PROGRAMLAMAYA GİRİŞ

Bölüm 1

1.1. GİRİŞ

Bilgi dünyasında yoğun bir şekilde kullandığımız bilgisayarlar ile ortaya çıkan programlar günlük hayatta biz insanların işlerinde büyük kolaylıklar sağlamaktadırlar. Alış-veriş merkezlerindeki kasiyerlerin kullandığı paket programlar, ticari ve sanayi alanlardaki makinelerin bilgisayar destekli kullanılan programları, eğitim alanlarında kullanılan otomasyonlar ve bilgisayarların kullanıldığı her alanda sayamadığımız bir çok programlar bilgisayar programcıları tarafından programlama dilleri kullanılarak yazılırlar. Otomasyon yazılımları, bir matematiksel işlemin veya bilimsel bir hesaplamanın bilgisayarla çözülmesi hızlı, daha kolay ve doğru yapılmasını sağlar.

Programlama dilleri zaman içerisinde gelişmiş, yeni versiyonlar ile değişmiş, kaybolmuş veya yenileri çıkmıştır. Bu nedenle programlama bilgisini asla bir programlama diline bağlı tutmamak gerekir. Eğer programlama mantığı oldukça iyi gelişirse, algoritmaları kolay kurup algılayarak, çok karmaşık sorunlar üzerinde fikir yürütülüp çözüm üretilebilir. Daha sonra da bilinen uygun bir programlama dilinin formatına uygun yazmak gerekecektir. Programlama temeli ve mantığı kavranıldıktan sonra, çoğunlukla 1-2 hafta gibi bir sürede bir programlama dilini orta düzeyde öğrenilebilir.

BÖLÜM -1- PROGRAMLAMAYA GİRİŞ

Bölümün Genel Amacı: Programlamayı genel olarak tanıma ve programlama mantığını kavrayarak, algoritma ve akış diyagramlarını kullanma ve bunları açıklama.

Bölümün Davranışsal Amaçları: Kitabınızın bu bölümünü başarıyla bitirip, uygulamaları yapıp, değerlendirme sorularına doğru cevap verdiğiniz taktirde, bölüm sonunda;

- Program kavramını açıklamanız,
- Programlama kavramını açıklamanız,
- Programlama türlerini açıklamanız,
- Program geliştirme sürecindeki adımları ve işlevlerini açıklamanız,
- Algoritma kavramını açıklamanız ve algoritmalar oluşturmanız,
- Programların akış diyagramını çizmeniz,
- Akış diyagramlarındaki şekilleri ve görevlerini açıklamanız,
- Programlama dillerini tanımanız ve temel işlevlerini öğrenmeniz,
- Bir program yazarken kullanacağınız dili bilmeniz beklenmektedir.

Değerlendirme: Modül sonundaki uygulamaları yapmanız sonuçları uygulama raporu ile karşılaştırmanız ve değerlendirme sorularına en az % 75 düzeyinde doğru cevap verebilmeniz gerekmektedir.

1.2. PROGRAMLAMA NEDİR?

Bilgisayarların isteğe uygun özel işlemler yapması için programlanması gerekir. Örneğin, bir şirkette kullanılan stok uygulaması, sipariş uygulaması yada değişik iş takipleri, hastane otomasyonları yada eğitim kurumlarının kullandığı öğrenci otomasyonları gibi. Programlamaya çok fazla örnek vermek mümkündür.

Program : Günlük hayattaki bir sorunu bilgisayarla çözmek, rutin işlemleri kolaylaştırmak için bilgisayarların isteğe uygun olarak özel bir takım işlemleri gerçekleştirmesi için programlanması gerekmektedir. İşte yazılan bu yazılımlar ile ortaya çıkan ürün bir programdır. Bilgisayar oyunu, muhasebe işlem programları ve ticari şirketlerde kullanılan paket programlar birer programdır.

Programlama Dili : Bilgisayarda çözülecek bir sorun için çözümün bilgisayara adım adım yazılmasını sağlayan biçimsel kuralları olan ve bu kurallara sıkı sıkıya bağımlılığı gerektiren bir tanımlar kümesidir. Yani, programcı ile bilgisayar arasında bir tercüman durumundadır.

1.3. PROGRAMLAMANIN TARİHİ

Oldukça eskiye dayanan programlamanın tarihine baktığımızda fazla kodlanmış makine komutları bulunmamaktaydı. 1940 – 1950 yılları arasında fazla kodlanmış makine komutlarıyla yazılan programlar, programcılar açısından oldukça zordu. Hızın düşük olması, bellek yetersizliği, işlem sayısının az olması bilgisayarın kullanım alanını sınırlıyor ve işlemler basit bir uygulamadan ileri gidemiyordu. Tabi bu durum mikroişlemciler ile alakalı idi. Rekabet halinde olan firmalar işlemcilerin mimarilerini geliştirip hızlarını arttırdıkça, işlemcilerin işleyebileceği komut sayıları artmış ve komut setleri daha kullanışlı hale gelmiştir.

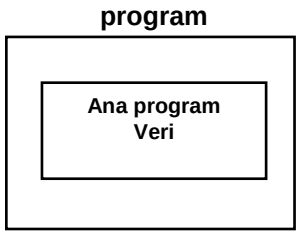
İlk programlar makine dili ile hazırlanıyordu. Makine dili de bir programlama dilidir ama makine dili ile program yazmak çok zahmetli bir iştir. Çünkü makine dilinde sıfırlar (0) ve birler (1) vardır. Yani işlemlerde DOĞRU (1) yada YANLIŞ (0) durumlarına göre hareket edilmektedir. Ayrıca, makine dili programları anlaşılması zor olan ve tamamıyla donanıma hitap eden programlardır. Günümüze baktığımızda geliştirilen üst düzey programlama dillerini kullanırken makine dilini bilmeye bile gerek yoktur.

Makine diline yakın Assembly programlama dilinde mikroişlemcilerin anlayacağı assebly kodları kullanılır. Bu ham şeklinde bulunan komutlar (MOV, ADD, PUSH gibi) mikroişlemcinin belli bir işlevi yerine getirmesini sağlamaktadır. Bu komutlara **mnemonic** adı verilir. Üst seviye programlama dillerinin geliştirilmesiyle programlar daha anlaşılabilir komutlarla yazılmaya başlanmıştır (Print, Read, Display, Circle, Get, vb). 1980'li yıllarda üst seviye programlama dilleri yaygınlaşarak, programlar belli bir dilin yapısı içinde tasarlanmış ve yazılmıştır.

1.4. PROGRAMLAMA TÜRLERİ

1.4.1 Yapısal Programlama

Programlama dilleriyle ilgilenenler kod yazmaya genellikle küçük ve basit kodlar yazarak başlarlar. Bu kodlar sadece bir "ana (main)" bloğundan oluşur. Bu blok içerisindeki komut ve deyimler programın tümünde tanımlı olan "global" verileri kullanırlar.

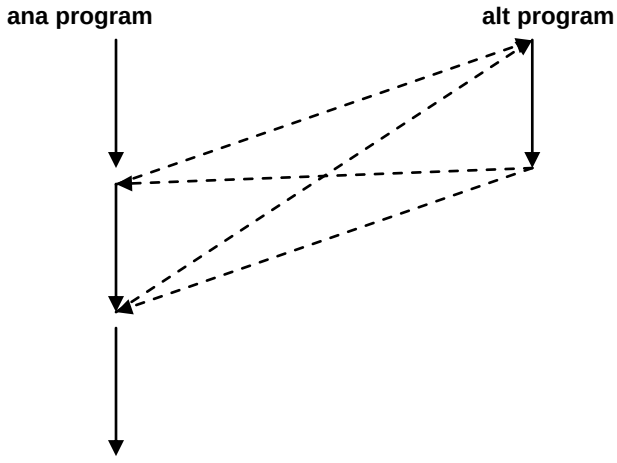


Şekil 1.1 Yapısal olmayan programlama

Şekilde 1.1'de görüldüğü gibi yapısal olmayan programlama tekniği kullanılarak yazılan kodlarda, ana program bloğu, global olarak tanımlanmış veriler üzerinde işlem yürütür.

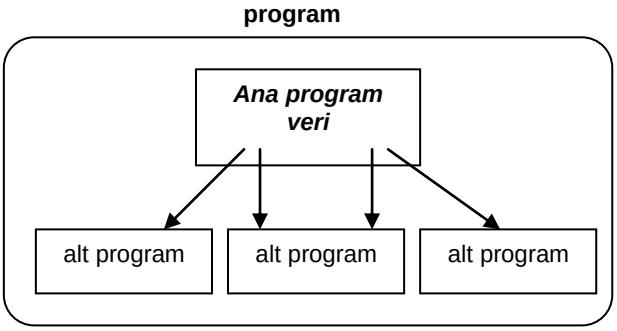
Bu programlama tekniği oldukça büyük programlarda büyük dezavantajlara sahiptir. Örneğin aynı komut ve deyimler bu teknikte yazılan kodlar içerisinde defalarca tekrarlanmak zorunda kalabilir. Aynı zamanda bu teknikte yazılan kodların okunabilirliği ve anlaşılabilirliği yazılan kod miktarı arttıkça zorlaşır. Dolayısıyla kod yazarken hata yapma olasılığı fazladır ve hata ayıklama (**debugging**) işlemi oldukça zordur. Bu nedenle de doğal olarak yazılan programın güvenirliği de düşük olacaktır.

Yapısal programa tekniğinde ise altprogramlar (**procedures**) ve fonksiyonlar (**functions**) kullanılır. Böylece program akışının kontrolünde büyük kolaylıklar sağlanmış olur.



Şekil 1.2. Alt programların çağrılması

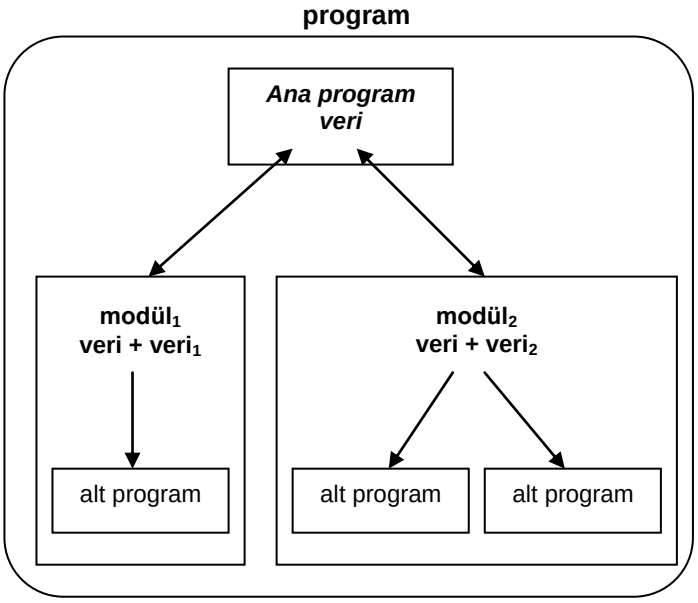
Şekil 1.2'deki program akışı esnasında, ana program içerisinden altprogramlar çağırılıyor. Bu altprogram çalıştırıldıktan sonra, program akışı tekrar geri dönecek ve programın işleyişi kaldığı noktadan devam edecektir. Yapısal programlama tekniğinde kod içerisinde aynı altprogram defalarca çağırılabilir. Böylece gereksiz kod tekrarıda ortadan kalkmış olur.



Şekil 1.3. Yapısal Programlama Tekniği

1.4.2 Modüler Programlama

Modüler programlama tekniğinde belli altprogramlar ayrı ayrı modüller içinde gruplandırılır. Her modül içerinde ana program içinde tanımlı global değişkenler geçerlidir. Aynı zamanda her modül kendi verisine de sahiptir.

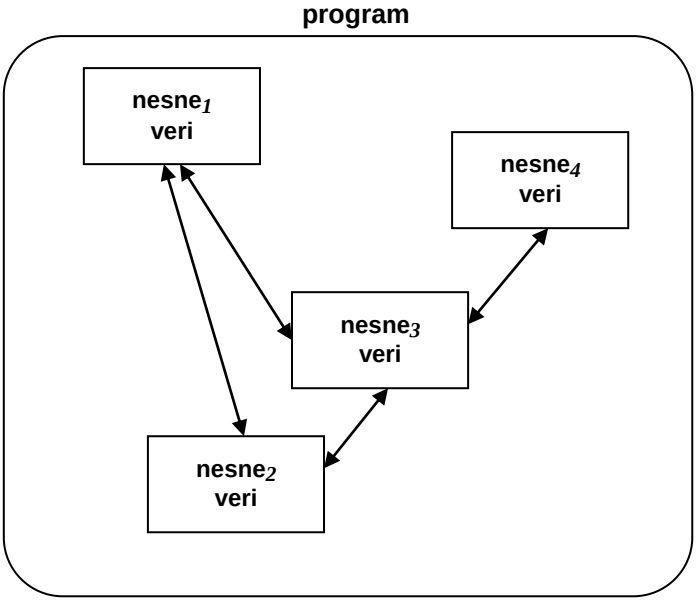


Şekil 1.4. Modüler Programlama

1.4.3 Nesne Tabanlı Programlama

Nesne tabanlı programlama (Object-Oriented Programming, OOP) dilleri 1980'li yıllarda C++ ile ortaya çıkmıştır. Nesneye yönelik programlama, programcının kendi sınıfını ve nesnesini oluşturup bunun üzerinde işlemler yapmasına olanak sağlar. Bu programlama sisteminin zor olması nedeniyle, çok sayıdaki nesneler önceden programcıya hazır bir şekilde sunulur. Günümüzdeki programcılar ise nesneye yönelik olan ve Visual özellikler içeren sürümleri kullanmaktadır. Visual C++, C++ Builder, Delphi, Java, Visual Basic vb. gibi örnekler sayılabilir.

Bu teknikte nesneler birbirlerine mesaj göndererek etkileşim içinde bulunurlar. Nesne yönelimli programlama tekniğinde açık bir biçimde altprogramları çağırmak yerine, direkt olarak ilgili nesneye bir mesaj gönderilir. Nesne kendine gelen mesajı alır ve öncelikle nesnenin bir örneğini (kopyasını) oluşturur. Bu kopya gerekli işlemleri yaptıktan sonra kendini yok eder. Tüm bu işlemlerden nesnenin kendisi sorumludur.



Şekil 1.5. Nesneye Yönelik Programlama

1.4.4. Olay Temelli Programlama

Bilgisayarda bir uygulamayı kullanırken, ekran üzerinde bir nesnenin hareket ettirilmesi veya bir tuşa basılarak düğmelerin seçilmesi işlemleri bir olaya dayanır. Olay temelli programlama (Event – Driven Programmming, EDP) kullanıcıların yapacakları işlemlere göre programın hareket etmesi temeline dayanır. Daha önceki programlarda ekrana gelen menülerde daha çok klavyenin fonksiyon ve yön tuşları aktif olarak kullanılıyordu. Olay temelli programlamada farenin menüler ve pencereler üzerinde oldukça yaygın olarak kullanılması sağlanmıştır. Bu şekilde işlemler daha basit, fonksiyonel ve daha kullanışlıdır.

1.5. PROGRAM GELİŞTİRME SÜRECİ

1.5.1. İyi Bir Programın Nitelikleri

Yazılan bilgisayar programlarının basit bir işlevi yerine getirmesi ona iyi bir program özelliği kazandırmaz. Yazılması düşünülen bir programın bir çok açıdan belli niteliklere sahip olması gerekmektedir. Bu nitelikleri şu şekilde sıralamak mümkündür.

- Estetik olarak görselliği ön plana çıkmalıdır.
- Kullanıcı açısından kullanımı kolay olmalıdır.
- İşlem ve hesaplamaları doğru yapmalıdır.
- Hızlı çalışmalıdır.
- Kolayca değiştirilebilmeli ve güncellenebilmelidir.
- Fazla kod yazılmadan etkin bir kodlamaya sahip olmalıdır.
- Yaygın kullanılan işletim sistemlerinde çalışabilmelidir.
- Büyük programlar için çoklu kullanıcı desteği olmalıdır.
- Ticari yazılan programlar ise iyi belgelenerek, lisanslı satılmalıdır.

1.5.2. Program Tasarlama

Bir yazılım geliştirirken takip edilmesi gereken adımlar şunlardır:

1. **Gereksinimlerin belirlenmesi :** Problemin tanımı verilir.
2. **Analiz :** Problemin çözümü için gerekli tüm girdi ve çıktılar analiz edilmelidir.
3. **Dizayn :** Problemin çözümünde kullanılacak uygun algoritmanın adım adım tanımlanması yapılmalıdır.
4. **Akış Diyagramı:** Algoritmaya göre uygun akış diyagramı çizilmelidir.
5. **Kod Yazımı :** Algoritmanın herhangi bir programlama dilinde yazılarak kaynak dosyanın hazırlanması gerekir.
6. **Test :** Bu basamakta ise yazılan programın bölümleri ve tamamı çalışır halde test edilir.
7. **Doğrulama :** Programın örnek girdilerle doğru çıktı ürettiği gözlenmelidir.
8. **Bakım :** Yazılan programda bulunan hatalar ayıklanır veya gerekli güncellemeler yapılır.
9. **Belgeleme *:*** Yazılan program için belgeleme yapılarak, toplu çoğaltmalara karşı engelleme konulur.

1.5.3. Arabirim Geliştirme ve Programın Görünüşü

Nesne tabanlı programlama dillerine baktığımızda iyi bir arabirim geliştirmek için bir çok nesnenin var olduğu görülmektedir. Araç çubukları, durum çubukları, menüler, iletişim kutuları gibi uygulamalar eklenebilir. Ayrıca, fare ve klavye işlemlerini eklemek için ayrıca ek bir bölüm yazmaya gerek yoktur.

Program yazma ve arabirim geliştirme işlemi, iyi bir programcılığın yanı sıra büyük bir sanatta gerektirir. “Program yapmak bir sanattır” ifadesini kullanmak pek de yanlış olmayacaktır. Arabirim geliştirme işleminde programın kullanılabilirliği, sadeliği, uyumluluğu ve grafik değerleri dikkate alınarak hazırlanmalıdır. Özellikle nesne tabanlı programlama türlerini kullanırken görsellik büyük ölçüde ön plana çıkmaktadır. Renklendirme, yazı tiplerinin biçimleri, gölgeleme, parlaklık, menülerin resim programlarıyla süslenmesi gibi konuları da dikkate almak gerekmektedir.

1.6 ALGORİTMALAR ve AKIŞ DİYAGRAMLARI

1.6.1 Algoritma Nedir?

Algoritma; Belirli bir görevi yerine getiren sonlu sayıdaki işlemler dizisidir. Başka bir deyişle; Bir sorunu çözebilmek için gerekli olan sıralı mantıksal adımların tümüne denir. Bu kavram M.S. 9.yy da, İranlı Musaoğlu Horzumlu Mehmet'in (Alharezmi adını araplar takmıştır) problemlerin çözümü için genel kurallar oluşturması ile ortaya çıkmış olup. Algoritma Alharezmi'nin Latince okunuşudur.

Peki bilgisayarda çözülecek bir sorunu nasıl algoritma ile ifade ederiz? Bunun için öncelikle bir sorun tanımlayalım. Başlangıç ta basit olması için şöyle bir problem üzerinde düşünelim: Bilgisayara verilecek iki sayıyı toplayıp sonucu ekrana yazacak bir program için algoritma geliştirmek isteyelim. Sorun son derece basit ancak sistem tasarımının net yapılabilmesi için sorun hakkında anlaşılamayan tüm belirsiz noktalar açıklığa kavuşturulmalıdır. Örneğin sayılar bilgisayara nereden verilecek, Klavye, Dosya veya belki başka bir ortam. Bu ve buna benzer soru ve tereddütleriniz varsa sorunun sahibine bunları sormalı ve sistem analizi yapmalısınız.

Sonra bulacağımız çözümü algoritma haline dönüştürebiliriz.

1. BAŞLA
2. A sayısını oku
3. B sayısını oku
4. TOPLAM=A + B işlemini yap
5. TOPLAM değerini ekrana yaz
6. SON

Bir başka örnek; Klavyeden girilecek iki sayıdan büyük olanından küçük olanını çıkarıp sonucu ekrana yazacak program için bir algoritma geliştirelim.

1. BAŞLA
2. A sayısını oku
3. B sayısını oku
4. Eğer A büyüktür B ise SONUC=A-B değilse SONUC=B-A
5. SONUC değerini ekrana yaz
6. SON

Bu algoritmalar oldukça basit algoritmalar olup algoritma kavramının yerleşmesini sağlayan örneklerdir.

Algoritmalar doğal dille yazılabileceği için fazlaca biçimsel değildir. Algoritmalar belli bir kurallar bütününü ifade ettiği için bir algoritmada aşağıdaki ifadelerin mutlaka doğrulanması gereklidir;

- Netlik
- Etkinlik
- Sonluluk
- Giriş/Çıkış Bilgileri

NETLİK

Algoritmada bulunan anlatım satırları kesin olmalıdır. Kesin olmayan anlatımlar algoritmada bulunmamalıdır. Başka bir deyişle her işlem (komut) açık olmalı ve farklı anlamlar içermemelidir.

Örnek;

```
z←x + y
sayı←sayı + 1
```

Bu örnekte sırasıyla; x + y işleminin sonucu z'ye taşınmaktadır. Öte yandan sayı + 1 işleminin sonunda elde edilen değer yeni sayı değeri olmaktadır.

ETKİNLİK

Algoritmada, her komut, bir kişinin kalem ve kağıt ile yürütebileceği kadar basit olmalıdır. Algoritmada tekrar anlatımlar olmamalıdır. Bir algoritma bünyesinde ne kadar az tekrar varsa algoritmanın etkinliği o kadar artar. Kaçınılmaz tekrarlarda ise bir algoritmayı etkin hale getirebilmek için; tekrar anlatımların alt algoritma yapılması gerekmektedir.

SONLULUK

Her türlü olasılık için algoritma sonlu adımda bitmelidir. Her algoritmanın bir bitiş ya da geriye dönüş noktası olmalıdır. Ana algoritmada bitiş noktası END, alt algoritmalarda ise geriye dönüş noktası RETURN komutları ile sağlanır. İşletim sistemleri gibi bazı programlar istisnai olarak sonsuza dek çalışırlar.

GİRİŞ/ÇIKIŞ BİLGİSİ

Bir algoritmada mutlaka Giriş ve Çıkış bilgisi olmalıdır.Giriş bilgisi, algoritmaya dışarıdan bilgi aktarımını, Çıkış bilgisi ise, algoritma içinde oluşan sonuçların algoritma dışına çıkartılabilmesi işlemidir.

Genelde Giriş ve Çıkışı işlemleri için Read ve Write (veya Print) kullanılır. Bir bilginin okunabilmesi için değişken kullanılır.

Read Değişken	Değişken ile belirtilene dışardan değer oku.
Write Değişken	Değişken ile belirtilendeki değeri dışarıya yaz.

ÖRNEKLER

Örnek 1.1:
İki sayının toplamını yazan bir algoritma
Çözüm:

1. İlk sayıyı oku
2. İkinci sayıyı oku
3. Sayıları topla
4. Sonucu görüntüle.

Örnek 1.2:
Kullanıcının girdiği 4 sayının ortalamasını hesaplayıp yazdıran algoritma
Çözüm:

1. Başla
2. Sayaç = 0 ve Toplam = 0
3. Sayıyı Oku
4. Sayıyı Toplam'a ekle
5. Sayaç'ı 1 arttır
6. Sayaç < 4 ise 3. adıma git
7. Ortalamayı hesapla (Ortalama = Toplam / 4)
8. Ortalamayı yazdır
9. Son

Örnek 1.3:
20'den 50'ye kadar olan sayıların toplamını bulan algoritma
Çözüm:

1. Başla
2. S=20 ve T=0 ata. (Sayı=20 T=0 ile başla)
3. T=T+S (T'ye sayıyı ekle T'yi göster.)
4. S=S+1 (Sayıyı bir artır.)
5. S<50 ise A3'ye git. (Eğer sayı 50'den küçük ise Adım 3'ye git)
6. T'yi göster. (T'nin değerini göster.)
7. Son

Örnek 1.4:
Klavyeden girilen iki sayıdan en büyüğünü bulup gösteren algoritma.
Çözüm:

1. Başla
2. S1=? S2=? (İlk sayıyı gir ;İkinci sayıyı gir.)
3. S1>S2 ise git Adım 5 (Sayı 1 sayı 2'den küçükse Adım 5'e git.)
4. S2>S1 ise git Adım 6 (Sayı 2 sayı 1'den küçükse Adım 6'e git.)
5. S1'i göster git Adım 7 (sayı 1 değerini göster ve işlemi durdur)
6. S2'yi göster. (Sayı 2 değerini göster)
7. Dur

Örnek 1.5:
Klavyeden girilen üç sayıdan en büyüğünü bulup gösteren algoritma
Çözüm:

1. Başla
2. X, Y, Z değerlerini oku
3. EB = X
4. Y > EB ise EB = Y
5. Z > EB ise EB = Z
6. EB' yi yazdır
7. Son

Örnek 1.6:

1'den 100'e kadar olan sayıların toplamını veren algoritma.

Çözüm:

1. Başla
2. Toplam T, sayılar da i diye çağırılsın
3. Başlangıçta T'nin değeri 0 ve i'nin değeri 1 olsun
4. i'nin değerini T'ye ekle
5. i'nin değerini 1 arttır
6. Eğer i'nin değeri 100'den büyük değil ise 3. adıma git
7. T'nin değerini yaz
8. Son

Algoritmaların yazım dili değişik olabilir. Günlük konuşma diline yakın bir dil olabileceği gibi simgelere dayalı da olabilir. Akış şeması eskiden beri kullanıla gelen bir yapıdır. Algoritmayı yazarken farklı anlamlar taşıyan değişik şekildeki kutulardan (Akış Şemaları) yararlanılır. Yine aynı amaç için kullanılan programlama diline yakın bir (sözde kod = pseudo code) dil , bu kendimize özgü de olabilir, kullanılabilir.

Aynı algoritmayı aşağıdaki gibi yazabiliriz.

1. Başla
2. T=0 ve i=0
3. i'nin değerini T'ye ekle
4. i'yi 1 arttır
5. i<101 ise 2.adıma git
6. T'nin değerini yaz
7. Son

Örnek 1.7:

İki tamsayının çarpma işlemini sadece toplama işlemi kullanarak gerçekleştiren algoritma.

Çözüm;

Girdi : iki tamsayı
Çıktı : sayıların çarpımı

1. Başla
2. a ve b sayılarını oku
3. c =0
4. b>0 olduğu sürece tekrarla

4.2. c=c + a

4.3. b = b-1

5. c değerini yaz
6. Son

Örnek 1.8:

Bir tamsayının faktöriyelini hesaplayan algoritma

Çözüm:

Girdi : Bir tamsayı
Çıktı : Sayının Faktöriyel
İlgili formül: Faktöriyel(n)=1*2*...*n

1. Başla
2. n değerini oku
3. F=1

4. n >1 olduğu sürece tekrarla
4.1. F=F*n
4.2. n= n-1
5. F değerini yaz
6. Son

Örnek 1.9:
İki tamsayının bölme işlemini sadece çıkarma işlemi kullanarak yapan algoritma (Bölüm ve kalanın ne olduğu bulunacak).
Çözüm;
1. Başla
2. a ve b değerlerini oku
3. m=0
4. a>=b olduğu sürece tekrarla
4.1 a=a-b
4.2 m = m + 1
5. kalan a ve bölüm m 'yi yaz
6. Son

Örnek 1.10:
100 tane sayıyı okuyup, ortalamasını bulan algoritma
Çözüm;
1. Başla
2. T=0, i=0
3. i<101 olduğu sürece tekrarla
3.1 m değerini oku
3.2 T = T + m
3.3 i = i + 1
4. T = T / 100
5. Ortalama T 'yi yaz
6. Son

1.6.2 Akış Diyagramları

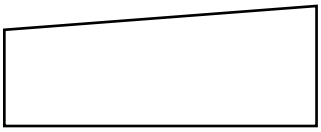
Akış Diyagramı; bir algoritmanın belirli bir anlamı olan şekillerle ifade edilmesidir. Önceki konuda eğer dikkat edildiyse algoritmaların, doğal dille yazıldığı için herkes tarafından anlaşılamayabilir ya da istenmese de başka anlamlar çıkarılabilir oluşudur. Ancak akış diyagramlarında her bir şekil standart belli bir anlam taşıdığı için farklı yorumlanıp anlaşılması olası değildir. Bir algoritmanın ifade edilebilmesi için sıklıkla kullanılan şekiller ve anlamları şunlardır:

Bir algoritmanın başladığı veya bittiği konumu gösterir.

Bir algoritmada aritmetik işlem yapılmasını sağlayan şekildir. Bu dörtgen kutu içerisine yapılmak istenen işlem yazılır.

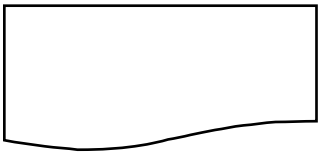
Algoritmada bir bilginin ekrana yazılacağı konumu gösteren şekildir. Ekrana yazılacak ifade ya da değişken bu şekil içerisine yazılır.

Bir algoritmada başka bir yerde tanımlanmış blokun yerleştiği konumu gösteren şekildir. Kutu içerisine blokun adı yazılabilir.

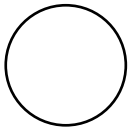


Klavyeden Bilgisayara bilgi girilecek konumu belirten şekildir. Girilecek bilginin hangi değişkene okunacağını kutu içerisine yazabilirsiniz.

Giriş - Çıkış komutunun kullanılacağı yeri belirler. Kutu içerisine hangi değişken veya değişkenlere okuma mı? yoksa yazma mı? yapılacağını belirtmeniz gerekir

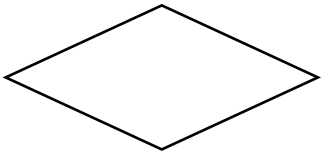


Bilginin Yazıcıya yazılacağı konumu gösteren şekildir.



Bir algoritmanın birden fazla alana yayılması durumunda bağlantı noktalarını gösteren şekildir. Tek girişli veya tek çıkışlı olarak kullanılırlar.

Bir işlemin belli bir sayıda veya belli bir koşul doğru olduğu sürece tekrar edilmesini sağlayan döngü komutunu gösteren şekildir. Bu döngüde altıgen içerisine ya koşul yada döngünün başlangıç, adım ve sonlanma değerlerini belirtebilirsiniz. DÖNGÜ olarak belirlenen blokta da tekrar edilmek istenen komutlar yer almaktadır.

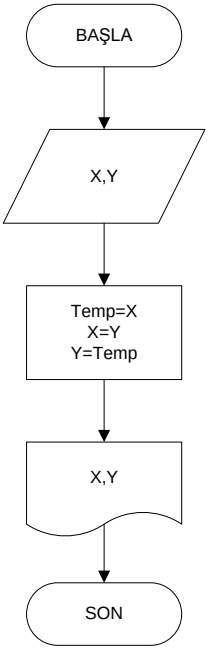


Bir algoritmada bir kararın verilmesini ve bu karara göre iki seçenekten birinin uygulanmasını sağlayan şekildir. burada eşkenar dörtgen içerisine kontrol edilecek mantıksal koşul yazılır. Program akışı sırasında koşulun doğru olması durumunda "Evet" yazılan kısma Yanlış olması durumunda "Hayır" yazılan kısma sapılır. Tek girişli ve çift çıkışlı bir şekildir.

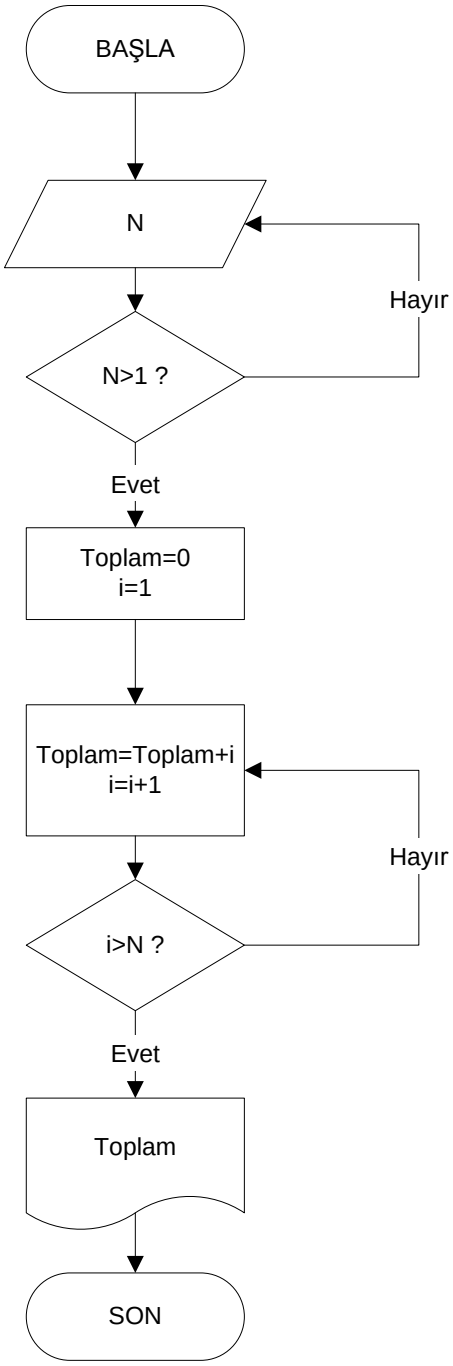
Bu şekiller kullanılarak algoritma ile oluşturulan çözümler akış diyagramlarına dönüştürülür. Bu diyagramlar herkes tarafından anlaşılabilir ve doğru olarak yorumlanabilir bir özellik arz ederler.

Şimdi akış diyagramlarına birkaç örnek inceleyelim;

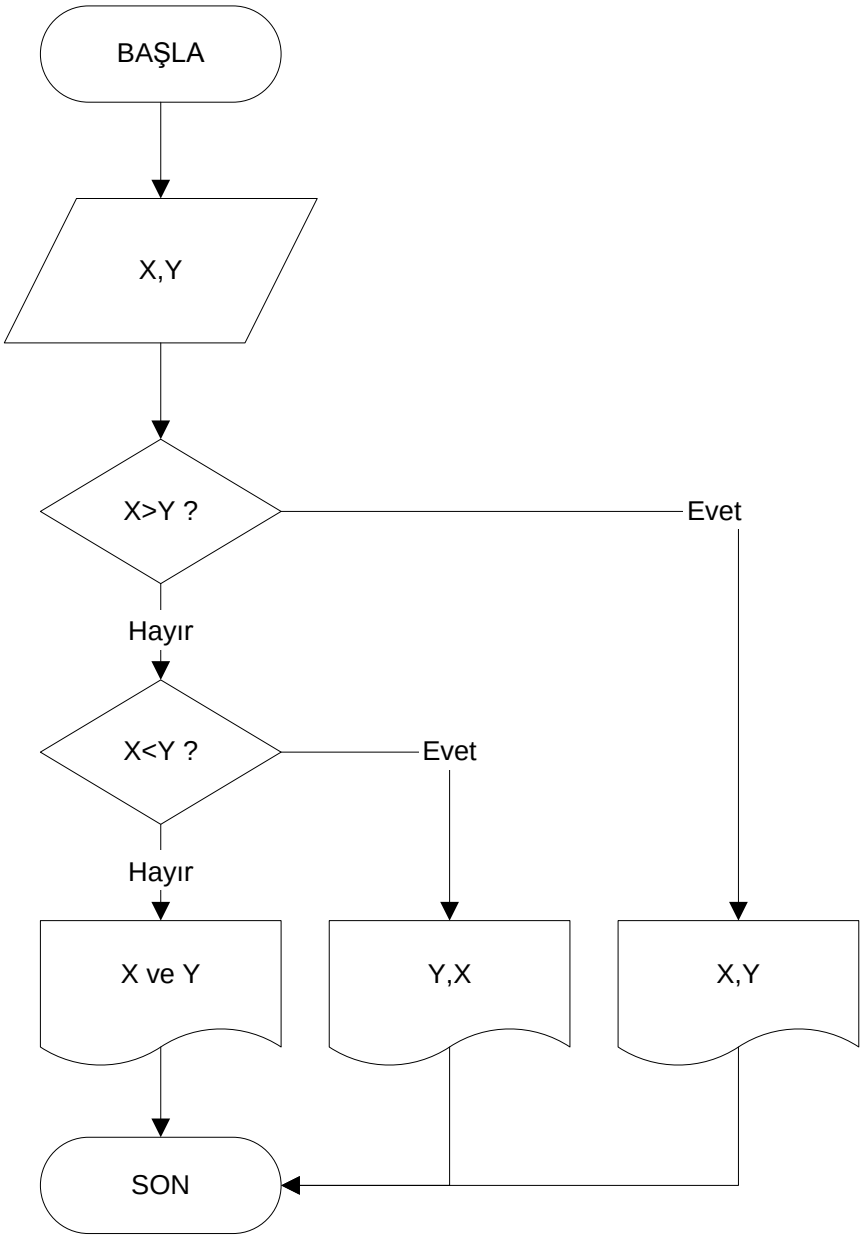
Örnek 1.11:
Öncelikle pek çok kez örnek olarak verilen akış diyagramlarının başında gelen bir problemi yapalım. Burada dışarıdan girilen iki sayıyı yer değiştirip çıktı olarak veren algoritmanın akış diyagramını yapalım.



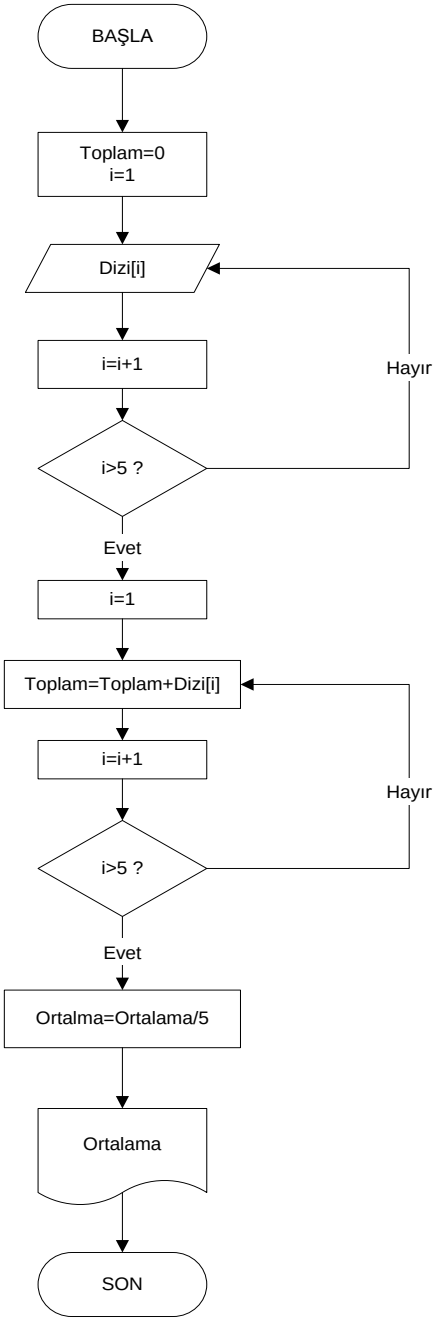
Örnek 1.12:
İkinci olarak dışardan girilecek bir N sayısı için 1 den N'ye kadar olan sayıların toplamını alıp çıktı olarak veren algoritmayı akış diyagramı olarak ifade edelim;



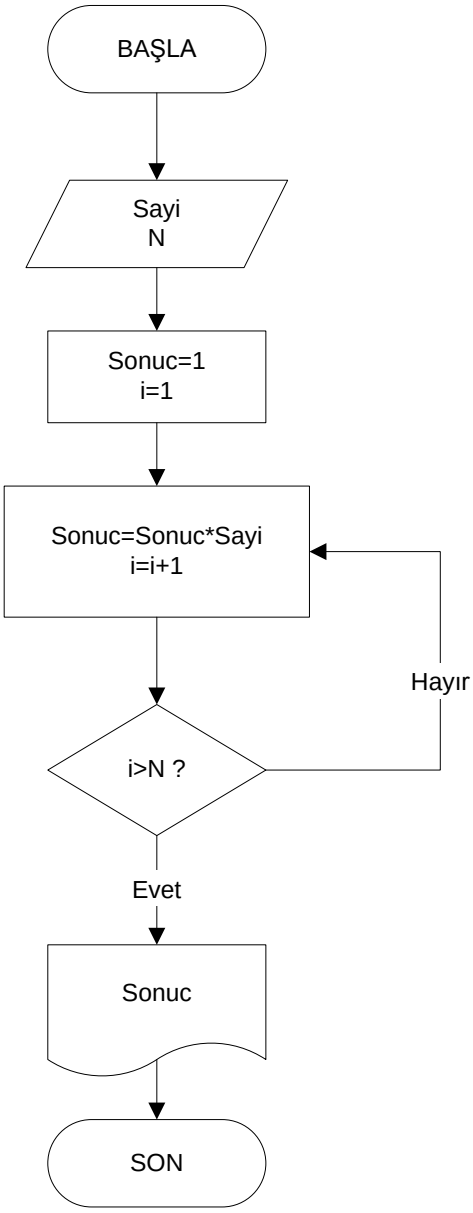
Örnek 1.13:
Şimdi de dışardan girilen iki sayıyı büyükten küçüğe doğru sıralayan programın akış diyagramını çizelim;



Örnek 1.14:
Bu örnekte dışardan girilen 5 adet sayının aritmetik ortalamasını alıp çıktı olarak veren akış diyagramı görülüyor;



Örnek 1.15:
Son örneğimizde dışarıdan girilen bir sayının N'ninci kuvvetini alan bir akış diyagramı görülüyor;



1.7 PROGRAMLAMA DİLLERİ

Programlama Dili bilgisayarda çözülecek bir sorun için çözümün bilgisayara adım adım yazılmasını sağlayan biçimsel kuralları olan ve bu kurallara sıkı sıkıya bağımlılığı gerektiren bir tanımlar kümesidir.

Belki daha kısa bir tanımı ile sizinle bilgisayar arasında bir tercümandır demek doğru olur. Bir sorun çözüleceği zaman öncelikle iyice anlaşılmış olmalıdır. Sonra bu sorunu çözebilecek bir çözüm zihinsel olarak hazırlanır. Bu çözüm bilgisayara uygun bir çözüm olmalıdır. Şöyle ki her çözüm bilgisayarda uygulanamaz. Çünkü her çözümün takip ettiği yol yeteri kadar basit olmayabilir. Üretilen çözüm son derece basit adımlarla anlatılabilmelidir. Algoritma kavramını hatırlayınız. Bu adımlar alt alta yazılmak suretiyle oluşturulan çözüm bilgisayar için uygundur. Ancak ihtiyaç var ise bu adımlar akış diyagramlarına çevrilebilir. Algoritmalar doğal bir dil ile yazılır ve sıkı sıkıya kuralları bulunmaz. Anlaşılmasının kolay olması yeterlidir. Akış diyagramlarında belirlenmiş semboller yer alır ve bu semboller tüm dünyada standarttır. Kısmen biçimsel olan bu diyagramlar, sorunun çözümünü daha evrensel bir dille ifade eder.

Son adım olarak, akış diyagramları veya algoritma ile elde edilen çözümün bir programlama dili ile bilgisayar ortamına aktarılması gerekir. Programlama dili son derece standart tanımlar içerir ve bir programı yazarken bu tanımlardan bir an için bile uzaklaşılamaz. O nedenle de bir program parçasından başkalarının başka şeyler anlaması mümkün değildir. Yazılan bu programlar bir derleyici vasıtası ile Makine diline çevrilir varsa hataların bulunmasını sağlar ve kullanıcı bu hataları düzeltir.

1.7.1 Programlama Dillerinin Bazı Özellikleri

İfade gücü: Dili kullanırken gerçek ifadelerin kullanılması ile ilgilidir. Örneğin bir matematikçi ve kimyacı kodlama yaparken kullandığı işaretleri ve terimleri kullanmak isteyecektir.

Veri Türleri ve Yapıları: Ön tanımlı değişken türlerinin fazla ve ihtiyaçları karşılaması, bir dilden beklenen bir özelliktir.

Giriş - Çıkış Kolaylığı: Dosyalara erişme, karmaşık işlemler yapma imkanlarını kasteden bu özellik, C'de pek gelişmemiştir. Özel kütüphaneler gerektirir. Veritabanı programlama dilleri bu konuda oldukça gelişmiştir.

Taşınabilirlik: Bir sistemde yazılmış kaynak kodun, başka sistemlerde de sorunsuz derlenebilmesidir. Genellikle dilin seviyesi azaldıkça taşınabilirlik azalır. C dili, orta seviyelidir ancak taşınabilirlik bakımından üstündür.

Alt Programlanabilirlik: Programın daha ufak programcıklardan oluşturulmasıdır. Böylece kaynak kod kısalır, algılanması güçlenir, test olanakları artar, kodun güncelleştirmesi ve yeniden kullanılması kolaylaşır.

Verimlilik: Derlenen kodun hızlı ve sorunsuz çalışabilmesidir.

Okunabilirlik: Kaynak kodun hızlı biçimde anlaşılabilmesidir. İyi bir programcının yazdığı kaynak kod, çok iyi işlev gören ama karışık bir koddan ziyade açık ve anlaşılabilir biçimdedir. Ancak bu ölçüt dile de bağlıdır.

Esneklik: Dilin, programcıyı kısıtlamamasıdır. Ancak esnek bir dil, daha az hata vermesine karşın hata oluşma riski daha fazladır.

Öğrenme Kolaylığı: Dilin konuşma diline yakınlığı, komutlarının sade ve anlaşılır olması gibi ölçütler o dilin öğrenilmesini etkiler.

Genellik: Bir dilin herhangi bir alanda kullanılabilmesidir. Bazı diller sadece mühendislik alanlarında kullanılmasına karşın, C genel amaçlı bir dildir.

Yapısal Programlanabilirlik: Programın bloklar halinde yazılması, atlamasız akışı ve altprogramların kullanılması anlamlarına gelen bir programlama tekniğidir. Kodun okunabilirliğini ve verimini artırır.

Nesne Yönelimlilik: Yeni diller ve eski dillerin yeni uyarlamaları artık nesne yönelimli olmaya başladılar. Verilerin birbirinden daha kesin çizgilerle ayrılmasını öngören bir programlama tekniğidir.

1.7.2 Programlama Dillerinin Sınıflandırılması

Programlama dilleri, makine dilinde programlama çok zor olduğu için geliştirilmiştir. Programlama dilleri kendi aralarında sınıflara ayrılmışlardır. İnsanın en zor öğrenebileceği, anlayabileceği yani 1100101 gibi makina kodlarına yakın diller en düşük seviyeli (low level) programlama dilleri, insanın en kolay anlayıp kullanabileceği ve insan diline yakın özellikler gösteren diller ise en yüksek seviyeli (high level) programlama dilleridir. Yazılan kodları, zaten makine dilinde değilse, makine diline çevirip koşmaya hazır hale getirmek, o dilin derleyicisinin veya yorumlayıcısının görevidir. Bu diller seviyelerine göre aşağıdaki gibi sınıflandırılabilir;

(Öğrenilmesi Kolay, Daha Yavaş)

Çok Yüksek Seviyeli Diller (İnsana en yakın) FOXPRO, ACCESS, PARADOX, VB...

Yüksek Seviyeli Diller : PASCAL, FORTRAN, COBOL, BASIC...

Orta Seviyeli Diller : C ,C++, C# , Java ,ADA...

Düşük Seviyeli Diller: Assembly...

Makine Dilleri (Makineye en yakın diller. 0 ve 1'lerin dizilimlerinden oluşurlar..)

(Öğrenilmesi Daha Zor, Daha Hızlı)

Burada insana yakın demekteki kasıt, insanın anlamasına uygun, anlamlı sözcüklerle kodlama yapmak, makineye yakın demekteki kasıt ise bilgisayarın çalışma mantığına uygun, ne yapılacağı değil, nasıl yapılacağını kodlamaktır. Bir dilin seviyesi yüksekse, o dili öğrenmek kolaydır, kaynak kod kısadır ama oluşacak çalıştırılabilir dosya uzundur, uzun ve karmaşık işlemler kısa kodlarla gerçekleştirilebilir; Alçak seviyeli dillerde ise programcı, makineye daha hakimdir, sorumluluğu daha fazladır, kod yazımı uzun ve zahmetlidir.

Programlama dillerinin kendi alanları vardır ve her dil kendi branşında kullanıldığı sürece başarılı ve etkili kullanılmış olur. Genel olarak programlama dilleri uygulama alanlarına göre aşağıdaki sınıflara ayrılabilir:

1. **Bilimsel ve Mühendislik Alanında :** Üniversitelerde ve bilimsel kuruluşlarda mühendislik veya matematik hesapları için kullanılırlar. Bu dillere Pascal, C, C++, Java, Fortran gibi diller örnek olarak verilebilir.
2. **Veritabanı Kullanımında :** Genellikle personel kayıtları, stok veya depo denetimi vb gibi veritabanı gerektiren işlemlerde kullanılan dillerdir. Bu dillere DBase, Sql, Foxpro, Paradox gibi diller örnek olarak verilebilir.
3. **Sistem Programcılığında :** İşletim sistemlerinin ve sistem programlarının yazılımında kullanılan dillerdir. Örnek olarak C, C++, Java ve makina dilleri verilebilir.
4. **Genel Amaçlı kullanım :** Çeşitli konularda uygulama geliştirmek için kullanılan dillere Örnek olarak C, C++, Java, VB ve Pascal'ı verebiliriz.
5. **Yapay Zeka Kullanımında:** Özellikle son zamanlarda popüler olan yapa zeka uygulamalarında kullanılan dillerdir. Örnek olarak Prolog, Lisp gibi diller verilebilir.

Son yıllarda programlama dillerinde nesneye yönelik tasarımlar yapılmış ve bu dillerin çoğunun nesneye yönelik programlama yapabilen uyarlamaları çıkmıştır. Nesneye yönelik programlama, programcının kendi sınıfını ve nesnesini oluşturup bunun üzerinde işlemler yapmasına olanak sağlayan ve programlama dillerinin geldiği son aşamalardan birisidir.

OOP (Object Oriented Programming) yani NYP (Nesneye Yönelik Programlama)'nın kullanılmasıyla ve Visual (Görsel) programcılığın da gelişmesi ile beraber ortaya oldukça güzel görünümlü ve kullanışlı programlar çıkmaya başlamıştır. Bu durum bilgisayar programlarına olan ilgiyi bir hayli arttırmış ve bu konulardaki araştırmaları hızlandırmıştır. Sonuçta, pek çok dilin artık nesneye yönelik olan ve görsel özellikler içeren sürümleri kullanılmakta ve tercih edilmektedir. Örneğin: Visual C++, C++ Builder, Delphi, Kylix, Java, Visual Basic vb. gibi diller.

İnternet'in de yaygınlaşmasıyla programlama dillerine yeni özellikler eklenmiş ve İnternet'te kullanılabilecek nitelikte görsel özellikli diller çıkarılmıştır. Örneğin web sayfalarının gösteriminde kullanılan HTML, kullanıcının dikkatini çeken, renkli ve hareketli arabirimiyle her geçen gün yeni ekler ile gelişmektedir. JavaScript ve VBScript gibi script diller HTML'ye getirdikleri ek özelliklerle çokça kullanılır olmuştur. Öte yandan Asp, Php, Perl vb. ile veri tabanlarını aktif kullanıma sunma, sunucu taraflı aktif sayfalar hazırlama olanağı elde edilmiştir.

Uygulama programlarına, ticari programlara veya işletim sistemlerinin kullanımına yönelik yazılım geliştirmek isteyen programcıların C, C++, Java, Delphi vb. gibi son zamanların en popüler dilleri üzerinde çalışması ve en az bunlardan bir tanesini öğrenmesi gerekir. Özellikle İnternet'in gelişmesi ile birlikte programlama dillerinin, yapılacak işe veya kullanılacak platforma göre değişik özellikler gösteren varyasyonları çıkmıştır. Bu durumda web teknolojisini izleyen ve İnternet üzerinde yazılım geliştirmek isteyen bir programcının javascript, vbscript, HTML gibi dilleri de bilmesi ve bu konularda kendini geliştirmesi de ayrı bir gereklilik olmuştur. Sıkça kullanılan programlama dilleri kısaca özetlenirse;

C
Yapısal programlama dilleri arasındadır. Öğrenilmesi zaman almasına rağmen oldukça kullanışlı ve esnek yapısı ile adından yıllarca bahsettirmiş, bilgisayar programcılığının temel dillerinden biridir. C ile bilgisayarınıza bir sistem yazmaktan bir oyun yazmaya kadar her türlü işlem yapılabilir. Bu özelliği sayesinde kullanım alanı çok geniş olan bir dildir. Bu kitabın odaklanacağı dilde bu olacaktır.

C++
Nesneye yönelik programlama yapabilen diller arasındadır. C'nin saydığımız tüm özelliklerine ek olarak güçlendirilmiş nesne yönetim özelliği ile şu anda bilgisayar dünyasının en çok kullanılan dillerinden biridir.

Pascal
Yapısal bir dildir, C diline benzerlik gösterir. Öğrenilmesinin kolay oluşu ve genelde, bilgisayar eğitimi veren okullarda okutulan bir ders olması sebebiyle kullanım alanı daha çok üniversiteler ve bilimsel hesaplamalar yapan kurumlardır.

C#
Nesneye dayalı bir programlama dilidir. C++'ın ve Java'nın pozitif yönlerini bünyesinde birleştirmiş yeni bir dildir. Programcıya internet uygulamaları ve yerel uygulamalar yazmakta bazı kolaylıklar getirmiştir. İleriye dönük olarak Microsoft firmasının Java teknolojisine rakip olarak ortaya sürdüğü bir programlama dilidir ve Microsoft'un bu konulardaki (İnternet uygulamaları) yelpazesini genişletmeye yönelik bir atılımdır. Microsoft teknolojileri kullanacak programcıların C#'ı öğrenmeleri zamanla gerekecektir ve öncelikle öğrenilmeye başlanması da avantaj getireceği açıktır.

Delphi
Pascal tabanlı bir dil olup nesneye yönelik programlama yapabilme özelliği taşır. Öğreniminin kolay oluşu ve genellikle üniversitelerde Pascal eğitiminin ağırlıklı verilmesi nedenleriyle çoğu bilgisayar programlama öğrencisinin tercih ettiği bir dildir. Görsel programlama özelliği taşır. Şu anda İnternet üzerinde en çok desteklenen ve üzerine bileşen (component) geliştirilen dillerin başında yer alır. Geniş bir kullanıcı kitlesi vardır.

Visual Basic
Basic tabanlı bir dil olup öğrenilmesi kolay, kullanım alanı geniş bir dildir. Özellikle görsel uygulamalarda projenin arabiriminin hızlı yazılmasını sağladığı için genelde kullanıcı arabirimi tasarımlarında kullanılır. Kapsamlı veya çok kullanıcılı uygulamalarda kullanılmaz. Kullanıcı sayısı az olan veya kısa sürede bitmesi gereken küçük ölçekli projelerde tercih edilir. Delphi'den sonra yoğun olarak kullanılmaktadır

VB.NET
Nesneye dayalı bir dildir. VisualBasic(VB) teki biçok özellik bu dilde yeniden yapılandırılarak değişmiştir. Yapısına bakılırsa VB den ayrı yeni bir dil geliştirilmiş denilebilir. Eklenen bazı özellikler ile VB de yapılamayan birçok işlem artık yapılabilmektedir ve OOP nin özellikleri desteklenerek daha verimli kod yazmaya olanak sağlanmıştır. VB programcılarının VB.NET'e geçişleri kolay olmayacak olsa da VB yerine VB.NET kullanımı gün geçtikçe artacaktır. VB.NET internet uygulamalarından yerel uygulamalara kadar kullanım imkanı geniş bir dildir.

Java

Açık olmak gerekirse, İnternet programcılığı, esnek programlama mimarileri, OOP gibi konularda söylenmesi gereken önemli bir nokta; Java dilinin, programlama dünyasına getirdiği önemli bir yeniliktir. Bu yenilik platformdan yani işletim sisteminden bağımsız olarak her sistemde çalışabilen programların yazılabilmesidir. İleriye yönelik bir bakış açısı ile bakıldığında cep bilgisayarlarının, kablosuz cihazların, cep telefonlarının sıklıkla kullanılacağı ve artık PC döneminin kapanmaya başladığı düşünülürse birçok cihazda çalışabilecek programların yazılmasının ne derece önemli olduğu anlaşılabilir. Java, son yıllarda programlama dünyasına gelmiş en iyi programlama araçlarından biridir ve yeni çıkacak programlama dillerinin çoğu Java'nın birçok özelliğinden esinlenmektedir.

Java, nesneye yönelik dillerdendir. Son yıllarda geliştirilmiş bir dil olup modern ve yenlikçi altyapısı, görsel özellikleri ve sürekli gelişen kütüphane (library) desteği ile gün geçtikçe kullanımı artan bir dil olmuştur. Java dili platform bağımsızlığı özelliği ile hemen hemen her alanda kullanılabilen esnek ve güçlü bir dildir.

Programlama dillerinin hepsini incelemeye imkan olmadığından burada çok kullanılan bir kısım dilden bahsedildi. Her dilin kullanım amacı ve yönelimi farklı olabilir önemli olan hedeflediğiniz konularda size yardımcı olacak dili seçip onunla çalışmanızdır.

1.8 DEĞERLENDİRME SORULARI

1. Bir program tasarlama aşamalarını adım adım sayınız.
2. İyi bir program hangi nitelikleri taşımalıdır, belirterek açıklayınız.
3. Girilen N adet sayıdan en büyüğünü ve en küçüğünü bulan algoritmayı yazınız.
4. Tamsayılarda üs alma işlemini gerçekleştiren algoritmayı yazınız.
5. 1-100 arasında tutulan bir sayıyı tahmin eden algoritmayı yazınız.
6. Binom açılımı yapan algoritmayı yazınız.
7. İkinci dereceden 1 bilinmeyenli denklemin köklerini bulan algoritmayı yazınız.
8. Bir dik üçgenin girilen iki kenar değerine karşılık 3. kenarın uzunluğunu bulan akış diyagramını çiziniz.
9. Dışardan girilen N elemanlı bir dizide istenilen bir değeri arayan ve varsa kaçıncı değer olduğunu bulan algoritmanın akış diyagramını çiziniz.
10. Dışarıdan girilen N sayısının bölenlerini bulup çıktı olarak veren algoritmanın akış diyagramını çiziniz.
11. Girilen sayının tek mi yoksa çift mi olduğunu bulan algoritmanın akış diyagramını çiziniz.
12. Dışardan girilen sayıyı yazıya çeviren algoritmanın akış diyagramını çiziniz.
13. Programlama Dillerinin özelliklerini açıklayınız.
14. Programlama Dillerini seviyelerine göre sınıflandırınız.
15. Programlama Dillerini uygulama alanlarına göre sınıflandırınız.

C DİLİNİN GENEL YAPISI

Bölüm 2

2.1. GİRİŞ

Genel amaçlı bir programlama dili olan C, esnek ve basit bir programlama dilidir. Bu esnek yapısı dilin mikro denetleyici programlamasından işletim sistemi yazımına, paket programlardan bilimsel programlara kadar değişik tipteki uygulamalarda kullanılmasına olanak sağlamaktadır. Bu dilin basitliği, oluşturulan kodun küçüklüğü ve üst seviyelerden alt, assembly' ye yaklaşan alt seviyelere kadar programlama kodun yazılabilmesi, açık bir dil olması, her çeşit programlamada kullanılabilmesi, C' yi popüler bir dil yapmıştır. Bu prog-ramlama dili günümüzde hemen hemen her alanda tercih edilen ve esnek bir dil olması nedeniyle bu kitapta C programlama dili üzerinde durulacaktır. C programlama dilini anlatmadan önce ilk olarak C dilinin tarihi gelişimi üzerinde durulacaktır.

BÖLÜM -2- C Dilinin Genel Yapısı

Bölümün Genel Amacı: C dilini tanıma ve genel yapısını öğrenme

Bölümün Davranışsal Amaçları: Kitabınızın bu bölümünü başarıyla bitirip, uygulamaları yapıp, değerlendirme sorularına doğru cevap verdiğiniz taktirde, bölüm sonunda;

- C Dilinin Tarihi Gelişimi
- C Dilinin Avantaj ve Dezavantajları
- C Dilinin Temel Kavramları
- C Dilini Genel Program Yapısı
- C Dilindeki Temel Tanımlamaları bilmeniz beklenmektedir.

Değerlendirme: Modül sonundaki uygulamaları yapmanız sonuçları uygulama raporu ile karşılaştırmanız ve değerlendirme sorularına en az % 75 düzeyinde doğru cevap verebilmeniz gerekmektedir.

2.2. C DİLİNİN TARİHİ GELİŞİMİ

Unix işletim sistemi ile C programlama dili bir birleriyle yakından ilişkilidir. Tarihleri 70 lerin başında başlar. Ve çıkış noktası AT&T BELL LABORATUVAR' ında Ken Thompson tarafından yazılan bir oyun programından kaynaklanmaktadır. Thompson , yazdığı oyunu bir PDP-7 bilgisayarında kullanmış ve kullanılan işletim sistemi (MULTICS) hiç hoşuna gitmemiştir ve bu işletim sisteminin basit ve kolay şeklini yazmaya karar vermiştir. Daha sonra çalışmalara ortak olan M. Ritchie ve Brain W. Kernighan tarafından UNICS *(uniplexed Information and Compuing Service)* adı verilen işletim sisteminin ilk hali oluşturuldu. Thompson ilk başlarda BCPL programlama dilini kullanarak B programlama dilini tasarladı daha sonra ise UNIX işletim sistemini daha kolay yazmak amacıyla ise C programlama dilini oluşturdu ve UNIX işletim sistemini yeniden yazarak daha aktif, esnek bir işletim sistemi ortaya çıkmıştır. C programlama dili bazı ufak tefek değişiklik, eklentiler ve editördeki bazı kullanıcı hatalarını azaltmak için yapılan düzenlemeler haricinde bu tarihten beri olduğu şekliyle kullanılmaktadır. Buda ortaya C dilinin özel olarak üretilen bir dil olmadığı ve ihtiyaçlardan doğan bir dil olduğu, görülmektedir.

2.3. C DİLİNİN AVANTAJ VE DEZAVANTAJLARI

C dili hem üst düzey hem de alt düzey programlamayı destekleyen bir dil olarak tasarlanmıştır. Programcı önceden hazırlanmış temel fonksiyonları kullanarak istediği işlemleri rahatlıkla kullanırken derleyici ve bu fonksiyonların anlamı hakkında hiçbir şey bilmez. Bu fonksiyonlar C programlama dilinin kütüphanesini oluşturmaktadır. C dili program yazma aşamasında bu kütüphanelerden faydalanır. Dolayısıyla bu dile eklenecek olan yeni kütüphaneler C dilinin gücünü arttıracaktır. C programlama dili az sayıda anahtar sözcüğü ve güçlü işlem operatörlerini içermektedir. Dolayısıyla C dilinin öğrenilmesi koyladır. C dilinde işlem operatörleri makine koduna dönüştürülürler buda C dilinin hızlı olmasını sağlamaktadır.

C dili programcının bilgisayardan bağımsız program yazmasına ve programın rahatlıkla başka sistemlere aktarılmasına olanak sağlayan bir dildir. Bu nedenle C programlama dilinin ilk kullanıldığı yer olan UNIX işletim sisteminden sonra diğer sistemlerde de kullanılmaya başlanmıştır.

C programlama dili programcıyı modüler programlamaya teşvik eder. Bunun için çeşitli bellek sınıfları çeşitli düzeylerde gizlilik sağlamaktadır. Modüler programcılığın temeli olan fonksiyonlar C dilinde oldukça rahat bir şekilde kullanılmaktadır.

C dilinde yazılan programların dezavantajlarından en önemlisi yazılan programın içeriği arttıkça ve karmaşıklaştıkça programın takibinin de zorlaşmasıdır. C dili Basic v.b diller gibi yürütme zamanı desteği sağlamamaktadır ayrıca derleyici işlem sırasında alt ifadeleri ve argümanların hesaplanma sıralarını değiştirebilmektedir.

Günümüzde C dilinin bu dezavantajlarına rağmen programcıya oldukça fazla avantaj sağlamasından dolayı tercih edilen ve kullanılan bir dil halini almıştır. Ayrıca Windows platformlarının gelişmesi ile birlikte görsel programcılığın gelişimi hızlanmıştır. Bununla birlikte C de görsel ortama taşınmış ve Visual C olarak programcıların tercih ettiği bir dil halini almıştır. Visual C dili, C dili ile temelde aynı fakat görsel işlevler için ekstra işlevleri

bulunmaktadır. Ayrıca C dili nesne tabalı programlamaya da müsait bir dildir ve nesne tabanlı C programlama dili olarak C++ dili geliştirilmiştir.

2.4. TEMEL KAVRAMLAR

C dilinin tarihi gelişimini , avantaj ve dezavantajlarını inceledikten sonra C dilinin genel kavramlarını ve kavramların nasıl kullanıldığını inceleyelim.

2.4.1.C Dilinde Programın Yapısı

Bir programcı yazacağı programı en az sayıda komut ile oluşturmalı ve en iyi algoritmayı oluşturmalıdır. Programcının kodu en aza indirebilmesinin tek yolu programı yazacağı dilin yapısına ve özelliklerine hakim olmasıdır.

Programcılığa yeni başlayanlar için burada C dilinde bir program yazılırken nelere dikkat edilmeli ve olmazsa olmazlar nelerdir bunları inceleyelim.

```
/* Açıklama Satırları
   Bu bloklar arasına yazılır */
#include <Kullanılan Kütüphane>
#define Sabit ve Fonksiyonlar
Değişkenler
main()
{
Program Kodları
        Printf("Ayhan Akbal");
        .
        .
}
```

Şekil 2.1: Basit C Programı Kodu

Yukarıda en basit haliyle bir C programı görülmektedir ve bu programı satır satır inceleyelim. Programda görülen ilk iki satır dikkat edilirse **/*** ve ***/** işaretleri arasında ***" Açıklama satırları Bu bloklar arasına yazılır"*** ifadelerinin bulunduğu alan C derleyicisi tarafından **/*** ve ***/** işaretleri nedeniyle derleyici dikkate alınmaz. Dolayısıyla programın herhangi bir yerinde açıklama yazılması gerekirse **/*** ve ***/** işaretleri arasına yazılabilir. **/*** ve ***/** işaretleri bir blok ifade ederken sadece bir satır açıklama satırı olarak kullanılacaksa bu durumda satır başına **//** işaretleri kullanıldığı takdirde derleyici **//** işretinin bulunduğu satırı dikkate almaz(sadece C++ derleyicilerde). Burada akla şu soru gelebilir,

“Açıklamalar derleyici tarafından dikkate alınamadıkları halde neden kullanılır?”. Bu sorunun cevabı bir programcının yazdığı program binlerce satırdan oluşabilir, dolayısıyla programcı yazdığı programı takibi, kontrolünde veya uzun süre sonra tekrar programını açtığı zaman ben burada ne düşünmüştüm gibi zorluklarla karşılaşabilir. Bu nedenle program yazılırken program kodunun en üstüne bu programın ne işe yaradığı hangi tarihte yazıldığı bilgileri ve program akışına göre çeşitli yerlerde açıklayıcı bilgiler verilerek sıkıntıların önüne geçilmiş olunacaktır. Ayrıca program kodu çalışırken bazı kodların derleyici tarafından dikkate alınmayarak ne gibi sonuçlar ürettiği incelenmek istenebilir, bu durumda bu kodların silinmesi yerine çalışması istenmeyen kodlar **/*** ve ***/** blokları arasına alınarak ihmal, istendiği zaman da bu işaretler kaldırılarak tekrar aktif edilebilir.

Yazılan programın bir sonraki satırda ise **#include <Kullanılan Kütüphaneler>** komutu görülmektedir. Bu satır yazılan C programı içerisinde birden fazla olabilir. Bu satırı anlayabilmenin yolu C dilinin çalışma prensibindedir. C dilini oluşturanlar herkesin sıklıkla kullandığı işlemleri tekrar tekrar yazılmaması için tek bir komut haline getirmişlerdir. Bu komutların kullanılabilmesi için bu kütüphanelerin programcı tarafından programa tanıtılması gerekmektedir.

Genel Kullanımı aşağıdaki gibidir.

#include <stdio.h>

Buradaki stdio.h, standart input output anlamına gelir ve yazılan C kodunda eğer programdan çıkış veya programa giriş yapılacaksa bu kütüphane kullanılmak zorundadır. Bu kütüphaneleri arttırmak mümkündür. Ayrıca C de tanımlı olmayan fakat ihtiyaç duyulan bir kütüphane programcı tarafından oluşturulabilir ve programlarına dahil edebilir. Fakat burada şöyle bir ayrım bulunmaktadır. Kullanıcı tarafından oluşturulan bir kütüphane kullanılacaksa **#include “KütüphaneAdı.h”** şeklinde ifade edilmektedir.

Bir sonraki satırda ise **#define Sabit ve Fonksiyonlar** satırda ise programda kullanılacak olan ve sık sık tekrar eden değer veya denklemler bu satırda tanımlanarak programın istenilen yerinde tekrar tekrar kullanılabilir.

Genel Kullanımı aşağıdaki gibidir.

#define PI 3,14

buradaki PI sabite verdiğimiz ad 3,14 ise bu sabitin aldığı değerdir,

#define f(x) x*x+3*x+2

bu tanımlamada f(x) fonksiyonun adı x*x+3*x+2 ise bu fonksiyonun denklemidir ve programın içinde istenildiği yerde x’ e bir değer verilerek kullanılabilir (a=f(3) şeklinde bir yazılım yazıldığı takdirde C programı yukarıda Define ile tanımlanmış fonksiyona 3 değerini gönderecek ve sonuç olarak a=20 olacaktır.) Define ile yapılacak tanımlamalar istenildiği kadar olabilir.

Bir sonraki satırda ise programda kullanılacak değişkenler ve tipleri tanımlanır. Bu tanımlamalar va tipleri bir sonraki konuda ele alınacaktır.

Bir sonraki satırdan itibaren C dilinde yazılan kodun ana program kısmının başlangıç yapılmıştır. C dilinde ana program yukarıdaki kütüphane ve tanımlamalar işlemlerinden sonra **main()** fonksiyonu ile başlar. Derleyici **main()** ifadesi ile karşılaştığında ana programın başladığını anlar. Bu main fonksiyonunun içersinde yazılan kodlar C programını oluşturacaktır.

Her C programında bir main() ana fonksiyonu kesinlikle bulunmalıdır ve program bu fonksiyon ile başlar ve bu fonksiyon ile son bulur.

C programı yazılırken her satır kesinlikle ";" noktalı virgül ile bitmek zorundadır.

C programında main() fonksiyonunu takip eden kod, fonksiyonun gövdesini oluşturur ve { } işaretleri arasında yer alır. Bu işaretlere program bloğu denir. Esas olarak "{" parantezi ile birden fazla fonksiyon birlikte kullanılabilir. Burada dikkat edilmesi gereken nokta her açılan program bloğu yani "{" ile başlayan kod "}" ile bitirilmelidir. Aksi takdirde program derleyicisi hata üretecektir.

2.4.2. C Program Tanımlamaları

Değişkenler, deyim etiketleri tip isimleri, fonksiyon isimleri v.b. programcı tarafından oluşturulan ifadelerdir. Bu isimlendirmelerde programcı bazı kurallarla uymak zorundadır. Bunlar,

- İfadeler ya bir harf veya "_" ile başlamalıdır. İfadeler rakam ile başlayamaz. Ger kalan karakterler rakam v.b olabilir.
- C tarafından kullanılan tip isimleri v.b. isimler kullanıcı tarafından kullanılamazlar.

Örneğin bir ifade **"12ayhan"** şeklinde kesinlikle olamaz iken **"ayhan12"** veya **"_12ayhan"** şeklinde kullanılabilir.

Noktalı Virgülün İşlevi

Noktalı virgül, ifadeyi sonlandırıcı işlev yapar. Şöyle ki:
x=xx+5;y=yy+5;
gibi iki ifadeyi birbirinden ayırır. Noktalı virgül olmadan yazılacak kod, anlamsız olacaktır:
x=xx+5
y=yy+5
İfadelerin arasında istenildiği kadar boşluk bırakılabilir, bu satır şu hale dönüştürülebilir:
x=a+2y=b+3 //???

Fakat şu asla unutulmamalıdır. C programlama dilinde her ifade kesinlikle noktalı virgül ile son bulur. C diline yeni başlayanların en çok karşılaşacakları yazım yanlışlarından birisi budur.Burada şu kesinlikle unutulmamalıdır. C programlama dilinde büyük küçük harf ayrımı bulunmaktadır.

"Ayhan ≠ AYHAN"

İfade isimleri oluşturulurken "_" alt çizi ile başlayan ifadelerden kaçınılmalıdır. Çünkü bazı C derleyicisine özgü anahtar sözcükler ile çakışabilmektedir.

2.4.2.1. Anahtar Sözcükler

C dilinde kullanılan en önemli anahtar sözcükler aşağıda listelenmiştir ve hepsi küçük harfle yazılmaktadır. Bu anahtar sözcükler kullanıcı tarafından ifadelerde kendi

tanımlamaları olarak kullanamazlar. Bu anahtar sözcükler, Veri tipleri, bellek sınıfları, deyim ifadeleri ve işleçler olmak üzere gruplara ayrılır. Bunlar aşağıdaki şekilde sınıflandırılır,

veritipi	char, const, double, enum, float, int, long, short, signed, struct, union, unsigned, void, volatile
deyim	break, case, continue, default, do
belleksınıfı	auto, extern, register, static, typdef, else, for, goto, if, return, swtich, while
işleç	sizeof

2.4.2.2. Değişmezler

C dilinde tamsayı, gerçek sayı, karakter ve diziler olmak üzere değişmezler bulunur.

- **Tam sayı değişmezler**

Tamsayı değişmezleri, ondalık, sekizli ve onaltılık sayı tabanlarında kullanılırlar ve derleyici programcı tarafından yazılan şekle göre sayı tabanlarının ayrımını yapar. C de bu veri tipleri **int, short int** ve **long int** olarak üç tipde tanımlanabilir.

123 : ondalık sayı

0123 : Sekizli sayı= 83 Ondalık sayıya karşılık gelir

085 : Geçersiz bir sayı

0xFF : onaltılık sayı=255 ondalıklı sayı

burada belirtilen sekizli sayı tabanı 0 ile 7 arasında ki rakamlardan oluşur, onaltılık sayı tabanı ise 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F değerleri alabildiği unutulmamalıdır. Normalde programcı tarafından belirtilen bu sayı tabanları ne olursa olsun derleyici tarafından bunlar ikilik sayı tabanına çevrilerek kullanılır.

- **Ondalıklı sayı değişmezler**

Ondalıklı sayıların kullanımı aşağıdaki şekilde kullanılabilir. C de bu veri tipleri **float, double** ve **long double** olarak üç tipde tanımlanabilir.

1.123

1.123E20 $=1.123x10^{20}$

1.123e20 $=1.123x10^{20}$

1.123e-20 $=1.123x10^{-20}$

- **Karakter değişmezler**

Karakter değişmezleri tek tırnaklar arasına yazılarak C dilinde kullanılır.

'A', 'a', '%' şeklinde kullanılır.

- **Karakter dizisi değişmezler**

Karakter dizisi değişmezleri çift tırnak arasına yazılır. C de “Ayhan AKBAL” şeklinde kullanılır. Karakter dizileri yazılırken C dilinde kullanılan bazı kaçış sıraları da kullanılabilir. Bu kaçış sıraları aşağıdaki gibi gruplandırılır.

\n :yeni satıra geç

\b :geri alma

\r :satırbaşı

\t :1 tab ileri hareket

\f :sayfa ilerletme

\a :beep sesi çıkart

\’ :tek tırnak

\” :çift tırnak

\? :soru işareti

\\ :ters slaş

şeklinde kaçışlar bulunur.

Örneğin C de ekrana adınızı soyadınızı alt alta yazmak istiyorsunuz bu durumda “**printf**” komutu kullanılarak karakter dizisi kullanılarak yapılabilir ve kullanımı ise aşağıdaki gibidir

Printf (“Ayhan \n Akbal “)

şeklinde bir yazım ifadesinde derleyici \n gördüğü yere kadar normal olarak yazılır ve \n den sonra bir alt satıra geçer ve yazar.

Ekran çıktısı

Şekil 3.2. Programın Ekran Çıktısı

Şeklinde olacaktır.

2.5. DEĞİŞKEN KAVRAMI VE TEMEL VERİ TİPLERİ

Değişken bir programcının programı içerisinde, programın aşamasına göre sürekli farklı değerler alabilen ifadeleridir. Örneğin, F=mxa şeklinde bir işlem yapılacak olsun,

“F”, “m”, “a” ifadeleri birer değişkendir, çünkü bu değişkenler her zaman farklı değerler alabilir. F=3x4, F=2x5 v.b şeklinde değerler alabilecektir. Dolayısıyla programcı bu değişkenleri önceden tanımlaması daha sonra programında kullanması gerekir.

Tanımlamada şöyle bir sıkıntı ortaya çıkacaktır. Bu değişkenler tamsayı mı?, ondalıklı sayımı? Yoksa bir karakter veya karakter dizisi mi? olacağı önceden programda belirtilmesi gerekir. Bu belirtmeleri yapabilmek için C dilinde ne tür veri tipleri bulunduğu ve veri tiplerinin hangi değerleri alabileceği önceden bilinmesi gerekir.

2.5.1. C Dilinde Kullanılan Veri Tipleri

Standart Veri Tipleri

Tüm diller bilgileri üzerinde çalıştığı bilgisayarın belleğinde saklanırlar. Saklanan bilgiler hafızada kapladığı yer ve tipine göre float, integer, double v.b. tipler olabilir. Her değişken bir veri tipi ile kesinlikle ilişkilendirilmelidir, bunun nedeni verilerin saklanması için gerekli saklama kapasitesini belirlenmesi ve belleğin en iyi şekilde kullanılmasıdır. Aşağıda C de kullanılan ver tipleri tablo halinde verilmiştir.

Veri Tipleri	**Veri Tipi Özelliği**	**Bit Sayısı**	**Alabildiği Değerler**	
			Minimum	**Maximum**
Unsigned char	İşaretisiz Karakter	8 bit	0	255
Char	Tek Karakter	8 bit	-128	127
enum	Sıralı Veriler	16 bit	-32.768	32.767
Unsigned int	İşaretsiz Tam Sayı	16 bit	0	65.535
Short int	Kısa Tam Sayı	16 bit	-32.768	32.768
İnt	Tam Sayı	16 bit	-32.768	32.768
Unsigned long	İşaretsiz Uzun Tam Sayı	32 bit	0	4.294.967.295
long	Uzun Tam Sayı	32 bit	-2.147.483.648	2.147.483.647
float	Ondalıklı Sayı	32 bit	$3.4x10^{-38}$	$3.4x10^{+38}$
Veri Tipleri	**Veri Tipi Özelliği**	**Bit Sayısı**	**Alabildiği Değerler**	
			Minimum	**Maximum**

Double	Uzun Ondalıklı Sayı	64 bit	$1.7x10^{-308}$	$1.7x10^{+308}$
Long double	Çok Uzun Ondalıklı Sayı	80 bit	$3.4x10^{-4932}$	$3.4x10^{+4932}$
Near pointer	Yakın İşaretçi	16 bit	-	-
Far pointer	Uzak İşaretçi	32 bit	-	-

Tablo 3.1: Standart Veri Tipleri

Karakter Tipleri

Tüm dillerde anlamlı yapılar oluşturabilmek için karakter dizileri kullanırlar. Örneğin Türkçe' de 29 harf 10 rakam kullanılır. Aynı şekilde C dilinin de tanıdığı bu şekilde karakterlerde vardır. Bunlar,

Harfler

ABCDEFGHIJKLMNOPRSTUVYZabcdefghijklmnoprstuvyz

Rakamlar

0123456789

Özel Semboller

+ - * / = , . _ : ; ? \ ” ’ $ & { } [ ] ( ) ^ @ < > # %-

ana başlıklar altında gruplandırılır. Bir sonraki bölümde değişkenler ve veri tipleri ayrıntılı bir şekilde anlatılacaktır.

2.6. DEĞERLENDİRME SORULARI

S1) C dili ilk olarak nere kullanılmıştır. a) Bilgisayarda b) UNIX işletim sisteminde c) Windows İşletim sisteminde d) ICC sistemlerde	S2) C dilinde açıklama satırları nasıl yazılır a) % ,% işaretleri arasına b) */ ,*/ işaretleri arasına c) /* ,*/ işaretleri arasına d) * ,* işaretleri arasına
S3) C dilinde kütüphaneler hangi komut ile programa tanıtılır? a) define b) #define c) #include d) main	S4) Yazılacak olan bir C dilinde epsilon diye bir sabit tanımlanacak ve bu sabitin değeri $8,85x10^{-12}$ dir buna göre C dilinde bu sabit nasıl tanımlanır? a) #define epsilon= 8.85x10^-12

	b) #include epislon 8.85x10_12 c) #define epsilon 8.85x10^-12 d) #main epsilon
S5) C diline göre aşağıdaki değişken tanımlamalarından hangisi yanlıştır? a) int PI b) char isim c) float kes123 d) int 123kes	S6) C dilinde tam sayı sabitler hangi tip ile ifade edilir? a) float b) char c) integer d) string

S7) C dilinde ekrana çıkış yapabilmek için hangi C komutu kullanılır?
a) scanf
b) printf
c) ineger
d) void main()
	S8) printf komutu kullanılırken bilgisayarın üç kere beep sesi çıkarabilmesi için yazılması gereken kod aşağıdakilerden hangisidir?
	a) printf("\n\n\n");
	b) printf("\a\a");
	c) printf("\a\a\a");
	d) printf("\t\t\t");

S9) C dilinde integer değişken tipleri hafızada ne kadar yer tutarlar? a) 8 Bit b) 32 Bit c) 16 Bit d) 63 Bit	S10) Bir programlama dilinde değişkenlerin tip tanımlamaları neden kullanılır? a) Programın akıcı olması için b) Programın bilgisayar belleğini optimum olarak kullanılması için c) Programcının kodu daha iyi anlayabilmesi için d) Programcının kodu doğru yazabilmesi için

DEĞİŞKEN, SABİT VE OPERATÖRLER

Bölüm 3

3.1. GİRİŞ

Çoğu programlama dilinde olduğu gibi C dilinde de değişkenler, sabitler ve operatörler kullanılmaktadır. Herhangi bir program yazıldığında mutlaka bunlardan biri veya daha fazlası mutlaka kullanılmak zorundadır. Çünkü yapılan işlemler esnasında çıkan sonuçların kimi yerde saklanması, işlemlerde kullanılabilecek bazı sabit değerlerin tanımlanması ve özellikle işlemleri yapabilmek için kullanılması gereken operatörlerin olması şarttır. Bu ve bunun gibi nedenlerden dolayı değişkenlere, sabitlere ve operatörlere ihtiyaç duyulur.

Değişkenler, sabitler ve operatörlerin neler oldukları, türleri ve nasıl kullanıldıkları bu bölümde detaylı olarak açıklanmış ve örneklerle bu açıklamaların uygulamaları da gösterilmiştir.

Bölümün Genel Amacı : Değişkenler, sabit ve operatörleri kullanmak.

Bölümün Davranışsal Amaçları : Bu bölümü başarıyla bitirip, örnekleri anlayarak ve bölüm sonundaki değerlendirme sorularını cevapladığınızda şunları öğrenmiş olacaksınız:

- C dilinde kullanılan değişkenler
- C dilinde kullanılan sabitler
- C dilinde kullanılan operatörler
- Değişkenleri, sabitleri kullanarak program geliştirme
- Operatörlerin işlemlerde nasıl kullanıldığı
- Değişkenlerin önemi

Değerlendirme: Bölüm sonundaki değerlendirme sorularını yapmanız ve sonuçlarınızı uygulama raporları ile karşılaştırmanız ve değerlendirme sorularına en az %75 doğru cevap vermeniz gerekmektedir.

3.2. DEĞİŞKENLER

Programlama dillerinin hepsinde kullanılan çok sayıda komut sayesinde yüzlerce işlem yapılır. Bu işlemlerin ara adımlarında bazı değerleri saklamak gerekebilir. Bu değerler ilerleyen adımlarda kullanılabilir yada başka bir amaç için gerekebilir. İşte bilgisayar aynı anda birden fazla işlem yaparken bazı verileri kullanırken diğer verileri hafızasında saklar. C programlama dilinde de verileri saklamak için verinin türüne göre tanımlanabilen ve kullanılabilen veri alanları vardır. Bu alanlar türüne göre verileri depolar ve başka bir sonuç bu alana aktarılabilir. Yani alan içerisindeki verinin değeri istenirse değiştirilebilir. Bu yüzden programlama dillerinde bu alanlara DEĞİŞKEN adı verilir. Çoğu programlama dillerinde değişkenler belli kurallara göre tanımlanır ve türleri vardır. Programlar yazılırken bu kurallara uymak gerekir. Aksi halde program çalıştırılamaz. C dilinde de kendine has kuralları vardır. Öncelikle değişken isimleri tanımlanırken bazı özel durumlara dikkat etmek gerekir. Bunlar;

1) Değişken isimleri bir harf veya "_" işareti ile başlayabilir.
2) İlk karakterden sonra harfler, rakamlar veya "_" işareti gelebilir.
3) Değişken ismi içerisinde boşluk bırakılmaz.
4) Değişken adı istenildiği kadar olabilir, ancak C bunun ilk 32 karakterini geçerli sayar.
5) Değişken adında büyük yada küçük harf kullanılabilir. Fakat büyük küçük harf duyarlılığı vardır. Yani, "adana" ile "ADANA" değişkenleri birbirinden farklıdır.
6) Kullanılacak değişken adı C diline ait bir komutun adı olamaz.
7) Değişken isimlerinde ingiliz alfabesi kullanılır. Değişken adında ç,ğ,ı,ö,ş,ü,Ç,Ğ,İ,Ö,Ş,Ü harfleri kullanılmaz.

Şekil 3.1'de değişkenleri anlatabilmek için küçük bir örnek verilmiştir.

Şekil 3.1: Değişkenler

Yanlış değişken isimlerine örnek şunları verebiliriz:

İSİM (İ harfi içeriyor)
Okul no (boşluk içeriyor)
3kisi (rakamla başlıyor)
3.kisi (rakamla başlıyor ve nokta içeriyor)
SHR (C dilinde kullanan bir komut)

C dilinde bir değişkeni tanımlamak genel olarak şöyledir;

Değişken_türü **değişken_adı**[=*sabit*] ;

Burada değişken türlerini bir alt konuda göreceğiz. Değişken adı kısmında da nelere uyulması gerektiğini yukarıda sıralamıştık. Sabit kısmı ise istenildiği takdirde verilebilir. Değişken tanımlama sırasında veya sonrasında değişkene uygun bir değer atanabilir. C dilinde tanımlı temel değişken tipleri vardır. Bunlar tablo 3.1 de görülmektedir.

TİP	UZUNLUK	DEĞER ARALIĞI
unsigned char	1 byte	0 / 255
char	1 byte	-128 / 127
enum	2 byte	-32768 / 32767
unsigned int	2 byte	0 / 65535
short int	2 byte	-32768 / 32767
int	2 byte	-32768 / 32767
unsigned long	4 byte	0 / 4.294.967.295
long	4 byte	-2.147.483.646 / 2.147.483.647
float	4 byte	$3,4*10^{-38}$ / $3,4*10^{38}$
double	8 byte	$1,7*10^{-308}$ / $1,7*10^{308}$
long double	10 byte	$3,4*10^{-4932}$ / $1,1*10^{4092}$

Tablo 3.1: C dilinde tanımlı temel değişken tipleri ve bunların hafızadaki uzunlukları ile alabilecekleri değer aralığı

Örnek 3.1 Değişken tanımlama

```
…
int  x,y;
float  m;
char  karakter;
char  ad[15];
long int buyuk;
```

x ve y adında iki tane tamsayı değişkeni tanımlandı. Her bir değişken 2 byte uzunluğunda.
m adında, hafızada 4 byte yer tutan reel sayı değişkeni tanımlandı.
karakter isimli 1 byte'lık tek karakterlik değişken tanımlandı.
tek boyutlu 15 haneli bir karakter dizisi tanımlandı.
buyuk isimli 4 byte'lık bir tamsayı değişkeni tanımlandı (int 'e göre çok daha büyük)

Örnek 3.2 Değişkenlere değer atama

```
#include<stdio.h>
#include<conio.h>
void main ( ){
clrscr ( );
int k,m,b;
k=7;
m=12;
b=k+m-2;
printf ("sonuç=%d \n", b);
k=m=b;
printf ("%d  %d  %d",k,m,b);
getch ( );
}
```

Programın işleyişi:
k,m,b adında 3 adet tamsayı değişkeni tanımlandı.
k değişkenine 7 değeri atandı. Bu atama tanımlama esnasında da yapılabilirdi.
m değişkenine 12 değeri atandı.
matematiksel bir işlem yapılıyor. *k* ile *m* değişkeni içerisindeki değerler toplanıyor ve bu değerden 2 sayısı çıkarılıyor. Sonuçta elde edilen sayı *b* değişkenine aktarılmış oldu.
b değişkeninin içeriği ekrana yazdırılıyor.
b değişkeninin içeriği *m* ve *k* değişkenlerine aktarılıyor .
önce *k*, sonra *m* ve en son *b* değişkeninin içeriği ekrana yazdırılıyor.
ekran görüntüsü herhangi bir tuşa basılana kadar donduruluyor.

Ekran çıktısı:

```
Sonuç=17
17  17  17
```

C dilinde tanımlanan değişkenler hafızada aldıkları yere göre statik ve dinamik, program içerisinde kullanıldığı yere göre lokal ve global değişkenler olarak gruplandırılabilir.

Statik değişkenler, tanımlandığı zaman hemen hafızada yer alırlar ve program sonlanana kadar yeri sabit kalır. Programın sonlandığında bellekte bu değişken için ayrılan yerler boşalır. Statik değişkenler tanımlandıkları anda herhangi bir değer ataması yapılmazsa ilk değeri başlangıçta sıfır olur. Statik değişkenlere başlangıç değeri verilirse, o değerler fonksiyon ilk çağrıldığında değişkene atanır; daha sonraki çağırmalarda başlangıç değeri verilmemiş gibi değerlendirilir.Eğer değişken tipi pointer (işaretçi) türünden ise bu durumda ilk değer sıfır değil NULL (boş) olur. Statik değişken **static** deyimi ile tanımlanır (Örnek 3.3).

Örnek 3.3 Statik değişken tanımlama

```
static int a;
a=a+k;
…
```

Dinamik değişkenler ise programın çalıştırılması sırasında oluşturulur. Bu değişkenler için heap bellek bölgesi ve bellek adresi kullanılır. C dilinde bu değişkenlerin tanımı malloc( ), calloc( ) gibi komutlarla olur. Dinamik değişkenler tanımlandıktan sonra heap bellek bölgesinden kullanılan alanların serbest bırakılması gerekmektedir.

Lokal ve global değişkenler ise; lokal değişkenler tanımlandıkları fonksiyon içerisinde kullanılırlar ve sadece burada kullanılırlar. Bu fonksiyonun dışında programda başka herhangi bir yerde kullanılamazlar. Yerel değişkenler tanımlandığı fonksiyon çalıştırıldığında bellekte yer kaplarlar (Örnek 3.4).

Global değişkenler, program içerisinde yer alan tüm fonksiyonların dışında tanımlanırlar ve programın istenilen her kısmında kullanılabilirler (Örnek 3.5). Bu tür tanımlamalar daha

çok, program içerisinde sıklıkla kullanılacak bir değişken varsa yapılır. Bu değişkeni global değil de, gerektiğinde her fonksiyon içerisinde yerel (lokal) olarak da tanımlayabiliriz. Ancak bu durumda çok fazla parametre yoğunluğu olacağından gereksiz yere hafıza işgal edilmiş olur. Bu da programın başarısızlığıdır. Bu nedenle aynı değişkeni ayrı yerlerde lokal olarak tanımlamaktansa, global olarak bir defa tanımlamak daha doğru olacaktır.

Örnek 3.4 Lokal değişkenler tanımlama ve kullanma

```
#include<stdio.h>
main ( )
{
int a,b,c;
a=2; b=5;

c=a*b;

printf("sonuç=%d",c);

}
```

üç adet tamsayı değişkeni tanımlandı (Lokal değişkenler).
a değişkenine 2, b değişkenine 5 değeri atandı.

a ile b tamsayı değişkenlerinin içeriği birbiriyle çarpılarak, sonuç c tamsayı değişkenine aktarılmıştır.

c değişkeninin içindeki değer ekrana yazdırıldı.

Ekran çıktısı:

```
sonuç=10
```

Örnek 3.5 Global değişkenler tanımlama ve kullanma

```
#include<stdio.h>
int a,b,c;
main ( )
{
int a,b,c;
a=2; b=5;
c=a*b;
printf("sonuç=%d",c);
}
```

üç adet tamsayı değişkeni tanımlandı (Global değişkenler).

Ekran çıktısı :

```
sonuç=10
```

3.3. SABİTLER

Program içerisinde kullanılacak sabit değerli değişkenler olabilir. Bunun için C dilinde yine değişken tanımına benzer şekilde sabitlerde tanımlanır. Bunun için değişkenin adının önüne **const** deyimi kullanmak gerekir. Sabitler tanımlandıkları andan itibaren program içerisinde işlemlerde kullanılabilir. Ancak içeriği değiştirilemez (Örnek 3.6).

Örnek 3.6 Sabit tanımlama

`const float pi=3.142857;`	matematikte kullanılan pi sayısı tanımlandı.
`const double e=2.71828182845905;`	matematikte kullanılan e sayısı tanımlandı
`const char [ ]= " Adınızı giriniz";`	Adınızı giriniz şeklinde bir karakter dizisi

3.4. C DİLİNDE KULLANILAN OPERATÖRLER

Operatörler, programlama dillerinde matematiksel, mantıksal sınama ve karşılaştırma gibi işlemler için kullanılan bazı alfa nümerik karakterlerdir. Operatörlerin tam olarak ne iş yaptığı programı yazan kişi tarafından çok iyi bilinmelidir. Aksi halde çok küçük bir hata, çok büyük problemlere neden olabilir. Operatörler dört temel grupta toplanabilirler. Bunlar; aritmetiksel, karşılaştırma, mantıksal ve atama operatörleridir.

3.4.1. Aritmetiksel Operatörler

Değişkenler veya sabitler üzerinde temel aritmetik işlemler için kullanılırlar. Bu operatörler şunlardır:

+ toplama
– çıkarma
***** çarpma
/ bölme
% artık bölme (mod işlemi: bir sayının diğer bir sayıya bölümünden kalanını bulma)
– – bir azaltma
++ bir arttırma

Aritmetiksel operatörler, tamsayı veya karakter değişkenleri için kullanılabilir. Ancak dikkat edilmelidir. Çünkü karakter değişkeni için örneğin ++ operatörü kullanıldığında, bu değişkenin içeriğindeki ASCII değeri 1 arttırılır. Bu da farklı bir karakter demektir. Ancak değişken tamsayı değişkeni ise bu değişken içindeki sayı değeri kaç ise ++ operatörü kullanıldığında, bu sayı 1 artar. Aritmetiksel operatörler için değişik kullanımların gösterildiği küçük program parçacıkları örnek 3.7 de verilmiştir. Operatörün herhangi bir eşitlik ifadesi olmadan tek başına değişkenin sonunda veya başında kullanılmasının bir farkı yoktur.

Örnek 3.7 Aritmetiksel Operatörleri kullanma

```
#include<stdio.h>
main ( )
{
int a=7,b=12,c;
char k='A';
a– –;
++b;
k++;
c=a%b;
}
```

a,b,c tamsayı değişkenleri tanımlandı, a ile b ye değerleri atandı.
k karakter değişkeni tanımlandı ve içine A karakteri yerleştirildi.
a değişkeninin içerisi bir azaltıldı (a=6 oldu.)
b değişkeninin içerisi bir arttırıldı (b=13 oldu).
k değişkeninin içerisi bir arttırıldı (k'nın içi B oldu, ASCII olarak A dan sonra B geldiğinden dolayı a değişkeninin içeriği b değişkeninin içindeki sayıya bölünerek, kalan değeri c tamsayı değişkenine aktarılmış oldu).

3.4.2. Karşılaştırma Operatörleri

Bu operatörler değişken veya sabitleri birbirleriyle karşılaştırmada kullanılır. Yani sayısal veya karakter değişkenler birbiriyle karşılaştırılabilir ancak, karakter dizileri karşılaştırılamaz. Böylelikle değişik şartların yerine getirilmesi sağlanarak döngüsel veya şartlı ifadelere bağlı programlarda kullanılabilir. Karşılaştırma operatörlerini anlatmak için örnek 3.8'e bakılabilir. C dilinde kullanılan karşılaştırma operatörleri ise şunlardır:

>	büyük mü ?
>=	büyük veya eşit mi?
<	küçük mü ?
<=	küçük veya eşit mi?
= =	eşit mi?
! =	farklı mı ?

Karşılaştırma operatörleri az öncede değinildiği gibi şartlı ifadelerde ve döngülerde çok kullanılırlar. Karşılaştırma sonucunda doğru (true) yada yanlış (false) değeri gönderilir. Örneğin if 'li bir ifadede şart doğru ise blok içerisindeki işlemler yapılırken, şartın sonucu yanlış ise bu blok atlanır.

Örnek 3.8 Karşılaştırma Operatörlerini kullanma

Programa bir kişinin dışarıdan bir dersinin vize ve final notunu girmesini isteyerek bu kişinin ortalamasını hesaplayalım. Ve kişinin ortalama notu 60 dan küçükse "sınıfta kaldın", tam tersi notu 60'ın üstünde ise "sınıfını geçtin" yazsın.

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

int  vize, final;
float  ortalama;


main ( ) {

clrscr ( );

printf (“Lütfen vize notunuzu giriniz….:”);
scanf (“%d”, &vize);

printf (“\n Lütfen final notunuzu giriniz….:”);
scanf (“%d”, &final);

ortalama=vize*0.4+final*0.6;

if (ortalama<60) { printf (“\n Sınıfta kaldın”);

else { printf (“\n Sınıfı geçtin”); }

getch ( );
}
```

Kullanılacak kütüphaneler tanımlandı (açıldı).
vize, final adında iki tamsayı değişkeni tanımlandı.
ortalama adında reel sayı değişkeni tanımlandı (sonuç tam çıkmayabileceğinden dolayı float olarak tanımlamakta yarar vardır).
ana fonksiyon tanımlandı.
ekran temizlendi.
dışarıdan girilen vize notu değeri **vize** değişkeninde saklandı.
dışarıdan girilen final notu değeri **final** değişkeninde saklandı.
ortalama not hesaplandı, sonuç **ortalama** değişkeninde saklandı.
ortalama değişkeni mantıksal olarak 60 sayısıyla karşılaştırıldı, sonuç doğruysa yani ortalama değişkeni içindeki değer 60 dan küçükse “Sınıfta kaldın”, değilse “Sınıfı geçtin” yazısı yazacaktır.

Ekran çıktısı :

```
Lütfen vize notunuzu giriniz….:35
Lütfen final notunuzu giriniz….:60
Sınıfta kaldın
```

Örnek 3.9: Karşılaştırma Operatörlerini kullanma
Bu program dışarıdan girilen bir tamsayının pozitif mi?, negatif mi? yoksa sıfır mı? olduğunu ekrana yazar.

```
# include<stdio.h>
# include<conio.h>
main ( ){
int sayi;
printf (“bir sayı giriniz :”);scanf(“%d”,&sayi);
if (sayi<0) { printf(“sayı negatif”); }
else if (sayi>0) { printf(“sayı pozitif”); }
else { printf(“sayınız sıfırdır”); }
getch ( );}
```

3.4.3. Mantıksal Operatörler

Bu operatörler, şartlı ifadeler ve döngülerde birden fazla sınanması gereken değişken veya sabit olması durumunda kullanılırlar. Bu operatörler şunlardır:

&&	VE	işlemi (AND)
||	VEYA	işlemi (OR)
!	DEĞİL	işlemi (NOT)

Örnek 3.10: Mantıksal Operatörleri kullanma
Programa bir kişinin dışarıdan bir dersinin vize ve final notunu girmesini isteyerek bu kişinin ortalamasını hesaplayalım. Ve kişinin ortalama notu 60 dan küçükse "sınıfta kaldın", tam tersi notu 60'ın üstünde ise "sınıfını geçtin" yazsın. Ancak bu kez final notunun da 60 dan küçük yada büyük olduğunu da kontrol etsin.

```
# include<stdio.h>
# include<conio.h>
int  vize, final;
float  ortalama;

main ( ) {

clrscr ( );

printf ("Lütfen vize notunuzu giriniz….:");
scanf ("%d", &vize);

printf ("Lütfen final notunuzu giriniz….:");
scanf ("%d", &final);

ortalama=vize*0.4+final*0.6;

if (ortalama<60 || final<60) { printf ("\n Sınıfta kaldın");

else { printf ("\n Sınıfı geçtin"); }

getch ( );
}
```

3.4.4. Atama Operatörleri

Bir değişkenin içerisine (kabul edilebilir) herhangi bir değeri eşitlemek için atama operatörleri kullanılır. Bu değer bir sabit değer olabileceği gibi, herhangi bir işlemin sonucu da olabilir.

Örnek 3.11 Atama Operatörlerini kullanma

int a;	int a=2,b=5,c;
a=30;	c=a*b+10;
(a)	(b)

Örnek 3.11'de de görüldüğü gibi birinci kısımda (a) verilen atama şekli direkt bir sabit değerin değişkene atanmasıdır. İkinci kısımda ise (b) matematiksel bir işlemin sonucu bir değişkene aktarılmıştır. Burada atama işlemi için görüldüğü gibi tek başına **=** işareti kullanılmıştır. Ancak C dilinde, sadece tek başına = işareti değil çift (bitişik) atama operatörleri de sıklıkla kullanılır. Bu operatörler ve anlamları şunlardır:

OPERATÖR	**ANLAMI**
+=	toplayarak atama
–=	çıkararak atama
*=	çarparak atama
/=	bölerek atama
%=	bölerek kalanını atama
&=	bit düzeyinde VE işlemi yaparak atama
|=	bit düzeyinde VEYA işlemi yaparak atama
=~	bit düzeyinde tümleme ve atama
<<=	sola öteleyerek atama
>>=	sağa öteleyerek atama

Örnek 3.12 Bitişik atama operatörlerini kullanma

```
int a=5, b=2;   // a ve b adında iki tamsayı değişkeni tanımlanarak, değerleri atandı.
b*=a;           // b ie a değişkeninin içeriği çarpılarak, sonuç b değişkenine aktarıldı.
…
```

Bu örnekte b*=a yerine b=b*a; veya b=a*b; de yazılsaydı, sonuç yine aynı olurdu. Fark, sadece ilk yazılan deyimde bitişik atama operatörü kullanılmış ve ifade biraz daha kısaltılmış oldu. C dilinde bu üç ifadede doğru işlem yapmaktadır.

Örnek 3.13 Bitişik atama operatörlerini kullanma

1 den 25 'e kadar olan ardışık sayıları toplayan C programını yazalım (Burada kullanılan **for** döngüsü ileride anlatılacaktır).

```
# include<stdio.h>
# include<conio.h>
main ( ) {
int i,toplam=0;
printf (“Bu program 1-25 arası sayıların toplamını verir”);
for(i=1;i<=25;i++){  // i değişkeni, başlangıç olarak 1 den başlayıp 1’er artarak
toplam+=i;        // 25’e kadar gider. Döngü içerisinde i 1 artar, i ‘nin her değeri toplam
}          // değişkeninin üzerine eklenir. Böylece ardışık tüm sayılar toplam
          // değişkeni üzerine eklenir, toplam değişkeni de başlangıçta 0 değerinde
           // olduğundan sonucu etkilemez.

printf (“\n toplama sonucu = %d dir”,toplam);      // toplama sonucu ekrana yazdırılıyor.
getch ( );}
```

Bu operatörlerin dışında kalan üç operatör daha vardır. Bunlar **?:, *, &** operatörleridir. ?: operatörü ileride anlatılacak olan if-else yapısına benzer şekilde belirtilen şarta göre işlem yaptırmada kullanılır. Bunun genel şeklini şöyle yazabiliriz:

(koşul ifadesi) **?** deyim1 **:** deyim2;

Bu ifadeye göre verilen koşul ifadesinin sonucu doğru (true) ise deyim1, yanlış (false) ise deyim2 icra edilir. Buradaki deyim1 ve deyim2, herhangi bir atama işlemi, matematiksel bir işlem veya genelde bir fonksiyon olabilir. Örneğin, aşağıdaki ifadede a değişkeninin içeriği 10’dan büyükse b değişkenine 2, değilse -2 değeri aktarılmaktadır.

(a>10) **?** b=2 **:** b=-2;

***** operatörünün aritmetiksel operatör olarak çarpma işlemi yaptığını biliyoruz. Aynı zamanda bu operatör pointer’larda (işaretçi), **&** operatörü ile de kullanılır. * operatörü işaretçi değişkenin tanımlanmasında kullanılır.

Örnek 3.14 Pointer değişkeni tanımlama

```
int a, *b;        // a adlı tamsayı değişkeni ve bir tamsayı atanacak hafıza
                  bölgesinin adresinin
                  // tutulacağı b işaretçi değişkeni tanımlandı.

float *c;         // bir reel sayı atanacak hafıza bölgesinin adresinin
                  tutulacağı c işaretçi değişkeni tanımlandı.
```

Yukarıdaki örneklerde görüldüğü gibi ***** operatörü işaretçi değişkenin tanımlanmasında kullanılmıştır. Bunun yanında & operatörü de vardır ki, bu operatör işaretçinin gösterdiği adresle ilgilenir (içeriğiyle ilgilenmez).

Örnek 3.15 & operatörünün kullanımı

```
int  *p, x=4, y;
y=x;
p=&x;
```

Yukarıdaki örnekte p işaretçi değişeni, x ve y tamsayı değişkenleridir. x'in başlangıç değeri 4 verilmiştir. İkinci satırda da y ile x eşitlenmiş ve y’nin değeri de 4 olmuştur. Üçüncü satırda ise, x değişkenin başlangıçta bulunduğu hafıza bölgesinin adresi p işaretçi değişkenine aktarılmıştır.

***** operatörü aynı zamanda bir işaretçi değişkeninin gösterdiği adresteki değeri, normal bir değişkene aktarmada da kullanılır.

Örnek 3.16 * operatörünün kullanımı

```
int  *p,m,n;
p=&m;          // m nin adresi p ye aktarıldı.
*p=100;        // p nin gösterdiği adresin içine 100 değeri atandı.
n=*p;          // p nin gösterdiği adresin içeriği n değişkenine atandı.
```

DEYİMLER

Bölüm 4

4.1. GİRİŞ

C programlama dilinde program kodu yazabilmek dahası bu kodların bizi değişik amaçlarımıza ulaşmamızı sağlayacak hale getirmek için şart ve döngü deyimlerini iyi bilmemiz ve etkili bir biçimde kullanmamız gereklidir. Şu bir gerçektir ki bir bilgisayar programının rutin ve basit ve işlemler yapması fazla işe yarayacak bir durum değildir. Bazen programımızın akışı bir şarta göre değişebilmeli bazen belli işlemler tekrar tekrar yapılabilmelidir. İşte bu tür işlemleri yapabilmek için bu bölümde gereken bilgileri alacak, şart ve döngü deyimlerini kullanacağız.

BÖLÜM -4- DEYİMLER

Bölümün Genel Amacı: Deyimleri bilme ve program içerisinde kullanma.

Bölümün Davranışsal Amaçları: Kitabınızın bu bölümünü başarıyla bitirip, örnek uygulamaları yaptığınız taktirde, bölüm sonunda;

- Deyim kavramını açıklamanız,
- Şart kavramını açıklamanız,
- İf şart deyimini kullanmanız,
- Switch deyimini kullanmanız,
- Koşul operatörünü kullanmanız,
- Goto deyimini kullanmanız,
- While döngüsünü kullanmanız,
- For döngüsünü kullanmanız,
- Break ve continue sözcüklerini kullanmanız beklenmektedir.

Değerlendirme: Modül sonundaki uygulamaları yapmanız sonuçları uygulama raporu ile karşılaştırmanız ve değerlendirme sorularına en az % 75 düzeyinde doğru cevap verebilmeniz gerekmektedir.

4.2. DEYİM NEDİR?

c' de ifadeler birbirleriyle noktalı virgül ile ayrılırlar. İfadeleri bitirmesi nedeniyle, noktalı virgüle sonlandırıcı (terminator) da denilmektedir.

Deyim (statement) ifade ile sonlandırıcıdan oluşan gruba verilen addır.

Örneğin:

c = a * b – d bir ifadedir; ancak

c = a * b - d; bir deyimdir.

C'de deyimler 4 bölüme ayrılır:

1) Yalın deyimler: Bunlar bir ifadenin sonuna sonlandırıcı (;) getirilerek oluşturulan deyimlerdir.

Örneğin:

x = y *5;

a++;

b = getch( );

gibi...

2) Bileşik deyimler: Birden fazla deyimin bir blok içinde toplanmasıyla oluşan program parçalarına bileşik deyim adı verilir. c' de bloklar aynı zamanda bileşik deyimlerdir.

{

deyim1 deyim2 deyim3

}

örneğin

{

c = a /2 ; b++;

}

3) Bildirim deyimleri: Bildirim amacıyla oluşturulmuş deyimlerdir:

Örneğin;

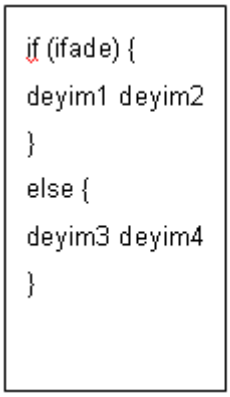
int d,temp, sayi;

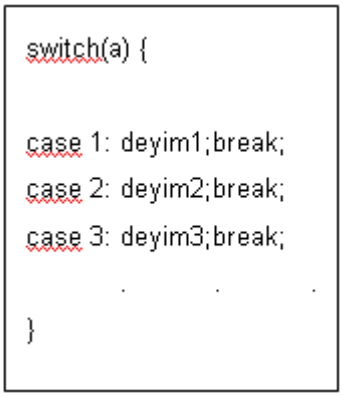
char ad;

float uzunluk, genislik;

gibi...

4) Kontrol deyimleri: Program akışını kontrol etmek için kullanılan deyimlerdir. Kontrol deyimleri en az bir anahtar sözcük içerir; bir ya da birden fazla blokdan oluşabilir.





Solunda bir ifade olmaksızın yalnızca bir sonlandırıcıdan oluşan deyime, **boş deyim (null statement)** denir. Derleyiciler boş deyimler için hiçbir işlem yapmazlar.

Örneğin:

```
a = b * 2;
;
c = a + 4;
```

4.3. İF DEYİMİ

C' de program akışını kontrol etmeye yönelik en önemli deyim if deyimidir. if deyiminin genel biçimi aşağıda verilmiştir.

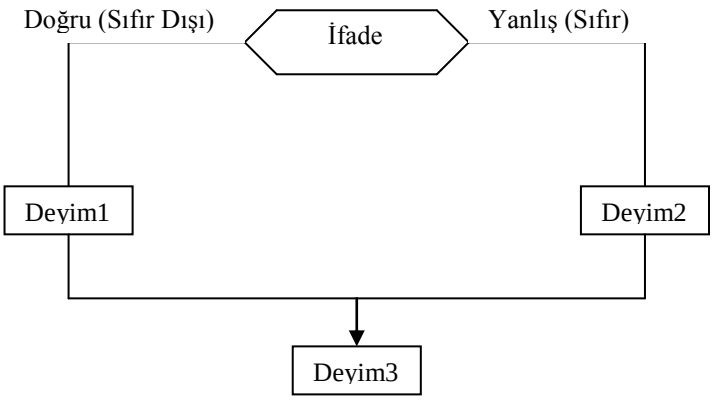
```
if (ifade)
        deyim1
else
        deyim2
```

deyim1 ve deyim2, yalın ya da bileşik olabileceği gibi başka bir kontrol deyimi de olabilir.

if deyiminin icrasında, önce derleyici if parantezinin içindeki ifadenin sayısal değerini hesaplar. Hesapladığı bu sayısal değeri mantıksal Doğru ya da Yanlış olarak yorumlar (0 ise Yanlış, 0 dışında bir değer ise Doğru). Eğer ifadenin sonucu Doğru ise, else anahtar sözcüğüne kadar olan kısım; Yanlış ise, else anahtar sözcüğünden sonraki kısım icra edilir.

Akış diyagramı ile gösterirsek:

```
if (ifade)
        deyim1
else
        deyim2
deyim3
```



NOT: Deyim3, if deyiminden sonra gelen ilk deyimdir:

Örneğin aşağıdaki program parçasının işleyişini adım adım görelim;

```
a=8;
…..
if (a * 5 < 50)
        deyim1
else
        deyim2
deyim3
……
```

1. Adım: if parantezinin içindeki ifadenin sayısal değeri hesaplanır.

a * 5 =40

40 < 50

Sonuç=1= DOGRU

2. Adım: ifade Doğru olduğuna göre, else anahtar sözcüğüne kadar olan kısım (deyim1) yapılır.

3. Adım: else kısmı atlanarak deyim3 yapılır.

if parantezi içindeki ifadeler daha karmaşık olabilirler:

```
harf = 'a';

if (harf >= 'A' && harf <= 'Z')
        deyim1
else
        deyim2
deyim3
```

Yandaki if deyiminde harf karakterinin büyük harf olup olmadığı test edilmektedir. Eğer harf büyük harf ise deyim2, küçük harf değilse deyim2 yapılır; daha sonra akış deyim3 ile devam eder.

1. Adım: if parantezinin içindeki ifadenin sayısal değeri bulunur.

harf >= 'A' ========0 harf <= 'Z'========0

0 && 0 =========0

Sonuç: YANLIŞ

2. Adım: Sonuç yanlış olduğu için else kısmı (deyim2) yapılır.

3. Adım: Deyim3 ile akışa devam edilir.

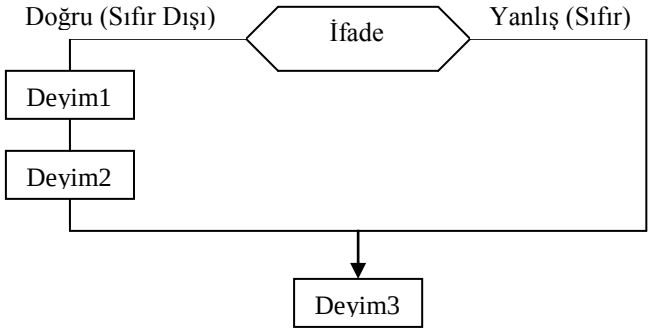
if deyiminin Doğru ve/veya Yanlış kısmı birden fazla deyimden oluşuyorsa (bileşik deyimse) bloklama yapılmalıdır.

```
if (c < d - 4) {
        deyim1
        deyim2
} else {
        deyim3
        deyim4
}
```

Bu örnekte c < d - 4 ifadesi Doğruysa deyim1 ve deyim2, Yanlışsa deyim3 ve deyim4 yapılır.

Bir if deyiminin else kısmı olmayabilir. Örneğin:

```
if (ifade) {
        deyim1
        deyim2
}
deyim3
```



if parantezinin içerisindeki ifade doğruysa deyim1 ve deyim2 yapılır; daha sonra programın akışı deyim3 ile devam eder. Eğer ifade yanlış ise blok atlanarak doğrudan deyim3 yapılır.

Bir if deyimi yalnızca else kısmına sahip olamaz. Bu durumda en uygun yol koşul ifadesini değiştirmektir.

```
if (ifade)
else
deyim1
```

şeklinde bir yazım hatalıdır

Yukarıdaki hatalı olan if deyiminin yerine aşağıdaki eşdeğerleri i kullanılabilir:

if (**!**ifade)	if (ifade)
deyim1	;
	else
	deyim1

if parantezinin içerisindeki ifadede değişken olması gibi bir zorunluluk yoktur.

if (10)

deyim gereksiz bir tarz olsa da yandaki gibi bir yazımı da kullanabiliriz.

4.4. switch DEYİMİ

if deyimi Doğru ya da Yanlış olmak üzere iki seçeneğe sahiptir. Oysa switch deyimi belli bir ifadenin çeşitli sayısal değerlerine karşı farklı işlemlerin yapılması için kullanılmaktadır. Genel biçimi aşağıdaki gibidir:

```
switch (ifade) {
        case <sabit ifadesi1>:   [break];
        case <sabit ifadesi2>:   [break];
        cas e <sabit ifadesi3>:  [break];
        [default:          ;]
}
```

switch, case ve default birer anahtar sözcüktür. Derleyici switch parantezinin içerisindeki ifadenin sayısal değerini hesaplar. Eğer bu sayısal değere eşit olan bir case ifadesi bulursa programın akışını oraya yönlendirir. Eğer böyle bir ifade yoksa programın akışı default ile belirtilen kısma geçer. Örneğin:

```
switch(n) {
case 1: printf("Pazartesi\n");
case 2: printf("Salı\n");
case 3: printf("Çarşamba\n");
case 4: printf("Perşembe\n");
case 5: printf( "Cuma \n) ;
default: printf("Hiçbiri\n");
}
```

burada n=1 ise programın akışını case 1: kısmına yöneltilir. Akış buradan sırasıyla aşağı inerek devam edecektir. Dolayısıyla ekranda:

Pazartesi

Salı

Çarşamba

Perşembe

Cuma

Hiçbiri

yazılarını görürsünüz. Döngülerden çıkmak için kullanılan break anahtar sözcüğü switch içerisinden çıkmak için de kullanılabilir. Bu durumda bir case ifadesi break ile sonlandırılırsa programın akışı diğer case ifadelerine geçmez. Yukarıdaki örnekteki her bir case in sonuna break koyarsak örneğimiz şu şekilde değişecektir.

```
switch(n) {
case 1: printf("Pazartesi\n");break;
case 2: printf("Salı\n"); break;
case 3: printf("Çarşamba\n"); break;
case 4: printf("Perşembe\n"); break;
case 5: printf( "Cuma \n) ; break;
default: printf("Hiçbiri\n"); break;
}
```

bu program bir öncekinin aksine tüm ifadeleri değil sadece n değerinin karşılığı olan günü yazacaktır. Yani n değeri 3 ise Çarşamba, 1 ise pazartesi, 12345 dışında bir değerse hiçbiri yazacaktır.

case anahtar sözcüğünün yanındaki ifadenin sabit ifadesi olması zorunludur. Çünkü derleyici derleme zamanında case ifadelerinin sayısal değerini hesaplamak zorundadır. Aşağıdaki örneği inceleyiniz:

```
switch (x - 1) {
        case 8  :       break;
        case 12:        break;
        case y-3 :      break;
        default  :      ;
}
```

Buradaki son case sabit ifade olmadığı için switch geçersizdir.

Birden fazla sayısal değer için aynı işlemlerin yapılması isteniyorsa aşağıdaki gibi bir yol izleyebilirsiniz. Bunun daha kısa bir yapılış yöntemi yoktur.

```
switch (a) {
case 3:
case 5: b=a+c; break;
case 8: b=a-c; break;
default: printf(" geçersiz işlem!.. \n");
}
```

Yukarıdaki örnekte a değişkeninin 3 ve 5 değerleri için aynı işlemler yapılmaktadır.

default anahtar sözcüğünün sonda olması ve case sabitlerinin artan sırada olması bir zorunluluk değildir. Yani yukarıdaki örnek şöyle de düzenlenebilirdi.

```
switch (a) {
default: printf("Geçersiz İşlem!..\n"); break;
case 8:b=a-c; break;
case 5: case 3:b=a+c;
}
```

switch deyiminin son case ifadesi için break gerekmediğine dikkat ediniz. Diğer kontrol deyimlerinde olduğu gibi switch deyimi de yalnızca bir deyim içeriyorsa bloklama yapmaya gerek yoktur. Örneğin:

```
switch(n)
case 11:
case 13: x = n - 1; z = n / 2;
```

burada switch içerisinde yalnızca x = n - 1; deyimi vardır. z = n / 2; dışındaki ilk deyimdir.

4.5. KOŞUL OPERATORU

Koşul operatörü C'nin karşılaştırma yapan özgün bir operatörüdür. Şartları sıradan birer ifade olabilir. Genel kullanım biçimi aşağıdaki gibidir:

ifade1 ? ifade2 : ifade3

Koşul operatörü ? ve : olmak üzere ayrık iki sembolden oluşur. Derleyici önce ? karakterinin solundaki birinci ifadenin sayısal değerini hesaplar. Genel biçimde bunu ifade1 olarak isimlendirdik. Eğer ifade1 sıfır dışı bir sayısal değere sahipse bu durum koşul operatörü tarafından Doğru olarak ele alınır ve yalnızca ifade2 yapılır. Eğer itade1

in sayısal değeri sıfır ise bu kez yalnızca ifade3 yapılacaktır. Diğer operatörlerde olduğu gibi koşul operatöründen de bir değer üretilir. Bu değer koşulun sağlanması durumuna göre ifade2 ya da ifade3 olabilir. Örneğin:

```
y = (x > 3 ? a + 1 : a – 1);
```

Burada önce x > 3 ifadesinin sayısal değeri hesaplanır. Bu ifade sıfır dışı bir değerse (yani doğruysa) y değişkenine a + 1 sıfır ise (yani yanlışsa) a - 1 atanacaktır. Aynı işlem if ile de yapılabilirdi:

```
if (x > 3)
y = a + 1; else
y = a - 1;
```

Koşul operatörü birçok durumda okunabilirliği kuvvetlendirmek için tercih edilir. Örneğin:

```
z=  (a > b) ? a : b;
```

z değişkenine max(a, b) atanmaktadır. Programcılar algılamayı kolaylaştırdığı için koşul operatörürünün ilk operandını parantez içerisine alırlar. Ancak böyle bir zorunluluk yoktur.

4.6. go to DEYİMİ

Yapısal programlama dillerinde programın akışının başka bölgelere gönderilmesi algılamayı ve tasarımı zorlaştırdığı için pek salık edilmez. goto anahtar sözcüğüprogram akışını istenilen bir bölgeye yönlendirmek için kullanılır. goto yanına bir etkiket ismi verilerek kullanılmalıdır.

```
goto <etiket ismi>;
```

Etiket, değişken isimlendirme kuralına uygun herhangi bir isim olabilir. Programın akışı etiket ile belirtilen bölgeye gider.

```
goto REPEAT;
```

REPEAT: goto etiketinin faaliyet alanı fonksiyon faaliyet alanıdır. Yani go to ile başka bir fonksiyonun bir bölgesine atlama yapılamaz. Aşağıdaki örnekte klavyeden alınan karakter q olduğunda içiçe iki döngüden çıkılmıştır.

```
#include <stdio.h>
void main(void) {
int i, j; char ch;
for(i = O; i < 10; ++i)
        for (j = O; j < 10;++j){
        ch = getch();
        if (ch == 'q')
        goto EXIT;
        printf("i = %d j= %d \n",i ,j ) ;
} EXIT:
}
```

4.7. while DÖNGÜSÜ

Diğer dillerde olduğu gibi C'de de while döngüleri koşul sağlandığı sürece yinelemeye neden olmaktadır. while döngülerini iki gruba ayırabiliriz:

1) Kontrolün başta yapıldığı while döngüleri

2) Kontrolün sonda yapıldığı while döngüleri (do - while) Sırasıyla inceleyelim.

4.7.1. Kontrolün Başta Yapıldığı while Döngüleri

Genel kullanımı aşağıdaki gibidir:

```
while  (ifade)
        deyim
…
```

while bir anahtar sözcüktür. C derleyicileri while anahtar sözcüğünden sonra parantezler arasında bir ifade bekler. Döngü bu ifadenin Doğru (sıfır dışı bir değer) olduğu sürece

yinelenir. Genel biçimde belirtilen deyim; yalın, bileşik ya da başka bir kontrol deyimi olabilir. Daha açık bir deyişle, if deyiminde olduğu gibi, eğer bir tek deyim söz konusuysa bloklama yapılmasına gerek yoktur; ancak birden fazla deyim blok içine alınmalıdır. Örneğin aşağıdaki while döngüsünde yalnızca deyim1 döngü içerisindedir. Döngü çıkışındaki ilk deyim ise deyim2' dir.

```
while  (ifade)
deyim1
deyim2
```

Aşağıdaki örnekte ise deyim1 ve deyim2' nin her ikisi de döngü içerisindedir. deyim3 döngüden sonraki ilk deyimdir.

```
white (ifade)  {
deyim1 .
deyim2
}
 deyim3
```

while döngüsünün içerisindeki deyim, herhangi bir kontrol deyimi de olabilir.

```
white (ifade1)
        if (ifade2)
                        deyim1
        else
                        deyim2

deyim3
…
```

Bu örnekte bloklama yapılmadığı için while içerisinde yalnızca if deyimi vardır. Deyim3, döngü çıkışındaki ilk deyimdir.

while döngüsü, "parantez içindeki ifadenin sayısal değeri sıfır dışı bir değer (Doğru) olduğu sürece yinelemeye neden olur" demiştik; örneklerle açıklayalım:

```
t =1;
while  (t <= 20) {
          printf( I/%d\n", t);
          ++t;
          }
```

Yukarıdaki while döngüsü t değişkeni 20'den küçük ya da eşit olduğu sürece yinelenecektir.

t değeri	t<=20 şartının sonucu	
1	1	Doğru, döngüye devam
2	1	Doğru, döngüye devam
3	1	Doğru, döngüye devam
...	...	
20	1	Doğru, döngüye devam
21	0	Yanlış, döngüden çık!..

t, döngünün içerisinde artırıldığına göre 20 yineleme sonra döngünün sonlanacağını söyleyebiliriz.

Bir başka while döngüsünü incelersek:

```
while  ((tus = getchar( )) ! = ‘q')
printf ("%c\n", tus);
```

Bu örnekte while döngüsünün devam edebilmesi için klavyeden girilen karakterin 'q' olmaması gerekir. Veya bir başka deyişle: Bu döngü 'q' karakterine basıldığında sonlanır!

4.7.2. Kontrolün Sonda Yapıldığı while Döngüleri

Bu tür while döngülerinde kontrol sonda olduğu için döngü içindeki deyimler en az bir kere işlem görür. Genel kullanımı şu şekildedir:

```
do
        deyim
        while  (ifade);
```

do, döngünün başını gösteren bir anahtar sözcüktür. Döngünün iç bölgesi do anahtar sözcüğünden while anahtar sözcüğüne kadar olan bölgedir. Diğerlerinde olduğu gibi, tek bir deyim için bloklamaya gerek yoktur; fakat birden fazla deyim için bloklama yapılmalıdır. Örneğin:

```
do
deyim1
while  (ifade);
deyim2
```

Burada yalnızca deyim1 döngünün içerisindedir.

```
do {
        deyim1
        deyim2
        } while  (ifade);
deyim3
```

Bu örnekte ise bloklama yapıldığı için deyim1 ve deyim2' nin her ikisi de döngünün içerisindedir. Deyim3, döngünün dışındaki ilk deyimdir.

4.8. for DÖNGÜLERİ

C deki for döngüleri diğer dillerdekilerle kıyaslandığında çok daha geniş işlevlere sahiptir. for döngülerinin genel biçimi şöyledir:

```
for (ifade1; ifade2; ifade3)
deyim
```

C derleyicileri for anahtar sözcüğünden sonra parantezler içerisinde iki tane sonlandırıcı (noktalı virgül) beklerler. Bu iki sonlandırıcı for döngüsünü ifade1, ifade2 ve ifade3 ile gösterdiğimiz 3 kısma ayırmaktadır. Şimdi for döngüsünü oluşturan bu 3 kısmın işlevlerini tek tek ele alıp açıklayalım:

ifade1: for döngüsünün birinci kısmı olan bu ifade, döngüye ilk girişte yalnızca bir kez yapılır. İfade1 uygulamalarda genellikle döngü değişkenine ilk değer vermek amacıyla kullanılılır.

Örneğin:

```
for (k = 0; ifade2; ifade3)
```

ifade2: for döngüsünün ikinci kısmı olan i fade2 döngüye ilk girişte ve sonraki her yineleme de işlem görür. for döngüleri ifade2'nin sayısal değeri Doğru (sıfır dışı bir değer) olduğu sürece yinelemeye neden olurlar. Uygulamada ifade2 genellikle yinelenecek miktarı belirleyen ilişkisel bir operatörle kullanılmaktadır.

Örneğin:

```
for (k =0;k < 100; ifade3)
```

ifade3: Bu kısım her yinelemenin sonunda bir kez işlem görür. Uygulamada genellikle döngü değişkeninin artırılması amacıyla kullanılmaktadır.

Örneğin:

```
for (k =0;k < 100; k++)
```

while döngülerinde olduğu gibi for döngülerinde de döngü içerisindeki deyim yalın, bileşik ya da başka bir kontrol deyimi olabilir. Yani tek bir deyim için bloklamaya gerek yoktur; ancak birden fazla deyim döngü içerisine alınacaksa bloklama yapılmalıdır.

Aşağıda O' dan 99' a kadar sayıları ekranda gösteren bir örnek verilmiştir, inceleyiniz:

```
{int k;
for (k =0; k < 100; ++k)
        printf( "%d\n", k);
}
```

for döngüsünün herhangi bir ya da birden fazla kısmı olmayabilir; ancak parantezler içinde iki sonlandırıcının mutlaka bulunması gerekir. A.şağıdaki örneği inceleyiniz:

```
k = 0;
for (; k < 100; ++k)
printf ("%d\n", k);
…
```

Burada for döngüsünün birinci kısmı yoktur. Birinci kısım, döngüye girişte yalnızca bir kez işlem gördüğüne göre döngünün başında ayrı bir deyim olarak da konulabilir. Şimdi aşağıdaki örneği inceleyiniz:

```
k = O;
for (; k < 100; ) {
pri ntf<"%d\n", k);
++k;
}
```

Burada ise for döngüsünün bir ve üçüncü kısımları yoktur. üçüncü kısım her yinelemenin sonunda bir kez işlem gördüğüne göre döngü içerisine son deyim olarak yazılabilir. Gerçekten de bu örnek for döngüsünün çalışmasını çok iyi bir biçimde yansıtmaktadır.

Nihayet üç kısmı da olmayan bir for döngüsü de mümkündür.

```
for(;;) {
….
}
```

bu biçimdeki for döngüleri sonsuz döngü sağlamak amacıyla kullanılırlar.

for döngülerinin içinde bulunan üç ifadenin aşağıdaki gibi bir ilişki içinde olması zorunlu değildir.

4.9. break ve continue ANAHTAR SÖZCÜKLERİ

Döngülerin işleyişinde etkili olan iki anahtar sözcük vardır: break ve continue.break anahtar sözcüğü döngüleri sonlandırarak program akışını döngünün dışındaki ilk deyim e atlatır.

Örneğin:

```
for (;;) {
            …
ch = getchar ( );
if (ch == 'q')
break;
}
…
```

Burada klavyeden girilen karakter 'q' olduğunda break anahtar sözcüğü ile döngü kırılmaktadır. for (;;) sonsuz bir döngü olduğundan çıkış da ancak break ile mümkün olabilir!

```
while  (k < 100) {
          ….
          if (fonk1 ( ) < 0)
                    break;
          ….
}
….
```

Bu örnekte ise fonk1 fonksiyonunun geri dönüş değeri sıfırdan küçük ise döngü kırılmaktadır. Aşağıdaki örnekte de 0' dan 100' e kadar olan tamsayılar ekrana yazdırılıyor.

continue anahtar sözcüğü o anda içinde bulunulan yinelemeyi keserek bir sonraki yinelemeye geçilmesine neden olur. Aşağıdaki örneği inceleyiniz:

```
for (k = 0; k < 100; ++k){
        if (k % 5 == 0)
        contınue;
        deyim1
}
```

Bu örnekte k bir tamsayı olduğuna göre; k % 5 == 0 koşulu sağlanıyorsa k, 5 sayısına tam bölünebiliyor demektir. Bu durumda continue ile döngünün devam etmesi engellenerek sonraki yinelemeye, yani k 1 artarak döngünün başına geçilmiştir.

for döngüsü içerisinde continue kullanıldığında for döngüsünün üçüncü kısmı işleme sokulduktan sonra yineleme yapılır. Dolayısıyla yukarıdaki örnekte continue anahtar sözcüğünden sonra k, bir artırılarak döngüye devam edilir.

4.9. DEĞERLENDİRME SORULARI

1. Klavyeden girilen 1-25 arasındaki bir tamsayının faktöriyelini alan programı yazınız.
2. klavyeden ardı ardına sayı girişi isteyen ve bu sayı 10 ile 15 arasında olmadığı sürece bu işleme devam eden programı yazınız.
3. 1den 25 e kadar olan sayıların kareleri toplamını bulan programı yazınız.

4 ile 7. sorularda akrana aşağıdaki çıktıları verecek programları uygun deyimleri kullanarak yazınız.

4.
```
1
1 2
1 2 3
1 2 3 4
```

5.
```
1 1 1 1
1 1 1
1 1
1
```

6.
```
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

7.
```
4 4 4 4
3 3 3 3
2 2 2 2
1 1 1 1
```

8. Klavyeden 10 tane tamsayı girilmesini isteyen ve bu girilen tamsayılardan kaç tanesinin negatif olduğunu bulan programı yazınız.
9. a,b,ve c klavyeden girilmek üzere, $ax^2+bx+c=0$ şeklindeki bir denklemin köklerini bulan programı yazınız.
10. Klavyeden girilen 1-12 arasındaki tamsayıların hangi aya denk geldiğini bulup ekrana yazan programı yazınız.
11. Dört işleme birer kod numarası vererek, klavyeden girilen iki sayıyı yine klavyeden girilen işlem koduna göre toplayan, çıkaran, çarpan veya bölen programı yazınız.
12. Klavyeden ardı ardına girilen sayıları toplayan ve girilen sayı negatif olduğunda duran programı yazınız.

13. Klavyeden bir not girilmesini isteyen ve bu not 0-49 arasındaysa “Başarısız”, 50-64 arasındaysa “Orta”, 65-84 arasındaysa “İyi”, 85-100 arasındaysa “Çok iyi “ Yazan programı yazınız.
14. Klavyeden girilen iki tamsayıdan büyük olanı bulup ekrana yazdıran programı yazınız.
15. Klavyeden girilen iki pozitif tamsayıdan birincisinin ikincisi cinsinden kuvvetini alan programı hazır fonksiyon kullanmadan yazınız.

HAZIR KÜTÜPHANE FONKSİYONLARI

Bölüm 5

5.1. GİRİŞ

C dilinin genel yapısı hatırlandığında, "***#include***" sözcüğü ile başlayan derleyici ön bildirilerde program içerisinde kullanılacak standart fonksiyonların bağlı bulunduğu dosyaların C kütüphanesinden programa eklenmesi gerekir. Bunun anlamı program içerisinde kullanılacak bir fonksiyonun bağlı bulunduğu kütüphanenin programa dahil edilmesi gerekir. Bu bölümde C kütüphaneleri ve bu kütüphanelere bağlı bazı standart fonksiyonların kullanımı anlatılacaktır. C dilinde tanımlı kütüphane dosyaları aşağıda verilmiştir.

Kütüphane Dosyası	Açıklama
Math.h	Matematiksel fonksiyonlar
Stdlib.h	Standart kütüphane alt programları
Stdio.h	C için stream alt programları
Conio.h	Ekran ve iskele G/Ç fonksiyonları
String.h	Karakter dizisi fonksiyonları
Io.h	Dosya işleme ve düşük seviyeli G/Ç fonksiyonları
Time.h	Tarih ve saat fonksiyonları

Tablo 5.1. Hazır Kütüphane Fonksiyonları

BÖLÜM -5- HAZIR KÜTÜPHANE FONKSİYONLARI

Bölümün Genel Amacı: Hazır kütüphane fonksiyonlarını açıklama.

Bölümün Davranışsal Amaçları: Kitabınızın bu bölümünü başarıyla bitirip, uygulamaları yapıp, değerlendirme sorularına doğru cevap verdiğiniz taktirde, bölüm sonunda;

- Math.h fonksiyonlarını açıklamanız,
- Stdlib.h fonksiyonlarını açıklamanız,
- Stdio.h fonksiyonlarını açıklamanız,
- Conio.h fonksiyonlarını açıklamanız,
- String.h fonksiyonlarını açıklamanız,
- Time.h fonksiyonlarını açıklamanız,
- Dos.h fonksiyonlarını açıklamanız beklenmektedir.

Değerlendirme: Bölüm sonundaki değerlendirme sorularını yapmanız ve sonuçlarınızı uygulama raporları ile karşılaştırmanız ve değerlendirme sorularına en az %75 doğru cevap vermeniz gerekmektedir.

5.2. MATH.H FONKSİYONLARI

Bu kütüphane geniş bir matematik fonksiyon grubuna sahiptir ve matematik, cebir ve trigonometri işlemlerini yapabilmemize olanak tanır. Bu fonksiyonlar arasında trigonometrik fonksiyonlar kullanılırken değişkenlerin radyan cinsinden olması gerektiği unutulmamalıdır. Matematiksel fonksiyonların çoğu math.h kütüphanesinde verilmiş olmasına rağmen matematiksel fonksiyonlardan bazıları stdlib.h' da tanımlanmıştır. Bu fonksiyonlar adı geçen kütüphane bölümünde ele alınacaktır. Tablo 5.2'de math.h fonksiyonlarından en çok kullanılanları verilmiştir.

Fonksiyon Tipi	**Fonksiyonlar**
Trigonometrik	sin(x), cos(x), tan(x)
Ters Trigonometrik	asin(x), acos(x), atan(x), atan2(y,x)
Hiperbolik	sinh(x), cosh(x), tanh(x)
Logaritmik	log(x), log10(x)
Üstel	exp(x), ldexp(x),pow(x,y), sqrt(x), hypot(x,y), poly(x,d,c[ ])
Yuvarlatma	ceil(x), floor(x)
Mutlak Değer	abs(x), labs(x), fabs(x)

Tablo 5.2. Math.h Fonksiyonları

5.2.1 Trigonometrik Fonksiyonlar

sin (x): Radyan cinsinden verilen x değerinin sinüs değerini hesaplar.

cos (x): Radyan cinsinden verilen x değerinin kosinüs değerini hesaplar.

tan (x): Radyan cinsinden verilen x değerinin tanjant değerini hesaplar.

5.2.2 Ters Trigonometrik Fonksiyonlar

asin (x): Verilen x değerinin arg sinüs değerini hesaplar, x değeri –1 ile +1 arasında olmalıdır. Sonuç $-\pi/2$ ile $+\pi/2$ arasında çıkar. Eğer x değeri tanım aralığının dışında bir değer verilirse *errno*'ya domain hatası anlamına gelen EDOM yerleştirilir ve geri dönen değer 0 olur.

acos (x): Verilen x değerinin arg kosinüs değerini hesaplar, x değeri –1 ile +1 arasında olmalıdır. Sonuç 0 ile π arasında çıkar. Hata durumunda *errno*'ya EDOM yerleştirilir.

atan (x): Verilen x değerinin arg tanjant değerini hesaplar. Çıkan değer -π/2 ile +π/2 arasındadır.

atan2 (y,x): Bu fonksiyon arctan(y/x) değerini hesaplar.

5.2.3 Hiperbolik Fonksiyonlar

sinh(x): Verilen x değerinin hiperbolik sinüs değerini hesaplar.

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

cosh(x): Verilen x değerinin hiperbolik kosinüs değerini hesaplar.

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

tanh(x): Verilen x değerinin hiperbolik tanjant değerini hesaplar.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

5.2.4 Logaritmik Fonksiyonlar

log(x): Verilen x değerinin e tabanına göre doğal logaritmasını hesaplar. Matematikte ln(x) olarak bilinir ve logaritması alınan değerler sıfır ve sıfırdan küçük olamaz.

log10(x): Verilen x değerinin 10 tabanına göre logaritmasını hesaplar. Yine bu fonksiyonun x değerleri sıfır ve sıfırdan küçük olamaz.

5.2.5 Üstel Fonksiyonlar

exp(x): Euler sayısının x inci kuvvetini hesaplar.

ldexp(x,n): Bu fonksiyon $x \times 2^n$ değerini hesaplar.

pow(x,y): Bu fonksiyon x^y değerini hesaplar. x=0 iken y $\leq$ 0 veya x<0 iken y bir tamsayı değilse hata oluşur. Sıfırın sıfırıncı kuvveti için 1 değeri üretilir.

sqrt(x): Verilen x değerinin karekökünü hesaplar. Burada x sıfıra eşit ve büyük olmalıdır.

Hypot(x,y): Hipotenüs değerini $\sqrt{x^2 + y^2}$ hesaplar.

5.2.6 Yuvarlama Fonksiyonları

ceil(x): Verilen x gerçel sayısını x'den küçük olmayan tamsayıya yuvarlama işlemini yapar.

Örnek 5.1

```
ceil(5.36)      6
ceil(4.01)      5
ceil(-19.423)   -19
ceil(1.000001)  2
```

floor(x): Verilen x gerçel sayısını x'den büyük olmayan tamsayıya yuvarlama işlemini yapar.

Örnek 5.2

```
floor(5.36)      5
floor(4.01)      4
floor(-19.423)   -20
floor(1.000001)  1
```

5.2.7 Mutlak Değer Fonksiyonları

abs(x): int tipinde verilen x değerinin mutlak değerini hesaplar. int tipindeki bir alan dışındaki sayı için yanlış sonuç elde edilir.

Örnek 5.3

```
abs(-456)      456
abs(23)        23
abs(-32768)    -32768 (Yanlış sonuç, çünkü int sınırı dışında bir tamsayı)
```

labs(x): long tipindeki x sayısının mutlak değerini hesaplar.

fabs(x): double tipindeki x sayısının mutlak değerini hesaplar.

5.3. STDLIB.H FONKSİYONLARI

Standart kütüphane olarak ifade edilen bu kütüphanede veri dönüşümü, bellek yerleşimi ve diğer işlemleri içeren makro ve bir grup güçlü fonksiyonlar bulunur. Ayrıca Math.h da bulunmayan bazı matematiksel fonksiyonlar bulunmaktadır. Bu fonksiyonlar veya makroların açıklaması Tablo 5.3'de toplu halde verilmiştir.

Fonksiyon veya Makro adı	**Açıklama**
exit( )	Programın çalışmasını durdur
lrotl( )	*Unsigned long* sayıyı sola döndür
lrorr( )	*Unsigned long* sayıyı sağa döndür
rotl( )	Tamsayıyı sola döndür
rotr( )	Tamsayıyı sağa döndür
atexit( )	Fonksiyondan çıkarken kaydet
atof( )	Karakter katarını *float* sayıya çevir
atoi( )	Karakter katarını *int* sayıya çevir
atol( )	Karakter katarını *long int* sayıya çevir
bsearch( )	İkili diziyi ara
calloc( )	Dinamik bellek kullanımını başlat
ecvt( )	*float* sayıyı karakter katarına çevir
free( )	Dinamik yerleştirilmiş diziyi serbest bırak

Tablo 5.3. Stdlib.h Kütüphanesi Fonksiyonları

5.3.1 Stdlib.h Kütüphanesinde Verilmiş Matematiksel Fonksiyonlar

Max(x,y) ve Min(x,y): Bu iki fonksiyon verilen iki sayı olan x ve y değerlerini karşılaştırarak en büyük ve en küçük olan değeri bulan fonksiyonlardır ve aşağıda gösterilen makrolar olarak tanımlanmıştır.

```
#define max(x,y)   ((x > y) ? (x) : (y))
#define min(x,y)   ((x < y) ? (y) : (y))
```

div(x,y): Verilen iki int tamsayıyı böler.

rand: Bu fonksiyon 0 ile RAND_MAX arasında 2^{32} peryotlu rastgele bir sayı üretir. Sembolik sabit RAND_MAX, stdlib.h içinde tanımlanmıştır ve değeri 2^{15}-1 dir.

Diğer rastgele sayı üretme fonksiyonları **random**, **srand**, **randomize** dir. Bu fonksiyonlar simülasyon programlarında belli aralıkta rasgele sayı üretmek için kullanılır.

5.4. STDIO.H FONKSİYONLARI

Bu kütüphane dosyalarla ilgili bir kütüphanedir. En çok kullanılan dosya komutları Tablo 5.4'de verilmiştir.

Fonksiyon	**Anlamı**
fclose	Dosyayı kapatır
fcloseall	Tüm dosyaları kapatır
fgetc	Dosyadan karakter alır
Fgetchar	Karakter alır
Fgets	Dosyadan karakter katarı alır
Fopen	Dosya açar
Fprintf	Dosyaya formatlı yazı yazar
Fputc	Dosyaya karakter yazar
Fputchar	Karakter gösterir
Fscanf	Dosyadan formatlı okur
Fwrite	Yapısal dosyaya veri kaydeder
Getc	Dosyadan karakter alır
Getchar	Karakter alır
Printf	Ekrana formatlı yazar
Putc	Fputc fonksiyonu ile aynı işlevi yapar, farkı makro olmasıdır
Putchar	Fputchar fonksiyonu ile aynı işlevi yapar, farkı makro olmasıdır
scanf	Ekrandan formatlı okur

Tablo 5.4. Stdio.h Kütüphanesi Fonksiyonları

5.5. CONIO.H FONKSİYONLARI

Bilgisayar ekranını en iyi kullanmak için gerekli bir kütüphanedir. Pencere açma, ekran modunu kontrol etme, renk belirleme gibi ekran işlemleri bu kütüphanede yapılır. Tablo 5.5 de en çok kullanılan conio.h fonksiyonlarından bazıları verilmiştir.

Tablo 5.5: Conio.h Kütüphanesi fonksiyonları

Fonksiyon
clreol
clrscr
delline
getch
gotoxy
highvideo
insline
lowvideo
normvideo
textbackground

textcolor
wherex
wherey
window

Tablo 5.5. Conio.h Kütüphanesi Fonksiyonları

- **clreol( ):** Kürsörün bulunduğu yerden itibaren satırın sonuna kadar olan tüm karekterleri ekrandan siler. Kürsör olduğu yerde kalır.
- **clrscr( ):** Ekranı temizler. Kürsor ilk satır, ilk sütuna konumlanır.
- **delinle( ):** Kürsörün bulunduğu satırı siler.
- **getch( ):** Karakteri göstermeden, bir karakter girilmesini bekler.
- **gotoxy(sütun,satır):** Kürsörün ekranda, istenilen satır ve sütuna gitmesini sağlar.
- **highvideo( ):** Karakterlerde parlak tonu seçer.
- **insline( ):** Kürsörün bulunduğu yere yeni bir satır açar.
- **lowvideo( ):** Karakterlerde soluk tonu seçer.
- **normvideo( ):** Görüntü rengini ilk renge dönüştürür.
- **textbackground(renk):** Ekranın zemin rengini, renk ile belirtilen renge çevirir.
- **textcolor(renk):** Ekranın yazı rengini, renk ile belirtilen renge çevirir.
- **wherex( ):** Kürsörün o anda bulunduğu sütun numarasını saklar.
- **wherey( ):** Kürsörün o anda bulunduğu satır numarasını saklar.
- **window(x1,y1,x2,y2):** Ekran üzerinde bağımsız pencere oluşturur. Oluşturulan pencere başlı başına bir ekran gibidir.

5.6 STRING.H FONKSİYONLARI

Hafızada bulunan karakter dizileri (string) ile ilgili işlemler için kullanılır. Tablo 5.6 da en çok kullanılan string.h fonksiyonlarında bazıları verilmiştir.

Fonksiyon Tipi	Fonksiyonlar
Birleştirme	strcat, strncat
Değiştirme	strlwr, strupr, strset, strrev
Arama	strchr
Kopyalama	strcpy, strncpy
Karşılaştırma	strcmp, strncmp

Tablo 5.6 String.h Kütüphanesi Fonksiyonları

5.6.1 Birleştirme Fonksiyonları

strcat(s1,s2): s1 ile belirtilen stringin sonuna, s2 ile belirtilen stringi ekler.

Strncat(s1,s2,n): s1 ile belirtilen stringin sonuna, s2 ile belirtilen stringin n tane karakterini ekler.

5.6.2 Değiştirme Fonksiyonları

strlwr(s1): s1 karakter dizisindeki tüm harfleri küçük harfe çevirir.

strupr(s1): s1 karakter dizisindeki tüm harfleri büyük harfe çevirir.

Strset(s1,karakter): Parametre olarak verilen stringin tüm karakterlerini yine parametre olarak verilen karaktere çevirir.

Strrev(s1): s1 stringini tersden okur.

5.6.3 Arama Fonksiyonları

strchr(s1,karakter): s1 stringi içersinde "karakteri" arar. Karakter bulununca arama durur ve ilk bulduğu yerin adresini geri döndürür.

5.6.4 Kopyalama Fonksiyonları

strcpy(s1,s2): s2 stringinin değerini s1 stringine kopyalar.

strncpy(s1,s2,n): Bir stringin baştan itibaren n karakterini başka bir stringe kopyalar.

5.6.5 Karşılaştırma Fonksiyonları

strcmp(s1,s2): İki stringin, ilk karakterden başlayarak ASCII kodlarını karşılaştırır. Karşılaştırma sonucu olarak, bu komut 0,-,+ değerlerinden birini üretir.

strncmp(s1,s2,n): İki stringin ilk n karakterini karşılaştırır.

5.7 TIME.H FONKSİYONLARI

Zaman ile ilgili işlemler için kullanılır. En çok kullanılan time.h fonksiyonlarından ikisi aşağıda verilmiştir.

- **clock( ):** Fonksiyonun çalıştırıldığı andaki sistemin saatini alır.
- **time( ):** Günün saat bilgisini saniye olarak verir.

5.8 DOS.H FONKSİYONLARI

En çok kullanılan dos.h fonksiyonlarından bazıları Tablo 5.7 de verilmiştir.

Fonksiyon
delay
sleep
sound
settime
gettime
setdate
getdate

Tablo 5.7. Dos.h Kütüphanesi Fonksiyonları

- **delay(x):** x milisaniye kadar programın çalışmasını durdurur.
- **sleep(x):** x saniye kadar programın çalışmasını durdurur.
- **sound(x):** x frekansında ses üretir.
- **settime(z):** Sistemin zaman bilgisini değiştirmek için kullanılır.
- **gettime(z):** Sistemin zaman bilgisini öğrenmek için kullanılır.
- **setdate(z):** Tarih bilgisini değiştirmek için kullanılır.
- **getdate(z):** Tarih bilgisini almak için kullanılır.

5.9 DEĞERLENDİRME SORULARI

Aşağıdaki program parçaları çalıştırıldığında ekran çıktıları ne olur?

1.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a,b;
void main()
{
clrscr();
a=floor(-4.00001);
b=floor(7.892);
printf("a=%d",a);
printf("\nb=%d",b);
getche();
}
```

2.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a,b;
void main()
{
clrscr();
a=ceil(-4.00001);
b=ceil(7.892);
printf("a=%d",a);
printf("\nb=%d",b);
getche();
}
```

3.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a,b;
void main()
{
clrscr();
a=ceil(-4);
b=ceil(7);
printf("a=%d",a);
printf("\nb=%d",b);
getche();
}
```

4.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a,b,c;
void main()
{
clrscr();
a=2;
b=3;
c=pow(a,b);
printf("c=%d",c);
getche();
}
```

5.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char s1[5],s2[]="FIRAT";
void main()
{
clrscr();
strcpy(s1,s2);
printf("s1=%s",s1);
getche();
}
```

6.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char s1[]="FIRAT",s2[]="UNIV";
void main()
{
clrscr();
strcat(s1,s2);
printf("s1=%s",s1);
getche();
}
```

7.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char s1[]="FIRAT";
int a;
void main()
{
clrscr();
a=strlen(s1);
printf("a=%d",a);
getche();
}
```

FONKSİYONLAR

Bölüm 6

6.1. GİRİŞ

Bir program bir görevi yerine getirmek için yazılır. Eğer yapılacak iş pek kolay değilse program oldukça uzun olabilir. Bazı programlar onbinlerce satır uzunluğunda olabilir. Böyle durumlarda, esas görevi daha küçük ve kolay idare edilebilir alt görevlere ayırmadan yerine getirmek hemen hemen olanaksızdır.

C, böyle alt görevleri ifade etmek ve birbirinden ayırmak için bir yöntem öngörmektedir. C sadece bir tek altprogram çeşidi sağlamaktadır, bu da fonksiyondır. Bu bölümde fonksiyon tanımı ve kullanımı hakkında bilgiler edinilecektir.

BÖLÜM -6- FONKSİYONLAR

Bölümün Genel Amacı: Fonksiyonların tanımlanması ve kullanılabilmesi

Bölümün Davranışsal Amaçları: Kitabınızın bu bölümünü başarıyla bitirip, uygulamaları yapıp, değerlendirme sorularına doğru cevap verdiğiniz taktirde, bölüm sonunda;

- Fonksiyon tanımlamanız
- Fonksiyon çağırmanız
- Fonksiyonları program içinde kullanmanız

Değerlendirme: Modül sonundaki uygulamaları yapmanız sonuçları uygulama raporu ile karşılaştırmanız ve değerlendirme sorularına en az % 75 düzeyinde doğru cevap verebilmeniz gerekmektedir.

6.2. FONKSİYONLAR

Fonksiyon, belirli sayıda verileri kullanarak bunları işleyen ve **bir** sonuç üreten komut grubudur.
Fonksiyonda elde edilen sonuç değerinin mutlaka **return** komutu kullanarak programa yollanması gerekir.

- Fonksiyonlar, programların etkinliğini artırmanın; kolay yazılmaları ve okunmalarını sağlamanın, bellekte daha az yer kaplamalarını sağlamanın en uygun yoludur.
- Programın herhangi bir yerinde yazılan bir fonksiyonu, programın her tarafında tekrar tekrar çağırarak çalıştırma olanağı bulunmaktadır.
- Fonksiyonlar, programın bütününden bağımsız olarak yazılırsa, diğer programlarda kullanması için sadece kopyalamak yeterli olacaktır.

Yazılan tüm fonksiyonlar sadece main() fonksiyonu içinden çağırılarak kullanılabilir.

Geri Dönüş Değerinin Türü Fonksiyon İsmi (VeriTipi Parametre Adı.....)
{
blok
return
}

C'de altprogramlara fonksiyon (function) denir. Her fonksiyon faydalı birtakım işlemleri (bir işlevi) yerine getirmek için tasarlanır ve çağrılır. Fonksiyonların, onları çağıran fonksiyonlardan aldıkları **girdileri** ve yine onları çağıran fonksiyonlara gönderdikleri **çıktıları** vardır.

Fonksiyonların girdilerine parametre ya da argüman denilir.

6.3 FONKSİYON PARAMETRELERİ

6.3.1 Fonksiyonların geri dönüş değerleri (Çıkış)

Bir fonksiyonun çağırılması sonunda onu çağıran fonksiyona dönüşünde gönderdiği değere geri dönüş değeri (return value) denir. Geri dönüş değeri, bir değişkene atanabilir ya da doğrudan matematik işlemlerde kullanılabilir.

x=fonk(); (değişkene atama)
[Önce fonk() çalışır. Geri dönen değer x'e atanır.]

x=fonk() + y; (işlemde kullanma)
[Önce fonk() çalışır. Geri dönen değerle y toplanıp, x'e atanır.]

biçiminde kullanılabilir.

Fonksiyonların geri dönüş değerlerinin kullanım amaçları

- Bazı fonksiyonlar bir tek değer elde etmek amacıyla tasarlanmıştır. Elde edilen değer, kendilerini çağıran fonksiyonlara geri dönüş değeri biçiminde iletilir.

a=sqrt(x); sqrt fonksiyonunun amacı x sayısının karakökünü bulmaktadır. Sonuç geri dönüş değeri biçiminde a değişkenine atanmaktadır.

- Bazı fonksiyonların geri dönüş değerleri yapılan işlemin başarısı hakkında bilgi verir.

Yani bu tür fonksiyonların geri dönüş değerleri test amacıyla kullanılmaktadır. Geri dönüş değeri: "İşlem başarılı olmuş mudur ya da neden başarısız olmuştur?..." gibi sorulara yanıt verir.

Örneğin:
p=**malloc(size)**;
ifadesiyle bellekte **size** byte uzunluğunda bir blok tahsis isteyen programcı bu işlemin başarılı bir biçimde getirilip getirilmediğini de test etmek zorundadır. Hemen arkasından **p** değişkeninin aldığı değeri kontrol edecek ve işlemin başarısı hakkında bir karara varacaktır.

- Kimi fonksiyonlar hem belli bir amacı gerçekleştirirler hem de buna ek olarak amaçlarını tamamlayan bir geri dönüş değeri üretirler.

Örneğin:
c=printf("Merhaba\n"); printf fonksiyonu ekrana merhaba yazısını yazmak için kullanılmıştır. Ancak ekrana yazdığı karakter sayısını da geri dönüş değeri olarak vermektedir.

- Bazen geri dönüş değerlerine hiç ihtiyaç duyulmaz.

Örneğin, yalnızca ekranı silme amacıyla tasarlanmış olan bir fonksiyonun herhangi bir geri dönüş değerine sahip olması gereksizdir.
clrscr(); clrscr fonksiyonu yalnızca ekranı siler; böyle bir fonksiyonun geri dönüş değerine gereksinimi yoktur.

Fonksiyonların geri dönüş değerleri herhangi bir türden olabilir. Geri dönüş değerlerinin türleri fonksiyonların tanımlanması sırasında belirtilir.

Bir fonksiyonun parametresi ve/veya geri dönüş değeri olmayabilir.

float a1(void) { } veya
float a1( ) { } a1 fonksiyonu parametreye (giriş değerine) sahiptir.
void x1 (void) { } x1 fonksiyonu geri dönüş değerine de parametreye de (giriş değeri) sahip değildir.

```
#include <stdio.h>
#include <math.h>
void uyari(void)
{
    puts("Uyari: Negatif sayinin karakoku alinmak
istedi.");
}

float x, sonuc; //Global degisken tanimlamasi
main()
{
   printf("Bir sayi giriniz : ");
   scanf("%f",&x);
   if (x>0.0)
{
   sonuc=sqrt(x);
   printf("sonuc= %f\n",sonuc);
}
else uyari();
}
```

Çıktısı:
Bir sayi giriniz : -9
Uyari: Negatif sayinin karakoku alinmak istendi.

Örnek 6. 1 Fonksiyon Tanımlama

Fonksiyon tanımlarken geri dönüş değerinin türü yerine hiç bir şey yazılmazsa C derleyicileri geri dönüş değerinin türünün int olduğunu varsayarlar.

Örneğin:

y1 ( ) { .... }
y1 fonksiyonunun parametresi yoktur; ancak geri dönüş değeri **int** türündendir.

C'de fonksiyon içinde fonksiyon tanımlanamaz.

Yanlış Fonksiyon içinde fonksiyon tanımlanmıştır.	**Doğru**
İnt fonksiyon1 () **{** **....** **int fonksiyon2 ()** **{** **....** **}** **....** **}**	**int fonksiyon1 ()** **{** **....** **}** **int fonksiyon2 ()** **{** **....** **}**

Çağrılan fonksiyon ile çağıran fonksiyon arasında bağlantı kurma işlemi, bağlama aşamasında, bağlayıcı program (linker) tarafından yapılır.

C programlarının çalışabilmesi için mutlaka main fonksiyonunun bulunması gerekir. main fonksiyonu yoksa bağlama aşamasında, bağlayıcı program tarafından bildirilecektir.

return ANAHTAR SÖZCÜĞÜ

İki önemli işlevi vardır.

1. Fonksiyonların geri dönüş değerini oluşturur.
2. Fonsiyonları sonlandırır.

return anahtar sözcüğünün **kullanılması zorunlu değildir**. Eğer return anahtar sözcüğü yoksa fonksiyon, ana bloğu bitince kendiliğinden sonlanır.

Fonksiyonların yalnız bir tane geri dönüş değeri olabilir. Diğer bir deyişle kendisi çağıran fonksiyona bir tane dönüş değeri gönderebilir.
Fonksiyonlardan birden fazla geri dönüş değeri çıkarabilmek göstericiler (pointers) ve yapılar (structures) ile mümkündür

6.3.2 Fonksiyonların giriş Değerleri

Fonksiyon parametrelerinin tanımlanması

1. Eski biçim	2. Yeni biçim

int topla (a, b)
int a, b;
{
return a+b;
}
	int topla (int a, int b)
	{
	return a+b;
	}

Arguman listesine, fonksiyona aktarılacak olan değerlerin saklanacağı yerel değişken adları yazılır. Fonksiyona aktarılan değerlere formal parametre ve saklandığı değişkenlere de formal değişken denir. Formal değişkenler, aynı yerel değişkenler gibi bildirildiği fonksiyon yürütüldüğü anda geçerlidir. Programın akışı fonksiyondan çıkıp, çağıran yere döndüğü anda yerel değişkenler ve formal değişkenler için kullanılan bellek alanı serbest bırakılır.

```
#include<stdio.h>
int topla( int a, int b)    /* fonksiyon tanimlanmasi*/
  {
  int c;
  c = a + b;
  return c;
  }
main()
  {
  int i=1;

  printf("fonksiyon calismadan once i'nin degeri=%d\n", i);

  i = topla(i,i);          /* fonksiyon calistiriliyor*/

  printf(" fonksiyon calistirildiktan sonra i'nin degeri= %d\n",
i);
}
```

Örnek 6. 2 Fonksiyon Tanımlama

```
int sayac = 0;          /*butun bloklarin disinda oldugu icin global
degisken*/

int fonksiyon(void);

main()

{

  sayac++;              /* global degisken*/

  printf("\n\nfonksiyon cagrilmadan once sayacin degeri= %2d\n",
sayac);

  fonksiyon();       /* fonksiyon cagrildi*/

  printf("fonksiyon cagrildiktan sonra sayacin degeri= %2d\n", sayac);

}

int fonksiyon(void)

{

  int sayac = 10;      /* yerel degisken*/

  printf("\tfonksiyonun icinde sayacin degeri= %2d \n", sayac);

}
```

Örnek 6. 3 Değişken Tanımlama

```
/*Kombinasyon hesabi*/

/*C(n,r)=n!/(r!*(n-r)!)*/

#include <stdio.h>

#include <conio.h>

int fak (int f);
```

```
main()
{
int n,r;
float c;
puts("\nkombinasyon hesabi: ");
puts("C(n,r)=n!/(r!*(n-r)!)");
printf("n'in degerini gir...: "); scanf("%d",&n);
printf("r'in degerini gir...: "); scanf("%d",&r);
c=(float) fak(n)/(fak(r)*fak(n-r));
printf("C=%5.2f\n",c);
getch();
}
int fak (int f)
{
int i, fa;
fa=1;
if (f<0) return 0; else if (f==0) return 1;
else
for (i=1; i<=f;i++) fa*=i;
return (fa);
}
```

Çıktısı:

```
/*n'in degerini gir...: 5
 *r'in degerini gir...: 2
 *C=10.00*/
```

Örnek 6. 4 Kombinasyon Programı

```
/*fibanocci serisi*/
/*1+1+2+3+5+8+13*/
#include <stdio.h>
int fibanocci(int n)
{
int a, b, c, top;
a=1; b=1; c=0; top=1;
do
{
  top+=b;
c=a+b;
a=b;
b=c;
} while (b<=n);
printf("toplam=%d ", top);
}
main()
{
```

```
  int nn=15;
/*  printf("son sayiyi gir :");
scanf("%d", &nn);*/
fibanocci(nn);
getch();
}
```

Örnek 6. 5 Fibanocci Sayı Programı

```
/*x^1/1!+x^2/2!+x^3/3!+ ... + x^n/n!*/
#include <stdio.h>
int fak (int f);
int us (int xx, int u);
main()
{
int i,x,n;
float toplam, ara;
puts("hesaplanacak seri: ");
puts("x^1/1!+x^2/2!+x^3/3!+ ... + x^n/n!");
printf("x'in degerini gir..."); scanf("%d",&x);
printf("n'in degerini gir..."); scanf("%d",&n);
toplam=0;
for (i=1;i<=n;++i)
```

```
{
ara=(float) us(x,i)/fak(i);
printf("Dongu %d.kez calisti. %d^%d/%d!=%5.2f\n",i,x,i,i,ara);
toplam+=ara;
}
printf("sonuc=%5.2f",toplam);
getch();
}
int fak (int f)
{
int i, fa;
fa=1;
if (f<0) return 0; else if (f==0) return 1; else
for (i=1; i<=f;i++) fa*=i;
return (fa);
}
int us (int xx, int nn)
{
int i, son;
son=1;
for (i=1;i<=nn; i++) son *=xx;
return(son);
}
```

Örnek 6. 6 Polinom Hesaplama Programı

```
/* Asal Sayıların bulunması*/
#include <stdio.h>

int asal(int);
main()
{
int i ;
int kac;
int j;

clrscr();
printf("ilk kac asal sayi listelenecek : ");
scanf("%d", &kac);

i = 2;
for (j = 1; j<=kac;i++)
if (asal(i))
   {
   printf("%d \t", i);
   j++;
}
}
int asal(int n)
{
   int i;
   if (n % 2 == 0)
      return (n==2);
   if (n % 3 == 0)
      return (n==3);
   if (n % 5 == 0)
      return (n==5);
   for (i=7; i*i <= n; i+=2)
      if (n % i == 0)
         return 0;
   return 1; /* sayı asal ise, 1 değil ise 0 değerini alır */
}
```

ÇIKTISI:

```
ilk   kac   asal   sayi   listelenecek   :   100
2   3   5   7   11   13   17   19   23   29
31   37   41   43   47   53   59   61   67   71
73   79   83   89   97   101   103   107   109   113
127   131   137   139   149   151   157   163   167   173
179   181   191   193   197   199   211   223   227   229
233   239   241   251   257   263   269   271   277   281
```

```
283  293  307  311  313  317  331  337  347  349
353  359  367  373  379  383  389  397  401  409
419  421  431  433  439  443  449  457  461  463
467 479 487 491 499 503 509 521 523 541
```

Örnek 6. 7 Asal Sayı Bulma Programı

C'de fonksiyon içinde fonksiyon tanımlanmaz. **Fakat program içinde tanımlanan bir fonksiyon diğer fonksiyon içinde kullanılabilir.**

```
#include<stdio.h>
#include<conio.h>
int carp_topla (int, int); //fonksiyon prototipi
int topla (int, int); /* fonksiyon tanimlanmasi*/   //fonksiyon prototipi

main()
{
int i=1;
clrscr();
printf("fonksiyon calismadan once i'nin degeri=%d\n", i);
i = topla(i,i); /* fonksiyon calistiriliyor*/
printf(" fonksiyon calistirildiktan sonra i'nin degeri= %d\n\n\n", i);

i = carp_topla(i,i); /* fonksiyon calistiriliyor*/
printf(" fonksiyon calistirildiktan sonra i'nin degeri= %d\n", i);
}

int topla(a, b) /* fonksiyon tanimlanmasi*/
{
   int c;
   c = a + b;
   return c;
}

int carp_topla(a, b) /* fonksiyon tanimlanmasi*/
{
   int c;
   c = topla(a,b)*topla(a,b); //yukarida tanimlanan topla fonksiyonu kullaniliyor
   return c;
}
```

Çıktısı

```
fonksiyon calismadan once i'nin degeri=1
fonksiyon calistirildiktan sonra i'nin degeri= 2
```

fonksiyon calistirildiktan sonra i'nin degeri= 16

C++'da aynı isimli birden fazla fonksiyon tanımlanabilmektedir (function overloading). Bu durumda C++ derleyicileri aynı isimli fonksiyonları parametre sayılarına ve türlerine göre birbirlerinden ayırırlar.

```
//Ayni isme, farkli parametreye sahip iki fonksiyonun kullanilmasi
#include<stdio.h>
#include<conio.h>

//Ayni isimde iki parametreli fonksiyon tanimlamasi
int topla (int x, int y, int z)
{int d;
x=1; y=2; z=3;
d=x+y+z;
return d;
}
//Ayni isimde uc parametreli fonksiyon tanimlamasi
int topla (int x, int y)
{int d;
x=1; y=2;
d=x+y;
return d;
}
main()
{
int i,a=1,b=2,c=10;
clrscr();
printf ("a = 1\nb = 2\nc = 3\n\n");
i = topla(a,b); /* ayni isime sahip topla fonksiyonunun
                   iki parametreli olani topla(int, int) calistiriliyor*/
printf ("iki parametreli fonksiyonunu\n"
        "calistirildiktan sonra \ni'nin degeri= %d\n\n\n", i);

i = topla(a,b,c); /* ayni isime sahip topla fonksiyonun
          Uc parametreli olani topla(int, int, int)calistiriliyor*/

printf("Uc parametreli topla fonksiyonunu\n"
   "calistirildiktan sonra \n i'nin degeri= %d\n", i);
}
```

Çıktısı:

```
a = 1
b = 2
```

```
c = 3

iki parametreli fonksiyonu
calistirildiktan sonra
i'nin degeri= 3


Uc parametreli topla fonksiyonu
calistirildiktan sonra
i'nin degeri= 6
```

Örnek 6. 8 Fonksiyon Tanımlama

6.3.3 Fonksiyonların kendi kendilerini çağırması (Recursive)

Bir fonksiyonun kendi kendini çağırması, işlev bakımından bir başka fonksiyonu çağırmasından farklı değildir. Bir fonksiyonun her çağrılması, fonksiyonla birlikte tanımlanmış parametrelerin ve diğer yerel (local) değişkenlerin, yığın (stack) üzerinde yeniden yaratılması ve çağırılan fonksiyonun çalışmasının tekrar başlamasına neden olur.

```
#include <stdio.h>

int fakt(int x)
{
  int sonuc;
  if (x<1) return(0);
  if (x==1) return(1);
  sonuc=fakt(x-1)*x;
  return(sonuc);
}
main()
{
  int faktor;
  faktor=fakt(5);
  printf("Faktoriyel=%d\n",faktor);
}
```

Örnek 6.9 Recursive Fonksiyon

Programın çalışması sırasında şu işlemler olmaktadır.

Çağırım ifadeleri	Geri Dönüşler
1. işlem --> faktor=fakt(5)	11. İşlem -->faktor=120
2. işlem --> faktor=fakt(4)	10. işlem--> sonuc=24*5; return(120)

3. işlem --> faktor=fakt(3)	**9. işlem--> sonuc=6*4; return(24)**
4. işlem --> faktor=fakt(2)	**8. işlem--> sonuc=2*3; return(6)**
5. işlem --> faktor=fakt(1)	**7. işlem--> sonuc=1*2; return(2)**
6. işlem --> çağırım yok	**6. işlem--> sonuc=???; return(1)**

6.4 DEĞERLENDİRME SORULARI

1. Klavyeden girilen iki tane sayıyı recursive fonksiyon kullanarak toplayan program yazınız.
2. n! değerini hesaplayan programı yazınız.
3. 1+4+9+ ... +100= değerini hesaplayan programı yazınız.
4. Toplama, çıkarama, çarpma ve bölme işlemi yapan ve program yazınız.
5. Saatte ortalama 60 km yol giden bir aracın, klavyeden girilen mesafeyi kaç saatte gideceğini hesaplayan program yazınız.
6. Klavyeden girilen 10 elemanlı bir diziyi büyükten küçüğe doğru sıralayan bir program yazınız.
7. [1-7] arasında girilen sayıya karşılık gelen haftanın gününü veren bir program yazınız.
8. Klavyeden girilen bir sayının sondan kaç basamağının sıfır olduğunu bulan bir program yazınız.
9. Klavyeden girilen iki sayının bölümünü bölme işlem operatörünü kullanmadan gerçekleştiren bir program yazınız
10. Klavyeden girilen sayıya kadar olan sayıların toplamını hesaplayan programı yazınız
11. Verilen dizide aranılan bir elemanın kaç defa tekrar ettiğini bulan bir program yazınız.
12. Klavyeden girilen bir sayının asal sayı olup olmadığını test eden bir fonksiyon yazınız
13. Klavyeden girilen bir integer dizinin en büyük ve en küçük elemanını bulan bir program yazınız.
14. Dışardan girilen N adet tamsayının aritmetik ortalamasını alıp ekrana yazdıran programı C++ dilinde kodlayınız.
15. $$f(x) = 1 + \frac{1}{x+1} + \sum_{n=1}^{\infty} \frac{x^n}{(x+(n+1))^n}$$

    Serisinin toplamını bulup yazdıracak programı yazınız.
    (x değeri dışarıdan okutulacak ve serinin sadece ilk 75 terimi alınacak)

DİZİLER

Bölüm 7

7.1. GİRİŞ

Bir dizi benzer elemanlardan oluşan bir kümedir. Daha önceki bölümlerde her bir veriyi kaydetmek ve o veriyi tanımlamak için bir değişken kullanıldı, benzer tipte çok fazla veriyi belleğe kaydetmek için dizilere başvurulur. Diziler bir boyutlu veya çok boyutlu olabilirler, boyut sayısı arttıkça işlemler biraz daha karmaşık hale gelebilir. C dilinde çok boyutlu dizilere "elemanları dizi olan dizi" , "dizi dizileri" veya "matris" denir, tek boyutlu diziler ise genel olarak "vektör" olarak adlandırılırlar.

BÖLÜM -7- DİZİLER

Bölümün Genel Amacı: Dizi kavramını anlayabilme ve ihtiyaca bağlı olarak kullanabilme.

Bölümün Davranışsal Amaçları: Kitabınızın bu bölümünü başarıyla bitirip, uygulamaları yapıp, değerlendirme sorularına doğru cevap verdiğiniz taktirde, bölüm sonunda;

- Dizi kavramını açıklamanız,
- Tek boyutlu dizi kavramını açıklamanız,
- Çok boyutlu dizi kavramını açıklamanız,
- Dizler ile karakter işleme yapabilmeniz,
- İşlemlerde dizi kullanımını yapabilmeniz

Değerlendirme: Modül sonundaki uygulamaları yapmanız sonuçları uygulama raporu ile karşılaştırmanız ve değerlendirme sorularına en az % 75 düzeyinde doğru cevap verebilmeniz gerekmektedir.

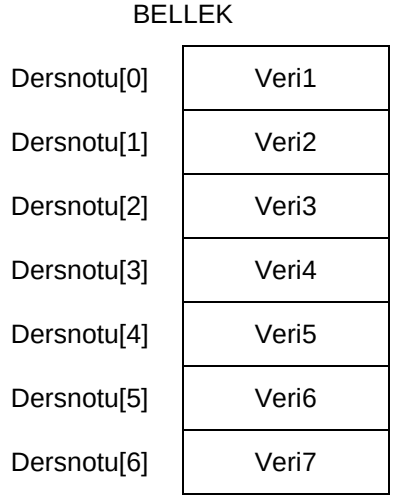
7.2. TEK BOYUTLU DİZİLER

Tek boyutlu dizilerin yazım biçimi aşağıdaki gibidir,

Tip dizi_adı[eleman_sayısı] ;

Bu gösterimde ilk olarak dizi elemanlarının bellekte depolanacağı verilerin tipi, bu elemanlara erişim yapılırken kullanılacak dizi adı, ve son olarak köşeli parantezler içerisinde dizinin eleman sayısı belirtilir.

Örnek olarak; int dersnotu[7] ; gösterilebilir. Bu örnek kullanımında dersnotu adında integer (tamsayı) tipinde bir dizi tanımlanmıştır. Bu dizinin elemanları bellekte aşağıdaki gibi yerleşir:



Şekil 7.2 Tek boyutlu dizi elemanların bellekteki görünümü

Dizinin ilk elemanının indisi 0 dır. Buna bağlı olarak dizinin son elemanının indisi de dizinin eleman sayısının 1 eksiğidir.
Dizinin herhangi bir elemanına erişmek veya değiştirmek için kaçıncı eleman olduğunu gösterir indis bilgisini vermek gerekir, örneğin dersnotu[4] dizinin 5. elemanını gösterir.

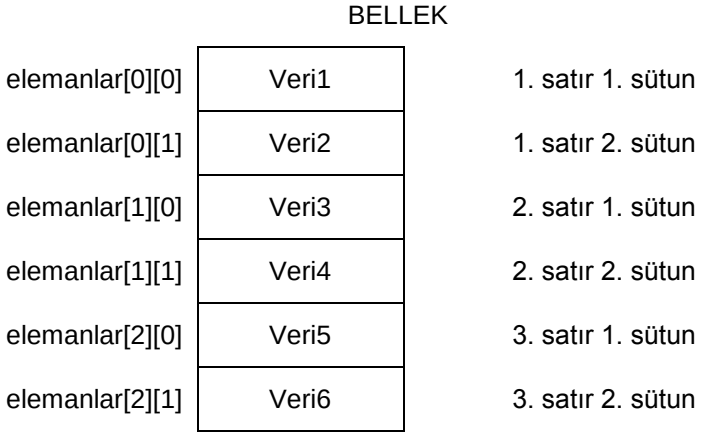
7.3. ÇOK BOYUTLU DİZİLER

Çok boyutlu bilgileri veya veri tablolarını saklamak için kullanılır. En çok 2 boyutlu diziler kullanılırlar, Bellekte satır düzeninde ve art arda depolanırlar. Çok boyutlu dizilerin yazım biçimi aşağıdaki gibidir,

Tip dizi_adı[satır_sayısı][sütun_sayısı] ;

Bu gösterimde ilk olarak dizi elemanlarının bellekte depolanacağı verilerin tipi, bu elemanlara erişim yapılırken kullanılacak dizi adı, ve son olarak köşeli parantezler içerisinde dizinin satır ve sütun eleman sayısı belirtilir. Bu gösterim şekli matrisler için kullanılır.

Örnek olarak; float elemanlar[3][2] ; gösterilebilir. Bu örnek kullanımda elemanlar adında float (gerçel) tipinde iki boyutlu bir dizi tanımlanmıştır.



Şekil 7.3 Çok boyutlu dizi elemanların bellekteki görünümü

Çok boyutlu diziler iki den fazla boyutlu olabilirler, örneğin;

```
Int a[3][2][2] ;
Double say[2][2][3][1] ;
```

7.4. DİZİLERE BAŞLANGIÇ DEĞERİ ATANMASI

Dizilere başlangıç değeri verilmesi için yazım biçimi;

```
Dersnotu[2]=55 ;
Elemanlar[1][0]=10.2;
a[0][1][0]=12 ;
```

Dizilere ilk değer tanımlama sırasında da verilebilir,

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

7.5. KARAKTER İŞLEME (STRINGLER)

C dilinde yazılabilecek programlarda sayıların dışında metinlerle de işlem yapılabilir. Bu tür verileri saklamak için char veri tipi kullanılır.
String dizilerini tanımlayabilmek için aşağıdaki yazım şekli kullanılır;

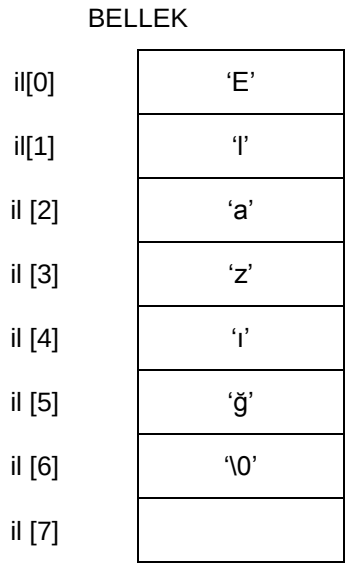
```
char string_adı;

char string_dizi_adı[eleman sayısı];
```

Örneğin;

```
char il;
```

char a[8] = {'E','L',,'A','Z','I','Ğ'};
char il[8]=”Elazığ”; Bu dizinin elemanları bellekte aşağıdaki gibi yerleşir:



Şekil 7.4 String dizilerinin bellekteki görünümü

Örnek 7.1 Bir diziye eleman girişi ve dizinin elemanlarını ekrana yazdırma

```
#include <stdio.h>
#include <conio.h>
main()
 {
       clrscr();
       int i;
       int dizi[15];
       printf("15 adet sayi gir \n");
       for (i=0;i<15;i++)
         {
           printf("\ndizi[%d]=",i);
           scanf("%d",&dizi[i]);
         }
        for (i=0; i<15; i++)
           printf("dizinin %d.elemani dizi[%d]=%d\n",i+1,i,dizi[i]);
           getch();
 }
```

Örnek 7.2 Bir diziye ilk değer atama ve dizinin elemanlarını ekrana yazdırma

```
#include <stdio.h>
#include <conio.h>
main()
{
clrscr();
  int i;
  int dizi[5]={100,200,300,400,500};
  for (i=0; i<5; i++)
     printf("\ndizinin %d.elemani dizi[%d]=%d\n",i+1,i,dizi[i]);
getch();
}
```

Ekran Çıktısı

```
dizinin 0.elemanı a[0]=100
dizinin 1.elemanı a[1]=200
dizinin 2.elemanı a[2]=300
dizinin 3.elemanı a[3]=400
dizinin 4.elemanı a[4]=500
```

Örnek 7.3 Bir elemanın dizide olup olmadığını bulan program

```
#include <stdio.h>
#include <conio.h>
main()
{
int i,j,ara;
int dizi[5];
clrscr();
printf("diziyi giriniz:\n");
for (i=0;i<5;i++) scanf("%d",&dizi[i]);
printf("Sayıyı giriniz...");
scanf("%d",&ara);
for(i=0;i<5;i++)
if (ara==dizi[i])
{
printf("Aradığınız sayı dizinin %d.ci elemanıdır\n",i+1);
break;
}
getch();
}
```

Örnek 7.4 Bir dizideki en küçük yada en büyük elemanı ve dizideki yerini bulma

```
#include<stdio.h>
# include<conio.h>
main()
{
int dizi[35],en,i,eleman,konum;
clrscr();
printf ("\nDizinin eleman sayısını giriniz...");
scanf("%d",&eleman);
printf("\nDizi elemanlarını giriniz...");
for (i=1; i<=eleman;i++) scanf("%d",&dizi[i]);
i=1;
en=dizi[1];
konum=i;
for(i=1;i<=eleman;i++)
   if(en>dizi[i])  // Eğer en<dizi[i] yazılırsa en büyük eleman bulunabilir
   {
   en=dizi[i];
   konum=i;
   }
printf("Dizinin en küçük değeri = %d\n", en);
printf("Elemanın dizideki yeri=%d\n", konum);
getch();
}
```

Örnek 7.5 Eleman sayısı dışarıdan girilen bir dizinin elemanlarını küçükten büyüğe doğru sıralama

```
#include<stdio.h>
#include<conio.h>
int eleman,i,,j,b[100],c;
void main()
{
clrscr();
printf(“Dizinin eleman sayısını giriniz:\n”);
scanf(“%d”,&eleman);
printf(“Dizi elemanlarını giriniz:\n”);
for(i=0;i<eleman;i++){
printf(“b[%d]:”,i);
scanf(“%d”,&b[i]);
}
for(i=0; i<(eleman-1); i++)
for(j=i+1;j<eleman;j++)
if(b[i]>b[j])
{
c=a[i];
a[i]=a[j];
a[j]=c;
}
printf(“\nSıralı Dizi:\n”);
for(i=0;i<eleman;i++)
printf(“%d\t”,b[i]);
getch();
}
```

Örnek 7.6 Elemanları belli bir matrisi ekrana yazdırma

```
#include <stdio.h>
int matris[2][3]={1,2,3,4,5,6};
int i, j;
main()
{
for (i=0;i<2;i++)
{
for(j=0;j<3;j++)
    printf("%d",matris[i][j]);
    printf("\n");
}
}
```

Örnek 7.8 Elemanları dışarıdan girilen iki matrisin toplanması

```
#include <stdio.h>
int matris1[3][3];
int matris2[3][3];
int matris3[3][3];
int i, j;
main()
{
for (i=0;i<3;i++)
for(j=0;j<3;j++)
{
printf("\nmatris1[%d][%d]=",i,j);
scanf("%d%d",&matris1[i][j]);
printf("\nmatris2[%d][%d]=",i,j);
scanf("%d%d",&matris2[i][j]);
}
for (i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
    matris3[i][j]=matris1[i][j]+matris2[i][j];
    printf("%d",matris3[i][j]);
}
    printf("\n");
}
}
```

Örnek 7.9 3x3 boyutunda bir birim matris oluşturan ve ekrana yazdıran program

```
#include<stdio.h>
#include<conio.h>
  int a[3][3];
  int i, j;
 void main()
  {
    clrscr();
     for(i=0;i<3;i++)
      for(j=0;j<3;j++)
         if(i==j)
           a[i][j]=1;
         else
           a[i][j]=0;
      clrscr();
       for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
          {
          printf("%d",a[i][j]);
          }
Printf("\n");
}
   getch();
  }
```

Ekran çıktısı:

```
100
010
001
```

Örnek 7.10 3x3 boyutunda a ve b matrislerini dışarıdan okutarak c=a*b matrisini hesaplatıp yazan program:

```
#include<stdio.h>
#include<conio.h>
  int x[2][2], y[2][2], z[2][2];
  int i, j,k;

 void main()
 {
   clrscr();
    printf("1. matrisin elemanlarını giriniz\n");
    for(i=0;i<2;i++)
     for(j=0;j<2;j++)
        {
         printf("x[%d][%d]=",i,j);
         scanf("%d",&x[i][j]);
        }

     printf("2. matrisin elemanlarını giriniz\n");
      for(i=0;i<2;i++)
       for(j=0;j<2;j++)
       {
        printf("y[%d][%d]=",i,j);
        scanf("%d",&y[i][j]);
       }
   printf("\n");

      for(i=0;i<2;i++)
       for(j=0;j<2;j++)
         for(k=0;k<2;k++)
             z[i][j]= z[i][j]+x[i][k]*y[k][i] ;

      for(i=0;i<2;i++)
       {
         for(j=0;j<2;j++)
         printf("z[%d][%d]=%d",i, j, z[i][j]);
         printf("\n");
        }
  getch();
 }
```

Ekran çıktısı:

```
1. matrisin elemanlarını giriniz
x[0][0]=2
x[0][1]=3
x[1][0]=5
x[1][1]=7

2. matrisin elemanlarını giriniz
y[0][0]=1
y[0][1]=2
y[1][0]=3
y[1][1]=4

z[0][0]=11    z[0][1]=16
z[1][0]=26    z[1][1]=38
```

Örnek 7.11 Girilen illeri ekranda gösterme (String dizi örneği)

```
#include<stdio.h>
#include<conio.h>
  char il[3][10];
  int i;

 void main()
 {
   clrscr();
    printf("İlleri giriniz:\n");
    for(i=0 ; i<3 ; i++)
        {
         printf("iladı[%d]:",i);
         scanf("%s",il[i]);
        }
    for(i=0 ; i<3 ; i++)
       printf("%s\n",il[i]);
    getch();
 }
```

Ekran çıktısı:

```
İlleri giriniz:
Elazığ
İstanbul
Kahramanmaraş

Elazığ
İstanbul
```

Kahramanmaraş

7.6. DEĞERLENDİRME SORULARI

1. Klavyeden rastgele girilen 10 sayıyı okuyup bu sayıları tersten yazdıran bir C programı yazınız?
2. Klavyeden girilen bir metnin uzunluğu bulan bir C programı yazınız?
3. Bir kare matrisin determinantını bulan bir C programı yazınız?
4. Klavyeden girilen bir metni tersten yazdıran bir C programı yazınız?
5. Klavyeden girilen bir cümledeki kelimelerin sadece ilk harflerini yazdıran bir C programı yazınız?
6. Klavyeden girilen bir cümledeki kelimelerin yerini bozmadan tersten yazdıran bir C programı yazınız? (Örn: Ali Topu At ➔ At Topu Ali)
7. Mevcudu klavyeden girilen bir sınıftaki öğrencilerin sınav sonuçlarını okutup ortalamasını ve en yüksek notu bulan programı yazınız?
8. 25 elemanlı bir dizideki negatif elemanların toplamını bulan programı yazınız?
9. 20 elemanlı bir dizide kaç negatif, kaç pozitif ve kaç tane 0 olduğunu bulan ve ekrana yazdıran programı yazınız?
10. 50 elemanlı bir dizide değeri 0'dan küçük olanları neg isimli diziye, değeri 0 ya da 0'dan büyük olanları poz isimli diziye aktaran programı yazınız?
11. Eleman sayısı dışarıdan girilen bir dizideki çift olan elemanların ortalamasını bulan programı yazınız?
12. n*n boyutundaki bir dizinin esas köşegeni üzerindeki elemanların ortalamasını ve toplamını bulan programı yazınız?
13. n*n boyutundaki bir matrisin satırlarını ve sütunlarını toplayıp ayrı aryrı sonuçları ekranda gösteren programı yazınız?
14. n*n boyutundaki bir matrisin simetrik olup olmadığını bulan programı yazınız?
15. n*n boyutundaki bir matrisin en büyük elemanını en küçük elemanına böldüren ve sonucunu ekrana yazdıran programı yazınız?

İŞARETÇİLER

Bölüm 8

8.1 GİRİŞ

Tüm bilgisayar programlama dillerinin temelinde işaretçiler veri tipleri bulunmaktadır. Turbo C' de işaretçilerin kullanımı en üst düzeydedir, hatta zorunludur.

Tipik olarak bir bilgisayar, ayrı ayrı veya bitişik gruplar halinde idare edilebilen ardışık olarak sayısallaştırılmış veya adreslenmiş hafıza hücrelerinin bir dizisine sahiptir. İşaretçiler, hafızadaki bir yerin adresini (byte sıra numarasını) içeren bir değişkendir.

BÖLÜM – 8

İŞARETÇİLER

Bölümün Genel Amacı: Turbo C programlama dilinde işaretçilerin yeri ve kullanma

Bölümün Davranışsal Amaçları: Kitabınızın bu bölümünü başarıyla bitirip, uygulamaları yapıp, değerlendirme sorularına doğru cevap verdiğiniz taktirde, bölüm sonunda;

- Tanımlanması ve Kullanımı
- İşaretçi Aritmetiği
- İşaretçiler ve Diziler
- İşlevleri Referans Yoluyla Çağırma
- İşaretçiler ve Yapılar
- Dinamik bellek kullanımı
- İşaretçilerle İlgili Diğer Konular

Değerlendirme: Modül sonundaki uygulamaları yapmanız sonuçları uygulama raporu ile karşılaştırmanız ve değerlendirme sorularına en az % 75 düzeyinde doğru cevap verebilmeniz gerekmektedir.

8.2 TANIMLANMASI VE KULLANIMI

Bir veri bloğunun bellekte bulunduğu adresi içeren (gösteren) veri tipidir. Tanımlama biçimi:

veri tipi *p;

p değişkeni <veri tipi> ile belirtilen tipte bir verinin bellekte saklandığı adresi içerir.

```
int *iptr;
float *fptr;
```

Bu kadar tanımla sonucunda bellekte p değişkeni mevcuttur. Ancak işaret ettiği veri bloğu yoktur. Bunun için iki yol vardır. Birincisi kullanılan herhangi bir değişkeni işaret etmek, ikincisi ise veri bloğunu boş belleği kullanarak oluşturmak.

8.2.1. İşaretçi değişkenin var olan bir değişkenin bulunduğu adresi göstermesi.

Bu işlemi yapabilmek için var olan değişkenin adresinin bilinmesi gerekmektedir.

& işleci : Bir değişkenin adresinin belirlenmesi için kullanılır. Kullanım biçimi:

&değişken

&i : i değişkenin adresini verir.

```
main()
{
  int i;
  int *iptr;
  i = 5;
  iptr = &i;
  clrscr();
  printf("i değişkeninin adresi    %p\n", &i);
  printf("iptr değişkeninin değeri %p\n", iptr);
}
```

Bellek modeline göre SSSS:OOOO veya OOOO biçiminde adres yazar.

8FF8:1000

8.2.2. Veri bloğunu boş belleği kullanarak oluşturmak.

Bu yolla veriler için dinamik yer ayırılır. Bunun için malloc işlevi kullanılır
void *malloc(n) : Boş bellekten n byte yer ayırıp başlangıç adresini döndürür.
iptr = (*int) malloc(2);
!!!!!!!!! Daha sonra dönüş yapılacak. sizeof, cast işleci (*tip) ...

8.2.3. Veriye işaretçi değişken yoluyla erişim

Bir işaretçinin gösterdiği adresteki veriye erişmek için işaretçi değişkeninin önüne * karakteri konur.

```
main()
{
  int i;
  int *iptr;
  iptr = &i;
  *iptr = 8;
  printf("i değişkeninin değeri %d\n", i);
  printf("iptr adresinin içeriği %d\n", *iptr);
}
```

Ekranda çıktı :

```
i değişkeninin değeri 8
iptr adresinin içeriği 8
```

!!! İşaretçi değişkenin gösterdiği adresin içeriği değişken ilklendirmeden kullanılmamalıdır

8.3 İŞARETÇİ ARİTMETİĞİ

İşaretçi değişkenler üzerinde toplama ve çıkartma işlemleri (++, --) geçerlidir. Ancak eklenecek değer tamsayı olmalıdır.
İşaretçi değişkenin değeri 1 arttırıldığı zaman değişken bir sonraki veri bloğunu işsaret eder. Değişkenin alacağı yeni değer işaretçi değişkenin ne tip bir veri bloğunu işaret ettiğine bağlıdır.

```
int *iptr, i;
...
iptr = &i;              i değişkenin adresinin 1000 olduğunu varsayalım. iptr nin değeri
1000 dir.
iptr++;              iptr nin değeri 1002 olur. ( int değeri işaret ettiği için)

aynı örneği double için yaparsak
double *iptr, i;
...
iptr = &i;              i değişkenin adresinin 1000 olduğunu varsayalım. iptr nin değeri
1000 dir.
iptr++;              iptr nin değeri 1008 olur. ( double değeri işaret ettiği için)
int *iptr, i, j;
...
iptr = &i;              i değişkenin adresinin 1000 olduğunu varsayalım. iptr nin değeri
1000 dir.
*(iptr+4)=2;         1008 adresinin içeriğini 2 yapar.
```

!!! Arttırma işaret edilen veri bloğuna göre yapılır Yani bir sonraki veri bloğunun gösterilmesi sağlanır.

iptr++ ; bir sonraki veri bloğunu göster
(*iptr)++; iptr değişkeninin gösterdiği adresteki değeri 1 arttır

8.4 İŞARETÇİLER VE DİZİLER

İşarteçiler üzerinde geçerli aritmetik yardımıyla dizilere işaretçi değişkenler ile erişmek mümkündür.

```
#include <stdio.h>
main()
{
  int i[10], j;
  int *iptr;

  for (j=0; j<10; j++)
       i[j]=j;

  /* Dizinin başlangıç adresine erişmek için ilk elemanın adresi kullanılabilir  &i[0]  veya
doğrudan */

  iptr = i;

  clrscr();

  for (j=0; j<10; j++) {
      printf("%d ", *iptr);
      iptr++;
  }
  printf("\n");
/* iptr artık dizinin başını göstermez */

  iptr = i;
  for (j=0; j<10; j++)
      printf("%d ", *(iptr+j));

  printf("\n");
/* iptr hala dizinin başını gösterir */
  getch();
}
```

Örnek 8.1: İşaretçi ve dizgi kullanımı.

```
#include <stdio.h>
main()
{
  char *a="1234567890";
  char b[11];
  char *p1, *p2;

  printf("%s\n", a);
  p1 = a;
  p2 = b;
  while (*p1 != '\0') {
     *p2 = *p1;
     p1++;
     p2++;
  }
  printf("%s\n", b);
}
```

8.5 İŞLEVLERİ REFERANS YOLUYLA ÇAĞIRMA

Şu ana yazdığımız işlevlerde gönderilen parametrelerin (diziler hariç) değerlerinin değiştirilmesi mümkün değil idi. İşlev çağırıldığı zaman parametrelerin bir kopyası çıkartılıp işleve gönderiliyordu. Bir işlevin birden fazla değer gönderebilmesi için işaretçilere gereksinimiz vardır.

```
void arttir(int);
main()
{
  int i;
  i = 5;
  printf("öncesi %d\n", i);
  arttir(i);
  printf("sonrası %d\n", i);
  getch();
}
void arttir(int k)
{
  k++;
}
```

Çıktı :

```
öncesi 5
sonrası 5
```

Gönderilen parametrenin kopyası işleve gönderildiği için işlev içerisinde yapılan değişiklikler işlevin çağırıldığı yeri etkilemez. Eğer parametredeki değişikliklerin işlevin çağırıldığı yerde de geçerli olmasını istiyorsak işleve parametrenin adresini göndermek gerekir.

```
void arttir(int*);
main()
{
  int i;
  i = 5;
  printf("öncesi %d\n", i);
  arttir(&i);
  printf("sonrası %d\n", i);
  getch();
}
void arttir(int *k)
{
  (*k)++;
}
```

Çıktı :

```
öncesi 5
sonrası 6
```

Örnek 8.2: Sayısal dizgiyi tamsayıya dönüştüren işlevde iyileştirme. Geçersiz karakterin konumu da verilsin.

```
int deger(char *s, int *konum)
konum = -1 ise tüm karakterler rakam
        >=0 ise geçersiz karakterin konumu
```

Örnek 8.3: Sıraya dizme. Yer değişikliği işlevde ve parametrelere referans yolu ile erişim.

```
#include <stdio.h>
#include <conio.h>
#define N 20
void degistir (int *, int *);

main()
{
   int s[N];
   int i, k;
   clrscr();
   for (i=0; i<N; i++) {
       s[i] = rand() % 100;
       printf("%4d",s[i]);
   }
   printf("\n");
   k=1;
   do {
      k=0;
      for (i=0; i<N-1; i++)
          if (s[i] > s[i+1]) {
             degistir (&s[i], &s[i+1]);
             k = 1;
          }
   } while (k);
   for (i=0; i<N; i++)
       printf("%4d",s[i]);
   printf("\n");
   getch();
}

void degistir (int *a, int *b)
{
  int gec;
  gec = *a;
  *a = *b;
  *b = gec;
}
```

!!! Dizilerde işaretçi olduğu için a değişkeni bir dizi(veya işaretçi ise) a[i] ile *(a+i) ifadeleri aynı anlamı taşır.

Örnek 8.4: İşleve gönderilen dizinin işlev içerisinde işaretçi olarak kullanımı.

```
#include <stdio.h>
#include <conio.h>
#define N 5

float ort (int *);

main()
{
   int s[N];
   int i, k;
   clrscr();
   for (i=0; i<N; i++) {
       s[i] = rand() % 100;
       printf("%4d",s[i]);
   }
   printf("\n");
   getch();
}
float ort (int *a)
{
  int i;
  float t = 0;
  for (i=0; i<N; i++)
      t = t + *(a+i);
  return t/N;
}
```

Örnek 8.5: işleve gönderilen işaretçinin işlev içerisinde dizi olarak kullanımı .

void malloc(n): En az n byte uzunluğunda bellekten yer ayırır. İşlevin değeri >0 ise bloğun bellekteki yeri, NULL yer yok demektir.

```
int *i;
i = (int *) malloc(2000) ;    2000 byte yer ayırıp bloğun başlangıç adresini i 'ye atar
                                ( 1000 elemanlı int dizisi )
double *x;
x = (double *) malloc(8*2000);  2000 elemanlı double dizi
```

sizeof(n) : n ifadesinin/tipinin byte olarak uzunluğunu verir.

```
i = (int *) malloc(1000*sizeof(int)) ;    1000 tane int değer içerecek bellek uzunluğu

x = (double *) malloc(2000*sizeof(double));  2000 elemanlı double dizi
```

void free (void *block) : mallock işlevivi tersi. Block değişkenin tuttuğu yeri boş belleğe gönderir

```
#include <stdio.h>
#include <conio.h>
#define N 8
float ort (int []);

main()
{
   int *s;
   int i, k;
   s = (int *) malloc(2*N);
   clrscr();
   for (i=0; i<N; i++) {
       s[i] = rand() % 10;
       printf("%4d",*(s+i));
   }
   printf("\n");
   printf("Ortamala = %.2f\n",ort(s));

   getch();
}
float ort (int a[])
{
  int i;
  float t = 0;
  for (i=0; i<N; i++)
      t = t + a[i];
  return t/N;
}
```

Örnek 8.6: Bir dizinin elemanlarının işaretçi olması.

Daha önce yapılan bir örnekte ay isimleri bir dizide saklanmıştı.

```
main()
{
  char *aylar[] = {"", "Ocak", "ªubat", "Mart", "Nisan",
                   "Mayıs", "Haziran", "Temmuz", "Ağustos",
                   "Eylül", "Ekim", "Kasım", "Aralık"};
  int i;
  printf("Ayın sırasını gir "); scanf("%d", &i);
  if (i>0 && i<13)
     printf("%s\n", aylar[i]);
  getch();
}
```

Benzer şekilde

```
float *a[100];
```

tanımlaması her bir elemanı bellekte bir 'float' sayıyı gösteren işaretçi olan 100 elemanlı bir dizidir.

Örnek 8.7 : Bir işaretçinin adresini içeren işaretçiler.

```
main()
{
  int i;
  int *iptr;
  int **iptrptr;
  i = 5;
  iptr = &i;
  iptrptr = &iptr;
  clrscr();
  printf("   i   ve   &i   : %d   %p\n", i, &i);
  printf(" *iptr  ve   iptr   : %d   %p\n", *iptr, iptr);
  printf("*iptrptr ve  iptrptr :  %p   %p\n", *iptrptr, iptrptr);
  getch();
}
```

Ekrana çıktı:

```
       i   ve   &i    : 5   8FDD:1000
   *iptr  ve   iptr   : 5   8FDD:1000
*iptrptr ve  iptrptr :  8FDD:1000   8FDD:0FFC
```

8.6 İŞARETÇİLER VE YAPILAR

Bir işaretçi işleve parametre olarak gönderildiğinde basit değişken gibi değişkenin kopyası alınıp gönderiliyordu. Yapının büyük olduğu durumlarda bu da sorun çıkartır. İşaretçinin bir yapı verisini göstermesi.

```
struct ogrenci{
  char no[10];
  int  notu;
};

struct ogrenci *a
```

Tanımlamasında a değişkenini oluşturan alanlara erişmek için, bilinen yol:

```
*a.notu=56;
strcpy((*a).no, "95001");
```

Bunun farklı kullanımı:

```
a->notu=56;
strcpy(a->no, "95001");
```

Örnek 8.8: Yapının adresinin işleve gönderilmesi.

```
#include <stdio.h>
typedef struct {
    char adi[35];
    char adres1[40];
    char adres2[40];
    char tel[15];
    float borc;
}
kisiler;
void yaz(kisiler *z);
main()
{
   kisiler  a;
   clrscr();
   printf("Adını gir  : "); gets(a.adi);
   printf("Adres-1    : "); gets(a.adres1);
   printf("Adres-2    : "); gets(a.adres2);
   printf("Telefonu   : "); gets(a.tel);
   printf("Borcu      : "); scanf("%f", &(a.borc));
   yaz(&a);
}
void yaz(kisiler *z)
{
  clrscr();
  printf("Adı        : "); puts(z->adi);
  printf("Adresi     : "); puts(z->adres1);
  printf("           : "); puts(z->adres2);
  printf("Telefonu   : "); puts(z->tel);
  printf("Borcu      : "); printf("%.0f\n", z->borc);
}
```

8.7 DİNAMİK BELLEK KULLANIMI

Üç boyutlu dizi tanımı ve kullanımı. Üç boyut --> Gerilim - Akım - Zaman

Örnek 8.9: Tek boyutlu diziyi üç boyutlu gibi kullanma.

```
/* Üç boyutlu dinamik dizi kullanımı. Her boyut farklı uzunlukta
   Dizi tek boyutlu gözüküyor. Ancak indis, hesaplanarak bulunuyor.  Yani

      *(a + i*y*z + j*z +k)
            ^      ^
            ^      ^ 3. boyutun uzunluğu
            ^2. boyutun uzunluğu

   şeklinde kullanılabilir. */

#define x    4
#define y    5
#define z    9
void matris_yaz(int *);
main(){
  int *a; int i, j, k;
  a=(int *) malloc(x * y * z * sizeof(int)); /* eleman sayisi kadar yer ac */
  clrscr();
  for (i=0; i<x; i++) {
      printf("i = %d \n", i);
      for (j=0; j<y; j++) {
          for (k=0; k<z; k++) {
              *(a + i*y*z + j*z +k) = i*j*k;
              printf("%5d ",*(a + i*y*z + j*z +k));
          }
          printf("\n");
      }
  }
  matris_yaz(a);
}
void matris_yaz(int *m)
{
  int i, j, k;
  clrscr();
  for (i=0; i<x; i++) {
      printf("i = %d \n", i);
      for (j=0; j<y; j++) {
          for (k=0; k<z; k++) {
              printf("%5d ",*m);
              *m++;
          }
          printf("\n");
      }
      getch();
  }
}
```

Örnek 8.10:

```
/* Uc boyutlu dinamik dizi kullanimi. Her boyut farkli uzunlukta
   Bu yontem ile diziye normal dizi gibi erismek mumkun
   Yani
      a[i][j][k]
   seklinde kullanilabilir.
*/

#define x    8
#define y    4
#define z   10
main()
{
/*
  typedef int *boyut1;
  typedef boyut1 *boyut2;
  typedef boyut2 *boyut3;
  boyut3 a;
*/
  double ***a;
  int i,j,k;

  a=(double *) malloc(x*sizeof(double*));          /* 1. boyut icin yer ayir  (işaretçiler)   */
  for (i=0; i<x; i++)                              /* 2. boyut icin yer ayir.  (işaretçiler)  */
     *(a+i)=(double *) malloc(y*sizeof(double*)); /* 1. boyutun her elemani n   */
                                                  /* elemanli diziyi gosterir   */

  for (i=0; i<x; i++)                  /* 3. boyut icin yer ayir (matrisin elemanlar)    */
     for (j=0; j<y; j++)
        *(*(a+i) + j) = (double *) malloc(z*sizeof(double));
  clrscr();
  for (i=0; i<x; i++)
     for (j=0; j<y; j++)
        for (k=0; k<z; k++)
           *(*(*(a + i) + j) + k) = i*j*k;
  for (i=0; i<x; i++) {
     printf("i = %d \n", i);
     for (j=0; j<y; j++) {
        for (k=0; k<z; k++)
           printf("%4.1f ",a[i][j][k]);
        printf("\n");
     }
     getch();
  }
}
```

8.8 İŞARETÇİLERLE İLGİLİ DİĞER KONULAR

İşaretçinin Belirli Bir Adresi Göstermesi

```
#include <dos.h>
#include <stdio.h>
char far *ekran;
void kaydir_Y(void);
void kaydir_A(void);
void main()
{
   int i, j;
   char c;
   ekran = MK_FP(0xB800, 0);
   clrscr();
   for (i=0; i<25; i++)
       for (j=0; j<80; j++)
           ekran[160*i+2*j] = 65+i;
   while (1) {
      c = toupper(getch());
      switch(c) {
          case 'A' :  /* yukari */
                      kaydir_Y();
                      break;
          case 'Z' :  /* asagi */
                      kaydir_A();
                      break;
          case 'Q' : exit(0);
      }
   }
}
void kaydir_Y(void)
{
  int i, j;
  for (i=8; i<=12; i++)
      for (j=30; j<60; j++)
          ekran[160*(i-1) + 2*j] = ekran[160*i + 2*j];
}
void kaydir_A(void)
{
  int i, j;
  for (i=12; i>=8; i--)
      for (j=30; j<60; j++)
          ekran[160*(i+1) + 2*j] = ekran[160*i + 2*j];
}
```

8.8.1. İşlev İşaretçileri

İşaretçinin bir işlevin bulunduğu adresi içermesi durumudur. Normal işaretçi gibi işlevin adresini içeren değişken tanımı yapılmalıdır. Örneğin;

int (*fnptr) (int, int)

fnptr değişkeni iki tane int parametresi olup bir int değer geri gönderen bir işlevin adresini içerebilir.

(int *fnptr (int, int) : iki int parametresi olup int işaretçi geri gönderir)

Örnek 8.11: Aynı isim ile farklı iki işlevi çağırma.

```
int kare(int);
int kub(int);
main()
{
  int (*islem)(int);                /* bir int değer alıp geriye int değer gönderen   bir işlevin
adresi  */
  int i;
  char c;
  clrscr();
  printf("1 / 2 : kare / küb hesabı : ");
  c = getch();
  printf("\nSayıyı gir : ");
  scanf("%d", &i);

  if (c == '1')
     islem = kare;      /* kare işlevinin adresi islem değişkenine kopyalanır */
  else
     islem = kub;
  printf("Sonuç = %d\n", islem(i));
  getch();
}
int kare(int s)
{
 return s*s;
}
int kub(int s)
{
 return s*s*s;
}
```

8.8.2. Void İşaretçiler

İşaretçiler void olarak tanımlanabilir. Bu biçimde tanımlanan işaretçilerin gösterdiği adresteki değere erişmek için veri tipi belirtilmelidir.

```
main()
{
  void *a;
  a = (char*) malloc(10);
  strcpy(a,"12345");
  printf("%s\n",  a);
  free(a);
  a = (double*) malloc(sizeof(double));
  *(double*)a = 3.123;                /* değere erişirken veri tipi belirt */
  printf("%f\n", *(double *)a);
  getch();
}
```

İÇİNDEKİLER

ÖNSÖZ

Gün geçtikçe gelişen teknoloji sayesinde insanlar, çeşitli problemlerine çözüm ararken bilişim teknolojilerini büyük bir yardımcı olarak görmektedir. Bu doğrultuda, hemen her alanda bilgisayarın karmaşık problemlerin üstesinden gelmesi beklenen bir durumdur. Beklentilere cevap verecek olan bilgisayar yazılımlarının geliştirilmesi şüphesiz bilgisayarların ve bilişim teknolojilerinin varlığı kadar önemlidir. Bilgisayarları, toplumun ve bireylerin amaçlarına hizmet eden birer makine haline getirmek, yazılımların ve onları geliştiren yazılım geliştiricilerin yani programcıların işidir.

İyi bir programcı olabilmenin ilk basamaklarını olaylara değişik açılardan bakabilmek, problemlere en ucuz ve en çabuk çözükleri üretebilmek ve tabiidir ki azimle çalışmak oluşturur. Bu kitapla ilk adımı atacağınız programcılık dünyasında program geliştirmenin ilk adımından programın elde edilmesi ve varsa, bir problemin çözülmesine kadar olan süreç enine boyuna incelenecektir.

Yrd. Doç. Dr. Hasan H. BALIK

EDİTÖR: YRD. DOÇ.DR. HASAN H. BALIK

"C" İLE PROGRAMLAMAYA GİRİŞ

Ahmet TEKİN
Ayhan AKBAL
Bahadır SEVİNÇ
Fatih ERTAM
Harun H. TUZSUZOĞLU
İhsan SERHATLIOĞLU
Kemal BALIKÇI
M. Fatih TALU
Musa ÇIBUK
Oğuzhan ÖZDEMİR
Resul DAŞ
Yaman AKBULUT
Zülfü GENÇ

ELAZIĞ - 2003