



**CEVİRİ KİTAPLAR SERİSİ**

**NO 01**

**PICMICRO MCU C**

**CCS C ile MICROCHIP PIC  
Programlama Kılavuzu**



**MICROCHIP**

## TEŞEKKÜRLER..

Bu kitap PICPROJE Ekibi tarafından eğitime katkı amacıyla yapılmıştır.

### Emeği geçen arkadaşlarımız.. (Alfabetik)

Grup Moderatörü	CoşkuN
	Bunalmis
	CaFFeiNe
	Gsever
	Kurumahmut
	Macera
	Microcozmoz
	Otm
	Oyhan
	Petek
	Picusta
	Respected
	SpeedyX
	Timpatı
	Ziya

[www.picproje.org](http://www.picproje.org)

TÜRKİYE'NİN ELEKTRONİK PLATFORMU

# İçindekiler

## Önsöz

Tarihçe  
Neden C?  
PC' ye karşı PICmicro MCU tabanlı program geliştirme  
Ürün geliştirme  
Terminoloji  
Kodu deneme ve test etme  
C kodlama standartları  
Temeller

## 1 C Temelleri

C programlarının yapısı  
Bir C programının bileşenleri  
#pragma  
main()  
#include  
printf Fonksiyonu  
Değişkenler  
Sabitler  
Yorumlar  
Fonksiyonlar  
C komutları

## 2 Değişkenler

Veri Tipleri  
Değişken Bildirimleri  
Değişken Atamaları  
Sıralama  
typedef  
Tip Dönüşümleri

## 3 Fonksiyonlar

Fonksiyonlar  
Fonksiyon Prototipleri  
Fonksiyon Argümanlarının Kullanımı  
Fonksiyonlardan Değer Döndürme  
Klasik ve Modern Fonksiyon Bildirimleri

## 4 Operatörler

Aritmetik  
İlişkisel  
Mantıksal  
Bit Operatörleri  
Arttırma ve Azaltma  
Operatör Öncelikleri

## 5 Program Kontrol İfadeleri

if  
if-else  
?  
for döngüsü  
while döngüsü  
do-while döngüsü  
Program Kontrol İfadelerinin İççe Kullanımı  
Break  
Continue  
Null  
Return

## 6 Diziler / Katarlar

Tek Boyutlu Diziler  
Katarlar  
Çok Boyutlu Diziler  
Öndeğerli Diziler  
Katar Dizileri

## 7 İşaretçiler

İşaretçi Temelleri  
İşaretçiler ve Diziler  
Fonksiyonlara Geçen İşaretçiler

## 8 Yapılar / Birleşimler

Yapı Temelleri  
Yapı İşaretçileri  
İççe Yapılar  
Birleşim Temelleri  
Birleşim İşaretçileri

## 9 PICmicro MCU'e özgü C

Girişler ve Çıkışlar  
C ve Assembly Birlikte Kullanımı  
Gelişmiş BIT İşleme  
Zamanlayıcılar  
A/D Çevrim  
Veri İletişimi  
I<sup>2</sup>C İletişimi  
SPI İletişimi  
PWM  
LCD Sürme  
Kesmeler  
Kütüphaneler

## Önsöz

### **Neden C?**

C dili Dennis Ritchie ve Brian Kernighan tarafından 1970'li yılların başlarında Bell laboratuvarlarında geliştirilmiştir. İlk uygulama platformlarından biri UNIX ortamında çalışan bir PDP-11 olmuştur. Tanıtımından bu yana, kendini kanıtlamış bir uygulama geliştirme dili olarak bilgi işlem endüstrisinin başından sonuna dek evrim geçirmiştir ve standartlaştırılmaktadır. PC, C++ ve diğer ANSI standardındaki versiyonları için düşük maliyetli bir geliştirme ortamı olmuştur.

C, programların bir bilgisayardan başka bir bilgisayara minimum değişiklik yapılarak aktarılabilmesi için taşınabilir bir dil olarak tasarlanmıştır. PC'ler ve anabilgisayarlar ile çalışırken bu çok iyi bir özelliktir. Fakat mikrokontrolörler ve mikro işlemciler farklı türlerdir. Ana program akışı değişmeden kalacaktır ama çeşitli ayarlar ve port/çevresel kontroller mikro işlemciye özgüdür. Buna örnek olarak PICmicro MCU için port yön yazmaçlarını verebiliriz, 1=Giriş 0=Çıkış iken H8 mikrokontrolörler için 0=Giriş 1=Çıkışı belirtir.

Üreticilerin daha fazla program ve RAM hafızaları ile yüksek işletim hızları sunması C dilinin mikrodenetleyici uygulamalarında kullanılmasına sebep olmuştur.

Bana aktarılan bir örnek – hiçbir şeye inanmayan biri olarak – şuydu: bir zamanlayıcı fonksiyonunu assemblyde iki haftada C' de ise birgünden kısa bir sürede yazabilirsiniz. Yani hemen koşup bir C derleyicisi almamı mı söylüyorsunuz – assemblyde yazmaktan niye sıkılayımki? Daha kısa kod üretiyor – assemblyde yazılmış bir program C' de yazılmış bir programa göre ortalama %20 daha az yer kaplıyor. Program hafızası yüksek modellerde iyi fakat düşük modellerde pek randımanlı değil.

## PC' ye karşı PICmicro MCU tabanlı program geliştirme

Mühendisler temel donanımının (klavye, işlemci, hafıza, G/Ç, yazıcı ve ekran gibi) hazır olmasından dolayı PC tabanlı birimlerle ürün geliştirmeye başlamışlardır. Ürün geliştirme yazılımı hazırlamaktan ve hatalarını gidermekten oluşuyordu.

PIC tabanlı ürün geliştirmek, MCU'nun dış dünya ile iletişiminin giriş çıkış donanımları şeklinde oluşturulmasıyla başlar. Bir PC programcısı "Merhaba Dünya" yazan bir programı derleyip çalıştırdığında ekranda mesajı hemen görür. PIC programcısı mesajı görebilmek için RS232 arabirimini yapmak, PIC içerisinde iletişim portunu ayarlamak ve geliştirme kartını PC üzerindeki iletişim portuna bağlamak zorundadır.

Bu kadar zahmete gerek varmı? dediğinizi duyar gibiyim. Eğer tüm PC' yi (monitör ve klavye ile birlikte) 40 bacaklı bir entegre kılıfında edinebilseydik onu kullanırdık: bu günkü imkanlarla henüz buna sahip değiliz. Bunun için düşük maliyet ve taşınabilir uygulamalar için PIC veya benzeri mikrodenetleyiciler kullanmaya devam edeceğiz.

PIC tabanlı sistemler için geliştirme araçları grafik kütüphaneler haricinde PC tabanlı sistemlerdeki gibi geliştiriciye bazı temel kolaylaştırıcı seçenekler sunar.

## Ürün geliştirme

Ürün geliştirme tecrübe ve şansın birleşimidir. Bazı çok küçük program parçaları uzun geliştirme sürelerine mal olur fakat ürünün mükemmelliğinde büyük pay sahibidirler.

Bir ürün geliştirmek için şunlara ihtiyaç vardır: zaman – huzur ve sessizlik – mantıksal düşünce ve en önemlisi ihtiyaçların tam olarak anlaşılmasıdır. Bana göre geliştirmeye başlamak için en kolay yol fikirlere beraber birkaç sayfa temiz kağıttır.

İşe mümkün olan çözümleri çiziktirmekle başlayın ve en basit, en güvenilir seçeneği bulmak için herbirini dikkatle gözden geçirin. Bu aşamada diğer fikirleri çöpe atmayın içlerinde güzel düşünceler olabilir.

Akış diyagramı, blok diyagram, G/Ç bağlantılarını çizin.

Tüm G/Ç pinleri ile prototip kartınızı oluşturun. Kartınızı G/Ç pinleri değiştirilebilecek şekilde dizayn etmeniz kullanışlı olacaktır.

Programınızı – test edilebilir bloklar şeklinde – yazmaya başlayın ve yavaş yavaş büyüterek programınızı oluşturun. Bu sizi 2000 satırlık bir programı tek seferde hatalardan arındırmaktan kurtarır!

Eğer ilk projenizse – O ZAMAN BASİT TUTUN – devasa bir uygulamadan önce komutlara alışmak, assembly tekniklerini ve hata ayıklamayı öğrenmek için bir LED i butonlarla yakıp söndürme gibi basit uygulamalar yapın.

Programınızı basit adımlarla test ederek oluşturun. Akış diyagramınızı elden geçirerek güncel kalmasını sağlayın.

## Fikir

Bir fikir belki EVREKA! tarzında sizden çıkabilir veya başka birinin bir ihtiyacı olarakta doğabilir.

Tasarım işlemine başlamadan önce gerekli olan temel terminoloji anlaşılmalıdır. Bu durumda PICmicro MCU tabanlı tasarım yapmaya başlamadan önce PIC dili (komut seti, terimler ve geliştirme kiti) tam olarak anlaşılmalıdır.

Şimdi genel terimlerle, PIC'ler hakkındaki gerçeklerle ve mikroişlemci ile mikrodenetleyici sistemler arasındaki farklarla başlıyoruz.

## Terminoloji

Kullanılan bazı temel terimler

**Mikrodenetleyici** bir yazılım olmadan hiçbir iş yapmayan plastik, metal ve kum yığınının başka birşey değildir. Yazılım ile kontrol edildiğinde sınırsız bir uygulama alanı vardır.

**G/Ç** dış dünya ile iletişime geçen MCU bacağıdır. Giriş veya çıkış olarak ayarlanabilir. Çoğu durumda G/Ç birimleri mikrodenetleyicinin iletişim, kontrol ve bilgi almasına imkan verirler.

**Yazılım** mikrodenetleyicinin işlem yapması ve çalışması için gerekli olan bilgidir. Başarılı bir uygulama veya ürün için hatalardan arındırılmış olması gerekir. Yazılım değişik dillerle yazılabilir C, pascal, basic veya assembly gibi (binary kodlardan oluşan makine dilinin bir üstü)

**Donanım** mikrodenetleyici, hafıza, arabirim bileşenleri, güç kaynağı, sinyal şartlandırma devreleri ile sistemin çalışmasını ve dış dünya ile iletişimini sağlayan bileşenlerden oluşur.

Başka bir bakış açısıyla (özellikle çalışmadığında) donanımı fırlatıp atabilirsiniz.

**Simülatör** MPLAB® geliştirme ortamı mikrodenetleyicinin dahili işlemlerine erişebilmeyi sağlayan kendi simülatörüne sahiptir. Olayların ne zaman meydana geleceğinin biliyorsanız bu tasarımınızı test etmeniz için iyi bir yoldur.

**In Circuit Emülatör** PC ile mikrodeneleyicinin takıldığı sokete bağlanan çok kullanışlı bir ekipmandır. Yazılımın PC üzerinde çalıştırılmasını fakat devre kartında mikrodeneleyici tarafından çalıştırılıyor gibi görünmesini sağlar. ICE programı adım adım çalıştırabilmenizi ve mikrodeneleyici kısmında neler olduğunu dış dünya ile nasıl iletişime geçtiğini izlemenizi sağlar.

**Programlayıcı** programı mikrodeneleyici hafızasına yüklemeye yarayan ve ICE yardımı olmadan çalışabilmesini sağlayan ünedir. Değişik şekillerde, boyutlarda ve fiyatlarda olabilir. Microchip' in PICSTART PLUS ve PROMATE II modelleri seri porta bağlanır.

**Kaynak dosyası** anlayabileceğiniz şekilde assembly dili ile yazılmış programdır. Kaynak dosyasının mikrodeneleyicinin anlaması için işlemde geçirilmesi gerekir.

**Assembler / Derleyici** kaynak dosyasını obje dosyasına çeviren yazılım paketidir. Hata kontrol özelliği vardır ve assembly işlemi esnasında oluşan hataları bildirir. MPASM Microchip' in tüm PIC' leri destekleyen en son assembler yazılımıdır.

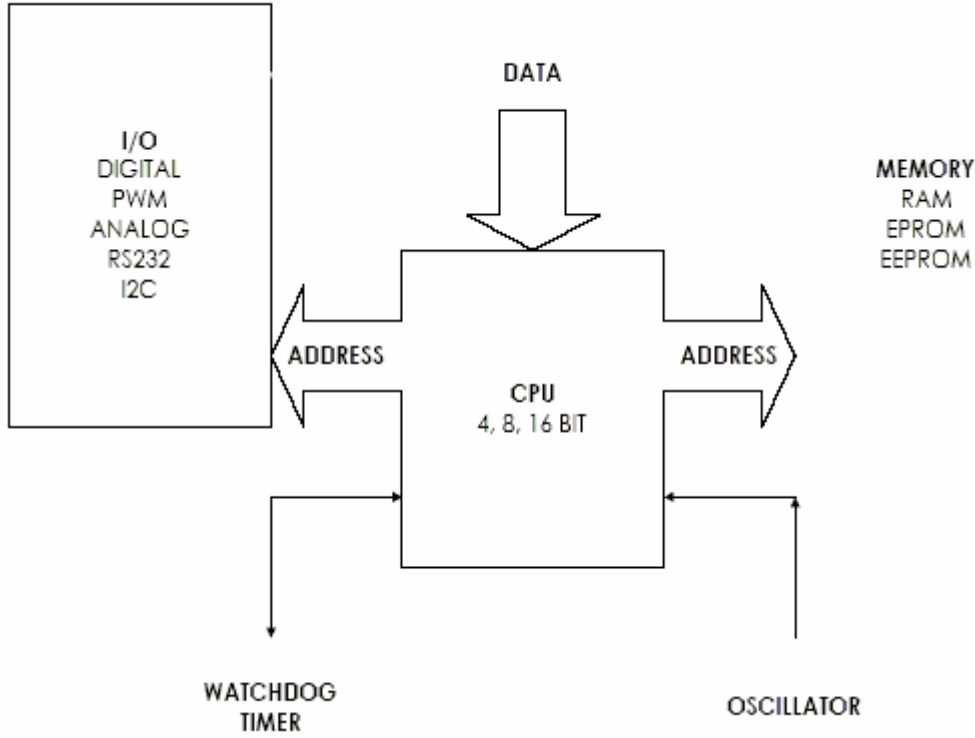
**Obje dosyası** Assembler / Derleyici tarafından üretilen ve programlayıcı, simülatör yada ICE' nin anlayabileceği formdaki dosyadır. Dosya uzantısı assembler direktifine bağlı olarak .OBJ veya .HEX olabilir.

**Liste dosyası** Assembler / Derleyici tarafından üretilen kaynak dosyasındaki tüm komutları bunların hexadesimal karşılıklarını ve yazdığınız açıklamaları içeren dosyadır. Dosya uzantısı .LST' dir.

**Diğer dosyalar** .ERR uzantılı dosyalar hataları içeren hata dosyalarıdır. .COD dosyaları emülatör tarafından kullanılır.

**Hatalar** sizin tarafınızdan yapılan yanlışlardır. Basit yazım hatalarından programlama dilinin yanlış kullanımına kadar çeşitli sebepleri vardır. Büyük bir çoğunluğu derleyici tarafından yakalanır ve .LST dosyasında gösterilir.

**Mikroişlemci** bir mikroişlemci veya dijital bilgisayar üç temel birimden oluşur. MİB (CPU), G/Ç ve hafıza – ile diğer destek devreleri



### TİPİK BİR MİKROİŞLEMCI SİSTEMİ

Giriş/Çıkış (G/Ç) dış dünya ile iletişimi sağlayan dijital, analog ve özel fonksiyonlardan oluşan birimdir.

Merkezi işlem birimi (MİB) hesaplama ve veri işleme işlemlerini yapan 4, 8 yada 16 bit veri formatları ile çalışabilen ve sistemin kalbi sayılan birimdir.

Hafıza RAM, ROM, EPROM, EEPROM veya bunların bir kombinasyonu şeklinde olabilir ve program ve verileri depolamaya yarar.

Osilatör mikroişlemcinin çalışabilmesi için gerekli saat frekansını sağlar. Osilatör değişik bileşenlerle oluşturulabilir veya hazır modül olarak elde edilebilir.

Mikroişlemci ile bağlantılı diğer devreler watchdog zamanlayıcısı – sistemin kitlenmesini önler –, adres ve veri yolları için tampon devreleridir – aynı yol üzerinde birden çok çipin birbirine zarar vermeden çalışabilmesini sağlar.

Mikroişlemci denildiğinde esas olarak MİB' den bahsedildiği anlaşılır. G/Ç ve hafıza ayrı çiplerden oluşur ve düzgün bir şekilde çalışabilmek için adres ve veriyolları ile adres çözme devrelerine ihtiyaç duyarlar.

### Mikrodenetleyiciler

PICmicro MCU ve diğer mikrodenetleyiciler tek çip içerisinde MİB (CPU), hafıza, osilatör, watchdog ve G/Ç birimlerini barındırırlar. Bunun sonucunda

yerden tasarruf, tasarım zamanının azalması ve harici birimlerle ilgili uyumluluk sorunlarının azalması sağlanır fakat bazı durumlarda sabit hafıza boyutu ve sınırlı G/Ç kapasitesi nedeniyle tasarımın sınırlarını daraltır.

PIC mikrodenetleyici ailesi geliştiricilere en çok ihtiyaç duyacakları geniş bir aralıkta G/Ç, hafıza ve özel fonksiyonları sunar.

## **Neden PIC kullanırız**

**Kod verimliliği** PIC Harvard mimarisini esas alan 8 bit bir mikrodenetleyicidir. – bunun anlamı içerisinde hafıza ve veri için farklı yollar bulunur. Program ve veri hafızasına aynı anda erişebildiğinden dolayı hızı yüksektir. Geleneksel mikrodenetleyicilerde program ve veri hafızası aynı veriyolunu kullanır. Bu PICmicro ile karşılaştırıldığında hızı en az 2 kat düşürür.

**Emniyet** tüm komutlar 12 veya 14bit genişliğindeki program hafızasına sığar. Bu yüzden programın VERİ bölgesine atlayıp VERİ' yi program komutları gibi çalıştırması gibi bir ihtimal yoktur. Bu Harvard mimarisinde olmayan 8bit program adres yolları kullanan mikrodenetleyicilerde oluşabilir.

**Komut seti** 16C5X ailesi ve 14bit genişlikteki 16CXX ailesi için program yazabilmek için öğrenmeniz gereken sadece 33 komut vardır. CALL, GOTO ve bit test amaçlı komutlar haricindeki (BTFSS, INCFSS vs.) komutlar tek komut adımında işletilirler.

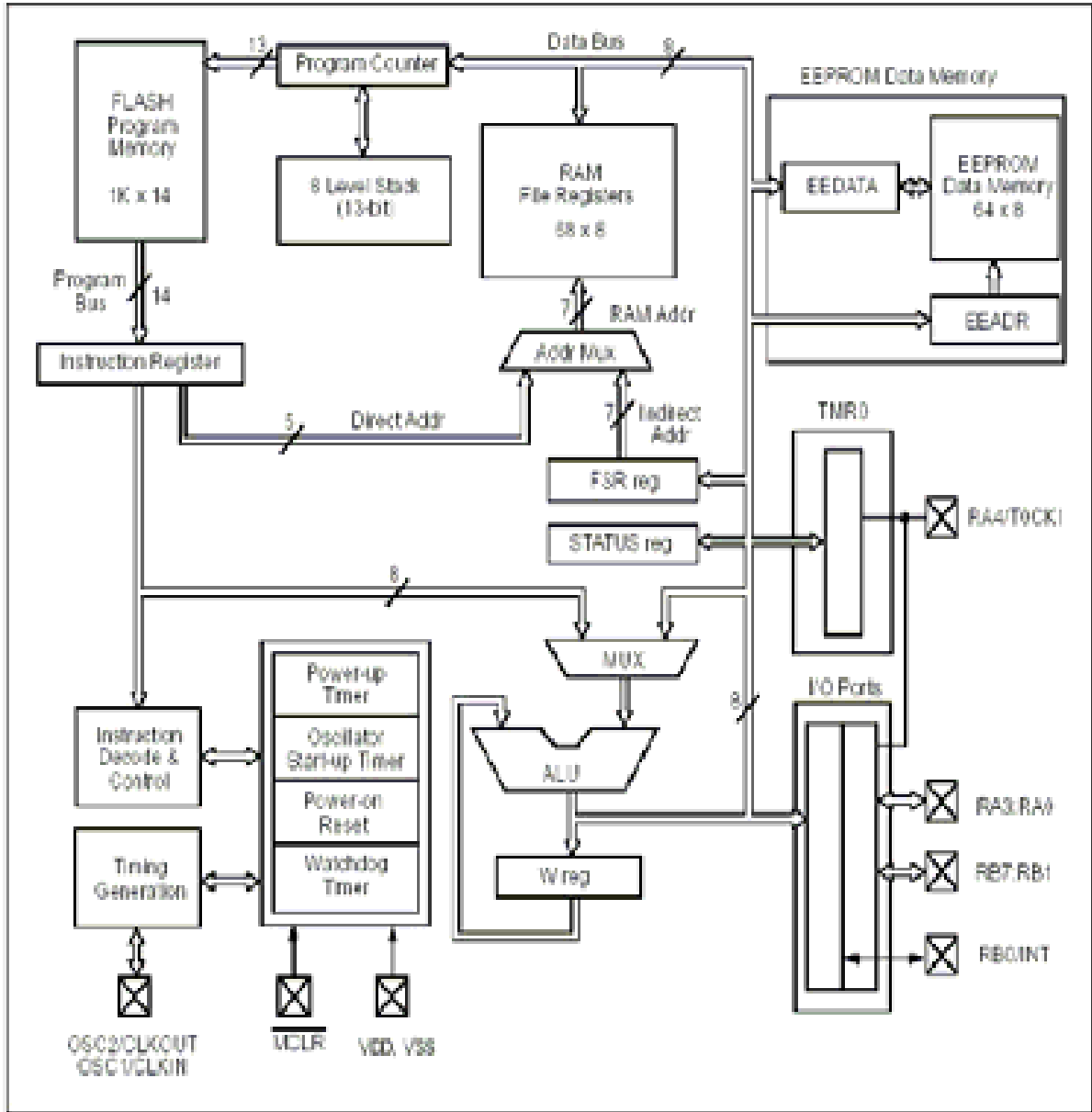
**Hız** PIC osilatör ile dahili saat yolu arasında 4'e bölücü bir devre içerir. Bu özellikle 4Mhz kristal kullanıldığında komut zamanının hesaplanmasını kolaylaştırır. Bu durumda her bir komut adımı (çevrimi) 1uS (yani 1µs) tutar. PIC 20Mhz gibi bir kristal kullanıldığında saniyede 5 milyon komut işletebilen oldukça hızlı bir işlemcidir. – hemen hemen 386SX33'ün iki katı!

**Statik işletim** PIC tamamen statik bir mikrodenetleyicidir; diğer bir deyişle saat frekansını durdurursanız tüm yazmaç içerikleri olduğu gibi kalır. Pratikte tam olarak bunu yapmazsınız, PIC'i uyku moduna geçirirsiniz – bu saat frekansını durdurur ve PIC' in uyku modundan önce hangi durumda olduğunu bilebilmesi için bazı bayrakları ayarlar. Uyku modunda iken PIC 1uA'den az akım çeker.

**Çıkış sürme yeteneği** PIC yüksek çıkış sürme kapasitesine sahiptir ve LED triyak vs. direk sürebilir. Herhangi bir G/Ç pininden 25mA kadar veya tüm çipten 100mA-200mA akım çekilebilir.

**Seçenekler** neredeyse tüm ihtiyaçlarınıza cevap verebilecek şekilde hız, sıcaklık, kılıf, G/Ç hatları, zamanlayıcı fonksiyonları, A/D ve hafıza seçenekleri mevcuttur.

**Çok yönlülük** PIC yüksek adetlerde düşük maliyetli ve çok yönlü bir mikrodenetleyicidir. Özellikle yerin önemli olduğu uygulamalarda birkaç mantık kapısının yerine bile kullanılabilir.



PIC16F84A (14BIT) BLOK DİYAGRAMI

**Güvenlik** PICmicro MCU endüstrideki en güvenli kod koruma özelliklerinden birine sahiptir. Koruma biti birkez programlandığında program hafızası okunamaz.

**Geliştirme** PIC geliştirme için pencereci veya FLASH yapıda, üretim için ise OTP (birkez programlanabilir) yapıda bulunabilir. Geliştirme araçları ev kullanıcıları için kolayca ve uygun fiyatlı olarak temin edilebilir.

## Kodu deneme ve test etme

C'yi kavramak için başlangıç masrafları olarak C derleyicisi, in circuit emülatör ve gerekli donanımı edinmek projenin değerlendirme safhasında insanın gözünü korkutabilir. C derleyicisi diskte verilir ve internetten sağlanabilir.

## C kodlama standartları

Program yazmak ev inşa etmeye benzer – temel sağlam ise işler iyi gider. Aşağıdaki öneriler C++ standartları dokümanından alınmış ve PIC'e adapte edilmiştir.

### İsimler – fonksiyonlarına uygun olmasını sağlayın

İsimler programlamanın kalbidir ve programda kullanılış amacı ile yerine getirdiği işleve uygun olarak verilmelidir. Okunabilirliği arttırmak için büyük küçük harfleri birlikte kullanabilirsiniz.

**HataKontrol, HATAKONTROL'** den daha kolay okunur.

Yine okunabilirliği arttırmak için ön ek olarak bir küçük harf kullanılabilir.

g	Global	<b>gLog;</b>
r	Referans	<b>rStatus();</b>
s	Statik	<b>sValueIn;</b>

### Parantezler { }

Parantezler geleneksel UNIX yöntemiyle;

```
if(koşul){
.....
}
```

veya kolay okunan şu yöntemle kullanılabilir

```
if(koşul)
{
.....
}
```

### Kenar boşlukları

Kenar boşlukları kullanarak yazıları içeriden başlatmak programın okunabilirliğini arttırmak için gereklidir.

### Satır uzunluğu

Monitör ve yazıcı uyumluluğu için satır uzunluğunun 78 karakteri geçmemesine dikkat edin.

### Else If formatı

Herhangi bir koşulu yakalamak için koyacağınız ekstra Else ifadesi önceki if' ler tarafından kapsanmaz.

```
if(koşul)
{
}
else if(koşul)
{
}
else
{
..... /* yukarıdakiler tarafından kapsanmaz */
}
```

### Koşul formatı

Derleyici izin veriyorsa sabit değerleri eşitlik/eşitsizlik karşılaştırmalarında sol tarafa yazın. = işaretinin biri unutulursa derleyici hata bildirir. Ayrıca değer göze batan yerde olmuş olur.

```
if(6 == HataNo)...
```

### Değişkenlere başlangıç değeri verin

Rastgele değer içermemesi için tüm değişkenlere önceden değer verin.

```
int a=6, b=0;
```

### Yorumlar

yorumlar hikayenin öbür yarısıdır. Programınızın bugün nasıl çalıştığını biliyorsunuz peki iki hafta veya iki yıl sonra hatırlayabilecek misiniz? Programınızı başka biri incelese neler olduğunu anlayabilecek mi? İleriki çalışmalarınızı kolaylaştırmak, hataları bulabilmek ve ileride ürününüzü geliştirebilmek için yorumları kullanın.

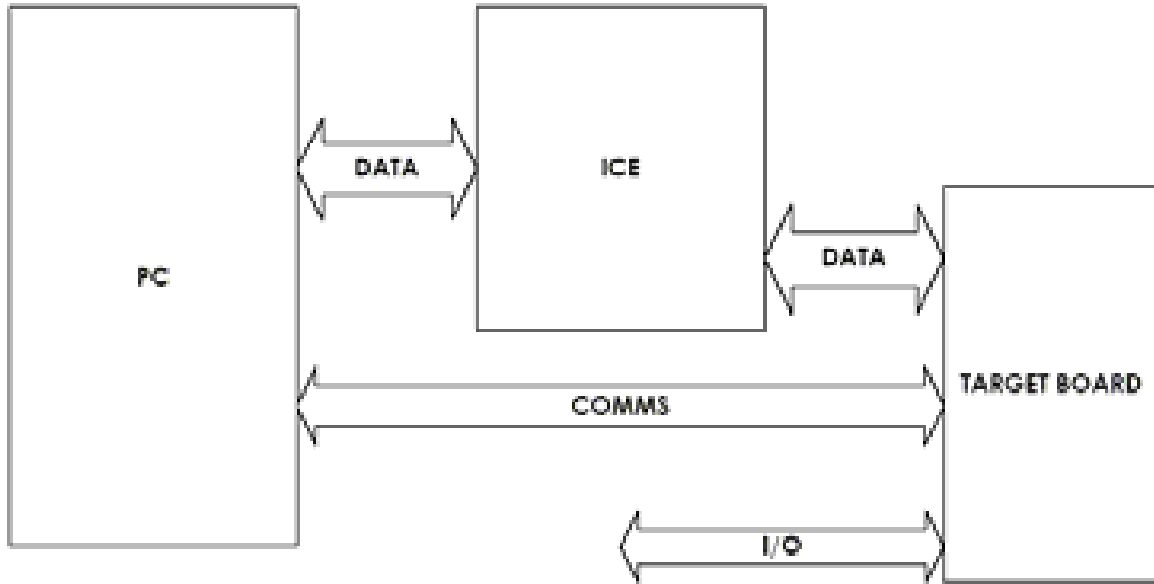
### Temeller

Bütün bilgisayar programları bir başlangıç noktasına sahiptir. Bu başlangıç noktası mikrodenetleyicilerde reset vektörüdür. 14bit PIC16CXX ailesinin reset vektörü 00h, 12bit PIC16C5X ve 12C50X ailesinin reset vektörü ise hafızanın en yüksek adresidir. – 1FFh, 3FFh, 7FFh

Bitiş noktası eğer program sadece bir kez çalışacaksa (örneğin iletişim için hızı ayarlayan bir rutin gibi) programın durduğu yerdir. Trafik ışık kontrolü gibi diğer programlar sürekli başlangıç noktasına dönerler.

Yüksek seviyeli dillerde en yaygın olarak kullanılan ilk programlama örneği ekrana "Merhaba Dünya" yazdırmaktır.

PC ile C kullandığınızda, açıkçası ekran, klavye ve işlemcinin tümü birbirine bağlıdır. Yapmanız gereken programlarla programları ve çevrebirimlerini birbirine bağlamaktır. PICmicro MCU veya diğer mikroişlemci/mikrodenetleyici sistemleri için program geliştirdiğiniz zaman ise programı yazmaktan başka mikroyu dış dünyaya bağlamak için fiziksel donanımı da oluşturmanız gerekir. Buna benzer bir sistem aşağıda görülmektedir.



PIC öğretirken kullandığım basit bir program bir butonla bir ledi yakma programıdır. Basit bir programla başlayın – 2000 satırlık bir program ile değil! Assembly'de şu şekilde:

```

main      btfss      porta, switch      ; switchi test et
          goto      main                ; basılana kadar bekle
          bsf       portb, led          ; ledi yak
lp1       btfsc      porta, switch      ; switchi test et
          goto      lp1                 ; bırakılana kadar bekle
          bcf       portb, led          ; ledi söndür
          goto      main                ; başa dön
  
```

C' de şu şekilde:

```

main()
{
    set_tris_b(0x00);          // portb' yi çıkış yap
    while(true)
    {
        if(input(PIN_A0))     // switchi test et
            output_high(PIN_B0); // kapalıysa ledi yak
        else
            output_low(PIN_B0); // açıksa ledi söndür
    }
}
  
```

derlendiğinde kod şu hale gelir:

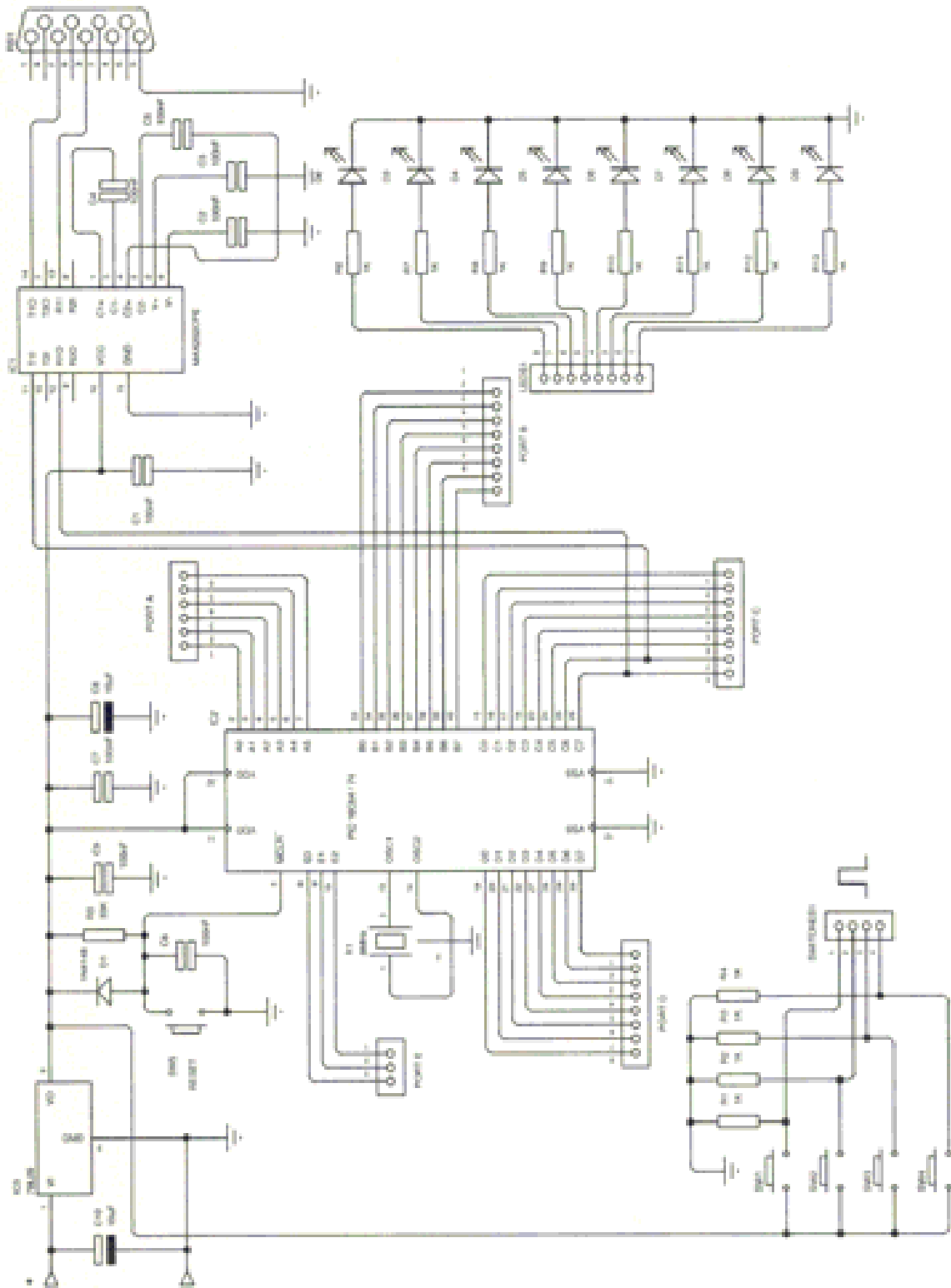
```

main()
{
    set_tris_b(0x00);          0007    MOVLW    00
                              0008    TRIS     6

    while(true)
    {
        if(input(PIN_A0))     0009    BTFSS   05,0
                              000A    GOTO    00D
            output_high(PIN_B0); 000B    BSF     06,0
                              000C    GOTO    00E
        else
            output_low(PIN_B0); 000D    BCF     06,0
                              000E    GOTO    009
    }
}

```

Gördüğünüz gibi derlenmiş hali hafızada assemblyden daha fazla yer kaplar – 14 word C, 9 word Assembly. Fakat programlar büyümeye başladığında C kod kullanımında daha etkili olur.



## 1. C Esasları

Bu bölümde C programlamanın bazı temel yönlerinden bahsedilecektir. Amaç size temel C bilgilerini vermek ve ilerki bölümlerdeki alıştırmaları anlamamanızı sağlamaktır.

Ele alınacak başlıklar şunlardır:

**Program yapısı**  
**Bir C programının bileşenleri**  
**#pragma**  
**main**  
**#include direktifi**  
**printf deyimi**  
**Değişkenler**  
**Sabitler**  
**Yorumlar**  
**Fonksiyonlar**  
**C komutları**

## 1.1 C Programının Yapısı

Tüm C programları önışlemci direktifleri, bildirimler, tanımlamalar, ifadeler, durum ifadeleri ve fonksiyonlar içerir.

### Önişlemci direktifi

Önişlemci direktifi C önişlemcisine gönderilen bir komuttur (program derlenirken ilk adımı oluşturur). En çok kullanılan iki önişlemci direktiflerinden **#define**, özel bir tanımlayıcının yerine geçen bir metin belirtmek için ve **#include** ise program içerisine harici bir dosyadan tanımlama ya da program parçaları dahil etmek için kullanılır.

### Bildirimler

Bildirimlerle programda kullanılan değişkenler, fonksiyonlar ve tiplerin nitelikleri ve isimleri ayarlanır. Global değişkenler fonksiyonların dışında tanımlanır ve tanımlandıkları yerden itibaren dosyanın sonuna kadar geçerlidirler. Local değişkenler ise fonksiyonların içinde tanımlanır ve tanımlandıkları yerden itibaren fonksiyonun sonuna kadar geçerlidirler.

### Tanımlamalar

Tanımlamalarla bir değişken veya fonksiyonun içeriği ayarlanır. Ayrıca değişkenler ve fonksiyonlar için gerekli hafıza ayrılır.

### İfadeler

İfade tek bir değer üreten işleçler ve işlenenler kombinasyonudur.

### Durum ifadeleri

Durum ifadeleri bir C programında program akışının kontrolünü sağlarlar.

## Fonksiyonlar

Fonksiyonlar özel bir işlem yapan bildirimler, tanımlamalar, ifadeler ve durum ifadeleri bütünüdür. Parantezler fonksiyonun gövdesini kapsar. Fonksiyonlar C' de bulunmak zorunda değildir.

## main Fonksiyonu

Tüm C programları **main** isimli bir fonksiyon içermek zorundadır ve program buradan başlar. **main** fonksiyonunu kapsayan parantezler programın başlangıç ve bitiş noktalarını belirler.

Örnek: Genel C program yapısı

```
#include <stdio.h>      /* önişlemci direktifi */
#define PI 3.142        /* standart C başlık dosyasını dahil eder */
float area;            /* global bildirim */
int square(int r);     /* prototip bildirimi */

main()
{
    /* main fonksiyonunun başlangıcı */
    int radius_squared; /* local bildirim */
    int radius = 3;     /* bildirim ve ayarlanması */
    radius_squared = square (radius);
                        /* fonksiyona değer gönderme */
    area = PI * radius_squared;
                        /* ilişkilendirme ifadesi */
    printf("Area is %6.4f square units\n",area);
}
                        /* main fonksiyonu ve program sonu */

square(int r)          /* fonksiyon başı */
{
    int r_squared;     /* sadece square fonksiyonu tarafından */
                        /* tanınan bildirimler */
    r_squared = r * r;
    return(r_squared); /* çağırılan ifadeye değer döndürme */
}
```

## 1.2 Bir C programının bileşenleri

Tüm C programları ifadeler ve fonksiyonlar gibi gerekli bileşenler içerir. İfadeler aslında programın işlemleri gerçekleştiren kısımlarıdır. Bütün C programları bir veya daha fazla fonksiyon içerir. Fonksiyonlar bir veya daha fazla ifadeden oluşan altprogramlardır ve programın diğer kısımları tarafından çağırılırlar.

Programları yazarken tablolar, boş satırlar ve açıklamalar okunurluluğu artırır – sadece ileriki tarihlerde sizin için değil, aynı zamanda programınızı inceleyen başkaları içinde. Aşağıdaki örnek bir C programının gerekli bileşenlerini göstermektedir.

```
#include <stdio.h>
/* İlk C programım */
main()
{
```

```
printf("Merhaba Dünya");  
}
```

`#include <stdio.h>` ifadesi derleyiciye 'stdio.h' dosyasının programa dahil edilmesini bildirir.

.h uzantısı başlık dosyasını belirtir. Başlık dosyası programda kullanılan standart fonksiyonlar hakkında bilgi içerir. Standart giriş çıkış başlık dosyası `stdio.h` adıyla anılır ve en çok kullanılan giriş çıkış fonksiyonlarını içerir. Sadece programda standart giriş çıkışlarla ilgili bölümler olduğunda kullanılması gerekir.

`/* ilk C programım */` ifadesi C için bir açıklama satırıdır. Geleneksel olarak açıklamalar `/*` ve `*/` karakterleri arasında bulunur. Yeni stil açıklamalar ise `//` karakterleri ile başlar ve satır sonuna kadar açıklama kabul edilir. Açıklamalar derleyici tarafından dikkate alınmaz ve bu nedenle derlenmiş kodun boyutu ve hızı üzerinde bir etkisi yoktur.

Bütün C programlarında `main()` fonksiyonu bulunmak zorundadır. Bu programa giriş noktasıdır. Bütün fonksiyonlar aynı formata sahiptir.

```
FunctionName()  
{  
    code  
}
```

Fonksiyon içerisindeki ifadeler sırayla işlenirler.

Süslü parantezler { ve } C'de kod bloklarının başlangıcını ve bitişini gösterirler.

Son olarak `printf("Merhaba Dünya");` ifadesi tipik bir C ifadesidir. Hemen hemen tüm C ifadeleri noktalı virgül (;) ile biter. Satır sonu karakteri (CR+LF) C tarafından satır sonlandırıcı olarak tanınmaz. Bu sebepten dolayı ifadelerin bir satıra sığması gibi bir zorunluluk yoktur ve bir satırda birkaç ifade bulunabilir.

Bütün ifadelerin sonunda, derleyiciye ifadenin sonuna geldiğini bildiren noktalı virgül (;) bulunur ve aynı zamanda ifadeyi diğer ifadeden ayırır. Noktalı virgül unutulursa bir sonraki satırda hata oluşur. `if` ifadesi bileşik bir ifadedir ve noktalı virgölün bileşik ifadelerin sonuna konması gerekir.

```
if (BuDoğruysa)  
    BunuYap();
```

### 1.3 #pragma

Pragma komutu derleyiciye derleme zamanında özel bir işlem yapmasını bildirir mesela hangi PICmicro MCU' nun kullanıldığını bildirmek gibi

```
#pragma device PIC16C54
```

CCS C' de pragma komutu zorunlu değildir ve aşağıdaki yazımda kabul edilir

```
#device PIC16C54
```

## 1.4 main()

Her programda sadece birkez görünen bir `main` fonksiyonu olmak zorundadır. () parantezler arasında parametreler bulunamaz. `void` sözcüğü parantezler içerisinde parametre olmadığını göstermek için kullanılabilir. `main` fonksiyon olarak sınıflandırılmıştır ve ardından gelen komutlar {} parantezleri arasında bulunmak zorundadır.

```
main
{
    komutlar
}
```

## 1.5 #include

Her programda sadece birtane `main` fonksiyonu olmak zorundadır.

Başlık dosyası, (\*.h uzantısıyla belirtilen) fonksiyonların aldığı argümanlar, fonksiyonların döndürdükleri değerler ve belirli bir PIC modelinin kaydedicilerinin adresleri gibi bilgileri barındırır.

```
#include <16C54.H>
```

Bu bilgi derleyici tarafından bütün donanımla ilgili tanımların ve kaynak programların bağlanması sırasında kullanılır.

```
#include <16c71.h>
#include <ctype.h>
#use rs232 (baud=9600, xmit=PIN_B0, rcv=PIN_B1)
main()
{
    printf("Enter characters:");
    while(TRUE)
        putc(toupper(getc()));
}
```

**PIN\_B0** ve **PIN\_B1** tanımlamaları 16C71.H adlı başlık dosyasında bulunur. **toupper** fonksiyonu da **CTYPE.H** başlık dosyasında belirtilmiştir. Her iki başlık dosyası da derleyicinin kullanılan fonksiyonlar hakkında bilgi sahibi olması için kullanılmalıdır. Ayrıca birçok C derleyicisi **printf** ve **putc** gibi G/Ç fonksiyonları için başlık dosyalarını kullanımına ihtiyaç duyar. Bu fonksiyonlar programlara **#use rs232** direktifi ile eklenirler ve harici bir başlık dosyasına ihtiyaç duymazlar.

```
#include <dosya1.h>
```

önişlemciye dosya1 adlı başlık dosyasının başlık dosyaları klasöründe bulunduğunu belirtir. Çift tırnak içerisindeki başlık dosyası ismi ise derleyiciye kaynak kodun bulunduğu klasöre bakmasını söyler.

```
#include "dosya2.h"
```

ilk olarak o anda kullanılan klasöre bakmasını belirtir. Dikkat ettiyseniz **#include** direktifinden sonra noktalı virgül konulmuyor. Bunun nedeni **#include** direktifinin bir C ifadesi değil önışlemci direktifi olmasıdır. Bütün başlık dosyasının içeriği derlenme zamanında kaynak dosyasına eklenir.

## 1.6 printf Fonksiyonu

**printf** fonksiyonu yazılabilir bilginin gönderilmesini sağlayan ve standart kütüphanede bulunan bir fonksiyondur. **printf()** için genel biçim şu şekildedir:

```
printf("kontrol karakterleri dizisi", argument_list);
```

Kontrol karakterleri dizisi çift tırnak içinde bulunan bir karakter dizisidir. Bu dizi içinde harf, rakam ve sembol kombinasyonları bulunabilir. Özel semboller % işareti ile birlikte kullanılırlar. Bir printf() fonksiyonu içinde mutlaka bir kontrol karakteri dizisi olmalıdır. Eğer biçim belirteçleri kullanılmadıysa argüman listesi bulunmayabilir. Argüman listesi sabitler ve değişkenlerden oluşabilir. Aşağıda **printf()** fonksiyonunun sabit ve değişkenlerle kullanımına dair iki örnek bulunmaktadır:

```
printf("Merhaba Dünya");  
printf("Microchip@ #d numara!", 1);
```

Biçim belirteci (**%d**) gösterilecek olan verinin tipine bağlıdır. Aşağıda C'de kullanılan bütün biçim belirteçleri ve ilgili veri tipleri gösterilmiştir:

### printf() Biçim Belirteçleri

```
%c    tek bir karakter character  
%d    İşaretli Desimal Tamsayı  
%f    Kayar noktalı (ondalık gösterim)  
%e    Kayar noktalı (Üstel veya bilimsel gösterim)  
%u    işaretli Desimal tamsayı  
%x    işaretli hexadesimal tamsayı (küçük harf)  
%X    işaretli hexadesimal tamsayı (Büyük harf)  
/ öneki uzun tamsayıları belirtmek için %d, %u, %x ile birlikte kullanılır
```

**NOT:** % karakterinden sonra kullanılan 0(sıfır)'ların sayısı, sayı dizilerinde kaç adet sıfırın gösterileceğini belirler. 0'ları takip eden sayı da, kaç hanenin gösterileceğini belirler.

```
printf(" 12'nin onaltılık karşılığı %02x\n", 12);
```

Bu program satırının çıktısı: **12'nin onaltılık (hexadesimal) karşılığı olan 0c dir.**

### Ters bölü karakter sabitleri(Escape Sequences):

```
\n    yenisatır
```

```
\t    yatay tab
\r    carriage return
\f    formfeed
\'    tek tırnak
\"    çift tırnak
\\    backslash
%%    yüzde işareti
\?    Soru işareti
\b    backspace
\0    boş(null) karakter
\v    dikey tab
\xhhh insert HEX code hhh
```

Format belirteçleri %[bayraklar][genişlik][.hassasiyet], şeklinde de kullanılabilir.

```
printf("Alan %6.4f birim karedir\n", alan);
```

**alan** değişkeni 6 haneli ve 4 ondalık hane kullanılarak yazılacaktır.

"Varsayılan" olarak printf'in çıktısı en son tanımlanmış RS232 portuna gönderilir. Çıktılar herhangi bir fonksiyon yoluyla başka bir yere de yönlendirilebilir.

Örneğin:

```
void lcd_putc(char c)
{
// Insert code to output one
// character to the LCD here
}
printf(lcd_putc, "value is %u", value);
```

## 1.7 Değişkenler

Değişken, bellekteki belirli bir konuma verilen isimdir. Bu bellek bölgesi değişkenin bildirimine göre değişik değerleri tutabilir. C dilinde bütün değişkenlerin kullanılmadan önce bildirimleri yapılmış olmalıdır. Değişken bildirimleri derleyiciye ne tür bir değişken kullanıldığını bildirir. Bütün değişken bildirimlerinin sonunda noktalı virgül (;) bulunmalıdır. Temel C veri tipleri **char**, **int**, **float**, ve **long**'dur. Değişken bildirimlerinin genel biçimi şu şekildedir:

**tip degisken\_ismi;**

Değişken bildirimine örnek olarak **char kar;** örneğini verebiliriz. Burada kar değişkeni karakter olarak tanımlanıyor (8 bit işaretsiz tamsayı)

## 1.8 Sabitler

Sabit, program tarafından değiştirilemeyen değer demektir. Örneğin 25 sayısı bir sabittir. -100 ve 40 gibi tamsayı sabitler olabileceği gibi 456.75 gibi kayar

noktalı sabitler de olabilir. Karakter sabitler tek tırnak içerisinde gösterilirler: 'A' veya '&' gibi.

Derleyici programınızda bir sabit ile karşılaştığında bu sabitin türü hakkında karar vermesi gerekir. C derleyicisi varsayılan olarak verinin yerleştirilebileceği en ufak veri tipini kullanacaktır. Örneğin 15 sayısı bir **int** , 64000 işaretli long tiptir.

Sabitler ayrıca **#define** ifadesi ile de belirtilebilir:

```
#define etiket deęer
```

Derleyici programı derlerken **etiket** kısmında yazılı olan isimle her karşılaştığında bunun yerine belirtilen deęeri koyacaktır.

```
#define TRUE 1
#define pi 3.14159265359
```

C sabitleri 16lı veya 8lik tabanda yazmanıza olanak verir: Onaltılık tabandaki sabitler '0x' öneki ile başlamalıdır. Örneğin 0xA4 geçerli bir onaltılık sabittir. Sayısal sabitlerin yanında C dizge sabitlerini de destekler. Dizge sabitleri çift tırnak arasında yazılan karakter takımlarıdır.

**#define** ifadesi ile tanımlanan sabitler aslında derlenmeden önceki aşamada yapılan yazınsal deęişimlerdir. # işareti ile başlayan satırlar ön-işlemci direktifleridir.

**#define sözcüğüyle** herhangi bir metni tanımlayabilirsiniz. Örneğin:

```
#define YASLI_DEGIL (YAS<65)
.
.
.
.
if YASLI_DEGIL
printf("GENC");
```

**#define** verisi bellekte saklanmaz. Derlenme anında işlenir.

Sabitleri **ROM**'a kaydetmek için **const** sözcüğünü kullanmalısınız. Örneğin:

```
char const id[5]={"1234"};
```

Buradaki diziyi bellekte saklamak için 5 adet bellek bölgesi kullanılır. Çünkü dizinin sonuna boşluk karakteri (\0) eklenmektedir.

## 1.9 Yorumlar

Yorumlar kaynak kodda yapılan işlemleri anlatmak ve çalışmasını göstermek için kullanılır. Yorumlar derleyici tarafından göz ardı edilir. Yorumlar bir C anahtar kelimesinin, fonksiyon veya değişken isminin ortası dışında herhangi bir yerde bulunabilirler. Yorumlar birçok satırdan oluşabilirler ve bazen kodların geçici olarak kaldırılması için de kullanılabilirler. Yorumlar satırlarca uzunlukta olabilirler ve bazen de bir kod satırının geçici olarak kaldırılması için kullanılabilirler. Yorumlar iç içe gömülemezler. Yorumların iki tür kullanım biçimleri vardır. İlk biçim bütün C derleyicileri tarafından desteklenmektedir.

```
/* Bu bir yorumdur*/
```

İkinci format ise birçok derleyici tarafından desteklenmektedir.

```
// Bu bir yorumdur
```

**Alıştırma:** Aşağıdaki yorum biçimlerinden hangileri geçerli, hangileri geçersizdir?

```
/* Kısa bir yorum */
```

```
/* Çok çok çok çok çok çok çok çok çok çok çok çok çok çok çok çok uzun bir yorum */
```

```
/* Bu yorum /* geçerli */ değildir */
```

## 1.10 Fonksiyonlar

Fonksiyonlar C programlarının temel yapıtaşlarıdır. Her bir C programı en az bir fonksiyon, **main()**, barındırır. Yazdığınız birçok program birden fazla fonksiyon bulunduracaktır. Birçok fonksiyonu barındıran bir C programının genel yapısı şu şekildedir:

```
main()
{
    fonksiyon1()
    {
    }
    fonksiyon2()
    {
    }
}
```

**main()** program çalıştırıldığında çağırılan ilk fonksiyondur. Diğer **fonksiyon1()** ve **fonksiyon2()** programın herhangi bir yerinde çağırılabilirler.

Geleneksel olarak **main()** fonksiyonu başka bir fonksiyon tarafından çağırılmaz, ancak C'de kısıtlamalar yoktur.

Aşağıda iki C fonksiyonu görülmektedir:

```
main()
{
```

```

printf("C'yi");
fonksiyon1();
printf("'cok");
}
fonksiyon1()
{
printf(" seviyorum ");
}

```

Bir fonksiyonda en son paranteze ( } ) gelince program tekrar fonksiyonun çağırılmış olduğu satırdan itibaren işletilmeye devam eder. Bölüm 3.1'e bakınız.

## 1.11 Makrolar

**#define** önceki bölümlerde de görüldüğü gibi güçlü bir direktiftir. C define direktifinin parametre almasını da sağlayarak daha da güçlü hale getirir. Parametrelerle birlikte kullanıldığında bunlara makro denilir. Makrolar programların okunabilirliğini artırmak veya yazma kolaylığı sağlamak için kullanılırlar. Basit bir makro örneği:

```

#define var(x,v) unsigned int x=v;
var(a,1);
var(b,2);
var(c,3);

```

aşağıdakine eşdeğerdir:

```

unsigned int a=1;
unsigned int b=2;
unsigned int c=3;

```

Aşağıdaki örnek de kısa bir fonksiyon tanımlanmaktadır:

```

#define MAX(A,B) (A>B)?A:B
z = MAX(x,y); // z ,x ve y değerlerinin en büyüğü olacaktır

```

## 1.12 Durumsal Derleme

C'de bazı derleme esnasında kodun belirli bölümlerinin derlemeye dahil edilmesi veya çıkarılması için ön-işlemci direktifleri bulunur. Aşağıdaki örneği inceleyelim:

```

#define HW_VERSION 5
#if HW_VERSION>3
output_high(PIN_B0);
#else
output_low (PIN_B0);
#endif

```

Yukarıdaki programda HW\_VERSION değerine bağlı olarak sadece bir satır derlenecektir. Aynı kodda onlarca **#if** ifadesi bulunabilir ve aynı kod değişik

donanımlar için derlenebilir. **#if** ifadeleri, normal **if** ifadelerinden farklı olarak kod derlenirken değerlendirilirler. **#ifdef** ifadesi herhangi bir kimlik ifadesinin daha önce tanımlanıp tanımlanmadığını belirlemede kullanılır.

Örnek:

```
#define TANIM
#ifdef TANIM
printf("X Fonksiyonu çalıştırılıyor");
#endif
```

Bu örnekte **#define TANIM** satırı kaldırıldığında printf fonksiyonunun bulunduğu satır derleyici tarafından göz ardı edilecektir (yani derlenmeyecektir).

## 1.13 Donanım Uyumluluğu

Derleyici kodun düzgün derlenebilmesi için donanım hakkında bilgiye ihtiyaç duyar. Tipik bir program şu şekilde başlar:

```
#include <16c74.h>
#fuses hs,nowdt
#use delay(clock=800000)
```

İlk satır kullanılan işlemcinin pinlerine ait define direktiflerini içeren dosyanın programa eklenmesini sağlar. İkinci satırda PICmicro@MCU'nun ayar değerleri atanır. Bu örnekte yüksek hızlı osilatör kullanılmakta ve watchdog timer devre dışı bırakılmaktadır. Son satır da osilatörün hızı belirtilmektedir. Aşağıda başka bir örnek görülmektedir:

```
#use rs232(buad=9600,xmit=PIN_C6,rcv=PIN_C7)
#use i2c(master,scl=PIN_B6,sda=PIN_B7)
```

Bu kitaptaki örneklerde yukarıda görülen donanım tanımlayan satırlar gösterilmemiştir.

Buna ilave olarak, C değişkenleri tanımlanıp donanım kontrol kayıtlarına (registers) yönlendirilebilirler. Bu değişkenler bayt veya bitler şeklinde olabilirler. Tanımlandıktan sonra programda diğer normal değişkenler gibi kullanılabilirler. Örnek olarak

```
#bit carry=3.0 ; STATUS registerindeki carry flagi değişken tanımlandı
#byte portb=6 ; PortB doğrudan değişkenmiş gibi tanımlandı
#byte intcon=11; INTCON kayıtlı değişken olarak tanımlandı
```

## 1.14 C Anahtar Kelimeleri

ANSI C standardı 32 adet C anahtar kelimesi tanımlar. C'de belirli kelimeler derleyici tarafından veri tiplerinin tanımlanmasında veya döngülerde kullanılırlar. Bütün C anahtar kelimeleri küçük harfle yazılmalıdır. Genellikle, C derleyicileri işlemcinin özelliklerinden faydalanabilmek için fazladan anahtar

kelimeler tanımlarlar. Aşağıda C anahtar kelimelerinin listesi bulunmaktadır. Kullanılan değişkenlere verilecek isimler anahtar kelimelerle aynı olamaz.

Auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

### **Alıştırmalar:**

1. Adınızı ekrana yazdıran programı yazınız.
2. Yıl adlı bir tamsayı değişkeni tanımlayan bir program yazınız. Ardından printf() komutuyla ekrana yılı şu şekilde yazdıran bir program yazınız. Programınızın çıktısı şu şekilde olmalıdır. **2006 yılındayız.**

## **2. Değişkenler**

C Dilinin önemli bir yanı dilin değişkenleri nasıl sakladığı ile ilgilidir. Bu bölümde, verilerin hafızada tutulması için C'de değişkenlerin nasıl kullanılacağı yakından incelenecektir.

Bu bölümde tartışılacak olan başlıklar şunlardır:

**veri tipleri**  
**bildirimler**  
**atamalar**  
**veri tipi aralıkları**  
**tip dönüşümleri**

## 2.1 Veri Tipleri

C Dili beş temel veri tipini ve 4 tür dönüştürücüyü destekler. Aşağıdaki tablo temel veri tiplerinin ve tip dönüştürücülerinin anlamlarını göstermektedir.

<u>Tip</u>	<u>Anlamı</u>	<u>Anahtar Kelimeler</u>
Karakter karakter verisi		<b>char</b>
Tam sayılar	işaretsiz tam sayılar	<b>int</b>
<b>Float</b>	kayan noktalı sayılar	<b>float</b>
<b>Double</b>	çift duyarlılıkta kayan noktalı sayılar	<b>double</b>
Boş	değersiz	<b>void</b>
İşaretili	pozitif veya negatif sayılar	<b>signed</b>
İşaretsiz	sadece pozitif sayılar	<b>unsigned</b>
Uzun	geniş aralıktaki sayılar	<b>long</b>
Kısa	dar aralıktaki sayılar	<b>short</b>

Her bir veri tipi kullanılan dönüştürücüye göre, belirli bir aralıktaki sayıları temsil eder. Sonraki tablo temel veri tiplerinin mümkün olan kombinasyonları için geçerli aralıkları göstermektedir.

<u>Tip</u>	<u>Bit Genişliği</u>	<u>Aralık</u>
<b>short</b>	1	0 veya 1
<b>short int</b>	1	0 veya 1
<b>int</b>	8	0..255
<b>char</b>	8	0..255
<b>unsigned</b>	8	0..255
<b>unsigned int</b>	8	0..255
<b>signed</b>	8	-128..127
<b>signed int</b>	8	-128..127
<b>long</b>	16	0..65536
<b>long int</b>	16	0..65536
<b>signed long</b>	16	-32768..32767
<b>float</b>	32	3.4E-38..3.4E+38

**NOT:** Asıl veri tipleri ve nümerik aralıkları için kullandığınız C derleyicisinin kılavuzlarına bakınız.

C, **unsigned int**, **short int** ve **long int** veri tiplerinin kısa gösterimine olanak verir. **int** kullanmaksızın kolayca, **unsigned**, **short**, ya da **long** şeklinde yazılabilir. Aritmetik işlemlerin işlemci için daha kolay olması bakımından, C'deki tüm negatif sayılar 2'nin tümleyeni biçiminde gösterilir. Bir sayının 2'nin tümleyeni eşdeğerini bulmak için önce sayının tüm bitleri evrilir ve sayıya 1 eklenir.

Örneğin, +29'u 2'nin tümleyini eşdeğerine çevirmek için:

```
00011101 = 29
11100010 tüm bitleri evir
      1    1 ekle
11100011 = -29
```

Aşağıda 12000 değerinin a değişkenine atanması örneği verilmiştir. 12000'in 16'lık sayı sistemindeki karşılığı 2EE0'dır. Verilen kod, sayının düşük değerli kısmını 11h yazmacına, yüksek değerli kısmını 12h yazmacına atar.

```
long a = 12000;
main()
{
    0007: MOVLW E0
    0008: MOVWF 11
    0009: MOVLW 2E
    000A: MOVWF 12
}
```

### ALİŞTIRMALAR:

1. Verilen ifadeyi kısa gösterimde yazınız.

### long int i;

2. İşaretili ve işaretsiz sayılar arasındaki farkı anlamak için, aşağıdaki programı yazınız. İşaretsiz tam sayı 35000, işaretili sayı biçiminde -30536 olarak gösterilecektir.

```
main()
{
    int i;                /* işaretili tam sayı */
    unsigned int u;      /* işaretsiz tam sayı */
    u = 35000;
    i =u;
    printf("%d %u\n", i, u);
}
```

## 2.2 Değişken Bildirimleri

Değişkenler iki temel yerde bildirilebilirler: fonksiyonun içinde ya da fonksiyonların dışında. Bunlar, sırasıyla yerel ve global isimlerini alırlar. Değişkenler aşağıdaki biçimde bildirilirler:

**tip değişken\_adi;**

Burada, **tip** geçerli bir C veri tipi ve **degisken\_adi** da bildirilen değişkenin ismidir.

Yerel değişkenler (fonksiyonların içinde bildirilenler) sadece bildirildikleri fonksiyonların içindeki ifadelerde kullanılabilir.

Yerel değişkenin değerine, bildirildiği fonksiyon dışındaki fonksiyonlar tarafından erişilemez. Yerel değişkenler hakkında hatırlanması gereken en önemli şey, onların fonksiyona giriş esnasında yaratılıp, fonksiyondan çıktığı zaman yok edilmesidir. Ayrıca yerel değişkenler, fonksiyonun başlangıcında diğer ifadelerden önce bildirilmelidir.

Farklı fonksiyonlarda, aynı isimli yerel değişkenlerin bildirilmesi geçerli bir bildirimdir. Aşağıdaki örneği göz önünde tutacak olursak:

```
void f2(void)
{
    int sayac;
    for (sayac = 0 ; sayac < 10 ; sayac++)
        print ("%d \n", sayac);
}

f1()
{
    int sayac;
    for (sayac=0; sayac<10; sayac++)
        f2();
}

main()
{
    f1();
    return 0;
}
```

Bu program 0'dan 9'a kadar olan sayıları ekrana 10 defa yazdıracaktır. Programın işleyişi her iki fonksiyonda da yer alan **sayac** değişkeninden etkilenmez.

Diğer taraftan global değişkenler bir çok farklı fonksiyon tarafından kullanılabilir. Global değişkenler herhangi bir fonksiyon tarafından çağrılmadan önce bildirilmelidirler. Daha önemlisi, global değişkenler programın işleyişi tamamlanmadıkça, çağrıldıkları fonksiyondan çıktıkları zaman yok edilmezler.

Aşağıdaki örnek global değişkenlerin nasıl kullanılacağını göstermektedir.

```
int max;
f1()
{
    int i;
    for(i=0; i<max;i++)
        printf("%d ", i);
}
```

```

main()
{
    max=10;
    f1();
    return 0;
}

```

Bu örnekte; main() ve f1() fonksiyonu max değişkenine referans göstermektedir. **main()** fonksiyonu **max** değişkenine değer atamakta ve **f1()** fonksiyonu for döngüsünü kontrol etmek için **max**'ın bu değerini kullanmaktadır.

### ALİŞTIRMALAR:

1. Yerel ve global değişkenler arasındaki temel farklılıklar nelerdir?
2. C'de yerel ve global değişkenler aynı ismi paylaşabilirler. Aşağıdaki programı yazınız.

```

int sayac;

f1()
{
    int sayac;
    sayac =100;
    printf("f1() deki sayac : %d\n", sayac);
}

main()
{
    sayac=10;
    f1();
    printf("main() deki sayac: %d\n", sayac);
    return 0;
}

```

**main()**'in içerisinde **sayac**'a referans global değişken biçimindedir. **f1()** fonksiyonundaki yerel **sayac** değişkeni global değişkenin değerini geçersiz kılar.

## 2.3 Değişken Ataması

Şu ana kadar değişkenlere ilk değer atamasını es geçip, sadece değişken bildirimlerinin nasıl yapılacağı üzerine tartıştık. Değişkenlere değer atanması aşağıda gösterildiği gibi basitçe gerçekleştirilir:

```

degisken_adi = deger;

```

Değişken ataması bir ifade olduğundan ötürü, ifadenin sonuna noktalı virgül konulur. Örnek olarak 100 değerini **sayac** tam sayı değişkenine atamak için:

```
sayac = 100;
```

Burada 100 deęeri bir sabittir.

Deęişkenler bildirildięi anda bunlara ilk deęer ataması yapılabilir. Bu şekilde deęişkenlerinize bilinen deęerlerle atama yapılması programınızı kolaylaştırıp daha güvenli hale getirir. Örnek olarak:

```
int a = 10, b = 0, c = 0x23;
```

C'de pek çok deęişik tipte sabitler mevcuttur. Bir karakter sabiti 'M' örneğindeki gibi bir karakter sabiti, karakterin **tek tırnak** içinde yazımı ile belirtilir. int tiplere deęer atarken tam sayılar kullanılır. Kayar noktalı sayılar, ondalık noktalı gösterimi kullanılmalıdır. Örneğin 100 sayısının kayar noktalı bir sayı olduğunu C'de göstermek için 100.0 gösterimi kullanılır.

Ayrıca, bir deęişken başka bir deęişkene atanabilir. Aşağıdaki kod parçası bu atamayı gösterir.

```
main()
{
    int i;
    int j;
    i=0;
    j=i;
}
```

### ALİŞTIRMALAR:

1. "**sayac**" isimli tamsayı tipli bir deęişkenin bildirimini yazınız. Bu deęişkene 100 deęerini atayıp, printf() fonksiyonu ile bu deęeri gösteriniz.

Programın çıktısı şu şekilde olmalıdır:

**sayac deęişkeninin deęeri 100'dür.**

2. **char**, **float** ve **double** tiplerinde **ch**, **f**, ve **d** isimli deęişkenlerin bildirimini yapıp, 'R' yi karakter, **50.5**'i float, **156.007**'yi double deęişkenine atayınız. Bu deęişkenlerin deęerlerini ekranda yazdırınız.

Çıktı aşağıdaki şekilde olmalıdır:

```
ch = R
f = 50.5
d = 156.007
```

## 2.4 Numaralandırma

C'de tam sayı sabitlerinin listesini yapmak mümkündür. Bu bildirim numaralandırma ismini alır. Numaralandırılarak oluşturulan sabitler listesi, tam sayı tiplerinin kullanıldığı her yerde kullanılabilir. Numaralandırma oluşturmanın genel şekli şöyledir:

```
enum isim {numaralandırma listesi} değişken(ler);
```

Değişken listesi numaralandırmanın seçime bağlı parçasıdır. Numaralandırma değişkenleri sadece numaralandırma listesinde tanımlanmış değerleri kullanabilir. Örneğin şu ifadede:

```
enum renk_tipi {kirmizi, yesil, sari} renk;
```

**renk** değişkenine sadece **kirmizi**, **yesil** ve **sari** değerleri atanabilir. (ör., **renk = kirmizi**);

Derleyici, ilk girdi 0'a eşit olmak üzere numaralandırma listesine tam sayı değerleri atar. Her bir girdi bir öncekinden bir büyüktür. Buna göre, yukarıdaki örnekte **kirmizi** 0, **yesil** 1 ve **sari** 2 değerlerini alır. Bu durum, sabit için bir değer atanarak önlenebilir. Aşağıda bu durum örneklendirilmiştir:

```
enum renk_tipi {kirmizi, yesil=9, sari} renk;
```

Bu ifade **kirmizi**'ya **0**, **yesil**'e **9** ve **sari**'ya **10** değerlerini atar.

Numaralandırma bir kez tanımlandığında, bu isim, programın farklı noktalarında ilave değişkenler yaratmak için kullanılabilir. Örneğin, **rengim** değişkeni **renk\_tipi** numaralandırması ile şu şekilde yaratılır:

```
enum renk_tipi rengim;
```

Değişken başka bir değişkenle sınırlanabilir:

```
if (renk==meyve)
// bir şeyler yap
```

Aslında, numaralandırma belgelendirmeye yardımcı olur. Bir değişkene değer atamak yerine, değerinin daha açık anlaşılması için numaralandırma kullanılabilir.

### ALIŞTIRMALAR:

1. PIC17CXX ailesinin numaralandırılmasını oluşturunuz.
2. En düşükten en yüksek değere paranızın numaralandırma listesini oluşturunuz.
3. Aşağıdaki kod parçası doğru mudur? Açıklayınız.

```
enum {PIC16C51,PIC16C52,PIC16C53} aygit;
aygit = PIC16C52;
printf("İlk PIC %s `d1.\n", aygit);
```

## 2.5 typedef

C'de yeni veri tipleri yaratmak için typedef ifadesi kullanılır. Bu aşağıdaki biçimde gerçekleştirilir:

**typedef eski\_isim yeni\_isim;**

Yeni isim değişkenleri bildirmek amacıyla kullanılabilir. Örneğin aşağıdaki program **signed char** tipi yerine **kucuktamsayi** ismini kullanır.

```
typedef signed char kucuktamsayi;
main()
{
    kucuktamsayi i;
    for (i=0; i<10; i++)
        printf("%d ", i);
}
```

**typedef** kullanırken iki noktaya dikkat etmelisiniz: **typedef** orijinal tipi veya ismin etkinliğini ortadan kaldırmaz yani; önceki örnekte **signed char** hala geçerli bir tiptir, aynı orijinal tipte değişik yeni isimler oluşturmak için birkaç **typedef** ifadesi kullanılabilir.

**typedef** genellikle iki sebepten ötürü kullanılır. Öncelikli olarak taşınabilir programlar oluşturmak amacıyla kullanılır. Eğer yazdığınız program 16-bit ve 32-bit'lik tam sayı tabanlı makinelerde kullanılacaksa, 16-bitlik tam sayıların kullanılacağından emin olmak isteyebilirsiniz. 16-bitlik makineler için oluşturulan programda tüm tam sayıları 16-bitlik olarak bildirmek için şu ifade kullanılırdı.

```
typedef int tamsayim;
```

Programı 32-bitlik bilgisayarlar için derlemeden önce typedef ifadesi şu şekilde değiştirilebilirdi:

```
typedef short int tamsayim;
```

Yani **tamsayim** tipinde bildirilmiş tüm değişkenler 16-bit genişliğe sahip olur.

**typedef** ifadelerini kullanmanın diğer nedeni, programın okunabilirliğini arttırmasındandır. Eğer kodunuz, sayaç benzeri pek çok değişken içeriyorsa, tüm **sayac** değişkenlerinizi bildirmek için typedef ifadesini kullanabilirsiniz.

**typedef int sayac;**

Kodunuzu okuyan birisi, **sayac** tipinde bildirmiş olduğunuz değişkenlerinizin sayıcı amaçlı kullandığınızı fark edecektir.

## ALIŞTIRMALAR:

1. **unsigned long tipi** için UL adında yeni bir isim oluşturunuz. Bu oluşturduğunuz yeni ismi kullanıp, buna bir değer ataması gerçekleştiriniz ve bu değeri ekrana yazdırınız.

2. Aşağıdaki kod parçası geçerli midir?

```
typedef int yukseklik;  
typedef yukseklik uzunluk;  
typedef uzunluk derinlik;  
derinlik d;
```

## 2.6 Tip Dönüşümleri

C, farklı veri tiplerinin bir ifadede karıştırılmasına izin verir. Örneğin verilen kod parçası geçerli bir yazımdır:

```
char ch = '0';  
int i = 15;  
float f = 25.6;  
double sonuc = ch*i/f;
```

Veri tiplerinin karıştırılması, derleyicinin farklılıkları nasıl çözümleneceğini bildiren sıkı dönüşüm kuralları ile yönetilir. Kurallar kümesinin ilk kısmı tip terfisi'dir. C derleyicisi, ifade çalıştırıldığında **char** veya **short int** tipini otomatik olarak **int** tipine dönüştürür. Tip terfisi sadece ifadenin çalıştırıldığı zaman geçerlidir; değişken fiziksel olarak daha büyük olamaz.

Otomatik tip dönüşümleri tamamladığı zaman, C derleyicisi tüm değişkenleri en geniş tipli değişkenin tipine çevirir. Bu işlem adım adım ilerleme temeline göre gerçekleştirilir. Aşağıdaki algoritma tip dönüşümlerinin nasıl yapılacağını göstermektedir.

**EĞER** bir operand long double ise  
değeri long double tipine çevrilir

**EĞER** bir operand double ise  
değeri double tipine çevrilir

**EĞER** bir operand float ise  
değeri float tipine çevrilir

**EĞER** bir operand unsigned long ise  
değeri unsigned long tipine çevrilir

**EĞER** bir operand long ise  
değeri long tipine çevrilir

**EĞER** bir operand unsigned ise  
değeri unsigned tipine çevrilir

Önceki örnekte ne türden terfiler ve dönüşümler olduğunu inceleyelim. İlk olarak, **ch int** tipine terfi ettirilir. İlk işlem **ch** ile **i**'nin çarpımıdır. Her iki

değişkende artık tam sayı olduklarından, her hangi bir tip dönüşümü gerçekleştirilmez. Sonraki işlem **f**'nin **ch\*i** ifadesine bölümüdür. Algoritmaya göre, eğer operandlardan biri float ise diğeri float tipine dönüştürüleceğinden, **ch\*i/f** ifadesinin sonucu float tipindedir, fakat **sonuc** double olduğu için double tipine dönüştürülecektir.

Derleyicinin tip değişimlerini yapmasına güvenmek yerine, tip dönüşümlerini aşağıda gösterildiği gibi açıkça yapabilirsiniz.

### (tip) deger

Bu (açık) tip dönüştürme adını alır ve değişkende geçici değişiklik yapılmasına neden olur. **tip** geçerli bir C veri tipidir ve **deger** değişken yada sabittir. Aşağıdaki kod parçası kayar noktalı bir sayıdaki tam sayı kısmın nasıl gösterileceğini örnelemektedir.

```
float f;  
f = 100.2;  
printf("%d", (int)f);
```

Bu kod çalıştırdıktan sonra ekrana 100 sayısı yazdırılır.

İki 8-bit tam sayı çarpıldığında sonuç 8-bit uzunlukta olacaktır. Sonuç değeri long tipinde bir değişkene atanacak olursa, aritmetik işlem yeni değişkene atanma işleminden önce gerçekleştirildiğinden sonuç hala 8-bitlik bir tam sayı olacaktır.

```
int a = 250, b = 10;  
long c;  
c = a * b;
```

Sonuç 196 olacaktır. long tipinde sonuca ihtiyacınız varsa, bir değişken için açıkça tip dönüşümü yapmanız gerekmektedir.

```
c = (long) a * b;
```

**a** long tipine açıkça dönüştürüldüğünden long tipinde çarpım gerçekleştirilmiştir ve sonuç beklenildiği gibi 2500 olacaktır.

## 2.7 Değişken Depolama Sınıfı

C'deki her bir değişken ve fonksiyonun iki özelliği vardır: tip ve depolama sınıfı. Tip kavramı daha önce açıklanmıştı. Otomatik, harici, durağan ve yazmaç olmak üzere dört değişik depolama sınıfı vardır. Bunlar C'de aşağıdaki anahtar kelimeler ile gösterilirler:

**auto extern static register**

**Auto**

Fonksiyonun içerisinde bildirilen değişkenler varsayılan olarak auto sınıfındadır.

```
{
char c;
int a, b, e;
}
```

ile aşağıdaki gösterim aynıdır.

```
{
auto char c;
auto int a, b, e;
}
```

Bir kod bloğuna girildiğinde, derleyici bildirilmiş değişkenler için RAM alanını ayırır. RAM alanları bu yerel kod bloğu için kullanılır ve diğer kod blokları tarafından kullanılabilir.

```
main()
{
char c = 0;
int a =1, b = 3, e = 5;
0007: CLRF 0E ; 0Eh yazmacının içeriği C'ye atanır
0008: MOVLW 01 ; w'ye 1 yükle
0009: MOVWF 0F ; a'ya w'deki değeri ata
000A: MOVLW 03 ; w'ye 3 yükle
000B: MOVWF 10 ; b'ye w'deki değeri ata
000C: MOVLW 05 ; w'ye 3 yükle
000D: MOVWF 11 ; c'ye w'deki değeri ata
}
```

## Extern

**extern** anahtar kelimesi harici bağlama sahip bir değişken ya da fonksiyon bildirir. Bunun anlamı, bu değişken ya da fonksiyonun tanımlandığı dosyalar haricinde de görülebilir olmasıdır. CCS C'de harici bağlama sahip fonksiyon yoktur.

## Static

**static** değişken sınıfı, önceden belirtilmedikçe 0 ile ilk değer atanmış global aktif değişkenleri tanımlar.

```
void test()
{
char x,y,z;
static int sayac = 4;
printf("sayac = %d\n",++sayac);
}
```

**sayac** değişkenine bir kez ilk değer ataması yapılmıştır ve sonra her fonksiyon çağrısında değeri bir kez arttırılır.

## Register

**register** deęişken sınıfı, büyük sistemlerde yüksek hızlı hafızaların sıklıkla kullanılan deęişkenler için ayrılması mantığından türetilmiştir. Bu sınıf sadece derleyiciye öneri amaçla kullanılır – CCS C içinde bu türden bir fonksiyon yoktur.

## 3. Fonksiyonlar

Fonksiyonlar C dilinin temel yapıtaşlarıdır. C'deki bütün ifadeler bir fonksiyonun içinde yer almalıdır. Bu bölümde fonksiyonlara argümanların iletilmesi ve bir fonksiyondan deęer döndürülmesi gibi konuları inceleyeceğiz.

Bu bölümde incelenen konular:

**Argümanların Fonksiyonlara İletilmesi**  
**Fonksiyonların Deęer Döndürmesi**  
**Fonksiyon Prototipleri**  
**Klasik ve Modern Fonksiyon Bildirimleri**

## 3.1 Fonksiyonlar

Önceki bölümlerde, ana programdan çağırılan birçok fonksiyon örneği görmüştük. Örneğin:

```
main() {
    f1();
}
int f1() {return 1;}
```

Gerçekte, bu programın hata yada en azından uyarı vermesi gerekir. Çünkü **f1()** fonksiyonu, değişkenlerde olduğu gibi, kullanımından önce bildirilmeli veya tanımlanmalıdır. Eğer standart C fonksiyonu kullanıyorsanız, programınızın başında **#include** deyimiyle programınıza eklediğiniz başlık dosyası derleyiciye fonksiyonu zaten tanıtmıştır. Eğer kendi fonksiyonlarınızdan birini kullanıyorsanız bu hatayı gidermenin iki yolu vardır: İlk yol, ikinci bölümde açıklanacak olan, fonksiyon prototipi kullanmaktır. Diğeri ise programınızı şu şekilde organize etmektir:

```
int f1()
{
    return 1;
}
main()
{
    f1();
}
```

Bir hata oluşmayacaktır çünkü **f1()** fonksiyonu **main()** içinde çağırılmadan önce tanımlanmıştır.

## 3.2 Fonksiyon Prototipleri

Derleyiciye bir fonksiyonun döndüreceği değeri bildirmenin iki yolu vardır. Genel biçim şudur:

```
tip fonksiyon_ismi();
```

Örneğin, **topla()** içindeki ifade derleyiciye **topla()**'nın bir tamsayı değeri döndüreceğini bildirebilir. Derleyiciye döndürülen değeri bildirmek için ikinci bir yol da fonksiyon prototipi kullanmaktır. Fonksiyon prototipi sadece fonksiyonun döndüreceği değeri belirtmekle kalmaz, aynı zamanda fonksiyonun kabul ettiği argüman sayısı ve tiplerini de bildirir. Prototip fonksiyon bildirimini ile örtüşmek zorundadır. Prototipler, fonksiyonlara iletilen argümanlarla, fonksiyon bildirimindeki argüman tipleri arasındaki hatalı tip dönüşümlerini göstererek programcıya program hatalarını bulmasında yardımcı olur. Ayrıca fonksiyona iletilen argüman sayısının, fonksiyon bildirimindeki ile aynı olmadığını da raporlar.

Fonksiyon prototipleri için genel biçim şu şekildedir:

```
tip fonksiyon_ismi(tip degisken1,tip degisken2,tip degisken3);
```

Yukarıdaki örnekte, her bir değişkenin tipi farklı olabilir. Bir fonksiyon prototipi örneği aşağıdaki programda gösterilmiştir. Fonksiyon uzunluk, genişlik ve yükseklik değerlerinden hacmi hesaplamaktadır.

```
int hacim(int s1, int s2, int s3);

void main()
{
    int hcm;
    hcm= hacim(5,7,12);
    printf("hacim: %d\n",hcm);
}
int hacim(int s1, int s2, int s3)
{
    return s1*s2*s3;
}
```

**return** ifadesinden sonra bir sabit ya da değişken yerine bir ifade kullanıldığına dikkat edin.

Prototiplerin önemi şimdiye kadar yazdığımız ufak programlarda tam olarak ortaya çıkmamış olabilir. Ancak programlar birkaç satırdan binlerce satıra doğru büyüdüğünde, prototiplerin hata gidermedeki önemi ortaya çıkacaktır.

### ALİŞTIRMALAR:

1. Derleyicinin hataları nasıl yakaladığını göstermek için, yukarıdaki programı hacim fonksiyonuna 4 parametre yollayacak şekilde değiştirin:

```
hcm = hacim(5,7,12,15)
```

2. Aşağıdaki program doğru mu? Değilse neden?

```
double fonk(void)

void main()
{
    printf("%f\n", fonk(10.2));
}
double fonk(double sayi)
{
    return sayi/2.0;
}
```

### 3.3 void

Bir fonksiyon hiçbir parametre almadığında ve hiçbir değer döndürmediğinde bildirimini şu şekilde yapılır:

## void hic(void)

Örnek olarak:

```
double pi(void)           //hiçbir parametre almadan
{
    return 3.1415926536; // fonksiyon tanımlanıyor
}
main()
{
    double pi_degeri;
    pi_degeri = pi();     //pi fonksiyonuyla pi değeri alınıyor
    printf("%d\n", pi_degeri);
}
```

## 3.4 Fonksiyon Argümanlarının Kullanımı

Fonksiyon argümanı fonksiyon çağırılırken, fonksiyona aktarılan değerdir. C fonksiyona birçok argümanın aktarılmasına izin vermektedir. Bir fonksiyonun kabul edebileceği argüman sayısı derleyiciye bağlı olmakla birlikte, ANSI C standardına göre bir fonksiyon en az 31 argüman kabul edebilmelidir.

Bir fonksiyon tanımlandığında, parametreleri alacak özel değişkenlerin bildirimini yapılmalıdır. Bu özel değişkenler formal parametrelerdir. Parametreler fonksiyonun isminden sonra parantez içinde belirtilirler. Örneğin, aşağıdaki fonksiyon çağırıldığı zaman kendisine iletilen iki tam sayının toplamını hesaplar ekrana yazar:

```
void topla(int a, int b)
{
    printf("%d\n", a+b);
}
```

Bu fonksiyonun bir programda çağırılmasına dair bir örnek:

```
void topla(int a, int b);           //Fonksiyon prototipi
main()
{
    topla(1,10);
    topla(15,6);
    topla(100,25);
}
void topla(int a, int b)
{
    printf("%d\n", a+b);
}
```

**topla()** çağırıldığında derleyici her bir argümanın değerini a ve b değişkenlerine kopyalayacaktır. Unutmamak gerekir ki fonksiyona iletilen değerlere **(1,10,15,6,100,25)** argüman, a ve b değişkenlerine formal parametre adı verilmektedir.

Fonksiyonlara argümanlar iki şekilde iletilebilir. İlk yola "değerle çağırma" denilmektedir. Bu metod, argümanların değerini fonksiyonun formal parametrelerine kopyalar. Formal parametrede yapılan herhangi bir değişiklik argüman değerini etkilemez. İkinci metodun adı "referans ile çağırma"dır. Bu metotta argümanın adresi fonksiyondaki formal parametreye kopyalanır. Bu fonksiyon içinde formal parametre, çağırıldığı yerdeki gerçek değişkene erişim için kullanılır. Bu, formal parametre kullanılarak değişkenin değerinin değiştirilebileceği anlamına gelmektedir. Bunu işaretçiler konusunda ayrıntılı olarak inceleyeceğiz. Şimdilik, fonksiyonlara argümanların iletilmesinde sadece değerle çağırma yöntemini kullanacağız.

### ALİŞTIRMALAR:

1. Bir tam sayı değeri alan ve bunu ekrana yazdıran fonksiyonu yazın.
2. Bu programda yanlış olan ne?

```
yazdir(int sayi)
{
    printf("%d\n", sayi);
}
main()
{
    yazdir(156.7);
}
```

## 3.5 Değer Döndürmek için Fonksiyonların Kullanımı

C dilinde herhangi bir fonksiyon çağırıldığı yere bir değer döndürebilir. Tipik olarak, fonksiyon eşittir (=) işaretinin sağ tarafına konulur. Döndürülen değer in illa bir atama deyiminde kullanılması gerekmez, ayrıca bir printf() ifadesinde de kullanılabilir. Derleyiciye fonksiyonun bir değer döndüreceğini belirtmek için kullanılan genel biçim:

```
tip fonksiyon_ismi(formal parametreler)
{
    <ifadeler>
    return deger;
}
```

**tip** fonksiyonun döndüreceği değer in veri tipini belirtmektedir. Bir fonksiyon dizi dışında herhangi bir veri tipi değeri döndürebilir. Eğer herhangi bir veri tipi belirtilmezse, C derleyicisi fonksiyonun tam sayı (integer - int) değeri döndürdüğünü varsayar. Eğer fonksiyonunuz değer döndürmüyorsa, ANSI C standardı fonksiyonun **void** döndürmesi gerektiğini belirtir. Bu derleyiciye fonksiyonun bir değer döndürmediğini anlatır. Aşağıdaki fonksiyon değer döndüren bir fonksiyona tipik bir örnektir.

```
#include <math.h>
main()
{
    double sonuc;
```

```
    sonuc = sqrt(16.0);  
    printf("%f\n", sonuc);  
}
```

Bu program kayar noktalı sayı değeri döndüren **sqrt()** fonksiyonunu çağırılmaktadır. Bu sayı değeri **sonuc** değişkenine atanmaktadır. **math.h** başlık dosyasının kullanıldığına dikkat edin. Bu dosya sayıların karekökünü alan **sqrt()** fonksiyonu hakkındaki bilgileri taşımaktadır.

Fonksiyonun döndüreceği değerin veri tipi ile bu döndürülen değerin atanacağı değişkenin veri tipinin aynı olmasına dikkat etmelisiniz. Aynı şey fonksiyona göndereceğiniz argümanlar için de geçerli. Öyleyse, bir fonksiyondan nasıl değer döndüreceğiz? Genel biçim şu şekildedir:

```
return degisken_ismi;
```

**degisken\_ismi** bir sabit, değişken veya döndürülen değerin veri tipindeki herhangi geçerli bir C ifadesi olabilir. Aşağıdaki örnekte her iki tipte fonksiyona ait örnek vardır:

```
fonk();  
topla(int a, int b);  
  
main()  
{  
    int sayi;  
    sayi = fonk();  
    printf("%d\n", sayi);  
    sayi = topla(5,127);  
    printf("%d\n", sayi);  
}  
fonk(){return 6;}  
topla(int a, int b)  
{  
    int sonuc;  
    sonuc= a + b;  
    return sonuc;  
}
```

Dikkat edilmesi gereken bir nokta, **return** ifadesine gelindiğinde fonksiyonun çağırıldığı yere döndürüldüğüdür. **return** ifadesinden sonraki ifadeler işletilmeyecektir. Bir fonksiyonun döndürdüğü değerin bir değişkene aktarılması veya bir ifadede kullanılması her zaman gerekli değildir. Ancak bu durumda döndürülen değer kaybolacaktır.

### ALİŞTIRMALAR:

1. 1 ve 100 arasında tam sayı değerlerini kabul eden ve bu sayının karekökünü döndüren fonksiyonu yazınız.
2. Bu fonksiyonda yanlış olan ne?

```
main()  
{  
    double sonuc;  
    sonuc = f1();  
}
```

```

        printf("%f\n", sonuc);
    }
    int f1()
    {
        return 60;
    }

```

### 3.6 Klasik ve Modern Fonksiyon Bildirimleri

C'nin orijinal versiyonunda formal parametrelerin bildirimlerinde farklı bir metod kullanılmıştır. Aşağıda gösterilen bu biçim, şimdi klasik biçim olarak adlandırılıyor.

```

tip function_name(deg1, deg2, ... degn)
tip deg1;
tip deg2;
.
tip degn;
{
    <ifadeler>
}

```

Bildirim iki bölümden oluştuğuna dikkat edin. Parantez içinde yalnızca parametrelerin isimleri bulunmaktadır. Parantezlerin dışında veri tipleri ve formal parametrelerin isimleri belirtilmiştir.

Önceki örnekte incelediğimiz modern biçim ise şu şekildedir:

```

tip fonksiyon_ismi(tip deg 1, ... tip var n)

```

Bu fonksiyon bildiriminde, veri tipleri ve formal parametre isimleri parantez içinde belirtilmiştir.

ANSI C standardı her iki fonksiyon bildirimine de izin vermektedir. Buradaki amaç, milyarlarca satır olan eski C programlarıyla uyumluluğu sağlamaktır. Eğer bir kodda klasik biçimi görürseniz endişelenmeyin: C derleyiciniz bu durumla başa çıkabilmelidir. Ancak yeni programlarınızda modern biçimi tercih etmelisiniz.

#### ALİŞTIRMALAR:

1. Fonksiyon prototipi nedir ve bunu kullanmanın faydaları nelerdir?
2. Aşağıdaki programı klasik fonksiyon biçimini kullanarak, modern biçime çevirin.

```

void main(void)
{
    printf("alan = %d\n", alan(10,15));
}
alan(1,w)
int 1,w
{
    return 1*w;
}

```

### 3.7 Dizge(string) Sabitlerin Fonksiyonlara İletilmesi

PICmicro® Mikrodenetleyicileri ROM erişiminde bazı kısıtlamalara sahip olduğundan, dizge(string) sabitleri fonksiyonlara normal yoldan iletilemezler. CCS C derleyicisi bu sorunu standart dışı bir şekilde aşmaktadır: Eğer dizge sabiti sadece bir karakter parametresine sahip fonksiyona iletilecekse, dizgedeki her bir karakter için ayrı ayrı çağırılır. Örneğin:

```
void lcd_yaz(char c)
{
    .....
}
lcd_yaz("abcd");
```

aşağıdakine eşdeğerdir:

```
lcd_yaz("a");
lcd_yaz("b");
lcd_yaz("c");
lcd_yaz("d");
```

## 4. C Operatörleri

İfadeler C'de önemli rol oynar. Bunun en büyük sebebi C'de diğer dillere göre daha çok operatör tanımlanmıştır. Bir ifade operatörler ve operandların bileşimidir. Birçok durumda C operatörleri cebir kurallarını takip eder ve göze anlaşılır gelir.

Bu bölümde aşağıda sayılan operatörleri inceleyeceğiz:

**Aritmetik**  
**İlişkisel**  
**Mantıksal**  
**Bit İşlemleri**

## Artırma ve Azaltma Operatörleri Operatörlerin Öncelik Sıralaması

### 4.1 Aritmetik Operatörler

C dili toplama, çıkarma, çarpma, bölme ve mod işlemi olmak üzere 5 aritmetik işlem operatörü tanımlar:

```
+ toplama  
- çıkarma  
* çarpma  
/ bölme  
% kalan
```

+, -, \* ve / operatörleri herhangi bir veri tipiyle kullanılabilir. %, operatörü yalnızca tamsayılarla birlikte kullanılabilir. Kalan operatörü tamsayı bölmede kalanı verir. Dolayısıyla bu operatör kayar noktalı sayı işlemlerinde anlam ifade etmez.

- Operatörü iki şekilde kullanılabilir: Çıkarma operatörü olarak ve ikinci olarak da bir sayının işaretini değiştirmek için kullanılabilir. Aşağıdaki örneklerde bu iki kullanım gösteriliyor:

```
a = a - b      ; çıkarma  
a = -a        ; a'nın işaretinin değiştirilmesi
```

Aritmetik operatörler her hangi bir sabit kombinasyonu ve/veya değişkenlerle kullanılabilir. Örneğin, aşağıdaki geçerli bir C ifadesidir:

```
sonuc = sayac - 163;
```

C ayrıca aritmetik işlem operatörlerinin kullanımıyla ilgili bazı kısayolların kullanımına olanak verir. Önceki örneklerdeki, **a = a - b**; ifadesi **a -=b**; şeklinde de yazılabilir. Bu metod +, -, \*, ve / operatörleri için de geçerlidir. Aşağıda bu kısa yollarla ilgili örnekler görülmektedir:

<b>a*=b</b>	eşdeğerdir	<b>a=a*b</b>
<b>a/=b</b>		<b>a=a/b</b>
<b>a+=b</b>		<b>a=a+b</b>
<b>a-=b</b>		<b>a=a-b</b>
<b>a%=b</b>		<b>a=a%b</b>
<b>a&lt;&lt;=b</b>		<b>a=a&lt;&lt;b</b>
<b>a&gt;&gt;=b</b>		<b>a=a&gt;&gt;b</b>
<b>a&amp;=b</b>		<b>a=a&amp;b</b>
<b>a =b</b>		<b>a=a b</b>
<b>a^=b</b>		<b>a=a^b</b>

C kodunu ve derlenmiş assembly kodunu karşılaştırarak aritmetik işlemlerin PIC ile nasıl gerçekleştirildiğini görelim:

```
int a,b,c;
```

**a = b + c;**

ifadeleri şu şekilde dönüşür:

```
0007:          MOVF      0F,W      ; b'yi yükle
0008:          ADDWF    10,W      ; c'yi b'ye ekle
0009:          MOVWF    0E        ; sonucu a'ya kaydet
```

**a = b - c;**

ifadeleri şu şekilde dönüşür:

```
0007:          MOVF      0F,W      ; b'yi yükle
0008:          MOVWF    0E        ; a'ya kaydet
0009:          MOVF      10,W      ; c'yi yükle
000A:          SUBWF    0E,F      ; a'dan çıkar
```

Derleyicinin ürettiği kodu anlamamanın önemi problemlerle uğraşırken ortaya çıkmaktadır. Birçok zaman derleyici liste dosyasına bakmak (.LST) birçok C hatasının bulunmasını sağlar. En çok yapılan hatalardan biri = ve == operatörlerinin kullanımındadır.

**a = b;**

ifadesi şu şekilde dönüşür:

```
0007:          MOVF      0F,W      ; b'yi yükle
0008:          MOVWF    0E        ; a'ya kaydet
```

buna karşın

**a==b;**

ifadesi şu şekilde dönüşür:

```
0007:          MOVF      0F,W      ; b'yi yükle
0008:          SUBWF    0E,F      ; a'dan çıkar
0009:          BTFSC    03,2      ; sonuc sıfır mı test et
000A:          GOTO     00D        ; evet - öyleyse dallan
```

İlk örnekte **a b**'ye eşitlenmektedir. İkinci de ise **a**'nın **b**'ye eşit olup olmadığı test edilmektedir.

### **ALIŞTIRMALAR:**

1. 5/5, 5/4, 5/3, 5/2, ve 5/1 işlemlerinde kalanı hesaplayan programı yazınız.
2. Bir yıl içindeki saniye sayısını hesaplayan programı yazınız.

## **4.2 İlişkisel Operatörler**

C'deki ilişkisel operatörler iki değeri karşılaştırır ve sonuçta doğru ya da yanlış gibi bir sonuç döndürür. İlişkisel operatörler şunlardır:

```
> büyüktür
>= büyüktür ya da eşittir
< küçüktür
<= küçük ya da eşittir
== eşittir
!= eşit değildir
```

İlişkisel operatörlerde döndürülen sonuç her zaman ya 0 ya da 1'dir. Ancak C her hangi sıfır-olmayan bir değeri doğru (1) olarak değerlendirir. Yanlış değer her zaman 0 olarak tanımlıdır.

Aşağıda ilişkisel operatörlerin kullanıldığı bazı ifadeler görülmektedir.

```
sayi > 15 ; eğer sayi 15'e eşit ya da 15'ten küçükse sonuç 0'dır.
(yanlış)
sayi != 15; eğer sayi 15'ten büyük ya da 15'ten küçükse sonuç 1'dir.
(doğru)
```

### ALİŞTIRMALAR:

1. Aşağıdaki ifadeyi farklı bir ilişkisel operatör kullanara yeniden yazın!

```
sayac != 0
```

2. Aşağıdaki ifade ne zaman doğru ya da yanlıştır? Neden?

```
sayac >= 35
```

## 4.3 Mantıksal Operatörler

Mantıksal operatörler VE(AND),VEYA(OR) ve DEĞİL(NOT) gibi temel lojik işlemleri desteklerler. Bu operatörler de sonuç olarak yanlış için 0, doğru için 1 değerini döndürürler. Lojik operatörler ve doğruluk tabloları aşağıda verilmiştir.

		AND	OR	NOT
p	q	p&&q	p  q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Mantıksal ve ilişkisel operatörler birarada kullanılarak ifadeler oluşturulabilir. Buna bir örnek verecek olursak:

```
sayac>maks || !(maks==57) && deg>=0
```

C'de ilişkisel ve mantıksal operatörlerin program kontrol ifadelerinde kullanımını bir sonraki bölümde göreceğiz.

### ALİŞTIRMALAR:

1. Aşağıdaki ifadeleri herhangi ilişkisel ve mantıksal operatör bileşimlerini kullanarak yeniden yazın.

```
sayac == 0
```

```
sonuc <= 5
```

2. C 'de özel VEYA (XOR) işlemi için bir operatör bulunmadığından, aşağıdaki doğruluk tablosuna göre bir XOR fonksiyonu yazın.

P	q	XOR
0	0	0
0	1	1
1	0	1
1	1	0

## 4.4 Bit İşlem Operatörleri

C 'de bitler üzerinde işlem yapabilmek üzere 6 özel operatör bulunmaktadır. Bu operatörler yalnızca tamsayı ve karakter veri tipleriyle birlikte kullanılabilirler.

Bit işlem operatörleri:

&	bit işlem AND
	bit işlem OR
^	bit işlem XOR
~	1'e tümleyeni
>>	sağa kaydırma
<<	sola kaydırma

Kaydırma operatörlerinin genel kullanım biçimi:

```
degisken<< ifade
degisken >> ifade
```

**ifade**' nin değeri **degisken**' in bitlerinin kaç kere kaydırılacağını belirtir. Her bir sola kaydırma işlemi, bütün bitleri bir pozisyon sola kaydırır ve sağ tarafa 0 değeri verir. Değişkenin en sonundaki bitin değeri kaybolur.

Sağa ve sola kaydırma işlemleri ile ilgili önemli bir özellik, sola kaydırmak 2 ile çarpma işlemine, sağa kaydırmak ise 2 ile bölme işlemine eşdeğerdir. Kaydırma işlemleri MİB'nin çalışma sisteminden dolayı aritmetik işlemlerden daha hızlıdır.

Aşağıda bütün bit işlem operatörlerine ilişkin örnekler verilmiştir:

	<b>AND</b>			<b>OR</b>
	00000101 (5)			00000101 (5)
&	00000110 (6)			00000110 (6)
	-----			-----
	00000100 (4)			00000111 (7)
	<b>XOR</b>			<b>NOT (1'e tümleme)</b>
	00000101 (5)		~	00000101 (5)
^	00000110 (6)			-----
	-----			11111010 (250)
	00000011 (3)			

<b>LEFT SHIFT</b>	<b>RIGHT SHIFT</b>
00000101 (5)	00000101 (5)
<< 2	>> 2
-----	-----
00010100 (20)	00000001 (1)

### NOT:

Bir işlenenin bit sayısından fazla sayıda kaydırma yapmayın – tanımsız sonuc

`a = b | c;`

şu şekilde dönüştürülür:

```

0007:    MOVF  0F,W      ; b'yi yükle
0008:    IORWF 10,W     ; c ile veya işlemine sok
0009:    MOVWF 0E       ; sonucu a'ya kaydet

```

`a = b & c;`

şu şekilde dönüştürülür:

```

0007:    MOVF  0F,W      ; b'yi yükle
0008:    ANDWF 10,W     ; c ile VE işlemine sok
0009:    MOVWF 0E       ; sonucu a'ya kaydet

```

`a = b >> 3;`

şu şekilde dönüştürülür:

```

0007:    MOVF  0F,W      ; b'yi yükle
0008:    MOVWF 0E       ; a'ya kaydet
0009:    RRF   0E,F     ; içeriği
000A:    RRF   0E,F     ; 3 kere
000B:    RRF   0E,F     ; sağa kaydır
000C:    MOVLW 1F       ; a'nın içeriği için
000D:    ANDWF 0E,F     ; maskeleye yap

```

`j = ~a;`

şu şekilde dönüştürülür:

```

0007:    MOVF  0F,W      ; a'yı yükle
0008:    MOVWF 0E       ; j'ye kaydet
0009:    COMF  0E,F     ; j'nin tersini al

```

### ALİŞTIRMALAR:

1. İşaretsiz karakter tipindeki bir değişkenin sadece MSB'ini tersleyen bir program yazınız.
2. Karakter tipindeki bir sayının ikili sayı şeklini gösteren bir program yazınız.

## 4.5 Artırma ve Azaltma Operatörleri

Bir değişkenin değerini nasıl bir arttırır veya azaltırsınız? Muhtemelen aklınıza şu ifadeler gelmektedir:

`a = a+1;` veya `a = a-1;`

C'nin tasarımcıları bir artırma ve azaltma için bir kısayol oluşturmuşlardır. Genel biçim şu şekildedir:

`a++;` veya `++a;` artırma için  
`a--;` veya `--a;` azaltma için

**++** veya **--** işaretleri değişkenlerin **önünde** yer aldığında, değişkenin değeri önce bir artırılır veya azaltılır ardından ifadede kullanılır. **++** veya **--** işaretleri değişkenlerin **sonunda** yer aldığında ise, değişkenin değeri önce ifadede kullanılır ardından bir artırılır veya azaltılır.

```
int j, a = 3;

0007:    MOVLW 03
0008:    MOVWF 0F    ; a'nın değeri yazmaca atandı
j = ++a;
0009:    INCF 0F,F    ; a = 4
000A:    MOVF 0F,W    ; a'nın değerini w yazmacına yükle
000B:    MOVWF 0E    ; w yazmacını j'ye kaydet

j = a++;
000C:    MOVF 0F,W    ; a'nın değerini w yazmacına yükle
000D:    INCF 0F,F    ; a = 5
000E:    MOVWF 0E    ; j = 4
```

### NOT:

Aşağıdaki biçimi kullanmayın:

```
a = a++;
```

Çünkü bu kod şu şekle dönüştürülür:

```
MOVF    0E,W    ; a'nın değeri w'ye yüklendi
INCF    0E,F    ; a'daki değer 1 artırıldı
MOVWF  0E    ; artımdan önceki değer tekrar a'daki değerinin
; üzerine yüklendi
```

Aşağıdaki örnekte her iki kullanım biçimi de gösterilmektedir:

```
void main(void)
{
    int i, j;
    i = 10;
    j = i++;
    printf("i = %d, j = %d\n", i, j);
    i = 10;
    j = ++i;
    printf("i = %d, j = %d\n", i, j);
}
```

İlk **printf()** ifadesi **i** için **11** ve **j** için **10** yazdıracaktır. İkinci **printf()** ifadesi ise **i** ve **j** için **11** yazdıracaktır.

## Parçaları Birleştirmek:

Yazılış	İşlem
<code>toplam = a+b++</code>	<code>toplam = a+b, b = b+1</code>
<code>toplam = a+b--</code>	<code>toplam = a+b, b = b-1</code>
<code>toplam = a+ ++b</code>	<code>b = b+1, toplam = a+b</code>
<code>toplam = a+ -- b</code>	<code>b = b-1, toplam = a+b</code>

## ALIŞTIRMALAR:

1. Aşağıdaki programdaki arttırma ve azaltma ifadelerini yeniden yazın.

```
void main(void)
{
    int a, b;
    a = 1;
    a = a+1;
    b = a;
    b = b-1;
    printf("a=%d, b=%d\n", a,b);
}
```

2. Aşağıdaki kodlar işletildiğinde a ve b değişkenlerinin son değeri ne olur?

```
a = 0;
b = 0;
a = ++a + b++;
a++;
b++;
b = -a + ++b;
```

## 4.6 Operatörlerin Öncelik Sıralaması

Öncelik, operatörlerin C derleyicisi tarafından işleme sıralamasını belirtmektedir. Örneğin, programınızda **a+b\*c** ifadesi ile karşılaşıldığında hangi işlem ilk olarak gerçekleştirilecektir? Toplama mı çarpma mı? Aşağıda C derleyicisinin operatörler için takip ettiği öncelik sıralaması görülmektedir:

Öncelik	Operatör	Örnek
1	<code>() ++ --</code>	<code>(a+b)/c</code> parantez
2	<code>sizeof &amp; * + - ~ ! ++ --</code>	<code>a=-b</code> artı/eksi/DEĞİL/tümleme artırma/azaltma/sizeof
3	<code>* / %</code>	<code>a*b</code> çarpma/bölme/elde
4	<code>+ -</code>	<code>a+b</code> toplama/çıkarma
5	<code>&lt;&lt; &gt;&gt;</code>	<code>a=b&gt;&gt;c</code> sola veya sağa kayırma
6	<code>&lt; &gt; &lt;= &gt;=</code>	<code>a&gt;=b</code> büyük/küçük/eşit

7	== !=	a= =b
8	&	a=b&c bit işlem VE
9	^	a=b^c bit işlem DIŞLAYAN VEYA (XOR)
10		a=b c bit işlem VEYA
11	&&	a=b&&c mantıksal VE
12		a=b  c mantıksal VEYA
13	= *= /= %= += -= <<= >>=	a+=b atama
	§= ^=  =	

Bu operatörlerin bir kısmını henüz görmedik. Ancak endişelenmeyin ileriki bölümlerde göreceğiz.

Parantez kullanılarak işlemlerin sıralaması belirtilebilir. Parantez kullanarak ifadelerin önceliğinin değiştirilmesine dair birkaç örnek daha:

```
10-2*5 = 0
(10-2)*5 = 40
```

```
sayac*toplam+88/deg-19%sayac
(sayac*toplam) + (88/deg) - (19%sayac)
```

## ALİŞTIRMALAR:

1. Aşağıdaki kodlar işletildiğinde a ve b değişkenlerinin son değeri ne olur?

```
int a=0,b=0;

a = 6 8+3b++;
b + = -a*2+3*4;
```

## 5 Program kontrol deyimleri

Bu bölümde C'de programınızın akışını kontrol etmeyi öğreneceksiniz. Ayrıca ilişkisel ve mantıksal kontrol operatörlerinin bu kontrol deyimleriyle nasıl kullanılacağını ve buna ek olarak nasıl kontrol döngüleri kurulacağını öğreneceksiniz.

Bu bölüm içerisinde ele alınacak kontrol deyimleri :

```
if
if-else
for
while
do-while
iç-içe döngüler
break
continue
switch
null
return
```

### 5.1 if deyimi

**if** deyimi bir şartlandırma deyimidir. **if** ifadesinde yer alan kod bloğu bir şartın sonucuna bağlı olarak icra edilir. Ayrıca, değer sıfır değil iken doğru (true), değer sıfır iken yanlıştır (false). En basit hali :

```
if (ifade)                NOT: ifadeden sonra ";" kullanılmaz.
    deyim;
```

Bu ifade geçerli bir C ifadesi olabilir. **if** deyimi ifadenin sonucunun doğru veya yanlış olduğunu değerlendirir. Eğer ifade doğru ise deyim çalıştırılır. Eğer ifade yanlış ise program, deyimi çalıştırmadan devam eder. Basit bir örnek :

```
if(num>0)
    printf("Sayı pozitif\n");
```

Bu örnek, ilişkisel operatörlerin program kontrol deyimleriyle nasıl kullanıldığını gösterir. **if** deyimi ayrıca kod bloklarının çalıştırılmasının kontrolünde de kullanılabilir. Genel formatı şöyledir :

```
if (ifade)
{
    .
    deyim;
    .
}
```

Kod bloğu { ve } sembollerinin içerisine yerleştirilir. Bu, derleyiciye, ifade doğru ise parantezler arasındaki kodların çalıştırılacağını anlatır. **if** ve kod bloğuna bir örnek :

```
if (count <0 )
{
    count =0;
    printf("Count down\n");
}
```

veya

```
if(TestMode ==1)
{
    ... yazdırılacak kodlar...
}
```

**if** deyiminde kullanılan diğer karşılaştırma operatörleri :

```
x == y x eşittir y
x != y x eşit değildir y
x > y x büyüktür y
x < y x küçüktür y
x <= y x küçük veya eşit y
x >= y x büyük veya eşit y
x && y mantıksal AND
x || y mantıksal OR
```

Herhangi bir fonksiyonun assembly diline çevirilmiş bir örneği :

```

int j, a =3;
0007: MOVLW 03          ;a'ya 3 yükle
0008: MOVWF 0F
if (j == 2)
0009: MOVLW 02          ;w'ye 2 yükle
000A: SUBWF 0E,W        ;j ile eşliğini test et
000B: BTFSS 03,2        ;eğer sıfır ise atla
000C: GOTO 00F
{
j = a;
000D: MOVF 0F,W         ;eğer sıfır ise
000E: MOVWF 0E         ;j'ye a'yı yükle
}

```

### Alıştırmalar :

- Aşağıdaki ifadelerin hangilerinin değerleri doğrudur?
  - 0
  - 1
  - 1
  - $5*5 < 25$
  - $1==1$

2. Sayının tek veya çift olup olmadığını bulan bir fonksiyon yazın. Fonksiyon çıktısı 0 iken sayı çift, 1 iken sayı tek. Bu fonksiyonu 1 ve 2 sayılarıyla çağırın.

## 5.2 if-else deyimleri

Eğer ifadenin sonucuna göre çalıştırılacak iki kod bloğunuz varsa ne yapardınız? Eğer ifade doğru ise ilk kod bloğu çalıştırılacak, eğer ifade yanlış ise ikinci kod bloğu çalıştırılacak. Muhtemelen if deyimi ile else deyimini birlikte kullanacaksınız. İf-else deyiminin genel formatı şöyledir :

```

if (ifade)
    deyim1;
else
    deyim2;

```

if-else deyimi için kod blokları kullanma formatı (bir satırdan fazla) :

```

if (ifade)
{
    .
    deyim;
    .
}
else
{
    .
    deyim;
    .
}

```

Bir if veya else ifadesini akılda tutmak için bir çok ifade gerekli olabilir. if veya else için sadece bir ifade olduğu zaman {} parantezler kullanılmayabilir. Tek if-else deyiminin bir örneği :

```

if (num<0)
    printf("Sayı negatif.\n");
else
    printf("Sayı pozitif.\n");

```

**Else** deyimini eklemek sizin için çift yönlü karar imkanı sağlar. Fakat, eğer birkaç if ve else deyimini birleştirip, birlikte birkaç karar almak isteseydiniz ne yapardınız? C'nin en önemli sağladığı olanak, if ve else'leri kullanarak istenilen sayıda durum kontrol edilip karar verebilmesidir. Genel hali şöyledir :

```

if (ifade1)
{
    .
    deyim(ler)
    .
}
else if (ifade2)
{
    .
    deyim(ler)
    .
}
else
{
    .
    deyim(ler)
    .
}

```

Burada kodun icrasında şarta göre seçip değerlendirmek için birkaç farklı ifade olabileceğini görüyoruz. Bunu bir örnekle açıklayalım :

```

if(num == 1)
    printf("deger 1\n");
else if(num == 2)
    printf("deger 2\n");
else if(num == 3)
    printf("deger 3\n");
else
    printf("deger farkli\n");

```

#### NOT:

if deyiminin içinde kullanılan karşılaştırma operatörlerinin tek hali olduğu gibi çift halide bulunmaktadır (tek **&**, **=** veya **|** operatörleri ve çift **&&**, **==** veya **||** operatörleri gibi). **Yapılan ortak hatalardan biri de; bu operatörlerden birinin yerine diğerini kullanmaktır. Bu kodlar belki hatasız derlenecektir fakat program hatalı işleyecektir.**

#### Alıştırmalar :

1. Bu kod parçacığı içindeki hatalar nelerdir?

```

if (sayac>20)
printf("sayac 20 den büyük");
sayac- ;

```

}

2. Verilen sayının içinde kaç tane 5, 10, 25, 50 ve 100 sayısı bulunduğunu bulan bir program yazınız.

### 5.3 ? operatörü

Koşul operatörü (conditional operator) C dilinin 3 işleneni alan tek operatördür. (ternary operator) Koşul operatörünün 3 işleneni, ifade tanımına uygun herhangi bir ifade olabilir. Koşul operatörünün genel yazımı aşağıdaki gibidir:

```
ifade1 ? ifade2 : ifade3
```

Koşul operatörü yukarıdaki biçimden de görüldüğü gibi birbirinden ayrılmış iki işareten oluşmaktadır. ? ve : işaretleri operatörün 3 işleneni birbirinden ayırır.

Derleyici bir koşul operatörü ile karşılaştığını ? işaretinden anlar ve ? işaretinin solundaki ifadenin (**ifade1**) sayısal değerini hesaplar. Eğer **ifade1**'in değeri **0** dışı bir sayısal değerse, bu durum koşul operatörü tarafından **doğru** olarak değerlendirilir ve bu durumda yalnızca **ifade2**'nin sayısal değeri hesaplanır.

Eğer **ifade1**'in değeri **0** ise bu durum koşul operatörü tarafından **yanlış** olarak değerlendirilir ve bu durumda yalnızca **ifade3**'ün sayısal değeri hesaplanır.

Diğer operatörlerde olduğu gibi koşul operatörü de bir değer üretir. Koşul operatörünün ürettiği değer **ifade1 doğru** ise (**0** dışı bir değer ise) **ifade2**'nin değeri, **ifade1 yanlış** ise **ifade3**'ün değeridir. Örnek:

```
m = x > 3 ? y + 5 : y - 5;
```

Burada önce **x > 3** ifadesinin sayısal değeri hesaplanacaktır. Bu ifade **0** dışı bir değerse (yani **doğru** ise) koşul operatörü **y + 5** değerini üretecektir. **x > 3** ifadesinin değeri **0** ise (yani **yanlış** ise) koşul operatörü **y - 5** değerini üretecektir. Bu durumda **m** değişkenine **x > 3** ifadesinin doğru ya da yanlış olmasına göre **y + 5** ya da **y - 5** değeri atanacaktır.

Aynı işlem **if** deyimi ile de yapılabilir :

```
if (x > 3)
m = y + 5;
else
m = y - 5;
```

Koşul operatörü Operatör Öncelik Tablosunun 13. öncelik seviyesindedir. Bu seviye atama operatörünün hemen üstüdür. Aşağıdaki ifadeyi ele alalım:

```
x > 3 ? y + 5 : y - 5 = m;
```

Koşul operatörünün önceliği atama operatöründen daha yüksek olduğu için, önce koşul operatörü ele alınır.  $x > 3$  ifadesinin **DOĞRU** olduğunu ve operatörün  $y + 5$  değerini ürettiğini düşünelim. Toplam ifadenin değerlendirilmesinde kalan ifade

$$y + 5 = m$$

olacak ve bu da derleme zamanı hatasına yol açacaktır. Çünkü  $y + 5$  ifadesi sol taraf değeri değildir, nesne göstermez. (Lvalue required hatası verecektir).

Koşul operatörünün birinci kısmını (**ifade1**) parantez içine almak gerekmez. Ancak, okunabilirlik açısından genellikle parantez içine alınması tercih edilir.

$$(x >= y + 3) ? a * a : b;$$

Koşul operatörünün üçüncü operandı (**ifade3**) konusunda dikkatli olmak gerekir. Örneğin :

$$m = a > b ? 20 : 50 + 5;$$

$a > b$  ifadesinin doğru olup olmamasına göre koşul operatörü **20** ya da **55** değerini üretecek ve son olarak da **m** değişkenine koşul operatörünün ürettiği değer atanacaktır. Ancak **m** değişkenine  $a > b ? 20 : 50$  ifadesinin değerinin **5** fazlası atanmak isteniyorsa bu durumda ifade aşağıdaki gibi düzenlenmelidir:

$$m = (a > b ? 20 : 50) + 5;$$

Koşul operatörünün 3 işleneni de bir fonksiyon çağırma ifadesi olabilir, ama çağırılan fonksiyonların geri dönüş değeri üreten fonksiyonlar olması (**void** olmayan) gerekmektedir. Üç işlenenden biri geri dönüş değeri **void** olan bir fonksiyona ilişkin fonksiyon çağırma ifadesi olursa koşul operatörü değer üretmeyeceğinden bu durum derleme zamanında hata oluşmasına neden olacaktır.

## 5.4 for döngüsü

For döngüleri yalnızca C dilinin değil, belki de tüm programlama dillerinin en güçlü döngü yapılarıdır. **for** döngülerinin genel biçimi şu şekildedir:

```
for (ifade1; ifade2; ifade3)
deyim1;

for (ifade1; ifade2; ifade3) {
    deyim1;
    deyim2;
    ...
}
```

Derleyici **for** anahtar sözcüğünden sonra bir parantez açılmasını ve parantez içerisinde iki noktalı virgül bulunmasını bekler. Bu iki noktalı virgül **for** parantezini

Üç kısma ayırır. Bu kısımlar yukarıda ifade1 ifade2 ve ifade 3 olarak gösterilmiştir.

**for** parantezi içinde mutlaka 2 noktalı virgül bulunmalıdır. **for** parantezi içinin boş bırakılması, ya da **for** parantezi içerisinde 1, 3 ya da daha fazla noktalı virgölün bulunması derleme zamanında hata oluşmasına yol açacaktır.

**for** parantezinin kapanmasından sonra gelen ilk deyim döngü gövdesini oluşturur. Döngü gövdesi basit bir deyimden oluşabileceği gibi, bileşik deyimden de yani blok içine alınmış birden fazla deyimden de oluşabilir. Döngü gövdesi tek bir ifadeden oluşacaksa ifadenin { } arasına alınmasına gerek yoktur.

**for** parantezi içerisindeki her üç kısmın da ayrı ayrı işlevleri vardır.

for parantezinin 2. kısmını oluşturan ifadeye kontrol ifadesi denir. Tıpkı **while** parantezi içindeki ifade gibi, döngünün devamı konusunda bu ifade söz sahibidir. Bu ifadenin değeri 0 dışı bir değer ise, yani mantıksal olarak doğru ise, döngü devam eder. Döngü gövdesindeki deyim(ler) icra edilir. Kontrol ifadesinin değeri 0 ise programın akışı for döngüsünün dışındaki ilk deyimle devam edecektir.

**for** döngüsünün 3. kısmı döngü gövdesindeki deyim ya da deyimler icra edildikten sonra, dönüşte çalıştırılır. Ve çoğunlukla döngü değişkeninin artırılması ya da azaltılması amacıyla kullanılır. (Böyle bir zorunluluk yok.)

```
for (ilk değer; koşul; işlem) {
    ...
    ...
    ...
}

int main()
{
    int i;

    for (i = 0; i < 2; ++i)
        printf("%d\n", i);
    printf("son değer = %d\n", i);
    return 0;
}
```

Yukarıdaki programı inceleyelim:

Programın akışı for deyimine gelince, önce for parantezi içindeki 1. ifade icra ediliyor. Yani **i** döngü değişkenine 0 değeri atanıyor.

Şimdi programın akışı for parantezinin 2. kısmına yani kontrol ifadesine geliyor ve **i < 2** koşulu sorgulanıyor. Kontrol ifadesinin değeri **0** dışı bir değer olduğu için, ifade mantıksal olarak doğru kabul ediliyor ve programın akışı döngü gövdesine geçiyor. Döngü gövdesi bloklanmadığı için, döngü gövdesinde tek

bir deyim var. (basit deyim). Bu deyim icra ediliyor. **printf("%d\n", i)** ifadesi **i=0** için işletiliyor. Yani ekrana i değişkeninin değeri yazılarak imleç alt satıra geçiriliyor (\n den dolayı alt satıra geçiyor).

Programın akışı bu kez for parantezinin 3. kısmına geliyor ve buradaki ifade bir deyimmiş gibi icra ediliyor, yani i değişkeninin değeri 1 artırılıyor. i değişkeninin değeri **1** oluyor.

2. ifade yeniden değerlendiriliyor ve **i < 2** ifadesi **doğru** olduğu için bir kez daha döngü gövdesi icra ediliyor.

Programın akışı yine for parantezinin 3. kısmına geliyor ve buradaki ifade bir deyimmiş gibi icra ediliyor, yani i değişkeninin değeri 1 artırılıyor. **i** değişkeninin değeri **2** oluyor.

Programın akışı yine for parantezinin 2. kısmına geliyor ve buradaki kontrol ifadesi tekrar sorgulanıyor. **i < 2** ifadesi bu kez **yanlış** olduğu için programın akışı döngü gövdesine girmiyor ve programın akışı döngü gövdesi dışındaki ilk deyimle devam ediyor.

Yani ekrana :

**0**

**1**

**son değer = 2**

yazılıyor.

Aşağıda farklı for döngüleri bulunmaktadır :

```
for (num=100; num>0; num=num-1)
for (count=0; count<50; count+=5)
for (count=1; count<10 && error==false; count++)
```

Bir örneği assembly'e dönüştürüp nasıl olacağını görelim :

```
int h,a;
for (h=0;h!=10;h++)
0007: CLRF      0E          ;h'ı temizle
0008: MOVLW 0A          ;W kaydedicisine 10 yükle
0009: SUBWF 0E,W        ;h dan çıkart
000A: BTFSC 03,2        ;0 mı diye test et
000B: GOTO      00F        ;eğer h=10, döngüden çık
a++;
000C: INCF      0F,F        ;a'yı arttır
      h++;
000D: INCF      0E,F        ;h'ı arttır
000E: GOTO      008        ;0008'e git (döngüye devam)
```

**Alıştırmalar :**

1. Aşağıdaki for() deyimleri nasıl çalışır?

```
for(i=1; ;i++)
for( ; ; )
for(num=1; num; num++)
```

2. Sayının tüm çarpanlarını görüntüleyen bir program yazınız.

## 5.5 while döngüsü

C'deki diğer bir döngü de **while** döngüsüdür. While döngüsü, ifade doğru olduğu sürede deyimi veya kod bloğunu tekrarlar. Bu nedenle ismi **while** dir. Burada genel kullanımı görülmektedir :

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

veya

```
while (ifade)
{
    deyim;
}
```

**İfade** geçerli herhangi bir C ifadesidir. Her seferinde **ifadenin** değeri **deyimden** veya kod bloğundan önce kontrol edilir. Bu demek ki, eğer **ifade** yanlış (false) ise **deyim** veya kod bloğu hiç bir şekilde icra edilmez.

While döngüsüne bir örnek :

```
#include <16C74.H>
#use RS232 (Baud=9600, xmit=pin_c6, RCV=pin_c7)
void main(void)
{
    char ch;
    printf("q ver\n");
    ch=getch();
    while(ch!='q')
    ch=getch();
    printf("iste q!\n");
}
```

Klavyeden istenilen karakteri alınca uyarılacaksınız. Öyleyse ifade değerlendirildi. **ch**'nin değeri **q** olmadığı sürece program klavyeden başka bir değer almaya devam edecektir. İlk **q** alındığında **printf** çalıştırılır ve program sonlanır.

Alıştırmalar :

1. Aşağıdaki while deyimleri nasıl işler?

a. `while(i<10)`  
{  
    printf("%d ",i);  
    i++;  
}

b. `while(1)`  
    printf("%d ",i++);

2. Klavyeden **ch=getch();** deyimi kullanarak aldığı verileri ekrana yazdıran ve önceden belirlenmiş bir karakter ile karşılaştığında sonlanan bir program yazınız.

## 5.6 do-while döngüsü

C'deki son döngü **do** döngüsüdür. Burada **do** ve **while**'ı birleştiriyoruz :

```
do
{
    deyimler
}
while (ifade)
```

Bu şekilde her zaman deyimler, ifade değerlendirilmeden önce icra edilecektir. **ifade** belki geçerli bir C ifadesidir. **do-while** için basit bir örnek :

```
#include <16C74.H>
#use RS232 (Baud=9600, xmit=pin_c6, RCV=pin_c7)
void main(void)
{
    char ch;
    do
    {
        ch = getch();
        while(ch != 'q');
        printf("iste q!\n");
    }
}
```

Bu program 5.5inci kısımdaki program ile aynı işi yapmaktadır.

Alıştırmalar :

1. 5.5inci kısımdaki Alıştırma 1'in hem a hemde b kısmını **do-while** kullanarak tekrar yazınız.
2. 5.5inci kısımdaki Alıştırma 2'yi **do-while** kullanarak tekrar yazınız.

## 5.7 İççe döngüler

Bir döngünün gövdesini başka bir kontrol deyimi oluşturabilir. Döngü gövdesini oluşturan kontrol deyimi bir if deyimi olabileceği gibi başka bir döngü deyimide olabilir. (while, do while, for deyimleri)

```
int main()
{
    int i = 0;
    int k = 0;

    while (i < 10) {
        while (k < 10) {
            printf("%d %d", i, k);
            ++k;
        }
    }
}
```

```

    }
    ++i;
    return 0;
}

```

İç içe döngülerde içerideki döngüde **break** deyimi kullanıldığında yalnızca içerideki döngüden çıkarılır, her iki döngüden birden çıkmak için goto deyimi kullanılmalıdır. (ileride göreceğiz)

**while** döngüsü bir bütün olarak tek deyim içinde ele alınır. Örnek:

```

while (1)
    while (1) {
        .....
        .....
        .....
    }

```

Burada ikinci **while** döngüsü tek bir kontrol deyimi olarak ele alınacağı için, bloklamaya gerek yoktur.

**while** döngüsünün yanlışlıkla boş deyim ile kapatılması çok sık yapılan bir hatadır.

```

int main()
{
    int i = 10;
    while (--i > 0);          /* burada bir boş deyim var */
    printf("%d\n", i);
    return 0;
}

```

Döngü while parantezi içerisindeki ifadenin değeri **0** olana kadar devam eder ve boş deyim döngü gövdesi olarak icra edilir. Döngüden çıktığında ekrana **0** basılır.

Sonlandırıcı ; **while** parantezinden sonra konulursa herhangi bir yazım hatası oluşmaz. Derleyici **while** döngüsünün gövdesinin yalnızca bir boş deyimden oluştuğu sonucunu çıkarır. Eğer bir yanlışlık sonucu değil de bilinçli olarak **while** döngüsünün gövdesinde boş deyim (null statement) bulunması isteniyorsa, okunabilirlik açısından, bu boş deyim **while** parantezinden hemen sonra değil, alt satırda ve bir tab içeriden yazılmalıdır.

## 5.8 Break deyimi

Break anahtar sözcüğü ile bir döngü sonlandırılabilir. Kullanımı

**break;**

şeklindedir. Programın akışı break anahtar sözcüğünü gördüğünde, döngü kırılarak döngünün akışı döngü gövdesi dışındaki ilk deyim ile devam eder. Yani koşulsuz olarak döngüden çıkarılır.

```
void main(void)
{
    int i;
    for(i=0;i<50;i++)
    {
        printf("%d ",i);
        if(i==15)
        break;
    }
}
```

Bu program ekrana 0-15 arası sayıları yazar. Break deyimini tüm C döngülerinde çalışır.

## 5.9 Continue deyimini

**continue** anahtar sözcüğü de tıpkı break anahtar sözcüğü gibi bir döngü içerisinde kullanılabilir. Programın akışı **continue** anahtar sözcüğüne geldiğinde sanki döngü yinelemesi bitmiş gibi yeni bir yinelemeye geçilir. Eğer **for** döngüsü içerisinde kullanılıyorsa yeni bir yinelemeye geçmeden önce döngünün 3. kısmı yapılır. Örnek :

```
int main()
{
    int i, k;
    char ch;
    for (i = 0; i < 100; ++i) {
        if (i % 3 == 0)
            continue;
        printf("%d\n", i);
    }
    return 0;
}
```

**break** anahtar sözcüğü bir döngüyü sonlandırmak için, **continue** anahtar sözcüğü de bir döngünün o anda içinde bulunan yinelemesini sonlandırmak için kullanılır.

**continue** anahtar sözcüğü özellikle, döngü içerisinde uzun **if** deyimlerini varsa, okunabilirliği artırmak amacıyla kullanılır.

```
for (i = 0; i < n; ++i) {
    ch = getch();
    if (!isspace(ch)) {
        ...
        ...
        ...
    }
}
```

Yukarıdaki kod parçasında döngü içinde, klavyeden getch fonksiyonu ile değer atanan ch değişkeni boşluk karakteri değilse, bir takım deyimlerin icrası istenmiş. Yukarıdaki durum **continue** anahtar sözcüğüyle daha okunabilir hale getirilebilir :

```
for (i = 0; i < n; ++i) {
    ch = getch();
    if (isspace(ch))
        continue;
    ...
}
```

n kere dönen for deyimi kalıpları

```
for (i = 0; i < n; ++i)
for (i = 1; i <= n; ++i)
for (i = n - 1; i >= 0; --i)
for (i = n; i > 0; --i)
```

Bir döngüden çıkmak için döngü değişkeni ile oynamak kötü bir tekniktir. Bu programları okunabilirlikten uzaklaştırır. Bunun yerine break anahtar sözcüğü ile döngülerden çıkılmalıdır.

## 5.10 Sonsuz döngülerden çıkış

1. **break** anahtar sözcüğü ile.

Bu durumda programın akışı döngü gövdesi dışındaki ilk deyim yönleneyecektir. (eğer iç içe döngü varsa break anahtar sözcüğü ile yalnızca içteki döngüden çıkılacaktır.)

2. **return** anahtar sözcüğü ile

bu durumda fonksiyonun (main de olabilir) icrası sona erecektir.

3. **goto** anahtar sözcüğüyle.

İç içe birden fazla döngü varsa en içteki döngü içinden en dıştaki döngünün dışına kadar çıkabiliriz. (goto anahtar sözcüğünün çok az sayıdaki faydalı kullanımından biri budur.)

4. **exit** fonksiyonu ile.

## 5.11 Switch deyimi

**if** deyimi birkaç seçenek arasından seçim yapmak için iyi bir deyimdir, fakat ortada birkaç seçenek var iken çok biçimsiz durmaktadır. **switch** deyimi birkaç if-else deyimi ile aynı anlamdadır. Genel hali şu şekildedir :

```
switch (degisken)
{
    case sabit1:
        deyim(ler);
        break;
    case sabit2:
```

```

        deyim(ler);
        break;
    case sabitN:
        deyim(ler);
        break;
    default:
        deyim(ler);
}

```

**Degisken**, integer veya karakter sabitlerinin listesi içinde sırayla test edilmektedir. Bir eş değere rastlandığı zaman, bu sabite ait deyimlerin bütünü **break**; sözcüğüne rastlanana kadar icra edilir. Eğer hiç bir eş değere rastlanamadıysa **default** bölümü altındaki deyimler icra edilir. **Default** bölümü isteğe bağlı olarak kullanılabilir. **Switch**'e bir örnek :

```

main()
{
    char ch;
    for(;;)
    {
        ch = getch();
        if(ch=='x')
            return 0;
        switch(ch)
        {
            case `0` :
                printf("Pazar\n");
                break;
            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;
            case `6` :
                printf("Cumartesi\n");
                break;
            default:
                printf("Geçersiz giriş\n");
        }
    }
}

```

Bu örnek 0 ve 6 arası sayıları okuyacak. Eğer girilen sayı bu sayı aralığı dışında ise, **Geçersiz giriş** mesajı görüntülenecektir. Girilen değer bu aralığın içinde ise, sayı haftanın günlerine dönüştürülecektir.

LCD ekranı üzerindeki satırlarda karakter sayısını tespit etmede kullanılan bir başka örnek te şu şekildedir. DIP swiç ve satırdaki karakter sayısı ayarları okunur

sonra diğer bitlerinden ayrıştırılır (maskeleme) ve bu bilgi ile fonksiyonun çağırıldığı yere uygun bir değer döndürülür.

```
byte cp1_sw_get()                //characters per line
{
    byte cp1;
    cp1=portd & 0b01110000;      //maskele
    switch(cp1)                  //şimdi decode edilmiş veriyi işle
    {
        case 0x00: cp1 = 8;    break;
        case 0x10: cp1 = 16;   break;
        case 0x20: cp1 = 20;   break;
        case 0x30: cp1 = 28;   break;
        default:   cp1 = 40;   break;
    }
    return(cp1);                 //çağırıldığın yere veri döndür
}
```

ANSI Standartlarındaki bir C derleyicisi en az 257 **case** deyimini desteklemelidir. Aynı **switch** içinde aynı iki tane **case** değeri olamaz. Ayrıca, **switch** deyimleri içiçe de olabilirler. Bir ANSI derleyicisi en az 15 içiçe **switch** deyimini desteklemelidir. Aşağıda içiçe bir **switch** örneği bulunmaktadır :

```
switch (a)
{
    case 1:
        switch (b)
        {
            case 0:
                printf("b yanlış");
                break;
            case 1:
                printf("b doğru");
                break;
        }
        break;
    case 2:
        .
        .
}
```

**break** sözcüğü **switch** deyimini içinde aynı zamanda isteğe bağlı kullanılır. Bu demek ki iki case deyimini kodun aynı parçasını paylaşabilirler. Bunu bir örnek üzerinde görelim :

```
void main(void)
{
    int a=6,b=3;
    char ch;
    printf("T = Toplama\n");
    printf("C = Çıkarma\n");
    printf("P = Çarpma\n");
    printf("B = Bölme\n");
    printf("Seçiminizi giriniz:\n");
    ch=getch();
    switch (ch)
    {
        case 'C':                // *
```

```

        b=-b;
    case 'T' :
        printf("\t\t%d", a+b);
        break;
    case 'P' :
        printf("\t\t%d", a*b);
        break;
    case 'B' :
        printf("\t\t%d", a/b);
        break;
    default:
        printf("\t\tNe dedin?");
}
}

```

Yukarıdaki programcıkta \* işaretli kısımda **break** sözcüğü olmadığı için programda **çıkarma** seçildiğinde, program **b=-b** yaptıktan sonra bir alttaki **case** bölümü olan **toplama** bölümüne geçecektir.

## 5.12 Null deyimi

**null** deyimi sadece bir noktalı virgülden (;) oluşur. Bir ifadenin bulunabileceği her hangi bir yerde bulunabilir. Assembly programlarında bir komut çevrimi beklemeye neden olan "**nop**" deyimi gibi null ifadesi işletildiğinde hiç bir işlem yapılmaz.

**do**, **for**, **if** ve **while** ifadelerin, ifade gövdelerinde, herhangi bir işletilebilir ifadenin bulunması gerekir. **null** ifadesi bu durumlarda gerekli yazımı gerçekleştirir.

```

    for (i=0;i<10;i++)
    ;

```

Bu örnekte **for line[j++]=0** döngü ifadesi line dizisinin 10 elemanını sıfırlar. İfade gövdesi null olduğundan ilave komutlara ihtiyaç yoktur.

## 5.13 return deyimi

**return** ifadesi bir fonksiyonun çalışmasını sonlandırır ve program en son fonksiyonun çağırıldığı yere dönerek çalışmasına devam eder. Fonksiyonlardan bir değer döndürülebilir. Değer döndürülmeyecekse fonksiyon prototipinde fonksiyonun döndüreceği değer **void** olarak tanımlanır.

```

GetValue(c)
int c;
{
    c++;
    return c;
}

void GetNothing(c)
int c;
{

```

```

c++;
return;
}

main()
{
int x;
x = GetValue();
GetNothing();
}

```

## 6. Diziler ve harf dizileri

Bu bölümde dizileri ve harf dizilerinden bahsedeceğiz. Bir dizi basitçe aynı veri tipine sahip birbiriyle ilişkili değişkenler listesidir. Bir harf dizisi sıfırla bitirilen ve en çok rastlanılan bir karakter dizisidir.

Bu konudaki başlıklar:

- Diziler**
- Harf dizileri**
- Tek boyutlu diziler**
- Çok boyutlu diziler**
- Hazırlama**

şekindedir.

### 6.1 Tek boyutlu diziler

Dizi hepsi aynı isimle adlandırılan ve aynı tipde olan değişkenler listesidir. Dizideki bağımsız değişken dizi elemanı diye adlandırılır. Bu ilişkilendirilmiş veri gruplarını işlemek için basit bir yoldur.

Tek boyutlu diziyi tanımlamanın genel formu aşağıdaki gibidir.

```
type değişken_ismi [adet];
```

Type geçerli bir C verisidir, değişken\_ismi dizinin ismi ve adette dizide kaç eleman olacağını belirtir. Örneğin 50 elemanı olan bir dizi aşağıdaki gibi ifade edilmelidir.

```
int yükseklik [50] ;
```

Bir dizi belirtildiği zaman, C ilk elemanın 0 inci indiste (sırada) olduğunu varsayar. Eğer dizinin 50 elemanı varsa en sondaki elemanın indisi 49 dur. Yukarıdaki örneği kullanarak diyelim ki yükseklik dizisinin 25 inci elemanına 60 değeri atanmıştır. Aşağıdaki örnekte bunun nasıl yapılacağı gösterilmiştir.

```
yükseklik[24] = 60 ;
```

C tek boyutlu dizileri boşluksuz olarak hafıza konumlarına yerleştirir. İlk eleman en düşük adrestedir. Aşağıdaki kodu çalıştırdığımız zaman ..

```
int num[10];
int i;
for (i=0;i<10;i++)
    num[i] = i;
```

dizinin hafızadaki görüntüsü aşağıdaki gibi olacaktır.

Eleman	1	2	3	4	5	6	7	8	9	10
	0	1	2	3	4	5	6	7	8	9

Herhangi bir dizi elemanı programda değişken yada sabit değer olarak herhangi bir yerde kullanılabilir. Aşağıdaki diğer örnekte basitçe dizi elemanına indisin karesini atamakta, sonrada bu elemanları yazmaktadır.

```
#include <16c74.h>
void main ( void)
{
    int num[10];
    int i;
    for ( i = 0 ; i<10 ; i++)
        num[i] = i * i ;
    for ( i = 0 ; i<10 ; i++)
        printf ( "%d " , num[i] );
}
```

Bir dizide 10 eleman olup kazayla 11 inci elemanına bir değer atadığınızda ne olabilir? C dili dizi indis limitlerini kontrol etmez. Bu sebepten dolayı dizide belirtilmeyen bir elemanı okur ya da yazabilirsiniz. Bu bir şekilde beklenmeyen sonuçlar doğurabilir. Sık sık bu olay programın çökmesine ve bazen de bilgisayarın çökmesinede sebep verebilir. C dili bir dizinin değerini başka bir dizine aşağıdaki şekil ile atama yapılmasına izin vermez.

```
int a [ 10 ] , b [ 10 ] ;
.
.
a = b ;
```

Yukarıdaki örnek yanlıştır. Bir dizinin içeriğini öteki diziye kopyalamak isterseniz , her bağımsız elemanı birinci diziden ikinci diziye kopyalamanız gerekmektedir. Aşağıdaki örnek içinde 20 eleman olan a[] dizisinin b[] dizisine nasıl kopyalanacağını gösterir.

```
for ( i = 0 ; i < 20 ; i++)
    b[i] = a[i];
```

### ALİŞTİRMA:

1. Aşağıdaki kod parçasında ne yanlıştır?

```

int i;
char count [10];
for(i = 0;i < 100;i++)
    count[i] = getch ( ) ;

```

2. getch() fonksiyonunu kullanarak klavyeden girilen ilk 10 karakteri okuyabilecek bir program yazın. Bu program bu karakterlerden uyan çıktığı zaman bir uyarıda bulunsun.

## 6.2Harf dizileri

En çok rastlanılan tek boyutlu diziler harf dizileridir. C dilinde harf dizisi ile ilgili bir veri şekli yoktur. Onun yerine tek boyutlu karakter dizilerini kullanır. Bir harf dizini 0 (sıfır) olarak gösterilir. Eğer her harf dizini sıfır ile sonlandırılacaksa, diziyi bir fazlası eleman ile tanımlamak gerekmektedir. Bu fazla eleman içeriği sıfır olacaktır. C dilindeki bütün harf dizi sabitleri otomatik olarak sıfır ile sonlandırılır.

Klavyeyi kullanarak programdaki bir harf dizisine nasıl giriş yapabiliriz? gets(str) fonksiyonu satır başı düğmesine basılana kadar klavyeden basılan karakterleri okur. Okunan karakter dizisi tanımlanan str dizisine kaydedilir. Str dizisinin uzunluğunun girilen karakter sayısından dizinin sıfırla sonlandırılacağından en az bir fazla olması gerekmektedir.

Gets() fonksiyonun nasıl kullanılacağını bir örnekle gösterelim.

```

void main ( void )
{
    char star[80] ;
    int i ;
    printf("Bir cumle gir ( < 80 karakter) :\n");
    gets(str);
    for(i=0;str[i];i++)
        printf ("%c",str[i]);
    printf("\n%s", str);
}

```

Burda harf dizisinin nasıl basılacağını iki şekilde görmekteyiz : %c yi kullanarak karakter dizisi şeklinde yada %s yi kullanarak harf dizisi şeklinde.

### ALIŞTIRMA:

1.Bu programdaki hata nerde? Strcpy() fonksiyonu ikinci argümenti birinciye kopyalıyor.

```

#include <string.h>
void main ( void )
{
    char str [16] ;
    strcpy (str,"PicProje nerede?");
    printf (str) ;
}

```

2. Ekrandaki harfleri okuyan vede bunları ekrana ters sırada yazabilen bir

program yazın.

### 6.3 Çok boyutlu diziler

C dili tek boyutlu diziler ile sınırlı değildir. İsterseniz iki yada daha fazla boyutta dizi yaratabilirsiniz. Örneğin, 5x5 elemanı olan integer dizisi yaratmak için, kullanmamız gereken şekil aşağıdaki gibidir:

```
int number[5][5]; //25 hafıza yeri
```

Ek boyutları basitçe köşeli parantez setini ekliyerek çoğaltabiliriz. Biz burda basitlik olsun diye iki boyutlu dizileri konuşacağız. İki boyutlu dizi en iyi satır sütun şeklinde anlatılabilir. Bunun için iki boyutlu diziler soldan sağa kısa zamanda ulaşılabilir. İzleyen örnekte 5x5 dizinin grafiksel gösterimini gösterir. İki boyutlu diziler tek boyutlu diziler gibi kullanılabilir. Örneğin aşağıdaki program 5x4 lük diziyi indislerin çarpımı ile doldurur sonrada dizi içeriğini sütun satır şeklinde gösterir.

```
void main ( void )
{
    int array [5] [4] ;
    int i , j ;

    for ( i = 0 ; i < 5 ; i++ )
        for ( j = 0 ; j < 4 ; j++ )
            array [i] [j] = i * j ;
    for ( i = 0 ; i < 5 ; i++)
    {
        for ( j = 0 ; j < 4 ; j++ )
            printf ("%d " , array [i] [j] ) ;
        printf ("\n");
    }
}
```

Programın çıktısı bu şekilde olmalıdır:

```
0 0 0 0
0 1 2 3
0 2 4 6
0 3 6 9
0 4 8 12
```

Gördüğümüz gibi, çok boyutlu dizileri kullandığınızda bağımsız değişkenlerin sayısı artmaktadır. Günümüzde kullanılan PIC MCU ların hafızaları oldukça geniştir.

#### ALİŞTİRMA:

1. 3x3x3 lük bir diziyi tanımlayan vede 1 den 27 e kadar olan sayıları bu diziyi atayan bir program yazın. Ekrana da çıktıyı yazdırın.

2. Birinci programı kullanarak her sütunun vede satırın toplamının çıktısını alın.

## 6.4 Dizileri hazırlama

Şimdiye kadar bağımsız dizi elemanlarına değerler atanmış olarak gördük. C dili dizilere bir değişkenmiş gibi değer atanmasına imkan verecek metoda sahiptir. Tek boyutlu dizilerin genel şekli aşağıdaki gibidir.

```
type dizi_ismi[adet] = {değer_listesi};
```

Değer\_listesi dizi tipine uygun virgüllerle ayrılmış sabitler listesidir. İlk sabit ilk elemanın yerine, ikinci sabit ikinci elemanın yerine ve sırayla bütün sabitler elemanların yerine yerleştirilir. İzleyen örnekte 5 elemanlı integer dizisinin hazırlanmasını göstermektedir.

```
int i[5]={1,2,3,4,5};
```

**i[0]** tanımlı dizi elemanın değeri **1** ve **i[4]** tanımlı elemanın değeri **5** olacaktır. Harf dizisi (karakter dizisi) iki türlü hazırlanabilir. Birincisi birbirinden bağımsız karakter listesi yapmak:

```
char str[3]={'a','b','c'};
```

İkinci tür ise ekteki örnekte gördüğünüz gibi tırnak içinde harf dizisi yaratmaktır.

```
char name[5]="Veli";
```

Kuyruklu parantezlerle harf dizisini kapatmadığımız dikkatinizi çekmiştir. Bu tip hazırlamada bunlar kullanılmaz çünkü C dilinde harf dizileri sıfırla bitmek zorundadır. Derleyici "Veli" nin sonuna otomatikman bir sıfır ekler.

Çok boyutlu diziler tek boyutlu diziler gibi hazırlanır. İki boyutlu dizileri hazırlarken sütun satır şeklinde hazırlanması daha kolaydır. İzleyen örnekte 3x3 dizinin hazırlanmasını gösterelim.

```
int num[3][3]={1,2,3,
               4,5,6,
               7,8,9};
```

### ALİŞTİRMA:

1. Aşağıdaki tanımlama doğrumudur?

```
int count[3]=10,0,5.6,15.7;
```

2. Bir sayının karesini ve kübünü belirten bir tablosu olan program yazın. Her satırda sayı, sayının karesi ve sayının küpü olmalıdır. Sayılarla ilgili bilgileri içeren 9X3 bir dizi yaratın.

`scanf ("%d", &num)` ; rutinini kullanarak kulanıcıdan sayı girmesini isteyin. Sonrada bu sayının, karesinin ve kübünün çıktısını alın.

## 6.5 Katar ( harf dizileri ) dizileri

Katar dizinleri C dilinde çok rastlanır. Öteki diziler gibi tanımlanır ve hazırlanır. Kullanma şekilleri diğer dizilere göre biraz daha değişiktir. Örneğin, izleyen tanımlama ne ifade eder?

```
char isim[10][40];
```

Bu durum isim dizisinin 10 adet ayrı 40 karakter uzunluğunda (sıfır dahil) ismi içerdiğini belirtir. Bu tablodan bir harf dizisine ulaşmak istediğinizde, ilk indisi belirtmelisiniz. Örneğin bu dizinin beşinci isminden çıktı almak için, izleyen şekli kullanmalısınız.

```
printf("%s", isimler[4]);
```

İki boyutdan büyük diziler içinde aynısını yapmalısınız. Örneğin hayvanlar dizisi aşağıdaki gibi tanımlandırılmalıdır.

```
char hayvanlar[5][4][80];
```

Belirtilmiş bir harf dizinine ulaşmak için ilk iki boyutu kullanmanız gerekmektedir. Mesela üçüncü listedeki ikinci harf dizinine erişmek için, hayvanlar [2][1] yazılması gerekir.

### Alıştırma:

1. 0 dan 9 kadar olan sayıların isimlerini içeren harf dizini tablosunu yazan bir program yazın. Kullanıcıya tek bir rakkam girmesine izin verin ve proramınız ilgili ismi gösterecek. Tablodaki indise ulaşmak için girilen karakterden '0' ı çıkartın.

## 6.6 Harf dizini fonksiyonları

Harf dizinleri program içerisinde bir çok yolla değiştirilebilir. Buna bir örnek strcpy komutu ile kaynak dizinin hedef dizine kopyalanmasıdır ki bu da sabit harf dizinini çalışabilir hafızaya aktarır.

```
#include <string.h> // harf dizini fonksiyonları için kütüphane
char string[8]; // harf dizinini tanımlama
strcpy (string, "Merhaba") ; // Karakterleri dizine kopyalama
```

PIC MCU larda sabit hafıza işaretçilerin geçerli olmadığını dikkat edin. Sabit harf dizinlerini strlen("hi") gibi fonksiyonlara aktaramazsınız.

Örnekler:

```
char s1[10], s2[10];

strcpy(s1, "abc");
```

```

strcpy(s2, "def");
strcat(s1, s2);
printf("%u", strlen(s1)); // çıkan çıktı 6 dir
printf(s1); // çıkan çıktı abcdef dir

if (strcmp(s1, s2) != 0) printf(" Aynıısı değildir");

```

Diğer harf dizin fonksiyonları aşağıdadır:

<b>strcat</b>	İki harf dizisini birleştirir
<b>strchr</b>	İlk karaktere bakar
<b>strrchr</b>	Son karaktere bakar
<b>strcmp</b>	İki harf dizisini karşılaştırır
<b>strncmp</b>	İki dizindeki karakter sayısını karşılaştırır
<b>stricmp</b>	Büyük küçük harflere dikkat etmeden iki dizini karşılaştırır
<b>strncpy</b>	Bir dizindeki belli bir sayıdaki karakterleri öteki dizine kopyalar
<b>strlen</b>	Bir dizinin uzunluğunu verir
<b>strlwr</b>	Büyük harfleri küçük harflerle değiştirir
<b>strpbrk</b>	İki dizideki ilk uyan karakteri bulur
<b>strstr</b>	Bir harf dizisindeki ilk belirtilen karakter sırasını bulur

Değiştirilecek harf dizisindeki büyüklüklerin uymasına dikkat edin.

## 7. İşaretçiler

Bu bölüm, C dilinin en önemli ve bir o kadar da kafa karıştırıcı özelliği olan işaretçiler hakkında bilgi içermektedir. Temel olarak işaretçi, bir nesnenin adresidir.

Bu bölümde inceleyeceğimiz bazı başlıklar;

**İşaretçi temelleri,  
İşaretçiler ve diziler,  
İşaretçileri Fonksiyonlarda kullanmak**

### 7.1 İşaretçilere giriş

Bir işaretçi, bir hafıza bölgesinin adresini içerisinde barındıran bir hafıza bölgesi değişkenidir.

Örneğin; a değişkeni b değişkeninin adresini barındırıyorsa; a, b'yi gösteriyor (işaret ediyor) demektir. Eğer b değişkeninin hafızadaki bulunduğu adres 100 ise, a değişkeninin değeri 100 olacaktır.

Bir işaretçi değişkeni tanımlamanın genel formatı şu şekildedir:

```
Tip *degisken_adi;
```

Tip, C için geçerli olan bir veri tipidir. Söz konusu değişken adının göstereceği değişkenlerin veri tipini belirler. Değişken adının başında bulunan \* işareti ise tanımlanan bu değişkenin bir işaretçi tipi değişken olduğunu belirtir.

Örneğin şu ifade bir tamsayı(integer) tipi işaretçi değişkeni tanımlar:

```
int *ptr;
```

İşaretçiler ile ilgili iki özel karakter vardır: \* ve & . Bir nesnenin adresi & operatörü ile elde edilebilir. \* operatörü ile ise bir hafıza bölgesinde bulunan değeri getirir.

Örneğin;

```
#include <16c74.h>
void main(void)
{
    int *a,b;
    b=6;
    a=&b;
    printf ("%d", *a);
}
```

Not: Varsayılan olarak derleyici bir işaretçi için 1 byte kullanır. Bu, sadece 0-255 arası bölgelerin gösterilebileceği (işaret edilebileceği) anlamına gelir. Daha büyük hafıza adresleyebilmek için 2 byte'lık işaretçi (16 bit) kullanmak gerekir. 16 Bit işaretçi kullanabilmek için şu komutu kullanmalısınız:

```
#device *=16
```

Ancak 16 bit aritmetik daha fazla işlem gerektirdiğinden daha çok program kodu oluşturulacağını göz önünde bulundurmalısınız.

İlk komut 2 değişken tanımlar: integer tipi pointer türündeki a değişkeni ve integer tipinde b değişkeni. Bir sonraki satır b değişkenine 6 değerini yükler. Sonraki satırda ise b değişkeninin adresi (&b) a değişkenine (işaretçisine) yüklenir. Bu satır şu şekilde okunabilir: a değişkenine b değişkeninin adresini yükle.

Son olarak da b değişkeninin değeri \*a komutuyla ekranda görüntülenir. Bu satır şu anlama gelmektedir: a işaretçisinde bulunan adresteki değeri yaz. Bu gibi bir değerden referans alınan bilgilere erişme işlemine indirection denilir. Bir görsel gösterim şekli şöyle olabilir:

Adres:	100	102	104	106
değişken:	i	j	k	ptr
içerik:	3	5	-1	102

```
int i, j, k;
int *ptr;
```

Başlangıçta, i değeri 3 ve &i değeri 100 dür (i değişkeninin hafızadaki yeri). ptr işaretçi değişkeni 102 değeri barındırdığından \*ptr 102 nolu adresteki değer olan 5 e işaret edecektir yani \*ptr 5 olarak değerlendirilecektir.

NOT: PICMicro'larda her banktan kolayca ulařılabilen (0x04 adresinde) `fsr` ve (0x04 adresinde) `indf` isminde iki kayıtçı bulunur. `Indf` kayıtçısı `fsr` kayıtçısı ierisine kaydedilmiř 8 bitlik veriyi adres kabul eder ve o adresteki deęeri barındırır. `Indf` kayıtçısının ierięi deęiřtirilirse doęrudan `fsr` ierindeki adreste bulunan deęer deęiřtirilmiř olur. 18FXXX serisi PICMicro'larda bu `fsr` ve `indf` iřlevini goren olduka yetenekli sanal kayıtçı gruplarından  grup bulunur.

Bir hafıza blgesinde iřareti kullanarak deęer yazmak ta mmkndr. Mesela bir nceki programı byle bir iřlem yapabilecek řekilde bir daha yapılandıralım:

```
#include <16c74.h>
void main(void)
{
    int *a,b;
    a=&b;
    *a=6;
    printf("%d",b);
}
```

Bu programda, nce `a` iřaretisine `b` deęiřkeninin adresini yklyoruz. Sonra `*a=6` diyerek `a` iřaretisinde bulunan adrese 6 deęerini yaz diyoruz. Bu son iki programdaki iřaretiler gereksiz yere kullanılmıřlardır. Burada sadece iřareti kullanımına rnek olması iin verilmiřlerdir.

### Alıřtırma:

FOR dngsyle 0'dan 9'a kadar sayan ve ekrana sayıları iřareti kullanarak gsteren bir program yazın.

## 7.2. İřaretilerin kullanımıyla ilgili kısıtlamalar

Genel olarak iřareti tipi deęiřkenler dięer tiplerdeki deęiřkenler gibi iřlem grebilir. Ancak iřaretilere (pointerlara) zg bir ka zel durum ve kısıtlama vardır. `*` ve `&` operatrlerine ilaveten pointerlar ile birlikte sadece řu drt operatr kullanabilirsiniz: `+`, `++`, `-`, `--`. Ayrıca iřaretilerden yalnızca tamsayı (integer) tipi deęerleri ıkarabilir veya ekleyebilirsiniz.

Bir pointer deęeri arttırıldıęında, bir sonraki hafıza birimini gsterir. Mesela `P` pointerinde 100 numaralı hafıza biriminin adres deęeri olduęunu dřnelim. İřareti `p` eęer **int16** tipinde tanımlanmıř ise **p++** ifadesinden sonra `p` pointerının deęeri 102 olacaktır. (**int16** tipi veriler hafızada 2 byte yer kapladıęından dolayı). Eęer `p` bir gerek sayı (reel sayı, float) olsaydı bu durumda `p` pointerının deęeri 104 olacaktı (nk float tipi veriler 4 byte uzunluęundadır).

Yalnızca **char** tipi veriler iin beklenen sonu gelir. Char tipi veriler hafızada 1 byte kapladıęından, bařlangıta deęeri 100 olan `p` iřaretisinin yeni deęeri `p++` iřleminden sonra 101 olacaktır.

Bir iřaretiden/ile diledięiniz tamsayı deęeri ıkartabilir/toplayabilirsiniz.

Mesela;

```
int *p;  
.   
p=p+200;
```

p=p+200 ifadesi ile P işaretçisi eski değerinden 200 sonraki hafıza biriminin adresiyle yüklenmiştir. Hem işaretçinin p kendisini, hem de gösterdiği adresteki değeri arttırmak veya azaltmak mümkündür.

Bir işaretçi tarafından gösterilen bir değeri arttırırken veya azaltırken dikkatli olmalısınız. Örneğin ptr değeri önce 1 iken aşağıdaki satırdan sonra ne olur?

```
*p++;
```

bu sadece p işaretçisinde kayıtlı adresin değerini (o adreste saklanan verinin değil) 1 arttırır. Ancak eğer biz işaretçideki adreste bulunan bilgiyi bir arttırmak istiyorsak şöyle yazmalıyız:

```
(*p)++;
```

Parantez sayesinde işaretçiyi değil de işaretçi içinde bulunan adresteki değeri arttırmak istediğimizi belirtmiş oluruz.

İşaretçiler ilişkisel işlemlerde de kullanılabilir. Ancak eğer işaretçiler birbirleri ile bir ilişkidelerse bu ancak anlamlı olur aksi takdirde hiç bir mana ifade etmez.

18FXXX mikrodenetleyicilerinden önceki PIC Mikrodenetleyicileri için işaretçiler program alanına (ROM'da) oluşturulamazlar.

```
char const name[5] = "JOHN";  
ptr=&name[0];
```

bu program, const kullanılmadan (yani dizi ROM içinde tanımlanmadığında) geçerli bir koddur. 18FXXX ailesinde donanıma eklenmiş tablo okuma yazma özellikleri ile program alanına işaretçi atamak mümkün hale getirilmiştir.

### **Alıştırma:**

Şu değişkenleri tanımlayın ve adreslerini bir pointer tipi değişkene yükleyin. Her bir pointer değişkeninin değerini %p kullanarak yazdırın. Sonrasında her bir pointeri bir arttırın ve tekrar yazdırın. Her bir veri tipinin uzunluğu ne kadardır sizin makinenizde?

```
char *cp, ch;  
int *ip, i;  
float *fp, f;  
double *dp, d;
```

2. Aşağıdaki anlatımda yanlışlık nerededir?

```
int *p, i;  
p = &i;  
p = p/2;
```

### 7.3 Pointerlar ve diziler

C dilinde, işaretçiler ve diziler birbirleri ile çok iç içedir. Bu içiçelik, C dilinin gücünü daha da ön plana çıkarmaktadır.

Bir diziyi indeksiz olarak kullanırsanız, kesinlikle diziye erişmek için bir işaretçi kullanıyorsunuz demektir. Geçen bölümde bir metnin sadece ismini içeren **gets()** fonksiyonunu kullanmıştık.

**Önemli not:** Bir dizi bir fonksiyona geçtiğinde yalnızca ilk elemanı gösteren bir işaretçi kullanılabilir. Fonksiyona dizi gönderirken 7.2'nin en sonunda yapılan açıklamayı hatırlayın.

İndekse sahip olmayan bir dizi adı işaretçi olduğundan, o değeri bir başka işaretçiye yükleyebilirsiniz. Bu, sizin diziye işaretçi aritmetiği ile erişmenizi sağlar.

Örneğin:

```
int a[5]={1,2,3,4,5};

void main(void)
{
    int *p,i;
    p=a;
    for(i=0;i<5;i++)
        printf("%d",*(p+i));
}
```

Bu, kusursuz bir C programıdır.

**printf()** komutunda **\*(p+i)** kullandığımızı fark etmişsinizdir. **i** dizinin indeksidir. Belki şaşıracaksınız ama bir işaretçiyi de sanki bir diziymiş gibi indeksleyebilirsiniz. Şu program da geçerli bir programdır:

```
int a[5]={1,2,3,4,5};
void main(void)
{
    int *p,i;
    p=a;
    for(i=0;i<5;i++)
        printf("%d",p[i]);
}
```

Hatırlanması gereken bir nokta vardır ki: bir işaretçi, ancak bir diziyi gösterdiğinde indekslenebilir. Dizileri gösteren işaretçiler sadece ilk elemanın yerini gösterdiğinden, işaretçiyi p++ ile arttırmak geçersiz bir işlem olacaktır. Dolayısıyla önceki programda kullanılacak bir p++; ifadesi geçersiz olacaktır. 18FXXX mikrodenetleyicilerinde p[i] değerini otomatik getiren

İşaretçiler ile dizileri karıştırmak bazı saçma sonuçlara sebep olabilir. Aşağıdaki örnekte sorunu göreceksiniz. İkinci programda problem düzeltilmiştir.

```

int *p;

int array[8];
p=array;

    0007: MOVLW 0F ;dizinin başlangıcını yükle
    0008: MOVWF 0E ;pointer

*p=3;

    0009: MOVF 0E,W ;
    000A: MOVWF 04 ;dolaylı registerı göster
    000B: MOVLW 03 ;3 yükle
    000C: MOVWF 00 ;belirtilen noktaya kayıt et

array[1]=4;

    000D: MOVLW 04 ;4 yükle
    000E: MOVWF 10 ;dizinin başlangıcına yükle

int *p;
int array[8];
p=array;

    0007: MOVLW 0F ;dizinin başlangıcını yükle
    0008: MOVWF 0E ;pointer

p[1]=3;

    0009: MOVLW 01 ;dizinin pozisyonunu yükle
    000A: ADDWF 0E,W ;Dizinin başlangıç pozisyonuna ekle
    000B: MOVWF 04 ;dizi pointerinin içine yükle
    000C: MOVLW 03 ;üç'e yükle
    000D: MOVWF 00 ;belirtilen lokasyona sakla

*(array+1) = 4;

    000E: MOVLW 10 ;dizi pozisyonuna yükle
    000F: MOVWF 04 ;ona işaret et
    0010: MOVLW 04 ;4 yükle
    0011: MOVWF 00 ;Belirtilen lokasyona kayıt et

```

### ALİŞTİRMA:

1. Bu bölümdeki kod yazılımı doğru mudur ?

```

int count[10];
.
count = count+2;

```

2. Bu segmentteki kod hangi değeri gösterir ?

```

int value[5]=(5,10,15,20,25);
int *p;
p = value;
printf("%d", *p+3);

```

## 7.4 İşaretçileri Fonksiyonlarda Kullanmak

3. Bölümde fonksiyonları çağırarak ilgili olarak kullanılabilen iki yöntemden bahsettik: “Değerle çağırma” ve “referansla çağırma”. Birinci yöntemi o bölümde anlatmıştık.

İkinci yöntemde işlenenlerin adresi fonksiyona taşınır; bir başka deyişle işaretçi fonksiyondan geçer. Bu noktada, işaretçi kullanarak değişken üzerinde yapılan herhangi bir değişiklik, çağırma rutinindeki değişkenin değerini değiştirir. İşaretçiler, tıpkı diğer değişkenler gibi fonksiyonlarda geçer. Aşağıdaki örnek bir harf dizisinin (string) işaretçi kullanan bir fonksiyona nasıl geçtiğini gösterir.

```
#include <16c74.h>
void puts(char *p);
void main(void)
{
    puts("Microchip is great!");
}
void puts(char *p)
{
    while(*p)
    {
        printf("%c", *p);
        p++;
    }
    printf("\n");
}
```

Bu örnekte, p işaretçisi harf dizisinin ilk karakteri olan “M” ye işaret eder. **While (\*p)** ifadesi de dizinin sonundaki null’u kontrol eder. Döngünün her bir aşamasında, işaretçi tarafından işaret edilen karakter yazdırılır ve sonra dizideki bir sonraki karakteri göstermesi için p bir arttırılır. İşaretçinin fonksiyona geçmesiyle ilgili bir başka örnek de:

```
void IncBy10(int *n)
{
    *n += 10;
}
void main(void)
{
    int i=0;
    IncBy10(i);
}
```

Yukarıdaki örnek, bir referans parametre tarafından çağırılan özel tipteki bir işaretçi parametresinin kullanılmasıyla daha okunur bir şekilde yeniden yazılabilir. Örnek:

```
void Incby10(int &n)
{
    n += 10;
}
void main(void)
```

```
{
    int i=0;
    Incby10(i);
}
```

Yukarıdaki her iki örnek de bir fonksiyondan parametre listesiyle nasıl değer döndürüleceğini gösterir.

### **Alıştırma:**

1. Bir float sayıyı fonksiyona geçiren bir program yazın. Fonksiyonun içerisinde fonksiyon parametresi olarak -1 atansın. Fonksiyon main'e döndükten sonra float değişkenin değerini yazın.

2. fl işaretçisini bir fonksiyona geçiren bir program yazın. Fonksiyonun içerisinde değişkene -1 değerini atayın. Fonksiyon main'e döndükten sonra float değişkenin değerini yazsın.

## **8.Yapılar ve Birlikler**

Yapılar ve birleşimler C dilinin en önemli kullanıcı tanımlı tiplerini temsil ederler. Yapılar değişik veri tiplerine sahip olabilen birbirleriyle ilişkili değişken gruplarıdır. Birlikler aynı hafıza alanını paylaşan değişkenler grubudur.

Bu bölümde ele alacağımız konular:

**Yapı esasları**  
**İşaretçilerden Yapılara**  
**İç içe geçmiş Yapılar**  
**Birlik Esasları**  
**İşaretçilerden Birlikler**

### **8.1 Yapılara Giriş**

Bir yapı, ortak bir ad üzerinden erişilebilen ve birbirleriyle ilişkili olan parçaların grubudur. Yapının içinde bulunan her parça birbirlerinden farklı olabilen veri tiplerine sahiptir.

C Yapıları aşağıdaki şekilde tanımlanır:

```
struct etiket_adi
{
    type element1;
    type element2;
    .
    type elementn;
} değişken_listesi;
```

“**Struct**” sözcüğü derleyiciye bir yapının tanımlanacağını belirtir. Yapı içindeki her tip bir veri tipidir. Bu tiplerin birbirleri ile aynı olması gerekmez. **etiket\_adi** yapının ismidir. **değişken\_listesi** opsiyoneldir. Yapı içindeki parçalardan genelde alanlar veya üyeler olarak bahsedilir. Biz parçalardan üye olarak bahsedeceğiz.

Genel olarak yapı içindeki bilgiler mantıksal olarak birbirleriyle ilişkilidirler. Örnek olarak; Bir yapıyı tüm müşterilerinizin isim, adres ve telefon bilgilerinizi tutması için kullanabilirsiniz.

Aşağıdaki örnek bir kütüphane takip kartı içindir.

```
struct katalog
{
    char yazar[40];
    char baslik[40];
    char yayinci[40];
    unsigned int tarih;
    unsigned char rev;
} kart;
```

Bu örnekte yapımızın adı **katalog**'dur. **katalog** herhangi bir değişkenin ismi olmayıp yalnızca yapımızın ismidir. **kart** değişkeni, **katalog** tipinin yapısı olarak bildirilmiştir. Yapının her hangi bir üyesine erişim için, değişkenin ismini ve üyenin ismini belirtmeniz gerekir. Bu isimler nokta ile bir birlerinden ayrılmıştır. Örnek olarak **katalog** yapısı içindeki revision üyesine erişmek için, **kart.rev='a'** kullanmalısınız. Burada **kart** değişken ismi ve **rev**'de üyesidir. Yapının üyelerine erişim için operatör kullanılmıştır. **katalog** yapısının **yazar** üyesini yazdırmak için şunu yazmalısınız;

```
printf("Yazar is %s\n",kart.yazar);
```

Şimdi biz bir yapının nasıl tanımlanacağını, bildirileceğini ve erişileceğini biliyoruz. Peki, katalog yapımızın hafızayı kullanım şekli nasıldır?

<b>yazar</b>	40 byte
<b>baslik</b>	40 byte
<b>yayinci</b>	40 byte
<b>tarih</b>	2 byte
<b>rev</b>	1 byte

Eğer **kart** yapısının **tarih** üyesinin adresini almak istiyorsak **&kart.tarih** ifadesini kullanmalıyız. Eğer **yayinci**'nin ismini yazdırmak istersek

```
printf("%s",kart.yayinci);
```

ifadesini kullanmalıyız. Eğer **baslik** dizisinin 3. elemanı gibi belirli bir elemana ulaşmak istiyorsanız ne yapmalısınız? Tabi ki **kart.baslik[2]** kullanmalısınız.

Baslığın birinci elemanı '0' , ikinci elemanı '1' ve üçüncüsü '2'.

Her hangi bir yapı tanımlamış iseniz , program içerisinde her hangi bir yerde daha fazla yapı değişkeni oluşturabilirsiniz. Bunun için;

```
struct yapı_ismi değişken_listesi;
```

Mesela, katalog yapısı daha önce program içerisinde tanımlanmış ise , iki adet değişkeni daha şu şekilde tanımlayabilirsiniz;

```
struct katalog book,list;
```

C dili size diğer veri tipleri gibi yapıları da bir dizi şeklinde bildirmenize izin verir. Örneğimiz katalog yapısının 50 elemanlı bir dizisini bildirmektedir.

```
struct katalog big[50];
```

Eğer dizi içindeki yapılara erişmek istiyorsanız, yapı değişkenini indekslemelisiniz. Örneğin bir yapı dizisi olan **big**'in 10. elemanının **baslik** üyesine nasıl erişirdiniz? Tabi ki **big[9].baslik** ifadesi ile.

Yapılar fonksiyonlara aktarılabilir. Bir fonksiyon herhangi bir veri tipini bize geri döndürdüğü gibi bir yapıyı da geri döndürebilir. Bir yapının değerlerini diğer bir yapıya basit bir atamayla atayabilirsiniz. Aşağıda bulunan kod parçası

```
struct temp
{
    int a;
    float b;
    char c;
} var1,var2;

var1.a=37;
var1.b=53.65;
var2 = var1;
```

Bu kod parçası yapıyı uyguladığında **var2** değişkeni **var1** değişkeni ile aynı içeriğe sahip olacaktır.

Aşağıda bir yapıya ilk değer verilmesine dair bir örnek verilmektedir:

```
struct ornek
{
    char kim[50];
    char karakter;
    int i;
} var1[2]={ "Rodger", 'Y', 27, "Jack", 'N', 30};
```

Not alınması gereken önemli bir nokta: Bir yapıyı fonksiyona aktardığınızda, tüm yapı "bir değer tarafından çağırılma" metodu ile aktarılır. Bu nedenle fonksiyon içindeki yapıda yapılacak modifikasyonlar çağırma rutini içinde bulunan yapının değerini etkilemeyecektir. Yapı içindeki elemanların sayısı yapının fonksiyona aktarılış biçiminden etkilenmezler.

Bunun Pic'lerde bir LCD yi set etme örneğindeki şekli;

```

struct cont_pins
{
    boolean en1; //tüm displaylere izin vermek için
    boolean en2; //40x4 displaylere izin vermek için
    boolean rs; //kaydedici seçimi
    int data:4;
} cont;

#byte cont = 8; //portd den kontrol (lcd portd ye bağlanmış)

```

Bu kod parçası **cont\_pins** için yapıyı set eder ve program ile birlikte çalışır.

**NOT:** Kod içinde data için yazılmış **:4** notasyonu bu nesne için 4 bitin ayrılacağını bildirir. Bu durumda D0 en1 ve D3-6 data olacaktır.

```

void LcdSendNibble(byte n)
{
    cont.data=n; //present data
    delay_cycles(1); //delay
    cont.en1=1; //set en1 line high
    delay_us(2); //delay
    cont.en1=0; //set en1 line low
}

```

## ALİŞTİRMA

1. İçinde bir karakter ve 40 adet karaktere sahip dizisi olan bir yapı içeren program yazınız. Keyboard'tan bir karakter okuyunuz ve bu karakteri getch() kullanarak karakter içine kaydediniz. Bir dizi okuyunuz ve gets() kullanarak dizi içine kaydediniz. Sonra üyelerin değerlerini print ediniz.

2. Bu kod parçasındaki yanlış nedir?

```

struct type
{
    int i;
    long l;
    char str[50];
} s;
.
.
i = 10;

```

## 8.2 İşaretçilerden Yapılara

Bazen bir işaretçi üzerinden bir yapıya ulaşmak çok kullanışlı olmaktadır. İşaretçilerden yapılara bildirim bir işaretçinin diğer veri tiplerinde olduğu gibidir. Örneğin, takip eden kod parçası yapı tipi **temp** olan, yapı değişkeni **p** ve yapı işaretçi değişkeni **q** olan yapıyı bildirir.

```

struct temp
{
    int i;
    char ch;
} p, *q;

```

**temp** yapısının bu şekilde tanımlanması kullanıldığında **q=&p** deyimi geçerlidir. Şu anda **q p**'yi işaret eder.

İşaretçinin gösterdiği yapının elemanlarına (örneğin **i**) ulaşmak için **->** işareti kullanılır. **ok** operatörünü (**->**) aşağıda gösterildiği şekilde kullanmalısınız;

```
q->i=1;
```

Bu işaretin aslı **\*q.i** gibidir. Ancak görsel açıdan daha anlaşılır olması için **->** kullanımı tercih edilmiştir. **Ok** operatörünün eksi işaretini takip eden büyüktür işareti olduğuna ve arada boşluk bulunmadığına dikkat ediniz.

Bu ifade **p** değişkeninin **i** elemanına **1** değerini atar.

C tüm yapıyı fonksiyona geçirdiğinden beri, geniş yapılar büyük data transferi yapmasından dolayı program işletim hızını yavaşlatabiliyorlardı. Bu nedenden dolayı bir işaretçinin bir yapıya, bir fonksiyona geçirilmesi daha kolaydır.

**Not alınması gereken önemli bir şey:** Bir yapı üyesine yapı değişkeni kullanarak erişirken nokta'yı kullanın. Bir yapı üyesine erişim için işaretçiden yapıya kullanılıyorsa **ok** operatörünü kullanmalısınız.

Bu örnek işaretçiden yapıya nasıl kullanıldığını gösterir.

```

#include <16c74.h>
#include <string.h>
struct s_type
{
    int i;
    char str[80];
} s, *p;

void main(void)
{
    p=&s;
    s.i=10;
    p->i=10;
    strcpy(p->str, "struct olayini seviyorum");
    printf("%d %d %s", s.i, p->i, p->str);
}

```

Bu iki satır **s.i=10** ve **p->i=10** eşittirler.

**ALİŞTİRMA:**

1. Bu kod parçası doğrumudur?

```
struct s_type
{
    int a;
    int b;
} s,*p;

void main(void)
{
    p=&s;
    p.a=100;
}
```

2. Üç long uzunluğunda bir yapı dizilimi oluşturan bir program yazınız. Yapıları yüklemek için PIC16C5X, PIC16CXX ve PIC17CXX ihtiyacınız olacaktır. Kullanıcı klavyenin 1,2 ve 3 girişlerini kullanarak yazdırmak istediği yapıyı seçecektir. Yapının biçimi:

```
struct PIC
{
    char name[20];
    unsigned char progmem;
    unsigned char datamem;
    char feature[80];
};
```

### 8.3 İç İçe Yapılar

Şu ana kadar bir yapı üyelerinin yalnızca C dili data tipi olduğunu gördünüz. Fakat yapı üyeleri başka bir yapı olabilirler. Buna iç içe yapılar denir. Örneğin;

```
#define NUM_OF_PICS 25
struct PIC
{
    char name[40];
    unsigned char progmem;
    unsigned char datamem;
    char feature[80];
};

struct urun
{
    struct PIC aygitlar[NUM_OF_PICS];
    char paket_tipi[40];
    float maliyet;
} list1;
```

**urun** yapısının üç elemanı vardır: **aygitlar** adıyla **PIC** yapı dizisi, **paket\_tipi** adıyla karakter dizisi ve **maliyet** adıyla gerçek sayı tipinde 4 baytlık değişken. Bu elemanlara **list1** değişkeni kullanılarak erişilebilir. Örneğin **list1** yapısının ihtiva ettiği **aygitlar** yapısının 10. elemanının **progmem** değerine erişmek için **list1.aygitlar[9].progmem** ifadesi kullanılmalıdır.

### 8.4 Birliklere Giriş

Bir birlik iki veya daha fazla deęişken tarafından paylaşılan bir hafıza bölgesidir. Hafıza bölgesini paylaşan deęişkenler deęişik veri tiplerinden olabilirler. Fakat her defasında yalnızca bir deęişken kullanabilirsiniz. Bir birlik bir yapıya çok benzemektedir. Birliğin genel yapısı şöyledir;

```
union etiket_adi
{
    type eleman1;
    type eleman2;
    .
    .
    type elemanN;
} deęişken_listesi;
```

Yine **etiket\_adi** tanımlanan birliğin adı ve **deęişken\_listesi etiket\_adi** tipinde bir birliği olan deęişkenlerdir. Birlikler ve yapılar arasındaki fark, birliklerin her üyesinin aynı bilgi alanını paylaşmasıdır.

Örneğin, takip eden birlik üç adet üye içerir: bir integer, bir karakter dizisi ve bir double

```
union u_type
{
    int i;
    char c[3];
    double d;
} temp;
```

Birliğin hafızadaki şekli aşağıda gösterilmiştir. Biz yapıyı anlatmak için bir önceki örneği kullanacağız. Integer iki byte, karakter dizisi 3 byte ve double 4 byte kullanır.

```
<----- double ----->
<--c[2]--><--c[1]--><--c[0]-->
<-----integer----->
```

Bir birliğin üyelerine erişmek için yapılarda olduğu gibi nokta kullanılır. **temp.i** ifadesi **temp** birliğinin iki baytlık integer tipindeki **i** üyesine ve **temp.d** dört baytlık double tipindeki **d** ye erişir. Eğer bir işaretçi üzerinden bir birliğe erişeceksiniz yapılarda olduğu gibi ok operatörünü (->) kullanmalısınız.

**Not alınması gereken önemli bir nokta**, bir birliğin compiler daki boyutu birlik içindeki en büyük boyutlu üyeye bağlı olarak sabitlenir. Double tipinde olanlarının boyutları 4 byte uzunluğunda olduğunu varsayarsak, temp birliği 4 byte uzunluğunda olacaktır.

Birlik kullanmanın güzel bir örneği, seri portuna harici 12-bit ADC bağlı olan 8-bit mikrokontrolör uygulamasında verilebilir. Mikrokontrolör A/D'yi iki baytta okur. Öyleyse biz iki unsigned chars ve bir signed short tipinde üyelere sahip bir birlik oluşturmalıyız.

```
union sample
{
    unsigned char baytlar[2];
    signed short word;
}
```

A/D okumak istediğinizde , A/D den iki byte okur ve bunları **baytlar** dizilimi içinde saklayabilirsiniz. Daha sonra 12-bitlik örneği kullanmak isterseniz , 12-bit sayıya ulaşmak için **word** u kullanmalısınız. Yani aynı adreste saklana bilgilere hem baytlar dizisi ile hem de word değişkeni ile ulaşabilirsiniz.