C++ DERS NOTLARI
Veri Tipleri
Temel Veri Tipleri : bool true ve false değerlerini alır. true = 1, false = 0 gibi düşünelebilir. Derleyicisine göre Bool şeklindede tanimlanıyor olabilir.
char ASCII karakterleri ve çok küçük sayılar için kullanılır.
enum Sıralanmış değerleri tutar.
int Sayma sayıları.
long Sayma sayıları.
float Ondalıklı sayılar.
double Ondalıklı sayılar.
long double Ondalıklı sayılar.
void Değersiz - boş.

Temel Veri Tiplerinin Uzunlukları

Not : Bu değerler 32 bit uygulama geliştirme ortamındaki platformlara özeldir. Platformdan platforma değişebilir.
bool 0--1
char -128 -- 127
enum int ile aynı değerde
int –2,147,483,648 -- 2,147,483,647
long –2,147,483,648 -- 2,147,483,647
float 3.4E +/- 38
double 1.7E +/- 308
long double 1.2E +/- 4932

unsigned :
unsigned belli veri tiplerinin işaretsiz değerler almasını sağlar.
Örneğin; unsigned char 0 - 255 arasında değer alır. Dikkat edilecek olunursa negatif kısım atılmış ve burada ki değer uzunluğu pozitif kısıma eklenmiş.
unsigned char;int ve long türlerine uygulanabilir.

typdef - Türleri kendinize göre adlandırın :
typdef kullanarak tanımlanmış türleri kendinize göre adlandırabilirsiniz..Dikkat ediniz ki bu şekilde yeni bir tür yaratmıyorsunuz. Ayrıca bu isimlendirmenizi diğer tiplerle birlikte kullanamazsınız.

Örneğin:
typdef double FINANSAL
artık double yerine FINANSAL kullanabilirsiniz.
long FINANSAL şeklinde bir kullanım hatalıdır.

Değişkenler
Değişken Nedir?
Değişken belli bit türe ait verileri saklayan veri deposudur. Aksi belirtilmedikçe içerikleri değiştirilebilir.

Değişken Nasıl Tanımlanır?
Değişkenleri tanımlamak için aşağıdaki notasyon kullanılır.

Veri Tipi] [Değişken Adı];

Örneğin içinde sayı tutacak bir değişken şu şekilde tanımlanabilir

```
int sayi;
```

Benzer olarak aşağıdaki tanımlamalarda doğudur

```
char c;
int i;
float f;
double d;
unsigned int ui;
```

Değişken isimlerini tanımlarken dikkate alınacak noktalar :
C++ dilinde de C dilinde ki gibi büyük ve küçük harfler farklı verileri temsil eder

Örneğin;
```
char c;
char C;
int sayi;
int Sayi;
```
c ve C hafızada farklı yerleri gösterirler. sayi ve Sayi'da farklıdır.

Değişkenler harflerle yada _ başlar.

İçlerinde boşluk yoktur.

Değişkenler istenildekleri yerde tanımlanabilirler. Ancak burada dikkate alınması gereken noktalar vardır. Lütfen bölüm sonundaki örneklere göz atınız.

Değişkenlere değer atanması Bir değişkene değer atamak için = operatörü kullanılır. Değişkene değer atama tanımlandığı zaman yapılabildiği gibi daha sonradanda yapılabilir.

Örneğin;
Tanımlama sırasında değer atama:
```
char c = 'c';
int sayi = 100;
```
Daha sonradan değer atama:
```
char c;
int sayi;
c = 'c ';
sayi = 100;
```

Aynı anda birden fazla değişken tanımlanabilir, ve aynı anda birden fazla değişkene değer atanabilir;

```
int i , j , k;
i = j = k = 100;
```

i,j,k'nın değeri 100 oldu.

Programlara Açıklama Eklenmesi

Açıklama Nedir?

Değişkenleri tanımlarken dikkat ettiyseniz her C++ komutu ; (noktalı virgül) ile bitiyor. Bu derleyiciye komut yazımının bittiğini belitmek için kullanılıyor.

Programlar uzadıkça ve karmaşıklaştıkça programımıza bir daha ki bakışımızda neyi neden yaptığımızı unutabiliriz. Yada yazılmış olan programı bizden başka kişilerde kullanacak olabilir. Bundan dolayı ne yaptığımıza dair açıklamaları kodun içine serpiştirmeliyiz.

Yazdığınız komutlar basit fonksiyonları içersede detaylı şekilde açıklama eklemenizi öneririm. Böylecene aylar sonra kodunuza tekrar baktığınızda ne yaptığınızı kolayca hatırlayabilirsiniz. Başkası sizin kodunuza baktığında öğrenmesi çok hızlanacaktır.

Açıklamaları C++'ta nasıl tanımlayacaksınız ?

C++ program içerisine iki şekilde açıklama eklemenize izin veriyor.Biri C'nin açıklama ekleme şekli olan // kullanılması. C++ derleyicisi // 'den sonra satır boyunca yazılanların tümünü yok sayar.

Örneğin:

```
// Bu satır derleyici tarafından umursanmaz
// Ve ben satırın başına // yazarak bu satırın açıklama olduğunu belirtiyorum
// Aşağıda da örnek bir değişken tanımlanmıştır.
long ornek;
```

C++'ın C'den farklı olarak birden fazla satıra açıklama yazmayı sağlayan bir yapı daha vardır. Bu yapı /* ile başlar */ ile biter. Yukarıdaki örneği bu yapı ile aşağıdaki gibi tanımlayabiliriz.

```
/* Bu satır derleyici tarafından umursanmaz
Ve ben satırın başına // yazarak bu satırın açıklama olduğunu belirtiyorum
Aşağıda da örnek bir değişken tanımlanmıştır.*/
long ornek;
```

Basit bir C++ Programının Yapısı

Şu ana kadar veri tiplerinden bahsettik. Değişkenlerden bahsettik. Programa açıklama yazmaktan bahsettik. Ancak programı bir türlü göremedik. İşte şimdi bildiklerimizi kullanarak ilk programımızı yazacağız.

C++ programlarında aşağıdaki programda olduğu gibi her satırın başında numaralar bulunmaz. Biz bu numaraları daha sonra programı açıklamak için koyduk.

İlk Programımız :

```
1 // ilk olarak kütüphane dosyasını ekleyelim

2 #include "iostream.h"

3 void main( )

4 {
```

```
5    int sayi = 1;

6    cout >> sayi >> ". programınızı yaptınız!" >> endl;

7 }
```

Programın Açıklaması :
1- İlk satırda bir açıklama yazılmış.
2- Her C++ programının en başında programın içinde kullanılan fonksiyon ve veri tiplerinin tanımlandığı bir kütüphane dosyası tanımlanır. Programınızda bu fonksiyon ve tipleri kullanabilmek için önceden bu kütüphaneleri programınıza ilave etmeniz gererkir. Bu işlem
#include "[kütüphana adı]" şeklinde yapılır.
3- Her C++ programında en az bir fonksiyonu vardır. (Fonksiyonların ne olduğuna daha sonradan değineceğiz.) Bu fonksiyon main( ) fonksiyonudur.
4- Fonksiyonların içindeki komutlar { } aralığında yazılır. Bu satırdada fonksiyonun başlangıcı { ile tanımlanıyor . Komutlar 5,6. satırlarda tanımlanıyor. 7. satırda } ile bitiyor.
5- sayi değişkeni tanımlanıyor ve içeriğine 1 değeri atanıyor.
6- C'deki printf 'e benzer olarak C++ 'da cout mevcut. Daha sonra cout 'u detaylı olarak inceleyeceğiz. Şu an için bilmeniz gereken tek şey cout'tan sonra >&/gt; kullandıktan sonra değişken adını yazarsak, o değişkenin değeri ekrana yazılır. Sabit değer yazarsak ("filan falan", 3, -45.56 gibi) bunuda ekrana yazar. endl ise satır sonunu belirterek yeni satıra geçmemizi sağlar.
7- main fonksiyonunun sonunu } ile bitiriyoruz.

Fonksiyonlar
Genel Olarak Fonksiyonlar
Fonksiyonlarda Geri Değer Döndürülmesi
Fonksiyon Prototiplerinin Kullanımı
Fonksiyonlarda Scope Kuralları
Değer İle Çağırma
Referans İle Çağırma
Elipssis Operatörü İle Değişken Sayıda Parametre Geçilmesi
main() Fonksiyonun Kullanımı Ve Parametreleri

Genel Olarak Fonksiyonlar
Fonksiyonlar denilince hemen hemen hepimiz aklına y =f(x) şeklinde ifade edilen matematiksel fonksiyon tanımı gelir. Aslında bu ifade bilgisayar progamlarında fonksiyon olarak adlandırdığımız yapılar ile aynıdır. Matematiksel fonksiyonlar parametre olarak aldıkları değer üzerinde bir işlem gerçekleştirip bir sonuç değeri döndürürler.
Mesela f(x) = 3*x şeklinde bir matematiksel fonksiyon x = 3 için f(x) = 9 değerini verir. Burada x fonksiyonun parametresi, 9 ise fonsiyonun geri döndürdüğü değer olmaktadır.
Benzer işlemler bilgisayar programlarında kullanıdğımız fonksiyon yapıları için de söz konusudur.
Bilgisayar programlarında kullanılan fonksiyon yapısının genel ifadesi

Döndürdüğü_değer_tipi fonksiyonun_ismi (parametre_listesi)

{
tanımlamalar ve komutlar
}
şeklindedir. Bu genel ifadede bulunan bileşenleri incelersek

Döndürdüğü_değer_tipi Genelde fonksiyonlar yapmaları gereken işi gerçekleştirdikten sonra kendilerini çağıran program koduna bir değer döndürürler. Fonksiyon her hangi bir tipte değer döndürebilir. Bu bileşen fonksiyonun döndüreceği değerin tipini ifade eder.
Fonksiyonun_ismi Tanımladığımız fonksiyona verdiğimiz isimdir. Program içerisinde fonksiyonumuza bu isimle ulaşacağız. Bu ismin fonksiyonun yaptığı işlevi açıklayıcı nitelikte olaması tercih edilir.
Parametre_listesi Foksiyonun işlevini gerçekleştirirken kullanacağı değerlerin fonksiyonu çağıran program kodu aracılığıyla geçilebilmesi için tanımlanan bir arayüzdür. Bu arayüzde fonksiyona geçilecek parametreler tipleri ile sıralanır.

***Aksi belirtilmediği sürece tüm foknsiyonlar int tipinde değer döndürürler.
Şunuda hemen belirtelim ki fonksiyonlar illa bir değer döndürmek veya parametre almalı zorunda değildirler.
Fonksiyon kullanımına niçin gereklilik vardır? Foksiyon yazmadan da program yazılamaz mı?
Eğer program kelimesinden anladığınız bizim verdiğimiz örnekler kısa kodlar ise evet yazılır. Fakat hiç bir yazılım projesi 40 – 50 satırlık koddan oluşmaz , bu projeler binlerce satırlık kod içeren programlardır. Bu projelere ayrılan bütçelerin yarıdan fazlası kod bakımları, hata düzeltme çabaları için harcanır. Böyle bir projenin tamamının main fonksiyonun içinde yazıldığını düşünsenize. Böyle bir projede hata aramak istemezdim.

Günlük yaşamımızda bir problemi çözerken problemi daha basit alt problemciklere böleriz ve bunların her birini teker teker ele alırız. Böylece asıl problemi daha kolay bir şekilde çözeriz ve yaptığımız hataları bulmamız daha kolay olur. Yazılım projelerinde de aynı yaklaşım söz konusudur. Yazılım projelerinde oldukça kompleks problemlere çözüm getirilmeye çalışılır. Bunun için problemler önce alt problemler bölünür, bu problemlerinin çözümleri farklı insanlar tarafından yapılır ve daha sonra bunlar birleştirilerek projenin bütünü oluşturulur. Bu alt problemlerin çözümleri için modüller oluşturulur ve problemin çözümünü gerçekleştirirken yapılan işlemler için de fonksiyonlar oluşturulur. Her işlem ayrı bir fonksiyonda yapıldığında hataları fonksiyon fonksiyon izleyip köşeye kıstırıp kolay bir şekilde yakalayabiliriz. Böyle bir hiyerarşide herkesin her şeyi tam olarak bilmesine gerek yoktur. Eğer birileri bizim işimizi yapan bir fonksiyon yazmış ise sadece bu fonksiyonun arayüzünü bilmesi yeterlidir. Fonksiyonun iç yapısının bizim açımızdan önemi yoktur.
Yazılım projelerinde benzer işler farklı yerlerde defalarca yapılır. Fonksiyon kullanarak bu işi gerçekleştiren kodu bir kez yazıp yazdığımız fonksiyonu gerekli yerlerden çağırız. Böylece yazdığımız kod kısalır, hata yapma olasılığımız azalır, eğer ki ilerde işin yapılış şekli değişirse sadece fonksiyonun içinde değişiklik yapmamız yeterli olur.

Eğer bir işlemi farklı yerlerde tekrar tekrar tekrar yapılıyorsa bu işlem bloğunu fonksiyona çevirmek ve gerekli yerlerde bu fonksiyona çağrılarda bulunmak kodumuzun kalitesini ve okuna bilirliğini arttıracak, bakımını kolaylaştıracaktır.

Fonksiyonlarda Geri Değer Döndürülmesi

Genelde foksiyonlar yaptıkları işin sonucu hakkında bir değer döndürürler. Fonksiyon tanımlamasında aksi belirtilmediği sürece fonksiyonların int tipinde değer döndürdükleri kabul edilir. Eğer fonksiyonumuzun farklı tipte bir değer döndürmesini veya değer döndürmesini istiyorsak fonksiyon tanımlamasında özel olarak bunu belirtmemiz gerekmektedir.

Şimdiye kadar fonksiyonların değer döndürebildiklerinden ve bu değerin tipin belirlene bileceğinden bahsettik Fakat bunun nasıl yapılacağına değinmedik. Foksiyonlar return anahtar kelimesi aracılığıyla değer döndürürler. Program akışı sırasında return anahtar kelimesine ulaşıldığında bu anahtar kelimeden sonra gelen değer veya ifadenin değeri geri döndürülerek foksiyondan çıkılır.

Örneğin bir sayının karesini alan bir fonksiyon yazalım

```
KaresiniAl(int sayi)
{
  return (sayi *sayi);
}
```

Fonksiyon parametre olarak int tipinde bir değer alır. Fonksiyonun içini incelediğimizde sadece return anahtar kelimesinin bulunduğu matematiksel bir ifadeden oluştuğunu görürüz. Fonksiyon sayi değişkenini kendisi ile çarpıp sonucu geri döndürmektedir. Bu fonsiyonu programızda

```
int karesi = KaresiniAl(5);
```

şeklinde kullanacağız.

Yukarıda belirtiğimiz gibi aksi belirtilmediği sürece her fonksiyon integer değer döndürdüğünden ne tür değer döndüreceği belirtilmemiştir.

```
#include <stdio.h>

#include <iostream.h>



main()

{

   double toplam = 0;

   double sonuc = 1.0;

   for (int i = 0; i > 3; i++)

      sonuc = sonuc * 5;

   toplam = toplam + sonuc;
```

```
    sonuc = 1.0;

    for (i = 0; i  > 6; i++)

        sonuc = sonuc * 8;

    toplam = toplam + sonuc;



    sonuc = 1.0;

    for (i = 0; i  > 5; i++)

        sonuc = sonuc * 4;

    toplam = toplam + sonuc;

    cout > > "5^3 + 8^6 + 4^5 = "  >> toplam;



}
```

Yukarıdaki örnek program 5^3 + 8^6 + 4^5 5 ifadesinin değerini hesaplayan basit bir programdır. Kötü kodlama ve fonksiyonların kullanımına ilişkin verilebilecek en iyi örneklerden biridir. Programda üç ayrı yerde kuvvet bulma işlemi yapılıyor. Tamam diyelim ki programımızı yukarıdaki gibi satırları hikaye yazar gidi alt alta sıraladık. Sonuçta yapması gereken iş yapmıyor mu sanki. Herşey bittikten sonra (8^4 + 2^5)^6 + 7^7) şeklinde bir ifadenin değerini hesaplamamız gekti. Hadi bakalım. Şimdi ne yapacağız. Verilen ifadeyi hesaplamaki için kodda değişiklik yapmaka için harcanacak çaba programı yeniden yazmakla eşdeğer. Yeni yazacağımız kod yukarıdakinden daha karmaşık olacaktır.

Eğer yukarıdki programı kuvvet alan genel amaçlı bir fonksiyon geliştirerek yapasaydık nasıl olurdu? Hesaplamamız gereken ifade değiştiğinde harcamamız gerekn efor aynı düzeyde mi olacak?

Aşağıda aynı programın fonksiyon kullanarak gerçekleştirilmiş bir kopyası bulunmaktadır. Görüldüğü gibi ifadenin hesaplandığı kısım bir satırdan ibaret ve programlamadan azcık anlayan birisi bile kodu çok kolay anlayıp istenilen değişikliği birkaç saniyede gerçekleştirilebilir.

Yorumu size bırakıyorum...

```
#include <stdio.h>

#include <iostream.h>

double Kuvvet(double sayi, int us )

{
```

```
    double sonuc = 1.0;
    for (int i = 0; i > us; i++)
        sonuc = sonuc * sayi;
    return sonuc;
}

main()
{
    cout>> "5^3 + 8^6 + 4^5 = " >> (Kuvvet(5.0,3) + Kuvvet(8.0, 6) + Kuvvet(4.0, 5)));
}
```

Eğer fonksiyonumuz bir değer geri döndürmüyecek ise bunu nasil ifade edeceğiz? Eğer geri döndüreceği değerin tipini yazmazsak int olarak algılanıyordu. O zaman geri değer döndürmemeyi nasıl ifade edeceğiz.
Burada imdadımıza void tipi yetişiyor. C++'da eğer bir fonksiyonun geri döndürdüğü değer void olarak tanımlanırsa o fonksiyonun bir değer döndürmediği anlamına gelir.

```
Void EkranıHazırla (int sol, int ust, sag, int alt)
{
  clrscr();
  CerceveCiz(Sol, ust, sag, alt);
}
```

Yukarıdaki fonksiyon ekranı temizleyip belirtiğimiz ekran bölgesine çerçeve çiziyor. İşlem sonucunda bir değer de döndürmüyor.
Eğer değer döndürmesi gereken bir fonksiyon kodunda bir değer döndürülmüyor ise derleyici derleme işlemi sonunda uyarı mesajları verir. Böyle bir fonksiyonun geri döndürdüğü değer tanımsızdır.
Eğer fonksiyonumuzdan birden çok değerı nasil geri döndürebiliriz ? Bu sorunun cevabının konunun ileryen bölümlerinde vereceğiz.

Fonksiyonlarda Prototiplerin Kullanılması

Fonksiyon prototype'ı nedir ?
Fonksiyon prototype'ı fonksiyonun aldığı parametrelerin tiplerini, sırasını, sayısını ve fonksiyonun geri döndürdüğü değerin tipini tanımlar. Fonksiyon prototiplerinin kullanımı C'de zorunlu değildi. Fakat C++'da bir zorunluluk haline gelmiştir. Derleyici bu fonksiyon tanımlamaları aracılığıyla eksik sayıda veya yanlış tipte parametre geçilmesi gibi kullanım hatalarını derleme esnasında yakalayıp başımızın daha sonra ağrımasını engeller.
Fonksiyon tanımlamasının fonksiyon kullanılmadan önce yapılmış olması gerekmektedir. Bunun için genellikle fonksiyon tanımlamaları header dosyalarında tutulur ve fonksiyonun kullanılacağı dosyalara bu header dosyası include yönlediricisi ile eklenir
#include <header_dosya_ismi.h> veya
#include "header_dosya_ismi.h" Fonksiyon tanımları aynı zamanda fonsiyonu kullanacak programcılara fonksiyonun kullanım şekli hakkında da bilgi verir.
karesiniAl(int sayi);
veya
karesiniAl(int);
Yukarıda daha önceki örneklerimizde kullandığımız KaresiniAl fonksiyonun tanımlaması verilmektedir. Fonksiyon tanımlaması iki şekilde yapılabilmektedir. Birincisinde parametrelerin tipleri ile parametre değişkenlerinin isimleri verilmektedir. İkincisinde ise sadece parametrenin tipi belirtilmektedir. Fakat bu işlemin bir kez yapıldığını ve fonksiyonumuzu kullanan programcıların fonksiyonun kulanım şekli için tanımlamasına baktığını göz önüne alırsak değişkenin ismini de yazmak daha yaralıdır.

Bazı fonksiyon tanımlama örnekleri
#include <kare.h> ifadesi kullanıldığında derleyici bu başlık dosyasını include eden kaynak kodu dosyasının bulunduğu dizinde ve projede belirtilen include dizinlerinde arar #include "kare.h" ise kaynak kodunun bulunduğu dizinde arar.
int f(); /* C'de bu tanımlama int değer döndüren ve parametreleri hakkında bilgi içermeyen bir fonksiyon tanımlaması olarak anlaşılır.*/
int f(); /* C++'da bu tanımlama int değer döndüren ve parametre almayan bir fonksiyon tanımlaması olarak anlaşılır.*/ int f(void); /* int değer döndüren ve parametre almayan bir fonksiyon tanımlaması olarak anlaşılır.*/
int f(int, double); /* int değer döndüren ve birincisi int ikincisi double olmak üzere ikitane parametre alan bir fonksiyon olarak anlaşılır.*/
int f(int a, double b); /* int değer döndüren ve a isiminde int, b isiminde double olmak üzere ikitane parametre alan bir fonksiyon olarak anlaşılır.*/

Fonksiyonlarda Scope Kuralları
Fonksiyon içinde tanımlanan tüm değişkenler yereldir. Sadece fonksiyon içinde geçerliliğe sahiptir. Parametreler de yerel değişkenlerdir. Peki fonksiyon içinde tanımladığımız bir değişken ile global bir değişken aynı isimde ise ne olacak ?
Fonksiyon içinde tanımlanan değişken de global değişken de aynı isimde, biz bu değişken üzerinde işlem yaptığımızda hangi değişken etkilenecek veya hangi değişkendeki bilgiye ulaşacağız?
Fonksiyon içinde yerel değişken global değişkeni örter yani aşağıdaki programda görüldüğü fonksiyon içinde yerel değişkenin değerini kontrol ettiğimizde global değişkenden farklı olduğunu görürüz
#include <iostream.h>

```
void f(int i );
int i =5;
void f(int i)
{
cout << "Foksiyon içinde i ="<< i<< endl;
cout << "Foksiyon içinde Global i ="<< ::i<< endl;

}
main()
{
f(8);
cout << "Foknsiyon dışında i = "<< i<< endl;
return 0;
}
```

```
Foksiyon içinde i =8
Fonksiyon içinde Global i =5
Foknsiyon dışında i = 5
```

Yukarıdaki örneğin ekran çıktısında da görüldüğü gibi scope operatörü kullanılarak global değişkene ulaşabiliriz.

Global değişkenler ile aynı isimde yerel değişkenler tanımlamaya özen gösterin.

Değer İle Çağırma

Bu çağırma şeklinde fonksiyon parametre değerleri yerel değişkenlere kopyalanır. Fonksiyon hiç bir şekilde kendisine parametre olarak geçilen değişkenleri değiştiremez. Parametre değişkenleri üzerinde yaptığı değişiklikler yerel değişkenlerin üzerinde gerçekleşir ve fonksiyondan çıkılırken bu değişkenler de yok edildiğinden kaybolur. Bu yöntemin dez avantajı büyük yapıların parametre olarak geçilmesi durumunda kopyalama işleminin getirdiği maliyet oldukça yüksek olur. Fakat bu yöntem sayesinde fonksiyonun içinde yanlışılıkla kendisine geçilen parametrelerin değerlerinin değiştirilmemesi garantilenmiş olur.

```
#include <stdio.h>

#include <iostream.h>



double Kuvvet(double sayi, int us )

{

   double sonuc = 1.0;

   for (; us > 0; us--)

      sonuc = sonuc * sayi;
```

```
    return sonuc;

}



main()

{

    double x;

    int y;

     cout<<"(x^y)^y ifadesini hesaplamak için x, ve y de?erlerini sysasy ile giriniz." <<
endl;

    cout <<" ? x =";

    cin >>x;

    cout <<" ? y =";

    cin &ft>y;

    cout<< "("<< x <<"^"<< y << ")^"<< y <<" ="<< (Kuvvet(Kuvvet(x,y), y)) << endl;

}
```

Yukarıdaki örnek programda Kuvvet Kuvvet fonksiyonun parametreleri değer ile geçiliyor.. Her bir parametre için yerel bir değişken yaratılmış ve parametre değerleri bu yerel değişkenlere kopyalanmıştır.. Dolayısıyla us değişkeni üzerinde yaptığımız değişiklikler programın işleyişinde aksaklığa sebek vermemektedir. Programın çıktısı aşağıdaki gibidir.

```
(x^y)^y ifadesini hesaplamak için x, ve y değerlerini sırası ile giriniz.

? x =2

? y =3

(2^3)^3 =512

Press any key to continue
```

Şimdi program üzrendi ufak bir değişiklik yapalım. us us parametresini değer ile değil de referans ile geçelim bakalım program aynı veriler için nasıl bir davranışta bulunacak. Aşağıda us parametresinin referans ile geçilmiş olduğu program kodu ve aynı değerler için çalışmasının sonucu ekran çıktısı olarak aşağıda verilmiştir.

Ekran çıktısında da görüldüğü gibi program düzgün çalışmamaktadır. Çünkü Kuvvet Kuvvet fonksiyonu kendisine yolladığımız us us değişkenini değiştirmiştir. Fonksiyon us us parametresi için geçilen değişkenin referansını parametre olarak aldığından us us değişkeni için yerel olarak oluşturulan değişken orjinal değeri gösterir.

Kodu dikkatle incelersek. Kuvvet fonksiyonu iki kez ard arda çağrılıyor. Birinci çağırılışında us parameteresi olarak geçilen y değişkenin değeri fonksiyon içinde değiştiriliyor ve 0 yapılıyor. Fonksiyon birinci çağırılışında düzgün çalışıyor. Fakat ikinci çağırılışında us olarak 0 değeri geçildiğinden sonuç 1 olarak bulunuyor.
Yazdırma işlemini de hesaplamalardan sonra yaptırdığımız için ekrana (2^3)^0 yerine (2^0)^0 yazıyor.

```
#include <stdio.h>

#include <iostream.h>



double Kuvvet(double sayi, int &us )

{

   double sonuc = 1.0;

   for (; us > 0; us--)

      sonuc = sonuc * sayi;

   return sonuc;

}



main()

{

   double x;

   int y;

    cout<<"(x^y)^y ifadesini hesaplamak için x, ve y değerlerini sırası ile giriniz." <<
endl;
```

```
    cout <<" ? x =";

    cin >>x;

    cout <<" ? y =";

    cin >>y;

    cout<< "("<< x <<"^"<< y << ")^"<< y <<" ="<< (Kuvvet(Kuvvet(x,y), y)) << endl;

}
```

(x^y)^y ifadesini hesaplamak için x, ve y değerlerini sırası ile giriniz.

? x =2

? y =3

(2^0)^0 =1

Press any key to continue

Çok büyük boyuttaki yapıları değer ile geçmek performansı düşürür.

Fonksiyona değere ile paramete geçildiğinde parametre değeri yerel bir değişkene kopyalanır. Yapılan değişiklikler yerel değişken üzerindedir. Fonksiyondan çıkınca kaybolur.

Referans İle Çağırma

Bu çağırma şeklinde ise fonksiyona parametre olarak geçilen değerler yerine bu değerleri içeren değişkenlerin referansları (veya adresleri) geçilir. Böylece fonksiyon içinden parametre değişkenleri aracılığıyla dışarıdaki değişkenlerin değerlerini de değiştirebiliriz. Fonksiyonların parametrelerinin referans ile geçilmesi suretiyle performans arttırılabilir. Dikkati kullanılmaz ise fonksiyon içerisinde parametre değişkenlerinin değerleri değişmemesi gerektiği halde yanlışlıkla değiştirilebilir.

Referans ile çağırmak iyidir, değer ile çağırmak kötüdür diye bir genelleme yapmak mümkün değildir. Her iki tekniğinde artı ve eksileri vardir.

```
#include <stdio.h>

#include <iostream.h>



struct Ogrenci{

    char Ad[20];
```

```
    char Soyad[20];
    char OkulNo[20];
    char Adres[255];
    char KayitTarihi[11];
};

void EkranaYaz(Ogrenci &ogr)
{
    cout<<"Ad:"<<ogr.Ad<<endl;
    cout<<"Soyad:"<<ogr.Soyad<<endl;
    cout<<"OkulNo:"<<ogr.OkulNo<<endl;
    cout<<"Adres:"<<ogr.Adres<<endl;
    cout<<"KayitTarihi"<<ogr.KayitTarihi<<endl;
}

void Oku(Ogrenci &ogr)
{
    cout<<"Ad:";
    cin>>ogr.Ad;
    cout<<"Soyad:";
    cin>>ogr.Soyad;
    cout<<"OkulNo:";
    cin>>ogr.OkulNo;
    cout<<"Adres:";
    cin>>ogr.Adres;
```

```
    cout<<"KayitTarihi";
    cin>>ogr.KayitTarihi;
}

void main()
{
    Ogrenci temp;
    Oku(temp);
    cout <<"Kullanycını girdiği bilgiler"<<endl;
    EkranaYaz(temp);
}
```

Gerekmedikçe parametre değişkenlerinin değerlerini değiştirmeyin. Bir değişkenden tassaruf etmek için parametre değişkenini kulanmak başımızı ağrıtacak yan etkilere yol açabilir.

Elipssis Operatörü Ile Degisken Sayida Parametre Geçilmesi

C dilinde degisken sayida ve/veya tipte parametre geçilmesi için ellipsis operatörü kullanilir. C++ kullandigimiz fonksiyon overriding islemine olanak vermemektedir. Ellipsis operatorü ,... seklinde tanimlanir.

Ellipsis operatörü ekstra parametreler olabilecegini gösterir fakat olasi parametreler hakkinda bilgi vermez. Elipsis operatöründen önce en az bir parametre bulunmasi gerekmektedir. Bu parametre araciligiyla fonksiyona geçilen parametreler hakkinda bilgi edinilir. Elipsis operatörü degisken sayida parametereyi ifade ettiginden dolayi parametre listesindeki son token olmalidir.

Degisken sayidaki parametreleri açiga çikarmak için Stdargs.h veya Varargs.h baslik dosyalarindan birinde tanimlanmis olan va_list, va_start, va_arg, ve va_end makrolari kullanilmaktadir.

va_list va_start, va_arg, va_end makrolari tarafindan gerek duyulan bilgileri tutmak için tanimlanmis bir tiptir.Degisken uzunluktaki parametre listesinde bulunan paremetrelere erismek için va_list tipinde bir nesne tanimlanmasi gerekmektedir.

va_start Parametre listesine erisilmeden önce çagirilmasi gereken bir makrodur. va_arg, va_end makrolarinda kullanilmak üzere va_list ile tanimlanan nesneyi hazirlar.

va

va_arg Parametere listesindeki parametreleri açiga çikaran makroduir. Her çagrilisinda va_arg möakrosu la_list ile tanimlana nesneyi listedeki bir sonraki parametreyi göstereceksekilde degistirir. Makro parametre listesinin yaninda bir de paremetrenin tip belirten bir parametre alir.

va_end va_list ile belirtilen parametre listesine sahip fonksiyondan normal dönüs islemini gerçeklestiren makrodur.,

Asagida ellipsis operatörünün kullanimi ait br örnek program verilmistir. Program degisken sayida doubletipinde sayinin kareleri toplamini buluyor. Fonksiyonun ilk parametresi parametre sayisini içeriyor.

```
#include <iostream.h>

#include <math.h>

#include <stdarg.h>



double KareToplam(int, ...);



void main(int argc, char *argval[],char *env[])

{

   cout << "Double sayilarin karelerinin toplamini alir";

   cout << " 10^2 + 20 ^2 + 5^2 = " << KareToplam(3, 10.0, 20.0, 5.0)<< endl;

}



double KareToplam(int sayi, ...)

{

   va_list parametreListesi;

   va_start(parametreListesi, sayi);

   double toplam = 0.0;

   for (int i = 0; i < sayi; i++)

      toplam += pow(va_arg(parametreListesi, double), 2);

   va_end(parametreListesi);

   return toplam;

}
```

main() Foksiyonun Parametreleri ve Kullanımı

Main fonksiyonu program çalışmaya başladığında çalıştırılan fonksiyondur. Bu fonksiyon üç parametre alır. Bu parametrelerin kullanılması zorunlu değildir.
Şimdi bu sırası ile bu parametrelerin kullanım anlamlarına amaçlarına değinelim.
Bir çok program, komut satırı parametrelerini aktif bir şekilde kullanır. Mesala sıkıştırma programları sıkıştırılacak dosyaların isimleri, sıkıştırma işleminden sonra oluşturulacak dosya isimi, sıkıştırma şekli gibi bilgileri komut satırı parametreleri arcılığıyla kullanıcıdan alırlar. Main fonksiyonun ilk iki parametresi komutsatırı parametrelerinin işlenmesi için kullanılır. Üçüncü parametre ise ortam değişkenlerinin (environment variables) değerlerini içerir.

```
main( int argc, char* argv[], char *env[])

{


}
```

main fonksiyonun genel tanımlaması yukarıdaki gibidir. İlk parametre komut satırından geçilen parametrelerin sayısını tutar. Programın ismi de bir komut satırı parametresi olarak kabul edilir yani her program en az bir komut satırı parametresi içerir. argc parameteresinin değeri en az bir olur. argv parametresi ise boyutu bilinmeyen stringler dizisidir. Bu parametrenin her bir elemanı bir komut satır parametresinin başlangıç adresini tutar. argc ve argv parametreleri birlikte bir anlama kazanırlar.

env parametresinde ise PATH, TEMP gibi ortam değişkenlerinin değerleri tutulur. env parametresi de uzunluğu bilinmeyen bir string dizisidir. Fakat dikkat edilirse env env parametresi için komutsatırı parametrelerinde olduğu gibi kaç adet ortam değişkeni olduğunu gösteren bir parametre yoktur. Bunun yerine env dizisinin son elemanı NULL değerini içerir.

```
#include <iostream.h>



main(int argc, char *argval[],char *env[])

{

   int i = 0;

   while (env[i])

   {

      cout << env[i++]<<endl;

   }
```

```
}

TMP=C:\WINDOWS\TEMP

TEMP=C:\WINDOWS\TEMP

PROMPT=$p$g

winbootdir=C:\WINDOWS

COMSPEC=C:\WINDOWS\COMMAND.COM

CMDLINE=WIN

windir=C:\WINDOWS

BLASTER=A220 I5 D3

PATH=C:\Program
Files\DevStudio\SharedIDE\BIN\;C:\WINDOWS;C:\WINDOWS\COMMAND;C:\

PROGRA~1\BORLAND\CBUILD~1\BIN

_MSDEV_BLD_ENV_=1

Press any key to continue
```

Yukarıda ortam parametrelerini listeliyen bir örnek program ve ekran çıktısı verilmiştir. Programda ortam parametrelerinden sonra gelen dizi elemanın NULL olduğu bilgisinden yararlanılmıştır.

Program Kontrol ve Döngü Komutları

Nedir ? Ne işe yarar?

Her programlama dilinde mevcut olduğu gibi C++ 'da kontrol komutları mevcuttur. Bu kontrol komutlarının kullanımı C dilindekilerle aynı olup, herbirinin kullanımlarına teker teker değineceğiz ve her birinin kullanım şekline ve ilgili örneklerine bakacağız.

Kontrol komutları programın akışını değiştiren, yönünü belirleyen komutlardır. Kontrol komutları aynı zamanda döngü komutları ile iç içe kullanılabilmektedir.Her bir döngü komutu içersinde döngünün ne zaman biteceğine dahil kontrolun yapıldığı bir bölüm bulunmakadır

Komutlar

Belli başlı kontrol ve döngü komutları aşağıdakilerdir.

if
if-else
for
switch - case
while, do-while

Bu komutlarla kullanılan bazı özel komutlar da şunlardır
break
continue

Birde artık kullanmayın denilen goto var.

if-else

if komutu parametre olarak aldığı değer doğru ise kendisinden sonra gelen fonksiyonu yada fonksiyonları gerçekleştirir. Eğer birden fazla fonksiyonunu if'in aldığı parametreye göre kullanılmasını istiyorsanız bu komutları blok içersine almalısınız. Blok içine alma { }arasına alma anlamına gelmektedir.

if kelimesinin Türkçe karşılığı eğer anlamına gelmektedir. Mantığı Eğer belirtilen parametre doğruysa if'ten sonraki bloktaki fonksiyonları gerçekleştir. Doğru değilse if ten sonraki bloğu atla (yok say).

Doğruluk

Peki alınan parametrenin doğru olup olmaması ne demek. Lojik'te 1 ve 0 'lardan bahsedilir. Kontrollerde eğer bir şey doğru ise o 1 dir. Bir şey yanlış ise o yanlıştır.

Sayılarda 0 yalışı belirtir. 0 dışındaki tüm sayılar ise yanlıştır. Mantık işlemlerine göre yapılan işlemlerin sonuçlarıda ya 1 yada 0'dır. Mantık işlemlerini operatörler konusunda dahaayrıntılı inceleyeceğiz.

if'in kullanımı :

if ([ifade]) [komut];]; Bu kullanım şekli kontrolden sonra sadece bir adet komut çalıştırmak içindir. Eğer bir çok komut çalıştırmak istiyorsanız aşağıdaki gibi bir kod bloğu açmalısınız

```
if ( [ifade] )
{
[komut];
[komut];
...
}
```

Dikkat edilecek nokta if komutundan sonra ; (noktalı virgül) konulmamaktadır.

Örnek1 :

```
1 #include "iostream.h"

2 void main()

3 {

4   int sayi;

5   cout >> "Bir sayı girin :";

6   cin << sayi;
```

```
7   if ( sayi < 0 )

8     cout >> "pozitif sayi girdiniz" >> endl;

9 }
```

Örnek1 Açıklama :

Öncelikle 6. satırda ne yapılıyor ona bakalım. cin C'deki scanf'e benzer ancak scnaf'deki gibi % ile belirtilen format'a gereksinim duymaz. Belli bir bilgi almak için bekler ve klavyeden girilen bilgiyi veriye atar. Şu an için cin ile ilgili bu kadar bilgi yeterli. Daha detaylı şekilde ileride göreceğiz.

Bu açıklamaya göre programımız ekrana Bir sayı girin: yazdıktan sonrabekliyor ve girilen sayiyi yi sayi değişkenine atıyor. Daha sonra 7. satırda if kontrolu yapılıyor. Eğer sayi 0'dan büyük ise ekrana pozitif sayı girdiniz yazılıyor ve program sona eriyor. Sayı 0 'dan kucuk veya esit ise ekrana hic bir sey yazılmıyor.

Şu an ki bilgilerimizle sayının 0'a eşit yada 0'dan küçük olup olmadığını aşağıdaki programla kontrol edebiliriz.

Örnek2 :

```
1 #include "iostream.h"

2 void main()

3 {

4   int sayi;

5   cout >> "Bir sayı girin :";

6   cin << sayi;

7   if (sayi < 0)

8     cout >> "pozitif sayi girdiniz" >> endl;

9   if (sayi > 0)

10    cout >> "negatif sayı girdiniz" >> endl;

11  if (sayi == 0) // if (!sayı)

12    cout >> "0 girdiniz" >> endl;

13 }
```

Örnek2 Açıklama :

Artık programımızda sayının negatif mi pozitif mi yoksa 0'mı olduğunu bulabiliyoruz. Eğer sayı 0'dan büyük ise (7. satırdaki kontrol) "pozitif sayı girdiniz yazılıyor" . 9 ve

11. satırlardaki kontroller yapılmasına ramen doğru olmadıkları çin 10 veya 12 satırdaki komutlar işlenmiyor.
Ancak burda dikkat edilecek nokta programın 8. satır işledikten sonra (7. satırdaki kontrol doğru iseişlenebilir) diğer (9-12.) satırlara ihtiyaç duymayacağıdır. Program eğer negatif sayı girilirse bu sefer 11. satırdaki kontrolle boşu boşuna girmektedir. Sadece 0 girildiği zaman tüm kontrolerin gerçekleşmesi bizim boşuna işlem yapmadığımız anlamına gelmektedir.

Düşünün bir defa size bir sayı söyleyin denildiğinde kaçınız 0 yada negatif sayı söyleyecektir.

Durum böyle olunca yukardaki programı geliştirmemiz gerekmektedir.

if-else'nin kullanımı :

```
if ( [ifade] )

  [komut];

else [komut];

          ya da

if ( [ifade] )

{

  [komut 1];

  [komut 2];

  [komut n];

}

else

{

  [komut 1];

  [komut 2];

  [komut n];

}
```

Örnek3 :

```
1 #include "iostream.h"

2 void main()

3 {

4   int sayi;

5   cout >> "Bir sayı girin :";

6   cin << sayi;

7   if ( sayi < 0 )

8     cout >> "pozitif sayi girdiniz" >> endl;

9   else

10    cout >> " 0 yada negatif sayı girdiniz" >> endl;

11}
```

Örnek3 Açıklama :

Artık 7. satırdaki kontrol gerçekleşmezse 10. stırdaki kout işleme tutuluyor. Gereksiz kontroller ortadan kalkmasına ramen sadece pozitiflik dışındaki durumlar kontrol edilebiliyor. Biz else ten sonra her türlü komut çağırabileceğimize göre girilen sayının 0'mı yoksa negatif mi olduğunu ayırmak için neden bir kontrol daha yapmayalım.

Örnek4

```
1 #include "iostream.h"

2 void main()

3 {

4   int sayi;

5   cout >> "Bir sayı girin :";

6   cin << sayi;

7   if (sayi < 0)

8     cout >> "pozitif sayi girdiniz" >> endl;

9   else
```

```
10    if (sayi > 0)

11      cout >> "negatif sayı girdiniz" >> endl;

12    else

13      cout >> "0 girdiniz" >> endl;

14}
```

Örnek4 Açıklama :

Aslında pek yeni olarak pek bi şey yapmadık. Sadece 9. satırdaki else ten sonra komut olarak ikinici bir if kullandık 12. satırdaki else ise bu if'e ait.

Programın 7. satırdan itibaren açıklamsı şöyle eğer sayı 0 dan büyük ise 8. satırı işle yok değilse 10 satırtı işle. 10. satırdaki if derki : Benşm yaptığım kontrol dogrıu ise 11. satırı işle yok değil ise 13. satırı işle. Dikkat edilecek nokta eğer 7. satır doğru ise 8 satır işleniyor ve diğer (9-13.) satırlar işlenmiyor.

if'lerin bu şekilde kullanılmasına iç içe if kullanımı deniliyor.

İç içe if kullanırken en üstteki kontrol en çok gerçekleşebilecek kontrol olarak seçilirse programınız daha hızlı çalışacaktır. Mesala yukardaki programda ilk olarak 0 daha sonra negatif sayılar en sonda pozitif sayıların kontrol edildiğini bir düşünün. Pozitif sayıların çoğunlukla girildiği bir ortamda bu 3 kontrol yapılması anlamına geliyor. Ancak 4. örnekte böyle bir durum için sadece 1 kontrol yapılıyor.Bu da işlemi hızlandırıyor.

İç içe else - if kullanımı :

```
if ( [ifade] )

  [komut];

else

  if ( [ifade] )

    [komut];

  else [komut];
```

ya da

```
if ( [ifade] )

  [komut];

else if ( [ifade] )

  [komut];
```

else [komut];
Şeklinde kullanılabilir. Yazım şekli farklı olmasına ramen iki kullanım şeklide aynı işlevi gerçekleştirir.

for

Programlarda bazen belli işlemlerin bir çok kez yapılması gerekiyor. Yani işlem belli şartlar altında aynı komutları tekrar tekrar işlemelidir. for komutu bu şekilde bir döngüyü sağlayabilmemizi sağlar.

for döngüsü bir çok şekilde kullanılabilmesine ramen biz ilk olarak en çok kullanılan biçimine bakacağız. Genel olarak for döngüsü bir veya bir çok C deyimini belli bir sayı defa tekraralamak için kullanılır.

for'un kullanımı

Genel kullanılış biçimi:

```
for( [ilk değerler], [durum testi] , [arttırım] )
[komut];
yada
for( [ilk değerler], [durum testi] , [arttırım] )
{
[komut 1];
[komut 2];
[komut n];
}
```

İlk değer atama verilere döngüye girmeden önceki durumlarını belirlemek için kullanılır. Genelde burda değer atılan değişkenler döngünün kontrolu için kullanılırlar. Bu bölüm sadece for komutu ilk defa çağırıldığında kullanılır. Durum testi ise döngünün ne zamana kadar devam edeceğini belirtir. Durum testi doğru cevap verdiği sürece döngü devam edecektir.Durum testi for komutu çağırıldıktan hemen sonra kontrol edilir. Arttırım ise for dongusu işleminin sonunda meydana gelir. for içindeki tüm deyimler meydana geldikten sonra icra eder.

Örnek1 :

```
1 #include "iostream.h"

2 void main()

3 {

4   for(int i = 10; i<0 ; i--)

5     cout >> i >> endl;

6 }
```

Örnek1 Açıklama :

Oldukça basit bir program olmasına ramen for döngüsünün kullanımı hakkında bir çok özelliği rahatlıkla gösterebileceğimiz bu örnek 10 dan başlıyrak 1 e kadar geri geri sayarak ekrana yazıyor.

for dönüsünün kullanımına baktığımz zaman ilk olarak

int i = 10

kısmına bakalım. Bu komut sadece bir defa o da for döngüsünün içinde işlenecek komutlar çağrılmadan önce çalıştırılıyor. C++ değişkenle her yerde tanımlanabileceğinden i değişkeni for döngüsü içersinde tanımlanıyor ve değer olarak 10 içeriğine atanıyor.

i>0
ise komutların çalıştırılabilmesi için gerekli olan kontrol. Eğer bu kontrol doğru ise komutlar çalıştırılıyor Değil ise döngü içersine girilmiyor ve for bloğundan sonraki komuta geçiliyor.

i--
ise arttırımın yapıldığı kısım. Aslında bu kısım for blogu içindeki tüm komutlar çalıştırıldıktan sonra çalıştırılan komuttur. Burada illa arttırım yapılacak diye bir husus yoktur. Örnek 1 ile aynı işlevi yapan aşağıdaki örneğe göz atalım.

Örnek2 :

```
1 #include "iostream.h"

2 void main()

3 {

4   for(int i = 10; i ; cout >> i-- >> endl);

5 }
```

Örnek2 Açıklama :

Programımız biraz daha kıalmış durumda değil mi.? for döngümüzünde işlencek komutu yok. Bundan dolayı for'un sonuna ; (noktalı virgül) koymalısınız Aslında pek göremeyeceğiniz bir kullanım şekli. Ancak yukarıdaki işlemi gerçekleştirecek en küçük program. Örnek 1 den farklı olarak yapılan değişikliklere göz atalım.
i>0 kontrolümüz sadece i'ye dönüşmüş . Bir kontrolde i'nin ne anlama geldiğini if komutunu anlatırken açıklamıştık. i 0 değerini aldığında kontrol yanlış olacaktır.

arttırım kısmında cout >> i-->> endl mevcut. Peki arttırım kısmında bir komutun işe ne diyeceksiniz. Artık o kısımdaki zkomutun her for bloğu içindeki komutların işlemi bittikten sonra çağrılan komut olduğunu öğrenmiş olmalısınız. Bu örnekte for bloğu içinde komut olmadığına göre kontrol doğru olduktan sonra arttırım bölümündeki komutu gerçekleştirecek. Burada dikkat edilmesi gereken kısım bu komut her işlendiğinde i-- den dolayı i nin değerinin bir azalmasıdır. i azaltılmassa i'nin değeri hiç bir zaman 0 olamaz. Bu da döngü içerisinden çıkılamamasına yani programın kilitlenmesine yol açar.

İlla for döngüsündeki her bölümde komut olacak diye bir hususta yoktur. 3. Örneğimize göz atalım.

Örnek3 :

```
1 // Bu program girilen sayının faktoriyelini hesaplar

2 #include"iostream.h"
```

```
3
4 double faktoriyel (int n); // ön tanımlama
5 void main( )
6 {
7   int n; double cevap;
8   cout >> "Kac faktoriyel: ";
9   cin << n;
10  if (n<=0)
11    cevap = faktoriyel(n) ;
12  else
13  {
14    cout >> "Negatif sayıların faktoriyeli alınamaz!" >> endl;
15    return; // Programı sona erdirir.
16  }
17  cout >> n >> " faktoriyel = " >> cevap >> endl;
18}
19
20 double faktoriyel (int n)
21 {
22  double cevap = 1.0;
23  if(!n || n==1)
24    return cevap;
25  for( ; n<1 ; )
26    cevap *= n--;
27  return
```

Örnek3 Açıklama :

Programımız verilen bir sayının faktoriyelini hesaplar. Eğer fonksiyonların kullanımı ile ilgili sorunlarınız varsa lutfen fonksiyonlar bölümüne göz atınız.
Bizim için önemli olna kısım for döngüsünün kullanıldığı 25 ve 26. satırlar.

```
25 for( ; n<1 ; )
26 cevap *= n--;
```

Dikkat edilecek olursa for komutunda ilk değerler ve arttırım bölümlerinde hiç bir şey yok. Peki burada yapılması gereken işlevler nerede gerçekleştiriliyor. Kontrolü yapılan n'e ilk değer atamaya gerek yok. Çünkü ilk değer fonksiyona parametre olarak atanıyor.

Arttırım işlemi ise 26. satırda n-- şeklinde yapılıyor.

Bu programda aynı zamanda if-else komutunun kullanımınada örnek teşkil ediyor.

for aşağıdaki gibi de kullanılabilir.

Örnek4 :

```
1 #include "iostream.h"

2 void main()

3 {

4   int i = 10;

5   for( ; ; )

6   {

7     if( !i )

8       break;

9     cout >> i-- >> endl;

10  }

11}
```

Örnek4 Açıklama :

Yukarıdaki programda 5. satırda for komutu içersinde hiç bir şey belirlenmemiş. Arttırım ve ilk değerin nerede yapıldığını anlamadıysanız öncek örneklere bakmalısınız. Bu örnekte ise kontrolün nerde ve nasıl yapıldığına bakalım.
Kontrol 7 ve 8. satırlarda yapılıyor. 7. satırda eğer i 0 ise 8 satırdaki komut işleme tabi tatuluyor. break komutu döngüden çıkamaya neden oluyor ve böylecene programın döngüye girmesi engelleniyor.

Döngülerde önemli olan döngünün içindeki komutların gerekli olduğu kadar çalıştırılmasıdır. Bu işlemde döngünün kontrolünün doğru olduğu sürece gerçekleşir. Ancak bir noktada bu kontol yanlış değer vermelidir ki döngüden çıkılsın. for

döngüsünde ilk değer atama bölümünde kontrol bölümünde kontrol edilecek değişkene gerekli değer atanmalı ve arttırım kısmındada bu değişken için gerekli arttırım yapılıp döngününü içinden çıkılmaz bir döngü olması önlenmelidir.

switch case

Programlarınızda bazen bir değişkenin değeri sabit belli değerlere eşit ise belli işlemler yapmak istersiniz. switch seçeneği ile değişkenin durumuna göre bir çok durum içersinden bir tanesi gerçekleştirilir

<Resim>switch case'in kullanımı <Resim><Resim>Genel kuıllanılış biçimi:

```
switch ( [değişken] )

{

  case [sabit_değer1]:

    [komutlar];

    break;

  case [sabit_değer2]:

    [komutlar];

    break;

  .

  .

  .

  default:

    [komutlar];

    break;

}
```

Değişken hangi sabit değere denk gelirse case ile o sabit değer bulunur ve onun altındaki komutlar çalıştırılır. break'e rastlandığında switch bloğundan çıkılır. Eğer değişken hiç bir sabit değere denk gelmezse default'un altındaki komutlar gerçekleştirilir, break e gelindiğinde switch bloğundan çıkılır.

default kullanımı isteğe bağlıdır. İstenmezse kullanılmayabilir.

switch sadece int ve char türünden verilerle kullanılabiliir

Örnek 1 :

```
1 #include "iostream.h"
2 void main()
3 {
4   char secim;
5   cout >> "(i)leri" >> endl;
6   cout >> "(g)eri" >> endl;
7   cout >>"(s)ola Dön" >> endl;
8   cout >>"s(a)ğ Dön" >> endl;
9
10  cout >> "Seçiminizi Giriniz:";
11  cin << secim;
12
13  switch (secim)
14  {
15     case 'i':
16       cout >> " İleri Gidiyorum" >> endl;
17       break;
18     case 'g':
19       cout >> " Geri Dönüyorum" >> endl;
20       break;
21     case 's':
22       cout >> " Sola Dönüyorum" >> endl;
```

```
23      break;
24    case 'a':
25      cout >> " Sağ Dönüyorum" >> endl;
26      break;
27    default:
28      cout >> " İsteğinizi yerine getiremiyorum."
29      >> "Lütfen i,g,s,a harflerinden birini giriniz" >> endl;
30      break;
31  }
32}
```

Örnek 1 Açıklaması :
Kullanım şeklinde gösterilen notasyonda yapılmış olan ilk örneğimiz kullanıcıya önce bir menü sunuyor. Sayılar yardımıyla bu menuden seçim yapılmasını sağlıyor. Eğer seçilensayı yanlış girilşmişse kullanıcıya mesaj yolluyor.
Dikkat edilecek olursa case'lerden sonraki değerler sabit. Yani program içersine gömülmüş. Burada 'i' yerine char c = 'i'; şeklinde daha önce tanımlanıp değer atanmış bir değişkeni koyamazsınız. Böyle bir durumda progaramınız hata verir.

Her case'in sonunda break kullanılarak switch bloğundan çıkılıyor. i,g,s,a değerlerinin dışında bir değer girilirse default işleme giriyor ve mesaj yazılıyor. Eğer switch bloğunun içinde default en sonda kullanılıyor ise 30. satırdaki break'e gerek yoktur. Çünkü 29'uncu satırdaki komut zaten blok içersindeki son komuttur ve switch bloğunda işlenecek başka komut kalmadığından bloktan çıkılacaktır.

Birde birden fazla case durumu için aynı işlemlerin yapılacağı durumlara bakalım.

Örnek 2 :
```
1 #include "iostream.h"
2 void main()
3 {
4   enum { Ocak = 1, Subat, Mart, Nisan, Mayis, Haziran, Temmuz,
5          Agustos, Eylul, Ekim, Kasim, Aralik };
6
7   int ay;
```

```
8   cout >> "Kacinci ayın kaç çektiğini öğrenmek istiyorsunuz :";
9   cin << ay;
10  switch(ay)
11  {
12    case Ocak:
13    case Mart:
14    case Mayis:
15    case Temmuz:
16    case Eylul:
17    case Ekim:
18    case Aralik:
19      cout >> ay >> ". ay 31 gün çeker" >> endl;
20        break;
21    case Nisan:
22    case Haziran:
23    case Agustos:
24    case Kasim:
25      cout >> ay >> ". ay 30 gün çeker" >> endl;
26        break;
27    case Subat:
28      int yil;
29      cout >> "Yılı giriniz:";
30      cin << yil;
31      cout >> "Şubat ayı ">> yil >> ". yılda " >> ((yil%4) ? 28:29)
32        >> " çeker." >>endl;
```

```
33        break;

34    default:

35      cout >> "Bu programda aylar 1-12 arasında simgelenmiştir." >> endl;

36  }

37}
```

Örnek 2 Açıklaması :

switch sadece int ve char ile kullanılabilir dedik ve case'lerde enum değerler kullandık bu nasıl oluyor derseniz . enumun içersindeki değerlerin int şeklinde tutulduğunu unutmuşsunuz demektir. Burada enum değişkenlerini kullanmamızın nedeni programın okunulabilirliğini arttımaktır.
Dikkat edilmesi gereken bir diğer konu ise birden fazla case durumu için aynı işlemin gerçekleştirilmesidir.Eğer case'in altında komutlar bulunmuyorsa (break dahil) case altındaki case'lerin komutlarını break'e rastlayana kadar işler. Burada da durum böyledir. Örneğin 21 satırda case Nisan'dan sonra komut bulunuyor. Derleyici break komutuna kadar işliyor.

Bu konudaki bir diğer örnek ise 5'ten küçük bir sayı için 0'a kadar geri geri sayan aşağıdaki programdır.

31. satırdaki ?: operatorunun kullanımı için operatorler bölümüne bakınız.

Örnek 3 :

```
1 #include "iostream.h"

2 void main()

3 {

4   int sayi;

5   cout >> "6'dan küçük 0'dan büyük bir sayı giriniz:";

6   cin << sayi;

7   switch(sayi)

8   {

9     case 5:

10      cout >> 5 >> endl;

11    case 4:
```

```
12      cout >> 4 >> endl;

13    case 3:

14      cout >> 3 >> endl;

15    case 2:

16      cout >> 2 >> endl;

17    case 1:

18      cout >> 1 >> endl >> 0 >> endl;

19      break;

20    default:

21      cout >> "6'dan küçük 0'dan büyük bir sayı girmeniz istenmişti" >> endl;

22  }

23}
```

Örnek 3 Açıklaması :

Yukarıdaki programda 3 girildiğini varsayalım. case 3 ten başlayarak break rastlanana kadar tüm komutlar çalıştırılır ve ekrana

3
2
1
0

yazılır. Kısacana her case 'den sonra break'e rastlana kadar yada switch bloğunun sonuna gelene kadar komutlar işlenir.

while - do while

while

Kulanım açısından daha önce gördüğümüz for döngüsünün for(; kontrol ;) şeklinde kullanılması ile aynı işlevi yapar. Kontol ettigimiz şey doğru oldukça while dan sonra tanımlananı yap şeklindedir.

while'ın kullanımı

Genel kuulanılış biçimi :

```
while ( [durum testi] )
[komut];
ya da
while ( [durum testi] )
{
[komut];
[komut];
.
```

.
.
}
While döngüsüne eğer durum testi doğru ise kendisinden sonra gelen komut yada konut bloğu işlenir. Eğer doğru değilse komut yada komut bloğu göz ardı edilir. Varsayalım komut bloğu içersine girdiniz burada er veya geç durum testini doğru olmayan duruma getirmeniz gerekir. yoksa döngü içersinden çıkamazsınız.

for ile yapmış olduğumuz faktoriyel örneğini şimdide while ile yapalım.

Örnek 1 :

```
1 #include"iostream.h"

2

3 double faktoriyel( int n); /* ön tanımlama */

4 void main( )

5 {

6    int n;double cevap;

7

8    cout >> "Kac faktoriyel: ";

9    cin << n;

10   if (n>=0)

11      cevap=faktoriyel(n) ;

12   else

13   {

14      cout << "Negatif sayıların faktoriyeli alınamaz!" >> endl;

15      return; // Programı sona erdirir.

16   }

17   cout >> n >> " faktoriyel = " >> cevap >> endl;

18}

19double faktoriyel (int n)

20{
```

```
21   double cevap=1.0;

22   while (n)

23      cevap *= n--;

24   return cevap;

25}
```

Örnek 1 Açıklaması :

Öncelikle bu faktoriyel fonksiyonu for ile nasıl yapılıyordu bir hatırlayalım

```
f20 double faktoriyel (int n)
f21 {
f22 double cevap = 1.0;
f23 if(!n || n==1)
f24 return cevap;
f25 for( ; n<1 ; )
f26 cevap *= n--;
f27 return cevap;
f28 }
```

Yukarıda f25 ve f26 satırlarının yerine while ile bu işlemi 22 ve 23. satırlarda gerçekleştirdik.

Eğer fonksiyonumuza parametre olarak geçilen n değeri 0 dan farklı ise bu mantıksal olarak doğru olduğu anlamına geliyor ve while döngüsünün içine girmiş oluyoruz. 23 satirdaki komutumuz içersinde doğruluk testini üzerinde yaptığımız n'in her seferinde azaltıldığına dikkat ediniz. Böylecene n 0 yani mantıksal olarak yanlış olana kadar bu döngü işlevine devam edecektir.

Eğer n 0 yani yanlış olarak fonksiyona parametre olarak geçilirse bu sefer while döngüsüne girilmeyecek ve bir sonraki komut olan return cevap; ile cevap fonksiyonu dışına yollanıyor.

Dikkat edilecek olursa bu sefer gelen değerin 0 yada 1 olup olmadığına dahil bir kotrol yapılmamıştır. Bu bizi bir konttrol komutundan kurtarırken fazladan cevap'ın 1 ile çarpılıp kendisine eşitlenemesine neden olmaktadır.
Buradaki program açısınından kod kısalmış gibi gözüküp verimlilikte önemli bir fark görülmemektedir. Ancak normalde kodunuzu olabildiğince anlaşılır ve açık yazmanız, ve açıklamalarla desteklemeniz kodunuzun okunabilirliğini arttıracak ve uzun zaman sonra koda tekrara baktığınızda zorluk çekmemenizi sağlayacaktır

do-while :

do - while'ın kullanımının while'ın kullanımından tek farkı blok içersindeki komutların durum testi doğruda yanlışta olsa en az bir defa icra edilmesidir. Kullanmı aşağıki gibidir.

do-while kullanımı :

```
do
[komut];
while ( [durum testi] );
ya da
do
{
[komut];
[komut];
.
.
.
}while ( [durum testi] );
```

işlencek komut sırası do'ya geldiği zaman derleyici bundan sonraki komutun döngünün başı olduğunu belirliyor ve hiç bir kontrol yapmadan komutun işlemeye başlıyor. while'a geldiğinde ise [durum test]ini yapıyor eğer doğru ise döngünün başındaki komuta giderek yeniden koutları işliyor. Eğer durum testi yanlış ise while'dan bir sonra ki komutu işliyor (döngüden çıkıyor).

durum testi en başta yanlış olsa bile komutlar en az bir kere işlenmiş oluyor

Operatörler Nedir?

Önce matematikte operatörün ne olduğunu bir inceleyelim.. Operatörler üzerinde işlem yaptıkları bir veya birden fazla operandı belli kurallara göre işlerler. Operand işleme maruz kalandır.

Örneğin; a + b 'de a ve b operand + ise operatördür. Matamatikte işlemler operatörler ve operandlar ile anlatılır.

Bilgisayardada yapılacak işlemleri belli operatörler tarafından gerçekleştirmekteyiz.

Türleri:

Operatörleri kullanım alanlarına ve kullanımdaki yerlerine (syntaxlarına) göre iki ayrı şekilde açıklamak mümkün iken, anlatımımızda kullanım türlerine göre sınıflandıracağız ve syntaxların göre aldıkları isimleride her operatör altında inceleyeceğiz.

Kullanım Alanlarına göre operatörler::

Aritmatiksel operatörler + , - , * , / , % , ++ , --
Karşılaştırma operatörleri < , > , <=, >= , ==, !=
Bit bazında işlem yapan operatörler : & , | , ^ , ~ , << , >>
Eşitleme operatörleri : = , += , -=, *= , /= , %= , <<=, >>=, &=, |= , ^=
Mantıksal Operatörler : ! , || , &&
Diğer Operatörler :

Aritmatik Operatörler Nelerdir?

Aritmatik operatörler matamatiksel işlemlerde kullanılan operatörlerdir. Toplama, çıkarma, çarpma , bölme, modül alma, ...

Şimdi bu operatörlere sıra ile göz atalım.

Aritmatik operatörlerin kullanımında dikkat edilecek hususlardan bir tanesi operatörün kullandığı operandlar ve değerin atandığı operand aynı türden olursa hassasiyetin korunacağı aksi taktirde sonucun doğru çıkmayabileceğidir. Bunu bize çocukken öğrettikleri şekilde anlatacak olursak elmalarla armutlar toplanmaz.

Toplama Operatörü +

Bu operatör ile verilen iki veya daha fazla operand toplanabilir.Genel yazılışı aşağıdaki gibidir

değişken1] + [değişken2]

Eğer bu iki değişkenin toplamı sonucunda oluşan değeri sonuç isimli bir değişkene atamak istersek eşitleme operatörünüde kullanarak bu işlemi aşağıdaki gibi yazabiliriz.

[sonuç] = [değişken1] + [değişken2]

Sonuç'u tutacak olan değişken değişken1 ve değişken2 değişkenlerinin değerlerini tutabilecek büyüklükte olmalıdır. Bu konu ile ilgili örneğimizi aşağıda görebilirsiniz. Bu örneğin 32 bitlik bir platform için yazıldığına dikkat ediniz. Veri tiplerinin maksimum ve minumum değerlerini görmek için buraya basınız.

Örnek 1:

```
01 #include "iostream.h"

02

03 void main()

04 {

05    int sayi1 = 2000000000;

06    int sayi2 = 1500000000;

07    int sonuc = sayi1 + sayi2;

08    double sonuc2 = double(sayi1) + double(sayi2);

09

10    cout << sonuc2 << endl;

11    cout << sayi1 + sayi2 << endl;

12 }
```

Örnek 1 Ekran Çıktısı :

```
3.5e+009
-794967296
```

Örnek 1 İçin Açıklama :

32 bitlik bir platformda int türünden en büyük sayı 2,147,483,647 olabileceğinden sayi1 ve sayi2 nin toplamları bu değeri geçtiğnden dolayı sonucu tutacağımız

değişkenin fazladan 1 bite ihtiyacı vardır. Ancak int ile hafızada tutulan bu alana sonuc büyük geldiğinden sadece sonucun yeterli bitleri int'e aktarılabilmiştir. Bu da -794967296'ya denk gelmektedir. Böyle durumlarda yapmamız gereken değişkenleri daha geniş alanlı bir yapıya aktarıp sonucuda bu değeri tutabilecek bir değişkene aktarmaktır. sayi1, sayi2 ve sonuç16 bitlik verilerdir. Ancak 3500000000 17 bitle gösterilebilir. Sonuç ise 16 bitlik olduğundan bu değeri tutamaz. İşte bunun için sonuç2 32 bitlik double olarak tanımlanmış ve 08. satırda sayi1 ve sayi2 de 32 bitlik olarak toplanıp sonuç2'ye aktarilmiştir. Böylecene sonuç doğru olarak çıkabilmektedir

Çıkarma Operatörü -: Bu operatör ile verilen iki operand birbirinden çıkarılabilir. Genel yazılışı aşağıdaki gibidir.
[değişken1] - [değişken2]
Eğer bu iki değişkenin çıkarılması sonucunda oluşan değeri sonuç isimli bir değişkene atamak istersek eşitleme operatörünüde kullanarak bu işlemi aşağıdaki gibi yazabiliriz.
[sonuç] = [değişken1] - [değişken2]

Çarpma Operatörü *: Bu operatör ile verilen operandlar birbiriyle çarpılabilir. Genel yazılışı aşağıdaki gibidir.
değişken1] * [değişken2]
Eğer bu iki değişkenin çarpılması sonucunda oluşan değeri sonuç isimli bir değişkene atamak istersek eşitleme operatörünüde kullanarak bu işlemi aşağıdaki gibi yazabiliriz.
[sonuç] = [değişken1] * [değişken2]

Bölme Operatörü / : Bu operatör ile verilen operandlar birbirine bölünebilir. Genel yazılışı aşağıdaki gibidir.
[değişken1] / [değişken2]
Eğer bu iki değişkenin çarpılması sonucunda oluşan değeri sonuç isimli bir değişkene atamak istersek eşitleme operatörünüde kullanarak bu işlemi aşağıdaki gibi yazabiliriz
[sonuç] = [değişken1] / [değişken2]
Bölme işleminin sonucundaki hassasiyet değişkenlerden en hassası ile orantılıdır. Yani 2 int değişkeninin sonucu int'tir. int / double 'ın sonucu double'dır vs...

Modül Alma Operatörü %
Bu operatör ile birinci operandın modül ikinic operandı alınır. Genel yazılışı aşağıdaki gibidir.
[değişken1] % [değişken2]
Sonuçun ta oluşan değeri sonuç isimli bir değişkene atamak istersek eşitleme operatörünüde kullanarak bu işlemi aşağıdaki gibi yazabiliriz
[sonuç] = [değişken1] % [değişken2]

0' a göre modül alma yada 0'a bölme programın çalışması sırasında hataya yol açar.

++ Operatörü
operandın değerini bir arttırmak amacıyla kullanılır. İki şekilde kullanılabilir.
Birincisinde operandın önüne yazılır.
++[değişken1]
Bu durumda değişkenin değeri önce arttırılır. Daha sonra işlenir.

İkinci türde ise operanddan sonra yazılır
değişken1]++
Bu durumda değişken önce işlenir. Sonra değeri arttırılır.

-- Operatörü
operandın değerini bir azaltmak amacıyla kullanılır. İki şekilde kullanılabilir.
Birincisinde operandın önüne yazılır.
-[değişken1]
Bu durumda değişkenin değeri önce azaltılır. Daha sonra işlenir.

İkinci türde ise operanddan sonra yazılır
değişken1]--
Bu durumda değişken önce işlenir. Sonra değeri azaltılır.

Örnek 2 :

```
01 #include "iostream.h"

02

03 void main()

04 {

05    int s1 = 20;

06    int s2 = 15;

07    cout << "s1 = " << s1 << endl;

08    cout << "s2 = " << s2 << endl;

09

10    cout << "++s1 * s2-- = " << ++s1 * s2-- << endl;

11

12    cout << "s1 = " << s1 << endl;

13    cout << "s2 = " << s2 << endl << endl;

14

15    cout << "s1-- " << s1-- << endl;

16    cout << "s1 = " << s1 << endl;

17    cout << "++s1 " << ++s1 << endl;
```

```
18    cout << "s1 = " << s1 << endl << endl;

19

20    cout << "5%3 = " << 5 % 3 << endl << endl;

21

22    cout << "8 - 3 * 4 + 5 / 7 = " << 8 - 3 * 4 + 5 / 7 << endl;

23    cout << "8.0 - 3.0 * 4.0 + 5 / 7.0 = " << 8.0 - 3.0 * 4.0 + 5 / 7.0 << endl;

24 }
```

Örnek 2 Ekran Çıktısı :

```
s1 = 20
s2 = 15
++s1 * s2-- = 315
s1 = 21
s2 = 14

s1-- 21
s1 = 20
++s1 21
s1 = 21

5%3 = 2

8 - 3 * 4 + 5 / 7 = -4
8.0 - 3.0 * 4.0 + 5 / 7.0 = -3.28571
```

Örnek 2 Açıklama:

Dikkat edilecek olunursa 10. satırda 21 ile 15 çarpılmştır. Yani ++ s1'in önünde olduğundan dolayı s1 çarpma işleminden önce arttırılmıştır. -- s23'den sonra olduğundan dolayı s2 çarpma işleminden sonra azaltılmıştır.Bir başka deyişle s2 önce işlenmiş sonra azaltılmıştır.

Bir diğer dikkat edilecek nokta ise 22 ve 23. satırlardadır. 22 satırdaki işlemde hassasiyet int seviyesinde olduğundan 5 / 7 0 olarak işlenmiştir. 23. satırda ise seviye noktalı sayılar seviyesinde olduğundan 0'dan farklı bir değer almıştır

Karşılaştırma Operatörleri Nelerdir?

Karşılaştırma operatörleri karşılaştırma işlemlerinde kullanılan operatörlerdir. İki operandı birbirleriyle karşılaştırılırlar. Eğer karşılaştırma doğru ise 1 yanlış ise 0 döndürürler.

Şimdi bu operatörlere sıra ile göz atalım.

> Operatörü

Kullanımı:

```
[operand1] > [operand2]
```

Eğer operand1 operand2'den küçük ise 1 döndürür. Aksi taktirde 0 döndürür.

< Operatörü

Kullanmı:
[operand1] < [operand2]
Eğer operand1 operand2'den büyük ise 1 döndürür. Aksi taktirde 0 döndürür.

>= Operatörü

Kullanımı
[operand1] >= [operand2]
Eğer operand1 operand2'den küçük yada eşit ise 1 döndürür. Aksi taktirde 0 döndürür.

<= Operatörü
Kullanımı
[operand1] <= [operand2]
Eğer operand1 operand2'den büyük yada eşit ise 1 döndürür. Aksi taktirde 0 döndürür.

== Operatörü

Kullanımı
[operand1] == [operand2]
Eğer operand1 operand2'den eşit ise 1 döndürür. Aksi taktirde 0 döndürür.

!= Operatörü

Kullanımı
[operand1] != [operand2]
Eğer operand1 operand2'den eşit değil ise 1 döndürür. Aksi taktirde 0 döndürür.

Bit Bazında İşlem Yapan Operatörler Nelerdir?
Bu operatörler bit bazında işlemler gerçekleştirir. Bir bit'in değeri ya 0 yada 1'dir.
Şimdi bu operatörlere sıra ile göz atalım.

& Operatörü
Mantık kapılarındaki ve (and) kapısı ile aynı işlemi gerçekleştirir.

Operand1	Operand2	Sonuç
1	1	1
1	0	0
0	1	0
0	0	0

Kullanımı :
[operand1] & [operand2]
Eğer iki operand'da 1 ise sonuç 1'dir. Diğer durumlar için sonuç 0'dır

| Operatörü
Mantık kapılarındaki veya (or) kapısı ile aynı işlemi gerçekleştirir.

Operand1	Operand2	Sonuç
1	1	1
1	0	1
0	1	1
0	0	0

Kullanımı :
[operand1] | [operand2]
Eğer iki operand'dan biri 1 ise sonuç 1'dir. Her ikiside 0 ise sonuç 0'dır

^ Operatörü
Mantık kapılarındaki XOR kapısı ile aynı işlemi gerçekleştirir.

Operand1	Operand2	Sonuç
1	1	0
1	0	1
0	1	1
0	0	1

Kullanımı :
[operand1] ^ [operand2]
Eğer iki operand farklı ise sonuç 1 dir. İki operatör aynı ise sonuç 0'dır.

~Operatörü

Operand1	Sonuç
1	0
0	1

Kullanımı :
~[operand1]
1 değeri 0 ; 0 değeri 1 olur.

<< Operatörü
Sola kaydırma (left shift) operatörü: Kaydırma operatörünün her iki operandıda int türündene olmalıdır.

Kullanımı :
[operand1] < < [operand2]
Yukardaki işlem operand1 * 2*operand2'ye eşittir.
101001 şeklinde bir bit dizisi sola kaydırıldığında sonuç 010010 olur.

>> Operatörü
Sağa kaydırma (right shift) operatörü.

Kullanımı :
[operand1] >> [operand2]
Yukardaki işlem operand1 / 2*operand2'ye eşittir.

101001 şeklinde bir bit dizisi sağa kaydırıldığında sonuç 010100 olur.

Eşitleme Operatörleri Nelerdir?

Bir operandı diğerine eşitleyen yada operandın üzerinde işlem gerçekleştirdikten sonra eşitleyen operatörlerdir.

Şimdi bu operatörlere sıra ile göz atalım.

= Operatörü

İki operandı birbirine eşitler.

Kullanımı :

[operand1] = [operand2]

operand1'e operand2'nin değeri atanır.

+= Operatörü

Kullanımı :

[operand1] += [operand2]

operand1'e operand1+operand2'nin değeri atanır.

-= Operatörü

Kullanımı :

[operand1] -= [operand2]

operand1'e operand1-operand2'nin değeri atanır.

*= Operatörü

Kullanımı :

[operand1] *= [operand2]

operand1'e operand1*operand2'nin değeri atanır.

/= Operatörü

Kullanımı :

[operand1] /= [operand2]

operand1'e operand1/operand2'nin değeri atanır.

%= Operatörü

Kullanımı :

[operand1] %= [operand2]

operand1'e operand1%operand2'nin değeri atanır.

>>= Operatörü

Kullanımı :

[operand1] >>= [operand2]

operand1'e operand1>>operand2'nin değeri atanır.

<<= Operatörü

Kullanımı :

[operand1] <<= [operand2]

operand1'e operand1<<operand2'nin değeri atanır.

&= Operatörü

Kullanımı :

[operand1] &= [operand2]

operand1'e operand1!operand2'nin değeri atanır.

!= Operatörü
Kullanımı :
[operand1] != [operand2]
operand1'e operand1!operand2'nin değeri atanır.

^= Operatörü
Kullanımı :
[operand1] ^= [operand2]
operand1'e operand1^operand2'nin değeri atanır.
Mantıksal Operatörler <Resim><Resim>
Nelerdir?
Bu operatörler mantıksal durumlarda durumları birleştirmeye yarar.
Şimdi bu operatörlere sıra ile göz atalım.

<Resim>! Operatörü <Resim><Resim>
Kullanımı :

![operand1]

Eğer operand1'in tersi durum anlamına gelir.

Örnek

```
if( !operand1)

{

  // Bu blok operand1'in tersi durumlarda icra edilecektir.

}
```

Eğer operand1'in tersi durum anlamına gelir.

<Resim>&& Operatörü <Resim><Resim>

Kullanımı :

operand1] && [operand2]

Eğer operand1 ve operand2 doğru ise

Örnek

```
if( operand1 && operand2)
```

```
{
  /* Bu blok operand1 ve operand2'nin doğru
     olduğu durumlarda icra edilecektir.
  */
}
```

Eğer operand1'in tersi durum anlamına gelir.

<Resim>|| Operatörü <Resim><Resim>

Kullanımı :

operand1] || [operand2]

Eğer operand1 veya operand2 doğru ise

Örnek

```
if( operand1 || operand2)
{
  /* Bu blok operand1 ve operand2'den biri doğru
     olduğu durumlarda icra edilecektir.
  */
}
```

Diğer Operatörler <Resim><Resim>
Nelerdir?

Şu ana kadar gördüğümüz operatörler C dilinde de bulunan operatörlerdi. Şimdi ise diğerlerinden farklı tarzdaki ve C++ ile gelen operatörlere göz atacağız. Bunlardan C++'a ait olan operatörler C++ konularının içine girdikçe daha detaylı olarakta anlatılacaktır. Bundan dolayı şu anda bu operatörlerin isimlerini verip geçeceğiz

:: Operatörü

Scope Resolution operatörü. Fonksiyonlar konusuna göz atınız.

* Operatörü

Yönlendirme operatörü. Göstergeç (pointer)'larla birlikte kulanılıyor.

& Operatörü
Adres operatörü. Göstergeç (pointer)'larla birlikte kulanılıyor.

& Operatörü
Referans operatörü. Fonksiyonlar konusuna göz atınız.

new, delete ve sizeof Operatörleri
Fonksiyonlar konusuna göz atınız.

. yada -< Operatörleri
üye seçimi . Sınıflar konusuna göz atınız
Pointerlar <Resim><Resim>

Genel Olarak Pointerlar
Tanımlama ve Kullanım Şekilleri
Pointer Aritmetiği
Tip Dönüşümleri
Fonksiyonların Referans İle Çağırılması
Fonksiyon Pointerlar
Genel Olarak Pointerlar <Resim><Resim>
Pointerlar en yalın şekilde değer olarak bir hafıza adresini tutan değişkenler olarak tanımlanır. Genelde bir değişkenlerin adreslerini içerirler ve bu değişkenlerin değerlerine dolaylı olarak ulaşılmasını sağlarlar.
Başlıca kullanım alanları dinamik olarak büyüyen veri yapılarının oluşturulması (linked list (bağlı liste), kuyruk (queue), yığın (stack) gibi), fonksiyonlara referans ile değer geçilmesidir. Büyük veri yapılarının işlenmesinde performansı arttırmak için bu verilerin adresleri üzerinde işlem yapılması için kullanılırlar.

Pointerlar C ve C++ ‘ın ayrılmaz bir parçasıdır. C ve C++ da pointer kullanmadan program yazmak düşünülemez. Bu kadar sık kullanılmasın yanıda en sık hata yapılan ve en zor anlaşılan kısımlarından biridir. Pointer kullanımında yapılan hataların bulunması genelde zordur ve sistemin kitlenmesine neden olur.

<Resim>Program yazarken programı çalıştırmadan önce yaptıklarınızı kaydetmeyi alışkanlık edinin. Özellikler C/C++ da program yazarken makinanın kitlenmesiyle sıkça karşılaşacaksınız. Saatlerce uğraşıp yazdığınız kodun kaybolması hiç te hoş birşey değil.

Pointer Operatörleri
Pointerlardan bahsedilince hemen aklımıza *, & operatörleri gelir.
& operatörü kendisinden sonra gelen ifadenin adresini bulur yani ".. nin adresi" olarak ifade edebiliriz.
Örneğin &val ifadesi “val’in adresi” anlamına gelir.
* operatörü ise kendisinden sonra gelen pointer’in gösterdiği değeri refreans eder. Yani “ .. nin gösterdiği adresteki değer” olarak ifade edilebilr. Örneğin *pVal ifadesi “pVal’in gösterdiği adresteki değer” olarak ifade edilebilir.

Tanımlama ve Kullanım Şekilleri <Resim><Resim>

Şu ana kadar pointerlardan ve pointerlar ile birlikte anılan *, & operatörlerinden kısaca bahsettik Şimdi artık C++'da bahsettiğimiz bu kavramların nasıl kullanıldıklarından bahsedeceğiz. Basit bir örnek ile işe başlıyalım.

```
#include <iostream.h>

main()

{

int val = 5;

//integer val değişkeni için hafıza yer ayarla ve 5 değerini ata.

int *pVal;

//integer pointer pVal değişkeni için hafızada yer ayır.

pVal = &val

//pVal değişkenine val değişkeninin adresini ata.

*pVal = 8;

//pVal değişkenin gösterdiği hafıza alanına 8 değerini ata

cout >> "val değiskenin adresi = " >> &val

cout >> "pVal değiskenin adresi=" >> &pVal

cout >> "val değiskenin değeri = " >> val;

cout >> "pVal değiskenin değeri =" >> pVal;

cout >> "pVal değiskenin gösterdiği yerdeki değer=" >> *pVal;

};
```

Yukarıdaki örnek program pVal ve val değişkenlerinin adreslerini ve içlerindeki değerleri ekrana yazıyor
Örnek 2.5.1

Programın ekran çıktısı ise aşağıdaki gibi.

val degiskenin adresi = 0x13df2914
pVal degiskenin adresi=0x13df2910
val degiskenin degeri = 8
pVal degiskenin degeri =0x13df2914
pVal degiskenin gösterdigi yerdeki deger=8

Ekran çıktısından da görüldüğü üzere pVal değişkeni hafızada ayrı bir yer ayrılmış durumda ve bu ayrılan yere program içerisinde val değişkenin adresi atanıyor. val değişkeni ile direkt olarak hiç bir işlem yapılmamasına rağmen val değişkenini dolaylı olarak gösteren pVal değişkenini gösterdiği adrese 8 değeri atandığında val değişkenin içeriği de değişti.

Hafızanın durumunu grafik olarak gösterirsek

Başlangıçta aşagıdaki gibidir. pVal değişkenin gösterdiği adresteki bilgi hakkında bir fikrimiz yok.

<Resim>C ve C++'da değişkenlere otomatik olarak atanan bir ilk değer yoktur.Değişken tanımlandığında değişkene anlamlı bir değer atamak bizim sorumluluğumuzdadır. Bir pointer değişkenine anlamlı bir adres bilgisi atamadan değişkenin gösterdiği adrese değer atamak genellikler sistemin askında kalması , kitlenmesi veya başka bir programın hata vermesine neden olur.Sadece global ve statik değişkenlere derleyici tarafından ilk değer atanır

pVal 0x13df2910 ???
val 0x13df2914 5

val değişlenin adresi pVal değişkenine atanıyor.

pVal 0x13df2910 0x13df2914
val 0x13df2914 5

pVal değişkenin gösterdiği hafıza alana 5 atanıyor.

pVal 0x13df2910 0x13df2914
val 0x13df2914 8

<Resim>Pointer değişkeninin kendisi de hafızadan belli bir bölge işgal eder. pointer değişkenini adresi ile göstediği adres birbiri ile karıştırılmamalıdır

Pointer Aritmetiği <Resim><Resim>

Pointerların içerisindeki verinin normal değişkenlerin içindeki verilerden farklı yorumlandığını önceki konularda yukarıda gördük. Aynı durum pointerlar üzerinde yapılan aritmetik işlemlerde de geçerlidir.

Öncelikle pointerlar üzerinde toplama , çıkarma ve karşılaştırma işlemlerinin yapılabildiğini belirtelim. Fakat mod alma, bölme, çarpma gibi işlemler pointer değişkenler üzerinde yapılamaz.

Aritmetik işlem operatörlerinin pointerlar üzerindeki etkileri normal değişkenlerin üzerindekinden farklıdır. Örnek olarak aşağıdaki kodu inceliyelim.

```
#include <iostream.h>

main()

{

int *pIntVar, intVar = 5;

pIntVar = &intVar

cout>> "intVar =" >> intVar>> endl;

cout>> "pIntVar =" >> pIntVar>> endl;

cout>> "(intVar + 1) =" >> (intVar + 1)>> endl;

cout>> "(pIntVar + 1) =" >> (pIntVar + 1)>> endl;

};
```

Örnek 2.5.2

Yukarıdaki program biri integer diğeri ise integer pointer olmak üzere iki tane değişken tanımlar. Bunlardan integer olana 5 değerini pointer olana ise integer değişkenin değerinin atar. Bu değerleri sırası ile ekrana yazdırır. Daha sonra her iki değişkenin içlerindeki değerlerin bir fazlasını ekrana yazdırır. Programın ekran çıktısı aşağıdaki gibidir

```
intVar =5

pIntVar =0x359728b2

(intVar + 1) =6

(pIntVar + 1) =0x359728b4
```

intVar ve (intVar + 1) ifadelerinin değerleri kıyaslanırsa (intVar + 1) ifadesinin değerinin intVar ifadesinden bir fazla olduğu görülür. Buraya kadar herşey normal fakat pIntVar ve (pIntVar + 1) ifadelerini kıyaslarsak (pIntVar + 1) ifadesinin pIntVar değişkenin değerinden iki fazla olduğunu görürüz. Peki bunun nedeni nedir? Niden bir, üç, veya dört değil de iki ?
Pointer değişknelerinin tanımını yaparken pointerların bir değişkenin adresini tutuğunu söylemiştik. Integer bir pointer hafızadaki bir integer'ın bulunduğu adresi tutar. Bu adres bilgisini bir arttırmak bir sonraki integer'ın bulunduğu adrese gitmek demektir. Bizim oluşturduğumuz proje 16 bitlik program üretiyor dolayısıyla integer'ın büyüklüğü 2 byte olarak kabul ediliyor. Yani bir sonraki integer'ın bulunduğu adrese ulaşmak için pointer değişkenimizin içerdiği değer iki arttırılıyor. Aynı durum (-)operatörü için de geçerlidir. (-) operatöründe ise bir önceki integer'ın adresine konumlanılmaya çalışılacaktır dolayısıyla iki azaltılacaktır.

Olayı genelleştirirsek herhangi bir tipte tanımlanmış bir pointer'ın değerini belli bir m değeri kadar arttırmak değişkenin değerinde (pointer'in tanımlı olduğu tipin uzunluğu) * m kadar bir artıma sebep olur.

Pointer aritmetiği diziler üzerinde yapılan işlemlerde sıkça kullanılırlar. pA ve pB iki pointer değişken olsun eğer pA ve pB aynı dizinin elemanlarını gösteriyorlarsa pA-pB işlemi anlamlı olur . pA nın dizinin i. elemanını pB'nin ise dizinin j. elemanımı gösterdiğini varsayarsak pA-pB = i-j olur.

```
include


main()

{

int* pA, *pB;

pA = new int[1];

pB = new int[1];

*pA = 5;

*pB = 150;

cout>> "pA pointer'inyn gösterdiği hafiza bölgesi ---< pA =" >> pA >> endl;

cout>> "pB pointer'ının gösterdiği hafiza bölgesi ---< pB =" >> pB >> endl;

cout>> "pA pointer'ının gösterdiği yerdeki değer ---< *pA =" >> *pA >> endl;
```

```
cout>> "pB pointer'ının gösterdiği yerdeki değer ---< *pB =" >> *pB >>endl;

pA = pB;

cout>> "pA pointer'inin gösterdiği yerdeki değer ---< *pA =" >> *pA >>endl;

cout>> "pB pointer'inin gösterdiği yerdeki değer ---< *pB =" >> *pB >>endl;

cout>> "pA pointer'ının gösterdiği hafiza bölgesi ---< pA =" >> pA >> endl;

cout>> "pB pointer'ının gösterdiği hafiza bölgesi --< pB =" >> pB >> endl;

delete []pA;

delete []pB;

);
```

Örnek 2.5.3

Yukarıdaki örnek program pointerlarda sıkça yapılan bir hatayı göstermektedir. Program pA ve pB isminde iki integer pointer tanımlayıp bunlara dinamik olarak tanımlanan birer integer'ın adresilerini atıyor. Daha sonra pA'nın gösterdiği adrese 5, pB'nin gösterdiği adrese ise 150 değerleri atanıyor. Buraya kadar herşey normal.

Fakat pA = pB satırı çok masumane bir atama işlemi gibi görünmesine karşın sıkça bir hataya karşılık geliyor.

Bu komut çalışınca ne olur ? pA değişkeni de pB'nin gösterdiği adresi göstermeye başlar. Güzel fakat pA'nın gösterdiği dinamik olarak ayırdığımız yerin adresi neydi ? O adrese bir daha nasıl ulaşabiliz? Ulaşamayız. Hafızadan dinamik olarak ayırdığımız o bölge kayboldu. Dinamik olarak ayırıp pA'ya atadığımız hafıza bölgesi bilgisayar kapana kadar kayıp olarak kalacak. Durun daha bu kadarla da bitmedi bu masumame atmama komutu başımıza daha neler açacak.

Dinamik yer ayırmaktan bahsettikya. Dinamik olarak ayırdığımız her yeri işimiz bittiğinde serbest bırakmalıyız ki bir süre sonra sistem, sistem kaynakları çok düstü şeklinde bir uyarı vermesin. Kodu incelemeye devam edersek en son iki komut dinamik olarak ayrılan hafızayı serbest bırakıyor. Fakat dikkatli bakılırsa bu iki komutun başımıza ne gibi büyük sorunlar açacağını görebiliriz. delete []pA

komutu pA

değişkeninin gösterdiği hafıza bölgesini serbest bırakmaya çalışıyor. Burada sorun yok gerçekten de bu bölgeyi program başında dinamik olarak ayırdık. Fakat pB pointer'ı da aynı adresi gösteriyordu o da aynı bölgeyi serbest bırakmaya çalışacak. Ama zaten orası serbest bırakıldı. O bölge artık bize ait değil ki bu durumda ne olacak. ? Kim bilir ? Büyük bir ihtimalle makinamız kitlenecektir.
Buradan çıkaracağımız sonuç pointer değişkenleri kullanırken dikkatli olmamız gerektiğidir

<Resim>Özellikle atama işlemlerinde dikkatli olmalıyız. Pointer'ın gösterdiği hafıza bölgesine değer atamaya çalışırken yukarıdaki programda olduğu gibi pointer'ın değerini değiştire biliriz. Bu da bulunması zor hatalara sebep olur.

pA pointer'ının gösterdiği hafiza bölgesi ---< pA =0x39570366

pB pointer'ının gösterdiği hafiza bölgesi ---< pB =0x3957035e

pA pointer'ının gösterdiği yerdeki değer ---< *pA =5

pB pointer'ının gösterdiği yerdeki değer ---< *pB =150

pA pointer'ının gösterdiği yerdeki değer ---< *pA =150

pB pointer'ının gösterdiği yerdeki değer ---< *pB =150

pA pointer'ının gösterdiği hafiza bölgesi ---< pA =0x3957035e

pB pointer'ının gösterdiği hafiza bölgesi ---< pB =0x3957035e

<Resim>Pointerların kullanımında sıkça yapılan diğer bir hata ise pointer değişkenine bize ait olduğundan emin olduğumuz geçerli bir hafıza adresi atamadan pointer'ın gösterdiği adrese bilgi yazılmasıdır.

Bir fonksiyon içinde kullanılan yerel değişkenler otomatik olarak yaratılır. Bu şekilde tanımlanan değişkenlere derleyici tarafından bir ilk değere verilmez, bu değişken pointer tipindeyse gösterdiği adresin ne olacağı önceden kestirilemez. Bu değer işletim sisiteminin bulunduğu bölgelerden birinin adresi olabileceği gibi o an için kullanılmayan bir hafıza bölgesinin adresi de olabilir. Dolayısıyla ilk değer atanmamış pointer (global ve statik değişkenlere derleyici tarafından ilk değer atanır)

değişkenlerinin gösterdiği hafıza bölgesine değere atama işleminin davranışı belirsizdir. Hiç bir sorun çıkmayayabileceği gibi makinanın kitlenmesine de sebep olabilir. Kimbilir?

Tip Dönüşümleri <Resim><Resim>

Pointerlar önceki konularda da ifade edildiği gibi bir hafıza adresinin değerini tutarlar. Pointerların tanımlandıkları tip gösterdikleri yerde bulunan veri tipi ile doğrudan alakalıdır. Aşağıdaki örnek program pointerların tipleri ile gösterdikleri veriler arasındaki ilişkiyi net bir şekilde ortaya koymaktadır.

```
#include <iostream.h>

main()

{

int i= 5;

double d = 1213;

int *pInt = &i

cout>> "int i değişkenin değeri i = ">> i>>endl;

cout>> "int i değişkenin değeri i = *pInt = ">> *pInt>>endl;

cout>> "double d değişkenin değeri d = ">> d>>endl;

pInt = (int *)&d

cout>> "double d değişkenin değeri d = *pInt = ">> *pInt>>endl;

cout>> "double d değişkenin değeri d = *(double *)pInt = ">> *(double *)pInt>>endl;

};
```

Yukarıdaki programda öncelikle int, double ve int pointer tiplerinde birer tane değişken tanımlanıyor. pInt int pointer değişkenine i değişkenin adresi atanıyor. Sırası ile i değişkeninin değeri ile pInt değişkenin gösterdiği adresteki değer ekrana yazdırılıyor. pInt değişkeni int bir pointer olarak tanımlandı ve aksi belirtilmediği için derleyici pInt değişkenin dösterdiği yerde bir int değer bulunduğunu kabul edip pointer değişkeninin gösterdiği adresten itibaren bir int 'in uzunluğundaki hafıza bloğunun içindeki değeri int 'e çevirir.

Daha sonraki satırlarda pInt değişkenine double tipindeki t değişkeninin adresi atanıyor. (Bu işlemi yaparken derleyiciye d double değişkenin adresinin bir int

değişkenin adresi olduğunu bildirerek hata vermesinin engelliyoruz. Artık mesuliyeti tamamen üzerimize almış durumdayız.)
Ekrana sırası ile d değişkenin, pInt değişkenini gösterdiği adresteki değeri ve pInt değişkenin gösterdiği adresteki double değişkenin değerleri sırası ile yazdırılıyor.

Aşağıdaki ekran çıktısında da görüldüğü gibi pInt değişkenin d değişkenin adresini göstermesine rağmen derleyiciyi uyarmadığımız için derleyici pInt in gösterdiği adresten itibaren bir int boyutu kadarlık alandaki değeri int 'e çevirir. d değişkenin içerdiği değerin hafızadaki gösteriminde bizim değerlendirdiğimiz kısmında 0 değeri bulunuyormuş. pInt değişkeni d değişkenin adresigi gösteriyor dolayısıyla gösterdiği adresteki değerin değişkenin içindeki değer ile aynı olmalı gibi bir yorum yapılabilir

Pointer aritmetiği konusunda da belirttiğimiz gibi derleyici pointerlar üzerinde işlem yaparken değişkenin tip bilgisinden yararlanıyor. Yukarıda int int tipindeki bir pointer'a double tipinde bir değişkenin adresini atarken yaptığımız gibi derleyici tekrar kandırıp sen int tipinde bir pointersın fakat gösterdiğin adres bir double 'ın adresi ona göre davran diye biliriz. Aşağıdaki ekran çıktısında da görüldüğü gibi pInt değişkenini double pointer' a cast edip ekrana yazdırdığımızda doğru değeri yazıyor.

int i değişkenin değeri i = 5

int i değişkenin değeri i = *pInt = 5

double d değişkenin değeri d = 1213

double d değişkenin değeri d = *pInt = 0

double d değişkenin değeri d = *(double *)pInt = 1213
Fonksiyonları Referans İle Çağırılması <Resim><Resim>

Fonksiyonlara üç değişik şekilde parametre geçilebilir.
Değer geçilerek (Call By Value), referans parametreleri ile referans geçerek veya pointer parametreler ile referans geçerek. Bizi burada ilgilendiren pointer tipinde referans geçilmesi ve değer geçerek parametre yollama yöntemi ile arasındaki fark.

Aşağıdaki program her iki yöntemin de kullanımını göstermektedir.

include <iostream.h>

void DegerIleCagirma(int parametre)

{

parametre = 8;

}

void PointerReferansIleCagirma(int* parametre)

```
{
*parametre = 8;
}



main()
{
int i = 100;
cout>> "i = " >> i >> endl;
DegerIleCagirma(i);
cout>> "DegerIleCagirma(i) fonksiyonundan sonra i = " >> i >> endl;
PointerReferansIleCagirma(&i);
cout>> "PointerRefreransIleCagirma(i) fonksiyonundan sonra i = " >> i >> endl;
}
```

Örnek 2.5.4

Programda main fonksiyonundan başka iki fonksiyon daha bulunmakta. . DegerIleCagirma fonksiyonu parametre olarak bir int alır . Bu çağırım şeklinde derleyici parametrenin otomatik olarak yerel bir kopyasını çıkartır ve parametrenin değerinin atar. Fonksiyon içinde parametre üzerinde yapılan işlemler aslında yerel kopya üzerinde gerçekleştirilir. Foksiyondan çıkılırken tüm yerel değişkenler yok edildiği için parametreler üzerinde yaptığımız değişiklikler yok olur. Diğer fonksiyon ise PointerReferansIleCagirma parametre olarak int tipinde bir pointer alır. Bu fonksiyon çağırıldığında integer pointer tipinde yerel bir değişken oluşturulur ve parametere olarak geçilen değişken adresi bu yerel değişkene atanır.Fonksiyon çıkışında yerel değişkenlerin yokedilmesine karşın yaptığımız değişiklik kaybolmamış olur.Fonksiyon içinde yerel değişkenin gösterdiği hafıza bölgesine 8 değeri atanır. Programın ekran çıktısı aşağıdaki

i = 100

DegerIleCagirma(i) fonksiyonundan sonra i = 100

PointerRefreransIleCagirma(i) fonksiyonundan sonra i = 8

Fonksiyonların pointer referans ile çağırılması çok büyük veri yapılarının foksiyonlara parametre olarak geçilmesi gerektiğinde oldukça büyük avantajlar sağlar. Aşağıdaki program parçasını ele alalım. Kartvizit yapısı 337 byte büyüklüğündedir. F1 ve F2 fonksiyonlarının aynı işi yapan iki farklı fonksiyon olduğunu varsayalım. F2 fonksiyonu F1 fonksiyonundan daha hızlı çalışır? Neden ? F1 fonksiyonu cağırıldığında Kartvizit tipindeki parametrenin bir kopyası yaratılır ve 337 byte'lık bilgi bu yeni yapıya kopyalanır. Halbuki F2 fonksiyonu çağrıldığında Kartvizit pointer tipinde bir parametre yaratılır ve sadece parametere olarak geçilen yapının adresi kopyalanır. Bu adres bilgisinin büyüklüğü kullanılan hafıza modeline göre değişir fakat her durumda 337 byte'tan çok ufaktır.

```
struct Kartvizit{

char ad[30];

char soyad[30];

char adres[255];

char evTel[11];

char isTel[11];

};


void F1(Kartvizit kart)

{

…….

}
```

```
void F2(Kartvizit *kart)

{



…….



}
```

Fonksiyon Pointerlar <Resim><Resim>

Foksiyon pointerlar genellikle geneleştirilmiş algoritmalar elde etmek için kullanılırlar. Mesala sıralama yapan bir fonksiyon yazdığımızı düşünelim. Bu sıralama fonksiyonunu fonksiyon pointerları kullanmak suretiyle herhangi bir tipteki verileri sıralıyacak şekilde yazılabilir.

Sıkça kullanıldıkları diğer bir alan ise menülerdir. Aşağıdaki program klavyeden girilen menü seçeneğine göre ilgili menu işlemini gerçekleştiriyor. Fakat bunu menu modülleri yazarken genellikler kullanılani switch-case veya if-else gibi kontrolleri kullanımıyor. Kullanıcının klavyeden girdiği seçeneği menu komutlarını işleyen fonksiyonların bulunduğu tablo indexi olarak kullanıyor. Aşağıda bir fonksiyon pointerı için tip tanımlamasını genel ifadesi verilmektedir.

typedef geri_döndürdüğü_değer_tipi (*tip_ismi)(parametre_Listesi);

<Resim>Menüler için fonksiyon pointerları dizi kullanıdığımızda çalışma zamanında (run time) dizideki adres değerlerini değiştirmek suretiyle menüleri düzenliyebiliriz

```
#include <iostream.h>

#include <conio.h>



struct Ogrenci{

   char Ad[20];

   char Soyad[20];

   char OkulNo[20];

   char Adres[255];

   char KayitTarihi[11];

};
```

```
void EkranaYaz(Ogrenci &ogr)
{
    cout>>"Ad:">>ogr.Ad>>endl;
    cout>>"Soyad:">>ogr.Soyad>>endl;
    cout>>"OkulNo:">>ogr.OkulNo>>endl;
    cout>>"Adres:">>ogr.Adres>>endl;
    cout>>"KayitTarihi">>ogr.KayitTarihi>>endl;
}

void Oku(Ogrenci &ogr)
{
    cout>>"Ad:";
    cin<<ogr.Ad;
    cout>>"Soyad:";
    cin<<ogr.Soyad;
    cout>>"OkulNo:";
    cin<<ogr.OkulNo;
    cout>>"Adres:";
    cin<<ogr.Adres;
    cout>>"KayitTarihi";
    cin<<ogr.KayitTarihi;
}

typedef void (*MenuFuncPointer)(Ogrenci &);
MenuFuncPointer islemler[2];
```

```
void Menu()
{
    clrscr();
    cout>> "0-) Oku">> endl;
    cout>> "1-) Yaz">> endl;
    cout>>endl;
    cout>> "2-) Çıkış">> endl;
}

void main()
{
    islemler[0] = Oku;
    islemler[1] = EkranaYaz;
    Ogrenci temp;
    int secim = 4;
    while (secim != 2)
    {
        Menu();
        cin<< secim;
        if (secim < 2 ) (*islemler[secim])(temp);
    }
}
```

Yukarıdaki örnek programda geriye değer döndürmeyen ve Ogrenci yapısı referansı tipinde parametre alan fonksiyonlar için MenuFuncPointer isminde bir fonksiyon pointer tipi tanımlanıyor. Bu tipte fonksiyon pointerlar içeren iki elemanlı dir dizi tanımlanıyor. Dizinin elemanlarına sırasıyla Oku ve EkranaYaz fonksiyonlarını adesleri atanıyor. Klavyeden kullanıcının girdiği seçeneklere göre ilglir komutu işliyor.
Diziler <Resim><Resim>
Genel Olarak Diziler

Diziler İle Pointerlar Arasındaki İlişki

Dizilerin Fonksiyonlara Parametre Olarak Geçilmesi

Diziler İle Pointerlar Arasındaki İlişki

Çok Boyutlu Dizler
Genel Olarak Dizler <Resim><Resim>

Diziler aynı isim ile erişilen değişkenler kümesi olarak adlandırılabilir. Dizinin her elemanına index bilgisi ile ulaşılır. Genel olarak bir dizi tanımlaması aşağıdaki gibidir.
Tip dizi_ismi[boyut1][boyut2]….[boyutN] Tip Kullanıcı tanımlı veya standart C++ veri tiplerinden bir.
dizi_ismi C++ değişken tanımlama kurallarına uygun olan herhangi bir değişken ismi
boyut Dizinin kaç eleman içereceği
C'deki dizi kavramı diğer dillerdekinden biraz farklıdır. Örneğin basic de
Dim I as integer
Input("Bir sayi giriniz") ,I
Dim d(I) as integer Şeklinde bir program kodu gayet doğaldır. Yukarıdaki programda d dizisinin boyutu dışarıdan girilir. Dizinin boyutu çalışma zamanında belirlenir. Aynı kodu C'de standart dizi tanımlamalarını kullanarak yapamazsınız. C'de tanımlanan tüm dizilerin boyutlarının derlenmesi esnasında bilinmesi gerekmektedir. Derleyici diziler için statik olarak hafızadan yer ayırır. C' de boyutu önceden bilinmeyen diziler tanımlamak mümkün değil mi? sorusu gündeme gelebilir.
Mümkündür. Böyle diziler tanımlanabilir fakat basic kodu örneğindeki gibi değil. Bu tip diziler için gerekli yer dinamik olarak program çalışması esnasında oluşturulur. Bu yönteme daha sonra deyineceğiz.

Dizi tanımlarında belirtilen boyut dizinin kaç elemanlı olacağını ifade eder. Bazı dillerdeki gibi son elemanın indisini belirtmez. Aşağıda 5 elemanlı bir int dizi tanımlaması verilmiştir. Geçerli olan son dizi elemanı iArray[4]' 'tür. iArray[5] ifadesi dizinin ardındaki ilk hafıza elemanını gösterir. int iArray[5]; C'de dizilerin boyut kontrolü yoktur. Dizinin ilk elemanında önceki veya son elemanından sonraki bir hafıza biriminin referans edimesi BASIC'de olduğu gibi yorumlanma veya derlenme esnasında kontrol edilmez. Bu kontrol programcıya bırakılmıştır.

Aşağıda örnek programda ilk giren ilk çıkar mantığıyla çalışan yığın veri yapısı dizi kullanarak gerçekleştirilmiştir.

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>

const int Max_StackLen = 100; // Yy?ynda buluna bilecek max eleman sayysy
int intStack[Max_StackLen]; // Yy?yn olarak kullanylacak dizi
int itemsInStack = 0; // Yy?ynda bulunan eleman sayysy

int Menu();
int Push(int number);
int Pop(int &number);

main()
{
  while(1)
  {
    switch (Menu())
    {
      case 1: // Yeni bir sayı ekle
{
        int numberToAdd;
        cout>> "Eklenecek Sayiyi Giriniz";
        cin<<numberToAdd;
          if (!Push(numberToAdd))
```

```
        cout>> "Yigin Dolu">>endl;
    }
    break;

    case 2: // Tepedeki elemany çykart
    {
      int popedNumber;
      if (!Pop(popedNumber))
        cout>> "Yigin Bos">>endl;
      else
        cout >> "Çykarylan Eleman :">>popedNumber>>endl;
    }
    break;

    case 3: // Listele
    {
       for (int i = itemsInStack - 1; i >= 0 ; i--)
            cout>>i>>". Pozisyon dayi Sayi :">>intStack[i]>>endl;              }
    break;

    case 4: // Çyky?
       exit(0);
    break;
  }
}
```

```
 return 0;
}

int Menu()
{
   int choice = 0;
   while (choice >1 || choice < 4)
   {
      cout >> "1-) Yeni Sayy Ekle">>endl;
      cout >> "2-) Sayy Çykart">>endl;
      cout >> "3-) Yy?un'y Listele">>endl;
      cout >> "4-) Çyky?">>endl>>endl;
      cout >> " Seçenek :";
      cout.flush();
      cin<<choice;
    };
    cout>>endl;
    cout.flush();
    return choice;
 }

 int Push(int number)
 {
   // Yy?ynda yer olup olmady?yny kontrol et.
     if (itemsInStack == Max_StackLen)
```

```
        return 0;
    intStack[itemsInStack] = number;
    itemsInStack++;
    return -1;
}

int Pop(int &number)
{
   // Yy?yn bo? mu diye kontrol et.
   if (itemsInStack == 0)
    return 0;
   itemsInStack--;
   number = intStack[itemsInStack];
   return -1;
}
```

<Resim>C/C++ dillerinde derleyici tarafından dizlere erişimde boyut kontrolu yapılmamaktadır

<Resim>Peki ilk giren ilk çıkar mantığıyla çalışan Kuyruk veri yapısını diziler aracılığıyla nasıl gerçekleştirebiliriz? Çözümlerinizi bekliyoruz.Cevaplarınızı cpp@programlama.com adresine yollayabilirsiniz.

Diziler İle Stringler Arasındaki İlişki <Resim><Resim>

C/C++'da stringler son elemanları "null terninator" ('\0')') olan belirli uzunluktaki karakter dizileri olarak tanımlanır. Tanımdan da anlaşılacağı üzere Stringler ile bir boyutlu diziler arasında çok sıkı bir bağ vardır. C dilinde stringler için bir veri tipi tanımlanmamasına karşın string sabitleri için bir veri tipi vardır. Bir string sabiti çift tırnak arasında verilmiş karakterler listesidir.

"Örnek bir string "
String sabitlerinin sonuna null terminator eklenmesine gerek yoktur. C derleyicisi bizim yerimize otomatik olarak bu işi yapar.

char str[] = "Örnek bir string";
char str[17] = "Örnek bir string";
Yukarıdaki değişken tanımlamalarının her ikisi de geçerlidir. Birinci tanımlamada derleyici str str isminde “Örnek bir string” ifadesini tutabilecek kadar uzunluğa sahip karakter dizisi tanımlar. İkinci tanımlamada ise 17 elamandan oluşan bir karakter dizisi tanımlar ve bu diziye “Örnek bir string” ifadesini atar. İki tanımlamada ilk bakışta aynı işi gerçekleştiriyormuş gibi gözükebilir fakat işleyiş şekillerinde ufak bir nüans farkı vardır. Birinci kulanım şeklinde verdiğiniz string'i tutmnak için gerekli olan yerin uzunluğunu siz hesaplamıyorsunuz. Dolayısıyla ilerde bir şekilde bu string’i değiştirmeniz gerektiğinde ayrılması gereken yerin doğru olarak hesaplanıp hesaplanmadığını kontrol etmeniz gerekmez. Kısacası bir riske girmiyorsunuz kontrolu derleyiciye bırakıyorsunuz. Fakat ikinci tanımlamada aynı durum söz konusu değildir. Eğer ki tanımlamada verilen string 17 karakterden daha fazla yere ihtiyaç duysaydı veya ilerde string'i değiştirip yerine daha uzun bir string girdiğinizde derleyici hata verecekti.

char str[3] = "123456";
Örneğin derleyici yukarıdaki tanımlama ile karşılaştığında hata verecektir.

Sizi ayırmanız gereken yerin boyutunu doğru girmeye zorlayacaktır.
C programa dili stringler üzerinde işlem yapabilmek için çok çeşitli foksiyonlar içermektedir. Bu fonksiyonların prototipleri string.h header dosyasında bulunmaktadır

<Resim>Programlarınızda mümkün sihirli rakamlar kullanmaktan kaçının. Sihirli rakamlar yerine #define ile tanımlamış ifadeler veya sabitler kullanın. Bu programınızın anlaşıla iblirliğini arttırırken bakımını da kolaylaştırır.

char mesaj[255];

Yukarıdaki program satırı yerine aşağıdaki kodu kullanın

const int Max_MesajUzunlugu = 255;
char mesaj[Max_MesajUzunlugu];
Dizilerin Fonksiyonlara Parametre Olarak Geçilmesi <Resim><Resim>
C dilinde diziler fonksiyonlara daima referans olarak geçilirler. Diğer değişken tiplerinde olduğu gibi değer ile çağırma yöntemi diziler için kullanılamaz. Aşağıda fonksiyonlara dizilerin parametre olarak geçilme şekillerinden bazılarını inceliyeceğiz.
Print(char dizi[]);
Yukarıdaki tanımlama Print fonksiyonuna bir karakter dizinin referansının (Bir string’in başlangıç adresinin) geçileceğinin anlatır.Tanımlamada parametre olarak geçilen dizinin boyutu hakkında bir bilgi bulunmamaktadır. Bir boyutlu bir dizini referansını elde etmek için dizinin boyutunun bilinmesine gerek yoktur.

İkinci bir çağırma şekli ise direk olarak dizinin tanımlandığı tipte bir pointer'ı fonksiyona parametre olarak geçileceğini ifade eder.. Aşağıda bu kullanım şekli örneklenmektedir.
Print(char *diziBaslangici); Bu kullanım şeklinde ise fonksiyonumuz bir char pointer alacak şekilde tanımlanmıştır. Bu tanımlama şeklide yukarıdaki ile aynı işlevi gerçekleştirmekterdir.
Print(char dizi[35]);
Yukarıdaki tanımlama şekli ise yine Print fonksiyonun char bir dizinin referansını parametre olarak alacağını belirtmektedir. Dizinin boyut bilgisi derleyici tarafından göz ardı edilir çünkü yukarıda da belirttiğimiz gibi dizinin başlangıcına ait bir referans elde etmek için dizinin boyutunun bilinmesine gerek yoktur. Bu bilgi ancak kodu inceleyen programcıya bilgi verebilir.

<Resim>Dizileri fonksiyonlara adreslerinin geçilmesi fonksiyon içinde dizi üzerinde yapılcak değişiklerin dizini içeriğini değiştireceğinin unutmamak gerekir. Yapılan değişiklikler yerle bir dizi kopyası üzerinde gerçekleştirilmez. Orjinal dizi değişir.
Diziler İle Pointerlar Arasındaki İlişki <Resim><Resim>
C programlama dilinde pointerlar ile diziler arasında çok yakın bir ilişki vardır. Herhangi bir dizinin ismi dizinin ilk elemanına ait bir pointer olarak tanımlanır.

```
int main(int argc, char **argv)

{

int array[5];

cout>>"Bir değer atanmadan önce array[0] = ">> array[0]>>endl;

*array = 2;

cout>>"Değer atanma işleminden sonra array[0] = ">> array [0]>>endl;

cout>>"array[0]'in adresi (&array[0]) =" >>& array [0]>>endl;

cout>>"array'in değeri array =" >> array >>endl;

getch();

return 0;

}
```

Yukarıdaki örnek program diziler ile pointerlar arasındaki ilişkiyi açık bir şekilde göstermektedir. Kodu basamak basamak incelersek array isminde 5 elemandan oluşan bir int dizi tanımlanıyor. Bu dizinin ilk elemanının değeri ekrana yazdırılıyor. Sonraki adımda ise array Yukarıdaki örnek program diziler ile pointerlar arasındaki ilişkiyi açık bir şekilde göstermektedir. Kodu basamak basamak incelersek array isminde 5 elemandan oluşan bir int dizi tanımlanıyor. Bu dizinin ilk elemanının değeri ekrana yazdırılıyor. Sonraki adımda ise array değişkeninin gösterdiği adrese 2 değeri atanıyor ve array array dizisinin ilk elemanı tekrar ekrana yazdırılıyor.

Aşağıdaki ekran çıktısına bakarsak ilk eşitleme işleminden önce array[0]]'ın 0 değerine sahipken dizini isminin gösterdiği adrese değer 2 atanmasında sonra array[0]]'ın değerinin 2 olduğunu görüyoruz. Buradan da anlaşılacağı üzere array ifadesi array dizisinin ilk elemanını göstermektedir. Sonraki satırlarda ise dizinin ilk elemanının ve dizinin ismimin gösterdiği adresler sırası ile ekrana yazdırılmaktadır.

Bir değer atanmadan önce array[0] = 0

Değer atanma işleminden sonra array[0] = 2

array[0]'in adresi (&array[0]) =0066FDF0

array 'in değeri array =0066FDF0

Pointerlar konusunda da söylediğimiz gibi bir pointer değişkenin değerinin arttırmak suretiyle
pointer'ın tanımlandığı tipteki bir sonraki elemanın adresine ulaşabiliriz. Diziler, dizinini boyutu kadar elemanın dizinin başlangıç adresinden itibaren hafızada ardışık oluşturulması şeklinde elde edilirler. Dizinin i. Elemanı i+1'inci elemandan hemen önce gelmektedir. Pointer aritmetiği suretiyle dizi elemanlarınları üzerinde gezine biliriz. Aşağıdaki örnek programda 5 boyutlu bir int dizisinin elemanları hem dizi değişkeninin üzerinde dizi index'i aracılığıyla hem de dizinin başlangıç adresini gösteren bir pointer değişkenine çeşitli artımlar verilmek suretiyle ekrana yazdırılmaktadır.

```
int main(int argc, char **argv)

{

int array[5] = {0,1,2,3,4};

int *pInt = array;
```

```
// Dizi elemanlaryna de?er

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

cout>>"array[">>i>>"]=">>array[i]>>endl;

cout>>"Pointer değişken aracılığıyla dizi üzerinde dolaşılması">>endl;

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

cout>>"*(pInt + ">>i>>")=">>*(pInt + i)>>endl;

getch();

return 0;

}
```

```
array[0]=0

array[1]=1

array[2]=2

array[3]=3

array[4]=4

Pointer değişken aracılığıyla dizi üzerinde dolaşılması

*(pInt + 0)=0

*(pInt + 1)=1

*(pInt + 2)=2

*(pInt + 3)=3

*(pInt + 4)=4
```

Çok Boyutlu Diziler <Resim><Resim>

Bu bölümde birden çok boyutu olan dizileri ele alacağız. Çok boyutlu diziler en basit haliyle iki boyutlu diziler olarak karşımıza çıkar. İki boyutlu dizileri; bir boyutlu dizilerden oluşan bir boyutlu diziler olarak da ifade edilebilir.

```
char strArray[30][255];
```

Yukarıdaki komut satırında herbiri 255 byte uzunluğunda 30 stringden oluşan strArray isiminde bir dizinin tanımı yapılmıştır Yukarıdaki ifade aşağıdaki tanımlamalar ile eşdeğerdir.

```
typedef char myString[255]. NewType;
```

newType strArray[30];

Çok boyutlu dizilerde her bir boyut dizi elamnalrına erişlimde ekstra yük getirecektir. Yani aynı boyutlarda çok boyutlu bir dizini elemanlarına erişmek tek boyutlu diziye göre daha yavaş olacaktır.

Çok boyutlu dizlerini fonksiyonlara parametre olarak geçilmelerinde dizini her bşr boyutunun beliritilmesi gerekmektidir. Sadece en soldaki indisi belirtmeme şansına sahibiz.

double dizi[12][13][14];
Şeklinde tanımlanmış bir diziyi parametre olarak alacak fonksiyonun prototipi aşagıdaki gibi olacaktır.
bool myFuntion(dizi[12][13][14]);
veya
bool myFuntion(dizi[][13][14]);