BÜLENT ECEVİT ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
MADEN MÜHENDİSLİĞİ BÖLÜMÜ

C++ PROGRAMLAMA DİLİ DERS NOTLARI

Yrd. Doç. Dr. Alaaddin ÇAKIR
Mart 2015

İÇİNDEKİLER

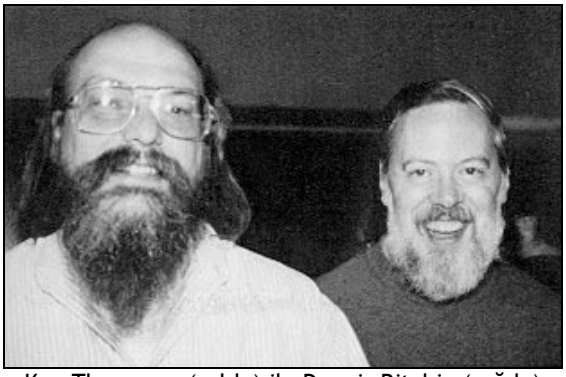
1. C ve C++ 4
1.1 C++'daki Yenilikler 5
1.2 Dev-C++ 6
2.1 Algoritma 7
3. C++ DİLİNİN TEMELLERİ 12
3.1 C++ Karakter Seti 12
3.1 C++ Programlarının Yapısı 13
3.2 Fonksiyonlar 15
3.2.1 main() Fonksiyonu 16
3.3 C++ Önişlemcisi 16
3.3.1 #include Emri 16
3.3.2 #define Emri 17
3.4 Açıklama Satırları 17
3.5 Temel Veri Türleri ve Değişkenler 17
3.5.1 Değişken Veri Türünün Bildirimi 18
3.5.2 Değişken Adlandırma Kuralları 19
3.5.3 Değişkenler 20
3.6 Sabitler 21
3.6.1 Tamsayı Sabitler 21
3.6.2 Ondalıklı Sabitler 21
3.6.3 Karakter Sabitler 21
3.6.4 Katar Sabitler 22
3.6.5 Tür Dönüşümü 23
3.7 Basit Veri Giriş Çıkışları 23
3.8 İşleçler (Operatörler) 24
3.8.1 Atama İşleci (=) 24
3.8.2 Aritmetik İşleçler 24
3.8.3 MOD Alma İşleci (%) 25
3.8.5 Aritmetik Atama İşleçleri 27
3.8.6 Karşılaştırma İşleçleri 28
3.8.7 Mantıksal İşleçler 29
3.8.8 Özel Amaçlı Ternary İşleci (? :) 29
3.8.9 İşlemlerin Öncelik Sırası 30
4. PROGRAM DENETİMİ 31
4.1. KARŞILAŞTIRMA İŞLEMLERİ 31

4.1.1. <if> YAPISI (Eğer...) 31
4.1.2. <if - else> YAPISI (Eğer... - Değilse...) 33
4.2 switch Deyimi 39
4.3. DÖNGÜLER 42
4.3.1. Belirli Sayıda Tekrar Döngüsü (for Döngüsü) 42
4.3.2. Koşullu Döngüler (while Döngüsü ve do...while Döngüsü) 45
4.3.3. Döngülerden Çıkış (break) ve Devam (continue) 49
4.3.4. İç İçe Döngüler 52
5. DİZİLER 54
5.1 Dizilere Başlangıç Değeri Atama 55
5.2 İki Boyutlu Diziler 64
HAZIR FONKSİYONLAR 71
Karakter İşleme Fonksiyonları 71
Dizgi İşleme Fonksiyonları 71
Zaman ve Tarih Fonksiyonları 71
Genel Amaçlı Fonksiyonlar 71
MATEMATİK FONKSİYONLARI 71

1. C ve C++

C programlama dili, Bell Laboratuvarları'nda çalışmakta olan Dennis Ritchie ve Ken Thompson tarafından, Unix sistemlerine programlama yapmak amacıyla 1969-1973 yılları arasında geliştirilir.

C'nin bu derece ilgi görmesinde, dil hakkında Dennis Ritchie ve Brian Kernighan'ın 1978 yılında yazdığı "C Programlama Dili" isimli kitabın etkisi büyüktür. Bu kitap; C dili, ANSI (Amerikan Ulusal Standartlar Enstitüsü) tarafından standartlaştırılana dek tek referans kaynağı olarak kullanılır.



Ken Thompson (solda) ile Dennis Ritchie (sağda)

C++, ilk olarak 1979 yılında Bell Laboratuvarları'nda araştırmacı olarak görev yapan Bjarne Stroustrup tarafından oluşturulmuştur. Stroustrup, doktora çalışmasında; ele aldığı bazı "nesne tabanlı" araçların özelliklerini, özellikle de sınıfları (class) destekleyen bir C sürümü geliştirmeye çalışmaktadır. Bu çalışma, C programlama dilinin yaratıcıları olan Dennis Ritchie ve ekibinin de içinde bulunduğu bir grup tarafından da desteklenmektedir. Geliştirilen bu sürümüne ilkin "Sınıflı C (cfront)" adı verilir. Ortaya çıkan ürünün potansiyeli kısa sürede keşfedilir ve 1983 yılında artırma operatörü (++) eklenerek ismi C++'a (C Plus Plus) çevrilir.

O günden bu güne C++ üç önemli evrim geçirir. Bunlardan ilki 1985, ikincisi 1990 yılındadır. Üçüncüsü ise, C++'ın standartlaştırılması sırasında meydana gelir. 1990 yılındaki evrimini takiben, C++'ı standartlaştırmak için bir çalışma başlatılır. ANSI ile ISO (Uluslararası Standartlar Örgütü) birleşerek bir standartlaştırma komitesi oluşturur. Teklif edilen standardın ilk tasarısı 25 Ocak 1994'de yapılır. Bu tasarıda ANSI/ISO C++ Komitesi, Stroustrup tarafından tanımlanan özellikleri korur ve bunlara bazı yeni özellikler ekler.

İlk tasarının hemen ardından standardın çok geniş bir şekilde yayılmasına neden olan bir olay meydana gelir. Alexander Stepanov tarafından Standart Şablon Kütüphanesi (Standart Template Library: STL) oluşturulur. STL, verileri işlemek için kullanılabilecek genel bir rutinler kümesidir; güçlü ve seçkindir. İlk tasarıdan sonra komite, C++'ın STL'yi de içermesine karar verir.

Sonuç olarak, C++'ın iki sürümü vardır. Bunlardan ilki, Stroustrup'un orijinal tasarımına dayanan geleneksel C++ sürümüdür. İkincisi ise, Stroustrup ve ANSI/ISO Standartlar Komitesi tarafından oluşturulmuş olan Standart C++'dır. C++'ın bu iki sürümü temelde birbirlerine çok benzemektedir; fakat Standart C++, geleneksel C++'da bulunmayan çeşitli ek özelliklere sahiptir ve onun geliştirilmiş şeklidir.

Genel bir anlatımla; C++, C'yi temel alarak genişletilmiş olup, C'nin kapsamına ek olarak, günümüzde yaygın olarak kullanılan Nesneye Dayalı Programlama yapısını da desteklemektedir. C++'ın kendine ait nesneye dayalı bir kütüphanesi vardır; fakat yine de C'nin standart kütüphanesindeki tüm fonksiyonları destekler; onunla aynı kontrol yapılarını kullanır; C tarafından tanımlanmış tüm veri tiplerini de içerir.

1.1 C++'daki Yenilikler

C++'daki önemli yeni özellikleri daha iyi açıklayabilmek amacıyla, önce geleneksel C++ yapısında, ardından Standart C++'a uygun olarak yazılmış iki program iskeleti örneği vereceğiz.

```
/*
geleneksel yapida bir c++ programi
*/

#include <iostream.h>

int main()
{
  /* program kodu */
  return 0;
}
```

Yukarıdaki programda #include deyimine özellikle dikkat edelim. Bu deyimle, C++'ın I/O (input/output) sistemini desteklemesini sağlayan iostream.h dosyasını dahil ediyoruz; ki, klasik C için stdio.h neyse, C++ için de bu dosya odur.

```
/*
standart c++ yapisinda ise
*/

#include <iostream>
using namespace std;

int main()
{
  /* program kodu */
  return 0;
}
```

Bir kütüphane fonksiyonunu kullanabilmemiz için onun başlık dosyasını programımıza dahil etmemiz gerekir. Bu da #include deyimiyle yapılır. Örneğin, C'de I/O fonksiyonları için gerekli olan stdio.h başlık dosyasını şu şekilde programa dahil etmemiz gerekir:

```
#include <stdio.h>
```

C++, ilk oluşturulduğu bir iki yıl içerisinde C ile aynı başlık stilini kullanmıştır. C++, sizin oluşturacağınız başlık dosyaları için ve eski sürümlere uyum sağlaması açısından hâlâ C stili başlıkları desteklemekle birlikte, kendi kütüphanesi tarafından kullanılan yeni bir başlık stili getirmiştir. Bu yeni tarz başlıklar, dosya adlarını değil, dosyalara derleyici tarafından bağlanabilmesi mümkün standart tanımlayıcıları belirler. Bu nedenle, bu stillerde .h uzantısı yoktur.

C++, C'nin fonksiyon kütüphanesinin tamamını kapsamaktadır ve C kütüphanesinin başlık dosyalarını da desteklemektedir. Örneğin, stdio.h ve ctype.h gibi başlık dosyaları hâlâ geçerlidir. Ancak C++'da, C'nin standart başlıklarındaki dosya adlarına bir "c" öneki eklenmiştir ve ".h" düşmüştür. Örneğin, math.h için C++'ın yeni tarz başlığı <cmath> şeklindedir. string.h ise <cstring> şeklini almıştır.

Bir diğer yeni özellik ise, namespace'lerdir. Namespace'ler, kütüphane fonksiyonları ve diğer elemanların tanımlanması için ayrılmış alanlardır. Namespace kullanmamızın amacı

ad çakışmalarını engellemek için tanımlayıcı adlarını uygun şekilde yerleştirmektir. Programımıza yeni bir başlık eklediğimizde, bu başlığın içeriği std namespace'in içine yerleştirilir ve burada saklanır. Bu iş için;

```
#using namespace std;
```

deyimini kullanabiliriz. Bu deyim derlendikten sonra eski veya yeni başlıklarla çalışıyor olmamızın bir önemi kalmaz.

Önemli yeni özelliklerden biri de C++'ın I/O konsolunda karşımıza çıkmaktadır. C++; printf() ve scanf() gibi I/O fonksiyonları yerine, "<<" ve ">>" gibi I/O operatörleri kullanarak daha yeni bir yol önermektedir.

printf() fonksiyonu yerine << çıkış operatörünü kullanarak, bir çıkış ifadesini veya geçerli herhangi bir C++ ifadesini çıkışa göndermek mümkündür. Örneğin aşağıdaki deyim, 'Bulent Ecevit Universitesi' ifadesinin ekranda gösterilmesini ve satır başı yapılmasını sağlar:

```
cout << "Bulent Ecevit Universitesi \n";
```

scanf() fonksiyonu yerine >> giriş operatörünü kullanarak klavyeden giriş yapabilir; örneğin aşağıdaki deyimlerle, klavyeden giriş yapılarak num değişkenine bir tamsayı değer atayabiliriz.

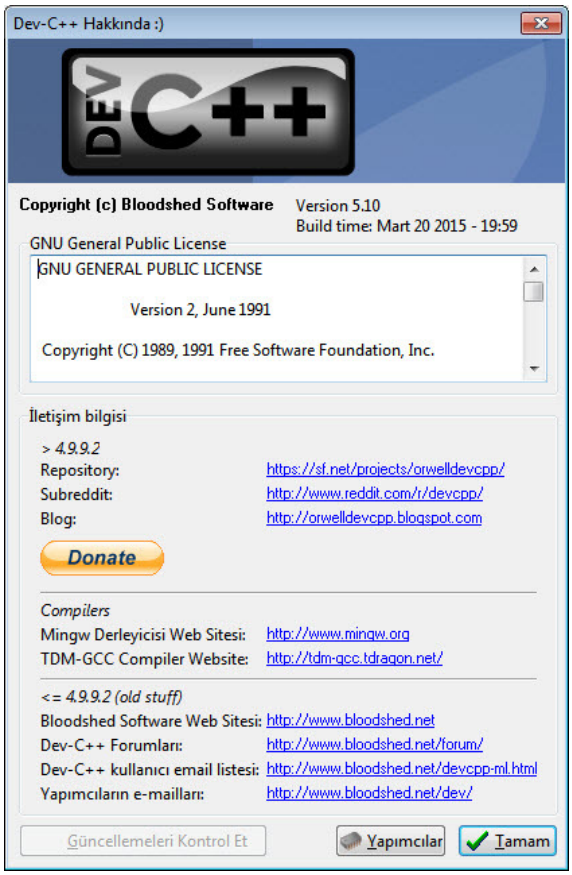
```
int num;
cin >> num;
```

C++'ın << ve >> gibi I/O operatörlerini kullanabilmek için <iostream> başlığını eklememiz gerektiğini bir kez daha belirtmek isteriz.

1.2 Dev-C++

Bu ders notları, Bloodshed Software (bloodshed.net) tarafından geliştirilen Dev-C++ Version 5.10 (Portable) sürümüne yönelik olarak hazırlanmıştır.

Söz konusu sürümün, şu ana dek hazırlanmış en güncel C++ sürümü olmasının yanı sıra; ücretsiz olması, Türkçe menülere sahip bulunması, kurulum gerektirmediği için USB flash belleklerden dahi kolayca çalıştırılabilmesi, hem 32 bit hem de 64 bit işletim sistemleri üzerinde sorunsuz olarak görev yapan dahili derleyicisiyle birlikte kullanılabilmesi ve ileri programlama tekniklerinin programa dahil edilebilmesi için devpaks.org sitesinden ücretsiz WebUpdate hizmeti alabilmesi gibi önemli avantajları vardır.



2. PROGRAMLAMAYA GİRİŞ

Bilgisayar yazılımları doğal olarak bir sorunu yani bir problemi çözmek üzere hazırlanır. Aslında problem çözümü sadece bilgisayar bilimleri alanında değil, diğer tüm bilim alanlarında ve hatta günlük yaşantımızda karşımıza çıkan bir olgudur. Problemin özelliğine bağlı olarak, her bir alanda kendi özel çözüm yöntemleri ve problemi anlama ve çözümleme yöntemleri bulunmaktadır. Bir fizikçinin probleme yaklaşım biçimi ile bir bilgisayar yazılımcısının yaklaşım biçimi farklılıklar gösterebilir.

Biz konuya program geliştirme açısından bakarak, problemin nasıl çözümleneceği konusuna eğileceğiz. Bu açıdan bakıldığında, problemin ortaya çıkışından, çözümünü gerçekleştirinceye dek olan sürecin iki temel aşamadan oluştuğunu söyleyebiliriz:

- Problem çözme aşaması
- Gerçekleştirim aşaması

Problemin çözüm aşaması da kendi içinde alt süreçlere ayrılmaktadır. Problem çözme aşamasında aşağıdaki adımlar yerine getirilir:

- Problemin tanımlanması
- Çözümün ana hatlarının ortaya konulması
- Ana hatlara bağlı olarak bir algoritmanın geliştirilmesi
- Algoritmanın doğruluğunun sınanması

Problemle ilgili algoritma ortaya konulduktan sonra, bu algoritmaya uygun olarak bilgisayar programlarının hazırlanması ve çalıştırılması gerekecektir. Gerçekleştirim adı verilen bu aşamada aşağıdaki adımlar yerine getirilir:

- Algoritma kodlan belirli bir programlama diline dönüştürülür.
- Program bilgisayarda çalıştırılır.
- Programın belgelemesi ve bakımı yapılır.

Bu iki süreç birleştirilerek, problemi bilgisayar aracılığıyla çözmek için, algoritma bulma ve programlama süreci şu şekilde ortaya konulabilir:

- Problemi belirleme ve anlama
- Algoritma hazırlama
- Program geliştirme
- Programı yürütme ve doğruluğunu değerlendirme.

Programlama süreci ardı ardına işlemlerden oluşmasına rağmen, bu sürecin herhangi bir adımından geriye dönerek işlemleri tekrarlamak mümkündür.

2.1 Algoritma

Algoritma, bir problemin çözülebilmesi için adım adım uygulanan kurallar dizisidir ve bir problemi çözmek için gerekli yöntemi tanımlar. Algoritma, doğru olarak icra edildiği zaman, verilen problemi çözecek adımlar dizisinden oluşur.

Bir algoritma için basitlik ve adım sayısı üzerinde önemle durulan iki konudur. Algoritmanın basitliği, kolay anlaşılır olması ile ilişkilidir. Adım sayısının ise az olması istenir. Aynı işi gören iki algoritma arasında, bilgisayarda daha az zaman gerektireceği için, adım sayısı az

olan algoritma tercih edilir. Algoritmalar, her türlü alternatif gidiş yolu düşünülerek sonuca ulaşıldığının garanti edilmesini hedef alır. Buna ek olarak bir algoritmada, girilen verilerin değerlendirilmesi ve buna karşılık olarak sonuçların elde edilmesi gereklidir. Algoritmada olması gereken özellikler aşağıda açıklanmaktadır.

- **Girdi/Çıktı:** Her algoritmanın bir giriş ve bir çıkış değeri olmalıdır. Girdi; algoritmanın üzerinde işlem yapması için aldığı veridir; çıktı ise algoritmanın girdiye karşı elde ettiği sonuçtur.
- **Açıklık:** Birden fazla anlama gelebilecek yani ikilemde kalınabilecek ifadelere sahip adımlar oluşturulmamalıdır. Bu nedenle her adım açık ve anlaşılır bir biçimde ifade edilmelidir.
- **Sonluluk:** Bir algoritma her ne koşulda olursa olsun sonlu sayıda işlem içermeli ve bu işlemlerin de süresi sonlu olmalıdır.
- **Etkinlik:** Bir algoritma aynı zamanda gereksiz tekrarlardan kaçınmalıdır ve gerektiği zaman başka bir algoritma içinde de kullanılabilmelidir.

Algoritmalar oluşturulurken sıklıkla ya doğal dilden ya da akış diyagramlarından yararlanılır. Eğer algoritma oluşturulurken bir dil kullanılmışsa bu dil, matematiksel ifadelerin de bulunduğu ve konuşurken kullandığımız bir dil olduğu gibi, özel olarak algoritma tasarımı için hazırlanmış bir dil de olabilir. Öncelikle, algoritmaların konuşma dili yardımıyla oluşturulmasına yönelik birkaç örneği inceleyelim.

ÖRNEK-1: 30 adet öğrencinin Matematik sınavında aldığı notları bilgisayara giriniz. Notların toplamını hesapladıktan sonra toplam değerini 30'a bölerek sınıf başarı ortalamasını bulunuz. Ortalama 70'in üstünde ise ekrana "başarılı sınıf' aksi takdirde ise "vasat sınıf' mesajını yazdırınız.

ÖRNEK-2: Girilen İki Sayıyı Toplayan Programın Algoritması

Girdi: Birinci sayı (x), İkinci sayı (y)
Çıktı: İki sayının toplamı (toplam)

1. Başla
2. x değerini gir.
3. y değerini gir.
4. toplam=x+y
5. toplam değerini ekrana yaz.
6. Bitir

ÖRNEK-3: Girilen İki Sayıyı Karşılaştırıp Büyük Olan Sayıyı Bulan Algoritma

Girdi: Birinci sayı (x), İkinci sayı (y)
Çıktı: Büyük sayı

1. Başla
2. x değerini gir.
3. y değerini gir.
4. Eğer x,y'den büyük ise adım 6'ya git.
5. Eğer y,x'den büyük ise adım 7'ye git.
6. X değerini ekrana yaz ve adım 8' e git.
7. Y değerini ekrana yaz ve adım 8' e git.
8. Bitir.

ÖRNEK-4: O’dan 100’e Kadar Olan Sayıların Karesini Alıp Ekrana Yazdıran Algoritma

Girdi: Sayılar (s)
Çıktı: Sayının karesi (k)

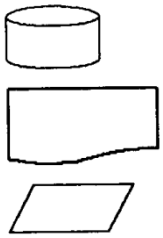
1. Başla
2. s değerine ilk değer olarak 1 değerini ver.
3. k=s*s yap.
4. k değerini ekrana yaz.
5. s değerini 1 arttır.
6. s>100 ise adım 8'e git.
7. Adım 3'e git.
8. Bitir.

ÖRNEK-5: Bir Sınıftaki Öğrencilerin Bir Dersten Aldıkları Notların Ortalamasını Ekrana Yazan Algoritma

Girdi: Öğrencilerin notları (nl, n2, n3... nSon), öğrenci sayısı (x), indis (i), Notların toplamı (toplam).
Çıktı: Ortalama (ort)

1. Başla
2. x değerini gir.
3. i değerine ilk değer olarak 1 değerini ver
4. Öğrencinin notunu (n) gir.
5. toplam=toplam+n yap.
6. i'yi 1 arttır.
7. Eğer i>x ise Adım 9'a git.
8. Adım 4'e git.
9. ort=toplam/x
10. ort değerini ekrana yaz
11. Bitir.

Akış diyagramı, bir algoritmanın adımlarının mantıksal sırasını, adımların birbiri ile bağlantısını, bir işlemden diğerine nasıl gidileceğini belirten kontrol mekanizmalarını, özel bazı şekil ve sembollerle anlatan bir ifade tarzıdır. Akış diyagramlarında kullanılan şekil ve semboller artık standart hale gelmiştir. Aşağıda, bir akış diyagramında sıklıkla kullanılan şekil ve semboller ile kullanıldıkları yerler verilmektedir.

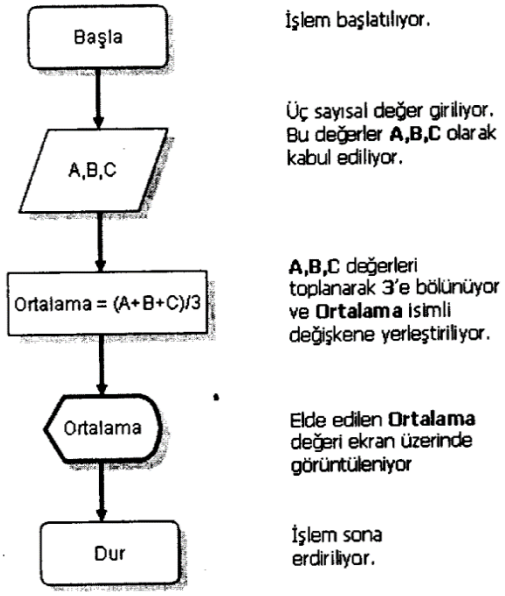


Veritabanı

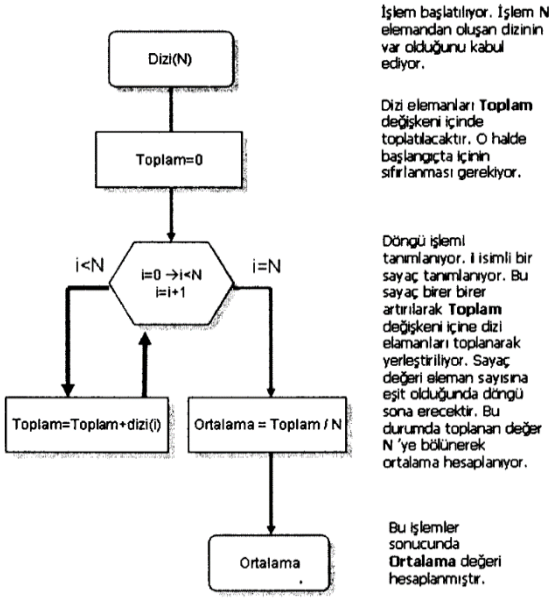
Kağıt çıktısı

Genel giriş/çıkış işlemeleri

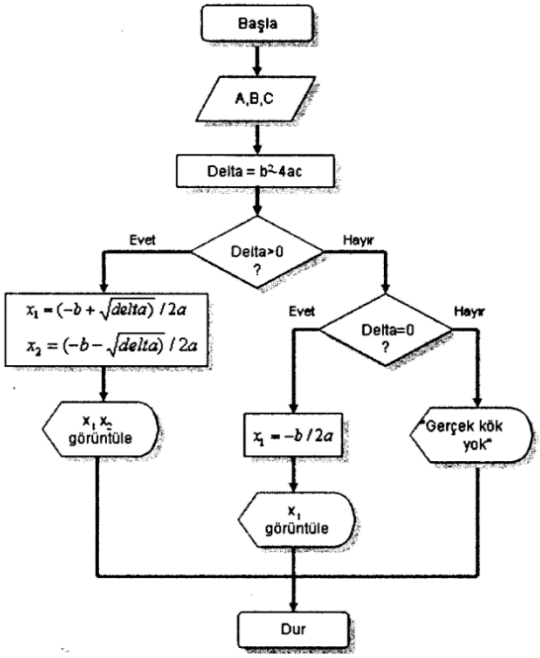
Çeşitli problemlerin çözümüne yönelik olarak hazırlanan aşağıdaki akış diyagramı örneklerini inceleyiniz.



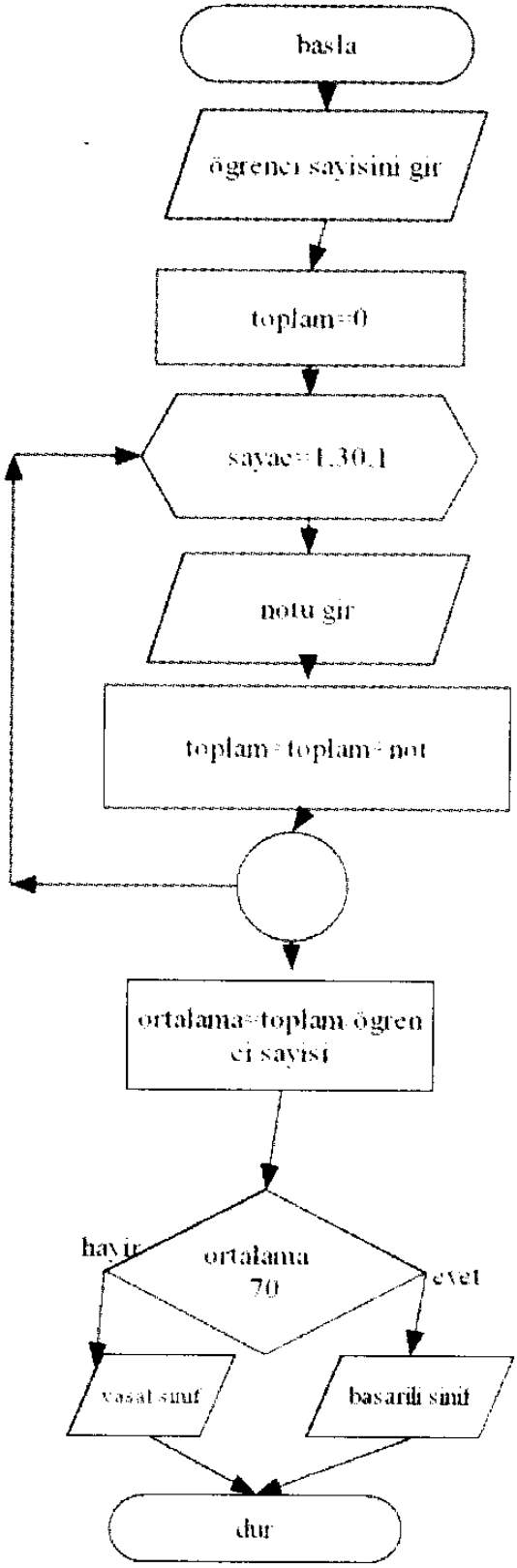
Şekil 1. Üç sayının ortalaması.



Şekil 2. N tane sayının ortalaması.



Şekil 3. İkinci dereceden bir denklemin kökleri.



Şekil 4. 30 öğrencinin aldığı notların ortalaması.

3. C++ DİLİNİN TEMELLERİ

3.1 C++ Karakter Seti

Her programlama dilinin kendine özgü bir takım karakterleri vardır. Bu karakterler ve ne anlama geldikleri önceden tanımlanmıştır. Örneğin; * (yıldız) karakteri BASIC dillerinde sadece çarpma anlamına gelirken, C++'da çarpma işleminin yanı sıra çok daha farklı işlevler yüklenmiştir. Bu durum diğer karakterler için de geçerli olabilir. Örneğin ^ (şapka) işareti BASIC dillerinde üs alma anlamına gelirken, C++ için bu karakterin böyle bir anlamı yoktur. Çizelge 3.1'de C++'da kullanılabilecek karakterler sınıflandırılmıştır.

Çizelge 3.1 C++ karakter seti.

RAKAMLAR
0 1 2 3 4 5 6 7 8 9
KÜÇÜK ALFABETİK KARAKTERLER
a b c d e f g h i j k l m n o p q r s t u v w x y x
BÜYÜK ALFABETİK KARAKTERLER
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ÖZEL KARAKTERLER
. , ; : ' " + - * / _ = ! ? # $ % & @ < > ( ) { } [ ] \ | ~ ^
ETKİSİZ BOŞLUK KARAKTERLERİ
\n \t \v \r
DİĞER KARAKTERLER
\a \b \0 \\ *

Etkisiz boşluk karakterlerinin ingilizce karşılığı 'whitespace' olup, bu ifade 'beyaz boşluk' şeklinde Türkçeye çevrilebilir. Etkisiz boşluk karakterlerinin program içinde kullanılıp kullanılmaması derleyici için farketmez. Ancak bu durum C++'la program yazan bizler için önemlidir. Yazdığımız programların okunabilirliğinin kolaylaştırılması amacıyla sıkça kullanacağımız karakterlerin başında etkisiz boşluk karakterleri gelmektedir.

Tablo 3.1'de yer alan Etkisiz Boşluk Karakterleri ile Diğer Karakterler'in yüklendiği işlevler Tablo 3.2'de verilmektedir.

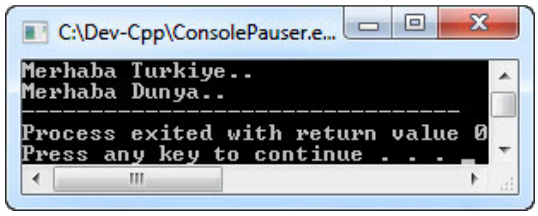
KARAKTER	ANLAMI	GÖREVİ
\n	Yeni Satır	Bir sonraki satıra geçmek
\t	Tab (Sekme)	Bir tab ölçüsünde boşluk bırakmak
\v	Dikey Tab	Bir tab ölçüsünde dikey boşluk bırakmak
\r	Satır Başı	İmleçi satır başına konumlandırmak
\a	Alarm	Uyarı amaçlı bir ses çıkarmak
\b	Backspace (Geri Al)	Kendinden önceki karakteri silmek
\0	Null	Karakter dizilerini string haline getirmek
\\	\ karakterini kullanabilmek için \\ şeklinde yazılır	
\"	" karakterini kullanabilmek için \" şeklinde kullanılır	

3.1 C++ Programlarının Yapısı

Öncelikle, çok basit bir örnek program yazalım ve C++ programlarının yapısını bu örnek program üzerinde inceleyelim.

Uygulama

```
// ilk C++ programim
/* bu program dev-c++ v5.10 yazilimi
kullanilarak hazirlanmistir */

#include <iostream>
using namespace std;

int main() {
cout << "Merhaba Turkiye..." << "\n";
cout << "Merhaba Dunya...";
return 0; }
```



1., 2. ve 3. satırlarda programımıza ait açıklama ifadeleri yer almaktadır.

```
// ilk C++ programim
/* bu program dev-c++ v5.10 yazilimi
kullanilarak hazirlanmistir */
```

C++'da yazılan bir program çalıştırıldığında '//' ile başlayan satırlar atlanır; çalıştırılmaz. '//' karakterleri ile başlayan satırlar, programın belirli satırlarında (kod parçalarında) ne iş yapıldığının açıklandığı yorum satırlarıdır. C++'da birden fazla satırı yorum olarak belirtmek için '/*' ve '*/' karakterleri kullanılır. '/*' karakterleri yorumun başladığı yeri, '*/' karakterleri ise yorumun bittiği yeri belirtir.

Çoğu kişi tarafından gözardı edilse de, yorumlar, bir programın akışının kolayca anlaşılabilmesi için çok önemlidir. Yorum satırları eklenmemiş bir programın aradan zaman geçtikten sonra çözümlenmesi ve üzerinde değişiklik yapılabilmesi çok zordur. Bu nedenle, programlarımızda mutlaka yorum yazmalıyız.

4. satır boş bırakılmış olup, herhangi bir kod görülmemektedir. Gerekli yerlerde bir satırın boş bırakılması, yazdığımız programın okunabilirliğini artırmak için başvurduğumuz yöntemlerden biridir ve bırakılan boşlukların, programımızın performansı üzerinde herhangi bir etkisi yoktur.

5. satırda 'iostream' isminde bir dosya programımıza eklenmektedir.

```
#include <iostream>
```

#include komutu, kendisinden sonra gelen dosyayı programımıza dahil eder. Diğer bir ifadeyle, bu satır, 'iostream adlı dosyayı bul ve programa ekle' anlamı taşımaktadır. Birtakım dosyaları neden programımıza eklemek zorunda olduğumuzu, ileride, daha ayrıntılı olarak ele alacağız.

6. satırda, C++'la gelen bir yenilik karşımıza çıkmaktadır.

```
using namespace std;
```

Normal olarak, programımızın 10. ve 11. satırında kullanacağımız 'cout' ifadesi için C++'dan izin istememiz gerekir. Sadece 'cout' ifadesi için değil, ileride göreceğimiz 'cin' vb. gibi ifadelerden önce de std:: kalıbını kullanmak zorundayız.

```
std::cout << "Merhaba Turkiye.." << "\n";
std::cout << "Merhaba Dunya..";
```

Ancak ‘using namespace std;’ ifadesi isim çakışmalarını önlemek amacıyla, std isim alanının kullanımının programa dahil edilmesini sağlar ve böylece 'cout', 'cin' vb. gibi ifadelerden önce std:: ifadesini kullanmaya gerek kalmaz.

7. satır da, 4. satır gibi, programımızın okunabilirliğini artırmak amacıyla eklenmiş boş bir satırdır.

8. satır, programımızın işlemeye başladığı satırdır. Her C++ programı 'main' isminde bir fonksiyon içerir. Fonksiyonlar ise, program içerisinde çağırarak kullandığımız, belli bir işi yaparak sonucu bize geri döndüren kod parçalarıdır. Bir C++ programı bir veya daha fazla fonksiyondan oluşur. Bir fonksiyon, altyordam (subroutine) olarak da adlandırılır ve bunlar programın bir başka yerinden isimleri kullanılarak çağırılabilir.

'main' fonksiyonu, programın ilk olarak çalışmaya başladığı özel bir fonksiyondur. 'main' ifadesinin başında yer alan 'int' ifadesi, fonksiyonun işini tamamladıktan sonra kendisini çağıran sisteme ne tür bir veri döndüreceğini tanımlayan bir ifadedir. Bu konuyu da, ileride, daha ayrıntılı olarak ele alacağız.

9. satırda yer alan '{' karakteri, 'main()' fonksiyonunun başlangıcını; 13. satırda yer alan '}' karakteri ise söz konusu fonksiyonun bittiğini göstermekte olup, bu karakterler arasında kalan kısımlar 'blok' olarak adlandırılmaktadır. Bir fonksiyon; '{' ve '}' işaretlerinin arasına yerleştirilen bir ya da daha fazla sayıda bloklardan oluşur.

Blokların içinde deyimler yer alır. Deyimler, programın amacına ulaşması için gereken işlemleri yerine getiren komutlardır. Deyimler noktalı virgül ';' işareti ile son bulur. Bir satırda, deyim sonları ';' işaretiyle belirlenmek koşuluyla, birden fazla deyime yer verilebilir. Bir satırın sonunda ';' işareti yer almıyorsa, bu satırdaki deyimin bir alt satırda da devam ettiği anlaşılır.

Bu durumda, fonksiyonumuz sadece 10. ve 11. satırları içeren 2 satırlık bir fonksiyondur; 9. satırda başlamakta ve 13. satırda bitmektedir.

10. ve 11. satırlar, programımızda asıl işi yapan satırlardır.

```
cout << "Merhaba Turkiye.." << "\n";
cout << "Merhaba Dunya..";
```

Bu satırlarda, 'cout' deyimiyle birlikte '<<' operatörü kullanılarak, verilen değerleri, ki bizim örneğimizde bu değerler 'Merhaba Turkiye..' ve 'Merhaba Dunya..' ifadeleridir, ekranda görüntülüyoruz. 10. satırın sonunda yer '\n' etkisiz boşluk karakteriyle de satır başı yaparak, ifadelerinizin ekranda alt alta iki satır şeklinde görüntülenmesini sağlıyoruz.

12. satırda fonksiyonumuzun görevi tamamlanmaktadır.

```
return 0;
```

'return 0' ifadesi, program akışının bu fonksiyondan çıkarak, kendisini çağıran satıra geri dönmesini sağlar. Bizim 'main()' fonksiyonumuz herhangi bir kodla çağırılmadığı için (çünkü

bu fonksiyon zaten programın çalışmaya başladığı yerdir) program sisteme geri döner, yani sonlanır.

13. satır, yukarıda da belirtiğimiz gibi, fonksiyonumuzun sonunu belirtmektedir.

C++ kaynak programları oluşturulduktan sonra bir isim verilerek kaydedilir. Derleyicinin programı algılayabilmesi için, kaynak dosyanın uzantılı '**.cpp**' olmalıdır.

Kaydedilen program derlenir. Derleme sonucunda herhangi bir hata ile karşılaşılmaz ise, '**.obj**' dosyası yaratılır.

Derlenen programa, **build** işlemi uygulanarak '**.exe**' uzantılı bir uygulama dosyası oluşturulur. Bu dosya çalıştırılarak programdan beklenen sonuçlar elde edilir.

3.2 Fonksiyonlar

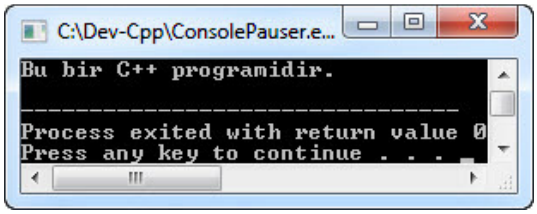
Fonksiyonlar, C++ programlama dilinin temel yapı taşlarıdır. Belirli bir adı olan program parçacıklarıdır ve bu isimlerle çağırılırlar. Fonksiyonların genel yapısı şu şekildedir.

türü fonksiyon adı (parametre listesi)
{
  deyimler
  return;
}

Fonksiyonlar, bir değer veya bir sonuç üretmek üzere tasarlanırlar. Bu nedenle bir veri türüne sahip olmaları gerekir. Örneğin; fonksiyon bir değer üretecek ise veri türü **int** olarak tanımlanır. Bu durumda, fonksiyondan sonuç döndürme işlemini gerçekleştirmek üzere, **return** deyimi kullanılır. Ancak, herhangi bir değer döndürülmeyecek ise, fonksiyonun veri türü olarak **void** tanımı yapılır.

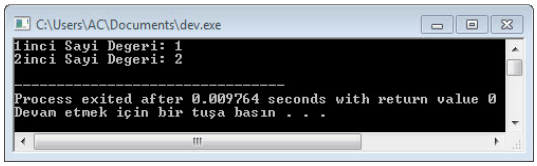
Uygulama

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

// Bu program bir mesaj goruntuler

int main() {
cout << "Bu bir C++ ";
cout << "programidir. " << "\n";
return 0; }
```



Uygulama

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

int fonk1();
int i=1; int j=2;

int main() {
cout <<"1inci Sayi Degeri: " << i <<"\n";
fonk1();
return 0; }

int fonk1() {
cout <<"2inci Sayi Degeri: " << j <<"\n";
return 0; }
```

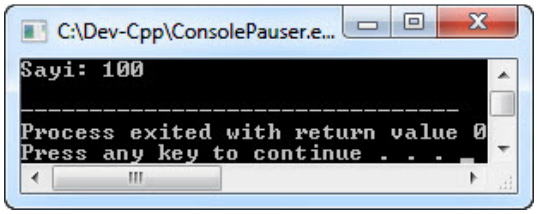


3.2.1 main() Fonksiyonu

Bir C++ programının içinde çok sayıda fonksiyon yer alabilir; ancak mutlaka bir **main()** fonksiyonu bulunmalıdır. Program yürütülmeye başladığında, öncelikle, programın içinde bağlanmış bulunan bir başlangıç yordamı çalışır ve bu yordam da **main()** fonksiyonunu çağırır.

Uygulama

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

int main() {
int num;
num=100;
cout << "Sayi: " << num << "\n";
return 0; }
```



3.3 C++ Önişlemcisi

C++ programlarının kendi derleyicileriyle olan ilişkisi **C++ önişlemcisi** yardımıyla sağlanır. Çeşitli emirlerden oluşan önişlemciler, C++ derleyicisinin kaynak kodunu denetlemekte kullanılır.

Önişlemci emirleri program içinde (**#**) işareti ile başlar. C++'ın en çok kullanılan önişlemci emirleri **#include** ve **#define** ile tanımlanmaktadır. Önişlemci emirleri (;) işareti ile sonlandırılmaz.

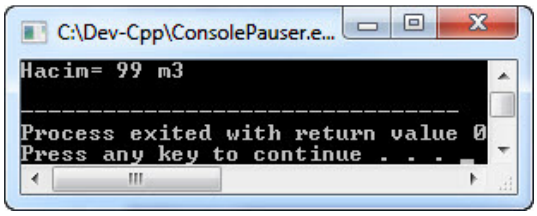
3.3.1 #include Emri

Bu emir, bir kaynak dosyasının programa dahil edilmesini sağlar. C++'a dahil edilen dosyalar **Başlık Dosyası** olarak adlandırılır. Başlık dosyalarında, standart kitaplık fonksiyonları hakkında bilgiler yer alır ve bu fonksiyonları kullanabilmek için söz konusu dosyaları programa dahil etmek gerekir.

Aşağıdaki programda main() fonksiyonu içinde bir mesajı yazdırabilmek için "<<" işleci kullanılmıştır. Bu işleci kullanabilmek için programa **<iostream>** başlık dosyasının dahil edilmesi gerekmektedir. Aksi takdirde program bu işleçleri yorumlayamayacaktır.

Uygulama

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

int main() {
int hacim;
hacim=99;
cout << "Hacim= " << hacim <<  " m3"
<< "\n";
return 0; }
```

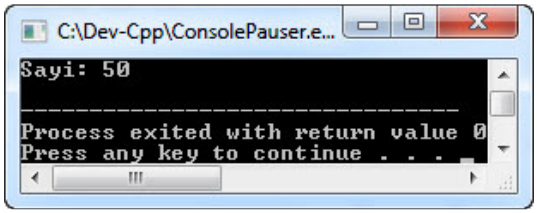


3.3.2 #define Emri

Bu emir, sembolik değişmezlerin tanımlanmasını sağlar. Söz konusu önişlemci, bir isim ve bir değer ile birlikte tanımlanır. Kullanılan ismin büyük harfle yazılması bir zorunluluk olmamakla beraber, genel bir alışkanlık olarak kabul edilmektedir.

Uygulama

```
#include <iostream>
#define SON 50
using namespace std;

int main() {
cout << "Sayi: " << SON <<  "\n";
return 0; }
```



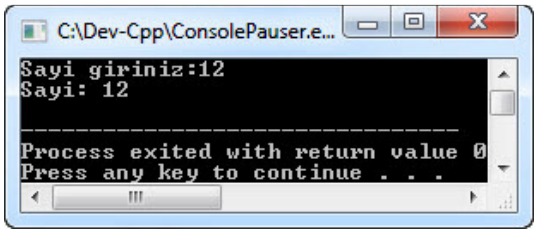
3.4 Açıklama Satırları

Özellikle uzun C++ programlarında, program bölümlerinin hangi işlemi yerine getirdiğinin bilinmesi için, satırlar arasına söz konusu kodların hangi amaçla yazıldığını belirten ifadeler kaydetmekte yarar vardır.

C++ programının herhangi bir yerine açıklama satırları eklemek için, açıklamanın başına (//) eklenir. Bu işaretle başlayan ifadeler C++ derleyicisi tarafından yok sayılır ve herhangi bir işleme tabii tutulmaz.

Uygulama

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

// Bu program klavyeden
// girilen bir degeri okuyarak
// ekranda goruntuler

int main() {
int sayi;
cout << "Sayi giriniz:";
cin >> sayi;
cout << "Sayi: " << sayi << "\n";
return 0; }
```



3.5 Temel Veri Türleri ve Değişkenler

C++'de kullanılacak olan tüm değişkenler, kullanılmadan önce programa bildirilmeli ve tanıtılmalıdır. Bu bildirim esnasında değişkenin veri türünün de belirlenmesi gerekmektedir.

<veri türü> <değişkenin adı>;

C++'da yaygın olarak kullanılan temel veri türleri şunlardır:

<int> - <long> - <short>
Tamsayıları tanımlamak için kullanılırlar.

int ve **long** veri türü bellekte 4 bayt, **short** veri türü ise 2 baytlık yer işgal eder.

<double> - <long double> - <float>
Ondalıklı sayıları tanımlamak için kullanılırlar.
double ve **long double** veri türü bellekte 8 bayt, **float** veri türü ise 4 baytlık yer işgal eder.

<char>
Bir alfabetik karakteri veya karakter katarlarını tanımlamak için kullanılırlar.
Her karakter bellekte 1 baytlık yer işgal eder.

C++'da kullanılan veri türleri Çizelge 1'de verilmektedir.

Çizelge 1. C++'da kullanılan veri türleri.

Tür (type)	Tanım	Kontrol Karakteri	Sınırlar
İŞARETLİ TÜRLER			
int	Tamsayı		-32 768 ile +32 767
short	Kısa Tamsayı		-32 768 ile +32 767
long	Uzun Tamsayı	l veya L	-2 147 483 648 ile +2 147 483 647
float	Kayan Ondalıklı Sayı	f veya F	$3.4*10^{-38}$ ile $3.4*10^{38}$
double	Çift Ondalıklı Sayı		$1.7*10^{-308}$ ile $1.7*10^{308}$
long double	Uzun Ondalıklı Sayı		$1.2*10^{-4932}$ ile $1.2*10^{3932}$
char	Karakter		-128 ile +127
İŞARETSİZ TÜRLER			
unsigned int	İşaretsiz Tamsayı		0 (sıfır) ile +4 294 967 295
unsigned short	İşaretsiz Kısa Tamsayı		0 (sıfır) ile +65 535
unsigned long	İşaretsiz Uzun Tamsayı		0 (sıfır) ile +4 294 967 295
unsigned char	İşaretsiz Karakter		0 (sıfır) ile +255
DİĞER			
bool	doğru ya da yanlış		doğru için 1, yanlış için 0 değeri alır

3.5.1 Değişken Veri Türünün Bildirimi

C++ içinde kullanılacak değişkenin veri türünü bildirmek için şu şekilde bir tanım yapılır:

<veri türü> <değişkenin adı>;

```
int yas;
float fiyat;
char harf;
```

Bir değişkene programın içinde herhangi bir yerde belirli bir değer atayarak içeriğini değiştirmek mümkündür. Ancak çoğu kez, değişkenin veri türü daha başlangıçta belirlenirken, bir değere de sahip olması istenir.

<veri türü> <değişkenin adı> = <değeri>;

```
int yas=51;
float fiyat=32.99;
char harf="a";
```

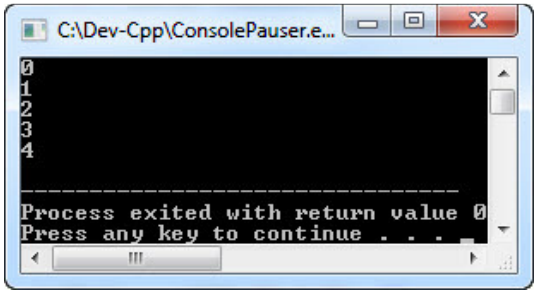
Programımızda birden fazla değişken kullanacaksak, aynı türden olması koşuluyla, bu değişkenleri yan yana yazarak da tanımlayabiliriz.

int sayi1, sayi2;
int sayi1=32, sayi2=99;

Değikenin veri tüeünün bildirildiği satırlar da, diğer C++ deyimleri gibi, ";" işareti ile son bulmalıdır.

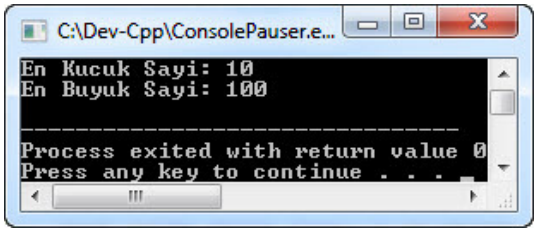
Uygulama

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

// Bu program 0 ile 4
// arasındaki tüm tamsayıları
// ekranda görüntüler

int main() {
int i;
for (i=0; i<=4; i++)
cout << i << "\n";
return 0; }
```



Uygulama

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

// Bu program iki degiskene baslangic
// degeri atar ve onlari ekranda
// goruntuler

int main() {
int enaz=10;
int encok=100;
cout <<"En Kucuk Sayi: " << enaz << "\n";
cout <<"En Buyuk Sayi: " << encok << "\n";
return 0; }
```



3.5.2 Değişken Adlandırma Kuralları

C++'da bir değişken adı tanımlarken dikkat etmemiz gereken bazı önemli kurallar vardır.

❖ C++, küçük ve büyük harflere karşı duyarlıdır. Örneğin;

char harf;
char Harf;
char HARF;

ifadelerinin üçü de farklı değişkenleri tanımlamaktadır. Bu nedenle değişken adlarında küçük harf - büyük harf kullanırken çok dikkatli olmalıyız.

❖ Değişken adlarında, Tablo 3.1'de verilen rakamlar, küçük alfabetik karakterler ve büyük alfabetik karakterler dışında, hiçbir simge veya sembol kullanılmamalıdır. Ancak, alt çizgi (_) karakteri bu kapsamın dışında bırakılmış olup, değişken adlarında kullanılabilir.

❖ Değişken adları mutlaka bir harf ile veya alt çizgi (_) karakteriyle başlamalı, asla bir sayı, simge veya sembol ile başlamamalıdır.

int sayi
int _sayi

❖ Bir değişkenin adı en fazla 255 karakterden oluşabilir.

❖ Değişken adında boşluk karakteri kullanılmamalıdır. Ancak, boşluk yerine alt çizgi (_) karakteri kullanılabilir.

int sayinin_karekoku

❖ Değişken adlarında, C++'a özgü anahtar sözcükler kullanılamaz. Bu sözcükler Tablo 3.2'de verilmektedir.

Tablo 3.2. C++'a özgü anahtar sözcükler.

asm	auto	bool	break	case
catch	char	class	const	const_cast
continue	default	delete	do	double
dynamic_cast	else	enum	explicit	extern
false	float	for	friend	goto
if	inline	int	long	mutable
namespace	new	operator	private	protected
signed	sizeof	static	static_cast	struct
switch	template	this	throw	true
try	typeid	typename	union	unsigned
using	virtual	void	volatile	while

3.5.3 Değişkenler

C++'da farklı amaçlara yönelik değişken tanımları yapılabilir. C++ programlarında, çeşitli türde değişkenler kullanılmakla birlikte, biz, şimdilik, sıklıkla kullanılan 2 tür değişkeni ele alacağız.

Yerel Değişkenler:
Program içinde birden fazla fonksiyon varsa, sadece tanımlandığı fonksiyonda geçerli olabilecek değişken türüdür. Bu tür değişkenler, fonksiyon sınırlarını belirten {} işaretleri içinde yer almalıdır.

Küresel Değişkenler:
Program içindeki tüm fonksiyonlarda geçerli olabilecek değişken türüdür. Bu tür değişkenler, fonksiyon sınırlarını belirten {} işaretleri dışında yer almalıdır.

Statik Değişkenler:
Bir fonksiyon içinde yerel olarak tanımlanmış bir değişkenin, program çalıştığı sürece içinde bulunduğu fonksiyonun tekrar tekrar çağırılması durumunda sabit kalması ve değişmemesi istendiğinde, o değişken statik değişken olarak tanımlanmalıdır.

3.6 Sabitler

Sabitler, programın başından sonuna kadar değeri değişmeyen program bileşenleridir. C++ dilinde aşağıda belirtilen veri türlerine sahip sabitler kullanılabilir:

> Tamsayı Sabitler
> Ondalıklı Sabitler
> Karakter Sabitler
> Katar Sabitler

3.6.1 Tamsayı Sabitler

Tamsayı sabitler; 'int' (tamsayı), 'short' (kısa tamsayı) ve 'long' (uzun tamsayı) olmak üzere üç türlüdür. Örnek olarak 1992 ifadesini ele alalım ve bir tamsayının türünün tanımlanması işini bu örnek üzerinde açıklayalım.

Bir sabitin hangi türe ait olduğunu belirtmek için o sabitin sonuna türünü belirtecek bir karakter eklenir. Eğer, sayısal bir ifadenin sonunda herhangi bir karakter yoksa, o ifadenin türü 'int' kabul edilir. Bu durumda, örneğimizdeki 1992 ifadesi 'int' türüne ait bir tamsayı olmaktadır. Bu ifadeyi 'long' türü olarak belirtmek için sonuna 'l' veya 'L' karakteri eklememiz gerekir: 1992l veya 1992L. Bu şekilde ifade artık 'int' türüne değil 'long' türüne ait olur.

Ayrıca, programın akışı içinde 'int' türü sınırlarını geçen tamsayılar da, sonlarında 'l' veya 'L' ekleri olmasa bile, otomatik olarak 'long' türüne çevrilir.

'short' türü içinse özel bir durum söz konusudur. Bir ifadenin değeri hesaplanırken 'short' türüne ait olsa da 'int' gibi işlem görür. Bu durumda 'short' türünde bir sabit yoktur diyebiliriz. Çünkü, 'short' sınırları içerisindeki sabitler, C++ tarafından 'int' türü olarak kabul edilmektedir.

3.6.2 Ondalıklı Sabitler

Ondalıklı sabitler; 'float' (kayan ondalıklı), 'double' (çift ondalıklı) ve 'long double' (uzun ondalıklı) olmak üzere üç türlüdür. Örnek olarak 1924.1992 ifadesini ele alalım ve bir tamsayının türünün tanımlanması işini bu örnek üzerinde açıklayalım.

Eğer ondalıklı bir sabitin sonunda herhangi bir karakter yoksa, o ifadenin türü 'double' olarak kabul edilir. Bu durumda, örneğimizdeki 1924.1992 ifadesi 'double' türüne ait bir ondalıklı sabit olmaktadır. Bu ifadeyi 'float' türü olarak belirtmek için sonuna 'f' veya 'F' karakteri eklememiz gerekir: 1924.1992f veya 1924.1992F. Bu şekilde ifade artık 'double' türüne değil 'float' türüne ait olur.

Sıkça kullanılmamakla birlikte, ondalıklı bir sabiti 'long double' türünde belirtmek için sonuna 'l' veya 'L' karakteri eklememiz gerekir: 1924.1992l veya 1924.1992L.

3.6.3 Karakter Sabitler

'char' (karakter) türünün -128 ile +127 veya 0 ile +255 arasında bir değer aldığını biliyoruz. Peki, bu sabitlerin ismi 'karakter' olduğuna ve alfabetik nitelik taşıdıklarına göre biz neden hâlâ sayısal ifadelerden söz ediyoruz? Çünkü, C++'da kullanılan her karakterin ASCII

(American Standard Code for Information Interchange: Bilgi Alışverişi için Amerikan Standart Kodları) tablosunda sayısal bir karşılığı vardır ve karakter sabitlerinin belirtilen aralıklarda tutulduğu bu sayılar kullanılan karakterlerin ASCII karşılıklarıdır. Diğer bir ifadeyle; biz, karakter sabiti olarak 97'den söz ederken, aslında 97 ifadesinin ASCII tablosundaki karşılığı olan 'a' karakterinden söz etmiş oluyoruz. ASCII tablosunu bu ders notlarının EKLER bölümünde bulabilirsiniz.

Karakter sabitlerini kullanırken, karakterlerin sayısal karşılıklarını kullanabildiğimiz gibi, doğrudan kendilerini de kullanabiliriz; ki C++'da yaygın olarak tercih edilen kullanım şekli de karakterlerin doğrudan kendilerinin kullanılmasıdır. Ancak, karakterlerin doğrudan kendilerini kullanacağımız zaman bu karakterleri tek tırnak (') içinde içinde belirtmek zorundayız.

Aşağıdaki satırlarda char türünde iki değişken tanımlanmakta ve bu değişkenlere sırasıyla 103 ve 'g' sabitleri atanmaktadır.

```
char karakter1=103;
char karakter2='g';
```

ASCII tablosunda 'g' karakterinin sayısal karşılığı 103 olduğu için, aslında bu iki satır aynı şeyi ifade etmektedir. Ancak, 'g' karakterinin tek tırnak işaretleri içinde yazıldığına özellikle dikkat edelim.

3.6.4 Katar Sabitler

'string' (karakter) sabitler, ardışık olarak sıralanmış karakter sabiti dizilerinden oluşmaktadır. C++'da çift tırnak (") içine alınmış her ifade 'string' türü bir sabittir. Aşağıdaki örnekleri inceleyiniz:

"beu" "1992" "1924.1992" "Bulent Ecevit Universitesi"

Görüldüğü gibi, çift tırnak içine alınan sayısal ifadeler de artık bir 'string' sabit olmuşlardır. Artık onlarla toplama, çıkarma gibi sayısal işlemler yapamayız.

Aslında C++'da 'string' adında bir tür yoktur. 'string' türü, derleyicinin birden çok karakter sabitine bir karakter dizisi şeklinde yaklaşmasıyla oluşur. Buna göre "beu muh fak" ifadesi;

'b' 'e' 'u' ' ' 'm' 'u' 'h' ' ' 'f' 'a' 'k' '/0'

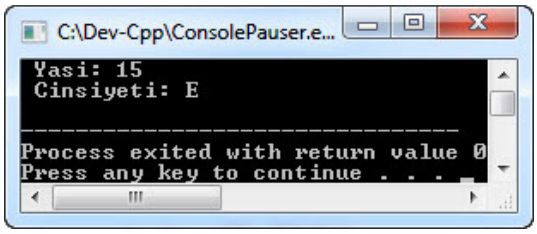
karakterlerinin bir araya gelmesinden oluşmaktadır. Derleyici tüm bu karakterlere bir dizi şeklinde davranmakta ve sonlarına '/0' karakterini ekleyerek onları bir araya getirmektedir.

Sabitler, C++ programında const sözcüğü ile tanımlanmakta ve kullanılacak değişmezin veri türünü bildirmek için şu şekilde tanımlar yapılmaktadır:

```
int const sabit adı = değeri;
char const sabit adı = 'değeri';
```

Uygulama

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

int const yas = 15;
char const cins = 'E';

int main() {
cout << " Yasi: " << yas <<"\n";
cout << " Cinsiyeti: " << cins <<"\n";
return 0; }
```

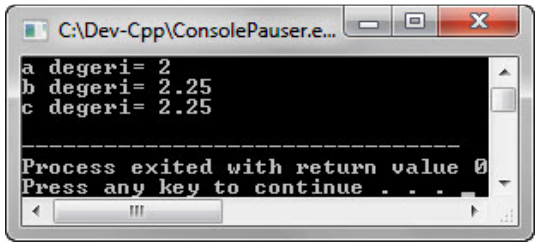


3.6.5 Tür Dönüşümü

Programlarımızda değişkenler veya sabitler birbirinden farklı türde olabilir. Eğer, böyle bir durum söz konusu ise, matematiksel işlemlerimizde hesap sonucumuzun hangi türde olacağı önemlidir. Bu nedenle, bir hata ile karşılaşmamak için tür dönüşümü yapılmalıdır.

Uygulama

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

int main(){
int sayi=9;
float a,b,c;
a=sayi/4;
b=sayi/4.0;
c=(float)sayi/4;
cout << "a degeri= " << a << endl;
cout << "b degeri= " << b << endl;
cout << "c degeri= " << c << endl;
return 0; }
```



Yukarıdaki uygulamada:

İlk işlemde, tamsayı olan <sayi> adındaki değişkeni tamsayı bir değere bölüyoruz; virgülden sonraki kısım yok sayılıyor ve sonuç tamsayı olarak <a> değişkene atanıyor.

İkinci işlemde, tamsayı olan <sayi> adındaki değişkeni ondalıklı bir bir değere bölüyoruz; virgülden sonraki kısım dikkate alınıyor ve sonuç ondalıklı bir değer olarak olarak <b> değişkene atanıyor.

Üçüncü işlemde, tamsayı olan <sayi> adındaki değişkeni önce <float> türü bir değişkene dönüştürüyoruz. Sonra, artık <float> olan değişkeni tamsayı bir değere bölüyoruz; sonuç ondalıklı bir değer olarak <c> değişkenine atanıyor.

3.7 Basit Veri Giriş Çıkışları

C++'da, standart giriş-çıkış nesnesine yönlendirme yapmak için "<<" işleci kullanılır.

Yönlendirme esnasında bir satır yazdırıldıktan sonra bir sonraki satırın başına atlamak için "\n" tanımı kullanılır.

Klavyeden veri girişinin söz konusu olduğu durumlarda ">>" işleci ile birlikte "cin" sözcüğü kullanılır.

Giriş-çıkış yapabilmek için, ilgili kitaplıkları içeren iostream dosyasının, aşağıda gösterildiği biçimde programa dahil edilmesi gerekmektedir.

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

3.8 İşleçler (Operatörler)

Bir veri üzerinde aritmetiksel işlem yaparak veriyi değiştirmek, iki veriden yeni bir veri elde edilmesini sağlamak, verileri karşılaştırmak veya veriler üzerinde mantıksal işlemler yapmak amacıyla kullanılan simgeler **işleç (operatör)** olarak adlandırılmaktadır.

3.8.1 Atama İşleci (=)

Bir değişkene herhangi bir değer atanacağı zaman (=) işleci kullanılır.

int sayi=12;
char harf='n';

3.8.2 Aritmetik İşleçler

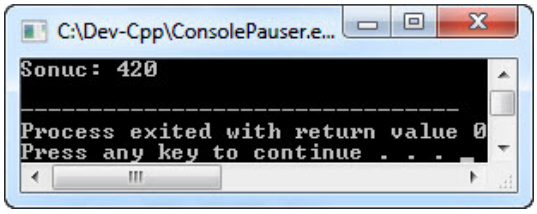
Matematiksel işlemler yaparken kullanılan işleçlerdir. (Tablo 3.2).

Çizelge 2. Aritmetik işleçler.

İŞLEÇ	ANLAMI
+	Toplama
-	Çıkarma
*	Çarpma
/	Bölme
%	Mod Alma (Bölme İşleminde Kalan)

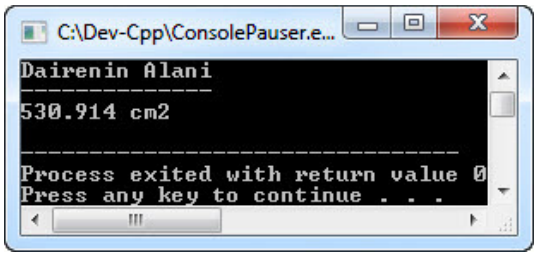
Uygulama

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

// Bu program iki sayisal degeri
// toplayarak sonucu goruntuler

int main() {
int sayi1=120;
int sayi2=300;
int toplam;
toplam=sayi1+sayi2;
cout << "Sonuc: " << toplam << "\n";
return 0; }
```



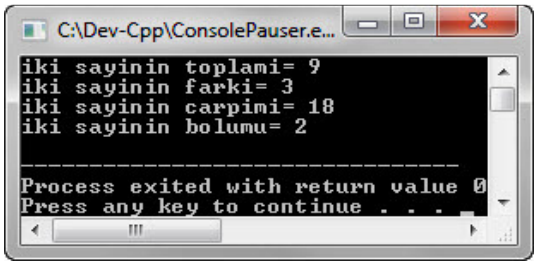
Uygulama

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

// Dairenin Alanı

int main() {
double pi=3.1415;
double yaricap=13;
double alan;
cout << "Dairenin Alani" << "\n";
cout << "--------------" << "\n";
alan=pi*yaricap*yaricap;
cout << alan << " cm2" << "\n";
return 0; }
```



Uygulama

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

int main() {
int sayi1=6,sayi2=3;
cout << "iki sayinin toplami= "
<< sayi1+sayi2 << endl;
cout << "iki sayinin farki= "
<< sayi1-sayi2 << endl;
cout << "iki sayinin carpimi= "
<< sayi1*sayi2 << endl;
cout << "iki sayinin bolumu= "
<< sayi1/sayi2 << endl;
return 0; }
```

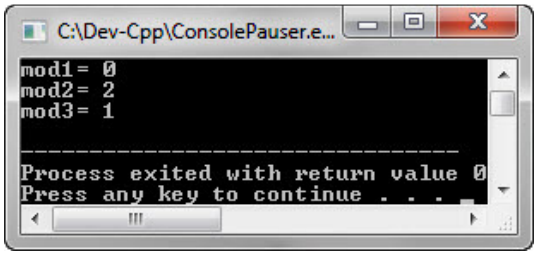


3.8.3 MOD Alma İşleci (%)

Birinci sayının ikinci sayıya bölümünde, kalan, mod değeridir.

Uygulama

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

int main(){
int x=8,y=4,z=3;
int mod1=x%y;
int mod2=x%z;
int mod3=y%z;
cout << "mod1= " << mod1 << endl;
cout << "mod2= " << mod2 << endl;
cout << "mod3= " << mod3 << endl;
return 0; }
```



3.8.4 Artırma ve Azaltma İşleçleri

(++) işleci, yanındaki değişkenin değerini bir artırır; (--) işleci ise bir azaltır. Söz konusu işleçlerin, değişkenin solunda veya sağında yer alması durumunda anlamı değişir. Artırma işlecini değişkenin solunda kullanırsak:

a=++b

artırma işleci önce <b>nin değerini bir artırır ve sonra <a> değişkenine atar. Bu durumda; <a> ve <b> değişkenlerinin değeri aynı olur. Diğer taraftan, artırma işlecini değişkenin sağında kullanırsak:

a=b++

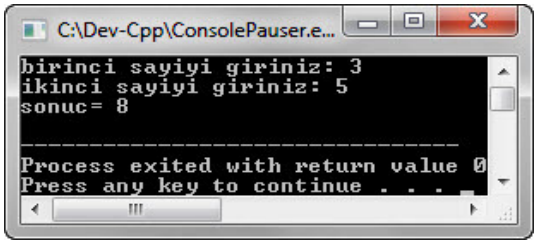
artırma işleci önce <b>nin değerini <a>ya atar; sonra <b>nin değerini bir artırır. Bu durumda; <b>nin değeri <a>nın değerinden bir fazla olur. Artırma ve azaltma işleçlerinin kullanımı konusunu bir tabloda özetleyelim:

Tablo. Artırma ve azaltma işleçlerinin kullanımı.

sağ++	x=10	y=x++	x=11 y=10 olur.
++sol	x=10	y=++x	x=11 y=11 olur.
sağ--	x=10	y=x--	x=9 y=10 olur.
--sol	x=10	y=--x	x=9 y=9 olur.

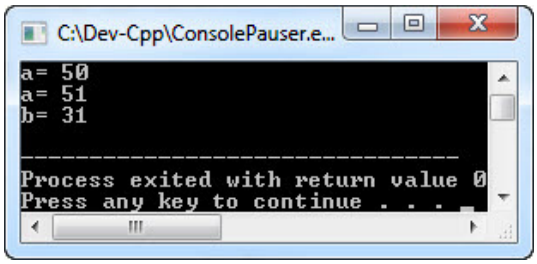
Uygulama

```
// girilen iki sayinin birer eksiginin
carpimi

#include <iostream>
using namespace std;

int main() {
int a,b;
cout << "birinci sayiyi giriniz: ";
cin >> a;
cout << "ikinci sayiyi giriniz: ";
cin >> b;
--a;
--b;
cout << "sonuc= " << a*b << endl;
return 0; }
```



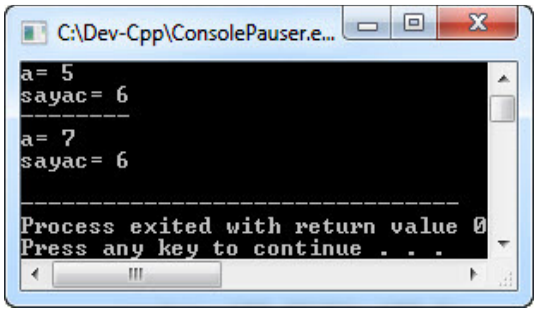
Uygulama

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

// Artirma Islecinin Kullanimi - 1

int main() {
int a=50;
int b=30;
cout << "a= " << a++ << "\n";
cout << "a= " << a << "\n";
cout << "b= " << ++b <<"\n";
return 0; }
```



Uygulama

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

// Artirma Islecinin Kullanimi - 2

int main() {
int sayac; int a;
sayac=5;  a=0;
a=sayac++;
cout << "a= "<< a++ << "\n";
cout << "sayac= " << sayac << "\n";
cout <<"--------"<<"\n";
sayac=5;  a=0;
a=++sayac;
cout << "a= " << ++a << "\n";
cout << "sayac= " << sayac << "\n";
return 0; }
```



3.8.5 Aritmetik Atama İşleçleri

C++'da bazı aritmetik işlemler, alışılagelmişin dışında, kısaltılarak farklı bir biçimde de ifade edilebilir.

Çizelge 3.3. Aritmetik atama işleçleri.

İŞLEÇ	ANLAMI	ARİTMETİK İŞLEM	KISALTILMIŞ İFADE
+=	Topla ve Ata	a=a+b	a+=b
-=	Çıkar ve Ata	a=a-b	a-=b
*=	Çarp ve Ata	a=a*b	a*=b
/=	Böl ve Ata	a=a/b	a/=b
%=	Mod Al ve Ata	a=a%b	a%=b

3.8.6 Karşılaştırma İşleçleri

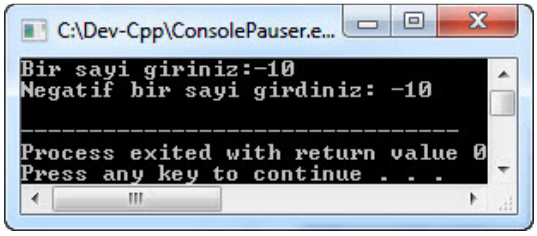
İki sayısal değeri veya iki karakteri karşılaştırmak amacıyla kullanılan işleçlerdir (Çizelge 3). Karşılaştırma yapmak amacıyla **if** deyimi kullanılır. Eğer karşılaştırmanın sonucu "doğru" ise bu deyimin ardından gelen satır işlem görür.; "doğru değil" ise **else** deyiminin ardından gelen satır işlem görür.

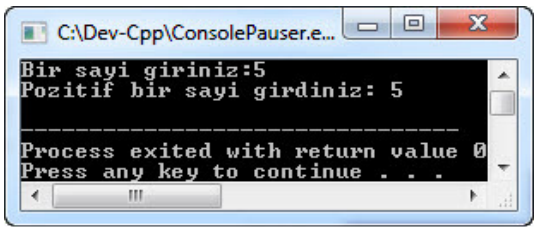
Çizelge 3.4 Karşılaştırma işleçleri.

İŞLEÇ	ANLAMI
>	Büyük
<	Küçük
>=	Büyük ya da Eşit
<=	Küçük ya da Eşit
==	Eşit mi
!=	Eşit Değil mi

Uygulama

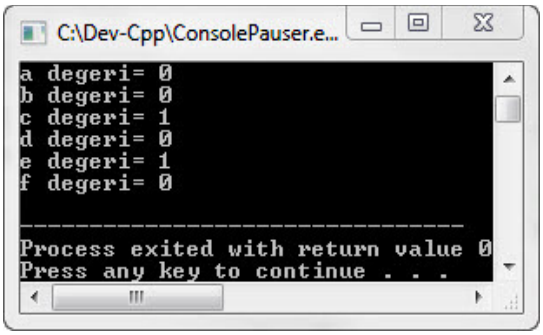
```
#include <iostream>
using namespace std;

// Karsilastirma Islecleri

int main() {
int sayi ;
cout << "Bir sayi giriniz:";
cin >> sayi;
if (sayi<0)
cout << "Negatif bir sayi girdiniz: "
<< sayi << "\n";
else
cout << "Pozitif bir sayi girdiniz: "
<< sayi << "\n";
return 0; }
```





Uygulama

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

int main() {
bool a=(6<3);
bool b=(5>8);
bool c=(10<=10);
bool d=(12>=15);
bool e=(1==1);
bool f=(0!=0);

cout << "a degeri= " << a << endl;
cout << "b degeri= " << b << endl;
cout << "c degeri= " << c << endl;
cout << "d degeri= " << d << endl;
cout << "e degeri= " << e << endl;
cout << "f degeri= " << f << endl;
return 0; }
```



3.8.7 Mantıksal İşleçler

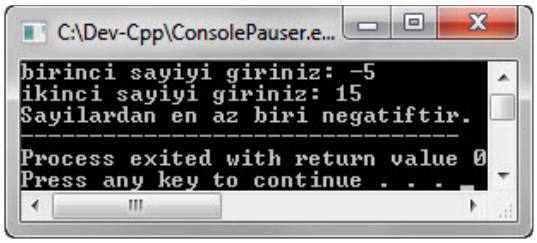
İki veya daha fazla sayıdaki koşulun birlikte sınanması amacıyla kullanılırlar (Çizelge 4). Bu durumda, iki veya daha fazla koşul, doğruluk değerleri göz önüne alırarak birlikte değerlendirilir.

Çizelge 4. Mantıksal işleçler.

İŞLEÇ	ANLAMI	AÇIKLAMA
&&	ve	Verilen ifadelerin tümünün koşulu sağlaması gerekmektedir.
||	veya	Verilen ifadelerin en az biri koşulu sağlamalıdır.
!	değil	Kendisinden sonra gelen ifadenin zıttını alır.

Uygulama

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

// mantiksal islecler

int main() {
int sayi1,sayi2;
cout << "birinci sayiyi giriniz: ";
cin >> sayi1;
cout << "ikinci sayiyi giriniz: ";
cin >> sayi2;
if (sayi1>0 && sayi2>0)
cout << "Her iki sayi da pozitiftir";
else
cout << "Sayilardan en az biri negatiftir.";
return 0; }
```



3.8.8 Özel Amaçlı Ternary İşleci (? :)

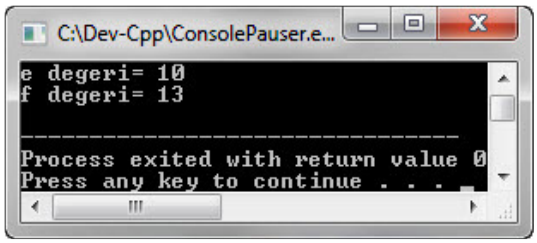
<if> yapısının yaptığı görevi yapmaktadır. Kullanımı şu şekildedir:

<koşul> ? <doğru_ise> : <yanlış_ise>

Koşul gerçekleşirse < : > işaretinin solundaki ifade, koşul gerçekleşmezse < : > işaretinin sağındaki ifade geçerli olur.

Uygulama

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

int main() {
int a=10, b=5, c=25, d=15, e, f;
e=a>b ? a:c;
f=c<d ? 9:13;
cout << "e degeri= " << e << endl;
cout << "f degeri= " << f << endl;
return 0; }
```



3.8.9 İşlemlerin Öncelik Sırası

C++'ın işlemleri düzenli bir şekilde yürütebilmesi amacıyla her bir işlem için bir öncelik sırası tanımlanmıştır. Bu sıralamaya göre en önce parantez içindeki işlemler yapılmakta, en son olarak da eşittir işleciyle işlemler dizisi sonuçlandırılmaktadır. Tablo'da matematiksel işlemlerin, bir anlamda da işleçlerin, öncelik sıraları verilmektedir. Aynı önceliğe sahip işlemlerde öncelik sırası soldan sağa doğrudur.

Tablo. İşlemlerin öncelik sırası.

ÖNCELİK	İŞLEÇ	AÇIKLAMA	OKUNUŞ
1	( )	Parantez	Soldan Sağa
2	++	Sonra Arttır	Soldan Sağa
	--	Sonra Eksilt	
3	++	Önce Arttır	Sağdan Sola
	--	Önce Eksilt	
	!	Değil	
4	*	Çarpma	Soldan Sağa
	/	Bölme	
	%	Mod Alma	
5	+	Toplama	Soldan sağa
	-	Çıkarma	
6	>	Büyüktür	Soldan Sağa
	<	Küçüktür	
	>=	Büyük ya da Eşittir	
	<=	Küçük ya da Eşittir	
7	==	Eşit mi	Soldan Sağa
	!=	Eşit Değil mi	
8	&&	Ve	Soldan Sağa
	||	Veya	
9	=	Atama İşleci	Sağdan Sola

4. PROGRAM DENETİMİ

4.1. KARŞILAŞTIRMA İŞLEMLERİ

C++ programlarının birçok yerinde iki ifadenin karşılaştırılması işlemine başvurmak gerekecektir. Karşılaştırma işlemlerinde karşılaştırma işleçleri kullanılabileceği gibi, mantıksal işleçler de kullanılabilir. Aşağıda, söz konusu işleçlerin kullanıldığı çeşitli karşılaştırma ifadeleri verilmektedir.

sayi>5 (sayi değişkeninin değeri 5'den büyük)

deger!=2 (deger değişkeninin değeri 2'ye **eşit değil**)

sayac==45 (sayac değişkeninin değeri 45'e **eşit**)

sonuc<=eski_sonuc (sonuc değişkeninin değeri eski_sonuc değişkeninin değerinden **küçük ya da eşit**)

sayi>=5 && sayac==45 (sayi değişkeninin değeri 5'den büyük ya da eşit **ve** sayac değişkeninin değeri 45'e eşit)

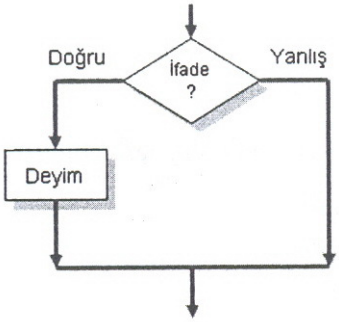
deger!=2 || sonuc==123 (deger değişkeninin değeri 2'ye eşit değil **veya** sonuc değişkeninin değeri 123'e eşit)

4.1.1. <if> YAPISI (Eğer...)

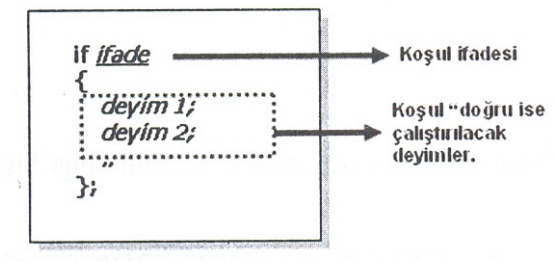
Karşılaştırma işlemi sonucunda bir eylemin yapılması söz konusu ise, diğer bir ifadeyle, belirli deyimlerin çalıştırılması gerekiyorsa if deyimine başvurulur. if deyimi şu şekilde tanımlanır;

if *ifade*
  deyim;

Bu tanıma göre, ifade içinde belirtilen koşulun doğru olması halinde, if içinde belirtilen deyim çalışır; aksi halde deyim işlem görmez.

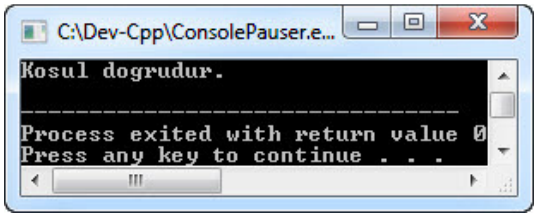


Karşılaştırma işlemi sonucunda birden fazla deyimin çalıştırılması söz konusu ise, bu durumda söz konusu deyimleri "{" ve "}" simgelerini kullanarak gruplandırmamız gerekir.



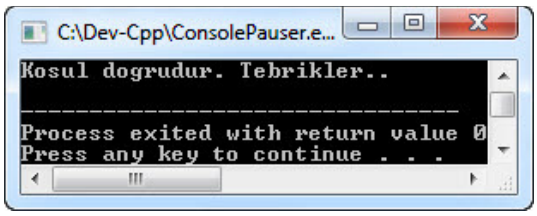
Uygulama

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

int sayi=5;

int main() {
if (sayi<10)
cout << "Kosul dogrudur." << "\n";
return 0; }
```



Uygulama

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

int sayi=5;

int main() {
if (sayi<10)
{
cout << "Kosul dogrudur. ";
cout << "Tebrikler.." << "\n";
}
return 0; }
```

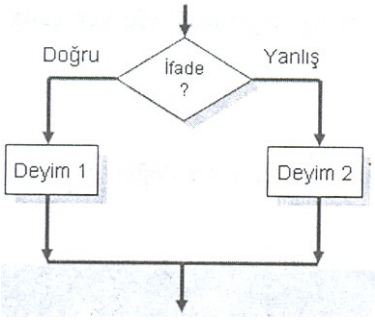


4.1.2. <if - else> YAPISI (Eğer... - Değilse...)

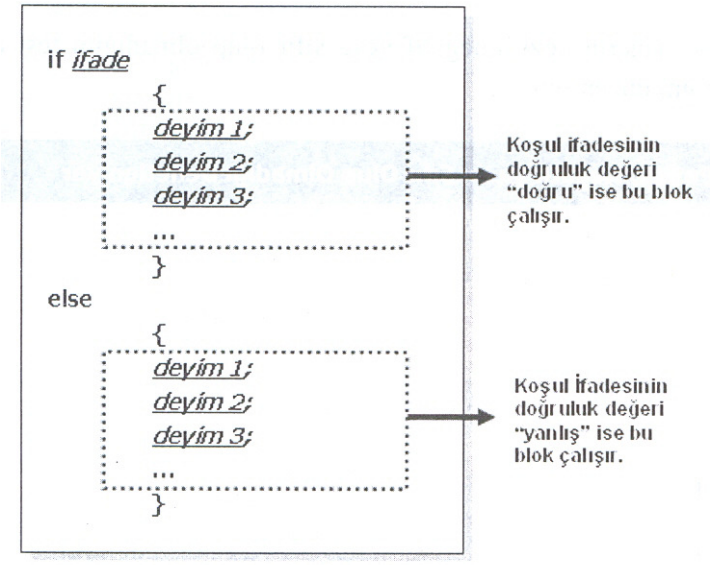
Bir koşulun gerçekleşmemesi durumunda yerine getirilecek eylemleri belirlemek için else sözcüğü kullanılır. Bu sözcük, if deyimi içinde şu şekilde yer alacaktır:

if *ifade*
   deyim1;
else
   deyim2;

Bu tanıma göre, ifade içinde belirtilen koşulun doğru olması halinde, if içinde belirtilen deyim1 çalışır; aksi halde deyim2 işlem görür.

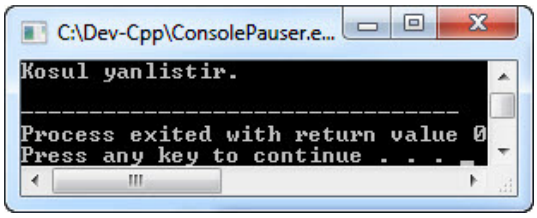


Program içinde else sözcüğünün kullanılması durumunda deyim gruplarına yer verilebilir. Deyim grupları "{" ve "}" işaretleri arasında tanımlanmalıdır. Bu durumda ifade doğru ise ifadeyi izleyen deyim bloğu, aksi halde else sözcüğünü izleyen deyim bloğu çalışır.



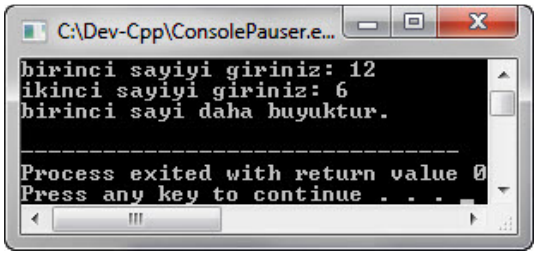
Uygulama

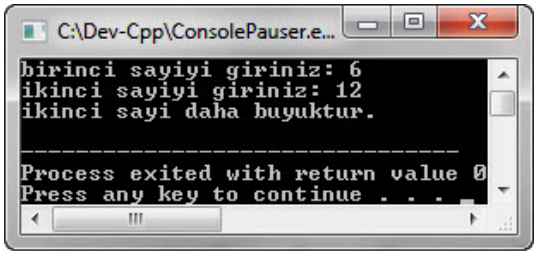
```
#include <iostream>
using namespace std;

int sayi=55;

int main() {
if (sayi<10)
cout << "Kosul dogrudur." << "\n";
else
cout <<"Kosul yanlistir." << "\n";
return 0; }
```



Uygulama

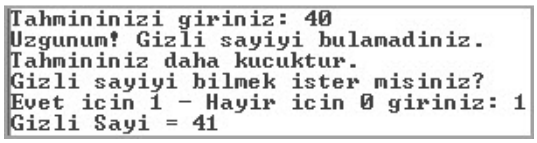
```
#include <iostream>
using namespace std;

main() {
int sayi1,sayi2;
cout << "birinci sayiyi giriniz: ";
cin >> sayi1;
cout << "ikinci sayiyi giriniz: ";
cin >> sayi2;
if (sayi1>sayi2){
cout << "birinci sayi daha buyuktur." << endl;
}
else {
cout << "ikinci sayi daha buyuktur." << endl;
}
return 0; }
```





Uygulama

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

int main() {
int gizli;
int tahmin;
int yanit;
gizli = rand() % 100;

cout << "Tahmininizi giriniz: ";
cin >> tahmin;
if (tahmin==gizli)    {
cout << "Tebrikler! Gizli sayiyi buldunuz. \n";
cout << "Gizli sayi= " << gizli << "\n";
return 0; }
else {
cout << "Uzgunum! Gizli sayiyi bulamadiniz. \n";
if (tahmin>gizli)
cout << "Tahmininiz daha buyuktur. \n";
else cout << "Tahmininiz daha kucuktur. \n"; }
cout << "Gizli sayiyi bilmek ister misiniz?" <<
"\n";
cout << "Evet icin 1 - Hayir icin 0 giriniz: ";
cin >> yanit;
if (yanit==1) cout << "Gizli Sayi = " << gizli <<
"\n";
if (yanit==0) cout << "Program kapaniyor... \n";
return 0; }
```



Uygulama

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

// 5000 TLden Sonra 1000 TL Prim

char adi[22];
char s_adi[22];
float saat;
float top_satis, prim, ucret;

int main(){
cout << "**** Ucret Hesabi **** \n\n";
cout << "Satis elemaninin adini giriniz: ";
cin >> adi;
cout << "Satis elemaninin soyadini giriniz: ";
cin >> s_adi;
cout << "Calistigi toplam saati giriniz: ";
cin >> saat;
ucret=50*(float)saat;
cout << "Sattigi urunler toplamini giriniz: ";
cin >> top_satis;
if(top_satis > 5000)
prim = 1000;
else
prim=0;
cout << "\n**** Odenecek Ucret **** \n";
cout << "\nSatis Elemani: " << adi << " " <<
s_adi << "\n";
cout << "\nMaas Bordrosu:\n";
cout << "--------------\n";
cout << "Ucret= " << ucret << " ve Alacagi Prim=
" << prim << "\n";
cout << "Toplam= " << ucret+prim << " TL
odenecektir." << endl;
return 0;}
```

```
**** Ucret Hesabi ****

Satis elemaninin adini giriniz: Ali
Satis elemaninin soyadini giriniz: Toputut
Calistigi toplam saati giriniz: 48
Sattigi urunler toplamini giriniz: 5500

**** Odenecek Ucret ****

Satis Elemani: Ali Toputut

Maas Bordrosu:
--------------
Ucret= 2400 ve Alacagi Prim= 1000
Toplam= 3400 TL odenecektir.
```

Uygulama

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

// ilk 100 KM'ye kadar 0.5 TL/KM
// 100 KM'den Sonra 0.2 TL/KM

const double yuzalti=0.5;
const double yuzustu=0.2;
int kilometre;
double ucretyuzalti, ucretyuzustu, toplamucret;

int main(){
cout << "Gidilen KM'yi giriniz: ";
cin >> kilometre;
if(kilometre<=100){
ucretyuzalti=kilometre*yuzalti;
ucretyuzustu = 0; }
else{
ucretyuzalti=100*yuzalti;
ucretyuzustu=(kilometre-100)*yuzustu; }
toplamucret=ucretyuzalti+ucretyuzustu;
cout << "\nToplam Ucret= " << toplamucret << "
TL" << "\n"; }
```

```
Gidilen KM'yi giriniz: 105

Toplam Ucret= 51 TL
```

Uygulama

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

// tek sayilarin toplami

int main(){
int n, x, toplam;
x=1; toplam=0;
cout << "n degerini giriniz: ";
cin >> n;

a:
if (x<=n){
toplam=toplam+x;
x=x+2;
goto a;}

cout << n << " sayisina kadar
olan tek sayilarin toplami= "
<< toplam << endl;
return 0;}
```

```
n degerini giriniz: 9
9 sayisina kadar olan tek sayilarin toplami= 25
```

Uygulama

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

int main(){
int x, carp;
carp=1;
cout << "Hangi sayinin faktoryeli
hesaplanacak? ";
cin >> x;

a:
if (x>1){
carp=carp*x;
x--;
goto a;}

cout << "Hesaplanan Faktoryel= " << carp
<< endl;
return 0; }
```

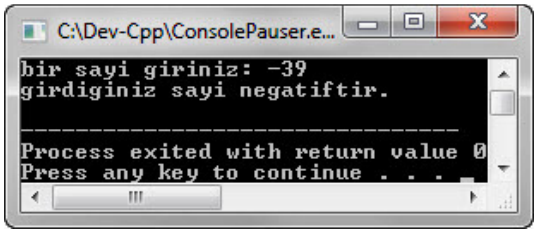
```
Hangi sayinin faktoryeli hesaplanacak? 5
Hesaplanan Faktoryel= 120
```

4.1.3. <if - else if> YAPISI (Eğer... - Değilse Eğer...)

Bazı durumlarda bir if deyimi içinde başka if deyimleri kullanmamız gerekebilir. <else if> olarak adlandırılan bu yapı, kendisinden önce bir <if> ifadesi varsa anlam kazanır.

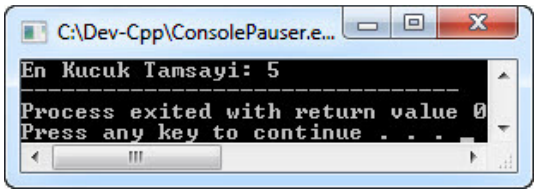
Uygulama

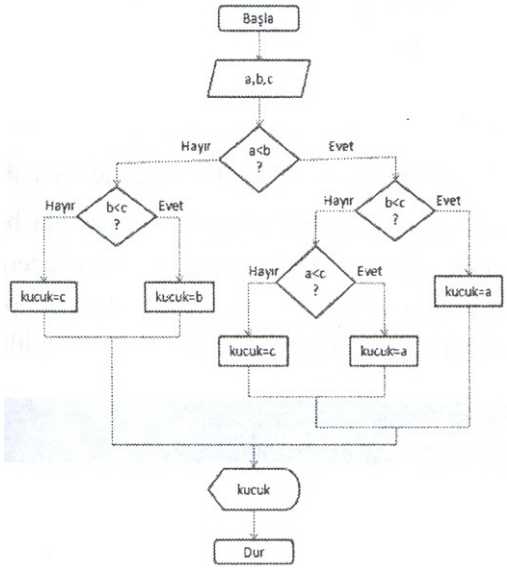
```
#include <iostream>
using namespace std;

int sayi;

int main() {
cout << "bir sayi giriniz: ";
cin >> sayi;
if (sayi>0)
cout << "girdiginiz sayi pozitiftir. ";
else if (sayi<0)
cout << "girdiginiz sayi negatiftir. ";
else
cout << "girdiginiz sayi sifirdir. ";
cout << "\n";
return 0; }
```



Uygulama

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

int main() {
int a=7; int b=5; int c=11; int kucuk;
if (a<b)
if (b<c)
kucuk=a;
else if (a<c)
kucuk=a;
else
kucuk=c;
else if (b<c)
kucuk=b;
else
kucuk=c;
cout << "En Kucuk Tamsayi: " << kucuk;
return 0; }
```





Uygulama

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

int basari;
char harfnot;
main(){
cout << "Basari notunu giriniz: ";
cin >> basari;
if(basari>=90)
harfnot = 'A';
else if (basari>=80)
harfnot = 'B';
else if (basari >=70)
harfnot = 'C';
else if (basari >=60)
harfnot = 'D';
else
harfnot ='F';
cout << "Ogrencinin Harf Notu: " << harfnot
<< endl; }
```

Uygulama

$$ax^2+bx+c=0 \rightarrow \Delta=b^2-4ac$$

$\Delta=0$ ise, eşitliğin tek gerçek kökü vardır.

$$x_1=\frac{-b}{2a}$$

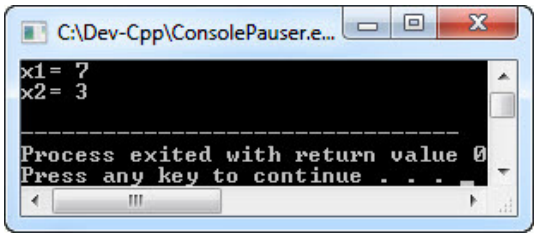
$\Delta>0$ ise, eşitliğin iki gerçek kökü vardır.

$$X_{1,2}=\frac{-b\mp\sqrt{\Delta}}{2a}$$

$\Delta<0$ ise, eşitliğin iki gerçek kökü yoktur.

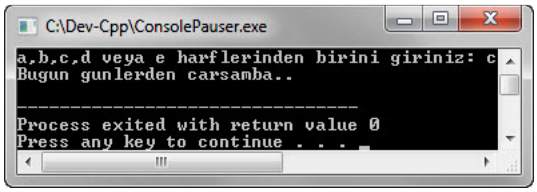
$x^2-10x+21=0$ denkleminin köklerini bulunuz.

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

int main() {
int a=1;
int b=-10;
int c=21;
float x1,x2,delta;
delta=b*b-4*a*c;
if (delta>0.0) {
x1=(-b+sqrt(delta))/2*a;
x2=(-b-sqrt(delta))/2*a;
cout << "x1= " << x1 << "\n";
cout << "x2= " << x2 << "\n"; }
else if (delta==0.0) {
x1=-b/2*a;
cout << "x1= " << x1 <<"\n"; }
else
cout << "Gercek kok yoktur." << "\n";
return 0;
}
```



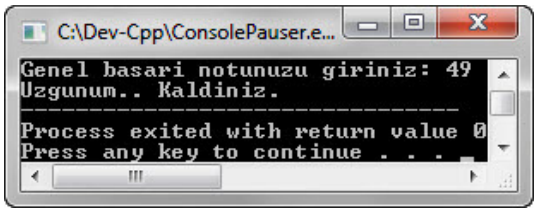
Uygulama

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

int main(){
char harf;
cout << "a,b,c,d veya e harflerinden birini
giriniz: ";
cin >> harf;
if (harf=='a'){
cout << "Bugun gunlerden pazartesi.." << endl; }
else if(harf=='b'){
cout << "Bugun gunlerden sali.." << endl; }
else if(harf=='c'){
cout << "Bugun gunlerden carsamba.." << endl; }
else if(harf=='d'){
cout << "Bugun gunlerden persembe.." << endl; }
else if(harf=='e'){
cout << "Bugun gunlerden cuma.." << endl;}
else {
cout << "Demek ki haftasonu.." << endl; }
return 0; }
```



Uygulama

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

int main(){
int basari;
cout << "Genel basari notunuzu giriniz: ";
cin >> basari;
if (basari>=50){
cout << "Tebrikler.. Gectiniz.";}
else if(basari<50){
cout << "Uzgunum.. Kaldiniz.";}
return 0; }
```



4.2 switch Deyimi

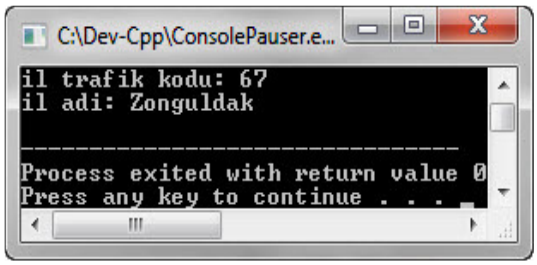
Eğer bir değişkenin değeri belirli sabitlerle karşılaştırılacak ve bunun sonucunda farklı işlemler yapılacak ise if deyimi yerine switch deyimi kullanılabilir.

Kullanım:
switch (*<u>değişken</u>*) {
case *<u>sabit1</u>*:
    <u>deyimler</u>;
    break;
case *<u>sabit2</u>*:
    <u>deyimler</u>;
    break;
...
default:
    <u>deyimler</u>;
}

Bu deyime göre; bir değişkenin değeri sabit1'e eşitse sadece ilgili case bloğundaki deyimler çalışır. Benzer biçimde, değişkenin değeri sabit2'ye eşitse bunu izleyen deyimler işlem görecektir. Eğer değişkenin değeri herhangi bir case içinde tanımlı bir sabite eşit değilse default başlıklı blok içinde yer alan deyimler çalışır.

Uygulama

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

int main() {
int kod;
cout << "il trafik kodu: ";
cin >> kod;
cout << "il adi: ";

switch(kod) {

case 6:
  cout << "Ankara"; break;
case 34:
  cout << "Istanbul"; break;
case 67:
  cout << "Zonguldak"; break;

default:
  cout << "Diger bir ilimiz"; }

cout << "\n"; }
```



Uygulama

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

int kod;

int main() {
cout << "Basinc kodunu giriniz: ";
cin >> kod;
switch (kod)   {
case 8:
cout << "Basinc Yuksek ; Sistemi kapat!"
<< endl; break;
case 7:
cout << "Sicakligi dusur ; Surekli kontrol et!"
<< endl; break;
case 6:
cout << "Dikkat ; Her 5 dakikada bir kontrol et!"
<< endl; break;
default:
cout << "Calisma kosullari normal." << endl; break; }
return 0; }
```

```
Basinc kodunu giriniz: 5
Calisma kosullari normal.
```

Uygulama

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

int basari;
double sayi1;
double sayi2;
int islem;

int main() {
cout << "Dort Islem Makinesi" << endl;
cout << "Birinci sayiyi giriniz: ";
cin >> sayi1;
cout << "Ikinci sayiyi giriniz: ";
cin >> sayi2;
cout << "Islem seciniz: (1=+, 2=-, 3=*, 4=/) ";
cin >> islem;
switch(islem){
case 1:
cout << "Toplama Sonucu= " << sayi1+sayi2
<< endl; break;
case 2:
cout << "Cikarma Sonucu= " << sayi1-sayi2
<< endl; break;
case 3:
cout << "Carpma Sonucu= " << sayi1*sayi2
<< endl; break;
case 4:
cout << "Bolme Sonucu= " << sayi1/sayi2
<< endl; break;
default:
cout << "Hatali sayi girdiniz." << endl; }
return 0; }
```

```
Dort Islem Makinesi
Birinci sayiyi giriniz: 4
Ikinci sayiyi giriniz: 4
Islem seciniz: (1=+, 2=-, 3=*, 4=/)4
Bolme Sonucu= 1
```

Uygulama

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

// defter='d' ; kalem='k' ; silgi='s'

int main(){
char ch;
float fiyat;

const float fiyatD=345.68;
const float fiyatK=123.45;
const float fiyatS=567.89;

cout << "Aldiginiz malin kodunu giriniz: ";
cin >> ch;
cout << endl;
switch(ch){

case 'D':
case 'd':
cout << "Urun: Defter" << endl;
cout << "Fiyati: " << fiyatD << " TL"<< endl;
break;

case 'K':
case 'k':
cout << "Urun: Kitap" << endl;
cout << "Fiyati: " << fiyatK << " TL" << endl;
break;

case 'S':
case 's':
cout << "Urun: Silgi" << endl;
cout << "Fiyati: " << fiyatS << " TL" << endl;
break;

default:
cout << "Birsey satin almadiniz." << endl;
break; }
cout << "Tesekkur ederiz." << endl;
return 0; }
```

```
Aldiginiz malin kodunu giriniz: s

Urun: Silgi
Fiyati: 567.89 TL
Tesekkur ederiz.
```

Uygulama

```
#include <iostream>
#include <cmath>
using namespace std;
int basari;

int main() {
cout << "Basari notunu giriniz: ";
cin >> basari;
switch (basari/10)
{cout << endl;
case 10: cout << "A" << endl ; break;
case 9: cout << "A-" << endl ; break;
case 8: cout << "B" << endl ; break;
case 7: cout << "B-" << endl ; break;
case 6: cout << "C" << endl ; break;
case 5: cout << "D" << endl ; break;
case 4: case 3: case 2: case 1: case 0: cout <<
"F" << endl ; break;
default: cout << "Gecersiz not!" << endl; }
return 0;}
```

```
Basari notunu giriniz: 89
B
```

4.3. DÖNGÜLER

Programın belirli bölümlerinin defalarca işlem görmesinin gerektiği durumlarda döngülerden yararlanılır. Döngü işlemleri, temel olarak üç şekilde gerçekleşebilmektedir.

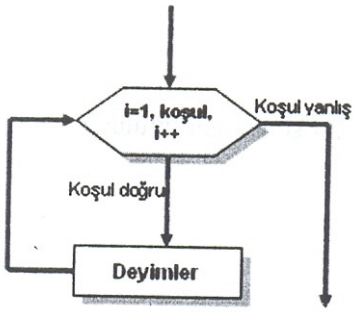
- for döngüsü
- while döngüsü
- do..while döngüsü

4.3.1. Belirli Sayıda Tekrar Döngüsü (for Döngüsü)

Bir ya da daha fazla sayıda deyimin belli bir koşulun gerçekleşmesine dek tekrarlanması söz konusu ise for deyimi kullanılır. Bu deyim şu şekilde tanımlanmaktadır:

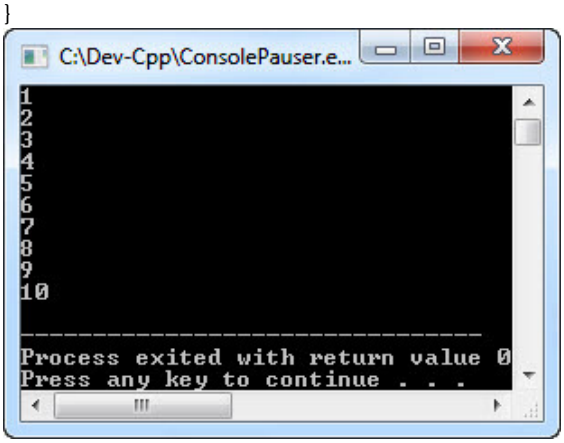
Kullanım:
for (*sayaç; koşul; artma*)
deyimler;

Bu deyime göre; döngü bir sayaca göre yapılacak ve her bir döngü sayılacaktır. Döngü işlemi koşul gerçekleşinceye dek sürecek ve koşul gerçekleştiğinde, diğer bir ifadeyle, doğruluk değeri yanlış olduğunda döngü terk edilecektir.



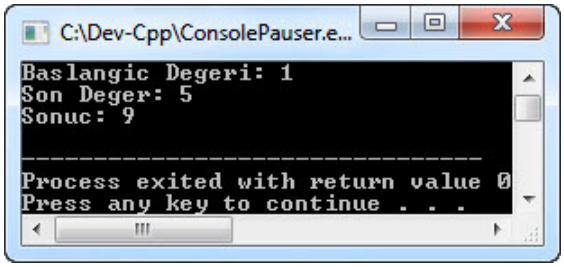
Uygulama

```
// 1den 10a kadar tamsayilar

#include <iostream>
using namespace std;

int i;
int main()
{
    for (i=1;i<=10;i++)
    cout << i << "\n";
    return 0;
}
```



Uygulama

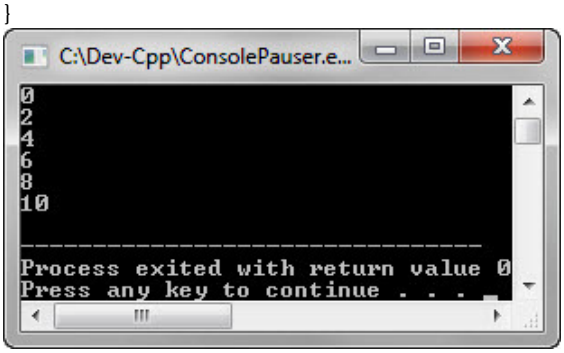
```
// girilen iki tamsayı arasında kalan
// tamsayıların toplamı

#include <iostream>
using namespace std;

int main()
{
    int i,baslangic,son;
    int toplam=0;
    cout << "Baslangic Degeri: ";
    cin >> baslangic;
    cout << "Son Deger: ";
    cin >> son;

// İki deger arasindaki toplam hesaplaniyor...

    for (i=baslangic+1;i<son;i++)
    toplam+=i;
    cout << "Sonuc: " << toplam
         << "\n";
return 0;
```




```
}
```

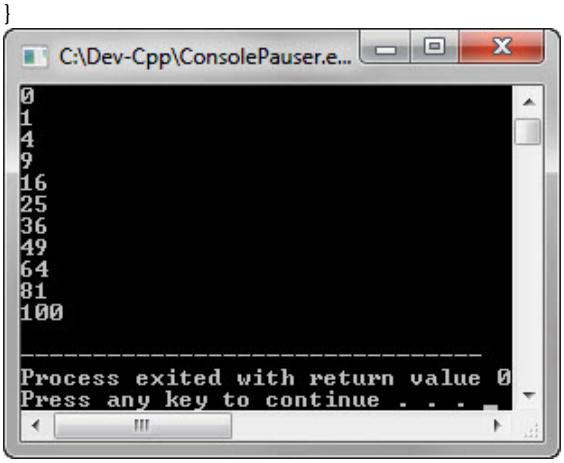
Uygulama

```
// 0dan 10a kadar çift tamsayilar

#include <iostream>
using namespace std;

int main()
{
        int i;
        for(i=0;i<=10;i+=2){
                cout << i << endl;
                }
}
```



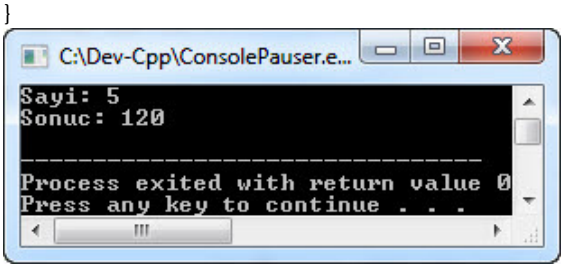
Uygulama

```
// 1den 10a kadar tamsayilarin kareleri

#include <iostream>
using namespace std;
int main()
{
        int i=0;
        for( ;i<=10; ){
                cout << i*i << endl;
                i++;
        }
        return 0;
}
```



Uygulama

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

// verilen bir sayisal
//degerin faktoriyelinin hesaplanmasi

int main ()
{
    int i,sayi;
    int faktoriyel=1;
    cout << "Sayi: ";
    cin >> sayi;

// Verilen sayinin faktoriyeli hesaplaniyor..

    for (i=1;i<=sayi;i++)
    faktoriyel*=i;
    cout << "Sonuc: " << faktoriyel
    << "\n";
    return 0;
}
```



4.3.2. Koşullu Döngüler (while Döngüsü ve do...while Döngüsü)

Bir ya da daha fazla sayıda deyimin belli bir koşulun gerçekleşmesine dek tekrarlanmasının belirli bir koşulun gerçekleşmesine bağlı olduğu durumlarda Koşullu Döngüler kullanılır.

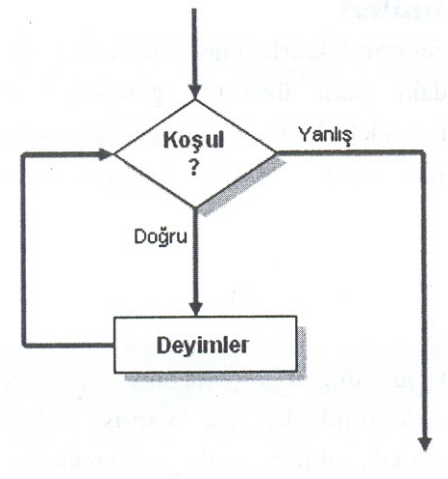
Bu döngüler iki türlüdür. Koşulun döngü başında denetlendiği durumlarda while döngüsü, koşulun döngünün sonunda denetlendiği durumlarda ise do...while döngüsü kullanılır.

4.3.2.1. while Döngüleri (Döngü Başında Denetim)

Bir döngünün çalışmasının, bir koşula bağlı olduğu durumlarda while döngüleri kullanılır. Bu tip döngülerde, döngünün başlaması ve sonrasında da devamı, döngü bloğunun başlangıcında tanımlanan koşulun sağlanıp sağlanmadığına bağlıdır.

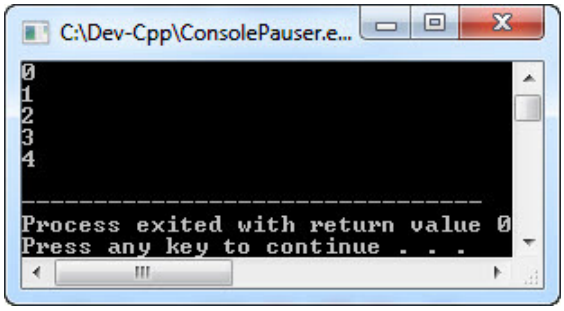
Kullanım:
while (*koşul*)
    deyimler;

while döngüsü içindeki deyimler, tanımlanan koşul "doğru" olduğu sürece çalışır; "yanlış" olduğu anda döngü terk edilir.



Uygulama

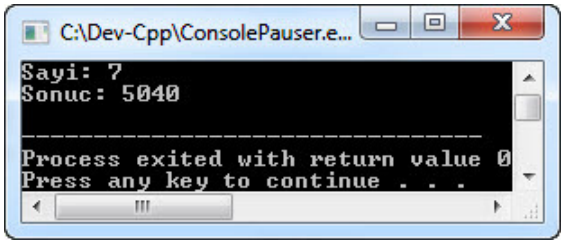
```
/* 5den kucuk olan
tamsayilarin listelenmesi */

#include <iostream>
using namespace std;


int i,n=5;
int main(){
      while(i<n)
      cout << i++ << "\n";
      return 0;
```




```
}
```

Uygulama

```
/* verilen bir sayisal degerin
faktoriyelinin hesaplanmasi */

#include <iostream>
using namespace std;


int main(){
	int i=1,sayi,faktoriyel=1;
	cout << "Sayi: ";
	cin >> sayi;

	while (i<=sayi)
	{
		faktoriyel*=i;
		i++;
	}
	cout << "Sonuc: "
	<< faktoriyel
	<< "\n";
	return 0;
```



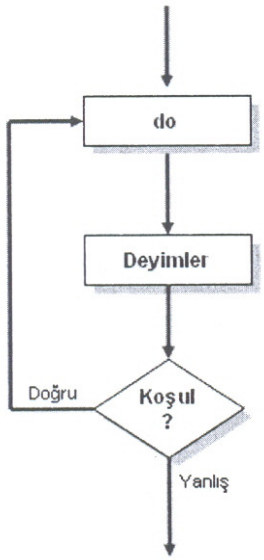

```
}
```

4.3.2.2. do..while Döngüleri (Döngü Sonunda Denetim)

Koşulun, döngü sonunda denetlenmesinin gerektiği durumlarda kullanılır. Bu tür döngülerde, koşul ne olursa oldun, döngü bloğundaki deyimler bir kez işlem görür. Ardından, blok sonunda while ile bir koşul denetimi yapılır. Yapılan denetim sonucunda tanımlanan koşul "doğru" olduğu sürece döngü devam eder; "yanlış" olduğu anda döngü terk edilir.

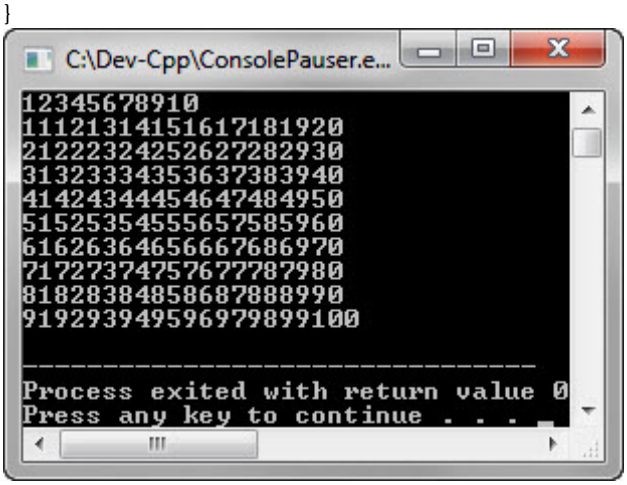
Kullanım:

do {
	deyimler
}while(*koşul*);



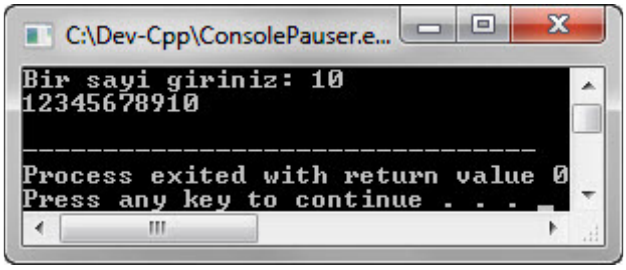
Uygulama

```
// 0dan 100e kadar tamsayilar

#include <iostream>
using namespace std;

int main()
{
        int i=0;
        do{
                cout << ++i;
                if (i%10==0)
                cout << endl;
                }while (i<100);
        return 0;
}
```



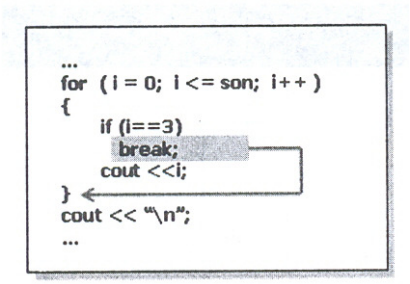
Uygulama

```
// girilen sayiya kadar tamsayilar

#include <iostream>
using namespace std;

int main()
{
	int sayi, i=0;
	cout << "Bir sayi giriniz: ";
	cin >> sayi;
	do{
		i++;
		cout << i;
		}while(i<sayi);
		cout << endl;
}
```

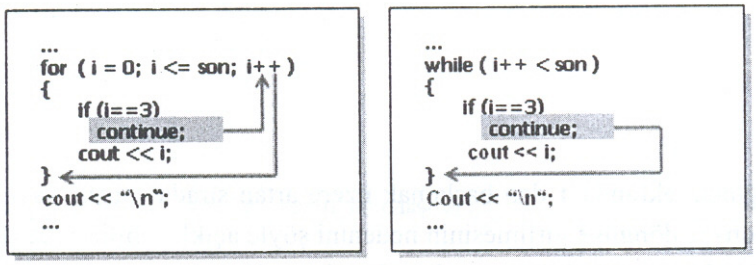


4.3.3. Döngülerden Çıkış (break) ve Devam (continue)

Döngü işleminin tamamlanmadan döngünün sona erdirilmesinin söz konusu olduğu durumlarda break deyimi kullanılır. Döngü içinde break deyimine sıra geldiğinde, döngü sonuna kadar olan tüm deyimler atlanır ve döngü terk edilerek bir sonraki adımdan devam edilir.

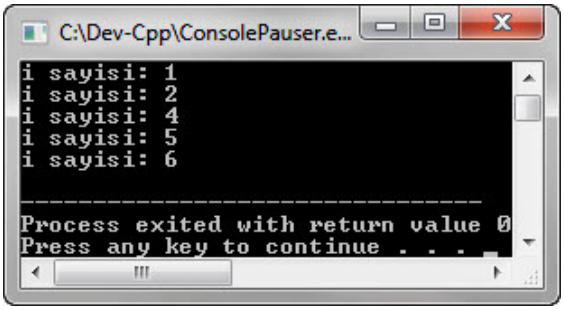


Bir döngüyü terk etmeden bir adımın atlanmasının söz konusu olduğu durumlarda continue deyimi kullanılır. Bu deyim döngünün çalışmasını sona erdirmez; sadece bir sonraki döngü adımına geçilmesini sağlar.



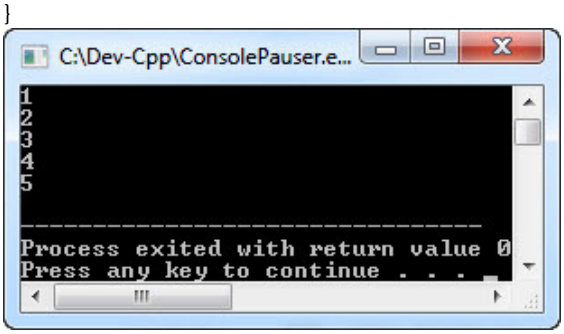
Uygulama

```
// 3 hariç, 0dan 7ye kadar tamsayilar

#include <iostream>
using namespace std;

int main()
{
	int i=0;
	while(++i<=10){
		if(i==3) continue;
		if(i==7) break;
		cout << "i sayisi: " << i << endl;
		}
}
```



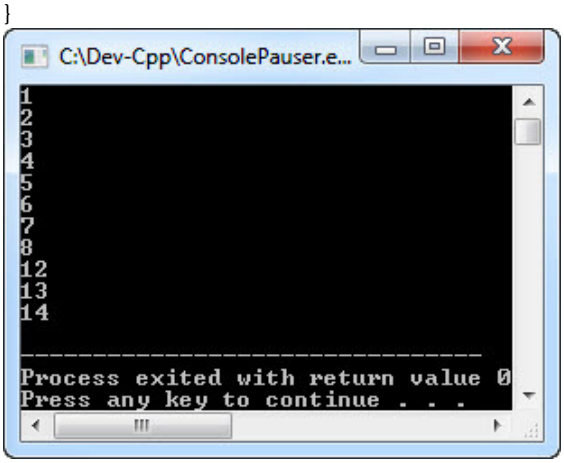
Uygulama

```
// break deyiminin kullanilmasi

#include <iostream>
using namespace std;

int i;
int main()
{
  for (i=1;i<100;i++)
    {
      cout << i << "\n";
      if (i==5)
      break;
    }
}
```



Uygulama

```
// continue deyiminin kullanilmasi

#include <iostream>
using namespace std;

int i;
int main()
{
  for (i=1;i<15;i++)
    {
      if (i>8 && i<12)
      continue;
      cout << i << "\n";
    }
}
```

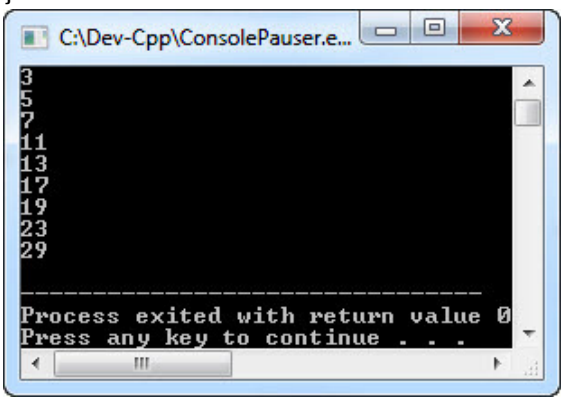


4.3.4. İç İçe Döngüler

Karşı karşıya olduğumuz problemin çözümüne yönelik olarak bir döngünün içinde başka döngüler de kullanabiliriz.

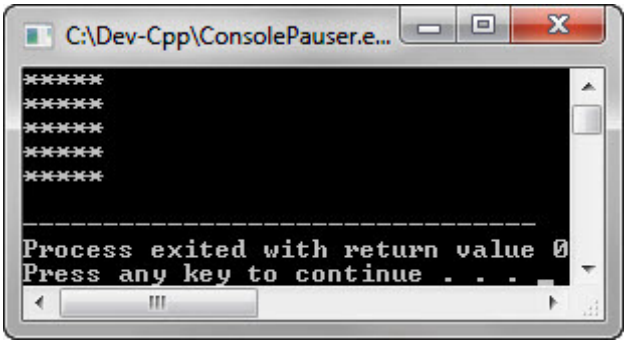
Uygulama

```
/* 1 ile 30 arasindaki
asal sayilar */

#include <iostream>
using namespace std;

main()
{
  int i, n=2;
  while (++n<=30)
    {
      i=1;
      while (++i<n)
      if (n%i==0)
      break;
      if (i==n)
       cout << n
       << "\n";
    }
    return 0;
}
```



Uygulama

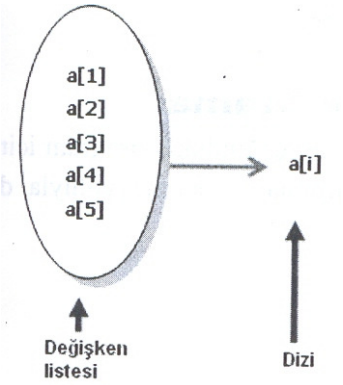
```
// icice dongulerin kullanilmasi

#include <iostream>
using namespace std;

int main()
{
	int i=5,j;
	while(i>=1){
		j=1;
		while(j<=5){
			cout << "*";
			j++;
		}
		cout << endl;
		i--;

	}
}
```



5. DİZİLER

Diziler, verileri bellekte saklayan ve gerektiğinde program içinde kullanan veri yapılarıdır. Diziler, bir boyutlu [a(i)] olarak düzenleyebileceğimiz gibi, gerektiğinde tıpkı bir matematiksel matris gibi çok boyutlu [a(i), b(i)] olarak da tanımlayabiliriz.

Dizi, belirli sayıda verinin bellekte saklandığı değişken listeleridir. Örneğin a(1), a(2) ve a(3) gibi herbiri birbirinden farklı değişkenlerden oluşan bir değişken listesini a(i) ismiyle bir dizi haline getirebilir ve program içinde ortak biçimde kullanabiliriz.



Bir grup sayısal değer ya da karakter verisini içeren diziler, 'Bir Boyutlu Diziler' olarak adlandırılır ve aşağıdaki şekilde tanımlanır:

tür *dizi adı* [*boyut*]

tür: Dizinin içerdiği değerlerin veri türü. Aynen değişken türlerinin tanımlandığı biçimde kullanılır.
dizi adı: Program içinde dizinin tüm elemanları bu ortak isim ile temsil edilir.
boyut: Dizi elemanları için bellekte ayrılacak yeri belirler. Ayrılan yerin tümüyle dolması gerekmez; örneğin 10 elemanlık bir boyuta sahip bir dizinin 3 elemanı olabilir.

Adı <sayilar> olan ve 12 adet tamsayıdan oluşan bir diziyi şöyle tanımlayabiliriz:

int sayilar[12];

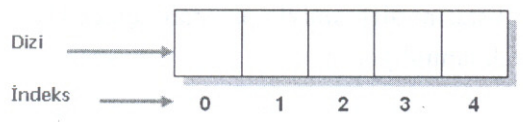
Adı <x> olan ve 18 adet ondalıklı sayıdan oluşan bir diziyi şöyle tanımlayabiliriz:

double x[18];

Adı <harfler> olan ve 6 adet karakterden oluşan bir diziyi şöyle tanımlayabiliriz:

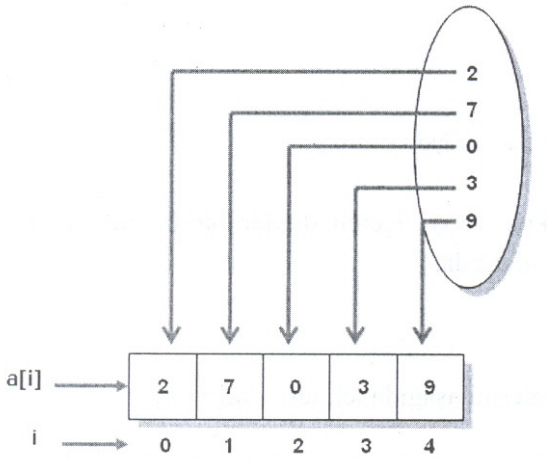
char harfler[6];

Diziler, indeksleri yardımıyla kullanılır. İndeks, bir dizinin her bir elemanına sırayla verilen bir numaradır. İndeksler, sıfırdan başlayarak oluşturulur; buna göre dizinin birinci elemanının indeksi 0 (sıfır), ikinci elemanının indeksi ise 1 (bir)'dir.



5.1 Dizilere Başlangıç Değeri Atama

Bir dizinin içereceği değerler çoğunlukla program içinde yapılan hesaplamalar sonucu elde edilir. Ancak, bazı durumlarda dizinin içini doğrudan atama yoluyla da doldurabiliriz.



Bir diziye başlangıç değeri vermek için, ilgili değişkene o değeri doğrudan atayabiliriz. Örneğin;

a[0]=2;

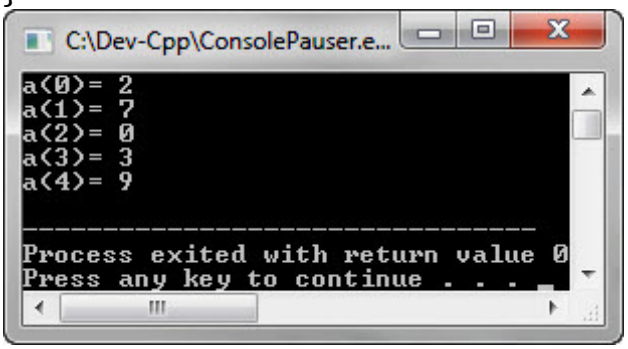
biçiminde bir tanımlama yaptığımızda, <a> isimli bir dizinin ilk elemanı olarak <2> değerini atamış oluruz. Dizilere aynı anda birden fazla değer de atayabiliriz. Bunun için söz konusu değerleri { } işaretleri arasına yazmamız gerekir. Örneğin;

a[5]={2,7,0,3,9};

biçiminde yazarak da diziye başlangıç değerleri atayabiliriz.

Uygulama

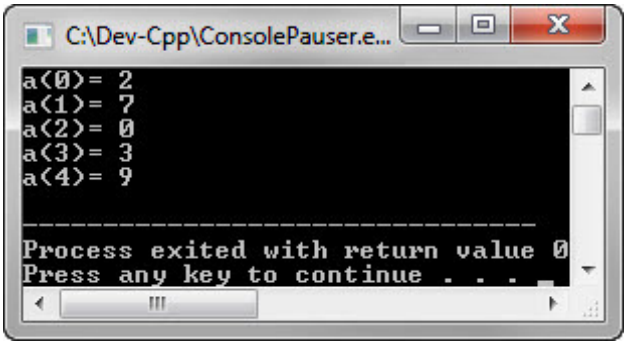
```
// bir diziye baslangic degerleri atanmasi

#include <iostream>
using namespace std;

int a[5]={2,7,0,3,9};

int main(){
        cout << "a(0)= " << a[0] << endl;
        cout << "a(1)= " << a[1] << endl;
        cout << "a(2)= " << a[2] << endl;
        cout << "a(3)= " << a[3] << endl;
        cout << "a(4)= " << a[4] << endl;
}
```



Yukarıdaki uygulama için <for> döngüsü kurarak da aynı sonuca ulaşabiliriz:

Uygulama

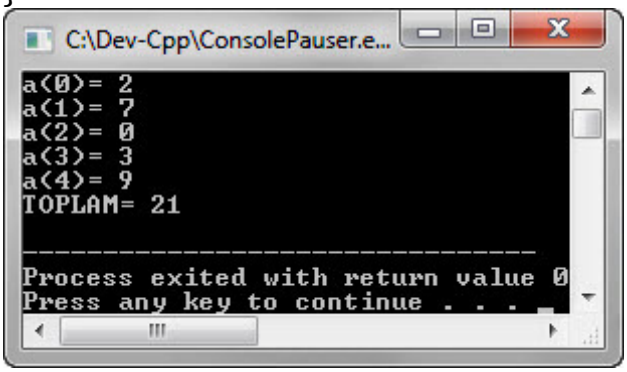
```
// dizi degerlerinin for dongusu
// yardimiyla goruntulenmesi

#include <iostream>
using namespace std;

int i;
int a[5]={2,7,0,3,9};

int main(){
        for (i=0;i<=4;i++)
        cout << "a(" << i << ")= " << a[i] << endl;
        return 0;
}
```



Bu dizideki sayıları toplayalım:

Uygulama

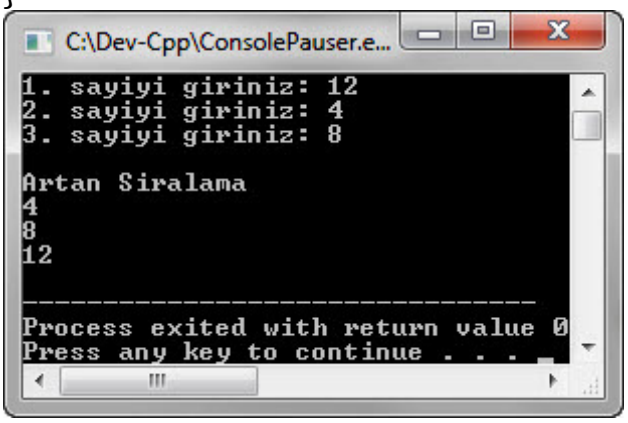
```
// dizi degerlerinin toplanmasi

#include <iostream>
using namespace std;

int main(){
	int a[5]={2,7,0,3,9};
		for (int i=0;i<=4;i++)
		cout << "a(" << i << ")= " << a[i] << endl;
	int toplam=0;
		for(int j=0;j<=4;j++){
		toplam=toplam+a[j]; // toplam+=a(j) de olabilir
	}
	cout << "TOPLAM= " << toplam << endl;
	return 0;
}
```



Klavyeden girilen üç sayıyı bir diziye atayalım ve bu sayıları küçükten büyüğe doğru sıralayalım:

Uygulama

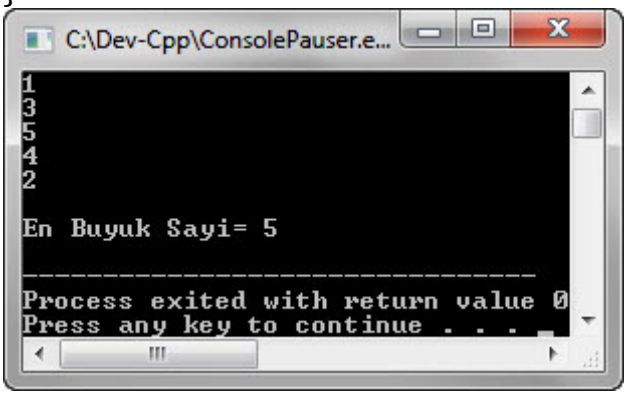
```
// kucukten buyuge siralama

#include <iostream>
using namespace std;

int sayi[3];
int i,b,gecici;
main(){
	do
	{
		cout << i+1 << ". sayiyi giriniz: ";
		cin >> sayi[i];
		i++;
	}
	while(i<3);
	for (i=0;i<2;i++)
	{
		for(b=i+1;b<3;b++)
		{
			if (sayi[i]>sayi[b])
			{
				gecici=sayi[i];
				sayi[i]=sayi[b];
				sayi[b]=gecici;
			}
		}
	}
	cout << "\n";
	cout << "Artan Siralama" << "\n";
	for(i=0;i<3;i++)
	{
		cout << sayi[i] << "\n";
	}
}
```



Uygulama

Klavyeden girilen beş sayının en büyüğünü bulalım:

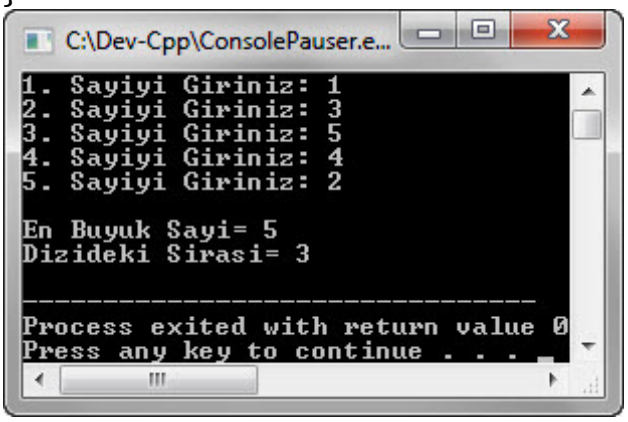
```
// en buyugunu bulma
#include <iostream>
using namespace std;

int i,enb;
int a[5];

main(){
for(i=0;i<5;i++)
cin >> a[i];
enb=a[0];
for (i=1;i<5;i++)
if(enb<a[i]) enb=a[i];
cout << endl;
cout << "En Buyuk Sayi= " << enb << endl;
return 0;
}
```



Uygulama

Klavyeden girilen beş sayının en büyüğünü ve dizideki sırasını bulalım:

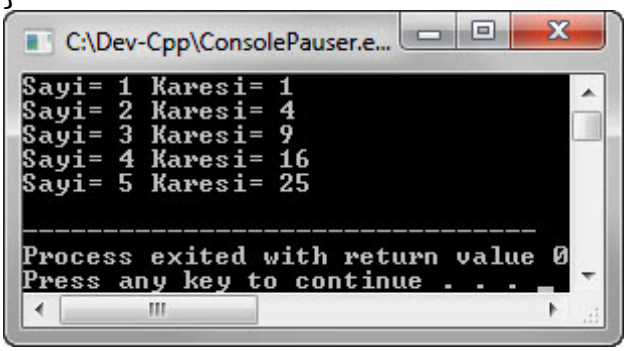
```
// en buyugunu ve sirasini bulma
#include <iostream>
using namespace std;

int i,enb,sira;
int dizi[5];

main(){
for(i=1;i<=5;i++){
cout << i << ". Sayiyi Giriniz: ";
cin >> dizi[i];
}
enb=dizi[0];
for (i=1;i<=5;i++){
if (dizi[i]>enb){
enb=dizi[i];
sira=i;
}
}
cout << endl;
cout << "En Buyuk Sayi= " << enb << endl;
cout << "Dizideki Sirasi= " << sira << endl;
return 0;
}
```



Uygulama

1'den 5'e (5 dahil) kadar olan sayıların karelerini hesaplayalım.

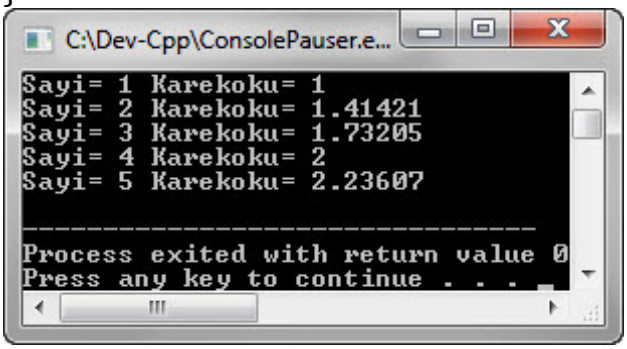
```
// 1-5 karelerini bulma
#include <iostream>
using namespace std;

int i;
int sayi[5];

main (){
i=0;
for(i=1;i<=5;i++){
sayi[i]=i*i;
cout << "Sayi= " << i << " Karesi= " << sayi[i] << endl;
}
return 0;
}
```



Uygulama

1'den 5'e (5 dahil) kadar olan sayıların kareköklerini hesaplayalım.

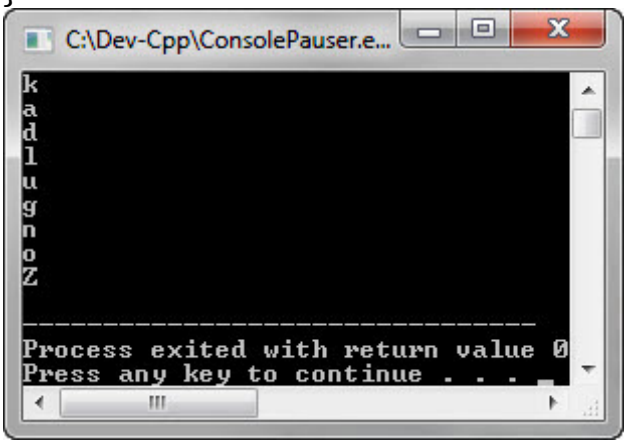
```
// 1-5 karekoklerini bulma

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

int i;
float sayi[5];

main (){
i=0;
for(i=1;i<=5;i++){
sayi[i]=sqrt(i);
cout << "Sayi= " << i << " Karekoku= " << sayi[i] << endl;
}
return 0;
}
```



Uygulama

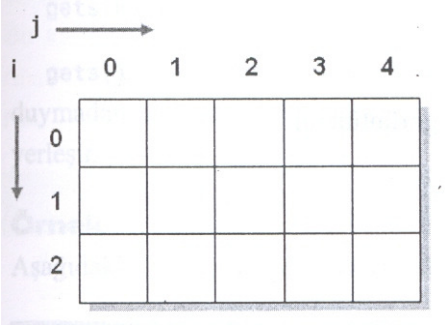
<Zonguldak> kelimesindeki harfleri sırasıyla bir diziye atayalım ve ekran çıktısında Zonguldak kelimesini tersten yazdıralım:

```
//bir karakter dizisi olusturulmasi

#include <iostream>
using namespace std;

int main(){
        char harf[9]={'Z','o','n','g','u','l','d','a','k'};
        for (int i=8;i>=0;i--){
                cout << harf[i] << endl;
                }
                return 0;
}
```



5.2 İki Boyutlu Diziler

İki boyutlu bir diziyi, bir tablo veya bir matris şeklinde düşünebiliriz. Bu durumda iki farklı indeks kullanmamız gerekecektir. Birinci indeksle satır elemanlarını, ikinci indeksle ise sütun elemanlarını tanımlayabiliriz. Aşağıdaki şekil iki boyutlu bir diziyi göstermektedir:



İki boyutlu bir diziyi şu şekilde tanımlayabiliriz:,

tür *dizi adı* [*boyut1*] [*boyut1*]

Örneğin tamsayı değerler içeren, adı <dizi> olan, 5 satır ve 2 sütundan oluşan bir diziyi şu şekilde tanımlayabiliriz:

int dizi[5][2]

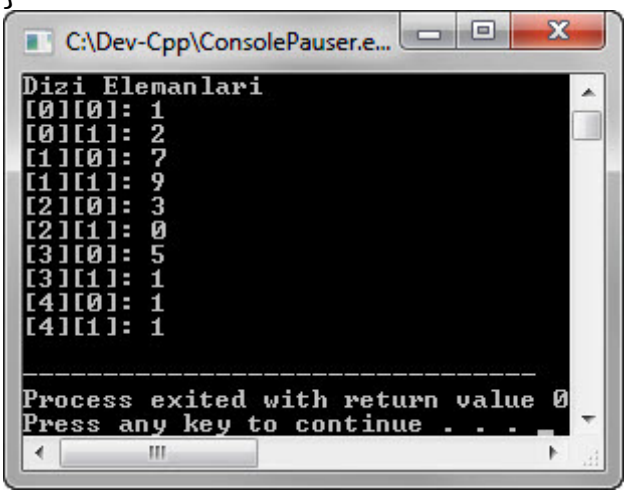
Aşağıda, 5 satır ve 2 sütundan oluşan bir tablo verilmektedir.

SATIR/SÜTUN	0	1
0	1	2
1	7	9
2	3	0
3	5	1
4	1	1

Bu tablodaki değerleri, iç içe for döngüleri kullanarak programa bir dizi olarak tanıtmak ve dizi elemanlarını ekranda yazdırmak için şöyle bir program hazırlayabiliriz:

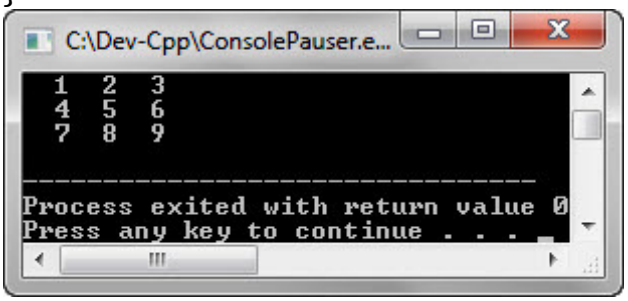
Uygulama

```
// iki boyutlu dizi hazirlanmasi-1

#include <iostream>
using namespace std;

int i;
int dizi[5][2]={{1,2}, {7,9}, {3,0}, {5,1}, {1,1}};

int main(){
	cout << "Dizi Elemanlari" << endl;
	for (int i=0;i<5;i++)
	for (int j=0;j<2;j++){
		cout << "[" << i << "][" << j << "]: ";
		cout << dizi[i][j] << endl;
	}
	return 0;
}
```



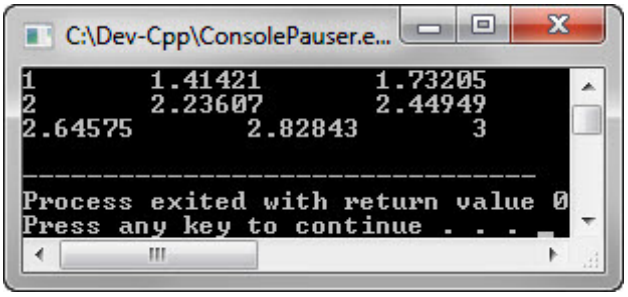
Uygulama

```
// iki boyutlu dizi hazirlanmasi-2
#include <iostream>
using namespace std;

int a[3][3]={1,2,3,4,5,6,7,8,9};
int i,j;

main (){
for(i=0;i<3;i++){
for(j=0;j<3;j++)
cout << "  " << a[i][j];
cout << endl;
}
return 0;
}
```



Uygulama

```
// iki boyutlu dizi icindeki degerlerin
// karekoklerinin hesaplanmasi

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

float a[3][3]={1,2,3,4,5,6,7,8,9};
int i,j;

main (){
for(i=0;i<3;i++){
for(j=0;j<3;j++)
cout << sqrt(a[i][j]) << "      ";
cout << endl;
}
return 0;
}
```



Uygulama

Aşağıda 2 satır ve 3 sütundan oluşan <a> ve <b> matrisleri verilmektedir. Bu matrislerin toplamı olan <c> matrisini hesaplayan bir program yazalım:

a Matrisi:

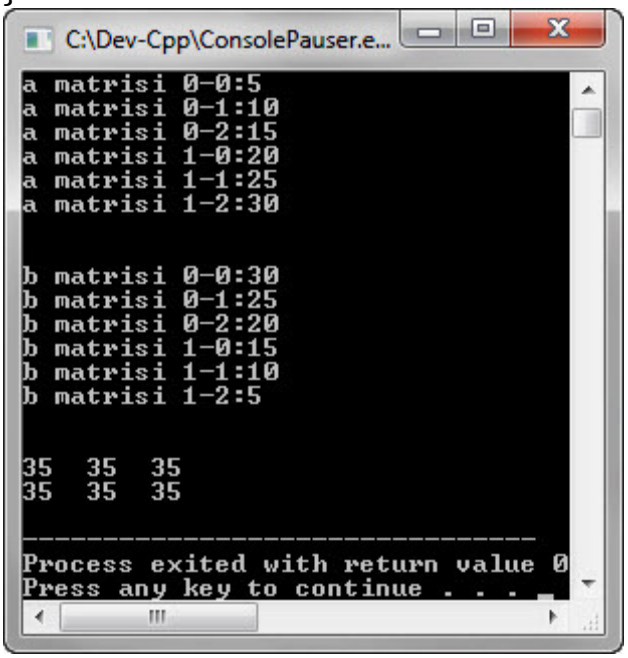
SATIR/SÜTUN	0	1	2
0	5	10	15
1	20	25	30

b Matrisi:

SATIR/SÜTUN	0	1	2
0	30	25	20
1	15	10	5

```
// iki matrisin toplami

#include <iostream>
using namespace std;

int main(){
        int a[2][3], b[2][3], c[2][3];
        int i,j;
        for(i=0;i<2;i++){
                for(j=0;j<3;j++){
                cout << "a matrisi " << i << "-" << j << ":";
                cin >> a[i][j];
                }
        }
        cout << "\n\n";
        for(i=0;i<2;i++){
                for(j=0;j<3;j++){
                cout << "b matrisi " << i << "-" << j << ":";
                cin >> b[i][j];
                }
        }
        for(i=0;i<2;i++){
                for(j=0;j<3;j++){
                c[i][j]=a[i][j]+b[i][j];
                }
        }
        cout << "\n\n";
        for(i=0;i<2;i++){
                for(j=0;j<3;j++){
                cout << c[i][j] << "  ";
        }
```

```
        cout << endl;
}
return 0;
}
```



Uygulama

Aşağıda 3 satır ve 3 sütundan oluşan <a> ve <b> matrisleri verilmektedir. Bu matrislerin çarpımı olan <c> matrisini hesaplayan bir program yazalım:

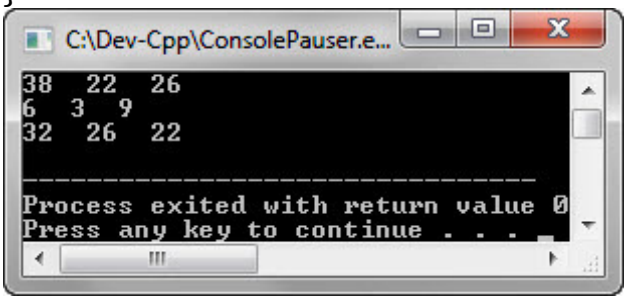
a Matrisi:

SATIR/SÜTUN	0	1	2
0	5	7	9
1	0	3	0
2	7	5	1

b Matrisi:

SATIR/SÜTUN	0	1	2
0	3	3	1
1	2	1	3
2	1	0	0

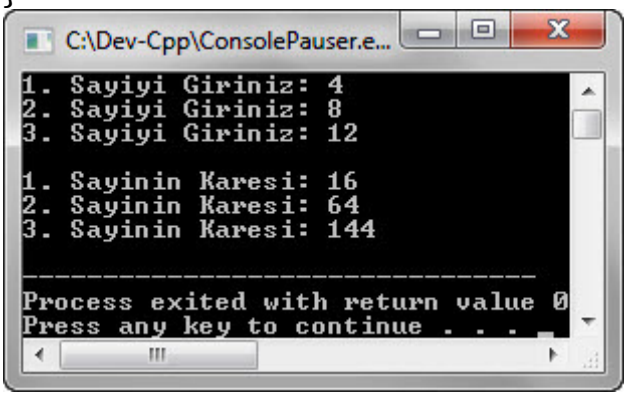
```
// iki matrisin carpimi

#include <iostream>
using namespace std;

int a[3][3]={5,7,9,0,3,0,7,5,1};
int b[3][3]={3,3,1,2,1,3,1,0,0};
int c[3][3];
int i,j,k;
int top;

int main(){
for(i=0;i<3;i++){
for(j=0;j<3;j++){
top=0;
for (k=0;k<3;k++){
top+=a[i][k]*b[k][j];
}
c[i][j]=top;
}
}
for(i=0;i<3;i++){
for(j=0;j<3;j++)
cout << c[i][j] << "  ";
cout << endl;
}
return 0;
}
```



Uygulama

Klavyeden girilen sayıları 2 satır ve 3 sütundan oluşan bir matrisin birinci satırına, bu sayıların karelerini de ikinci satırına yerleştiren bir program yazalım.

```
// birinci satirin karelerinin
// ikinci satıra yazilmasi

#include <iostream>
using namespace std;

int j, dizi[2][3];
main(){
for(j=0;j<3;j++){
cout << j+1 << ". Sayiyi Giriniz: ";
cin >> dizi[0][j];
dizi[1][j]=dizi[0][j]*dizi[0][j];
}
cout << endl;
for(j=0;j<3;j++){
cout << j+1 << ". Sayinin Karesi: " << dizi[1][j];
cout << endl;
}
return 0;
}
```



HAZIR FONKSİYONLAR

C++ derleyicileri çok kullanılan bazı fonksiyonları içerirler. Bu fonksiyonlar derleyici dosyalarında kütüphaneler şeklinde saklanmaktadır. Bu kütüphane fonksiyonları alanlarına göre gruplanarak kütüphane dosyaları şeklinde saklanmaktadır. Kullanılacağı yerlerde ilgili kütüphane dosyası programa dahil edilerek fonksiyon çağırılabilir.

Karakter İşleme Fonksiyonları

Karakterler üzerinde işlem yapan fonksiyonların bulunduğu kütüphanedir. Bu fonksiyonları kullanabilmek için cctype kütüphanesini programa tanıtmalıyız.
#include <cctype>

Dizgi İşleme Fonksiyonları

Karakterlerin bir araya gelerek oluşturduğu diziler 'dizgi' olarak adlandırılır. Dizgiler üzerinde işlem yapan fonksiyonları kullanabilmek için cstring kütüphanesini programa tanıtmalıyız.
#include <cstring>

Zaman ve Tarih Fonksiyonları

Zaman ve tarihle ilgili fonksiyonları kullanabilmek için ctime kütüphanesini programa tanıtmalıyız.
#include <ctime>

Genel Amaçlı Fonksiyonlar

Tamsayı mutlak değer [int abs(x)], bölme işleminde bölüm ve kalan [div(x,y)], rastgele değer üretme [randomize(x)], en küçük sayıyı bulma [min(x,y)], en büyük sayıyı bulma [max(x,y)] gibi çeşitli konularla ilgili fonksiyonlar cstdlib kütüphanesinde saklanırlar. Bu fonksiyonları kullanabilmek için cstdlib kütüphanesini programa tanıtmalıyız.
#include <cstdlib>

MATEMATİK FONKSİYONLARI

Sayılar üzerinde matematiksel işlem yapan fonksiyonlar cmath kütüphanesinde saklanırlar. Matematiksel fonksiyonlar double türündedir. Bu yüzden bu fonksiyonlara parametre olarak verilen değişkenler de double türünde olmalıdır. Önemli matematiksel fonksiyonlar aşağıda tanıtılmaktadır.

FONKSİYON	TANIM
sin(x)	Radyan olarak verilen x açısının sinüsünü hesaplar.
cos(x)	Radyan olarak verilen x açısının kosinüsünü hesaplar.
tan(x)	Radyan olarak verilen x açısının tanjantını hesaplar.
sinh(x)	Radyan olarak verilen x açısının hiperbolik sinüsünü hesaplar.
cosh(x)	Radyan olarak verilen x açısının hiperbolik cosinüsünü hesaplar.
tanh(x)	Radyan olarak verilen x açısının hiperbolik tanjantını hesaplar.
asin(x)	x değerinin ters sinüsünü hesaplar.
acos(x)	x değerinin ters cosinüsünü hesaplar.
atan(x)	x değerinin ters tanjantını hesaplar.

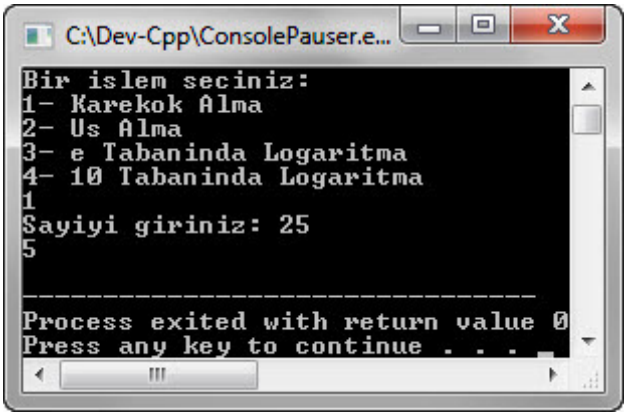
atan2(x)	y/x'in ters tanjantını hesaplar.
pow(x,y)	x'in y kuvvetini hesaplar.
pow(x,(1/y))	x'in y kökünü hesaplar.
sqrt(x)	x'in karekökünü hesaplar.
log(x)	ln(x)'i yani x'in e tabanındaki doğal logaritmasını hesaplar.
log10(x)	log(x)'i yani x'in 10 tabanındaki logaritmasını hesaplar.
exp(x)	e sayısının x kuvvetini hesaplar.
fabs(x)	Ondalıklı x'in mutlak değerini hesaplar.
floor(x)	x'ten küçük olan en büyük tamsayıyı bulur.
ceil(x)	x'ten büyük olan en küçük tamsayıyı bulur.

Önemli Bir Not ve Uyarı:

Trigonometrik fonksiyonlarda verilen ifadelerin değerini hesaplayabilmek için söz konusu değerleri radyana çevirmek gerekmektedir. Derece olarak verilen bir x ifadesinin radyan cinsinden değerini bulmak için $\frac{\pi}{180}x$ dönüşümü yapılmalıdır.

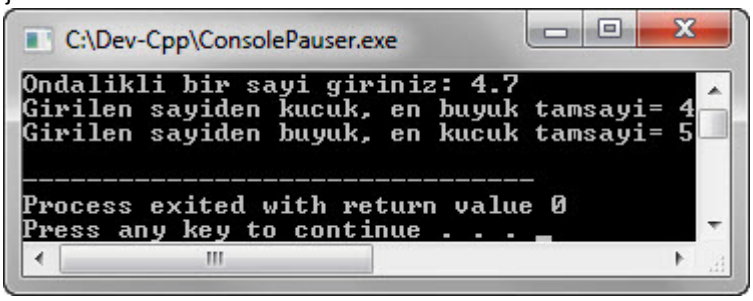
Uygulama

```
// karekok, us alma ve log

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

int main(){
	int secim;
	double i,j;
	cout << "Bir islem seciniz:" << endl;
	cout << "1- Karekok Alma" << endl;
	cout << "2- Us Alma" << endl;
	cout << "3- e Tabaninda Logaritma" << endl;
	cout << "4- 10 Tabaninda Logaritma" << endl;
	cin >> secim;

	if(secim==1){
		cout << "Sayiyi giriniz: ";
		cin >> i;
		cout << sqrt(i) << endl;
	}
	else if(secim==2){
		cout << "Taban sayisini giriniz: ";
		cin >> i;
		cout << "Us sayisini giriniz: ";
		cin >> j;
		cout << pow(i,j) << endl;
	}
	else if(secim==3){
		cout << "Sayiyi giriniz: ";
		cin >> i;
		cout << log(i) << endl;
	}
	else if(secim==4){
		cout << "Sayiyi giriniz: ";
		cin >> i;
		cout << log10(i) << endl;
	}
	else cout << "Hatali secim yaptiniz.";
	return 0;
}
```



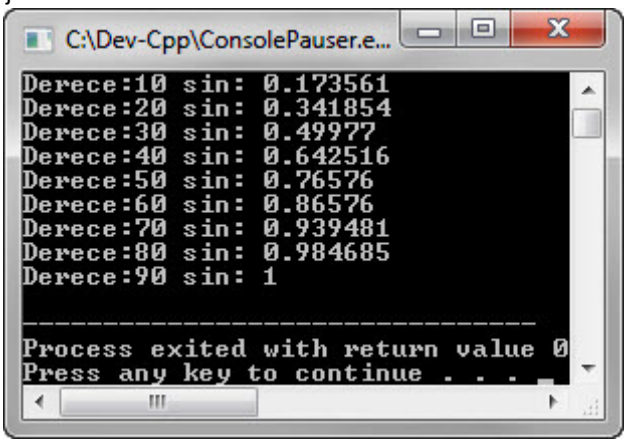
Uygulama

```
//yuvarlatma

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

int main(){
        double sayi;
        cout << "Ondalikli bir sayi giriniz: ";
        cin >> sayi;
        cout << "Girilen sayiden kucuk, en buyuk tamsayi= " << floor(sayi) << endl;
        cout << "Girilen sayiden buyuk, en kucuk tamsayi= " << ceil(sayi) << endl;
        return 0;
}
```



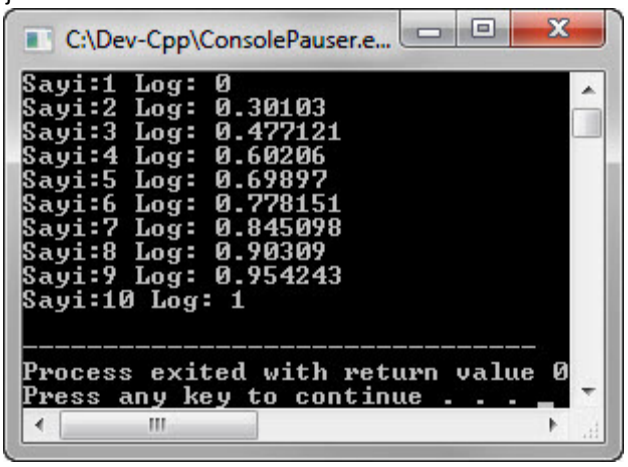
Uygulama

```
// acilar

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

int main(){
	double deger=10.0;
	double derece;
	do{
		derece=3.14/180*deger;
		cout << "Derece:" << deger << " sin: " << sin(derece) << endl;
		deger+=10;
	}
	while(deger<=90.0);
	return 0;
}
```



Uygulama

```
// logaritma(10)

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

int main(){
	double deger=1.0;
	double sayi;
	do{
		cout << "Sayi:" << deger << " Log: " << log10(deger) << endl;
		deger+=1;
	}
	while(deger<=10.0);
	return 0;
}
```



Uygulama:
2'nin, 1'den 10'a kadar olan kuvvetlerini hesaplayan bir program yazınız.

Uygulama:
e sayısının, 1'den 10'a kadar olan kuvvetlerini hesaplayan bir program yazınız.

Uygulama:
1'den 10'a kadar olan sayıların kareköklerini hesaplayan bir program yazınız.

Uygulama:
2pi, pi, pi/2, pi/3, pi/4 ve sıfır değerlerinin cosinüslerini hesaplayan bir program yazınız. Aynı uygulamayı sinüs ve tanjant için uyarlayınız.