ALGORİTMALAR VE PROGRAMLAMAYA GİRİŞ



DR. ÖGR. ÜYESİ HAYRETTİN EVİRGEN

Üniteler

1. Algoritma Kavramı Ve Algoritmaların İfade Edilme Yöntemleri

2. Programlamanın Temelleri

3. Karar Yapıları (ıf, If….else)

4. Kümelenmiş If….else Yapıları

5. Döngüler

6. Diziler

7. Fonksiyonlar

8. Sıralama Ve Arama Algoritmaları

9. Yapılar (structures)

10. İşaretçiler (poınters)

11. C++ Kullanılan Hazır Fonksiyonlar

12. Sınıf Ve Nesneler

13. Kalıtım

14. Dosyalar

1. ALGORİTMA KAVRAMI VE ALGORİTMALARIN İFADE EDİLME YÖNTEMLERİ

Giriş

Algoritma kavramı için, bilgisayarların bir problemi çözebilmesi için gerekli tüm adımları açıklayan bir tarif olarak düşünülebilir. Daha açık ifade edilmesi gerekirse, algoritmalar bilgisayara nelerin yapılması gerektiğini söyleyen talimatlar dizisidir. Hepimiz yemek kitaplarından veya internet üzerinden yemek tariflerini incelemişizdir. Bu tariflerde yemeği yapabilmemiz için gerekli malzeme ve yemeğinin nasıl yapılacağını adım adım açıklayan bir liste vardır. Algoritma da böyledir. Bir programlama algoritması, bir şeyin nasıl yapılacağını açıklar. Bu açıklamalar, bilgisayarınızın anladığı dile dönüştürüldüğünde, bilgisayarlar bunu her defasında açıklamalara uygun ve tam olarak çalıştırır.

Bugün algoritma konusu sadece bilgisayar bilimi ile uğraşanları değil, her alandan kişileri, kuruluşları da ilgilendiren bir konu olduğunu söylemek abartı olmayacaktır. Bundan birkaç kuşak önce algoritma denince insanların kafasında bir şey canlanmıyordu. Fakat bugün hangi işi yaparsak, hangi meslekten olursak olalım algoritmaları her gün her an kullanıyoruz. Algoritmalar her yerde. Bilgisayarlarımızda, Cep telefonumuzda, her gün kullandığımız asansörde, bindiğimiz otomobilde, uçakta, evde kullandığımız eşyaların birçoğunda.

Uzun zamandan beri teknolojisinin arkasında önemli algoritmalar yatan arama motorlarını kullanıyoruz ve bu şekilde istediğimiz bilgiye kolayca ulaşabiliyoruz. Alışverişlerimizin birçoğunu elektronik ticaret sitelerinden güvenli bir şekilde yapıyoruz. Çünkü bu siteler bizim için sır olan kredi kartı ve benzeri bilgilerimizi şifreleme algoritmaları ile İnternet üzerinden güvenli bir şekilde, üçüncü kişilerden koruyarak taşıyor. Bilgisayarlarımız, hata düzeltme algoritmalarını bize fark ettirmeden, her an kullanarak hayatımızı önemli ölçüde kolaylaştırıyor. Hayatımızda sıkıştırma algoritmaları olmasaydı verilerimizi internet üzerinden bir noktadan başka bir noktaya hızlı bir şekilde, daha az maliyetle nasıl gönderecektik?

Hayatımızda, yukarıda saydığımız algoritmalar gibi, en az onlar kadar etkili, yaygın kullanılan çok sayıda algoritma var ve bilim insanları her gün gece gündüz çalışarak bunlara yenilerini ekliyor. Bilgisayar bilimciler birbirlerinin çalışmalarını temel alarak, yeni problemlere çözüm olması için yeni algoritmalar geliştiriyor. Bu süreç hiçbir zaman olmadığı kadar baş döndürücü bir hızla devam ediyor. Algoritmalar hayatımızı, yaşama biçimimizi sarsıcı bir şekilde değiştirmeye devam ediyor.

1.1. Yaygın Olarak Kullanılan Algoritma Tanımlarından Bazıları

- Algoritma, belirli bir problemi çözmek için gerekli özgün adımlar serisidir (Dr. James F. Wirth).

- Matematikte ve Bilgisayar biliminde bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumda sonlanan, sonlu elemanlar kümesidir(Wikipedia).

- Algoritma, bir problemi çözmek için gerekli yöntemi tanımlar (Goldschlager/Lister).

- Algoritma, belirli bir veri kümesi üzerinde kullanılacak olan aritmetiksel işlemlerin sırası ve türünü belirleyen kurallar kümesidir.

- Algoritma bilgisayara neleri yapması gerektiğini söyleyen talimatlar dizisidir.

- Algoritmalar, bir problemi sonlu zamanda çözebilmek için geliştirilmiş, açık, yürütülebilir, sıralı, basit ve gerektikçe tekrarlanan adımlardan oluşan yöntemlerdir.

- Belli bir problemi çözmek veya belirli bir amaca ulaşmak için tasarlanan yol(Wikipedia)

1.2. Algoritmaların Genel Özellikleri

a. Kesin Olmalıdır: Talimatlar bir bilgisayar tarafından yerine getirebilecek kadar kesin ve belirsizlikten uzak olmalıdır. Algoritmanın her adımı anlaşılır, şekilde ifade edilmiş olmalıdır.

b. Etkin Olmalıdır: Gereksiz tekrarlardan uzak olmalıdır. Birçok farklı problemin çözümünde kullanılabilecek genel özelliklere sahip olmalıdır. Örneğin, öğrencilerine yüzlük sistemde not veren bir üniversitenin, öğrenci işleri programının algoritması tasarlanırken yüzlük sistemin yanında dörtlük, beşlik vb. diğer sistemler de de kullanılacak şekilde bir algoritma tasarımı yapılmalıdır.

c. Sonluluk: Algoritma belirli sayıda adımdan oluşmalı, sonsuz döngüye(loop) girmemelidir.

d. Giriş bilgisine karşılık bir çıkış bilgisi sağlamalıdır: Algoritma, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumda sonlanan ve bu noktada bir giriş verisine karşılık bir çıkış verisi üreten özelliğe sahiptir.

e. Başarım ve Performans: Algoritmanın bilgisayarın belleğinde depolamaması gerek bilgi, bilgisayarın belleğinden fazla olmamalıdır. Gereksiz tekrarlar içermemelidir. Algoritmanın çalışma zamanının makul sürelerden fazla olmamalıdır.

1.3. Algoritma Dokümanlarında Olması Gereken Özellikler

a. Algoritma adı: Algoritmaya anlamlı bir isim verilmelidir.

b. Yaptığı iş: Algoritmanın Ne iş yapıldığı, kullanılan sabitler, değişkenler, sınıflar, nesneler hakkında yeterli bilgi bulunmalıdır.

c. İşlem adımları: Algoritmada işlem adımlarına numara verilmeli veya girintiler kullanarak anlaşılabilir olması sağlanmalıdır.

d. Açıklama: Algoritma metinlerinde yeterli açıklayıcı bilgi bulundurulmalıdır.

1.4. Algoritmaların İfade Edilme Şekilleri

Bir programlama algoritmasının, bilgisayar kodu olmadığını bilmek önemlidir. Algoritmalar basit bir şekilde, Satır Algoritması(Doğal dil) ile veya sözde kodlarla(Pseudocode) veya akış şemalarından(Flow Chart) biriyle ifade edilebilir.

Algoritmalar ifade edilirken, adımları numaralandırmak genellikle iyi bir fikirdir. Ancak, bazen algoritmalar ifade edilirken adımların numaralandırılma tekniği kullanılmaz. Bununun yerine algoritmanın okunabilirliğini arttırmak için, bazı satırlarının daha içeriden bazı satırlarının da daha dışarıdan yazılması yaklaşımı kullanılır.

1.4.1. Algoritmaların Satır Algoritmaları (Doğal Dil) İle İfade Edilmesi

Algoritmaların doğal dil(günlük yazı/konuşma dili) ile ifade edilme şeklidir. İfadenin doğal dil ile yapılmasından dolayı yazılması ve bu şekilde yazılan algoritmaların okunması, anlaşılması kolaydır. Satır algoritmalarda satır numaralarının belirtilmesi gerekmektedir.

Problem 1.1 Klavyeden kısa ve uzun kenarları girilen dikdörtgenin çevresini ve alanını hesaplayan algoritmanın satır algoritmasını yazınız?

Satır Algoritması

1. BAŞLA
2. YAZ "Uzun Kenar "
3. OKU ukenar
4. YAZ "Kısa Kenar "
5. OKU kkenar
6. Cevre=2*(ukenar + kkenar)
7. Alan=ukenar*kkenar
8. YAZ Cevre
9. YAZ Alan
10. SON

1.4.2. Algoritmaların Sözde Kod(Pseudocode) İle İfade Edilmesi

Sözde kod, herhangi bir katı programlama dili sözdizimi veya temel teknoloji hususları gerektirmeyen, gayri resmi bir programlama açıklaması yoludur. Bir programın taslağını oluşturmak için kullanılır. Sözde kod bir programın akışını özetler. Sistem tasarımcılarının, programcıların bir yazılım projesinin gereksinimlerini anlamalarını ve kodu buna göre geliştirmelerini sağlamak için sözde kod yazılır. Sözde kod program ile algoritma veya akış şeması arasında köprü görevi görür.

Sözde kod gerçek bir programlama dili değildir. Bu nedenle yürütülebilir bir programda derlenemez. Sözde kodlar, Algoritmalar belirli bir programlama diline dönüştürülmeden önce, kısa terimler veya basit İngilizce sözdizimleri kullanılarak hazırlanır. Bu, üst düzey akış hatalarını belirlemek ve program geliştiricilerin veri akışlarını anlamaları için yapılır. Kavramsal hatalar sözde kodlar hazırlanırken önceden düzeltildiği için bu, gerçek programlama sırasında kesinlikle zaman kazanmanıza yardımcı olur.

Sözde kodun hazırlamanın ana amacı, bir programın her satırının tam olarak ne yapması gerektiğini açıklamaktır. Sözde kod aşamasında hataları yakalamak veya yanlış program akışlarını tespit etmek, bu hataları daha sonra yakalamaktan daha az maliyetli olduğundan, programların kodlanması aşamasına geçmeden önce sözde kod hazırlamak faydalıdır. Sözde kod hatalardan arındırılıp kabul edildiğinde, bir programlama dilinin kelime ve sözdizimi kullanılarak program yazılmasına geçilir.

İp Ucu:

Sözde Kod;

1. *Yürütülebilir bir programda derlenemez*
2. *Programcıların sadece kod geliştirme sürecinin algoritma kısmına odaklanmasını sağlar.*
3. *Her tür programcı tarafından anlaşılır.*

Problem 1.2 Klavyeden kısa ve uzun kenarları girilen dikdörtgenin çevresini ve alanını hesaplayan algoritmanın sözde kodunu yazınız?

Sözde Kod

```
1. Begin
2.    Print "Uzun Kenar ";
3.    Read ukenar;
4.    Print "Kısa Kenar ";
5.    Read kkenar;
6.    Cevre =2*(ukenar + kkenar);
7.    Alan=ukenar*kkenar;
8.     Print Cevre;
9.     Print Alan;
10. End
```

1.4.3. Algoritmaların Akış Şemaları İle İfade Edilmesi

Akış şemaları, algoritmaları görsel olarak temsil etmek için ideal diyagramlardır. Farklı alanlardaki bir süreci veya diyagramı analiz etmek, tasarlamak, belgelemek veya yönetmek için kullanılırlar. Diğer diyagram türlerine benzer şekilde, neler olduğunu görselleştirmeye yardımcı olurlar. Bu, bir süreci anlamaya ve içindeki kusurları ve darboğazları bulmaya yardımcı olur.

Program akış şeması, bir binanın mimari planına benzer. Bir tasarımcı, bir bina inşa etmeye başlamadan önce mimarı plan çizer. Aynı şekilde, bir programcı akış şemasına dayalı bir bilgisayar programı yazmadan önce bir akış şeması çizer. Tıpkı bir plan çizmek gibi, akış şeması, Amerikan Ulusal Standart Enstitüsü, tarafından verilen standart akış şeması sembollerini içeren tanımlanmış kurallara göre çizilir. MS Word veya hatta MS Visio'da akış şemalarını kolayca yapabilirsiniz (nasıl kullanılacağı hakkında daha fazla bilgi edinebilirsiniz)

Bir akış şeması, belirli bir sorunun çözümünü elde etmek için gerçekleştirilecek işlem sırasını gösteren diyagramlara sahiptir. Programcılar ve analistler arasında iletişimi sağlar. Bir akış şeması çizildikten sonra, programı herhangi bir üst dilde yazmak nispeten daha kolay hale gelir. Başka bir deyişle, karmaşık bir programın iyi belgelenmesi için akış şemaları zorunludur. Algoritmalar temsil edilirken iki farklı akış şeması çizilebilir: Yüksek Seviye Akış Şeması, Ayrıntılı Akış Şeması.

Yüksek Seviye Akış Şeması

Bu akış şeması, bir algoritmada önemli adımları göstermektedir. Ayrıca her bir adımın ara çıktılarını ve ilgili alt adımları verir. Bundan başka, sürecin temel bir resmini sunar ve süreç içinde meydana gelen değişiklikleri tanımlar.

Ayrıntılı Akış Şeması

Bu akış şeması, süreçteki tüm adımlar ve faaliyetler dâhil sürecin ayrıntılı bir resmini verir. Sürecin adımlarını ayrıntılı bir şekilde incelemek ve sorunları veya verimsizlik alanlarını saptamak için çok yararlıdır.

Akış Şeması Sembolleri

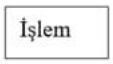
Başlangıç ve Bitiş Sembolleri: Genellikle daireler, oval şekiller veya yuvarlatılmış dikdörtgenlerle gösterilir. Normalde, başlangıç sembolünün içine "BAŞLAT" sözcüğü yazılır. Son sembolün içinde "SON" sözcüğü yazılır.



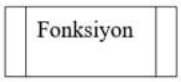
Oklar: Kontrol akışını gösterirler. Bir ok bir simgeden gelip başka bir simgeyle sona erdiğinde, program kontrolünün okun işaret ettiği simgeye geçeceği anlamına gelir.



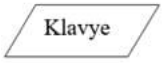
Dikdörtgenler: Bir eylemi veya işlemi temsil ederler.



Dış Fonksiyon: Genellikle çift dikey kenarlı dikdörtgenlerle gösterilirler.



Girdi: Genellikle paralelkenarlar ile gösterilir.



Koşul veya Karar: Elmas olarak tasvir edilmiştir. Bir ok girişi, iki ok çıkışı vardır. Bir ok "Evet" veya "Doğru" anlamına gelir. Diğer ok “Hayır” veya “Yanlış” a karşılık gelir.



Bağlayıcılar: Bunlar bir dairenin içindeki tanımlayıcı bir etiketle gösterilir. Etiketli konektörler ok yerine çok sayfalı diyagramlarda kullanılır.



Görüntüleme: Kâğıt ve ekrana yapılacak çıktıların sembolleri



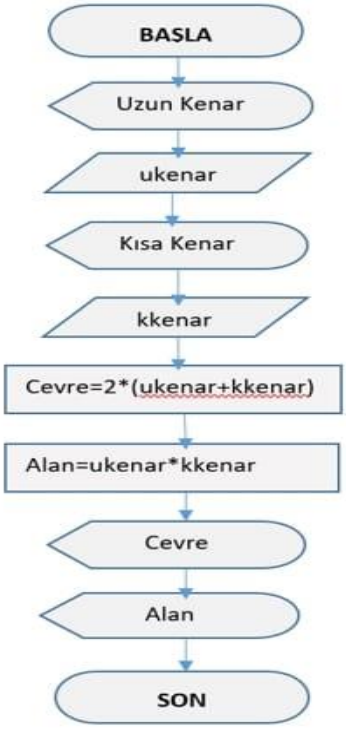
Döngü: Döngülerin gösterimi için kullanılan sembol



Manyetik Disk: Veritabanı sembolü



Problem 1.2 Klavyeden kısa ve uzun kenarları girilen dikdörtgenin çevresini ve alanını hesaplayan algoritmanın akış şemasını çiziniz?



Uygulamalar

Ekrana Merhaba C++ Yazdıran Program

Ekrana Merhaba C++ yazılmasını sağlayan algoritmanın satır algoritmasını, sözde kodunu, akış şemasını ve C++ kodunu yazınız?

a. Satır Algoritması

1. BAŞLA

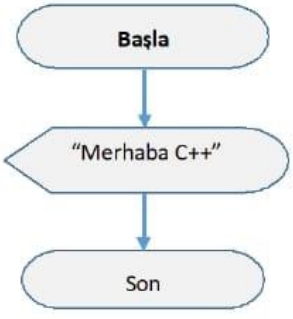
2. YAZ "Merhaba C++"

3. SON

b. Sözde Kod

```
1. Begin
2.    print "Merhaba C++";
3. End
```

c. Akış Şeması



d. C++ Kodu

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Merhaba C++ \n";
    return 0;
}
```

Tam Sayı Tipi Değişken Tanımlamak, Değişkene Değer Atamak ve Toplama Yapmak

Tamsayı türünde iki değişkenin tanımlanan ve bu değişkenlere değer atanan algoritmanın satır algoritmasını, sözde kodunu, akış şemasını ve C++ kodunu yazınız?

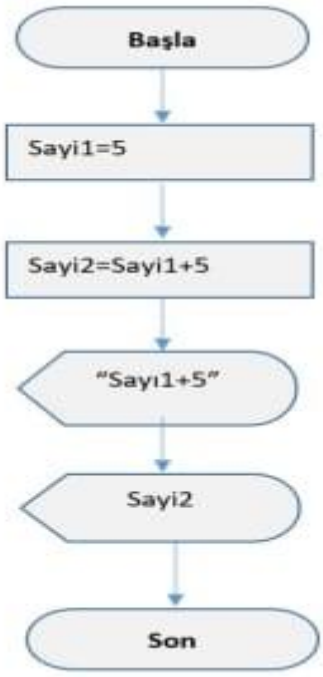
a. Satır Algoritma

1. BAŞLA
2. Sayi1=10;
3. Sayi2=sayi+5
4. YAZ "Sayi1+5="
5. YAZ sayi2
6. SON

b. Sözde Kod

1. Begin
2. Sayi1=5;
3. Sayi2=Sayi1+5;
4. Print "Sayi1+5=";
5. Print sayi2;
6. End

c. Akış şeması



d. C++ Kodu

```
#include <iostream>
using namespace std;
int main()
{
    int sayi1; // sayi1 değişkeni tanımlanıyor
    int sayi2; // sayi2 değişkeni tanımlanıyor
    sayi1 = 10; // sayi1 değişkenine 10 değeri atanıyor
    sayi2 = sayi1 + 5; // sayi2 değişkenine değer atanıyor

    cout << "sayi1 + 5 = "; // metin yazdırılıyor
    cout << sayi2 << endl; // sayi2'nin değeri yazdırılıyor
    /* endl alt satıra geçmek için kullanılır
    "\n" yazmak ile aynıdır  */
    return 0;
}
```

Klavyeden Girilen İki Sayıyı Toplamak

Klavyeden girilen iki sayıyı toplayan ve sonucu ekrana yazdıran algoritmanın satır algoritmasını, sözde kodunu, akış şemasını ve C++ kodunu yazınız?

a. Satır Algoritması

1. BAŞLA

2. YAZ “Birinci Sayıyı Giriniz “

3. OKU birinci sayı

4. YAZ “İkinci Sayıyı Giriniz “

5. OKU ikinci sayı

6. Toplam=birinci sayı + ikinci Sayı

7. YAZ Toplam

8. SON

b. Sözde Kod

```
1. Begin
2.     Print “Birinci sayıyı Giriniz “;
3.    Read Sayı1;
4.     Print “İkinci sayıyı Giriniz “;
5.     Read sayı2;
6.    Toplam=sayi1+Sayi2;
7.    Print Toplam;
8. End
```

c. Akış Şeması



d. C++ Kodu

```
#include <iostream>
using namespace std;
int main( )
{
    int sayi1,sayi2,toplam;
    cout<<"Birinci Sayıyı Giriniz :";
    cin>>sayi1;
    cout<<"İkinci  Sayıyı Giriniz :";
    cin>>sayi2;
    toplam=sayi1+sayi2;
    cout<<toplam<<endl;
    return 0;   }
```

Klavyeden Girilen Fahrenheit Cinsinden Sıcaklığı Santigrad Cinsine Çeviren Program

Klavyeden girilen fahrenheit cinsinden sıcaklığı santigrat cinsine çeviren, sonucu ekrana yazdıran algoritmanın satır algoritmasını, sözde kodunu, akış şemasını ve C++ kodunu yazınız.

a. Satır Algoritma

1. BAŞLA
2. YAZ "Sicakligi fahrenheit cinsinden giriniz:"
3. OKU fsicaklik
4. csicaklik=(fsicaklik-32)*5/9
5. YAZ "Girilen sicaklik: ", csicaklik, " santigrad derece"
6. SON

b. Sözde Kod

1. Begin
2. Print "Sicakliği fahrenheit cinsinden giriniz";
3. Read fsicaklik;
4. csicaklik=(fsicaklik-32)*5/9;
5. Print "Girilen sicaklik: ", csicaklik, " santigrad derece";
6. End

c. Akış Şeması



d. C++ Kodu

```
#include <iostream>
 using namespace std;
  int main()
  {
int fsicaklik;
// Ekrana mesaj yazılıyor
cout << "Sicakligi fahrenheit cinsinden giriniz: ";
// Klavyeden girilen değer okunuyor
cin >> fsicaklik;
// Girilen değer Santigrad cinsine çevriliyor
int csicaklik = (fsicaklik - 32) * 5 / 9;
// Sonuç ekrana yazılıyor
cout << "Girilen sicaklik: " << csicaklik < " santigrad derece" << endl;
return 0;
  }
```

1.5. Algoritma Geliştirme Adımları

Bir algoritmanın (bir planın) geliştirilmesi, bir problemin çözümünde önemli bir adımdır. Bir algoritmaya sahip olduktan sonra, onu bazı programlama dillerinde bir bilgisayar programına çevirebiliriz. Algoritma geliştirme sürecimiz beş ana adımdan oluşmaktadır. Bu adımlar; problemi tanımlamak, analiz, üst düzey algoritma geliştirmek, Algoritmayı hassaslaştırmak, algoritmayı test etmek.

1. Problemi Tanımlamak: Bu adım göründüğünden çok daha zordur. Etkin, başarımı ve performansı yüksek bir algoritma geliştirmek için, problemin analizi yapan gurup tarafından tam olarak anlaşılması gerekir. Aksi durumda beklenen algoritma geliştirilmesi ve buna bağlı olarak doğru çalışan bir bilgisayar programı yazılması imkânsızdır.

Algoritma geliştirmenin bu aşamasında müşteri, problemin tanımını oluşturmaktan sorumludur. Ancak bu genellikle sürecin en zayıf kısmıdır. Zayıflık, tanımının aşağıdaki hata türlerinden bir veya daha fazlasından olumsuz etkilenmesinden kaynaklanmaktadır: (1) açıklama, belirtilmemiş varsayımlara dayanır, (2) açıklama belirsizdir, (3) açıklama eksiktir veya (4) açıklamanın iç çelişkileri vardır. Bu eksiklik, sorunu tanımlayan müşterinin dikkatsizliğinden veya kullanılan dilin belirsizliğinden kaynaklanabilir.

Analiz grubunun sorumluluğunun önemli bir kısmı, problemin tanımlanası sırasında yapılan hataları tespit etmek ve hataları gidermek için müşteri ile çalışmaktır.

2. Analiz: Analiz adımının amacı, problemi çözmek için hem başlangıç hem de bitiş noktasını belirlemektir. Bu süreç, neyin verildiğini ve neyin kanıtlanması gerektiğini belirleyen bir matematikçinin yaşadığı sürece benzer. İyi bir problem tanımı, bu adımı gerçekleştirmeyi kolaylaştırır.

Başlangıç noktasını belirlerken, aşağıdaki sorulara cevap aranarak başlanabilir: Hangi veriler mevcut? Bu veriler nerede? Problemle ilgili hangi formüller var? Verilerle çalışmak için hangi kurallar var? Veri değerleri arasında hangi ilişkiler vardır?

Bitiş noktasını belirlerken, çözümün özelliklerini tanımlamamız gerekir. Başka bir deyişle, işimiz bittiğini nasıl bileceğiz? Aşağıdaki soruları sormak genellikle bitiş noktasını belirlemeye yardımcı olur: Ne gibi yeni gerçeklerimiz olacak? Hangi öğeler değişecek? Bu öğelerde ne gibi değişiklikler yapılacak? Artık hangi şeyler olmayacak?

3. Üst Düzey Algoritma Geliştirmek: Algoritma, bir sorunu çözmek için bir plandır ve bu planlar birkaç ayrıntı düzeyinde yapılabilir. Algoritmanın geliştirilmesine çözümün büyük bölümünü içeren yüksek seviyeli bir algoritma hazırlamakla başlamak genellikle daha iyidir, ancak ayrıntılar daha sonra ele alınmalıdır.

4. Algoritmayı Hassaslaştırmak: Üst düzey bir algoritma, bir sorunu çözmek için izlenmesi gereken önemli adımları gösterir. Algoritma geliştirmenin bu dördüncü adımında algoritmaya ayrıntı eklememiz gerekiyor. Ancak, ne kadar ayrıntı eklemeliyiz? Ne yazık ki, bu sorunun cevabı duruma bağlıdır.

Bu adımda amacımız, bilgisayar programı yazarken kullanacağımız algoritma geliştirmek olduğundan, programlayıcının yeteneğine uyacak detay seviyesini ayarlamamız gerekir. Şüphe duyduğumuzda veya öğrenirken, çok az ayrıntıya sahip olmaktan, çok fazla ayrıntıya sahip olmak her zaman daha iyidir.

Basit problemlerin çözümü için geliştirilen algoritmalarda, tek bir adımda üst düzey algoritmadan ayrıntılı bir algoritmaya geçilebilir. Ancak, bu her zaman mümkün olmayabilir. Daha büyük, daha karmaşık problemler için, bu işlemi birkaç kez tekrarlamak, önce orta seviye algoritmalar geliştirmek gerekir. Bu tekrarların her defasında, önceki algoritmaya daha fazla ayrıntı eklemeliyiz, daha fazla ayrıntıya ihtiyaç kalmadığı noktada durmalıyız.

5. Algoritmanın Sınanması: Son adım algoritmayı sınamaktır. Ne arıyoruz? İlk olarak, geliştirilen algoritmanın orijinal problemi çözüp çözemeyeceğini belirlemeliyiz. Bunun için algoritma üzerinde adım adım çalışmamız gerekir. Algoritmanın probleme bir çözüm sağladığı konusunda tatmin olduktan sonra başka şeyler aramaya başlamalıyız. Aşağıdaki sorular, bir algoritmayı test etmek için sorulabilecek sorulara örnek olabilir.

Geliştirilen algoritma çok çok özel bir problemi mi çözüyor, yoksa daha genel bir problemi mi çözüyor? Çok çok özel bir problemi çözüyorsa, genelleştirilmeli mi? Örneğin, 3 metrelik yarıçapa sahip bir dairenin çevresini hesaplayan bir algoritma (formül 2 *π * 3) çok özel bir problemi çözer, ancak herhangi bir dairenin çevresini (formül 2 * π * r) hesaplayan bir algoritma daha genel bir problemi çözer.

Bu algoritma basitleştirilebilir mi? Bir dikdörtgenin çevresini hesaplamak için bir formül: uzunluk + genişlik + uzunluk + genişlik, daha basit bir formül şu şekilde olabilir: 2.0 * (uzunluk + genişlik)

Bu çözüm başka bir sorunun çözümüne benziyor mu? Nasıl benzerlikler var? Nasıl farklılar var? Örneğin, aşağıdaki iki formülü göz önünde bulunduralım: Dikdörtgen alan = uzunluk * genişlik Üçgen alanı = 0.5 * taban * yükseklik. *Benzerlikler:* Her biri bir alan hesaplar. Her biri iki ölçümü çarpar. *Farklılıklar:* Farklı ölçümler kullanılır. Üçgen formülü 0.5 içerir. *Hipotez:* Belki de her alan formülü iki ölçümün çarpılması ile hesaplanabilir.

Algoritmanın sınanma işlemi, bu sorular çoğaltılarak ve cevapları tespit edilerek tamamlanır ve bu adım geçildikten sonra bilgisayar programcılarına programlarını geliştirirken kullanacakları kusursuz bir algoritma dokümanı sunulur.

Bölüm Özeti

Bu bölümde, günlük hayatımızda kullandığımız algoritmaların hayatımızı nasıl kolaylaştırdığını, algoritmaların tanımını, başarımı ve performansı yüksek, etkin bir algoritmanın genel özelliklerini ve de algoritma dokümanlarında bulunması gereken özellikleri öğrendik.

Algoritmaların ifade edilmesi için kullanılan üç ayrı yöntem(Satır algoritması, Sözde kod, Akış şeması) konusunda bilgi sahibi olduk ve örneklerle konuyu pekiştirdik.

Bir algoritma geliştirme döngüsündeki farklı adımları öğrenerek, basit algoritmalar hazırlayacak beceri kazandık.

ÜNİTE SORULARI

1. Aşağıda algoritmalarla ilgili verilen tanım cümlesinde bırakılan boşluğa hangi kelimeler grubu gelmelidir?

"algoritmalar bilgisayara ……. talimatlar dizisidir."

- Akış şemalarını ifade eden
- Satır algoritmalarını ifade eden
- Sözde kodları ifade eden
- Nelerin yapılması gerektiğini söyleyen
- Nelerin yapılmaması gerektiğini söyleyen

2. Aşağıda verilen ifade Algoritmaların hangi özelliğini tanımlar?

"**Gereksiz tekrarlardan uzak olmalıdır. Birçok farklı problemin çözümünde kullanılabilecek genel özelliklere sahip olmalıdır."**

- Kesin olama özelliği
- Etkin olma özelliği
- Sonlu olma özelliği
- Giriş bilgisine karşılık çıkış bilgisi sağlama özelliği
- Başarım ve Performans

3. Aşağıdakilerden hangisi Algoritma dokümanlarında olması gereken özelliklerden değildir?

- C++ Kodu
- Algoritma Adı
- Algoritmanın yaptığı iş
- Algoritmanın işlem adımları
- Açıklamalar

4. Aşağıdakilerden hangisi satır algoritmalarını ifade etmek için kullanılır?

- C++
- Doğal dil(günlük yazı/konuşma dili)
- Akış şeması
- Geometrik şekiller
- Sözde Kod

5. Aşağıda verilen geometrik şekil algoritmaların akış şemaları ile ifadesinde hangi işleme karşılık gelir?



- Karar
- Döngü
- İşlem
- Giriş
- Başla/Son
    -
    - CEVAP ANAHTARI1. d2. b3. a4. b5. c

2. PROGRAMLAMANIN TEMELLERİ

Giriş

Çok basit bir program yazmak için bile, o dile ait bazı esasları bilmek gerekir. Bu bölümde C++ dilinin temel özelliklerinden bazıları tanıtılacaktır. Tanıtılacak esaslar, Temel program yapısı, değişkenler, giriş çıkış, açıklamalar, aritmetik operatörler, arttırma operatörü, veri dönüşümleri ve kütüphane fonksiyonlarıdır.

Bu bölümde ele alınacak konular zor değildir. Ancak, öğrenciler arasında C ve C++ dillerinin zor olduğu konusunda yaygın bir kanaat vardır. C++'ı tanıdıkça bu dil size daha az rahatsız edici görünecektir.

Bu ve bundan sonraki bölümlerde, programlama kavramları ele alınırken verilecek örnekleri çalıştırmak için DEV C++ derleyicisini kullanabilirsiniz. Derleyiciler, sizin yazacağınız kaynak kodları çalıştırılabilir bir dosya haline getirir. C++ 'da kaynak dosyalar, .cpp uzantılı metin dosyaları, çalıştırılabilir dosyalar ise, .exe uzantılıdır.

2.1. DEV C++ Kurulumu ve Kullanılması

Bu ve bundan sonraki bölümlerde, programlama kavramları ele alınırken verilecek örnekleri çalıştırmak için DEV C++ derleyicisini kullanabilirsiniz. Derleyiciler, sizin yazacağınız kaynak kodları çalıştırılabilir bir dosya haline getirir. C++ 'da kaynak dosyalar, .cpp uzantılı metin dosyaları, çalıştırılabilir dosyalar ise, .exe uzantılıdır. Aynı zamanda Dev C++ size kaynak dosyalarınızı oluşturabilmeniz için bir editör, hata ayıklayıcı ve benzeri birçok özellik sunar. Program geliştiricilere bu şekilde birçok kolaylık sağlayan yazılımlara ***Tümleşik Geliştirme Ortamı***(*Integrated Development Environment- IDE*) denir.

C++ programlarını geliştirmek için piyasada ücretli veya ücretsiz birçok tümleşik geliştirme ortamı vardır. Örneğin Visual Studio, Eclipse, NetBeans, Dev C++ bunlardan birkaçıdır. Biz bu derste örnek uygulamalarımızı Dev C++ ortamında geliştireceğiz. Bu IDE 'yi seçmemizin nedeni, bu geliştirme ortamının sadeliği ve kolay olmasıdır.

2.1.1. DEV C+'ın Kurulumu

İlk geliştirilmesi Bloodshed firması tarafından yapılan DEV C++, 2011 yılından sonra Orwell firmasına geçmiş ve bundan sonraki geliştirmeler bu firma tarafından yapılmıştır. İnternet üzerinden indirip bilgisayarımıza kurabileceğiniz DEV C++'ı https://sourceforge.net/projects/orwelldevcpp/ adresinden indirebilirsiniz.

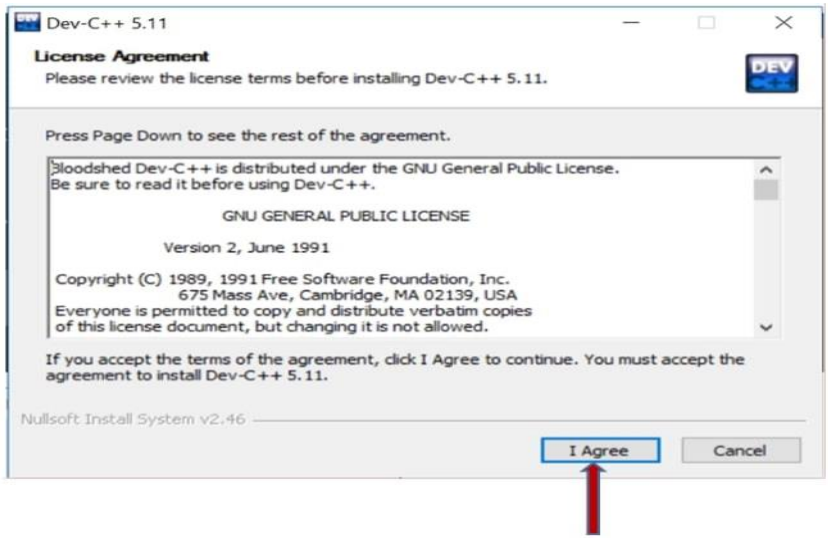
1. önce DEV C++'ı https://sourceforge.net/projects/orwelldevcpp/ adresinden bilgisayarınıza indiriniz.



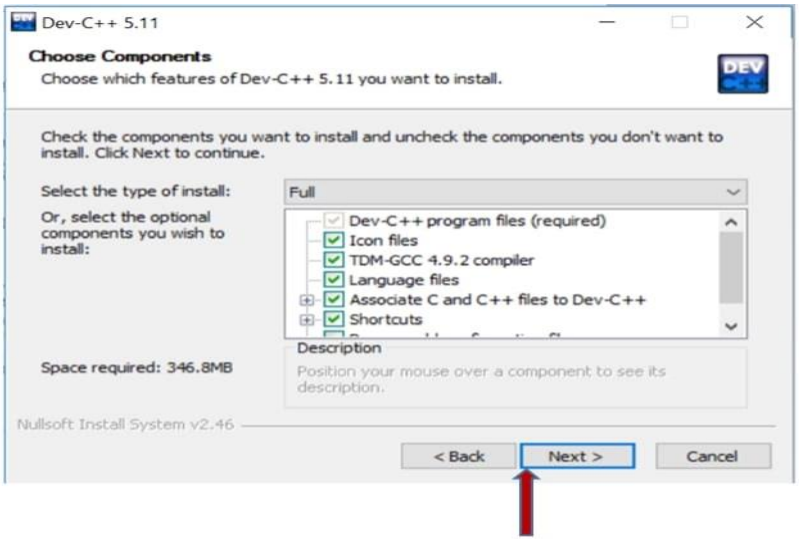
2. İndirilen dosyayı çalıştırın ve aşağıdaki pencereden dil seçimini yapınız.



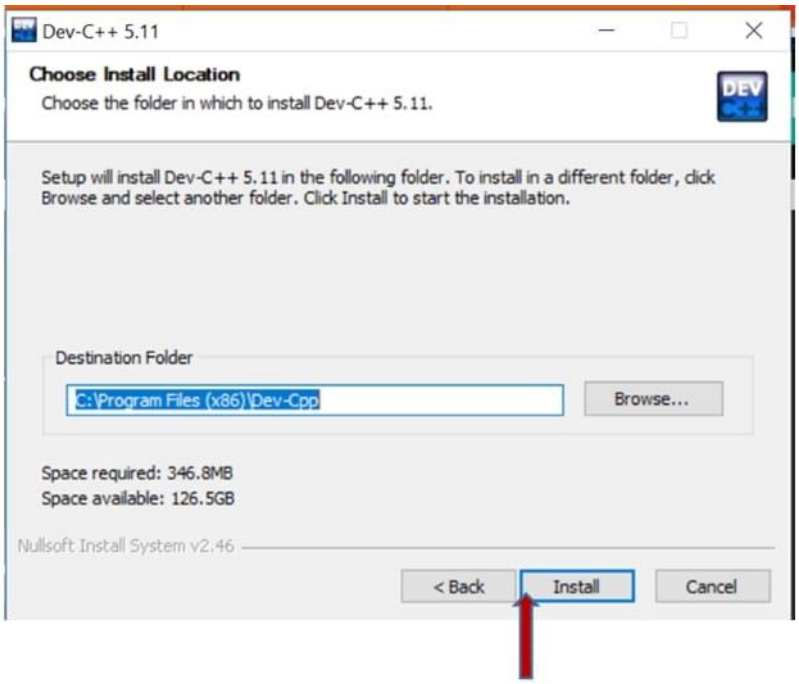
3. Sözleşmeyi okuyup onaylayınız.



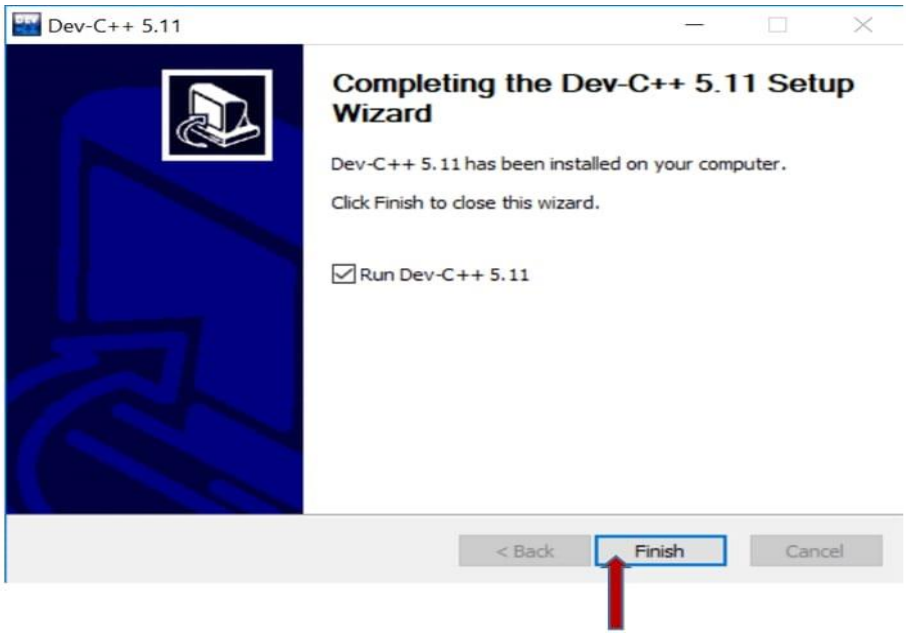
4. Kurulum tipini **Full** seçiniz ve **ileri** butonunu tıklayınız.



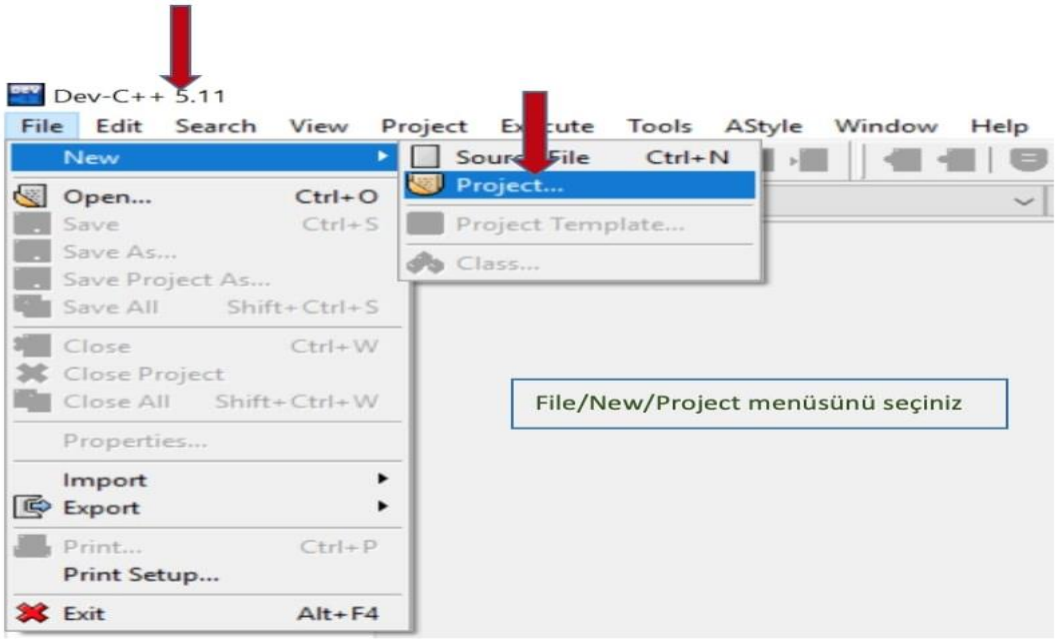
5. Hedef klasör seçiminde bir değişiklik yapmadan **Install** butonuna tıklayınız.

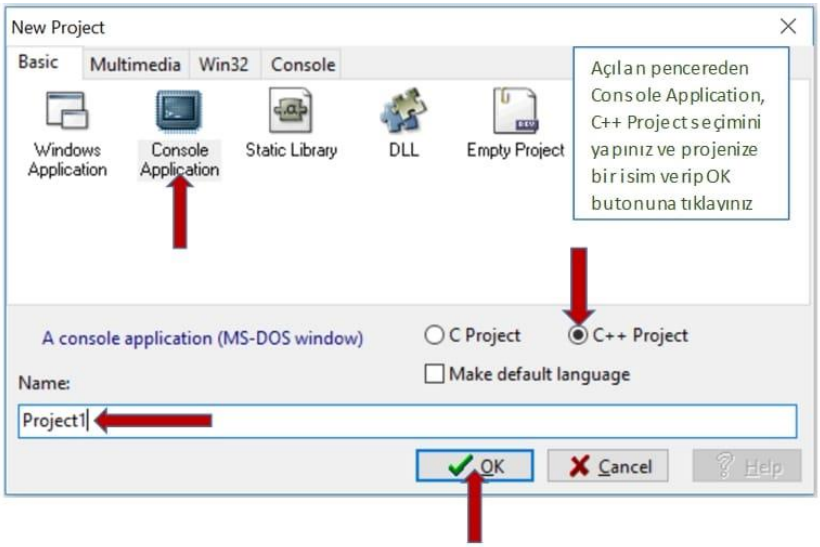


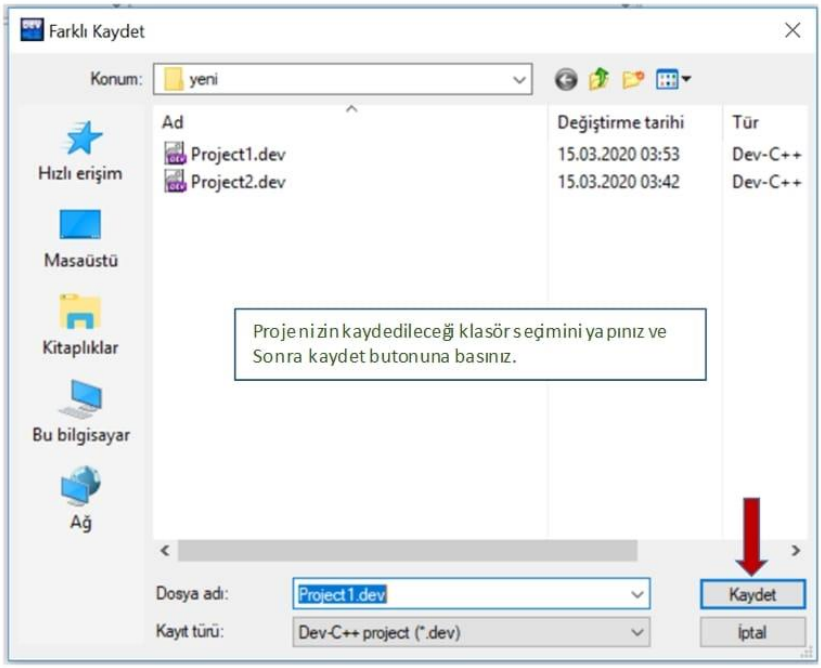
6. Kurulum tamamlandıktan sonra gelen pencereden **Finish** butonuna tıklayınız.



2.1.2. DEV C++ Kullanımı







File/New/Project menüsünü seçiniz

2.2. C++ Program Yapısı

Aşağıda verilen C++ programı, ekrana "*ilk C++ programı*" cümlesini yazar. Program basit olmasına karşılık, C++ programlama dili hakkında birçok bilgiyi içerir. Bu bölümde aşağıda verilen C++ programı ile ilgili ayrıntılar ele alınacaktır.

```
#include <iostream>

using namespace std;

int main()

{

cout << "ilk C++ Programı \n";
```

2.2.1. Fonksiyonlar

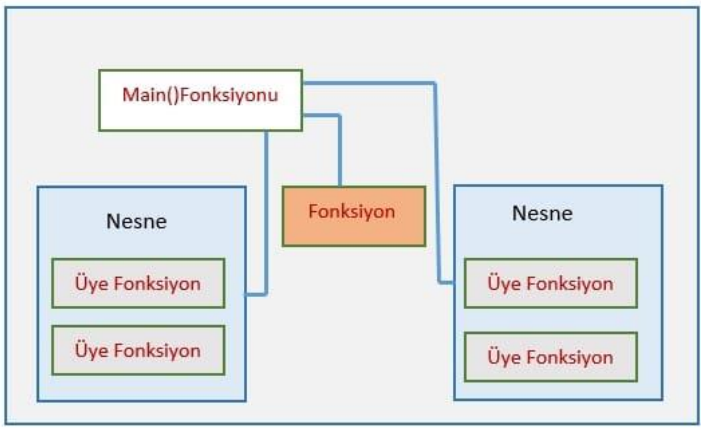
Fonksiyonlar C++'ın en temel bloklarından biridir. Örnek programımız hemen hemen main fonksiyonundan oluşmaktadır.

Her C++ programı, çalıştırıldığında main fonksiyonu içerisindeki ifadeler çalıştırılır. Eğer programda main isimli bir fonksiyon yoksa program çalıştırıldığında hata verir. main() kelimesinin peşinden gelen parantezler bir fonksiyonun ayırt edici özelliğidir. Fonksiyonda parantezlerin kullanılmaması, derleyicinin ***main*** ifadesini bir değişken veya başka bir program öğesi olarak yorumlamasına neden olur.

Fonksiyon isminden önce gelen *int* kelimesi, bu fonksiyonun *int* tipinde bir değer döndüreceğine işaret etmektedir.

2.2.2. Küme Parantezleri ve Fonksiyonun Gövdesi

C++'ta fonksiyon gövdesi küme parantezleri içine alınır. Bu parantezler diğer dillerdeki BEGIN ve END anahtar sözcüklerine karşılık gelir. Küme parantezleri program ifadelerinden oluşan bir bloğu çevreler ya da sınırlandırır. Her fonksiyonun gövdesi etrafında bu parantez çiftlerinden biri olmak zorundadır. Yukarıdaki fonksiyon gövdesinde yalnızca iki ifade bulunduğuna dikkat ediniz. C++ 'ta main fonksiyonu çeşitli nesnelerin fonksiyonlarını veya bağımsız fonksiyonları çağırmak için kullanılır. Bu durum aşağıda şekil 2.1'de gösterilmiştir.



Şekil 2.1 C++ 'ta main fonksiyonun yapısı

2.2.3. C++ Program İfadeleri

Program İfadeleri C++ programlarının en temel birimleridir. Aşağıda iki tane ifade gösterilmiştir.

```
cout << "ilk C++ Programı \n";
return 0;
```

İfadeler bilgisayara ne yapması gerektiğini bildirir ve C++ kullanılan ifadelerin büyük çoğunluğu C'deki ifadelerle aynıdır.

Noktalı virgül(;), ifadenin bittiğini gösterir. Eğer ifadenin sonuna noktalı virgül konulması unutulursa derleme sırasında hata mesajı alınır.

Fonksiyon gövdesindeki en son ifade return 0*;* ifadesidir. Bu ifade *main()* fonksiyonuna, kendisini çağırana 0 değerini döndürmesini bildirir. C++' ın eski sürümlerinde, main fonksiyonunun dönüş tipi *void* olarak belirleniyordu ve *return* ifadesine gerek kalmıyordu. Ancak standart C++'ta bu doğru değildir.

2.2.4. C++ Programı İçerisinde Boşluklar

C++ derleyicisi bazı özel karakterleri dikkate almaz; boşluk, satır başı, satır atlama, sekmeler vb. Yukarıda verilen programı aşağıdaki şekilde yazıp tekrar çalıştırırsanız aynı sonucu elde edersiniz. Ancak, bu yazım şekli programın okunabilirliğini zorlaştırmaktadır. Bu nedenle aşağıdaki şekilde program yazılması tavsiye edilmez.

```
#include <iostream>
using namespace std;
int main ()
{ cout <<
"ilk C++ Programı \n"
; return
0;}
```

Boşlukların derleyici açısından önemli olmadığı kuralının birkaç istisnası vardır. *#include* ile başlayan programın ilk ifadesi bir önişlemci direktifi olduğu için tek bir satırda yazılmak zorundadır. Bundan başka *"ilk C++ Programı \n"* gibi karakter katarı sabitleri de ayrı satıra bölünerek yazılmaz. Eğer uzun bir karakter katarı sabitine ihtiyacınız varsa, satır sonuna ters bölü işareti (\) ekleyebilir veya karakter katarını, tırnakla çevrilmiş iki ayrı karakter katarına bölebilirsiniz.

2.2.5. Açıklamalar

Hangi program olursa olsun açıklamalar programın önemli bir parçasıdır. Açıklamalar programı yazana kolaylık sağladığı gibi kaynak dosyanın okunabilirliğini arttır. Okunabilir kaynak program dosyalarına daha sonra bakım yapmanın maliyeti düşük olur. Açıklamaları derleyici dikkate almaz. Açıklamaların dosya büyüklüğüne etkisi yoktur ve çalışma hızını etkilemezler.

Aşağıda C++ programlarında açıklamaların nasıl yapılacağı gösterilmiştir.

```
Tek satırda yapılacak açıklama:
Birden fazla satırda yapılacak açıklama:
/*.....
.....*/
#include <iostream> //ön işlemci direktifi
using namespace std; //using direktifi
/* Bu program ekrana
Ilk C++ Programı" yazar */
int main()
{
cout << "ilk C++ Programı \n|";
return 0;
}
```

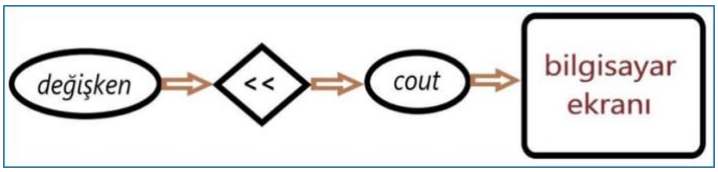
Program içerisinde Açıklama satırı kullanılmasına Örnek vermek için

2.2.6. C++ Programlama Dilinde *cout* Kullanarak Ekran Çıktısı Almak

Cout, aslında bir nesnedir. C++'ta önceden tanımlıdır ve standart çıktı akışına (standart output stream) karşılık gelir.

<< operatörüne ekleme operatörü denir.

<< operatörü, sağ taraftaki değişkenin değerini sol taraftaki nesneye yönlendirir.



Şekil 2.2. cout nesnesinin çalışma şekli

2.2.7. Karakter Katarı Sabitleri

Tırnak içerisinde yer alan *"ilk C++ Programı \n "* deyimi karakter dizini sabitlere bir örnektir. Bilindiği üzere değişkenlerden farklı olarak sabitler, program çalıştığında yeni bir değer almazlar. Sabit değeri program yazılırken belirlenir ve program çalıştığı sürece sabitler değerlerini korur.

Karakter dizininin sonundaki *"\n"* karakterleri, bir satır atlatılır. Burada, *"\n"* ekran çıktısının bazı derleyicilerin program sonunda verdiği mesajlarla karışmaması için konulmuştur.

2.3. Direktifler

Yukarıdaki programın başında yer alan ilk iki satır birer direktiftir. Bunlardan ilki ön işlemci direktifi ikincisi ise using direktifidir. Direktifler temel C++ programının bir parçası değildir ama gereklidirler.

2.3.1. Ön İşlemci Direktifleri

Programdaki *#include <iostream>* bir ön işlemci direktifidir.

Ön işlemci, derleyiciye verilen bir komuttur. Derleyicinin ön işlemci denilen bir parçası gerçek derleme işlemi başlamadan önce bu direktiflerle ilgilenir.

Ön işlemci direktifi olan #include derleyiciye, kaynak dosyası içine başka bir dosya eklemesi gerektiğini bildirir.

Programın derleme işlemi tamamlandığında *#include* direktifinin yerine belirtilen dosya içeriği yerleştirilir.

#include, pek çok ön işlemci direktifinden yalnızca bir tanesidir. Ön işlemci direktiflerinin kullanımı C++'da C kadar yaygın değildir.

2.3.2. using Direktifi

Bir C++ programı değişik isim uzaylarına(namespace) bölünebilir. İsim uzayı, programdaki belirli isimlerin derleyici tarafından tanınmasını sağlayan program parçasıdır. İsim uzayının dışında bu isimler tanınmaz.

```
using namespace std;
```

Bu direktifi takip eden tüm program ifadelerinin, *std* isim uzayında olduğunu belirtilir.

cout bu isim uzayında tanımlıdır. Eğer *using* direktifi kullanılmamış olsaydı her cout kullanıldığında cout 'un başına *std* kelimesini eklemek gerekecekti.

```
std::cout<<"ilk C++ Programı \n .";
```

Defalarca std:: yerine using direktifini bir defa yazmak bir çok açıdan daha yaralıdır.

2.3.3. #define Direktifi

C++ programlama dili de kullanımı tavsiye edilmemekle birlikte sabit tanımı # define ile tanımlanabilir. const'an farklı olarak #define ile yapılan sabit tanımlarında tip belirtilmez.

```
#define pi=3.141618
```

2.3.4. const Niteleyici

Sabit tanımlamak için kullanılır ve değişkenin değerinin program sonuna kadar değişmeyeceğini bildirir. Bu tür değişkenlerin içeriğini değiştirme girişiminde bulunulduğunda hata mesajı ile karşılaşılır.

```
const float pi=3.141618;
```

2.3.5. Başlık Dosyaları

#include ön işlemci direktifi derleyiciye programı derlemeden önce *iostream* dosyasını programa eklemesini bildirir. *iostream,* en temel giriş/çıkış işlemlerinden sorumludur. *cout* ve << operatörü için gerekli tanımları içerir. Bu tanımlar olmadan derleyici, *cout* 'u tanıyamaz. Buna benzer birçok başlık dosyası vardır. Güncel C++ başlık dosyalarından dosya uzantısı kaldırılmıştır. Ancak, C günlerinden kalma bazı eski başlık dosyaları.h uzantısı içerir. *iostream* içerisinde ne olduğunu merak ediyorsanız derleyicinin *include* dizininde bu kaynak dosyayı *notepad* ile açıp inceleyebilirsiniz.

2.3.6. iomanip Başlık Dosyası

endl manipülatörü dışındaki diğer manipülatörlerin tanımları, *iostream* başlık dosyasında değil, *iomanip* isimli başka bir başlık dosyasındadır. Bu durumda, setw manipülatörünü kullandığımızda *iomanip başlık dosyasını* da programımıza eklemek zorundayız. Yukarıdaki programda *setw* manipülatörünü kullanmak için *iomanip* programa ilave edilmiştir.

2.3.7 setw Manipülatörü

Ekrana basılacak karakterlerin belirli bir formatta n karakterlik bir alanda yazılmasını sağlar setw(n), Burada n setw(n) fonksiyonunun argümanıdır.

setw(10) kendisinden sonra gelen değişkeni 10 karakterlik alanda yazdırır. Sayılar sağa, metinler sola dayalıdır.

```
#include <iostream>

#include <iomanip> // setw için

using namespace std;

int main()

 {

int not1=50, not2=67,not3=80;

cout<<setw(8)<<"Adi"<<setw(8)<<setw(8)<<"notu"<<endl;

cout<<setw(8)<<"==========================================="<
<endl;

cout<<setw(8)<<"Aysegül"<<setw(8)<<not1<<endl;

cout<<setw(8)<<"Hasan"<<setw(8)<<not2<<endl;

cout<<setw(8)<<"Abdullah"<<setw(8)<<not3<<endl;

cout<<setw(8)<<"==========================================="<
<endl<<endl;

return 0;

}
```

ö	ğ	r	e	n	c	i		p	u	a	n				
A	y	ş	e	g	ü	L								5	0
H	a	s	a	n										6	7
A	b	d	u	l	l	a	h							8	0

setw manipülatörünün çıktıya etkisi

2.4. Değişkenler

Değişkenler, programlama dillerinin en temel parçalarındadır. Her değişkenin sembolik bir ismi vardır. Değişkenler bilgisayarın belliğinde belirli bir bölgede yerleşiktir. Bir değişkene bir değer verildiğinde bu değer gerçekte, bellekte o değişken için ayrılmış olan yere atanır. Programlama dilleri hemen aynı değişken tiplerini kullanır. Örneğin tamsayılar, kayan noktalı sayılar, karakterler vb.

2.4.1. Tamsayı Değişkenler

Tamsayıların kesirli kısmı bulunmaz. Beş(5) kavramını tam sayı kullanarak ifade edebiliriz, fakat beş buçuk(5.5) kavramını tamsayı tipinde değişken kullanarak ifade edemeyiz. Bu tür sayıları ifade edebilmek için farklı veri tipleri vardır.

Tamsayı değişkenler farklı büyüklüklerde olabilir. En yaygın olarak kullanılan *int* tipidir. Tamsayıların bellekte kapladığı yer sisteme bağlı olarak değişmektedir. Örneğin 32 bit Windows sistemlerde bir *int* tipindeki değişken 4 bayt yer kaplar. Bu, int tipindeki bir değişken ile -2,147,483,648 - 2,147,483,647 Aralığındaki sayıları saklayabileceğimizi gösterir. *int* tipi sayılar Windows işletim sisteminin ilk sürümlerinde 2 bayt yer kaplamaktaydı. Aşağıda *int* tipinde birkaç değişken tanımlayan ve kullanan bir program verilmiştir.

```
#include <iostream>
using namespace std;
int main()
{
int sayi1;
int sayi2;
sayi1=20;
sayi2=Sayi1+10;
cout << "sayi1 +10 =";
cout << sayi2 << endl;
return 0;
}
```

Tamsayı tipindeki değişkenlerin C++'ta kullanılması

```
İnt sayi1;
İnt sayi2;
```

Yukarıdaki ifadelerle ile *sayi1* ve *sayi2* adında iki tamsayı değişken tanımlanır. *int* anahtar kelimesi değişkenin tipini belirtir. Deklarasyonlar(declarations) olarak adlandırılan bu ifadeler, diğer ifadelerde olduğu gibi noktalı virgül ile sona ermelidir.

Bir değişken kullanılmadan önce deklare edilmelidir. Değişken deklarasyonunu programın herhangi bir yerinde yapabilirsiniz.

Deklarasyon, bir değişkenin ismini ve tipini programa tanıtır. Bir deklarasyon değişken için bellekte yer de ayırıyorsa aynı zamanda tanımdır. Biz daha çok hem deklarasyon hem de tanım olan ifadelerle ilgileneceğiz. Yukarıdaki verilen programda,

```
İnt sayi1;
İnt sayi2;
```

ifadeleri hem deklarasyon hem de tanımdır.

2.4.2. Değişken İsimleri

Bir C++ programında kullanılacak olan değişkenlerin programın başında, yürütülen deyimlerden önce tanımlanmaları gerekir.

Değişken tanımlanması,

a) bilgisayar belleğinde o değişkenler için yer ayrılmasını sağlar,

b) ilgili bellek gözlerinde sembolik isimler verilmesini sağlar. Değişken İsimleri:

- Harf, rakam ve alt çizgi '_' içerirler.
- Harf veya '_' ile başlamak zorundadır. '_' ile başlaması önerilmez.
- Büyük/küçük harf ayrımı vardır.
- C dilinin anahtar sözcükleri değişken ismi olarak kullanılamaz.
- Özel karakterler ve Türkçe 'ye özgü harfler (ş,ğ,ü,ı,ç) kullanılamaz.

Geçerli isimler:

a1, sayi, sayi_3, ilkDeger, SonSayi, a1c32, deneme_degeri

Geçersiz isimler:

1a1, değer, int

2.4.3. Değer Atama

```
Sayi1=10;
Sayi2=20;
```

Yukarıda verilen ifadeler değişkenlere değer atar. Eşit işareti(=) tahmin edilebileceği gibi, sağdaki değerin soldaki değişkene atanmasını sağlar.

2.4.4. Tam Sayı Sabitler

Sabitler program boyunca değerleri değiştirilmeyen, yani sabit kalan değişkenler olarak ta düşünülebilir.

2.4.5. Çıktı Varyasyonları

```
cout<< "Sayi1 +10 eşittir ";
```

İfadesi, önceden de açıkladığımız gibi, bir karakter katarı sabitini ekrana yazdırır.

```
cout<< sayi2<<endl
```

İfadesi, sayi2 değişkeninin değerini ekrana yazdırır.

Dikkat edilecek olursa cout ve ekleme(<<) operatörü, tamsayı ve karakter katarı sabitlerini farklı şekilde ele alır. Bu operatörlere karakter katarı gönderilirse, karakter katarını olduğu gibi ekrana basar. Tamsayı gönderilirse, ekrana tamsayının değerini basar.

2.4.6. endl Manipülatörü

endl manipülatörü çıktıda satır atlamasını sağlar. Bu şekilde, bunu takip eden metin bir sonraki satırda görüntülenir. endl manipülatörü \n karakterleri ile aynı etkiye sahiptir ve bir manipülatör örneğidir. Manipülatörler çıktı akışını değişik yollardan değiştiren komutlardır.

2.4.7. Diğer Tamsayı Tipleri

int tipinin yanı sıra iki tane tam sayı tipi daha vardır. Bunlar long, short tipleridir. int tipinin büyüklüğünün siteme bağlı olduğunu daha önce belirtmiştik. Ancak long ve short tamasayı tipleri hangi sistemde kullanılırsa kullanılsın aynı büyüklüğe sahiptirler. long her zaman bellekte 4 byte yer kaplar, short tamsayı tipi ise bellekte 2 Bayt yer kaplar.

2.4.8. Karakter Değişkenleri

char tipi -128 ile 127 arasındaki tamsayıları saklar. Bu tipteki değişkenler bellekte 1 byte (8bit) yer kaplar ve daha çok ASCII karakterleri saklamak için kullanılır. Karakter sabitlerinde bir karakter etrafında tek tırnak işareti kullanılır. Söz gelişi, ‘a’, ‘b’ gibi (Bunun çift tırnak kullanan karakter katarı sabitlerinden farklı olduğuna dikkat etmelisiniz) Aşağıdaki program bazı karakter sabitlerine ve karakter değişkenlerine örnekler içerir:

```
#include <iostream>

using namespace std;

int main()

{

 char karakter1 ='A';

 char karakter2 ='\t';

 cout << karakter1;

 cout << karakter2;

 karakter1= 'B';

 cout << karakter1;

 cout << '\n';

 return 0;

}
```

char tipindeki değişkenlere de tanımlandıkları anda ilk değer atanabilir. Bu programda char tipindeki iki değişkene ilk değer atanmıştır.

Kaçış Sekansı	Karakter
\a	*Zil*
\b	*Backspace*
\f	*Formfeed*
\n	*Newline*
\r	*return*
\t	*sekme*
\\	*Ters bölü*
\'	*Tek tırnak*
\"	*Çift Tırnak*

Kaçış Sekansları

2.4.9. Kayan Noktalı Temel Veri Tipleri

Kayan noktalı değişkenler, sayıları ondalık kısmıyla birlikte simgeler. Bu tür sayıların hem tam kısmı hem de ondalık kısmı vardır.

C++'da kayan noktalı değişkenler; float, double, long double şeklindedir.

float tipi: $3.4x10^{-38}$ ile $3.4x10^{38}$ arasındaki sayıları saklar. Ondalık kısmı noktadan sonra en fazla 7 basamak uzunluğunda olabilir. Bellekte 4 byte kaplar.

double ve long double : float' a göre bellekte daha fazla yer kaplarlar ve daha geniş değer aralığını kapsarlar. 8 byte yer kaplarlar ve noktadan sonra 15 basamak ondalık kısma sahiptirler.

```
#include<iostream>
using namespace std;
int main()
{
 float yaricap;
 const float PI=3.14159F;
 cout<<"Yari çap Giriniz..:";
 cin>>yaricap;
 cout<<endl;
 float alan=PI*yaricap*yaricap;
 cout<<alan<<endl;
 return 0;
}
```

2.4.10. Çoklu Tanımlama

```
int not1=50, not2=67,not3=80;
```

Her üç değişken tek satırda bir tek int anahtar kelimesi ile tanımlanmıştır. Bu şekilde yapılan tanımlamalarda değişkenler birbirinden virgül ile ayrılır. Bu şekilde yapılan tanımlamalara çoklu tanımlama denir.

2.4.11. Değişken Tiplerinin Özeti

Temel C++ Değişken Tipleri				
Anahtar Kelime	Alt Sınır	Üst Sınır	Maks. Ondalık kıs.	Bellek Alanı (byte)
bool	false	true	yok	1
char	-128	127	yok	1
short	-32,768	32,767	yok	2
int	-2,147,483,648	-2,147,483,647	yok	4
long	-2,147,483,648	-2,147,483,647	yok	4
float	3.4 x10-38	3.4 x 10 38	7	4
double	1,7 x 10 -308	1.7 x 10 308	15	8

İşaretsiz (unsigned) Tamsayı Tipleri				
Anahtar Kelime	Alt Sınır	Üst Sınır	Maks. Ondalık kıs.	Bellek Alanı (byte)
unsigned char	0	255	yok	1
unsigned short	0	65,535	yok	2
unsigned int	0	4,294,967,295	yok	4
unsigned long	0	4,294,967,295	yok	4

Aşağıda verilen ve işaretli-işaretsiz sayıların sınanması için yazılan programı inceleyiniz.

```
//işaretli ve işaretsiz tamsayıların sınanması
#include<iostream>
using namespace std;
int main()
{
 int sayi1=1500000000;
 unsigned int sayi2=1500000000;
 sayi1=(sayi1*2)/3;
 sayi2=(sayi2*2)/3;
 cout<<"isaretli sayi :"<<sayi1<<endl;
 cout<<"isaretsiz sayi :"<<sayi2<<endl;
 return 0;
}
```

2.4.12. Tip Dönüşümü

C++, C 'de olduğu gibi, birkaç veri tipi içeren deyimleri ele alırken diğer dilleri oranla esnek davranır. Aşağıdaki programı inceleyiniz:

```
//Tip dönüşümü-Karışik tipler
#include<iostream>
using namespace std;
int main()
{
 int sayac=7;
 float kutle=155.5F;
 double toplamkutle=sayac*kutle;
 cout<<"Toplam Kutle :"<<toplamkutle<<endl;

 return 0;
}
```

Programda int tipindeki değişken float tipindeki bir değişkenle çarpılıyor ve sonuç double bir değişkene atanıyor. Birçok programlama dili bu tür işlemlere sınır koyar. Ancak C++ bu satırı hatalı olarak işaretlemez ve program çalışır.

Bazı durumlarda tip dönüşümleri otomatik olarak gerçekleşir. Derleyici karışık tipli deyimlerle yüz yüze geldiğinde tipler, aşağıdaki tabloda gösterilen sıralamaya göre ele alınır.

Veri Tiplerinin Sıralanması	
Veri Tipi	Sıralama
long double	En Yüksek
double	
float	
long	
int	
short	
char	En Düşük

+ ve * gibi aritmetik operatörler aynı tip operandlar üzerinde işlem yaparlar. Aynı deyim içerisinde farklı tipte iki operand karşı karşıya gelirse, düşük tipteki değişken yüksek tipteki değişkenin tipine dönüştürülür ve işlem sonra yapılır.

2.4.13. cin İle Giriş Almak

cin anahtar kelimesi standart giriş akışına karşılık gelen, C++ da önceden tanımlı bir nesnedir. >> operatörü, extraction(çıkarma) operatörüdür. Solundaki akış nesnesinden değeri alır, sağındaki değişkene yerleştirir. Aşağıdaki, programda cin nesnesi ve çıkarma operatörünün kullanımı gösterilmiştir.

```
#include <iostream>

using namespace std;

int main()

{

 int ftemp;

 cout << "fahrenheit sicaklik Giriniz : ";

 cin >> ftemp;

 int ctemp= (ftemp-32) * 5/9;

 cout << " derece : " <<ctemp << endl;

 return 0;

}
```

2.4.14. Kullanım Anında Değişken Tanımlama

Yukarıdaki programda *ctemp* değişkeni programın en başında değil, kullanılacağı yerde tanımlanmıştır. Önceden de belirtildiği gibi C++ ta değişkenler programın başında tanımlanabildiği gibi kullanılacağı yerde de tanımlanabilir. Değişkenleri kullanıldığı yerde tanımlamak program listesini okumayı kolaylaştırır.

2.4.15. Basamaklanmış ekleme (<<) Operatörleri

Yukarıdaki programda ikinci *cout* 'a dikkat ederseniz bu satırda birden fazla << operatörü kullanılmıştır. Bu satır, birden fazla *cout* kullanılarak yapılabileceği gibi yukarıdaki gibi aynı ifade içerisinde birden fazla ekleme operatörü yazarak da yazılabilir. Bu şekilde yazılmış ifadelere Basamaklanmış Ekleme Operatörü denir.

2.4.16. Öncelikler

*(ftemp-32) * 5/9*

Deyimindeki parantezlere dikkat edin. Parantezler olmasaydı * operatörü – operatöründen daha yüksek önceliğe sahip olduğu için ilk önce çarpma işlemi yapılacaktı. Parantezler olduğu için ilk önce çıkarma sonra çarpma işlemi yapılacaktır. * ve / operatörleri arasındaki öncelik ise, * ve / aynı öncelik derecesine sahip olduğu için sol taraftaki işlem, yani önce çarpma işlemi yapılacaktır. İşlem öncelikleri ile ilgili kuraların matematikteki öncelik sırasına uygun olduğuna dikkate edin.

2.5. Operatörler

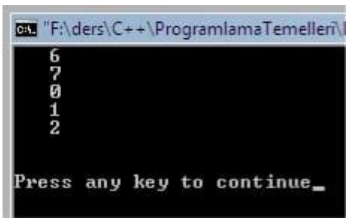
2.5.1. Aritmetik Operatörler

C++'ta dört tane normal aritmetik operatör kullanılır. Toplama için +, çıkarma için -, çarpma için * ve bölme için / kullanılır. Bu operatörler hem tamsayı hem de kayan noktalı sayı tiplerine uygulanır. Kullanımları hemen hemen diğer dillerdeki kullanımları ile aynıdır. Bununla birlikte kullanımları çok açık olmayan başka operatörler de vardır.

Yukarıda sayılan 4 operatörün dışında sadece tamsayı değişkenler üzerinde işlem yapan beşinci bir operatör daha vardır. Buna kalan operatörü (Modül operatörü) denir ve % işareti ile simgelenir. Bölme işleminden kalanı hesaplar. Aşağıdaki programı inceleyiniz.

```
#include<iostream>
using namespace std;
int main()
{
 cout <<" "<< 6%8 << endl
 <<" "<< 7%8 << endl
 <<" "<< 8%8 << endl
 <<" "<< 9%8 << endl
 <<" "<< 10%8 << endl <<endl << endl ;
 return 0;
}
```

Programın çalıştırılması sonucunda elde edilecek çıktı aşağıda verilmiştir.



2.5.2. Aritmetik Atama Operatörleri

C++ yazdığınız kodun daha kısa ve daha okunabilir yazabilmek için birçok yöntem sunar. Bu yöntemlerden birisi aritmetik atama operatörüdür. C++'ta beş adet aritmetik atama operatörü vardır.

```
toplam=toplam+sayi;
```

İfadesini atama operatörü kullanarak aşağıdaki gibi yazabiliriz.

```
toplam +=sayi;
```

Aşağıdaki programı ve çıktısını inceleyiniz.

```
#include<iostream>

using namespace std;

int main()

{

 int sayi=30;

 sayi +=10;

 cout << sayi<<endl;

 sayi-=5;

 cout << sayi<<endl;

 sayi*=2;

 cout << sayi<<endl;

 sayi/=3;

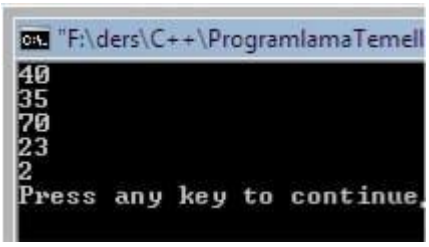
 cout << sayi<<endl;

 sayi%=3;

 cout << sayi<<endl;

 return 0;

}
```



2.5.3. Arttırma ve Azaltma Operatörleri

Yukarıda bir değişkenin değerini bir arttırmak için çeşitli yollar açıkladık. Fakat bu işlemi daha kısa yoldan yapmanın yolu arttırma operatörünü kullanmaktır. Arttırma operatörü yanında bulunduğu değişkenin değerini bir arttırır. Azaltma operatörü ise yanında bulunduğu değişkenin değerini bir azaltır. Aşağıdaki örneği dikkatlice inceleyiniz. Arttırma veya eksiltme operatörünün, değişkenin önünde veya sonunda olmasının sonuca nasıl etki ettiğini tartışınız.

```
#include <iostream>
using namespace std;
int main() {
    int a,b,c,d=10,e=10,f=10,g=10;
    a=++d;   //önce a'nın değeri bir artar sonra atama işlemi yapılır.
    cout<<" a=++d > "<<a<<endl;
    a=e++;  //Önce atama işlemi yapılır. Sonar a'nın değeri bir artar.
    cout<<" a=e++ > "<<a<<endl<<endl;

    a=--f; //Önce f'nin değeri bir eksilir Sonra atama işlemi yapılır.
    cout<<" a=--f > "<<a<<endl;
    a=g--; //Önce atama işlemi yapılır. Sonra g'nin değeri bir azalır.
    cout<<" a=g-- > "<<a<<endl;
    return 0;
}
```

Yukarıdaki programın ekran çıktısı aşağıdaki gibi olur:

```
a=++d > 11    a=--f > 9
a=e++ > 10    a=g-- > 10
```

2.5.4. Karşılaştırma Operatörleri(İlişkisel Operatörler)

Karşılaştırma operatörleri genellikle karar yapılarında ve döngülerde kullanılır ve iki sayıyı ve iki karakteri karşılaştırır. C++'ta altı adet karşılaştırma operatörü(ilişkisel operatör) vardır.

Küçüktür Operatörü (<)	Büyük Eşittir Operatörü (>=)
Büyüktür Operatörü (>)	Eşit Operatörü (==)
Küçük Eşittir Operatörü (<=)	Eşit Değil Operatörü (!=)

2.5.5. Mantıksal Operatörler

İki veya daha fazla koşul karşılaştırılırken kullanılan operatörlerdir.

Operatör	Operatörün İşlevi
Ve Operatörü (&&)	Karşılaştırılan ifadelerin tamamı doğru ise bir aksi halde sıfır döndürür
Veya Operatörü (‖)	Karşılaştırılan ifadelerden biri doğru ise bir aksi halde sıfır döndürür
Değil operatörü (!)	Kendisinden sonra gelen ifadenin tersini alır. Bir ise sıfır, Sıfır ise bir döndürür.

Bölüm Özeti

Bu bölümde, dönem boyunca, bu ders kapsamında C++ programlama dili ile geliştireceğimiz programlar için ihtiyaç duyacağımız editör, derleyici, hata ayıklayıcı vb. özellikler sunan tümleşik geliştirme ortamı(IDE) sunan DEV C++ aracını nasıl temin edeceğimizi ve bilgisayarlarımıza nasıl kuracağımızı öğrendik.

Buna ek olarak, C++ programlarının büyük kısmının fonksiyonlardan oluştuğunu ve en temel fonksiyonun main() fonksiyonun olduğunu öğrendik ve basit bir C++ programı yazarak, main fonksiyonunun yapısını inceledik.

Bölümün devamında, program ifadelerini, cout kullanımını, ekleme operatörünü, cin kullanımını, çıkarma operatörünü ve direktifleri inceledik. Okunabilirliği arttırabilmek için, program içerisine açıklama satırları yazılması gerektiğini öğrendik.

Daha sonra bir programlama dilinin en temel parçası olan değişkenleri, sabitleri, C++'ta kullanılan temel veri tiplerini ve operatörleri inceledik.

Kaynakça

ÜNİTE SORULARI

1. C++'ta yazılan programlarda kaynak dosyaların uzantısı aşağıdakilerden hangisidir?

- .exe
- .cpp
- .com
- .docx
- .xlsx

2. C++ programlarının çalışması program içerisindeki hangi noktadan başlar?

- #include<iostream>
- Using namespace std;
- main() fonksiyonu
- return 0;
- cout

3. C++ programlarında, program ifadelerinin sonuna hangi noktalama işareti konulur?

- İki nokta üst üste
- Diyez işareti
- Soru işareti
- Virgül işareti
- Noktalı virgül işareti

4. C++ programlarında, program içerisindeki tek satırlık açıklama satırları aşağıdakilerden hangisi ile başlar?

- %
- #
- ()
- //
- ??

5. Aşağıdakilerden hangisi cout nesnesi ile birlikte kullanılan ekleme operatörüdür?

- %
- &
- >>
- <<
- -=

6. Aşağıdakilerden hangisi derleyiciye, kaynak dosyası içerisine başka bir dosya eklenmesi gerektiğini bildirir?

- İnclude
- define
- using
- cout
- cin

7. Aşağıdakilerden hangisi hatalı bir değişken adıdır?

- A
- A1
- Sayi_3
- 1A
- A1c32

8. Aşağıdakilerden hangisi cin nesnesi ile birlikte kullanılan çıkarma operatörüdür?

- %
- &
- >>
- <<
- -=

9. Aşağıdakilerden hangisi C++ programlarında sabitleri tanımlamak için kullanılır?

- int
- long
- const
- float
- define

10. Aşağıdakilerden hangisi karşılaştırma operatörüdür?

- ++
- –
- +=
- =
- <=
    -
    - CEVAP ANAHTARI1. b2. c3. e4. d5. d6. a7. d8. c9. c10. e

3. KARAR YAPILARI (IF, IF….ELSE)

Giriş

Bundan önceki bölümde ele aldığımız problemlerde algoritma, ilk satırdan başlayarak, herhangi bir satır atlanmadan sonuna kadar işletiliyordu. Anacak, her zaman problemleri çözerken algoritma içerisinde yer alan ifadeleri sıralı olarak çalıştırmak, o algoritmadan beklenen sonuçların ya da beklenen çıktıların elde edilmesini sağlamaz. Bazı durumlarda algoritmadan beklenen sonuçların ya da beklenen çıktıların elde edilebilmesi için belli koşullar altında bazı ifadelerin çalıştırılması, belli ifadelerin de çalıştırılmaması gerekebilir. Algoritmalarda bu yapıları oluşturabilmek için karar yapıları kullanılır. **C++** programlama dilinde program akışını kontrol etmekte en çok kullanılan yapı **if** yapısıdır. **if** yapısı bütün programlama dillerinde de yapısal farklılık göstermeden yaygın olarak kullanılmaktadır.

Bu bölümde, hemen hemen bütün programlama dilleri tarafından desteklenen ve koşul sağlandığında, çalıştırılacak ifadelerin seçilmesini sağlayan **if** deyimini, koşul sağlandığında ve sağlanmadığı durumda çalıştırılacak ifadelerin ayrı ayrı seçilmesini sağlayan **if… else** deyimini ele alacağız.

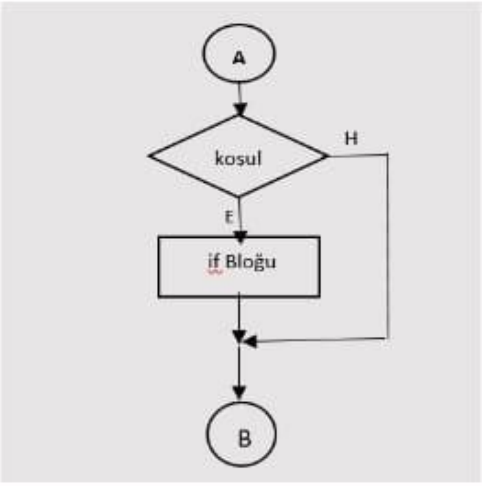
3.1. if Deyimi

if ifadesi, karar yapılarının en basit olanıdır ve programlarımıza karar verme mekanizması eklememizi sağlar. Aşağıda tek ifadeli **if** bloğunun yapısı gösterilmiştir. Bu yapıda, eğer test edilen koşul doğru(true) ise ifade çalıştırılır. Eğer test edilen ifade yanlış(false) ise ifade çalıştırılmaz. Tek ifadeli **if** bloklarında ifadenin küme parantezleri arasına alınması gerekmez. Bu tür yapılarda ifade, küme parantezleri arasına alınsa da program derleme sırasında hata vermez.

Dikkat !! İf bloklarında küme parantezlerinden sonra noktalı virgül yazılmaz.

```
İf (sayi<50)  ├ Koşul deyimi
   İfade;     ├ Tek ifadeli if bloğu
```

Tek ifadeli if yapısının söz dizimi



Şekil 3.1 *if ifadesinin akış şeması*

İpucu: Program içerisinde çalışma sırası if bloğuna geldiğinde önce koşul değerlendirilir. Değerlendirme sonucu doğru(true) ise, if deyiminden sonraki ifade çalışır. Eğer ifade doğru değil(false) ise, if deyiminden sonraki satır çalıştırılmaz.

Bir sinema salonunun bilet gişesinde, bilet satışı yapılırken seyircinin, yaşına göre indirim alabileceği konusunda mesaj veren bir programda, if ifadesi aşağıdaki gibi kullanılabilir.

```
if(yas <= 15)
cout<<"indirimli bilet alabilirsiniz";
```

İf deyiminin kullanımı

<= operatörü, küçük ya da eşit anlamında karşılaştırma(ilişkisel) operatördür. Karşılaştırma(ilişkisel) operatörlerinin ve mantıksal operatörlerin listesi ikinci bölümün sonunda verilmişti. Fakat karar yapılarında, koşulun test edilmesi sırasında, karşılaştırma operatörleri ve mantıksal operatörler çok sık kullanıldıkları için, burada bir defa daha, tablo 3.1'de karşılaştırma(ilişkisel) operatörler, tablo 3.2'de mantıksal operatörler gösterilmiştir.

Operatör Anlamı	
>	Büyük
>=	Büyük eşit
<	Küçük
<=	Küçük eşit
==	Eşit
!=	Eşit değil

Tablo 3.1 *Karşılaştırma(ilişkisel) operatörler*

Operatör	Kullanım Şekli	Anlamı
VE (&&)	(a>b) && (a>c)	(a>b) doğru ve (a>c) doğru(iki önerme de doğru) ise doğru
VEYA (||)	(a>b) || (a>c)	(a>b) doğru veya (a>c) doğru(önermelerden biri doğru) ise doğru
DEĞİL (!)	!(a>b)	(a>b) doğru ise yanlış, (a>b) yanlış ise doğrudur.

Tablo 3.2 *Mantıksal Operatörler*

if yapılarında, koşulun belirlenmesi sırasında, problemin özelliğine göre, birden fazla karşılaştırma(ilişkisel) operatörü kullanılabilir. Bu durumda Karşılaştırma operatörleri, mantıksal operatörlerle gruplanır.

Örneğin, yukarıdaki indirimli bilet satışı probleminde, bilet ücretlerindeki indirimli satışı, altmış beş ve daha büyük yaştaki seyirciler için de isteyebiliriz. Bu durumda koşul parantezi içerisindeki ifadeyi aşağıda gösterildiği şekilde değiştirmemiz gerekir.

```
if( (yas <= 15) || (yas>=65) )
cout<<"indirimli bilet alabilirsiniz";
```

3.1.1. Program

Klavyeden yaşı girilen seyircinin indirimli bilet alma şansı olduğunu(yaş<= 15 veya >=65 ise şansı var) ekran yazan algoritmanın satır algoritmasını, sözde kodunu, akış şemasını ve C++ kodunu yazınız.

3.1.1.1. Satır Algoritması

1. BAŞLA

2. YAZ “Seyircinin yaşını Giriniz: “

3. OKU yas

4. EĞER (yas<=15 || yas>=15)

YAZ “İndirimli bilet alabilirsiniz”

5. SON

3.1.1.2. Sözde Kod

Begin

Print “seyircinin yaşını Giriniz”;

Read yas;

if (yas<=15 || yas>=15)

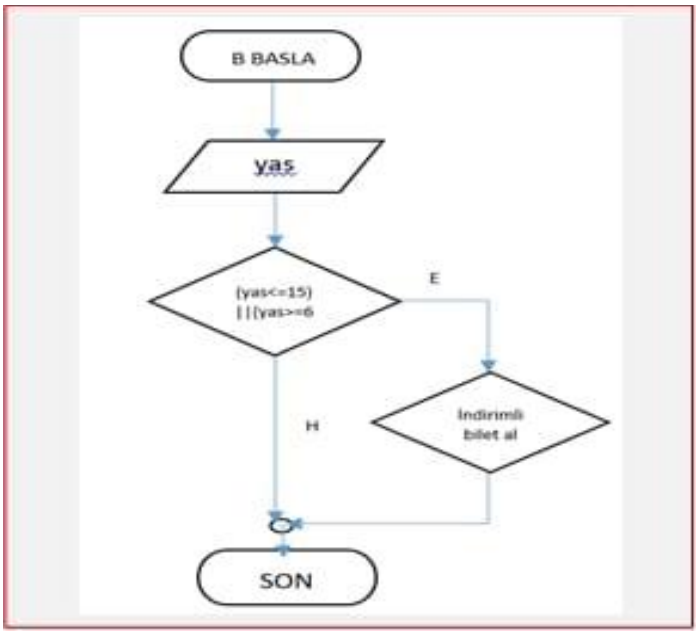
Begin

Print “İndirimli bilet alabilirsiniz.”;

End

End

3.1.1.3. Akış şeması



Şekil 3.2 *Program 3.1. Akış şeması*

3.1.1.4. C++ Kodu

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int yas;
    cout<<"seyircinini Yaşı : ";
    cin>>yas;
    if((yas<=15)||(yas>=65))
    cout<<"indirimli bilet al"<<endl;

    return 0;
}
```

Program 3.1. C++ Kodu

```
seyircinini Yaşı : 12
indirimli bilet al
```

Program 3.1.'in ekran çıktısı

Yukarıda verilen akış şemasından ve C++ kodun da anlışılacağı gibi, karar gösteriminin Evet (E) çıkışında bir ifade bulunmaktadır. Sınama işlemi sonunda doğru sonuç elde edilirse (iki koşulda biri doğru olursa) ifade çalıştırılır(ekrana *indirimli bilet al* yazılır) ve programın çalışması sona erer.

Karar gösteriminin Hayır (H) çıkışında ise, her hangi bir ifade bulunmamaktadır. Koşulun sağlanmadığı, yani klavyeden girilen yaşın en az iki koşuldan birine uymadığı durumda, algoritma herhangi bir çıktı üretmeden(ekrana herhangi bir şey yazmadan) sonlanır.

Bu algoritmada if bloğunda sadece bir ifade bulunmaktadır. Bu sebeple ifade blok parantezleri içerisine alınmamıştır. Eğer ifade, blok parantezleri içerisine alınsaydı da program hatalı olmayacak ve çalıştırıldığında aynı sonucu verecekti.

3.1.2. Program 3.2

Klavyeden girilen bir sayının 50'den küçük olması durumunda ekrana, "girdiğiniz sayı 50'den küçük" yazdıran algoritmanın satır algoritmasını, sözde kodunu, akış şemasını ve C++ kodunu yazınız?

3.1.2.1. Satır Algoritması

6. BAŞLA
7. YAZ "Bir Sayı Giriniz: "
8. OKU sayi
9. EĞER (sayi<50)

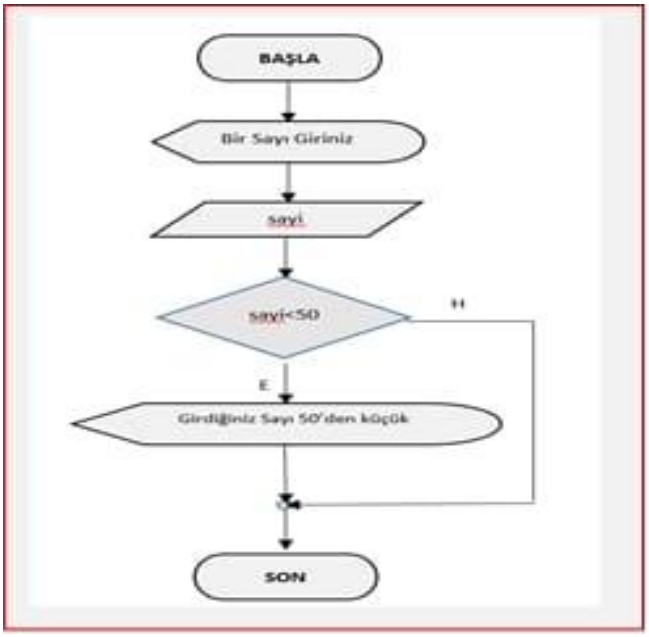
   YAZ "Girdiğiniz Sayı 50'den Küçük."
10. SON

3.1.2.2. Sözde Kod

```
Begin
 Print "Bir Sayı Giriniz";
 Read sayi;
 if (sayı<50)
 Begin
 Print "Girdiğiniz Sayı 50'den Küçük.";

End
 End
```

3.1.2.3. Akış Şeması



Şekil 3.3. *Program 3.2 akış şeması*

Yukarıda verilen akış şemasından da görülebileceği gibi karar gösteriminin Evet (E) çıkışında bir ifade bulunmaktadır. Sınama işlemi sonunda doğru sonuç elde edilirse (Klavyeden girilen sayı 50'den küçük ise) ifade çalıştırılır(ekrana *Girdiğiniz sayı 50'den küçük* yazılır) ve sonra programın çalışması sona erer. Karar gösteriminin Hayır (H) çıkışında ise, her hangi bir ifade bulunmamaktadır. Koşulun sağlanmadığı, yani klavyeden girilen sayının 50 veya 50'den büyük olması durumunda bu algoritma herhangi bir çıktı üretmeden(ekrana herhangi bir şey yazmadan) sonlanır.

Bu algoritmada if bloğunda sadece bir ifade bulunmaktadır. Bu sebeple ifade blok parantezleri içerisine alınmamıştır. Eğer ifade blok parantezleri içerisine alınsaydı program hatalı olmayacak ve çalıştırıldığında aynı sonucu verecekti.

3.1.2.4. C++ Kodu

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL, "Turkish");
    int sayi;
    cout<<"Sayi Giriniz : ";
    cin>>sayi;
    if(sayi<50)
       cout<<"Girilen Sayi 50 den küçük\n";
    return 0;
}
```

Program 3.2'in C++ kodu

3.1.2.5. Program Çıktısı

Yukarıda verilen C++ programı çalıştırılıp klavyeden 5 girildiğinde elde edilen çıktı aşağıda verilmiştir. Klavyeden girilen sayı programda belirtilen koşulu sağladığı için **if** bloğundaki ifade çalıştırılmıştır.

```
Sayi Giriniz : 5
Girilen Sayi 50 den küçük
```

Yukarıda verilen C++ programı çalıştırılıp klavyeden 65 girildiğinde elde edilen çıktı aşağıda verilmiştir. Klavyeden girilen sayı programda belirtilen koşulu sağlamadığı için **if** bloğundaki ifade bu defa çalıştırılmamıştır.

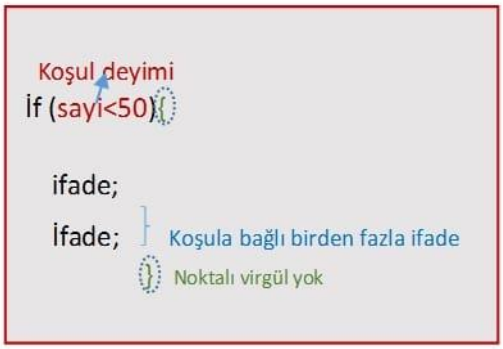
```
Sayi Giriniz : 65
```

```
İpucu: if(sayi<50)
           cout<<"sayı 50 den küçük";
           İfade;
```

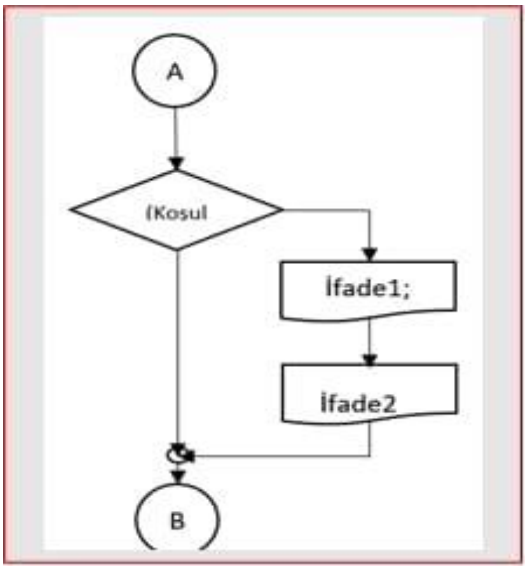
Bu if bloğunda, koşul sağlanmadığı durumda programın akışı doğrudan bloğun sonundaki ifadeye aktarılır.

3.2. if Bloğu İçerisinde Birden Fazla İfade Kullanılması

Önceki kısımda verilen kod örneğinde olduğu gibi **if** gövdesi içerisindeki kod tek bir ifadeden oluşabilir. Koşula bağlı tek olan bu ifadenin küme parantezleri ile sınırlandırılması gerekmez. Ancak, **if** gövdesi içerisinde kod birden fazla ifadeden oluşursa bu durumda, koşula bağlı çalıştırılması gereken ifadeler küme parantezleri içerisine alınmalıdır.



Şekil 3.4. *Birden fazla ifade bulunan if gövdesi söz dizimi*



Şekil 3.5. *Birden fazla ifade bulunan if gövdesi söz dizimi*

3.2.2. Program 3.3

Klavyeden girilen bir sayının 50'den küçük olması durumunda ekrana girilen sayıyı ve "**sayı 50'den küçük**" ifadesini yazdıran algoritmanın sözde kodunu, akış şemasını ve C++ kodunu yazınız.

3.2.2.1. Satır Algoritması

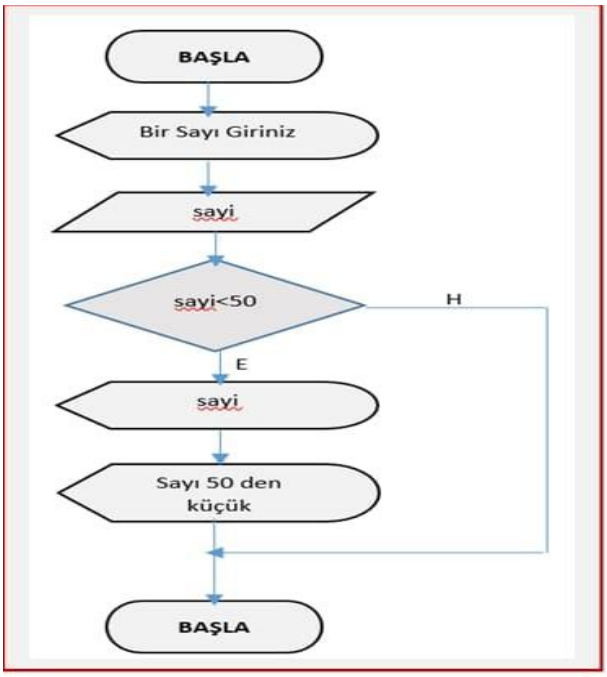
1. BAŞLA
2. YAZ "Bir Sayı Giriniz: "
3. OKU sayi
4. EĞER (sayi<50){

   YAZ "Girdiğiniz Sayı",sayi
   YAZ "sayı 50 'den küçük"}
5. SON

3.2.2.2. Sözde Kod

```
Begin
 Print "Bir sayı Giriniz";
 Read sayi;
 if (sayi<50)
 Begin
 print "Girdiğiniz Sayı",sayi;
 print "sayı 50 'den küçük";
 End
End
```

3.2.2.3. Akış şeması



Şekil 3.6 *Program 3.3'ün akış şeması*

3.2.2.4. C++ Kodu

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL, "Turkish");
    int sayi;
    cout<<"Sayi Giriniz : ";
    cin>>sayi;
    if(sayi<50){
       cout<<"\nGirilen Sayı :"<<sayi<<endl;
       cout<<"Girilen Sayı 50 den küçük\n";
}
    return 0;
}
```

Program 3.3' C++ kodu

3.2.2.5. Program Çıktısı

Yukarıda verilen C++ programı çalıştırılıp klavyeden 5 girildiğinde elde edilen çıktı aşağıda verilmiştir. Klavyeden girilen sayı programda belirtilen koşulu sağladığı için **if** bloğundaki **ifadeler** çalıştırılmıştır.

```
Sayi Giriniz : 5

Girilen Sayı :5
Girilen Sayı 50 den küçük
```

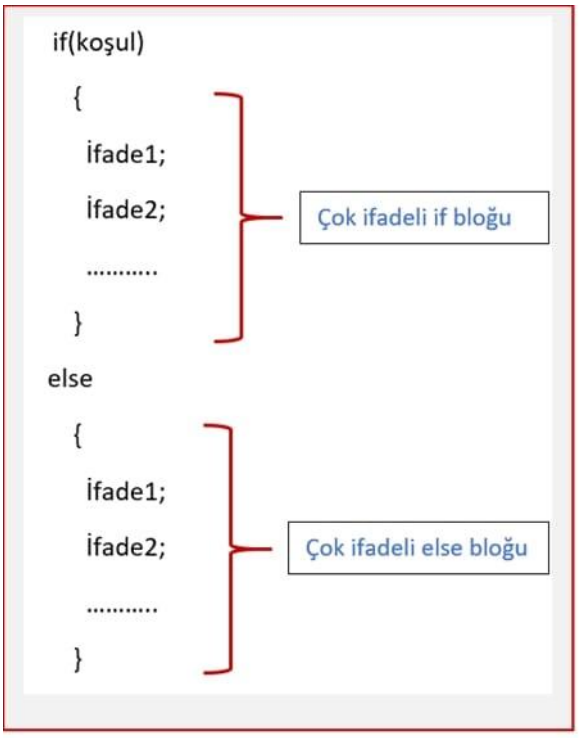
Yukarıda verilen C++ programı çalıştırılıp klavyeden 65 girildiğinde elde edilen çıktı aşağıda verilmiştir. Klavyeden girilen sayı programda belirtilen koşulu sağlamadığı için **if** bloğundaki **ifadeler** bu defa çalıştırılmamıştır.

```
Sayi Giriniz : 65
```

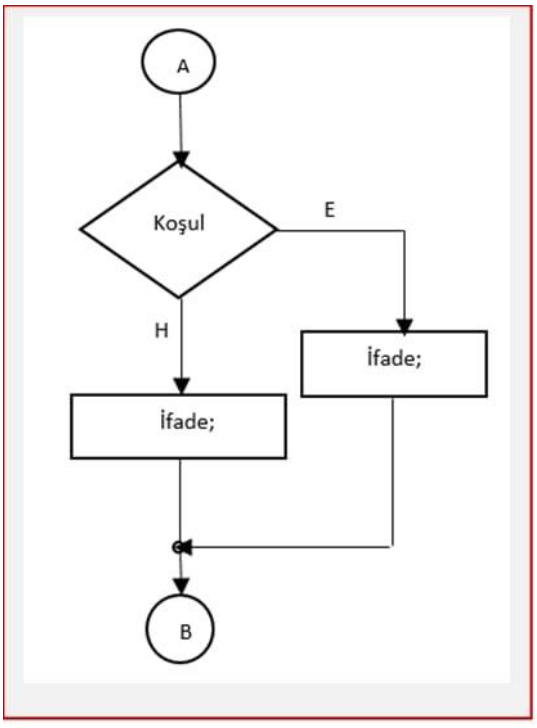
3.3. if…else Yapısı

Basit karar yapısının(if ifadesinin) yer aldığı algoritmalarda koşul sağlandığı durumda çalıştırılacak ifadeler yer almaktadır. Ancak, bazen bilgisayarda problemleri çözerken hem koşulun sağlandığı, hem de koşulun sağlanmadığı

durumlarda çözüm sunmamız gerekebilir. Örneğin, klavyeden girilen iki sayının eşit olması durumunda ekrana "**Girilen Sayılar Eşit**", sayıların eşit olmaması durumunda ekrana "**Girilen Sayılar Eşit Değil**" yazan algoritma **if….else** ifadesine iyi bir örnektir.



Şekil 3.7 *if….else söz dizimi*



Şekil 3.8 *if….else akış şeması*

3.3.1. Program 3.4

Klavyeden girilen iki sayının eşit olması durumunda ekrana "**Girilen Sayılar Eşit**", sayıların eşit olmaması durumunda ekrana "**Girilen Sayılar Eşit Değil**" yazan algoritmanın sözde kodunu, akış şemasını ve C++ kodunu yazınız.

3.3.1.1. Satır Algoritması

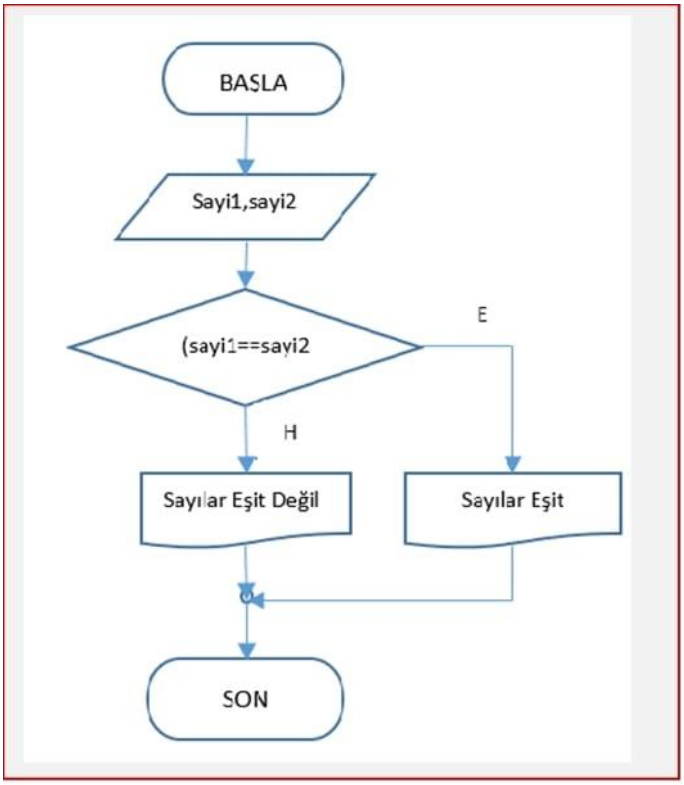
1. BAŞLA
2. YAZ "İki Sayı Giriniz: "
3. OKU sayi1,sayi2
4. EĞER (sayi1==sayi2)

   YAZ "Girilen Sayılar Eşit"
   ELSE
   YAZ "Girilen Sayılar Eşit Değil"
5. SON

3.3.1.2. Sözde Kod

```
Begin
 Print "İki Sayı Giriniz";
 Read sayi1,sayi2;
 if (sayı1==sayi2) then
 Begin
 Print "Girilen Sayılar Eşit";
 else
 Print "Girilen Sayılar Eşit Değil";
 End
End
```

3.3.1.3. Akış Şeması



Şekil 3.9. *Program 3.4'ün Akış şeması*

3.3.1.4. C++ kodu

```
#include <iostream>
#include<locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int sayi1,sayi2;
    cout<<"Birinci Sayi :";
    cin>>sayi1;
    cout<<"İkinci  Sayi :";
    cin>>sayi2;
    if(sayi1==sayi2)
    {
        cout<<"Sayılar Eşit";
    }
        else
        {
            cout<<"Sayılar Eşit Değil";
        }
    return 0;
}
```

Program 3.4'ün C++ Kodu

3.3.1.5. Program Çıktısı

Yukarıda verilen C++ programı çalıştırılıp klavyeden her iki sayı için de 5 girildiğinde elde edilen çıktı aşağıda verilmiştir. Klavyeden girilen sayılar programda belirtilen koşulu sağladığı için bilgisayar ekranına **"Sayılar Eşit"** yazılmıştır.

```
Birinci Sayi :5
İkinci  Sayi :5
Sayılar Eşit
```

Yukarıda verilen C++ programı çalıştırılıp klavyeden her birinci sayı için 5, ikinci sayı için 6 girildiğinde elde edilen çıktı aşağıda verilmiştir. Klavyeden girilen sayılar programda belirtilen koşulu sağlamadığı için else sonraki ifade çalıştırılmış ve bilgisayar ekranına **"Sayılar Eşit Değil"** yazılmıştır.

```
Birinci Sayi :5
İkinci  Sayi :6
Sayılar Eşit Değil
```

3.4 Basit if Yapısı ile İlgili Örnek Programlar

Örnek 1. Klavyeden girilen bir sayının faktöriyelini alan algoritmanın C++ programını yazınız?

Çözüm:

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int sayi, sonuc = 1, i;
    cout << "Sayıyı Girin : "; cin >> sayi;
    for (i = 1; i <= sayi; i++)
    {
        sonuc = i*sonuc;
    }
    cout << sayi << " sayısının faktoriyeli : " << sonuc;
    return 0;
}
```

```
Sayıyı Girin : 12
12 sayısının faktoriyeli : 479001600
--------------------------------
```

Örnek 2. Klavyeden Girilen Sayının tek mi Çift mi olduğunu hesaplayan ve sonucu ekrana yazdıran algoritmanın C++ kodunu yazınız?

Çözüm:

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

int main() {
    setlocale(LC_ALL,"Turkish");
    int sayi;
    cout<<" Sayı Giriniz : ";
        cin>>sayi;

        if(sayi%2==1)
        {
            cout<<"\n Girilen Sayı Tektir"<<endl;
                }
        else
        {
            cout<<"\n Girilen Sayı Çifttir"<<endl;
                }
    return 0;
}
```

Örnek 3. Klavyeden girilen a sayısının klavyeden girilen b sayısına tam olarak bölünüp bölünmediğini hesaplayan ve sonucu ekrana yazdıran C++ programını yazınız?

```
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int a,b;
    cout<<" Sayı Giriniz : ";
        cin>>a;
    cout<<" Böleni Giriniz : ";
        cin>>b;

        if(a%b==1)
        {
            cout<<"\n Girilen Sayı "<<b<<" e tam olarak bölünmez"<<endl;
                }
        else
        {
            cout<<"\n Girilen Sayı "<<b<<" e tam olarak bolunur"<<endl;
                }
    return 0;
}
```

Çözüm:

Örnek 4. Else ve else if kullanmadan klavyeden girilen iki sayı arasında büyüklük, küçüklük eşitlik durumlarını test edip sonucu ekrana yazdıran C++ programını yazınız?

Çözüm:

```cpp
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int a,b;
    cout<<" ilk Sayıyı Giriniz    : ";
        cin>>a;
    cout<<" ikinci Sayıyı giriniz : ";
        cin>>b;
        if(a>b)
        cout<<"\n Birinci sayı büyüktür"<<endl;
        if(b>a)
        cout<<"\n İkinci sayı büyüktür"<<endl;
        if(a==b)
        cout<<"\n Sayılar eşittir"<<endl;
        return 0;
}
```

Bölüm Özeti

Bundan önceki bölümlerde, yazdığımız programlarda ifadeler, birbiri ardınca sıralı bir akış içerisinde lineer olarak çalışıyordu. Bu bölümde, sıralı olan bu akışı değiştirmek için yeni yöntemler öğrendik.

Sıralı olan akışı değiştirmek için **if** deyiminin kullanılmasını, algoritmaların ifade edilmesi sırasında, satır algoritması yazarken, sözde kod yazarken ve akış şeması çizerken **if** yapısının nasıl ifade edileceğini öğrendik.

Aynı şekilde, Sıralı olan akışı değiştirmek için **if….else** deyiminin kullanılmasını, algoritmaların ifade edilmesi sırasında, satır algoritması yazarken, sözde kod yazarken ve akış şeması çizerken **if….else** yapısının nasıl ifade edileceğini öğrendik.

Bunlara ilave olarak, **if** koşul parantezinde birden fazla önerme olduğu zaman bu önermelerin **mantıksal operatörlerle** nasıl birleştireceğini öğrendik.

ÜNİTE SORULARI

1. C++ programlarında, belli koşullar altında belli ifadelerin çalıştırılması, belli ifadelerin de çalıştırılmamasını sağlayan deyim aşağıdakilerden hangisidir?

- cout
- cin
- return 0;
- endl
- if

2. Aşağıdakilerden hangisi C++ programlarında, koşul sağlandığında ve sağlanmadığı durumda çalıştırılacak ifadelerin ayrı ayrı seçilmesini sağlayan ifadedir?

- if
- main()
- int
- if….else
- else

3. Aşağıdaki şekillerden hangisi akış şemalarında if karar deyiminin ifade edilmesi için kullanılır?

- Silindir
- Eşkenar dörtgen
- dikdörtgen
- Üçgen
- Kare

4. Aşağıda verilen C++ programında kaçıncı satırda hata yapılmıştır?

```
#include <iostream>
int main() {
    int yas=9;
    if(yas>=18);
    cout<<"Reşitsiniz";
    return 0;
}
```

- 1
- 2
- 3
- 4
- 5

5. Aşağıda verilen söz dizimi hangi deyime aittir.

if(koşul)
İfade;

- İf
- İf….else
- include <iostream>
- else
- main()

6. Aşağıdakilerden hangisi C++ programlama dilinde kullanılan karşılaştırma(ilişkisel) operatör değildir?

- >=
- &&
- !=
- ==
- <

7. Aşağıda kaynak kodu verilen C++ programının ekran çıktısı ne olur?

```
#include <iostream>
int main() {
    if(sayi>=100)
        cout<<" sayi yüzden büyüktür";
    return 0;
}
```

- Program çalışır. Herhangi bir çıktı üretmez.
- Yazılan kaynak eksik veya hatalıdır. Program Çalışmaz.
- Sayı yüzden büyük değildir.
- Sayı yüzden büyüktür.
- cout<<”sayı yüzden büyüktür”.

8. Aşağıdakilerden hangisi değildir operatörüdür?

- !
- =
- ==
- I I
- <=

9. Aşağıda verilen söz dizimi hangi deyime aittir.

if(koşul)
İfade;
else
İfade;

- İf
- İf….else
- include <iostream>
- else
- main()

10. Aşağıdaki C++ yazılmış programın ekran çıktısı ne olur?

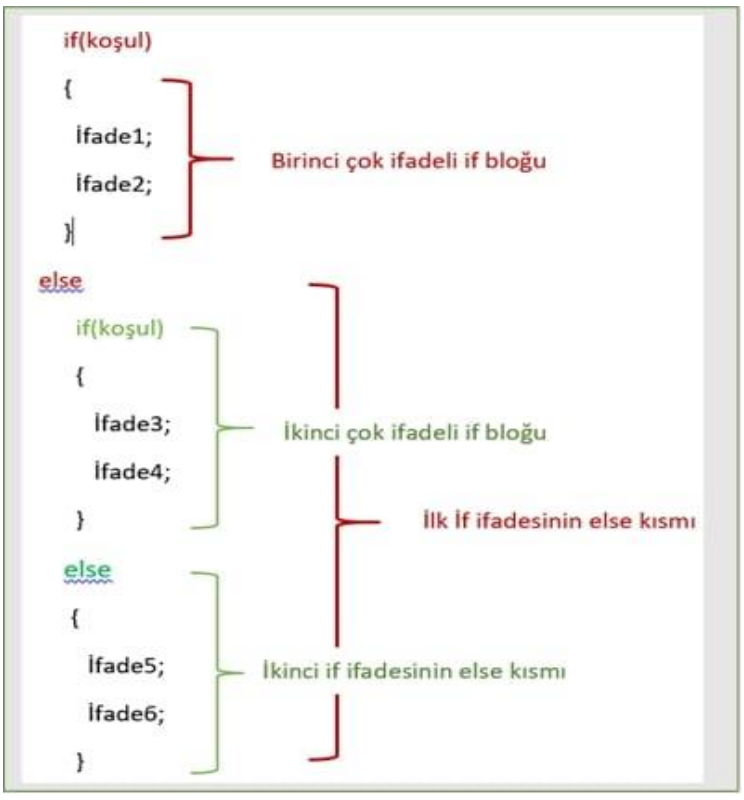
```
#include <iostream>
int main() {
    int sayi=10;
    if(sayi>=10)
    cout<<"sayı ona eşit veya ondan büyüktür";
    else
    cout<<"Sayı ondan küçüktür";
    return 0;
}
```

- Program Çalışır. Herhangi bir çıktı üretmez.
- Program Çalışmaz. Kaynak kod eksik veya hatalıdır.
- Sayı ona eşit veya ondan büyüktür.
- Sayı ondan küçüktür.
- 10
    -
    - CEVAP ANAHTARI1. e2. d3. b4. d5. a6. b7. b8. a9. b10. b

4. KÜMELENMİŞ *IF….ELSE* YAPILARI

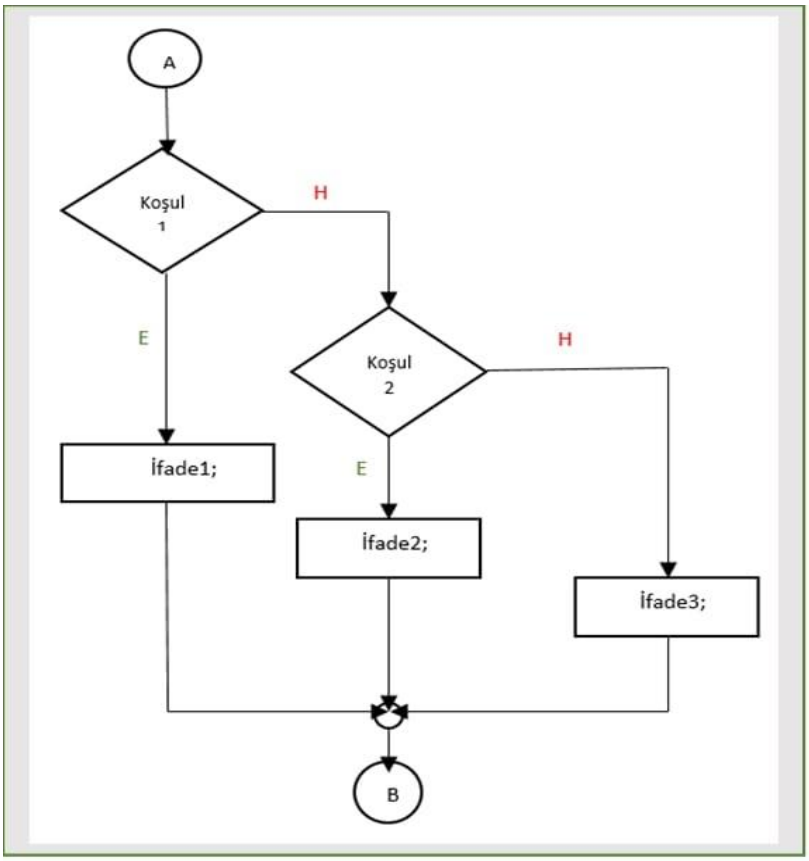
Giriş

Bölüm 3'te ***if*** ve ***if….else*** yapılarını inceledik. Hatırlanacağı gibi ***if….else*** yapısı ile program içerisinde bir koşulun, doğru ya da yanlış olmasına göre değerlendirme yapılabiliyordu. Yani sadece iki farklı durum değerlendirilebiliyordu. Ancak, bazen programlarda, koşula bağlı olarak değerlendirilmesi gereken farklı durum sayısı ikiden fazla olabilir. Örneğin klavyeden girilen üç sayıdan büyük olan sayıyı ekrana yazdıran bir C++ programı yazmamız gerektiğinde ***if….else*** yapısı problemin çözümü için yetersiz kalacaktır. Çünkü bu problemin çözümü için değerlendirilmesi gereken ikiden fazla, üç farklı durum vardır. Bu şekilde problemin çözümü için ikiden fazla durumun değerlendirilmesi gerektiğinde Kümelenmiş ***if….else*** yapıları kullanılır. **Kümelenmiş if** yapısının söz dizimi ve akış şeması aşağıda verilmiştir.



***Şekil 4.1** Kümelenmiş **if….else** yapısı*

İpucu !! *Kümelenmiş if….else yapılarında, if….else içerisine yerleştirilmiş bir if….else, onun da içerisine yerleştirilmiş bir if….else bulunur ve yapı bu şekilde devam eder. Kümelenmiş if else yapılarında ilk sınama koşulu doğru ise programın çalışması doğrudan kümelenmiş bloğun dışındaki ilk ifadeye aktarılır. İlk sınama yanlış(false) ise bir sonraki koşul test edilir. Eğer test edilen bütün koşulların sonucu yanlış(false) olursa o zaman en sondaki else içerisindeki*



Şekil 4.2 kümelenmiş if yapısının blok akış şeması

4.1. Kümelenmiş if….else Yapısı

4.1.1. Program 4.1

Üç sayıdan büyük olanı ekrana yazdıran algoritmanın akış şemasını çiziniz ve C++ programını yazınız?

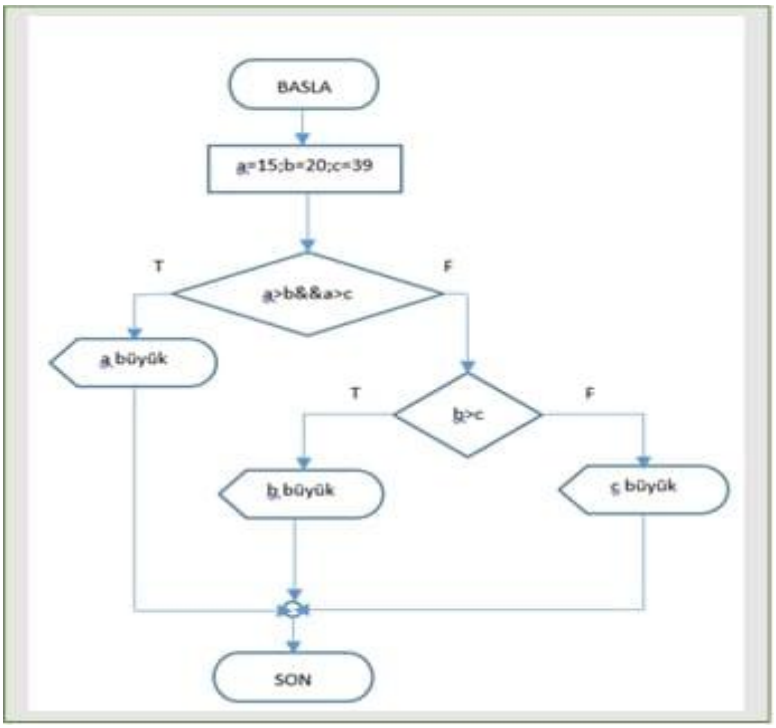
Not: *Program yazılırken a, b ve c sayılarının eşit olmadığı varsayılacaktır.*

Çözüm:

Önce **a** sayısının **b** ve **c** sayılarından büyük olmadığını kontrol edebiliriz (*birinci if*). Eğer a sayısı b ve c sayılarından büyükse program ekrana "*a büyük*" yazacak ve programın çalışması sona erecektir. Eğer a sayısı b veya c sayılarının birinden veya her ikisinden küçük ise bu defa b sayısının c sayısından büyük olup olmadığı araştırılır. Eğer b sayısı c'den büyükse bu durumda ekrana "*b büyük*" yazılır (*else if*). Son olarak, bir önceki değerlendirme sonucunda b sayısı c sayısından büyük çıkmamış ise programın else kısım çalışır ve ekrana "*c büyük*" yazılır.

İpucu !! *İç içe else yapılarında potansiyel bir tehlike vardır. Bu tehlike yanlışlıkla **else'i** yanlış **if** ile eşleme ihtimalidir.*

4.1.1.1. Program 4.1.'in Akış Şeması



Şekil 4.2 *Problem 4.1 akış şeması*

4.1.1.2. Program 4.1.'in C++ Kodu

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int a=15,b=20,c=39;
    if(a>b && a>c)
    {
        cout<<"a büyük\n";
    }
    else if(b>c)

        cout<<"b büyük\n";

        else

            cout<<"c büyük\n";

        return 0;
}
```

Problem 4.1'in C++ kodu

Yukarıda verilen örneğe yakından bakıldığında programda iç içe iki ***if*** yapısı olduğu görülecektir. İkinci ***if*** yapısı birinci ***if*** yapısının bir parçasıdır. Birinci *if* yapısı çalıştırıldığında önce koşul sağlanıp sağlanmadığı kontrol edilir. Koşul saylanıyorsa buradaki ilk ifade çalıştırılır. Koşul sağlanmıyorsa else kısmından sonraki ikinci ***if*** yapısı çalıştırılır. Bu kısım yukarıdaki programda ***else if(b>c)*** ile başlar ve ***return 0;*** kadar devam eder.

4.1.2. Program 4.2

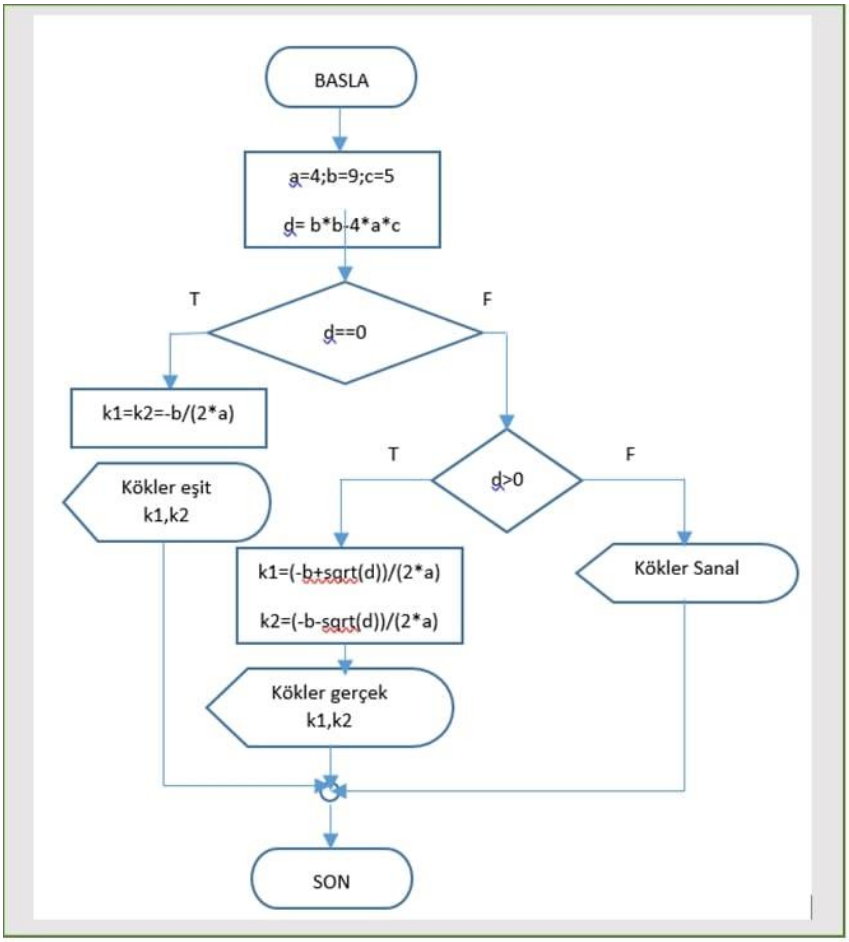
Katsayıları verilen ikinci derece denklemin köklerini hesaplayan programı C++ 'ta yazınız?

Çözüm:

Şimdi yazacağımız program (**program 4.2**), ikinci derece bir denklemin köklerini hesaplayıp ekrana yazdıran bir program olacak. İkinci derece denklemin köklerinin hesabı **kümelenmiş if** yapılarının anlaşılması için iyi bir örnektir.

Bu problemin çözümümde karekök alma işlemi yapılması gerektiği için programa *#include<math.h>* satırı eklenmiştir. Bu başlık dosyası içerisinde matematik fonksiyonları tanımlıdır. Bilindiği gibi ikinci derece bir denklemin köklerini hesaplamak için denklemdeki katsayıların bilinmesi gerekir. Programda bu katsayılar doğrudan atama yoluyla belirlenmiştir.

4.1.2.1 Program 4.2'nin akış şeması



Şekil.4.3 *Problem 4.2'nin akış şeması*

Şekil 4.3'te verilen akış şemasından da görülebileceği gibi önce denklemin diskriminant değeri hesaplanmıştır. Daha sonra diskriminant değerinin sıfıra eşit olup olmadığı test edilmiştir(*birinci if*). Bu karşılaştırmanın sonucu doğru(True) olarak elde edildiği durum için denklemin kökleri k1=k2=-b/(2*a) formülü ile hesaplanmıştır. Bulunan sonuçlar ekrana yazdırılmış ve akış if bloğunun sonuna aktarılmıştır. Bundan sonra karşılaştırmanın sonucu yanlış(False) olarak elde edildiği durum için diskriminantın sıfırdan büyük olup olmadığı test edilmiştir(ikinci if). Bu karşılaştırmanın sonucunun doğru(True) olması durumunda denklemin birbirinden farklı iki kökü vardır. Denklemin kökler k1=(-b+sqrt(d))/(2*a), k2=(-b-sqrt(d))/(2*a formülleri ile hesaplanmış hesaplanan sonuçlar ekrana yazdırılıp programın akışı if bloğunun dışına aktarılmıştır. ***else if(d>0)*** karşılaştırmasının sonucunun yanlış(false) olması durumunda denklemin kökleri sanaldır. Bu durumda programın else kısmındaki ifadeler çalışır ve ekrana "Kökler Sanal" yazılır. Programın akışı if bloğunun sonuna yönlendirilir.

4.1.2.2 Program 4.2'nin C++ Kodu

```
#include <iostream>
#include <locale.h>
#include <math.h>
using namespace std;

int main() {
    setlocale(LC_ALL,"Turkish");
    float b=9,a=4,c=5,d,k1,k2;
    d=b*b-4*a*c;
    if(d==0)
    {
        cout<<"Kökler geçek ve eşit\n";
        k1=k2=-b/2*a;
        cout<<"k1 = "<<k1<<endl;
        cout<<"k2 = "<<k2<<endl;
    }
    else if(d>0)
    {
        cout<<"Kökler geçek ve eşit değil\n";
        k1=(-b+sqrt(d))/(2*a);
        k2=(-b-sqrt(d))/(2*a);
        cout<<"k1 = "<<k1<<endl;
        cout<<"k2 = "<<k2<<endl;
    }
    else
    {
        cout<<"Kökler Sanal\n";
    }
    return 0;
}
```

Problem 4.2' ye ait C++ kodu

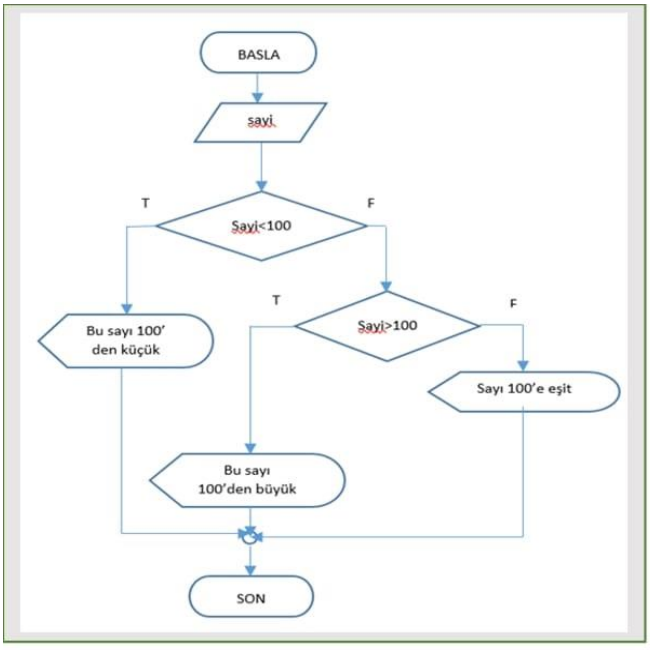
4.1.3. Program 4.3

Klavyeden girilen sayının 100'den küçük, 100'den büyük veya 100' e eşit olduğunu tespit eden ve sonucu ekrana yazdıran C++ programını yazınız?

Çözüm:

Klavyeden sayı girildikten sonra sayının 100'den küçük olup olmadığı kontrol edilecek (*birinci if*). Eğer test sonucunda sayının 100'den küçük olduğu tespit edilirse program ekrana "*Bu sayı 100'den küçük*" yazılmasını sağlayacak ve programın çalışması sona erecektir. Eğer bu ilk testin sonucunda sayının 100'den küçük olmadığı tespit edilirse sayının 100'den büyük olup olmadığı kontrol edilir(*else if*). Eğer sayı 100'den büyük ise program ekrana "*Bu sayı 100'den büyük*" yazılmasını sağlayacak ve programın çalışması sona erecektir. Bu ikinci kontrolde de sayının 100'den büyük olmadığı tespit edilirse programın else kısmı çalışacak ve program ekrana "Sayı 100'e eşit" yazılmasını sağlayacak ve programın çalışması sonlanacaktır.

4.1.3.1. Program 4.3'ün Akış Şeması



Şekil.4.4 *Problem 4.3'nin akış şeması*

4.1.3.2. Program 4.3'ün C++ Kodu

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

int main()
{
    setlocale(LC_ALL,"Turkish");
    int sayi;
    cout << "Bir sayi girin: ";
    cin >> sayi;
    if (sayi < 100)
    {
        cout << "Girdiginiz sayi: " << sayi << endl;
        cout << "Bu sayi 100'den kucuk." << endl;
    }
    else if (sayi > 100)
    {
        cout << "Girdiginiz sayi: " << sayi << endl;
        cout << "Bu sayi 100'den buyuk." << endl;
    }
    else
    {
    cout << "Girdiginiz sayi: " << sayi << endl;
    cout << "Sayılar 100'e eşit'" << endl;
    }

    system("PAUSE");
    return 0;
}
```

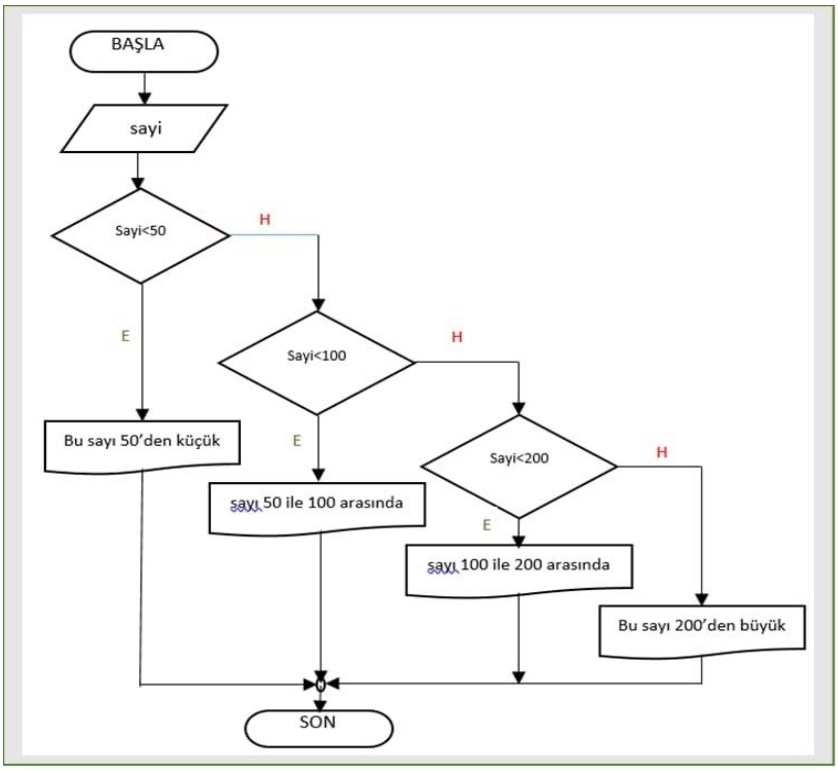
Problem 4.3' ye ait C++ kodu

4.1.4. Program 4.4

Klavyeden girilen bir sayının 50'den küçük veya 50 ile yüz arasında veya 100 ile 200 arasında veya 200'de büyük olduğunu tespit eden ve sonucu ekrana yazdıran algoritmanın akış şemasını çizin ve C++ programını yazınız?

Çözüm:

4.1.4.1. Program 4.4'ün Akış Şeması



Şekil 4.5 *Program 4.4'ün Akış Şeması*

4.1.4.2. Program 4.4'ün C++ Kodu

```
#include <iostream>
using namespace std;
int main()
{
    int sayi;
    cout << "Bir sayi girin: ";
    cin >> sayi;
    cout << "Girdiginiz sayi: " << sayi << endl;
    if (sayi < 50)
        cout << "Bu sayi 50'den kucuk." << endl;
    else if (sayi < 100)
        cout << "Bu sayi 50 ile 100 arasinda." << endl;
    else if (sayi < 200)
        cout << "Bu sayi 100 ile 200 arasinda." << endl;
    else
        cout << "Bu sayi 200'den buyuk." << endl;
    system("PAUSE");
    return 0;
}
```

Program 4.4'ün C++ Kodu

4.1.5. Program 4.5

Bir öğrencinin klavyeden sayısal notu girildiğinde, girilen sayısal notu harf notuna dönüştüren C++ programını yazınız?

```
#include <iostream>
using namespace std;
int main()
{
    short int ogrnot;
    cout<<"not giriniz : =";
    cin>>ogrnot;
    cout<<"\n\n";
    if(ogrnot>= 90)
    cout<<"notunuz : AA\n";
    else if(ogrnot>=85)
    cout<<"notunuz : BA\n";
    else if(ogrnot>=80)
    cout<<"notunuz : BB\n";
    else if(ogrnot>=75)
    cout<<"notunuz : CB\n";
    else if(ogrnot>=65)
   cout<<"notunuz : CC\n";
    else if(ogrnot>=58)
   cout<<"notunuz : DC\n";
    else if(ogrnot>=50)
   cout<<"notunuz : DD\n";
    else
 cout<<"notunuz : FF\n";
  cout<<"\n\n";
  system("PAUSE");
  return 0;
}
```

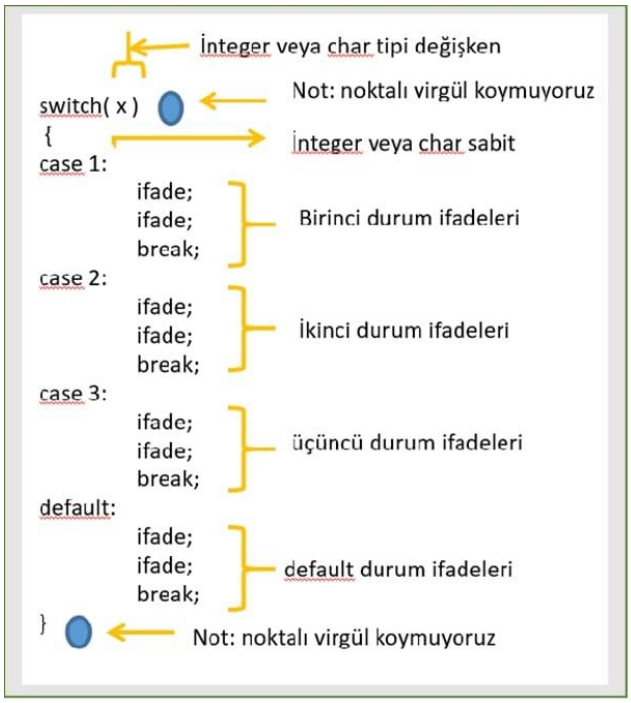
4.2. Switch İfadesi

switch yapısının kullanılması kümelenmiş ***if….else*** veya ***else….if*** yapılarına benzer. Önceki bölümden hatırlanacağı gibi ***if….else*** veya ***else….if*** yapılarında, problemin çözümü için gereken sayıda koşulu iç içe yazarak uygun olan koşula bağlı ifade veya ifadeler çalıştırılıyordu. *Ele alınan problemin çözümü için büyük bir karar ağacı olması ve bütün kararların aynı değişkenin değerine bağlı olması durumda da bu problemler* ***if….else*** *veya* ***else….if*** *yapılarıyla çözülebileceği gibi* ***switch*** *yapısı kullanılarak da çözülebilir.* Hatta bu türden

problemleri ***if….else*** veya ***else….if*** yapılarıyla çözmek yerine switch yapısı ile çözmek kodun okunabilirliğinin artması ve programcıya kod yazarken kolaylık sağlaması açısından avantaj olabilir.

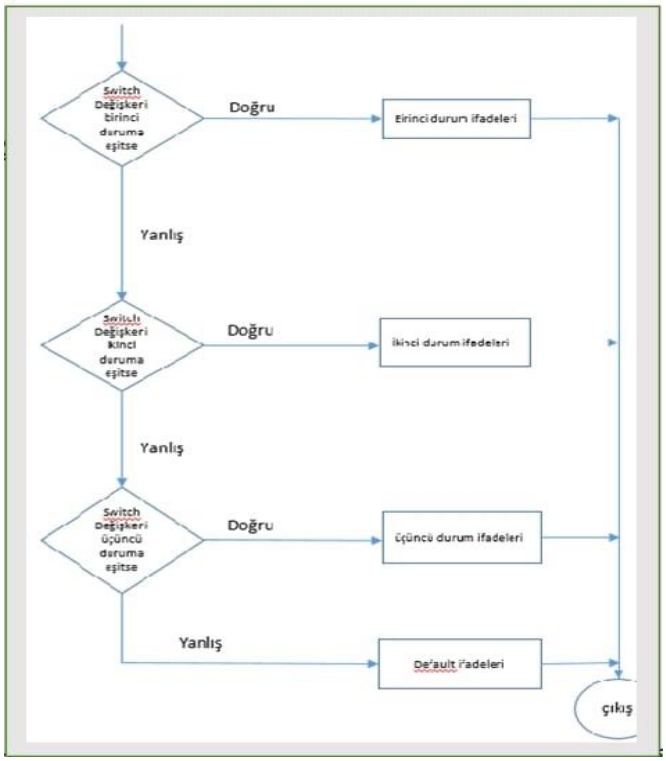
> **İpucu !!** ***Switch*** yapısını, ***if….else*** veya ***else….if*** yerine kullanılacak daha kapsamlı bir yapı düşünmeyiniz. Switch yapısı, daha çok problemin çözümü için büyük bir karar ağacı olması ve bütün kararların aynı değişkenin değerine bağlı olması durumda kullanılır. Switch kullanarak çözülen problemlerin tamamı ***if….else*** veya ***else….if*** yapılarından biri ile çözülür.

Aşağıdaki şekilde ***switch*** yapısı gösterilmiştir.



Switch *ifadesinin söz dizimi*

Aşağıda ***switch*** ifadesinin akış şeması verilmiştir.



Şekil 4.5. switch *ifadesini akış şeması*

switch yapısında, ***switch*** anahtar kelimesinden sonra sınanacak bir değişken gelir. Aşağıda verilen örnekte bu değişkenin sayı olduğu görülmektedir.

switch(sayi)

Programda çalışma sırası switch(sayi) ifadesine geldiğinde parantez içerisindeki değişkenin değeri hesaplanır. Daha sonra elde edilen sonuca göre ***case*** ile başlayan ifadelerden biri çalıştırılır ve bu işlemden sonra programın çalışması ***break*** ifadesi kullanılarak ***switch*** bloğu dışına aktarılır. ***Break*** *ifadesi* ***switch*** yapısından çıkılmasını sağlar. Eğer başlangıçta yapılan hesaplama sonucunda elde edilen değişken değeri herhangi bir **case** değerine eşit olmaz ise programın çalışması, yapının ***default*** bölümüne aktarılır ve bu bölümdeki ifade veya ifadeler çalıştırılarak programın akışı ***switch*** bloğundan sonra gelen ilk ifadeye aktarılır.

İpucu !! ***Switch*** yapısında ***default*** ifadesinin kullanılması zorunlu değildir. Ancak, ***switch*** bloğunun başında hesaplanan değişken değeri yapı içerindeki hiçbir ***case*** değerine eşit olmadığı durumda programın çalışması ***switch*** bloğundan sonra gelen ilk ifadeye aktarılır. Bu durumda ***switch*** bloğunda herhangi bir ifade çalıştırılmamış olur.

Örnek: Klavyeden haftanın gün sırası girildiğinde günü ekrana yazdıran programı **switch** yapısı kullanarak yazınız. Program hatalı gün sırası girildiğinde ekrana hata mesajı vermelidir.

Çözüm:

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int gun;
    cout<<"Haftanın Kaçıncı Günü :";
    cin>>gun;
    switch(gun)
    {
        case 1:
            cout<<"Pazartesi"<<endl;
            break;
        case 2:
            cout<<"Salı"<<endl;
            break;
        case 3:
            cout<<"Çarşamba"<<endl;
            break;
        case 4:
            cout<<"Perşembe"<<endl;
            break;
        case 5:
            cout<<"Cuma"<<endl;
            break;

        case 6:
            cout<<"Cumartesi"<<endl;
            break;
        case 7:
            cout<<"Pazar"<<endl;
            break;
        default:
            cout<<"Hatalı Gün"<<endl;
    }
    return 0;
}
```

İpucu !! her ***case*** değişkeninden sonra bir sabit olmalıdır. Sabitten sonra iki nokta işareti(:) gelir. ***case*** sabitlerinin veri tipleri ***switch*** değişkeninin veri tipi ile aynı almalıdır.

Yukarıdaki örnek çalıştırıldığında ve klavyeden gün sırası girildiğinde günün ismi ekrana yazdırılır. Örneğin klavyeden 6 girildiğinde ekrana *Cumartesi* yazdırılır. Eğer klavyeden 1'den küçük 7'den büyük bir sayı girilirse ekrana "*Hatalı Gün*" mesajı yazdırılır. Yukarıda verilen örneği bilgisayarınızda yazıp klavyeden farklı sayılar girerek çalıştırınız ve sonuçlarını inceleyiniz.

Örnek: *Klavyeden iki sayı ve bir dört işlem operatöründen biri girildiğinde operatöre göre dört işlem yapan ve sonucunu ekrana yazdıran bir C++ programı yazınız. (**switch** yapısını kullanınız).*

```
#include <iostream>
#include<conio.h>
#include<locale.h>
using namespace std;
int main()
{
    setlocale(LC_ALL,"Turkish");
    int sayi1,sayi2,sonuc;
    char islemtipi;
    cout<<"sayı giriniz :";
    cin>>sayi1;
    cout<<"sayı giriniz :";
    cin>>sayi2;
    cout<<"operator giriniz :";
    islemtipi=getche();
```

```
    switch(islemtipi)
        {
        case '*':
            sonuc=sayi1*sayi2;
            break;
        case '/':
            sonuc=sayi1/sayi2;
            break;
        case '+':
            sonuc=sayi1+sayi2;
            break;
        case '-':
          sonuc=sayi1-sayi2;
          break;
        default:
    cout<<"girdiğiniz karakter dört islem tipi değil\n";
        }
            cout<<"sonuc :"<<sonuc<<endl;
  system("PAUSE");
  return 0;
}
```

Yukarıda verilen örneği yazıp çalıştırınız. Verilen örnekte klavyeden iki sayı girildikten sonra, çarpma, bölme, toplama ve çıkarma operatörlerinden biri ***switch*** değişkeni olarak yine klavyeden girilmektedir. Bu örnekte ***switch*** değişkeni tipi **char** olarak seçilmiştir. Bu örnekten ***switch*** yapılarında değişken olarak ***char*** tipinin kullanılabileceği gösterilmiştir.

Klavyeden toplama işareti girildiğinde girilen sayılar toplanır ekrana yazdırılır. Benzer şekilde çarpma, bölme ve çıkarma operatörlerinin girişi yapıldığında ise operatörlere uygun sonuçlar ekrana yazdırılır. Klavyeden bu dört temel operatörün dışında bir karakter girildiğinde ekrana ***default:*** bölümünde verilen hata mesajı yazdırılır.

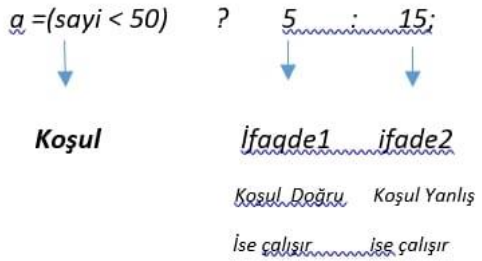
İpucu !! Bir else....if yapısında içinde alakasız değişkenlerin olduğu birçok deyim kullanılabilir. Ancak, switch ifadesinde seçeneklerin her biri aynı değişkene bağlı olmak zorundadır. Bir seçeneği diğerinden ayıran tek özellik değişkenin değeridir. Case sabiti tamsayı veya karakter sabiti olmak zorundadır.

4.3. Koşul Operatörü

Koşul operatörü, üç operand üzerinde işlem yapan iki sembolden meydana gelir. çalışması if….else yapısına benzer. Durum operatörü aşağıdaki şekilde kullanılır.

(ifade1) ? ifade2 : ifade3;

Kuşul operatöründe önce ifad1'in değeri hesaplanır. Eğer bu ifadenin değeri doğru ise ifade2, yanlış ise ifade3 çalıştırılır. Aşağıda koşul operatörünün kullanımını gösteren bir örnek verilmiştir.



Koşul Operatörü için Program:

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

main()
{
    int sayi=-9;
    int a=0;

    a=(sayi>0) ? 5 : 15;

    cout<<"a = "<<a<<endl<<endl;

    return 0;

}
```

Program çalıştırıldığında oluşan ekran çıktısı

```
a = 15
```

Bölüm Özeti

Problemlerin çözümünde koşula bağlı karar verilmesi gereken durumun ikiden fazla olduğu durumda kümelenmiş **if…else** yapılarının nasıl kullanılacağını ve bu problemlere ait algoritmaların akış şemalarının nasıl çizileceğini öğrendik.

Problemin çözümü için büyük bir karar ağacı olması ve bütün kararların aynı değişkenin değeri bağlı olması durumunda, daha çok **switch** deyiminin kullanılması gerektiğini öğrendik. switch deyiminin akış şemasının çizimi ve C++ kodlarının yazılmasını örnekler çözerek pekiştirdik.

Genellikle birçok kaynakta karar yapılarının en son anlatılan konusu koşul operatörü konusudur. Biz de aynı geleneği bozmayarak kümelenmiş if else yapılarının sonunda koşul operatörünü anlattık ve konuyla ilgili bir örnek çözdük.

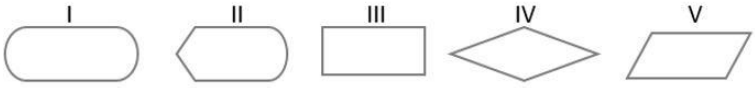
ÜNİTE SORULARI

1. Aşağıdaki şekillerden hangisi akış şemalarında karar ifadesini göstermek için kullanılır?



- I
- II
- III
- IV
- V

2. Aşağıdakilerden hangisi akış şemalarında başla simgesine karşılık gelir?



- I
- II
- III
- IV
- V

3. switch yapılarında her "case :"'dan hemen sonra gelen değer için aşağıdakilerden hangisi doğrudur?

- Sabit değer olmalıdır.
- Değişken değer olmalıdır.
- case 'den sonra gelen değer ile switch değişkeni farklı tipte olmalıdır.
- switch yapılarında "case:" 'dan hemen sonra bir değer gelmez.
- Switch yapılarında "case:" kullanılmaz.
    -
    - CEVAP ANAHTARI1. d2. a3. a

5. DÖNGÜLER

Giriş

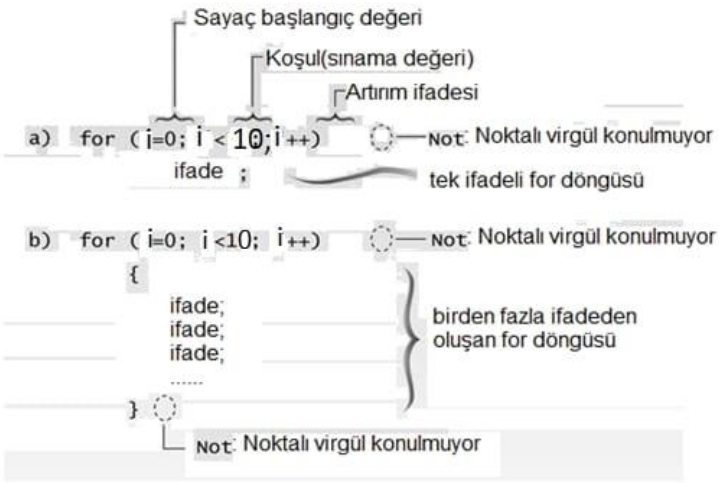
Döngüler, karar yapıları olduğu gibi program içerisinde yapılan hesaplamalara bağlı olarak akışı kontrol eder ve programın bir bölümünün belli sayıda tekrar etmesini sağlar. *Üç ve dördüncü bölümden hatırlanacağı gibi karar yapılarında kodun belli bölümünün çalıştırılıp belli bölümünün de çalıştırılmaması, program içerisinde belli deyimlerin doğru(True) ve yanlış(False) olmasına bağlıydı.* Döngülerde de programın belli bölümünün kaç defa çalışacağı karar yapılarında olduğu gibi program içerisinde belli deyimlerin doğru(True) ve yanlış(False) olmasına bağlıdır. Tekrarlar, her iterasyon sonunda kontrol edilen koşul doğru olduğu sürece devam eder. Koşulun yanlış olduğu noktada döngü biter ve akış döngünün sonundaki ifadeye geçer.

C++' ta for döngüsü, while döngüsü ve do….while döngüsü olmak üzere üç döngü çeşidi vardır.

5.1. for Döngüsü

for döngüsü hemen bütün programlama dillerinde kullanılan bir döngü çeşididir. for döngüsü anlaşılması en kolay döngü çeşitlerinden biridir. for döngünün, bütün döngü kontrol elemanları bir yerde toplanmıştır. Diğer döngü yapılarında kontrol elemanları döngünün dışında, döngünün başında, döngü gövdesinde, döngü sonunda yer alabilmektedir. Bu durum döngülerin anlaşılmasını, kodun okunabilirliğini for döngüsüne göre zorlaştırmaktadır. Buna rağmen ele alınan problemin özelliğine göre for döngüsü dışındaki döngüler de programlarda yağın olarak kullanılmaktadır. Bu bölümde, ilerleyen kısımlarda hangi tür problemlerin hangi döngülerle çözülmesi gerektiği konusunda açıklamalarda bulunulacaktır.

İpucu: for döngüsü genellikle döngüye girmeden önce döngü bloğundaki kodun kaç kere çalıştırılacağı bilindiği durumlarda kullanılır.



Şekil 5.1. *for döngüsü blok yapısı*

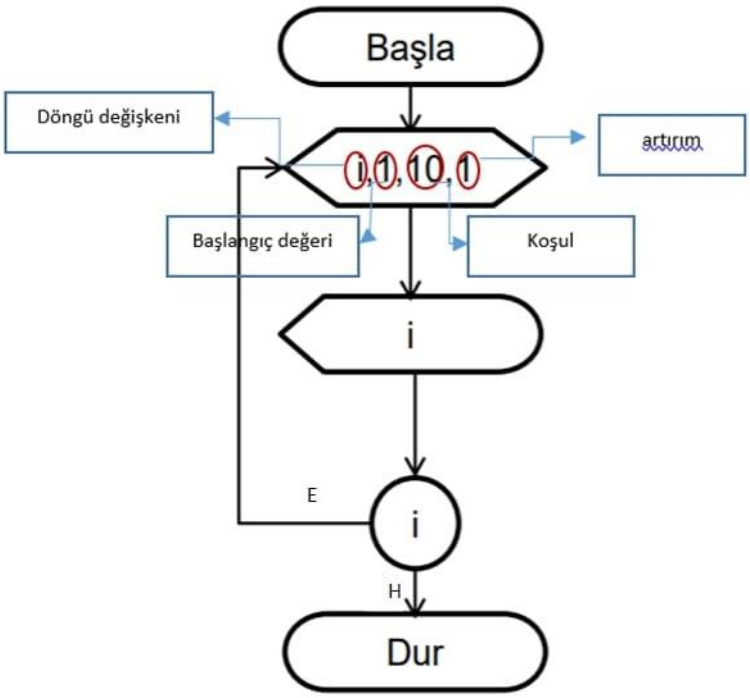
Şekil 5.1'de iki ayrı for döngüsü gösterilmiştir(a ve b). Her iki döngüde de for ifadesi döngüyü kontrol eder. for döngüleri, for anahtar kelimesi ile başlar. for parantezinde ilki *sayaç başlangıç değeri*, ikincisi *koşul* ve üçüncüsü *artırım* olmak üzere üç ifade yer alır. for döngülerinde sayaç başlangıç değeri, koşul ve artırım deyimleri genellikle aynı değişken ile tanımlanır. Değişkenin adı *Döngü Değişkenidir*. Şekil 5.1'de for döngüsünde bu değişken i olarak belirlenmiştir *for(i = 0; i < 10; i ++)*. for döngülerinde döngü değişkeni döngü gövdesindeki ifadeler çalıştırılmadan tanımlanmalıdır.

İpucu: 1. for döngülerinde for parantezinden sonra noktalı virgül konulmaz.

2. for gövdesinde tek bir ifade yer alıyor ise, bu ifadenin küme parantezi içerisine alınması gerekmez. Bu ifade küme parantezi içerisine alınırsa hata olmaz.

3. for gövdesinde birden fazla ifade yer alıyor ise bu ifadelerin küme parantezi içerisine alınması gerekir.

4. Küme parantezlerinin sonuna noktalı virgül konulmaz.



Şekil 5.2. *for döngüsü akış şeması*

5.1.1. for döngüsünün Çalışma Şekli

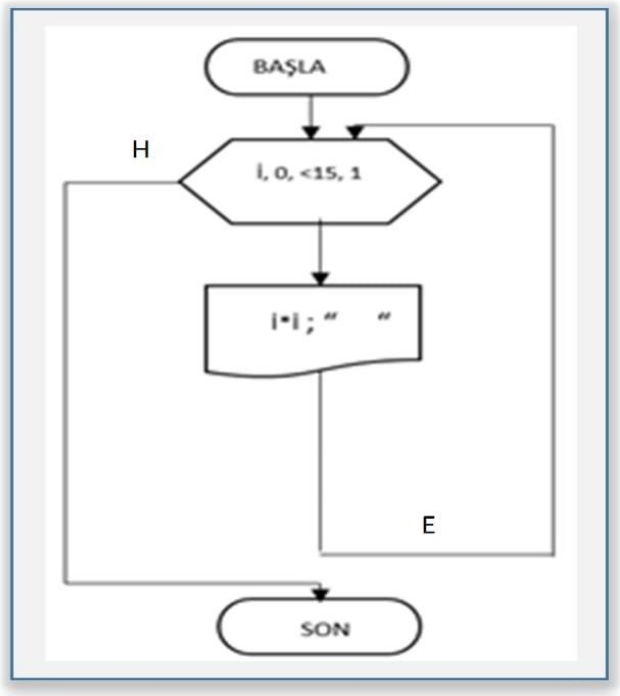
Programın çalışması, for anahtar kelimesine geldiğinde önce sayaç başlangıç değerine atama yapılır. Sonra koşul değeri ile başlangıç değeri test edilir. Eğer test doğru(True) çıkarsa döngü gövdesindeki ifadeler çalıştırılır. Sonra artırım değeri çalıştırılır ve test işlemi tekrarlanır. Sonuç doğru(True) çıkarsa döngü gövdesindeki ifadeler tekrar çalıştırılır. Bu işlem test sonucu yanlış(false) çıkana kadar tekrarlanır. Test sonucu yanlış çıktığında döngüden çıkılır ve programın akışı döngü bloğunu takip eden ilk ifadeye aktarılır.

İpucu: for döngüsüne girişte yapılan ilk testte(Başlangıç değeri koşul ile karşılaştırıldığında), test sonucu yanlış(false) çıkarsa döngü gövdesindeki ifadeler çalıştırılmadan programın akışı döngü bloğunu takip eden ilk ifadeye aktarılır.

5.1.2. Program 5.1

o ile 15 arasındaki sayıların karesini alan ve sonucu ekrana yazdıran algoritmanın akış şemasını çiziniz ve C++ kodunu yazınız.

Çözüm:



Şekil 5.3 *Program 5.1'in akış şeması*

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

int main()
{
    int i;

    for (i = 0; i < 15; i++)
        cout << i * i<<" ";
    system("PAUSE");
    return 0;
}
```

Program 5.1'in C++ kodu

Programın çıktısı aşağıdaki gibidir:

```
0 1 4 9 16 25 36 49 64 81 100 121 144 169 196
```

Programın Çalışması: for parantezinde, döngü değişkeninin başlangıç değeri sıfır olarak atanmıştır. Bu atamadan sonra koşul kontrol edilir. Başlangıç değeri(Sıfır) koşul değerinden küçük(On Dört) olduğu için döngü gövdesindeki ***cout << i * i<<" ";*** çalıştırılır. Ekrana sıfır yazılır. Sonra artırım değeri bir artar(Bir Olur). Koşul test edilir. Koşul sağlandığı için döngü gövdesindeki ifade bir kere daha çalıştırılır. Ekrana bir yazılır. Bu işlem artım değeri on beş olana kadar tekrarlanır. Artım değeri 15'e ulaştığında önceki iterasyonlarda olduğu gibi bu değer koşul ile test edilir ve karşılaştırma sonucu yanlış(False) olduğu için programın çalışması döngü bloğunun sonundaki ifadeye aktarılır.

İpucu: Bir C++ programı içerisinde, çalışma sırası for parantezine geldiğinde;

1. Döngü değişkeni parantez içerisinde tanımlanmış ise önce bu değişkene ilk değer atanır(Başlangıç değeri) ve sonra başlangıç değeri koşul değeri ile karşılaştırılır. Karşılaştırma sonucu doğru(true) olursa for gövdesindeki ifadeler çalıştırılır.
2. for parantezinde başlangıç değeri atama işlemi bir defa gerçekleşir.
3. Döngünün ikinci defa çalıştırılıp çalıştırılmayacağına, for parantezi içerisindeki artırım değeri ifadesi çalıştırılarak ve artırım değerinin yeni değeri ile koşul değeri test edilerek karar verilir. Döngü gövdesindeki ifadelerin tekrar çalıştırılıp çalıştırılmayacağına her defasında bu kontrol yapılarak karar verilir.
4. For döngüsünden çıkış, artım değeri ile koşul değerinin karşılaştırılması sonucu elde edilecek yanlış(false) değerine ulaşıldığında gerçekleşir.

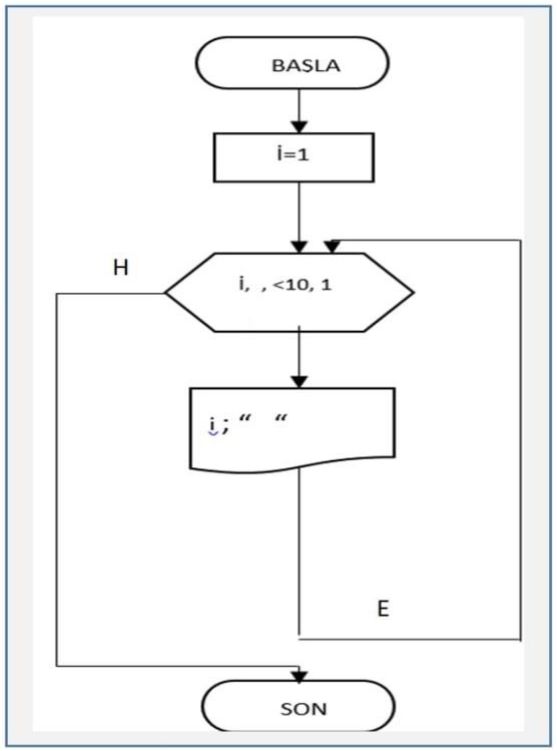
5.1.3. Program 5.2

Birden On'a kadar olan sayıları ekrana yazdıran algoritmanın akış şemasını ve C++ programını yazınız;

Not: döngünün başlangıç değerini for parantezinden önce, for parantezi dışında

tanımlayınız.

Çözüm:



Şekil 5.3 *Program 5.2'in akış şeması*

```
#include <iostream>
using namespace std;
int main()
{
    int i=1;
    for(;i<=10;i++){
        cout << i<<" ";
            }
    cout<<endl;
    system("PAUSE");
    return 0;
}
```

Program 5.2'in C++ kodu

Programın çıktısı aşağıdaki gibidir:

```
1 2 3 4 5 6 7 8 9 10
```

Problemin çözümü için çizilen akış şeması ve yazılan C++ koduna yakından bakılırsa döngünün başlangıç değeri for parantezinden önce atanmıştır. Problemi ilginç yapan da bu yaklaşımdır. for döngülerinde döngü değişkeninin tanımı, başlangıç değerinin atanması, döngüden çıkış koşulunun belirlenmesi ve artırım for parantezinin içerisinde yapılabileceği gibi(genellikle bu şekilde kullanılır) bütün bu tanımlamaların tamamı for parantezi dışında da yapılabilir. Program 5.2 için yazılan aşağıdaki C++ programını yazınız ve çalıştırınız.

```
#include <iostream>
using namespace std;
int main() {
    int i=1;
    for(;;){
        cout<<i<<" ";
        i++;
        if(i>10)
        break;

    }
    return 0;
}
```

Program çalıştırıldıktan sonra elde edilen ekran çıktısını bir önceki(program 5.2) programın ekran çıktısı ile karşılaştırınız.

```
1 2 3 4 5 6 7 8 9 10
```

Karşılaştırma sonucunda Yazılan bu yeni programın(elemanların tamamı for parantezi dışında tanımlanmıştır), çıktısı ile program 5.2'nin çıktısının aynı olduğunu göreceksiniz.

5.1.4. break Deyimi

break ifadesi döngü bloğu çalışırken belli koşullar altında döngüden tamamen çıkılmasını sağlar. Yukarıda en son yazılan C++ programında for parantezi ***for(; ;)*** şeklinde yazılmıştır. Aslında bu sonsuz bir döngü oluşturur. Yani program, sürekli döngü bloğu içerisinde çalışmaya devam eder ve sonsuz sayıda rakam üretir. Fakat bizim, yazdığımız bu programdan beklediğimiz çıktı, bir ile on arasındaki sayıların ekrana yazılmasıdır. Bunu sağlayabilmek için birden on 'a kadar sayılar ekrana yazıldıktan sonra, sonsuz olan bu döngüden çıkmak için **break** ifadesi kullanılmıştır.

İpucu: ***break*** deyimi, içerisinde yer aldığı döngüyü sonlandırmak için kullanılır. Bir döngü içerisinde çalışma sırası break deyimine geldiğinde program akışı döngünün tamamlanıp tamamladığına bakılmadan döngü sonundaki il ifadeye geçer.

5.1.5. continue Deyimi

continue deyimi döngü içerisinde program akışının başa dönmesini sağlar. Döngü içerisinde çalışma sırası ***continue*** deyimine geldiğinde, ***continue*** deyiminden sonraki satırlar çalıştırılmaz ve programın akışı döngü içerisindeki ilk ifadeye aktarılır.

5.1.6. Program 5.3

Bir ile yüz arasındaki sayılardan ilk 5 ve son beşini tek bir for döngüsü kullanarak ekrana yazdıran C++ programını yazınız.

Çözüm:

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

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

        if(i>5 &&  i<95)
        continue;
        cout<<i<<" ";
    }
    return 0;
}
```

Programın çıktısı aşağıdaki gibidir:

```
1 2 3 4 5 95 96 97 98 99 100
```

Programın ürettiği çıktıdan da anlaşılacağı gibi ***if (i>5 && i<95)*** koşulu sağlandığında (parantezin içerisi bire eşit olduğunda) ***continue*** deyiminin altındaki ifade çalışmayacak, beşten büyük ve doksan beşten küçük sayılar ekrana yazılmayacaktır. Çünkü bu koşul doğru(true) olduğunda ***continue*** deyimi çalışacak ve program akışını döngünün başına geri döndürecektir.

İpucu: ***continue*** deyimi döngü içerisinden çıkmadan belli koşullar altında döngü gövdesindeki ifadelerin bir kısmının veya tamamının çalıştırılmamasını sağlar.

5.1.7. Program Blokları Arasında Tanımlanan Değişkenlerin Erişilebilirliği

Döngüler içerisinde birkaç ifadenin küme parantezleri ile sınırlandığı program bölümlerine ***program bloğu*** denir. Bu bloklar içerisinde tanımlanan değişkenlere, yalnızca tanımlı oldukları bloklar içerisinden erişilebilir. Blok içerisinde tanımlı olan değişkenlere blok dışından erişilemez. Bu yaklaşım aynı isimli değişkenlerin farklı bloklarda tanımlanıp kullanılabilmesini sağlar.

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

    for (i = 0; i < 15; i++)
    {
        int kare = i * i;
        cout << kare;
        cout << " ";
    }
    cout << endl;
    cout << kare << endl; // Kare değişkeni burada kullanılamaz
    system("PAUSE");
    return 0;
}
```

Program bloğu içerisinde tanımlanan değişken blok dışında kullanılamaz

5.1.8. for Parantezi İçerisinde Birden Fazla Başlangıç ve Artırım Deyimi Kullanmak

for parantezi içerisinde kullanılan deyimlerden başlangıç ve artırım deyimleri için birden fazla değişkene değer atanabilir. Koşul için bu kural geçerli değildir. Koşul değişkenine tek bir değer atanabilir. Parantez içerisindeki deyimlere birden fazla ilk değer ataması yapılacaksa atamalar virgül ile birbirinden ayrılmalıdır.

Aşağıda verilen örnekte i, j değişkenlerine farklı iki başlangıç değeri verilmiş ve bu atamalar virgül ile birbirinden ayrılmıştır. Diğer taraftan artırım deyimi için de i, j değişkenlerine farklı değerler atanmış ve bu atamalar da virgül ile birbirinden ayrılmıştır. Örnekten de görülebileceği gibi koşul olarak tek değişken ve tek bir atama yapılmıştır.

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

int main()
{
    int i, j;

    for (i = 14, j = 0; i >= 0; i--, j++)
    {
        cout << i * j << " ";
    }
    cout << endl;
    system("PAUSE");
    return 0;
}
```

Birden fazla ilk değer atama örneği

5.1.9 Farklı for Döngüsü Örnekleri

a-)

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    for (i = 0; i <10; i++); // Burada Noktalı virgül kullanılmamalıdır.
    {
        cout << i<<endl;
        cout << " ";
    }
    cout << endl;
    system("PAUSE");
    return 0;
}
```

b-)

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

int main()
{
    // i değişkeni döngüde tanımlanıyor
    for (int i = 0; i < 15; i++)
    {
        int kare = i * i;
        cout << kare;
        cout << " ";
    }
    cout << endl;

    cout << i << endl; // i değişkeni döngüde tanımlandı.Döngü dışında kullanılamaz

    system("PAUSE");
    return 0;
}
```

c-)

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

int main()
{
    // for döngüsü 14'ten 0'a kadar geri geri sayar
    for (int i = 14; i >= 0; i--)
    {
        cout << i * i << " ";
    }

    cout << endl;

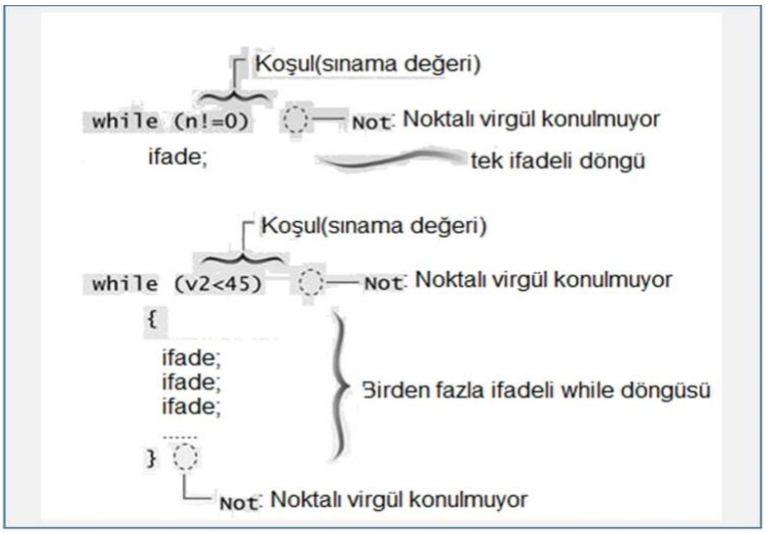
    system("PAUSE");
    return 0;
}
```

d-)

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    for(; ;){ //Sonsuz Döngü oluşturur.
        cout<<"İstanbul Üniversitesi"<<endl;
    }
    return 0;
}
```

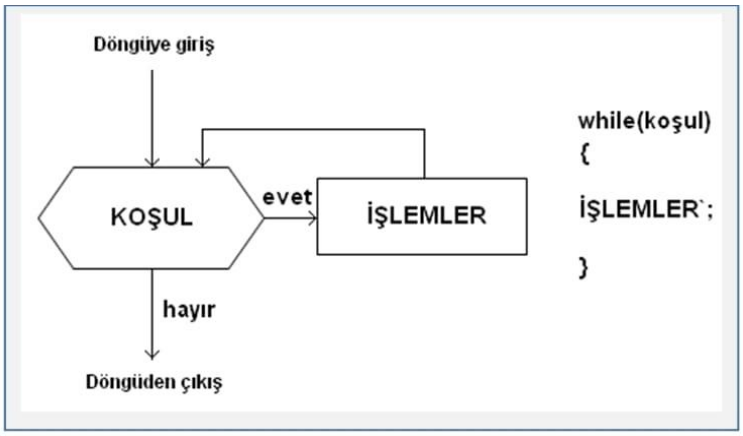
5.2. while Döngüsü

While döngüsü, daha çok döngüye girmeden önce döngü gövdesindeki ifadelerin kaç defa çalıştırılacağını bilmediğimiz durumda kullanılan bir döngüdür. Aşağıda while döngüsünün söz dizimi verilmiştir ***şekil 5.4.***



Şekil 5.4 *while döngüsünün söz dizimi*

While döngüsünde, while deyiminden sonra, parantez içerisinde koşul ifadesi yer alır. Fakat bu döngüde, ilk değer atama ve artırım deyimleri yer almaz. Bu açıdan bakıldığında while döngüsü for döngüsünün basitleştirilmiş halidir. while döngüsünde de koşul sınama değeri doğru(true) olduğu sürece, for döngüsünde olduğu gibi, döngü gövdesindeki ifadeler tekrar, tekrar çalıştırılır. Şekil 5.5'de while döngüsünün blok akış şeması verilmiştir.



Şekil 5.5*. while döngüsü blok akış şeması*

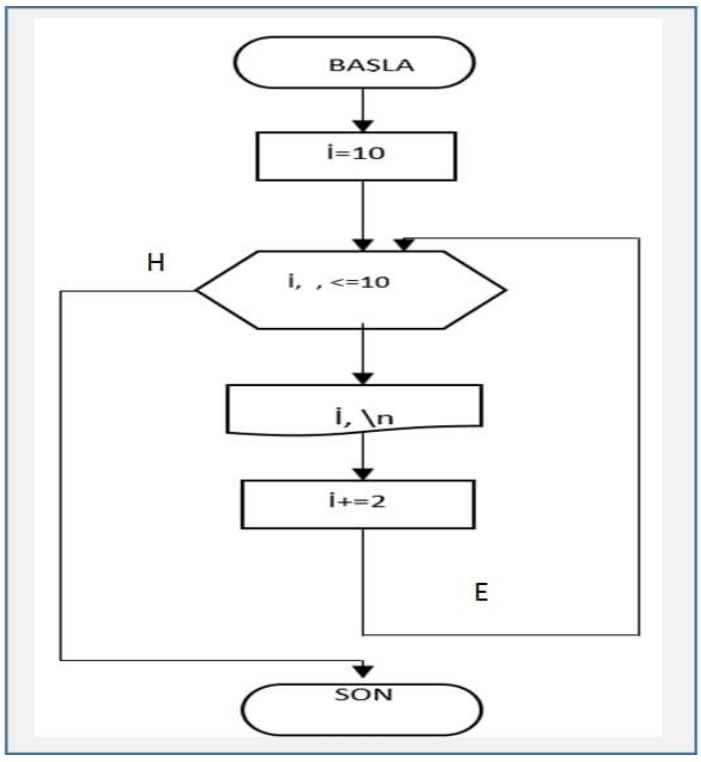
While döngüsünde ilk değer atama ifadesi olmasa da döngü değişkenine döngüye girmeden önce mutlaka bir ilk değer atanmalıdır. Aksi halde döngü içerisine, while parantezi içerisindeki koşul sağlanamayacağı için girilmez ve programın akışı doğrudan döngü bloğu sonuna aktarılır. While döngü bloğu içerisinde mutlaka döngü değişkeninin değeri değiştirilmelidir ki, döngüden çıkış değerine ulaşılabilsin.

While döngülerinde de for döngüsünde olduğu gibi döngü gövdesinde tekbir ifade yer alırsa bu ifadenin küme parantezleri içerisine alınması gerekmez. Fakat unutulmamalıdır bu tek ifade küme parantezleri içerisine alınsa da hata olmaz. Eğer while döngü gövdesinde birden fazla ifade yer alıyor ise bu ifadeler mutlaka küme parantezleri içerisine alınmalıdır.

5.2.1. Program 5.3

Sıfır ile 10 arasındaki çift sayıları ekrana yazdıran algoritmanın akış şemasını çiziniz ve C++ kodunu yazınız.

Çözüm:



Şekil 5.6. program 5.3'ün akış şeması

```
#include <iostream>
using namespace std;
int main()
{
    int i = 0;
    while (i <= 10)
    {
        cout << i <<endl;

        i+=2;
    }
    cout << endl;
    system("PAUSE");
    return 0;
}
```

Program 5.3'ün C++ kodu

Program 5.3'ün ekran çıktısı:

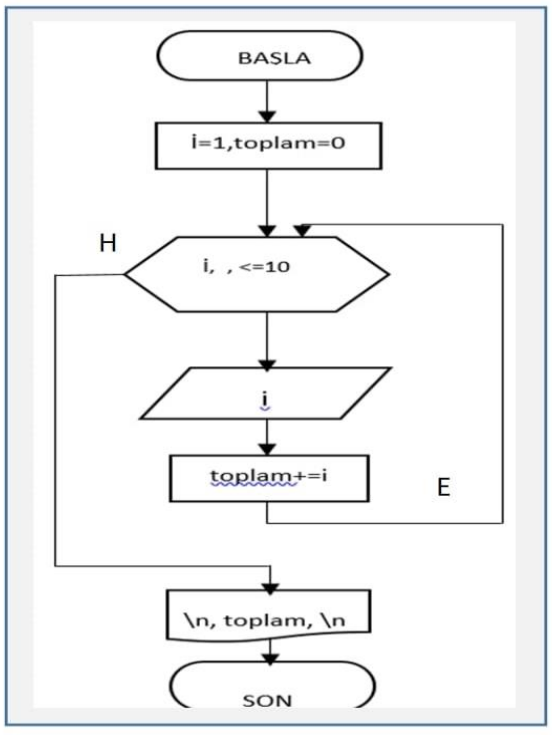
```
0
2
4
6
8
10
```

Şekil 5.6'dan veya program 5.3'ün C++ kodundan görüleceği gibi bu problemde, **for** döngülerinde olduğu gibi, döngü gövdesinin kaç defa çalıştırılacağı önceden bellidir. Burada bu örneğe yer verilmesinin nedeni ***while*** döngüleri ile bu türden problemle de çözülebilir. Programda döngüye girilmeden önce, döngü değişkenine ilk değer ataması yapılmıştır. Daha sonra while parantezindeki koşul test edilmiş ve koşul sağlandığı için döngü bloğundaki ifadeler çalıştırılmıştır. Bu işlem döngü değişkeni 12 olana kadar devam etmiş ve döngü değişkeni 12 olduğunda programın akışı döngü bloğunun dışındaki ilk ifadeye aktarılmıştır.

5.2.2. Program 5.4

Bir while döngüsü içerisinde klavyeden sıfır girinceye kadar girilen sayıları toplayıp sonucu ekrana yazdıran algoritmanın akış şemasını çiziniz ve C++ kodunu yazınız?

Çözüm:



Şekil 5.7 *Program 5.4'ün akış şeması*

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

int main()
{
    int i = 1,toplam=0;

    while(i!=0)
    {
        cout << "0'dan farkli bir sayi girin: ";
        cin>>i;
    /*  toplam=toplam+i;  toplama işlemi için
        bu ifadede kullanılabilir. */
        toplam+=i;
    }

    cout <<endl<<toplam<<endl;

    system("PAUSE");
    return 0;
}
```

Program 5.4'ün C++ kodu

Program 5.4'ün Program 5.3'ten farkı, Program 5.4'te döngüden çıkmak için kontrolün tamamen kullanıcının seçeneğine sunulmasıdır. Program çalışırken döngü içerisinden her iterasyonda bir sayı girilmesi, girilen sayının toplama eklenmesi öngörülmüştür. Programın(Program 5.4), daha önce bu bölümde yazılan uygulamalardan farkı, önceden döngü gövdesindeki tekrar sayısının belirlenmemiş olması ve döngüden çıkışın tamamen kullanıcıya bırakılmış olmasıdır. Kullanıcı klavyeden sıfır girdiğinde döngüden çıkılacak ve programın çalışması döngü sonundaki ilk ifadeye(toplama sonucunun ekrana yazdırılması) aktarılacaktır. Bu yaklaşım program kontrolünün uygulamadan kullanıcıya geçmesini birazda olsa sağlamıştır.

5.3. Sonsuz Döngüler

Sonsuz döngü, döngü çalışmaya başladıktan sonra, döngü bloğunun dışarıdan müdahale edilmediği sürece çalışmaya devam etmesi anlamına gelmektedir. Aşağıda iki farklı sonsuz döngü örneğinin C++ kodları verilmiştir.

```cpp
#include <iostream>
using namespace std;
int main() {
    int i=0;
    while(1)
    cout<<i++;
    return 0;
}
```

Döngünün sonsuz olması while parantezindeki (1) den kaynaklanmaktadır. Bu koşul programın çalışması sırasında sağlanmaz.

```
9745984599460046014602460346044605460648
2746284629463046314632463346344635463648
5746584659466046614662466346644665466648
8746884689469046914692469346944695469648
1747184719472047214722472347244725472647
4747484749475047514752475347544755475647
7747784779478047814782478347844785478647
0748084809481048114812481348144815481648
3748384839484048414842484348444845484648
```

Programın ürettiği çıktıdan bir kesit

```
#include <iostream>
using namespace std;
int main() {
    int i=0;
    while(true)
     cout<<i++;
    return 0;
}
```

Yanda verilen bu program da sonsuz döngü oluşturacaktır çünkü while parantezinin içi hiçbir zaman ***false*** olmayacaktır.

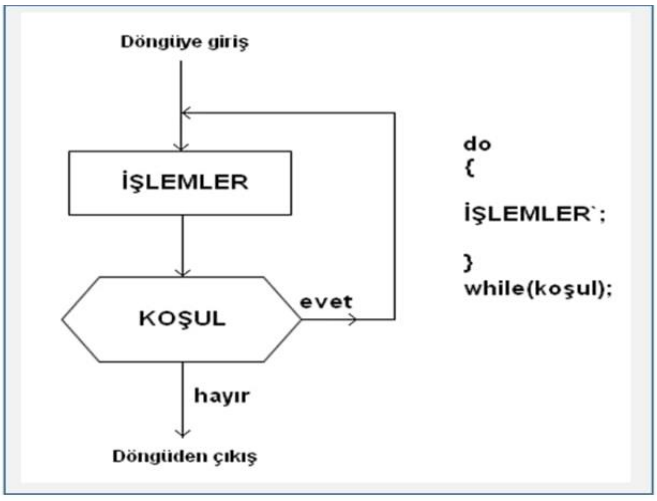
```
9745984599460046014602460346044605460647
2746284629463046314632463346344635463647
5746584659466046614662466346644665466647
8746884689469046914692469346944695469647
1747184719472047214722472347244725472647
4747484749475047514752475347544755475647
7747784779478047814782478347844785478647
0748084809481048114812481348144815481647
3748384839484048414842484348444845484647
```

Programın ürettiği çıktıdan bir kesit(önceki programla aynı çıktı üretilmiştir)

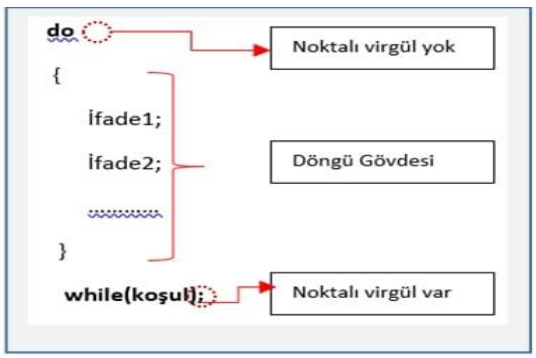
5.4. do….while Döngüsü

"***do … while***" döngüsü diğer döngüler gibi aynı işlemleri birçok kez tekrarlamak için kullanılır. Farklı olarak, bu döngüde koşul sınaması yapılmadan döngüye girilir ve döngü gövdesindeki ifadeler en az bir kere işletilir. ***do….while*** döngülerinde koşul kontrolü döngü sonundaki ***while*** parantezindeki deyim ile yapılır. Bu parantezin sonunda noktalı virgül vardır. Bu döngü yapısında da koşul sağlandığı sürece çevrim tekrarlanır.

Koşul tek bir karşılaştırmadan oluşabileceği gibi birden çok koşulun mantıksal operatörler ile birleştirilmesi ile de oluşturulabilir.



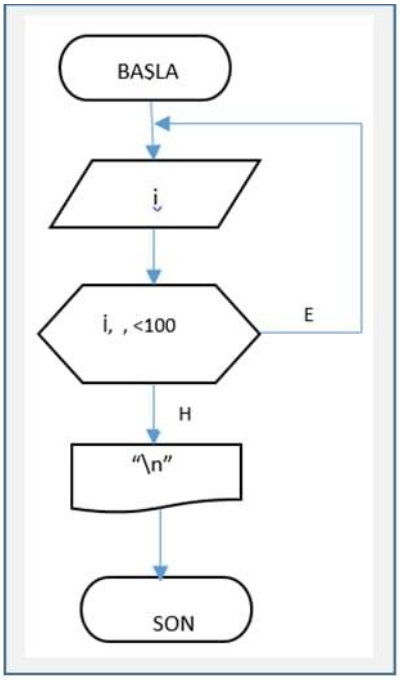
Şekil 5.8 *do….while döngüsünün akış şeması*



Şekil 5.9. *do….while döngüsünün söz dizimi*

5.4.1. Problem 5.5

Bir do….while döngüsü içerisinde, kullanıcının 100'den büyük veya 100'e eşit bir sayı girişine kadar klavyeden sayılar girmesini sağlayan algoritmanın akış şemasını çiziniz ve C++ kodunu yazınız?



Şekil 5.10 *Problem 5.5'in akış şeması*

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

int main()
{
    int i;

    do
    {
        cout << "100'den kucuk bir sayi girin: "
        cin >> i;
    } while (i < 100);

    cout << endl;

    system("PAUSE");
    return 0;
}
```

Problem 5.5'in C++ kodu

5.4.2. Problem 5.6

Bir do….while döngüsü içerisinde klavyeden girilen sayının karesini alan, sonucu ekrana yazdıran ve işleme devam edip etmeme tercihini kullanıcının seçimine bırakan algoritmanın C++ kodunu yazınız?

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    char c;

    do
    {
        cout << "Bir sayi girin: ";
        cin >> i;
        cout << "Girilen sayinin karesi: " << i * i << endl << endl;

            cout << "Baska bir sayi daha girmek istiyor musunuz? (e/h)? ";
            cin >> c;

    } while (c != 'h');

    cout << endl;
    system("PAUSE");
    return 0;
}
```

Problem 5.6'nın C++ Kodu

5.4.3. Problem 5.7

Birden yirmiye kadar olan sayıları her satırda 5 sayı olacak şekilde ekrana yazılmasını sağlayan algoritmanın C++ kodunu yazınız?

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

int main() {
    int i=0;
    do{
        cout<<++i;
        if(i%5==0)
            cout<<en
    }while(i<20);
    return 0;
}
```

Problem 5.7'nin C++ Kodu

Bölüm Özeti

Programların geliştirilmesi sırasında, genel olarak döngülerin kullanılma nedenlerini öğrendik. Yapısal programlamada, döngülerin daha kısa kod yazılması için önemli araçlar olduğunu gördük.

Daha sonra C++ 'ın desteklediği üç döngü yapısını inceledik: for döngüleri, while döngüleri ve do while döngüleri. Bu döngülerin birbirleri arasındaki farkları ve hangi döngünün hangi tür problemle karşılaşınca tercih edileceğini kavradık. Her üç döngünün blok akış şemalarının çizilmesi ve C++ kodlarının yazılması da ayrıca bu bölümde ele alındı.

Bundan başka, sonsuz döngülerin nasıl oluştuğu ve çalışma şekilleri de gözden geçirildi.

ÜNİTE SORULARI

1. Aşağıdaki kod parçasındaki hata nedir?

int x=1;
while(x<10);
x++;

- x değişkenine bir atanması

- while parantezinde koşulun x<0 olarak belirlenmesi
- x ‘in birer birer arttırılması
- while bloğunun küme parantezi içerisine alınmamış olması
- while-koşul parantezinden sonra noktalı virgül kullanılması

2. Aşağıdaki kod parçasının çıktısı nedir?

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

- Herhangi bir çıktı üretmez?
- Ondan sıfıra kadar(sıfır hariç) sayıları ekrana yazar
- Sıfırdan(sıfır dahil) ona kadar olan sayıları(on hariç) ekrana yazar.
- Sıfırdan(sıfır dahil) ona kadar olan sayıları(on dahil) ekrana yazar.
- Ekrana sadece 10 yazar.

3. Aşağıdaki döngü için doğru olan şık aşağıdakilerden hangisidir?

```
int x=0;
while(1){
x+=1;
cout<<x;}
```

- Bir çıktı üretmez.
- Ekrana sadece bir adet 1 yazar.
- Ekrana sadece bir adet 0 yazar.
- Ekrana sonsuz sayıda 1 yazar.
- Ekrana 1’den başlayıp sonsuza kadar olan sayıları yazar.

4. for döngüsünün koşul parantezindeki döngü değişkenine döngü boyunca(döngünün 10 defa döndüğünü varsayınız) kaç defa ilk değer ataması yapılır?

- 0
- 1
- 2
- 9
- 10

5. Aşağıdaki kodun çıktısı ne olur?

```
for(int i=1;i<10;i++){
        if(i>2 && i<8)
            continue;
            cout<<i<<endl;}
```

- Çıktı üretmez
- Birden ona kadar tüm sayıları ekrana yazar
- Ondan bire kadar olan sayıları ekrana yazar.
- İki ile sekiz arasındaki sayıları ekrana yazar.
- Ekrana yukarıdan aşağıya 1289 yazar.
    -
    - CEVAP ANAHTARI1. e2. d3. e4. b5. e

6. DİZİLER

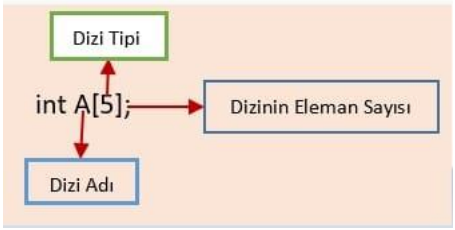
Giriş

Programlama dillerinde, aynı tipte veri parçalarını bir araya getirip gruplayan veri yapılarına dizi denir. Bir dizi içinde gruplanmış veri parçaları, *int* veya *float* gibi temel veri tiplerinden biri olabilir. Bu bölümde *int* ve *char* gibi temel veri tipindeki dizileri ele alacağız.

Diziler birkaç veriyi veya binlerce veriyi gruplayıp saklayabilir. Bu açıdan dizler, yedinci bölümde ele aldığımız yapılara benzer. Ancak, dizilerle yapılar arasında önemli iki fark bulunmaktadır. Bu farklardan birincisi, yapılar farklı tipteki verileri gruplayabilir, diziler ise aynı tipteki verileri gruplamak için kullanılır. Yapılarla diziler arasındaki ikinci fark, yapının üyelerine isimle ulaşılırken, dizi elemanlarına bir indeks ile ulaşılır.

6.1. Tek Boyutlu Dizileri tanımlamak

Değişkenlerde olduğu gibi, dizileri de kullanmadan önce tanımlamak gerekir. Bir dizi tanımlanırken, önce dizinin depolayacağı verilere bağlı olarak dizinin tipi belirlenir. Diziye bir isim verilir ve bilgisayar programlarında bu isimle dizi elemanlarına ulaşılır. Tek boyutlu dizilerde, bir köşeli parantez içerisine dizinin eleman sayısı yazılır. Köşeli parantez içerisine yazılan sayı ya sabit bir sayı olmalıdır ya da değeri sabit olan bir deyim olmalıdır. Ayrıca, bu sayı tam sayı olmak zorundadır. Beş adet tamsayı tutacak A dizinin söz dizimi şekil 6.1'de gösterilmiştir.



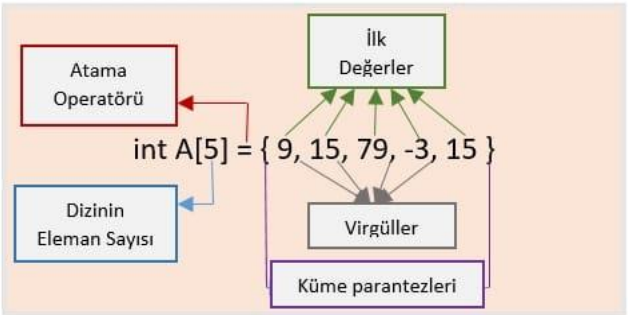
Şekil 6.1 Dizi tanımının söz dizimi

Beş elemanlı bu A dizisinin elemanları sırasıyla A[1], A[2], A[3],A[4], A[5] şeklinde ifade edilir. *Dizi, art arda gelen aynı tip verileri saklayan bellek elemanları veya bellek hücreleri olarak tanımlanır.* Bir dizi bellekte sabit büyüklükte bir yere yerleşir. Dizi tanımlandıktan sonra program içerisinde dizinin bellekte kapladığı alan miktarında artma veya azalma yapmak mümkün değildir.

Örneğin, 100 elemanlı bir dizi tanımlanırsa, program içerisinde bu dizinin sadece 10 elemanı kullanılsa bile geri kalan 90 elemanı bellekte yer işgal etmeye devam eder. Bu nedenle diziler tanımlanırken, dizilerde saklanılacak veri miktarı iyi analiz edilmelidir.

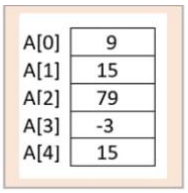
6.2. Tek Boyutlu Dizileri İlk Kullanıma Hazırlamak

Bir dizi tanımlandığı sırada dizi elemanlarının her birine bir değer atanabilir. Bu işleme diziyi ilk kullanıma hazırlama işlemi denir. Şimdi, yukarıda Dizileri Tanımlama bölümünde tanımı yapılan *int A[5];* dizisini ilk kullanıma hazırlayalım. Dizileri ilk kullanıma hazırlama işleminin söz dizimi aşağıda *şekil 6.2'de* gösterilmiştir.



Şekil 6.2 Diziyi ilk Kullanıma Hazırlama

İnt A[5]; dizisi beş elemanı vardır. Şekil 6.3'ten de görüldüğü gibi, dizinin birinci elemanının indisi sıfırdır. Beş elemana sahip bu dizinin son elemanının indisi de dörttür. Bu biraz akıl karışıklığına neden olabilir. Akıl karışıklığından kurtulmak için, tek boyutlu dizilerde dizinin son elamanının indisinin, dizinin eleman sayısından bir eksik olduğunu aklımızda tutmak yararlı olabilir. Bu bölümün ilerleyen kısımlarında, dizinin elemanlarına ulaşmak istediğimizde sürekli indisleri kullanacağız. Şekil 6.3'te dizi elemanlarının bellekte sembolik olarak yerleşimi gösterilmiştir.



Şekil 6.3 int A[5]={9,15,79,-3,15} dizisinin elemanları

6.2.1. Program 6.1

int A[5]={9,15,79,-3,15} dizisi verilmiştir.

a-) C++ kodunu yazınız?

b-) C++ kodunu yazınız?

c-) Diziye elman atama işlemi yapmadan(, dizi elemanlarını ekrana yazdıran algoritmanın C++ kodunu yazınız?

d-) Beş elemanlı int A[5]={9,15,79}; dizisine üç eleman atayıp, dizi elemanlarını ekrana yazdıran algoritmanın C++ kodunu yazınız?

e-) int A[5]={0}; ataması yaparak dizi elemanlarını ekrana yazdıran algoritmanın C++ kodunu yazınız?

f-) int A[5]={9,15,79,-3,15,6}; dizisine belirlenen eleman sayısından fazla eleman atayınız ve C++ uygulamasını yeniden yazınız?

Çözümler:

a-)

Daha önce for döngülerini incelemiştik. Hatırlanacağı gibi biz, for döngülerini daha çok döngüye girmeden, döngünün kaç defa dönmesi gerektiğini bildiğimiz durumda kullanıyorduk. Problem 6.1'de verilen A[5]={9,15,79,-3,15} dizisinin eleman sayısı beş olduğuna göre dizinin elemanlarının tamamının ekrana yazdırılabilmesi için döngünün beş defa dönmesi gerektiğini öngörmek zor değildir. Aşağıda verilen C++ kodundan da anlaşılacağı gibi, döngünün ilk dönüşünde dizinin sıfır indisine sahip elamanı ekrana yazdırılır. Döngünün İkinci dönüşünde ise, dizinin ikinci elamanı ekrana yazdırılır ve bu işlem dizinin son elemanının ekrana yazdırılmasına kadar devam eder.

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

    int A[5]={9,15,79,-3,15};
        for(int i=0;i<5;i++)
        cout<<"   "<<A[i]<<endl;
    return 0;
}
```

Program 6.1-(a)'nın C++ kodu

```
9
15
79
-3
15
```

Ekran çıktısı

b-)

Dizi, hem tanım olarak, hem de dizi elemanlarına atanan değerler açısında bu problemin (**a)** şıkkının çözümünde kullanılan dizi ile aynıdır. Buradaki farklılık dizinin elemanlarının tersten yazdırılmasının istenmesi. Çözüm için önce, dizinin son elemanının(***A[4]****'ün*) yazdırılması, sonra sondan bir önceki elemanın(***A[3]****'ün*) yazdırılması ve bu işlemin ilk eleman(**A[0]**) yazdırılana kadar bu şekilde devam etmesi gerekiyor.

A[5]={9,15,79,-3,15} dizisinin elemanlarının tersten yazdırılması için Problem 6.1'in (**a**) şıkkı için yazılan C++ kodunda, for parantezinde bazı değişiklikler yapılması gerekmektedir. Yapılması gereken değişikliklerden ilki,

döngünün başlangıcının dizinin son elemanına erişecek şekilde düzenlenmesi (**i=4**), ikincisi, Programın hangi koşul altında döngüdeki ifadeleri çalıştıracağını belirlemek(**i>=0**) ve son olarak artış değerini istenen çözüme göre yeniden düzenlemek. Bu problemin çözümü için artış değeri ***i***--olarak belirlenmiştir.

6.3.1. (b)'nın çözümü için yazılması istenilen C++ kodu, yukarıda açıklaması yapılan değişiklikler dikkate alınarak, yeniden yazılmış ve aşağıda ekran çıktısı ile birlikte gösterilmiştir.

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

    int A[5]={9,15,79,-3,15};
    for(int i=4;i>=0;i--)
        cout<<"   "<<A[i]<<endl;
    return 0;
}
```

Program 6.1-(b)'nın C++ kodu

```
15
-3
79
15
9
```

Ekran çıktısı

c-)

Problem 6.1'in (a) şıkkı için yazılan programdan dizi elemanlarına değer atama bölümü kaldırılmıştır. Programda for parantezinden önce sadece dizinin tanımı(***int A[5];***) bırakılmıştır. Program çalıştırıldıktan sonra, ekran çıktısı olarak rastgele değerler çıktığı gözlenmiştir. Bunun nedeni, Dizi tanımlandığında, tanımlanan dizi için bellekte yer ayrılır. Fakat dizi elemanlarına değer atanması yapılmadığı için dizi için, ayrılan adreslerdeki rasgele sayılar çıktı olarak üretilir. 6.3.1. (c)'nın çözümü için yazılması istenilen C++ kodu, yukarıda açıklaması yapılan değişiklikler dikkate alınarak, yeniden yazılmış ve aşağıda ekran çıktısı ile birlikte gösterilmiştir.

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

    int A[5];
        for(int i=0;i<5;i++)
        cout<<"   "<<A[i]<<endl;
    return 0;
}
```

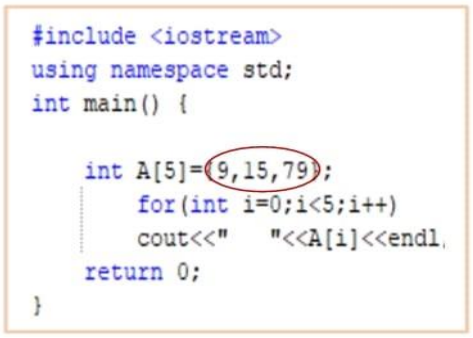
Program 6.1-(c)'nın C++ kodu

```
1
0
4254425
0
86
```

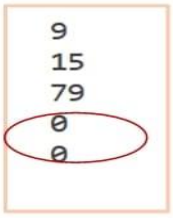
Ekran çıktısı

d-)

int A[5]; dizisini ilk kullanıma hazırlarken eksik sayıda eleman atanırsa (**int A[5]={ 9, 15, 79**}), atama yapılmayan elemanlara otomatik olarak sıfır değeri atanır. 6.3.1. (d)'nın çözümü için yazılması istenilen C++ kodu, yukarıda açıklaması yapılan değişiklikler dikkate alınarak, yeniden yazılmış ve aşağıda ekran çıktısı ile birlikte gösterilmiştir.



Program 6.1-(d)'nın C++ kodu

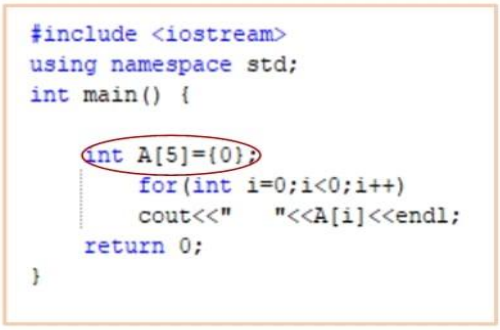


Ekran çıktısı

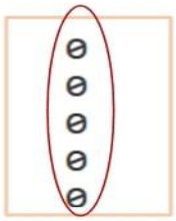
Program 6.1-(d)'nın C++ kodu Ekran çıktısı

e-)

Eleman sayısından bağımsız olarak, bir diziye sadece bir adet sıfır(***{0}***) ataması yapılırsa o dizinin bütün elemanlarının değeri sıfır olur. Problem 6.1'de ele aldığımız örnek diziye de ***int A[5]={0};*** şeklinde bir atama yapılmış ve programın ekran çıktısında dizinin bütün elemanlarının sıfır olduğu gözlenmiştir.



Program 6.1-(e)'nın C++ kodu

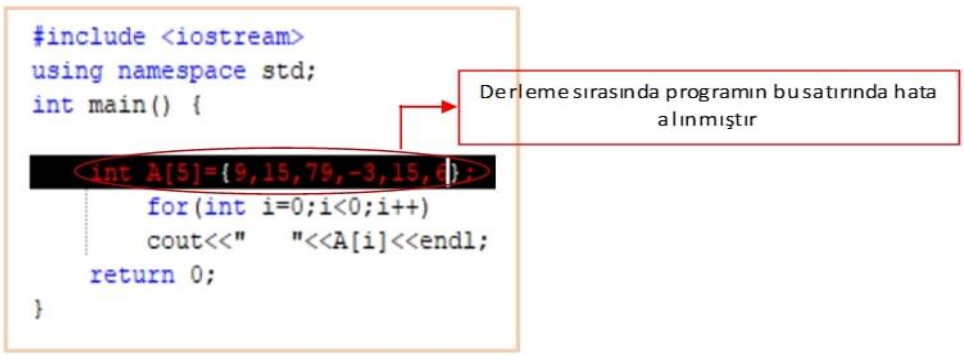


Ekran çıktısı

Program 6.1-(e)'nın C++ kodu Ekran çıktısı

f-)

Bir dizinin ilk kullanıma hazırlanması sırasında diziye, tanımında belirlenen eleman sayısında daha fazla değer ataması yapılırsa program derleme sırasında hata verir. Problem 6.1'de ele aldığımız örnek diziye de ***int A[5]={*** *9,15,79,-3,15,6****};*** şeklinde bir atama yapılmış ve programın derlenmesi sırasında hata ile karşılaşılmış ve herhangi bir ekran çıktısına ulaşılamamıştır.



Program 6.1-(f)'nın C++ kodu

6.3. Tek Boyutlu Dizi Elemanlarına Klavyeden Değer Girmek

6.3.'te verilen dizi örneğinde, dizi elemanlarının değerleri program içerisinde atanmıştı. Ancak, birçok problemin çözümünde, dizi elemanlarının değerlerinin klavyeden girilmesi gerekebilir. C++'ta dizi elemanlarına klavyeden değer girilmesi, değişkenlerin değerlerinin klavyeden girilmesine çok benzer.

cin>>A[0];

Dizilerin elemanlarına, klavyeden değer girilmesi için önce dizinin adı ve sonra da köşeli parantezin içinde, dizinin hangi elemanına değer girilecekse, o elemanın indisinin bilinmesi gerekir.

Bir dizinin bütün elemanlarına değer girilebilmek için ya dizinin her elanı için ayrı ifade yazmamız gerekecek(1. Yöntem) ya da bunu daha kısa yoldan bir for döngüsü içerisinde yapacağız(2. Yöntem). Bu iki örnekte aşağıda gösterilmiştir.

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int A[5];
    // Dizinin elemanlarına klavyeden değer girmek için gerekli ifadeler
        cout<<"Dizinin Birinci  elemanı: "; cin>>A[0];
        cout<<"Dizinin İkinci   elemanı: "; cin>>A[1];
        cout<<"Dizinin Üçüncü   elemanı: "; cin>>A[2];
        cout<<"Dizinin Dördüncü elemanı: "; cin>>A[3];
        cout<<"Dizinin Beşinci  elemanı: "; cin>>A[4];

    // Dizinin elemanlarına girilen değerleri göstermek için gerekli ifadeler
        cout<<endl<<endl;
        cout<<A[0]<<endl;
        cout<<A[1]<<endl;
        cout<<A[2]<<endl;
        cout<<A[3]<<endl;
        cout<<A[4]<<endl;

    return 0;
}
```

1. *Yöntem. Dizi elemanlarına klavyeden değer girilmesi*

İki programda daire içerisine alınan ifadeler aynı işlemi yapmaktadır

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int A[5];
    // For döngüsü İle dizi elemanlarına değer girilmesi
    for(int i=0;i<5;i++)
    {
        cout<<i+1<<". Eleman : ";cin>>A[i];
    }
    /* For döngüsü ile dizinin elemanlarına klavyeden girilen
       degerlerin gosterilmesi*/
    for(int i=0;i<5;i++)
    {
        cout<<"  "<<A[i]<<endl;
    }
    return 0;
}
```

2. *Yöntem Dizi elemanlarına klavyeden for döngüsü kullanılarak değer girilmesi*

Yukarıda dizi elemanlarına değer girilmesi için yazılan iki C++ programı da aynı işlemi yapmaktadır. İncelendiğinde kolayca görülebileceği gibi, 1. Yöntemde dizinin bütün elemanlarına giriş yapabilmek için çok daha fazla ifade yazılmıştır. Dizinin eleman sayısı arttığında bu ifadeler de eleman sayısı kadar artacaktır. Örneğin 1000 elemanı olan diziye klavyeden değer girilebilmesi için bu yöntem kullanılırsa 1000 ifadeye ihtiyaç duyulacaktır.

Girişler için for döngüsü kullanılırsa(2. Yöntem) Tek bir ifade ile dizinin bütün elemanlarına klavyeden değer girişi yapılabilir. Bu yöntemde dizinin eleman sayısının artması programa yeni ifadeler eklenmesini gerektirmez. Bunun için for parantezindeki koşul ifadesinin değiştirilmesi yeterli olacaktır. Örneğin 1000 elemanlı bir dizinin bütün elemanlarına değer girebilmek için koşul ifadesinin ***i<1000*** şeklinde değişmesi yeterli olacaktır.

6.4. Tek Boyutlu Dizi Elemanlarının Ortalamasını Almak

Bu uygulama, bir okulda derslere ait sınıf ortalamalarının hesaplanması için, C++ dili ile yazılmıştır. Uygulama çalıştırıldığında kullanıcıdan önce ortalaması hesaplanacak dersin adının, sonra sınıftaki öğrenci sayısının, daha sonra da öğrencilerin dersten aldıkları notların girilmesi istenmektedir. Bu bölümde daha önce ele alınan uygulamalarda dizilerin eleman sayıları sabit olarak girilmekteydi. Bu uygulamada ise kullanıcıdan dizinin eleman sayısının(ogrSayi) klavyeden girilmesi istenmektedir. Uygulamada yapılan bu değişiklik, programa önemli bir esneklik kazandırmış ve

öğrenci sayısının farklı olduğu derslerin ortalamasının koda müdahale etmeden alınması sağlanmıştır. Buna ilave olarak, ders ismi girişinin kullanıcıdan alınması da uygulamaya ek esneklik kazandırmıştır.

```
#include <locale.h>
using namespace std;
int main()
{
    setlocale(LC_ALL,"Turkish");
    int ogrSayi;
    string ders;
    cout<<"Ortalaması Alınacak Dersi Giriniz : ";
    cin>>ders;
    cout<<"Öğrenci Sayısı Giriniz            : ";
    cin>>ogrSayi;
    int dersNotu[ogrSayi];
    for (int i = 0; i < ogrSayi; i++)
    {
        cout <<i + 1<<". ogrencinin "<<ders<<" Notunu girin: ";
        cin >> dersNotu[i];
    }
    int toplam = 0;
    for (int i = 0; i < ogrSayi; i++) toplam += dersNotu[i];
    float sinif_ort = toplam / ogrSayi;
    cout << "Sinif ortalamasi: " << sinif_ort << endl;
    system("PAUSE");
    return 0;
}
```

Dizinin eleman sayısı klavyeden girilerek uygulamaya esneklik kazandırılmıştır.

Öğrencilerin derslerden aldıkları notların ortalamasını hesaplayan program

```
Ortalaması Alınacak Dersi Giriniz : Matematik
Öğrenci Sayısı Giriniz            : 2
1. ogrencinin Matematik Notunu girin: 37
2. ogrencinin Matematik Notunu girin: 54
Sinif ortalamasi: 45.5
```

```
Ortalaması Alınacak Dersi Giriniz : Fizik
Öğrenci Sayısı Giriniz            : 4
1. ogrencinin Fizik Notunu girin: 67
2. ogrencinin Fizik Notunu girin: 29
3. ogrencinin Fizik Notunu girin: 53
4. ogrencinin Fizik Notunu girin: 74
Sinif ortalamasi: 55.75
```

Programın iki farklı derse ait ortalama hesabı ekran çıktısı

6.5. Çok Boyutlu Diziler

Bazı problemin çözümünde tek boyutlu diziler yetersiz kalabilir. Bu nedenle pek çok programlama dili birden fazla boyuttan oluşan dizilerin kullanılmasına izin verir ve bu şekilde tek boyutlu dizilerle çözümü modellenemeyen birçok problem iki veya daha fazla boyuta sahip dizilerle çözülebilir. Ayrıca, matematik derslerinden aşina olduğumuz matris işlemlerinin çözümü için de yine programlama dillerinin sağladığı bu özelliklerden yararlanılır.

İki boyutlu bir matris kullanılarak çözülebilecek problemlere örnek teşkil etmesi için, bir pazarlama firmasının üç farklı bölgede ürünlerini sattığını ve bu firma aylık olarak bu bölgelerde satış miktarlarını, bilgisayarda, bir program aracılığı ile takip etmek istediğini varsayalım. Bu problemin çözümü için, hem bölge hem de ay bilgisi tutulması gerektiğinden, yani tutulması gereken bilginin tablo karakteri göstermesinden dolayı, iki boyutlu bir matris kullanılması gerektiğinin tahmin edilmesi zor değildir. İki veya daha çok boyutlu matrisler, bu ve benzeri problemlerin çözümü için kullanılabileceği gibi iki matrisin toplanması, matrisin transpozunun alınması, birim matris oluşturulması ve benzer matris işlemlerinin yapılmasında da yaygın olarak kullanılır.

6.6. Çok Boyutlu Dizilerin Tanımlanması

Çok boyutlu dizilerin tanımı temelde tek boyutlu dizilerin tanımına benzer. İki tanımlama arasındaki temel farklılık, tek boyutlu dizilerde indis sayısı bir olmasına karşılık, örneğin iki boyutlu dizilerde indis sayısı ikidir. İki boyutlu dizinin birinci indisi satırlara ikinci indisi de sütunlara karşılık gelir. Bilgisayar hafızasında(RAM) tek boyutlu bir yapı vardır ve bundan dolayı çok boyutlu yapılar hafızada tek boyuta indirgenerek saklanır. Dolayısı ile iki boyutlu dizilerde birinci

indisin satır, ikinci indisin sütün olduğunu söylemek bir kabulden ibarettir. Kolayca tahmin edilebileceği gibi, üç boyutlu dizilerde de indis sayısı üçtür.

Aşağıda C++ yazım kurallarına göre tanımlanmış bir matris gösterilmiştir:

```
int A[2][3];
```

Bu ifadeyi C++ programında yazarsak hafızada 2x3 boyutunda bir matris yer açılır. Açılan bu yer altı hücrelidir ve bu hücrelerde altı farklı tamsayı tutulabilir.

Aşağıda 2x3 matrisin satır ve sütun indisleri gösterilmiştir.

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]

Aşağıda dizinin bütün elemanlarına değer atanmış ve atanan değerler tabloda gösterilmiştir.

```
A[0][0]=12;
A[0][1]=15;
A[0][2]=27;
A[1][0]=3;
A[1][1]=83;
A[1][2]=9;
```

12	15	27
3	83	9

6.6.1. Problem 6.2

ogrNot[4][2] şeklinde İki boyutlu bir dizi tanımlayın ve bu dizinin her bir satırındaki dizi elemanlarına bir öğrencinin vize ve final notunu atayınız. Daha sonra bu dört öğrencinin ağırlıklı not ortalamasını hesaplatıp ekrana yazdıran bir C++ programı yazınız?

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    // Çok boyutlu diziye ilk değer veriliyor
    int ogrNot[4][2] = { {15, 30}, {45, 77}, {38, 55}, {100, 90} };

    for (int i = 0; i < 4 ; i++)
    {
        float ort = ogrNot[i][0] * 0.4F + ogrNot[i][1] * 0.6F;

        cout << i + 1 << ". ogrencinin ortalamasi: "<<ort<<endl;
    }
    system("PAUSE");
    return 0;
}
```

İki Boyutlu dizi ilk kullanıma hazırlanıyor.

Ortalama hesaplanıp, ekrana yazdırılıyor

Dört öğrencinin not ortalamalarını hesaplayan program

```
1. ogrencinin ortalamasi: 24
2. ogrencinin ortalamasi: 64.2
3. ogrencinin ortalamasi: 48.2
4. ogrencinin ortalamasi: 94
Press any key to continue . . .
```

Öğrencilerin not ortalamalarını hesaplayan programın çıktısı

6.6.2. Problem 6.3

Birim matris oluşturan algoritmanın C++ kodunu yazınız ve akış şemasını çiziniz?

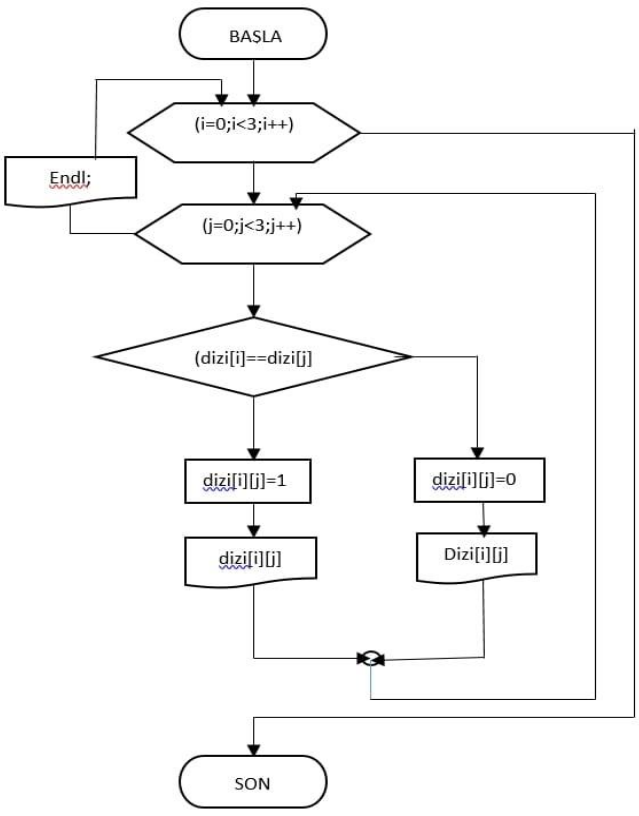
```
#include <iostream>
using namespace std;

int main() {
    int dizi[3][3],i,j;
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            if(i==j){
             dizi[i][j]=1;
             cout<<"  "<<dizi[i][j];
            }
            else
            {
             dizi[i][j]=0;
             cout<<"  "<<dizi[i][j];
            }
        }
        cout<<endl;
    }
        return 0;
}
```

```
1  0  0
0  1  0
0  0  1
```

Problem 6.3'ün ekran çıktısı

Şekil 6.4 oldukça ilginç bir örnektir. Akış şemasından da görülebileceği gibi, şekilde iç içe iki döngü vardır. Birinci döngü(dıştaki döngü) satırları oluştururken, ikinci döngü(içteki döngü) sütunları oluşturmaktadır. Birinci döngüde i'nin değeri sıfır olunca akış ikinci döngüye geçmektedir. İkinci döngüye ilk girişte, J'nin değeri de sıfır olduğu için dizi[i][j]=1 ifadesi çalıştırılmış ve arkasından bu değer ekrana yazdırılmıştır. Sonra içteki döngü bir kere daha çalışmış, fakat bu defa i'nin değeri hala sıfır iken j'nin değeri bir olmuştur i ve j'nin eşitliği sağlanmadığı için karar bloğunun else kısmındaki ifade çalışmış ve ekrana sıfır yazılmıştır. i'nin değeri sıfırken, j iki olduğunda, içteki döngü üçüncü ve son defa çalıştırılmış ve ekrana tekrar sıfır yazdırılmıştır. J, üç olduğunda matrisin ilk satırı tamamlanmıştır ve akış yeni satıra geçilerek, dıştaki döngüye geçmiştir. Akış dıştaki döngüye tekrar geldiğin de i'nin değeri bir olmuş ve akış tekrar içteki döngüye geçmiş ve içteki döngüye ilk girişte, j'nin değeri sıfır olduğu için karar bloğunun else kısmındaki ifadeler çalıştırılmış ve ikinci satırın il elemanı olarak ekrana sıfır yazdırılmıştır. Döngülerin çalışması tamamlanınca ekrana 3x3 bir birim matris yazdırılmıştır.



Şekil 6.4 Problem 6.3'ün Akış şeması

Bölüm Özeti

Bu bölümde yapısal programlarda tek boyutlu ve çok boyutlu dizilerin kullanım amaçlarını ve değişkenlere göre avantajlı yanlarını öğrendik.

Ayrıca, hem tek boyutlu dizileri hem de çok boyutlu dizileri ilk kullanıma hazırlamayı, dizi elemanlarına klavyeden değer girmeyi, dizi elemanlarının değerlerini bellekten okuyup hesaplama yapmayı öğrendik

İki boyutlu dizi örneğinden yola çıkarak içerisinde dizi olan algoritmaların akış şemalarını çizmeyi ele aldık.

ÜNİTE SORULARI

1. Tek boyutlu ve çok boyutlu dizilerde, dizi tanımlanırken eleman sayısı hangi temel veri tipi ile belirlenir?

- bool
- float
- char
- int
- double

2. Bir diziye ilk değer ataması yapılırken dizinin eleman sayısından daha fazla değer atanırsa aşağıdakilerden hangisi gerçekleşir?

- Derleme sırasında hata döner
- Diziye yapılan atamada dizi boyutu kadar değer dizi elemanlarına atanır.
- Dizinin bütün elemanları sıfır olur.
- Dizinin bütün elemanları rastgele sayılardan oluşur.
- Dizinin bütün elemanları bir olur

3. int A[4][3]; şeklinde tanımı yapılan dizinin kaç elemanı vardır?

- 7
- 4
- 3
- 12
- 15

4. int A[5]={0}; ifadesinden sonra A dizisi için aşağıdakilerden hangisi doğrudur?

- Derleme sırasında, tanımlama şeklinden dolayı hata döner
- Dizinin bütün elemanları sıfır olur.
- Dizinin ilk elemanı sıfır, diğer dört elemanı rastgele sayılardan oluşur.
- Dizinin son elemanı sıfır, diğer dört elemanı rastgele sayılardan oluşur.
- Dizinin beş elemanı da rasgele sayılardan oluşur.

5. Aşağıda verilen kod bloğunun çıktısı hangisidir?

```
 int dizi[4]={1,2,3,4};
for(int i=0;i<4;i++){
                if(i==2)
                cout<<endl;
                cout<<dizi[i]<<" ";
}
```

- 1 2 3 4
- 4 3 2 1
- 1 2
  3 4
- 3 4
  1 2
- 4 1 2 3

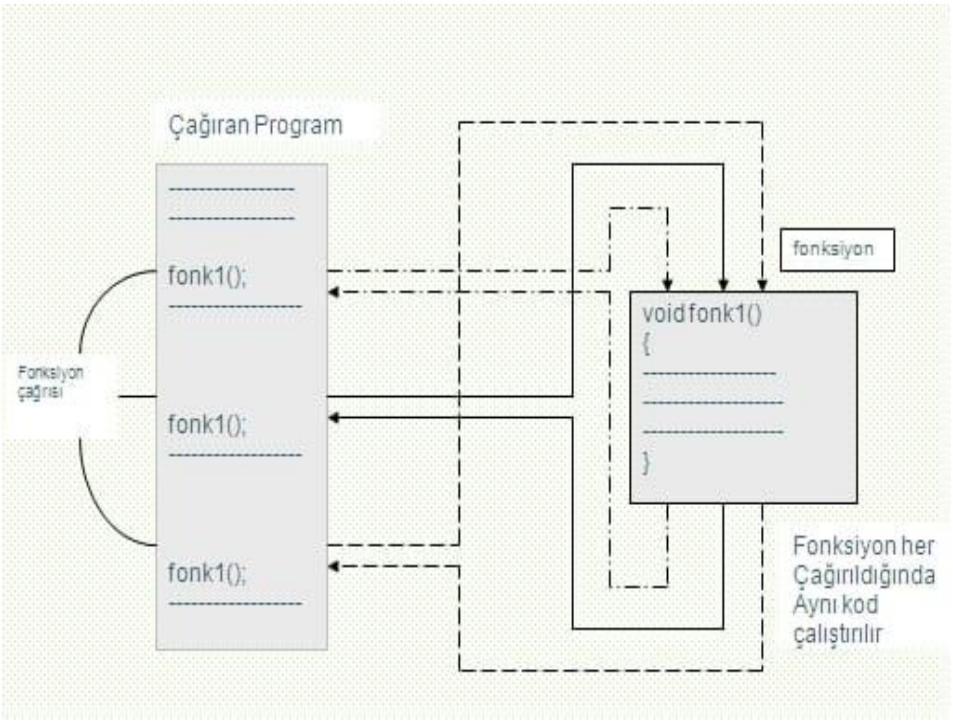
-
- CEVAP ANAHTARI1. d2. a3. d4. b5. c

7. FONKSİYONLAR

Giriş

Bir fonksiyon, bir veya birkaç program ifadesini gruplar ve bu gruba bir isim verir. Bu program ifadeleri grubu, daha sonra programın başka bölümleri tarafından çağırılabilir. Fonksiyonların kullanılmasının ilk nedeni programın kavramsal olarak düzenlenmesi ihtiyacından ortaya çıkmıştır. Programların fonksiyonlar halinde yazılması yapısal programlamanın da temelidir.

Bundan başka, programları fonksiyonlar halinde yazılırsa daha kısa kod yazarak işlemlerinizi yürütebilirsiniz. Bir fonksiyon, programın içerisinde birkaç defa çağırılıp çalıştırılabilir. Dolayısıyla bu daha kısa, daha verimli program yazmamızı sağlar. Şekil 7.1'de bir programda, fonksiyonun değişik bölümler tarafından nasıl çağırıldığı gösterilmiştir.



7.1. Basit Fonksiyonlar

7.1.1. Program 7.1

İçerisinde ekrana 50 adet "=" karakteri basan bir fonksiyon bulunan bir C++ Programı yazınız.

```
#include<iostream>

using namespace std;

void cizgi(); //fonksiyon deklarasyonu(prototip)

int main()

{

cizgi();

cout<<"Veri Tipi"<<" "<<"boyut"<<endl;

cizgi();

cout<<"char -128 128"<<endl

 <<"short -32 768 32 767" <<endl

 <<"int sistemlere bağlı "<<endl

 <<"long -2 147 483 648 2 147 483 647"<<endl;

cizgi();

return 0;

}

void cizgi() //fonksiyon tanımı

{

 for(int i=0;i<=50;i++)

 cout<<"=";

 cout<<endl;

}
```

Program 7.1'in C++ kodu



Program 7.1'in ekran çıktısı

Yukarıdaki program iki fonksiyondan oluşmaktadır: main() ve cizgi(). Şimdiye kadar programlarımızda yalnızca tek bir fonksiyon; main fonksiyonu yer alıyordu. Bu uygulamada iki fonksiyon kullanılmıştır. İkinci fonksiyonu; cizgi() fonksiyonunu programa dâhil etmek için üç temel bileşen kullanılmıştır. Bu bileşenler fonksiyonun deklare edilmesi, fonksiyonun çağırılması ve fonksiyonun tanımıdır.

7.1.2. Fonksiyonların Deklare edilmesi

Fonksiyonlarda değişkenler gibi önce derleyiciye bildirilir ve daha sonra kullanılabilir. Bunun iki yolu vardır. Birinci yol fonksiyonu önce deklare etmektir. Diğer yol ise fonksiyonu tanımlamaktır. Yukarıdaki örnekte birinci yol tercih edilmiş; cizgi fonksiyonu önce deklare edilmiştir.

void cizgi();

Bu deklarasyon derleyiciye, daha sonraki bir noktada cizgi isminde bir fonksiyon sunmayı planladığımızı anlatır. void anahtar kelimesi fonksiyonun döndürdüğü bir değer olmadığını, boş parantezler ise fonksiyonun argüman almadığını bildirir. Fonksiyon deklarasyonu noktalı virgül ile biter. Yani fonksiyon bildirimi bir ifadedir.

Ayrıca, fonksiyon deklarasyonu bir prototiptir. Çünkü fonksiyon deklarasyonu, fonksiyonun bir taslağını, modelini sunar.

7.1.3. Fonksiyonu Çağırmak

7.1 'de verilen programı incelediğimizde, cizgi() fonksiyonunun main() fonksiyonu içinden üç kez çağırıldığını görüyoruz. Bu üç çağrının her biri aşağıdaki şekilde görünür:

cizgi();

Çağrının söz dizimi, deklarasyonda kullanılan ifade ile aynıdır. Çağrı noktalı virgül ile sona erer. Çağrı ifadesini çalıştırmak fonksiyonu çalıştırmak anlamına gelir; yani bu durumda kontrol fonksiyona geçer. Fonksiyonun çalışması bittiğinde kontrol fonksiyon çağrısını takip eden ifadeye geçer.

7.1.4. Fonksiyonun Tanımı

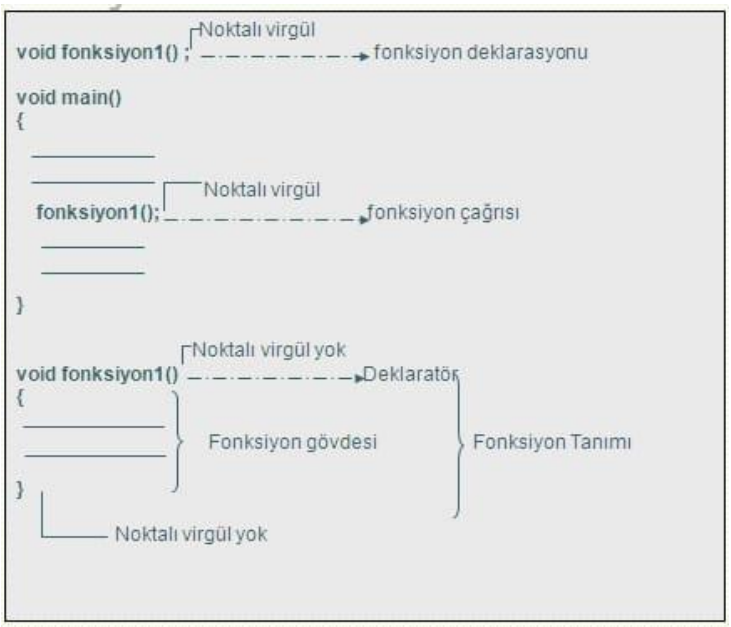
Fonksiyon tanımı, fonksiyonun asıl kodunu içerir. Çizgi() fonksiyonunun tanımı aşağıdaki gibidir.

```
void cizgi()
{
        for(int i=0;i<=50;i++)
          cout<<"=";
           cout<<endl;
}
```

Fonksiyon tanımı

Tanım deklaratör olarak adlandırılan bir satırdan ve bunu takip eden fonksiyon gövdesinden ibarettir. Fonksiyon gövdesi, küme parantezleri ile sınırlandırılmış, fonksiyonu oluşturan ifadelerden meydana gelir.

Burada dikkat edilmesi gereken husus, deklaratörün deklarasyonla uyuşmak zorunluluğudur. Deklaratör, aynı fonksiyon ismini kullanmalı, aynı tipte ve aynı sırada argüman içermeli(Eğer argüman kullanılmışsa) ve döndürdüğü değer aynı tipte olmalıdır. Aşağıda fonksiyon deklarasyonu, fonksiyon çağrısı ve fonksiyon tanımının söz dizimi verilmiştir.



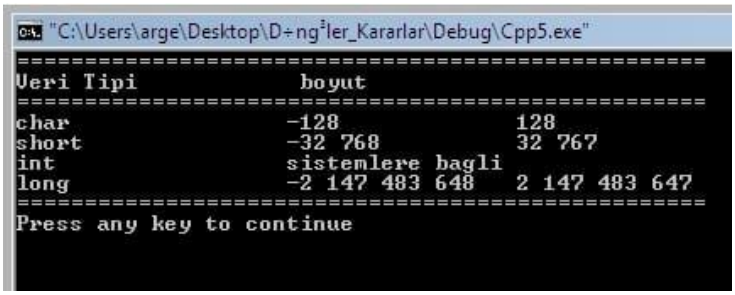
Fonksiyon çağrıldığında kontrol, fonksiyonun gövdesindeki ilk ifadeye aktarılır. Sonra fonksiyon gövdesindeki diğer ifadeler çağırılır. Kapanan parantezle karşılaşılınca kontrol fonksiyonu çağıran programa geri döner.

7.1.5. Deklarasyondan Kurtulmak

Fonksiyon tanımını, program listesi içinde, fonksiyona yapılan ilk çağrıdan önce yazarak fonksiyon deklarasyonundan kurtulabiliriz. Bu yaklaşım kısa programlar için daha basit olabilir. Yukarıda verilen programın deklarasyon kaldırılarak yazılan versiyonu aşağıda verilmiştir.

```
#include<iostream>
using namespace std;
void cizgi()
{
 for(int i=0;i<=50;i++)
 cout<<"=";
 cout<<endl;
}
int main()
{
cizgi();
cout<<"Veri Tipi"<<" "<<"boyut"<<endl;
cizgi();
cout<<"char -128 128"<<endl
 <<"short -32 768 32 767" <<endl
 <<"int sistemlere bagli "<<endl
 <<"long -2 147 483 648 2 147 483 647"<<endl;
cizgi();
return 0;
}
```

Program 7.1'in deklarasyon kullanmadan yazılan versiyonu



Yukarıdaki programın ekran çıktısı

7.2. Fonksiyonlara Argüman Aktarmak

Argüman, bir programdan bir fonksiyona aktarılan veri parçasıdır. Argümanlar bir fonksiyonun değişik değerlerle çalışmasına imkân verir, hatta onu çağıran programın isteklerine bağlı olarak değişik işler bile yapabilir.

7.2.1. Sabitleri Aktarmak

Yukarıdaki fonksiyonu her defasında 50 '=' basan bir fonksiyon yerine herhangi bir karakteri herhangi bir sayıda basacak bir fonksiyon haline getirebiliriz. Aşağıda, sabitleri argüman olarak aktaran bir program verilmiştir.

```
#include<iostream>
using namespace std;
void cizgi(char, int);
int main(){
cizgi('+',50);
cout<<"Veri Tipi"<<" "<<"boyut"<<endl;
cizgi('-',25);
cout<<"char -128 128"<<endl
 <<"short -32 768 32 767" <<endl
 <<"int sistemlere bagli "<<endl
 <<"long -2 147 483 648 2 147 483 647"<<endl;
cizgi('*',50);
return 0;
}
void cizgi(char ch, int n)
{
for(int i=1;i<=n; i++)
cout<<ch;
cout<<endl;
}
```

Fonksiyonlara sabitleri argüman olarak aktaran program



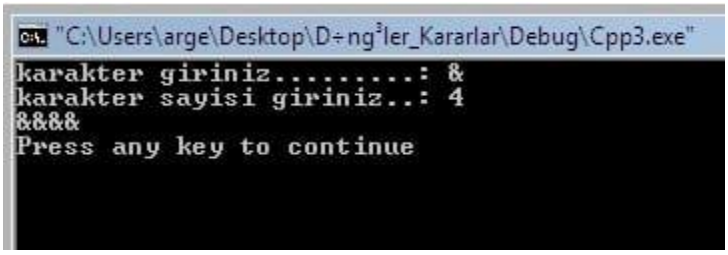
Fonksiyonlara sabitleri argüman olarak aktaran programın ekran çıktısı

7.2.2. Değişkenleri Argüman Olarak Aktarmak

Aşağıdaki örnekte karakterler ve karakterlerin kaç kez tekrarlanacağını kullanıcının belirlemesine izin veren bir örnek verilmiştir. Argüman olarak kullanılan değişkenlerin veri tipleri, tıpkı sabitlerde olduğu gibi, fonksiyon deklarasyonunda ve tanımında belirtilen tiplerle eşleşmelidir.

```
#include<iostream>
using namespace std;
void cizgi(char,int);
int main()
{
 char ch; int n;
 cout<<"karakter giriniz.....: ";
 cin>>ch;
 cout<<"karakter sayisi giriniz..: ";
 cin>>n;
 cizgi(ch,n);
 return 0;
}
void cizgi(char fch, int fn)
{
 for(int i=1;i<=fn; i++)
 cout<<fch;
 cout<<endl;
}
```

Fonksiyonlara değişkenleri argüman olarak aktaran program



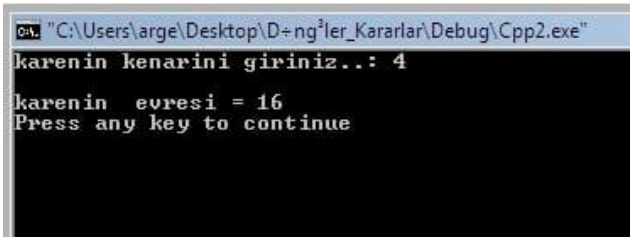
Fonksiyonlara sabitleri argüman olarak aktaran programın ekran çıktısı

7.3. Fonksiyonlardan Değer Döndürmek

Bir fonksiyon çalışmasını tamamladıktan sonra, kendisini çağıran programa tek bir değer döndürebilir. Genellikle döndürülen bu değer, fonksiyonun çözdüğü problemin cevabını içerir. Aşağıdaki programdaki fonksiyon değer döndüren bir fonksiyondur.

```
#include<iostream>
using namespace std;
float fcevre(float);
int main()
{
 float kenar, cevre;
 cout<<"karenin kenarini giriniz..: ";
 cin>>kenar;
 cevre=fcevre(kenar);
 cout<<endl<<"karenin çevresi = "<<cevre<<endl;
 return 0;
}
float fcevre(float a)
{
 float sonuc;
 sonuc=4*a;
 return sonuc;
}
```

Fonksiyonlardan değer döndüren program



Fonksiyonlardan değer döndüren programın ekran çıktısı

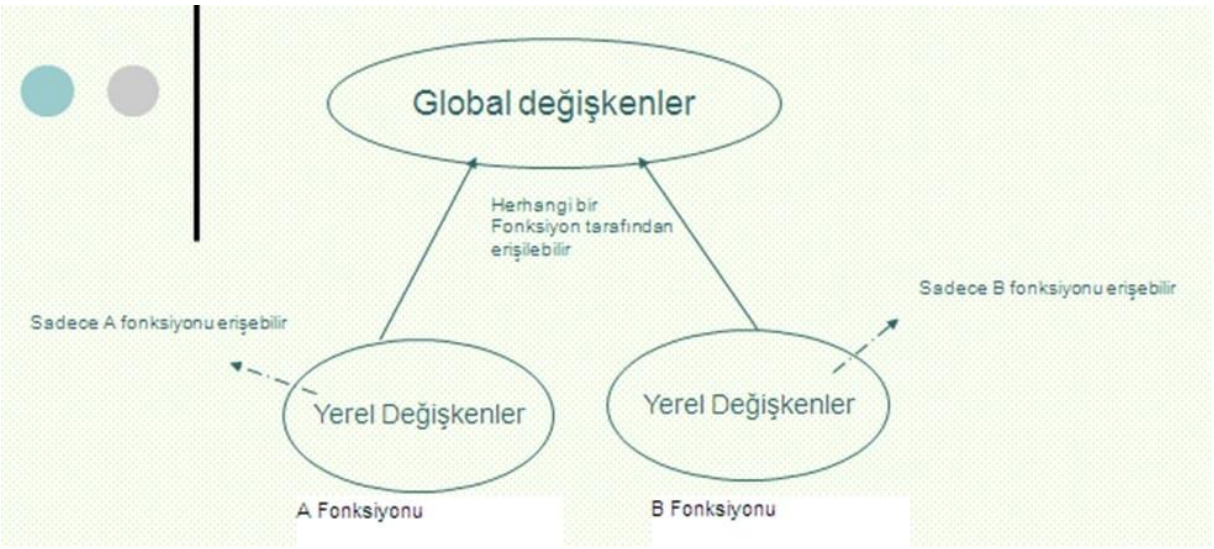
Bir fonksiyon bir değer döndürecekse, bu değerin veri tipi açıkça belirlenmelidir. Fonksiyon deklarasyonu bunu, veri tipini fonksiyon deklarasyonunda ve tanımında fonksiyon isminden önce yerleştirerek gerçekleştirir.

7.3.1. return İfadesi

fcevre() fonksiyonuna bir argüman aktarılır ve bu değer ***a*** parametresinde saklanır. Fonksiyon içerisindeki ifadeler çalıştırılır. Hesaplanan değer ***sonuc*** değişkeninde saklanır. Bu değişkenin değeri daha sonra, fonksiyonu çağıran main() fonksiyona return ifadesi ile döndürülür.

return sonuc;

Hem main() fonksiyonunda hem de fcevre() fonksiyonunda cevre değişkenini saklamak için bir yerin mevcut olduğuna dikkat ediniz. fcevre fonksiyonunda sonuc değişkeni main fonksiyonunda da cevre değişkeni hesaplanan değeri saklar. Fonksiyon döndüğü zaman sonuc içindeki değer cevre içine kopyalanır. Fonksiyonu çağıran program, fonksiyonun içindeki sonuc değişkenine erişemez. Sadece değişkenin değeri döndürülür. Bu işlem aşağıda gösterilmiştir.



Bir fonksiyona birçok argüman aktarılırken, fonksiyondan sadece bir değer döner. Fonksiyonlardan birden fazla değer döndürmenin başka yöntemleri vardır. Ancak bu yöntemler ileri C++ konuları içinde yer alır.

7.3.2. Gereksiz Değişkenlerden Kurtulmak

Yukarıdaki programda programın daha iyi anlaşılması için kullanılan, ancak gerekli olmayan birkaç tane değişken vardır. Aşağıdaki programda gereksiz değişkenler çıkartılmıştır.

```
#include<iostream>

using namespace std;

float fcevre(float);

int main()

{

 float kenar;

 cout<<"karenin kenarini giriniz..: ";

 cin>>kenar;

 cout<<endl<<"karenin çevresi = "<<fcevre(kenar)<<endl; //fcevre fonksiyonu cout'a doğrudan eklenir.

 return 0;

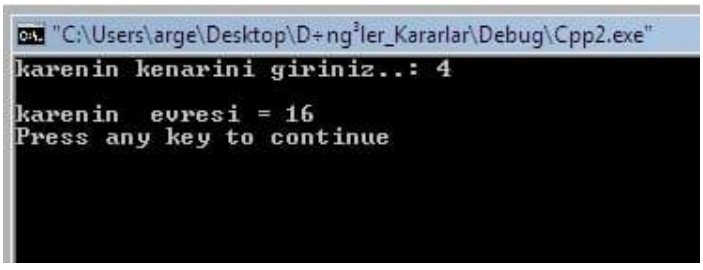
}

float fcevre(float a)

{

 return 4*a; // dogrudan deger döndürülür

 }
```



Gereksiz değişken kullanılmadan yazılan programın ekran çıktısı

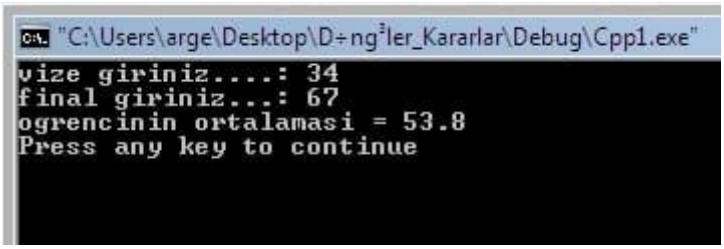
main içerisindeki cevre değişkeni çıkartılmış ve bunun yerine *cout<<endl<<"karenin çevresi = "<<fcevre(kenar)<<endl;* ifadesi yazılmıştır. fcevre() fonksiyonundan sonuç değişkeni çıkartılmış ve doğrudan *return 4*a;* ifadesi yazılmıştır. Bu programın çalıştırılması sonucunda yukarıdaki programın çalıştırılmasından elde edilen sonuçlar elde edilir.

7.4. Yerel(Inline) Fonksiyonlar

Kısa fonksiyonlar kullanıldığında, programın çalışmasını hızlandırmak için fonksiyon gövdesini doğrudan fonksiyonu çağıran programın kodunun içine yerel(inline) olarak yerleştirmek daha iyi olabilir. Bu şekilde daha hızlı çalışan programlar elde edebiliriz. Bunun için yalnızca yazılan fonksiyonun inline olduğunu belirtmek yeterlidir. Derleyici derleme sırasında inline fonksiyonu çağıran fonksiyonun içerisine yerleştirir.

```
#include<iostream>
using namespace std;
inline float ortalama(float vize, float final)
{
 return 0.4*vize+0.6*final;
}
int main()
{
 int vize, final;
 cout<<"vize giriniz....: ";cin>>vize;
 cout<<"final giriniz...: ";cin>>final;
 cout<<"ögrencinin ortalaması = "<<ortalama(vize,final)<<endl;
 return 0;
}
```

İçerisinde inline fonksiyon bulunan C++ programı



İçerisinde inline fonksiyon bulunan C++ programın ekran çıktısı

7.5. Fonksiyonların Aşırı Yüklenmesi

Aşırı yüklenen fonksiyonlar konusu fonksiyonların ilgi çeken konularından biridir. Fonksiyonların aşırı yüklenmesi, ayni isimdeki fonksiyonların farklı parametre sayısı veya farklı parametre tipinden yararlanılarak programların içerisine birden fazla yazılmasını sağlar. Programların içerisinde aşırı yüklenen fonksiyonların yazılması programların okunabilirliğini arttırır. Aşağıda, içerisinde aşırı yüklenen fonksiyon bulunan bir program gösterilmiştir.

```
/* Fonksiyonlar aşırı yükleniyor */
#include <iostream>
using namespace std;
void cizgiciz();
void cizgiciz(char);
void cizgiciz(char, int);
int main()
{
    cizgiciz();
    cizgiciz('-');
    cizgiciz('=', 20);

    return 0;
}
void cizgiciz()
{
    cizgiciz('*', 10);
}
void cizgiciz(char k)
{
    cizgiciz(k, 10);
}
void cizgiciz(char k, int s)
{
    for (int i = 0; i < s; i++) cout << k;
    cout << endl;
}
```

Fonksiyonların aşırı yüklenmesi

7.6. Kapsam(Scope) ve Depolama Sınıfı

Bir *değişkenin kapsamı*, değişkene programın hangi bölümlerinin erişebileceğini belirler. Değişkenin *depolama sınıfı* ise değişkenin ne kadar süre ile var olacağını gösterir. Aşağıda, programlarda kullanılan değişkenler bu açıdan ele alınmıştır.

7.6.1. Yerel Değişkenler

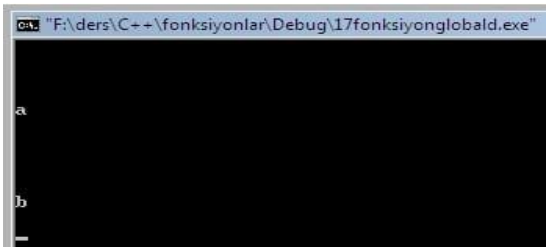
Bir fonksiyonun içinde tanımlanan değişkenlere yerel(local) değişkenler denir. Çünkü bu değişkenler yerel kapsama sahiptir. Şimdiye kadar yazdığımız programlarda kullanılan değişkenlerin tamamı kullanıldıkları fonksiyonun gövdesi içende tanımlanmıştır. Bir yerel değişken, içinde tanımlandığı fonksiyon çağırılana kadar ortaya çıkmaz. Yanı bu değişkenler için fonksiyon çağırılana kadar bellekte yer ayrılmaz. Kontrol, yerel değişkenin tanımlı olduğu fonksiyona geçtiğinde değişken ortaya çıkar; yani bellekte değişken için yer ayrılır. Kontrol tekrar fonksiyonu çağıran programa geçtiğinde, değişkenler ortadan kaldırılır ve değerleri yitirilir. Bu değişkenlere *otomatik* ismi de verilir. Çünkü değişkenler fonksiyon çağırıldığında otomatik olarak oluşturulur ve fonksiyon geri döndüğünde otomatik olarak ortadan kaldırılır. Değişkenlerin oluşturulması ve yok edilmesi arasında geçen süreye yaşam süresi denir. Yerel bir değişkenin yaşam süresi, içinde tanımlı olduğu fonksiyonun çalışma süresi ile aynıdır. Değişkenlerin yaşam sürelerinin bu şekilde kısıtlanmış olması, bellek alanını daha verimli olarak kullanılması sonucunu doğurur.

Bir değişkenin kapsamı(Scope) değişkenin erişilebilirliği olarak adlandırılır. Bir fonksiyon içinde tanımlı değişkenler, sadece tanımlı oldukları fonksiyon içinde görülebilirler. Bu durum, bir fonksiyon içerisinde tanımlı değişkenin değerinin, başka bir fonksiyon tarafından değiştirilememesi sonucunu doğurur. Yani güvenliği arttırır.

7.6.2. Global Değişkenler

Global değişkenler bir fonksiyonun dışında tanımlanan değişkenlerdir. Bir global değişkene dosya bir dosya içerisindeki bütün fonksiyonlar tarafından erişilebilir. Bu değişkenlere bazen harici(External) değişkenler de denir. Çünkü bu değişkenler herhangi bir fonksiyonun dışında tanımlanır. Aşağıda global değişken kullanan bir program verilmiştir.

```
#include<iostream>
using namespace std;
#include<conio.h>
char ch = 'a';
void oku();
void yaz();
int main(){
 while(ch != '\r') {
 oku();
 cout<<endl<<endl<<endl<<endl;
 yaz();
 cout<<endl<<endl;
 }
return 0;
}
void oku() {
 ch=getch();
 }
void yaz() {
 cout<<ch; }
```



İçerisinde global değişken tanımlanmış programın ekran çıktısı

Eğer bir global değişkene ilk değer atanmış ise:

int sayi=15;

değer atama işlemi, program ilk kez yüklendiğinde yapılır. Eğer global değişkene program içerisinde herhangi bir değer atanmamış ise bu durumda değişkenin değeri sıfır olur. Yerel değişkenlere değer atanmadığı zaman, değerlerinin rastgele, anlamsız birer değer olduğunu hatırlayınız.

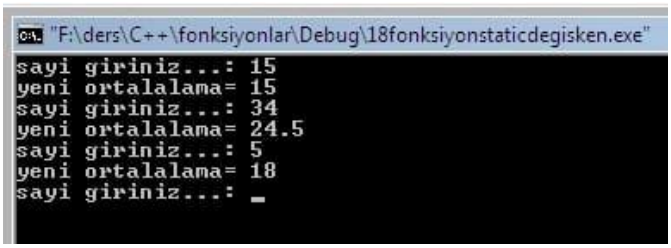
Global değişkenler programın yaşam süresi boyunca varlıklarını sürdürürler. Program çalıştırıldığında global değişken için bellekte yer ayrılır ve bu yer, program çalıştığı sürece bu değişken için ayrılır. Yine bu değişkenler tanımlandığı dosya boyunca erişilebilirdir.

7.6.3. Statik Yerel Değişkenler

Statik yerel değişkenler, otomatik yerel değişkenler ile aynı erişilebilirliğe sahiptir. Ancak statik yerel değişkenlerin yaşam süresi global değişkenlerle aynıdır. Yaşam süresi açısından bakıldığında statik yerel değişkenlerin global değişkenlerden farkı global değişken program çalıştırıldıktan hemen sonra ortaya çıkar, statik yerel değişken ise tanımlı

olduğu fonksiyon çağırılana kadar ortaya çıkmaz. Aşağıda statik yerel içerisinde statik yerel değişken tanımlı bir C++ programı verilmiştir.

```
#include<iostream>
using namespace std;
float ortalama(float);
int main() {
float sayi=1,ort;
while(sayi !=0) {
cout<<"sayi giriniz...: "; cin>>sayi;
if(sayi!=0){
ort=ortalama(sayi);
cout<<"yeni ortalalama= "<<ort<<endl; } }
 return 0;
}
float ortalama(float yenisayi) {
 static float toplam=0;
 static int sayac=0;
 sayac++;
 toplam+=yenisayi;
 return toplam/sayac;
}
```



İçerisinde statik yerel değişken tanımlı programın ekran çıktısı

Bölüm Özeti

Bu bölümde fonksiyonların, yapısal programlamanın temel taşı olduğunu, programlar içerisinde yapılan tekrarlardan programcıları koruduğunu, fonksiyonlar halinde yazılan programların bilgisayarda çalışırken maliyetlerinin daha düşük olduğunu öğrendiniz.

Fonksiyon deklare etmeyi, tanımlamayı, çağırmayı, fonksiyonlara argüman aktarmayı, çağıran fonksiyona değer döndürmeyi örnekler çözerek öğrendiniz.

Ayrıca fonksiyonları aşırı yükleyerek daha okunabilir programlar yazma becerisi kazandınız. Yazacağınız fonksiyonların kısa olması durumunda bu fonksiyonları yerel olarak yazarak programlarınıza performans kazandırdınız.

Bunlara ilave olarak, bu bölümün sonunda, değişkenlerin program içerisinde tanımlandıkları yerlerin önemli olduğunu, yerel değişkenlere, sadece tanımlandıkları yerden ulaşılabildiğini, global değişkenlere ise bütün fonksiyonlardan ulaşılabildiğini gördük. Bunların hangilerini ne zaman tercih edeceğimiz konusunda bilgi edindik.

ÜNİTE SORULARI

1. void ciz(); şeklinde bir fonksiyon deklarasyonu yapılmıştır. Buna göre, bu deklarasyon dikkate alınarak, yapılacak bir fonksiyon tanımı için aşağıdakilerden hangisi söylenebilir?

- Tanımlanacak fonksiyon, çağıran fonksiyona değer döndürebilir
- Tanımlanacak fonksiyon iki parametre alır
- Tanımlanacak fonksiyon parametre almaz
- Tanımlanacak fonksiyon bir parametre alır
- Tanımlanacak fonksiyon çağırıldığında çalışmaz

2.

```
1  #include <iostream>
2  using namespace std;
3  int topla(int,int);
4  int main() {
5      int s1=4,s2=9;
6      cout<<"Sayıların toplamı = "<<topla(s1,s2)<<endl;
7      return 0;
8  }
9  int topla(int s1, int s2)
10 {
11     return s1+s2;
12 }
```

Yukarıda verilen programda fonksiyon deklarasyonu kaçıncı satırda yapılmıştır?

- 11
- 9
- 6
- 4
- 3

3.

```
1  #include <iostream>
2  using namespace std;
3  int topla(int,int);
4  int main() {
5      int s1=4,s2=9;
6      cout<<"Sayıların toplamı = "<<topla(s1,s2)<<endl;
7      return 0;
8  }
9  int topla(int s1, int s2)
10 {
11     return s1+s2;
12 }
```

Yukarıda verilen programda, fonksiyon çağrısı hangi satırda yapılmıştır?

- 11
- 9
- 6
- 4
- 3

4.

```
1   #include <iostream>
2   using namespace std;
3   int topla(int,int);
4   int main() {
5       int s1=4,s2=9;
6       cout<<"Sayıların toplamı = "<<topla(s1,s2)<<endl;
7       return 0;
8   }
9   int topla(int s1, int s2)
10  {
11      return s1+s2;
12  }
```

Yukarıdaki verilen programda, çağıran fonksiyona değer döndüren satır hangisidir?

- 11
- 9
- 6
- 4
- 3

5. Aşağıda, üç adet aşırı yüklenmiş fonksiyon çağrısı verilmiştir. Bu çağrılardan hangisinde çağırılan fonksiyona aktarılan argümanların ikisi de sabit olarak belirlenmiştir.

I. Kare();
II. Kare(islem,40);
III. Kare('+' ,40);

- Yalnızca I
- Yalnızca II
- I ve II
- Yalnızca III
- II ve III
    -
    - CEVAP ANAHTARI1. c2. e3. c4. a5. d

8. SIRALAMA VE ARAMA ALGORİTMALARI

Giriş

Sıralama ve arama algoritmaları bilgisayar biliminin en önemli konularının içerisinde sayılabilir. Bu bölümde sıralama ve arama algoritmalarına örnekler verilerek düzensiz (sırasız) durumdaki verilerin nasıl sıralanacağını ve bulunması istenen bir değerin sıralı ve sırasız düzende bulunan veri yapısı içerisinde nasıl aranacağını öğreneceğiz.

Sıralama algoritmaları düzensiz olarak verilmiş olan bir grup sayıyı veya metinsel ifadeyi küçükten büyüğe veya büyükten küçüğe sıralanmasını gerçekleştiren algoritmalardır. Bu amaçla geliştirilmiş çeşitli sıralama algoritmaları bulunmaktadır. Seçerek sıralama (selection sort), Kabarcık Sıralaması (Bubble Sort), Yerleştirmeli Sıralama (Insertion Sort), hızlı sıralama (quick sort), birleştirerek sıralama (merge sort), yığın sıralaması (heap sort) gibi algoritmalar temel sıralama algoritmalarıdır. Bu bölümde konunun daha iyi anlaşılabilmesi için, elemanları sayı türünde veri olan dizilerin sıralanmasına örnekler verilecek ve bu örneklerde Seçerek sıralama (selection sort), Kabarcık Sıralaması (Bubble Sort), Yerleştirmeli Sıralama (Insertion Sort) algoritmaları kullanılacaktır.

Ayrıca, bu ünitede bir veri yapısı içerisinde aranan bir elemanın var olup olmadığının tespiti için kullanılan arama algoritmalarına örnekler verilecektir. Arama algoritmalarının kavranabilmesi için verilen örneklerde iki ayrı temel arama algoritması kullanılacaktır. Kullanılacak bu arama algoritmaları doğrusal arama (sequential search) ve ikili arama (binary search) algoritmalarıdır.

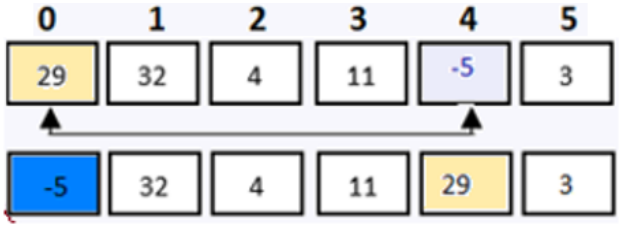
8.1. Seçerek Sıralama (Selection Sort)

Seçerek sıralama algoritması sıralama problemine getirilen en basit çözümdür. Algoritmanın çalışması sırasında dizinin iki bölüme ayrıldığı varsayılır. Bu bölümlerden ilki sıralanmış dizi, ikincisi ise sıralanmamış dizidir. Başlangıçta sıralanmış dizi boş, sıralanmamış dizi ise, dizinin elemanlarının tamamından oluşur.

Seçerek sıralama algoritmasının çalıştırılmasına, dizinin ilk elemanının seçilmesi ile başlanır. Seçilen dizi elemanı ile bu elemandan sonra gelen dizi elemanları karşılaştırılır ve en küçük eleman bulunur. Bulunan bu en küçük eleman seçilen eleman ile yer değiştirir. Bu işlemin sonucunda dizinin en küçük elemanı belirlenerek dizinin başına yazılmış olur. Dizinin başına yazılmış olan bu eleman sıralanmış dizinin ilk elemanıdır.

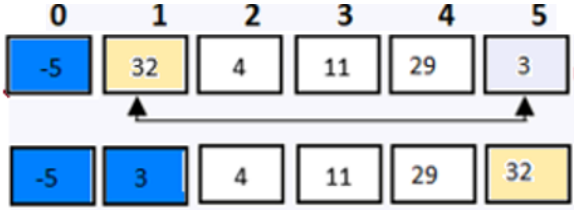
Seçerek sıralama algoritmasının çalışması şekil 8.1'de verilen örnek üzerinde adım adım gösterilmiştir. Şekilden(1. Adım'a bakınız) de görülebileceği gibi başlangıçta dizinin **0**. Elemanı olan **29** seçilmiştir. Sonra, seçilen elemandan sonra gelen dizi elemanları içerisinden en küçük olan eleman bulunmuştur. Örnekte bu eleman **-5**'tir. **-5** dizinin dördüncü elemanıdır.

İlk adımda bulunan en küçük sayı **(-5)**, dizinin **0**'ıncı elemanı **29** ile yer değiştirir ve dizinin elemanlarının sıralaması aşağıdaki gibi olur.

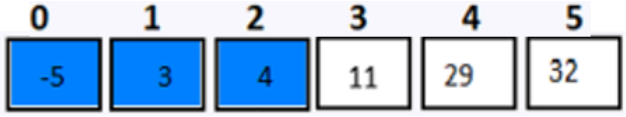


Yukarıdaki işlemden sonra dizinin **1.** Elemanı olan **32** seçilir ve **32**'den sonra gelen dizi elemanları içerisinden en küçük eleman bulunur. Örnekte bu eleman **3**'tür. **3** dizinin **5**. elemanıdır.

Bu adımda bulunan en küçük sayı **(3)** dizinin **5**. elemanı **32** ile yer değiştirir ve dizinin elemanlarının sıralaması aşağıdaki gibi olur.



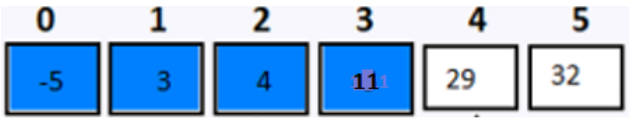
Üçüncü adımda dizinin **2**. elemanı olan **4** seçilir ve **4**'den sonra gelen dizi elemanları içerisinden en küçük eleman araştırılır. Örnekte 4'ten sonra 4'ten küçük eleman yoktur. Dolayısı ile bu adımda dizi elemanları yer değiştirmez. Bu adımdan sonraki dizi elemanlarının sıralı (mavi renkte olan kısım) ve sıralı olmayan kısmı (beyaz renkte olan kısım) aşağıda gösterilmiştir.



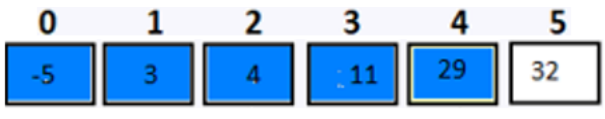
Bundan sonraki dördüncü ve beşinci adımlarda da üçüncü adıma benzer şekilde seçilen elemandan sonraki dizi elemanları seçilen elemandan daha küçük olmadığı için dizi elemanları arasında herhangi bir yer değişikliği işlemi yapılmamıştır.

Aşağıda bu adımlar sonunda dizinin durumu gösterilmiştir.

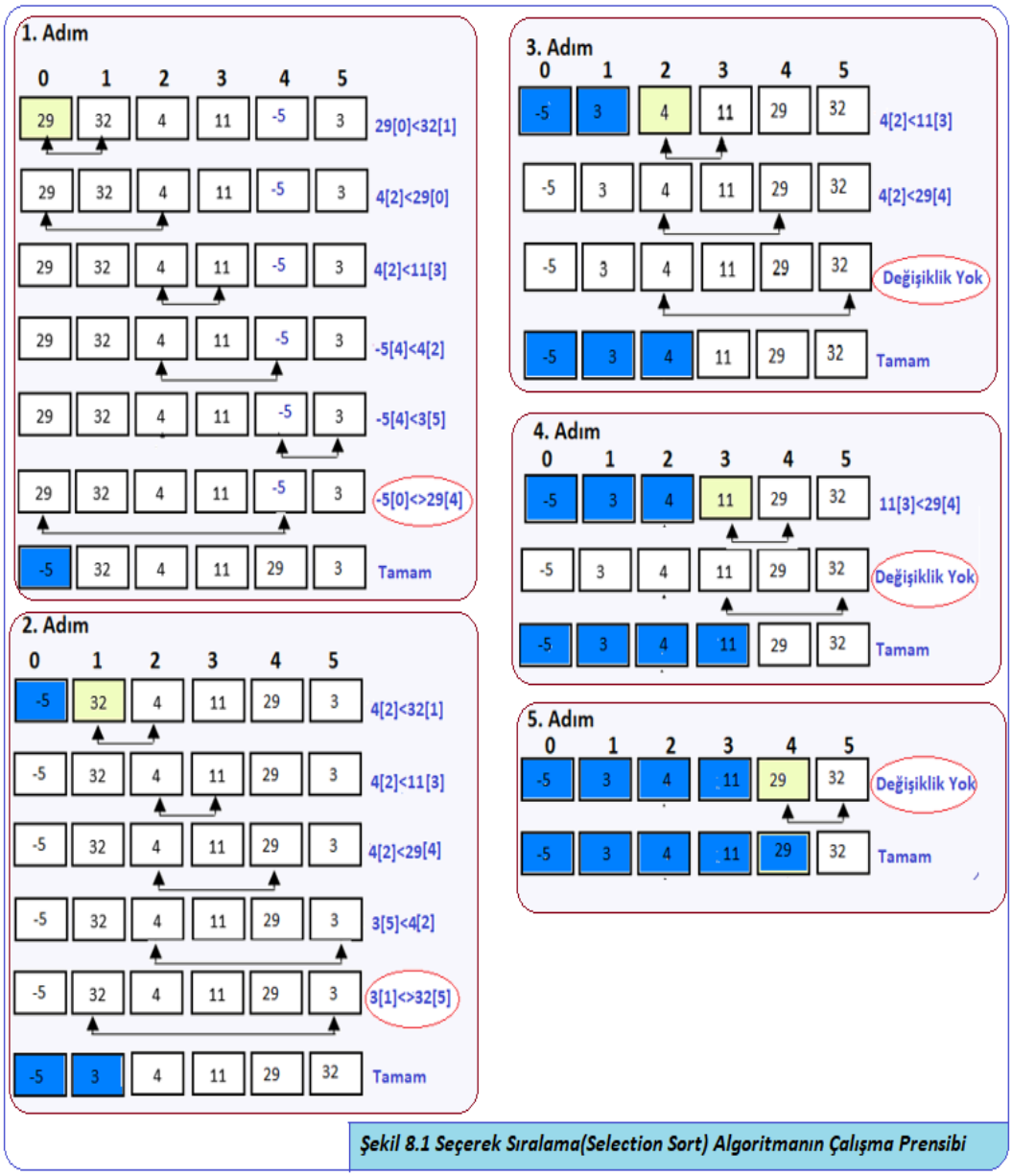
4. Adım sonunda dizi:



5. Adım Sonunda dizi:

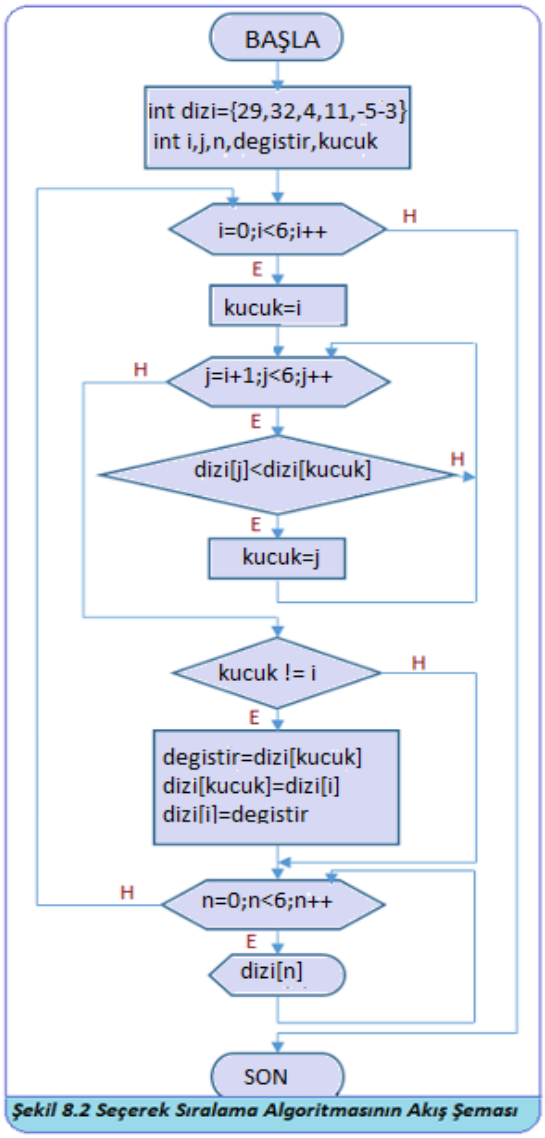


Şekil 8.1'de Seçerek Sıralama (Selection Sort) Algoritmasının çalışma şekli bütün olarak gösterilmiştir.



Şekil 8.1 Seçerek Sıralama(Selection Sort) Algoritmanın Çalışma Prensibi

Şekil 8.2'de Seçerek Sıralama (Selection Sort) algoritmasının akış şeması gösterilmiştir. Akış şeması dikkatli olarak incelendiğinde verilen dizinin elemanlarını sıralayabilmek için algoritmada iç içe çalışan iki adet for döngüsü kullanıldığı görülmektedir. Algoritmanın sonundaki for döngüsünün ise, sıralanmış diziyi ekrana yazdırmak için kullanıldığına dikkat ediniz.



Şekil 8.2 Seçerek Sıralama Algoritmasının Akış Şeması

Aşağıda Seçerek sıralama algoritmasının C++ kodu ve ekran çıktısı verilmiştir.

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       int dizi[6]={29,32,4,11,-5,3};
5       int i,j,n,degistir,kucuk;
6       for(i=0;i<6;i++){
7           kucuk=i;
8           for(j=i+1;j<6;j++){
9               if(dizi[j]<dizi[kucuk]){
10                  kucuk=j;
11              }
12          }
13          if(kucuk!=i){
14              degistir=dizi[kucuk];
15              dizi[kucuk]=dizi[i];
16              dizi[i]=degistir;
17          }
18              for(n=0;n<6;n++)
19              cout<<dizi[n]<<" ";
20              cout<<endl;
21      }
22      return 0;
23  }
```

Program 8.1 Seçerek Sıralama Algoritmasının C++ Kodu

```
-5 32 4 11 29 3
-5 3 4 11 29 32
-5 3 4 11 29 32
-5 3 4 11 29 32
-5 3 4 11 29 32
-5 3 4 11 29 32
```

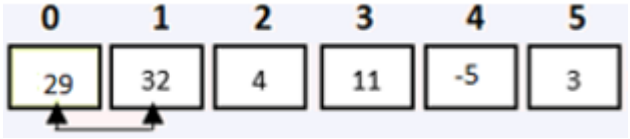
Program 8.1'in Ekran Çıktısı

1 **Sıra Sizde!** Yukarıda verilen ve Seçerek sıralama yöntemi ile küçükten büyüğe sıralan dizinin **büyükten küçüğe** sıralanmasını sağlayan algoritmanın akış şemasını çiziniz ve C++ kodunu yazınız?

8.2. Kabarcık Sıralaması (Bubble Sort)

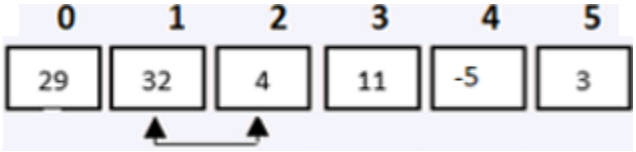
Kabarcık Sıralaması algoritması ilk önce Yer Değiştirme (Exchange Sort) algoritması olarak isimlendirilmiş, daha sonra dizi içerisindeki büyük elemanların algoritmanın her adımında dizi sonuna kadar doğrusal olarak ilerlemesinden dolayı algoritmaya Kabarcık Sıralaması (Bubble Sort) denmiştir.

Kabarcık sıralaması algoritmasında dizinin küçükten büyüğe sıralanması aşağıdaki şekilde gerçekleşir; Önce dizinin ilk elemanı (dizinin 0. elemanı) seçilir ve bu eleman kendisinden sonra gelen elemanla (dizinin 1. Elemanı) karşılaştırılır. Karşılaştırma sonunda ilk eleman, kendisinden sonra gelen elemandan büyük ise ilk eleman ile kendisinden sonra gelen eleman yer değiştirir. İlk eleman kendisinden sonra gelen elemandan küçük ise yer değiştirme işlemi yapılmaz.

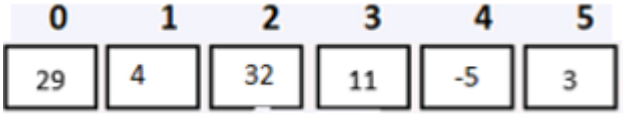


Bu işlemden sonra dizinin birinci elemanı dizinin ikinci elemanı ile karşılaştırılır. Karşılaştırma sonucunda birinci eleman ikinci elemandan büyük ise birinci eleman ile ikinci eleman yer değiştirir. Eğer dizinin birinci elemanı dizinin ikinci elemandan daha küçük ise yer değiştirme işlemi yapılmaz.

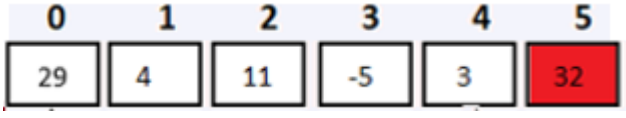
Verilen örnekte dizinin birinci elemanı (32) kendisinden sonra gelen elemandan büyüktür. Dolayısı ile dizinin birinci elemanı ile kendisinden sonra gelen eleman yer değiştirecektir.



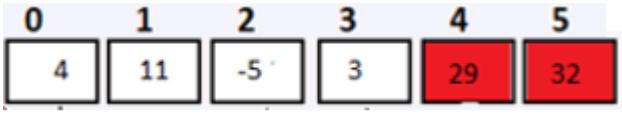
Yer değiştirme işleminin sonunda dizi elemanlarının sıralanışı aşağıdaki gibi olacaktır.



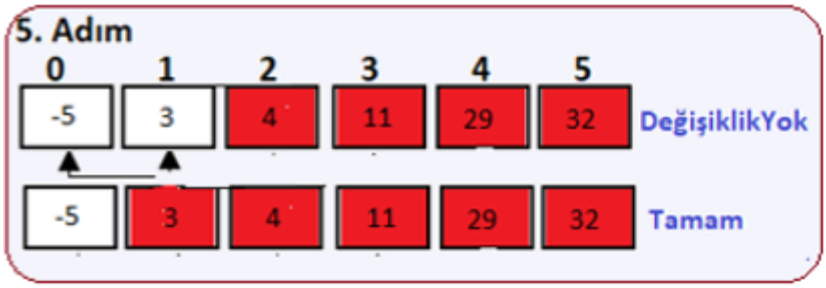
Kabarcık sıralaması yöntemi ile sıralama işleminde, yukarıdaki işlemler dizi sonuna kadar tekrarlanır. 1. Adımın sonunda dizinin en büyük elemanı dizinin sonuna yerleştirilmiş olur (Bakınız Şekil 8.2, 1. Adım). Bir sonraki adımda karşılaştırma işleminin sınırı bir eleman geri çekilir.



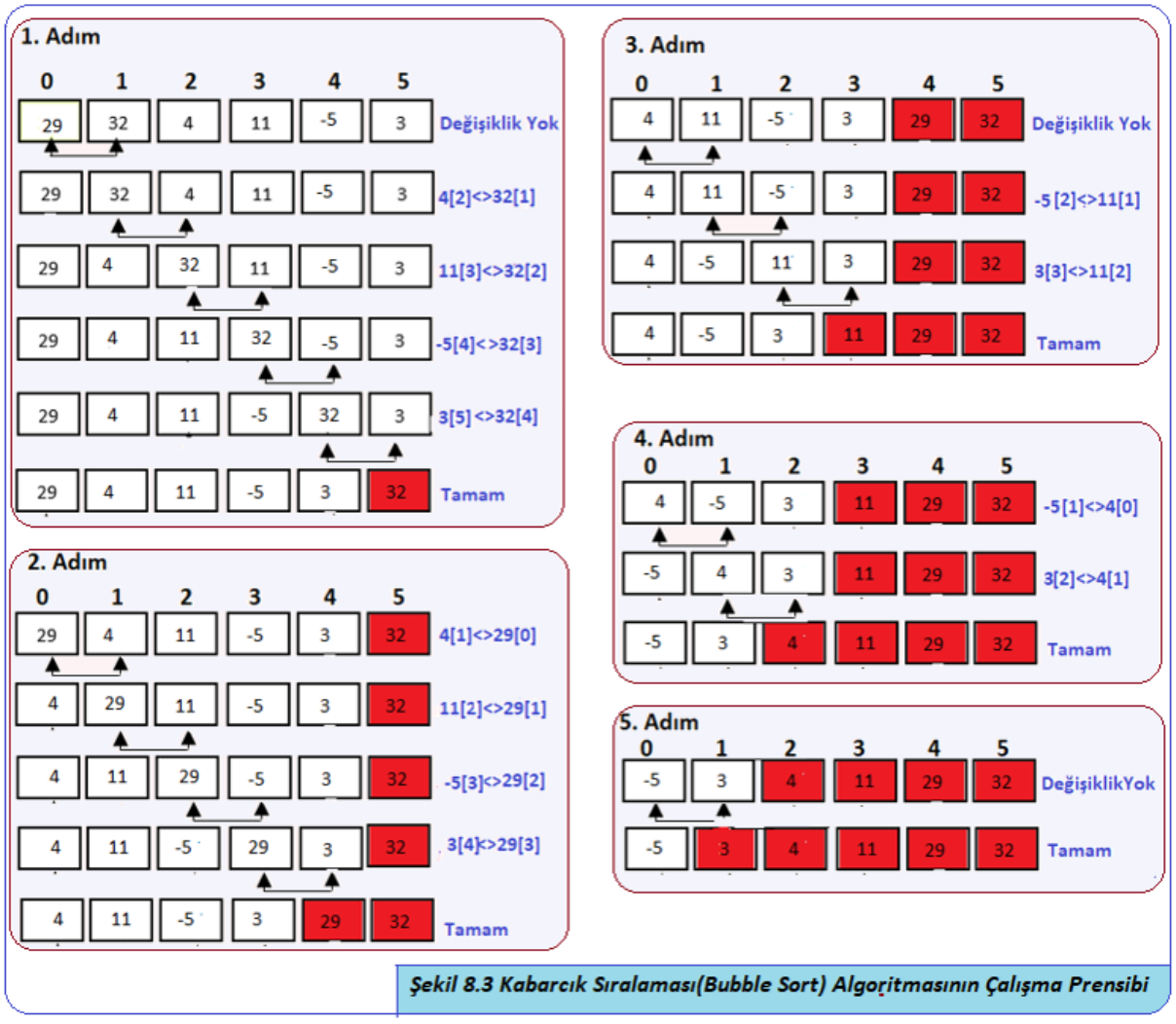
İkinci adımda karşılaştırma işlemine yine dizinin ilk elemanından başlanır ve yukarıda açıklanan kurallara göre ikili karşılaştırmaların sonunda yer değişiklikleri yapılarak dizinin ikinci büyük elemanı 4. elemanın yerine geçer.



Yukarıda açıklanan işlemler, bu örnek için üçüncü adımda, dördüncü ve beşinci adımda tekrarlanarak beşinci adımın sonunda başlangıçta sırasız olarak verilen dizi küçükten büyüğe sıralanmış olacaktır.



Şekil 8.2'de Kabarcık Sıralamasının (Bubble Sort) Algoritmasının çalışma şekli bütün olarak gösterilmiştir.



Şekil 8.3 Kabarcık Sıralaması(Bubble Sort) Algoritmasının Çalışma Prensibi

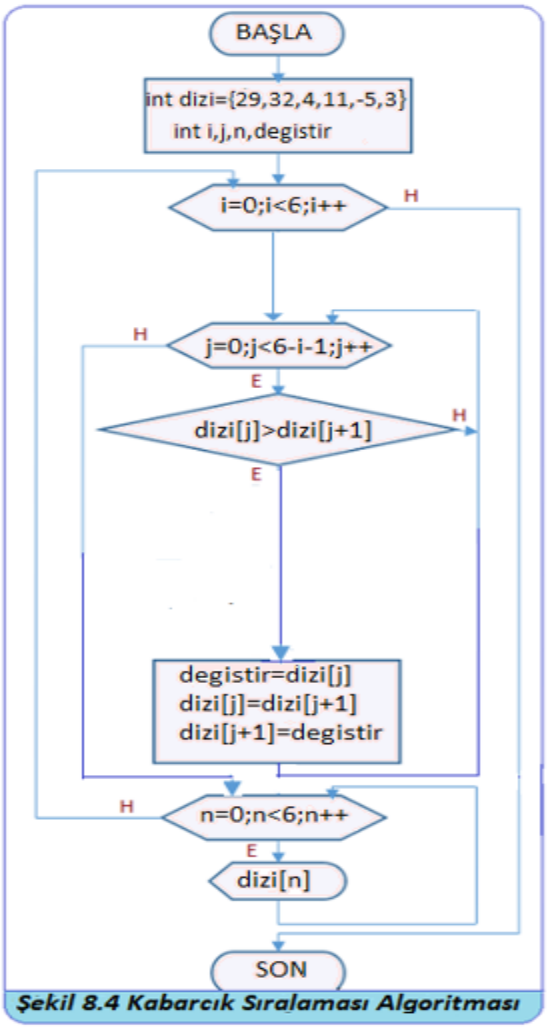
```cpp
#include <iostream>
using namespace std;
int main() {
    int dizi[6]={29,32,4,11,-5,3};
    int i,j,n,degistir;
    for(i=0;i<6;i++){
        for(j=0;j<6-i-1;j++)
        {
            if(dizi[j]>dizi[j+1])
            {
                degistir=dizi[j];
                dizi[j]=dizi[j+1];
                dizi[j+1]=degistir;
            }
        }
        for(n=0;n<6;n++)
            cout<<dizi[n]<<" ";
            cout<<endl;
    }
    return 0;
}
```

Program 8.2 Kabarcık Sıralaması Algoritmasının C++ Kodu

```
29 4 11 -5 3 32
4 11 -5 3 29 32
4 -5 3 11 29 32
-5 3 4 11 29 32
-5 3 4 11 29 32
-5 3 4 11 29 32
```

Program 8.2'nin Ekran Çıktısı

Yukarıda Kabarcık sıralamasının C++ kodu ve kodun ekran çıktısı verilmiştir. Aşağıda ise Kabarcık sıralamasının akış şeması verilmiştir. Akış şeması incelendiğinde kabarcık sıralamasının da seçerek sıralamada olduğu gibi iç içe çalışan iki adat for döngüsü ile gerçekleştirildiği görülmektedir. Akış şemasında sonda yer alan for döngüsünün sıralanmış diziyi ekrana yazdırmak için kullanıldığına dikkat ediniz.



Şekil 8.4 Kabarcık Sıralaması Algoritması

2 **Sıra Sizde!** Yukarıda verilen ve kabarcık sıralaması yöntemi ile küçükten büyüğe sıralan dizinin, aynı yöntemle **büyükten küçüğe** sıralanmasını sağlayan algoritmanın akış şemasını çiziniz ve C++ kodunu yazınız?

8.3. Yerleştirmeli Sıralama (Insertion Sort)

Yerleştirmeli sıralama algoritmasının çalıştırılmasına dizinin ikinci elemanı seçilerek başlanır. Seçilen ikinci eleman kendinden önceki elemanlarla karşılaştırılır ve dizinin büyük elemanları sağa kaydırılır. Bu kaydırma sonucunda açılan uygun pozisyona o anda sıralanmakta olan eleman yerleştirilir. Şekil 8.5'te 6 elemanlı sıralı olmayan bir dizi verilmiştir ve bu dizinin elemanlarının yerleştirmeli sıralama yöntemi ile sıralanması adım adım gösterilmiştir.

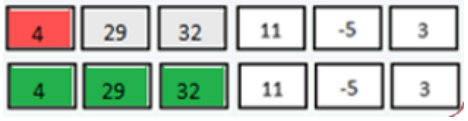
İlk adımda dizinin 1. elemanı (32) seçilmiş ve kendisinden gelen önceki elemanla (29) karşılaştırılmış 29 32'den küçük olduğu için dizi elemanlarının yerlerinde herhangi bir değişiklik yapılmamıştır.



İkinci adımda dizinin 2. Elemanı (4) seçilmiş ve kendisinden önce gelen dizinin 1. ve 0. elemanları ile karşılaştırılmıştır.



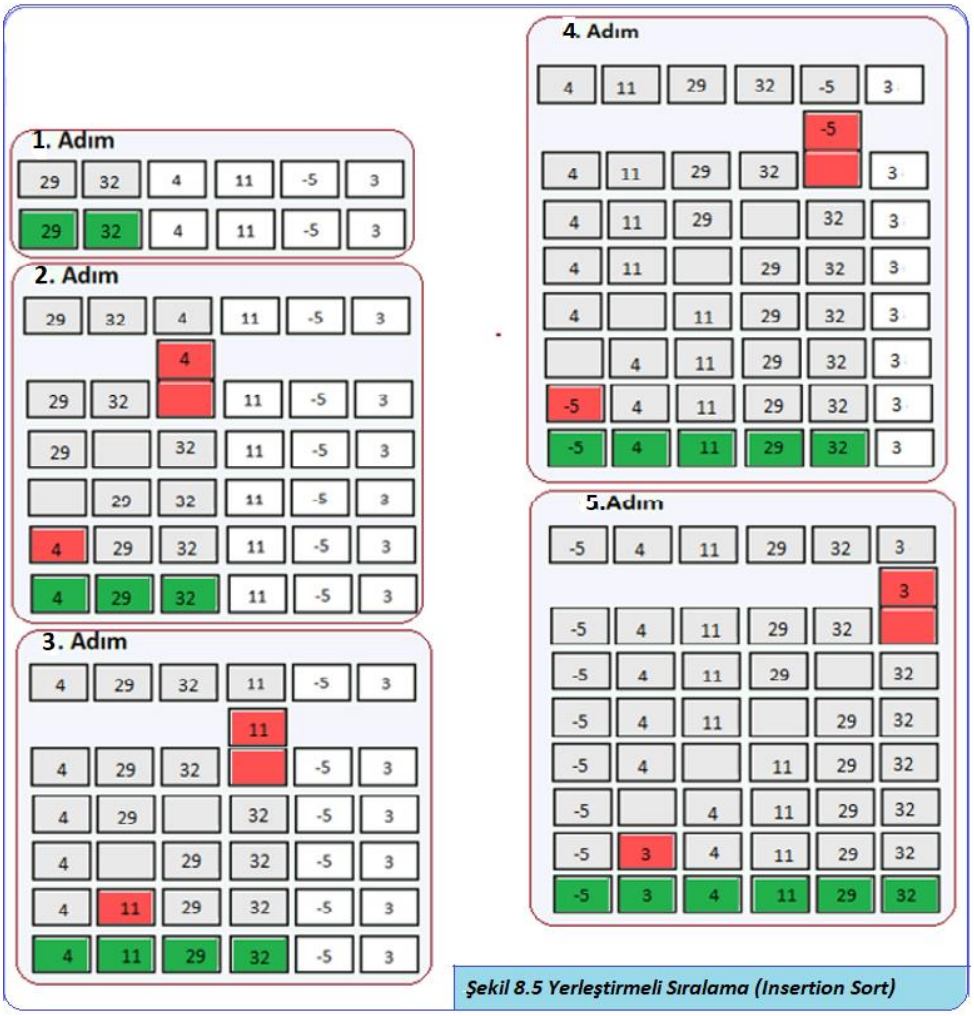
Karşılaştırma sonucunda önce dizinin 1. elemanı olan 32, dizinin ikinci elemanı olan 4'ün yerine, sonra da dizinin 0. elemanı olan 29, dizinini birinci elemanı olan 32'nin yerine gelmiştir. Seçilmiş olan dizi elemanı bu durumda dizinin 0. elemanı olmuştur. İkinci adımın sonunda dizinin sıralaması aşağıdaki gibi olmuştur.



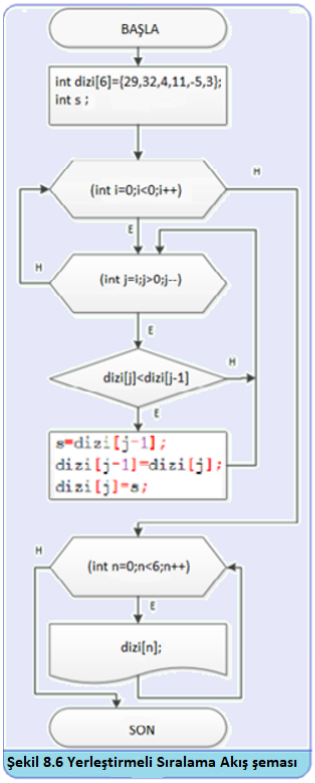
Bu işlemler aynı kurallarla üçüncü, dördüncü ve beşinci adımlarda sürdürülerek beşinci adımın sonunda dizi küçükten büyüğe sıralanmıştır.



Şekil 8.3'de Yerleştirmeli Sıralama (Insertion Sort) Algoritmasının çalışma şekli bütün olarak gösterilmiştir.



Şekil 8.5 Yerleştirmeli Sıralama (Insertion Sort)



Şekil 8.6 Yerleştirmeli Sıralama Akış şeması

```cpp
#include <iostream>
using namespace std;
int main() {
    int dizi[6]={29,32,4,11,-5,3},s,n;
    for(n=0;n<6;n++)
        cout<<dizi[n]<<" ";
        cout<<endl;
    for(int i=1;i<6;i++){
        for(int j=i;j>0;j--){
        if(dizi[j]<dizi[j-1])
        {
            s=dizi[j-1];
            dizi[j-1]=dizi[j];
            dizi[j]=s;
        }
        /*  else
        {
            break;
        } */
        }
        for(n=0;n<6;n++)
        cout<<dizi[n]<<" ";
        cout<<endl;
    }
    return 0;
}
```

Program 8.3 Yerleştirmeli Sıralama C++ Kodu

```
29 32 4 11 -5 3
29 32 4 11 -5 3
4 29 32 11 -5 3
4 11 29 32 -5 3
-5 4 11 29 32 3
-5 3 4 11 29 32
```

Program 8.3'ün Ekran Çıktısı

Şekil 8.6'de Yerleştirmeli Sıralama (Insertion Sort) algoritmasının akış şeması gösterilmiştir. Akış şeması incelendiğinde seçerek sıralama ve kabarcık sıralamasında olduğu gibi yerleştirmeli sıralamada da verilen dizinin elemanlarını sıralayabilmek için algoritmada iç içe çalışan iki adet **for** döngüsü kullanılmıştır. Algoritmanın sonundaki **for** döngüsün bu örnekte de, sıralanmış diziyi ekrana yazdırmak için kullanılmıştır.

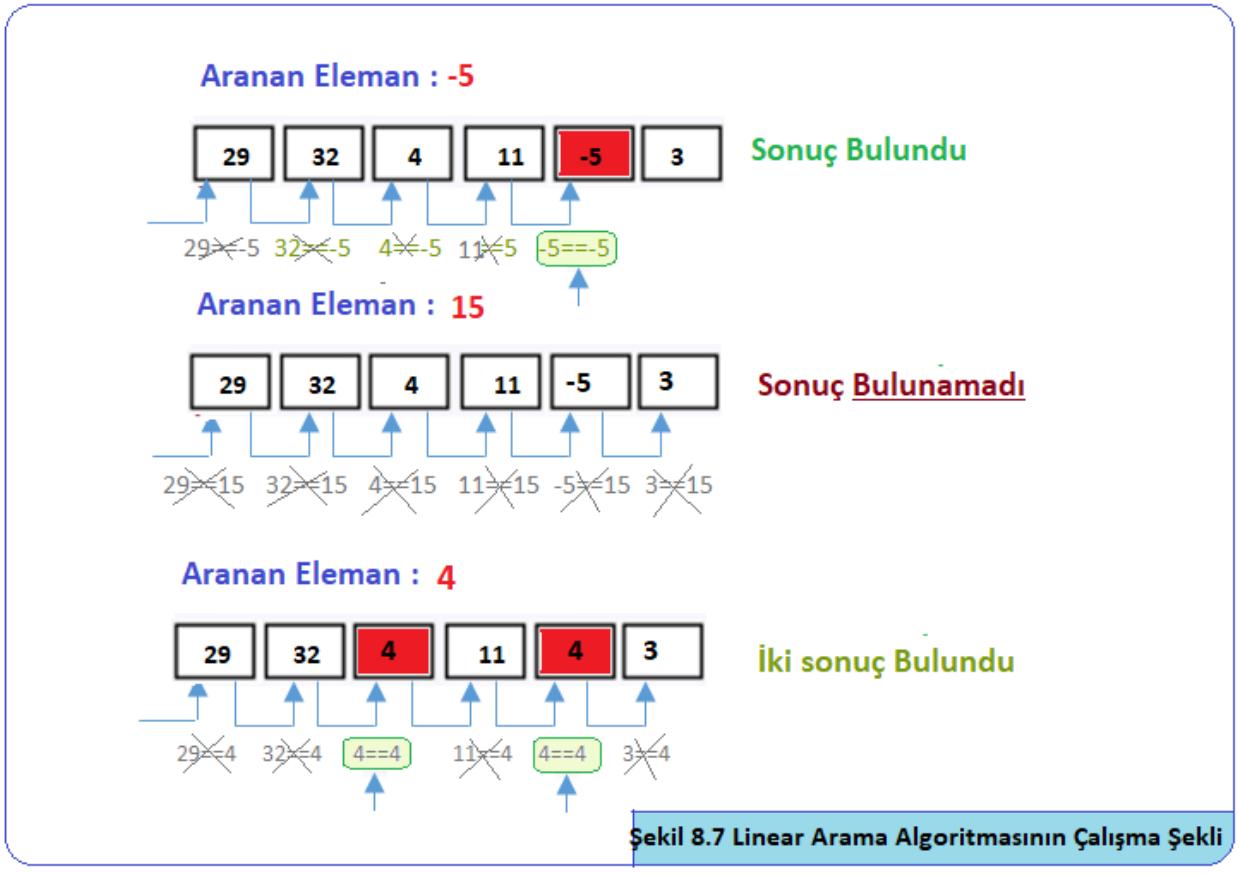
Program 8.3 Yerleştireli sıralama algoritmasının C++ koduna Örnek olarak verilmiştir.

3 **Sıra Sizde!** Yukarıda verilen ve yerleştirmeli sıralama (Insertion Sort) yöntemi ile küçükten büyüğe sıralan dizinin, aynı yöntemle **büyükten küçüğe** sıralanmasını sağlayan algoritmanın akış şemasını çiziniz ve C++ kodunu yazınız?

8.4. Doğrusal Arama (Sequential Search)

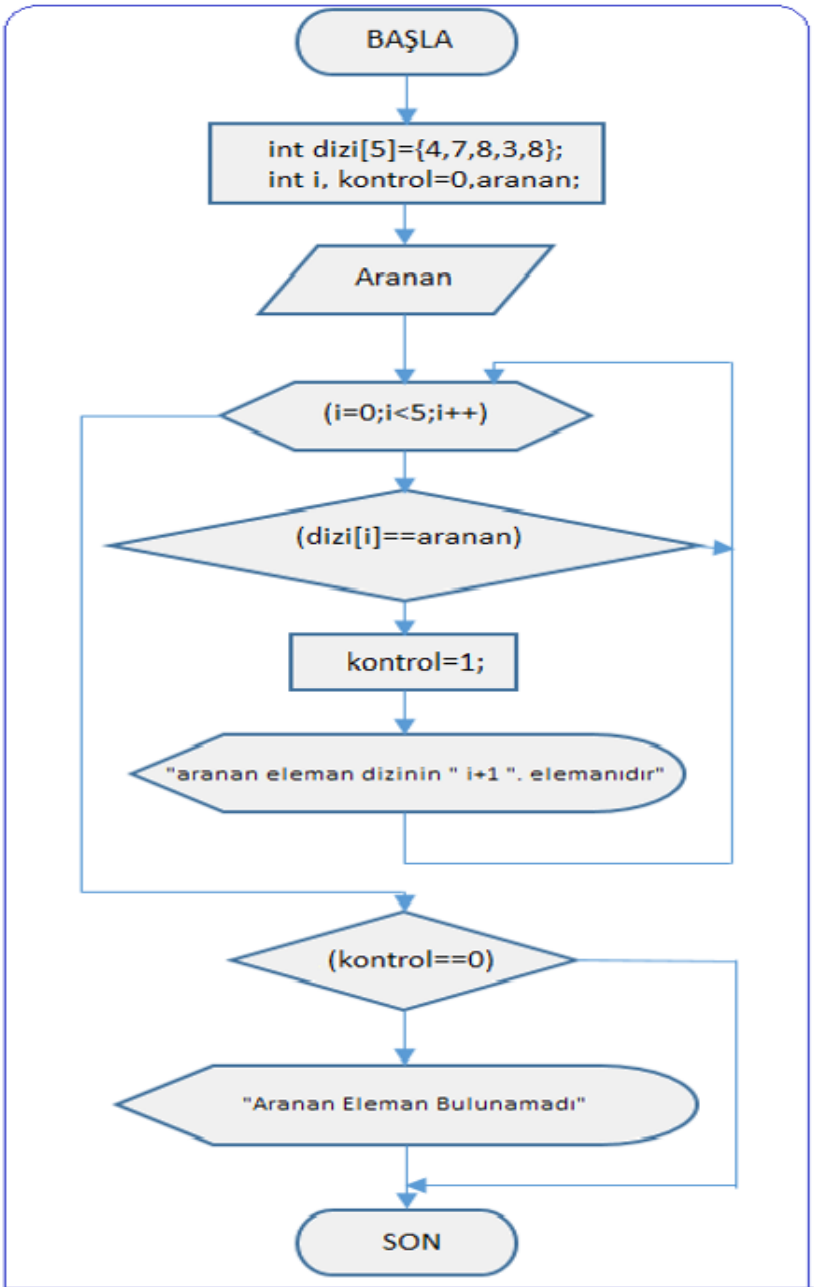
Lineer arama olarak da isimlendirilen bu algoritma en temel arama algoritması olarak bilinir. Doğrusal arama algoritması sıralı veya sıralı olmayan diziler üzerinde arama yapmak için kullanılabilir. Doğrusal aramada aranan eleman, sırayla dizinin elemanları ile karşılaştırılır. Aranan elemanın değerinin dizi elemanlarından birine eşit olduğu noktada algoritmanın çalışması sonlandırılır. Bazı durumlarda aranan eleman dizi elemanları arasında bulunmayabilir. Bu durumda karşılaştırmalar dizinin ilk elemanından son elemana kadar sürdürülür.

Üzerinde doğrusal arama yapılan dizinin elemanları arasında birden fazla aynı değere sahip eleman varsa ve bunların tamamının konumunun bulunması amaçlanıyorsa bu durumda da karşılaştırma işlemi dizinin ilk elemanından son elemana kadar sürdürülür. Şekil 8.7'de doğrusal Aramanın çalışma şekli gösterilmiştir.



Şekil 8.7 Linear Arama Algoritmasının Çalışma Şekli

Şekil 8.8'de doğrusal aramanın akış şeması, program 8.4'te doğrusal aramanın C++ kodu verilmiştir.



Şekil 8.8 ***Lineer Arama Algoritmasının Akış Şeması***

```cpp
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
     int dizi[5]={4,7,8,3,8},i,kontrol=0,aranan;
    cout<<" Aranan Sayıyı Giriniz : ";
    cin>>aranan;
    for(i=0;i<5;i++)
    {
        if(dizi[i]==aranan){
            cout<<"aranan eleman dizide indisi "<<i<<" olan elemandır"<<endl;
            kontrol=1;
            // break;
                }
    }
    if(kontrol==0)
    cout<<"aranan eleman bulunamadı"<<endl;
    return 0;
}
```

Program 8.4 Lineer Arama Algoritmasının C++ kodu

```
 Aranan Sayıyı Giriniz : 3
aranan eleman dizide indisi 3 olan elemandır

--------------------------------
Process exited after 7.068 seconds with return value 0
Press any key to continue
```

Program 8.4'ün Ekran Çıktısı

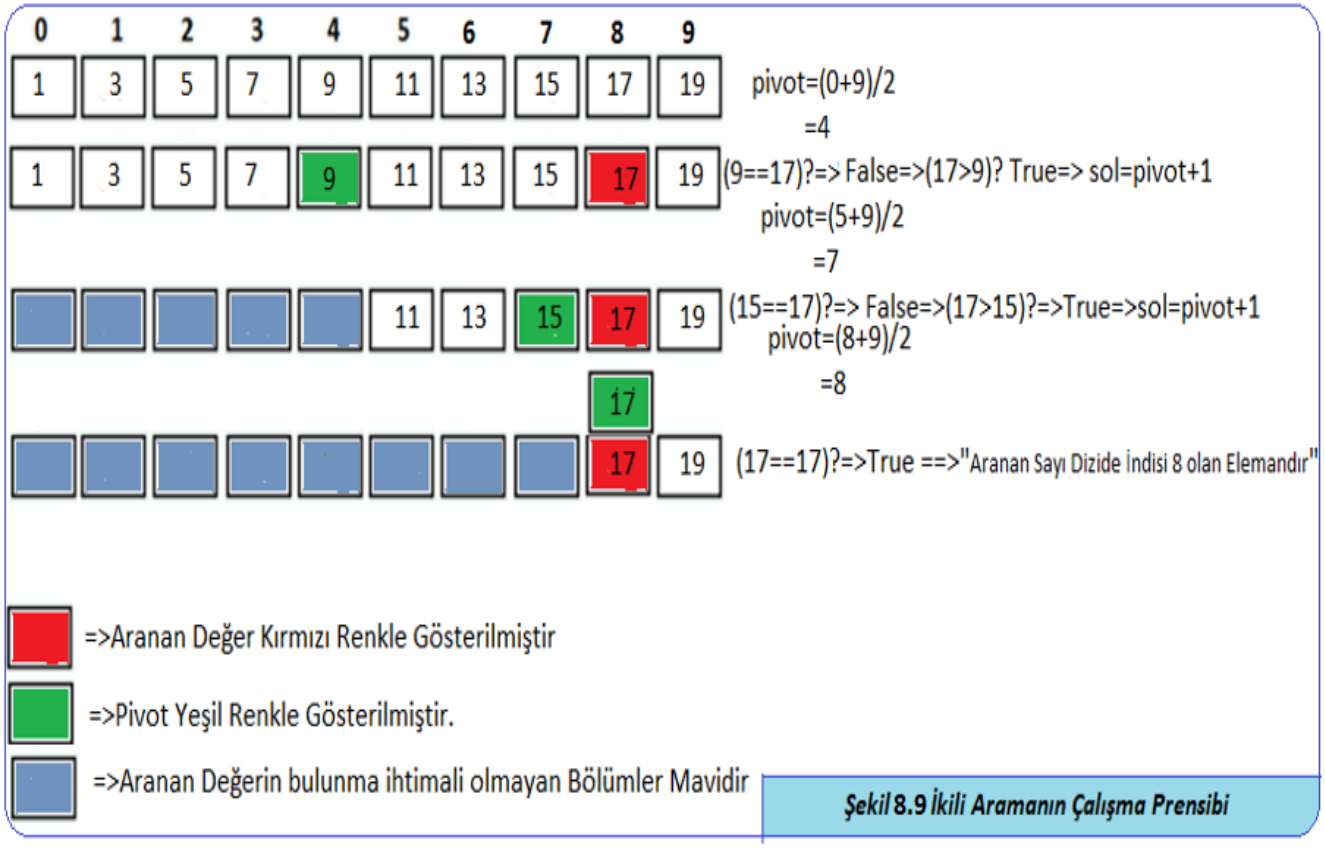
8.5. İkili Arama (Binary Search)

İkili arama algoritması doğrusal arama algoritmasından farklı olarak sadece sıralı diziler üzerinde arama yapan bir algoritmadır. İkili arama parçala fethet prensibi ile çalışır. Sıralı bir dizi üzerinde ikili arama yöntemi ile arama yaparken dizi önce ortadan ikiye bölünür ve aranan değerin ortadaki bu sayıya eşit olup olmadığına bakılır. Eğer bu karşılaştırma sonunda arana değer dizinin ortasındaki değere eşit ise aranan sayı bulunmuştur, arama sonlandırılır. Eşit değil ise aranan değerin ortadaki bu sayıdan büyük veya küçük mü olduğu araştırılır.

Eğer Aranan değer ortadaki sayıdan büyük ise arama ortadaki değerin sağında yapılır. Bunun için ortadaki değerin sağında kalan kısım için yeni bir orta değer bulunur ve arama için ilk adımda açıklanan işlemler tekrarlanır.

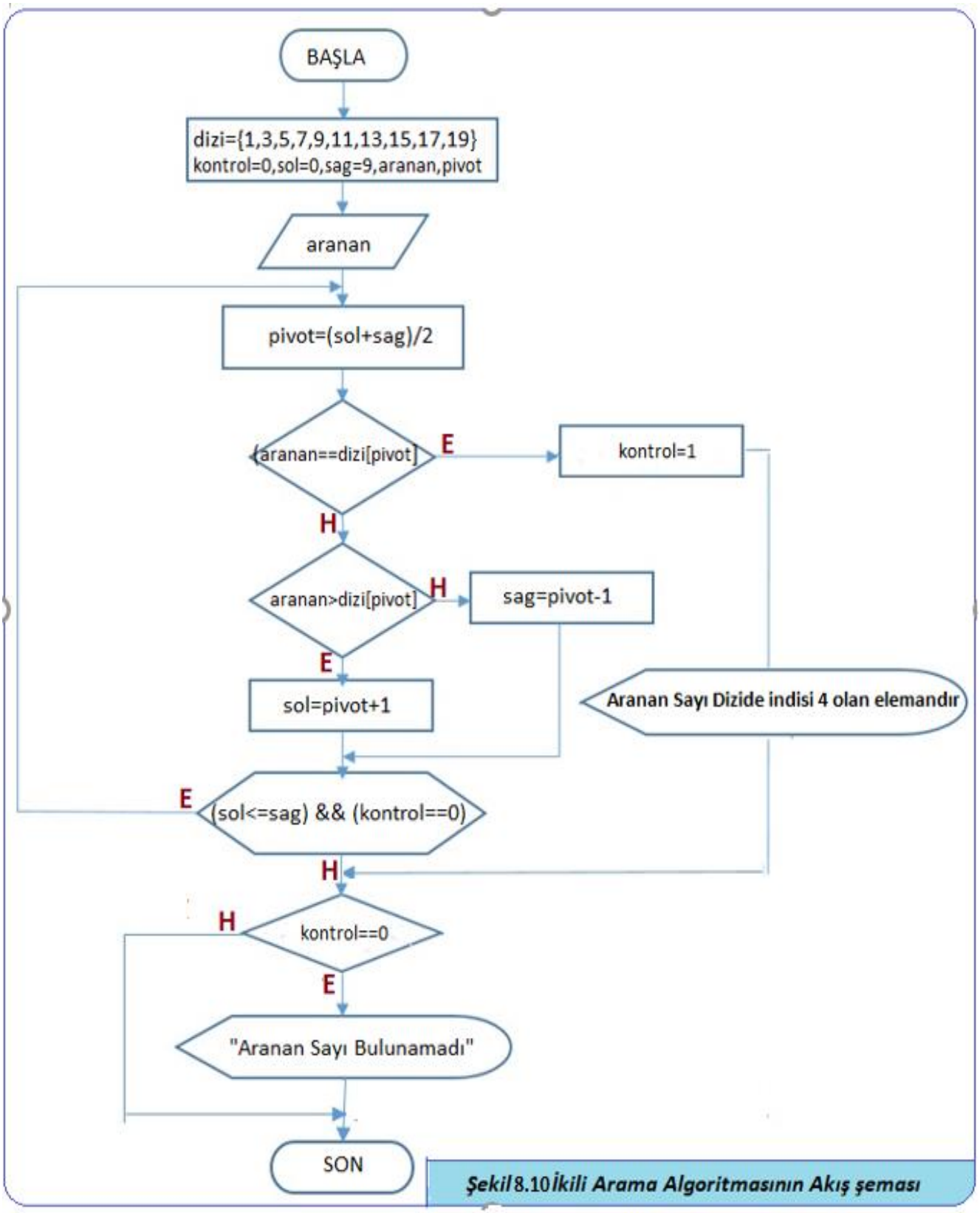
Eğer Aranan değer ortadaki sayıdan küçük ise arama ortadaki değerin solunda yapılır. Bunun için ortadaki değerin sağında kalan kısım için yeni bir orta değer bulunur ve arama için ilk adımda açıklanan işlemler tekrarlanır.

Arama işlemi, dizi üzerinde aranan değer bulunana kadar yukarıda açıklanan prensiplerle sürdürülür. Şekil 8.9'da ikili arama algoritmasının çalışma prensibi gösterilmiştir.



Şekil 8.9 İkili Aramanın Çalışma Prensibi

Şekil 8.10'da ikili arama algoritmasının akış şeması verilmiştir.



Şekil 8.10 İkili Arama Algoritmasının Akış şeması

İkili arama algoritmasının C++ kodu ve C++ kodunun ekran çıktısı aşağıda verilmiştir.

```cpp
#include <iostream>
#include <locale.h>
using namespace std;
int main ( )
{
    setlocale(LC_ALL,"Turkish");
    int dizi[10]= {1,3,5,7,9,11,13,15,17,19};
    int aranan,kontrol=0,sol=0,sag=9,pivot;
    cout<<"Aranan sayiyi girin: ";
    cin>>aranan;
    do {
        pivot=(sol+sag)/2;
        if(aranan ==dizi[pivot]){
            kontrol=1;
            cout<<"aranan eleman dizide indisi ";
            cout<<pivot<<" olan elemandır"<<endl;
            break;
        }
        else if(aranan>dizi[pivot]){
            sol=pivot+1;}
        else{
            sag=pivot-1;}
    }while ((sol<=sag) && (kontrol==0));
        if (kontrol==0)
        cout<<"Aranan sayı bulunamadı!";
    return 0;
}
```

Program 8.5 İkili Arama Algoritmasının Akış Şeması

```
Aranan sayiyi girin: 7
aranan eleman dizide indisi 3 olan elemandır

--------------------------------
Process exited after 6.902 seconds with return value 0
Press any key to continue
```

Program 8.5'in Ekran Çıktısı

Bölüm Özeti

Bu bölümde ilk önce sıralama algoritmalarını ele aldık ve konuyu üç temel sıralama algoritması üzerinden açıkladık. Sırala algoritmalarının nasıl çalıştığını açıklarken önce algoritmanın çalışma prensibini şekil üzerinde gösterdik. Ele alınan sıralama algoritmasının çalışma mantığını öğrenmiş olmak bize hem algoritmanın akış şemasını çizerken hem de C++ kodunu yazarken önemli kolaylıklar sağladı.

Sıralama ve Arama Algoritmaları bölümünde ele alınan ilk sıralama algoritması Seçerek Sıralama algoritması oldu. Seçerek sıralama algoritmasının çalışma prensibinin ilk önce dizinin ilk elemanının ele alınması ve bu elemandan sonra geriye kalan dizi elemanları içerisinden en küçük elemanın bulunup ele alınan elemanla en küçük elemanın yer değiştirmesi, daha sonra dizinin ikinci elemanın ele alınıp işlemlerin bu şekilde sürdürülmesi şeklinde olduğu görüldü.

Ele alınan ikinci sıralama algoritması, Kabarcık sıralaması oldu. Kabarcık sıralaması algoritmasının çalışma prensibinin dizi elemanlarından komşu iki elemanın karşılaştırılması, karşılaştırma sonucunda elemanların uygun yerlerine yerleştirilmesi ve bu işlemin dizi elemanlarının sıralanması tamamlanıncaya kadar sürdürülmesine dayandığı gösterildi.

Sıralama algoritmalarında ele alınan üçüncü ve son algoritma yerleştirmeli sıralama algoritması oldu. Bu algoritmanın çalışma prensibinin ise, dizinin ilk elemanın yerinde bırakılması, dizinin ikinci elemanın seçilmesi, seçilen ikinci elemanın kendinden önceki elemanlarla karşılaştırılması, dizinin büyük elemanlarının sağa kaydırılması, daha sonra dizinin üçüncü sıradaki elemanının seçilmesi ve seçilen elemanın kendinden önce gelen elemanlarla karşılaştırılarak işlemlerin dizi sıralanıncaya kadar sürdürülmesi şeklinde olduğu açıklandı.

Sıralama algoritmalarından sonra bu bölümde ele alınan ikinci önemli ana başlık Arama algoritmaları konusu oldu. Arama algoritmalarından doğrusal arama ve ikili arama algoritmaları ele alınarak diziler üzerinde arama işlemlerinin nasıl yapılacağı bu bölümde anlatılan ikinci önemli konuydu.

Doğrusal arama algoritmasının, aranan elemanın dizinin ilk elemanından başlanarak sırayla karşılaştırılması ve aranan elemana eşit dizi elemanı bulunduğunda aramanın sonlandırılması prensibine dayandığı şekil üzerinde anlatıldı, algoritmanın akış şeması çizildi ve C++ kodu yazıldı.

Bu bölümün son konusu olarak ikili arama algoritması ele alındı. İkili arama algoritması sıralı veriler üzerinde arama yapan bir algoritmadır. Algoritmanın çalışma prensibi, ilk önce dizinin ortasındaki elemanın aranan eleman olup olmadığına bakılır. Eğer aranan eleman dizinin ortasında ise arama işlemi sonlandırılır. Eğer aranan eleman ortadaki eleman değilse aranan veri, ya ortadaki elemanın solunda ya da ortadaki elemanın sağında olabilir. Dolayısıyla bu bilgi kullanılarak dizinin yarısı arama işleminden çıkartılır. Dizinin kalan kısmında yeni bir orta değer hesaplanır ve işlemler bu şekilde, aranan eleman bulunana kadar sürdürülür.

ÜNİTE SORULARI

1. Elemanları [3, 9, 11, 4,2] olan bir dizi, seçerek sıralama algoritması ile küçükten büyüğe doğru sıralanmak istenildiğinde, algoritmanın adımları sonrasında elde edilecek diziler aşağıdaki seçeneklerin hangisinde doğru sırayla verilmiştir?

- [2 3 4 9 11], [2 3 4 11 9], [2 3 11 4 9], [2 9 11 4 3]
- [2 9 11 4 3], [2 3 11 4 9], [2 3 4 11 9], [2 3 4 9 11]
- [2 3 4 9 11], [2 3 11 4 9], [2 3 4 11 9], [2 9 11 4 3]
- [2 9 11 4 3], [2 3 4 11 9], [2 3 4 11 9], [2 3 4 9 11]
- [2 3 4 11 9], [2 9 11 4 3], [2 3 4 11 9], [2 3 11 4 9]

2. Elemanları [5, 19, 1, 14, 9] olan dizi üzerinde doğrusal arama yapılarak önce 19 ve daha sonra 14 sayısının bulunup bulunmadığı kontrol edilecektir. Bu aramalar için toplam kaç karşılaştırma işlemi yapılır?

- 2
- 4
- 6
- 8
- 9

3. Elemanları [5, 19, 1, 14, 9] olan dizi üzerinde doğrusal arama yapılarak önce 14 ve daha sonra 29 sayısının bulunup bulunmadığı kontrol edilecektir. Bu aramalar için toplam kaç karşılaştırma işlemi yapılır?

- 1
- 3
- 6
- 9
- 10

4. Elemanları [5, 19, 1, 14, 9] olan dizi üzerinde doğrusal arama yapılarak önce 29 ve daha sonra 39 sayısının bulunup bulunmadığı kontrol edilecektir. Bu aramalar için toplam kaç karşılaştırma işlemi yapılır?

- 1
- 3
- 5
- 10
- 12

5.

I. [99, 66, 8, 27, 45, 80]
II. [90, 65, 27, 18, 36,33]
III. [43, 90, 66, 75, 88, 37]

Yukarıdaki dizilerden hangisi üzerinde doğrusal arama algoritması uygulanabilir?

- Yalnızca I
- Yalnızca II
- Yalnızca III

- I ve I
- I, II ve III

6.

I. [13, 16, 18, 27, 55, 80]
II. [80, 55, 27, 18, 16, 13]
III. [13, 80, 16, 55, 18, 27]

Yukarıdaki dizilerden hangisi üzerinde ikili arama algoritması uygulanabilir?

- Yalnızca I
- Yalnızca II
- I ve II
- II ve III
- I, II, III

7. Elemanları [10, 15, 20, 25, 30, 35, 40] olan dizi üzerinde ikili arama yapılarak önce 22 ve daha sonra 35 sayısının bulunup bulunmadığı kontrol edilecektir. Bu aramalar için toplam kaç karşılaştırma işlemi yapılır?

- 1
- 2
- 3
- 4
- 5

8. Aranan elemanın dizinin ilk elemanından başlanarak sırayla dizi elemanları ile karşılaştırılması ve aranan elemana eşit dizi elemanı bulunduğunda aramanın sonlandırılması esasına dayanan algoritma aşağıdakilerden hangisidir?

- Seçerek Sıralama Algoritması
- Kabarcık Sırlaması Algoritması
- Yerleştirmeli Sıralama Algoritması
- İkili Arama Algoritması
- Doğrusal Arama Algoritması

9. Dizinin ilk elemanın yerinde bırakılması, dizinin ikinci elemanın seçilmesi, seçilen ikinci elemanın kendinden önceki elemanlarla karşılaştırılması, dizinin büyük elemanlarının sağa kaydırılması, daha sonra dizinin üçüncü sıradaki elemanının seçilmesi ve seçilen elemanın kendinden önce gelen elemanlarla karşılaştırılarak işlemlerin dizi sıralanıncaya kadar sürdürülmesi şeklinde olduğu algoritma aşağıdakilerden hangisidir?

- Seçerek Sıralama Algoritması
- Kabarcık Sırlaması Algoritması
- Yerleştirmeli Sıralama Algoritması
- İkili Arama Algoritması
- Doğrusal Arama Algoritması
    -
    - CEVAP ANAHTARI1. b2. c3. d4. d5. e6. c7. c8. e9. c

9. YAPILAR (STRUCTURES)

Giriş

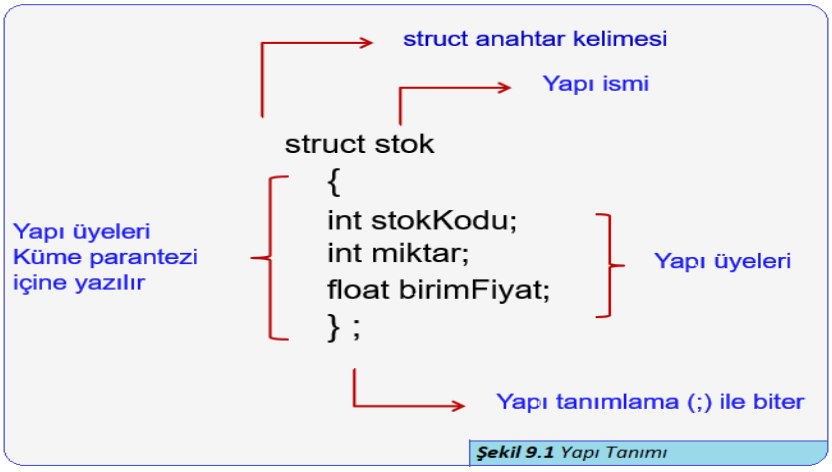
Bundan önceki bölümlerde int, float, double, char vb. C++' ın desteklediği temel veri tiplerini kullanmıştık. Hatırlanacağı gibi, temel veri tipleri kullanılarak tanımlanan değişkenler tek bir bilgiyi temsil edebiliyordu. Bu bölümde temel değişkenleri bir araya getirerek kendi veri tiplerimizi oluşturacağız ve oluşturulan bu veri tipleri ile birden fazla verinin nasıl temsil edileceğini öğreneceğiz. C++'ta yapı(struct) kendi veri tiplerimizi oluşturmanın araçlarından biridir.

Yapı(struct), birbiri ile ilişkili verileri bir isim altında toplayarak tanımlamak olarak tarif edilebilir. Bu şekli ile yapılar dizilere benzer. Diziler gibi birden fazla veriyi tutabilir ve yine diziler gibi bellekte sıralı şekilde bulunurlar. Ancak, diziler Altıncı bölümden de vurguladığımız gibi, aynı tür verileri bir isim altında temsil etmek için kullanılıyordu. Yapılar ise, aynı türdeki verileri bir isim altında toplayabildiği gibi, farklı türdeki verileri de aynı isim altında

toplayabilirler. Bunu başka bir şekilde ifade etmek gerekirse; bir yapının altında tanımlanan değişkenler aynı tipte olmak zorunda değildir. Örneğin bir değişken int, diğeri float ve bir diğeri de char tipinde olabilir.

9.1. Yapıların Oluşturulması ve Kullanılması

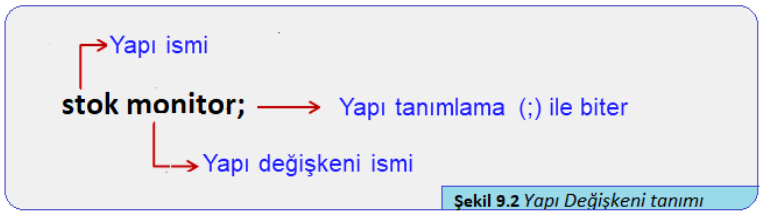
Bir yapının tanımı **struct** anahtar kelimesi ile başlatılır. Daha sonra yapıya bir isim verilir ve küme parantezi içerisinde yapının üyeleri tanımlanır. Yapıların tanımında kullanılan kapanış küme parantezinin sonuna, kararlardan, döngülerden ve fonksiyonlardan farklı olarak noktalı virgül konulur. Şekil 9.1'de yapı tanımının söz dizimi gösterilmiştir.



Şekil 9.1 *Yapı Tanımı*

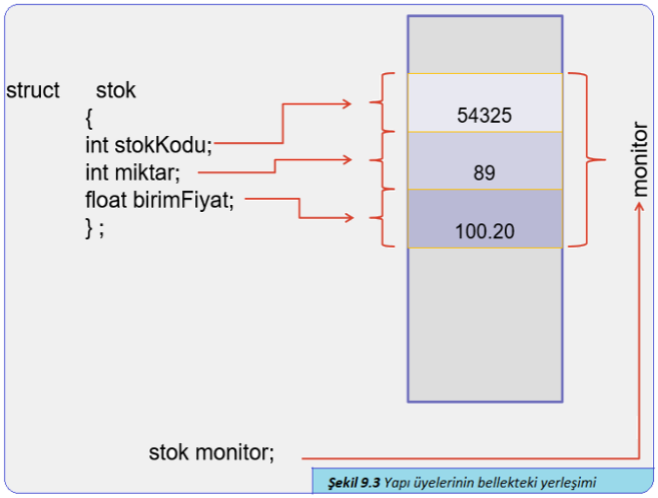
Şekil 9.1'de tanımı yapılan yapının, ikisi (*stokKodu*, *miktar*) int ve biri de float (*birimFiyat*) olmak üzere üç üyesi vardır. Bu tanımlama sadece bir taslak görevi görmektedir. Yapılan bu tanımlama sonucunda bellekte herhangi bir yer ayrılmaz. Bu yapıya ait bellekte bir yer ayrılabilmesi için en az bir yapı değişkeni tanımlanması gerekir. Geçtiğimiz bölümlerden hatırlayacağınız gibi, bu temel veri tipleri kullanılarak yapılan tanımlamalar için de böyleydi. Yani sadece program içerisinde ***int*** yazınca bellekte bir yer ayrılmıyordu. Bellekte yer ayrılabilmesi için, örneğin ***int sayi***; gibi bir tanımlama yapmak gerekiyordu.

Yapı değişkenlerinin tanımlanması, temel veri tipleri kullanılarak değişken tanımlanırken yapıldığı gibi, yapı ismi, boşluk, yapı değişkeninin ismi ve sonuna noktalı virgül konularak gerçekleştirilir. Aşağıda şekil 9.2'de yapı değişkeninin tanımlanması gösterilmiştir.



Şekil 9.2 *Yapı Değişkeni tanımı*

Bu tanım, bilgisayarın ana belleğinde monitör adlı yapı değişkeni için yer ayrılmasını sağlar. Ayrılan yer 12 Beyte büyüklüğündedir. Çünkü **monitor** adlı yapı değişkeninin ikisi ***int*** ve biri de ***float*** tipinde üç üyesi (***stokKodu***, ***miktar***, ***birimFiyat***) vardır ve bunlar bellekte dörder Byte 'tan toplam 12 Byte yer kaplar.



Şekil 9.3 *Yapı üyelerinin bellekteki yerleşimi*

Yapı değişkeninin bellekte ne kadar yer kapladığını ***sizeof()*** operatörünü kullanarak gösterebiliriz. ***sizeof()*** operatörü bir veri yapısının bellekte ne kadar yer kapladığını Byte cinsinden geri döndürür.

```
#include <iostream>
#include <locale.h>
using namespace std;
struct stok
{
    int stokKodu;
    int miktar;
    float birimFiyat;
};
int main() {
    setlocale(LC_ALL,"Turkish");
    stok monitor;
    cout<<" Yapı Değişkeninin Bellekte Kapladığı Alan :"<<sizeof(monitor)<<" Byte"<<endl;
    return 0;
}
```

Program 9.1 *Yapı değişkeninin bellekte kapladığı alan*

```
 Yapı Değişkeninin Bellekte Kapladığı Alan :12 Byte

--------------------------------
Process exited after 0.1329 seconds with return value 0
Press any key to continue . . .
```

Program 9.1'in Ekran Çıktısı

Yapı değişkenlerine isim verilmesi sırasında uyulması gereken kurallar, temel veri tipleri ile tanımlanan değişkenlere isim verirken uyulması gereken kurallarla birebir aynıdır. Bir yapı ile aynı ifadenin içerisinde birden fazla yapı değişkeni tanımlanabilir.

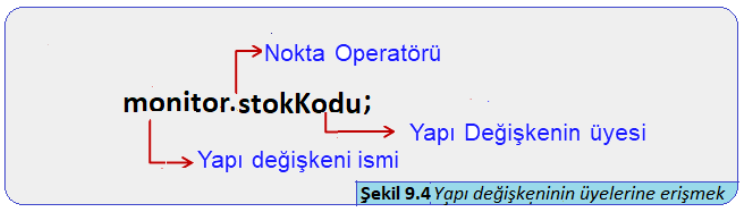
stok monitor, klavye, hardDisk, anaBellek;

Aynı ifadenin içerisinde birden fazla yapı değişkeni tanımlanması gerekiyorsa yapı değişken isimleri birbirlerinden virgül ile ayrılması gerekir. C programa dilinde yapı değişkenleri tanımlanırken tanımın başında ***struct*** bildirimi yapılması gerekir.

struct stok monitor, klavye, hardDisk, anaBellek;

C++ 'ta bu zorunluluk kaldırılmıştır. Ancak, C++ 'ta da C 'de olduğu gibi ***struct*** kullanılarak yapı değişkeni tanımlanabilir ve bu durumda da derleyici hata vermez.

Yapı değişkenlerinin üyelerine nokta operatörü(.) kullanılarak erişilir. Şekil 9.4 'te yapı değişkenin üyelerine erişilirken nokta operatörünün(üye erişim operatörü) nasıl kullanılacağı gösterilmiştir.



Şekil 9.4 *Yapı değişkeninin üyelerine erişmek*

Yapı değişkenlerinin üyelerine erişilirken sırası ile yapı değişkeninin ismi, nokta operatörü, yapı değişkeninin üyesi yazılarak erişilir.

```
stok monitor1,monitor2,monitor3;

monitor1.stokKodu=1001;
monitor1.miktar=10;
monitor1.birimFiyat=2500;
```

Yukarıda üç yapı değişkeni (***monitor1, monitor2, monitor3***) tanımlanmıştır. Daha sonra ***monitor1*** yapı değişkeninin üyelerine ilk değer ataması yapılmıştır.

Program 9.2, yapı değişkeninin üyelerine ilk değer atama işleminin nasıl yapılacağını göstermek için yazılmıştır.

```
#include <iostream>
#include <locale.h>
using namespace std;
struct stok
{
    int stokKodu;
    int miktar;
    float birimFiyat;
};
int main() {
    setlocale(LC_ALL,"Turkish");

    stok monitor1,monitor2,monitor3; //üç yapı değişkeni tanımlandı

    monitor1.stokKodu=1001;     //  monitor1 yapı değişkeninin
    monitor1.miktar=10;          //  üyelerine
    monitor1.birimFiyat=2500;   //  ilk değer ataması yapıldı
    cout<<"Stok Kodu    :"<<monitor1.stokKodu<<endl;   //monitor1 Yapı değişkeni
    cout<<"Stok Miktarı :"<<monitor1.miktar<<endl;     //üyeleri ekranda
    cout<<"Birim Fiyatı :"<<monitor1.birimFiyat<<endl; //görüntülendi

    monitor2=monitor1;      //atama operatörü kullanılarak monitör1, monitor2 'ye atandı

    cout<<endl<<endl<<"Stok Kodu    :"<<monitor2.stokKodu<<endl; //monitor2 Yapı değişkeni
    cout<<"Stok Miktarı :"<<monitor2.miktar<<endl;               //üyeleri ekranda
    cout<<"Birim Fiyatı :"<<monitor2.birimFiyat<<endl;           //görüntülendi

    monitor3={1002,25,2250}; //monitor3 yapı değişkeninin üyelerine küme parantezi ilk değer atandı.

    cout<<endl<<endl<<"Stok Kodu    :"<<monitor3.stokKodu<<endl;  //monitor3 Yapı değişkeni
    cout<<"Stok Miktarı :"<<monitor3.miktar<<endl;                //üyeleri ekranda
    cout<<"Birim Fiyatı :"<<monitor3.birimFiyat<<endl;            //üyeleri ekranda
    return 0;
}
```

Program 9.2 Yapı değişkeninin üyelerine ilk değer atanması

```
Stok Kodu    :1001
Stok Miktarı :10
Birim Fiyatı :2500


Stok Kodu    :1001
Stok Miktarı :10
Birim Fiyatı :2500


Stok Kodu    :1002
Stok Miktarı :25
Birim Fiyatı :2250

--------------------------------
Process exited after 0.1006 seconds with return value 0
Press any key to continue . . .
```

Program 9.2'nin Ekran Çıktısı

Program 9.2'den görülebileceği gibi yapı değişkenlerinin üyelerine tek tek değer atanabileceği gibi bir yapı değişkeni atama operatörü kullanılarak doğrudan diğer yapı değişkenine atanabilir.

monitor2=monitor1;

Bu atamadan sonra monitor2'nin üyelerinin değerleri monitor1'in üyeleri ile aynı olur. *Bakınız: Program 9.2*

Bundan başka yapı değişkenlerinin üyelerine dizilerde olduğu gibi küme parantezi içinden de atama yapılabilir.

monitor2={1002,25,2250};

Yapı değişkenlerinin üyelerine küme parantezi içerisinden atama yapılırken üyelerin değerleri arasına virgül konulması unutulmamalıdır.

Örnek: Bir yapının üyeleri öğrenci numarası, iki kısa sınav notu, vize notu, final notu ve yılsonu ortalaması olsun. Kısa sınavların her birinin yıl içi notuna katkısı %10, vizenin yıl içi notuna katkısı %80 olsun. Toplam yıl içi notunun yılsonu ortalamasına katkısı %40 ve finalin yılsonu ortalamasına katkısı da %60 olsun. ***ogrenci*** isimli bu yapıya ait ***ogrenci1*** isimli bir yapı değişkeni tanımlayınız ve yapı değişkeninin üyelerine ilk değer atayınız. Yukarıda belirtilen ağırlıkları dikkate alarak öğrencinin yılsonu ortalamasını hesaplayıp sonucu ekrana yazdıran C++ programı yazınız?

Çözüm:

```
#include<iostream>
using namespace std;
struct ogrenci
{
    int ogr_no;
    int q_1;
    int q_2;
    int vize;
    int final;
    float ortalama;
};
int main()
{
    ogrenci ogrencil;
    ogrencil.ogr_no=1;
    ogrencil.q_1=70;
    ogrencil.q_2=65;
    ogrencil.vize=80;
    ogrencil.final=90;
    ogrencil.ortalama=((ogrencil.q_1*0.1)+(ogrencil.q_2*0.1)+(ogrencil.vize*0.8))*0.4+
                      ogrencil.final*0.6;
    cout<<"Ogrenci No...........: "<<ogrencil.ogr_nc<<endl;
    cout<<"Ogrnci  Not Ort........"<<ogrencil.ortalama<<endl<<endl;
return 0; }
```

Program 9.3 Öğrencinin not ortalamasını hesplayan program

```
Ogrenci No...........: 1
Ogrnci  Not Ort........85


--------------------------------
Process exited after 0.09024 seconds with return value 0
Press any key to continue . . .
```

Program 9.3'ün Ekran Çıktısı

Örnek: Bir kenarı metre ve santimetre olarak girilen karenin çevresini hesaplayan C++ programını yazınız?

Not: *kare isimli bir yapı oluşturunuz. Yapının üyeleri metre ve cm olsun. Daha sonra bu yapıya ait yapı değişkenleri tanımlayınız ve yapı değişkenlerinin üyelerinin değerlerini klavyeden giriniz.*

Çözüm:

```cpp
#include<iostream>
#include<locale.h>
using namespace std;
struct kare
{
int metre;
float cm;
};
int main()
{
    setlocale(LC_ALL,"Turkish");
    kare kenar,cevre;
    cout<<" metre Giriniz....: "; cin>>kenar.metre;
    cout<<" cm Giriniz .....: "; cin>>kenar.cm;
    cevre.cm=0;
    cevre.cm=kenar.cm*4;
    cevre.metre=0;
    if(cevre.cm>=100.0)
    {
        int ekmetre=cevre.cm/100;
        cevre.cm=cevre.cm-(ekmetre*100.0);
        cevre.metre+=ekmetre;
    }
    cevre.metre+=kenar.metre*4;
    cout<<" Karenin Çevresi ="<<cevre.metre<<" m. "<<cevre.cm<<" cm."<<endl;
    return 0;
}
```

Program 9.4 *Bir kenarı metre ve santimetre olarak ayrı girilen Karanin çevre hesabı*

```
 metre Giriniz....: 4
 cm Giriniz .....: 90.3
 Karenin Çevresi =19 m. 61.2 cm.

--------------------------------
Process exited after 8.392 seconds with return value 0
Press any key to continue
```

Program 9.4'ün Ekran Çıktısı

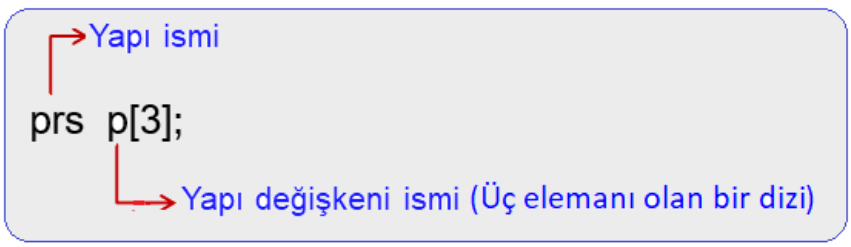
9.2. Yapıların Dizilerle Kullanılması

Altıncı bölümde C++'ın desteklediği temel veri tipleri kullanılarak bir dizinin nasıl tanımlanacağı ve C++ programları içerisinde bu dizilerin nasıl kullanılacağını öğrenmiştik. Temel veri tiplerini kullanarak tanımladığımız, kullandığımız dizilere benzer şekilde yapılarla oluşturduğumuz kendi veri türlerimizi kullanarak da dizi tanımlayabilir ve C++ programlarında tanımladığımız bu dizileri kullanabiliriz.

Aşağıda verildiği gibi bir yapı tanımlandığını varsayalım.

```cpp
struct prs
{
    string tc;
    string ad;
    float  saatUcreti;
};
```

Bu yapıya ait bir diziyi aşağıda gösterildiği gibi tanımlayabiliriz. Bu tanımlama, ***p*** isminde 3 elemanlı ve elemanlarının türü ***prs*** olan bir diziyi belirtir.



Aşağıda verilen ***Program 9.5*** yapıların dizilerle nasıl kullanılacağını gösteren basit, öğretici bir örnektir. Program 9.5 te önce ***prs*** isminde bir tür oluşturulmuş ve daha sonra türü ***prs*** olan üç elemanlı ***p*** dizisi oluşturulmuştur.

p dizisinin elemanlarına nokta operatörü (.) kullanılarak aşağıdaki şekilde ulaşılabilir.

```
p[0].tc="987654321";
p[0].ad="Hasan Ak   ";
p[0].saatUcreti=325.5;
p[1].tc="123456789";
p[1].ad="Mehmet Kara";
p[1].saatUcreti=355.75;
p[2].tc="432198765";
p[2].ad="Fatma Pembe";
p[2].saatUcreti=300.65;
```

Program 9.5 yukarıdaki bilgiler kullanılarak aşağıdaki gibi yazılmıştır. Bu programda dizi şeklinde tanımlanmış yapı değişkeninin elemanlarına ilk değer ataması yapılmıştır. Bölüm 6'dan öğrendiğimiz şekilde, dizi şeklinde tanımlanmış yapı değişkenlerinin elemanlarını, temel veri tipleri ile tanımlanmış dizilerde olduğu gibi ***cin*** nesnesini ve çıkarma operatörünü(>>) kullanarak klavyeden de girebiliriz.

Program 9.5'in son bölümünde ***for döngüsü*** kullanılarak ***p*** dizisinin elemanları ekranda gösterilmiştir. Bu işlemin sonucu için program 9.5'in ekran çıktısını inceleyiniz.

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   struct prs
5   {
6       string tc;
7       string ad;
8       float  saatUcreti;
9   };
10  int main() {
11   prs p[3];
12   p[0].tc="987654321";
13   p[0].ad="Hasan Ak   ";
14   p[0].saatUcreti=325.5;
15   p[1].tc="123456789";
16   p[1].ad="Mehmet Kara";
17   p[1].saatUcreti=355.75;
18   p[2].tc="432198765";
19   p[2].ad="Fatma Pembe";
20   p[2].saatUcreti=300.65;
21   cout<<"TC        "<<"AD SOYAD    "<<"SAAT UCRETI"<<endl;
22   cout<<"----------------------------------------"<<endl<<endl;
23  for(int i=0;i<3;i++) {
24      cout<<p[i].tc<<" "<<p[i].ad<<" "<<p[i].saatUcreti<<endl;
25  }
26      return 0;
27  }
```

Program 9.5 *Yapıların Dizilerle Kullanılması*

```
TC        AD SOYAD    SAAT UCRETI
----------------------------------------

987654321 Hasan Ak    325.5
123456789 Mehmet Kara 355.75
432198765 Fatma Pembe 300.65

--------------------------------
Process exited after 0.04084 seconds with return value 0
Press any key to continue . . .
```

Program 9.5'in Ekran Çıktısı

9.3. Yapıların Fonksiyonlara Aktarılması

Fonksiyonların parametre olarak aldığı değerler ve/veya geri dönüş değerleri yapı olabilir. Yapıları fonksiyonlara aktarmak diğer temel veri tipleri ile tanımlanmış değişkenleri fonksiyonlara aktarmaya benzer.

Program 9.6'da ***void yazdir(ogrenci);*** ifadesi ile ***yazdir*** isimli bir fonksiyon deklare edilmiştir. Bu deklarasyondan fonksiyonun bir yapı değişkenini parametre olarak alacağı anlaşılmaktadır. Ayrıca, fonksiyonun tipi **void** olarak belirlendiği için geri dönüş değeri olmayacaktır.

Daha sonra main() fonksiyonu içerisinde ***ogrenci ogrenci1;*** ifadesi ile ogrenci1 isimli bir yapı değişkeni tanımlanmış ve bu yapı değişkeninin üyelerine değerler atanmış, ortalama isimli üyenin değeri de hesaplanarak bulunmuştur.

Programın devamında ***yazdir(ogrenci1);*** ifadesi ile fonksiyon çağrılmıştır. Çağrılan fonksiyon oldukça basittir. Bir geri dönüş değeri yoktur. Sadece kendisine gönderilen yapı değişkeninin üyelerinden ikisini (ogrencif.ogr_no, ogrencif.ortalama) ekrana yazdırmaktadır. Program 9.6'yı ve ekran çıktısını inceleyiniz.

```
#include<iostream>
using namespace std;
struct ogrenci
{
    int ogr_no;
    int q_1;
    int q_2;
    int vize;
    int final;
    float ortalama;
};
void yazdir(ogrenci);
int main()
{
    ogrenci ogrenci1;
    ogrencil.ogr_no=1;
    ogrencil.q_1=70;
    ogrencil.q_2=65;
    ogrencil.vize=80;
    ogrencil.final=90;
    ogrencil.ortalama=((ogrencil.q_1*0.1)+(ogrencil.q_2*0.1)+(ogrencil.vize*0.8))*0.4+
                      ogrencil.final*0.6;
    yazdir(ogrencil);
return 0; }

void yazdir(ogrenci ogrencif)
{
    cout<<endl<<" Ogrenci No...........: "<<ogrencif.ogr_nc<<endl;
    cout<<" Ogrnci  Not Ort........"<<ogrencif.ortalama<<endl<<endl;
}
```

Program 9.6 *Yapıların fonksiyonlara aktarılması*

```
 Ogrenci No...........: 1
 Ogrnci  Not Ort........85


--------------------------------
Process exited after 0.09545 seconds with return value 0
Press any key to continue . . .
```

Program 9.6'nın ekran çıktısı

Program 9.7, program 9.6'dan bir miktar farklıdır. Program 9.7 de 9.6 gibi yapıların fonksiyonlarla kullanılmasına örnek olması amacıyla yazılmıştır. Hatta 9.7'nin fonksiyonlarından biri (***yazdir***) işlev olarak 9.6'nın ***yazdır*** fonksiyonuna benzemektedir.

9.7'deki farklılık, programdaki ***hesapla*** fonksiyonunun bir yapı değişkenini çağıran fonksiyona geri döndürmesidir.

Fonksiyonlar konusunu ele aldığımız 7. Bölümden hatırlayabileceğiniz gibi, orada (7.3) "Bir fonksiyon, çalışmasını tamamladıktan sonra, kendisini çağıran programa tek bir değer döndürebilir." Şeklinde bir ifade kullanmıştık. Yapı

değişkenlerini, çağıran fonksiyona geri döndürdüğümüzde de bu şekilde bir ifade kullanabiliriz. Ancak burada küçük bir fark bulunmaktadır. Çağıran fonksiyona yapı değişkeni tek olarak döndürülmüş olsa da, geri dönen yapı değişkenine, sahip olduğu birden fazla üye değişken açısından bakıldığında geri dönen değerin birden fazla olduğunu söylemek yanlış olmaz. Program 9.7'yi ve aynı programın ekran çıktısını inceleyiniz.

```
#include <iostream>
using namespace std;
struct tarih
{
    int gun;
    int ay;
    int yil;
};
tarih hesapla(tarih,tarih);
void yazdir(tarih);

int main() {
    tarih dgunu, bgun, sonuc;
    cout<<"Dogum gununu Gun  : ";cin>>dgunu.gun;
    cout<<"Dogum gununu Ay   : ";cin>>dgunu.ay;
    cout<<"Dogum gununu Yil  : ";cin>>dgunu.yil;cout<<endl;
    cout<<"Bugunun Tarihi Gun: ";cin>>bgun.gun;
    cout<<"Bugunun Tarihi Ay : ";cin>>bgun.ay;
    cout<<"Bugunun Tarihi Yil: ";cin>>bgun.yil;cout<<endl;
    sonuc=hesapla(dgunu,bgun);
    yazdir(sonuc);
    return 0;
}
tarih hesapla(tarih d,tarih b)
{
    tarih s={0,0,0};
    if(d.gun>b.gun)
    {
        b.gun+=30;
        b.ay-=1;
        }
    if(d.ay>b.ay)
    {
        b.ay+=12;
        b.yil-=1;
    }
    s.gun=b.gun-d.gun; s.ay=b.ay-d.ay;s.yil=b.yil-d.yil;
    return s;
}
void yazdir(tarih sy)
{
    cout<<"Yasiniz :"<<sy.yil<<" "<<sy.ay<<" "<<sy.gun<<endl;
}
```

Program 9.7 *Yapı Değişkenlerinin çağıran fonksiyona geri döndürülmesi*

```
Dogum gununu Gun  : 28
Dogum gununu Ay   : 05
Dogum gununu Yil  : 1999

Bugunun Tarihi Gun: 14
Bugunun Tarihi Ay : 04
Bugunun Tarihi Yil: 2020

Yasiniz :20 10 16

--------------------------------
Process exited after 30.53 seconds with return value 0
Press any key to continue . . .
```

Program 9.7'nin Ekran Çıktısı

9.4. İç İçe(Nested) Yapılar

C++'ta bir yapı içerisinde başka bir yapı bulunabilir. Program 9.8'de iki adet yapı görünüyor. Birinci yapı ***tarih*** ikinci yapı ise ***personel***. Personel içerisinde, tarih yapı türünde bir eleman bulunmaktadır.

Şimdi iç içe yapıların program içerisinde nasıl kullanılacağını açıklayalım. main() fonksiyonu içerisinde önce personel tipinde personel1 yapı değişkeni tanımlanmıştır. Sonra sırası ile aşağıda gösterildiği gibi personel1 yapı değişkeninin üyelerinin girişi yapılmıştır.

```
cout<<"personel Adi giriniz       :";
cin>>personel1.ad;
cout<<"personel Dog_Gun giriniz :";
cin>>personel1.dogtar.gun;
cout<<"personel Dog_Ay giriniz  :";
cin>>personel1.dogtar.ay;
cout<<"personel Dog_Yil giriniz :";
cin>>personel1.dogtar.yil;
```

Burada bu başlık altında şimdiye kadar öğrendiklerimizden farklı olan, personel1 yapı değişkeninin doğum tarihi (dogtar) elemanın ele alınış şeklidir. Dikkat edilirse dogtar yapı değişkeninin elemanlarına ulaşmak için bu defa iki nokta operatörü kullanılmıştır.

İpucu: *Bir yapı içerisinde kendisinden daha önce tanımlanmış bir yapı kullanılarak, eleman tanımlanmış ise, bu yapı ile tanımlanmış yapı değişkenlerin elemanlarına iki adet nokta operatörü kullanılarak ulasılabilir*

(1)
```
struct personel
{
   char ad[20];
   struct tarih
    {
       int gun;
       int ay;
       int yil;
            };
};
```
(1)

(2)
```
struct tarih
{
  int gun;
  int ay;
  int yil;
};

struct personel
{
  char ad[20];
  tarih dogtar;
```
(2)

Yapılar iç içe kullanılırken bir yapının bildirimi doğrudan diğer yapı içinde yapılabilir**(1).**
Bu durumda yapı değişkeninin üyelerine ulaşmak için bir adet nokta operatörü kullanılır.
Önceden tanımlanmış yapı sonradan tanımlanmış yapının içerisinde kullanılabilir**(2)**

```
1   #include <iostream>
2   using namespace std;
3   struct tarih
4   {
5         int gun;
6         int ay;
7         int yil;
8         };
9   struct personel
10  {
11        char ad[20];
12        tarih dogtar;
13        };
14     int main(){
15      personel personel1;
16      cout<<"personel Adi giriniz       :";
17      cin>>personel1.ad;
18      cout<<"personel Dog_Gun giriniz :";
19      cin>>personel1.dogtar.gun;
20      cout<<"personel Dog_Ay giriniz  :";
21      cin>>personel1.dogtar.ay;
22      cout<<"personel Dog_Yil giriniz :";
23      cin>>personel1.dogtar.yil;
24      cout<<endl<<endl<<"Personel Adi : "<<personel1.ad<<endl;
25      cout<<"Dogum Tarihi : "<<personel1.dogtar.gun<<"\\"<<personel1.dogtar.ay<<"\\"<<personel1.dogtar.yil<<endl;
26      return 0;
27  }
```

Program 9.8 *İç içe yapı örneği*

```
personel Adi giriniz      :Mehmet
personel Dog_Gun giriniz :28
personel Dog_Ay giriniz  :04
personel Dog_Yil giriniz :1999


Personel Adi : Mehmet
Dogum Tarihi : 28\4\1999

--------------------------------
Process exited after 22.07 seconds with return value 0
Press any key to continue . . .
```

Program 9.8'in ekran çıktısı

Bölüm Özeti

C++'ta yapı oluşturmayı ve bu şekilde temel veri tiplerini kullanarak kendimize ait veri tipi tanımlamayı öğrendik. Oluşturulan kendimize ait veri türleri ile değişken(yapı değişkeni) tanımlamayı, yapı değişkenlerinin üyelerine nasıl ulaşacağımızı öğrendik. Yapıları kullanarak C++'ta program yazabilme becerisi kazandık. Yapı değişkenlerinin bellekte nasıl yerleştiğini ve ne kadar yer tuttuğunu öğrendik

Yapı türünde bir dizi tanımlamayı, bu dizilerin elemanlarına nasıl erişileceğini, erişilen dizi elemanlarına ilk değer atamasının nasıl yapılacağını öğrendik.

Yapıları kullanarak fonksiyon deklare etmeyi, fonksiyon tanımlamayı, fonksiyon çağırmayı, fonksiyonlardan yapı değişkenlerini döndürmeyi öğrendik.

C++ yapılar iç içe tanımlanabilir. İç içe yapı tanımlamanın iki farklı yöntemini, iç içe tanımlanmış yapıların üyelerine erişmeyi öğrendik.

Yapılar bir yönüyle sınıf ve nesnelere benzemektedir. Bu konuyu öğrenmemizin önemli yararlarından biri de sınıf ve nesneler konusunu öğrenirken bize önemli ölçüde yarar sağlayacak olmasıdır.

ÜNİTE SORULARI

1.

```
1   #include<iostream>
2   using namespace std;
3   struct ogrenci
4   {
5       int ogr_no;
6       int q_1;
7       int q_2;
8       int vize;
9       int final;
10      float ortalama;
11  }
12  int main()
13  {
14
15  return 0;
16  }
```

Yukarıda verilen C++ programında bir yapı tanımlanmış ve yapı tanımlanırken bir satırda yazılım (syntax) hatası yapılmıştır. Hatalı satır aşağıdakilerden hangisidir?

- 3. Satır
- 5. Satır
- 7. Satır
- 9. Satır
- 11. Satır

2. Bir yapı tanımı için aşağıdakilerden hangisi hatalı olarak yazılmıştır?

- Yapı tanımına struct anahtar kelimesi ile başlanır.
- Yapı tanımında struct anahtar kelimesinden sonra yapının ismi gelir.
- Yapı tanımında struct anahtar kelimesinden sonra yapı değişkeninin ismi gelir.
- Bir yapı tanımlanırken yapı elemanlarının tanımlandığı satırlar noktalı virgül ile biter.
- Bir yapıda yer alan yapı elemanları temel veri tiplerinden herhangi birinin tipinde olabilir.

3.

```
#include<iostream>
using namespace std;
struct ogrenci
{
     int ogr_no;
     int q_1;
     int q_2;
     int vize;
     int final;
     float ortalama;
};
int main()
{  ogrenci ogrenci1;
   cout<<sizeof(ogrenci1)<<endl;
return 0;
}
```

Yukarıda verilen programda **sizeof()** fonksiyonu kullanılarak yapı değişkeninin bellekte kapladığı alan Byte olarak ekrana yazdırılmıştır. Yapı değişkeninin bellekte kapladığı alan aşağıdakilerden hangisidir?

- 4
- 8
- 16
- 24
- 32

4. Bir yapı için aşağıdakilerden hangisi hatalı olarak yazılmıştır?

- Yapılar birbirleri ile ilgili değişkenlerin tek isim altında toplanmasını sağlar
- Yapı tanımında bildirilen değişkenlere yapının üyeleri(elemanları) denir.
- Yapılarda üye tanımlaması sırasında sadece bir veri tipi kullanılabilir.
- Farklı yapıların üye isimleri aynı olabilir.
- Yapılarda üye tanımlaması sırasında farklı veri tipleri kullanılabilir.

5. Aşağıdaki seçeneklerdeki ifadelerden hangisi yanlıştır?

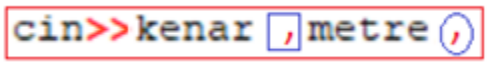
- Yapılar farklı türdeki verileri bir isim altında temsil edebilir.
- Yapılar aynı türdeki verileri bir isim altında temsil edebilir.
- Yapı değişkenlerinde tek bir veri tutabileceği gibi birden fazla veri de tutabilir.
- Yapı değişkenlerinin elemanları, dizi elemanlarında olduğu gibi, bellekte sıralı şekilde durmaktadır.
- Bir yapı tanımlandığında bellekte bu yapı için yer ayrılır.

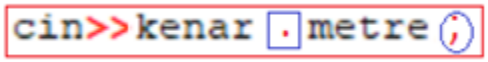
6.

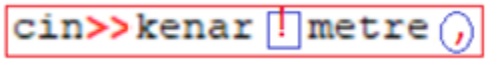
```
cin>>kenar metre
```

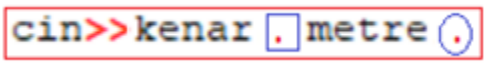
İfadesi bir C++ programında yapı değişkeninin üyesine klavyeden değer girilmesi için yazılmıştır. İfadenin doğru olması için sırası ile kare ve daire içerisine ne yazılmalıdır?

-
```
cin>>kenar,metre;
```

- 

- 

- 

- 

7.

```
struct prs
{
   char  prsNo[11];
   char  prsAd[25];
   float prsUcret;
};
```

Bir C++ programında yukarıda gösterildiği gibi **prs** isimli bir yapı tanımlanmıştır ve program gövdesinde **prs p[3];** ifadesi ile bir yapı değişkeni tanımlanmıştır. Bu yapı değişkeni için bilgisayar belleğinde kaç Byte 'lık yer ayrılır?

- 120
- 10
- 80
- 40
- 12

8. Tanımlanmış bir yapı içerisinde ikinci bir yapı tanımlanmış ise, bu iç içe yapı ile tanımlanmış yapı değişkenlerinin üyelerine kaç adet nokta operatörü kullanılarak ulaşılabilir?

- 1
- 2
- 3
- 4
- 5

9. C++ programlarında Yapı değişkeni program gövdesinin haricinde nerede tanımlanabilir?

- Başka herhangi bir yerde tanımlanamaz.
- Programın ilk satırından son satırına kadar her yerde tanımlanabilir.
- C++ programlarında main içerisinde tanımlanabilir.
- Yapı tanımının yapıldığı noktada kapanan küme parantezinden sonra ve virgülden önce tanımlanabilir.
- include <iostream> satırından önce tanımlanabilir.
  -
  - 1. e2. c3. d4. c5. e6. c7. a8. b9. d

10. İŞSARETÇİLER (POINTERS)

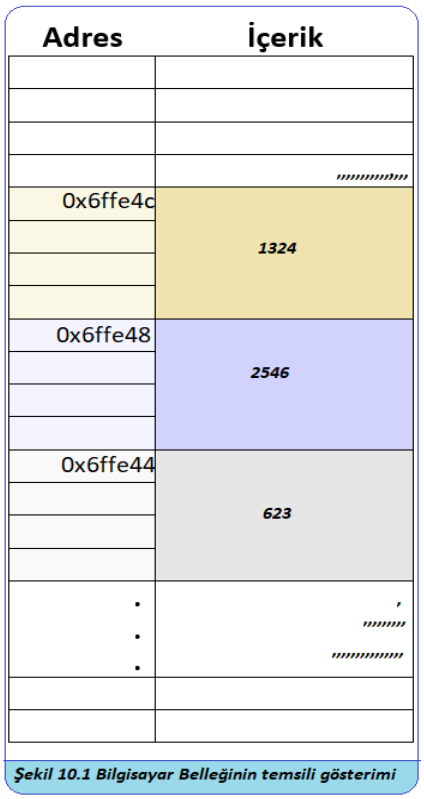
Giriş

Bazı kaynaklarda işaretçiler konusu ***Göstericiler*** başlığı altında incelenmektedir. Konuyla ilgili farklı kaynaklara müracaat edildiğinde bir karışıklığa yol açmamak için, daha başlangıçta bu hususu vurgulamakta fayda olduğu düşünüldüğünden konuya bu şekilde bir açıklama cümlesiyle giriş yapıldı.

İşaretçiler konusu C++ ve C programlama dillerinin en popüler konularından biri, belki de en popüler konusudur. Ayrıca işaretçiler, başka programlama dillerinde yer almayan, başka bir deyişle başka programlama dilleri tarafından desteklenmeyen bir özelliktir.

İşaretçiler konusunda, bilgisayar programlamaya yeni başlayanlar arasında, "*işaretçileri öğrenmek zordur*" şeklinde yaygın, yanlış bir kanaat vardır. Ancak burada şunu belirtmeliyiz; C++ 'in diğer konularına göre, işaretçiler konusunu öğrenmek için biraz daha fazla zaman ayırmanız gerekebilir.

Şimdi asıl konumuza gelip işaretçi nedir? Ne işe yarar? Sorularının cevaplarını arayalım. İşaretçi, içerisinde başka değişkenlerin adresini tutan bir değişkendir. Bilgisayar belleğinde her bir Byte 'ın bir adresi vardır ve bu adresler heksadesimal (16 Tabanlı Sayı Sisteminde) sayılarla ifade edilir. Şekil 10.1'de Bilgisayar belleği temsili olarak gösterilmiştir.



Şekil 10.1 Bilgisayar Belleğinin temsili gösterimi

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int sayi1=1324,sayi2=2546,sayi3=623;
6      cout<<&sayi1<<endl;
7      cout<<&sayi2<<endl;
8      cout<<&sayi3<<endl;
9      return 0;
10 }
```
Bilgisayar Belleğinde çalışan Program

```
0x6ffe4c
0x6ffe48
0x6ffe44

--------------------------------
Process exited after 0.04614 seconds with return value 0
Press any key to continue . . .
```
Bellekte Çalışan Programın Çıktısı

Programların derlenmesi sırasında derleyici, program içerisinde tanımlanmış değişkenler için, değişkenin boyutuna göre, bilgisayar belleğinde yer ayırır. Örneğin program içerisinde ***char*** tipinde bir değişken tanımlanmış ise bu değişken için bellekte bir Byte yer ayrılır, ***int*** tipinde bir değişken tanımlanmış ile bilgisayar belleğinde dört Byte yer ayrılır, Eğer double tipinde bir değişken tanımlanmışsa bu defa bellekte sekiz Byte yer ayrılır. Bir Byte 'tan fazla yer ayrılan değişkenlerin adresi, o değişken için ayrılan ilk Byte 'ın adresidir. Yukarıda verilen örnekte üç tane ***int*** tipinde değişken tanımlanmış, Bu değişkenler için bilgisayar belleğinde dörder Byte 'lık yer ayrılmış ve değişkenlerin adresleri olarak, ayrılan dörder Byte 'tan ilk Byte 'ların adresi olduğu gösterilmiştir. Bu adreslerin heksadesimal olarak ifade edildiğine dikkat ediniz.

C++'ta işaretçileri kullanmadan da program yazabiliriz. Nitekim bundan önceki bölümlerde böyle programlar yazdık. Ancak daha hızlı çalışan programlar yazmak, sistemden bellek almak, veri yapılarını tanımlamak gibi işlemlerde işaretçilerin kullanılmasının önemli ölçüde yararı vardır. C ve C++'ın belki de en güçlü yanı işaretçileri kullanmasından kaynaklanmaktadır.

Bu bölümde adres operatörünün kullanılmasını, işaretçilerin tanımlanması ve kullanılmasını, dizilerde ve fonksiyonlarda işaretçilerin kullanılmasını öğreneceğiz.

10.1. Adres Operatörü (&)

Adres operatörü, sabitlerin veya değişkenlerin bellekte hangi adreslere yerleştiklerini öğrenmek için kullanılır. Adres operatörünün simgesi (&) işaretidir. Bir değişken veya sabitin adresini öğrenmek için değişkenin önüne **&** yazılır. Örneğin program içerisinde tamsayı tipinde ***sayi1*** isimli bir değişkenimiz olduğunu varsayalım. Bu değişkenin adresini öğrenmek için:

&sayi1

yazmamız yeterlidir. Program 10.1'de tamsayı (***int***) tipinde üç değişken tanımlanmıştır. Daha sonra adres operatörü kullanılarak bu değişkenlerin bilgisayar belleğinde kullandıkları adresler ekrana yazdırılmıştır. Program 10.1'in aşağıda verilen ekran çıktısı incelendiğinde, değişkenlerin ekrana yazdırılan adreslerinin heksadesimal olarak ifade edildiği (**0x6ffe4c, 0x6ffe48, 0x6ffe44**) görülecektir. Dikkat edilirse ekrana yazdırılan adresler dörder Byte olarak azalan sıradadır. Bunun nedeni yerel değişkenlerin belleğin aşağı yönde büyüyen kısmında saklanmasıdır. Program 10.1'deki

değişkenler global değişken olarak tanımlansaydı adresleri artan sırada olacaktı. Çünkü global değişkenler belleğin yukarı yönde büyüyen kısmında saklanır.

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       int sayi1=3;
5       int sayi2=5;
6       int sayi3=7;
7       cout<<"  "<<&sayi1 <<endl;
8       cout<<"  "<<&sayi2 <<endl;
9       cout<<"  "<<&sayi3 <<endl;
10      return 0;
11  }
```

Adres Operatörü

Adresi Ekrana yazılacak Değişken

Program 10.1 Adres Operatörü

```
0x6ffe4c
0x6ffe48
0x6ffe44

--------------------------------
Process exited after 4.348 seconds with return value 0
Press any key to continue . . .
```

sayi1, sayi2 ve sayi3 değişkenleri için ayrılan bellek adresleri

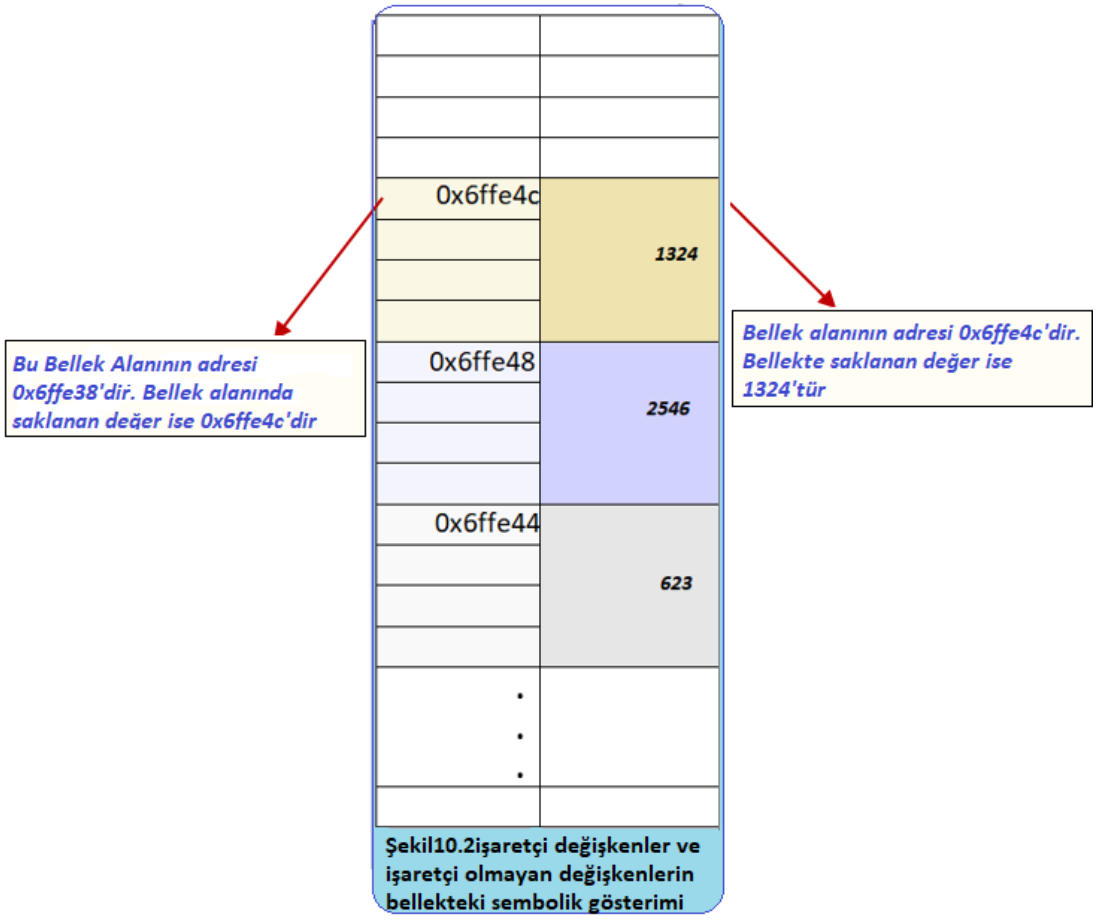
Program 10.1'in Ekran Çıktısı

Burada şu iki hususu vurgulamakta yarar var. Bu hususlardan birincisi derleyicinin sabitler veya değişkenler için belleğin hangi adreslerinin kullanacağını bizim müdahalemiz olmadan, bizim adımıza takip etmesidir. Vurgulanması gereken ikinci husus ise, muhtemelen program 10.1'i siz kendi bilgisayarınızda yazıp çalıştırdığınızda, ekranınızda değişkenlerin adresleri muhtemelen farklı olacaktır. Çünkü sabit ve değişkenler için hangi adreslerin kullanılacağı, bilgisayarınızın fiziksel özellikleri, kullandığınız işletim sistemi, sizin programı çalıştırdığınızda bilgisayarınızda o anda çalışan diğer programlar gibi parametrelere bağlıdır.

10.2. İşaretçilerin Tanımlanması ve Kullanılması

İşaretçi değişkenlerinin tanımlanması diğer değişkenlerin tanımlanmasına benzer. Ancak bir işaretçi değişkeni tanımlanırken değişkenin başına asteriks (*) işareti konulur (*Bkz. Program 10.2 satır 6*). Bu işaretin adı işaretçi operatörüdür. Bir program içerisinde tanımlanmış değişkenin, işaretçi değişken olup olmadığına değişkenin tanımlanması sırasında başına işaretçi operatörü konulup konulmadığına bakılarak karar verilir.

İşaretçi değişkenleri adres değerlerini tutan değişkenlerdir. Bir adres değerini tutan değişkene işaretçi değişkeni ya da kısaca işaretçi denir. İşaretçiler tanımlanırken değişkenlerin tanımlanması sırasında olduğu gibi tipleri (***char, int, float vb.***) belirlenir. Ancak belirlenen bu tipler işaretçinin tuttuğu değerin türünü belirtmez. Yukarıda da belirlendiği gibi işaretçiler her zaman adres bilgisi tutar. Yani işaretçi değişkenin char, int, float vb. olması işaretçinin içeriği ile ilgili değildir. Ancak ileride açıklanacak bir istisnayı bir kenara bırakacak olursak işaretçiler adreslerini tutacakları değişkenlerin tipleri ile aynı tipte tanımlamalıdır. Yani işaretçinin adresini tutacağı değişken ***int*** ise işaretçinin tipi de ***int*** olmalı, değişken ***float*** ise işaretçi de ***float*** olarak tanımlanmalıdır. İşaretçi tanımında kullanılan char, int, float vb. tip tanımlamaları, işaretçinin tuttuğu adreste bulunan verinin tipi ile doğrudan ilgilidir. ***Şekil 10.2***'de işaretçi değişkenler ve işaretçi olmayan değişkenler ve bunların değerleri gösterilmiştir.



Şekil10.2işaretçi değişkenler ve işaretçi olmayan değişkenlerin bellekteki sembolik gösterimi

```
1   #include <iostream>
2   using namespace std;
3   int sayi1,sayi2;
4   int main() {
5       int sayi3,sayi4;
6       int* i1, *i2,*i3,*i4;
7       i1=&sayi1;i2=&sayi2;
8       i3=&sayi3;i4=&sayi4;
9       cout<<i1<<endl;
10      cout<<i2<<endl<<endl;
11      cout<<i3<<endl;
12      cout<<i4<<endl;
13      return 0;
14  }
```

Global Değişkenler

Yerel değişkenler

İşaretçi değişkenlerin tanımlanması

Değişkenlerin Adreslerinin işaretçi değişkenlere atanması

İşaretçi değişkenlerin değerlerinin ekrana yazdırılması

***Program 10.2a* İşaretçi Değişkenleri**

```
0x4a7030
0x4a7034

0x6ffe2c
0x6ffe28

--------------------------------
Process exited after 3.314 seconds with return value 0
Press any key to continue . . .
```

İşaretçi değişkenlerin değerleri

***Program 10.2*'a'nın Ekran Çıktısı**

Program 10.2a'nın üçüncü satırında iki global değişken (sayi1, sayi2) ve beşinci satırında iki yerel değişken (sayi3, sayi4) tanımladık. Programın 6. satırında dört adet işaretçi tanımladık (i1, i2, i3, i4). Yedinci ve sekizinci satırlarda global ve yerel değişkenlerin adreslerini işaretçi değişkenlere atadık. Dokuz, on, on bir ve on ikinci satırlarda işaretçi değişkenlerin değerlerini ekrana yazdırdık. Bkz. Program 10.2a'nın ekran çıktısı. Ekrana yazdırılan bu adreslerin işaretçi

değişkenlerin değerleri olduğunu ve bu heksadesimal değerlerin sırası ile global ve yerel değişkenlerin adresleri olduğunu unutmayınız. Program 10.2a'nan ekran çıktısı incelendiğinde global değişkenlerin adreslerinin artan sırada yerel değişkenlerin adreslerinin ise azalan sırada olduğu görülmektedir.

```
2  #include <locale.h>
3  using namespace std;
4  int main() {
5      setlocale(LC_ALL,"Turkish");
6      int s1=1324,s2=2546,s3=623;
7      int *ss1, *ss2, *ss3;
8      ss1=&s1;
9      ss2=&s2;
10     ss3=&s3;
11     cout<<"İşaretçinin Adresi     İşaretçinin İçinde Tuttuğu Adres"<<endl;
12     cout<<"--------------------+---------------------------------+"<<endl;
13     cout<<&ss1<<"            +  "<<ss1<<endl;
14     cout<<&ss2<<"            +  "<<ss2<<endl;
15     cout<<&ss3<<"            +  "<<ss3<<endl;
16     cout<<"--------------------+---------------------------------+"<<endl;
17     return 0;
18 }
```

Program 10.2b İşaretçilerin adresleri ve işaretçilerin içinde tuttukları adresler

```
İşaretçinin Adresi     İşaretçinin İçinde Tuttuğu Adres
--------------------+---------------------------------+
0x6ffe28            +  0x6ffe3c
0x6ffe20            +  0x6ffe38
0x6ffe18            +  0x6ffe34
--------------------+---------------------------------+
```

program 10.2b'nin ekran çıktısı

Program 10.2b'den görülebileceği gibi, işaretçi değişken işaretçi olmayan bir değişkenin adresini tutarken aynı zamanda işaretçi değişkenin de kendine ait bir adresi vardır. Başka bir şekilde ifade etmek gerekirse, her işaretçi değişkeni, birincisi işaretçinin içerisinde tuttuğu adres, ikincisi işaretçinin bellekte tutulduğu adres olmak üzere iki adrese sahiptir. Bu adresleri birbirine karıştırmamak gerekir.

Bu bölümün buraya kadar ele aldığımız kısmında, işaretçi değişkenlerin tanımlanmasını, işaretçi değişkenlerin adres bilgilerini nasıl tuttuklarını ve adres bilgilerine nasıl ulaşabildiğimizi öğrendik. Şimdi işaretçi değişkenlerin hangi nedenlerle kullanıldığını öğreneceğiz.

```
1  #include <iostream>
2  #include <locale.h>
3  using namespace std;
4  int main() {
5      setlocale(LC_ALL,"Turkish");
6      int sayi=120;
7      int *isr=&sayi;
8      cout<<"sayi degşkeninin adresi adres operatörü ile yazdırıldı     : "<<&sayi<<endl;
9      cout<<"sayi degişkeninin adresi işaretçi değişkeni ile yazdırıldı : "<<isr<<endl;
10     return 0;
11 }
```

Program 10.3 İşaretçi Değişken

```
sayi degşkeninin adresi adres operatörü ile yazdırıldı     : 0x6ffe44
sayi degişkeninin adresi işaretçi değişkeni ile yazdırıldı : 0x6ffe44

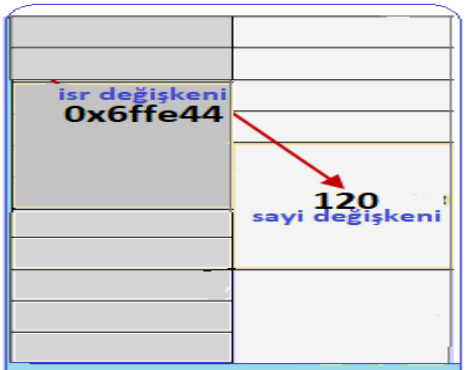
--------------------------------
Process exited after 0.07254 seconds with return value 0
Press any key to continue . . .
```

Program 10.3' ün Ekran Çıktısı

İşaretçiler, adreslerini tuttukları değişkenler üzerinde işlem yapmak için kullanılırlar. Program 10.3'te Altıncı satırda ***sayi*** değişkeni tanımlandı ve değişkene 120 değeri atandı. Yedinci satırda ***isr*** isimli bir işaretçi tanımlandı ve bu işaretçi değişkenine sayi değişkeninin adresi atandı.

```
6       int sayi=120;
7       int *isr=&sayi;
```

Bu işlemlerin sonunda şekil 10.3'ten de görülebileceği gibi ***isr*** işaretçi değişkeni bellekteki ***sayi*** değişkenine işaret eder.



Şekil 10.3 *isr* değişkeni *sayi* değişkenini işaret ediyor

isr işaretçi değişkeninin ***sayi*** değişkenine işaret etmesinden yararlanılarak sayi değişkeni kullanılmadan işaretçi değişkeni kullanılarak sayi değişkeninin değeri değiştirilebilir. Program 10.3b'yi inceleyiniz.

```
1   #include <iostream>
2   #include <locale.h>
3   using namespace std;
4   int main() {
5       setlocale(LC_ALL,"Turkish");
6       int sayi=120;
7       int *isr=&sayi;
8       cout<<"Sayı Değişkeninin ilk  değeri = "<<sayi<<endl;
9       *isr=987;
10      cout<<"Sayı Değişkeninin yeni değeri = "<<sayi<<endl;
11      return 0;
12  }
```

Program 10.3b İşaretçi değişkeni kullanarak sayi değişkeninin değerini değiştirmek

```
Sayı Değişkeninin ilk  değeri = 120
Sayı Değişkeninin yeni değeri = 987

--------------------------------
Process exited after 4.205 seconds with return value 0
Press any key to continue . . .
```

Program 10.3b'nin Ekran Çıktısı

Programın dokuzuncu satırındaki ifade ile sayi değişkeninin değeri 987 yapılmıştır.

```
9       *isr=987;
```

Yukarıdaki ifadede ***isr*** işaretçinin önündeki * operatörü(içerik operatörü), ***sayi*** değişkeninin işaret ettiği adrese karşılık geldiği için ***sayi*** değişkeninin kullanılması gerektiği her noktada ****isr*** ifadesini kullanabiliriz. Örneğin, sayi değişkeninin değerini ***s1*** isimli bir değişkene kopyalamak istediğimizde aşağıdaki ifade yazabilir. Bunun için Program 10.3c'yi inceleyiniz.

```
int s1=*isr;
```

```
1   #include <iostream>
2   #include <locale.h>
3   using namespace std;
4   int main() {
5       setlocale(LC_ALL,"Turkish");
6       int sayi=120,s1;
7       int *isr=&sayi;
8       cout<<"Sayı Değişkeninin ilk  değeri = "<<sayi<<endl;
9       *isr=987;
10      cout<<"Sayı Değişkeninin yeni değeri = "<<sayi<<endl;
11      s1=*isr;
12      cout<<"s1 Değişkeninin değeri        = "<<s1<<endl;
13      return 0;
14  }
```

Program 10.3c Gösterici değişkeni kullanarak diğer değişkenlere değer atamak

```
Sayı Değişkeninin ilk  değeri = 120
Sayı Değişkeninin yeni değeri = 987
s1 Değişkeninin değeri        = 987

--------------------------------
Process exited after 3.71 seconds with return value 0
Press any key to continue . . .
```

Program 10.3c'nin Ekran Çıktısı

Program 10.3c'nin Program 10.3b'den farkı, Program 10.3c'ye On bir ve on ikinci satırların eklenmiş olmasıdır. On birinci satırda içerik operatörü kullanılarak ***sayi*** değişkeninin değeri ***s1*** değişkenine atanmış ve on ikinci satırda ***s1*** değişkeninin değeri ekranda gösterilmiştir.

İşaretçiler başlığı altında incelediğimiz bu bölümün buraya kadar olan kısmında üç tane operatör kullandık. Bu operatörlerden birincisi Adres Operatörüdür (**&**). Adres operatörünü, değişkenlerin bellekteki adreslerini göstermek için kullandık. Operatörlerin ikincisi, İşaretçi tanımlamam operatörüdür (*****). İşaretçi tanımlama operatörünü, değişkenlerin tanımlanması sırasında, tanımlanan değişkenin işaretçi değişkeni olacağını belirtmek için kullandık. Kullandığımız üçüncü operatör ise içerik operatörüdür (*). Asteriks(*****) işareti tanımlama satırında kullanılmıyorsa içerik operatörüdür. İçerik operatörünü, işaretçinin işaret ettiği değişkenin değerlerini maniple etmek için kullandık. Burada dikkat edilmesi gereken en önemli husus işaretçi tanımlama operatörü ile içerik operatörünü karıştırmamaktır.

10.3. void İşaretçiler

Bu bölümde şu ana kadar bir işaretçinin içine yerleştirdiğimiz adres, işaretçi ile aynı tipteydi. Yani ***int***' a işaret eden bir işaretçiye ***int*** bir değişkenin adresini, ***float***' a işaret eden bir işaretçiye ***float*** bir değişkenin adresini, vb. atadık. Bu konunun bir istisnası vardır. ***void*** tipindeki işaretçiler herhangi bir veri tipine işaret edebilir. Bu nedenle ***void*** tipi işaretçilere genel amaçlı işaretçiler denir.

Burada dikkat edilmesi gereken en önemli husus, ***void*** işaretçilerin türü olmadığı için içerik operatörü olarak kullanılamayacakları hususudur. Yani ***void*** işaretçiler bir değerin işaretçisi olamazlar ve ayrıca aritmetik işlemlerde kullanılamazlar.

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int ins;
5      float fos;
6
7      int* isri;
8      float* isrf;
9      void* isrv;
10
11     isri = &ins;
12     isrf=&fos;
13
14     isrv=&ins;
15     cout<<isrv<<endl;
16     isrv=&fos;
17     cout<<isrv<<endl;
18     return 0;
19 }
```

Program 10.4 void işaretçiler

```
0x6ffe34
0x6ffe30

--------------------------------
Process exited after 0.05195 seconds with return value 0
Press any key to continue . .
```

Program 10.4'ün Ekran çıktısı

Program 10.4'te dördüncü satırda ***int*** türünde, beşinci satırda ***float*** düründe iki değişken tanımlanmıştır. Yedi, sekiz ve dokuzuncu satırlarda sırasıyla ***int, float, void*** işaretçiler tanımlanmıştır. On bir ve on ikinci satırlarda sırasıyla ***int*** ve ***float*** değişkenlerin adresleri yine sırasıyla ***int*** ve ***float*** işaretçi değişkenlere atanmıştır. Aynı şekilde on dört ve on altıncı satırlarda ***int*** ve ***float*** değişkenlerin adresleri ***void*** işaretçiye aktarılmış ve bu adresler ekrana yazdırılmıştır. ***Program 10.4*** ve program ***10.4'ün ekran çıktısını*** inceleyiniz.

Program 10.4'te ki uygulamadan ***void*** tipindeki işaretçilerin herhangi bir veri tipine işaret edebildiği görülmektedir.

10.4. İşaretçi Aritmetiği

Değişkenler üzerinde aritmetiksel işlemler yapılabildiği gibi, işaretçiler üzerinde de aritmetiksel işlemler yapılabilir. Örneğin işaretçi değişkenleri tamsayılarla arttırılıp azaltılabilir. Değeri arttırılan bir işaretçi bellekte bir sonraki alanı işaret eder. Değeri azaltılan bir işaretçi ise bellekte bir önceki alanı işaret eder.

Program 10.5'te dördüncü satırda beş elemanı olan ***int*** türünde bir dizi tanımlanmış ve dizi elemanlarına ilk değer ataması yapılmıştır. Programın beşinci satırında bir işaretçi değişken tanımlanmış ve altıncı satırda dizinin ilk elemanının adresi işaretçi değişkene atanmıştır. Yedinci satırda işaretçi değişkenin değeri ve işaret ettiği değişkenin değeri (dizinin 0. Elemanın değeri) yazdırılmış, sekizinci satırda arttırma operatörü kullanılarak işaretçi değişkenin değeri arttırılmıştır. Dokuzuncu satırda işaretçi değişkenin yeni değeri ve işaret ettiği değişkenin değeri (dizinin 1. Elemanın değeri) yazdırılmıştır.

Program 10.5'in ekran çıktısı incelendiğinde görülebileceği gibi, ***isr*** değişkeni tamsayı türünden bir işaretçi olduğu için işaretçi bir arttırılmış buna karşılık adres dört artmıştır.

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       int dizi[]={1,3,5,7,9};
5       int *isr;
6       isr=dizi;
7       cout<<isr<<" Adresindeki Değer = "<<*isr<<endl;
8       ++isr;
9       cout<<isr<<" Adresindeki Değer = "<<*isr<<endl;
10      return 0;
11  }
```

Program 10.5 İşaretçilerin arttırılıp azaltılması

Adreslerin dörder Byte olarak ilerlediğine dikkat ediniz

önce dizinin 0. elemanı sonra 1. elemanı yazdırıldı

```
0x6ffe20 Adresindeki Deger = 1
0x6ffe24 Adresindeki Deger = 3

--------------------------------
Process exited after 2.573 seconds with return value 0
Press any key to continue .
```

Program 10.5'in Ekran Çıktısı

10.5. İşaretçiler ve Diziler

Bölüm 6'da Diziler konusunu ele almıştık ve bir dizi tanımlanırken önce dizide yer alacak elemanların tipini, sonra ismini ve daha sonra da dizinin eleman sayısını(boyutunu) belirliyorduk. Örneğin 5 elemanlı, int tipinde sayıları tutacak bir diziyi aşağıdaki gibi tanımlıyorduk.

```
int dizi[5];
```

Bu tanımlama bellekte program için araka arkaya 4 Byte'lık 5 adet alan ayrılmasını sağlıyordu. Bu alanların adreslerine adres operatörü ile ulaşabiliriz.

Dizinin Elemanlarının Sırası	*Dizinin Elemanları*	*Dizi Elemanlarının Adresi*
İlk eleman	dizi[0]	&dizi[0]
İkinci eleman	dizi[1]	&dizi[1]
Üçüncü eleman	dizi[2]	&dizi[2]
Dördüncü eleman	dizi[3]	&dizi[3]
Beşinci Eleman	dizi[4]	&dizi[4]

Tablo 10.1 Dizi Elemenlarının Adreslerine Adres operatörü ile erişim

İşaretçilerle diziler arasındaki ilişkiyi, ***bir dizinin ismi, o dizinin ilk elemanının adresini tutan bir göstericidir*** şeklinde ifade edebiliriz. Dolayısı ile bu tanımdan faydalanılarak dizi elemanlarının adreslerinin iki farklı şekilde yazdırılabileceğini söyleyebiliriz. Tablo 10.2'de dizi elemanlarının adreslerinin iki farklı şekilde nasıl yazdırılacağı ve

dizi elemanlarının değerlerine, 6.bölümden bildiğimiz dizi operatörü ile ulaşılmasından farklı olarak, nasıl erişilebileceği gösterilmiştir.

Dizi Elemanlarının Sırası	Dizi Elemanlarının Adresi (1)	Dizi Elemanlarının Adresi (2)	Dizi Elemanlarının Değeri
İlk Eleman	&dizi[0]	dizi	*dizi
İkinci Eleman	&dizi[1]	dizi + 1	*(dizi + 1)
Üçüncü Eleman	&dizi[2]	dizi + 2	*(dizi + 2)
Dördüncü Eleman	&dizi[3]	dizi + 3	*(dizi + 3)
Beşinci Eleman	&dizi[4]	dizi + 4	*(dizi + 4)

Tablo 10.2 Dizi Elemanlarının Adreslerine ve Değerlerine Erişim

Program 10.6 dizi elemanlarının adreslerini ve dizi elemanlarını değerlerini ekrana yazdırmak için yazılmıştır.

Siz Program 10.6'yı *for* döngüsü veya *for* döngüleri kullanarak yeniden yazınız.

```
1  #include <iostream>
2  #include <locale.h>
3  using namespace std;
4  int main() {
5      setlocale(LC_ALL,"Turkish");
6      int dizi[5]={12,4,7,19,11};
7      cout<<"\n--Adres Operatörü ile Dizi Elamanlarının Adresinin Yazdırılması--\n";
8      cout<<&dizi[0]<<endl;
9      cout<<&dizi[1]<<endl;
10     cout<<&dizi[2]<<endl;
11     cout<<&dizi[3]<<endl;
12     cout<<&dizi[4]<<endl;
13     cout<<"\n--Dizi Adını(işaretçi) kullanarak Dizi Elemanlarının Adresinin Yazdırılması--\n";
14     cout<<dizi<<endl;
15     cout<<dizi+1<<endl;
16     cout<<dizi+2<<endl;
17     cout<<dizi+3<<endl;
18     cout<<dizi+4<<endl;
19     cout<<"\n--İşaretçi kullanarak Dizi Elemanlarının Değerlerinin Yazdırılıması--\n";
20     cout<<*(dizi)<<endl;
21     cout<<*(dizi+1)<<endl;
22     cout<<*(dizi+2)<<endl;
23     cout<<*(dizi+3)<<endl;
24     cout<<*(dizi+4)<<endl;
25     return 0;
26 }
```

Program 10.6 Farklı İki Yöntemle Dizi Elemenlarının Adresini ve İşaretçi Kullanarak Dizi Elemanlarının Değerlerini Yazdırmak

```
--Adres Operatörü ile Dizi Elamanlarının Adresinin Yazdırılması--
0x6ffe30
0x6ffe34
0x6ffe38
0x6ffe3c
0x6ffe40

--Dizi Adını(işaretçi) kullanarak Dizi Elemanlarının Adresinin Yazdırılması--
0x6ffe30
0x6ffe34
0x6ffe38
0x6ffe3c
0x6ffe40

--İşaretçi kullanarak Dizi Elemanlarının Değerlerinin Yazdırılıması--
12
4
7
19
11
```

Program 10.6'nın Ekran Çıktısı

10.6. İşaretçiler ve Fonksiyonlar

Program 10.7'den de görülebileceği gibi bir fonksiyonu referans ile çağırmak, o fonksiyona parametre olarak bir gösterici göndermekle aynı anlama gelmektedir. Fonksiyonu referansla çağırırken fonksiyona gönderilen adres orijinal verinin adresidir (*Program 10.7'de konuyu açıklamak için basit bir değişken kullanılmıştır*). Dolayısıyla fonksiyona gönderilen adresteki veri üzerinde yapılan her değişiklik aslında orijinal veriyi değiştirir. Program 10.7'yi ve Program 10.7'nin ekran çıktısını inceleyiniz.

```cpp
#include <iostream>
using namespace std;
int main() {
    void cevir_m_cm(double*);
    double metre=10.3;
    cout<<" "<<metre <<" metre "<<endl;
    cevir_m_cm(&metre);
    cout<<" "<<metre <<" cm "<<endl;
    return 0;
}
void cevir_m_cm(double* s_cm)
{
    *s_cm*=100;
}
```

Program 10.7 Fonksiyona parametre olarak işaretçi göndermek

```
 10.3 metre
 1030 cm

--------------------------------
Process exited after 33.68 seconds with return value 0
Press any key to continue .
```

Program 107'ninEkran Çıktısı

Program 10.7'de ***main()*** fonksiyonu içerisinde ilk değer ataması yapılan ***double*** değişkenin adresi parametre olacak şekilde ***cevir_m_cm()*** çağırıldı. ***cevir_m_cm()*** fonksiyon içerisinde veri değiştirildi(). Daha sonra ***main()*** fonksiyonu içerisinde değişen veri gösterildi.

Basit değişkenlerin fonksiyonlara aktarılması sırasında olduğu gibi, fonksiyonlara dizi aktarılırken de işaretçileri kullanmak yaygın bir yaklaşımdır. Program 10.8, fonksiyonlara dizi aktarılırken işaretçilerin nasıl kullanılacağını açıklamak için yazılmıştır.

```cpp
#include <iostream>
#include <locale.h>
using namespace std;
const int Eleman=6;
int main() {
    setlocale(LC_ALL,"Turkish");
    void cevir(double*);
    double dizi[Eleman]={11.3,9.8,5.5,12.5,7.8,14.6};
cout<<"Orijinal Dizi(Elemanlar metre)"<<endl;
    for(int i=0;i<Eleman;i++)
    cout<<dizi[i]<<"  ";
    cout<<endl<<endl;
    cevir(dizi);
cout<<"Fonksiyonda Elemanları santimetreye çevrilmiş Dizi"<<endl;
    for(int i=0;i<Eleman;i++)
    cout<<dizi[i]<<"  ";
    cout<<endl;
    return 0;
}
void cevir(double* d_el)
{
    for(int i=0;i<Eleman;i++)
    *(d_el+i)*=100;
}
```

Program 10.8 Dizileri Fonksiyonlara Aktarmak

```
Orijinal Dizi(Elemanlar metre)
11.3  9.8  5.5  12.5  7.8  14.6

Fonksiyonda Elemanları santimetreye çevrilmiş Dizi
1130  980  550  1250  780  1460
```

Program 10.8'in Ekran Çıktısı

Program 10.8'in sekizinci satırında ***dizi*** isminde altı elemana sahip ve elemanları double olan bir dizi tanımladık ve dizi elemanlarına metre cinsinden ilk değerler atadık. Onuncu satırda diziyi fonksiyona göndermeden for döngüsü ile dizinin elemanlarını ekrana yazdırdık. On üçüncü satırda diziyi, dizi ismini parametre olarak kullanarak cevir() fonksiyonuna

gönderdik. Bu fonksiyon içerisinde içerik operatörünü kullanarak dizi elemanlarını santimetreye çevirdik. On beşinci satırda dizi elemanlarını santimetre olarak tekrar yazdırdık.

Bu örnekte iki husussa dikkat etmelisiniz. Bunlardan birincisi fonksiyona sadece dizinin ismi aktarıldı(Dizinin isminin, diziye ait ilk elemanın adresi olduğundan yararlandık). İkincisi ise dizinin son sınırı için hem main fonksiyonundan, hem de cevir fonksiyonundan ulaşılabilmesi için global değişken (***Eleman***) tanımladık.

Aşağıda verilen ***program 10.9***, elemanları küçük harf olarak atanmış bir karakter dizisini büyük harfe ve elemanları büyük harfe dönüştürülmüş bu dizinin elemanlarını tekrar küçük harfe dönüştürmektedir. Bunun için programda ***main()*** fonksiyonu dışında iki ayrı fonksiyon yazılmıştır. ***bharf()*** isimli fonksiyon kendisine gelen ve elemanları küçük harf olan diziyi büyük harfe çevirmektedir. ***kharf()*** isimli fonksiyon ise bu işlemin tam tersini yapmaktadır.

```
#include <iostream>
using namespace std;
void bharf(char*,int);
void kharf(char*,int);
int main() {
    char dizi[6]={'a','n','K','a','r','a'};
    int i=6;
    bharf(dizi,i); //dizinin elmanlarını büyük harfe çevirmek için
    for(i=0;i<6;i++)
    cout<<dizi[i]<<"  ";
    cout<<endl;
    kharf(dizi,i);//Büyük harfe dönüşen dizi elemanlarını küçük harfe çevirmek için
    for(i=0;i<6;i++)
    cout<<dizi[i]<<"  ";
    cout<<endl;
    return 0;
}
void bharf(char *kdizi, int ii){          //Küçük harfi
    for(int i=0;i<ii;i++){                //büyük harfe
        if(kdizi[i]>='a' && kdizi[i]<='z')//çeviren fonksiyon
            kdizi[i]=kdizi[i]-32;
        }
}
void kharf(char *bdizi, int ii){           //Büyük harfi
    for(int i=0;i<ii;i++){                 //küçük harfe
        if(bdizi[i]>='A' && bdizi[i]<='Z') //çeviren fonksiyon
        bdizi[i]=bdizi[i]+32;
        }
}
```

Program 10.9 küçük harf verlen diziyi büyük harfe, büyük harf verilen diziyi küçük harfe çevirmek

```
A  N  K  A  R  A
a  n  k  a  r  a
```

Program 10.9'un Ekran Çıktısı

```
void bharf(char *kdizi, int ii){
    for(int i=0;i<ii;i++){
        if(kdizi[i]>='a' && kdizi[i]<='z')
            kdizi[i]=kdizi[i]-32;
        }
}
void kharf(char *bdizi, int ii){
    for(int i=0;i<ii;i++){
        if(bdizi[i]>='A' && bdizi[i]<='Z')
        bdizi[i]=bdizi[i]+32;
        }
}
```

ARAŞTIRINIZ! *32'nin çıkarılmasının sebebi nedir?*

ARAŞTIRINIZ! *32'nin eklenmesinin sebebi nedir?*

Program 10.9'un küçük bir problemi var. Bu problem, program Türkçe karakterleri küçük harften büyük harfe veya büyük harften küçük harfe dönüştürmüyor. Bunun için bir araştırma yaparak programa Türkçe karakterleri de dönüştürecek bir özellik katmayı deneyiniz.

10.7. Karakter Katarlarının Fonksiyon Argümanı Olarak Kullanılması

Program ***10.10'***un listesinde yer alan ***yazdır()*** fonksiyonu karakter katarı içerisindeki her karaktere sırayla erişerek karakter katarını ekrana yazdırır. Programda ***kk1*** dizi adresi, yazdır fonksiyonu çağrılırken argüman olarak kullanılmıştır. Bu adres bir sabittir, fakat fonksiyona değer olarak aktarılır ve fonksiyon içerisinde bir kopyası çıkarılır.

Bu programda kopya işaretçi olan ***kk***'dır. Bir işaretçi sabitten farklı olarak değiştirilebilir. İşaretçinin bu özelliğinden faydalanılarak fonksiyonda ***kk*** arttırılır ve ****kk++*** ifadesi de karakter katarının arka arkaya gelen karakterlerini döndürür. Döngü karakter katarının ***null*** değerine kadar devam eder ve karakter katarının tamamı ekrana yazdırılır.

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       void yazdir(char*);
5       char kkl[]="Karakter Katari Sabiti";
6       yazdir(kkl);
7       return 0;
8   }
9   void yazdir(char* kk)
10  {
11      while(*kk)
12      {
13              cout<<*kk++;
14              }
15      cout<<endl;
16  }
```

Program 10.11Karakter Katarları ve fonk. Arg.

```
Karakter Katari Sabiti

--------------------------------
Process exited after 0.06507 seconds with return value 0
Press any key to continue .
```

Program 10.11'in Ekran Çoktısı

Bölüm Özeti

Diğer dillerde işaretçi olmamasına karşılık C++ ve C'nin en önemli özelliği işaretçileri destekliyor olmalarıdır. Bu dillerde işaretçi olmadan da birçok işlem yapılabileceğini bundan önceki bölümlerde gördük. Fakat veri yapılarını tanımlamak, sistemden bellek almak, bazı operatörleri kullanabilmek için işaretçileri kullanmamız gerekmektedir. Bu bölümün işaretçilere ayrılmasının nedeni programlama ile uğraşan herkesin işaretçiler konusunda bir fikri olması gerekliliğindendir.

İşaretçilerin bir değişken olduğunu ve içerisinde başka bir değişkenin adresini tuttuğunu, adres operatörünün(&) değişkenlerin bellek adreslerini öğrenmek için kullanıldığını, işaretçi tanımlarken değişkenin başına asteriks(*) işareti konulması gerektiğini, bu şekilde tanımlanan değişkenin adres bilgisi tutması gerektiğini, işaretçiye değer atarken kullanılan asteriks(*) işareti ile gösterici tanımlarken kullanılan asteriks(*) işaretinin farklı anlamlara geldiğini, gösterici kullanılırken, göstericiye kendi tipinde bir değer atanması gerektiğini, aksi halde veri kaybı yaşanabileceğini, işaretçilerle aritmetiksel işlemler yapılabildiğini öğrendik.

Bunlardan başka işaretçilerin fonksiyonlara aktarılabileceğini ve bunun işaretçilerin en faydalı kullanım şekillerinden biri olduğunu, Bir dizinin isminin, o dizinin ilk elemanının bellek adresi olduğunu, dizi ismi de bir işaretçi olduğu için işaretçilerle yapılan matematiksel işlemlerin dizi ismi ile de yapılabileceğini gördük.

Bunlara ek olarak C++ Programlarında dizi adresinin(dizinin ismi), fonksiyon çağırılırken argüman olarak Kullanılabileceğini bir örnekle gösterdik.

ÜNİTE SORULARI

1. Bellekte bir byte 'tan fazla yer alan değişkenlerin adresi için aşağıdakilerden hangisi söylenebilir?

- Bu değişkenlerin adresi bellekte kapladıkları alanın son Byte' nın adresidir.
- Bu değişkenlerin adresi bellekte kapladıkları alanın ilk Byte' nın adresidir.
- Bu değişkenlerin adresi bellekte kapladıkları alanın ikinci Byte' nın adresidir.
- Bu değişkenlerin adresi bellekte kapladıkları alanın ortasındaki Byte' ın adresidir.
- Bu değişkenlerin adresi Bellekte kapladıkların alanların adreslerinin toplamı ile ifade edilir.

2. Adres operatörü için aşağıdakilerden hangisi doğru değildir?

- Sabitlerin bellekte hangi adreslere yerleştiklerini öğrenmek için kullanılabilir.

- Değişkenlerin bellekte hangi adrese yerleştiklerini öğrenmek için kullanılabilir.
- C++ programlama dilinde adres operatörü & işareti ile temsil edilir.
- Tamsayı tipinde(int) sayi isimli bir değişkenimiz varsa bu değişkenin adresini bilgisayar ekranında göstermek için cout<<&sayi; yazmak yeterlidir.
- Adres operatörü kullanılarak değişkenin değeri yazdırılır.

3. İşaretçi değişkenler tanımlanırken değişkenin başına hangi işaret konulur?

- # İşareti
- & İşareti
- * İşareti
- ? İşareti
- Herhangi bir işaret konulmaz.

4. Bellek adreslerinin değerlerini tutan değişkenlere işaretçi değişken denir. İşaretçi değişkenlerin tipleri için aşağıdakilerden hangisi doğrudur?

- Bütün işaretçi değişkenler tamsayı (int) tipindedir.
- Bütün işaretçi değişkenler karakter(char) tipindedir.
- Bütün işaretçi değişkenler gerçek sayı(float) tipindedir.
- Bütün işaretçi değişkenlerin tipleri string olarak tanımlanır.
- İşaretçi değişkenlerin tipleri genellikle adreslerini tutacakları değişkenin tipiyle aynı olur.

5.

```
1  #include <iostream>
2  using namespace std;
3  int sayi1,sayi2;
4  int main() {
5      int sayi3,sayi4;
6      int* i1, *i2,*i3,*i4;
7      i1=&sayi1;i2=&sayi2;
8      i3=&sayi3;i4=&sayi4;
9      cout<<i1<<endl;
10     cout<<i2<<endl<<endl;
11     cout<<i3<<endl;
12     cout<<i4<<endl;
13     return 0;
14 }
```

Yukarıda verilen C++ programında işaretçi değişkenler kaçıncı satırda tanımlanmıştır?

- 3
- 5
- 6
- 7
- 8

6.

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int s1=5;
5      int s2=7;
6      int *isr;
7      isr=&s1;
8      cout<<" "<<*isr<<endl;
9      isr=&s2;
10     cout<<" "<<*isr<<endl;
11     return 0;
12 }
```

Yukarıda verilen programın ekran çıktısı için aşağıdakilerden hangisi söylenebilir?

- Ekrana yalnızca s1 değişkeninin değeri (5) yazdırılır.
- Ekrana yalnızca s2 değişkeninin değeri (7) yazdırılır.
- Ekrana s1 ve s2 değişkenlerinin değerleri yazdırılır.
- Ekrana s1 ve s2 değişkenlerinin adresleri yazdırılır.
- Ekrana *isr işaretçisinin adresi yazdırılır.

7.

```
#include <iostream>
using namespace std;
int main() {
    int s1=5;
    int s2=7;
    int *isr;
    isr=&s1;
    cout<<" "<<isr<<endl;
    isr=&s2;
    cout<<" "<<isr<<endl;
    return 0;
}
```

Yukarıda verilen programın ekran çıktısı için aşağıdakilerden hangisi söylenebilir?

- Ekrana yalnızca s1 değişkeninin değeri (5) yazdırılır.
- Ekrana yalnızca s2 değişkeninin değeri (7) yazdırılır.
- Ekrana s1 ve s2 değişkenlerinin değerleri yazdırılır.
- Ekrana s1 ve s2 değişkenlerinin adresleri yazdırılır.
- Ekrana *isr işaretçisinin adresi yazdırılır.

8. Tanımlanan bir işaretçinin herhangi bir veri tipine işaret edebilmesi için tipinin ne olması gerekir?

- int
- void
- double
- char
- float

9.

```
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int dizi[5]={12,4,7,19,11};
    cout<<*(dizi)<<endl;
    return 0;
}
```

Yukarıdaki programın ekran çıktısı aşağıdakilerden hangisidir?

- Dizini elemanlarının tamamı ekrana yazdırılır.
- 12
- 4
- 7
- 19

10.

```
void  fonk(char *bdizi, int ii){
    for(int i=0;i<ii;i++){
        if(bdizi[i]>='A' && bdizi[i]<='Z')
        bdizi[i]=bdizi[i]+32;
        }
           }
```

Yukarıda verilen fonksiyon kendisine gelen karakter dizisini ve dizinin eleman sayısını parametre olarak almaktadır. Fonksiyon aşağıdaki işlemlerden hangisini gerçekleştirmektedir?

- Dizi elemanlarına sayısal değer ekler.
- Bu fonksiyonun kodunda yapılan syntax hatasından dolayı program çalışmaz.
- Dizi elemanlarının tamamı büyük harfe çevrilir.
- Dizi elemanlarının tamamı küçük harfe çevrilir.
- Program çalışır, dizi elemanları üzerinde herhangi bir işlem yapmaz.
  -
  - CEVAP ANAHTARI 1. b2. e3. c4. e5. c6. c7. d8. b9. b10. d

11. C++ KULLANILAN HAZIR FONKSİYONLAR

11.1. cmath Kütüphanesi ve Matematik Fonksiyonlar

C++ çok sayıda matematiksel fonksiyona sahiptir. Bu fonksiyonların C++ programları içerisinde kullanılabilmesi için, programa ***cmath*** kütüphanesi eklenmelidir. Çünkü fonksiyonların prototipleri ***cmath*** dosyası içerisinde yer almaktadır. Matematiksel fonksiyonların çoğu double veri tipindedir. Bu nedenle matematiksel fonksiyonlar parametre olarak double türündeki değişkenleri alır ve bu fonksiyonların geri dönüş değerleri double tipindedir. Bu bölümde ***cmath*** kütüphanesinde yer alan matematiksel fonksiyonlardan önemli olan bazılarının C++ programlarında kullanılması ile ilgili örnekler verilecektir.

11.1.1. exp()

exp() fonksiyonu matematiksel **e** sabitinin(2.718…) **x.** dereceden kuvvetini döndürür.

Örnek: e sabitinin klavyeden girilen kuvvetini hesaplayan C++ programı yazınız?

```
1   #include <iostream>
2   #include <cmath>
3   #include <conio.h>
4   #include <locale.h>
5   using namespace std;
6   int main() {
7       setlocale(LC_ALL,"Turkish");
8       double sayi;
9       char c='e';
10      do{
11      cout<<endl<<"e sabitinin kaçıncı kuvvetini hesaplatacaksınız : ";
12      cin>>sayi;
13      cout<<endl<<"e üzeri "<<sayi<<": "<<exp(sayi)<<endl;
14      cout<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h)   : ";
15      c=getche();
16      cout<<endl;
17      }while(c!='h');
18      return 0;
19  }
```

Program 11.1 exp() fonksiyonuna örnek C++ programı

exp() fonksiyonu 13. Satırda kullanılmıştır. do… while() döngüsü içerisinde kullanıcı **"hesaplamaya devam edecek misiniz(çıkış: h) : "** sorusuna hayır(**'h'**) cevabını verinceye kadar klavyeden girilen sayılar fonksiyona parametre olarak gönderilmiştir. Dönenen değer yine 13. Satırda ekrana yazdırılmıştır. Aşağıda program 11.5'in çalıştırılması sonucunda klavyeden girilen 0, 1, 5, 10 sayılarına karşılık e^0, e^1, e^5, e^{10} değerleri ekran yazdırılmıştır.

```
e sabitinin kaçıncı kuvvetini hesaplatacaksınız : 0

e üzeri 0: 1

hesaplamaya devam edecek misiniz(çıkış:h)  : e

e sabitinin kaçıncı kuvvetini hesaplatacaksınız : 1

e üzeri 1: 2.71828

hesaplamaya devam edecek misiniz(çıkış:h)  : e

e sabitinin kaçıncı kuvvetini hesaplatacaksınız : 5

e üzeri 5: 148.413

hesaplamaya devam edecek misiniz(çıkış:h)  : e

e sabitinin kaçıncı kuvvetini hesaplatacaksınız : 10

e üzeri 10: 22026.5

hesaplamaya devam edecek misiniz(çıkış:h)  :
```

Program 11.1'in Ekran Çıktısı

11.1.2. sqrt()

sqrt() fonksiyonu kendisine parametre olarak gönderilen sayının karekökünü döndürür.

Örnek: Klavyeden girilen sayının karekökünü **sqrt()** fonksiyonu kullanılarak hesaplayan C++ programı yazınız?

```
1  #include <iostream>
2  #include <cmath>
3  #include <conio.h>
4  #include <locale.h>
5  using namespace std;
6  int main() {
7      setlocale(LC_ALL,"Turkish");
8      double sayi;
9      char c='e';
10     do{
11     cout<<endl<<"Karekökünü Hespalanmasını İstediğiniz Sayıyı Giriniz : ";
12     cin>>sayi;
13     cout<<endl<<" "<<sayi<<" \'in karekökü = "<<sqrt(sayi)<<endl;
14     cout<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h)  : ";
15     c=getche();
16     cout<<endl;
17     }while(c!='h');
18     return 0;
19 }
```

Program 11.2 sqrt() fonksiyonunun Kullanımına Örnek

Yukarıdaki program klavyeden girilen sayının karekökünü almakta sonucu ekrana yazdırmaktadır. Programın çalışması, kullanıcı **"hesaplamaya devam edecek misiniz (çıkış: h) : "** sorusuna hayır (**'h'**) cevabını verinceye kadar devam etmekte ve klavyeden girilen sayıların karekökünü almaktadır. Aşağıda programın çıktısı ile ilgili bir örnek verilmiştir.

```
Karekökünü Hespalanmasını İstediğiniz Sayıyı Giriniz : 4

 4 'in karekökü = 2

hesaplamaya devam edecek misiniz(çıkış:h)  : e

Karekökünü Hespalanmasını İstediğiniz Sayıyı Giriniz : 7

 7 'in karekökü = 2.64575

hesaplamaya devam edecek misiniz(çıkış:h)  : h
```

Program 11.2'nin Ekran Çıktısı

11.1.3. pow()

pow() fonksiyonu parametre olarak x ve y gibi sayı alır ve x üzeri y'yi (x^y) döndürür. Fonksiyonun prototipi double pow(double x, double y) şeklindedir.

Örnek: $1/1^4 + 1/2^4 + 1/3^4 + 1/4^4 + 1/5^4$ serisinin toplamını hesaplayıp sonucunu ekrana yazdıran C++ programını yazınız?

```
#include <iostream>
#include <cmath>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    double toplam=0.0;
    for(int i=1;i<=5;i++)
    {
        toplam+=(1.0/pow(i,4));
    }
    cout<<"Serinin toplamı : "<<toplam<<endl;
    return 0;
}
```

Program 11.3 pow() Fonksiyonunun Kullanımı

```
Serinin toplamı : 1.08035

--------------------------------
Process exited after 2.46 seconds with return value 0
Press any key to continue . . .
```

Program 11.3'ün Ekran Çıktısı

11.1.4. ceil() Fonksiyonu

ceil(), fonksiyonu x 'ten sonraki ilk tamsayı değerini döndürür. Örneğin **ceil()** fonksiyonuna 10.3 sayısı parametre olarak verildiğinde 11 sayısı geri döner. Fonksiyona -4.8 parametre olarak verildiğinde -4 sayısı geri döner. **ceil()** fonksiyonuna parametre olarak tamsayı verilirse geri dönen değer fonksiyona parametre olarak verilen sayıdır. **ceil()** fonksiyonu kendisine parametre olarak verilen sayıyı yukarı(kendisinden büyük ilk sayıya) yuvarlar. Fonksiyonun prototipi **double ceil(double sayi)** şeklindedir.

Örnek: Klavyeden girilen sayıları **ceil()** fonksiyonu kullanarak yukarı yuvarlanmasını sağlayan C++ programını yazınız?

```
#include <iostream>
#include <cmath>    // matematik kütüphanesi
#include <conio.h> //getche() Kullanabilmek için
#include<locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    double sayi;
    char c='e';
    do{
    cout<<endl<<"Yuvarlanacak Sayıyı Giriniz : ";
    cin>>sayi;
    cout<<endl<<"ceil("<<sayi<<"):"<<ceil(sayi)<<endl;

    cout<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h) : ";
    c=getche();
    cout<<endl;
    }while(c!='h');
    return 0;
}
```

Program 11.4 ceil() Fonksiyonunun Kullanımına Örnek

```
Yuvarlanacak Sayıyı Giriniz : 10.3

ceil(10.3):11

hesaplamaya devam edecek misiniz(çıkış:h) : e

Yuvarlanacak Sayıyı Giriniz : 9

ceil(9):9

hesaplamaya devam edecek misiniz(çıkış:h) : e

Yuvarlanacak Sayıyı Giriniz : -4.8

ceil(-4.8):-4

hesaplamaya devam edecek misiniz(çıkış:h) :
```

Program 11.4'ün Ekran çıktısı

11.1.5. floor() Fonksiyonu

floor() fonksiyonu kendisine parametre olarak gönderilen sayıyı alıp, kendisinden küçük ilk tamsayıyı geri döndürür. Prototipi **double floor(double sayi)** şeklindedir. Örneğin **floor()** fonksiyonu parametre olarak 10.3 sayısını alınca geriye 10 sayısını döndürür. Fonksiyon 9 sayısını parametre olarak alırsa 9 sayısını geri döndürür. **floor()** fonksiyonu -4.8 sayısını parametre olarak alırsa -5 sayısını geri döndürür. Bir cümle halinde floor fonksiyonunu tarif etmek gerekirse, **floor()** fonksiyonu kendisine parametre olarak verilen sayıyı aşağı ç(kendisinden küçük ilk sayıya) yuvarlar. **floor()** fonksiyonuna parametre olarak tamsayı verilirse, geri dönen değer fonksiyona parametre olarak verilen sayı olur.

```
1   #include <iostream>
2   #include <cmath>     // matematik kütüphanesi
3   #include <conio.h> //getche() Kullanabilmek için
4   #include<locale.h>
5   using namespace std;
6   int main() {
7       setlocale(LC_ALL,"Turkish");
8       double sayi;
9       char c='e';
10      do{
11      cout<<endl<<"Yuvarlanacak Sayıyı Giriniz : ";
12      cin>>sayi;
13      cout<<endl<<"floor("<<sayi<<"):"<<floor(sayi)<<endl;
14      cout<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h) : ";
15      c=getche();
16      cout<<endl;
17      }while(c!='h');
18      return 0;
19  }
```

Program 11.5 floor Fonksiyonunun Kullanımına Örnek

```
Yuvarlanacak Sayıyı Giriniz : 10.3

floor(10.3):10

hesaplamaya devam edecek misiniz(çıkış:h) : e

Yuvarlanacak Sayıyı Giriniz : 9

floor(9):9

hesaplamaya devam edecek misiniz(çıkış:h) : e

Yuvarlanacak Sayıyı Giriniz : -4.8

floor(-4.8):-5

hesaplamaya devam edecek misiniz(çıkış:h) :
```

Program 11.5'in Ekran Çıktısı

11.1.6. log() Fonksiyonu

log() fonksiyonu kendisine parametre olarak gönderilen sayının ***e*** tabanında logaritmasını döndürür. Başka bir ifade ile log(x) fonksiyonu **lnx** 'i hesaplar

Örnek: Klavyeden girilen sayının doğal logaritmasını hesaplayan ve sonucu ekrana yazdıran C++ programını **log()** fonksiyonunu kullanarak yazınız?

```
#include <iostream>
#include <cmath>    // matematik kütüphanesi
#include <conio.h> //getche() Kullanabilmek için
#include<locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    double sayi;
    char c='e';
    do{
    cout<<endl<<"Logaritması hesaplanacak Sayıyı Giriniz : ";
    cin>>sayi;
    cout<<endl<<"log("<<sayi<<"):"<<log(sayi)<<endl;
    cout<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h) : ";
    c=getche();
    cout<<endl;
    }while(c!='h');
    return 0;
}
```

Program 11.6 log() fonksiyonunun kullanımına örnek

```
Logaritması hesaplanacak Sayıyı Giriniz : 3

log(3):1.09861

hesaplamaya devam edecek misiniz(çıkış:h) : e

Logaritması hesaplanacak Sayıyı Giriniz : 2.718282

log(2.71828):1

hesaplamaya devam edecek misiniz(çıkış:h) :
```

Program 11.6'nin Ekran çıktısı

11.1.7. fabs() Fonksiyonu

fabs() fonksiyonu kendisine gönderilen parametre negatifse geri dönüş değeri gönderilen parametreyi pozitif olarak geri döndürür. Gönderilen parametre pozitif ise geri dönen değer yine pozitiftir. Başka bir ifade ile **fabs()** fonksiyonu gönderilen parametrenin mutlak değerini alır.

```
#include <iostream>
#include <cmath>    // matematik kütüphanesi
#include <conio.h> //getche() Kullanabilmek için
#include<locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    double sayi;
    char c='e';
    do{
    cout<<endl<<"Mutlak değeri hesaplanacak Sayıyı Giriniz : ";
    cin>>sayi;
    cout<<endl<<"abs("<<sayi<<"):"<<fabs(sayi)<<endl;
    cout<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h) : ";
    c=getche();
    cout<<endl;
    }while(c!='h');
    return 0;
}
```

Program 11.7 fabs() Fonksiyonunun kullanılmasına Örnek

```
Mutlak değeri hesaplanacak Sayıyı Giriniz : -5

abs(-5):5

hesaplamaya devam edecek misiniz(çıkış:h) : -

Mutlak değeri hesaplanacak Sayıyı Giriniz : -89

abs(-89):89

hesaplamaya devam edecek misiniz(çıkış:h) : h

--------------------------------
Process exited after 34.47 seconds with return value 0
Press any key to continue . . .
```

Program 11.7'in Ekran Çıktısı

11.1.8. log10() Fonksiyonu

log10() fonksiyonu kendisine parametre olarak gönderilen sayının 10 tabanındaki logaritmasını hesaplar ve geri döndürür.

Örnek: Klavyeden girilen sayının 10 tabanındaki logaritmasını hesaplayan ve sonucu ekrana yazdıran programı yazınız.

```
1  #include <iostream>
2  #include <cmath>     // matematik kütüphanesi
3  #include <conio.h> //getche() Kullanabilmek için
4  #include<locale.h>
5  using namespace std;
6  int main() {
7      setlocale(LC_ALL,"Turkish");
8      double sayi;
9      char c='e';
10     do{
11     cout<<endl<<"Logaritması hesaplanacak Sayıyı Giriniz : ";
12     cin>>sayi;
13     cout<<endl<<"log10("<<sayi<<"):"<<log10(sayi)<<endl;
14     cout<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h) : ";
15     c=getche();
16     cout<<endl;
17     }while(c!='h');
18     return 0;
19 }
```

Program 11.8 log10() fonksiyonunun Kullanılmasına Örnek

```
Logaritması hesaplanacak Sayıyı Giriniz : 3

log10(3):0.477121

hesaplamaya devam edecek misiniz(çıkış:h) : e

Logaritması hesaplanacak Sayıyı Giriniz : 2.718282

log10(2.71828):0.434295

hesaplamaya devam edecek misiniz(çıkış:h) : e

Logaritması hesaplanacak Sayıyı Giriniz : 10

log10(10):1

hesaplamaya devam edecek misiniz(çıkış:h) : e

Logaritması hesaplanacak Sayıyı Giriniz : 1

log10(1):0

hesaplamaya devam edecek misiniz(çıkış:h) :
```

Program 11.8'in Ekran Çıktısı

11.1.9. Trigonometrik Fonksiyonlar

Trigonometrik fonksiyonlar kendisine parametre olarak verilen değeri radyan olarak kabul eder. Bu fonksiyonlar radyan olarak verilen parametre değerini alır, sonucu hesaplar ve geri dönüş değerleri yine radyan cinsindendir. Trigonometrik fonksiyonlar kullanılarak açıların derece cinsinden kosinüs, sinüs, tanjant vb. değerleri hesaplanması gerektiği durumlarda ***radyan=(π/180)*derece*** dönüşümü kullanılmalıdır.

cos(x) fonksiyonu radyan olarak verilen x açısının kosinüsünü hesaplar ve hesaplanan değeri geri döndürür.

sin(x) fonksiyonu radyan olarak verilen x açısının sinüsünü hesaplar ve hesaplanan değeri geri döndürür.

tan(x) fonksiyonu radyan olarak verilen x açısının tanjantını hesaplar ve hesaplanan değeri geri döndürür.

Örnek: Klavyeden girilen açının sinüs, kosinüs ve tanjantını hesaplayan ve sonucu ekrana yazdıran programı yazınız.

```
#include <iostream>
#include <cmath>
#include <conio.h>
#include<locale.h>
float const PI=3.141593;
using namespace std;
int main ( )
{
    setlocale(LC_ALL,"Turkish");
    double aci_1,aci_2;
    char c='e';
    do{
    cout<<"Trigonemetrik Değerleri Hesaplanacak Açı: ";
    cin>>aci_1;
    aci_2=aci_1;
    aci_1 *= (PI/180.0);
    cout<<"cos("<<aci_2<<")= "<<cos(aci_1)<<endl;
    cout<<"sin("<<aci_2<<")= "<<sin(aci_1)<<endl;
    cout<<"tan("<<aci_2<<")= "<<tan(aci_1)<<endl;
    cout<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h) : ";
    c=getche();
    cout<<endl;
    }while(c!='h');
    return 0;
}
```

Program 11.9 Trigonometrik fonksiyonların kullanılmasına örnek

```
Trigonemetrik Değerleri Hesaplanacak Açı: 0
cos(0)= 1
sin(0)= 0
tan(0)= 0

hesaplamaya devam edecek misiniz(çıkış:h) : e
Trigonemetrik Değerleri Hesaplanacak Açı: 45
cos(45)= 0.707107
sin(45)= 0.707107
tan(45)= 1

hesaplamaya devam edecek misiniz(çıkış:h) :
```

Program 11.9'un Ekran Çıktısı

11.2. cctype Kütüphanesi ve Karakter Fonksiyonları

C++ programlama dilinde karakterler üzerinde işlem yapan fonksiyonların prototipleri ***cctype*** kütüphanesi içerisinde yer almaktadır. Bu nedenle C++ programları içerisinde, karakterler üzerinde işlem yapan hazır fonksiyonların kullanılabilmesi için, programa ***cctype*** kütüphanesi eklenmelidir.

Karakter fonksiyonları tek bir karakter üzerinde işlem yapan fonksiyonlardır. Bu fonksiyonlardan bir kısmı parametre olarak aldığı karakteri değiştirirken, diğer bir kısmı da parametreyi sorgular ve geriye true veya false değeri döndürür.

Bu bölümde yukarıda bahsedilen ve cctype kütüphanesinde prototipleri bulunan karakter fonksiyonlarına örnekler verilecektir.

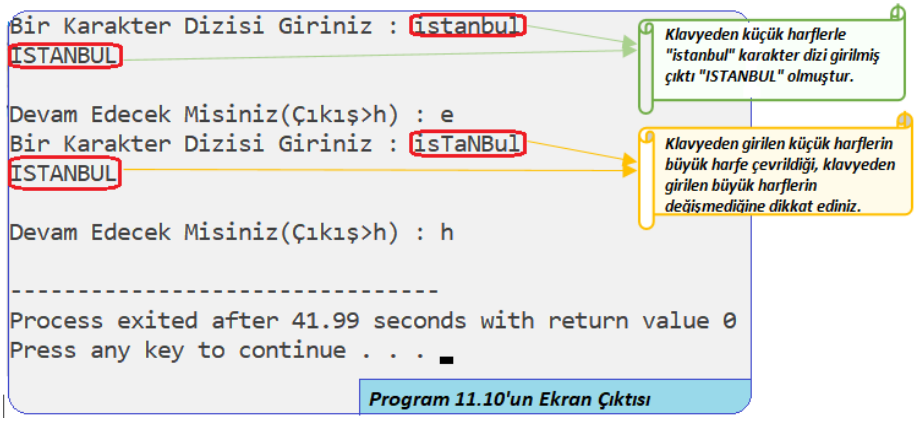
11.2.1. toupper() Fonksiyonu

toupper() fonksiyonu, parametre olarak aldığı karakteri büyük harfe çevirir. Eğer fonksiyonun aldığı parametre(karakter) büyük harf ise bir işlem yapılmaz.

Örnek: Klavyeden girilen karakter dizisinin küçük harflerini büyük harfe çeviren C++ programını yazınız?

```
1   #include <iostream>
2   #include <cctype>
3   #include <cstring>
4   #include <conio.h>
5   #include <locale.h>
6   using namespace std;
7   int main() {
8       setlocale(LC_ALL,"Turkish");
9       char dizi[21];
10      char c='e';
11      do{
12      cout<<"Bir Karakter Dizisi Giriniz : ";
13      gets(dizi);
14      /*gets() fonksiyonu klavyeden istenilen sayıda karakterin okunarak
15      belleğe yazılmasını sağlar*/
16      for(int i=0;i<strlen(dizi);i++) //strlen() dizinin eleman sayısını hesaplar
17      {
18          if(islower(dizi[i]))
19          dizi[i]=toupper(dizi[i]);}
20      /*puts() fonksiyonu parametre olarak bir adres(dizi ismi veya gösterici)
21      alır. bu adresten NULL karaktere kadar olan kısmı ekrana yazdırır. */
22      puts(dizi);
23      cout<<endl<<"Devam Edecek Misiniz(Çıkış>h) : ";
24      c=getche();
25      cout<<endl;
26      }while(c!='h');
27      return 0; }
```

Program 11.10 toupper() fonksiyonunun kullanılmasına örnek



Program 11.10'un Ekran Çıktısı

11.2.2. tolower() Fonksiyonu

tolower() fonksiyonu, parametre olarak aldığı karakteri küçük harfe çevirir. Eğer fonksiyonun aldığı parametre(karakter) küçük harf ise bir işlem yapılmaz.

Örnek: Klavyeden girilen karakter dizisinin büyük harflerini küçük harfe çeviren C++ programını yazınız?

```cpp
#include <iostream>
#include <cctype>
#include <cstring>
#include <conio.h>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    char dizi[21], c='e';
    do{
    cout<<"Bir Karakter Dizisi Giriniz : ";
    gets(dizi);
    /*gets() fonksiyonu klavyeden istenilen sayıda karakterin okunarak
    belleğe yazılmasını sağlar*/
    for(int i=0;i<strlen(dizi);i++) //strlen() dizinin eleman sayısını hesaplar
    {
        if(isupper(dizi[i])) /*fonksiyonun parametre olarak aldığı karakter
        büyük harf ise geriye dönen değer bir, küçük harf ise geriye dönen değer sıfır olur*/
        dizi[i]=tolower(dizi[i]);}
    puts(dizi);
    /*puts() fonksiyonu parametre olarak bir adres(dizi ismi veya gösterici)
    alır. bu adresten NULL karaktere kadar olan kısmı ekrana yazdırır. */
    cout<<endl<<"Devam Edecek Misiniz(Çıkış>h) : ";
    c=getche();
    cout<<endl;
    }while(c!='h');
    return 0; }
```

Program11.11 tolower() fonksiyonunun kullanımına örnek

```
Bir Karakter Dizisi Giriniz : AUZEF
auzef

Devam Edecek Misiniz(Çıkış>h) : e
Bir Karakter Dizisi Giriniz : AUZeF
auzef

Devam Edecek Misiniz(Çıkış>h) : h

--------------------------------
Process exited after 39.03 seconds with return value 0
Press any key to continue . . .
```

Klavyeden büyük harflerle "AUZEF" karakter dizisi girilmiş ve çıktı "auzef" olmuştur.

Klavyeden girilen büyük harflerin küçük harflere çevrildiğine ve küçük harflerin herhangi bir değişikliğe uğramadan ekrana basıldığına dikkat ediniz.

Soru : Klavyeden girilen cümlenin küçük harflerini büyük harfe, büyük harflerini de küçük harfe çeviren C++ programını yazınız?

Sıra Sizde

11.2.3. Kendisine Gönderilen Parametreyi Sorgulayıp Geriye True (sıfırdan farklı bir değer) veya False (sıfır) Değeri Döndüren Fonksiyonlar.

isalpha(x) : x bir harf ise sıfırdan farklı bir değer, x harf değilse sıfır döndürür.

isdigit(x) : x bir sayı ise sıfırdan faklı bir değer, x sayı değil ise sıfır döndürür.

isxdigit(x) : x onaltılık tabanda bir rakam ise sıfırdan farklı bir değer, değilse sıfır döndürür.

isalnum(x) : x bir rakam veya harf ise sıfırdan farklı bir değer, değilse sıfır döndürür.

isprint(x) : x yazılabilir bir karakter ise sıfırdan farklı bir değer, değil ise sıfır döndürür.

iscntrl(x) : x kontrol karakteri ise sıfırdan farklı bir değer, değil ise sıfır döndürür.

isspace(x) : x yatay tab (\t), dikey tab (\v), yeni satir (\n), satır başı (\r), form besleme (\f)

karakterlerinden biriyse sıfırdan farklı bir değer, değil ise sıfır döndürür.

islower(x) : x küçük harf ise sıfırdan farklı(true) bir değer, değilse sıfır döndürür.

isupper(x) : x büyük harf ise sıfırdan farklı (true) bir değer, değilse sıfır döndürür.

```
#include <iostream>
#include <cctype>
using namespace std;
int main() {
    char x='a', y='5',z='&',n='\t',k='d',b='D';
    cout<<endl<<"          x=  "<<x<<endl;
    cout<<"          y=  "<<y<<endl;
    cout<<"          z=  "<<z<<endl;
    cout<<"          n=  "<<"\'t"<<endl;
    cout<<"          k  "<<k<<endl;
    cout<<"          b  "<<b<<endl<<endl<<endl;
    cout<<"isalpha(x) : "<<isalpha(x)<<"        "<<"isalpha(y)    : "<<isalpha(y)<<endl;
    cout<<"isdigit(x) : "<<isdigit(x)<<"        "<<"isdigit(y)    : "<<isdigit(y)<<endl;
    cout<<"isxdigit(x): "<<isxdigit(x)<<"      "<<"isxdigit(z)   : "<<isxdigit(z)<<endl;
    cout<<"isalnum(x) : "<<isalnum(x)<<"        "<<"isalnum(z)    : "<<isalnum(z)<<endl;
    cout<<"isspace(n) : "<<isspace(n)<<"        "<<"isspace(z)    : "<<isspace(z)<<endl;
    cout<<"islower(k) : "<<islower(k)<<"        "<<"islower(b)    : "<<islower(b)<<endl;
    cout<<"isupper(k) : "<<isupper(k)<<"        "<<"isupper(b)    : "<<isupper(b)<<endl;
    return 0;
}
```

Program 11.12 Geriye true veya false değeri döndüren karakter fonksiyonlarının kullanımı

```
          x=  a
          y=  5
          z=  &
          n= \t
          k=  d
          b=  D


isalpha(x) : 2        isalpha(y)    : 0
isdigit(x) : 0        isdigit(y)    : 1
isxdigit(x): 128      isxdigit(z)   : 0
isalnum(x) : 2        isalnum(z)    : 0
isspace(n) : 8        isspace(z)    : 0
islower(k) : 2        islower(b)    : 0
isupper(k) : 0        isupper(b)    : 1

--------------------------------
Process exited after 0.6332 seconds with return value 0
Press any key to continue . . .
```

Program 11.12'nin Ekran Çıktısı

11.3. cstring Kütüphanesi ve String Fonksiyonları

String fonksiyonları, karakter dizisinin uzunluğunun bulunması, metinlerin kopyalanması, metin sonuna eklemeler yapılması gibi işlemleri kolay ve hızlı yapabilmek için kullanılır. String fonksiyonlarını kullanabilmek için C++ programı içerisine **cstring** kütüphanesinin eklenmesi gerekir.

11.3.1. strlen(ad)

strlen(ad) fonksiyonu kendisine parametre olarak gönderilen karakter dizisinin boyunu(eleman sayısını) geri döndüren fonksiyondur.

Örnek: Klavyeden girilen bir isme ait karakter dizisinin boyunu bulan ve sonucu ekrana yazdıran C++ programını yazınız?

Çözüm:

```cpp
#include <iostream>
#include <cstring>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    char ad[20];
    int elSayisi;
    cout<<endl<<" Ad Soyad giriniz :";
    gets(ad);
    elSayisi=strlen(ad);
    cout<<endl<<"Girilen Ad Soyad "<<elSayisi
        <<" karakterden oluşmaktadır"<<endl;
    return 0;
}
```

Program 11.13 strlen() fonksiyonun C++'ta kullanılması

```
 Ad Soyad giriniz :Hasan Dereli

Girilen Ad Soyad 12 karakterden oluşmaktadır

--------------------------------
Process exited after 10.39 seconds with return value 0
Press any key to continue . . .
```

Program 11.13'ün Ekran Çıktısı

Yukarıdaki programın yazılması sırasında klavyeden iki kelimeden oluşan karakter dizisi girildiğinde, boşlukların karakter sayısından çıkartılması gözardı edilmiştir. Siz programı yeniden yazarak karakter dizisinin eleman sayısını hesaplarken boşlukların çıkarılmasını ve sonucun ekrana vazılmasını sağlayınız

11.3.2. strcpy(string1,string2)

strcpy(string2,string1) fonksiyonu string1'in string2'ye kopyalanmasını sağlar.

Örnek: Klavyeden girilen karakter dizisini başka bir değişkene kopyalanmasını sağlayan ve kopyalanan karakter dizini(string2) ekrana yazdıran C++ programını yazınız?

```cpp
#include <iostream>
#include <cstring>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    const int MAX=80;
    char str1[MAX],str2[MAX];
    cout<<"Üniversite ve Fakülte Adı(str1[]) : ";
    gets(str1);
    strcpy(str2,str1);
    cout<<endl<<"strcpy(str2,str1) işleminden sonra str2 : ";
    cout<<str2<<endl;
    return 0;
}
```

Program 11.14 strcpy() fonksiyonunun kullanılmasına Örnek

```
Üniversite ve Fakülte Adı(str1[]) : istanbul universitesi AUZEF

strcpy(str2,str1) işleminden sonra str2 : istanbul universitesi AUZEF

--------------------------------
Process exited after 18.79 seconds with return value 0
Press any key to continue . . .
```

Program 11.14'ün Ekran Çıktısı

11.3.3. strncpy(string2,string1,n)

strncpy(string2,string1,n) fonksiyonu string1'in **n** karakterinin string2'ye kopyalanmasını sağlar.

Örnek: Klavyeden girilen karakter dizisinin ilk 7 karakterinin başka bir dizi değişkenine kopyalanmasını sağlayan ve kopyalanan karakter dizinin (sitring2) ekrana yazdıran C++ programını yazınız?

```cpp
#include <iostream>
#include <cstring>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    const int MAX=80,n=8;
    char str1[MAX],str2[8];
    cout<<"Üniversite ve Fakülte Adı(str1[]) : ";
    gets(str1);
    strncpy(str2,str1,n);
    cout<<endl<<"strncpy(str2,str1,n) işleminden sonra str2 : ";
    cout<<str2<<endl;
    return 0;
}
```

Program 11.15 strncpy() fonksiyonunun kullanılması

```
Üniversite ve Fakülte Adı(str1[]) : istanbul universitesi AUZEF

strncpy(str2,str1,n) işleminden sonra str2 : istanbul

--------------------------------
Process exited after 15.58 seconds with return value 0
Press any key to continue . . .
```

str1'in str2'ye ilk 8 karakteri kopyalanmıştır

Program 11.15'in ekran Çıktısı

11.3.4. strcat(string2,string1)

strcat(string2,string1) fonksiyonu string1'in string2'ye eklenmesini sağlar.
Fonksiyon **strncat(string2,string1,n)** şeklinde kullanılırsa string1'in **n** karakteri string2'ye eklenir.

Örnek: Klavyeden girilen ilk karakter dizisini, sonra girilen karakter dizisine ekleyen C++ programını yazınız?

```cpp
#include <iostream>
#include <cstring>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    const int MAX=25;
    char str1[MAX],str2[MAX];
    cout<<"Fakülte str1[])     : ";
    gets(str1);
    cout<<"Üniversite str2[]) : ";
    gets(str2);
    strcat(str2,str1);
    cout<<endl<<"strcat(str2,str1) işleminden sonra str2 : ";
    cout<<str2<<endl;
    return 0;
}
```

program 11.16 strcat() fonksiyonunun kullanılması

```
Fakülte str1[])    : acikogretim fakultesi
Üniversite str2[]) : istanbul universitesi

strcat(str2,str1) işleminden sonra str2 : istanbul universitesi acikogretim fakultesi

--------------------------------
Process exited after 25.48 seconds with return value 0
Press any key to continue . . .
```

str2

Program 11.16'nın ekran çıktısı

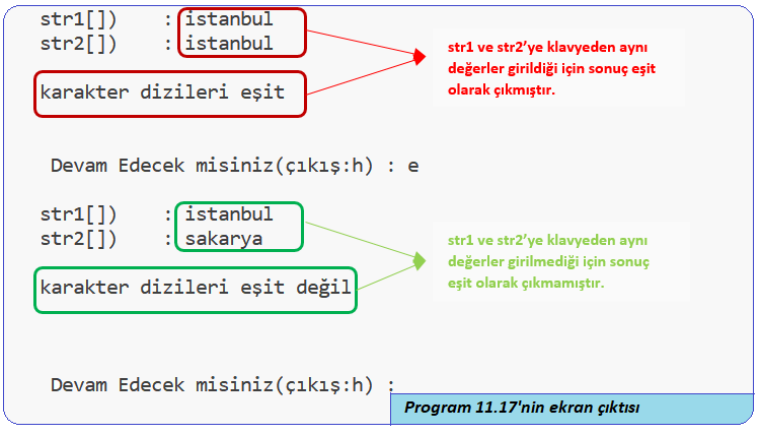
11.3.5. strcmp(string2,string1)

strcmp(string2,string1) fonksiyonu string1 ile string2'nin karşılaştırılmasını sağlar.
Fonksiyon **strncmp(string,string1,n)** şeklinde kullanılırsa sitring2 ile string1'in **n** karakteri karşılaştırılır.

Örnek: Klavyeden girilen ilk karakter dizisini karşılaştıran bir C++ programını yazınız?

```cpp
#include <iostream>
#include <cstring>
#include <locale.h>
#include <conio.h> //getche() Kullanabilmek için
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    const int MAX=25;
    char str1[MAX],str2[MAX];
    char c='e';
    do{
    cout<<"str1[])     : ";
    gets(str1);
    cout<<"str2[])     : ";
    gets(str2);
    if(!(strcmp(str2,str1)))
    cout<<endl<<"karakter dizileri eşit"<<endl;
    else
    cout<<endl<<"karakter dizileri eşit değil"<<endl<<endl;
    cout<<endl<<endl<<"hesaplamaya devam edecek misiniz(çıkış:h) : ";
    c=getche();
    cout<<endl<<endl;
    }while(c!='h');
    return 0;}
```

Program 11.17 strcmp() Fonksiyonunun Kullanılması



Program 11.17'nin ekran çıktısı

strcmp() fonksiyonu str1 ve str2 değişkenlerinin bütün elemanlarını teker teker karşılaştırır. Karşılaştırma sırasında eşitliği bozan herhangi bir karakter bulunmazsa (!(strcmp(str2,str1))) parantezinin değeri **0(sıfır)** olur ve ekrana "karakter dizileri eşit" yazılır. Karşılaştırma sırasında eşitliği bozan bir veya birden fazla karakter bulunursa parantezin değeri 1(bir) olur ve ekrana "karakter dizileri eşit değil" yazılır.

Dikkatli bir şekilde baktığınızda (!(strcmp(str2,str1))) parantezinde **!** İşareti kullanıldığını görebilirsiniz. (!(strcmp(str2,str1))) parantezde **!** İşaretinin kullanılması, **strcmp()** fonksiyonunun döndürdüğü değerin tersinin alınması ihtiyacından kaynaklanmaktadır. Çünkü **strcmp()** fonksiyonu str1 ve str2 'nin karakterlerini tek tek karşılaştırır ve karşılaştırma sonucunda iki karakter dizisi arasında herhangi bir fark yoksa geri dönen değer sıfır (0) olur. Bilindiği gibi C++ dilinde 0(false) olumsuzdur. Dolayısıyla iki karakter dizisinin eşitliği durumunda geri dönen değer olan **sıfır**ın(**0**) tersinin alınması gerekmektedir. Karakter dizilerinin eşit olmaması durumunda da yine geri dönen değirin sıfır yapılması için aynı şekilde parantezdeki **!** İşaretine ihtiyaç vardır.

11.3.6. strchr(string, 'a')

strchr(string,'a') fonksiyonu string içerisinde bir karakter arar.

Örnek: Klavyeden girilen bir karakter dizi içerisinde yine klavyeden girilen bir karakteri arayan ve sonucu ekrana yazdıran C++ programını yazınız?

```cpp
#include <iostream>
#include <cstring>
#include <locale.h>
using namespace std;
int main()
{
    setlocale(LC_ALL,"Turkish");
    char str[20];
    char ch,ch2;
    cout<<" str[] : ";
    gets(str);
    cout<<" Dizi içerisinde Aranan ilk Karakter : ";cin>>ch;
    cout<<" Dizi içerisinde Aranan ilk Karakter : ";cin>>ch2;
    if (strchr(str, ch) != NULL)
        cout << ch << " "
             << " harfi string içerisinde var" << endl;
    else
        cout << ch << " "
             << " harfi string içerisinde yok" << endl;
    if (strchr(str, ch2) != NULL)
        cout << ch2 << " "
             << " harfi string içerisinde var" << endl;
    else
        cout << ch2 << " "
             << "  harfi string içerisinde yok" << endl;
    return 0; }
```

Program 11.8 strchr() Fonksiyonunun Kullanımı

```
str[] : sakarya

Dizi içerisinde Aranan ilk Karakter : a
Dizi içerisinde Aranan ilk Karakter : d

a  harfi string içerisinde var
d   harfi string içerisinde yok

--------------------------------
Process exited after 11.06 seconds with return value 0
Press any key to continue . . .
```

Program 11.18'in Ekran Çıktısı

11.3.6. strrev(string)

strrev(string) fonksiyonu karakter dizisini tersine çevirir.

Örnek: Klavyeden girilen bir karakter dizini tersten yazdıran C++ programını yazınız?

```cpp
#include <iostream>
#include <cstring>
#include <locale.h>
#include <conio.h> //getche() Kullanabilmek için
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    const int MAX=25;
    char str[MAX];
    char c='e';
    do{
    cout<<"str[]   : ";
    gets(str);
    cout<<endl<<"Ters Çevrilmiş Karakter Dizisi : "<<strrev(str)<<endl;
    cout<<endl<<endl<<" Devam Edecek misiniz(çıkış:h) : ";
    c=getche();
    cout<<endl<<endl;
    }while(c!='h');
    return 0;}
```

Program 11.19 strrev() fonksiyonunun kullanılması

```
str[]   : istanbul
Ters Çevrilmiş Karakter Dizisi : lubnatsi   ters çevrildi

 Devam Edecek misiniz(çıkış:h) : e

str[]   : sakarya
Ters Çevrilmiş Karakter Dizisi : ayrakas   ters çevrildi

 Devam Edecek misiniz(çıkış:h) :
```

Program 11.19'un Ekran Çıktısı

Bölüm Özeti

Fonksiyonlar yapısal programlamanın en önemli araçlarındandır. Yedinci bölümde Fonksiyonlar başlığı altında problemleri parçalara ayırarak bilgisayar programlarını fonksiyonlar halinde yazabileceğimizi görmüştük. Bu yaklaşım sayesinde okunabilirliği çok daha yüksek, daha az karmaşık program yazma şansımız olmuştu. Programları fonksiyonlar halinde yazmanın getirdiği diğer bir avantaj da bu yaklaşımın bilgisayarın fiziksel kaynaklarını daha verimli kullanılmasına yol açmış olmasıydı.

Bu bölümde C++ derleyicilerinin programcılara sunduğu hazır programları ele aldık. Hazır fonksiyonların derleyicilerde, kütüphaneler altında yer aldığını, Bu kütüphaneler, ihtiyaç duyulması halinde programa dâhil edilerek bu hazır fonksiyonları çağırıp kullanabileceğimizi örneklerle açıkladık.

Bu bölümde hazır matematik fonksiyonlarını kullanabilmek için cmath kütüphanesini, karakter fonksiyonlarını kullanabilmek için cctype kütüphanesini, string tipinde veriler üzerinde işlem yapabilmek için cstring kütüphanesini ele aldık. Her bir kütüphane için kütüphanenin desteklediği önemli hazır fonksiyonlara örnekler vererek bu hazır fonksiyonların nasıl kullanılacağını açıkladık.

ÜNİTE SORULARI

1. C++'ta matematiksel fonksiyonları kullanabilmek için aşağıdaki kütüphanelerden hangisi eklenmelidir?

- locale.h
- conio.h
- iostream
- cmath
- cstring

2. Kendisine parametre olarak gönderilen sayının karekökünü döndüren fonksiyon aşağıdakilerden hangisidir?

- sqrt()
- max()
- floor()
- pow()
- ceil()

3. Kendisine parametre olarak gönderilen sayıyı alıp, kendisinden küçük ilk tamsayıyı geri döndüren fonksiyon aşağıdakilerden hangisidir?

- ceil()
- floor()
- sqrt()
- pow()
- fabs()

4. Aşağıdaki fonksiyonlardan hangisi kendisine parametre olarak gönderilen sayının mutlak değerini alır?

- ceil()
- floor()
- sqrt()
- pow()
- fabs()

5. Aşağıdaki fonksiyonlardan hangisi parametre olarak aldığı karakteri büyük harfe çevirir?

- toupper()
- tolower()
- floor()
- ceil()
- isdigit()

6. Aşağıdaki fonksiyonlardan hangisi aldığı parametre harf ise sıfırdan farklı bir değer, aldığı parametre harf değilse sıfır değeri döndürür?

- isdigit()
- isalnum()
- isprint()
- isalpha()
- isspace()

7. Aşağıdaki fonksiyonlardan hangisi aldığı parametre yazılabilir bir karakter ise sıfırdan farklı bir değer, aldığı parametre yazılabilir bir karakter değil ise sıfır döndürür?

- isdigit()
- isalnum()
- isprint()
- isalpha()
- isspace()

8. Aşağıdaki fonksiyonlardan hangisi parametre olarak aldığı karakteri küçük harfe çevirir?

- toupper()
- tolower()
- floor()
- ceil()
- isdigit()

9. C++'ta string fonksiyonları kullanabilmek için aşağıdaki kütüphanelerden hangisi programa eklenmelidir?

- locale.h
- conio.h
- iostream
- cmath
- cstring

10. Aşağıdaki fonksiyonlardan hangisi kendisine parametre olarak gönderilen iki string'ten, ikinci sırada gönderilen string'in birinci sırada gönderilen string'e eklenmesini sağlar?

- strcpy()
- strncpy()
- strcat()
- strcmp()
- strchr()
    -
    - CEVAP ANAHTARI1. d2. a3. b4. e5. a6. d7. c8. b9. e10. c

12. SINIF VE NESNELER

Giriş

Bu bölümde sınıf ve nesne kavramlarını ele alacağız. Önce basit sınıflardan başlayarak giderek daha karmaşık sınıflar yazacak ve bu sınıflardan nesneler tanımlayacağız. Bölüm tamamlandığında nesne yönelimli programlama (***Object Oriented Programming***) kavramları ve teknikleri ile ilgili temel bilgileri öğrenmiş olacağız.

Nesne yönelimli programlamanın esası, belirli özelliklere ve yöntemlere sahip bir nesne oluşturmaktır. Nesneye yönelik programlamada (***Object Oriented Programming***) programı tasarlarken, tüm dünyayı nesne şeklinde görmeye çalışmalıyız. Örneğin araba, renk, kapı sayısı ve benzerleri gibi belirli özelliklere sahip bir nesnedir. Ayrıca, hızlanma, fren vb. gibi belirli işlevlere sahiptir. Nesneye yönelik programlamanın temelini oluşturan bazı kavramlar vardır. Bu kavramlar sınıf, nesne, soyutlama, Paketleme, kalıtım ve çok biçimlilik şeklinde sıralanabilir. ***Bu kavramlardan soyutlama, Paketleme, kalıtım ve çok biçimlilik nesneye yönelik programlama teorisinin dört temel özelliği olarak anılır.*** Konunun başında bu kavramları açıklamanın konun anlaşılmasına katkı sağlayacağı düşüncesiyle önce bu kavramların tanımlarını yapalım.

Nesne (Object)

Nesne, nesne yönelimli programlamanın en temel birimidir ve ait olduğu sınıfın bir örneğidir. Bu yaklaşımda hem veriler hem de veriler üzerinde çalışan işlevler nesne olarak adlandırılan bir birim olarak paketlenir.

Sınıf (Class)

Sınıf tanımlandığında, bir nesne için bir plan tanımlanır. Sınıfın tanımını yapmak aslında herhangi bir veri tanımlamaz, ancak sınıf adının ne anlama geldiğini, yani sınıftaki bir nesnenin ne içereceğini ve böyle bir nesne üzerinde hangi işlemlerin gerçekleştirilebileceğini tanımlar.

Soyutlama (Abstraction)

Veri soyutlama, dış dünyaya sadece gerekli bilgileri sağlamak ve arka plan ayrıntılarını gizlemek, yani ayrıntıları sunmadan programdaki gerekli bilgileri temsil etmek anlamına geldiğini de söyleyebiliriz.

Örneğin, bir veritabanı sistemi, verilerin nasıl saklandığına, nasıl oluşturulduğuna ve nasıl korunduğuna ilişkin belirli ayrıntıları gizler. Benzer şekilde, C ++ sınıfları bu yöntemler ve veriler hakkında iç ayrıntı vermeden dış dünyaya farklı yöntemler sunar.

Paketleme (Encapsulation)

Paketleme, verileri ve bu veriler (değişkenler) üzerinde çalışan işlevleri (fonksiyonlar) aynı isim altında tanımlamak, paketlemek anlamına gelir. Prosedürel dillerle çalışırken, hangi işlevlerin hangi değişkenler üzerinde çalıştığı her zaman açık değildir, ancak nesne yönelimli programlama size verileri ve ilgili işlevleri aynı nesneye yerleştirmek için önemli bir çerçeve sağlar.

Nesneye yönelik programlama nesnelerin özelliklerini (değişkenleri) ve işlevlerini (fonksiyonlarını – metotlarını) diğer nesnelerden saklanmasını sağlar. Bu, özellik ve işlevleri public (açık), private (özel) ve protected (korumalı) şeklinde tanımlayarak gerçekleştirilir.

- Public (genel) : Özelliklere ve işlevlere her nesneden erişilebilir.
- Private (özel): Özelliklere ve işlevlere yalnızca işlevin tanımlandığı sınıfa ait nesnelerle erişilebilir.
- Protected (korumalı): Özelliklere ve işlevlere yalnızca metodun tanımlandığı sınıfa ait nesnelerle erişilebilir ve bu sınıftan türetilmiş alt nesnelerinden erişilebilir.

Kalıtım (Inheritance)

Nesneye yönelik programlamanın en kullanışlı yönlerinden biri de kodların yeniden kullanılabilir olmasıdır. Adından da anlaşılacağı gibi kalıtım, temel sınıf olarak adlandırılan mevcut bir sınıftan yeni bir sınıf oluşturma işlemidir. Bu, nesne yönelimli programlama için çok önemli bir kavramdır, çünkü bu özellik kod boyutunu azaltmaya, kodun okunabilir olmasına önemli katkılar sağlar.

Çok Biçimlilik (Polymorphism)

Nesneye yönelik programlamada, oluşturulan nesnelerin gerektiğinde başka bir nesne gibi davranabilme özelliğine çok biçimlilik denmektedir. Çok biçimlilik kavramı için nesnelerin aynı mesaja farklı cevap vereme özelliğidir de diyebiliriz.

Nesneye yönelik programlama yaklaşımı günümüzde düzenli kod yazmak, kod bloklarına daha çabuk erişim sağlamak, kod tekrarını önlemek, güvenlik düzeyi daha yüksek programlar yazılmasını sağlamak ve vb. nedenlerle, özellikle büyük ölçekli projelerde kullanmaktan kaçmayacağımız bir tekniktir. Başlangıçta sadece C++ gibi, Java gibi programlama dilleri tarafından desteklenen nesneye yönelik programlama yaklaşımı, günümüzde hemen bütün programlama dilleri tarafından desteklenmektedir. Bu diller, yukarıda sayılan özellikleri ile programcılara daha sonradan bakımını daha düşük maliyetlerle yapılabilecekleri (geliştirilebilir), daha az zamanda tamamlayabilecekleri, güvenlik düzeyi yüksek, karmaşıklığı daha az ve bilgisayarın fiziksel kaynaklarını daha verimli kullanan projeler yapmalarını sağlayacak programlama ortamları sunmaktadır. Nesneye yönelik programlama yaklaşımı diğer bir yönüyle kendisinden önce kullanılan yapısal programlama yaklaşımını ve başlangıçtan günümüze kullanılan daha önceki programlama yaklaşımlarını da kapsayan bir yaklaşımdır.

Nesneye Yönelik programlama yaklaşımından önce yazılım projelerinde yapısal programlama yaklaşımı kullanılmaktaydı (Bu bölüme kadar biz de örneklerimizi yapısal programlama tekniklerini kullanarak oluşturduk). C,

Pascal ve benzeri prosedürel dillerin desteklediği yapısal programlama yaklaşımında, bir program dosyası aslında bir komut listesi olarak hazırlanmaktaydı. Zaman içerisinde bu listeler, elektronik ortama taşınacak işlerin hacminin artmasıyla çok fazla uzadı ve sonradan ortaya çıkan bakım maliyetleri katlanılamaz boyutlara ulaştı. Daha sonra yine aynı yapısal programlama yaklaşımı içerisinde kalınarak program içerisindeki tekrarlardan kurtulmak, programı bilgisayar ortamında koştururken bilgisayarın fiziksel bileşenlerini daha verimli kullanabilmek gibi nedenlerle programlar fonksiyonlar halinde yazılmaya başlandı. Ancak, yine de eklenen bu önemli yeni özelliğine rağmen yapısal programlama yaklaşımı gerçek hayat problemlerini temsil etmekte çok başarılı olamadı ve de en önemlisi yapısal programlama yaklaşımı ile geliştirilen yazılım projelerinde programcılar güvenli projeler oluşturduklarından hiçbir zaman emin olamadı. Yukarıda sayılan bu ve benzeri eksikliklerine rağmen yapısal programlama yaklaşımı, kendi döneminde önemli yazılım projelerinde kullanılarak programlama dilleri tarihi içerisinde yerini almıştır.

Nesneye yönelik programlama yaklaşımında sınıflar, aslında yapısal programlamadan miras aldığı fonksiyonlar (Nesneye yönelik programlamada **metot** olarak anılacaktır.) ile verileri (özellikleri) bir çatı altında toplar. Daha doğru bir ifadeyle sınıflar, metotlar ile özellikleri bir çatı altında toplayan tanımlamalardır.

C++ ve nesneye yönelik benzer programlama dillerini kullanan programcılar, sınıf adı verilen kendilerine ait kullanıcı tanımlı tiplerini oluşturur. Programcı tarafından oluşturulan sınıflar veriyi yöneten metotların kümesi ve verilerden oluşur. Daha sonra program içerisinde oluşturulan bu sınıfa ait nesneler oluşturulur.

Bu bölümde yukarıda da belirtildiği gibi C++ programlama dilini kullanarak sınıfların ve bu sınıflara ait nesnelerin tanımlanmasını program içerisinde bu yaklaşımın nasıl kullanılacağını öğreneceksiniz.

12.1. Sınıf ve Nesnelerin Tanımlanması ve Kullanılması

12.1.1. Sınıf ve Nesnelerin Tanımlanması

Aşağıda kaynak kodu verilen C++ programında, bir sınıf ve bir de nesne tanımlanmıştır. Bu program basit olmakla birlikte, C++'ta sınıflar ve nesneler hakkında birçok bilgi içermektedir. Programda ***Kare*** isimli bir sınıf tanımlanmış ve Kare isimli bu sınıfta tek bir veri öğesi yer almıştır. Bundan başka sınıf içerisinde üç tane fonksiyon (metot) yazılmıştır. Fonksiyonlardan ilki, veri öğesine bir değer atamak için, ikinci fonksiyon karenin çevresini yazdırmak için ve üçüncü fonksiyon da karenin alanı yazdırmak için yazılmıştır.

İpucu: Veri ve fonksiyonları tek bir bütün halinde ele almak nesne yönelimli programlamanın en önemli özelliğidir.

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

class Kare
{
    private:
        int karekenar;

    public:
        void KenarUzunlukata(int sayi)
            { karekenar = sayi; }

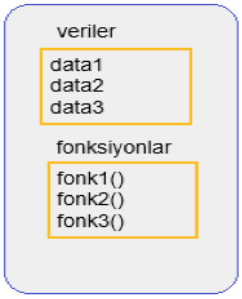
        void cevreyaz()
            { cout << 4*karekenar << endl; }
        void alanyaz()
        {
            cout<<karekenar*karekenar<<endl;}
    };

int main()
{
    Kare kare1;
    kare1.KenarUzunlukata(3);
    kare1.cevreyaz();
    kare1.alanyaz();

    return 0;
}
```

Program 12.1 Karenin Çevresi ve Alanı

Veri ve fonksiyonları tek bir bütün haline getirmek nesne yönelimli programlamanın temel yaklaşımıdır.



Veri ve fonksiyonların sınıf içerisinde gösterilmesi

Yukarıdaki programda, ilk bölümde ***Kare*** isimli bir sınıf tanımlanmış daha sonra ***main()*** bölümünde Kare ***sınıfına*** ait ***kare1*** isimli bir örnek(nesne) tanımlanmıştır.

Program 12.1'in kaynak kodundan alınan sınıf tanımlama bölümü, aşağıda gösterilmiştir. C++ programlarında sınıf tanımı **class** bildirimi ile başlar ve bildirimden sonra sınıfın adı **(Aşağıdaki örnekte bu ad, *Kare* olarak belirlenmiştir.)** yazılır. Daha sonra sınıf bloğunda yapılacak tanımlamalar, aşağıda da gösterildiği gibi, küme parantezleri arasına yazılır. Kapanan küme parantezi sonunda noktalı virgül konulur.

```
class Kare
{
    private:
        int karekenar;

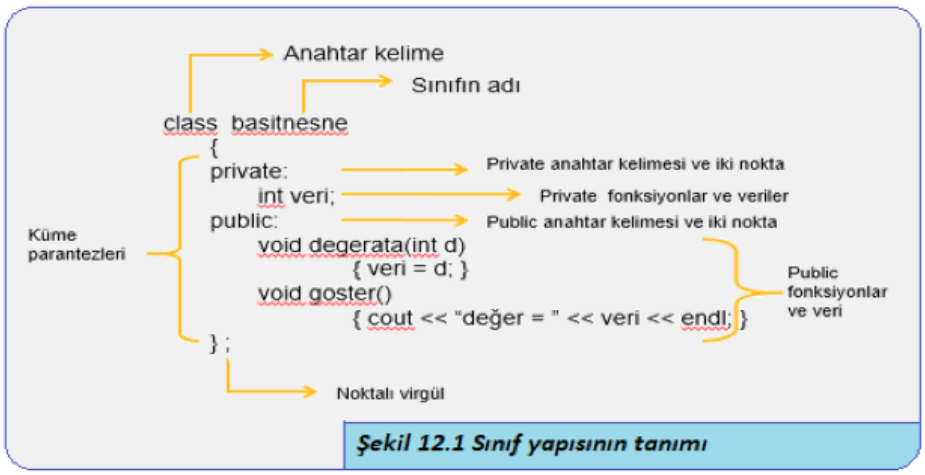
    public:
        void KenarUzunlukata(int sayi)
            { karekenar = sayi; }

        void cevreyaz()
            { cout << 4*karekenar << endl; }
        void alanyaz()
        {
            cout<<karekenar*karekenar<<endl;}
    };
```

Program 12.1'den Alınan Sınıf Tanımı Bölümü

İpucu: Sınıf içerisinde tanımlanan verilere ve fonksiyonlara o sınıfın üyeleri(elemanları) denilir.

Program 12.1'e ait sınıf tanımından da görülebileceği gibi, nesneye yönelik programların gövdesinde ilki ***private***(özel), ikincisi ***public***(genel) olmak üzere, daha önceki bölümlerde kullanmadığımız iki yeni bildirim kullanılmıştır. Nesneye yönelik programlamanın en temel özelliklerinde birisi verinin saklanmasıdır. Nesneye yönelik programlamada verinin saklanması, veriyi sınıf içerisinde ***private*** olarak tanımlayarak gerçekleştirilir. Bu şekilde tanımlanan verilere, sadece o sınıfın fonksiyonları erişebilir. Dolayısı ile veriler, diğer sınıflara ait fonksiyonlar tarafından erişilip değiştirilemez ve veriler güvenlik altına alınmış olur. Verilerin güvenlik altına alınması, programın içerisinde o verilere ulaşması gerekmeyen bölümlerden ulaşılmasını önlemek anlamına gelmektedir.



Şekil 12.1 Sınıf yapısının tanımı

Veriler ve fonksiyonlar sınıf gövdesinde private olarak tanımlanabildiği gibi public olarak da tanımlanabilir. Bu şekilde tanımlanan veri ve fonksiyonlara programın her yerinden ulaşılabilir.

Program 12.1'de ***int*** tipinde tek bir veri öğesi tanımlanmıştır. Ancak, bir sınıfta istenildiği kadar veri öğesi tanımlanabilir. Sınıf içerisinde tanımlanan veri öğelerine ***üye veriler*** veya ***veri üyeleri*** denilir.

İpucu: Sınıf gövdesinde private bildirimi içerisinde verilerin tanımı yapılabildiği gibi, ihtiyaç duyulması halinde bu bildirim altında fonksiyonların tanımlamaları da yapılabilir.

İpucu: *private* veri veya fonksiyon tanımlanması gerekiyorsa *private* yazmaya gerek yoktur. Sınıf adından sonra {} içine direk veri ve fonksiyon yazıldığında bu tanımlamalar otomatik olarak private kabul edilir.

Bir sınıfın içerisinde yer alan fonksiyonlara o sınıfın ***üye fonksiyonları*** denilir. Bazen de fonksiyonlar **metot** olarak ta isimlendirilir. Program 4.1'de Kare sınıfının üç üye fonksiyonu(metot) vardır. Bu üye fonksiyonlar aşağıda gösterilmiştir.

```
public:
    void KenarUzunlukata(int sayi)
        { karekenar = sayi; }

    void cevreyaz()
        { cout << 4*karekenar << endl; }
    void alanyaz()
    {
        cout<<karekenar*karekenar<<endl;}
```

Program 4.1'deki üye fonksiyonlar

Tanımlamaya yakından bakıldığında, üye fonksiyonların(metotlar) **public** anahtar kelimesi altında yazıldığı görülmektedir. Genellikle bir sınıf tanımlanırken veriler **private** altında, üye fonksiyonlar da public altında yazılır. Fakat bu bir kural değildir. Bazen üye veriler **public** altında, üye fonksiyonlar da **private** altında tanımlanabilir.

İpucu: Üye veriler, yanlışlıkla değiştirilmemeleri için, güvenlik nedeniyle private altında tanımlanır. Üye fonksiyonlar da, üye veriler üzerinde işlem yaptıklarından, sınıf dışından erişilebilmeleri için public olarak tanımlanırlar.

12.1.2. Tanımlanan Sınıfın Kullanılması

Bir sınıf tanımlamak yapı tanımlamak gibidir. Tanımlanan sınıf, kendisine ait bir nesne oluşturulana kadar sadece oluşturulacak nesnenin neye benzeyeceğini tanımlar. Karenin çevresini ve alanını hesaplayan programda main() fonksiyonu içerisindeki ilk ifadede Kare sınıfına ait bir nesne tanımlanmıştır:

Kare kare1;

İfadesiyle programda bir nesne tanımlanmıştır. Bu işleme örnek oluşturmak denir. Nesnelere bazen örnek değişken(***instance variable***) adı da verilebilir.

12.1.3. Üye Fonksiyonların Çağırılması

Üye fonksiyonlar(metotlar) bir nesne üzerinde işlem yapmak için çağırılır. Başka bir şekilde ifade etmek gerekirse, sınıfın üye fonksiyonlarına sadece o sınıfa ait nesneler tarafından erişilir. Sınıfa ait nesne ile yine o sınıfa ait bir üye fonksiyonu çağırmak için, nesne ile üye fonksiyonu ***nokta operatörü*** ile ilişkilendirmek gerekir. Nokta operatörüne ***sınıf üyesi erişim operatörü*** de denir. Ele aldığımız örnekte, üç üye fonksiyon da tanımlanan nesne tarafından aşağıdaki şekilde çağırıldığını görüyoruz:

kare1.KenarUzunlukata(3);

Programda çalışma sırası yukarıdaki ifadeye geldiğinde, *kare1* nesnesinin *KenarUzunlukata()* üye fonksiyonu çalışır. Bu fonksiyon kenar1 değişkeninin değerini üç(3) yapar.

İkinci çağrıda;

kare1.çevreyaz();
kare1 nesnesinin çevresi ekrana yazılır.

Üçüncü çağrıda;

kare1.alanyaz();
kare1 nesnesinin alanı ekrana yazılır.

İpucu: Nesne yönelimli dillerde üye fonksiyonları çağırma işlemine ***mesaj*** denir. Bu nedenle ***kare1.alanyaz();*** *çağrısı* ***kare1****'e, verisini göndermesini bildiren bir* ***mesaj*** gönderimi *olarak düşünülebilir.*

12.1.4. Program 12.2

Bir Kare sınıfı tanımlayınız. Kare sınıfında, sınıf elemanlarının tamamı public olarak tanımlanmış olsun. Sınıfa ait bir nesne oluşturunuz ve klavyeden nesneye kenar uzunluk ölçüsünün girişinin yapılmasını sağlayınız. Program çalıştırıldığında karenin çevresini ve alanını ekrana yazdırınız?

Çözüm:

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

class kare
{
    public:
        int kareKenar;
    int cevre(){

    return 4*kareKenar;}

    int alan(){
    return kareKenar*kareKenar;}

}x;
int main() {

    cin>>x.kareKenar;
    cout<<"Çevre    :"<<x.cevre()<<endl;
    cout<<"Alan     :"<<x.alan()<<endl;
    return 0;
}
```

Program 12.2 Üye fonksiyonların ve üye verilerin public olduğu C++ programı

Program 12.2'de üye veriler ve üye fonksiyonlar ***public*** olarak tanımlanmıştır. Bundan dolayı sınıfın bütün elemanlarına dışarıdan erişilebilmektedir. Biz de üye veriye(***KareKenar***) public olarak tanımlandığı için, tanımlanan nesne üzerinden, dışarıdan(***main() içerisinden***) doğrudan erişilebildik. Bu bir kolaylık gibi görünse de verinin güvenliğini tehdit edici bir yaklaşımdır.

Bundan başka, Program 12.2'de nesnenin tanımlandığı yere de dikkat ediniz. Program 12.1'de nesne ***main()*** fonksiyonunda tanımlanmıştır. Program 12.2'de ise bu tanımlama, sınıf gövdesinin bittiği noktada yapılmıştır. Bu iki yaklaşım da doğrudur. C++'ta Nesnelerin tanımı, sınıfın tanımlandığı yerde veya ***main()*** fonksiyonu içerisinde yapılabilir.

12.1.5. Program 12.3

Bilgisayara stok kodu, ürün adı, ürünün birim fiyatı girişinin klavyeden yapıldığı ve girilen verilerin ekranda gösterildiği programı nesneye yönelik programlama yaklaşımı ile yazınız.

Çözüm:

```cpp
#include <iostream>
#include <string>
using namespace std;
class Stok  //sinifi tanimla
    {
    private:
        int stokKodu;   //Stok kodu
        string urunAdi;     //Ürün adı
        float brimFiyat;    //Birim fiyatı
    public:
        void stokAta(int stk, string uad, float bf) //deger atama fonsiyonu
        {
        stokKodu = stk;
        urunAdi = uad;
        brimFiyat = bf;
        }
        void goster()   //veriyi görüntüleyen fonksiyon
        {
        cout << "Stok Kodu   : " << stokKodu<<endl;
        cout << "Urun Adi    : " << urunAdi<<endl;
        cout << "Birim Fiyat : " << brimFiyat << endl;
        }
    };
int main()
    {
    Stok  stok1; string ua; int sn; float bf;   //Stok sinifina ait nesneyi tanimla
    cout<<"stok kodu   : ";cin>>sn;
    cout<<"Urun adi    : ";cin>>ua;
    cout<<"Birim Fiyat : ";cin>>bf;

    stok1.stokAta(sn, ua, bf);  //üye foksiyonu çagir
    cout<<"----------------------"<<endl;
    stok1.goster();          //üye foksiyonu çagir
    return 0;
    }
```

Program 12.3 Stok Programı

12.1.6. Program 12.4

Öğrencilerin öğrenci numarası, ad, soy ad ve yaşlarının bilgisayara girişinin klavyeden yapıldığı ve girilen verilerin ekranda görüntülendiği C++ programını yazınız?

Çözüm:

```cpp
#include <iostream>
#include<string>
#include <locale.h>
using namespace std;
class ogrenci
{
string ogrNo,ogrAd,ogrSoyad;
int ogrYas;
public:
   void ogr(string oNo, string oAd, string oSad,int oYas)
    {
     ogrNo=oNo;
     ogrAd=oAd;
     ogrSoyad=oSad;
     ogrYas=oYas;
     }
     void goruntule()
     {
         cout<<"Öğrenci No     : "<<ogrNo<<endl;
         cout<<"Öğrenci Ad     : "<<ogrAd<<endl;
         cout<<"Öğrenci Soyad  : "<<ogrNo<<endl;
         cout<<"ogrYas         : "<<ogrYas<<endl;
       }
};

int main() {
     setlocale(LC_ALL,"Turkish");
     ogrenci ogr1,ogr2;
     ogr1.ogr("00001","Ahmet","Kara",20);
     ogr2.ogr("00002","Ayşe","Ak",19);
     ogr1.goruntule();
     ogr2.goruntule();
     return 0;
}
```

Program 12.4 Sınıf ve Nesne Örneği_ Öğrenci Uygulaması

Sıra Sizde: Aşağıda, Program 12.4 için bir çözüm sunulmuştur. Fakat üye verilere değerleri, main() fonksiyonu içerisinde atama yoluyla gerçekleştirilmiştir. Programa ait soru metninde veri girişlerinin klavyeden yapılması istenmiştir. Program üzerinde veri girişlerinin klavyeden yapılabilmesi için gerekli değişiklikleri yapınız.

12.2. Kurucu Fonksiyonlar

C++'da nesnenin otomatik olarak ilk kullanıma hazırlama işlemi kurucu(constructor) fonksiyon tarafından gerçekleştirilir. Ne zaman bir nesne oluşturulursa kurucu fonksiyon otomatik olarak çalıştırılır.

Bir program içerisinde kullanılmak üzere bir sayaç sınıfı oluşturulduğunu düşünelim. Her zaman sayaçların ilk oluşturulduklarında sıfır değerini almasını isteriz.

Bunu sağlamak için sayaca ilk değeri atayan bir üye fonksiyon yazabiliriz ve sayaç nesnesi tanımlandığında bu üye fonksiyonu çağırarak sıfır değeri atayabiliriz. Ancak, her nesne oluşturulduğunda bu üye fonksiyonun da bir defa çalıştırılması gerekir. Bu işlem hataya açık bir durumdur ve bu nedenle sınıf içerisinde başlangıç değeri verme işlemi yapılmaz.

Ancak, kurucu fonksiyonları kullanarak üye değişkenlere ilk değer atanabilir. Her nesne oluşturulduğunda, otomatik olarak kurucu fonksiyonla kendisini ilk kullanıma hazırlaması daha uygun ve güvenilirdir.

Kurucu (Yapıcı) Fonksiyonların Özellikleri:

a. Kurucu fonksiyonlar üyesi olduğu sınıfla aynı isme sahip olmalıdır. Bu özellik derleyicinin fonksiyonun kurucu olduğunu anlamasına neden olur.
b. Sınıfa ait nesne oluşturulduğunda kurucu fonksiyon otomatik olarak çağırılır.
c. Bir sınıf içerisinde birden fazla kurucu fonksiyon tanımlanabilir. Bir sınıf içerisinde birden fazla kurucu fonksiyon tanımlanmışsa bu fonksiyonların isimleri aynı olmalıdır.
d. Kurucu fonksiyonlar değer döndürmezler.
e. Kurucu fonksiyonlar nesne oluşturulduğu zaman sadece bir kere çalıştırılırlar.
f. Eğer bir sınıfa ait kurucu fonksiyon tanımlanmaz ise derleyici tarafında otomatik oluşturulur.

12.2.1. Program 12.5

Bir C++ uygulaması geliştiriniz. Uygulamada üye değerlere kurucu fonksiyon tarafından ilk değer ataması yapılsın.

Çözüm:

```
#include <iostream>
using namespace std;
class ucret
{
    int calismaSaati;
    int saatUcreti;
    int brut;
    public:
        ucret():calismaSaati(0),saatUcreti(0),brut(0)
        {}
        void goster( )
        {
            cout<<calismaSaati<<endl;
            cout<<saatUcreti<<endl;
            cout<<brut<<endl;
        }
};

int main() {

    ucret prsl;
    prsl.goster();
    return 0;
}
```

Program 12.5 Kurucu Fonksiyon örneği

```
Calisma Sati : 0
Saat Ucreti  : 0
Brut Ucret   : 0
```

Program 12.5'in Ekran Çıktısı

İlk değer atama işlemi üye fonksiyon bildiriminden sonra, fakat fonksiyon gövdesinden önce yapılır. Atanacak değerden önce iki nokta gelir. Değer üye verinin peşinden parantez içerisinde yazılır. Eğer birden fazla üyeye değer atanacaksa, bu değerler virgül ile ayrılır. Program 12.5'teki örnek kurucu fonksiyon tarafından birden fazla üyeye ilk değer ataması yapılmasına örnek olarak verilmiştir.

Sıra Sizde: Program 12.5'te aşağıdaki değişiklikleri yapıp programı tekrar çalıştırınız. Yukarıdaki programın ekran çıktıları ve aşağıdaki değişiklikleri yaptıktan sonra çalıştıracağınız programdan elde edilen ekran çıktılarını karşılaştırınız.

1. Kurucu fonksiyonda üye değerlere sıfır yerine 10, 20, 30 gibi farklı değerler atayınız ve programı tekrar çalıştırınız.
2. Kurucu fonksiyonu kaldırınız ve programı tekrar çalıştırınız?

12.2.2. Program 12.6

Bir C++ uygulaması geliştiriniz. Geliştirdiğiniz bu programda kurucu fonksiyon ile programın üye değişkeni sıfırlansın.

Çözüm:

```
#include <iostream>
using namespace std;
class sayac
    {
private:
    unsigned  int  sayici;
public:
 sayac():sayici(0)
        {    }
   void artir_sayici()
        {sayici++;}

   int  al_sayici()
        {return sayici;}
    };
int main()
    {
sayac  s1,s2;
cout<<"\n s1 degeri "<< s1.al_sayici();
cout<<"\n s2 degeri "<<s2.al_sayici();

s1.artir_sayici();
s2.artir_sayici();
s2.artir_sayici();

cout<<"\n s1 degeri "<< s1.al_sayici();
cout<<"\n s2 degeri "<< s2.al_sayici();
cout<<endl;
system("pause");
return 0;
    }
```

Program 12.6 Kurucu Fonksiyonla Üye değ. Sıfırlanması

```
s1 degeri 0
s2 degeri 0
s1 degeri 1
s2 degeri 2
```

Program 12.6 'nın Ekran çıktısı

12.3. Yıkıcı Fonksiyonlar

Yıkıcı Fonksiyonlar nesneler yok edileceği zaman çalışan fonksiyonlardır. Eğer programcı tarafından tanımlanmazsa otomatik olarak tanımlanırlar. Yıkıcı fonksiyonların da isimleri kurucu fonksiyonlarda olduğu gibi sınıf ismi ile aynı

olmalıdır. Yıkıcı fonksiyonların isimlerinin başında, kurucu fonksiyonlardan farklı olarak ~(Tilda) işareti bulunur. Bir sınıf içerisinde sadece bir yıkıcı fonksiyon olabilir. Yıkıcı fonksiyonlar parametre almazlar ve değer döndürmezler.

Yıkıcı fonksiyonlar yaygın olarak, bir nesne için kurucu fonksiyonların ayırdığı bellek alanını iade etmek için kullanılır.

12.3.1 Program 12.7

İçerisinde yıkıcı fonksiyon bulunan bir C++ programı yazınız?

Çözüm:

```
using namespace std;
class yoket
    {
        int sayi;
        public:
                yoket();
                ~yoket();
                void goruntule();

    }sayil;

    yoket::yoket(){
        cout<<"kurucu fonksiyon calisti"<<endl;
        sayi=5;
    }
    yoket::~yoket(){
        cout<<"Yıkıcı fonksiyon calisti"<<endl;
            }
    void yoket::goruntule(){
        cout<<"sayi degeri : "<<sayi<<endl;
    }
int main() {
    sayil.goruntule();
    return 0;
}
```

Program 12.7 İçerisinde yıkıcı fonksiyon bulunan C++ programı

```
kurucu fonksiyon calisti
sayi degeri : 5
Yikici fonksiyon calisti
```

Program 12.7'nin Ekran Çıktısı

12.4. Bir Nesneyi Başka Bir Nesneye Atamak

Bazı problemlerin çözümünde kullanılmak üzere, bir sınıftan birden fazla nesne oluşturabiliriz ve bu nesnelerden birinin değerlerini, diğer başka nesnelere atamak isteyebiliriz. C++ ta bu işlem atama operatörü kullanılarak yapılmaktadır.

12.4.1. Problem 12.8

Nesneden nesneye atama işlemin Problem 12.3'ün sınıf yapısını değiştirmeden *main()* fonksiyonu içerisinde yeni bir nesne oluşturup(*stok2*), daha önceden oluşturulan stok1'in değerleri stok2'ye aktarılmıştır. Programın son kısmında da göster fonksiyonu yardımıyla her iki nesne de görüntülenmiş ve aynı değerlere sahip olduğu görülmüştür.

```
class Stok  //sinifi tanimla
    {
    private:
        int stokKodu;   //Stok kodu
        string urunAdi;     //Ürün adı
        float brimFiyat;    //Birim fiyatı
    public:
        void stokAta(int stk, string uad, float bf) //deger atama fonsiyonu
        {
        stokKodu = stk;
        urunAdi = uad;
        brimFiyat = bf;
        }
        void goster()   //veriyi görüntüleyen fonksiyon
        {
        cout << "Stok Kodu   : " << stokKodu<<endl;
        cout << "Urun Adi    : " << urunAdi<<endl;
        cout << "Birim Fiyat : " << brimFiyat << endl;
        }
    };
int main()
    {
    string ua; int sn; float bf;
    Stok  stok1,stok2;  //Stok sinifina ait nesneyi tanimla
    cout<<"stok kodu   : ";cin>>sn;
    cout<<"Urun adi    : ";cin>>ua;
    cout<<"Birim Fiyat : ";cin>>bf;
    stok1.stokAta(sn, ua, bf);  //üye foksiyonu çagir
    cout<<"----------------------"<<endl;
    stok1.goster();           //üye foksiyonu çagir
    stok2=stok1;
    cout<<"----------------------"<<endl;
    stok2.goster();
    return 0;   }
```

Program 12.8 Bir nesneyi Başka Bir nesneye Atama Örneği

```
Stok kodu   : 0423
Urun adi    : Bilgisayar
Birim Fiyat : 5090
--------------------
Stok Kodu   : 423
Urun Adi    : Bilgisayar
Birim Fiyat : 5090
--------------------
Stok Kodu   : 423
Urun Adi    : Bilgisayar
Birim Fiyat : 5090
```

Program 14.8'in ekran çıktısı

12.5. const Fonksiyonlar ve const Nesneler

Daha önceki bölümlerde değişkenlerin program içerisinde değerlerinin değişmez olmasını sağlamak, programı veri açısından güvenlik düzeyini arttırmak için ***const*** bildirimini kullanmıştık. Nesneye yönelik programlamada da const nesneler ve const fonksiyonlar kullanarak aynı amaçları gerçekleştirebiliriz.

12.5.1. Problem 12.9

Bir C++ programı yazınız. Program içerisinde tanımlanan sınıfın içerisinde const bir üye fonksiyon bulunsun. Ayrıca, programda iki nesne tanımlayınız. Tanımlanan nesnelerin birincisi const diğeri const olmasın. Programın aşağıdaki ihtimallerden hangisinde hata verdiğini kontrol ediniz.

	const		const	Sonuç
Nesne	evet	Fonksiyon	evet	?
Nesne	hayır	Fonksiyon	hayır	?
Nesne	hayır	Fonksiyon	evet	?
Nesne	evet	Fonksiyon	hayır	?

```
1  #include <iostream>
2  using namespace std;
3  class prs{
4      public:
5          double maas;
6          prs(){
7          maas=0.0;
8          }
9          void maasAta(double m){
10             maas=m;
11         }
12         float geridon() const{
13         return maas;
14         }
15 };
16 int main() {
17 const prs prs1;
18 prs prs2;
19 prs1.geridon();
20 prs1.maasAta(7500.85);
21 prs2.
22 prs2.maasAta(8500.25);
```

public void prs::maasAta (double m)

(2) Resources | Compile Log | Debug | Find Results | Close

File	Message
C:\Users\Yedek\Desktop\Dersİçeriği\12_Sınıflar Nesn...	In function 'int main()':
C:\Users\Yedek\Desktop\Dersİçeriği\12_Sınıflar Nesneler...	[Error] passing 'const prs' as 'this' argument of 'void prs::maasAta(double)' discards qualifiers [-fpermissive]

Program 12.9A const fonksiyon ve const nesneler (20. Satır Hatalı, Nesne const-Fonks. Değil

Yapacağınız kontrollerin sonunda ihtimallerden sonuncusunda hata alacağınızı program 12.9A'dan görebilirsiniz. Denemeleriniz sonucunda tablo aşağıdaki gibi oluşacaktır:

	const		const	Sonuç
Nesne	evet	Fonksiyon	evet	Hata Yok
Nesne	hayır	Fonksiyon	hayır	Hata Yok
Nesne	hayır	Fonksiyon	evet	Hata Yok
Nesne	evet	Fonksiyon	hayır	Hata

Program 12.9B, 12.9A'dan 20. Satırın kaldırılmasıyla oluşturulmuştur. Program bu yeni şekli ile derlendiğinde hata vermemiş ve beklenen ekran çıktısı alınmıştır.

```
1  #include <iostream>
2  using namespace std;
3  class prs{
4      public:
5          double maas;
6          prs(){
7          maas=0.0;
8          }
9          void maasAta(double m){
10             maas=m;
11         }
12         float geridon() const{
13         return maas;
14         }
15 };
16 int main() {
17 const prs prs1;
18 prs prs2;
19 prs1.geridon();
20 //prs1.maasAta(7500.85);
21 prs2.geridon();
22 prs2.maasAta(8500.25);
23 cout<<prs2.maas<<endl;
24     return 0;}
```

Problem 12.9B const fonksiyon ve const nesne örneği

12.6. Arkadaş Fonksiyonlar

Arkadaş fonksiyonlar sınıfın dışında tanımlanan ve sınıfın içerisindeki private veriler ulaşmak için kullanılan fonksiyonlardır. Bir sınıfa ait arkadaş bir fonksiyon tanımlanacak ise ***friend*** anahtar kelimesinden yararlanılır. Arkadaş fonksiyonların bildirimleri sınıf içerisinde private veya public alanlarının içerisinde yapılabilir. Fonksiyon tanımlaması ise sınıf dışında yapılır. Aşağıda şekil 12.2'de arkadaş fonksiyonların bildiriminin prototipi gösterilmiştir.

```
class A{
    private:
        .....
        .....
        .....
    public:
        .....
        .....
        .....
        friend int goster();
};

int goster()
{
..........
..........
..........
}
```

Şekil 12.2 Arkadaş fonksiyonların tanımlanması

C++ ta arkadaş fonksiyon tanımlanabildiği gibi arkadaş sınıflar da tanımlanabilir. Arkadaşlık bir sınıf tarafından verilir ve bir fonksiyon veya başka bir sınıf tarafından alınır. İki sınıfın kendi aralarında arkadaş olması ve bu sınıflardan birinin başka bir sınıfla arkadaşlık ilişkisi bulunması, bu üçüncü sınıfın diğer sınıfla arkadaş olacağı sonucunu doğurmaz.

Aşağıda içerisinde arkadaş sınıf tanımlaması yapılmış ve kullanılmış bir C++ programı verilmiştir.

```
1  #include <iostream>
2  using namespace std;
3  class A {
4  private:
5      int a;
6    public:
7      A(){
8          a=4;
9      }
10     friend int kare();
11 }k;
12 int kare(){
13     return k.a*k.a;
14 }
15 int main() {
16     cout<<" Sayinin Karesi : "<<kare()<<endl;
17     return 0;
18 }
```

Program 12.10 Arkadaş fonksiyonlar

```
Sayinin Karesi : 16

--------------------------------
Process exited after 0.5454 seconds with return value 0
Press any key to continue . . .
```

Program 12.10'un ekran çıktısı

12.7. Nesne Dizileri

Nesne dizilerini ele almadan önce program 12.11' inceleyelim. Program 12.11'de önce Ogrenci isimli bir sınıf oluşturulmuştur. Bu sınıfta ***private*** üye olarak iki özellik (***string Ad, int Ortalama***) ve ***public*** üye olarak üç fonksiyon (***void getAd(), void getOrtalama, void goruntule()***) yer almaktadır.

Sınıfı yukarıda açıklandığı gibi tanımladıktan sonra Ogrenci sınıfına ait üç nesne oluşturuldu (***ogr1, ogr2, ogr3***). Daha sonra ***Ogrenci*** sınıfına ait ***getAd(), getOrtalama()*** isimli üye fonksiyonlar çağırılarak nesnelerin private olan özelliklerinin klavyeden girişleri yapıldı. Programın sonunda da **goruntule()** üye fonksiyonu çağırılarak bu üç nesneni bilgisayara girilen özellikleri ekranda gösterildi.

```cpp
#include <iostream>
#include <locale.h>
using namespace std;
class Ogrenci
{
    string Ad;
    int Ortalama;
    public:
        void getAd(){
            getline( cin, Ad);
        }
        void getOrtalama(){
            cin >> Ortalama;
        }
        void goruntule(){
            cout << "Ad       : " << Ad << endl;
            cout << "Ortalama : " << Ortalama << endl;
        }
};
int main(){
    setlocale(LC_ALL,"turkish");
    Ogrenci ogr1,ogr2,ogr3; int i=1;
    cout << "Öğrenci " << i << endl;
    cout << "Öğrenci Adı         : ";cin.ignore();ogr1.getAd();
    cout << "Öğrenci Ortalaması : ";ogr1.getOrtalama();i+=1;
    cout << "Öğrenci " << i << endl;
    cout << "Öğrenci Adı         : "; cin.ignore(); ogr2.getAd();i+=1;
    cout << "Öğrenci Ortalaması : "; ogr2.getOrtalama();
    cout << "Öğrenci " << i << endl;
    cout << "Öğrenci Adı         : ";cin.ignore(); ogr3.getAd();
    cout << "Öğrenci Ortalaması : ";ogr3.getOrtalama();
    ogr1.goruntule();ogr2.goruntule();ogr3.goruntule();
    return 0;}
```

Program 12.11 Nesne Dizisi Kullanılmadan Öğrenci Bilgilerinin Bilgisayara Girilmesi

Yukarıda verilen örnekte sadece üç öğrencinin verilerinin bilgisayara girilmesi istenmişti. Her zaman bu şekilde aynı sınıfa ait nesnelerin bir veya bir kaçıyla ilgili işlem yapmamız gerekseydi, problemlerimizi program 12.11'deki gibi bir yaklaşım belirleyerek çözebilirdik.

Fakat bazen 50 bazen 100 bazen binler hatta on binlerce öğrencinin bilgilerinin bilgisayara girilmesi gerekebiliyor. Böyle bir durumda program 12.11'i kullanmamız gerekseydi 10 000 nesne tanımlayıp her bir nesnenin özelliklerinin girişini yapabilmek için 12 ifade yazmamız gerekecekti. Yazılması gereken toplam ifade sayısının ne kadar artacağını kestirmeyi size bıraktık.

Anlatılan öyküden bu ve benzeri işlemler için program 12.11'de kullanılan yaklaşımdan başka bir yaklaşım bulma ihtiyacında olduğumuz kesin olarak anlaşılmaktadır. Bunun için program 12.12 hazırlandı ve bu programda öğrenciler için nesnelerinden oluşan bir dizi tanımlanarak yine 3 öğrencinin adının ve ortalamasının bilgisayara girilmesini sağlayan örnek gösterildi. Program 12.12'yi inceleyerek çözümün ne kadar kısaldığını ve pratik hale geldiğini anlamaya çalışınız. Ayrıca, çok küçük müdahalelerle bu programla istenildiği kadar öğrencinin bilgelerinin bilgisayara girileceği açıktır.

```cpp
#include <iostream>
using namespace std;
class Ogrenci
{
    string Ad;
    int Ortalama;
    public:
        void getAd(){
            getline( cin, Ad);
        }
        void getOrtalama(){
            cin >> Ortalama;
        }
        void goruntule(){
            cout << "Ad       : " << Ad << endl;
            cout << "Ortalama : " << Ortalama << endl;
        }
};
int main(){
    setlocale(LC_ALL,"turkish");
    int sayi=3;Ogrenci ogr[sayi];
    for( int i=0; i<sayi; i++ ){
        cout <<endl<< "Öğrenci " << i+1 << endl;
        cout << "Öğrenci Adı         : ";cin.ignore();ogr[i].getAd();
        cout << "Öğrenci Ortalaması : ";ogr[i].getOrtalama();
    }
    for(int i=0;i<sayi;i++) {
        cout <<endl<<"Öğrenci " << i+1 << endl;
        ogr[i].goruntule();
    }
    return 0;}
```

Program 12.12 Nesne Dizileri

Program 12.12 incelendiğinde nesne dizilerinin tanımlanması ve kullanılmasının daha önce kullandığımız dizilere çok benzediği görülmektedir. Bundan başka nesne dizilerinin kullanılmasının çok daha kısa, okunabilir program yazmak için çok pratik faydalar sağladığı görülmektedir.

Bölüm Özeti

C++ programlama dili aslında birçok yönden C programlama diline benzemektedir. C++ programlama dilinin C programlama dilinden farklı en temel özelliği nesneye yönelik programlama yaklaşımını destekliyor olmasıdır. Nesneye yönelik programlamanın ilk adımı ve en önemli aracı sınıflardır. Bu bölümde Bir sınıfın, bir nesnenin biçimini belirtmek için kullanıldığını verilerle, verileri işlemek için kullanılacak işlevleri tek bir düzgün pakette birleştirdiğini öğrendik. Bundan sonrası için sınıf içindeki verilerin (özelliklerin) ve işlevlerin (fonksiyonların) sınıfın üyeleri olarak isimlendirileceğini biliyoruz.

On ikinci bölümün devamında, program içerisinde tanımı yapılmış olan sınıfa ait nesnelerin nasıl tanımlanacağını ele aldık. Nesneye yönelik programlama yaklaşımında nesnelerin tanımlanması işlemine ***örnek(instance) oluşturmak***, işlem neticesinde oluşturulan nesnelere de ***örnek değişken***(***instance variable***) adı da verilmektedir. Esasında programlar içerisinde nesne tanımlamanın, herhangi bir temel veri tipindeki değişken tanımlamaya çok benzediğine dikkat ediniz. Ayrıca, kullanıcı tarafından tanımı yapılmış bir yapı ile yapı değişkeni tanımlama işleminin de tanımı yapılmış sınıfla nesne tanımlanmasına benzediğini hatırdan çıkarmayınız.

Bu bölümde daha sonra, tanımlanmış olan nesneler aracılığı ile o sınıfa ait üye fonksiyonlara erişimin nasıl yapılacağı örneklerle açıklandı. Üye fonksiyonların çağırılması için oluşturulan C++ ifadelerinde üye fonksiyon ile nesnenin isminin nokta operatörünün ilişkilendirdiği gösterildi.

C++'da nesnelerin otomatik olarak ilk kullanıma hazırlanması işleminin kurucu (constructor) fonksiyonlar ile yapıldığı yine bu bölümde ele alındı ve kurucu fonksiyonların isimlerinin sınıflarla aynı olması gerekliliği vurgulandı. Benzer şekilde çalışması sona ermiş nesneler için bilgisayar belleğinde ayrılmış olan adreslerin serbest bırakılması amacıyla da de yıkıcı fonksiyonların kullanıldığı örneklerle açıklandı.

Sınıflar ve Nesneler bölümünün son dört konusu Bir nesneyi Başka Bir Nesneye Atamak, const nesneler ve const sınıflar, Arkadaş fonksiyonlar ve nesne dizileri oldu.

Bir nesneyi başka bir nesneye kopyalamak başlığı altında aynı sınıfa ait nesnelerin birbiri arasında, değişkenlerde olduğu kopyalanabileceği örnekle gösterildi.

Daha önceki bölümlerde değişkenlerin program içerisinde değerlerinin değişmez olmasını sağlayarak, programın veri açısından güvenliğinin arttırılması gerektiğini, bunun için const bildiriminin kullanılabileceğini öğrenmiştik. Bu bölümde de nesneye yönelik programlamada **const** nesneler kullanarak aynı amaçların gerçekleştirebileceği bir örnekle açıklandı.

Arkadaş fonksiyonların sınıfın dışında tanımlandığı ve sınıfın içerisindeki private verilere ulaşmak için kullanıldığı, bir sınıfa ait arkadaş fonksiyonun ***friend*** anahtar kelimesinden yararlanarak tanımlanabileceği ve arkadaş fonksiyonların bildirimlerinin sınıf içerisinde private veya public alanlarının içerisinde yapılabileceği yine bir örnek üzerinden gösterildi.

Bu bölümün son konusu nesne dizileri oldu. Nesne dizilerinin tanımlanması ve kullanılmasının daha önceki bölümlerde kullanılan dizilere çok benzediği ve daha kısa kod yazarak program yazmaya önemli katkı sağlayacağı gösterildi.

ÜNİTE SORULARI

1. Aşağıdakilerden hangisi nesneye yönelik programlama yaklaşımını temel özelliklerinden biri değildir?

- Soyutlama
- Çok Biçimlilik
- Yapılar
- Kalıtım
- Paketleme

2. "Oluşturulan nesnelerin gerektiğinde başka bir nesne gibi davranabilme özelliği" nesneye yönelik programlama yaklaşımının hangi temel özelliğini açıklar?

- Soyutlama
- Çok Biçimlilik
- Yapılar
- Kalıtım
- Paketleme

3. "Temel Bir sınıftan yeni bir sınıf oluşturma işlemi" nesneye yönelik programlama yaklaşımının hangi temel özelliğidir?

- Soyutlama
- Çok Biçimlilik
- Yapılar
- Kalıtım
- Paketleme

4. "Verileri ve bu veriler üzerinde çalışan işlevleri aynı isim altında tanımlamak" nesneye yönelik programlama yaklaşımının hangi temel özelliğidir?

- Soyutlama
- Çok Biçimlilik
- Yapılar
- Kalıtım
- Paketleme

5. Aşağıdakilerden hangisi nesneye yönelik programlamada sınıf oluşturulurken o sınıfa ait nesnelerin özelliklerini ve işlevlerini diğer nesnelerden korumak için kullanılan bir tanımlama şeklidir?

- private (özel) tanımlayarak
- global olarak tanımlayarak
- yerel olarak tanımlayarak
- static olarak tanımlayarak
- static yerel tanımlayarak

6. Nesneye Yönelik programlamada sınıfa ait nesne ile yine o sınıfa ait üye fonksiyonu çağırmak için nesne ile üye fonksiyon aşağıda verilenlerden hangisi ile ilişkilendirilir?

- : (iki nokta üst üste)
- . (nokta)
- ! (Ünlem)
- ? (Soru İşareti)
- = (Atama Operatörü)

7. Nesneye Yönelik programlama tekniği ile geliştirilmiş bir programda Kare isimli bir sınıf ve bu sınıfa ait Kare1 isimli bir nesne tanımlanmıştır. Kare sınıfında, karenin çevresini hesaplayıp ekrana yazdıran Cevre() isimli bir üye fonksiyon da vardır. Aşağıdaki şıklardan hangisinde karenin çevresini hesaplayıp sonucu ekrana yazdırmak için yapılan fonksiyon çağrısı doğru olarak verilmiştir?

Not: *Karenin kenar uzunluğun sınıfta tanımlı olan başka bir üye fonksiyon tarafından daha önce atandığını varsayarak soruyu cevaplandırınız.*

- Kare.Kare1;
- Kare kare1;
- Kare.Cevre();
- Kare1.Cevre();
- Cevre().Kare1;

8. C++'ta nesnenin otomatik olarak ilk kullanıma hazırlanması işlemi aşağıda verilen fonksiyonlardan hangisi ile yapılır?

- protected fonksiyonla
- public fonksiyonla
- private fonksiyonla

- yıkıcı fonksiyonla
- kurucu fonksiyonla

9. Aşağıda verilenlerden hangisi kurucu fonksiyon özelliklerinden değildir?

- Sınıfa ait nesne oluşturulduğunda kurucu fonksiyon otomatik olarak çağırılır.
- Kurucu fonksiyonlar üyesi olduğu sınıfla aynı isme sahip olamaz.
- Kurucu fonksiyonlar değer döndürmez.
- Kurucu fonksiyonlar nesne oluşturulduğunda sadece bir kez çalıştırılır.
- Bir sınıfa ait kurucu fonksiyon tanımlanmaz ise derleyici tarafından otomatik olarak oluşturulur.

10. Nesneye yönelik programlamada sınıflarda yıkıcı fonksiyon tanımlanırken fonksiyonun önüne hangi işaret konulur?

- **~** (Tilda işareti)
- **–** (Tire İşareti)
- **?** (Soru İşareti)
- **!** (Ünlem İşareti)
- **%** (Yüzde işareti)
    -
    - CEVAP ANAHTARI1. c2. b3. d4. e5. a6. b7. d8. e9. b10. a

13. KALITIM

Giriş

Kalıtım, sınıf ve nesnelerden sonra nesneye yönelik programlamanın en popüler konusu en güçlü özelliğidir. Kalıtım, bir sınıfın diğer sınıfın özelliklerini ve işlevlerini miras yoluyla alma tekniğidir. Bu şekilde önceden yazılmış ve doğrulanmış olan kodu tekrar tekrar yazmadan kullanma şansına sahip oluruz.

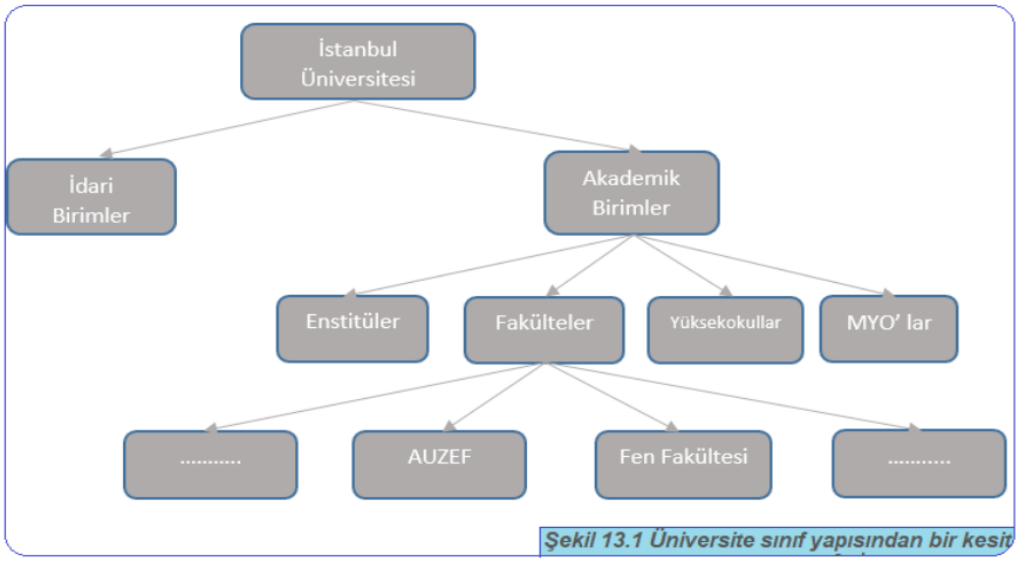
Kalıtımda, miras yoluyla özellikleri ve işlevleri alınan sınıfa ***temel sınıf*** veya ***üst sınıf*** denir. Söz konusu bu özellik ve işlevleri alan sınıfa da **türetilmiş sınıf** denir. Türetilmiş sınıf türetildiği temel sınıfın bütün özelliklerini ve işlevlerini kullanabilme özelliğine sahip olur.

Temel Sınıf: Temel sınıf, Nesne Tabanlı Programlama dilinde, diğer sınıfların türetildiği sınıfın adıdır. Temel sınıfı devralan sınıf, bu temel sınıfın tüm üyelerine sahiptir ve ayrıca bazı ek özelliklere de sahip olabilir. Temel sınıf üyeleri ve üye işlevleri, türetilmiş sınıfın Object öğesine devralınır. Yukarıda da belirtildiği gibi temel sınıf, üst sınıf olarak da adlandırılır.

Türetilmiş Sınıf: Var olan bir sınıftan (Temel Sınıftan, üst sınıftan) oluşturulan sınıftır. Türetilmiş sınıf, bir temel sınıfın tüm üyelerini ve üye işlevlerini devralır. Türetilmiş sınıf, temel sınıfına göre daha fazla işlevselliğe sahip olabilir ve temel sınıfına kolayca erişebilir. Türetilmiş alt sınıf da denir.

Nesneye yönelik programlamanın daha önce kullanılan yapısal programlama yaklaşımına göre, kalıtım tekniği ile programlama sürecine sağladığı önemli avantajlar vardır. Bu avantajlardan bazıları yazılmış olan kodun yeniden kullanılmasına izin verilmesi, bunun sonucu olarak sınıf kütüphanelerinin dağıtımının kolaylaşması, yazılım projelerinin geliştirilmesi ve bakımı sırasında zamandan ve paradan tasarruf edilmesinin sağlaması, bu teknik kullanılarak yazılan programların okunabilirliğinin artması şeklinde sayılabilir.

Aşağıda şekil 13.1'de üniversite sınıf yapısından temsili bir kesit gösterilmiştir.



Şekil 13.1 Üniversite sınıf yapısından bir kesit

Şekil 13.1'de en alt seviyedeki türetilmiş sınıfların fakülteler olduğu görülmektedir. Çok büyük olmasının önüne geçmek için, şekilde bütün fakültelere yer verilmemiştir. Şekle baktığımızda temel sınıfın İstanbul Üniversitesi olduğunu görüyoruz. Bu temel sınıfa ait özelliklerin İstanbul üniversitesinin adresi, adrese ek olarak üniversitenin kuruluş yılı, eski dönemlere ait ve hâlihazırdaki üst yönetim organları, yöneticiler vb. olduğunu varsayabiliriz. Bunun yanında mesela bunların dökümünün alınması ile ilgili bir işlev olması gerektiğini de söyleyebiliriz. Burada tamamını saymadığımız bütün bu özellik ve işlevler Üniversitenin Rektörlük birimini ilgilendirdiği gibi örneğin fakülteleri de doğrudan ilgilendirmektedir.

Konunu anlaşılması için, temel sınıfın özelliklerinden olan üniversite rektörünün isminin, bütün fakülteleri, öğrencilerin diplomalarının imzalanması açısından ilgilendirdiğini hatırlayalım. Dolayısı ile bütün fakülteler, fakültelerin yanında enstitüler, yüksekokullar MYO 'lar temel sınıfın bu özelliğini kendilerine ait sınıf veya sınıflardan birinde bu özelliği bulundurmadan kullanabileceklerdir. Tabi temel sınıfın özelliklerinin yanında işlevlerinin (üye fonksiyonlar) de aynı şekilde türetilmiş sınıflar tarafından kullanabileceklerini hatırdan çıkarmamalıyız. Bu basit örnek bile kod yazarken, verileri depolarken nesneye yönelik programlama yaklaşımının kalıtım özelliğinin bize ne kadar katkı sağladığını açık olarak anlatmaktadır. Sonuç olarak kalıtım yoluyla elde ettiğimiz temel avantajın, kodun tekrar kullanılabilirliği olduğunu burada söylemeliyiz.

C ++ 'da temel sınıftan türetilmiş sınıf oluşturmak için aşağıdaki sözdizimi kullanılır:

```
class Türetilmiş_Sınıf : Erişim_Belirteci Temel Sınıf
```

İpucu! *Temel sınıftan türetilmiş sınıf oluşturulması sırasında erişim belirteci yazılmamış ise erişim belirtecinin varsayılanı*

Kalıtım yoluyla sınıfların türetilmesi göstermek amacıyla çizilen şekil 13.1 için aşağıdaki açıklamaları okuyunuz:

En üstteki İstanbul Üniversitesi sınıfı tüm sınıfların temel sınıfıdır.

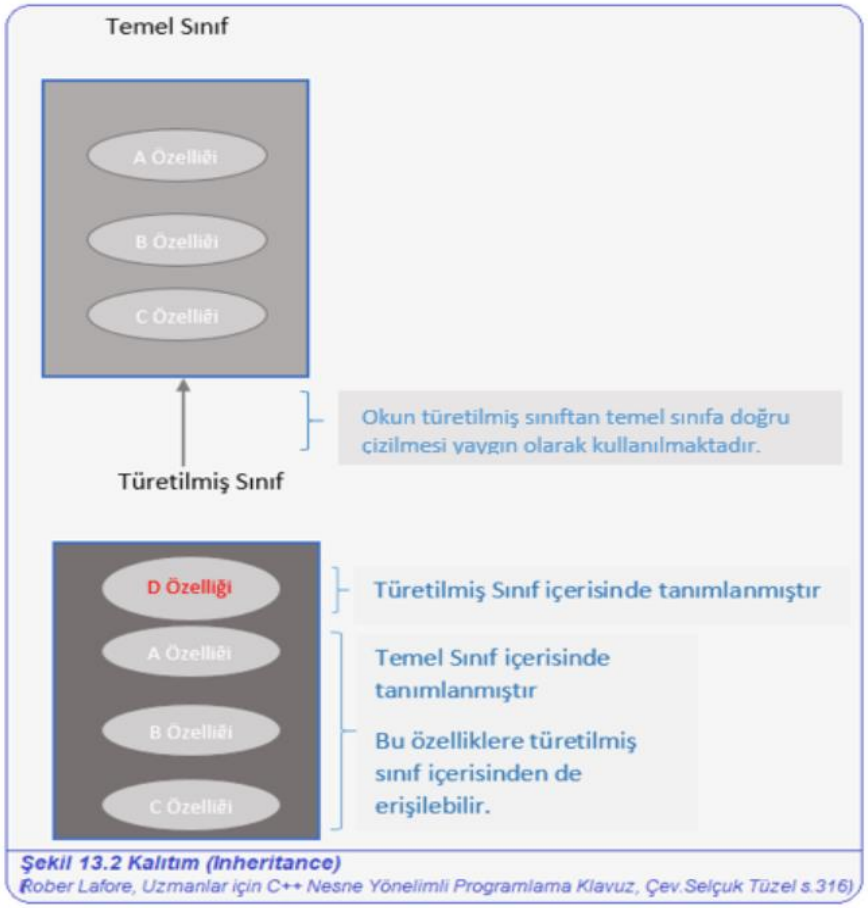
İdari Birimler ve Akademik Birimler İstanbul Üniversitesi sınıfı için türetilmiş sınıftır.

Enstitüler, Fakülteler, Yüksekokullar ve MYO 'lar için Akademik Birimler üst sınıftır.

İstanbul Üniversitesi sınıfından bulunan tüm özellikler, bu sınıftan türetilmiş İdari Birimler ve Akademik Birimler sınıflarının da özellikleridir. Ancak İdari Birimler ve Akademik Birimler sınıflarında bulunan bir özellik İstanbul Üniversitesi sınıfında olmayabilir. Türetilen sınıflar temel sınıfların bütün özelliklerini taşırken kendilerine ait özellikler de barındırabilirler.

AUZEF sınıfı Fakülteler sınıfına göre daha az nesneyi ifade eder. Çünkü AUZEF sınıfında sadece o sınıfa ait nesneler olurken Fakülteler sınıfında her bir fakülteye ait sınıftan nesneler oluşturulur.

Temel sınıf ile Türetilmiş sınıf arasındaki kalıtım ilişkisi şekil 13.2'de gösterilmiştir.



Şekil 13.2 Kalıtım (Inheritance)
Rober Lafore, Uzmanlar için C++ Nesne Yönelimli Programlama Klavuz, Çev.Selçuk Tüzel s.316)

13.1 Türetilmiş Sınıf Tanımlama

Sınıflar ve Nesneler başlıklı önceki ünitede public ve private erişim belirteçlerinin sınıf tanımlanırken nasıl kullanıldığını ve sınıf içerisinde public ve private olarak tanımlanmış olan üyelere erişimle ilgili kuralları açıklamıştık. Bu kuralları burada da özetleyecek olursak, bir sınıfa ait public elemanlara o sınıfın üye fonksiyonları ile erişilebileceği gibi sınıf dışından, üye fonksiyon kullanmadan da o sınıfın public elemanlarına erişilebilir. Bir sınıfa ait private üye değişken veya fonksiyonlarına ise sadece o sınıftan veya sınıfa ait arkadaş fonksiyonlardan erişilebilir, sınıf dışından asla erişilemez.

Sınıflar ve Nesneler ünitesinde (12. Bölüm) kullanmadığımız fakat sınıfların tanımlanması sırasında kullanılan üçüncü bir erişim belirteci daha vardır. Bu erişim belirteci protected erişim belirtecidir. Bir sınıf içerisinde protected olarak tanımlanmış üyeler, erişilebilirlik açısından sınıfın private üyelerine çok benzer. Protected üyelerin private üyelerden tek farkı türetilmiş sınıflardan protected üyelere erişilebilir.

İpucu! *Bir sınıfa ait üye fonksiyon veya değişkenler tanımlanırken erişim belirteci kullanılmamış ise, sınıfın o üyelerinin private olarak tanımlandığı varsayılır.*

Bir sınıf bir temel sınıftan türetilirken, temel sınıf public, protected veya private miras yoluyla miras alınabilir. Kalıtımın türü, erişim belirteci tarafından belirtilir.

Public Kalıtım – Bir temel sınıftan public erişim belirteci (public niteleyici) kullanarak bir sınıf türetildiğinde, ana sınıfın public üyeleri türetilmiş sınıfın da public üyeleri olur ve temel sınıfın protected üyeleri türetilmiş sınıfın protected üyeleri olur. Bir temel sınıfın private üyelerine hiçbir zaman doğrudan türetilmiş bir sınıftan erişilemez, ancak temel sınıfın public ve private üyelerine çağrılar yoluyla erişilebilir.

Protected Kalıtım – Bir temel sınıftan protected erişim belirteci (protected niteleyici) kullanarak bir sınıf türetildiğinde, temel sınıfın public ve protected üyeleri türetilmiş sınıfın protected üyeleri haline gelir.

Private Kalıtım – Bir temel sınıftan private erişim belirteci (private niteleyici) kullanarak bir sınıf türetildiğinde, temel sınıfın public ve private üyeleri türetilmiş sınıfın private üyeleri haline gelir.

Yukarıda, bir temel sınıftan bir türetilmiş sınıf oluşturulması sırasında kullanılan erişim belirtecine göre temel sınıfın üyelerinin türetilmiş sınıf için nasıl nitelenmesi gerektiği açıklandı. Tablo 13.1'te Bu açıklamalara göre temel sınıf elemanlarına erişim sınırlılıkları özet olarak gösterildi.

Temel Sınıf üyelerinin erişim belirteç tipi	Kalıtım Tipi		
	Public	Protected	Private
Public	public	protected	private
Protected	protected	protected	private
Private	Erişim Yok	Erişim Yok	Erişim Yok

Tablo 13.1 Temel Sınıf üyelerine Kalıtım tipine göre erişim seviyeleri

13.1.1 Public Erişim Belirteci Kullanarak Sınıf Türetme

Temel sınıftan türetilmiş sınıfa miras alınırken erişim belirteci olarak public kullanılır. Public niteleyicisi ile sınıf türetmek için kullanılan söz dizimi aşağıdaki gibidir.

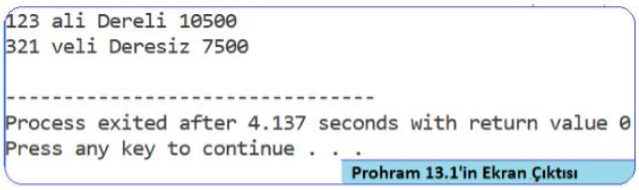
```
class türetilecek_sınıf_ismi : public temel_sınıfın_ismi
```

Türetilen sınıf için, türetildiği ana sınıfın public olan bütün elemanları türetilmiş sınıfın public, protected elemanları protected eleman olur. Türetilmiş sınıfın bütün nesneleri temel sınıfın public elemanlarına ve protected elemanlarına erişebilir. Türetilmiş sınıf nesneleri temel sınıfın private elemanlarına erişemez.

Program 13.1 public erişim belirteci ile temel sınıftan türetilmiş sınıf oluşturmaya örnek olması için yazılmış ve kaynak kodları ile ekran çıktısı aşağıda verilmiştir.

```
1  #include <iostream>
2  using namespace std;
3  class personel
4  {
5      public:
6      string prsNo,ad,soyad;
7      float maas;
8          personel(){};
9          void goruntule()
10         {
11             ccut<<prsNc<<" "<<ad<<" "<<soyad<<" "<<maas<<endl;
12         }
13 };
14 class muhendis:public personel
15 {
16     public:
17       muhendis(string n,string a,string s,float m)
18         {
19             prsNo=n;
20             ad=a;
21             soyad=s;
22             maas=m;
23         }
24 };
25 class isci:public personel
26 {
27     public:
28     isci(string n,string a,string s,float m)
29         {
30             prsNo=n;
31             ad=a;
32             soyad=s;
33             maas=m;
34         }
35 };
36 int main() {
37     muhendis m1("123","ali","Dereli",10500);
38     isci i1("321","veli","Deresiz",7500);
39     m1.goruntule();
40     i1.goruntule();
41     return 0;
42 }
```

Program 13.1 Public ile Temel sınıftan türetilmiş sınıf oluşturma



Program 13.1'in işlem adımları aşağıda sıralanmıştır.

Program 13.1 'de temel sınıf olarak personel sınıfı oluşturuldu ve her personelin numarası, adı ve soy adı olacağı için PrsNo, ad, soyad üye değişkenleri ve goruntule() üye fonksiyonu oluşturuldu.

Daha sonra personel temel sınıfından muhendis adında bir sınıf, public erişim belirteci kullanılarak türetildi ve yapıcı fonksiyon ile personel sınıfının üye değişkenlerine atama yapıldı.

Aynı şekilde işlemler isci sınıfı için de tekrarlandı.

main() fonksiyonu içerisinde m1 ve i1 nesneleri tanımlandı ve atanan değerler görüntüle fonksiyonu ile yazdırıldı.

13.1.2. Private Erişim Belirteci Kullanarak Sınıf Türetme

Temel sınıftan türetilmiş sınıfa miras alınırken erişim belirteci olarak private kullanılır. C++ 'ta Temel sınıftan türetilmiş sınıfa miras alınırken erişim belirteci yazılmaz ise varsayılan olarak erişim belirtecinin private olduğu kabul edilir. private niteleyicisi ile sınıf türetmek için kullanılan söz dizimi aşağıdaki gibidir.

```
class Türetilecek_Sınıf_İsmi : private Temel_Sınıfın_İsmi
```

Private türetmede ana sınıfın public ve protected üyeleri türetilmiş sınıf için private hale gelir. Burada dikkat edilmesi gereken husus türetme şekli ne olursa olsun ana sınıfın private elemanlarına erişilemez.

Program 13.2 private erişim belirteci ile temel sınıftan türetilmiş sınıf oluşturmaya örnek olması için yazılan programın kaynak kodları ve ekran çıktısı aşağıda verilmiştir.

Program 13.2'nin işlem adımları aşağıda verildiği gibi olacaktır.

Program 13.2 'de temel sınıf olarak personel sınıfı oluşturuldu ve PrsAd, prsSoyad üye değişkenleri public olarak temel sınıfta tanımlandı.

Daha sonra personel temel sınıfından prsAta adında bir sınıf, private erişim belirteci kullanılarak türetildi ve yapıcı fonksiyon ile personel sınıfının üye değişkenlerine atama yapıldı. (Temel sınıf olan personel sınıfına ait public üye değişkenlerin PrsAta sınıfı için private niteliğe dönüştüğünü hatırınızdan çıkarmayınız)

main() fonksiyonu içerisinde türetilmiş sınıfa ait kişi nesnesi tanımlandı PrsAd ve prsSoyad üye değişkenlerine türetilen sınıfın kurucu fonksiyonu ile atama yapıldı.

Türetilen sınıfa ait görüntüle üye fonksiyonu ile personelin ad ve soyadı görüntülendi.

```cpp
#include <string>
#include <iostream>
using namespace std;
class personel
{
      public:
            string prsAd,prsSoyad;
            X(){};
      };
class prsAta:private personel
{
      public:
            prsAta(string a,string s)
            {
                  prsAd=a;
                  prsSoyad=s;
                  }
            void goruntule()
            {
                cout<<prsAd<<"\t"<<prsSoyad<<endl;
             }
      };
int main()
{
    prsAta kisi("Mehmet","Oduncu");
    kisi.goruntule();
    system("PAUSE");
    return 0;
}
```

Program 13.2 private il temel sınıftan türetilmiş sınıf oluşturma

```
Mehmet  Oduncu
Press any key to continue . . .

--------------------------------
Process exited after 2.869 seconds with return value 0
Press any key to continue
```

Program 13.2'nin Ekran çıktısı

13.1.3. Protected Erişim Belirteci Kullanarak Sınıf Türetme

Temel sınıftan türetilmiş sınıfa miras alınırken erişim belirteci olarak protected kullanılır. protected niteleyicisi ile sınıf türetmek için kullanılan söz dizimi aşağıdaki gibidir.

```
class Türetilecek_Sınıf_ismi : protected Temel_Sınıf ismi { }
```

public ile sınıf türetme işleminde ana sınıfın public elemanlarının türetilmiş sınıf için public, ana sınıfın protected elemanlarının türetilmiş sınıf için protected olduğunu ve private ile sınıf türetmede ana sınıfın public ve protected elemanlarının türetilen sınıf için private olduğunu daha önce gördük.

Protected ile sınıf türetmede ise ana sınıfın public veya protected elemanları türetilmiş sınıfın protected elemanı haline gelir.

Program 13.3 protected erişim belirteci ile temel sınıftan türetilmiş sınıf oluşturmaya örnek olması için yazılan programın kaynak kodları ve ekran çıktısı aşağıda verilmiştir.

```cpp
#include <iostream>
#include <string>
using namespace std;
class calisan
{
      protected:
            string ad,soyad;
      public:
            calisan(){};
            void isim(string a,string b)
            {
                ad=a;
                soyad=b;
             }
      };
class muhendis:protected calisan
{
    int maas;
      public:
            muhendis(){};
            void muh_maas(int m)
            {
                maas=m;
                }
            void erisim(string a, string b)
            {
                isim(a,b);
                }
            friend void listele(muhendis &);
      };
void listele(muhendis &x)
{
     cout<<x.ad<<endl;
     cout<<x.soyad<<endl;
     cout<<x.maas<<endl;
     }
int main()
{
     muhendis kisi[3];
     kisi[0].erisim("ahmet","ak");
     kisi[1].erisim("hasan","kara");
     kisi[2].erisim("ayse","pembe");
     kisi[0].muh_maas(3000);
     kisi[1].muh_maas(4000);
     kisi[2].muh_maas(5000);
     for(int i=0;i<3;i++)
     listele(kisi[i]);
     system("PAUSE");
     return 0;
}
```

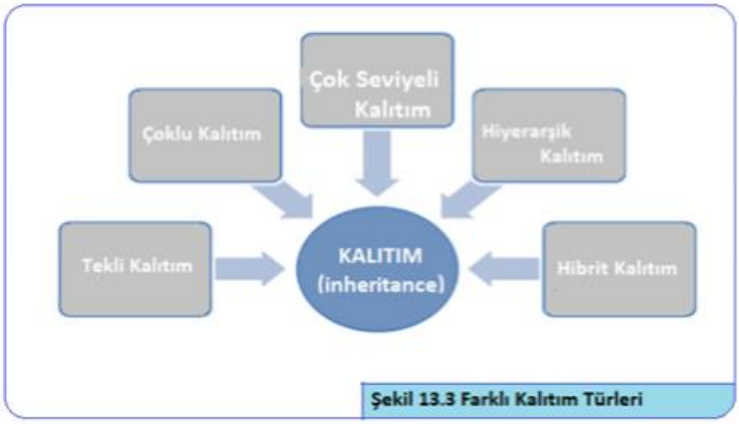
Program 13.3 Protected ile temel sınıftan türetilmiş sınıf oluşturma

13.2 Kalıtım Türleri

Bu ünitede şu ana kadar kalıtım hakkındaki temel bilgileri öğrendik. C++ programlama dilinde sınıfların türetilme şekillerine veya türetilmesi sırasında kaç sınıftan miras aldığına bağlı olarak birbirinden farklı kalıtım türleri vardır. Bu farklı kalıtım türleri aşağıda listelenmiştir:

1. Tekli Kalıtım
2. Çoklu Kalıtım
3. Çok seviyeli Kalıtım
4. Hiyerarşik Kalıtım
5. Hibrit Kalıtım

Bu farklı kalıtım türleri temsili olarak şekil 13.3'te gösterilmiştir.

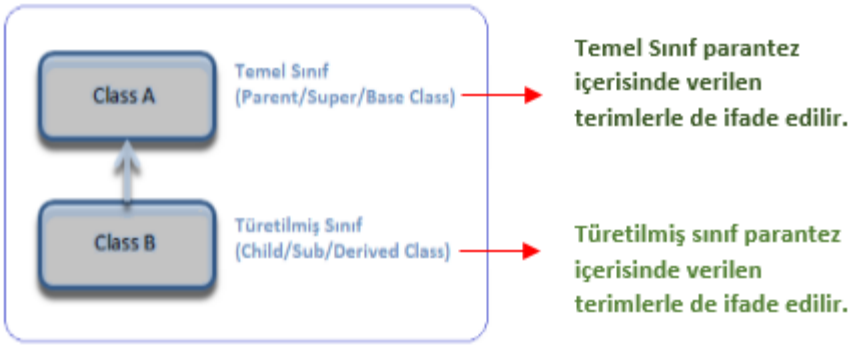


Şekil 13.3 Farklı Kalıtım Türleri

13.2.1. Tekli Kalıtım

Tekli kalıtımda, bir sınıf sadece bir temel sınıftan türetilir. Bu, bir üst sınıftan türetilen yalnızca bir alt sınıfın olduğu anlamına gelir. Tekli kalıtım için kullanılan söz dizimi aşağıdaki gibidir.

```
class Türetilmiş_Sınıf : Erişim_Belirteci Temel Sınıf { };
```



Program 13.4 Tekli Kalıtıma örnek olarak yazılmıştır.

```
#include <iostream>
using namespace std;
class A
{
    public:
        A()
        {
            cout<<"Kurucu A"<<endl;
        }
};
class B:public A
{
    public:
        B()
        {
            cout<<"Kurucu B"<<endl;
        }
};
int main() {
    B b1;
    return 0;
}
```

Program 13.4 Tekli Kalıtım

```
Kurucu A
Kurucu B

--------------------------------
Process exited after 3.546 seconds with return value 0
Press any key to continue . . .
```

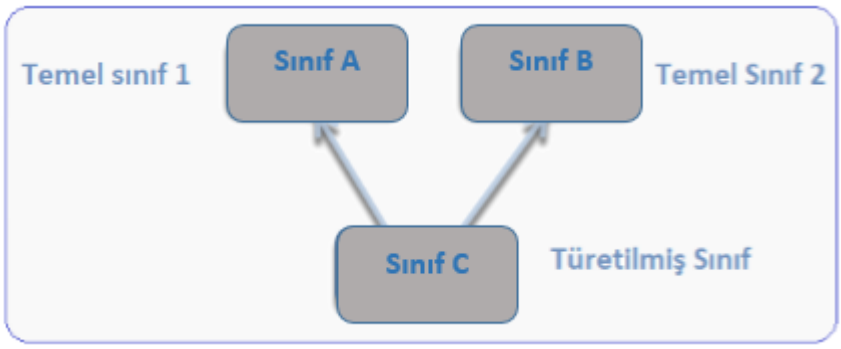
Program 13.4'ün Ekran çıktısı

13.2.2. Çoklu Kalıtım

Çoklu kalıtım, bir sınıfın birden fazla ana sınıftan türetildiği bir kalıtım türüdür. Aşağıdaki şekilde gösterildiği gibi, C sınıfı ana sınıfı A sınıfı ve B sınıfı olan bir türetilmiş sınıftır. Çoklu kalıtım için kullanılan söz dizimi aşağıdaki gibidir.

```
class C : public A, public B {      };
```

Çoklu kalıtım şeması aşağıda gösterilmiştir.



Program 13.5 Çoklu Kalıtıma örnek olarak yazılmıştır.

```
#include <iostream>
using namespace std;
class A {
public:
  A(){
     cout<<"Kurucu A class"<<endl;
  }
};
class B {
public:
  B(){
     cout<<"Kurucu B class"<<endl;
  }
};
class C: public A, public B {
public:
  C(){
     cout<<"Kurucu C class"<<endl;
  }
};
int main() {
    C obj;
    return 0;
}
```

Program 13.5 Çoklu kalıtım örneği

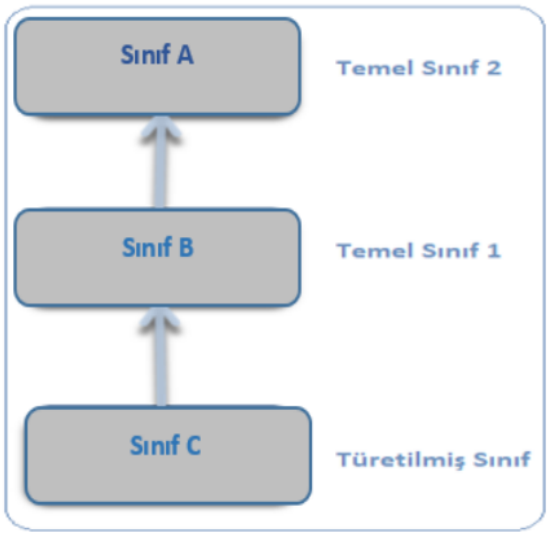
```
Kurucu A class
Kurucu B class
Kurucu C class

--------------------------------
Process exited after 3.538 seconds with return value 0
Press any key to continue . . .
```

Program 13.5'in ekran çıktısı

13.2.3. Çok Seviyeli Kalıtım

Çok düzeyli kalıtımda, bir sınıf önceden türetilmiş başka bir sınıftan türetilir. Aşağıdaki şemadan, C sınıfının B Sınıfından ve B Sınıfının da A Sınıfından türetildiği görülmektedir.



Çoklu seviyeli kalıtım için kullanılan söz dizimi aşağıdaki gibidir:

```
class A{
...
};
class B : public A{
...
};
class C : public B{
...
};
```

```
#include <iostream>
using namespace std;
class A
{
    public:
        A()
        {
            cout<<"kurucu A"<<endl;
        }
};
class B:public A
{
    public:
        B()
        {
            cout<<"kurucu B"<<endl;
        }
};
class C:public B
{
    public:
        C()
        {
            cout<<"Kurucu C"<<endl;
        }
};
int main() {
    C n1;
    return 0;  }
```

Program 13.6 Çok Seviyeli Kalıtım

```
Kurucu A class
Kurucu B class
Kurucu C class

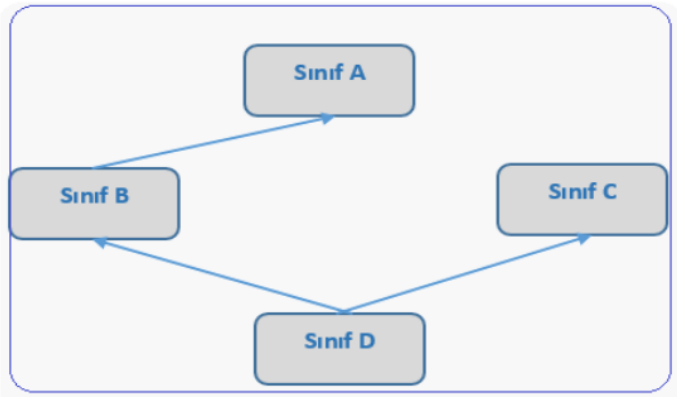
--------------------------------
Process exited after 3.538 seconds with return value 0
Press any key to continue . . .
```

Program 13.6'in ekran çıktısı

13.2.4. Hibrit Kalıtım

Hibrit kalıtım genellikle birden fazla kalıtım türünün birlikte kullanılmasıyla ortaya çıkar. Aşağıdaki gösterimde, melez bir kalıtım elde etmek için çoklu kalıtım (B, C ve D) ve çok seviyeli kalıtım (A, B ve D) yer almaktadır. Bu şekilde bir uygulamada birden fazla kalıtım türü yar alıyorsa bu kalıtım türüne Hibrit kalıtım denmektedir.

Hibrit Kalıtım Şeması:



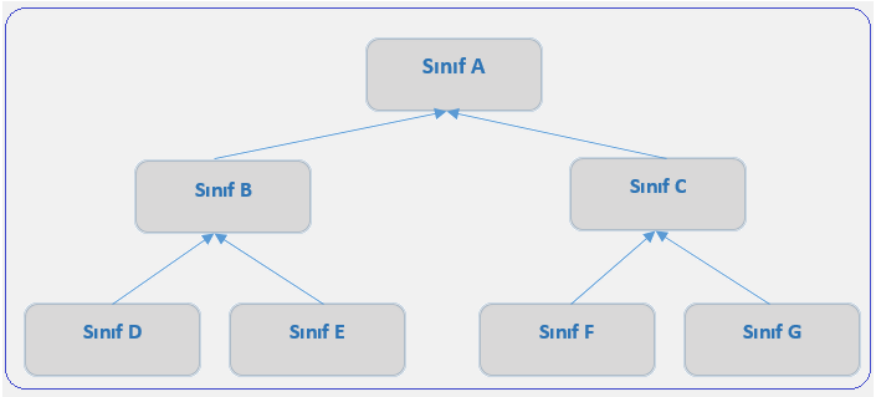

```
#include <iostream>
using namespace std;
class A
{
    public:
        A()
        {
            cout<<"kurucu A"<<endl;
        }
};
class C
{
    public:
        C()
        {
            cout<<"kurucu C"<<endl;
        }
};
class B:public A
{
    public:
        B()
        {
            cout<<"kurucu B"<<endl;
        }
};
class D:public B,public C
{
    public:
        D()
        {
            cout<<"Kurucu C"<<endl;
        }
};
int main() {
    D nl;
    return 0;
    }
```

program 13.7 Hibrit Kalıtım Örneği

13.2.5. Hiyerarşik Kalıtım

Hiyerarşik kalıtım, Aşağıdaki şemada gösterildiği gibi, birden fazla sınıfa tek bir temel sınıftan miras alınarak oluşturulur. Bundan dolayı kalıtım alma şeklinde hiyerarşik bir düzen vardır.

Hiyerarşik Kalıtım Şeması:




```cpp
#include <iostream>
using namespace std;
class sekil {
public:
int a,b;
void ata(int x,int y)
{
    a=x;
    b=y;
    }
};
class dikdortgen:public sekil{
public:
int dalan()
{
    int alan=a*b;
    return alan;
    }
};
class ucgen:public sekil{
    public:
    int ualan()
    {
        int alan=0.5*a*b;
        return alan;
    }
};
class kare:public sekil{
    public:
    int kalan()
    {
        int alan=a*a;
        return alan;
    }
};
int main() {
    dikdortgen d; ucgen u; kare k;
    d.ata(5,7);
    u.ata(4,7);
    k.ata(4,4);
    cout<<"Dikdortgenin Alani :"<<d.dalan()<<endl;
    cout<<"Ucgenin Alani      :"<<u.ualan()<<endl;
    cout<<"Karenin Alani      :"<<k.kalan()<<endl;
    return 0;
}
```

Program 13.8 Hiyerarşik Kalıtıma örnek

```
Dikdortgenin Alani :35
Ucgenin Alani      :14
Karenin Alani      :16

--------------------------------
Process exited after 2.573 seconds with return value 0
Press any key to continue .
```

Program 13.8'in ekran çıktısı

C ++ dili yukarıda açıklanan tüm bu kalıtım türlerini desteklemektedir. Buradan hareketle, C ++ 'ın kalıtım için çok güçlü bir desteği olduğunu, C ++ kullanarak birçok problemin daha etkin bir şekilde modelleyebileceğini söyleyebiliriz.

Bölüm Özeti

Bu bölümde Kalıtımın, Nesne yönelimli programlamanın temel özelliklerinden biri olduğunu, programcıların nesneye yönelik programlamanın bu özelliğini kullanarak mevcut bir sınıftan (temel sınıf) yeni bir sınıf (türetilmiş sınıf) oluşturabildiklerini öğrendik. Bu yöntem kullanılarak yazılmış olan kodun yeniden kullanılmasına izin verilmesi, bunun sonucu olarak sınıf kütüphanelerinin dağıtımının kolaylaşması, yazılım projelerinin geliştirilmesi ve bakımı sırasında zamandan ve paradan tasarruf edilmesi, bu teknik kullanılarak yazılan programların okunabilirliğinin artması gibi avantajların programlama sürecine kazandırıldığının farkına vardık.

Türetilmiş sınıfların, temel sınıftaki tüm özellikleri devraldığını ve ayrıca kendine özgü ek özelliklere sahip olabileceklerini öğrendik. Temel sınıflar türetilirken farklı erişim belirteçleri (public, private, protected) kullanılarak yazacağımız programlara farklı güvenlik özellikleri kazandırabileceğimizi gördük.

C++ programlama dilinde sınıfların türetilme şekillerine veya türetilmesi sırasında kaç sınıftan miras aldığına bağlı olarak, birbirinden farklı kalıtım türleri vardır. Bu farklı kalıtım türleri tekli kalıtım, çoklu kalıtım, çok seviyeli kalıtım, hiyerarşik kalıtım, Hibrit kalıtım şeklinde sıralanabilir. Bu bölümde farklı kalıtım türlerinin nasıl kullanılacaklarını öğrendik.

C ++ 'ın kalıtım için çok güçlü bir desteği olduğunu, C ++ kullanarak birçok problemin daha etkin bir şekilde modelleyebileceği konusundaki farkındalığımız arttı.

ÜNİTE SORULARI

1. Kalıtımda miras yoluyla özellikleri ve işlevleri alınan sınıfa verilen ad aşağıdakilerden hangisidir?

- Türetilmiş Sınıf
- Üye Değişken
- Sub sınıf
- Temel sınıf
- Üye fonksiyon

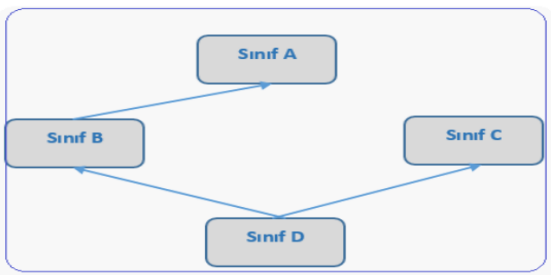
2. C++ 'ta temel sınıftan türetilmiş sınıf oluşturmak için yazılan söz dizimi aşağıdakilerden hangisi ile başlar?

- Temel sınıfın ismi
- Türetilecek sınıfın ismi
- Erişim Belirteci
- İki nokta üst üste işareti
- class bildirimi

3. Bir temel sınıftan kalıtım yoluyla türetilen sınıfa ait nesnelerle, temel sınıfın hangi üyelerine erişilemez?

- private üyelerine
- public üyelerine
- protected üyelerine
- Temel sınıfın hiçbir üyesine erişilemez.
- Temel sınıfın üyelerinin tamamına erişilebilir.

4.



Yukarıda verilen kalıtım şeması aşağıdaki kalıtım türlerinden hangisine aittir?

- a. Hiyerarşik kalıtım
- Hibrit kalıtım
- Çok seviyeli kalıtım
- Çoklu Kalıtım
- Tekli kalıtım

5. class C : public A, public B { }; şeklinde verilen söz dizimi aşağıda verilen kalıtım türlerinden hangisine aittir?

- Hiyerarşik kalıtım
- Hibrit kalıtım
- Çok seviyeli kalıtım
- Çoklu Kalıtım
- Tekli kalıtım
    -
    - CEVAP ANAHTARI1. d2. e3. a4. b5. d

14. DOSYALAR

Giriş

Verilerin elektronik ortamda saklanması ile ilgili çalışmaların başlama tarihi, programlama macerasının başlangıcına kadar geriye gider. Yirminci Yüzyılın başlarında kullanılmaya başlayan delikli kartları ve 1960'ların başlarında kullanılmaya başlanan manyetik bantları bir tarafa bırakarak, 1970'lerin başılarını bu konun başlangıç tarihi olarak sayabiliriz. Çünkü 1970'lerin başlarında harddiskler hayatımıza girmiş ve bugünkü anlmada veriler elektronik ortamda saklanmaya başlamıştı. Daha sonra 1980'lerde ilişkisel veritananı çalışmalarının başlaması, 1990'larda SQL dilinin ortaya çıkması, aynı dönemlerde object oriented veritabanı çalışmalarının yapılması, günümüzde çok büyük miktarlardaki verinin elektronik ortamlara kolay ve güvenli şekilde taşınmasını ve bu ortamlarda işlenmesini sağlayan teknolojilerin hayatımıza girmesini sağladı.

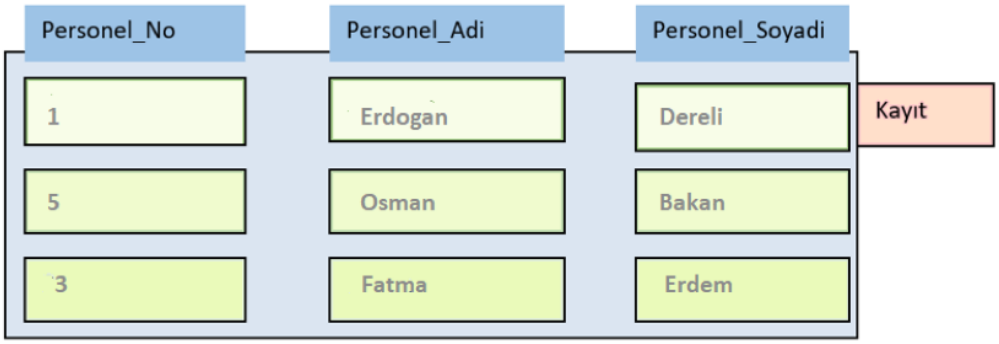
1970 'li yıllarda bilgisayar ortamında verilerin saklanması için dosya sistemleri kullanılmaktaydı (Bugün bazı özel alanlarda dosya sistemlerinin kullanılmasına az da olsa devam edilmektedir). Dosya kavramı için, "verinin ikinci seviye bellek ünitelerinde(Harddisk, flashdisk vb.) saklandığı yapıdır" şeklinde bir tanım yapabiliriz. Ya da dosya, veri saklama birimlerinde depolanan veri topluluklarına verilen isimdir de denebilir.

Yukarıdaki paragrafta birden çok sayıda *veri* kavramı geçti. Dosya sistemleri veya veri tabanlarından söz edilirken bu kavrama sık sık başvurulur ve çoğu kere de *veri* kavramının *bilgi* kavramı ile karıştığı görülür. Bunun önüne geçebilmek için aşağıda veri ve bilgi kavramlarının tanımını verelim.

Veri (data), Bilgisayarın alabildiği, işleyebildiği, sonuç üretebildiği ve saklayabildiği her şey veridir. Örneğin işlenmemiş deney sonuçları, gözlem sonuçları ve izlenimlerimiz birer veridir. Veri için anahtar kelime işlenmemiş olma durumudur diyebiliriz.

Bilgi ise, verinin işlenmesi sonucu ortaya çıkan her türlü gerçek, malumat ve kavrayıştır. Bugün teknolojinin geldiği noktada her türlü veri çok hızlı ve doğru bir biçimde işlenip bilgiye dönüştürülüyor ve karar verme noktasında olanlara sunuluyor. Yakın gelecekte bu konuda yeterli farkındalık geliştiremeyen bütün organizasyonlar ne yazık ki rekabet avantajlarını süratle kaybedeceklerdir.

Veri ve bilgi tanımından sonra şimdi de dosya nedir sorusunun cevabını netleştirmeye çalışalım. Bunun için, bir işyerinde çalışanların bilgilerinin tutulduğu bir dosya düşünelim. Vereceğimiz örnek oldukça basit ama konuya açıklık getirecek özellikte olsun. Dosyada her çalışanın personel numarası, adı ve soyadı yer alsın. Şekil 14.1 'de dosyanın yapısı gösterilmiştir.



Şekil 14.1 Dosya Örneği

Dosyada yer alan her satıra **kayıt** denir. Şekil 14.1'den de görülebileceği gibi kayıtlar birbirleri ile ilişkili alanlardan (field) oluşur. Ayrıca her bir kaydın, işyerinde çalışan bir kişiye ait olduğuna da dikkat ediniz.

Dosya sistemlerinin kullanılması sırasında bazı dezavantajlar ortaya çıkmıştır. Yeri gelmişken bu dezavantajların önemlilerini de burada sıralayalım:

Veri tekrarı: Aynı veri çeşitli dosyalarda birden fazla yer alabilmektedir buda sistemin hantallaşmasına neden olur. Mesela bir stok dosyasında stok numarası verisinin malzeme dosyasında, fatura dosyasında ve ambar girişi dosyasında yer alması gibi.

Verinin birkaç dosyada güncellemesi: Veri birden fazla dosyada tekrar edilebildiği için, verinin bir dosyada güncellenip diğerlerinde güncellenmemesi Veri Bütünlüğünün (Data Integrity) bozulmasına neden olabilir. Buna bağlı olarak birbiri ile çelişen raporlar üretilmektedir.

Belleğin tekrarlı bilgi nedeniyle israfı: Aynı verinin birden fazla dosya içinde bulunması nedeniyle kullanılan veri hard diskte fazla yer işgal edecek. Yani hard disk tekrarlı veriler için kullanılmış olacaktır.

Sadece belirli bir dilin kullanılması: Her programlama dilinin kendisine ait bir dosyalama sistemi vardır. Aynı işyerinde farklı diller kullanılarak geliştirilen programlar aracılığı ile oluşturulan dosyalardan ortak rapor üretilemeyeceği için birçok bilginin tekrar tekrar farklı dosyalara girilmesi gerekebilir.

Dosya sistemlerinin yukarıda sayılanlar ve benzeri dezavantajlarının ortaya çıkmasından dolayı verilerin saklanmasıyla ilgili teknolojiler bugün gelişerek ve dönüşerek çok farklı noktalara taşınmıştır. Ancak yine de dosya sistemleri yukarıda da belirtildiği gibi kullanılmaya devam edilmektedir.

C++ iki temel dosya tipini desteklemektedir. Bunlardan birincisi sıralı erişimli dosyalar, ikincisi ise doğrudan erişimli dosyalardır. Biz bu bölümde C++'ın desteklediği dosya tiplerinden sıralı erişimli dosyaları ele alacağız ve sıralı erişimli dosyalara kayıt yapılması, önceden dosyaya yapılmış kayıtların okunması, kayıt silinmesi ve kayıtların güncellenmesi konularını ele alacağız ve bu başlıkları birer örnek çözerek açıklayacağız.

14.1 C++'ta Dosya Giriş/Çıkış İşlemleri

C++ 'ta dosyalara giriş çıkış işlemleri üç adet sınıftan oluşturulan nesnelerle tanımlanır.

ofstream: Dosya oluşturmak ve dosyaya Yazmak için kullanılır

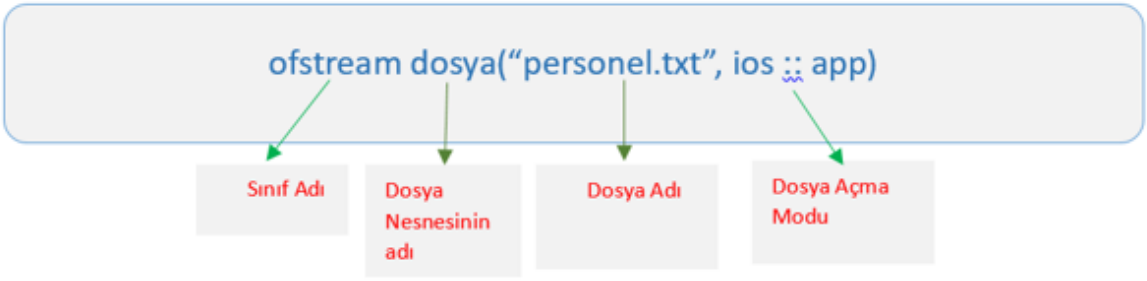
ifstream: Dosyadan okuma yapmak için kullanılır.

fstream: Dosya oluşturmak, dosyaya veri yazmak ve dosyadan veri okumak için kullanılır.

C++'ta program içerisinde dosya işlemleri yapabilmek için **fstream** dosyasının programa eklenmesi gerekmektedir.

14.2 C++' ta Sıralı Erişimli Dosyalara veri Yazmak

Verilerin dosyaya yazılabilmesi için önce dosyanın oluşturulması gerekmektedir. Bunun için *ofstream* sınıfı kullanılmaktadır.



- Dosyayı açtıktan sonra çıktılar **cout** veri yoluna gönderiliyor gibi dosya nesnesine gönderilmesi gerekir.

- ***dosya<<prsNo<<prs.Ad<<prs.Soyad;***

Dosya Açma Modları

C++ 'ta programcıların tercihine göre dosyalar üzerinde farklı işlemlerin yapılması için özel komutlar vardır. Bu komutlara ***mod*** denir. Tablo 14.1'de dosya açma modları verilmiştir.

Sıra No	Dosya Açma Modu
1	**ios::out** ofstream sınıfı için varsayılan mod. Dosyayı oluşturur ve yazma modunda açar
2	**ios::app** Dosya yoksa dosyayı oluşturur ve ekleme modunda açar. Dosya varsa dosyayı ekleme modunda açar.
3	**ios::in** ifstream için varsayılan mod. Dosyayı okuma modunda açar
6	**ios::trunc** Dosya varsa dosyayı siler
7	**ios::ate** Başlangıç konumunun dosyanın sonuna set edilmesini sağlar.
8	**ios::binary** Dosyayı binary modda açar

open() Fonksiyonu Kullanarak Dosya açmak

```
ofstream dosya;
dosya.open("personel.txt", ios :: app);
```

open() fonksiyonu ile dosya açılırken önce bir dosya ismi, sonra da isteğe bağlı olarak dosya açma modu kullanılmaktadır.

C++ 'ta Dosya Açma İşlemlerinde Hata Kontrolü

C++'ta Dört adet önceden tanımlı akış nesnesi vardır.

cin: Standart giriş
cout: Standart çıkış
cerr: Standart hata
clog: Bellekte saklanan cerr

```
if(!dosya)
{
  cerr<<"dosya açılamadı";
  exit(1)
                    }
```

C++ 'ta Dosya Kapamak

C++ 'ta Program bitmeden dosyayı kapatmak için **close()** fonksiyonu kullanılır.

```
dosya.close();
```

Program 14.1 : Bir iş yerinde çalışan personelin personel numarası, adı ve soyadının kaydedileceği sıralı erişimli dosya oluşturan ve bu dosyaya işyerinde çalışan personelin bilgilerini kaydeden C++ programını yazınız.

```
2  #include<fstream>
3  #include<string>
4  #include<conio.h>
5  #include<locale.h>
6  using namespace std;
7  struct pers{
8          string prsNo;
9          string prsAd;
10         string prsSoyad;
11             };
12 int main() {
13     setlocale(LC_ALL,"Turkish");
14     pers prs;
15     char ch='e';
16     ofstream dosya("personel.txt",ios::app);
17     if(!dosya)
18     {
19             cerr<<"dosya acilmadi"<<endl;
20             exit(1);
21             }
22     do
23     {
24     cout<<"Personel No       :"; cin>>prs.prsNo;
25     cout<<"Personel Adı       :"; cin>>prs.prsAd;
26     cout<<"Personel Soyadı    :"; cin>>prs.prsSoyad;
27     dosya<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
28     cout<<"veri girişine devam edecek misiniz  : "; ch=getche();cout<<endl;
29     }while(ch=='e');
30     dosya.close();
31     return 0;
32 }
```

Program 14.1 Sıralı erşimli dosyaya veri yazılması

Program 14.1'de önce programın çalışması için gerekli olan başlık dosyaları programa dâhil edildi. Arkasından string tipinde üç elemanı(prsNo, prsAd, prsSoyad) olan bir yapı tanımlandı. main() fonksiyonu içerisinde ***prs*** isimli bir yapı değişkeni tanımlandı. Daha sonra ***ofstream*** sınıfından ***dosya*** isimli bir nesne oluşturuldu ve parametre olarak ***personel.txt*** dosya ismi ve dosya açma modu olarak ta **ios::*app*** belirlendi.

ofstream dosya("personel.txt",ios::app);

program ilk defa çalıştırıldığında çalışma sırası bu satıra geldiği zaman disk üzerinde(Bir yol tanımlaması yapılmadığı için programın çalıştığı klasörde) personel.txt dosyası oluşturulacak ve ekleme modunda açılacaktır. Eğer program daha önceden çalıştırılmış ve personel.txt dosyası oluşmuş ise bu ifade daha önce oluşturulmuş olan dosyayı ekleme modunda açacaktır.

Programda 17-21. Satırlar arasındaki ifadelerle disk üzerinde dosya kontrolü yapılacak ve hata dönmez ise program 22. Satırdan çalışmaya devam edecektir. Eğer bu kontrolden hata dönerse programın çalışması sona erecektir.

22. ve 29. satırlar arasındaki do….while döngüsü içerisinde klavyeden ‘e’ tuşuna basıldığı sürece personel bilgisi girilecek ve her bir personele ait veri girildikten sonra;

dosya<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;

ifadesi verilerin personel.txt dosyasına yazılmasını sağlayacaktır.

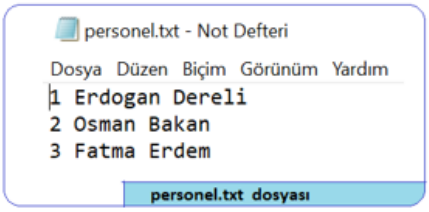
Veri girişi tamamlandığında kullanıcı ‘e’ tuşu dışında bir tuşa basarak programın çalışmasını sona erdirecektir.

Aşağıda, programın çalışması sırasındaki ekran görüntüsü ve programın çalışması tamamlandıktan sonra oluşan personel. txt dosyası verilmiştir.

```
Personel No       :1
Personel Adı      :Erdogan
Personel Soyadı   :Dereli
veri girişine devam edecek misiniz  : e
Personel No       :2
Personel Adı      :Osman
Personel Soyadı   :Bakan
veri girişine devam edecek misiniz  : e
Personel No       :3
Personel Adı      :Fatma
Personel Soyadı   :Erdem
veri girişine devam edecek misiniz  : h

--------------------------------
Process exited after 70.72 seconds with return value 0
Press any key to continue . . .
```

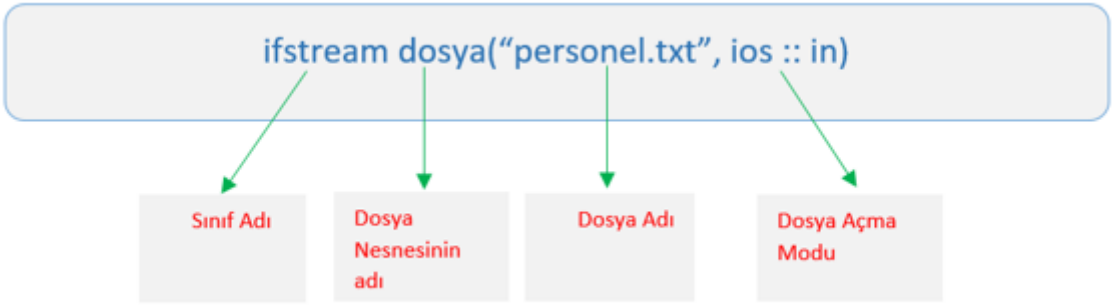
Program 14.1'in Veri Giriş Ekranı



personel.txt dosyası

14.3 C++’ ta Sıralı Erişimli Dosyalardan Veri Okumak

C++’ta sıralı erişimli dosyalardan veri okumak için ***ifstream*** sınıfı kullanılır. Daha önce oluşturulmuş olan dosyadan verileri okumak için *ifstream* sınıfından bir nesne oluşturulur. Dosya, oluşturulan *ifstream nesnesinin* varsayılanı modunda (ios::in) açılır. Dosya açıldıktan sonra girdiler cin veriyolundan okunuyormuş gibi oluşturulan nesneden okunur. Aşağıda ifstream sınıfından nesne oluşturulması ve dosyanın okumada modunda açılması gösterimiştir.



Program 14.2 : Program 14.1’de oluşturulan ve içerisine bir işyerinde çalışan personelin bilgilerinin kaydedildiği *personel.txt* dosyasını okuma modunda açan ve verilerin bu dosyadan okuyarak ekrana yazılmasını sağlayan programı yazınız?

```cpp
#include <iostream>
#include<fstream>
using namespace std;
struct pers{
      string prsNo;
      string prsAd;
      string prsSoyad;
        };
int main() {
    pers prs;
    ifstream dosya("personel.txt",ios::in);
    if(!dosya)
    {
        cerr<<"dosya acilmadi"<<endl;
        exit(1);
    }
    while(!dosya.eof())
     {
        dosya>>prs.prsNo>>prs.prsAd>>prs.prsSoyad;
        if(!dosya.eof())
           cout<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
            }
    dosya.close();
    return 0;
}
```

Program 14.2 Sıralı erişimli Dosyadaki verileri listelemek

Program 14.2'de önce programın çalışması için gerekli olan başlık dosyaları programa dâhil edildi. Arkasından string tipinde üç elemanı(prsNo, prsAd, prsSoyad) olan bir yapı tanımlandı. main() fonksiyonu içerisinde ***prs*** isimli bir yapı değişkeni tanımlandı. Daha sonra ***ifstream*** sınıfından ***dosya*** isimli bir nesne oluşturuldu ve parametre olarak ***personel.txt*** dosya ismi ve dosya açma modu olarak ta **ios::*in*** belirlendi.

ifstream dosya("personel.txt",ios::in);

program ilk defa çalıştırıldığında çalışma sırası bu satıra geldiği zaman disk üzerinde bulunan *personel.txt* dosyası okuma modunda açılacaktır.

Eğer program çalıştığı sırada *personel.txt* dosyası disk üzerinde bulunmuyorsa, programda 12. ve 16. satırlar arasında yazılan ifadelerden dolayı ekrana "*dosya açılmadı*" mesajını yazacak ve programdan çıkılacaktır.

Program 14.2'de 17. ve 22. Satırlar arasında while döngüsü oluşturulmuş ve bu döngünün dosya sonuna kadar çalışması sağlanmıştır. Program 14.2'de yazılan while döngüsü aşağıda gösterilmiştir.

```cpp
    while(!dosya.eof())
     {
        dosya>>prs.prsNo>>prs.prsAd>>prs.prsSoyad;
        if(!dosya.eof())
           cout<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
            }
```

Döngü içerisinde 19. Satırda yazılan ifade ile dosyadaki kayıtların sıra ile okunması, 21 satırda yazılan ifadeyle de okunan kaydın ekrana yazılması sağlanmıştır.

Aşağıda, program 14.1'in ekran çıktısı gösterilmiştir.

```
1 Erdogan Dereli
2 Osman Bakan
3 Fatma Erdem

--------------------------------
Process exited after 2.688 seconds with return value 0
Press any key to continue . . .
```

Program 14.2'nin Ekran Çıktısı

14.4. C++' ta Sıralı Erişimli Dosyalardan Kayıt Silmek

Program 14.3 'te, *personel txt* dosyasından, klavyeden personel numarası girilen kaydın silinebilmesi için önce *personel.txt* dosyası okuma modunda açılmış ve geçici olan bir dosya (*gpersonel.txt*) yazma modunda

açılmıştır. *personel.txt* dosyasından okunan kayıtlar, silinecek kayıt hariç *gpersonel.txt* dosyasına yazılmıştır. İşlem tamamlandıktan sonra *personel.txt* dosyası silinmiş ve *gpersonel.txt*'nin ismi *personel.txt* olarak değiştirilmiştir. Bu işlemlerin sonunda dosyadan klavyeden girilen kaydın silinmesi sağlanmıştır.

```
#include <iostream>
#include<fstream>
#include<conio.h>
#include<string>
using namespace std;
struct pers{
        string prsNo;
        string prsAd;
        string prsSoyad;
            };
int main() {
    string s,x;
    pers prs;
    cout<<"Bilgileri Silinecek Personelin Numarasi  :";
    cin>>s;

    ifstream dosya("personel.txt",ios::in);
    if(!dosya)
    {
        cerr<<"Personel dosyasi acilmadi\n";
        exit(1);
    }
    ofstream dosya1("gpersonel.txt",ios::app);
    if(!dosya)
    {
        cerr<<"Personel dosyasi acilmadi\n";
        exit(1);
    }

    while(!dosya.eof())
     {
        dosya>>prs.prsNo>>prs.prsAd>>prs.prsSoyad;
        x=prs.prsNo;
        if((s!=x)&&(!dosya.eof())){
dosya1<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
        }

    }
    dosya.close();
    dosya1.close();
            remove("personel.txt");
            rename("gpersonel.txt","personel.txt");

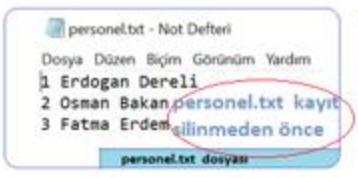
    return 0;
}
```

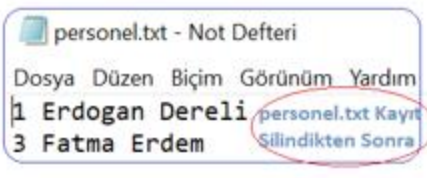
Program 14.3 Sıralı Erişimli Dosyadan Kayıt Silmek

```
Bilgileri Silinecek Personelin Numarasi :2

--------------------------------
Process exited after 10.01 seconds with return value 0
Press any key to continue .
```

Personel Numarası 2 olan kayıt silindi





14.5. C++' ta Sıralı Erişimli Dosyalarda Kayıt Güncellemek

Program 14.4, *personel.txt* dosyasındaki kayıtların güncellenmesi için yazılmıştır. Bu programda da program 14.3'e benzer şekilde kayıtlar *personel.txt* 'den sırayla okunarak gpersonel.txt 'ye yazılması sağlanmış ve sıra güncellenmesi gereken kayda geldiğinde güncellenecek kayıtla ilgili bilgiler kullanıcıya ekranda gösterilmiş, programda kullanıcının güncel bilgileri klavyeden giriş yapabilmesi için düzenleme yapılmıştır. Kullanıcının güncellenecek kayıtla ilgili yeni bilgilerin girişini yapmasından sonra bu bilgilerin de *gpersonel.txt* 'ye yazılması sağlanmıştır. *personel.txt* 'deki tüm kayıtlar, güncellenen kayıt dahil *gpersonel.txt* 'ye yazılması tamamlandıktan sonra *personel.txt* silinmiş ve *gpersonel.txt* 'nin ismi *personel.txt* olarak değiştirilerek güncelleme işlemi tamamlanmıştır.

```cpp
#include <iostream>
#include<fstream>
#include<conio.h>
#include<string>
using namespace std;
struct pers{
        string prsNo;
        string prsAd;
        string prsSoyad;
            };
int main() {
    string s,x;
    pers prs;
    cout<<"Bilgileri Guncellenecek Kisinin Numarasi :";
    cin>>s;

    ifstream dosya("personel.txt",ios::in);
    if(!dosya)
    {
        cerr<<"Personel dosyasi acilmadi\n";
        exit(1);
    }
    ofstream dosya1("gpersonel.txt",ios::app);
    if(!dosya)
    {
        cerr<<"Personel dosyasi acilmadi\n";
        exit(1);
    }
    while(!dosya.eof())
    {
       dosya>>prs.prsNo>>prs.prsAd>>prs.prsSoyad;
       x=prs.prsNo;
       if((s!=x)&&(!dosya.eof())){
           dosya1<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
       }
       if((s==x)&&(!dosya.eof()))
       {
           cout<<"Personel No       :"; cout<<prs.prsNo<<endl;
           cout<<"Personel Adi      :"; cout<<prs.prsAd<<endl;
           cout<<"Personel Soyadi   :"; cout<<prs.prsSoyad<<endl;
           cout<<prs.prsNo<<" numarali personelin guncel bilgilerini giriniz "<<endl;
           cout<<"Personel No       :"; cout<<prs.prsNo<<endl;
           cout<<"Personel Adi      :"; cin>>prs.prsAd;
           cout<<"Personel Soyadi   :"; cin>>prs.prsSoyad;

           dosya1<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
       }
    }
    dosya.close();
    dosya1.close();
            remove("personel.txt");
            rename("gpersonel.txt","personel.txt");

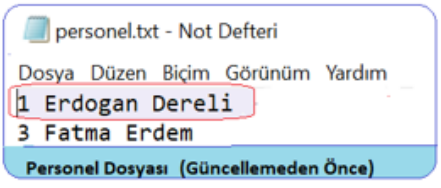
    return 0;
}
```

Program 14.4 Kayıt Güncelleme Programı

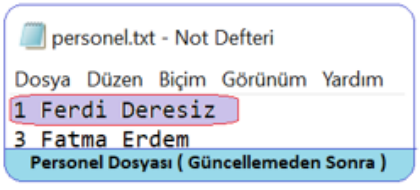
```
Bilgileri Guncellenecek Kisinin Numarasi        :1
Personel No       :1
Personel Adi      :Erdi
Personel Soyadi   :Dereli
1 numarali personelin guncel bilgilerini giriniz
Personel No       :1
Personel Adi      :Ferdi
Personel Soyadi   :Deresiz

--------------------------------
Process exited after 28.61 seconds with return value 0
Press any key to continue . . .
```

Program 14.4'ün ekran görüntüsü



Personel Dosyası (Güncellemeden Önce)



Personel Dosyası (Güncellemeden Sonra)

Program 14.5: Bu ünitede 14.2'de text dosyaya kayıt ekleme, 14.3'te text dosyadan okunan kayıtların listelenmesi, 14.4'te text dosyadan kayıtların silinmesi ve 14.5'te text dosyadaki kayıtların güncellenmesi ile ilgili ayrı ayrı programlar yazılmıştır. Program 14.5'te yukarıda yazılan programları birer fonksiyon haline getiriniz, ek olarak menü isimli bir fonksiyon oluşturunuz ve programda switc –case yapısı kullanarak yukarıda sayılan bütün bu özelliklere kullanıcının tek ekrandan ulaşmasını sağlayınız?

```
#include <iostream>
#include <string>
#include <fstream>
#include <conio.h>
using namespace std;void goster();void ekle();void kayitsil();void guncelle();void menu();
struct pers{
      string prsNo;
      string prsAd;
      string prsSoyad;
          };
main(){
    while(true){
        char secim;
        menu();
        cout<<endl<<"Seciminiz : ";cin>>secim;
        switch(secim){
            case '1':
                goster();
                break;
            case '2':
                ekle();
                break;
            case '3':
                kayitsil();
                break;
            case '4':
                guncelle();
                break;
            case '5':
                system("cls");
                break;
            default:
                exit(1);
       }   }
    return 0;}
```

Program 14.5 Liste, Ekle, Sil, Güncelle toplu program

```
 _____________________________
+[1] Personel Listesi         +
|[2] Personel Ekle            |
|[3] Personel Sil             |
|[4] Personel Guncelle        |
|[5] Ekrani Temizle           |
|[C] Cikis                    |
+_____________________________+

Seciminiz : 1
1 Ferdi Deresiz
3 Fatma Erdem
2 Hasan Kimsesiz

 _____________________________
+[1] Personel Listesi         +
|[2] Personel Ekle            |
|[3] Personel Sil             |
|[4] Personel Guncelle        |
|[5] Ekrani Temizle           |
|[C] Cikis                    |
+_____________________________+

Seciminiz :
```

Program 14.5'in Ekran Çıktısı

```
36  void goster(){
37      pers prs;
38      ifstream dosya("personel.txt",ios::in);
39      if(!dosya)
40      {
41          cerr<<"dosya acilmadi"<<endl;
42          exit(1);
43      }
44      while(!dosya.eof()){
45          dosya>>prs.prsNo>>prs.prsAd>>prs.prsSoyad;
46          if(!dosya.eof())
47              cout<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
48              }
49      dosya.close();
50  }
```

Program 14.5 Listeleme Fonksiyonu

```
51  void ekle(){
52      pers prs;
53      char ch='e';
54      ofstream dosya("personel.txt",ios::app);
55      if(!dosya)
56      {
57              cerr<<"dosya acilmadi"<<endl;
58              exit(1);
59              }
60      do
61      {
62      cout<<"Personel No       :"; cin>>prs.prsNo;
63      cout<<"Personel Adi      :"; cin>>prs.prsAd;
64      cout<<"Personel Soyadi   :"; cin>>prs.prsSoyad;
65
66      dosya<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
67      cout<<"veri girisine devam edecek misiniz  : "; ch=getche();cout<<endl;
68      }while(ch=='e');
69      dosya.close();
70  }
```

Program 14.5 Kayıt Ekleme Fonksiyonu

```
71  void kayitsil(){
72      string s,x;
73      pers prs;
74      cout<<"Bilgileri Silinecek Personelin Numarasi  :";
75      cin>>s;
76
77      ifstream dosya("personel.txt",ios::in);
78      if(!dosya)
79      {
80          cerr<<"Personel dosyasi acilmadi\n";
81          exit(1);
82      }
83      ofstream dosya1("gpersonel.txt",ios::app);
84      if(!dosya)
85      {
86          cerr<<"Personel dosyasi acilmadi\n";
87          exit(1);
88      }
89      while(!dosya.eof())
90       {
91          dosya>>prs.prsNo>>prs.prsAd>>prs.prsSoyad;
92          x=prs.prsNo;
93          if((s!=x)&&(!dosya.eof())){
94              dosya1<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
95          }
96
97      }
98      dosya.close();
99      dosya1.close();
100             remove("personel.txt");
101             rename("gpersonel.txt","personel.txt");
102     }
```

Program 14.5 Kayıt Silme Fonksiyonu

```
103 void guncelle(){
104     string s,x;
105     pers prs;
106     cout<<"Bilgileri Guncellenecek Kisinin Numarasi :";
107     cin>>s;
108     ifstream dosya("personel.txt",ios::in);
109     if(!dosya)
110     {
111         cerr<<"Personel dosyasi acilmadi\n";
112         exit(1);
113     }
114     ofstream dosya1("gpersonel.txt",ios::app);
115     if(!dosya)
116     {
117         cerr<<"Personel dosyasi acilmadi\n";
118         exit(1);
119     }
120     while(!dosya.eof())
121      {
122         dosya>>prs.prsNo>>prs.prsAd>>prs.prsSoyad;
123         x=prs.prsNo;
124         if((s!=x)&&(!dosya.eof())){
125             dosya1<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
126         }
127         if((s==x)&&(!dosya.eof()))
128         {
129             cout<<"Personel No        :"; cout<<prs.prsNo<<endl;
130             cout<<"Personel Adi       :"; cout<<prs.prsAd<<endl;
131             cout<<"Personel Soyadi    :"; cout<<prs.prsSoyad<<endl;
132             cout<<prs.prsNo<<" numarali personelin guncel bilgilerini giriniz "<<endl;
133             cout<<"Personel No        :"; cout<<prs.prsNo<<endl;
134             cout<<"Personel Adi       :"; cin>>prs.prsAd;
135             cout<<"Personel Soyadi    :"; cin>>prs.prsSoyad;
136             dosya1<<prs.prsNo<<" "<<prs.prsAd<<" "<<prs.prsSoyad<<endl;
137         }   }

138     dosya.close();
139     dosya1.close();
140             remove("personel.txt");
141             rename("gpersonel.txt","personel.txt");
142     }
```

Program 14.5 GÜncelleme Fonksiyonu

```
143  void menu(){
144       cout<<" _____"<<endl;
145       cout<<"+[1] Personel Listesi          +\n";
146       cout<<"|[2] Personel Ekle            |\n";
147       cout<<"|[3] Personel Sil             |\n";
148       cout<<"|[4] Personel Guncelle        |\n";
149       cout<<"|[5] Ekrani Temizle           |\n";
150       cout<<"|[C] Cikis                    |\n";
151       cout<<"+_____+"<<endl;
152  }
```

Program 14.5 menu Fonksiyonu

Sıra Sizde: Program 14.5'te yukarıdaki özelliklere ek olarak personelin ev, iş ve cep telefonlarının ve mail adreslerinin de kaydedilmesini sağlayarak programa bir telefon rehberi programı özelliği katınız.

Sıra Sizde : Program 14.5 'i yapı yerine sınıf ve nesneleri kullanarak yeniden yazınız?

Bölüm Özeti

Bu bölümde C++ akışlarını kullanarak dosya bağlantılı faaliyetlerin nasıl yürütüldüğü öğrendik. Bu faaliyetler dosyaların oluşturulması, dosyalara veri kaydedilmesi, hataların kontrol edilmesi, Dosyalardan kayıt silinmesi ve dosyalardaki kayıtların güncellenmesi başlıkları altında incelendi. Son olarak ta bütün bu operasyonların bir menü altın da nasıl toplanacağı, programın fonksiyonlar halinde nasıl yazılacağı gösterildi.

ÜNİTE SORULARI

1. C++'ta aşağıdaki sınıflardan hangisi ile oluşturulan nesneler kullanılarak sıralı erişimli dosya oluşturulabilir?

- istream
- ifstream
- istrigstream
- ofstream
- iostream

2. C++ 'ta aşağıdaki sınıflardan hangisi ile oluşturulan nesneler kullanılarak sıralı erişimli dosyadan okuma yapılabilir?

- ostream
- ifstream
- ostrigstream
- ofstream
- iostream

3. C++'ta aşağıdaki sınıflardan hangisi ile oluşturulan nesneler kullanılarak sıralı erişimli dosyalara veri eklenebilir?

- istream
- ifstream
- istrigstream
- ofstream
- iostream

4. C++'ta aşağıda verilen dosya açma modlarından hangisi ile dosyalar, okuma modunda açılır?

- İn
- app
- out
- ate

- nocreate

5. C++'ta aşağıda verilen dosya açma modlarından hangisi ile dosyalar, ekleme modunda açılır?

- in
- app
- out
- ate
- nocreate
    -
    - CEVAP ANAHTARI1. d2. b3. d4. a5. b