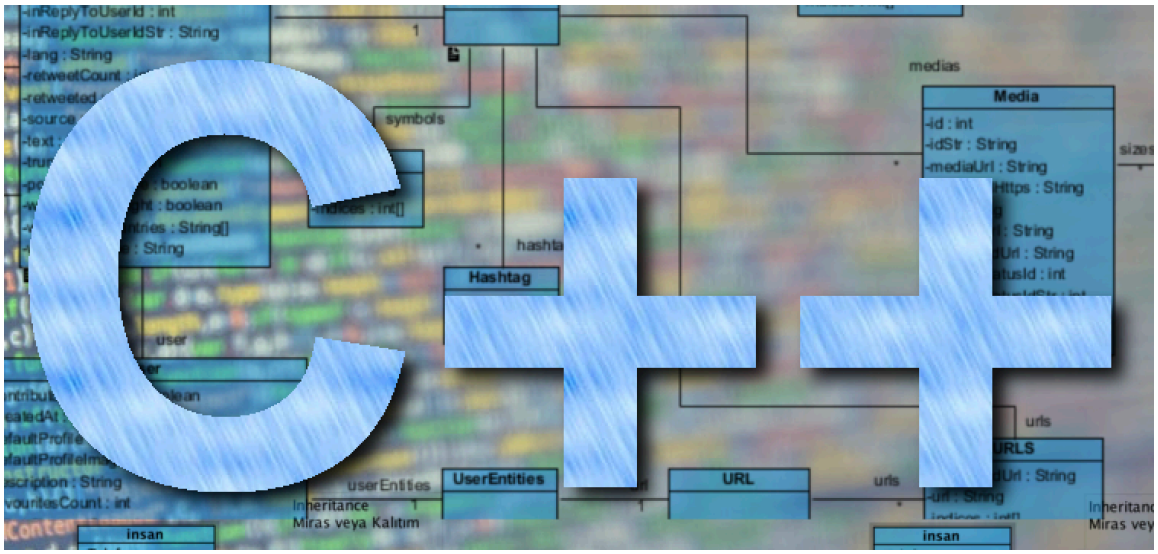
Başlangıç Seviyesinde C++ ile Programlamaya Giriş



Yazarlar
Şadi Evren ŞEKER
Rabia YÖRÜK

2018, İstanbul

İÇİNDEKİLER

C++ ÖĞRENME KILAVUZU

BAŞLANGIÇ DÜZEYİNDE C++ ÖĞRENİMİNE GİRİŞ

GİRİŞ

Bu kitabın asıl amacı programlamaya giriş yapmak isteyen, bu dersi alan veya tekrar etmek, pekiştirmek isteyen, bir şekilde bilişim dünyasına adım atmak isteyip nereden başlaması gerektiğini bilemeyen her yaşta (öğrenci ya da yetişkin fark etmeksizin) insana rehber olmaktır. Bu amaç doğrultusunda, bir programlama dili olan C++ 'ı öğrenmeniz aşamalar ile sağlanarak ilerlenecektir. Bu kitap, aynı zamanda öğrenecek olduğunuz programlama dilini, ileride kendinizi geliştirerek, farklı alanlarda kullanabilmeniz de baz alınarak anlatılmıştır.

GENEL BAKIŞ

Genel olarak kitabın içeriğine değinecek olursak, ilk olarak kitabımızda anlatılan C++ programlama dilini, kendi sanal ortamınızda uygulayabilmeniz ve pekiştirebilmeniz için bir takım program kurulumundan bahsedeceğiz. Daha sonra kodların dünyasına giriş yaparken olmazsa olmazlardan, uygulamamıza bir **"Merhaba Dünya"** yazdıracağız ve C++ programlama diline adım atmış olacaksınız. Kitabın devam içeriği aşağıda mevcut bulunmaktadır:

- **Değişkenler**
- **Temel Giriş/Çıkış (I/O)**
- **Koşullar (If – Else – Else If – Switch/Case)**
- **Döngüler (For – Do/While – While)**
- **Fonksiyonlar**
- **Diziler (Arrays)**
- **Göstericiler (Pointers)**
- **Gösterici Aritmetiği ve Fonksiyon Göstericiler**
- **Diziler (Strings)**
- **Öz Yineli Fonksiyonlar**
- **Dosya İşlemleri**

PROGRAMLAMA DİLİ; C++ NEDİR?

Programlama dilleri insanlar tarafından sanal ortamda yazılan **kaynak kodları (Source Code)**, bir **derleyici (Compiler)** yardımı ile **işletim sisteminin (Operating System)** anlayacağı dile çevirmek amacıyla kullanılır. Bu derleyiciler, yazılan kodları farklı işletim sistemleri için derleyebileceği gibi, Android ya da İos gibi mobil işletim sistemleri için derlenirse de çalışabilmektedirler. Fakat bir işletim sistemi için derlediğimiz kod Android ya da İos cihazlarda çalışmazken, aynı durum tam tersi için de geçerlidir. Yani Android ya da İos için derlediğimiz kodları da bir işletim sisteminde çalıştıramamaktayız. Burada C++ 'ın derleyicisi (Compiler) almış olduğu kodları işletim sisteminize göre derleyerek, cihazın **donanımı (Hardware)** üzerinde çalışmasını sağlayacaktır.

C++ programlama dili aslında C tabanlı ve nesne yönelimli bir programlama dilidir. İlk aşamada C++ 'ı daha iyi kavrayabilmeniz için kod türlerinden aşağıda kısaca bahsedilmiştir:

MAKİNA KODLAMASI (MACHİNE CODE): Makineler tasarlandığı zaman, sadece 1 ve 0'lardan oluşan kodlar ile tasarlanmıştır. Bu sebeple onlar esas itibariyle ikilik tabanda **(binary)** sayılardan ibarettir ve bu sayılar makinede işlenen dijital sinyallerin birer gösterimidir. Yani diğer bir dille makinede kullanılan ve her birisi farklı anlamlara gelen sinyallere makine dili ismi verilebilir.

DÜŞÜK SEVİYE KODLAMA (LOW LEVEL CODİNG): Makine kodlamasının bir üst seviyesinde yer alan düşük seviye kodlamadan anlaşılması gereken ilk şey, kodu yazan kişinin detaylarıyla ve daha uzun bir yoldan bu kodu yazması gerektiğidir. Örneğin bu kodlama yöntemi için kullanılan **Assemble** de bir mesajı yazdırırken, birçok detayı programcının kendisi belirtmesi gerekirken, daha üst seviyelerde buna ihtiyaç duyulmayacaktır.

ORTA – ÜST SEVİYE KODLAMA (MİDDLE – HİGH LEVEL CODİNG): Bu gruptaki programlama dillerine bakacak olursak, üst seviyedeki kodlamalarda **Java** gibi programlama dilleri kullanılırken, bizim öğreneceğimiz **C++** ve onun temelini oluşturan **C** orta seviyeli kodlama grubuna dahil olmaktadır. Orta seviyeli kodlamalar, düşük seviyeye daha yakın olarak düşünülebilir. Ayrıca C++ gibi bir programlama dilinde RAM 'e müdahale edebilmek mümkün iken fakat aynı şeyi yapmak üst seviyeli Java gibi bir programlama dilinde sadece kısıtlı olarak mevcuttur.

Kitabın ilk bölümlerinde C++ dilini temelde uygulayarak öğrenmeniz hedeflendiği için üzerinde çokça durulacaktır fakat ilerleyen bölümlerde nesne yönelimli programlamaya da değinilecektir.

GEREKSİNİMLER

Bu dili öğrenebilmek ve dahi uygulayabilmek için herhangi kişisel bir gereksinime ihtiyacınız yoktur. Ancak kendinizin de daha sonra uygulayarak pekiştirebilmeniz için, anlatımlar bir **IDE (Integrated Development Environment – Tümleşik Geliştirme Ortamı)** üzerinde yapılmıştır. Kısaca **kodlarımızı yazacağımız ve çalıştıracağımız ortam** olarak nitelendirebiliriz. Kitap anlatımı için seçtiğimiz uygulama **CODELİTE** olup, kişiden kişiye kullanılmak istenilen uygulama değişebilir. CODELİTE 'ı tercih etmemizin nedenleri arasında açık kaynak kodlu bir uygulama olmasını ve hemen hemen tüm işletim sistemleri için formatının bulunmasını gösterebiliriz. Kitabımızın içeriğinde CODELİTE 'ın nasıl kurulması gerektiğinden bahsedilmiştir. Fakat özet itibari ile IDE bizim kodlarımızı yazmamız için yardımcı olacak bir ortamdır ve kesinlikle olmazsa olmaz değildir. C++ bir dildir ve bu dilin nerede yazıldığının bir önemi yoktur. İsterseniz Notepad 'te de ya da bilgisayarınızın terminalinde de kodlarınızı yazarak çalıştırabilmeniz mümkündür.

EĞİTİME NASIL ÇALIŞILMALI?

Bireyden bireye değişmekle birlikte, eğitim sırasında en etkili öğrenme biçimi olarak gördüğümüz yöntem, herhangi bir bölümü okuduktan sonra ya da okurken eğitime ara vererek kendi sanal ortamınızda bu uygulamayı kendiniz yazmaya çalışmanızdır. Böylece hem yazılan kodları uygulamış olacak hem de aklınızda daha kalıcı hale getirilmesi sağlanacaktır. Kitabımız üzerinde sizin de yapmanız için örnekler ve ödevler bulunmaktadır. Kendiniz kodları uyguladıktan sonra kitabımızdan kodların doğruluğunu kontrol edebilirsiniz.

KODLAMA ORTAMI

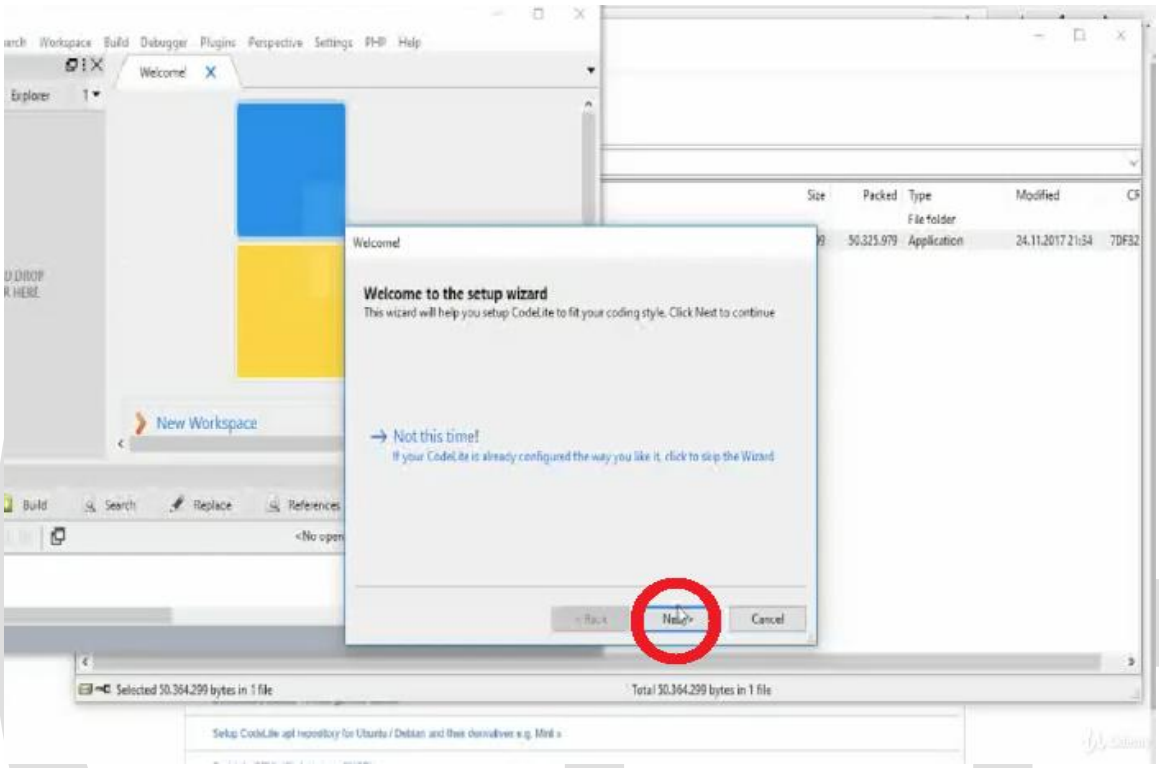
CODELİTE KURULUMU

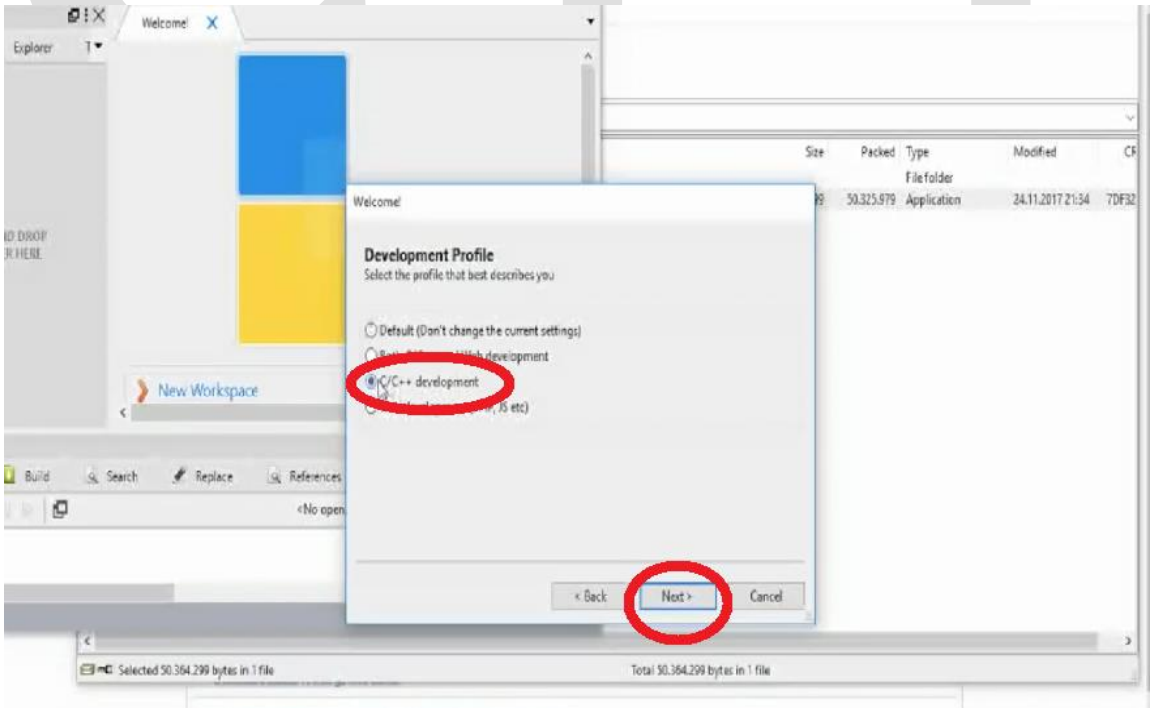
CODELİTE uygulamasının kurulum aşamaları aşağıda maddeler halinde anlatılmıştır:

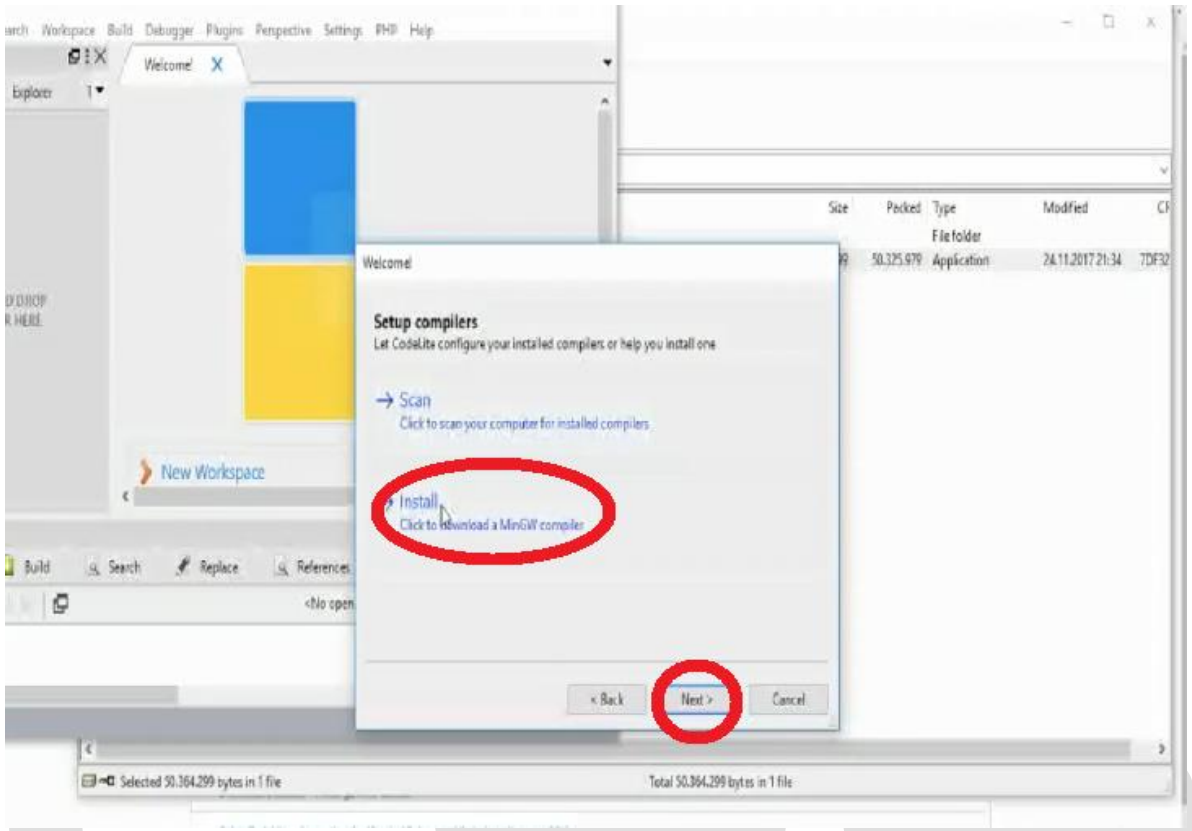
1. İlk olarak bir arama motoru vasıtası ile **https://codelite.org/** adresine gidiniz.
2. Açılan sayfada '**download**' sekmesine gittiğiniz zaman, indirmeniz için hazır bulunan linklere ulaşmış olacaksınız.
3. Bu sayfadan kendi cihazınız için uygun olan versiyonu seçerek, ilgili linke tıklayabilirsiniz.
4. Dosya bilgisayarınıza indirildikten sonra, bilgisayarınızın '**karşıdan yüklemeler'** klasörüne gidiniz.
5. Uygulama ilk olarak karşımıza bir **Win-rar** dosyası olarak çıkmaktadır. Sağ tıklayarak '**dosyayı çıkart**' seçeneğine tıklamanız gerekmektedir.
6. Açılan klasörde **.exe** uzantılı dosyayı çalıştırırsanız, uygulamanın kurulumu için bir sekme açılacaktır.
7. Burada '**next**' tuşuna basarak, daha sonra anlaşmayı kabul etmeniz gerekmektedir (ilgili kutuyu işaretleyiniz).
8. En son '**instal (yükle)**' tuşuna basarak kurulumu tamamlayınız. Böylece bilgisayarınıza CODELİTE uygulaması kurulmuş olacaktır.

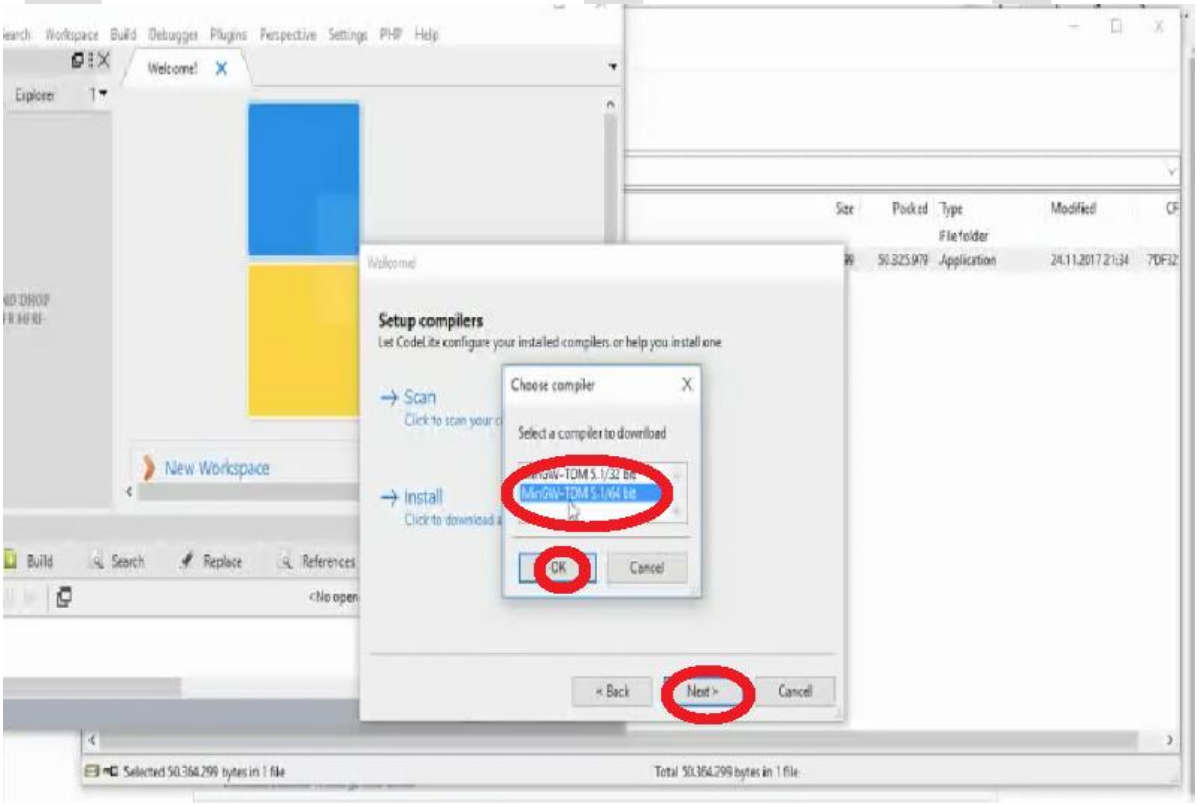
KISA BİLGİ: Kitap hazırlandığı zaman 11.0.6 versiyonu en yeni versiyonu olduğu için, aşağıda bazı alınmış ekran görüntülerinde bunu fark edebilirsiniz. Fakat indirilen sürümün eski ya da yeni olmasının bir farkı yoktur. Temelde hepsi bizim C++ kodlarını yazmamız için gerekli olan ortamı sağlayacaklardır. Siz kitabı okuduğunuz zaman cihazınızda hangi sürümü mevcut ise onu kullanabilirsiniz.

Açılan ekranda temel ayarları yapmanıza yardımcı olabilmek için işaretlemeniz gereken yerler sırasıyla kırmızı ile daire içine alınmıştır.

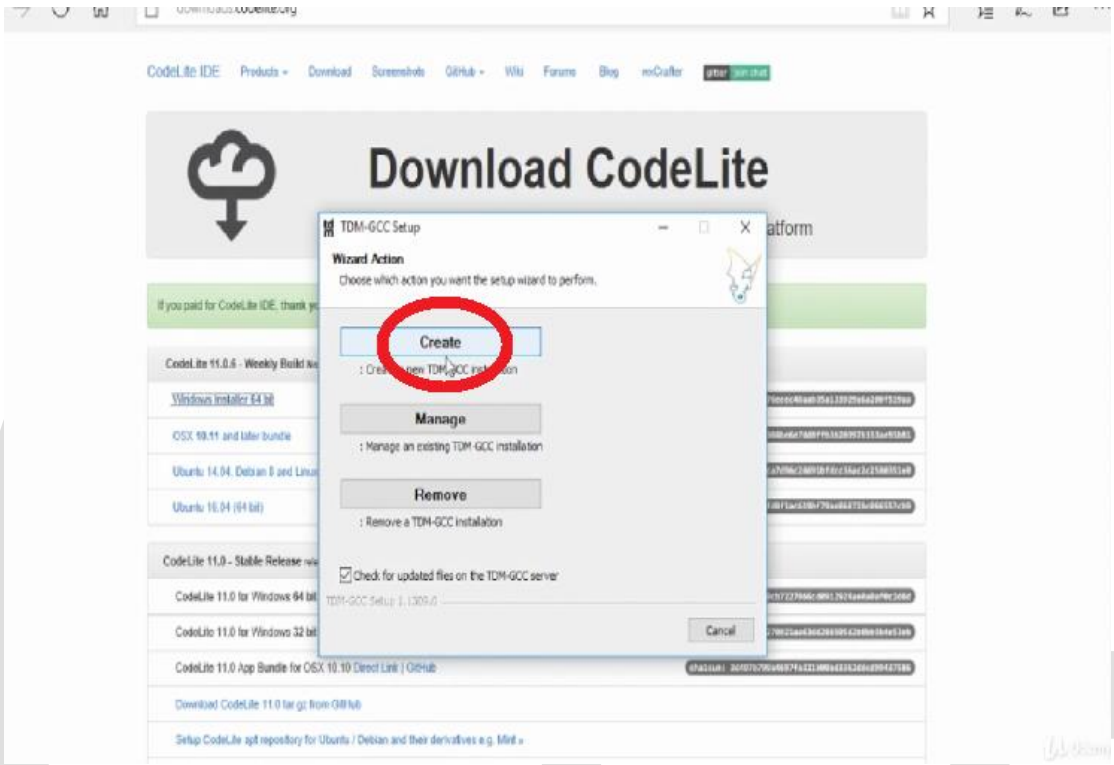


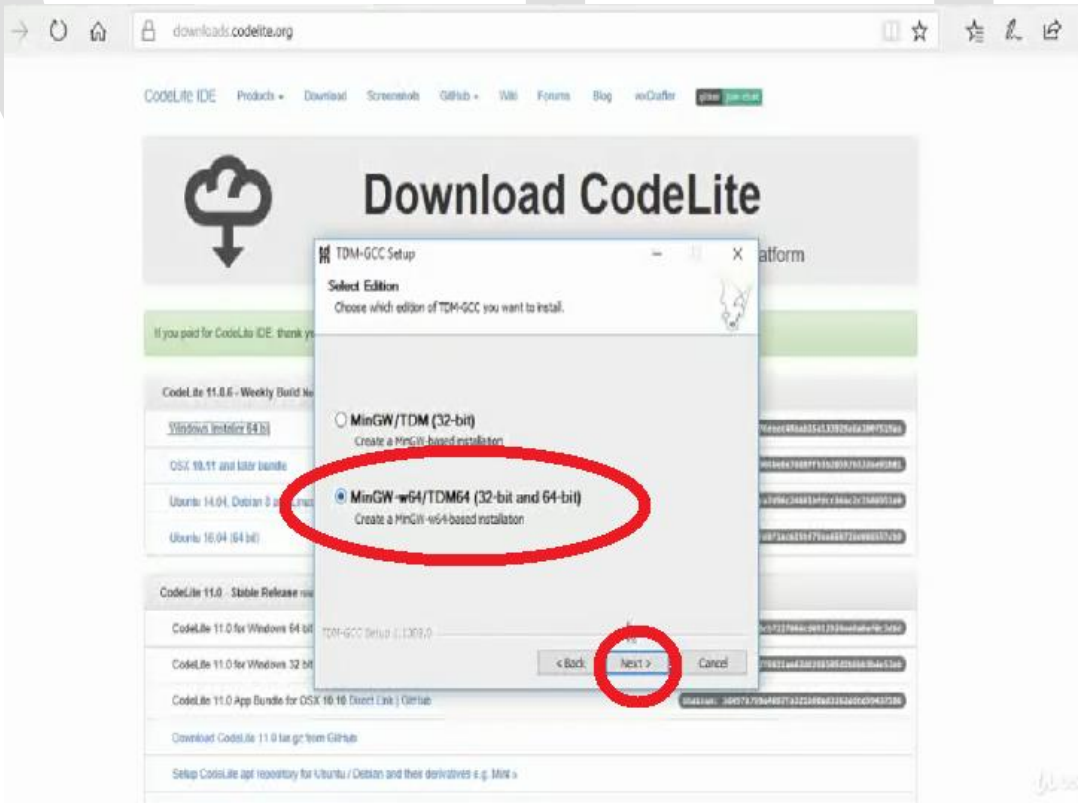


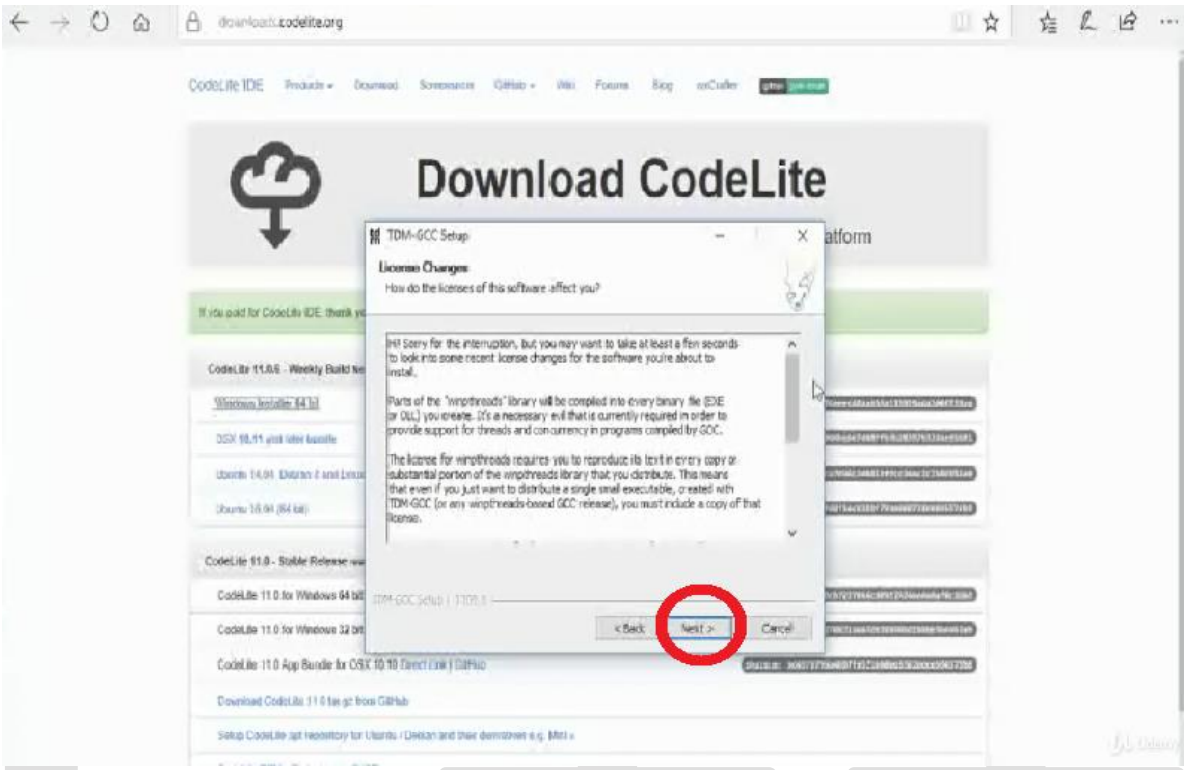

Welcome!
Setup compilers
Let CodeLite configure your installed compilers or help you install one
Scan
Click to scan your computer for installed compilers
Install
Click to download a MinGW compiler
< Back
Next >
Cancel
New Workspace
Selected 50.364.299 bytes in 1 file
Total 50.364.299 bytes in 1 file

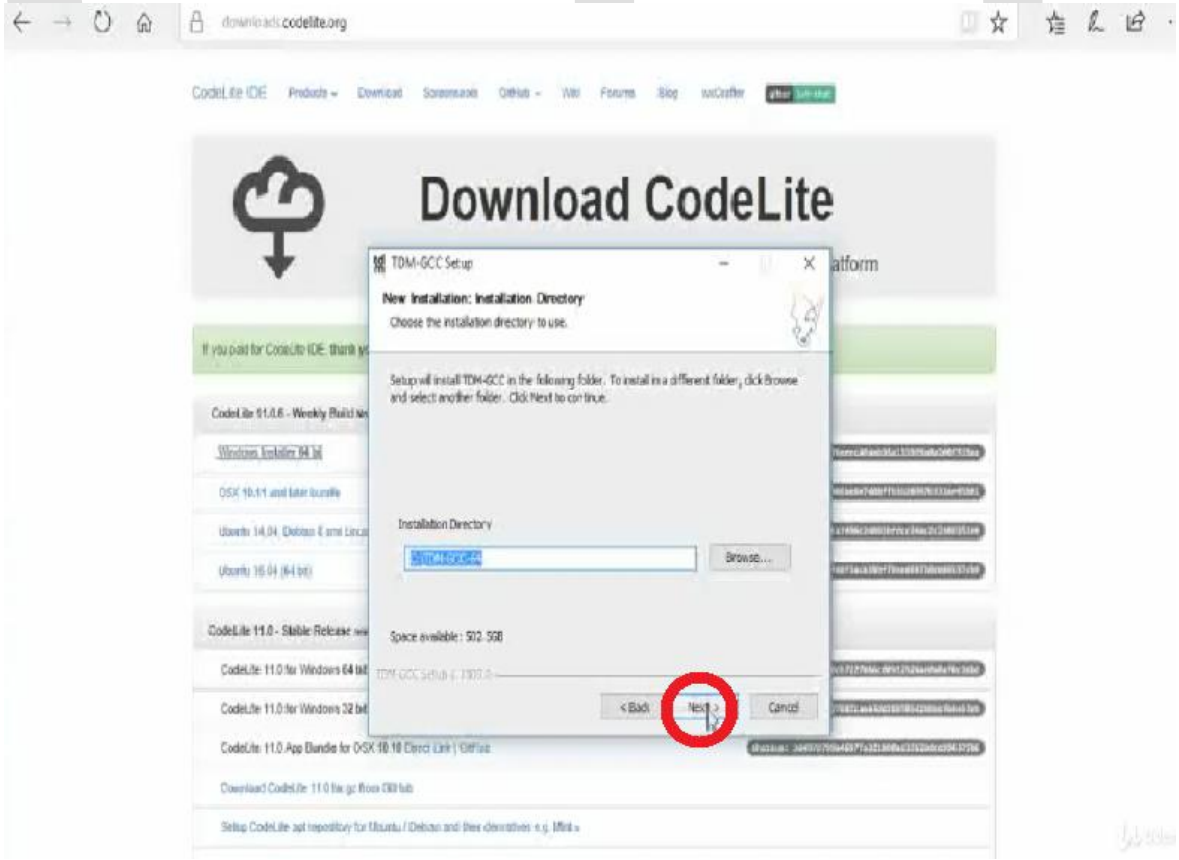

Welcome!
Setup compilers
Let CodeLite configure your installed compilers or help you install one
Choose compiler
Select a compiler to download
Scan
Install
OK
Cancel
< Back
Next >
Cancel
New Workspace

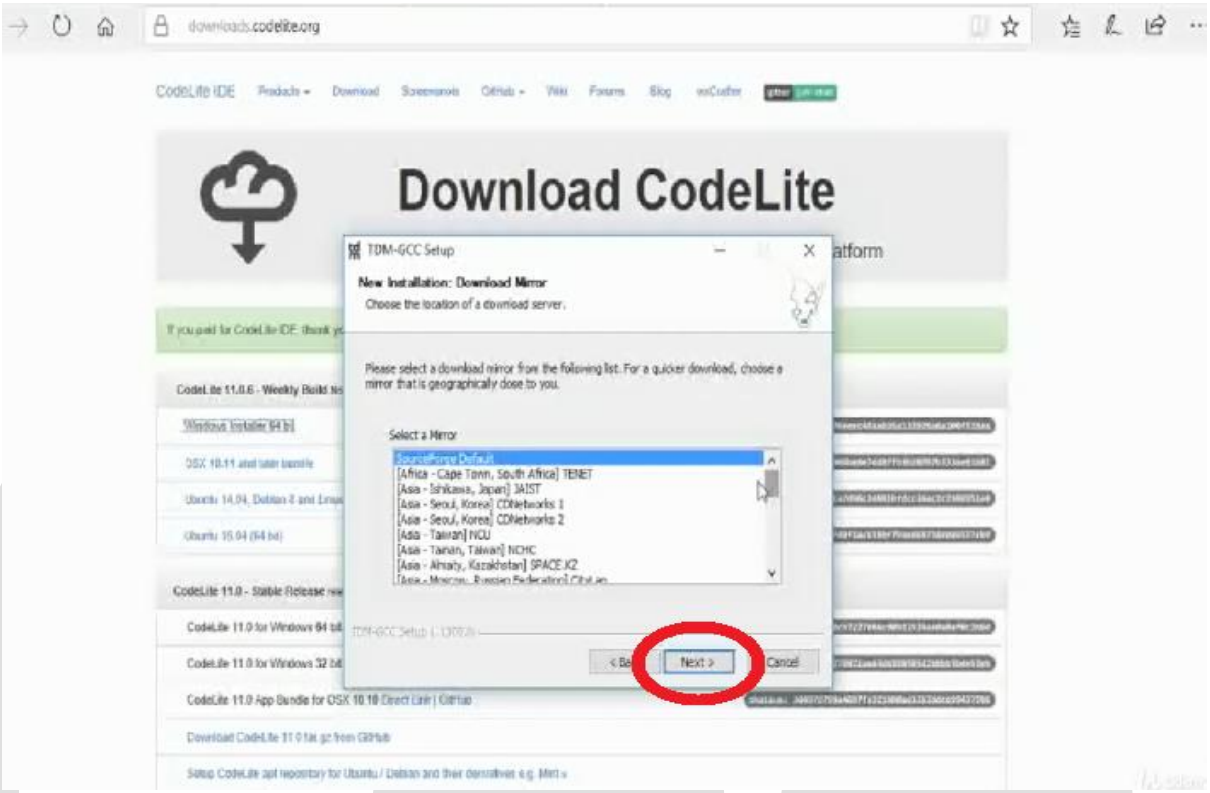
Bu işlemleri tamamladıktan sonra bir Compiler indirme sayfasına yönlendirileceksiniz. İndirme otomatik olarak başlayacak olup, tamamlandıktan sonra işlemler şu şekilde olacaktır.



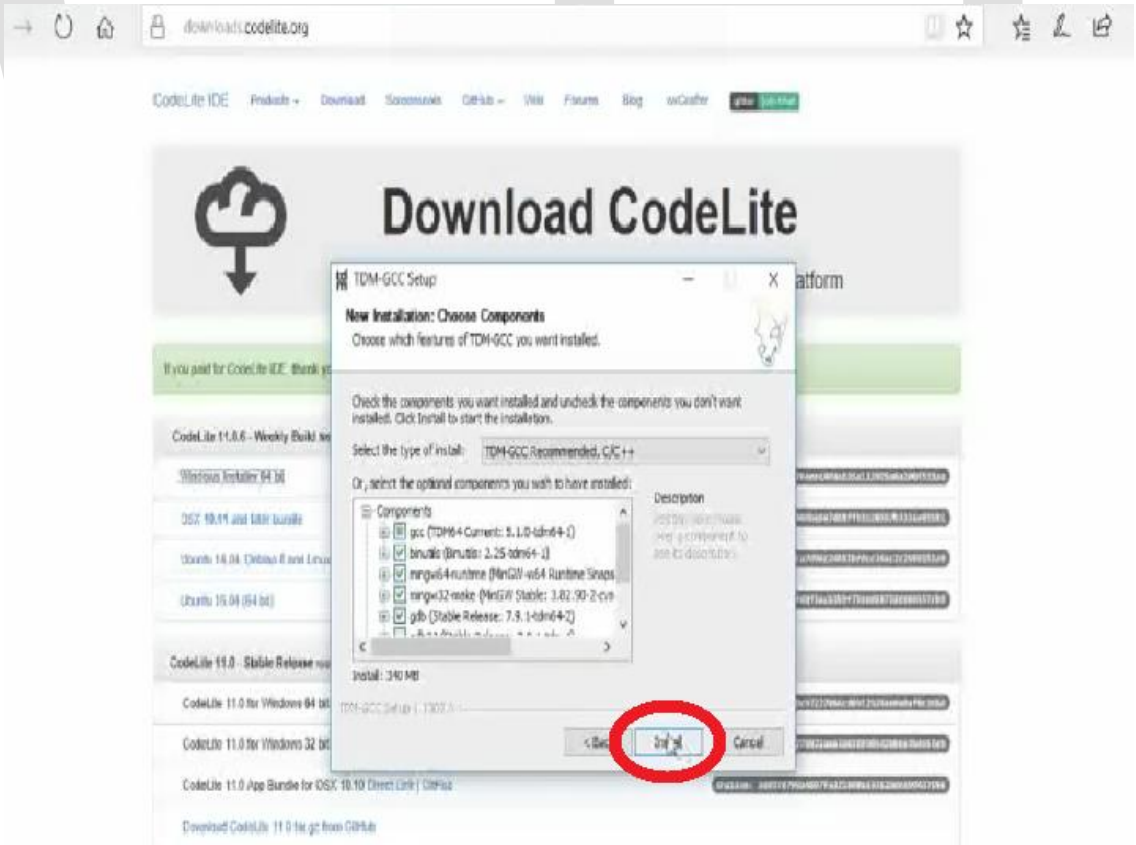



downloads.codelite.org
CodeLite IDE
Download CodeLite
TDM-GCC Setup
License Changes
How do the licenses of this software affect you?
< Back
Next >
Cancel


downloads.codelite.org
CodeLite IDE
Download CodeLite
TDM-GCC Setup
New Installation: Installation Directory
Choose the installation directory to use.
Setup will install TDM-GCC in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Installation Directory
Browse...
Space available : 502.5GB
< Back
Next >
Cancel



İşlemin son aşamasını ‘finish’ butonuna basarak bitiriyoruz.



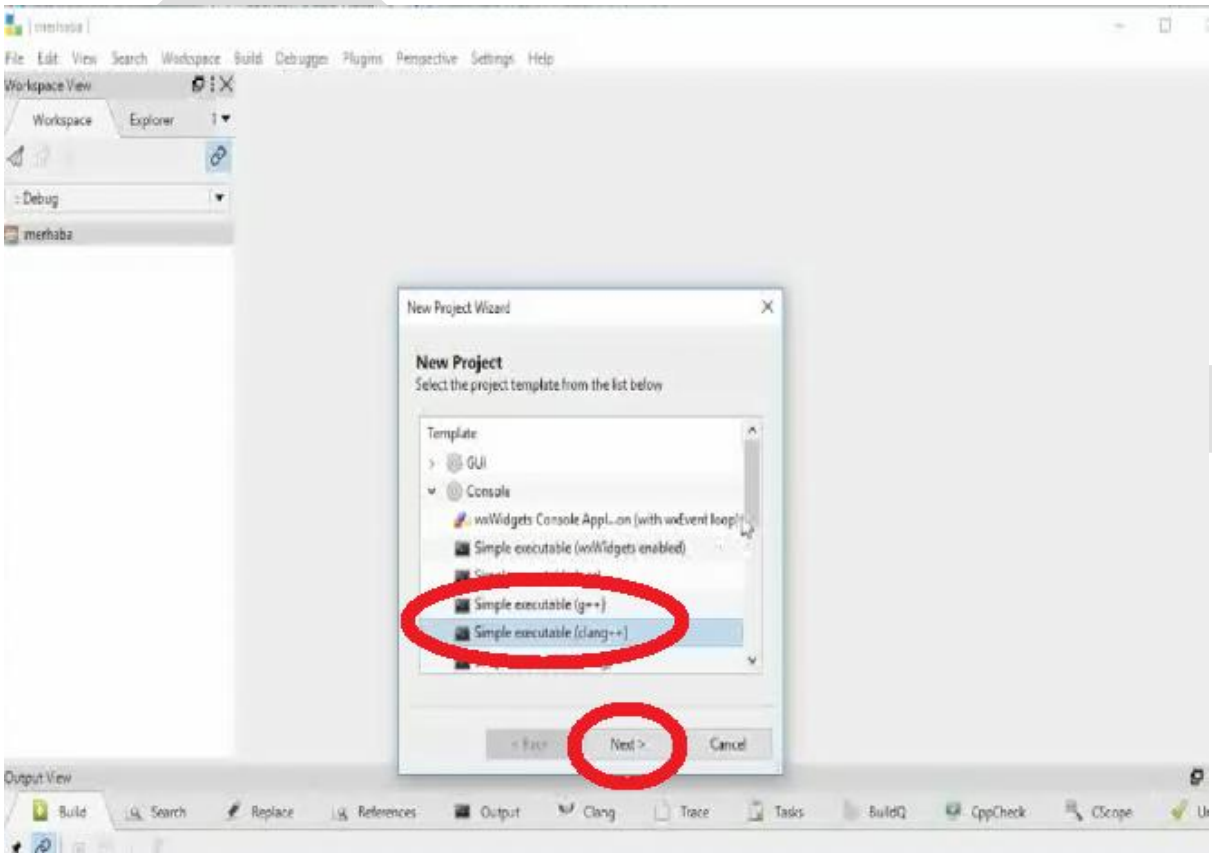
Açılan sayfa bir önceki kurulumla aynı olup, bu sefer instal yerine ‘scan’ butonuna basıyoruz. Ve yüklemiş olduğumuz derleyiciyi seçiyoruz.

Bir sonraki ekran için, uygulamanın nasıl görünmesini istediğinizi seçebilirsiniz. Codelite uygulamanız kullanmak için hazır olarak bulunmaktadır.

İLK ÇALIŞMA

Başlarken yeni bir workspace (çalışma alanı) oluşturmanız gerekmektedir. Bu yüzden '**New Workspace**' yazan alana tıklayın ve çalışmanıza bir isim verin.

Çalışma sahanızın içerisine bir proje oluşturmak için sağ tıklayınız ve '**New Project**' seçiniz. Burada bizim oluşturacak olduğumuz proje bir kütüphane vb. farklı alanlar olabilir. Biz başlangıç aşaması için '**Console**' seçeneğini ve altında C++ için bulunan **Clang++** 'ı seçerek ilerleyeceğiz.



Ayarlarda herhangi bir değişiklik yapmadan ilerlediğiniz takdirde kod yazmanız için sanal ortamınız hazır olarak karşınızda olacaktır. Sayfada bu sanal ortama ulaşmak için tek yapmanız gereken, **çalışma alanı ismi/ Proje ismi / src / main.cpp** şeklinde ilerleyerek **main.cpp** basarak çalıştırmanızdır.

MERHABA DÜNYA VE YORUMLAR

İLK KODUMUZUN ÇALIŞTIRILMASI

Daha önce hiç kod yazmamış olanlar ya da unutanlar, tekrar etmek isteyenler için ilk kod genelde en zor aşamadır fakat bu kılavuz ile birlikte bunun o kadar da zor olmadığını kavramanızı amaçlamaktayız. Codelite uygulamanızı açıp bir proje oluşturduktan sonra ilk karşımıza çıkan ve gözümüze çarpan '**Hello World**' yazısıdır. Yazılım dünyasının birçok köşesinde klişeleşmiş ve artık bir standart haline gelmiş olan bu kodu yazmaya başlamadan önce, halihazırda karşınıza çıkan hazır kodların ne anlama geldiklerini anlamanız gerekmektedir. Böylece bir projenin nasıl çalıştırılması gerektiğini ve işimizi kolaylaştıran birkaç ipucuna sahip olmuş olacaksınız.

main.cpp

```cpp
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << "Hello World" << std::endl;
    return 0;
}
```

İlk olarak C++ kodlarını yazmaya başlarken kullanacağımız kütüphaneyi tanımlamamız gerekmektedir. 'İOSTREAM' olarak tanımlanan kütüphane, bizim 'main' gibi temel kodları kullanmamızı sağlar.

main.cpp

```cpp
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << "Hello World" << std::endl;
    return 0;
}
```

Hemen altında bulunan parantez içerisinde belirtilen kodlar, daha çok işletim sistemi dersi almak isteyen kişileri ilgilendirmekle birlikte eğer projemizden silecek olursak, bir aksaklık olmayacaktır. Yani projemizi çalıştırdığımızda yine çalışacaktır.

main.cpp

```cpp
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << "Hello World" << std::endl;
    return 0;
}
```

Kodları yazarken kullanılan 'std::' ön ekini her özellik tanımlarken yazmak istemiyorsanız eğer en başta belirteceğiniz 'Using Namespace Std' ile bu kısayolu oluşturabilirsiniz.

main.cpp

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

int main(int argc, char **argv)
{
    cout << "Hello World" << endl;
    return 0;
}
```

Böylece en başta yazacak olduğunuz Using Namespace Std bütün dosyayı kaplayacak ve her özellik için ayrı ayrı belirtmenize gerek kalmayacaktır.

main.cpp

```cpp
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
    cout << "Hello World" << endl;
    return 0;
}
```

Yanda gördüğünüz iki tane küçüktür işareti, o kodun içerisindeki yazının ekranda gösterilmesi içindir. Hemen sağ tarafında bulunan 'endl (end of line)' kodu ise satır sonu demektir yani bu mesajın bittiğini ve alt satıra geçilmesi gerektiğini ifade eder. Cout ve endl arasındaki metin kod çalıştırıldığı zaman ekranda gösterilir.

YORUM SATIRI OLUŞTURMAK

main.cpp

```cpp
#include <iostream>
using namespace std;
int main(int argc, char **argv)
// tek satırda yorum

{
    cout << "Hello World" << endl; // burada da yorum var
    return 0; // gjbgkbjlfgsjrı (ne yazarsanız çalışmayacak :)

    /* eğer birden fazla
     * satırda
     * istiyorsanız
     * bu şekilde yazabilirsiniz.. İsterseniz her satırın
     * başındaki '*' ı kaldırarakta
     * kullanabilirsiniz.
     */
}
```

Son olarak nasıl yorum yazıldığını bilmeniz ilk aşamada kazanılması gereken özelliklerden birisidir. Eğer bir yorum satırı oluşturmak istiyorsanız '// ' yaparak kolaylıkla oluşturabilirsiniz. Çift slash yaptıktan sonra yazdığınız hiçbir şey işleme alınmayacaktır. Yorum satırları yapmak istiyorsanız da başlarken '/*' kullanıp, arasına notlarınızı istediğiniz kadar aldıktan sonra '*/' koymanız yeterlidir.

Yorum (comment) bir kodun açıklanması ya da kolay hatırlanabilmesi için yazan kişinin aldığı kişiye özel yazıdır. Program üzerinde yazdığınız yorum etkisiz kabul edilir ve çalıştırılmaz. Sadece kod içeriğine bakan kişi tarafından görülebilir.

Yorum yazmak her ne kadar ilk başlarda gereksiz gibi görünse de eğer bir projeyi büyük kapsamlı olarak yürütüyorsanız, başka insanların daha sonradan bu kodları anlayabilmesi için çok önemlidir.

DİLİN TEMELLERİ

DEĞİŞKENLER (VARİABLES) VE TANIMLAMA

Görecek ya da görmüş olduğunuz bütün programlama dillerinde 'değişenler' konusu temelde büyük önem taşımaktadır. Değişkenleri genel tabiri ile bir kutu olarak düşünmek mümkündür. Siz bu kutunun içerisine bir değer atarsınız. Daha sonra projenin gidişatına göre tekrar aynı değişkene başka bir değer atayabilirsiniz. Buna o değişkeni güncellemekte denebilir.

Değişkenler kendi aralarında, içerlerine atanan değerin özelliğine göre birçok tipe ayrılmaktadırlar. Değişkenlerin mantığını kavramanız için ilk olarak 'int (integer – tamsayı)' adını verdiğimiz bir değişken tipi ile örnekler göstereceğiz. 'int' tipi değişken C altyapısına sahip çoğu dilde bulunan ana bir değişkendir.

Bir değer atayacağımız zaman:

int a; (Burada 'a' (değişkenin ismi) için hangi tip (int) değişken ile ifade edildiğini belirttik.)

a = 10; [a için 10 değerini atadığımız (atama işlemlerine assignment da denebilir) gibi herhangi bir sayı da olabilir.]

Bu gösterim haricinde farklı kullanım alanları da mevcuttur.

int a = 10;

şeklinde gösterilerek de aynı değer atanabilir.

Birden fazla değişken atanacağı zaman,

int a = 10;

int b = 15;

Şeklinde gösterilmesi ile yan yana yazımı

int a = 10, b = 15; (anlamı : a diye bir değişken ata ve 10 değerini alsın. Daha sonra b diye de bir değişken ata ve 15 değerini alsın.)

arasında herhangi bir farklılık söz konusu değildir. Kod yazarken genelde satır başından başlanmasının sebebi, daha sonra baktığımız zaman kolaylık sağlamasıdır.

KISA BİLGİ:

- Ekranda yazdırmak istediğiniz kodu ya da değeri, nasıl kodlayacağınızı "İLK KODUMUZUN ÇALIŞTIRILMASI" başlığı altında incelemiştik. Fakat aynı kodu (cout << "mesaj" << endl ya da cout << değer << endl) değer ya da mesaj olarak ayrı ayrı verebileceğimiz gibi bir arada da verebiliriz. Bunun için yazılacak kod aşağıda verilmiştir:

cout << "mesaj" << değer << endl

- Eğer bir mesajı ekranda yazdırmak istiyorsanız, mesaj çift tırnak içine alınır fakat atanmış olan bir değeri ekranda yazdırmak istiyorsanız kullanılmaz.

Kod yazıldıktan sonra sonuna " ; (noktalı virgül) " konulması, o kodun yazılıp bittiğini belirtir.

BİR DEĞİŞKENE BİRDEN FAZLA DEĞER ATANMASI

Bir başka değinilmesi gereken nokta ise eğer "a" değişkenine birden fazla değer atarsak, bu o değişkenin birden fazla değer aldığı anlamına değil, o değerin güncellendiği anlamına gelir.

Yani sanal ortama aşağıdaki gibi bir değer yazıldığı ve ekrana yansıtıldığı zaman;

```
int a = 10;
a = 20;
cout << a << endl
```

Burada yazılan ikinci değer (20) ekranda gözükecektir.

DEĞİŞKEN İSİMLERİ VE BELİRLEYİCİLER (İDENTİFİERS)

DEĞİŞKEN İSİMLERİ

Kitabın bu bölümüne kadar değişkenlere tanımladığımız isimler tek bir harften ibaretti. Fakat genelde harf kullanımı, kodu yazan kişi tarafından tercih edilmez çünkü kodlara hem kendisi hem de başka biri geri dönerek baktığı zaman, atanan değerin ne olduğu anlaşılsın istenir.

Bu hususta vereceğimiz isim, istediğimiz herhangi bir isim olabilir ve isim harflerden oluşabileceği gibi sayılardan da oluşabilir. Ancak C++ kodlarının kendi özel ismi olmadığı şartı ile. Örneğin;

```
int yas;
int yıl;
int vas18ken;
```

gibi isimler verilebilir. Gördüğümüz kodlardan örnek vermek gerekirse:

```
int int ;
```

gibi C++ kodlarının isimleri değişken ismi olarak atanamamaktadır. Bu gibi isimler C++ tarafından rezerve edilmiş anahtar kelimeler (reserved keyword) olarak geçmektedir.

Bir diğer dikkat edilmesi gereken nokta ise sembol ve işaretler. Bunlarda aynı şekilde isim verirken kullanılabilir fakat birtakım operatörler hariç olarak. İlerleyen konularda bu operatörleri göreceğiz fakat genel olarak bahsetmek gerekirse:

Toplam sembolü " + "

Çıkarma sembolü " – "

Çarpma sembolü " * "

Bölüm sembolü "/ "

Kalan sembolü " % "

gibi operatör sembolleri kullanılmamalıdır.

DEĞİŞKEN TİPLERİ (İLKEL VERİ TİPLERİ)

Örneklerimizde kullandığımız int (integer - tamsayı) değerleri gibi farklı özelliklere sahip diğer değişken tiplerini de bu başlık altında inceleyeceğiz.

GRUPLAR	DEĞİŞKEN TİPLERİN İSİMLERİ	BOYUTLARI HAKKINDA
CHARACTER TYPES (KARAKTER TİPLERİ)	char	8 bit
	char16_t	16 bit
	char32_t	32 bit
	wchar_t	
İNTEGER TYPES - SİGNED (TAMSAYI TİPLERİ – YÖNLÜ)	signed char	8 bit
	signed short int	16 bit
	signed int	16 bit
	signed long int	32 bit
	signed long long int	64 bit
İNTEGER TYPES – UNSİGNED (TAMSAYI TİPLERİ - YÖNSÜZ)	unsigned char	8 bit
	unsigned short int	16 bit
	unsigned int	16 bit
	unsigned long int	32 bit
	unsigned long long int	64 bit
FLOATİNG-POİNT TYPES (KAYAN NOKTA TİPLERİ)	float	32 bit
	double	64 bit
	long double	80 bit
BOOLEAN TYPE	bool	1 bit
VOİD TYPE (BOŞLUK TİPİ)	void	Depolama alanı yok

CHAR DEĞİŞKEN TİPİ: Bu değişken tipinde sadece tek bir karakter tanımlanabiliyor. Yani klavyenizin üstünde gördüğünüz her tuş tek bir karakteri ifade ediyor. Aynı şekilde işletim sistemlerinde, programlama dillerinde vb. de kullanılan farklı semboller de bulunmaktadır.

İNTEGER DEĞİŞKEN TİPİ: İnteger değişken tipinde daha önceki örneklerde de gösterdiğimiz üzere değişkene bir tamsayı ataması yapılmaktadır. Tabloda görüldüğü üzere integer değerler ikiye ayrılmaktadırlar.

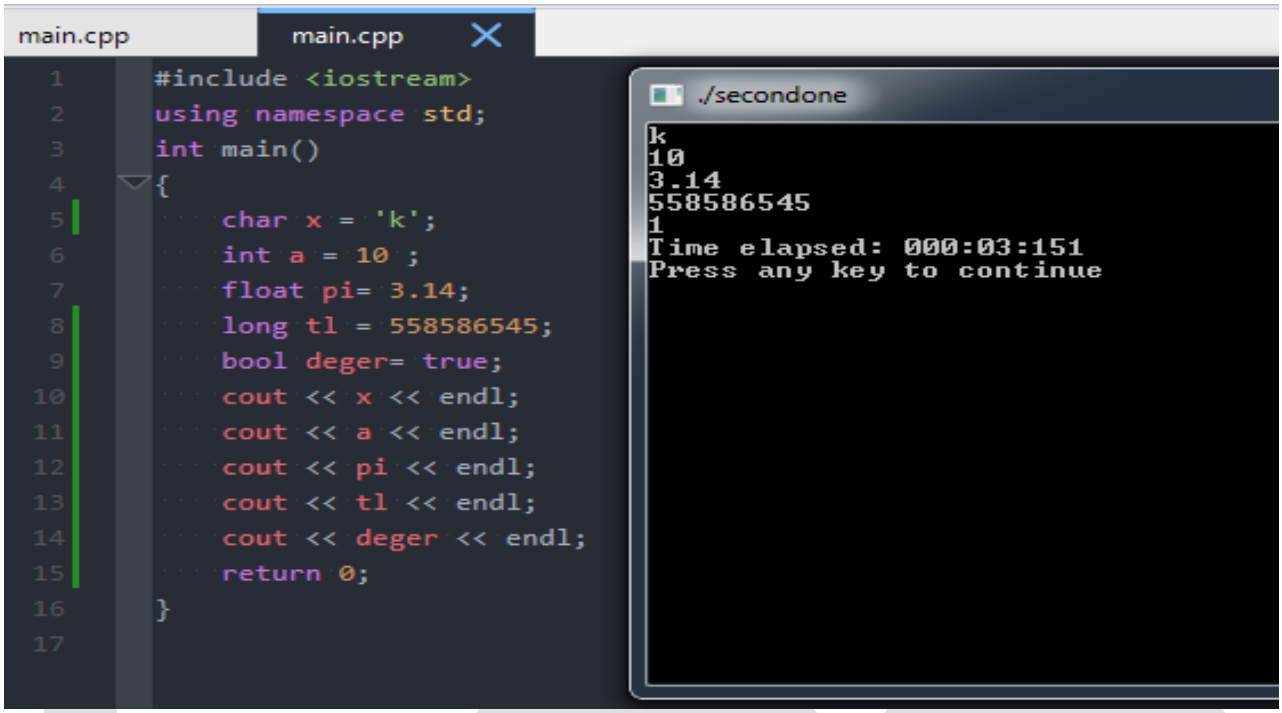
FLOATİNG – POİNT DEĞİŞKEN TİPİ: Bu değişken tipinde atadığımız değer bir ondalık sayıdır. Kendi aralarında float < double < long double gibi bir sıralama yapmak mümkündür. Long double en büyük ondalık değeri alan değişkendir.

BOOLEAN DEĞİŞKEN TİPİ: Boolean değişken tipinde ikilik sistem mantığı kullanılır. Bu değişken true (doğru) ve false (yanlış) veya 0 ve 1 olmak üzere iki farklı değer alabilir.

VOİD DEĞİŞKEN TİPİ: Void kelimesinin Türkçe anlamı ‘tipsiz’ demektir. Yani eğer atanmış olunan karakterin hangi değişken tipine ait olduğunu bilmiyorsanız ‘void’ değişken tipini kullanmanız C++ programlama dilinin bir kolaylığıdır.

Kodlama yaparken ondalık sayılar arasında “ . (nokta) “ işareti kullanılır.

Örneğin sanal ortam üzerinde kodların yazılması ve ekranda gösterilmesi aşağıdaki şekilde olduğu gibidir:



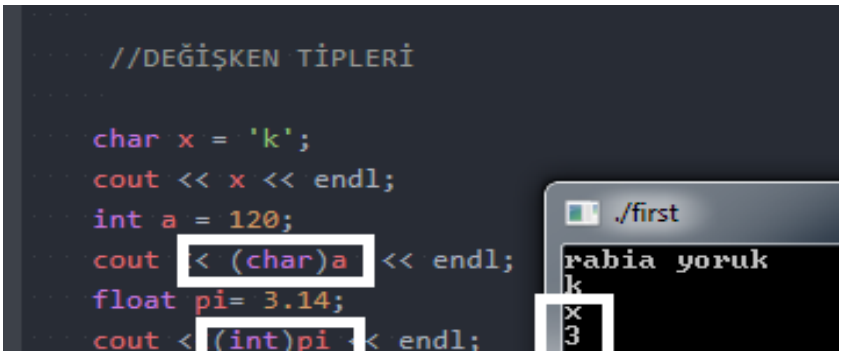
TİP DÖNÜŞÜMLERİ

Birçok farklı değişken tipi olduğu gibi, aynı zamanda bu değişken tiplerinin kendi aralarında dönüşebilmesi de mümkündür.

Tanımlamış olduğumuz bir değişkeni, farklı bir değişken tipinde karşılığı olarak ekrana yazdırmak istersek, buradaki tip değişimini sağlamak için birkaç yol mevcuttur. Örneğin ilk olarak kodu ekrana yazdırmak için kullandığımız kodların arasına parantez içinde istediğimiz değişken tipini belirtmektir. Aşağıda görüldüğü gibi int değişken tipinde atadığımız değeri ekrana yansıtırken char olarak gösterilmesini belirtiyoruz ve sanal ortam bunu bizim için char tipine çeviriyor.

Aynı zamanda burada yapılan diğer bir çeviri de integer (tamsayı) bir değer ile float (ondalık sayı) bir değer arasında. Float (ondalık sayı) olarak atadığımız değişken değeri '3.14' , integer (tamsayı) değişken tipine çevrilirken o sayının sadece tamsayı kısmı alınarak işlem yapılır.

Ya da diğer değişkenin daha önce



bir yol ise bir içerisine (int), tanımlanmış

(char) farklı tipte bir değer atayacak olursak bu kod çalıştırıldığı zaman aşağıdaki gibi, kodun karşılığı bir integer değer olarak görülür. Aynı şey tam tersi durum içinde söz konusudur. Char değişkeninin içerisine integer olarak yazdığımız değer char tipinde ekrana basılmıştır.

Fakat sayı olarak yazdığımız 120 değeri nasıl oluyor da x harfine karşılık geliyor? Bu sorunun cevabı ise ASCİİ tablosu ile açıklanmaktadır.

ASCİİ TABLE

ASCİİ tablosu dönüşümler için bize yardımcı olmak için hazırlanmış bir tablodur. Kendi sanal ortamınızda da aşağıdaki verileri kullanarak Dec (integer) ve Char sütunları ile değişken tipleri arasındaki dönüşümü sağlayabilirsiniz.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

.

İŞLEMLER / OPERATÖRLER

OPERATÖRLER			
ARİTMETİK	KARŞILAŞTIRMA	MANTIKSAL	BİT DÜZEYİNDE
+ : TOPLAMA	> : BÜYÜK	&& : VE	& : VE
- : ÇIKARMA	< : KÜÇÜK	|| : VEYA	| : VEYA
* : ÇARPMA	>= : BÜYÜK EŞİT	! : DEĞİL	^ : YA DA
/ : BÖLME	<= : KÜÇÜK EŞİT		<< : SOLA ÖTELE
% : MOD BÖLME	== : EŞİT		>> : SAĞA ÖTELE
++ : BİR ARTTIRMA	!= : FARKLI		
-- : BİR AZALTMA			
= : ATAMA			

Bu bölümde yukarıda anlamları belirtilen ve daha önceki bölümlerde gösterdiğimiz sayısal değişken tiplerinin değerleri ile nasıl işlem yapılacağı gösterilecektir. Bu hususta bize yardımcı olacak olan semboller yukarıda gösterilmiş olup, sanal ortamınızda kodlarınızı yazarken kullanacağınız bu sembollere operatörler adı verilir..

ARİTMETİK OPERATÖRLERİ

TOPLAMA– ÇIKARMA– ÇARPMA– BÖLME- KALAN OPERATÖRLERİ

Sayısal tipteki iki değeri toplamak, birbirinden çıkarmak, birbiri ile çarpmak ve bölmek için kullanılan genel 4 işlem matematik operatörleridir. Bunlar hariç olarak bir de kalan operatörü mevcuttur. Bu operatör matematiksel işlemler için kullandığımız mod kavramı ile aynı görevi üstlenmektedir. Yani bir sayının başka bir sayı ile bölümünden sonucunu değil, kalan değeri bize vermektedir.

Toplama ve Çıkarma Operatörlerinin Kullanımı

```
cout << "islemler ve operatorler" << endl;

//toplama ve çıkarma operatörleri

int a = 70;

int b = 80;

cout << a + b << endl;
cout << b - a << endl;
```

```
./islemlerveoperatorler
islemler ve operatorler
150
10
Time elapsed: 000:03:713
Press any key to continue
```

Çarpma, Bölme ve Kalan Operatörlerinin Kullanımı

```
// çarpma , bölme ve kalan operatörleri

int d =100;

int f= 25;

cout << d*f << endl;

cout << d/f << endl;

int l =15;

cout << d%l << endl;
```

```
./islemlerveoperatorler
islemler ve operatorler
2500
4
10
Time elapsed: 000:03:853
Press any key to continue
```

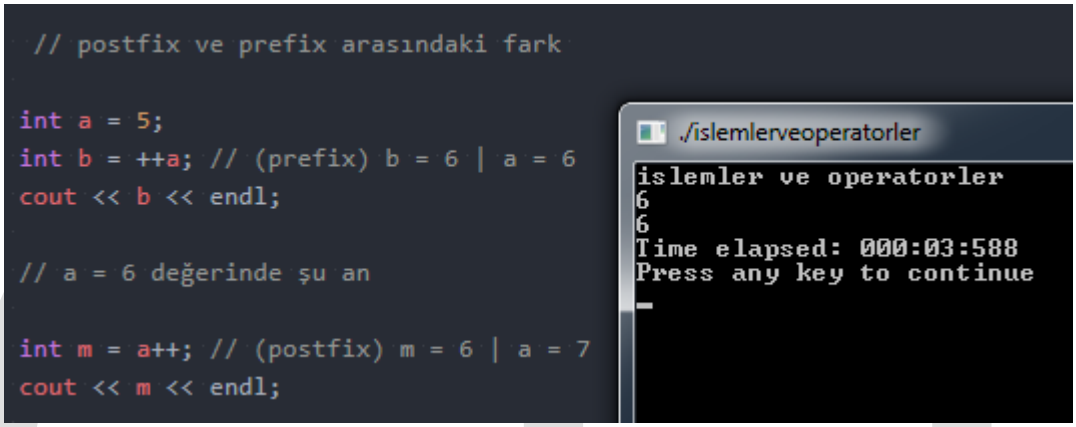
ARTTIRMA VE AZALTMA OPERATÖRLERİ

Eğer bir sayıyı yalnızca 1 arttırmak veya 1 azaltmak istiyorsanız değişken isminin yanına iki tane toplam veya çıkarma işlemi sembolü koymanız yeterlidir. Bu gösterim tarzına prefix denir. Buna ek olarak yukarıda ki kod sırasıyla yazıldığında herhangi bir değişiklik olmayan ama aralarında çok ufak bir fark bulunan bir gösterim tipi daha vardır ki o da postfix 'tir. Postfix gösteriminde ise değişken isminin başına iki tane sembol konulmalıdır.

```
//attırma ve azaltma operatörleri
int c = 5;
c++;

cout << c << endl;

int y = 6;
y++;

cout << y << endl;

int s = 90;
s--, //postfix gösterimi
--s; //prefix gösterimi

cout << s << endl;
```

```
./islemlerveoperatorler
islemler ve operatorler
6
7
88
Time elapsed: 000:03:603
Press any key to continue
```

Postfix ve Prefix ile gösterimin arasındaki küçük farkı açıklamak için örneğimiz aşağıda mevcuttur. Prefix gösteriminde önce a'nın değeri 1 arttırılır ve daha sonra ekrana yansıtılır. Fakat Postfix gösteriminde a'nın değeri alınarak önce ekrana yansıtılır ve daha sonra 1 arttırılır. Bu yüzden 2. Gösterimde ekrana basılan değer 6 olarak kalmıştır.



Karşılaştırma, Mantıksal ve Bit düzeyinde olan operatörlerin kullanımını bir sonraki bölümde Bitwise Operatörler (ikilik tabanda işlemler) konusunda anlatılacaktır.

BİTWİSE OPERATÖRLER (İKİLİK TABANDA İŞLEMLER)

İKİLİK TABAN NEDİR?

Matematikte taban aritmetiği olarak da geçen binary sistemde amaç sayıların eşitini ikinin kuvvetleri ile yazmaktır. Örneğin;

5 sayısını ikilik taban sistemine göre yazalım:

$5 = 2^0 + 2^2$ şeklinde gösteririz fakat bu sistemde sayılar ifade edilirken 0 ve 1 baz alınarak ifade edilir. Yani eğer bir sayıda, ikinin kullanacağım kuvveti için 1 ve kullanmayacağım kuvveti için yerine 0 yazmanız gerekmektedir. Bundan yola çıkarak örneğimizi şöyle açıklayıcı hale getirebiliriz:

5 = 0 1 0 1 = $5 = 2^3 + 2^2 + 2^1 + 2^0$

5 = 0101 şeklinde ifade edebiliriz. Bir diğer örneğimizde incelemek isterseniz 17 sayısını ele alalım:

$17 = 2^4 + 2^0$

Başka bir deyiş ile sayıları ifade ederken yukarıdaki gibi yazdığımız zaman, yazılan değerler için ikilik tabanda (yani 4 ve 0.kuvveti için) 1 , kullanmayacağımız diğer kuvvetleri (1, 2 ve 3 için) 0 değeri verilmektedir. Buna göre:

17 = 1 0 0 0 1

Şeklinde ifade edebiliriz.

İkilik tabanın iyi anlaşılması için işlemi tam tersi şekilde gerçekleştirecek olursak, rastgele yazdığımız 1 ve 0'lardan oluşan bir sayıyı, ikilik taban yardımıyla eşit olan sayıya çevirmeyi deneyelim:

$1\,0\,1\,0\,0\,1 = 2^5 + 2^3 + 2^0 = 41\ sayısına\ eşittir.$

Sıra geldi bu gibi işlemleri kodlarımızda nasıl kullanacağımıza. İkilik tabanı anlatmamızın öncelikli nedeni, Bitwise operatörlerini kullanırken çıkan sonuçları anlayabilmeniz içindir.

SAĞA VE SOLA ÖTELEME OPERATÖRLERİ (SHİFT OPERATÖRLERİ)

İlk olarak sağa ve sola kaydırma operatörlerinden başlayalım. Sola kaydırma operatörü, değişkene atanan değerin, ikilik tabandaki karşılığına 0 eklemek ya da diğer bir tabiri ile ne kadar belirtildiyse o kadar hane sola kaydırmak anlamına gelir. Aynı şekilde sağa kaydırma operatörü ise sonundan kaç tane belirtildiyse, o kadar hanenin silinmesi anlamına gelmektedir. Kodlarımızda bu operatörleri kullanıp, ekrana yansıttığımız zaman, ekrana çıkan değer, haneleri kaydırılmış olan sayının yeni karşılığıdır. Örneklerle açıklamak gerekirse ilkin:

```
cout << "bitwise operatorleri" << endl;

//sola kaydırma

int r = 19;
cout << r << endl;

int b = r << 2;
cout << b << endl;

// sağa kaydırma

int g = 18;
cout << g << endl;

int u = g >> 2;
cout << u << endl;
```

```
./bitwiseoperatorleri
bitwise operatorleri
19
76
18
4
Time elapsed: 000:02:902
Press any key to continue
```

- 19 olarak atadığımız değerin ikilik tabanda değeri = 1 0 0 1 1

Bu sayıyı iki sola kaydırmak demek, 2 hane daha eklemek anlamına geldiğine göre, yeni sayımız: 1 0 0 1 1 0 0

Yeni sayımızı tekrar ikilik sistem yardımı ile geri çevirirsek, ekranda basılan değer olan 76 sayısını elde etmiş oluruz.

- Diğer örneğimiz olan, bu sefer sağa kaydırma işlemini ele alacak olursak, öncelikle girilen sayımızı ikilik sistemde çevirelim. 18 = 1 0 0 1 0

Bu sayıyı belirtildiği gibi iki sağa kaydırmak demek, sonundan iki hane silmek anlamına geldiğine göre yeni sayımız: 1 0 0

Yeni sayımızı tekrar ikilik sistem yardımı ile geri çevirirsek, ekranda basılan değer olan 4 sayısını elde etmiş oluruz.

VE – VEYA – YA DA OPERATÖRLERİ

Diğer Bitwise (İkilik taban operatörleri) operatörlerini inceleyecek olursak sırasıyla Bitwise And, Bitwise OR ve Bitwise XOR olarak da geçen ve, veya, ya da operatörlerinin işlevlerine bakacağız. Ama bunların kodları yazarken ki kullanımına geçmeden önce ikilik tabanda 0 ve 1'lerden oluşan karşılıklarını vermemizde fayda olduğunu düşünüyoruz. Ve, veya, ya da gibi operatörleri kullanırken iki sayıya ihtiyaç duyarız ve bu iki sayı ile bir sonuca ulaşabilmek için her birinin kendine özel olarak bir kuralı olması gerekir. Bu kurallar dahilinde örneklerle açıklayacak olduğumuz aşağıdaki tablolar ortaya çıkmıştır:

SAYI 1	SAYI 2	SAYI 1 & (VE) SAYI 2
1	1	1
0	0	0
0	1	0
1	0	0

Ve operatörünü kullanırken ikilik tabanda yazmış olduğumuz 1 ve 0'lardan oluşan bir sayı için 1 ve 1 'in alt aşta gelme ihtimali haricinde bütün ihtimaller 0 rakamını verir. Örneğin;

Sayı 1 = 1 0 1 0 0

&

Sayı 2 = 1 0 1 1 0 =

1 0 0 0 0 şeklinde bir sonuca ulaşırız. Başka bir örnekle ikili tabanda oluşturduğumuz ve tekrar dönüştürdüğümüz farklı sayılar ile yapalım;

Sayı 1 = 1 1 1 0 1 = 29

&

Sayı 2 = 1 0 0 1 1 = 19

1 0 0 0 1 = 17

Kodlarımızı yazarken 29 ve 19 sayılarını ve operatörüne tabi tutarsak, 17 sayısına ulaşmış oluruz. Sizlere arka planda bunun nasıl olduğunu anlatabilmek için aynı mantık ile veya - ya da operatörlerinde de örnekler ile inceleyeceğiz. Mantık aynı olmakla birlikte, onların tabloları farklıdır.

Veya operatörü için geçerli olan tabloya baktığımızda bu sefer 0 ve 0 rakamları alt alta geldiğinde sadece 0 olduğunu ve diğer durumlarda sonucun hep 1 rakamına ulaştığını görüyoruz.

SAYI 1	SAYI 2	SAYI 1 | (VEYA) SAYI 2
0	0	0
1	0	1
0	1	1
1	1	1

Bundan yola çıkacak olursak bunu yine örnekle açıklayalım:

Sayı 1 = 1 1 0 1 1 = 27

|

Sayı 2 = 1 0 0 1 1 =19

1 1 0 1 1 = 27 sonucuna ulaşıyoruz.

Ya da operatörü için geçerli olan tablo:

SAYI 1	SAYI 2	SAYI 1 ^ (YA DA) SAYI 2
0	0	0
0	1	1
1	0	1
1	1	0

Şayet eğer iki rakam birbirinden farklı yani 1 ve 0 ise sonuç 1, diğer durumlarda (rakamların aynı olduğu) sonuç 0'dır.

Sayı 1 = 1 1 1 1 1 =31

^

Sayı 2 = 1 0 1 0 1 =21

0 1 0 1 0=10

şeklinde sonuçlanır. 0 ile başlayan bir sayı elbette mevcut değildir fakat rakamlar sizlere açıklama ve örnek amaçlı olduğu için sonuç bu şekilde çıkmıştır. Ekrandaki hali:

```
//and ( & ) operatörü
int k = 22 & 13 ;
cout << k << endl;
// veya ( | ) operatörü
int a = 12 | 6;
cout << a << endl;
// ya da ( ^ ) operatörü
int m = 5 ^ 28;
cout << m << endl;
```

```
./bitwiseoperatorleri
bitwise operatorleri
4
14
25
Time elapsed: 000:04:914
Press any key to continue
```

Son kalan Büyük – Küçük – Büyük eşit ve Küçük eşit operatörleri, koşullar konusu ile birlikte kullanılarak anlatılacaktır.

STANDART GİRDİ VE ÇIKTILAR – TEMEL GİRDİ VE ÇIKTILAR(I/O)

Kitabın en başında belirttiğimiz üzere, kısayol olarak bize kolaylık sağlaması için açtığımız workspace üzerine ilk olarak 'using namespace std;' kodumuzu belirtiyoruz. Böylece yazmış olduğumuz cout ve endl gibi kodlardan önce 'std::' ön kodunu eklememiz gerekmiyor. Ve biz başka herhangi bir ön koşul eklemediğimiz sürece, cout ve endl yardımı ile ekrana basılan değer, bizim atadığımız değerler oluyor. Kitabın başından beri kullanmış olduğumuz bu girdi – çıktı işlemine Standart Girdi – Çıktı işlemleri denir.

Temel Girdi – Çıktı işlemleri bizim kodları çalıştırdıktan sonra klavyeden bir değer girmemizi sağlayan ve tekrar çalıştırılan işlemlerdir. Yani bunu günlük hayatta uygulamalarımızı yazarken, örneğin kullanıcıdan veri alabilmek için kullanırız. Bu işlemi ise cin>> kodu ile sağlarız. Açılımı C input (girdi – girilen değer anlamında) şeklindedir.

Ayrıca klavyeden girilen değer üzerinde işlem yapabilmekte mümkündür. Daha önceki bölümlerde öğrenmiş olduğumuz operatörleri kullanarak, kullanıcıdan alınan değeri toplayıp çarpıp bölebilerek tekrar ekrana bastırabiliriz.

```
cout << "temel girdi çıktı işlemleri" << endl;

int a;

cin >> a; //klavyeden girilecek değer

cout << "klavyeden girdiğiniz değer:" << a << endl;

cout << "klavyeden girilen değerin 10 fazlası" << a + 10 << endl;
```

KOŞULLAR (CONDİTİONS)

İF– ELSE – ELSE İF YAPILARI

Koşul ifadeleri bir blok tanımlamayı gerektirir. Kodlarımızı yazarken açtığımız ve kapadığımız süslü parantezler (Curly Bracket) " { } " içerisine yazdığımız kodları kapsayan birer blok tanımlar. Bizim bu zamana kadar açtığımız bloklar main değerine bağlı olan bloklardır. Bundan sonra ki kodlarımızda kendimiz bloklar oluşturacağız ve dolayısıyla ilk oluşturacağımız bloklar koşul (condition) blokları olacaktır.

Koşul bloklarından bahsederken büyük (>) , büyük eşit (>=) , küçük (<) , küçük eşit (<=) , eşit değildir(!=) ve eşittir [normal tek olarak kullanılan eşittir "bir değeri atama" anlamına gelirken , iki tane olarak kullandığımız eşittir sembolü bize gerçek manasında değerlerin birbirine eşit olduğunu gösterir (==).] olan karşılaştırma operatörlerinden yararlanacağız. Bu operatörler iki sayıyı kendi arasında karşılaştırmakta kullanılabileceği gibi aynı zamanda koşul ifadelerinde atanan bir değer ile bir sayıyı karşılaştırmak için de kullanılabilir.

İf kelimesi Türkçe de ‘eğer’ anlamına gelmektedir. Bundan yola çıkarak kodlarımızı yazarken içerisine bir koşul belirtmemiz gerekiyor. Belirttiğimiz koşul için iki ihitimal vardır. Bunlar daha önceki bölümlerde bahsettiğimiz Boolean ifadeleridir yani true (doğru) ve false (yanlış). Şayet o koşul sağlanıyorsa (true değeri atanıyorsa) kod bloğumuz çalışıyor fakat eğer çalışmaz ise (false değeri atanıyor) o koşul sağlanamadığı için başka bir kod bloğuna geçiş yapılıyor.

```
cout << "if - else - else if yapilari" << endl;

int c =15;

if ( c > 10 ) //boolean true | false
    {
    cout << "Kosulumuz saglaniyor" << endl;
    }
```

```
./ifelseif
if - else - else if yapilari
Kosulumuz saglaniyor
Time elapsed: 000:03:572
Press any key to continue
```

Eğer koşulu sağlamayacak bir şey yazacak olursak, bu durumda ekranda herhangi bir ifade ile karşılaşmayacaktık yani ekrana yazdırma kodumuz çalışamayacaktı.

Fakat bu gibi durumlarda yani if koşulunun sağlanmadığı durumlarda devreye else if ve else koşulları girmektedir. Else if koşulu bize if koşunun sağlanmadığı durumlarda çalışacak olan alternatif kod bloklarını sunar ve bir if bloğunu yazdıktan sonra istediğimiz kadar else if koşulu ile devam edebiliriz. Şayet else if bloklarındaki değerlerde sağlanmıyorsa else kod bloğu çalıştırılır. Örnek vermek gerekirse ekranda bu şekilde görülür;

```
cout << "if - else - else if yapilari" << endl;

int c =15;

if ( c > 20 ) //boolean true | false

    {
    cout << "c degeri 20 dan buyuktur" << endl;
    }

else if ( c == 10 ){
 cout << "c degeri 10 degerine esittir" << endl;
}

else if ( c == 12 )
    cout << "c degeri 12 degerine esittir" << endl;

else {
    cout << "c degeri 20 den buyuktur" << endl;
    }
```

```
./ifelseif
if - else - else if yapilari
c degeri 20 den buyuktur
Time elapsed: 000:04:040
Press any key to continue
```

- Koşul bloklarını yazarken izlenmesi gereken ıra if << else if << else şeklindedir.
- Yazılan koşul bloklarda birden fazla else if olabileceği gibi, aynı zamanda birden fazla if koşul bloğu da olabilir. Fakat bu kısımda dikkat edilmesi gereken bir husus vardır: eğer birden fazla if yazıyorsak ilk kısmına, devamında yazılacak olan tüm else if ve else koşul blokları, son yazılan if bloğuna bağlanır.
  Diğer bir olası durum ise koşul bloklarının tümünün if kod bloklarından oluşacak olmasıdır. Hiç else if ve else kod blokları olmadan da sadece if kod bloğu ile bir kodlama yapılabilir.

MİNİ PROJE (NOTLARIN HARF KARŞILIĞI) = "Kullanıcıdan notlarını 100'lük sisteme göre alan ve aldığımız notların karşılığına göre kullanıcıya harf karşılığını veren" kod bloklarını yazalım."

90+ = A

70 – 90 = B

50 – 70 = C

0 – 50 = F

```
// IF VE ELSE YAPILARI
cout << "if - else - else if yapiari" << endl;
int a;
cout << "LUtfen notunuzu giriniz:" << endl;
cin >> a; // KULLANICININ NOTU GİRMESİ İÇİN

if ( a >= 90){

    cout << "A aldiniz.." <<endl;

    }
    else if ( a >= 70 ){

        cout << "B aldiniz.." << endl;

        }
        else if (a >= 50){

            cout << "C aldiniz.." << endl;

            }
    else {

        cout << "F aldiniz ve kaldiniz.." << endl;
    }
```

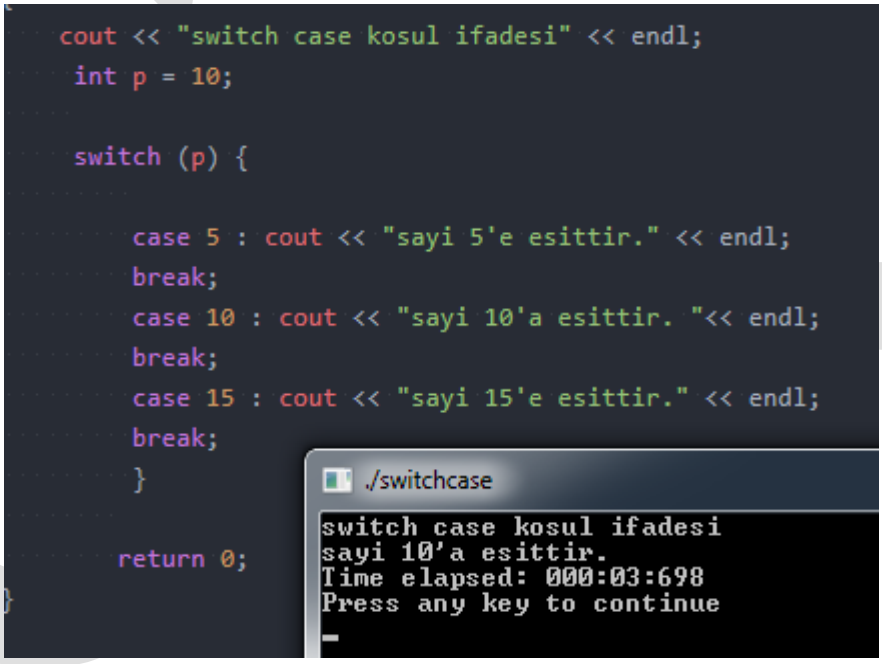
```
./projekosullar
if - else - else if yapiari
LUtfen notunuzu giriniz:
70
B aldiniz..
Time elapsed: 000:13:994
Press any key to continue
```

SWİTCH – CASE YAPILARI

Koşul bloklarının bir diğeri ise Switch – Case kod bloğudur. Daha önce İf – Else – Else İf koşul blokları ile yapabildiğimiz her şeyi, Switch – Case yapısı ile de yapabiliriz. Switch kodu hangi değerin doğruluğunun kontrol edileceğini belirtilen koddur.

Örneğin atamış olduğumuz a = 1 şeklinde bir değeri, switcch kodunun içerisine yazarsak " switch (a) " bu a değerinin doğruluğunun kontrol edileceği anlamına gelir.

Case kodu ise a değeri için 'true (doğru)' değeri sağlandığı zaman çalıştırılacak olan koddur. Eğer yanlış ise diğer seçenekler için birden fazla case kodu oluşturabiliriz.



Switch kodundan sonra açılan "{ }" kod bloğunun içerisindeki tüm case'ler, a değerini kontrol edilmesi için yazılan case'lerdir.

Burada farklılığına değinilmesi gereken nokta ise 'Break' (İngilizcede kırmak anlamına gelir) komutudur. Break komutu " eğer bu case doğru (true) ise diğer case'leri çalıştırma" anlamı taşır. Eğer break komutunu koyulmaz ise ilk case doğru olup çalıştırıldıktan sonra kodların çalışması durdurulmaz ve ikinci gelen case'i de çalıştırır.

Diğer bir bahsedilmesi gereken kod ise ‘Default’ kodudur. Eğer case kod bloklarının hiçbiri sağlanmazsa diyerek son olarak default kodunu ekliyoruz. Hiçbir case kontrolünde true (doğru) karşılığını elde edemezsek, çalıştırılmasını istediğimiz kod bloğu bu kısımda yer alıyor.

```
    int d;
    cout << "lutfen bir sayi giriniz."<< endl;
    cin >> d;

    switch (d) {

        case 5: {
            cout << "sayi 5'e esittir." << endl;
            }
            break;
        case 10: {
            cout << "sayi 10'a esittir." << endl;
            }
            break;
        case 15: {
            cout  << " sayi 15'e esittir." << endl;
            }
            break;
            default: {
                cout << "sayi 5, 10 ve 15'e esit degildir." << endl;
                }

        }

        return 0;
}
```

```
./switchcase
switch case kosul ifadesi
lutfen bir sayi giriniz.
4
sayi 5, 10 ve 15'e esit degildir.
Time elapsed: 000:17:316
Press any key to continue
```

MİNİ PROJE (MANTIK İŞLEMLERİ – LOGİC OPERATORS) = “Klavyeden girilen 2 sayıyı alarak mantıksal olarak karşılaştıran kod örneğini yazınız. Karşılaştırmalar için kullanılacak olan operatörler şunlardır: == , != , < , > , <= , >= ”

Kodlarımızı yazarken bütün olasılıklar değerlendirilerek yazılacaktır. Bundan dolayı ekran çıktısında aynı anlama gelen birden fazla cümle öbeği olacaktır. Böylece bütün mantık işlemlerini uygulamış olacağız.

```cpp
#include <iostream>
using namespace std;
int main()

{
    cout << "Mantiksal islemler projesi" << endl;

    int b,r ;

    cout << "birinci sayiyi giriniz." << endl;
    cin >> b;
    cout << "ikinci sayiyi giriniz." << endl;
    cin >> r;

    if (b == r) {
        cout << "sayilar biribirine esittir." << endl;
        }

        else {
        cout << "sayilar birbirine esit degildir." << endl;
            }

    if (b != r) {
        cout << "sayilar birbirinden farklidir." << endl;
        }

    else {
        cout << "sayilar birbirinden farkli degildir." << endl;
        }

    if (b > r) {
        cout << "birinci sayi ikinci sayidan buyuktur." << endl;
        }
```

```
    if (b > r) {
        cout << "birinci sayi ikinci sayidan buyuktur." << endl;
        }

    else {
        cout << "birinci sayi ikinci sayidan buyuk degildir." << endl;
        }

    if (b < r) {
        cout << "birinci sayi ikinci sayidan kucuktur." << endl;
        }

    else {
        cout << "birinci sayi ikinci sayidan kucuk degildir." << endl;
        }

     if (b >= r) {
         cout << "birinci sayi ikinci sayiya buyuk eşittir." << endl;
         }

     else {
         cout << "birinci sayi ikinci sayiya buyuk esit degildir." << endl;
         }

     if (b <= r) {
         cout << "birinci sayi ikinci sayiya kucuk esittir." << endl;
         }

    else {
         cout << "birinci sayi ikinci sayiya kucuk esit degildir." << endl;
        }

    return 0;
```

```
./projekosullarmantik
Mantiksal islemler projesi
birinci sayiyi giriniz.
5
ikinci sayiyi giriniz.
10
sayilar birbirine esit degildir.
sayilar birbirinden farklidir.
birinci sayi ikinci sayidan buyuk degildir.
birinci sayi ikinci sayidan kucuktur.
birinci sayi ikinci sayiya buyuk esit degildir.
birinci sayi ikinci sayiya kucuk esittir.
Time elapsed: 000:13:307
Press any key to continue
```

MİNİ PROJE (SAYI ARALIĞI) = “Klavyeden 3 sayı alarak mantık bağlaçları ile karşılaştıran kod örneğini yazınız.” Örneğin; klavyede yazılan sayılar a, b ve c olsun. Aşağıdaki durumların karşılaştırılması baz alınarak kodlar yazılmalı:

- a’nın b ve c arasında olup olmadığı
- a’nın b’ye eşit ve aynı zamanda c’den küçük olup olmadığı
- a’nın b’den veya c’den küçük olup olmadığı
- 3 sayının birbirine eşit olup olmadığı

İlk koşulun sağlanması için yazılan kodlar:

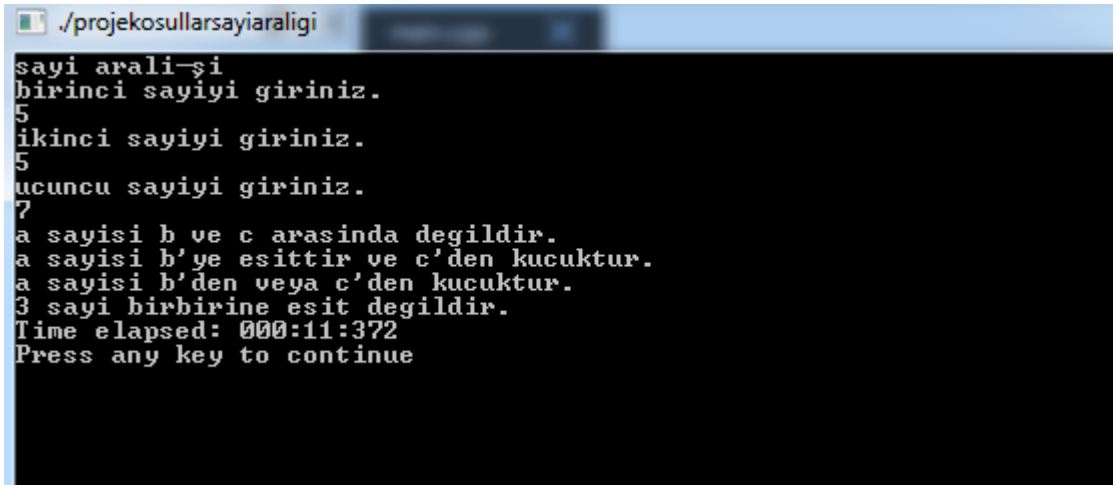
```
    int a,b,c;

    cout << "birinci sayiyi giriniz." <<endl;
    cin >> a;
    cout << "ikinci sayiyi giriniz." <<endl;
    cin >> b;
    cout << "ucuncu sayiyi giriniz." <<endl;
    cin >> c;

    if (a>b && c>a || a<b && a>c) /* b -- a -- c || c -- a -- b (a'nın b ve c'nin arasında
olma ihtimalleri*/
        {
            cout << "a sayisi b ve c arasindadir." <<endl;
        }
    else{
            cout << "a sayisi b ve c arasinda degildir." <<endl;
        }
```

- Eğer ‘if’ kodunun içinde çalışacak olan tek bir kodunuz var ise süslü parantez koymaya gerek yoktur.
- Kodları yazarken yapabileceğiniz iki tür hata tipi vardır:
  1. Syntax Error : Yazım hatasıdır.
  2. Logic Error : Mantık hatasıdır.

İkinci, üçüncü ve dördüncü şartların sağlanması için yazılan kodlar:

```
        }
    else{
            cout << "a sayisi b ve c arasinda degildir." <<endl;
        }
    if (a == b && c>a)
        {
            cout << "a sayisi b'ye esittir ve c'den kucuktur." <<endl;
        }
    else {
            cout << "a sayisi b'ye esit degildir veya c'den kucuk degildir." <<endl;
        }
    if (b>a || c>a)
        {
            cout << "a sayisi b'den veya c'den kucuktur." <<endl;
        }
    else {
            cout << "a sayisi b'den ve c'den kucuk degildir." <<endl;
        }
    if (a == b && a == c) // eğer a b'ye ve c'ye eşit ise zaten b'de c'ye eşittir.
        {
            cout << "3 sayi birbirine esittir." <<endl;
        }
    else {
            cout << "3 sayi birbirine esit degildir." <<endl;
        }

    return 0;
}
```

Klavyeden rastgele 5, 5 ve 7 sayılarını girdiğimiz zaman ekrana yansıyan sonuçlar:



MİNİ PROJE= “Klavyeden 3 sayı alarak aralarından en küçük ve en büyüğünü ekrana yazan kod örneğini yazınız.”

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "3 sayidan en buyugu ve kucugu" << endl;

    int x,y,z;

    cout << "birinci sayiyi giriniz." << endl;
    cin >> x;

    cout << "ikinci sayiyi giriniz." << endl;
    cin >> y;

    cout << "ucuncu sayiyi giriniz." << endl;
    cin >> z;

    if (x>y && x>z ) {
    cout << "en buyuk sayi:" << x;

    if (y>z) {
        cout << "en kucuk sayi" << z;
        }
        else {
        cout << "en kucuk sayi:"<< y;
            }
        }
```

```cpp
    if (y>x && y>z) {
        cout << "en buyuk sayi:" << y;

    if (x>z) {
        cout << "en kucuk sayi:" << z;
        }
        else {
        cout << "en kucuk sayi:" << x;
            }
        }

    if (z>x && z>y) {
        cout << "en buyuk sayi:" << z;

    if (x>y) {
        cout << "en kucuk sayi:" << y;
        }
        else {
        cout << "en kucuk sayi:" << x;
            }
        }


    return 0;
}
```

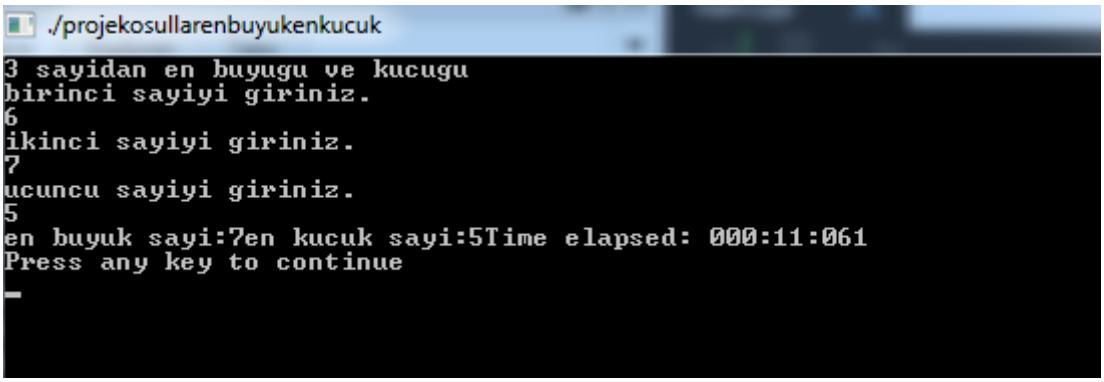
Ekranda görüntüsü:

```
./projekosullarenbuyukenkucuk
3 sayidan en buyugu ve kucugu
birinci sayiyi giriniz.
6
ikinci sayiyi giriniz.
7
ucuncu sayiyi giriniz.
5
en buyuk sayi:7en kucuk sayi:5Time elapsed: 000:10:047
Press any key to continue
```

Aynı zamanda aynı kodları farklı şekillerde de yazmak mümkündür. Örneğin yukarıda yazmış olduğumuz kodları bir de farklı şekilde yazmayı deneyelim:

```
*/
cout << "3 sayidan en buyugu ve kucugu" << endl;

int x,y,z;

cout << "birinci sayiyi giriniz." << endl;
cin >> x;

cout << "ikinci sayiyi giriniz." << endl;
cin >> y;

cout << "ucuncu sayiyi giriniz." << endl;
cin >> z;

int enbuyuk = x;

if (y > enbuyuk) {
    enbuyuk=y;
    }
if (z > enbuyuk) {
    enbuyuk=z;
    }
int enkucuk = x;

if (y < enkucuk) {
    enkucuk=y;
    }
if (z < enkucuk) {
    enkucuk=z;
    }
    cout << "en buyuk sayi:" << enbuyuk;
    cout << "en kucuk sayi:" << enkucuk;
return 0;
```

Ekrana yansıyan görüntü diğeri ile aynı olacaktır:



Bir kodun iyi yazılabilmesi demek, o kodun en kısa ve anlaşılır olarak nasıl yazılacağını bilinmesi demektir. Bu da kendinizi test ederek, aynı kodu farklı yazım şekilleri ile öğrenebilmenizi gerektirir. Bu yüzden de örnek kodları yazarken mümkün olduğunca fazla alternatifi barındırmaya çalıştık.

Son olarak en başta belirttiğimiz üzere, sırayla if – else if ve else olarak düzenlediğimiz kodların üçüncü farklı yazım şekli:

```
*/
        //üçüncü gösterim tarzı

        int enbuyuk = x;

        if (x>y && x>z) {
        enbuyuk= x;
            }
        else if (y>x && y>z) {
        enbuyuk= y;
            }
            else {
                enbuyuk=z;
                }
        cout << "en buyuk sayi:" << enbuyuk;

        int enkucuk = x;

        if (x<y && x<z) {
        enkucuk= x;
            }
        else if (y<x && y<z) {
        enkucuk= y;
            }
            else {
                enkucuk=z;
                }
        cout << "en kucuk sayi:" << enkucuk;

    return 0;
}
```

```
./projekosullarenbuyukenkucuk
3 sayidan en buyugu ve kucugu
birinci sayiyi giriniz.
6
ikinci sayiyi giriniz.
7
ucuncu sayiyi giriniz.
5
en buyuk sayi:7en kucuk sayi:5Time e
Press any key to continue
```

MİNİ PROJE (İŞÇİ PROBLEMİ): “Bir işçinin işi kaç bitirme süresini ve toplam işçi sayısını alarak işin bitme süresini hesaplayan örnek kodları yazınız.”

Örneğin bir işçi işi 10 günde bitiriyorsa, aynı işi 2 işçi kaç günde bitirir gibi bir problemin çözümünü kodlar yardımı ile hesaplatacağız.

```
using namespace std;
int main()
{
    //birinci hesaplama örnek kodu
    cout << "Isci problemleri ve basit hesaplamalar" << endl;

    cout << "Bir isci isi kac gunde bitiriyor?" << endl;
    // o işin kaç günde yapıldığı
    int kacgun, iscisayisi;
    cin >> kacgun;

    cout << "Toplam kac isci calisacak?" << endl;
    // ve kaç kişi ile yapıldığı
    cin >> iscisayisi;

    float sonuc = kacgun / iscisayisi; /* float olarak atamamızın nedeni işlemin
     * sonucu ondalıklı bir sayı da çıkabilmesidir.*/
    cout << "isin bitme suresi" << sonuc << "gundur.." <<endl;
```

```
./projekosullarproblemler
Isci problemleri ve basit hesaplamalar
Bir isci isi kac gunde bitiriyor?
10
Toplam kac isci calisacak?
2
isin bitme suresi5gundur..
Time elapsed: 000:03:760
Press any key to continue
```

aynı sonucu aşağıdaki şekilde, ekrana yazdırırken de hesaplatabilirdik:

```
cout << "isin bitme suresi" << (float) kacgun/iscisayisi << "gundur." <<endl;
```

ve bu şekilde sadece tek bir satırda işlemi bitirmiş olurduk.

MİNİ PROJE (HİPOTENÜS- ALAN-ÇEVRE HESAPLAYAN KOD): “Bir dik üçgenin iki dik kenarını kullanıcıdan alarak, hipotenüsü, üçgenin alanını ve çevresini hesaplayan örnek kodu yazınız.”

Hipotenüs, bir dik üçgenin en uzun kenarıdır. Ve diğer iki kenarın karesi alınıp toplanır ve daha sonra karekökü alınarak hesaplanır.

$$c = \sqrt{(a^2 + b^2)}$$

Bu tarz kullanılacak olan matematiksel işlemleri (karekök alma, küpünü almak gibi) 4 işlem ile değil de, direk kısayoldan yapabilmek için, C ve C++ dilinde kullanılan #include <math.h> kütüphanesinden sqrt() fonksiyonunu çalışma alanınıza ekleyerek (kodları yazmadan önce) kullanabilirsiniz.

```
#include <iostream>
#include <math.h>

using namespace std;
int main()
{
    cout << "Hipotenus hesaplama" << endl;

    int a,b,c;
    /*atamış olduğumuz c değerini hipotenüs olarak
    belirledik. Seçime göre değişebilir tabi ki.*/

    cout << "bir dik kenar giriniz:" << endl;
    cin >> a;

    cout << "diger dik kenar giriniz:" << endl;
    cin >> b;

    c = sqrt(a*a + b*b);
    cout << "hipotenus:" << c << endl;


    return 0;
}
```

```
./projekosullarhipotenus
Hipotenus hesaplama
bir dik kenar giriniz:
3
diger dik kenar giriniz:
4
hipotenus:5
Time elapsed: 000:07:816
Press any key to continue
```

Çevresi için almış olduğumuz iki kenar ve bulmuş olduğumuz hipotenüs ile toplamalıyız. Alanını hesaplamak için ise hali hazırda kullanıcıdan almış olduğumuz iki dik kenarı birbiri ile çarpmamız ve ikiye bölmemiz gerekmektedir. Bunun için yazılması gereken örnek kod:

```cpp
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    cout << "Hipotenus hesaplama" << endl;

    int a,b,c;
    /*atamış olduğumuz c değerini hipotenüs olarak
    belirledik. Seçime göre değişebilir tabi ki.*/

    cout << "bir dik kenar giriniz:" << endl;
    cin >> a;

    cout << "diger dik kenar giriniz:" << endl;
    cin >> b;

    c = sqrt(a*a + b*b);
    cout << "hipotenus:" << c << endl;

    cout << "ucgenin cevresi:" << a + b + c << endl;

    cout << "ucgenin alani:" << (float)a*b/2 << endl;
    /* değer ikiye bölündüğü için ondalıklı çıkabilir. Bu yüzden
    float olarak belirtiyoruz. */

    return 0;
}
```

```
./projekosullarhipotenus
Hipotenus hesaplama
bir dik kenar giriniz:
3
diger dik kenar giriniz:
4
hipotenus:5
ucgenin cevresi:12
ucgenin alani:6
Time elapsed: 000:05:070
Press any key to continue
```

MİNİ PROJE (YOL-HIZ PROBLEMLERİ): “Mesafeyi ve hızı alarak süreyi hesaplayan kodu yazınız.”

Örneğin; İstanbul ve Ankara arası 400 km olarak ölçülmektedir. Bu yolu ortalama olarak 120 km/h hızla giden bir sürücü ne kadar sürede hedefe varır?

Fakat burada dikkat edilmesi gereken bir dakika kısmı vardır. Normalde ilk akla gelen şekilde bölme yaparak kodlarımızı yazarsak;

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Yol - Hiz problemleri" << endl;

    int mesafe, hiz;

    cout << "mesafeyi giriniz:" << endl;
    cin >> mesafe;

    cout << "hizinizi giriniz:" << endl;
    cin >> hiz;

   cout << "tahmini varis sureniz:" << (float)mesafe/hiz << endl;


    return 0;
}
```

```
./projekosullaryolhiz
Yol - Hiz problemleri
mesafeyi giriniz:
400
hizinizi giriniz:
120
tahmini varis sureniz:3.33333
Time elapsed: 000:05:865
Press any key to continue
```

Yukarıda görüldüğü üzere ilk akla gelen bölme işlemi ile yaptığımız zaman her ne kadar tahmini varış süresini saat olarak bulabilmiş olsa da, dakika kısmı tam olarak kullanıcının anlayabileceği şekilde net değil. Yani bizim 3.33333 olarak bulduğumuz değerde 3 saatte (tamsayı kısmı) varış süresi olduğunu anlayabiliyoruz. Fakat dakika kısmını ifade eden ondalık kısmını da anlaşılır bir vaziyette yazmamız ve bunun için dakika dönüşümü yapmamız gerekir.

1. Öncelikle sayının tamsayı kısmı ile ondalık kısmını ayırmalıyız. Hali hazırda tam sayı kısmı saati ifade ediyordur zaten.
2. Daha sonra ondalık kısmını saat sistemine göre ayarlayabilmek için 60 ile çarpmamız gereklidir. 60 ile çarpmamızın anlamı, sayıyı yüzlük sistemden 60 dakikalık saat sistemine çevirecek olmamızdan kaynaklanır.

Böylece kodumuz daha anlaşılır olarak kullanıcıya sunulabilir hale gelecektir. Örnek kodumuz aşağıdaki gibidir:

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Yol - Hiz problemleri" << endl;

    int mesafe, hiz;

    cout << "mesafeyi giriniz:" << endl;
    cin >> mesafe;

    cout << "hizinizi giriniz:" << endl;
    cin >> hiz;

    int saat = mesafe/hiz; //eğer sonuç 3.333 gibi ise 3 olarak görünecektir.

    float dakikakismi = (float)mesafe/hiz - (int)mesafe/hiz;
    int dakika= dakikakismi * 60;

    cout << "tahmini varis sureniz:" << (int)saat << "saat ve" << dakika << "dakikadir."<< endl;


    return 0;
}
```

```
./projekosullaryolhiz
Yol - Hiz problemleri
mesafeyi giriniz:
400
hizinizi giriniz:
120
tahmini varis sureniz:3saat ve19dakikadir.
Time elapsed: 000:04:758
Press any key to continue
```

MİNİ PROJE (DİK ÜÇGEN KONROLÜ): “Kullanıcıdan 3 adet sayı alarak, bu sayıların bir dik üçgen kenar uzunlukları olup olmadığını kontrol eden örnek kodu yazınız.”

Bu kodu yazabilmek için yardım alabileceğiniz Pisagor bağıntısı aşağıdaki gibidir. Eğer verilen 3 kenar bu denklemi sağlıyor ise, verilen kenarlar bir dik üçgenin kenarlarıdır.

$$c^2 = a^2 + b^2$$

```cpp
using namespace std;
int main()
{
    cout << "Dik ucgen kontrolu" << endl;

    int a,b,c; // c yi hipotenüs olarak düşünelim bağıntıyı kullanırken

    cout << "ucgenin kenarlarini giriniz:" << endl;
    cin >> a;
    cin >> b;
    cin >> c;

    if (a*a + b*b == c*c) {
        cout << "dik ucgendir" << endl;
        }

        /* eğer kullanıcı yanlışlıkla hipotenüsü en son girmez ise
         * diyerek bir mekanizma geliştirmek için
         * if kodunda yazdığımız kodu, diğer
         * olabilecek ihtimaller içinde şu şekilde geliştirebiliriz:
         * (a*a + b*b == c*c || b*b + c*c == a*a || c*c + a*a == b*b)
*/
    else {
        cout << "dik ucgen degildir" << endl;
        }

    return 0;
}
```

```
./projekosullardikucgenkontrol
Dik ucgen kontrolu
ucgenin kenarlarini giriniz:
3
8
11
dik ucgen degildir
Time elapsed: 000:15:678
Press any key to continue
```

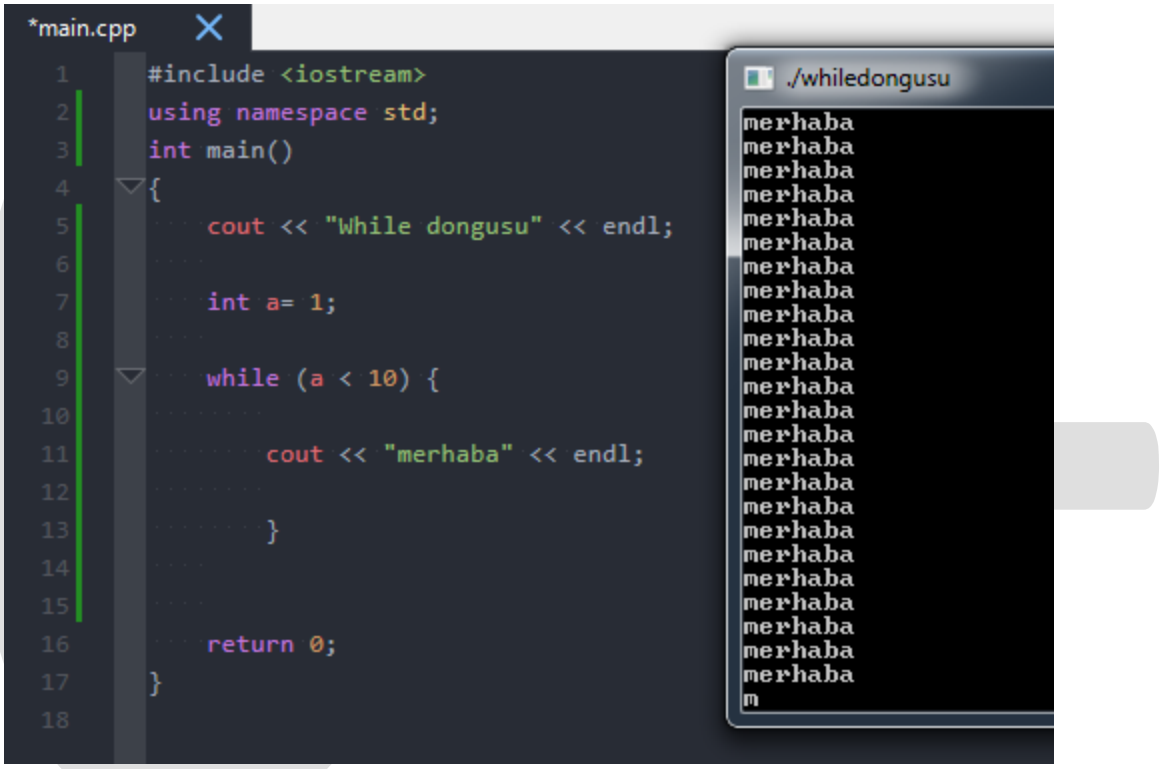
DÖNGÜLER (LOOPS)

Döngü ya da İngilizce de Loop olarak da geçen kavramın anlamı kod tekrarı demektir. Kodun sürekli olarak kendini tekrar ettirmesi de denebilir. Daha önceki bölümlerde açmış olduğumuz kod bloklarını (süslü parantezleri) bir if koşuluna bağlayarak çalışıp çalışmayacağını kontrol edebiliyorduk. Bundan sonraki bölümlerde bu kod bloklarını While - Do While - For gibi döngülere bağlayarak çalışacağız.

WHİLE DÖNGÜLERİ

While kodunun içerisine yazmış olduğumuz işlem ya da değer doğru olduğu sürece, o kod bloğunu sürekli olarak çalıştırmaya devam edecektir.

Bir örnek ile açıklamak gerekirse eğer, aşağıda atanmış olan a=1 değeri her zaman doğru olacağı için, while döngüsü sürekli olarak çalıştırılacaktır. Bu nedenle ekranda sürekli olarak yazılan bir merhaba yazısı görürüz. Bu gibi durumlara sonsuz döngü denir.


*main.cpp
#include <iostream>
using namespace std;
int main()
{
cout << "While dongusu" << endl;
int a= 1;
while (a < 10) {
cout << "merhaba" << endl;
}
return 0;
}
./whiledongusu
While dongusu
merhaba
merhaba
merhaba

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "While dongusu" << endl;

    int a= 1;

    while (a < 10) {

        cout << "merhaba" << endl;
        a++;
        cout << a << endl;
        }


    return 0;
}
```

```
./whiledongusu
While dongusu
merhaba
2
merhaba
3
merhaba
4
merhaba
5
merhaba
6
merhaba
7
merhaba
8
merhaba
9
merhaba
10
Time elapsed: 000:00:640
Press any key to continue
```

GÖRÜLDÜĞÜ ÜZERE A'NIN DEĞERİ SÜREKLİ OLARAK 1 ARTMIŞ VE EN SONUNDA 10 DEĞERİNİ ALARAK KOD ÇALIŞMAYI DURDURMUŞTUR.

FOR DÖNGÜLERİ

For döngüsünün while döngüsünden ayrılan yanı, while döngüsünde ayrı ayrı atadığımız değer ve yazdığımız kodları, for döngüsünde tek bir satırda yazabilmemizdir. Yani birbirlerinden ' ; (noktalı virgül) ' ile ayrılan aşağıdaki kod kümesi ortaya çıkmaktadır:

```cpp
// for döngüleri

for (int d = 1; d < 10 ; d++) {
    cout << "udemy" << endl;
    cout << d << endl;
    }

return 0;
}
```

```
1
udemy
2
udemy
3
udemy
4
udemy
5
udemy
6
udemy
7
udemy
8
udemy
9
Time elapsed: 000
Press any key to
```

HATIRLAYALIM!

Bir döngünün en temel üç özelliği içerisinde barındırdığı aşağıdaki değerlerdir:

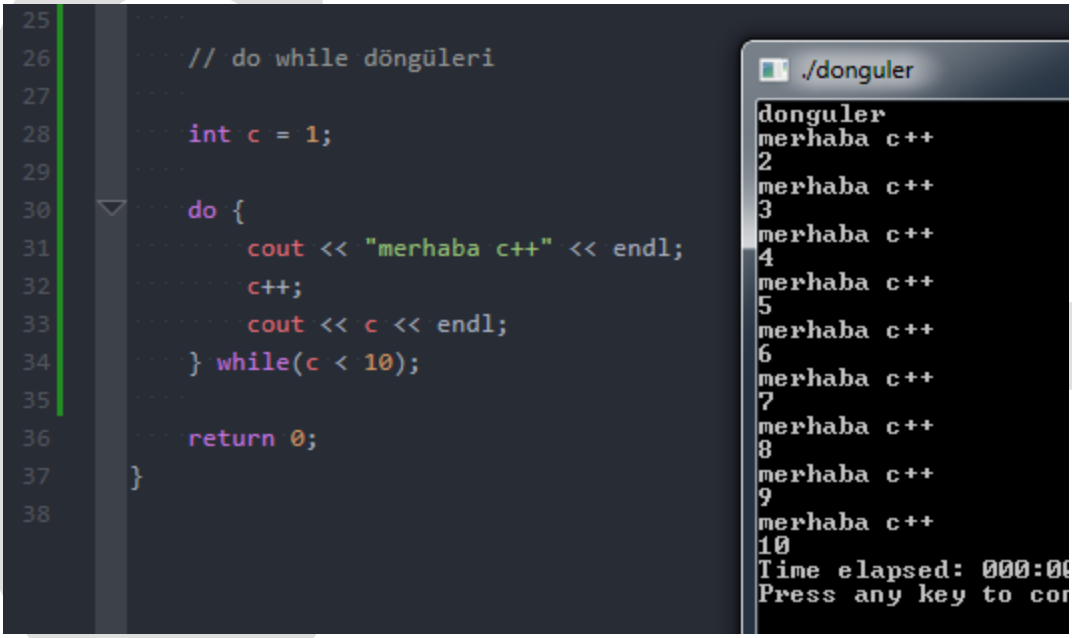
1. İlk değer
2. Koşul değeri
3. Adım değeri (döngüyü kısıtlamak için kullanılan kod)

DO WHİLE DÖNGÜLERİ

Do While döngüsü mantık olarak aynı olsa da, kod yazarken diğerlerinin formatında değildir. Do While döngüsünün diğerlerinden farklı koşulu en son belirtecek olmamızdır. Kodları yazarken öncelikle

1. Bir değer atıyoruz (int a = 5; gibi).
2. Do kod bloğu açıyoruz ve içerisine yazmak ya da kodlar çalıştırıldığı zaman ne yapılmasını istediğimizi yazıyoruz.
3. While kodunu açarak içerisine koşulumuzu belirtiyoruz.

Örnek kodumuz aşağıdaki gibidir:



Genel olarak baktığımızda bu döngüler arasındaki temel bir fark daha vardır. For döngüsü yazıldığı zaman atanan değer ilk kontrol edilir ve daha sonra ekrana basılır. Fakat do-while döngülerinde değer ekrana basıldıktan sonra kontrol edilir. Yani siz for ve while döngülerini birbirlerine çevirebilirken, for ve do while döngülerini birbirlerine net olarak çeviremezsiniz.

Örneğin döngüyü yazarken atadığımız değer d=100 olsaydı o döngüyü for döngüsü ile çalıştıramazdık çünkü ilk olarak değeri kontrol edecek ve koşul sağlanmadığı için döngüyü çalıştıramayacaktı. Fakat aynı şeyi do – while döngüsünde yaptığımız zaman yani c = 100 olarak değer atadığımız zaman, ilk önce kodu çalıştırıp sonrasında değeri kontrol ettiği için bir kereliğine ekranda “merhaba c++” yazısını görebilmiş olacaktık.

ÖRNEKLER

1: 1’den 100’e kadar olan tek sayıları ekrana bastıran örnek kodu ekrana bastırınız.

Bu kodu while döngüsü ile yazdığımızda örnek kodumuz aşağıdaki gibi olacaktır:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Donguler - Ornekler" << endl;

    // 1'den 100'e kadar olan tek syıları bastıran kod örneği

    int a=1;

    while (a<100){

        cout << a << " , " ;
        a += 2;

        /* arttırma operatörü: a++ (bir arttırma)
        eğer daha fazla arttırma istersek a += (sayı) şeklinde belirmeniz yeterlidir.
        Örneğin bu kodda 2 atlayarak a'ya değer ataması yapacaktır. */

        }


    return 0;
}
```

```
./orneklerdonguler
Donguler - Ornekler
1 , 3 , 5 , 7 , 9 , 11 , 13 , 15 , 17 , 19 , 21 , 23 , 25 , 27 , 29 , 31 , 33 ,
35 , 37 , 39 , 41 , 43 , 45 , 47 , 49 , 51 , 53 , 55 , 57 , 59 , 61 , 63 , 65 ,
67 , 69 , 71 , 73 , 75 , 77 , 79 , 81 , 83 , 85 , 87 , 89 , 91 , 93 , 95 , 97 ,
99 , Time elapsed: 000:00:811
Press any key to continue
```

Aynı kodu farklı şekillerde yazmakta mümkündür. Yazmış olduğumuz kodun açıklaması da yorum kısmında mevcuttur:

```
    // aynı kodun farkllı yazım şekli ise aşağıda verilmiştir.

    int b = 1;

    while (b < 100) {

        if (b%2==1) { /* kalan operatörünü kullandık. a sayısını yine birer birer
         * artmasını istedik fakat eğer a'yı 2'ye böldüğümüz
         * zaman kalan 1 olursa a'yı ekrana bastır dedik.*/
            cout << b << "," ;
            }
        b++;
        }

    return 0;
}
```

```
./orneklerdonguler
Donguler - Ornekler
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,5
7,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99,Time elapsed: 0
00:00:764
Press any key to continue
```

2: 50’den 70’e kadar olan çift sayıları ekrana bastıran örnek kodu yazınız.

```
    cout << "Donguler - Ornekler" << endl;

    for (int a=50; a < 70; a++) {

        if (a%2==0) {

            cout << a << " , " ;

        }
    }

    return 0;
}
```

```
./ornekdonguleriki
Donguler - Ornekler
50 , 52 , 54 , 56 , 58 , 60 , 62 , 64 , 66 , 68 , Time elapsed: 000:00:749
Press any key to continue
```

3: 100’den 70’e kadar olan ve 7’ye bölünebilen sayıları içeren örnek kodu ekrana bastırınız.

```
    for (int d=100; d>70; d--) {

        if (d%7==0) {

            cout << d << " , " ;

        }


    return 0;
}
```

```
./ornekdonguleriki
Donguler - Ornekler
98 , 91 , 84 , 77 , Time elapsed: 000:00:718
Press any key to continue
```

Yukarıda yazdığımız örnek kodda 100’den 70’e kadar bir azalma söz konusu olduğu için “d--” azaltma operatörünü kullandık.

4: 1 ile 100 arasındaki 3 ve – veya 7'ye bölünebilen sayıları ekrana bastıran örnek kodu yazınız.

```
    // dördüncü örnek

    //ve için yazılan kod (yani hem 3'e hemde 7'ye bölünebilen sayılar)

    for (int f=1; f< 100; f++) {

        if (f%3==0 && f%7==0) {

            cout << f << " , " ;

          }
        }
        cout << endl;
        cout << endl;
        cout << endl;
        cout << endl;// 4 satır boşluk bırakarak diğer kodu çalıştıracak

    //veya için yazılan kod (yani ya 3'e ya da 7'ye bölünebilen sayılar)

    for (int f=1; f< 100; f++) {

        if (f%3==0 || f%7==0) {

            cout << f << " , " ;

          }
    }

    return 0;
}
```

```
C:\Windows\syste
"----------Build
mingw32-make.exe
mingw32-make.exe
mingw32-make.exe
C:/TDM-GCC-64/bi
C:/TDM-GCC-64/bi
mingw32-make.exe
====0 errors, 0
```

```
./ornekdonguleriki
Donguler - Ornekler
21 , 42 , 63 , 84 ,


3 , 6 , 7 , 9 , 12 , 14 , 15 , 18 , 21 , 24 , 27 , 28 , 30 , 33 , 35 , 36 , 39 ,
 42 , 45 , 48 , 49 , 51 , 54 , 56 , 57 , 60 , 63 , 66 , 69 , 70 , 72 , 75 , 77 ,
 78 , 81 , 84 , 87 , 90 , 91 , 93 , 96 , 98 , 99 , Time elapsed: 000:00:842
Press any key to continue
```

5: Kullanıcıdan 5 sayı alarak bu sayıların ortalamasını hesaplayan ve ekrana bastıran örnek kodu yazınız.

Aşağıdaki kodları yazarken artık döngülerin yapısını bildiğimiz ve bunları kullanabileceğimiz için 5 defa "cin >>" ile kullanıcıdan sayı istemek yerine aşağıdaki kısayola başvuruyoruz:

Bir döngü içerisine herhangi bir değer atıyoruz (int a=0) ve bu değerin 5 kere tekrarlanabileceğini a<5 diyerek belirtiyoruz.

Değinmemiz gereken diğer bir nokta ise bu döngünün özelliğidir. Bu şekilde kullanıcıdan değer alınarak biriktirilen ve daha sonra işleme sokulan döngülere accumulator yani biriktirici döngü adı verilir.

Eğer sayıların ortalamasını almak istiyorsak, bunun formülü aşağıdaki gibidir:

$$\frac{sayıların\ toplamı}{kaç\ tane\ sayı}$$

Örnek kodumuz aşağıdaki gibidir:

```
        int toplam=0; /* toplamın ilk değeri 0'dır. eğer döngünün içinde tanımlansaydı,
        döngü her başladığında değeri tekrar sıfır olurdu ve toplam değerine ulaşılamazdı. */
        for (int a=0; a< 5 ; a++)
            {
            int okunandeger;
            cout << "lutfen bir sayi giriniz" << endl;
            cin >> okunandeger;
            toplam += okunandeger;
            /* diğer bir gösterimi=
             * toplam + okunandeger =(yeni)toplam
             * yani böylece her yeni okunan değer
             * o toplama eklenecek ta ki
             * 5 sayı girilene kadar*/
            }
            cout << "ortalamasi:" << toplam/5 << endl;

    return 0;
}
```

```
./ornekdonguleruc
Donguler - Ornekler
lutfen bir sayi giriniz
5
lutfen bir sayi giriniz
4
lutfen bir sayi giriniz
3
lutfen bir sayi giriniz
2
lutfen bir sayi giriniz
1
ortalamasi:3
Time elapsed: 000:43:649
Press any key to continue
```

6: Kullanıcıdan sayı değerleri alınız ve ortalamasını ekrana bastırınız. Ta ki kullanıcı ' -1 'sayısını girene kadar.

Aşağıdaki örnek kodumuzda kullandığımız for (;;) şeklinde kullanımı da doğrudur. Eğer belirtmeniz gereken herhangi bir ön koşulunuz bulunmuyor ise bu şekilde de kullanabilirsiniz.

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Ornekler - Donguler" << endl;

    int okunandeger=0;
    int toplam=0;
    int sayi=0;

    for (;;) // eğer bir koşulunuz yoksa bu şekilde kullanabilirsiniz
    {
    cout << "lutfen bir sayi giriniz:" << endl;
    cin >> okunandeger;

        if (okunandeger == -1)
            break; //eğer kullanıcı -1 girerse döngü kırılacak

    toplam += okunandeger;
    sayi++;
    }
        cout << "ortalamasi: " << toplam/sayi <<endl;

    return 0;
}
```

```
./ornekdongulerdort
Ornekler - Donguler
lutfen bir sayi giriniz:
1
lutfen bir sayi giriniz:
3
lutfen bir sayi giriniz:
4
lutfen bir sayi giriniz:
8
lutfen bir sayi giriniz:
-1
ortalamasi: 4
Time elapsed: 000:07:083
Press any key to continue
```

BREAK VE CONTINUE KOMUTLARI

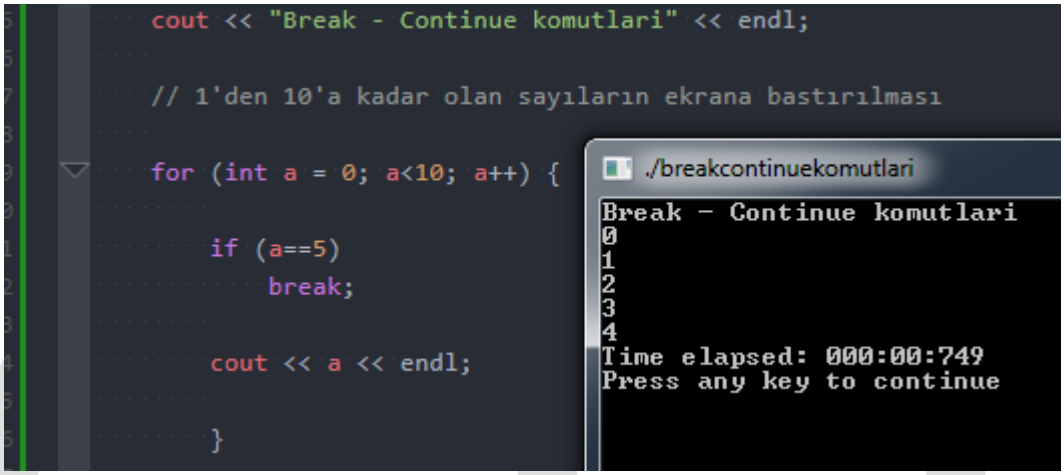
Koşullar bölümünde if koşulunu anlatırken, break (kır) komutuna giriş yapmıştık. Bu bölümde Break ve Continue komutlarını daha detaylı olarak anlatacağız.

Break komutu anlamındanda anlaşılacağı üzere bulunduğu kodu, koşulunu sağlamak şartı ile kırmakla görevlidir. Daha iyi anlatabilmek için kısa bir örnek yapalım. Bu örnekte 1'den 10'a kadar olan sayıları ekrana bastıralım:

```cpp
    cout << "Break - Continue komutlari" << endl;

    // 1'den 10'a kadar olan sayıların ekrana bastırılması

    for (int a = 0; a<10; a++) {

        cout << a << endl;

        }
```

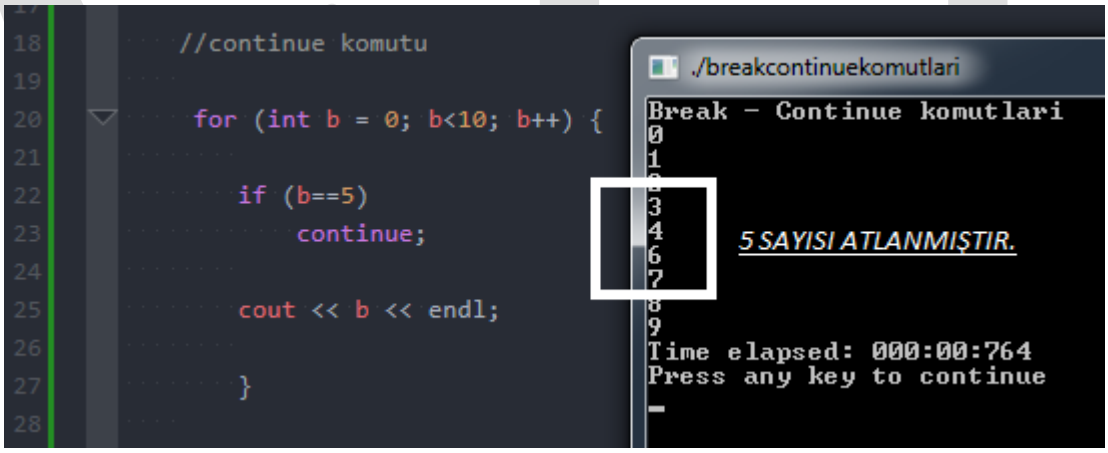
```
./breakcontinuekomutlari
Break - Continue komutlari
0
1
2
3
4
5
6
7
8
9
Time elapsed: 000:00:702
Press any key to continue
```

Fakat daha sonra buraya bir koşul ve break komutunu eklersek, koşul sağlanacağı zaman döngü durmuş olacaktır. Örnek kodumuz aşağıdaki gibidir:



Buna çok benzer olan fakat kodları yazarken tam tersi etkiye sahip olan continue komutu ise İngilizce de devam et anlamına gelir. Kodlarımızı yazarken eğer continue komutunu kullanırsak, bunun anlamı “bu koşul sağlandığında, bu adımı atla ama diğerlerinden devam et” demektir.

Örneğin yukarıdaki örnekte break komutu yerine continue komutunu kullanmış olsaydık, döngü kırılmayacaktı sadece koşulun sağlandığı, 5 sayısını atlayacak ve diğer sayılardan döngümüz devam edecekti. Örnek kodumuzun ve ekrana yansıyan görüntüsü aşağıdaki gibidir:



Döngüler konusundaki örneklerimizle devam edelim.

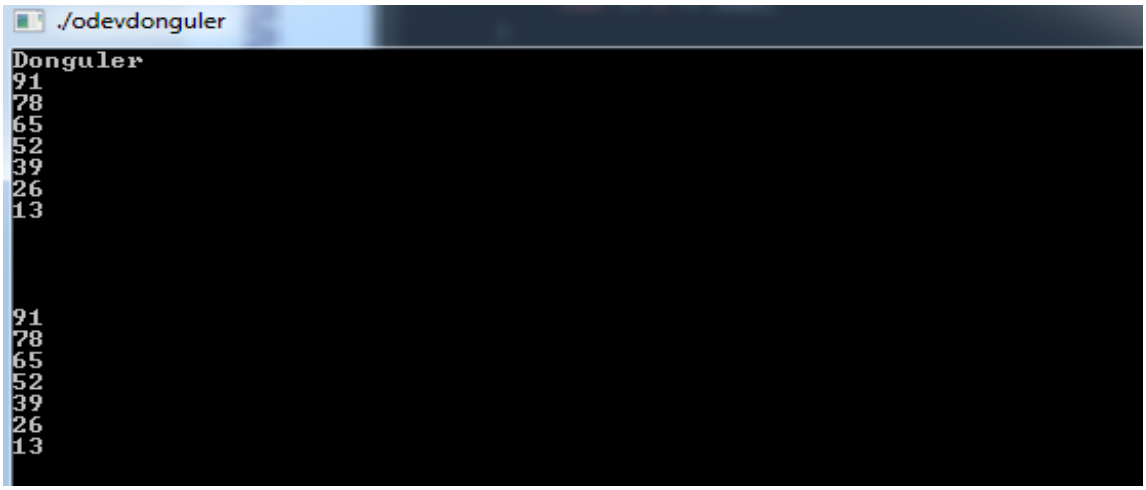
7: 100’den 0’a kadar olan ve 13’e tam bölünebilen sayıları ekrana bastırınız.

```
//ilk gösterim yolu

for (int a=100; a>0; a--) {

    if (a%13==0)
        cout << a << endl;
    }
```

```
 //ikinci gösterim yolu

 for (int b=91; b>0; b-=13) /* b-=13 anlamı b= b-13 tür. yani her atanan değerden 13 çıkarılacaktır.
  * ve ilk sayıyı 13 ün katı olan 91 olarak atadığımız için
  * her 13 eksilttiğimizde, oluşan sayı yine 13 ün katı olacaktır.
 */
     {
         cout << b << endl;
     }
```

```
// üçüncü gösterim yolu

//aynı kodların while döngüsü ile yazılmış hali

int d=100;
 while (d>0) {
     d--;
     if(d%13==0) {

         cout << d << endl;
         }

     }
```

Bütün kodlar aynı anda çalıştırıldığında ekranda yazılmış hali aşağıdaki gibidir. Bütün sonuçlar birbiri ile aynıdır:



8: Ekrana 4 kolon şeklinde aşağıdaki serileri bastıran örnek kodu yazınız.

1. Kolonda 1'den 100'e kadar olan 15'in katları
2. Kolonda 1'den 30'a kadar olan 5'in katları
3. Kolonda 100'den 50'ye kadar olan 10'un katları
4. Kolonda 2'den 64'e kadar olan 2'nin üsleri

Eğer bu kodları while döngüsü ile yazarsak, örnek kodumuz:

```
    int a=15; // 1'den 100'e kadar ilk 15'in katı 15 olduğu için
    int b=5;  // 1'den 30'a kadar ilk 5'in katı 5 olduğu için
    int c=100; // 100'den 50'ye kadar ilk 10'un katı 100 olduğu için
    int d=2;  // 2'den 64'e kadar ilk 2'nin katı 2 olduğu için

  while (a<100) //bir tanesi ile kontrol etmemiz yeterlidir.
      {
      cout << a << "\t" << b << "\t" << c << "\t" << d << endl;

      // " \t " kodu aralarında 3-4 satırlık boşluk bırakmak içindir.

      a += 15;
      b += 5;
      c -= 10;
      d *= 2;

      }
```

```
./odevdonguleriki
Donguler
15      5       100     2
30      10      90      4
45      15      80      8
60      20      70      16
75      25      60      32
90      30      50      64
Time elapsed: 000:00:733
Press any key to continue
```

Kolonlarımızı (dikey sütunlar) oluşturmak için, her bir ekrana yansıtılacak değerin arasına " \t " kodunu yazdık. Bu kod klavyemizde bulunan tab (caps lock tuşunun üzerindeki tuş) tuşu ile aynı göreve sahip olup, değerler arasına 3-4 satırlık boş yer bırakmamızı sağlar.

Aynı örnek kodu for döngüsü ile yazmayı denersek, örnek kodlarımızı adım adım inceleyelim:

```
for (a=1; a <= 6; a++) {
    }
```

1.Adım: For döngüsü için koşullarımızı yazarken, ilk olarak a değişkenin değerinin sürekli olarak 1 artacağını belirtiyoruz ve bu yüzden "a++" koşulunu yazıyoruz. Daha sonra bu artışın 6 ya eşit olduğunda son bulacağını belirtiyoruz "a <=6" ile. 6 ile son buluyor diye belirtmemizin sebebi, 1'den 100'e kadar olan 15'in katlarının 6 tane olduğunu bilmemizden kaynaklanıyor. Koşullarımızı yazarken en son olarak da "a=1" değerini atıyoruz. Çünkü bir sonraki adım da while döngüsünde olduğundan farklı bir yöntem kullanarak ekrana basılmasını sağlayacağız. Bu yöntem örneğin 1.satır için, a her bir arttığı zaman a'yı 15 ile çarpmak olacak yani;

a=1 iken a*15=15

a=2 iken a*15=30 gibi.

Ayrıca diğer while döngüsü ile yazmış olduğumuz kodda olduğu gibi koşul kısmında sadece bir tane değişkenin koşulunu belirtmemiz yeterlidir.

2.Adım:

```
*/
int us=2;

for (int a=1; a <= 6; a++)
    {

    cout << a*15 << "\t" << a*5 << "\t" <<  100-((a-1)*10) << "\t" << us << endl;
    us *= 2;

    }

return 0;
}
```

```
./odevdonguleriki
Donguler
15      5       100     2
30      10      90      4
45      15      80      8
60      20      70      16
75      25      60      32
90      30      50      64
Time elapsed: 000:00:484
Press any key to continue
```

Yazmış olduğumuz

cout << a*15 << "\t" << a*5 << "\t" << 100-((a-1)*10) << "\t" << us << endl;

kodunda 1 ve 2. Satır yukarıda bahsettiğimiz yöntem ile yazılmıştır. Her biri 15 ve 5 ile çarpılmıştır her döngü başladığında. 3. Satır için a sayısını 10'ar arttırmak isteseydik eğer (a-1)*10 (a'nın ilk değeri 1 olduğu için 1 çıkarttık) dememiz gerekirdi. Fakat 100'den 50'ye doğru geriye gittiğimiz için her sayıyı 100'den çıkartıyoruz ki döngü sondan başlayabilsin.

4.Satıra gelecek olursak henüz fonksiyonlar konusunu öğrenmediğimiz için, bir sayının üssünü alan fonksiyon kullanmak yerine farklı bir yola başvurduk. Bu yol için ilk önce döngünün dışında bir us adını verdiğimiz değişken tanımladık ve ona 2 değerini atadık. Döngünün içerisinde ise us değerinin her döngü bittiğinde kendini 2 ile çarpması için " us *= 2; " kodunu yazdık. Böylece tüm kolonlarımız tam olarak yazılmış oldu ve ekrandaki görüntüsünün while döngüsü ile bir farkı yoktur.

While döngüsü ile yazmış olduğumuz örnek kodlar her ne kadar daha kolay görünüyor olsa d, for döngüsünde kullanmış olduğumuz yöntem bilgisayarın diline daha yakındır. Bu yönteme sayılabilirlik (countability) adını veriyoruz. Döngülerin sayma teorisine (işlemsel olmasına) indirgenmesi anlamına geliyor.

9: Klavyeden bir sayı okuyarak, girilen sayı kadar fibonacci serisinin elemanlarını ekrana bastırınız.

Fibonacci serisinin ilk iki sayısı 1'dir. Ve diğer elemanları kendinden önce gelen son iki elemanın toplamı ile oluşmaktadır.

Fibonacci serisi= 1 1 2 3 5 8 13 21 ..

Kodları sürekli olarak güncelleyerek bu seriyi oluşturabilmek için 3 tane sayıya ihtiyacımız vardır. Yani a + b = c mantığını uygulayabilmemiz ve daha sonra bu işlemi kaydırmamız gereklidir. Birinci kolona a, ikinci kolona b ve üçüncü kolona ise c dersek;

* a b c
* 1 1 2
* 1 2 3
* 2 3 5

Burada sürekli olarak a=b ve b=c olmaktadır. Dolayısıyla biz de bu koşulu kendi kodlarımızda sağlamak için atamalıyız.

Örnek kodumuz aşağıdaki gibidir:

```
    int n;
    cout << "bir sayi giriniz" << endl;
    cin >> n;

    int a = 1;
    int b=1;

    /* kullanıcı 1, 0 veya negatif bir sayı girdiğinde ilk iki değeri
    otomatik olarak basmaması için aşağıda koşul belirttik. */

    if (n==1) {
        cout << 1 << endl;
        }
    else if (n <= 0);
    else {
    cout << a <<endl << b << endl;
    }

    for (int i=0; i < n-2; i++) /* n-2 yazmamızın sebebi bizim seriyi başlatmak için
    (kurala uymayan kısım- iki tane 1) ilk iki tane1'i kendimiz yazdırmamızdır. Yani
    kullanıcı 8 sayısını girdiği zaman ilk iki değerden sonra diğer 6 sayıyı basmasını sağlamış
    oluyoruz böylelikle. */
        {

        int c= a + b;
        a=b;
        b=c;
        cout << c << endl;
        }

    return 0;
}
```

```
./fibonacciserisi
fibonacci serisi
bir sayi giriniz
8
1
1
2
3
5
8
13
21
Time elapsed: 000:01:997
Press any key to continue
```

10: Kullanıcıdan bir sayı alıp, alınan sayı kadar sayıyı okuyunuz. Bu sayıların içerisindeki pozitif, negatif ve sıfır sayılarının oranını ekrana bastırınız.

Örneğin;

Kullanıcının girdiği değer: 6

Ve 6 tane de sayı giriyor: -1 , 6 , -3 , 2 , 4 , 0

Pozitif sayıların oranı: 0.5

Negatif sayıların oranı: 0.33

Sıfırların oranı: 0.16

Şeklinde ekrana bastırmamız isteniyor.

İlk olarak kullanıcıdan kaç sayı gireceğini alıyoruz ve negatif için esayi, pozitif için asayi ve sıfır için ssayi tamsayı değerlerini atıyoruz. Daha sonra koşulun içerinde bu sayıların 0'dan büyük olduğu durumda asayi, 0'dan küçük olduğunda esayi ve diğer durumlar için (0 olduğu durum) ssayi'yı arttırmamız gerektiğini koşul ifadesinin kod bloğunda belirteceğiz. For döngüsünün içerisine bunlara ek olarak, en başa bir integer değeri atıyoruz (bizim örneğimizde int g) ve bu değer okunan değeri temsil ediyor ki aşağıda koşulları yazarken karşılaştırabilelim. Kod bloğumuzu kapattıktan sonra ekrana yüzdelik değerlerini bastırabilmemiz için cout ve endl kodlarından yararlanıyoruz. Girilen sayılar arasında, belirtilen sayının (asayi, esayi ya da ssayi olabilir) yüzdeliği hesaplamak için ise sayı / girilen sayı olarak işlemi tanımladık. Bölme işleminin olduğu neredeyse her işlemde, ondalık bir sonuç çıkma ihtimali olduğu için işlemimizin başına ondalık değişken tipimiz olan float diye belirttik.

Örnek kodumuz kodların açıklaması ile birlikte aşağıdaki şekildedir:

```
int main()
{
    cout << "girilen pozitif - negatif - sifir orani" << endl;

    int n;
    cout << "lutfen kac sayi gireceginizi giriniz" << endl;
    cin >> n;
    int esayi=0, asayi=0 , ssayi=0; // eksi sayisi, artı sayısı ve sıfır sayısı

    for (int i=0; i<n; i++) {
    int g; // geçici olarak her döngüde okunan sayı
    cin >>g;

    if(g>0) {
        asayi++;
        }
    else if (g<0) {
        esayi++;
        }
    else {
        ssayi++;
        }
    }

    //bölme işlemi olduğu için float kullanıyoruz
    //sayılar arasında yüzdeyi bulabilmek için toplam sayıya (n) bölüyoruz
        cout << "pozitifler:" << (float)asayi/n << endl;
        cout << "negatifler:" << (float)esayi/n << endl;
        cout << "sifirlar:" << (float)ssayi/n << endl;
    return 0;
}
```

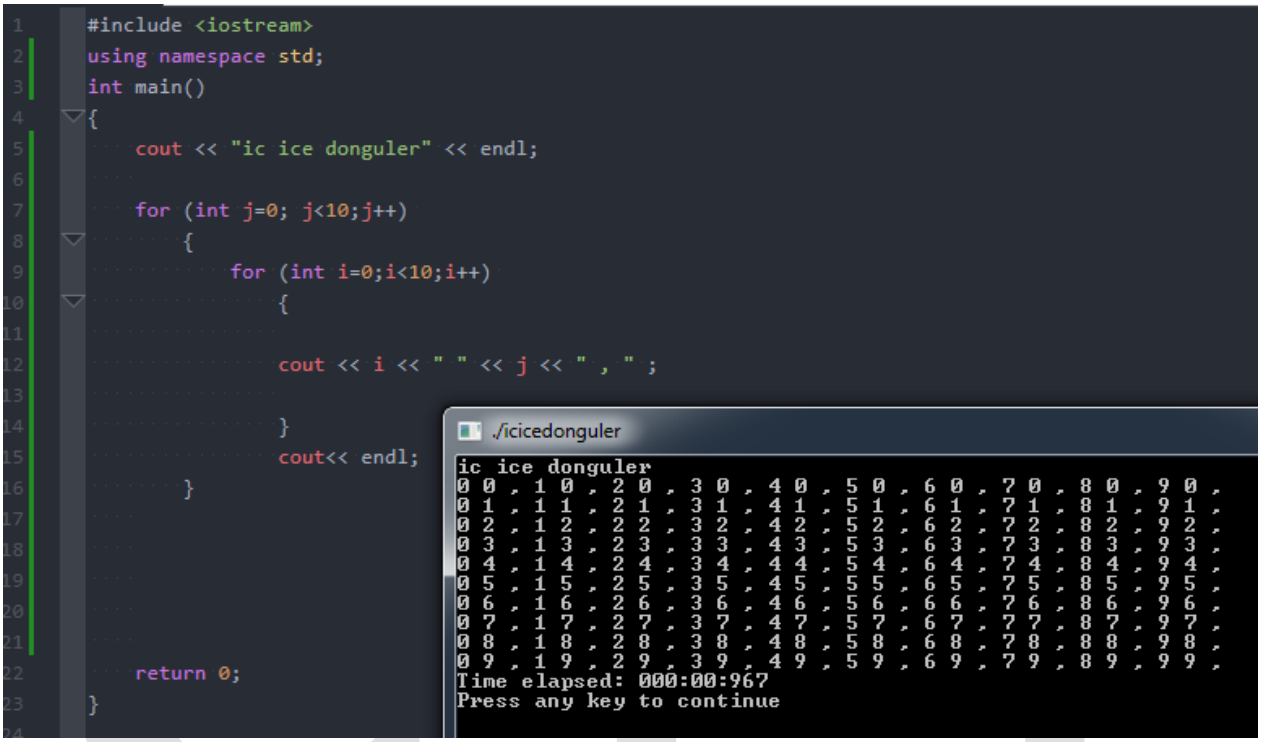
```
./dongulerpozitifnegatifsifir
girilen pozitif - negatif - sifir orani
lutfen kac sayi gireceginizi giriniz
6
-1 6 -3 2 4 0
pozitifler:0.5
negatifler:0.333333
sifirlar:0.166667
Time elapsed: 001:09:624
Press any key to continue
```

İÇ İÇE DÖNGÜLER

İÇ İÇE BİRDEN FAZLA DÖNGÜ OLUŞTURMA

Döngüler konusuna giriş yaptığımızda belirttiğimiz üzere, döngü bir şeyin tekrar etmesi anlamına gelir. İç içe döngülerden kastettiğimiz ise bu döngülerin de tekrar tekrar etmesi ile oluşur.

Bir örnek üzerinden anlatmak gerekirse örnek kodumuz aşağıdaki şekildedir:



Örneğimizde görüldüğü üzere kodlarımızda bir for döngünün içerisine bir değer atadık önce ve bu değerin 10 kere tekrar edeceğini koşulumuzda belirttik. Daha sonra bu for döngüsünün içerisine başka bir for döngüsü daha yazdık ve iç içe döngümüzü elde etmiş olduk. Aynı şekilde bu döngünün de 10 kere tekrar edilmesini " i<10 " diyerek belirttik. Burada dikkat edilmesi gereken husus şu ki, kodlarımız çalıştırıldığı zaman döngüler 10 * 10'dan 100 kere tekrar etmiş olduğunu gördük. Daha açıklayıcı olmak gerekirse, birinci döngüde 1 sayısı için ikinci döngü 10 kere tekrarlandı. Daha sonra birinci döngüde 2 sayısı için tekrar ikinci döngü 10 kere tekrarlandı ve bu şekilde aslında iç içe döngümüz ekranda da görüldüğü üzere 100 kere tekrarlanmış oldu.

Bu kodları yazarken bir nevi bir iki boyutlu tablo oluşturmuş olduk. Buradan yola çıkarsak, iç içe döngüler kodlarımızı yazarken herhangi tek boyutlu olan şeyi, birden fazla boyutlu olarak ekrana aktarmamız için bize yardımcı olur. Şimdiye kadar anlatılmış olan döngülerin hepsi tek boyutlu döngülerdi ve dolayısıyla tek boyutlu dizilerden söz ediyorduk (1'den 10'a kadar olan sayılar gibi tek bir dizi). İç içe döngülerde ise bu durum iki ve daha fazla boyutlu oluyor. Örneğin iki boyutludan kasıt burada, iki sıra dizi halinde bir tablo olarak ekrana bastırılmasıdır. İstenildiği sayıda iç içe döngü oluşturabilmektedir fakat bununla birlikte boyutunda arttığını göz önünde bulundurmalısınız.

ÖRNEKLER

1: Ekrana çarpım tablosunu bastıran örnek kodu yazınız.

Örneğin aşağıdaki 4*4'lük çarpım tablosu gibi:

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Örnek kodlarımız açıklaması ile birlikte kodlarımızdadır:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "ic ice donguler ornekler" << endl;

    for (int i=1; i<=4;i++)
    /* bizim çarpım tablomuz 1'den başaladığı için değeri 1'e atayarak başladık
     * ve ayrıca değerin 4'e <= olmasının sebebi yine çarpım tablomuzun
     * 4'ler kısmında bitiyor olmasıdır. */
        {

        for (int a=1;a<=4;a++)
            {

            cout << i*a << " , " ;
    // yan yana yazılmaları için endl koymuyoruz, eğer koyarsak satır aşağı kısıma geçer

            }
            cout << endl;
        }

    return 0;
}
```

```
./icicedongulerornekbir
ic ice donguler ornekler
1 , 2 , 3 , 4 ,
2 , 4 , 6 , 8 ,
3 , 6 , 9 , 12 ,
4 , 8 , 12 , 16 ,
Time elapsed: 000:00:655
Press any key to continue
```

Aynı kod örneğini yazarken, bir değişiklik yapsaydık ve kullanıcının bize girdiği sayıya kadar çarpım tablosunun oluşturulmasını isteseydik;

Örneğin i<=4 dediğimiz yere sayısal bir değer atamazdık. Onun yerine kullanıcıdan alınacak bir değerin gelmesini sağlardık.

Örnek kodlarımızın aşağıdaki gibi düzenlemesi gerekirdi:

```
    int boyut;
    cout << "lutfen bir sayi giriniz.." <<endl;
    cin >> boyut;
    for (int i=1; i<=boyut; i++) {

        for (int a=1; a<=boyut;a++) {

            cout << i*a << " , " ;

            }
        cout << endl;
        }

    return 0;
}
```

```
./icicedongulerornekbir
ic ice donguler ornekler
lutfen bir sayi giriniz..
6
1 , 2 , 3 , 4 , 5 , 6 ,
2 , 4 , 6 , 8 , 10 , 12 ,
3 , 6 , 9 , 12 , 15 , 18 ,
4 , 8 , 12 , 16 , 20 , 24 ,
5 , 10 , 15 , 20 , 25 , 30 ,
6 , 12 , 18 , 24 , 30 , 36 ,
Time elapsed: 000:04:727
Press any key to continue
```

2: Kullanıcıdan bir sayı alarak, bu sayı boyutu kadar ekrana ters üçgen bastırınız

Görüntünün aşağıdaki gibi olması bekleniyor:

```
* * * *
  * * *
    * *
      *
```

Burada yazacağımız kod için ilk olarak şekli analiz etmemiz gerekiyor. Yanda örnek olarak gösterdiğimiz şekil, kullanıcının 4 sayısını girmesi ile oluşturulacak olan şekildir. İlk satırda 4, sonra 3 ve daha sonra da 2 ve 1 şeklinde azalan bir yapıya sahip. Ayrıca diğer bir analiz etmemiz gereken kısım ise üçgenimiz ters bir üçgen olduğu için, giderek artan boşluk sayılarıdır. İlk satırda hiç boşluk olmaması ile birlikte daha sonra 1 – 2 – 3 şeklinde artan boşluklar yer almıştır şeklimizde. Şeklimizin ekrana yansıtırken kullanabilmek açısından yıldız ya da çarpım karakterini kullanacağız.

Örnek kodumuzu inceleyecek olursak, boşluk sayısını belirleyecek olan for döngüsü için farklı olarak b<a koşulu yer almaktadır. Çünkü şayet a=1 ise ekrana hiç boşluk basılmayacaktır ya da a=2 ise sadece bir tane basılacaktır:

a=3 ise 2

a=4 ise 3 şeklinde artarak ilerleyen yapıyı sağlamış olacağız.

Yıldız sayımızı belirleyen kod bloğunda ise c < = boyut – a +1 şeklinde bir koşul yer aldı. Kullanıcının girdiği boyuttan a tamsayı değerini (her döngüde birer artan) çıkarttık. Bunun sebebi yıldız sayısının azalan bir yapıya sahip olabilmesi içindir. Bu koşul her ne kadar istediğimiz gibi azalan bir yapıya sahip olsa da örnek değerler verildiği zaman görülüyor ki, bizim girdiğimiz boyuttan bir eksik şekilde yıldızı ekrana yansıtılıyor. Bu yüzden koşulumuza +1 ifadesini de dahil ederek bu sorunu ortadan kaldırmış olduk. Daha açıklayıcı olabilmek için sorunu bir örnekle ele almak gerekirse;

Eğer kullanıcı 5 değerini girerse: c (yıldız sayısı) <= 5 – 1 'den 4 tane yıldız basarak döngüye başlamış olacaktı. Daha sonra da:

5-2=3

5-3=2

5-4=1 yıldız basarak işlemi tamamlayacaktı. Örnekten anlaşıldığı üzere azalan yapıyı sağlarken, boyutu bir eksik olarak sağlamış oluyorduk ve +1 ekleyerek sorunu aşağıdaki gibi ortadan kaldırmış olduk:

(Kullanıcının yine boyut olarak 5 sayısını girdiği varsayılmıştır.)

C <= 5 – 1 + 1 =5

C <=5 – 2 + 1 =4

C <= 5 – 3 + 1 =3

C <= 5 – 4 + 1 =2

C <= 5 – 5 + 1 =1

Ekranda gösterilecek olan hali;

```
* * * * *
  * * * *
    * * *
      * *
        *
```

Örnek kodumuz aşağıdaki gibi olacaktır:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "ic ice donguler ornekler" << endl;

    int boyut;

    cout << "lutfen boyut bildiren bir sayi giriniz" << endl;
    cin>> boyut;

    for ( int a =1; a<=boyut; a++) //bu for döngüsü iki tane for'u kapsamaktadır
        {

    for( int b=1; b<a; b++) //ilk for döngüsü boşluk sayısını belirleyecek
                {
                cout << " ";
                }

    for ( int c=1; c<=boyut-a+1; c++) //ikinci for döngüsü ise yıldız sayısını belirleyecekti
                {
                cout << "*";
                }
                cout << endl;
        }

    return 0;
}
```

```
./icicedongulerornekiki
ic ice donguler ornekler
lutfen boyut bildiren bir sayi giriniz
5
*****
 ****
  ***
   **
    *
Time elapsed: 000:03:027
Press any key to continue
```

3: Ters köşegeni (anti diagon) 1 olan ve diğer bütün elemanları 0 olan matrisi ekrana bastırınız.

Ekrana bastırılması istenen görüntü aşağıda verilmiştir:

0 0 0 1

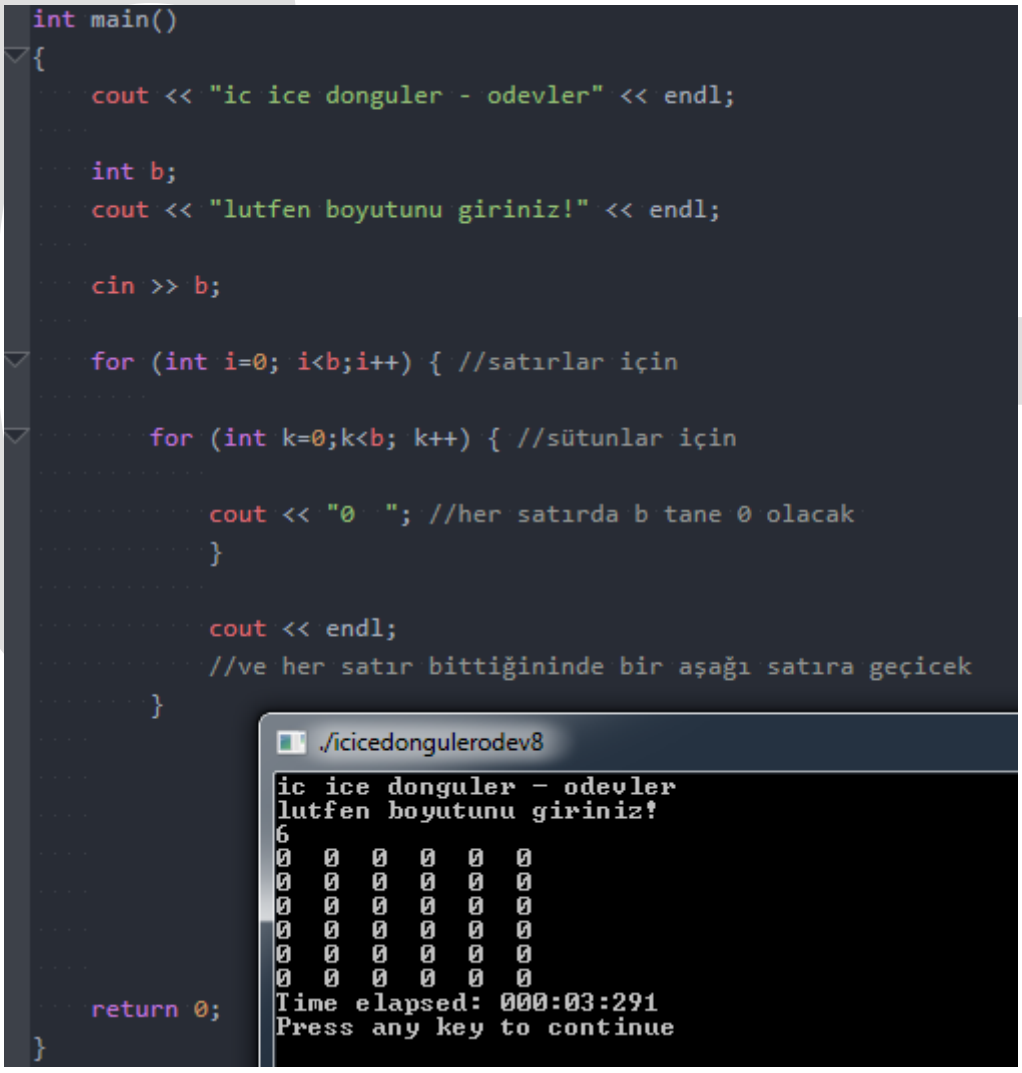
0 0 1 0

0 1 0 0

1 0 0 0

Yukarıda verilen örnekte 4'lük bir köşegen gösterilmiştir. Ancak ekrana bastırılmasını istediğimiz örnek için kullanıcıdan bir parametre alacağız ve kullanıcı hangi sayıyı girerse ona göre bir köşegenin ekrana bastırılmasını sağlayacağız.

Öncelikle bir matrisin nasıl basılması gerektiğini kavrayabilmek için sadece sıfırlardan oluşan bir matrisin örnek kodlarını inceleyelim.



Yukarıdaki örnekte görüldüğü üzere, kullanıcıdan alınan parametre değeri ile bir matrisi oluşturabilmek için iç içe iki tane döngüden yardım almaktayız. Bunlardan birincisi satırların ve ikincisi ise sütunların oluşması içindir. Bu satır ve sütunların eşit sayıda ve parametreye bağlı olması içinse i ve k tamsayı değerlerini, kullanıcıdan alınan parametre ile (b) bağlıyoruz.

"1"sayısını kullanarak bir köşegen yapabilmek için döngülerin içerisine bir koşul ifadesi atamamız gerekmektedir. Ve bu koşulda (satırları i ve sütunları da k olarak düşünürsek) i'nin k'ye eşit olduğu durumlar için 1 (ortasından geçmesini sağlıyor), diğer durumlar için ise "0" basmasını söylemeliyiz. Fakat örnekte bizden istenen ters köşegen olduğu için koşulumuzu farklı bir ifadeye bağlamamız gerekmektedir.

Analiz etmemiz gerekirse eğer, 4'lük bir ters köşegeni ele alalım (satırları i, sütunları k olarak düşünüyoruz).

0 0 0 1 (i=0 k=3)

0 0 1 0 (i=1 k=2)

0 1 0 0 (i=2 k=1)

1 0 0 0 (i=3 k=0)

Görüldüğü üzere 4'lük bir köşegen oluşturmak istediğimiz zaman, i ve k'nın toplam değeri, oluşturmak istediğimiz boyut değerinden bir eksiktir. Farklı boyutlardaki köşegenler için de denediğiniz zaman ifadenin değişmediğini görebilirsiniz.

O zaman ters köşegenin ekrana basılabilmesi için yazmamız gereken koşul i==k değil, i+k=b-1 şeklinde düzenlememiz gerekmektedir.

Örnek kodlarımız aşağıdaki gibidir:

```
cout << "ic ice donguler - odevler" << endl;

int b;
cout << "lutfen boyutunu giriniz!" << endl;

cin >> b;

for (int i=0; i<b;i++) { //satırlar için

    for (int k=0;k<b; k++) { //sütunlar için
        if (i+k==b-1)
            cout << "1  ";
            else
        cout << "0  "; //her satırda b tane 0 olacak
        }

        cout << endl;
        //ve her satır bittiğininde bir aşağı satıra geçicek
    }

 return 0;
```

```
./icicedongulerodev8
ic ice donguler - odevler
lutfen boyutunu giriniz!
6
0  0  0  0  0  1
0  0  0  0  1  0
0  0  0  1  0  0
0  0  1  0  0  0
0  1  0  0  0  0
1  0  0  0  0  0
Time elapsed: 000:09:329
Press any key to continue
```

Matris oluşumlarını daha iyi anlayabilmek için aynı örneği biraz değiştirerek, bu defa 1'lerin altında kalan tüm sayılarında 1 olduğu bir matrisi ekrana bastıralım.

Bunun için satır ve sütunları yine analiz ederek bir koşul oluşturmamız gerekirse, analizi aşağıdaki gibidir (4'lük bir köşegen ele alınarak yapılmıştır):

0 0 0 1 (i=0 k=3)

0 0 1 1 (i=1 k=2) (i=1 k=3)

0 1 1 1 (i=2 k=1) (i=2 k=2) (i=2 k=3)

1 1 1 1 (i=3 k=0)

Yukarıdaki analizden yola çıkılırsa, koşulu büyük eşittir işareti ile değiştirmemiz gerektiği sonucuna varmış oluruz. (i+k>=b-1)

Örnek kodlarımız aşağıdaki gibidir:

```
    cout << "ic ice donguler - odevler" << endl;

    int b;
    cout << "lutfen boyutunu giriniz!" << endl;

    cin >> b;

    for (int i=0; i<b;i++) { //satırlar için

        for (int k=0;k<b; k++) { //sütunlar için
            if (i+k>=b-1) // sadece ters köşegen için i+k==b-1
                cout << "1  ";
                else
            cout << "0  "; //her satırda b tane 0 olacak
            }

            cout << endl;
            //ve her satır bittiğininde bir aşağı satıra geçicek
        }

     return 0;
}
```

```
./icicedongulerodev8
ic ice donguler - odevler
lutfen boyutunu giriniz!
8
0  0  0  0  0  0  0  1
0  0  0  0  0  0  1  1
0  0  0  0  0  1  1  1
0  0  0  0  1  1  1  1
0  0  0  1  1  1  1  1
0  0  1  1  1  1  1  1
0  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1
Time elapsed: 000:04:430
```

4: Kullanıcıdan alınan sayı kadar boyuta sahip bir dik üçgeni ve ters dik üçgeni ekrana bastıran örnek kodu yazınız.

Sorulardan da anlaşıldığı üzere, bir dik üçgen, dörtgen gibi satır ve sütuna ya da daha genel bir ifade ile kullanıcıdan gelmesi gereken iki veri ile kurulan sistemlerde iki boyutlu oldukları için, iç içe döngüleri kullanıyoruz.

İlk olarak yazılacak olan örnek kodumuzu anlayabilmek açısından, kullanıcıdan değer okumadan, kendimiz değer vermişiz gibi düşünelim. Örneğin 5 boyutlu bir dik üçgenin ekrana basılmasını istiyoruz. İç içe döngümüzü temel kuralları ile yazdıktan sonra doğal olarak şayet i=0 ise i<5 dememiz gereklidir ki döngü 5 kere çalışarak yıldız basılmış olsun.

Kodlarımız ile göstermek gerekirse:

```
{
    cout << "ic ice donguler odevler" << endl;
    // dik üçgen ve ters dik üçgen yazılması
    //VİDEOLARDAKİ ÇÖZÜM (2.YOL)
    /*
     *
     * *
     * * *
     * * * *
     * * * * * şeklinde bir dik üçgen isteniyor.
     */

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

```
./icicedongulerodevbirbir
ic ice donguler odevler

*
**
***
****
Time elapsed: 000:00:546
Press any key to continue
```

Ekrana bastırdığımız zaman, gördüğümüz üzere, boyutu 5 olan bir dik üçgen beklerken, i=0 olarak başladığı zaman ilk satır 0 olacak şekilde üçgenimiz ekrana basılıyor. Bu sorunu j<i+1 diyerek (her satıra +1 yıldız eklenmesi) şeklinde çözüme kavuşturabiliriz.

Genel olarak kodlarımızın yapısı anlaşıldığı için geriye kullanıcıdan aldığımız değere göre bir dik üçgen bastırmak gerekiyor. Bunu da kendimizin koşul olarak belirttiğimiz i<5 'de 5 yerine bir değişken atayarak gerçekleştiriyoruz. Kullanıcıdan aldığı değere göre ekrana dik üçgen bastıran örnek kodumuz aşağıdaki gibidir

```
{
    cout << "ic ice donguler odevler" << endl;
    // dik üçgen ve ters dik üçgen yazılması
    //VİDEOLARDAKİ ÇÖZÜM (2.YOL)
    /*
     *
     * *
     * * *
     * * * *
     * * * * * şeklinde bir dik üçgen isteniyor.
     */
    int a;
    cout << "lutfen boyut belirten bir sayi giriniz" << endl;
    cin>>a;
    for (int i=0;i<a;i++)
        {
        for (int j=0;j<i+1;j++)
        {
            cout << "*";
        }
        cout << endl;
        }

    return 0;
```

```
./icicedongulerodevbirbir
ic ice donguler odevler
lutfen boyut belirten bir sayi giriniz
6
*
**
***
****
*****
******
Time elapsed: 000:02:325
Press any key to continue
```

Örnekte istenen ikinci örnek kodumuza gelecek olursak, ekrana aşağıdaki gibi bir ters üçgen bastıracağız:

```
        *   1 yıldız 4 boşluk
      * *   2 yıldız 3 boşluk
    * * *   3 yıldız 2 boşluk
  * * * *   4 yıldız 1 boşluk
* * * * *   5 yıldız
```

Normal bir dik üçgen bastırırken sadece yıldız sayılarına dikkat etmemiz gerekirken, ters dik üçgen yani tam tersi düzleme basılacak olan bir üçgen bastırılırken boşluk sayısını da dikkate almamız gerekmektedir. Dikkat edilmesi gereken diğer bir nokta ise (yukarıdaki gösterimde kullanıcının boyutunu 5 olarak girilmesi baz alınmıştır) yıldız ve boşluk sayılarının hepsinin birbirini 5'e yani kullanıcının girmiş olduğu boyut sayısına tamamlıyor (eşit) olmasıdır. Buradan yola çıkarak, az önceki örnekte girdiğimiz yıldız sayısını hatırlayalım. J<i+1 şeklinde bir yıldız sayısını belirten koşul yazılmıştı. i + 1 yıldız sayımızı belirttiğine göre, boşluk sayısın bulabilmemiz için n'den i+1'i çıkarmamız yeterli olacaktır yani boşluk belirtecek olan koşulumuzun son hali " n – (i + 1)".

Örnek kodumuz aşağıdaki gibidir:

```
 */
int n;
cout << "lutfen boyut belirten bir sayi giriniz" << endl;
cin>>n;
for (int i=0;i<n;i++)
    {
        for (int j=0; j<n-(i+1);j++)
            {
            cout << " ";
            }
        for (int j=0;j<i+1;j++)
            {
             cout << "*";
            }
        cout << endl;
    }
// ters dik üçgenin ekrana bastırılması
/*
       *
     * *
   * * *
 * * * *
* * * * * şeklinde bir üçgen istenmektedir. */
```

```
./icicedongulerodevbirbir
ic ice donguler odevler
lutfen boyut belirten bir sayi giriniz
8
       *
      **
     ***
    ****
   *****
  ******
 *******
********
Time elapsed: 000:02:434
Press any key to continue
```

6: Kullanıcıdan A'dan Z'ye kadar bir harf girmesini isteyin ve girilen harfe kadar olan harfler ile bir piramit oluşturun. Piramit kullanıcıdan alınan harf ilke başlayacaktır ve o harfler kendini tekrar edecektir. Ekrana basılması istenen görüntü aşağıdaki gibidir:

Kullanıcının A harfini girdiği baz alınmıştır.

A

A B A

A B C B A

A B C D C B A

A B C D E D C B A

A B C D E F E D C B A

Burada kullanıcıdan alacak olduğumuz harf bize piramidin en üstünde hangi harfin olacağını, hem de dolaylı olarak kaç satırdan oluşacağını belirtmektedir.

Değişken tiplerinin birbirlerine olan dönüşümlerini anlatırken karakter ataması için kullandığımız char değişken tipinin bir tamsayı değerine (integer) ascii tablosu yardımı ile dönüştüğünden bahsetmiştik. Aynı durum integer olarak atanan bir değişken için de geçerlidir ve integer değeri de ascii tablosu yardımı ile bir char karakterine dönüştürebilmekteyiz. Buradan yola çıkarsak bizim harf piramidini oluşturmak için kullanıcıdan okuyacak olduğumuz karakter değeri bir tamsayıya karşılık gelmektedir. Kodlarla ifade etmek gerekirse, ilk örnek kodlarımız aşağıdaki gibi olmalıdır:

```
{
    cout << "ic ice donguler odevler" << endl;

    //harf piramidi oluşturulması

    char c;
    cout << "lutfen bir harf giriniz.." << endl;

    cin>>c;
    cout << (int)c << endl;


    return 0;
}
```

```
./icicedongulerodeviki
ic ice donguler odevler
lutfen bir harf giriniz..
S
83
Time elapsed: 000:08:502
Press any key to continue
```

Kodları yazarken daha anlaşılır olması için, harf piramidini sayısal karakterler olarak ele alacağız. Örneğin yazmış olduğumuz harf piramidini, sayısal değer olarak algılayıp, en son değişken tipleri arasındaki dönüşümden faydalanacağız.

Sayısal olarak hali aşağıdakine benzer olacaktır.	
0	0.satır
0 1 0	1.satır
0 1 2 1 0	2.satır
0 1 2 3 2 1 0	3.satır

Burada dikkat etmemiz gereken ve kodlarımızı yazarken ana düşünceyi meydana getirecek olan ayrıntı satır numaraları kadar, maksimum değer aldığıdır. Örneğin, bizim örneğimizde 4 satır bulunmaktadır ve 4.satırda en büyük değerimiz 4'tür.

Dolayısıyla yazmamız gereken kod: satırda en büyük değeri alana kadar yazan ve daha sonra azalan koddur. Bu kodu oluşturduktan sonra yapmamız gereken tek şey parantez içinde onu char değişken tipine dönüştürmektir.

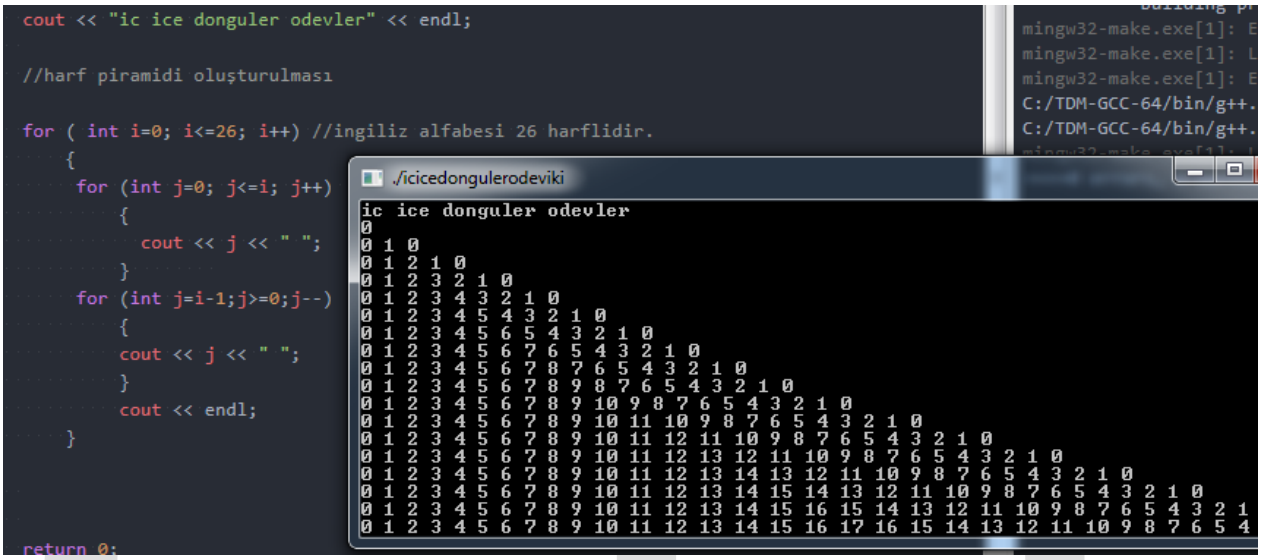
Kodlarımızın ilk aşamasında;

```
cout << "ic ice donguler odevler" << endl;

//harf piramidi oluşturulması

for ( int i=0; i<=26; i++) //ingiliz alfabesi 26 harflidir.
    {
     for (int j=0; j<=i; j++)
          {
            cout << j << " ";
          }
          cout << endl;
    }



return 0;
}
```

```
./icicedongulerodeviki
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10 11
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12 13
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
```

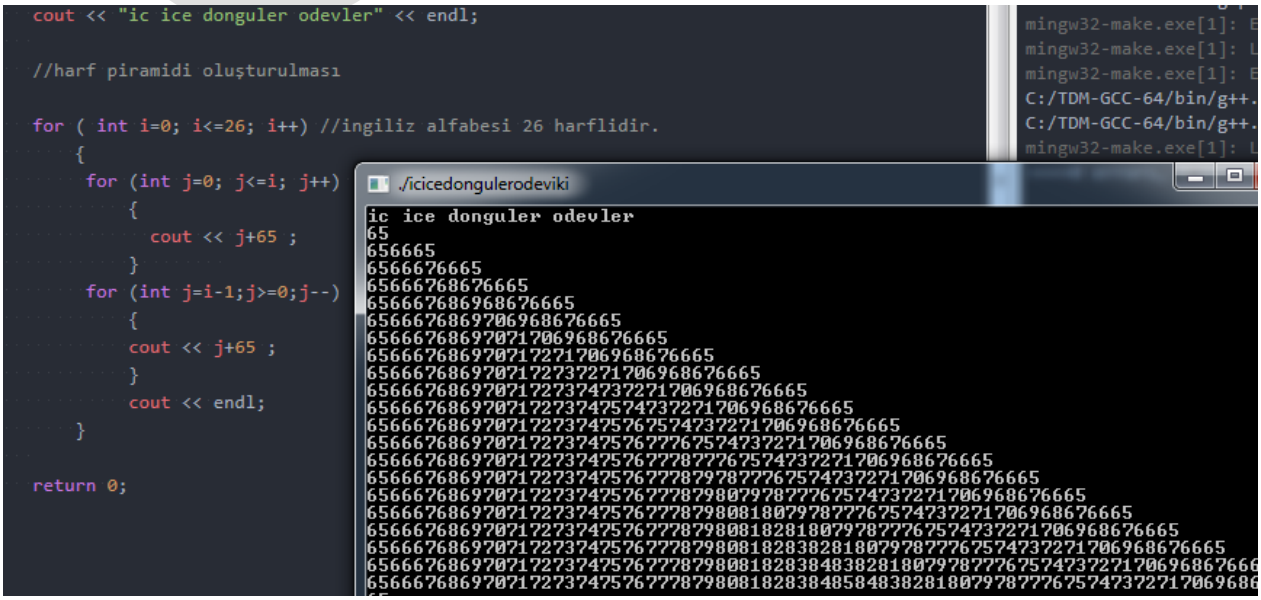
İkinci for döngüsünde j <= i olmasının nedeni, ' i ' değerinin satır numarasını belirlemesidir. Daha sonra ' j ' değeri de satır numarasından yola çıkarak maksimum değeri ekrana basmamızı sağlayacaktır. Fakat yazmış olduğumuz kodun eksik kısım, değerlerin sadece artan olarak ekrana basılmasıdır. Bizim oluşturmamız gereken kodun, piramidin yapısı itibari ile bir de azalan kısmının yer alması gerekmektedir.

Eksik olan kısmı giderebilmemiz için yapmamız gereken şey azalan karakterler için bir tane daha for döngüsü eklemektir. Burada azalan değerleri koşul olarak belirtirken kullanacağımız ifade i-1'dir. Yani o kod bloğu her çalıştığında sayının bir eksiğini sıraya ekleyecektir. Ta ki sayımız 0'a eşit olana kadar. Örnek kodumuzun ikinci aşaması aşağıdaki gibidir:



Ascii tablosunu hatırlayacak olursak, büyük harfler için başlayan kısımda A'nın değeri 65 ile başlıyor ve diğerleri de birer artarak devam ediyordu. Daha a kolay kavrayabilmemiz için yazmış olduğumuz 0 1 2.. sayılarımızı Ascii tablosundaki karşılıklarına çevirebilmek için, ekrana bastırmak için kullandığımız ifademizde j'ye ek olarak +65 eklememiz gerekmektedir. Böylece ekrana basılan değerler, büyük harflerin Ascii tablosundaki karşılığı olacaktır. Ayrıca piramidin daha anlaşılır durması için kodlarımızdaki boşluk " " ifadesini de kaldırmamızda yarar vardır.

Örnek kodlarımızın üçüncü hali aşağıdaki gibidir:



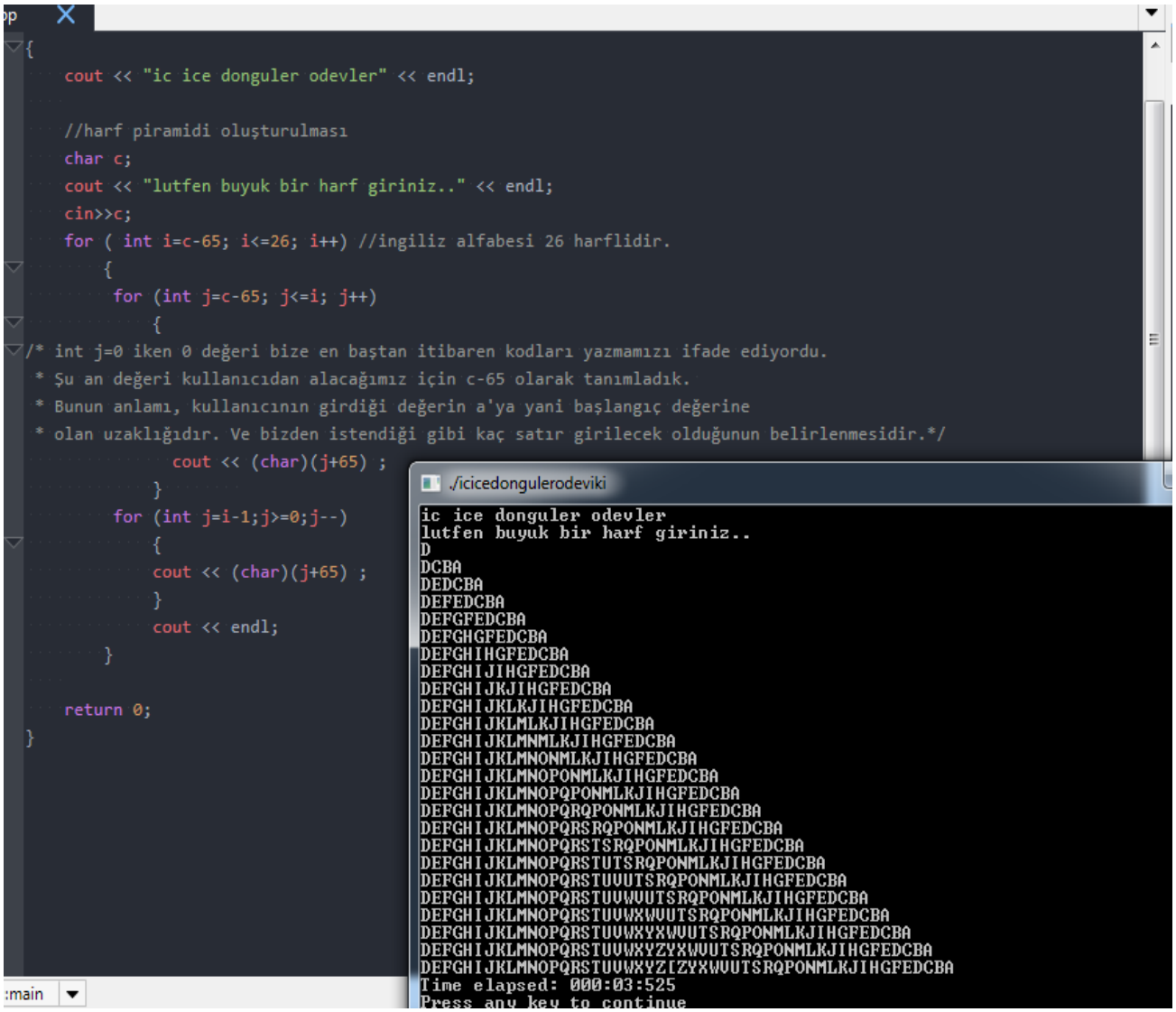
Yapmamız gereken diğer şey, ekrana bastırırken kullanmış olduğumuz j+65 değerini, char olarak belirtmemizdir. Ekranda görecek olduğumuz değerler, sayıların Ascii tablosundaki harf karşılıklarıdır.

```
cout << "ic ice donguler odevler" << endl;

//harf piramidi oluşturulması

for ( int i=0; i<=26; i++) //ingiliz alfabesi 26 harflidir.
    {
     for (int j=0; j<=i; j++)
          {
            cout << (char)(j+65) ;
          }
     for (int j=i-1;j>=0;j--)
          {
          cout << (char)(j+65) ;
          }
          cout << endl;
    }

return 0;
```

```
./icicedongulerodeviki
ic ice donguler odevler
A
ABA
ABCBA
ABCDCBA
ABCDEDCBA
ABCDEFEDCBA
ABCDEFGFEDCBA
ABCDEFGHGFEDCBA
ABCDEFGHIHGFEDCBA
ABCDEFGHIJIHGFEDCBA
ABCDEFGHIJKJIHGFEDCBA
ABCDEFGHIJKLKJIHGFEDCBA
ABCDEFGHIJKLMLKJIHGFEDCBA
ABCDEFGHIJKLMNMLKJIHGFEDCBA
ABCDEFGHIJKLMNONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPQPONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPQRQPONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPQRSRQPONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPQRSTSRQPONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPQRSTUTSRQPONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPQRSTUVUTSRQPONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPQRSTUVWUTSRQPONMLKJIHGFEDCBA
ABCDEFGHIJKLMNOPQRSTUVWXWUTSRQPONMLKJIHGFEDCBA
```

En başta belirttiğimiz üzere, ödevin bizden istediği şey kullanıcının girdiği bir harfe göre piramidin oluşturulmasıydı. Yani bizim atamış olduğumuz 26 (İngiliz alfabesi harf sayısı) değerini, kullanıcıdan alınacak olan bir değer ile değiştirirsek, örnek kodlarımız son halini almış olacaktır.

Örnek kodlarımızın son hali aşağıdaki gibi olacaktır:



```
{
    cout << "ic ice donguler odevler" << endl;

    //harf piramidi oluşturulması
    char c;
    cout << "lutfen buyuk bir harf giriniz.." << endl;
    cin>>c;
    for ( int i=c-65; i<=26; i++) //ingiliz alfabesi 26 harflidir.
        {
         for (int j=c-65; j<=i; j++)
             {
/* int j=0 iken 0 değeri bize en baştan itibaren kodları yazmamızı ifade ediyordu.
 * Şu an değeri kullanıcıdan alacağımız için c-65 olarak tanımladık.
 * Bunun anlamı, kullanıcının girdiği değerin a'ya yani başlangıç değerine
 * olan uzaklığıdır. Ve bizden istendiği gibi kaç satır girilecek olduğunun belirlenmesidir.*/
              cout << (char)(j+65) ;
             }
         for (int j=i-1;j>=0;j--)
             {
             cout << (char)(j+65) ;
             }
             cout << endl;
        }

    return 0;
}
```

```
./icicedongulerodeviki
ic ice donguler odevler
lutfen buyuk bir harf giriniz..
D
DCBA
DEDCBA
DEFEDCBA
DEFGFEDCBA
DEFGHGFEDCBA
DEFGHIHGFEDCBA
DEFGHIJIHGFEDCBA
DEFGHIJKJIHGFEDCBA
DEFGHIJKLKJIHGFEDCBA
DEFGHIJKLMLKJIHGFEDCBA
DEFGHIJKLMNMLKJIHGFEDCBA
DEFGHIJKLMNONMLKJIHGFEDCBA
DEFGHIJKLMNOPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSTSRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSTUTSRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSTUVUTSRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSTUVWVUTSRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSTUVWXWVUTSRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSTUVWXYXWVUTSRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSTUVWXYZYXWVUTSRQPONMLKJIHGFEDCBA
DEFGHIJKLMNOPQRSTUVWXYZ[ZYXWVUTSRQPONMLKJIHGFEDCBA
Time elapsed: 000:03:525
Press any key to continue
```

FONKSİYONLAR (FUNCTİONS)

BASİT FONKSİYON YAPILARI

Fonksiyonlar diğer bir adıyla metodlar, belirli bir işi yapan ve bunu belirtilen şekilde tekrar eden komutlardır. Fonksiyonlar, bir kere tanımladıktan sonra, o fonksiyon her çağırıldığı zaman o iş aynı şekilde tekrarlanmaktadır. Yani kısaca fonksiyonların işlevi, bir işi birden fazla kez çalıştırmak istediğimiz zaman, özellikle büyük projelerde bize kısayol sağlamaktır. Peki bu fonksiyonlar kod dünyasında tam olarak nedir?

Matematikten aşina olduğumuz fonksiyonlar ile programlama da bulunan fonksiyonlar arasında çok bir fark bulunmamaktadır. Matematikte;

F(x)= x + 5

Dediğimiz zaman f fonksiyonu x + 5 parametresine bağlı olarak yazılmış bir fonksiyon olarak görmekteyiz. F(3) kaçtır gibi bir soru karşımıza çıktığı zaman, x yerine 3 koymamız cevaba ulaşabilmemiz için yeterli olmaktadır.

Benzer şekilde görmekte olduğumuz C++ ve diğer C Syntax dillerde de fonksiyonlar bir değişkene bağlıdırlar. İlk olarak fonksiyonun döndürüleceği tipi belirleriz (f (x)'te olan f gibi). Örneğin "int" olarak belirleriz. Daha sonra bu fonksiyonun ismini yazarız (normal bir integer değişkene isim atadığımız gibi) ve en son bu fonksiyona bağlı olan parametresini de (int x gibi) belirleyerek kod bloğumuzu hazır hale diğer bir tabiriyle parametrize edilmiş hale getirmiş oluruz.

Yapısal programlama dilinde 3 temel özelliğimiz bulunmaktadır. Bunlar:

1. Bir kod bloğunun koşula bağlanması (if – else gibi koşul ifadeleri)
2. Bir kod bloğunun tekrar etmesi (for -while döngüleri)
3. Bir kod bloğunun parametrize edilmesi (bu bölümde yani fonksiyonlarda karşımıza çıkacak olan son özelliği de görmüş olacağız.)

Kitabımızın başından itibari ile kodlarımızı yazarken, en başta bir temel olarak kullandığımız "int main()" yapısı bir fonksiyon belirtmektedir.

HATIRLATMA!

C++'da ilk kodumuzu çalıştırırken belirttiğimiz üzere, int main () kod bloğu ana fonksiyondur ve diğer fonksiyonlar ele alınmaksızın o ilk olarak çalıştırılır.

Fonksiyonlarımızı sanal kod ekranımızda yazarken basit şekilde aşağıdaki gibi düzenlemekteyiz:

```
int f (int x)
        {
        cout << x << endl;
        }

int main()
{
    f(5);
    f(16);
    /*f(5) ve f(16) olarak atamış olduğumuz fonksiyonlar
    yukarıdaki kod bloğunun içerisine direk x olarak bastırdığımız için,
     * cout << 5 << endl;
     * cout << 16 << endl;
    dememiz ile aynı sonucu vermektedir. */
}
```

```
./basitfonksiyonlar
5
16
Time elapsed: 000:00:640
Press any key to continue
```

FONKSİYONLARIN DEĞER DÖNDÜRMESİ VE ÇAĞIRILMASI

Fonksiyonlar genel olarak tanımlandıkları zaman bir kod bloğunu çalıştırıyorlar ve bir daha çağırıldıklarında da içine tanımlanan değerler ile ekrana değer döndürebiliyorlardır (kodu tekrarlamak). Farklı yerlerde, gerektiği zaman atanan bu fonksiyonların, içine tanımlanan değerlerle ekrana değer döndürmesi "return" kavramı ile karşılık bulmaktadır.

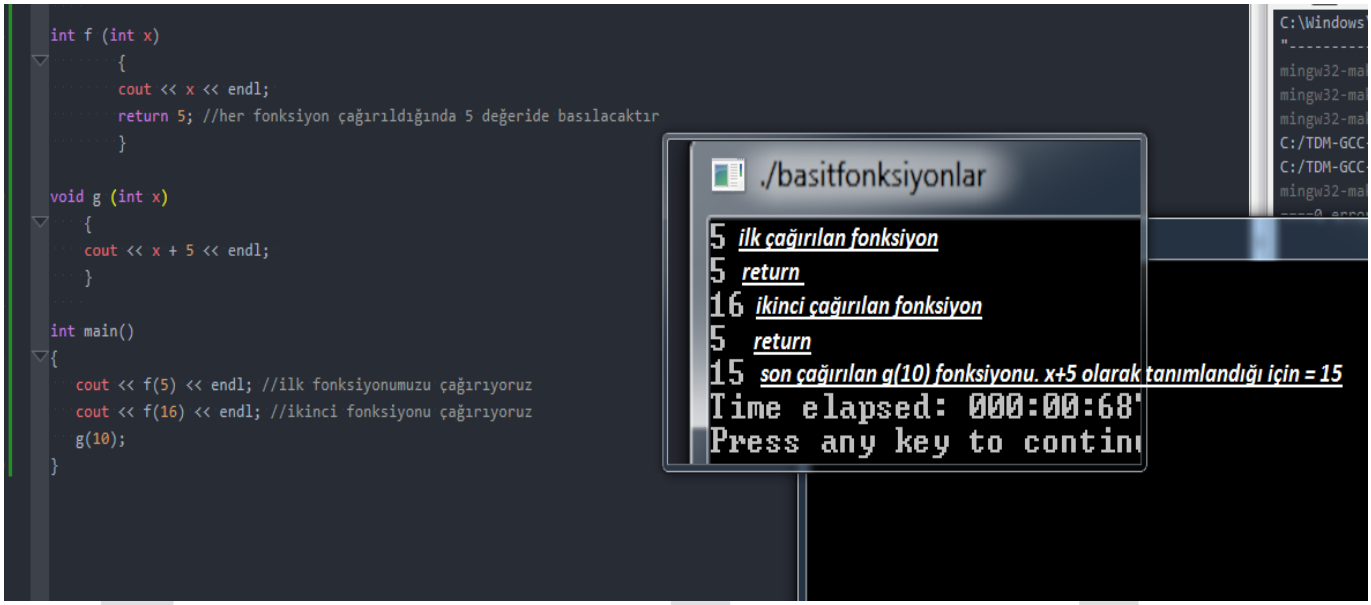
Kodlarımızda return kodunu kullanırken return 5 , return x+3 gibi içerisine değerler atayarak kullanırız. Bunların anlamı, o fonksiyon çağırıldığı zaman ekrana 5 ya da x + 3 değerinin döndürülmesi anlamına gelmektedir.

Bunun yanı sıra diğer bir döndürme tipi olarak "void" de bulunmaktadır. Void hiçbir değer döndürmeyen bir fonksiyondur.

Dikkat edilmesi gereken nokta, eğer bir fonksiyon yazılacak ise;

- İlk olarak o fonksiyonun ekrana döndürülüp, döndürülmeyeceğini belirlemek
- Buna bağlı olarak int vb. ve void ile fonksiyonlarımızı çalıştırmak
- Eğer döndürülecekse int vb. fonksiyonlarda bunu return kodu ile sağlamaktır.

Bir örnek ile pekiştirmek gerekirse, ekrana basılan değerler resimde açıklaması ile verilmiştir;



Örnek: Kullanıcıdan iki sayı alarak kombinasyonunu ekrana bastıran örnek kodu ekrana bastırınız.

Örnek kodumuzu yazarken kullanacak olduğumuz, kombinasyon denklemi aşağıda verilmiştir:

$$c(n,r) = \frac{n!}{r!\,(n-r)!}$$

Öncelikle belirtmemiz gerekir ki, kombinasyon (C) bir fonksiyondur ve 2 tane (n ve r olmak üzere) parametre alır. Kodlarımızı yazmaya başlarken ilk önce işlemi tanımlamayacağız yani faktöriyel işlemlerinin yapıldığı kısmı kodlayacağız.

Faktöriyel olarak adlandırdığımız herhangi bir fonksiyon atıyoruz ve bu fonksiyonun içerisine, faktöriyel işleminin tanımlayabileceğimiz bir koşul bloğu açıyoruz.

Faktöriyel kavramı bir sayının 1'e kadar süregelen çarpımına karşılık gelen işlemdir. ' ! ' işareti ile gösterilir. Örneğin 6 sayısının faktöriyeli:

6! = 6 * 5 * 4 * 3 * 2 * 1 yani 720 'dir.

```
// fonksiyonlar - örnekler 1

int faktoriyel (int x)
    {
    int carpim =1; //faktöriyel açılımı 1'e kadar gittiği için
    // 0 olsaydı tüm çarpım işlemleri sıfırlanırdı
    for (int i=x; i>0;i--)
        {
        carpim*=i; //carpim = carpim * i
        }
    return carpim;

    }

int main()
{
    cout << "fonksiyonlar ornekler" << endl;

    cout << faktoriyel(3) << endl;
    cout << faktoriyel(6) << endl;
```

```
./ornekfonksiyonlarbir
fonksiyonlar ornekler
6
720
Time elapsed: 000:00:718
Press any key to continue
```

Faktöriyel fonksiyonumuzda yazmış olduğumuz koşulların doğru olup olmadığını test etmek için, ekrana 3 ve 6 sayılarının faktöriyellerini bastırmış olduk. Koşullarımız arasında yer alan

“ çarpim*=i; “ kodumuz sayesinde azalan sayıyı sürekli olarak diğer çarpımlar ile çarpıyoruz. Ve aynı zamanda belirtmiş olduğumuz i>0 ile de 1’e geldiği zaman durmasını sağlamış oluyoruz.

Tanımlamış olduğumuz faktöriyel işlemini, kombinasyon denkleminde kullanabilmek için kombinasyon adını verdiğimiz bir fonksiyon kodu yazıyoruz ve işlemi içerisine tanımlıyoruz. Kombinasyon fonksiyonumuz iki tane parametreye sahip olduğu için hem n hem de r olmak üzere iki tane integer (tamsayı) değer atamış bulunmaktayız.

Örnek kodumuz aşağıdaki gibidir:

```
// fonksiyonlar - örnekler 1

int faktoriyel (int x)
    {
    int carpim =1; //faktöriyel açılımı 1'e kadar gittiği için
    // 0 olsaydı tüm çarpım işlemleri sıfırlanırdı
    for (int i=x; i>0;i--)
        {
        carpim*=i; //carpim = carpim * i
        }
    return carpim;

    }

int kombinasyon (int n, int r) //Fonksiyon kümelerine istedğimiz kadar parametre atayabilmekteyiz
    {
       return faktoriyel(n) / (faktoriyel(r) * faktoriyel(n-r));
    }

int main()
{
    cout << "fonksiyonlar ornekler" << endl;

    cout << faktoriyel(3) << endl;
    cout << faktoriyel(6) << endl;

    cout << kombinasyon(5,2) << endl;
    //atamış olduğumuz parametre sayısı kadar değer vermek zorundayız
```

```
./ornekfonksiyonlarbir
fonksiyonlar ornekler
6
720
10
Time elapsed: 000:00:889
Press any key to continue
```

ÖZ YİNELİ FONKSİYONLAR (RECURSİVE FUNCTİONS)

Öz yineli fonksiyonlar kısa tabiri ile bir fonksiyonun kendi kendini yineliyor ya da kendi cinsinden bir fonksiyonu çağırıyor olması demektir.

Matematikte ve kodlarımızda işlemlerimiz için kullanıyor olduğumuz faktöriyel kavramı ile öz yineli fonksiyon kavramını açıklayacağız.

6 sayısının faktöriyelini aldığımız düşünelim.

6! = 6 * 5 * 4 * 3 * 2 * 1 şeklinde ifade ederiz. Aynı ifadeyi

6! = 6 * 5! olarak da ifade etmemiz mümkündür. Çünkü sonuç itibari ile 5 faktöriyelinde açılımı 1' e kadar devam eden çarpım işlemidir.

Ya da bir çarpım işlemi oluşturacak olursak;

2 * 4 işlemi aslında 4 tane ikinin toplanması anlamına gelmektedir.

2 * 4= 2 + 2 + 2 + 2

Aynı işlemi 2 + (2*3) olarak da tanımlayabiliriz.

2 * 4 = 2 + (2 * 3)

2 * 3= 2 + (2 * 2)

2 * 2= 2 + (2 * 1)

2 * 1= 2 + (2 * 0)

Çarpım işleminin genel halini yukarıda oluşturmuş bulunmaktayız.

carp(2 , 4) = 2 + carp (2 , 3)

carp(2 , 3) = 2 + carp(2 , 2)

carp(2 , 2) = 2 + carp(2 , 1)

carp(2 , 1) = 2 + carp(2 , 0)

Kodumuzu **carp** adını verdiğimiz bir fonksiyon olarak atadık.

carp(2 , 4) = 2 + 6

carp(2 , 3) = 2 + 4

carp(2 , 2) = 2 + 2

carp(2 , 1) = 2 + 0

İşlemi alt basamaktan başlayarak tekrar yazıyoruz, carp (2,0) yerine 0 gibi. Böylece çarpım işlemini farklı bir şekilde yazılmış halini elde etmiş oluyoruz.

Bu şekilde kullanmakta olduğumuz geriye dönme işlemlerine call back stack (geri dönüş yığını) adı verilmektedir.

Örnek kodumuz aşağıdaki gibidir:

```
int faktoriyel (int x)

{
    if (x == 1)
        return 1;
    return x * faktoriyel(x-1);
}


int main()
{
    cout << "ozyineli fonksiyonlar" << endl;

    cout << faktoriyel(5) << endl;
}
```

```
./ozyinelifonksiyonlar
ozyineli fonksiyonlar
120
Time elapsed: 000:00:7
Press any key to conti
```

Bu bilgileri yani özyineli fonksiyonları kodlarımızda nasıl çalıştığını anlamak için yukarıdaki örneği inceleyiniz. Örneğin biz faktöriyel 5'in çağırılmasını istiyorsak, bunun anlamı:

Faktöriyel (5) = 5 * faktöriyel (4) şeklinde olacaktır. Fakat daha sonra aynı işlemler faktöriyel (4), faktöriyel (3), faktöriyel (2), faktöriyel (1) için de yapılacaktır. En son 1 sonucuna ulaşıldığı zaman cevaplar yerine yazılarak, en baştaki faktöriyel (5)'in cevabına (120) ekranda ulaşılmış olunacaktır.

Faktöriyel (5) = 5 * faktöriyel (4)

Faktöriyel (4) = 4 * faktöriyel (3)

Faktöriyel (3) = 3 * faktöriyel (2)

Faktöriyel (2) = 2 * faktöriyel (1)

Faktöriyel (1) = 1

Örnek: İlk 20 Mersanne sayılarını veren kodu yazınız.

$$Mersanne\ sayıları\ formülü = 2^n - 1$$

Örneğin: 1, 3, 7, 15, 31, 63 … şeklinde devam ederek gitmektedir.

Formüldeki 2^n ifadesini kodlarımızda yazabilmek için çarpım işleminden faydalanacağız. Sonuç itibari ile örneğin 2'nin 5. Kuvveti, 5 tane 2'nin çarpılması anlamına gelmektedir.

$2^5 = 2 * 2 * 2 * 2 * 2$

Örnek kodumuz açıklaması ile birlikte ekranda verilmiştir:

```
int ust (int t, int u) // t = taban , u = ussu
    {
        int sonuc =1;
        for (int i=1; i<=u; i++)
    {
       sonuc = sonuc * t ; // t^u = t*t*t*t*... (u tane)
    }
    return sonuc;
    }
int main()
{
    cout << "oz yineli fonksiyonlar-mersanne sayilari" << endl;

    for (int i = 1; i<= 20; i++) //ilk 20 sayı istendiği için
    {
    cout << ust(2,i)-1 << endl; // 2^n - 1 formülünü bastırıyoruz
    }
}
```

```
./ozyinelifonksiyonlarmersanne
oz yineli fonksiyonlar-mersanne sayilari
1
3
7
15
31
63
127
255
511
1023
2047
4095
8191
16383
32767
65535
131071
262143
524287
1048575
Time elapsed: 000:00:655
```

Örnek: ilk 20 asal sayıyı yazan örnek kodu yazınız.

Kendisi ve 1 hariç başka bir sayıya bölünmeyen sayılara asal sayı denir ve genel bir formülü bulunmamaktadır.

```
bool asalmi (int x)
    {
    for (int i=2; i<x; i++)
    {
    if (x%i==0) /*eğer 2 (ve artan değerler) değerlerine bölündüğünde
     * kalıyorsa false (yanlış-asal değil)
     * olarak ata diyoruz. */
        {
        return false;
        }
    }
    return true; //diğer durumlarda asal sayı olduğu için true (doğru) döndürüyoruz.
    }

int main()
{
    cout << "ozyineli fonksiyonlar- asal sayilar" << endl;
    int c = 0; //bir parametre atadık ve aşağıda 20 ye kadar devam etmesini sağladık
    for (int i=2; c<20; i++) {
        if (asalmi(i)) {
            cout << i << endl;
            c++;
            }
        }
}
```

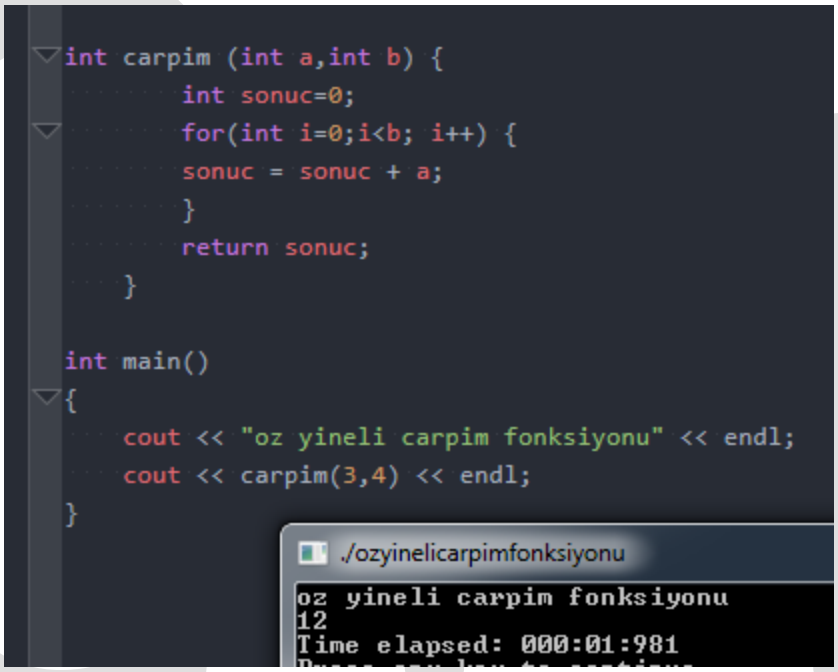
```
./ozyinelifonksiyonlarasalsayilar
ozyineli fonksiyonlar- asal sayilar
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
Time elapsed: 000:00:639
```

Örnek: Sadece toplama işlemini kullanarak çarpma işlemi yapan örnek kodu yazınız.

Örneğin; f(3,4) olan işlemin sonucunu 12 olarak bulmalısınız.

Kodlarımızı yazmaya başlamadan önce hatırlamamız gereken ilk şey, her çarpma işleminde ilk sayının, ikinci sayı kadar kendi ile toplandığıdır. Yani biz 3*4 işlemini 3+3+3+3 şeklinde de yazabiliriz.

Örneğimizi döngüler ile yazacak olsaydık, çarpım değerleri olması için iki tane değer alarak başlardık ve birinci sayıyı, ikinci sayı kadar toplamasını söylerdik. Örnek kodlarımız yandaki gibi olurdu.



Ama örneği öz yineli fonksiyonlar ile yazmak istersek, mantığı aynı olmakla birlikte kodlarımız değişecektir. Öncelikle örneğimizin analizini öz yineli fonksiyonlara uygun olarak yazalım:

Eğer a ve b şeklinde iki tane değer üzerinden gidecek olursak,

f (a,b) = a+a+a+a+… (b kadar) olacaktır.

Şayet biz bir tane a'yı çıkaracak olursak ve geri kalan kısmını yine fonksiyonel olarak yazarsak, öz yineli fonksiyonların mantığına uygun olarak yazmamız mümkün olacaktır:

f (a, b)= a + f (a, b-1) (b-1 olmasının sebebi, 1 tane a'yı başa yazmış olmamızdır.) Böylece kodlarımız çalışırken bu sefer f (a, b-1) sonucunu bulabilmesi için bir ifade daha yazması gerekecektir.

Burada dikkat edilmesi gereken bir nokta, sayılardan herhangi birini sıfır olup olmadığıdır. Çünkü eğer sıfır ise, sonucun ekrana sıfır olarak atanması gerekecektir. Bunun için de bir koşul yazılması gerekmektedir.

Örnek kodlarımız aşağıdaki gibidir:

```
//öz yineli fonksiyonlar ile yazılan
int recursive (int a,int b) {
    if (b==0||a==0)
        return 0;
    return a + recursive(a,b-1);
    }

int main()
{
    cout << "oz yineli carpim fonksiyonu" << endl;
    cout << recursive(3,4) << endl;
    }
```

```
./ozyinelicarpimfonksiyonu
oz yineli carpim fonksiyonu
12
Time elapsed: 000:00:842
```

Fakat fonksiyonda çalıştırılması için ikinci sayıya negatif bir sayı verildiğinde hata almamak için bir ekleme daha yapılması gerekmektedir. Normalde sayıları -1 ile çarpmak bu problemi çözmüş olsa da, örneğimiz itibari ile çarpma işlemi kullanamadığımız için aynı mantık ile 0'dan sayıları çıkartıyoruz. Fakat bu işlemi ikinci sayının negatif olduğu durumlarda yapmamız yeterlidir. Çünkü birinci sayının negatif olduğu durumda bir sorun çıkmayacaktır. Bunun da nedeni, en başta belirttiğimiz üzere, ikinci sayı kadar işlem yapmamızdır. Negatif bir sayı kadar toplama olamayacağı için, onu pozitif bir sayı olarak alıyoruz.

Böylece örneğin f (-5,3) olarak girilen sayılar f(5,-3) olarak algılanmaktadır ve sorun ortadan kalmaktadır. Yada her ikisi de negatif ise f (-5, -3), bunu f(5,3) olarak algılayarak doğru olarak hesaplamaktadır.

Örnek kodlarımız son hali ile aşağıdaki gibi olmalıdır:

```
//öz yineli fonksiyonlar ile yazılan
int recursive (int a,int b) {
    if (b==0||a==0)
        return 0;
    if (b<0)
        return recursive(0-a,0-b);
    return a + recursive(a,b-1);
    }

int main()
{
    cout << "oz yineli carpim fonksiyonu" << endl;
    cout << recursive(3,4) << endl;
    cout << recursive(-5,-3) << endl;
    cout << recursive(5,-3) << endl;
    }
```

```
./ozyinelicarpimfonksiyonu
oz yineli carpim fonksiyonu
12
15
-15
Time elapsed: 000:00:687
```

Örnek: Rastgele sayı (random number) üreten örnek kodu ekrana bastırınız.

Bu fonksiyonu yazabilmemiz için c++ dilinde yer alan srand (time (NULL)) kodunu kullanmamız gereklidir. Bu kod, rand (yani random) fonksiyonunu zamana bağlı olarak üretmemizde bize yardımcı olacaktır. Bunun anlamı, random fonksiyonumuza çalıştığı zaman neyi kullanacağını söylememizdir.

Kodlarımızı yazarken bir diğer kullandığımız ayırt edici koşul ise hangi sayılar arasından rastgele sayı seçildiğidir ve bunu yüzde (%) olarak ifade ederiz. Örneğin %6 dediğimiz zaman 0 ile 6 arasındaki sayılar arasından bir seçim yapılacaktır. Aşağıda kodlarımızı yazarken zar örneğinden yola çıktığımız için 6 rakamının da sayılara dahil olması gerekiyordur. Bundan dolayı +1 olarak belirtmiş bulunmaktayız.

Rastgele sayı belirleyen örnek kodumuz aşağıdaki gibidir:

```
#include <iostream>
#include <ctime>
#include <stdlib.h>
using namespace std;
int main()
{
    cout << "oz yineli fonksiyonlar - rastgele sayi atama" << endl;

    int rg; //rastgele=rg
    srand (time(NULL));
    rg= rand() %6 +1; // zar örneği
    //1 ve 6 arasındaki sayılardan seçim
    cout << rg << endl;

    rg= rand() %2; // yazı - tura örneği için
    //0 ve 1 arasından döndürülecektir.
    cout << rg << endl;

    rg=rand() %100;
    cout << rg << endl;
}
```

```
./ozyinelifonksiyonlarrastgelesayi
oz yineli fonksiyonlar - rastgele sayi atama
3
1
82
Time elapsed: 000:00:670
Press any key to continue
```

Kodlarımızı bir kere daha çalıştırdığımız zaman sayılarımızın rastgele değiştiğini gözlemliyoruz:

```
./ozyinelifonksiyonlarrastgelesayi
oz yineli fonksiyonlar - rastgele sayi atama
1
0
8
Time elapsed: 000:00:687
Press any key to continue
```

DİZİLER (ARRAYS)

DİZİLER VE İNDİSLER

Diziler hafızadan ardışık olarak tutulan değişkenlerdir ve " [ ] " sembolü dizileri gösterir. Örneğin;

int a[3]

a'nın hafızasında 3 tane sayı olduğunu belirtiyor.

int a[3] = {3,7,2}

a[3]	3	7	2
İndis	0	1	2
Erişim	a[0]	a[1]	a[2]

İndis , dizide kaçıncı eleman olduğunu göstermektedir. Dizilerde sayım işlemi sıfırdan başlamaktadır.

Dizileri birer değişken olarak görmemiz mümkündür. Yani kodlarımızda atamış olduğumuz a'nın 0, 1 ve 2. Değerini değişkenlerde olduğu gibi sonradan tekrar başka bir sayı ile güncelleyebiliriz.

```
int main()
{
    cout << "diziler" << endl;

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

    cout << a[1] << endl;
    //1.elemanını yani 2'yi bastırmasını istiyoruz.

    cout << a[0] + a[2] << endl;
    //değişkenlerde olduğu gibi toplayabiliyoruz.
    //0. ve 2. elemanını toplamasını istedik.

    a[2]=8; //değerini güncelledik ve tekrar bastırılmasını istiyoruz.

    cout << a[2] << endl;
}
```

```
./diziler
diziler
7
5
8
Time elapsed: 000:00:749
Press any key to continue
```

Diziler büyük ölçüde döngüler ile birlikte kullanılırlar. Örneğin;

for (int i=0; i<3; i++) dediğimiz zaman i'nin artan değerlerinde a'nın 0, 1 ve 2 değerlerini ekrana bastırmak için kullanılabilir.

```
int a[3]={3,7,2};
for (int i=0;i<3;i++)
    {
    cout << a[i] << endl;
    }
```

```
./diziler
diziler
3
7
2
Time elapsed:
Press any key
```

1. Dizilerin içerisine sadece integer (tamsayı) değeri değil, her türlü değişken tipi atanabilmektedir.
2. Dizilerin en büyük avantajı, kaçıncı elemanı istersek o elemana kolaylıkla ulaşabilmemizdir.

Örnek: Kullanıcıdan 5 sayı alarak bu sayılar arasından en büyük, en küçük ve onların ortalama değerlerini ekrana bastıran örnek kodu yazınız.

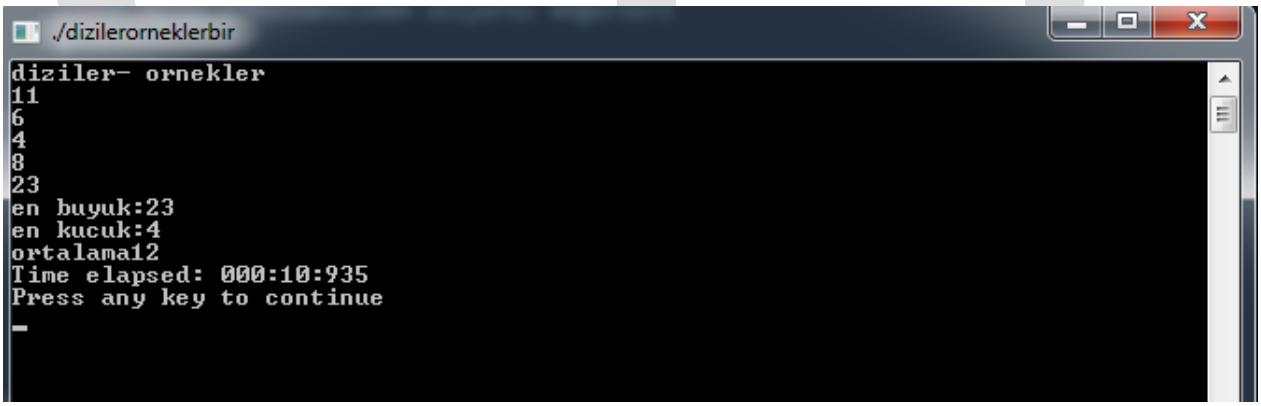
En büyük değeri ekrana bastıran örnek kod:

```
int a[5];
for (int i=0;i<5;i++)
    {
    cin>> a[i]; //kullanıcıdan alıyoruz değerleri
    }

int eb=a[0];
for (int i=1; i<5;i++)
    {
    if (eb< a[i]) //eğer başka bir sayı en büyükse yerine onu ata
        {
        eb=a[i];
        }
    }
cout << "en buyuk:" << eb << endl;
```

En küçük değeri ve ortalamayı ekrana bastıran örnek kod:

```
    int ek=a[0];
    for (int i=1; i<5;i++)
        {
        if (ek > a[i]) //eğer başka bir sayı en küçükse yerine onu ata
            {
            ek=a[i];
            }
        }
    cout << "en kucuk:" << ek << endl;

    int ortalama=a[0];
    for (int i=0; i<5;i++)
        {
        ortalama= ortalama+a[i]; //sayıların toplanması
        }
    cout << "ortalama" << ortalama/5 << endl;
}
```

Ekrana rastgele sayılar girdiğimizde oluşan sonuçlar:



Örnek: Asal mersanne sayılarını veren örnek kodu yazınız.

Mersanne sayılarının formülü aşağıdaki gibidir:

$$Mersanne\ sayıları\ formülü = 2^n - 1$$

Kodlarımızı yazarken öncelikle, önceki örneklerimizde yaptığımız mersanne sayılarını ve asal sayıları bulan kodlarımızdan yardım alacağız. Diğer örnekten farklı olarak burada yapacağımız işlem mersanne sayıları arasından asal olanları belirleyerek ekrana ilk 20 tanesini bastırmak olacaktır.

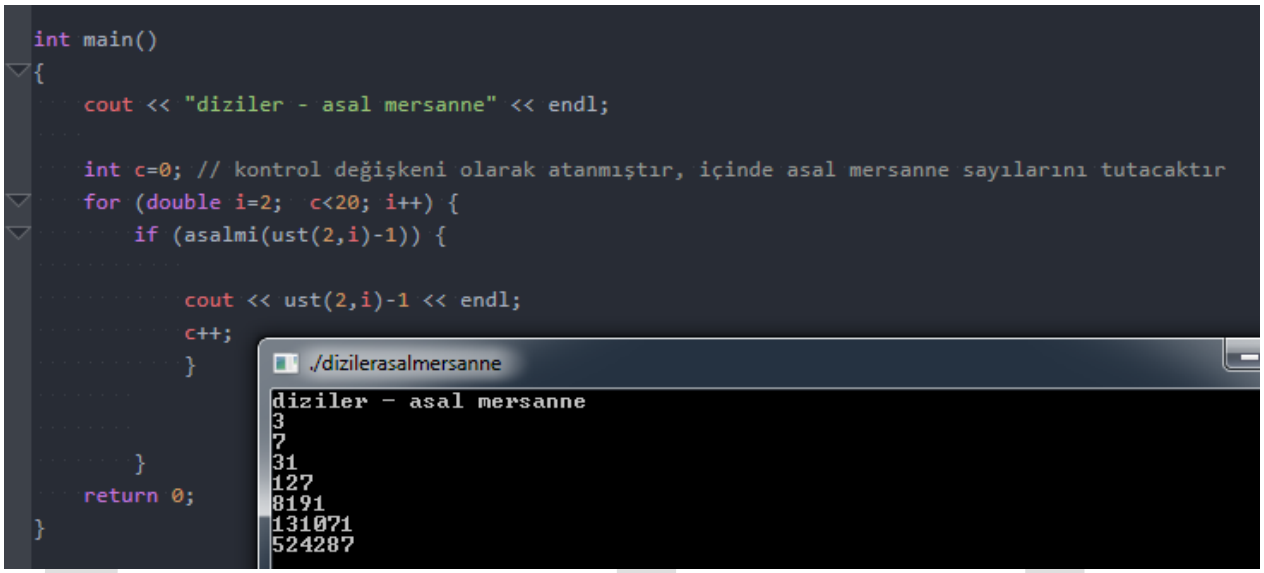
İlk örnek kodlarımızda görüldüğü üzere, mersanne sayılarını ve asal sayıları bulan kodlarımızı kullandık:

```
//mersanne sayılarının örnek kodları

int ust (int t, int u) // t = taban , u = ussu
{
int sonuc =1;
for (int i=1; i<=u; i++) {
        sonuc = sonuc * t ; // t^u = t*t*t*t*... (u tane)
}
return sonuc;
}

//asal sayılarının örnek kodları

bool asalmi (int x)
    {
    for (int i=2; i<x; i++)
    {
    if (x%i==0) /*eğer 2 (ve artan değerler) değerlerine bölündüğünde
     * kalıyorsa false (yanlış-asal değil)
     * olarak ata diyoruz. */
        {
        return false;
        }
    }
    return true; //diğer durumlarda asal sayı olduğu için true (doğru) döndürüyoruz.
    }
```

İlk 20 asal mersanne sayılarını bastırabilmek için int main() ana kodumuzda koşul belirttik. Burada kodlarımızı yazarken bir c değeri atadık ve bu değer bizim kontrol değerimiz oldu. Bulduğumuz asal ve mersanne sayılarını bu değişkenin içerisinde tuttuk.

Ayrıca “asalmı” olarak kodladığımız, asal sayıları kontrol eden bool değişkenin içerisine mersanne sayılarını oluşturan kodumuzu yerleştirdik. Ve c<20 diyerek, ilk 20 tanesini ekranda basılmasını sağladık.

Örnek kodlarımız aşağıdaki gibidir:



Fakat ilk 20 asal mersanne sayıyı yazılmasını beklerken, ekranda oluşan değerler 20’den daha azdır. Bunun sebebi, asal olan mersanne sayılarının ilk 7-8 tanesinden sonrasının hızlı bir şekilde çok basamaklı ve c++’ın şimdilik hesaplayamayacağı boyutta olmasından kaynaklanmaktadır. İlerleyen sevilerde bunların bir çeşit özel kütüphaneler ile mümkün olacağını öğreneceksiniz.

Örnek: Bir dizideki en büyük 3 sayının toplamını ekrana bastıran örnek kodu yazınız.

Daha önceki örneklerimizde dizideki sayılar arasından en büyük sayının nasıl seçilebileceğini göstermiştik. İlk sayı yine aynı yol ile seçilecek olup, diğer iki sayının nasıl seçileceğini anlatacağız.

Bunu yapabilmek için kodlarımızda 3 tane sayıyı hafızada tutmamız gerekmektedir ve if koşul blokları kullanarak dizideki diğer sayılar ile, hafızada tutulanları karşılaştıracağız. En son soruda istenen itibari ile toplamlarını ekrana bastıracağız.

Örnek kodlarımız açıklaması ile birlikte kod ekranında verilmiştir:

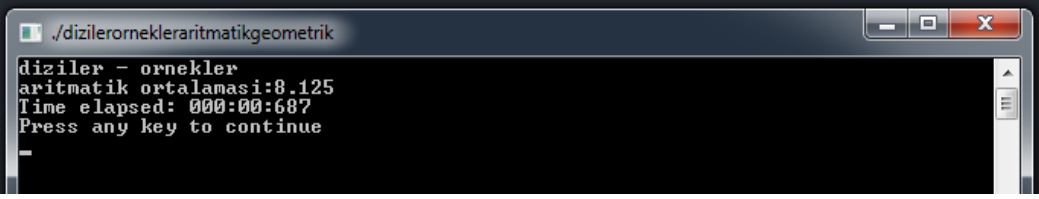
```
    cout << "diziler - ornekler" << endl;

    int a[7] ={4,5,8,18,41,3,6};

    //ilk olarak en büyük bütün sayıları, dizinin ilk elemanı olarak atadık
    //daha sonra karşılaştırıken hafızada tutulan sayılar ile yer değiştirecektir.

    int eb1= a[0];
    int eb2= a[0];
    int eb3= a[0];

    for (int i=1; i<6;i++)
        {
        if (eb1< a[i]) //eğer dizideki diğer sayılar 1.hafızada tutalandan büyükse
        {
        eb3=eb2;
        eb2=eb1;
        eb1= a[i]; //hafızada tutalan 1.sayı ile yer değiştir.
        }
        else if (eb2< a[i]) //eğer dizideki diğer sayılar 2.hafızada tutalandan büyükse
        {
        eb3=eb2;
        eb2=a[i]; //hafızada tutalan 2.sayı ile yer değiştir.
        }
        else if (eb3< a[i])  //eğer dizideki diğer sayılar 3.hafızada tutalandan büyükse
        {
        eb3=a[i]; //hafızada tutalan 2.sayı ile yer değiştir.
        }
        }
        cout << "en buyuk sayilar:" << eb1 << ", " << eb2 << ", " << eb3 << endl;

        cout << "en buyuk sayilarin toplami:" << eb1+eb2+eb3 << endl;
        }
```

Sonuçların ekrana basılması:

Sıralı örnekler:

Dizideki sayıların aritmetik ortalamasını veren örnek kod:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "diziler - ornekler" << endl;

    int a[8] ={3,5,21,12,6,4,13,1};
    int toplam=0;
    for (int i=0; i<8;i++)
        {
            toplam= toplam + a[i]; //toplam değerinin hesaplanması için
        }
    cout << "aritmatik ortalamasi:" << (float) toplam/8 << endl; //bölme işlemi olduğu için kalanlı çıkablir
    //bu yüzden float değişken tipini kullanıyoruz.
}
```



Dizideki sayıların geometrik ortalamasını bastıran örnek kod:

Geometrik ortalama: Sayıların birbirleriyle çarpımlarının, n birim sayısı olmak üzere, n'inci dereceden köküne denir.

```
int a[8] ={3,5,21,12,6,4,13,1};
int carpim=1;

for (int i=0;i<8;i++) {
    carpim *= a[i]; //değerlerin çarpılması
    }
    cout << "geometrik ortalamasi:" << pow(carpim,(float) 1/8) << endl;
    /*1/8 değeri 8.dereceden kök almamızı belirtir. 1/x dedikten sonra kaçıncı
    dereceden kökünü almak istersek x yerine onu yazmamız gerekmektedir. */
```

```
./dizilerornekleraritmatikgeometrik
diziler - ornekler
geometrik ortalamasi:5.74058
Time elapsed: 000:00:858
Press any key to continue
```

Dizideki tek sayıların ortalamasını bulan örnek kodu yazınız:

```
//dizideki tek sayıların ortalaması

 int a[8] ={3,5,21,12,6,4,13,1};
 int toplam=0;
 int teksayilar=0; //kaç tane tek sayi oladuğunu bilmediğimiz için bir paramtre değeri atadık
  for (int i=0;i<8;i++) {
      if(a[i]%2==1) {
          toplam += a[i];
          teksayilar++;
          }
   }
cout << "dizideki tek sayilarin ortalamasi:" << (float)toplam/teksayilar << endl;
```

```
./dizilerornekleraritmatikgeometrik
diziler - ornekler
dizideki tek sayilarin ortalamasi:8.6
Time elapsed: 000:00:795
Press any key to continue
```

Dizideki sayılardan en büyüğü ve en küçüğünün ortalamasını bulan örnek kod aşağıdaki gibidir:

```
int a[8] ={3,5,21,12,6,2,13,1};

int eb= a[0];
int ek = a[0];

for (int i=0; i<8; i++) {

    if(a[i]%2==0) { //eğer sayı çift ise (ilk kural)

        if(eb< a[i]) { //en büyük çift sayiyi belirlemek için

            eb= a[i];
            }
        if (ek> a[i]) {

            ek= a[i];
            }
    }

    }
    cout << "eb ve ek sayilarin ortalmasi:" << (float)(eb+ek)/2 << endl;
}
```

```
./dizilerornekleraritmatikgeometrik
diziler - ornekler
eb ve ek sayilarin ortalmasi:7
Time elapsed: 000:00:795
```

Örnek: Kullanıcıdan kaç sayı gireceğini ve daha sonra da o sayıları alın. Ekrana girdiği sayıların toplamını bastıran örnek kodu yazınız.

Bu örneğimizi hem dizi kullanmadan hem de kullanarak birden fazla farklı yol ile çözmemiz mümkündür. Öncelikle sadece döngüleri kullanarak çözmeyi deneyelim. İkinci çözüm yolumuzda ise dizileri kullanarak çözeceğiz.

İlk çözüm yolumuz:

```
int main()
{
    cout << "diziler - ornekler" << endl;

    cout << "lutfen kac sayi gireceginizi belirtiniz:" << endl;
    int c;
    cin >> c;

    int toplam=0;
    cout << "lutfen sayilari giriniz:" << endl;

    for (int i=0; i < c; i++) {

        int g; // girilecek olan sayilar
        cin >> g;
        toplam += g; //girilen sayilari bir degerin içerisinde topluyoruz
        }
    cout << "sayilarin toplami:" << toplam << endl;
```

```
./dizilerorneklerdizitoplama
diziler - ornekler
lutfen kac sayi gireceginizi belirtiniz:
4
lutfen sayilari giriniz:
1
2
3
4
sayilarin toplami:10
Time elapsed: 000:02:098
```

Daha önceki kodlarımızda yaptığımız üzere, kullanıcının kaç sayı gireceğini ve daha sonra bu girilen sayıları bir değişkenin içerisinde toplanmasını istedik. Son olarak da toplam değerini ekrana bastırdık.

İkinci çözüm yolumuz (dizileri kullanarak):

İlk olarak kullanıcıdan kaç sayı gireceğini alıyoruz ve bu sayıyı bir dizinin içerisinde tutuyoruz. Daha sonra ilk döngümüzde kullanıcıdan gireceği değerleri alıyoruz ve bu sayıları hafızda tutması için yine bir dizinin içerisinde tutuyoruz.

İkinci yazmış olduğumuz döngüde ise bu değerleri topluyoruz ve son olarak ekrana toplam değerinin basılması istiyoruz. Dizilerin daha anlaşılır bir şekilde kullanılabilmesi ve sorunsuz çalışması için kodlarımızı ayır ayrı iki farklı döngüde yazdık fakat istersek sayıları tutma ve toplama işlemini aynı döngünün içerisinde birleştirmekte mümkündür.

Örnek kodlarımız aşağıdaki gibidir:

```
//ikinci çözüm yolumuz - dizileri kullanarak

 cout << "lutfen kac sayi gireceginizi belirtiniz:" << endl;

int n;
cin >> n;
int a[n]; //kullancının kaç sayı gireceğini alıyoruz

cout << "lutfen sayilari giriniz:" << endl;

for (int i=0;i<n;i++) {

    cin >> a[i]; //girilecek olan sayıları alıyoruz
}

int toplama=0;
for (int i=0;i<n;i++) {

    toplama += a[i]; //bu sayıları topluyoruz

    }
 cout << "sayilarin toplami:" << toplama << endl; //ekrana bastırıyoruz


return 0;
}
```

```
./dizilerorneklerdizitoplama
diziler - ornekler
lutfen kac sayi gireceginizi belirtiniz:
3
lutfen sayilari giriniz:
1
5
78
sayilarin toplami:84
Time elapsed: 000:07:675
```

Örnek: Kullanıcıdan 5 sayı alarak bu sayılardan 4 tanesi ile yazılabilecek en büyük ve en küçük değerleri ekrana bastıran örnek kodu yazınız.

Bu örneği çözerken kullanacak olduğumuz algoritma diğerlerinden daha farklı olacaktır. Dilerseniz daha önce kullandığımız yöntemleri kullanarak da çözüme ulaşmak mümkündür.

Örneğin kullanıcıdan aşağıdaki sayı değerlerini aldık:

1 5 8 4 9

Öncelikle bu sayıların toplamını elde edeceğiz yani bu örnekten yola çıkarsak 27.

Daha sonra bu sayılar arasından en büyük sayıyı (9) ve en küçük sayıyı (1) kodlarımızda belirleyeceğiz. Böylelikle eğer biz toplam değerinden en büyük sayıyı çıkartırsak, bu sayılardan 4 tanesi ile oluşturulabilecek en küçük sayıyı bulabilmiş olacağız. Aynı algoritma bu sayılardan 4 tanesi ile oluşturulabilecek en büyük değer içinde geçerlidir. Eğer toplam değerinden en küçük sayıyı çıkartacak olursak, bu sayılardan 4 tanesi ile oluşturulabilecek en büyük değere de ulaşmış olmaktayız.

Örnek kodlarımız adım açıklamaları ile birlikte aşağıda verilmiştir:

```
cout << "diziler - ornekler" << endl;

cout << "lutfen 5 sayi giriniz!!" << endl;
int a[5];

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

cin >> a[i]; //kulanıcıdan 5 değer alıyoruz
}

int eb= a[0];
int ek= a[0];
/* ilk olarak en büyük ve en küçük değerleri
 * dizinin ilk elemanlarını atıyoruz
 * daha sonra diğer elemanlar ile karşılaştırarak
 * bu değerlerden daha büyükleri varsa
 * güncelleştirmesini sağlayacağız.
 * */
int toplam=a[0];

for (int i=1; i<5; i++) {
    toplam += a[i]; //değerleri topluyoruz

    if (eb<a[i]) {
    eb= a[i]; } //eb değerini güncellemek için
    if (ek>a[i]){
        ek= a[i]; }
        //ek değerini güncellemek için

    }
cout << "en buyuk deger:" << toplam - ek << endl;
cout << "en kucuk deger:" << toplam - eb << endl;
```

```
./dizilerorneklertoplamaoyunu
diziler - ornekler
lutfen 5 sayi giriniz!!
1
2
3
14
4
en buyuk deger:23
en kucuk deger:10
Time elapsed: 000:06:146
Press any key to continue
```

ÇOK BOYUTLU DİZİLER

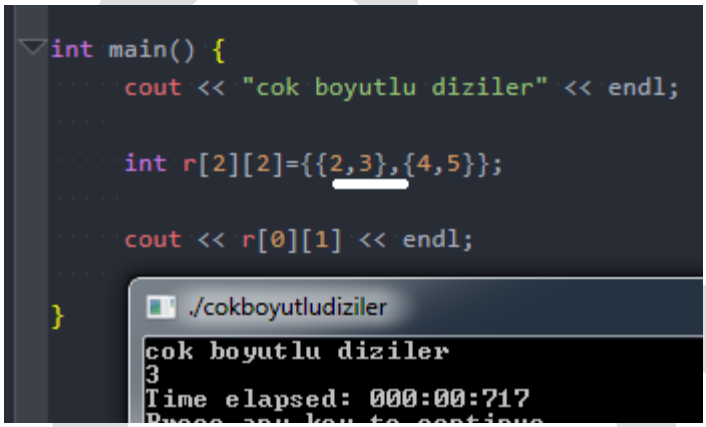
Bir dizi bir veya birden çok boyutlu olabilir. Örneğin eğer aşağıdaki gibi tanımlanıyorsa iki boyutludur:

int x[2][3];

İki boyutlu dizilerin ilk boyutuna satır ikinci boyutuna sütun denmektedir.

İnt x[2][2] = {{2,3},{4,5}};

anlamı, 2 satır ve 2 sütundan oluşmuş olduğudur. Eşit olduğu değerlerde {2,3} kısmı dizinin 0. Eleman değerleri olup, {4,5} kısmındaki değerler ise birinci eleman değerleridir.



Değerleri ekrana bastırmak için r[0][1] kullanmak demek, dizinin 0.elemanındaki 1.elemanını ekranda bastırmak demektedir. Böylece 3 elemanının olduğunu görmüş bulunmaktayız.

Aynı diğer dizilerde olduğu gibi çoklu dizilerde de daha sonradan dizinin elemanlarını güncellemek, o değeri kullanıcıdan almak mümkündür. Örneğin bir değeri daha sonradan aşağıdaki şekilde güncelleyebiliriz:

r[0][1] = 33;

Bu sefer farklı bir döngü kullanarak dizideki tüm elemanları bastıran kodu yazalım:

Bunun için iki tane iç içe iki tane for döngüsü elde etmemiz gereklidir. İlk yazılan döngü de hangi satırın ilk basılacağı belirlenecek olup (bunu koşul olarak int i değerine bağlayacağız ve i++ diyerek sırası ile satırları basmasını sağlayacağız), daha sonra içerideki döngüde ise sırayla o satırın elemanları basılması söylenecektir (yine bunu da i değerine bağlayacak olup, sırayla basılması sağlanacaktır).

Örnek kodumuz aşağıda verilmiştir:

```
int main() {
    cout << "cok boyutlu diziler" << endl;

    int a[2][2]={{1,2},{3,4}};

    for (int i=0; i<2; i++) {
   // {1,2} ve {3,4} arasındaki seçim için sırası ile artan değere bağladık

        for (int j=0; j<2;j++) {
   // 1 ve 2 mi yoksa 3 ve 4 mü önce basılacak diyerek onu da sırası ile j değişkeine bağladık
            cout << a[i][j] << " " ;
            }
            cout << endl; //her satır bittiginde alt satıra geçmesi için
}
```

```
./cokboyutludiziler
cok boyutlu diziler
1 2
3 4
Time elapsed: 000:00:686
Press any key to continue
```

Örnek: İki diziyi karşılaştırarak, birinci dizinin, ikinci dizinin bir parçası olup olmadığını bulan örnek kodu yazınız.

Kodumuzu yazarken öncelikle yapmamız gereken şey iki dizi atamaktır. Atamış olduğumuz dizilerin kontrolünü sağlamak için;

```
int a[3]={1,2,3};

int b[7]={8,9,1,2,3,7,6};
```

Teker teker dizide öncelikle 1 var mı diye aranacak olup, şayet varsa 1'den sonraki elemanı 2 mi? Evet ise cevap sonrası 3 mü? Şeklinde doğrulayarak olacaktır.

Bunları yaparken aynı zamanda a ve b değişkenlerinin boyutlarını da yazmamız gerekmektedir. Çünkü kullanıyor olduğumuz diziler sadece birer gösterici olup, bize bir boyut ifade etmemektedirler. Fakat kodlarımızın C++ dilinde doğru çalışabilmesi için, ayrıca bir değişken atayarak boyutlarını da belirtiyor ve problemi ortadan kaldırıyoruz.

Örnek kodumuz aşağıda verilmiştir. Alt dizinin 2.elemandan başladığı doğrudur çünkü dizilerde eleman sayılarının 0'dan başladığını daha önce belirtmiştik.

```
{
    cout << "diziler - ornekler" << endl;

    int a[3]= {1,2,3};
    int b[10]= {6,7,1,2,3,8,9,4,11,30};

    int aboyut=3;
    int bboyut=10;

    for(int i=0; i<bboyut; i++) {
        bool esit=true;
        for (int j=0; j<aboyut; j++) {
            if (a[j] != b[i+j]) {
/* b[i+j] dememizin sebebi sayıları kontrol ederken 0. değise 1'e, 1. değilse 2'ye
ve bu şekilde devamını da kontrol etmesini istememizdir. */

                esit=false; //eğer eşit değilse yanlış olarak döndür ve
                break; // eğer eşit değilse döngüyü kır

                }
            }
    if(esit) {
        cout << "alt dizisidir ve " << i << " 'den baslar" << endl;
        }
        }
        }
```

```
./dizilerornekleraltdizi
diziler - ornekler
alt dizisidir ve 2 'den baslar
Time elapsed: 000:00:609
Press any key to continue
```

Örnek: İki kişinin zar oyunu oynadığını düşününüz. Kullanıcıdan kaçar zar atılacağını alıp, iki kişi için bu zarları atan, sonra da kazananı bulan örnek kodu yazınız.

Yapılacak olan zar atma işleminde kullanılacak kodlar, daha önce de random (rastgele) sayı atma işleminde kullanılmış olanlardır. (rand() %6+1)

Örneğin;

Kullanıcıdan alınan sayı= 5

1. Kullanıcı için= 1 6 3 2 4
2. Kullanıcı için= 2 4 3 2 1

Sonuç = 1 kullanıcı kazanır.

gibi bir örnek kodun ekranda olması istenmektedir.

Sonucun belirlenmesi şu şekildedir:

Örneğin yukarıdaki örnekten yola çıkarsak, 1 kullanıcı ilk olarak 1, 2.kullanıcı ise 2 değerini zarda atmıştır. Dolayısı ile 2.kullanıcıdan yana olarak oyun 1-0 'dır. Daha sonra 1.kullanıcı 6 değerini, 2.kullanıcı ise 4 değerini zarda atmıştır ve oyun berabere (1 – 1) olmuştur. Bu şekilde en son oyundan sonra sonuç belirlenerek kazanana ulaşılacaktır.

Örnek kodlarımızı yazarken dizileri kullanacağımız nokta, zar atılan değerlerin yani skorların yan yana ve alt alta yazıldığı ve daha sonra bunların birbirleri ile karşılaştırıldığı kısımdır. Biz 2.kullanıcıya (satıra) geçtiğimiz zaman, 1.kullanıcının skorunun unutulmaması için, hafıza da 1.kullanıcının skorunu dizilerin tutmasını sağlamış olacağız.

Örnek kodlarımız adım adım aşağıda verilmiştir:

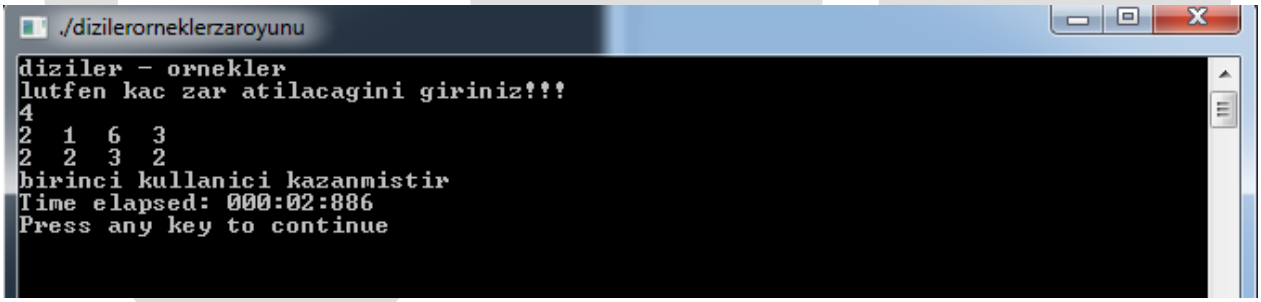
1.Adım:

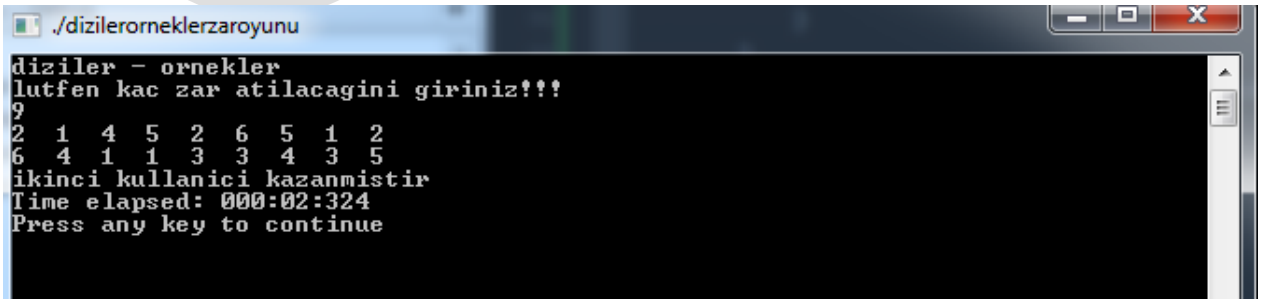
```
{
    cout << "diziler - ornekler" << endl;
    cout << "lutfen kac zar atilacagini giriniz!!!" << endl;
    int n;
    cin >> n; //kullanıcıdan kaç zar atılacağını girmesini istiyoruz.

    srand(time(NULL));
    int skor; // sonucu belirlemek için bir değişken atıyoruz
    int a[n];
    int b[n]; //zar değerlerinin tutlması için
    for (int i=0; i<n; i++) {

        int z1 = rand()%6+1; //birinci kullanıcının zarlari
        int z2 = rand()%6+1; //ikinci kullanıcının zarlari

        a[i]= z1; //her atanan değeri hafızada tutabilmek için
        b[i]= z2;

    //skorun artıtılması-azaltılması için
        if (z1>z2){
            skor++;
            }
        else if (z2>z1) {
            skor--;
            }
        }
```

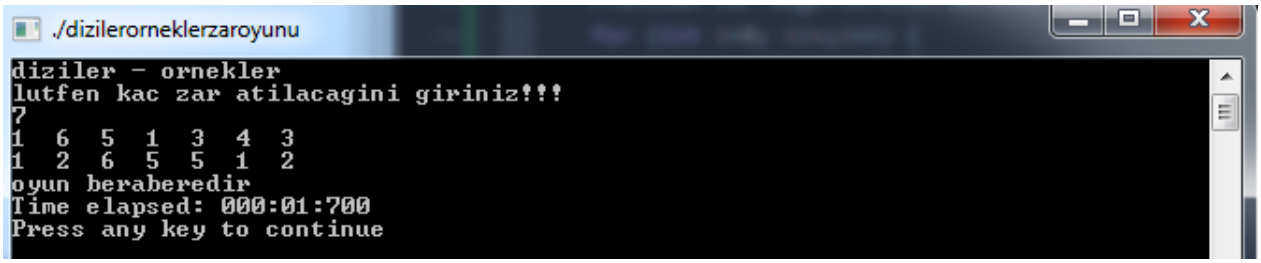
2.Adım:

```
    }
    //atılan zar değerlerinin ekranda bastırılması için
    for (int i=0; i<n;i++) {
        cout << a[i] << "  " ;
        }
        cout << endl;
    for (int i=0; i<n;i++) {
        cout << b[i] << "  " ;
        }
        cout << endl;
        //skorun belirlenebilmesi için
    if (skor>0) {
        cout << "birinci kullanici kazanmistir" << endl;
        }
        else if (skor<0){
        cout << "ikinci kullanici kazanmistir" << endl;
            }
    else {
        cout << "oyun beraberedir" << endl;
        }
}
```

Sonuçların ekranda farklı sayılar ile çalıştırılmış hali:







Örnek: İki boyutlu bir matrisin transpozunu alan örnek kodlarını yazınız.

Transpoz: Matematikte (özellikle sayısal cebirde) bir matrisin satır ve sütunlarını yer değiştirmesi anlamına gelmektedir.

Örneğin;

Matris: 5 4 3 Bu matrisin transpozunu alacak olursak => 5 7 4

4 0 4 4 0 10

7 10 3 3 4 3

Öncelikle anlaşılabilmesi için örnek kodlarımızı basit bir şekilde kodlayacağız. Daha sonra yazacak olduğumuz ikinci örnek kodlarımız, daha kullanışlı olacaktır.

1. Kodlarımızı yazarken öncelikle bir iki boyutlu (3'er değer alan) bir matris tanımlıyoruz ve orijinal matrisin (transpozu alınmamış) değerlerini burada belirtiyoruz.
2. Daha sonra iç içe 3 tane for döngüsü kullanacağız. İlk iç içe for döngüsü, orijinal matrisi görebilmemiz için yazılacaktır yani ekrana yansıtma kodlarını barındıracaktır.
3. İkinci iç içe for döngüsü için öncelikle başka bir dizi daha atıyoruz (her biri 3'er değer alan). Ve bu sefer kodlarımızın içeriğinde orijinal matris ile ikinci oluşturduğumuz diziyi eşitliyoruz. Böylece ikinci iç içe for döngüsünde matrisin transpozu alınmış oluyor.
4. Son kullanacak olduğumuz iç içe for döngüsünü ise, transpozu alınmış olan iki boyutlu matrisin ekrana bastırılmasını sağlamaktadır.

Örnek kodlarımız aşağıdaki gibidir:

```
int a[3][3]= {5,4,3,4,0,4,7,10,3};

//orjinal matrisin ekrana basılması için
for(int i=0;i<3;i++) {
    for (int j=0;j<3;j++) {

        cout << " " << a[i][j];
        }
    cout << endl;
    }

//transpozun yapıldığı iç içe for döngüsü

 int b[3][3];
 for(int i=0;i<3;i++) {
    for (int j=0;j<3;j++) {

       b[j][i]=a[j][i];
        }
    }
cout << endl;//iki matris arasında boşluk olmasını sağlamak için
cout << endl;
cout << endl;

//transpozu alınmış matrisin basılmasını sağlıyoruz

  for(int i=0;i<3;i++) {
    for (int j=0;j<3;j++) {

        cout << " " << b[i][j];
        }
    cout << endl;
    }
```

```
./matristranspozu
matris transpozunu alma
 5 4 3
 4 0 4
 7 10 3



 5 4 3
 4 0 4
 7 10 3
Time elapsed: 000:00:280
```

Transpoz işlemi yukarıda yazmış olduğumuz örnek kodlar ile de sağlanabilmekle birlikte, bu kodları kullanmamız büyük projelerde büyük alan kaplayacaktır. Bunun sebebi iki dizi birden atadığımız için, örneğin 3000 tane gibi değere sahip olan bir projede, iki dizi için de Ram'de bu alanı ayırmak kullanışlı olmayacaktır.

Bu yüzden ikinci çözüm yolumuzda, transpoz aşamasında ikinci bir dizi daha atamak yerine, geçici bir integer değişkende değerlerimizi tutacağız. Yani sırası ile;

- int g = a [j][i] (orijinal matrisi koyuyoruz.)
- a [j][i]=a[i][j] (satırları sütunlara, sütunları satırlar ile yerlerini değiştiriyoruz. Ve bu sırada boşta kalan değerler integer g değişkeni ile tutuluyor.)
- a [i][j]=g (son olarak g değerine transpozu olan matrisi atıyoruz.)

Örneğin elinizde bulunan iki bardaktaki sıvıları yer değiştirmek için nasıl ki 3. Bir bardağa ihtiyaç duyarsınız, bu örnekte kullanmış olduğumuz in g değişkenini de, 3.bir bardak olarak düşünmeniz mümkündür.

Örnek kodlarımız aşağıdaki gibidir:

```
//transpozun yapıldığı iç içe for döngüsü

 // int b[3][3];
 for(int i=0;i<3;i++) {
    for (int j=i+1;j<3;j++) {
        int g= a[j][i];
        a[j][i]=a[i][j];
        a[i][j]=g;
        }
    }
cout << endl;//iki matris arasında boşluk olmasını sağlamak için
cout << endl;
cout << endl;

//transpozu alınmış matrisin basılmasını sağlıyoruz

  for(int i=0;i<3;i++) {
    for (int j=0;j<3;j++) {

        cout << " " << a[i][j];
        }
    cout << endl;
    }
```

```
./matristranspozu
matris transpozunu alma
 5 4 3
 4 0 4
 7 10 3


 5 4 7
 4 0 10
 3 4 3
Time elapsed: 000:00:858
```

Örnek: İki boyutlu iki matrisin toplamını veren örnek kodları yazınız (matrislerin boyutları eşit olmalıdır).

Örneğin;

Ekranda basılması istenen matrisi verecek olan işlem aşağıdaki gibidir:

```
0 1 2  +  6 5 4  =  0+6 1+5 2+4  =  6  6  6
9 8 7     3 4 5     9+3 8+4 7+5     12 12 12
```

Diğer örneklerde olduğu gibi örnek kodlarımızı yazarken, iki tane iki boyutlu dizi atadık ve matrislerde verilen değerleri sırası ile belirttik. Daha sonra 3. Bir dizi daha tanımlayarak, bu dizide de satır ve sütunların toplanmasını ve ikinci bir iç içe döngü ile ekrana basılmasını istedik.

Örnek kodlarımız aşağıdaki gibidir:

```
cout << "matrislerin toplami" << endl;

int a[2][3]= {0,1,2,9,8,7};
int b[2][3]= {6,5,4,3,4,5};
int c[2][3]; // toplamını yapması için

for (int i=0;i<2;i++) {
    for (int j=0;j<3;j++) {

        c[i][j]= a[i][j] + b[i][j];
    }
    }

 for (int i=0;i<2;i++) {
    for (int j=0;j<3;j++) {

        cout << "  " << c[i][j];
    }
    cout << endl;
    }

return 0;
```

```
./matrislerintoplami
matrislerin toplami
  6   6   6
  12  12  12
Time elapsed: 000:00:687
Press any key to continue
```

Yandaki örnekte kullandığımız gibi, iki tane iç içe döngüde aşamaları ayırmak yerine, yazdırma işlemini de ayni iç içe döngünün içerisinde yapabilirsiniz. Ama bu yöntem büyük projelerde kodlar çalışmadığı zaman aşamaların net gözükmesi istendiği için tercih edilmemektedir. Ayrıca Yine başka bir dizi atamak yerine a[i][j] += b[i][j] şeklinde birleştirilmiş olarak kullanmamız mümkündür.

GÖSTERİCİLER-İŞARETÇİLER (POİNTERS)

GÖSTERİCİ KAVRAMINA GİRİŞ

Kodlarımızı yazmaya ilk başladığımız zaman atamış olduğumuz her değişkeni bir kutu olarak düşünebileceğimizi ve bu kutunun içerisine bir değer koyabileceğimizi söylemiştik. Bu kutunun ve dolayısıyla yazdığımız her kodun Ram üzerinde belirli bir yeri vardır. Buna bağlı olarak da Ram üzerinde o değere erişebilmemiz için bir adrese sahiptir. Göstericiler (pointers), Ram'deki bir yeri gösteren değişken olarak tanımlanabilmektedir. Eğer değişken belirli bir değere eşit ise (int b= 1;), Ram'de belirli bir adresi vardır fakat eğer sadece değişken atanmış ve içi boş ise (int b;) o zaman Ram'de belirli bir adresi yoktur, sadece bir yer kaplar.

ff506	a=10
ff808	p

Kullanılan değerler tamamen örnek olması açısından yazılmıştır, gerçek değildir.

Kodlarımızı yazarken bir değişkenin başına " * " işareti koyarsak (int *p), bunun anlamı o değişkenin bir pointer yani işaretçi olduğu anlamına gelmektedir. Eğer bir değişkenin başına " & " işaretini yerleştirirsek bu işaretin anlamı, değişkenin Ram' deki adresini göstermesidir. Yani &a dediğimiz zaman bizim ekranımızda ff506 gözükecektir. Ve yazmış olduğunuz bir işaretçiye de herhangi bir adres belirtmemiz mümkündür [int *p; ve p=&a (önce bir işaretçi atıyoruz ve bu işaretçinin a'yı göstermesini istiyoruz)] . Örneğin yukarıdaki örnekte p'nin önüne * işareti koyulmuş ve *p Ram'de ff506 adresine sahip olan yeri işaret ediyorsa (p=&a) , ekranda göreceğimiz değer 10 olacaktır. &p şeklinde bir şey görmek istersek, o zamanda p'nin Ram'deki adresini (ff808) ekranda göreceğiz.

```
{
    cout << "gostericiler - pointers" << endl;

    int a= 10;
    int *p; //p'nin gösterdiği yerdeki değer
    p= &a; //p, a'nı adresini gösteriyor

    cout << "a: " << a << endl; //a'nın değeri
    cout << "p: " << p << endl; //p'nin adresi (fakat a'yı işaret edecektir)
    cout << "*p: " << *p << endl; //p'nin işaret ettiği yerdeki değer
    cout << "&a: " << &a << endl; //a'nın adresi
    cout << "&p: " << &p << endl; //p'nin kendi adresi
}
```

```
./gostericiler
gostericiler - pointers
a: 10
p: 0x22fe3c
*p: 10
&a: 0x22fe3c
&p: 0x22fe30
Time elapsed: 000:00:686
Press any key to continue
```

DİZİLERİN GÖSTERİCİLER İLE BİRLİKTE KULLANILMASI

Her dizi bir işaretçi ve her işaretçi de bir dizidir. Yani bizim işaretçileri, değişkenlere işaret ettiğimiz gibi, aynı zamanda dizilere de işaret etmemiz mümkündür. Bunu örnekler ile anlatmak gerekirse, aşağıda yazmış olduğumuz diziyi, bir işaretçi ile gösterelim.

Örneğin;

int a[3]= {1,4,8};

int *p; (bir işaretçi atıyoruz)

p = a; (a adresini işaretçiye gösteriyoruz)

Aynı şeyi farklı bir gösterim yöntemi ile de yapmak mümkün: p=&a[0];

Böylece aslında yapmış olduğumuz işlemde, biz a değişkenin yapabileceği şeylerin (örneğin sayıları güncellemek ya da erişmek), aslında p'nin de yapabilmesini sağlamış oluyoruz.

Kodlarımızı ekranda da göstermek gerekirse aşağıdaki gibi olacaktır:

```
{
    cout << "dizilerin gostericiler ile kullanilmasi" << endl;

    int z[3]={1,4,8};

    int *p;
    p=&z[0]; //ya da p=z;

    cout << z[2] << endl; // z dizisinin 2.elemanını(3'ü) bastırıyoruz

    cout << p[2] << endl;
    /*aslında p bir diziye sahip değilken
     * z'yi işaret ettiği için
     * z'nin 2.elemanını basıyor
     * */

    p[1]=6; //p dolayısıyla z dizini işaret ettiği için, onun 1.elemanını 8 olarak güncelleyecektir

    cout << z[1] << endl;
```

```
./dizileringostericilerilekullanilmasi
dizilerin gostericiler ile kullanilmasi
8
8
6
Time elapsed: 000:00:687
Press any key to continue
```

FONKSİYONLARIN GÖSTERİCİLER İLE KULLANIMI-REFERANS/DEĞER İLE ÇAĞIRMA

C++ çok sık olarak göstericilerle kullanılan özelliklerden bir diğeri ise referans ile çağırmadır. Bu zamana kadar fonksiyonlar ile yaptığımız çağırma işlemi değer ile çağırmaydı (call by value). Referans ile çağırma işlemi göstericileri kullanarak tanıyacağımız çağırma işlemi olacaktır.

Aşağıda yazmış olduğumuz örnek kodda, değer ile çağırma ve referans ile çağırma arasındaki farkı ekrana basılan değerlerde göstermeye çalışacağız:

```
// değer ile çağırma
int g (int x) {
   x=10;
    }

//referans ile cagirma
int k (int *x) {
    *x=30;
    }

int main()
{
    cout << "gostericilerin fonksiyonlar ile kullanilmasi" << endl;

    int a=20;
    int *p;
    p=&a;

    //deger ile cagirma
    g(a);
    cout << a << endl;
    //referans ile cagirma
    k(p);
    cout << a << endl;
}
```

```
./gostericilerreferansilecagirilma
gostericilerin fonksiyonlar ile kullanilmasi
20
30
Time elapsed: 000:00:764
Press any key to continue
```

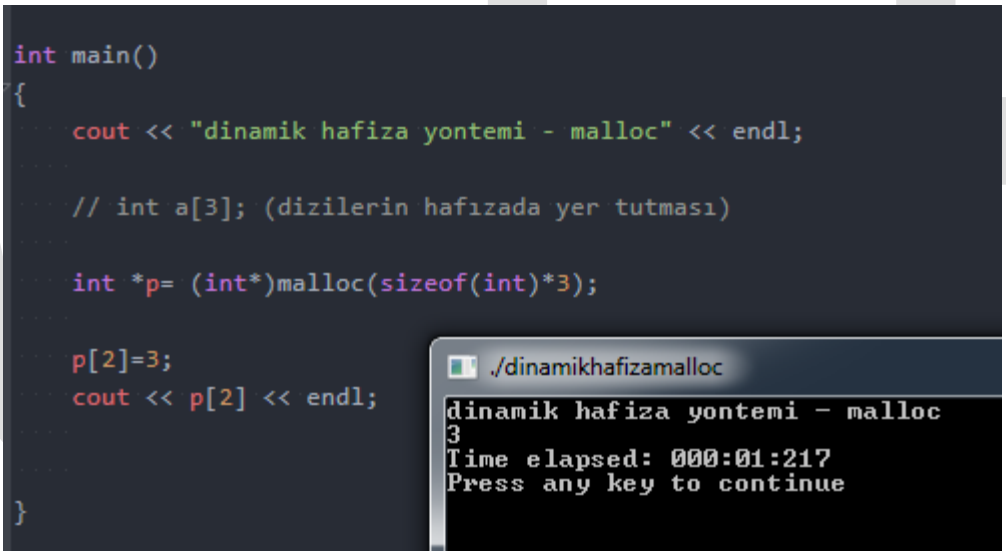
Değer ile çağırma işlemi yaptığımız zaman (yani daha önceki bölümlerde, fonksiyonlarda yapmış olduğumuz çağırma işlemi), int a=20; olarak yapmış olduğumuz güncelleştirme işlemi kullanılıyor ve ekranda güncellenen değeri görmüş oluyoruz. Ama aynı işlemi referans ile (göstericiler yardımı ile) yaptığımız zaman, bize fonksiyonun içerisindeki değeri getiriyor ve bizim güncellemiş olduğumuz değeri kullanmıyor.

Referans ile çağırmanın bize sağladığı yarar şudur:

Normal şartlarda bir fonksiyon, bize sadece 1 değeri getirmektedir. Fakat biz fonksiyonları göstericiler ile kullandığımız zaman birden fazla değeri döndürmemiz mümkündür.

DİNAMİK HAFIZA YÖNTEMİ-MALLOC

Bildiğimiz üzere diziler, atamış olduğumuz değişkenler için bize hafızada yer tutuyorlardı. Aynı hafıza da yer tutma işlemini dinamik hafıza yöntemi olarak geçen Malloc (memory allocation) yöntemi ile de yapmamız mümkündür. Bunun için bazı kodlardan yardım almaktayız:



Yazmış olduğumuz int *p= (int*)malloc(sizeof(int)*3); kodunun anlamı, ilk olarak integer (tamsayı) değeri tutacak kadar bir yer ayırması gerektiğini ve daha sonra da bu değerin boyutunu "*3" diyerek belirtiyoruz.

Böylece dizilerde olduğu gibi, işaretçiler ile de hafızada yer tutmamız mümkün hale geliyor.

Bir önceki bölümde görmüş olduğumuz referans ile gösterme yöntemini de yine dinamik hafıza yöntemi ile elde etmemiz mümkündür. Örnek kodumuz aşağıdaki gibi olmalıdır:

```
int f (int *p)
        *p=10;


int main()
{
    cout << "dinamik hafiza yontemi - malloc" << endl;

    int *p= (int*)malloc(sizeof(int)*3);
    p[2]=3;
    cout << p[2] << endl;

    int *q= (int*)malloc(sizeof(int)*3);
    *q= 50;
    f(q);

    cout << *q << endl;
```

```
./dinamikhafizamalloc
dinamik hafiza yontemi - malloc
3
10
Time elapsed: 000:00:609
Press any key to continue
```

Örnek: Göstericileri kullanarak kullanıcıdan 5 sayı okuyunuz ve sonra okunan sıra ile ekrana geri bastırınız.

Yukarıda belirtmiş olduğumuz örneği, konuları daha iyi kavrayabilmek için başlangıç seviyesinde, dizilerle, dinamik hafıza ile ve göstericiler aritmetiği kullanarak 4 farklı çözüm yolu ile derleyeceğiz.

1. Çözüm yolu: Başlangıç seviyesinde;

```
int main()
{
    cout << "gosterici airtmetigi - kullanicidan sayi okuma" << endl;

    int a,b,c,d,e;
    cout << "lutfen 5 sayi giriniz!" << endl;
    cin >> a >> b >> c >> d >> e ;
    cout << a <<" " << b <<" "<< c <<" "<< d << " "<< e << endl;
}
```

```
./gostericiaritmetigi
gosterici airtmetigi - kullanicidan sayi okuma
lutfen 5 sayi giriniz!
1
2
3
4
6
1 2 3 4 6
Time elapsed: 000:05:960
Press any key to continue
```

2. Çözüm yolu: Diziler ile;

```
//2.çözüm yolu (diziler ile)

int a[5];
cout << " lutfen 5 sayi giriniz!" << endl;
for (int i=0; i<5;i++) {
    cin >> a[i];
    }
for (int i=0; i<5;i++) {
   cout << a[i] <<" " ;
    }
```

```
./gostericiaritmetigi
gosterici airtmetigi - kullanicidan sayi okuma
 lutfen 5 sayi giriniz!
1
2
3
4
7
1 2 3 4 7 Time elapsed: 000:12:184
Press any key to continue
```

3. Çözüm yolu: Dinamik hafıza ile;

```
//3.çözüm yolu (dinamik hafıza ile)

int *q;
q=(int*) malloc(sizeof (int)*5);

 for (int i=0; i<5;i++) {
     cin >> q[i];
     }
 for (int i=0; i<5;i++) {
    cout << q[i] <<" " ;
     }
}
```

```
./gostericiaritmetigi
gosterici airtmetigi - kullanicidan sayi okuma
1
2
3
4
8
1 2 3 4 8 Time elapsed: 000:07:410
Press any key to continue
```

4. Çözüm yolu: Gösterici aritmetiği ile;

Gösterici aritmetiği olarak belirttiğimiz işlemde, yine bir gösterici tanımlıyoruz fakat bu sefer göstericimizi bir integer (tamsayı) değeri ile topluyoruz. Bu kodlarımızı yazarken yardımını aldığımız döngüleri kullandığımız için, koşullarımıza yazdığımız int i (sırayla değeri artan) değeri ile toplamış bulunmaktayız. Ve daha sonra bu değerin önüne “*” işareti koyarak, bu toplamın yeni yerini kullanıcının girmesini istiyoruz.

Örnek kodumuz aşağıda verilmiştir:

```
    */
    // 4.çözüm yolu (gösterici aritmetiği ile)
    cout << " lutfen 5 sayi giriniz!" << endl;
    int *q;
    q=(int*) malloc(sizeof (int)*5);

    for (int i=0; i<5;i++) {
        cin >> *(q+i);
        }
    for (int i=0; i<5;i++) {
       cout << *(q+i) <<" " ;
        }

}
```

```
./gostericiaritmetigi
gosterici airtmetigi - kullanicidan sayi okuma
 lutfen 5 sayi giriniz!
1
2
3
4
9
1 2 3 4 9 Time elapsed: 000:09:376
Press any key to continue
```

FONKSİYONLARIN DİZİLERİ PARAMETRE OLARAK ALMASI

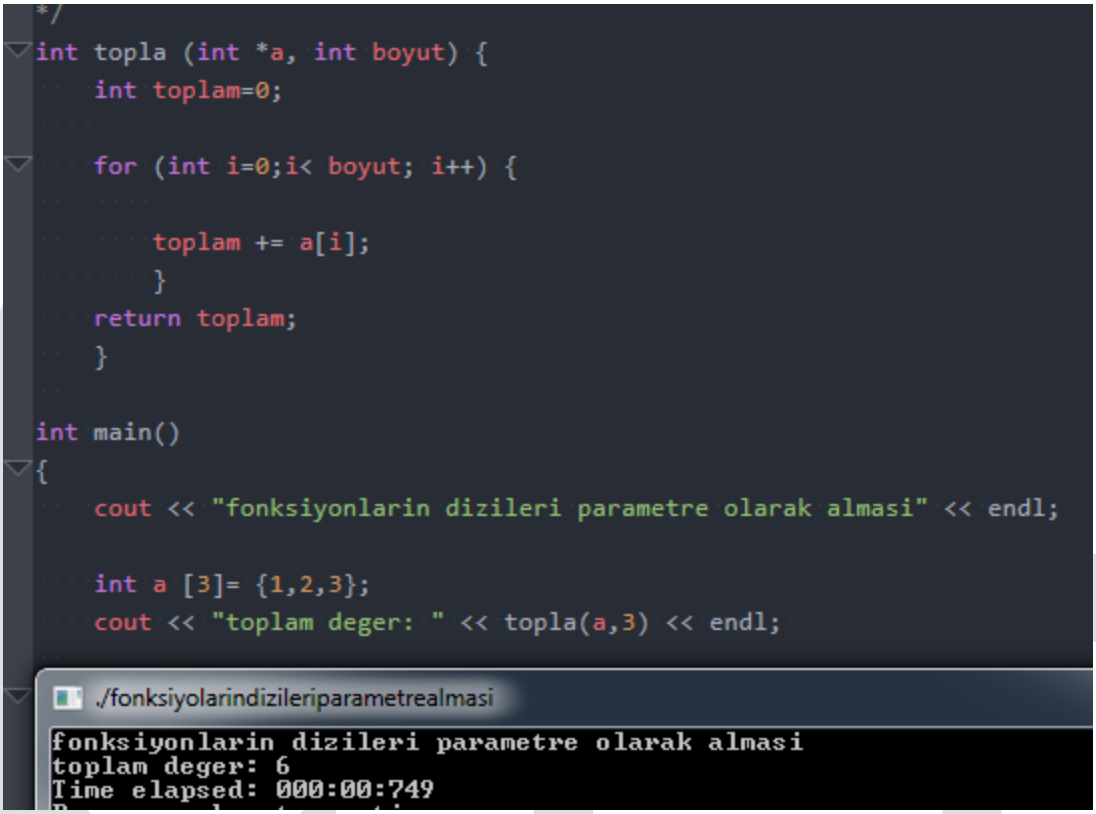
Göstericileri kullanarak bir diziye ve fonksiyonlara ulaşabilmemiz ve onların elemanları ile işlem yapabilmemiz mümkündü. Aynı şekilde hiç göstericileri kullanmadan da fonksiyonların dizilere ulaşması ve onları parametre olarak kullanması mümkündür.

```
int f (int *p) {
    p[2]=10;
    }

int main()
{
    cout << "fonksiyonlarin dizileri parametre olarak almasi" << endl;

    int *p;
    int a [3]= {1,2,3};
    p=a;
    cout << a[2] << endl;

    f(p);
    cout << a[2] << endl;

    f(a);
    cout << a[2] << endl;
```

```
./fonksiyolarindizileriparametrealmasi
fonksiyonlarin dizileri parametre olarak almasi
3
10
10
Time elapsed: 000:00:718
Press any key to continue
```

Ekranda görüldüğü üzere bir gösterici yardımı ile dizilere ve fonksiyona erişebiliyoruz. Fakat son örneğe dikkat edilmesi gerekirse, aynı şekilde fonksiyonlarda dizilere (gösterici yardımı olmaksızın) erişebilmekteler.

Bundan yola çıkarak bir toplama örneği yapmak gerekirse, örnek kodumuz aşağıdaki gibidir:



Örnek: Dizideki en büyük ve en küçük sayıları bularak bunlar arasındaki farkı ekrana bastıran örnek kodu yazınız.

Daha önceki örneklerimizde de kullanmış olduğumuz karşılaştırma yöntemi ile sayılar arasındaki en büyük ve en küçük değeri ilk olarak bulacağız. eb (en büyük) ve ek (en küçük) değerleri için öncelikle a[0] yani a dizisinin 0.elemanını atıyoruz. Daha sonra if koşul bloklarını kullanarak bu sayıları güncelliyoruz. Diğer örneklerimizden tek farkı bu kodları fonksiyon blokları içerisinde döndürmemiz ve karşılaştırılan sayıları bir diziden almamızdır.

Örnek kodlarımız aşağıdaki gibidir:

```
    for (int i=0; i< boyut; i++) {
        //daha sonra eb ve ek değerlerini güncelleyebilmek için karşılaştırıyoruz

        if (eb<a[i]) {
            eb=a[i];
            }
        if (ek>a[i]) {
            ek=a[i];
            }
        }
    return eb-ek;
    }

int main() {
cout << "gostericiler - odev" << endl;

int a[9] = {1,2,8,45,6,-2,7,52,1};
cout << "eb-ek farki:" << f(a,9) << endl;
```

```
./gostericilerodev
gostericiler - odev
eb-ek farki:54
Time elapsed: 000:02:106
Press any key to continue
```

DİZGİLER (STRİNGS)

DİZGİ KAVRAMI VE KARAKTER DİZİLERİ

Dizgi kavramı aslına bakıldığı zaman, kolay anlaşılabilmesi açısından bir karakter dizisi olarak düşünülebilir. Normal dizilerde olduğu gibi bir dizi ataması yapılır fakat bu sefer karakterler ile hareket ettiğimiz için, char değişken tipi kullanılır. Ve hafızada tutulacak olan değişken sayısı, her harf başına hesaplanmaktadır.

char a[8]= “yazılım” ; şeklinde yazılabileceği gibi, harf harfte atama yapılabilmektedir. Yani;

char b[3];

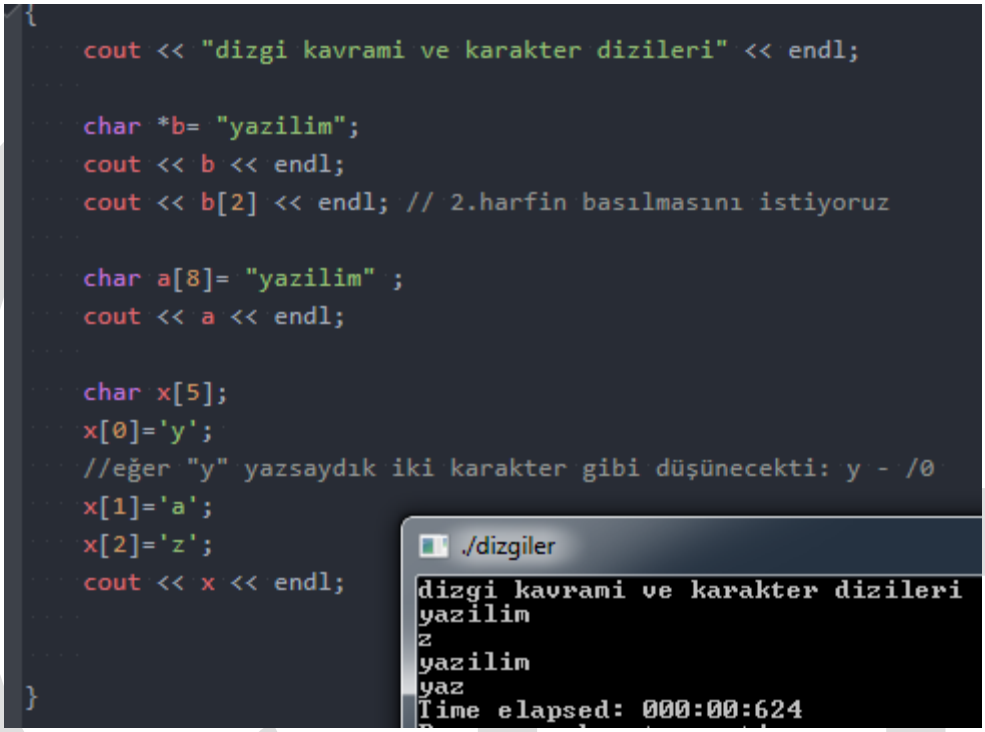
b[0]=’y’;

b[1]=’a’;

b[2]=’z’; şeklinde de ifade etmemiz mümkündür.

Yukarıdaki “yazılım” örneğinde olduğu gibi, harflerden fazla olarak karakter yer tuttuğu zaman burada devreye “end of string” kodu dediğimiz “/0” devreye girmektedir. Boş olan ya da değer atanmamış yerlere bu kod gelmektedir. Eğer kodlarımızı yazarken çift tırnak kullanırsak end of string her durumda sona kendiliğinden eklenecektir fakat tek tırnak koyduğumuz zaman böyle bir durum söz konusu değildir.

Anlattığımız kodları özetleyecek bir örnek yapmak gerekirse, örnek kodlarımız aşağıdaki gibi ekrana yansıyacaktır:



Eğer sanal ortamınızda dizgiler için yazdığınız kodlar çalışmıyorsa aşağıda verilen kütüphaneyi eklemeniz gerekmektedir:

```
#include <string.h>
```

DİZGİLERİN KARŞILAŞTIRILMASI-SIĞ KOPYALAMA-BUS ERROR 10

Yukarıdaki örnekte de olduğu üzere, aynı kelimeyi birden fazla char değişkeni ile dizgi olarak yazmamız mümkündür. Fakat bunları kodlarımızda karşılaştırdığımız zaman, kodlarda birebir aynı gözükmelerine rağmen, eşit değildir olarak karşılaşacağızdır:

```
int main()
{
    cout << "Dizgilerin karsilastirilmasi-Sig Kopyalama-Bus Error 10" << endl;

    char *b= "yazilim";
    cout << b << endl;

    char a[8]= "yazilim" ;
    cout << a << endl;

    if (b==a) {
        cout << "esittirler" << endl;
        }
    else {
        cout << " esit degillerdir" << endl;
        }
```

```
./dizgilerinkarsilastirilmasi
Dizgilerin karsilastirilmasi-Sig Kopyalama-Bus Error 10
yazilim
yazilim
 esit degillerdir
Time elapsed: 000:00:609
Press any key to continue
```

Bunun sebebi aslında bir işaretçi olmalarıdır. Her ne kadar birebir aynı görünseler bile, ikisinin de Ram'de sahip olduğu adresler birbirinden farklıdır (yani toplamda iki farklı adrese sahibiz). Bu yüzden de kodlarımızda "eşit değillerdir" yazısı işle karşılaşmaktayız.

Bunu giderebilmek için ilk yol olarak aklımıza c=s; şeklinde bir kodu kodlarımıza eklemek gelebilir (işaretçiler-pointers bölümünden) fakat bu bir çözüm yolu değildir. Çünkü ikisine birbirini eşitlememiz aslında bir işaretçiyi, diğerinin adresine yönlendirmiş olduğumuz anlamına gelmektedir (yani elimizde sadece 1 adres kalmış olur). Örneklemek gerekirse kodlarımız ekrana basıldığı zaman aşağıdaki gibi gözükecektir:

```
    cout << "Dizgilerin karsilastirilmasi-Sig Kopyalama-Bus Error 10" << endl;

    char *b= "yazilim";
    cout << b << endl;

    char a[8]= "yazilim" ;
    cout << a << endl;
    b=a;
    b[2]='x';
    cout << a << endl;
    if (b==a) {
        cout << "esittirler" << endl;
        }
    else {
        cout << " esit degillerdir" << endl;
        }
```

```
./dizgilerinkarsilastirilmasi
Dizgilerin karsilastirilmasi-Sig Kopyalama-Bus Error 10
yazilim
yazilim
yaxilim
esittirler
Time elapsed: 000:00:718
```

Görüldüğü gibi ekranda “eşittirler” olarak gözükmüştür fakat sadece elimizde bir adres olduğu için böyle bir sonuç karşımıza çıkmıştır. Bunu daha iyi anlayabilmek için de “yazılım” kelimesinden bir harfi değiştirdik ve diğer kodun ekrana basılmasını istediğimizde, kelimeyi bize değişmiş olarak (yaxılım) vermesinden anlamamız mümkündür (elimizde tek bir kelime yani adres olduğu için).

1. İşaretçilerin birbirlerine eşitlenmesine sığ kopyalama ya da İngilizcesi ile shallow copy adı verilmektedir.
2. Günlük hayatta örneğin bir yönetici (admin) tarafından kullanıcıların şifreleri karşılaştırılmak istendiğinde, birden fazla aynı şifreden olması mümkündür. Fakat aynı şekilde tek bir şifre için değil, bütün şifreler için (her kullanıcı için) ayrı ayrı Ram’de yer kaplamaktadırlar. Burada kullanacak olduğumuz karşılaştırma işlemini “strcmp” (string compare) kodu ile sağlamaktayız.
3. “Strcmp” kodu, belirtmiş olduğumuz ayrı dizgileri karşılaştırmak için kullanılmaktadır. Ve aynı zamanda, karşılaştırdığı sonra kendine göre 0 ve 1 gibi değerler döndürmektedir. Buna göre değerlerin birbirine eşit olup olmadığı anlaşılmaktadır.
4. bu kodu Kullanabilmek için kodlarımızı yazarken string.h kütüphanesini sanal çalışma alanımıza eklememiz gerekmektedir (#include <string.h>)

Örnek kodumuzu ekrana yansıttığımız zaman aşağıdaki gibidir:

```
int main()
{
    cout << "Dizgilerin karsilastirilmasi-Sig Kopyalama-Bus Error 10" << endl;

    char *b= "yazilim";
    cout << b << endl;

    char a[8]= "yazilim" ;
    cout << a << endl;
    /* b=a;
    b[2]='x';
    cout << a << endl;
     */

    if(strcmp(b,a)==0) {
        cout << "esittirler" << endl;
        }
    else {
        cout << " esit degillerdir" << endl;
        }
}
```

```
./dizgilerinkarsilastirilmasi
Dizgilerin karsilastirilmasi-Sig Kopyalama-Bus Error 10
yazilim
yazilim
esittirler
Time elapsed: 000:00:733
```

Burada b=a kodunu kaldırdığımız zaman, ekranda verilebilecek olan Bus Error 10 hatasına dikkat edilmesi gerekmektedir. Bu hatanın nedeni, b [2] =’x’ kullanarak dizideki bir karakteri değiştirirken, b=a şeklinde, b bir karakter dizisine eşitti. Dolayısıyla ekrana bastırırken, pointer da tanımlı olan karakter değil de, (eşitlikten dolayı) string de tanımlı olan karakterler ekrana basılıyordu.

Burada string ile karakter dizisinin farkını görebilmemiz mümkündür. Temelde aynı olmakla birlikte, aslında işaretçilerin içeriği değiştirilemezdir (immutable). Karakter dizileri ise bir dizi olduğundan dolayı içeriğine müdahale edebilmeniz mümkündür.

Dolayısıyla b=a ifadesini ekrandan kaldırdığımız zaman, yani b işaretçisini bir diziye eşitlemediğimiz sürece, içeriğini değiştirmemiz hatalı bir kullanımdır. Bus Error 10 hatasını geçebilmek için (b=a’yı kaldırdığımız takdirde), b işaretçisinin içeriğini değiştirmek yerine, a dizisinin içeriğini değiştirerek hatayı ortadan kaldırabilirsiniz.

STRİNG FONKSİYONU YAZMAK-STRCPY VE STRLEN

Bir string fonksiyonunu yazmak, parametre olarak string aldığımız bir fonksiyon yazmak demektir. Örneği verilen bir string değişkeninin boyutunu hesaplamak istersek;

Öncelikle b değişkeninin sıfırıncı değerine bakılacaktır (dolayısı ile string değişkeninin de). Daha sonra b değerinin artan değerlerinde (++b ile bu koşulu sağlıyoruz), string fonksiyonu kontrol edilecektir. Bu while döngüsü, end of string (\0) atanana kadar geçerli olacaktır.

Örnek kodumuz aşağıdaki gibidir;

```
int boyut (char *s) {

    int b=0;
    char c= s[0];
    while(c!='\0') { //end of string gelene kadar kodumuz çalışacak

        c=s[++b]; // döngü her çalıştığında s'in değeri bir artmış olacak

        }
    return b;
    }

int main()
{
    cout << "string fonksiyonu yazmak" << endl;

    char *s="yazilim"; //immutable
    cout << s << endl;

    char c[8]="yazilim";

    cout << c << endl;
    cout << s << endl;
    cout << boyut(c) << endl;
    cout << boyut(s) << endl;
```

```
./stringfonksiyonuyazmak
string fonksiyonu yazmak
yazilim
yazilim
yazilim
7
7
Time elapsed: 000:00:749
Press any key to continue
```

Aynı şekilde bir string fonksiyonun boyutunu görmek istersek, “strlen()” fonksiyonundan yardım almamız mümkündür. “boyut(c)” ya da (s) ile bastırdığımız kodlar ile aynı işlevi görmektedir.

Bir diğer string için kullanabileceğimiz kodumuz “strcpy()” dir. Aynı şekilde bize bir string değişkeninin boyutunu basabileceği gibi, asıl yaptığı işlem deep copy (derin kopyalama)’dır. Yani elimizde bulunan bir string değişkenini kopyalayarak, başka bir string değişkeni oluşturabilmektedir. Bir değişkeni kopyaladığı için iki tane değer alır. İlk girdiğiniz değer oluşacak kopya iken, ikici girilecek olan değer kopyası oluşturulacak olan string değişkenidir.

Örnek kodlarımız ve ekrana yansıtılmış hali aşağıdaki gibidir:

```
int boyut (char *s) {

    int b=0;
    char c= s[0];
    while(c!='\0') { //end of string gelene kadar kodumuz çalışacak

        c=s[++b]; // döngü her çalıştığında s'in değeri bir artmış olacak

        }
    return b;
    }

int main()
{
    cout << "string fonksiyonu yazmak" << endl;

    char *s="yazilim"; //immutable
    cout << s << endl;

    char c[8]="yazilim";

    cout << c << endl;
    cout << s << endl;
    cout << boyut(c) << endl;
    cout << boyut(s) << endl;

    cout << strlen(c) << endl; //boyutu bastırma ile aynı görevi yapmaktadır

    char *x = (char *)malloc(sizeof(char)*8); //boyutunu belirledik
    strcpy(x,s); // deep copy - s değişkenini x'e kopyalama işlemini yapar

    cout << x << endl;
}
```

```
./stringfonksiyonuyazmak
string fonksiyonu yazmak
yazilim
yazilim
yazilim
7
7
7
yazilim
Time elapsed: 000:00:717
Press any key to continue
```

STRİNG TİPİ

Şimdiye kadar olan bölümlerde string değişken tipini genel olarak karakter tipi vb. olarak kullanıldı fakat ayrıca bir değişken tipi olarak atayabileceğiniz “string” de mevcut bulunmaktadır.

String değişken tipi aslında bir char ya da int gibi bir ilkel değişken tipi değildir. Genel olarak kullanımına bakıldığı zaman daha çok nesne yönelimli programlama dillerinde kullanılmaktadır.

Öncelikle bir string değişken tipinin karakter (char) tipinde olan değişkeni içerisinde bulundurmamız mümkündür.

Ayrıca “s.” Diyerek kendi özgü olan fonksiyonları da çağırmanız mümkündür. Örneğin “s.size” dediğiniz zaman boyutunu, “s.append” dediğiniz zaman ise, kodlarınıza ek olarak bir yazı yazdırabilmeniz yanına mümkündür.

Ya da belirli bir metnin sadece bir kısmını “s.substr” ile bastırabilmeniz mümkündür (belirttiğiniz karakterlerden başlayarak ve biterek).

Örnek kodlarımız ve ekrana yansıtılan görüntüleri aşağıdaki gibidir:

```
int main()
{
    cout << "string tipi" << endl;

    string s; //string gelişmiş bir değişken tipidir
    char *p= "yazilim";
    char c[8]= "yazilim";
    s = p; //string tipinde tanımlanan bir değişken, bir işaretçiye eşitlenebiliyor

    /*string tipi kullanıldığı zaman
     * "s. " diyerek string'in herhangi bir
     * özelliğini çağırabilmeniz mümkündür
     */

    cout << s.size() << endl; //boyutunu çağırdık

    s.append("deneme") ; // string ile değerin yanına, bir yazı daha eklemeniz mümkündür
    cout << s << endl;

    cout << s.substr(2,8) << endl; //metinin belirttiğiniz elemanlarından keserek, ekrana yazdırır

    return 0;
}
```

```
./stringtipivecharfarklari
string tipi
7
yazilimdeneme
zilimden
Time elapsed: 000:00:655
Press any key to continue
```

Örnek: Kullanıcıdan 12 saatlik sistemde saati alıp (hh.mm.ssAM veya hh.mm.ssPM), bu saati 24 saatlik sisteme çeviren örnek kodu yazınız.

Örneğin:

Kullanıcıdan alınan saat: 07:05:02PM

Çıktı: 19:05:02

Kullanıcıdan alınan saat: 07:02:00AM

Çıktı: 07:02:00

Öncelikle belirlememiz gereken nokta, kullanıcıdan alınan değeri string değişken tipinde tutmamız gerektiğidir. Çünkü alınacak ve çıktı olarak verilecek olan kodlarımızda hem sayı hem noktalama işareti (iki nokta) hem de harfleri tutabilecek olan değişken tipi şu an ancak string olabilir.

Yazılacak olan kodları analiz etmemiz gerekirse, kullanıcının girecek olduğu değerler hangi saat diliminde olursa olsun PM ve AM yazılarının silinerek ekrana basılması gerekmektedir.

Diğer bir durum ise eğer kullanıcı PM saat diliminde bir değer girmiş ise, ilk sayılara +12 eklememiz gerekmektedir. Bu şekilde 12 saatlik sistemde olan saat, 24 saatlik sisteme uyarlanmış olacaktır. Tam tersi durumda ise (yani kullanıcıdan AM saat diliminde bir değer girilirse), o zamanda sadece yazı kısmı silinmesi gerekecektir.

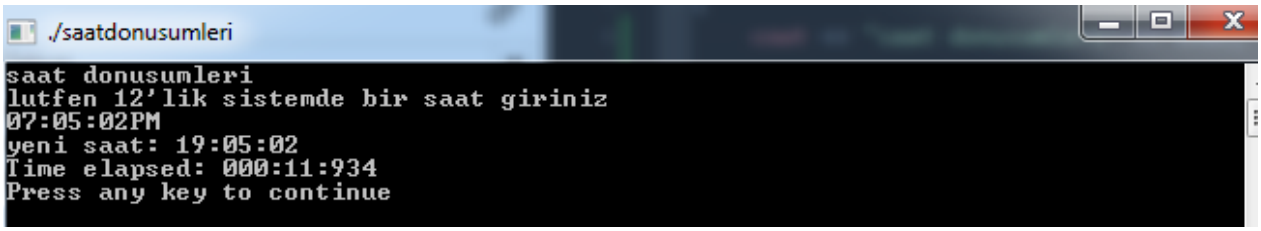
Kodlarımızı yazarken dikkat edilmesi gerek bir diğer hususta boyut konusunda ortaya çıkmaktadır. String değişken tipini kullanırken elimizde kaç tane karakter var ise bir fazlasını almamız gerekmektedir. Çünkü string değişken tipinde ayrıca end of string (string sonu – \0) adını verdiğimiz bir değer daha eklenmektedir. Bizim değerlerimizde sayılar, noktalama işaretleri ve harfler dahil olmak üzere 10 tane değer bulunurken, end of string ile birlikte hafızada 11 değerlik bir yer ayırmamız gerekmektedir.

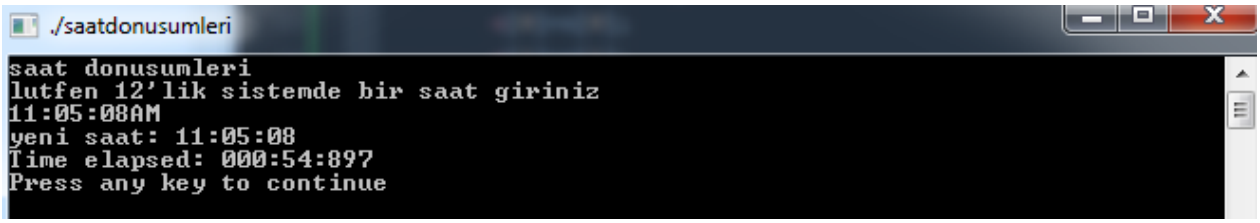
Adım adım gitmemiz gerekirse kodlarımızı düzenlerken;

1. Kullanıcıdan alacak olduğumuz verilerin boyutunu belirtiyoruz.
2. Saat dilimini belirlemek için, bir koşul belirtiyoruz. Koşulumuzu tanımlarken kullanıcının hangi formatta (PM – AM) girdiğini anlayabilmemiz için P harfi mi yoksa A harfi mi şeklinde bir kontrol tanımlıyoruz.
3. Eğer kullanıcının girdiği saat dilimi AM ise, saatler aynı kalacağı ve sadece AM yazısı silineceği için öncelikle koşulun else kısmını düzenliyoruz. Buna göre, kullanacağımız yöntem end of string ile birliktedir. Daha önce belirttiğimiz üzere, eğer biz bir string karakterine end of string (\0) getirirsek, bunun anlamı o string değişkeninin bittiğini göstermektir.
4. Bundan yola çıkarak PM ve AM yazılarını silebilmek için, A ve P harflerini koşul durumlarında güncellemek ve yerlerine end of string gelmesini sağlamalıyız.
5. Geriye düzeltilmesi gereken tek bir işlem kalıyor ki o da, eğer kullanıcı PM saat diliminde bir değer girerse, ilk 2 değeri birlik ve onluk taban olarak ayırarak (düzgün çalışabilmesi için) 12 eklemek.

Örnek kodlarımız ve birkaç örnek ile birlikte çalıştırılmış hali aşağıdaki gibidir:

```
int main()
{
    cout << "saat donusumleri" << endl;

    cout << "lutfen 12'lik sistemde bir saat giriniz" << endl;

    char s[11]; //boyutunu belirtiyoruz
    cin >> s; //kullanıcıdan alıyoruz

    if (s[8]=='P') { //8.karaktere denk geldiği için
        char x[3]; //girilen verilerin ilk üç değerini alıyoruz (değiştirebilmek için)
        x[0]=s[0];
        x[1]=s[1];
        x[2]= '\0'; //bir dizgi olduğu için son karakteri end of string omalı

        int saat=0;
        saat += (x[0]-48) *10;
        saat += x[1]-48;
        saat += 12;

        s[0]=(char)48 + saat/10;
        s[1]=(char)48 + saat%10;


        s[8]='\0'; // P ve geri kalan karakterler basılmayacaktır

        cout << "yeni saat: " << s << endl;
            }
    else { //geriye kalan tek durum else olduğu için else yapısını kullandık

        s[8]='\0'; // A ve geri kalan karakterler basılmayacaktır
        cout << "yeni saat: " << s << endl;
        }
```





Örnek: Kullanıcıdan alınan bir kelimenin palindrome olup olmadığını ekrana yazdıran örnek kodu yazınız.

Örneğin;

Kayak, kaçak, ağa, asa, kabak, mum …

Bir kelimenin okunuşu ile tersten okunuşu aynı ise bu kelimeye palindrome adı verilmektedir.

Bunu sağlayabilmek için başta ve sondaki harflerin eşitliğini kontrol etmemiz gerekmektedir. Bunun için de iki tane işaretçi atayacağız (biri baştan ve diğeri sondan başlamak üzere). Ve bu işaretçilerin ortaya doğru ilerleyerek her harf birbirine eşit mi diyerek kontrol etmelerini sağlayacağız. Döngünün tek bir harf kalana kadar devam etmesini sağlayacağız. Örneğin kabak kelimesindeki p ve q işaretçileri için, bilgisayarın kontrol adımları aşağıdaki gibi olacaktır:

kabak

1.adım: p q (p=q)

2.adım: p q (p=q)

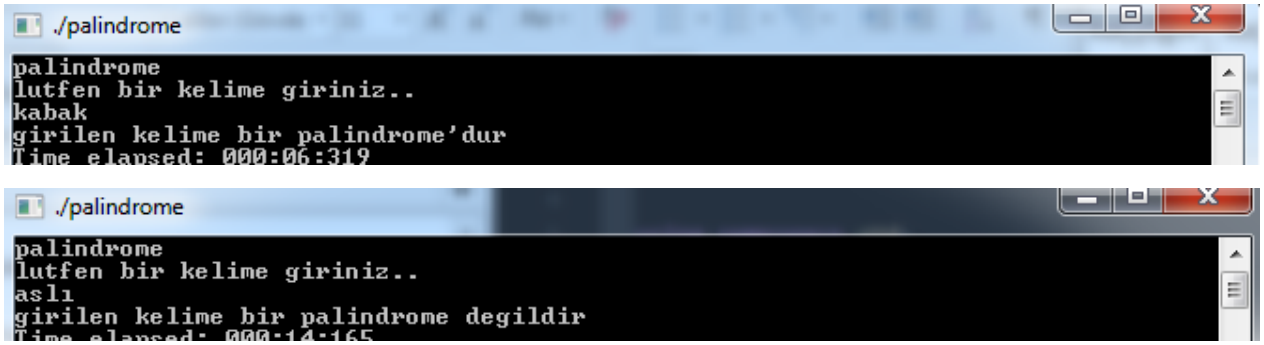
Kodlarımızı yazarken dikkat etmemiz gereken ilk nokta, q işaretçisini sondan başlatmaktır. Dizginin en sonundan başlaması için bir while döngüsü atayacağız ve end of string (\0) kodunu görene kadar ilerlemesini isteyeceğiz. Böylece q işaretçisi kelimenin en sonundan başlamış olacaktır.

İkinci bir nokta ise bool değeridir. Kelimeler ilk girildiği zaman bir boolean değeri atıyoruz ve true (doğru) kabul ediyoruz. Daha sonra döngümüzün içinde de gördüğümüz üzere, p ve q işaretçilerinin birbirinden farklı olduğu tek bir durum için bunu false olarak döndürüyoruz.

Örnek kodlarımız açıklamaları ile birlikte görüntüde verilmiştir;

```
    cout << "palindrome" << endl;
    cout << "lutfen bir kelime giriniz.." << endl;

    char c[100]; //100 karakterlik bir boyut ayırdık
    cin >> c;

    char *p,*q;
    p=c; //dizginin en başından başlaması için
    q=c; /* dizginin en sonundan başlaması için bir
     * while döngüsü atayacağız ve end of string kodunu
     * görene kadar ilerlemesini isteyeceğiz.
     * Böylece q işaretçisi kelimenin en sonundan başlamış olacaktır */

while(*q!='\0') { //q'nun gösterdiği yerdeki değere bakmamız gerektiği için işaretçi kullanıyoruz
        q++;
        }
    q--; //en son eleman \0 olduğu için bir geri gelmesini sağlıyoruz

    bool palindrome=true; //ilk başta tüm kelimeleri palindrome kabul ediyoruz

    while (q>p) { //p ve q birbirine eşit olmadığı zaman (ortada karşılaşmadıklarında)

        if(*p!=*q) { //işaretçi olmalarının sebebi, oradaki değerleri karşılaştırmak istememiz
        palindrome=false; //eğer bir tane bile farklı eleman bulursak false döndürüyoruz
        }
        p++; //diğer harflere geçmeleri için
        q--;
        }
    //sonuçları ekrana yansıtmak için
    if (palindrome)
        cout << "girilen kelime bir palindrome'dur" << endl;
    else
        cout << "girilen kelime bir palindrome degildir" << endl;
```

Ekrana birkaç farklı kelime yazdığımızda ulaşılan sonuçlar aşağıdaki gibidir:



DOSYA İŞLEMLERİ

DOSYALARA GİRİŞ, TEMEL DOSYA TİPLERİ VE BASİT BİR DOSYA AÇMA KODU

Öncelikle c++ dilinde dosya tiplerini kullanabilmek için dosya kütüphanesi olan fstream'i include etmeniz gerekmektedir. Bunun için kodlarınızın başına;

#include <fstream> demelisiniz.

Bir dosyayı açmak için ofstream dosya; diyerek (out file stream) aslında o dosyayı açtığınız ve içine bir şeyler yazacağınızı belirtmiş oluyorsunuz. Aynı şekilde bir dosyayı okurken de ifstream diyerek de dosyayı okuyabilirsiniz.

```
int main()
{
   //dosyayı yazma kismi
   ofstream dosya;
   dosya.open ("deneme.txt");

   if(dosya.is_open()){ //dosyanın açılabilirliğini kontrol ediyor.
       dosya << "rabia yoruk/n";
       dosya.close();
       }
       else {

           cout << "dosya acilmiyor" << endl;
           }

    //dosyayı okuma kısmı
    string satir;

    ifstream dosya2 ("deneme.txt");
    if (dosya2.is_open()) {

        while (getline(dosya2, satir)) { //fstream kaynaklı bir koddur.

            cout << satir << endl;
            }
        dosya2.close();
        }
}
```

```
./dosyaacmayazma
rabia yoruk/n
Time elapsed: 000:
Press any key to c
```

Örnek: Verilen bir metin dosyasındaki bütün karakterleri tersine çeviren örnek kodu yazınız.

Soruyu genel olarak analiz etmemiz gerekirse, örneği yapabilmemiz için öncelikle bir dosyayı açacağız ve bu dosyanın içindeki metin dosyasında, string karakterlerine erişeceğiz. Bu string karakterlerini tersine çevirerek, daha sonra da dosyaya geri yazacağız.

Kodlarımızı yazmaya başlamadan önce, dosyalar ile işlem yapacağımız için fstream kütüphanesini sanal ortamınıza eklemeniz gerekmektedir.

Bir metine ulaşabilmek ya da bir metin oluşturabilmek için, sanal ortamınız içerisinde boş bir dosya açmanız gerekmektedir. Ve açılan dosyayı, kodlarımızda kolay erişebilmemiz için debug klasörünün altına kayıt edilmelidir.

Açılan dosyaya bir metin girdikten sonra, dosyayı açmak için gerekli kodlarımızı girmemiz gerekmektedir [ifstream girdi("girdi.txt");]. Dosyadan okuyacak olduğumuz metin string tipinde olduğu için bir string değişkeni tanımlıyoruz ve dosyadan metne ulaşabilmek için gerekli kodlarımızı [while(getline(girdi,s))] yazarak dosyayı kapatıyoruz [girdi.close();].

Soruda istenene göre, metni tam tersine çevirebilmemiz için metine ulaşmamız yeterli değildir. Bu yüzden metnin içerisindeki karakterlere ulaşacak olan bir kod yazmamız gerekmektedir. Bu yüzden bir başka string değişkeni atıyoruz. Atamış olduğumuz değişkende, karakterlere teker teker ulaşabilmek için bir while döngüsü oluşturuyoruz. Burada yapmamız gereken şey, karakterleri teker teker döngüde incelenirken, end of string kodu ile karşılaşmadığı sürece devam etmesini sağlamamız olacaktır.

While döngüsünde kullanmak üzere p ve q olarak iki tane işaretçi atıyoruz. Böylece karakterlerin yerlerini değiştirmemizi sağlayacaklardır.

P işaretçisinin gösterdiği yerdeki değer, end of string kodundan farklı olduğu sürece devam etmeli ve döngü bittiği zaman ise q işaretçisini bir azaltmalıdır. Bunun sebebi metni tersine çevirirken, aslında baştaki ve sondaki harfleri birbirlerinin yerlerine atamamızdır (kodlarımıza başlarken p ve q işaretçilerini metnin başına ve sonunu işaret etmeleri için tanımlıyoruz).

```
string ters(string s) {
char c = s[0];
int i=1;
char *p;
char *q;
p= & s[0];
q= & s[0];
```

```
while (*p!='\0')
{
p++;
}
p--;
```

Daha önceki yer değiştirme örneklerinde yaptığımız gibi geçici bir char değişkeni atıyoruz ve işaretçilerin birbirleri üzerine yazılmaları sırasında, boşta kalan değerleri tutmasını sağlıyoruz.

```
while (p>q){
    char c = *p;
    *p = *q;
    *q= c;
    p--;
    q++;
    }
```

Bu aşamada kodlarımızı çalıştırdığımız zaman, dosyanın içerisinde bulunan metni tersine çevirmesi için yeterlidir. Fakat eğer isterseniz, metni tersine başka bir dosyada çevirmesini isterseniz de, yazmak için kullandığımız bir ofstream dosyası açarak bunu yapabilmeniz mümkündür. Sadece bunu yaptığınız zaman, ekranda basılmasını sağlamak için, o dosyanın içerisine metni atmayı unutmayınız.

```
    ifstream girdi("girdi.txt");
    string s;
    ofstream cikti("cikti.txt");
    while(getline(girdi,s)) {
        cout << ters(s) << endl;
        cikti << ters(s) << endl;
        }

    girdi.close();
    cikti.close();
}
```

Örnek kodlarımız aşağıdaki gibidir:

```
string ters(string s) {
    char c = s[0];
    int i=1;
    char *p;
    char *q;
    p= & s[0];
    q= & s[0];

    while (*p!='\0')
    {
        p++;
    }    p--;

    while (p>q){
        char c = *p;
        *p = *q;
        *q= c;
        p--;
        q++;
        }
    return s;
    }

int main() {
    cout << "dosyadaki metni tersine cevirme" << endl;
    ifstream girdi("girdi.txt");
    string s;
    ofstream cikti("cikti.txt");
    while(getline(girdi,s)) {
        cout << ters(s) << endl;
        cikti << ters(s) << endl;
    }
    girdi.close();
    cikti.close();
```

Oluşturmuş olduğumuz metin:

```
baslangic
seviyesinde
programlama
dilini basari ile
tamamladiniz
```

Kodlarımızın ekranda tersine çevrilen görüntüsü: