C Programlama Ders Notları

Revizyon ve Baskı Bilgisi
54/17.10.2007 01:35:00 PM

İçindekiler

Önsöz

Bu ders notlarının yazılmasında Brian W Kernighan ve Dennis Ritchie isimli yazarların “The C Programming Language” isimli kitabından faydalanılmıştır. Bu kitabı İnönü Üniversitesi Kütüphanesinde bulabilmeniz mümkündür. Bu ders notları hem içerik olarak hem de takip ettiği müfredat olarak yukarıda bahsi geçen kitabı örnek olarak almıştır.

Yaralanılabilecek Diğer Kaynaklar

Program Nedir?

Program, günlük hayatta bir sorunu bilgisayar ile çözmek, rutin işlemleri kolaylaştırmak için yazılan yazılımlardır. Bir program bilgisayar üzerinde çalışır ve insanların günlük hayatlarını kolaylaştırır.

Kişinin program yazması için öncelikle Genel Programlama Bilgisine sahip olması gerekir. Peşinden bir Programlama Dili bilmek gereklidir. Burada önemli olan programlama bilgisidir. Bu konuda kendinizi iyi hissedebiliyorsanız herhangi bir programlama dili ile programlarınızı yazabilirsiniz. Dil tercihi yazılacak programa, soruna ve platforma uygun olarak yapılabilir.

Programlama Dili Nedir?

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

Belki daha kısa bir tanımı ile sizinle bilgisayar arasında bir tercümandır demek doğru olur.

Bir sorun çözüleceği zaman öncelikle iyice anlaşılmış olmalıdır. Sonra bu sorunu çözebilecek bir çözüm zihinsel olarak hazırlanır. Bu çözüm bilgisayara uygun bir çözüm olmalıdır. Şöyle ki her çözüm bilgisayarda uygulanamaz. Çünkü her çözümün takip ettiği yol yeteri kadar basit olmayabilir. Üretilen çözüm son derece basit adımlarla anlatılabilmelidir. Bu adımlarla çözümün anlatılmasına Algoritma denir Bu adımlar alt alta yazılmak suretiyle oluşturulan çözüm bilgisayar için uygundur. Ancak ihtiyaç var ise bu adımlar Akış Çizgesine çevrilebilir. Algoritma doğal bir dille yazılır ve sıkı sıkıya kuralları bulunmaz. Anlaşılmasının kolay olması yeterlidir. Akış Çizgesinde belirlenmiş semboller yer alır ve bu semboller tüm dünyada standarttır. Kısmen biçimsel olan bu çizge sorunun çözümünü daha evrensel bir dille ifade eder.

Son adım olarak sıra Akış Çizgesi veya Algoritma ile elde edilen çözümün bir Programlama dili ile Bilgisayar ortamına aktarılmasına gelmiştir. Programlama dili son derece standart tanımlar içerir ve bir programı yazarken bu tanımlardan bir an için bile uzaklaşamazsınız. O nedenle de bir program parçasından başkalarının başkadan başkalarının başka şeyler anlaması mümkün değildir. Yazılan bu programlar bir derleyici vasıtası ile Makine diline çevrilir varsa hataların bulunmasını sağlar ve kullanıcı bu hataları düzeltir.

C Nasıl Bir Programlama Dilidir?

C Makine Dili ile üst düzey programlama dili arasında olan son derece esnek kullanımlı bir programlama dilidir. Esnek olması yazacağınız programda daha dikkatli olmanız gerektiği anlamına gelir. Unutulacak bir işaret belki derleyici hatasına neden olmayacaktır ancak programınızı da doğru çalıştırmayacaktır.

Genel Amaçlı, oldukça çok sayıda ifade, denetim komutları bulunduran, güçlü veri yapılarına sahip olan bir programlama dilidir. İlk başlarda UNIX üzerinde tasarlanıp geliştirilen bu dil şimdi tüm işletim sistemlerin de yaygın olarak kullanılmakta ve diğer bazı programlama dillerinde olduğu gibi modernlik kavramını yitirmemektedir.

"Bir Assembler derleyicisinin sağladığı esneklik ve gücü sağlarken üst düzey bir programlama dilinin sağladığı kolay anlaşılabilirlik özelliğini de sunmaktadır"

C'de Program Yazma...

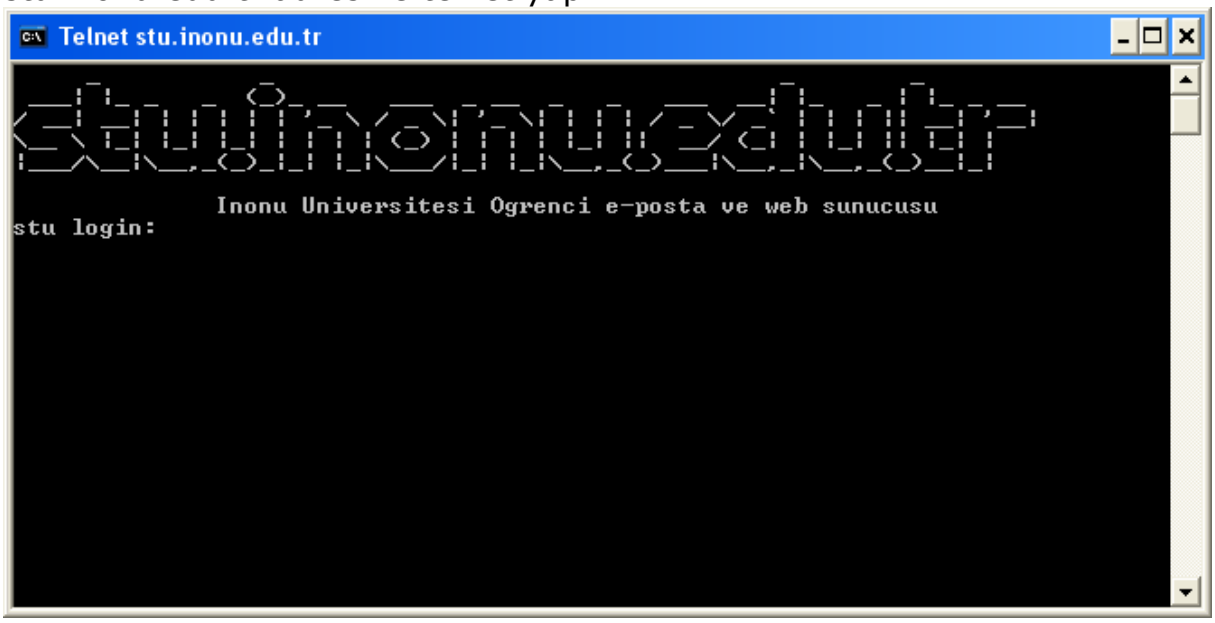
Çok basit anlamda bir metin düzenleyici ve bir derleyiciniz var ise C'de program yazabilirsiniz.

Eğer TC (Turbo C) ortamını kullanıyorsanız metin düzenleyici ve derleyici iç içedir.

İnönü Üniversitesi İnternet Merkezinde Program Yazma

Eğer İnternet Merkezindeki Bilgisayarlardan faydalanarak C Programlarını LINUX üzerinde geliştireceksiniz şu adımları takip etmeniz gerekecek:

1. Öncelikle Kendinize -hala yoksa- bir mail adresi alın. Bu aynı zamanda sizin LINUX makine üzerinde kullanıcı şifreniz olacaktır.
2. İster Windows makinelerden isterseniz de LINUX makinelerden stu.inonu.edu.tr adresine telnet yapın



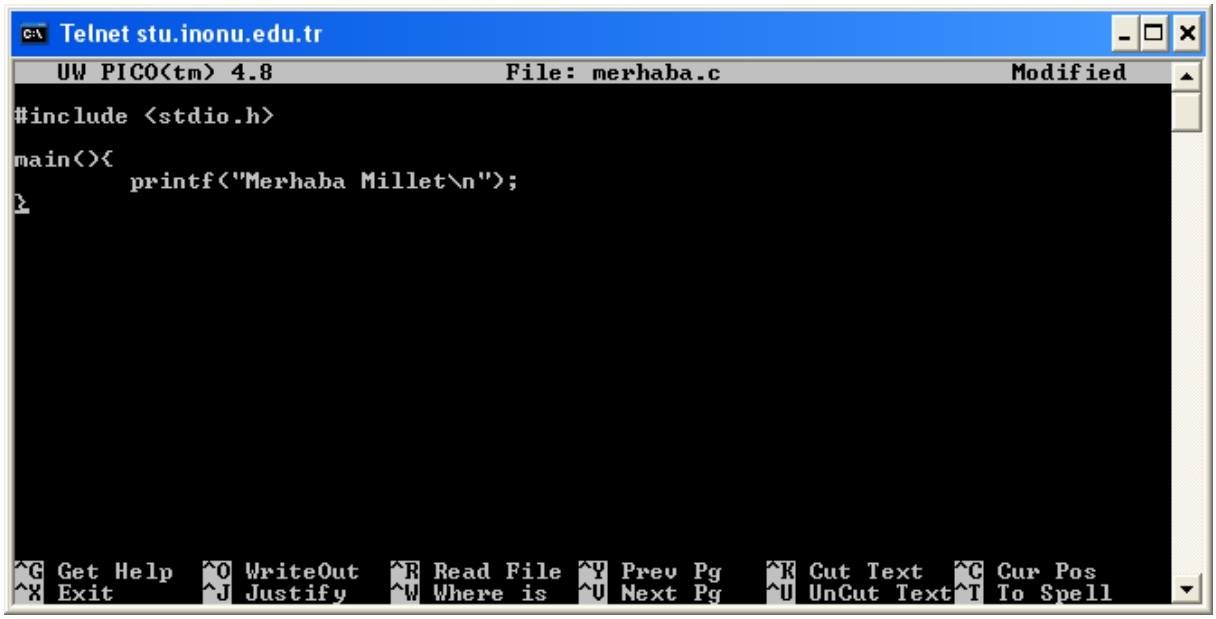
3. Kullanıcı adı ve şifrenizi kullanarak sisteme login olun.
4. **`pico merhaba.c`**
   yazarak Pico editorunun boş bir sayfa ile gelmesini sağlayın.

Örneğin şu programı yazın:

```
#include <stdio.h>

main(){
printf("Merhaba Millet\n");
```

}



5. Ctrl-X tuşlarına basarak pico editöründen çıkın. Soracağı **"Kaydedilsin mi?" ("Save modified buffer....?")** sorusuna **"Y"** ile karşılık verin. Dosya adı ne olacak şeklindeki soruya uygun bir dosya adı ile cevap verin. Burada **`merhaba.c`** zaten yazıyordur. Enter tuşuna basıp geçebilirsiniz.
6. Linux'a dönünce komut satırından
   `gcc merhaba.c -o merhaba`
   yazarak programı derleyin hata mesajı alırsanız 4. adıma geri dönün. Eğer –o kısmından sonra bir şey yazmazsanız bu programın derlendikten sonra **`a.out`** ismiyle kaydedilmesini sağlamış olursunuz. Burada unutulmaması veya karıştırılmaması gereken şey **merhaba.c** programın kaynak kodlarıdır. Sizin yazdığınız C komutlarıdır. **Merhaba** dosyasına kaydedilen program ise makine diline çevrilmiş olan programdır. Ve programı çalıştırmak için **merhaba** dosyasına üzerinde değişiklik yapmak için **merhaba.c** dosyasına ihtiyaç vardır.

`./merhaba`
yazarak programınızı çalıştırın.
Bu komut yazılıp ENTER tuşuna basıldıktan sonra ekrana **Merhaba Millet** yazıldığı görülecektir.

7.
8. Tüm çalışmalarınız bitirdikten sonra sistemden çıkmak için
   `logout` `veya` **`exit`**
   yazmayı unutmayın.

Kendi Yerel Bilgisayarınızda program yazma

Yazılmış Programı Kağıt üzerinde çalıştırma

Algoritma Çalıştırma konusunda gördüğümüz gibi bazen de yazılan programlar kağıt üzerinde çalıştırılmak zorunda kalınabilir. Gerçi bir fiziksel bilgisayar ortamı var ise bu işleme gerek olmayabilir. Ancak mantıksal bir hatanın nerde olduğunu görmenin en iyi yollarından biri programı el ile kağıt üzerinde çalıştırmaktır. Yani programın her bir adımı tek tek programcı tarafından çalıştırılır ve değişkenlere ilişkin değerler bir tabloda sıra ile gösterilir. Bu hem programda nerede hata yaptığınızı anlatır hem de programın ne sonuçlar üreteceğini size gösterir.

Genel Program Yapısı

Bu bölümde Genel bir C Programının yapısını C programlama dilinin detaylarına fazla girmeden göstermeye çalışacağız. Bu noktada amacımız mükemmel programlar ve eksiksiz tanımlar yapmak değil olabildiğince hızlı bir şekilde C programlarının genel görünüşleri ve işleyişleri hakkında bilgi vermektir. İlerleyen bölümlerde bu programlarda yazdığımız bir takım yapılar, komutlar detaylıca anlatılacaktır.

Bu nedenle bu bölümde ele alıp işlemiş olabileceğimiz bir konu diğer bölümlerde daha detaylı işlenebilecektir. Umarız bu sizlerin daha iyi öğrenmenizi ve daha fazla zevk almanızı sağlayacaktır.

Başlarken

Bir Programlama dilini öğrenmenin en iyi ve en kolay yolu o dilde programlar yazmaktır. Bir çok programlama dilinde genellikle kullanılan şu programı yazalım. Ekrana Merhaba yazan bir programdan bahsediyoruz.

Bu bir çok programlama dilinde temel olarak ele alınan en basit konudur. Önemli olanlardan biri bu programı yazabilmek, başarılı bir şekilde derleyebilmek, çalıştırabilmek ve elde edilen sonuçları görebilmeyi anlatır.

```
#include <stdio.h>
main(){
   printf("Merhaba Dünya\n");
}
```

Bu programı nasıl derleyip çalıştıracağınız kullandığınız işletim sistemi ve derleyiciye bağlıdır. Bir önceki konuda bunun hakkında yeterli bilgi verdik.

Diğer sistemlerde bu işlem farklılıklar gösterebilecektir. Yukarıdaki programı yazarak derleyiniz ve çalıştırmaya çalışınız.

Program hakkında yapılacak bir takım açıklamalar şunlar olabilir. Bir C programı boyutu ne olursa olsun bir ya da daha fazla fonksiyon denilen bloktan oluşur. Bu fonksiyonların her biri gerçek işlemlerin yapıldığı yerlerdir. Bu örnekte main böyle bir fonksiyondur. Normal olarak fonksiyonlarınıza istediğiniz adı verme özgürlüğüne sahipsiniz. Ancak main özel bir isimdir. C programınız bu fonksiyonun başladığı yerden itibaren icra edilmeye başlanır. Bu aynı zamanda her programın bir main fonksiyonunun olması gerektiği anlamına gelir. Genellikle bu fonksiyon diğer fonksiyonları çağırarak, C komutlarını icra ederek veya hazır diğer fonksiyonları çağırarak yapması gereken işlemleri yapar.

Fonksiyonlar arasında haberleşme ve değerlerin transferi parametre-argümanlar maharetiyle gerçekleşir. Fonksiyon adındaki parantezler arasında parametre listesi yer alır. Bu örneğimizde main fonksiyonunun parametresi olmadığı için parantezler arasında herhangi bir ifade bulunmamaktadır. Küme parantezleri arasında yazılan C ifadeleri bu fonksiyonu meydana getiren komutların başladığı ve bittiği yerleri tanımlar.Bir fonksiyon normal olarak isminin anılması ile icra edilir. Eğer fonksiyonun parametreleri var ise parantezler arasında bu parametre listesi virgüllerle ayrılmış olarak verilir. Fonksiyonu çağıran özel ayrılmış bir kelime ya da komut bulunmamaktadır. Fonksiyon parametre listesi almasa bile parantezler yazılmak zorundadır.

printf(“Merhaba Dünya\n”); satırı adı printf olan bir fonksiyon çağırır. Bu çağrıda parametre olarak Tırnak içerisinde yazılı olan ifade kullanılır. printf ekrana bir bilginin yazılmasını sağlayan bir kütüphane fonksiyonudur. Bu durumda bu fonksiyon, ekrana, parametre olarak gönderilen ifadeyi yazacaktır.

“...” şeklindeki çift tırnakların arasına yazılan ifadelere karakter sabiti veya alfabetik sabit denir. Şimdilik biz bu tür sabitleri sadece printf ve benzeri fonksiyonlarda parametre olarak kullanacağız.

Bu karakter dizisi içinde yer alan \n ifadesi yeni satır veya bir alt satır anlamına gelen dizedir. Bu dizenin karşılaşıldığı karakter sabitinde bir alt satıra geçilir. Eğer bu \n ifade içinde yer almazsa program çıktısında bir alt satıra geçilmediğini göreceksiniz. Yeni bir satıra geçebilmek için programda \n ifadesini uygun bir yerde kullanmak zorundasınız. Aksi halde
printf(“Merhaba Dünya
”);
şeklinde bir ifade yazarak yeni bir satırı elde etme şansınız olmadığı gibi derleyici hataları ile karşılaşacaksınız.

printf fonksiyonu asla otomatik bir yeni satıra geçiş işlemi sağlamaz. O nedenle bir satır bilgi yazabilmek için birden fazla printf kullanılabilir. Örneğin programımız şu şekilde yazılabilir.

```
main(){
    printf(“Merhaba, “);
    printf(“Dünya”);
    printf(“\n”)
}
```

şeklindeki bir çıktı demin ki çıktının aynısını üretecektir.

\n ifadesinin tek bir karakter anlattığını bilmenizde fayda var. Escape sequence denilen bu sistemde görünmeyen karakterleri ifade edebilmek mümkündür. Bu tarz karakterlerin arasında şunların da bulunduğunu

söyleyebiliriz: \t tab için, \b geriye doğru silme işlemi için (Backspace), \" çift tırnağın kendisi için, \\ \ kendisi için kullanılan dizelerdir.

Değişkenler ve Aritmetik işlemler

Sonraki program Fahrenheit sıcaklık değerleri ile santigrat sıcaklık ölçü karşılıklarını veren bir tabloyu ekrana yazabilecektir. Bu işlem için kullanılan formül şu şekildedir. C=(5/9)(F-32).

```
/* Fahrenheit ve santigrat Tablosu
    f=0, 20, ... ,300 değerleri için
main(){
    int alt, ust, adim;
    float fahr, celcius;
    alt=0;
    ust=300;
    adim=20;

    fahr=alt;
    while (fahr<=ust){
        celcius=(5.0/9.0) * (fahr-32.0);
        fahr=fahr + step;
    }
}
```

Bu programdaki ilk iki satır açıklama satırı olarak anılırlar. Bu satırlar çoğunlukla programın veya o kesimin ne iş yaptığını kısaca açıklayan satırlardır. /* ile */ ifadeleri arasına yazılan tüm karakterler derleyici tarafından göz ardı edilirler. O nedenle programın daha kolay anlaşılması için rahatlıkla kullanılabilirler.

C Programlama dilinde değişkenler kullanılmadan önce tanımlanmalıdırlar. Bu tanım genellikle programın en başında herhangi bir komuttan önce yer alır. Eğer bir değişkeni tanımlamayı unutmuşsanız derleyicinin size durumu hata raporu ile bildirdiğini görürsünüz. Bir tanım tür ve bu türde tanımlanmak istenen değişken listesinden oluşur. Şöyle ki:

int alt,ust, adim;
float fahr,celcius;

int türü yanındaki değişkenlerin tamsayı olduğunu vurgulamaktadır. float ise kesirli sayıları vurgular. int ve float'ın hassasiyeti kullandığınız bilgisayara ve işletim sistemine hatta derleyiciye bağlı olarak değişebilir.

Genellikle int türündeki bir sayı –32768 ile +32767 aralığında bulunabilir. float türdeki bir değişken ise 32 bit yer işgal eder ve 10^{-38} ile 10^{+38} aralığında değerleri taşıyabilir.

C dilinde başka temel değişken türleri bulunur.char, double, long, short gibi ifadelerle bunlar anlatılırlar. Bu değişkenlerin büyüklükleri makineye bağımlıdır. Bunların haricinde diziler (arrays), yapılar (structures), unions tarzında değişken türlerinin tanımlanabileceği ifadeler bulunacaktır.

Programda aşağıdaki işlemler aritmetik değer atama işlemleridir. Değer hesaplama bu kısımda bahsi geçen komutlar ile başlar.

```
alt=0;
ust=300;
adim=20;
fahr=alt;
```

Bu komutlar değişkenlerimize ilk değer atamalarını sağlar her komutun noktalı virgül ile bittiğine dikkatlerinizi çekmek isteriz.

Tablonuzun her satırındaki değer aynı yöntemle hesaplanır.Öyleyse her satırı her seferinde hesaplayan bir döngü komutu işlemimizi rahatlıkla ifade edebilir. Bu amaçla **`while`** komutu kullanılmıştır.

```
while (fahr <= ust){
    ...
}
```

Bu komutta parantezler arasında yer alan koşul test edilir. Eğer sonuç doğru ise (yani fahr içindeki değer ust içindeki değerden büyük veya eşit ise) küme parantezleri arasında kalan işlem gerçekleştirilir. Bu koşul tekrar kontrol edilir ve bu işlem tekrarlanır ta ki yapılan kontrol yanlış oluncaya kadar. Koşul yanlış sonuca ulaşırsa program akışı döngü komutundan sonraki komuttan devam eder. Bu programda döngü komutundan sonra komutlar bulunmadığı için program sonlandırılır.

while döngü komutunun gövdesi bir komuttan oluşabileceği gibi küme parantezleri içerisine yerleştirilmiş birden fazla komuttan da oluşabilir. İstenirse tek komut ta küme parantezi içerisine yerleştirilerek anlatılabilir. Genellikle while komutunun içindeki komutlar okunması ve bir bakışta kolay anlaşılması için bir tab içeriden yazılırlar. Bu biçimleme programınızın mantıksal yapısını vurgular. Buna rağmen C dilinin satırda yazılacak bilgilerin hangi sütuna konulmasının belirlenmesi hususunda kısıtlamasının olmadığını bilmelisiniz. Düzenli girinti ve çıkıntıların programda vurgulanabilmesi programlarınızın aşka insanlar tarafından kolay okunmasını sağlar.Bir satıra bir komut yazmanızı ve operatörler arasına boşluk bırakarak yazmanızı tavsiye ederiz. Küme parantezlerinin konumu çok önemli değildir. Ancak okunaklılığı arttırmak için bir stil sahibi olmanızda fayda bulunmaktadır.

İşlemlerin çoğu Döngülerin içinde halledilir. Programda celcius=(5.0/9.0) * (fahr-32.0) komutunda 5.0/9.0 yazılmıştır. Halbuki 5/9 yazmak daha kolay

olabilecekken neden böyle bir yol seçilmiştir. Çünkü C dilinde tamsayı bölme işleminden elde edilen sayıda tam kısım alınır kesirli kısımlar imha edilir. Kesirli bir ondalık basamağının sıfır olsa bile belirtilmesi bu ifadenin bir kesirli bölme olduğunu C diline anlatır ve sonuç kesirli sayı olarak istediğimiz şekilde elde edilir.

Tür çevrimleri ile ilgili konulara daha sonra değinilecektir. Ancak burada şunu söyleyebiliriz

fahr=alt veya while (fahr<=ust) komutlarında int değerler işleme alınmadan önce float türüne dönüştürülürler.

printf fonksiyonunun tam olarak nasıl çalıştığı daha sonraki konularda anlatılacaktır. Ancak şu konuyu burada hatırlatmakta fayda var. Esasında printf C dilinin bir parçası değildir. Daha doğrusu C dili komutu değildir. Esasında oldukça kullanışlı bir çıkış fonksiyonudur.

Alıştırma 1

Yukarıdaki programın aksine celcius dereceden fahrenheit dereceye değerlerin verildiği bir tablo oluşturunuz.

Değişken Tanımlama Kuralları

Değişken, bir programda değişik zamanlarda değişik değerler tutan bellek birimleridir. Bir değişken programlar için çok önemlidir. Çünkü matematiksel, alfabetik vb. tüm işlemler bu değişkenler üzerinde yapılır ve sonuçlar bu değişkenler üzerinde tutulur. Her programlama dilinde değişkenler, içlerinde saklayacağı bilginin türüne göre sınıflandırılır. Değişkenler haricinde bir bilgisayar programında sabitler bulunur. Bu sabitler yine türlerine göre sınıflandırılır ancak ifade ettikleri değer sabittir.

Değişken Tanımlarında değişken listesi, değişkenin türü ve gerekiyorsa ilk değerler verilir.

Sabitler

Bir Program içerisinde kullanılan tüm alfabetik ve sayısal değerler sabit olarak adlandırılır. Bu açıdan sabitler sayısal ve alfabetik sabitler olarak ikiye ayrılır.

SABİTLER		
SAYISAL SABİTLER		**ALFASAYISAL SABİTLER**
TAM SAYILAR	**KESİRLİ SAYILAR**	
8	**12.45**	"Enformatik Bölümü"
12	**3.345e+03**	"İnönü Üniversitesi"
12564 gibi	**72.0 gibi**	"4.56" Gibi

Dikkat edilirse alfa sayısal sabitler -"-(Tırnak) ifadeleri arasında yazılarak ifade edilirler. Tırnak içerisinde yazılan her şey Alfa sayısal sabit olup bu değerleri aritmetik işleme sokmak ya da bunları bir sayı gibi değerlendirmek mümkün değildir. Tırnak içindeki ifade her ne kadar sayı olsa da aslında o bir alfa sayısal ifadedir. İki harf veya daha fazla harften oluşan alfabetik sabitler çift tırnak içinde anlatılırken tek harfli sabitler tek tırnak içinde anlatılır. Ekranda belki görünmeyen ancak özel etkileri olan bir takım karakterlerimizde iki harf gibi görünseler de tek karakterlik değişkenler grubunda ele alınırlar ve tek tırnak içinde yazılırlar. '\n', '\t', '\0', '\\' '\'' '\"' gibi. Ayrıca '\ddd' şeklinde yazılan bir ifade aslında ddd değeri ile ifade edilen sıradaki karakteri anlatır. Örneğin \014 FormFeed karakterini anlatan bir sabittir.

Sekizli ve Onaltılı gösterimde tutulan sabitlerin de belirtilmesi gerekebilir. Herhangi bir değerin önündeki 0 (sıfır) sekizli bir sabiti anlatır. 0x şeklinde başlayan bir sayı ise on altılı bir sabiti ifade ediyor demektir. Örneğin 31 değeri onlu bir sabiti anlatırken 037 sekizli bir sayıyı anlatıyor demektir. Ve 0x15 gibi bir sayı da on altılı bir sabiti anlatıyor demektir.

\0 sabiti bir karakter katarında katarın sonlandığı noktayı gösteren özel sıfır değerli karakterdir.

Sabit ifade sadece sabit değerleri anlatan ifadelerdir. Bu şekildeki ifadeler derleme aşamasında hesaplanırlar. Çalışma zamanı için böyle bir tanımın etkisi bulunmaz. Hatta derleyici derleme aşamasında ilgili sabit tanımların yerine sabit değerleri yerleştirerek derleme işlemine devam eder.

#define MAXSAT 1250
char satirlar[MAXSAT+1]; ifadesinde define ile yapılan tanımda adı MAXSAT olan sabite 1250 değeri yüklenir ve aslında derleyici MAXSAT gördüğü tüm yerlere 1250 yazar ve gerekiyorsa hesap yapar ve sonra derler.

Değişken Türleri

C'de Temel olarak alfa sayısal ve sayısal değişken türleri bulunmaktadır. Aslında yukarıdakine benzer bir tablodan bahsetmek mümkün. Ancak sayısal değiş mümkün. Ancak sayısal değişken türleri sayıca daha fazladır.

C'de Bir değişken Tanımlamak için Genel olarak şu biçim kullanılır:
Tür_adı Değişken_ad1 [,Değişken_ad2[...]];

Burada Tür_adı ile verilen bilgi C Değişken türlerinden biridir. Değişken adı ise sizce isimlendirilecek olan ihtiyacınıza cevap verecek bir veya daha fazla değişkendir. Değişkenler birden fazla olacak ise virgül ile ayrılmalıdırlar. Sonuna noktalı virgül konulması zorunludur.

Değişken Adlandırma Kuralları

Bir değişkene isim verilirken harf ile başlamanız gerekmektedir. Sonraki karakterler harf-rakam karışık olabilir.

Değişken adları içerisinde Türkçe harf olarak isimlendirebileceğimiz ve sadece Türkçe alfabede bulunan ç, Ç, Ö, ö, Ş, ş, Ğ, ğ, ü, Ü, İ, ı harfleri kullanmamanız gerekmektedir. Ayrıca değişkenlerinize vereceğiniz isimler içerisinde *, /, +, },], gibi özel simgeler ve boşluk karakteri bulunamaz. Boşluk Karakteri yerine _ -alt tire- sembolü kullanılabilir.

C için ayrılmış sözcük olarak kullanılan ifadeler değişken adı olarak kullanılamaz.

Alfa sayısal Değişkenler

char değişken_adi;

Kuralına göre tanımlanan değişkenlerdir. C dilinde bu şekilde tanımladığınız bir alfa sayısal değişken içerisinde sadece 1 harf saklayabilirsiniz. Eğer birden fazla harf saklanması gerekiyorsa şu tanımı kullanmanız daha uygun olacaktır:

char isim[28];

Bu tanımda ise toplam 28 karakter uzunluğunda adı isim olan bir alfa sayısal değişken tanımlanmıştır.

Sayısal Değişkenler

Tür Adı	Tanım ve Açıklama
int	Tamsayı değişken tanımlama için kullanılır.
Unsigned int	İşaretsiz yani Pozitif Tamsayı değişken tanımlama için kullanılır.
long int	Uzun Tamsayı (Yani daha büyük tam sayılar) değişken tanımlama için kullanılır.
Unsigned long int	İşaretsiz uzun Tamsayı değişken tanımlama için kullanılır.
Float	Kesirli değişken tanımlama için kullanılır.
Double	Kesirli çift duyarlıklı değişken tanımlama için kullanılır.
long double	Kesirli çift duyarlıklı uzun değişken tanımlama için kullanılır.

Tanımlamalar

Kullanılmadan önce tüm değişkenler tanımlanmalıdır. Hatta bazı değişken tanımları sadece kullanıldıkları fonksiyon içinde yer verilebilir. Bir tanım bir tür bilgisi ve yanında bu türden olacak değişken listelerinden oluşur. Örneğin:

```
int alt, ust, adim;
char c, ad[25];
```

Değişkenler muhtelif tanım satırları içerisinde dağıtılmış olabilirler. Yani yukarıdaki tanım şu şekilde yazılmış olabilirdi:

```
int alt;
int ust;
int adim;
char c;
char ad[25];
```

Bu gösterim programınızın kaynak kodlarında daha fazla yer işgal edecektir. Ancak her değişkene açıklama satırları koymak vb. düzeltmelerin kolay yapılmasını sağlamak amacıyla uygun bir gösterim olabilirdi.

Değişkenler tanımlandıkları sırada ilk değer ataması ile sıfırlanabilirlerde. Eğer bir değişken tanımlama satırında değişken adından sonra = işareti ve

onun yanına bir sabit yazıyorsanız bir değişkeni herhangi bir değerle sıfırlamış olursunuz.

```
char  bs = '\\'
int i = 0; gibi.
```

Bu tür bir ifadede söz konusu değişken external ya da static tanımlanmış ise bu değer sıfırlama işlemi bir kez ve program çalışmaya başlamadan önce yapılır. Diğer normal tanımlanmış değişkenler. Fonksiyon her çağrıldığında yeniden sıfırlama işlemine tabii tutulurlar. Eğer her hangi bir değişken sıfırlanmadan (başlangıç değeri verilmeden) kullanılacak olursa ilk değeri tanımsız bir değer olacaktır. External ve static değişkenlerin ilk değerleri 0 (Sıfır) olarak atanacaktır. Böyle olmasına rağmen hangi tür değişken ile çalışıyor olursanız olun sizin bir değer atama işlemi yapmanız en iyisidir.

Yeni değişken türleri ile tanıştıkça değer sıfırlama işlemlerini anlatmaya devam edeceğiz.

Aritmetik İfadeler

Bir matematiksel işlem yapabilen değimlerdir. Çoğunlukla bir değer aktarma ve değer işleme bölümlerinden oluşur. İşlenen değerler bir aktarma değimi ile (çoğunlukla =) başka bir değişkene aktarılır. Ancak bazen bu bulunan değer doğrudan ekrana yazdırılabilir veya başka bir noktaya yönlendirilebilir.

Örneğin `k=5*1+2` ifadesi bir değer aktarma değimidir ve bir aritmetik ifadedir.

Ancak Tek başına `1*25/100` ifadesi de bir aritmetik ifadedir. Bir aritmetik ifade içerisinde sayısal değişkenler, sayısal sabitler ve aritmetik operatörler yer alır.

Aritmetik Operatörler

C dili aritmetik operatörler yönü ile çok zengin bir dildir. İşlemleri kolaylaştıran bir çok ifadeyle karşılaşmak mümkündür. İkili (Binary) operatörler +,-, /, * ve % operatörleridir. Tekil (Unary) operatörler ise – ve + olarak karşımıza çıkmaktadır. Tamsayı bölme yapıldığında elde edilen sonucu tamdır. Kesirler kesilip atılır. X % Y işlemi X'in Y'ye bölümünden kalanı bulur. Örneğin artık yıl 4'e tam bölünebilen, fakat 100'e tam bölünemeyen ancak 400'e tam bölünebilen yıllardır.

```
if (yil %4 == 0 && yil % 100 != 0 || yil % 400 == 0)
   artık yıldır
```

```
else
```
artık yıl değildir.

% operatörü float ve double'a yani kesirli değerlere uygulanamaz.

+ ve – operatörleri aynı önceliğe sahiptir. Ancak bu operatörlerin önceliği *, /, % operatörlerinden daha düşüktür. İşaret değiştirme operatörü olan – en öncelikli işlemdir. Eş öncelikli operatörler soldan sağa doğru işlenirler. Ancak parantezler kullanılarak işlem öncelikleri değiştirilebilir. Genelde a + (b + c) işlemi (a + b) + c işlemi ile aynı sonucu üretir. Ancak bazen değişken içerisine değerin sığmaması ve alt taşma veya üst taşmadan dolayı bu değerler birbirine eşit olmayabilir.

Mantıksal ifadeler

Sonucunda doğru ya da yanlış değer üretilen ifadelere mantıksal ifade denir. Ancak C dili açısından özellikle doğru ya da yanlış diye bir kavram bulunmaz. Onun yerine 0 ve 0'dan farklı değerler söz konusudur. Bu bakımdan 0 değeri yanlış olarak algılanmalıdır. Diğer değerler doğru olarak algılanmalıdır.

Bu sebeple bir C Mantıksal ifadesi aslında herhangi bir sayısal değer'dir ve illa karşılaştırma operatörleri veya mantıksal operatörlerle ifade edilmiş olmak zorunda değildir. C dilinin bu özelliğini bir çok program parçasını incelediğinizde görmeniz mümkündür.

```
if (1)
printf("Merhaba\n");
else
printf("Yanlış\n");
```

ifadesinde ekrana her halükarda Merhaba yazılacaktır. Çünkü 1 değeri mantıksal doğru değerini de taşımaktadır.

Karşılaştırma İşleçleri ve Mantıksal Operatörler

Karşılaştırma işleçleri

İki Değeri birbirleri ile karşılaştırıp mantıksal doğru ya da mantıksal yanlış değerlerinden birini elde eden operatörlerdir. Bu iki değerin birbirine göre eşitlik, büyüklük veya küçüklük gibi ilgilerini karşılaştırır.

Op.	Açıklama	Örnek
>	Operatörün solundaki değer sağındaki değerden büyüktür.	k>123
<	Operatörün solundaki değer sağındaki değerden küçüktür.	k<123
>=	Operatörün solundaki değer sağındakinden büyük ya da eşittir.	k>=123

<=	Operatörün solundaki değer sağındaki değerden küçüktür.	k<=123
==	İki değer biri birine eşittir	k==123
!=	İki değer biri birinden farklıdır.	k!=123

Bu değer karşılaştırma işlemleri ya sayısal değerler için ya da alfa sayısal tek karakterler için geçerlidir. iki alfa sayısal dizenin karşılaştırılması için birtakım fonksiyonların kullanılması gerekmektedir. Tek karakterler ise 'Tek tırnak' simgeleri arasında yazılırlar.

Öncelik olarak tüm karşılaştırma operatörleri aynı önceliğe sahiptir. Ancak tahmin edeceğiniz gibi karşılaştırma işlemlerinin önceliği aritmetik operatörlerinkinden düşüktür. Yani karşılaştırma işleminde öncelikli olarak var ise aritmetik işlemler gerçekleştirilir.

Mantıksal Operatörler

Mantıksal Operatörler birden fazla mantıksal ifadeyi biri birine bağlamak için kullanılabilecek mantıksal değimlerdir. Bu değimler ve, veya, değil işlemlerine karşılık gelen ifadelerdir.

<table>
<tr><th>Op.</th><th>Açıklama</th><th>Örnek</th></tr>
<tr><td>&&</td><td>ve operatörü. İki Mantıksal ifadenin de doğru olmasını gerektiren operatördür. Yani iki farklı mantıksal ifade doğru ise elde edilecek sonuç yanlış olmaktadır.
<table>
<tr><th>P</th><th>Q</th><th>P && Q</th></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
</table></td><td>(k>123) && (k<321)</td></tr>
<tr><td>||</td><td>veya Operatörü İki Mantıksal ifadeden sadece birinin doğru olmasını gerektiren operatördür. Yani iki farklı mantıksal ifadeden biri doğru ise elde edilecek sonuç doğru aksi halde sonuç yanlış olmaktadır.
<table>
<tr><th>P</th><th>Q</th><th>P || Q</th></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
</table></td><td>(k>123) || (k<321)</td></tr>
<tr><td>!</td><td>değil operatörü. Bir mantıksal ifadenin mantıksal değilini elde eden operatördür. ! (k>=123) şeklinde kullanılır ve k<123 ifadesine eşdeğerdir.</td><td>! (k>=123)</td></tr>
</table>

Bu işlemlerde öncelik soldan sağadır. Hesaplama; doğru yada yanlış değerin bulunduğu anda eğer değer ne yapılacağını belirleme için yeterli ise durdurulur. Bu kriterler çalışan bir program yazmak için önemlidir.Örneğin bu tanımı netleştirecek bir komut inceleyelim

```
For ( i=0; i<lim-1 && (c=getchar()) != '\n' && c !=EOF;++i) s[i]=c;
```

Bu örnekte yeni bir karakter okumadan önce tampon sahada (s dizgisi) yeterli yer olup olmadığı denetlenmelidir. O nedenle i<lim-1 kontrolü ilk yapılır. Eğer bu kontrol yanlış sonuç verirse daha fazla karakter okumamalıdır. Ve program sonraki adımlardan devam edebilmelidir.

Benzeri bir şekilde c içine okunan değer EOF karakteri olmamalıdır yada satır sonu işareti olmamalıdır. && işleminin önceliği || işleminden yüksektir. Bu nedenle **`i<lim-1 && (c=getchar()) != '\n' && c != EOF;++i`** tarzındaki işlemlerin ek parantezlere ihtiyacı bulunmaz. Ancak **`!=`** işleminin önceliği değer atamadan yüksek olduğu için parantezlere ihtiyaç vardır.

`(c=getchar()) != '\n'` işlemi istenilen sonuca ulaşmamızı sağlar.

! operatörü mantıksal doğru değerini mantıksal yanlışa veya mantısal yanlışı mantıksal doğruya çevirir.

Tür Dönüşümleri

Eğer bir işlemde işlenen değerlerin türleri farklı ise genel bir ortak türe dönüştürülür ve öyle işlem yapılırlar. Bu dönüşümde şu kurallar uygulanır. Genelde otomatik olarak yapılan tek dönüşüm işlemi tamsayı değerin kesirli değere dönüştürülmesidir. Örneğin f + i işlemi sırasında i kesirli sayıya çevrilir. Ancak bunun yanı sıra bir kesirli değişken/değer dizilerde indis değeri olarak kullanılırsa otomatik dönüşüm gerçekleşmez.

Öncelikli olarak; char ve int türleri rahatlıkla biri biri içinde aritmetik ifadelerde kullanılabilirler. Her char otomatik olarak bir int değerine dönüştürülür. Bu bir takım karakter işlemlerinde inanılmaz kolaylıklar sağlar. Ayrıca karakter dizilerini integer değere çeviren atoi fonksiyonu tanımlanmıştır. Karakterler üzerinde aritmetik işlem yapan bir örnek büyük harfi küçük harfe çevirmek ya da tersi işlemidir.

```
char ch;
if (ch>='a' && ch<='z')
   ch+=32;
```

ifadesi ch içindeki harfi küçük harf ise büyük harfe dönüştürür.

Bu işlem ASCII karakter kümesi ve bu kümenin özelliğinden kaynaklanır. ASCII karakter kümesinde a ile A harfleri arasında 32 fark vardır.

Char ve short tanımlar int türe çevrilir float türler double'a çevrilir.

Eğer türlerden biri double ise diğeri de double çevrilir ve sonuç double olarak bulunur.

Eğer türlerden bir long ise diğer tür long'a çevrilir ve sonuç long olarak bulunur.

Türlerden bir unsigned ise diğeri de unsigned türe çevrilir ve sonuç unsigned olarak bulunur.

Diğer durumlarda işlenenler int türdedir ve sonuç int olarak bulunur.

Sonuç olarak bazı durumlarda da tür dönüşümleri zorunlu olarak program/programcı tarafından yapılmalıdır. Bu durumda

(tür-adı) ifade

tanımı ile ifade ile belirtilen değerler istenilen türe dönüştürülürler. Bu işleme cast denilir.

Artırma ve Eksiltme operatörleri

C dilinde değişkenlerin değerini artıran veya eksilten alışılmadık iki operatör bulunur. Arttırma operatörü ++ işlemi yanındaki değişkenin değerini bir arttırırken eksiltme operatörü olan – yanındaki değişkenin değerini bir eksiltir. Değişkenlerin değerlerini ++ operatörü ile arttırırız. Alışılmadık diğer bir konu ise ++ veya – operatörlerinin hem önek (değişkenden önce ++n) hem de sonek (değişkenden sonra n++) olarak kullanılabiliyor olmasıdır. Her iki durumda da n'in değeri bir arttırılır. Fakat ++n işleminde n'in değeri kullanılmadan önce arttırılırken; n++ işleminde n değeri önce kullanılır. sonra 1 arttırılır. Bu durumda şunu söyleyebiliriz: n'in değeri 5 olmak kaydıyla

X=n++ işlemi sonucunda x'in değeri 5 n'in değeri ise 6 olurken

X=++n işlemi sonucunda x'in değeri 6 n'i değeri ise 6 olur. Görüldüğü üzere her iki durumda da n'in değeri aynı olurken x'in değeri değişmektedir. ++ veya -- operatörleri sadece değişkenlerle birlikte kullanılabilir. Yani x=(ı+j)++ uygun bir kullanım değildir.

Mantıksal Bit operatörleri

C dilinde bir grup operatör bit düzeyinde işlem yapabilmektedir. Bu operatörler float ve double türleri üzerinde işlem yapamamaktadır.

- & Bit düzeyli VE işlemi
- | Bit düzeyli VEYA işlemi
- ^ Bit düzeyli XOR işlemi (Dışlayan ya da)
- << bitleri sola kaydır
- >> bitleri sağa kaydır

& operatörü çoğunlukla sayısal bir değer üzerinde istenilen bitlerin maskelenmesini sağlar. | operatörü bu işlemin tersinin yapılmasını sağlar. Bu operatörlerin && ve || ile karıştırılmaması gereklidir.

Değer Atama ifadeleri

i=i+1 şeklindeki ifadeleri C dilinde i+=2 biçiminde anlatabiliriz. Burada += şeklinde bir değer atama operatörü kullanılmıştır. İkili operatörlerin hepsinin bu şekilde değer atama değimleri bulunur. Şöyle ki;

ifade1 op= ifade2 ifadesi ifade1 = ifade1 op ifade2 şeklinde anlatılabilir.

Koşullu İfadeler

```
if (a>b)
  z=a;
else
  z=b;
```

ifadesi a veya b değerlerinden büyük olanın değerini bulan ifadedir. Bu işlemin karşılığı olarak şu şekilde bir ifade ile yazmak mümkün olmaktadır.

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

bu ifadenin genel gösterimi şu şekildedir:

ifade1 ? ifade2 : ifade3

Bu gösterimde öncelikle ifade1 mantıksal ifadesi hesaplanır. Eğer mantıksal olarak doğru ise veya 0'dan farklı bir değer hesaplanırsa ifade2 ile gösterilen

Giriş Çıkış İşlemleri

Giriş Çıkış Komutları bir Programlama dili için oldukça önemlidir. Çünkü oluşan çıktıların kullanıcı veya programcıya ulaşması gerektiği gibi kullanıcının da Programa veri vermesi gerekebilir. Ancak gerçekte giriş/çıkış imkanları C dilinin bir parçası değillerdir. C Programlama dilinde Giriş çıkış işlemleri gerçekte komutlar ile değil **stdio.h** kütüphanesinde bulunan fonksiyonlar ile gerçekleştirilir. Bu bölümde standart G/Ç kütüphanesinde yer alan ve C programlarında kullanılan fonksiyonlardan bahsedeceğiz. Fonksiyonlar G/Ç işlemlerini kolaylıkla yapabilecek tanımlara sahipler. Uygulamanın ne kadar kritik olmasına bakmaksızın tüm programcılar bu rutinleri rahatlıkla ve etkin bir şekilde kullanabilirler. Ve son olarak bu fonksiyonlar taşınabilir şekilde tasarlanmışlardır. Yani herhangi bir programcı yazdığı C programını C derleyicisinin bulunduğu herhangi bir işletim sistemi platformuna taşıyabilir.

Bu bölümde G/Ç kütüphanesi içinde yer alan tüm fonksiyonları anlatamayacağız. Bizim için daha önemlisi yoğun kullanılan fonksiyonlarla tanışmak ve bir C programının işletim sistemi çevre ortamıyla etkileşimini sağlayacak kurallardan bahsetmektir.

Standart Kütüphaneye Erişim

Standart G/Ç ortamına müracaat eden tüm C program kaynak kodlarında aşağıdaki satır bulunmalıdır:

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

Bu satır bir C programının en başlarında yer alır. stdio.h dosyası içinde G/Ç kütüphanesi tarafından kullanılan bir takım makroların ve değişkenlerin tanımlanmıştır. <> işaretlerini kütüphane adının etrafında kullanmayı “” işaretlerini kullanmaya tercih ederiz. Çünkü derleyiciye bu kütüphane dosyasının sistem tarafından tanımlanan include klasöründe (başlık dosyalarının bulunduğu klasörde) olduğunu belirtmek isteriz. Eğer “” işaretleri kullanılmış olsaydı bu kütüphane dosyası klasörün içindedir anlamına gelecektir.

Standart Giriş, Standart Çıkış ve Hata Çıkış noktaları

Bir C Programında oluşan değerler standart olarak ekrana yazılır. Bu birime standart çıkış birimi denir (Standart Output).

Bir C Programında girilecek değerler standart olarak klavyeden okunur. Bu birime standart giriş birimi denir (Standart Input).

Bir C Programında oluşan hatalar standart olarak ekrana yazılır. Bu birime standart hata birimi denir (Standart Error).

Bir giriş çıkış fonksiyonu kullanıldığında bu fonksiyonların özelliğine göre standart Giriş veya Standart Çıkış ortamından değer iletişiminde bulunulur.

Standart Giriş Çıkış Fonksiyonları

En basit giriş yöntemi tek bir karakterin standart giriş biriminden okunmasıdır. getchar() fonksiyonu giriş biriminde sıradaki diğer karakteri döndürür. Bir çok işletim sistemi ortamında C programlama dili terminalden/klavyeden değer okuyacağına başka bir dosyadan karakterleri okuyabilir. Eğer bir programda getchar() fonksiyonu kullanılmış ise şu şekilde yazılan bir komut satırı

prog <girisdosyasi

prog adındaki programın ekran yerine girisdosyasi adlı dosyadan girdileri okuyacaktır. Burada "<" işareti girdinin yönlendirildiği ve diğer bir doyanın girdi dosyası olduğunu anlatır. Eğer bu şekildeki bir programa veriler başka bir programdan gelecek ise boru (pipe |) işlemi yapılarak sorun halledilebilir. Şöyle ki:

baskaprog | prog

komut satırı iki programı da çalıştırır ve baskaprog adındaki programın çıktısını prog adındaki programa girdi olarak yollar.

Bir giriş dosyasında dosya sonu olunca getchar fonksiyonu EOF değerini döndürür. EOF değeri Standart Kütüphane içinde –1 olarak tanımlanmıştır. Ancak programda EOF tanımı kullanılması taşınabilirlik açısından daha uygundur.

Çıkış işlemi için putchar(c) fonksiyonunu kullanabiliriz. Bu fonksiyonu tek bir karakteri standart çıkış birimine yazacaktır. Çıkış karakterleri ">" işareti ile yönlendirilebilir. Örneğin:

prog > cikisdosyasi

şeklindeki komut satırında prog isimli programın çıktısı terminale yazılacağına cikisdosyasi isimli dosyaya yazılacaktır. Eğer bu programın çıktısı başka bir programa gönderilecekse boru işareti kullanılabilir.

Bunun haricinde dosyalar üzerinde Okuma/Yazma yapan komutlar ilerdeki konularımızda ele alınıp örneklenecektir. Bu komutları da iki farklı kategoride incelemek hiç de yanlış olmasa gerek.

```
char c;
...
c=getchar();
```

Biçiminde gösterilebilecek şekilde yazılan komutumuz standart giriş biriminden bir karakter okur. Yada

```
char c;
...
c=getc(stdin);
```

şeklinde kullanıldığını görebilirsiniz.

```
char c;
...
/* Tek karakter yazar */
putc(c,stdout);
```

`putchar(c);` Şeklinde yazılır ve c ile ifade edilen tek karakteri standart çıkış biriminden yazar.

Örnek 1

Aşağıdaki Program Klavyeden girilen harfleri ekrana süzerek yazmaktadır. Şöyle ki klavyeden girilen harflerden a harfini ekrana yazmamaktadır ve bu işlemi enter tuşuna basılıncaya kadar devam etmektedir.

```
#include <stdio.h>

main(){
char c;
while((c=getchar())!='\n')

     if (c!='a') putchar(c);
     printf("\n");

}
```

Bir çok program tek bir giriş kaynağından okur ve tek bir çıkış noktasına bu bilgileri yazar. Bu amaçla da getchar, putchar veya printf gibi G/Ç fonksiyonları kullanılır.

Örnek 2

Aşağıdaki örnek klavyeden girilen karakterleri küçük harfe çevirerek ekrana yazan bir programdır.

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

```
main(){
   int c;
   while((c=getchar()) != EOF)
      putchar(isupper( c ) ? tolower( c ) : c);
}
```

Bu örnekteki isupper ve tolower makroları stdio.h kütüphanesinde tanımlanmış olan fonksiyonlardır. Bu fonksiyonlardan isupper kendisine gönderilen karakterin büyük harf olması durumunda doğru değer yanlış olması durumunda yanlış değer döndürür. Tolower fonksiyonu ise kendisine gönderilen karakteri küçük harfe çevirir.

Biçimlendirilmiş Çıkışlar

İstenilen değerlerin uygun şekilde ve uygun yerlerde görünebilmesi için Biçimlendirilmiş çıkış Komutu kullanılabilir.

```
printf("Biçim Tanımları ve Sabit
ifadeler"[,Değişken1[,Değişken2...]])
```

şeklinde kullanılan komut bu amaç için son derece uygundur. Bu Fonksiyon Sabit ifadelerin ve değişkenlerin belirlenmiş biçim tanımlarına göre ekrana yazılmasını sağlar. Biçim tanımları ve Sabit ifade dediğim karakter dizisinde iki farklı türde bilgi bulunur. Bunlar normal karakterler ile ifade edilen ifadeler ve c için bir anlam taşıyan dönüşüm ifadeleri. Bu dönüşüm ifadeleri istenilen bir değişkenin parametre olarak yazılmış ise istenilen bir biçimde ekrana yazılmasını sağlar.

```
printf("Merhaba %s Şu anda Hava %d sıcaklığında\n", ad, hava);
```

Bu deyim **Merhaba Ahmet Şu anda Hava 23 sıcaklığında** gibi bir ifadeyi ekrana yazacaktır. Bu satırda yer alan ve % ile başlayıp bir dönüştürme karakteri ile biten ifade dönüşüm yapan biçim tanımıdır. % işareti ile dönüştürme karakteri arasında yer alan bölümde şu karakterler bulunabilir:

- - işareti : yazılacak ifadenin sola yanaşık yazılmasını sağlayacaktır.
- Sayısal bir değer : yazılacak ifadenin ekranda kaç basamaklık alana sığacağını belirleyen sayısal bir değerdir. Eğer ifade belirlenen yere sığamayacaksa sol taraftan kırpılacaktır. . işareti : alanı kesirli sayısal alandan ayıran nokta.
- l işareti : Biçim tanımının uzun olup olmadığını veren bir tanımlayıcı .

Biçimlendirme tanımları ya da dönüştürme karakterleri için aşağıdaki tabloda yer alan tanımları kullanılmaktadır.

Biçim	Tanım

d	Karşılık olarak gelen değişkenin değerini sayı olarak ekrana yazar.
o	Karşılık olarak gelen değişkenin değerini 8'lik sayı olarak ekrana yazar.
x	Karşılık olarak gelen değişkenin değerini 16'lık sayı olarak ekrana yazar.
u	Karşılık olarak gelen değişkenin değerini işaretsiz bir değer olarak ekrana yazar.
c	Karşılık olarak gelen değişkenin değerini tek bir karakter olarak ekrana yazar.
s	Karşılık olarak gelen değişkenin değerini bir karakter dizesi olarak ekrana yazar. Bu karakter dizesi dize sonu anlamına gelen null karakteri bulununcaya kadar yazılır.
e	Karşılık olarak gelen değişkenin değerini bilimsel gösterimdeki sayı olarak ekrana yazar.
f	Karşılık olarak gelen değişkenin değerini kesirli sayı olarak ekrana yazar.
G	%e veya %f'den kısa olanını kullanmasını sağlar
Eğer % işaretinden sonra kullanılan işaret dönüştürme karakteri değil ise ve normal bir karakter olarak % işareti kullanılacaksa %% şeklinde kullanılmalıdır	
Kullanılan Özel Karakterler	
\n	Yeni Satır Karakteri
\t	TAB Karakteri
\\	\ Karakteri
\"	" (Tırnak) Karakteri

Biçimlendirilmiş Girişler

İstenilen Değer ve ifadelerin bilgisayara girilmesini sağlar. Kullanım olarak printf ile aynı şartları sağlar. Ve bilgisayara giriş yönü ile çalışır.

```
scanf("Biçim Tanımları"[,Değişken1[,Değişken2...]])
```

şeklinde kullanılır. scanf standart giriş biriminden karakterleri okur ve verdiğiniz biçim tanımlarına göre okunan karakterleri yorumlar ve değerleri ilgili değişkenlere yerleştirir. Ancak dikkat edilmelidir ki Değişken olarak ifade edilen değişkenlerin adresleri bu fonksiyona gönderilmelidir. Burada değişkenleri her biri biçim tanımları ile belirlenen tanımlara birebir örtüşmek zorundadır. printf'de görülen tüm biçim tanımlayıcıları scanf için de geçerlidir. Değişkenler ise adresleri ile verilmelidir. Eğer değişken basit değişken ise bu adres değeri "&" operatörü ile bulunmalıdır. Eğer değişken

karakter dizesi gibi bir değişken ise sadece değişken adını vermek yeterli olacaktır.

Örnek 3

Bu Program klavyeden farklı özelliklere sahip değerler okuyup farklı özellikler ile ekrana yazmaktadır.

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

main(){
char ifade[100];
int k,uzunluk;
float kesir;
char cevap;
printf("Ad, Sıcaklık, d1, kesir, cevap değerlerini
giriniz?\n");
scanf("%s %d %d %f %c", ifade, &k, &uzunluk, &kesir, &cevap));
/* Bu komut aralarında boşluk olmak üzere bir dizi değeri
karşılık değişkenlere aktaracaktır.*/
printf("Sonuç\n%s\n%d\n%d\n%f\n%c", ifade, &k, &uzunluk,
&kesir, &cevap));
}
```

Alıştırma 2

int k;
char ad[20];
scanf(“....”,&k,ad);
Şeklinde yazılmış bir ifade de .... ile belirlenmiş yere ne konmalıdır.

Alıştırma 3

Klavyeden girilecek iki tam sayı ve bir kesirli sayı için yazılması gerekecek **scanf** fonksiyonunun parametreleri nelerdir?

Bellek Biçim dönüşüm fonksiyonları

Scanf ve printf fonksiyonlarının benzeri olarak sprintf ve sscanf şeklinde fonksiyonlar tanımlanmıştır. Ancak bu fonksiyonlar çıktıyı ekrana yazacağına bir karakter dizesine yerleştirir veya girdiği klavyeden okuyacağına bir karakter dizesinden okur. Fonksiyonları tanımları şu şekilde verilebilir:

sprintf(karakter dizesi değişkeni, "Biçim ve sabit tanımları", değişken listesi)

sscanf(karakter dizesi değişkeni, "Biçim tanımları", değişken listesi)

Karakter dizesinin Okunup-Yazılması

Bazen tek bir karakter veya tek bir kelime değil de Enter tuşuna basılıncaya kadar ya da satır sonu işaretini buluncaya kadar tüm yazılanların okunması gerekebilir. Bu nedenle şu fonksiyonları kullanmak yerinde olacaktır.

```
char ad[30];
...
gets(ad);//Bir karakter dizesini klavyeden okur
puts(ad);//Bir karakter dizesini ekrana yazar
```

Örnek 4

Bu Örnek'te Klavyeden girilen bir karakter dizesinin tersi bulunarak ekrana yazılmaktadır.

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

main(){
char ifade[100];
int k,uzunluk;
puts("İfadenizi Giriniz\n");
gets(ifade);
uzunluk=strlen(ifade);
for(k=uzunluk-1;k>=0;k--)

     putchar(ifade[k]);

putchar('\n');}
```

Önemli bir takım fonksiyonlar

stdio.h kütüphanesi içinde kullanışlı olabilecek bir takım fonksiyonlar bulunur.

Karakter Sınama ve dönüştürme fonksiyonları

isalpha(c) c karakter bilgisi taşıyorsa 0'dan farklı bir değer değilse 0 döndürür.

isupper(c) c karakteri büyük harf ise 0'dan farklı bir değer değilse 0 döndürür

islower(c) c karakteri küçük harf ise 0'dan farklı bir değer değilse 0 döndürür

isdigit(c) c rakam bilgisi taşıyorsa 0'dan farklı bir değer değilse 0 döndürür.

isspace(c) c boşluk bilgisi taşıyorsa 0'dan farklı bir değer değilse döndürür.

Toupper (c) c karakterini büyük harfe çevirir.

Tolower (c) c karakterini küçük harfe çevirir.

ungetc

standart giriş ortamından veya herhangi bir dosyadan okunmuş karakteri geri o ortama yazan fonksiyondur.

ungetc(c, dosyaisaretcisi);

Program Akışı ve Program Akış Denetimi

C Programlarında Program satırları diğer programlama dillerinde olduğu gibi yukarıdan aşağıya icra edilirler. Her komut sonunda ; (noktalı virgül) karakteri bulunur. Blok yapılı bir dildir ve bu bağlamda değişkenler tanımlandıkları blok içerisinde etkilidirler. Birden fazla komutlar {...} simgeleri arasında ele alınarak ek bir komutmuş gibi bloklanabilir. Belki daha önce tanıştığınız komutları burada daha ayrıntılı bir şekilde incelemeye çalışacağız.

Komutlar ve Bloklar

C Dilinde Noktalı virgül komut sonlandırma belirtecidir.

```
x=0;
i++;
printf(.....);
```

yukarıdaki her bir ; komutun bittiğini ve diğer komuta geçildiğini sembolize eder. Komutları bloklamak için {...} küme parantezleri kullanılmaktadır. Bu parantezler arasında yazılan komutlar kaç tane olursa olsun tek bir komutmuş gibi algılanır. if, else, for, while gibi komutların hemen peşinden ; kullanılmaz. Ayrıca bir bloğu bitiren } işaretinin peşinden de noktalı virgül konulmaz. Bu da çözümlemede istenilen kısımların kapalı bir kutu imiş gibi gösterilmesini sağlar.

If-Else komutu

Karar vermeyi sağlayan Komut'tur. Temel Yapısı

```
if (mantıksal_ifade)

     Komut-1;

else

     Komut-2;
```

Bu yapıda else kısmı seçimliktir. Yani kullanılmasa da olur. Komutun çalışması şu şekilde izah edilebilir:

- Öncelikle mantıksal ifade hesaplanır ve bir değer bulunur.
- Bu değer 0 haricinde bir değer ise (yani mantıksal ifade doğru ise) Komut-1 ile gösterilen komut uygulanır

- Bu değer 0 ise (yani mantıksal ifade yanlış ise) else'den sonraki komut-2 ile gösterilen komut uygulanır.
- Eğer else kısmı yok ise ve değer 0 ise Komut-1 uygulanmadan ;'den sonraki komutlardan devam edilir.

Gerçekte if komutu bir sayısal değişkenin değerini kontrol ediyorsa bir takım kısaltmalar kullanmak mümkündür. Örneğin;

if (değişken !=0) ifadesi yerine
if (değisken) yazılabilir. Ancak tercihen birinci yazılan daha anlaşılır ve alışılagelmiş olanıdır.

Bu yapıda Komut-1 veya Komut-2 ile gösterilen yerlerde birden fazla komut uygulanmak zorunda ise bu komutlar küme parantezleri içerisine alınıp tek komutmuş gibi gösterilebilir.

Bu yapıda else kısmı seçimlik olduğu için iç içe if yazımlarında sorunlarla karşılaşabilirsiniz. Özellikle bir if komutunun else tümcesi yok ise belirsizlik ortaya çıkar. Bu durumda else son if'e aittir. Daha doğrusu if'ler içten dışa doğru kapatılır. Şöyle ki:

```
if (n>0)
   if (a>b)
      z=a;
   else
      z=b;
```

komut topluluğunda else komutu ikinci sıradaki if komutuna aittir. Daha anlaşılabilir olsun diye komutların yazımı sırasında girinti çıkıntılara dikkat edilmiştir. Eğer anlatılmak istenen bu değil de başka bir sıra idiyse küme parantezleri kullanılabilirdi.

```
if (n>0) {
   if (a>b)
      z=a;
}
else
   z=b;
```

bazı durumlarda bu belirsizlikler hatalara dahi neden olabilir. Şöyle ki

```
if (n>0)
    for(i=0; i<n; i++)
        if (s[i]>0) {
           printf("....");
           return(i);
        }
else     /* Yanlış İfade */
       printf("n negatif olamaz \n");
```

Girinti çıkıntıların konumuna göre yaptığınız ile derleyicinin anladığı farklı olacaktır. Derleyici ikinci if komutunun else komutunu yazdığınızı sanacak ve kodu buna göre üretecektir.Böylece programınızda ciddi mantıksal bir hata yapmış olursunuz. Bu tür bir hatayı ayıklamak oldukça zordur.

Örnek 5

Bu örnekte klavyeden girilen iki sayısal değer karşılaştırılıyor, büyük olandan küçük olan çıkarılıyor ve sonuç ekrana yazılıyor.

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

main(){
int a,b;
printf("A için bir değer giriniz?");
scanf("%d",&a);
printf("B için bir değer giriniz?");
scanf("%d",&b);
if (a>b)

     printf("A-B=%d\n", a-b);

else

     printf("B-A=%d\n", b-a);

}
```

Else – if komutu

Temel yapısı şu şekildedir.

```
if (koşul)
    komut;
else if (koşul)
    komut;
else if (koşul)
    komut;
else
    komut;
```

Bu şekilde çoklu seçim şeklinde bir dizi if değimi arasından bir tanesi seçilecektir. Koşullar sıra ile kontrol edilir, koşullardan ilk doğru olanın ilişiğindeki komut icra edilir. Bu doğru koşulun bulunup çalıştırılmasından sonra tüm bloğun çalışmasını bitirir. Her bir komut ile ifade edilen bölüm tek başına bir komut olabileceği gibi küme parantezleri arasındaki bir çok komut ta olabilir.

Komutların sonundaki else değimi koşullardan hiçbirinin gerçekleşememesi durumunda icra edilir. Bazen koşullardan hiçbirinin doğru olmadığı durumlarda yapılacak değişik bir komut bulunmayabilir. Bu durumda bu satırlar (en son else satırı) yazılamayabilir. Bazen de bu durum ile asla karşılaşılmayacaksa "imkansız durumu" görebilmek için yazılabilir.

Etiketler ve goto komutu

C için hiç tercih edilmeyen bir komut olan bu komut program akışını bir noktadan diğer bir noktaya yönlendirir. C Programlama dili çok etkili blok yapılı bir dildir. Güçlü Döngü Komutları, Fonksiyonel yapı ve hatta harici kütüphaneler oluşturup bunları programa bağlama işleminden dolayı böyle bir komuta kesinlikle ihtiyaç bulunmamaktadır. Hatta onsuz program yazmanın C'de çok kolay olduğunu söyleyebiliriz.

Yine de eski Programlama alışkanlıkları olanlar için bu komut kullanılabilir bir komut olarak C dili içerisinde vardır.

```
etiket:
.......
goto etiket;
```

Şeklinde kullanılmaktadır. Öncelikle goto komutu ile sapılacak veya dallanılacak konuma bir ad verilir. Bu konum için verilecek ad değişken isimlendirme kurallarındaki gibi adlandırılabilir. Bu adı (: -iki nokta üst üste) sembolü takip etmelidir. Döngü oluşturmak için tercih edilmese de belki kullanılması gerekecek birkaç nokta söylenebilir. Şöyle ki:

İç içe birden fazla döngü olan bir yapıda en dış döngüye ulaşmak için...

Hata ayıklama amacıyla oluşturulacak yapılarda kullanmak için...

Zorunlu olmasa da bir grup sayının içinde aradığınızı bulduğunuzda bu durumu anlatabilmek için...

Alıştırma 4

Klavyeden 0-Sıfır girilinceye kadar girilen bir grup sayının karelerinin ortalamasını bulacak program için C Programlama dili ile geliştiriniz.

Alıştırma 5

Klavyeden girilecek iki Pozitif tam sayının OBEB (Ortak Bölenlerin En Büyüğü)'ini bulacak **C Programını** geliştiriniz. Örneğin elimizde 3654 ve 1365 değerleri olsun. Bu değerlerin OBEB'i şu şekilde bulunmaktadır.

3654 / 1365 Kalan 924
1365 / 924 Kalan 441
924 / 441 Kalan 42
441 / 42 Kalan 21
42 / 21 Kalan 0

Bu işlem sırasını ve yineleme özelliklerini kullanarak iki sayının OBEB'ini bulunuz. Bu örnekte OBEB 21 değeridir.

Alıştırma 6

Klavyede girilen kesirli bir sayıyı a/b şeklinde rasyonel olarak ifade edip ekrana yazabilecek programı c ile geliştiriniz. Örneğin klavyeden girilecek olan 0.5 değeri için 1/2 değerini ekrana yazmalıdır.

switch komutu

Switch Komutu birden fazla değerden seçim yapmak için kullanılabilecek özel karar verme komutudur. İç içe fazla sayıda if..else komutunu kullanmayı gerektiren durumlarda kullanılabilecek etkili bir karar verme aracıdır.

```
switch(ifade){
case sabit_değer1: [Komut1;]
case sabit_değer2: [Komut2;]
case sabit_değer3: [Komut3;]
case sabit_değer4: [Komut4;]
case sabit_değer5: [Komut5;]
......
case sabit_değerN: [KomutN;]
default: [Komut;]
}
```

Örnek 6

Aşağıdaki program bir ifade içindeki rakamları, boşlukları ve diğer karakterleri sayan bir programdır.

```
#include <stdio.h>
int c,i,nwhite,nother,ndigit[10];
main(){
   nwhite = nother = 0;
   for (i=0; i<10;i++)
      ndigit[i]=0;
   while ((c=getc(stdin)) != EOF )
      switch ( c ){
         case '0' :
         case '1' :
         case '2' :
```

```
            case '3' :
            case '4' :
            case '5' :
            case '6' :
            case '7' :
            case '8' :
            case '9' :
                    ndigit[c-'0']++;
                    break
            case ' ' :
            case '\n':
            case '\t' :
                    nwhite++;
                    break;
            default :
                    nother++;
                    break;
        }
        printf("digits = ");
        for (i=0;i<10;i++)
            printf(" %d",ndigit[i]);
        printf("\nwhite space = %d, other =
%d\n",nwhite,nother);
}
```

Bu komut ifade ile belirtilen tamsayı ifadeyi hesaplar. Yukarıdaki örnek bu c karakteridir. Sonrasında bu hesaplanan değer case ile belirtilmiş olan değerlerden birine eşit mi diye kontrol eder. Eğer eşit olan bir case ifadesine rastlarsa onun karşısındaki ve diğer altındaki komutları icra eder. Eğer burada belli bir kısım komutların uygulanması bir kısmının uygulanmaması isteniyorsa switch değiminden break komutuyla çıkılır. Eğer bu case değerlerinden hiçbiri eldeki değere eşit değilse default ile gösterilen komut uygulanır.

Her case değiminin yanında bir komut bulunmak zorunda olmadığı gibi birden fazla komutu {...} ifadeleri arasına yazmadan da kullanabilirsiniz. Ancak bu durumda komutların uygulamasının bitmesini istediğiniz yere break komutunu koymayı unutmamanız gerekecek.

break komutu switch bloğundan hemen çıkmanızı sağlayan bir komuttur. Her bir case etiket gibi davrandığı için biri karşısındaki işlem gerçekleştirildikten sonra diğerlerinden devam eder. Devam edilmemesini sağlamak amacıyla break veya return değimleri ile buradan çıkılması gerekir. Daha sonra göreceğimiz üzere break komutu while, for, do döngülerinden de çıkılmasını sağlar.

Bu case değimlerinden devam edip gidilmesi kargaşaya yol açacak gibi görünse de bazen olumlu sonuçları olabilir. Şöyle ki birden fazla eylemi tek

bir case satırında kullanmanız mümkün olabilmektedir. Veya birden fazla case satırı için aynı satırların çalıştırılmasını sağlamaktadır.

İyi programlama örnekleri için son case satırından sonra da break komutu kullanmak gerekebilir. Mantıksal olarak gereksiz görünse de bu daha sonra buraya eklenecek yeni case satırlarına yanlışlıkla dallanmayı engelleyecek be mantıksal hataların oluşumunu azaltacaktır.

Alıştırma 7

Klavyeden girilen iki sayı ve bir operatöre göre işlem yapıp sonucu ekrana yazan bir C Programı geliştiriniz.

Alıştırma 8

Klavyeden girilecek olan ay ve yıl bilgisine göre ayın kaç gün çektiğini bulacak C Programını geliştiriniz.

Döngü Komutları - while

Bir koşul sağlanıncaya kadar bazı komutların tekrar edilmesini sağlayan döngü komutudur. Bu Döngü komutunun özünde sadece koşul sınama bulunmaktadır. Değer arttırma veya ilk değer atama gibi gerekli olabilecek diğer işlemler ayrıca belirtilmelidir.

```
while(koşul)
{

      komut1;
      komut2;
      ......

}
```

Şeklinde kullanılır ve koşul doğru olduğu sürece komutların uygulanmasını sağlar. Koşul ile belirtilen ifade mantıksal bir ifade olup doğru olduğu sürece komut1, komut2 vb komutların tekrar tekrar uygulanmasını sağlar. Ta ki koşul yanlış oluncaya kadar. Koşul yanlış olursa veya sayısal olarak 0 değerini taşırsa program akışı döngü komutundan sonraki komuttan devam eder.

Örnek 7

Aşağıdaki örnekte Fibonecci dizisini bir N değerine kadar ekrana yazan program while komutu kullanılarak yazılmıştır.

```
#include <stdio.h>
int a,b,c;
```

```
int n;
main(){
   printf("n değerini klavyeden giriniz");
   scanf("%d",&n);
   a=1;
   b=1;
   printf("%d\n%d\n",a,b);
   c=a+b;
   while(c<=n){
      printf("%d\n",c);
      a=b;
      b=c;
      c=a+b;
   }
}
```

Alıştırma 9

Sıfır – 0 girilinceye kadar klavyeden girilen tüm değerlerin ortalamasını bulup ekrana yazan C programını geliştiriniz.

Alıştırma 10

```
t=0;
while(t<10){
   printf("%d\n",t);
   t++;
} şeklindeki bir ifadeyi for komutu ile yazmaya çalışınız.
```

Döngü Komutları - for

C Programlama dili içerisinde son derece güçlü bir döngü komutudur. Belli bir koşul sağlanıncaya kadar otomatik olarak artan değerlerle döngü oluşturmayı sağlar.

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

     komut1;
     komut2;
     ......
     }
```

Şeklinde kullanılır.

Bu komut while ile aşağıdaki gibi ifade edilebilir

ifade1;

```
while(ifade2){
   komut1;
   komut2;
   ifade3;
}
```

Teknik olarak ifade olarak tanımlanmış ifadelerden ifade1 ve ifade3 aritmetik ifadeleri anlatır. İfade2 ise mantıksal bir karşılaştırma ifadesidir. Bu üç ifadeden her hangi biri yazılmadan pas geçilebilir. Ancak bu durumda bile noktalı virgüller yerlerinde kalmalıdır.eğer ifade1 ve ifade3 yazılmazsa ve hatta ifade2 for komutu içine yerleştirilmezse ortaya çıkan komut doğrudur. Şöyle ki

```
for(;;){
   ...
}
```

şeklinde yazılan komut bir sonsuz döngüyü anlatır. Bu tür bir döngüden break veya return gibi bir komutla çıkılacağı kabul edilir.

Burada değişken ile belirtilen ilk kısım tek bir değer aktarma ifadesi yerine virgüllerle ayrılmak sureti ile birden fazla değer aktarma ifadesini barındırabilir. Bu kısım bir ilk değer aktarma ifadesi olup döngü uygulanmaya başlamadan önce icra edilir.

İkinci kısımda kullanılan koşul döngünün sonlanma koşuludur. Bu koşul doğru olduğu sürece döngü yinelenir.

Son Kısımda kullanılan değer aktarma değimleri aralarında virgül bulunmak koşulu ile birden fazla olabilir. Bu kısım ise döngü içindeki komutların her uygulanmasından sonra icra edilen işlemlerdir. Çoğunlukla değer artıran ifadelerdir.

```
for(x=0;x<10;x++){
printf("Sıradaki Sayı %d\n",x);
}
```

Örneği Sıfırdan 9'a kadar sayıları ekrana alt alta yazan bir program paçasıdır.

Acaba nerede while nerede for komutlarını kullanmalıyım? Eğer değişkenlere ilk değer atama ifadeleri yok ise ve döngünün kaç defa uygulanacağı hakkında sayısal net bir tanım söylenemiyorsa kullanıcı davranışlarına bağlı bir koşul ile ifade edilebiliyorsa while komutu uygun bir komuttur. Eğer ilk değer atama ifadeleri var ve döngünün çalışması konusunda sayısal bir takım değerler ifade ediliyorsa for komutu daha uygun bir komuttur. Çünkü for komutu gerek ilk değer atamalarını gerekse de koşulu ilk satır üzerinde gösterdiği için daha kolay anlaşılır ve derli toplu olarak karşımızda duruyor. Bu da döngü komutumuzun merkezileşmesini ve programın kontrolünün kolaylaşmasını sağlar.

Örnek 8

Bu örnekte klavyeden girilen bir N değerinin faktöryeli for komutu kullanılarak hesaplanmıştır.

```
#include<stdio.h>
int f,n,x;
main(){
   printf("n değerini giriniz");
   scanf("%d",&n);
   for(f=1,x=1;x<=n;x++) f*=x;
   printf("Faktöryel=%d\n",f);
}
```

Bu örnekte aslında faktöryeli hesaplayan kesim sadece for komutunun yazılı olduğu satırdır.

Örnek 9

Klavyeden girilecek X değerinde N değerine kadar tüm doğal sayıları ekrana listeleyen C Programını geliştiriniz.

```
#include<stdio.h>
int x,n,k;
main(){
   printf("X değerini giriniz");
   scanf("%d",&x);
   printf("N değerini giriniz");
   scanf("%d",&n);
   for(k=x;k<=n;k++) printf("%d\n",k);
}
```

Alıştırma 11

Alıştırma 4 örneğini goto komutu kullanmadan while veya for ile yeniden yazınız

Alıştırma 12

Alıştırma 5 örneğini goto komutu kullanmadan while veya for ile yeniden yazınız

Alıştırma 13

Sadece toplam ve çıkarma kullanarak iki sayıyı çarpan ve sonucu ekrana yazan C Programını geliştiriniz.

Alıştırma 14

Sadece toplam ve çıkarma kullanarak iki sayıyı bölerek bölüm ve kalanı bulan ve sonucu ekrana yazan C Programını geliştiriniz.

Alıştırma 15

Klavyeden girilecek 20 sayının tek olanlarını ayrı çift olanlarını ayrı toplayıp sonucu ekrana yazan bir C programı geliştiriniz.

Alıştırma 16

F(x)= $\sum_{x=1}^{n} \frac{1}{x^2}$ şeklinde tanımlanmış bir fonksiyonda girilen n değeri için sonucu hesaplayan ve ekrana yazan C programını geliştiriniz. Programı yazarken kesinlikle goto komutu kullanmayınız.

Alıştırma 17

Klavyeden girilen bir tam sayının tüm tam bölenlerini bulup ekrana listeleyen bir C Programı geliştiriniz.

Alıştırma 18

```
for (x=1,f=1;x<10;x++) f*=x;
şeklindeki program parçasını while komutunu kullanarak yazınız
```

Alıştırma 19

Cos(x)=$1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}$ şeklinde tanımlanmış bir açılım için klavyeden girilecek bir x değerine göre ilk 10 terim için sonucu hesaplayıp ekrana yazacak C programını geliştiriniz. Üs alma işlemi için math.h kütüphanesindeki pow fonksiyonunu kullanınız.

Döngü Komutları - do...while

while ve for komutlarında döngü sonlanma koşulu döngünün en başında sınanır. Ancak bu üçüncü döngü komutu olan do komutu döngü sonlanma

koşulunu döngü sonunda sınar. Bu nedenle ve döngü komutunun genel görünüşünden dolayı döngü komutu içindeki komutlar en az bir kez icra edilirler.

```
do
{
komut1;
komut2;
.......
}
while (koşul);
```

Öncelikli olarak komutlar icra edilir sonra koşul sınanır. Doğru ise komutlar tekrar uygulanır yani aslında döngü komutları tekrar icra edilir. Eğer ifade yanlış olmuşsa döngü son bulur.

Tahmin edebileceğiniz gibi do komutu for ve while komutuna göre oldukça az kullanılan bir komuttur. Ancak tabi ki bazı durumlarda oldukça etkili bir döngü komutudur.

Örnek 10

Aşağıdaki örneği inceleyelim. Bu örnekte kullanıcı klavyeden 0 ile 100 arasında bir sayı girmeye zorlanıyor. While ile yazılmış olan koşul inceleme satırına dikkat ediniz. Eğer x<0 veya x>100 ise döngü içindeki komut tekrar ediliyor. Çünkü bu istemediğimiz bir durum. Asıl istenen kullanıcının geçerli bir vize notu girebilmesidir.

```
#include <stdio.h>
int x;
main(){
   do {
      printf("0 ile 100 arasındaki vize notunu giriniz");
      scanf("%d", &x);
    }while(x<0 || x>100);
}
```

yukarıdaki do...while bloğu klavyeden girilecek değerin 0...100 aralığında olmasını zorlar. Ve bu aralıkta değilse tekrar değer girilmesini sağlar.

Döngü Çıkış Komutu - break

Normal olarak bir döngü koşulda oluşan duruma göre sonlanır veya devam eder. Döngünün başı ve sonunu belirleyen ifade veya küme parantezlerine göre döngü içi ve dışı ayrılmış durumdadır. Eğer bir döngüden herhangi bir anda çıkma ihtiyacı oluşursa bu komut kullanılabilir. Bu komut icra edildikten sonra döngünün sonunu belirten noktadan sonraki ilk komuttan program akışı devam eder.

```
break;
```

Şeklinde Kullanılır. Bu komut en içteki döngüden çıkmayı sağlar.

Örnek 11

```
#include <stdio.h>
int i;
main(){
   i=0;
   while (1){
      if(i >100) break;
      printf("%d\n",i);
      i++;
   }
}
```

Döngü devam Komutu - continue

Break komutu tarzında bir komuttur. Bir döngüde bu komutun bulunduğu noktadan öteye komutların kullanılmamasını sağlar. Onun yerine döngü başına dönüp oradan program akışını devam ettirmeyi sağlar.

```
continue;
```

Şeklinde Kullanılır.

Tüm döngü komutlarında kullanılabilirken break'den farklı olarak switch komutu ile kullanılamaz. Döngüdeki bir sonraki iterasyonun yapılmasını sağlar. Örnek olarak:

Örnek 12

Aşağıdaki örnek i değeri çift olan durumlar için döngü başından devam edilmesini sağlar. Yani ekrana sadece tek sayıların yazılmasını sağlar.

```
#include <stdio.h>
int i;
main(){
   i=0;
   while (i < 100){
      if(i % 2 == 0) continue;
      printf("%d\n",i);
      i++;
   }
}
```

Fonksiyonlar ve Genel Program Yapısı

Fonksiyonlar büyük programları küçük parçalara bölen görevlerdir. Ayrıca daha önce insanların üretmiş olabilecekleri çözümleri sizin tekrar baştan keşfetmemenizi sağlarlar. İlgili fonksiyonlar genellikle detayları gizler. Zaten bu detayların programcı tarafından bilinmesinin bir anlamı yoktur.

C fonksiyon kullanımını verimli ve kolay kullanımlı bir şekilde tasarlamıştır. Genellikle C programları küçük fonksiyonlardan oluşmuştur. Bu arada bir C programı bir yada daha fazla kaynak dosyasından oluşabilir. Bu kaynak dosyalar ayrı ayrı derlenmiş ve birlikte yükleniyor olabileceği gibi hep birlikte de bağlanmış olabilir.

Aslında C programlama dili ile yazılan tüm programlar bir grup fonksiyondan ibarettir. Hatta C dilinin komutlarından hariç çok kullanışlı bir yığın fonksiyon tanımları söz konusudur. Belki de C dilini güçlü kılan yapılardan biri de bu fonksiyon tanımlama ve fonksiyonları istenilen yerlerde kullanmadır.

Bir çok programda kütüphanelerin içinde yer alan fonksiyonları kullanmışınızdır. getc, putc, sin, cos vb. fonksiyonlar bunlardandır. Bu bölümde biz daha çok bu fonksiyonları kullanmaktansa bunları tanımlamayı öğreneceğiz. Bu şekilde kendinize ait program kütüphaneleri oluşturabilir ve bu program parçalarını bir yerden başka bir yere taşıyabilirsiniz.

Ana program bloğu (main)

Bu kadar fonksiyon içerisinde şunu belirtmek gerekir: Aslında C programının ana bloğu da bir fonksiyon tanımıdır. Yani main diye adlandırılan bir fonksiyondur. Her Programda mutlaka bulunması gerekir ancak parametre almak zorunda değildir ve değer döndürmek zorunda da değildir.

```
tanımlar.......
main(){
......
/* Program ile ilgili ana giriş ifadeleri buraya
yazılacaktır.*/
......
}
```

C programı genel yapısı

```
#include tanımları
#define tanımları
Global değişken tanımları
fonksiyon_adı(){
.......
/*Fonksiyon tanımı buraya*/
.......
}
/* Bu programda kullanılabilecek tüm fonksiyon tanımları */
main(){
......
/* Program ile ilgili ana giriş ifadeleri buraya
yazılacaktır.*/
......
}
```

#include Tanımları

Programda kullanılacak harici ya da önceden tanımlanmış fonksiyonların bulunduğu kütüphanenin tanımlandığı ifadelerdir.

Bu cins kütüphane stdio.h, string.h, conio.h time.h vb. Kütüphaneler olup bu kütüphanelerin herbiri bir işlem grubu belirleyen fonksiyonların bir arada bulunduğu dosyalardır.

Bu kütüphaneler, önceden tanımlanmış kütüphaneler (Programlama dili geliştiricileri tarafından) olabileceği gibi, sizin oluşturduğunuz bir kütüphaneler de olabilir.

#define Tanımları

Bir Program içerisinde kullanmak isteyebileceğiniz sabit ifadeleri tanımlamak için kullanılan sözcüktür. Hemen ilk kısımda yer alır.

Diğer bölümler

Tüm program içerisinde kullanılacak değişkenler main bloğundan önce tanımlanmalıdır. Bu şekilde tanımladığınız değişkenler Global olurlar ve tüm program içerisinde tanımlı hale gelirler.

Daha sonra fonksiyonlar tanımlanır. Bu fonksiyonların tanımlanması bir sonraki bölümde ele alınmıştır.

Ana Program bloğunu tanımlanması ve kodlarının yazılması ile program tamamlanmış olur. aslında **main()** diye adlandırılan bu blok normal bir fonksiyon tanımında olduğu gibi tanımlanır. Ancak adı özeldir ve

değiştirilemez veya değişik amaçlar için kullanılamaz. Ayrıca bir C programı çalıştırılmaya başlandığında ilk olarak bu blok uygulanır. Diğer fonksiyonlar veya program parçaları buradan çağrılır veya buradan icra edilirler. Bu anlamda özel bir fonksiyondur. main() fonksiyonu. Eğer çalıştırılırken parametre ile kullanılan bir program olarak çalıştırılırsa ()'ler arasında gelen ifadeler o programın parametreleridir.

Temeller

Program birbirinden bağımsız bir kısım fonksiyon tanımlarından oluşur. Fonksiyonlar arasında haberleşme, parametreler ve fonksiyonların döndürdüğü değerler ile gerçekleşir. Bazen de bu değer transferi harici değişkenler ile gerçekleşir.

Fonksiyonlar kaynak kodda herhangi bir sırada yazılabilir. Hatta kaynak kod birden fazla dosyaya bölünebilir. Önemli olan fonksiyonların bölünmemesidir.

Return değimi fonksiyondan değer döndürmeyi sağlayan ve çağrıldığı yere değeri döndüren değimdir. Return değimini herhangi bir ifade izleyebilir.

```
return (ifade);
```

Çağıran fonksiyon dönen değeri dikkate almama hususunda serbesttir. Fonksiyonda return yanına değer yazmak da zorunlu değildir bu durumda çağıran fonksiyona değer döndürülmez.

Fonksiyon, çağrıldığı noktaya değer ya da hizmet döndüren program parçası olarak tanımlanabilir. Kullanım amacı bir işlemin çok sık tekrar edilmesinden dolayı paketlenmesini sağlamak ve kullanım anında sadece bu paketin adını kullanmaktır. Bu yöntem ile aynı işi yapan bir program parçası çok defa yazılacağına bir defa yazılıp çok yerde kullanıma sunulmuş olur.

Bu yöntemle yapısal programlar geliştirebilirsiniz. C dili ile en pratik program yazmanın yolu fonksiyonlardan geçer. Özellikle çok yoğun ve karmaşık programların yazılmasında hem yazımı kolaylaştırmak hem de anlaşılırlığı arttırmak için fonksiyon kullanımı son derece idealdir. Ayrıca sorunları çözümlerken fonksiyonel kavramlarla çözümlemek sorunun kolay çözümlenebilmesine neden olacaktır.

Bir Fonksiyon tanımı genel olarak aşağıdaki yapıya sahiptir.

```
[fonksiyonun türü] fonksiyonun_adi([P1[,P2..]])
[değiskentürü P1;
degiskentürü P2;
.................]
{
```

```
Fonksiyona ait değişkenler ve tanımları
Fonksiyona ait yapılacak işlemler ve komutlar
[return değer];
}
```

Bu tanımda fonksiyon eğer bir değer döndürmüyor ise fonksiyonun_türü denilen kısım boş olabileceği gibi eğer fonksiyon parametre almıyor ise P1, P2 gibi ifadelere ihtiyaç bulunmamaktadır.

Eğer fonksiyon değer döndürmüyor ise en sondaki **return değer** ifadesine gerek yoktur. burada **değer** denilen ifade fonksiyonun döndüreceği değeri sembolize eder.

Eğer fonksiyonun parametreleri yok ise bu parametrelerin tür tanımı yapılmaz. Burada tanımlanan bu parametreler sadece fonksiyon içerisinde kullanılabilir.

Parametre: Bir fonksiyonda farklı değerler için farklı sonuçlar elde etmemizi sağlayan ve fonksiyona gönderilen değerlere verilen addır.

Fonksiyon parametreleri hakkında

Şu ana kadar gördüğümüz fonksiyonlar parametreleri değerle çağrılmışlardı. Bu yönteme call by value denir ve çağrılan fonksiyon tüm parametreler için değeri alır ve özel yerel bir değişkene kopyasını oluşturur. Bu işlemde o değişkenin adresi kullanılmaz. Bu işlemden de bu fonksiyonun bu değişkenin değerini değiştiremeyeceği gibi bir sonuç çıkar. Her fonksiyon bu tip değişkenleri ve kendi yerelinde tanımlanmış değişkenleri sadece kendi içerisinde değiştirebilir.

Bir Fonksiyon parametresi olarak bir dizinin adı kullanıldığında bu dizinin giriş adresi fonksiyona iletilir. Dizinin tüm değerleri fonksiyona kopyalanmaz. Fonksiyon dizinin elemanlarını indis değerleri ile marifeti ile rahatlıkla değiştirebilir. Buradan şu sonuç çıkarılabilir diziler referans değeri ile yani adres ile fonksiyona iletilirler. Sonraki konularımızda işaretçilerin kullanımının anlatılması ile bu konu daha net olarak açıklanacaktır.

Bu arada değişken sayıda parametre alan fonksiyonları portatif olarak yazmanın bir yolu maalesef bulunmuyor. Bu nedenle de değişken sayıda değerlerden en büyüğünü veya en küçüğünü bulabilecek fonksiyonu yazmak C'de yazmak çok da kolay olmasa gerek.

Alıştırma 20

Kendisine gönderilen a ve b gibi iki değişkenin içerisindeki değerleri çeviren swap adında bir fonksiyon tanımlayınız. Bu tanımın program içinde nasıl kullanılacağını gösteriniz.

Alıştırma 21

Kendisine gönderilen bir tamsayı dizisini sıralayarak çağırana geri döndüren fonksiyonu C dili ile geliştiriniz.

Değişkenlerin etkili olduğu bölgeler

C Blok yapılı bir programlama dilidir. Bu nedenle değişkenler aslında tanımlandıkları blok içerisinde kullanılabilirler. Bir başka blok içerisinde o değişken tanımsız kabul edilir. Bir değişkenin bir blok içerisinde tanımlanıp tanımlanmadığı derleme aşamasında anlaşılabilir. Eğer **Syntax Error** hatası ile karşılaşılıp undefined variable gibi hatalar alınıyorsa değişken tanımlanmamış veya yanlış bloklarda bu değişken kullanılmaya çalışılıyor demektir.

Bu tanımlara göre **"Her değişken sadece tanımlandığı fonksiyon içerisinde kullanılabilir"** demektir. Ancak Global olarak tanımlanmış değişkenleriniz var ise bu değişken tüm fonksiyonlarınız içerisinde kullanılabilir.

Genel Değişkenler

C Programı bir grup harici nesneden oluşur. Bunlar bazen değişkenler bazen de fonksiyonlardır. Bir fonksiyonun içinde tanımlanan değişkenler yerel değişken olarak adlandırılırlar. Bazı durumlarda bunlara dahili değişken de denebilir. Fonksiyonların dışında tanımlanan değişkenler Genel değişkenler olup bazen de harici değişken olarak adlandırılırlar.

Genel değişkenler tüm fonksiyonlar tarafından erişilebilir olduğundan fonksiyon parametrelerine ve değer döndürme mekanizmasına alternatif olarak kullanılabilir. Bir fonksiyon genellikle genel olarak tanımlanmış herhangi bir değişkene adını anmak suretiyle erişebilir.

Eğer fonksiyonlar arasında ortak kullanılacak değişkenler çok ise uzun parametre listeleri yerine genel değişken tanımlamaları daha etkili olacaktır. Ancak bu da fonksiyonun portatifliğini ve program yapısı konusunda dağınıklığı belirtecektir. Bu şekilde fonksiyonlar arasında bir çok veri bağı bulunacaktır.

Genel değişkenlerin kullanılmasının temel nedenlerinden biri de değişken değerlerinin sıfırlanmasını sağlamaktır.

Genel olarak tanımlanmış bir değişken tüm diğer fonksiyonlarda tanımlı olacağı gibi programın çalışmasının bittiği ana kadarda bellekte tutulurlar. Halbuki bir fonksiyonun içinde tanımlanmış yerel değişkeniniz ise fonksiyona girdiğinizde belleğe alınır ve fonksiyondan çıkıncaya kadar

bellekte kalır. Bu nedenle iki ya da daha fazla fonksiyon verileri paylaşacaksa bu verilerin genel değişkenler ile paylaşılması mümkündür.

extern değişkenler

Eğer bir program birden fazla dosya üzerinde tanımlanmışsa ve bazı değişkenler tüm dosyalarda kullanılabilmek ve erişilebilmek istiyorlarsa bu değişkenler extern olarak tanımlanırlar. Örneğin main fonksiyonunun olduğu bir program bloğunda genel değişkenler düzeyinde

int kalem; şeklinde bir tanım yapmış olsanız ve diğer bir program dosyasından bu değişkeni kullanmak isteseniz o dosyada şu şekilde tanımlamanız gerekecektir.

```
extern int kalem;
```

bu şekilde bu değişkenin bu dosya içinde değil diğer bir dosya içinde tanımlanmış olduğunu ve bir ilişki içinde olduğunu görebilirsiniz.

static değişkenler

static tanımı extern gibi bir değişkeninize özel bir sıfat katan değimdir.

Static değişkenler genel tanımlanabildikleri gibi yerel de tanımlanabilirler. Yerel olarak tanımlanan bir static değişken sadece o fonksiyon içinde kullanılabilir ve diğer fonksiyonlarda tanınmaz ancak diğer yerel değişkenlerden farklı olarak değişken fonksiyon bitiminde bellekten atılmaz. Hatta içerisindeki değer dahi statik olarak korunur. Fonksiyonun sonraki tekrar çağrılmasında içindeki değer kullanılır. Bu da şu anlama gelir: yerel static bir değişken özel ama kalıcı bellek sahaları oluşturur.

Bu şekilde tanımlanmak istenen bir değişken için önüne static ifadesini koyanız yeterli olacaktır.

```
static int k;
```

```
static char buf[128] gibi.
```

register değişkenler

register tanımı ile verilen bir değişken program tarafından sıkça kullanılacaktır anlamına gelir. Eğer mümkün olursa bu değişkenler bilgisayarın yazmaçlarına (register) yerleştirilir. Bu sayede bu programlar hem hızlı hem de daha küçük olurlar. Register tanımı ile yapılan bir değişken tanımlama satırı şu şekildedir:

```
register int x;
```

```
register char harf;
```

Uygulamada register değişkenleri ile ilgili olarak bir takım kısıtlamalar bulunacaktır. Bu kısıtlamalar donanımdan kaynaklanana kısıtlamalardır. Bir fonksiyonun içinde sadece birkaç tane değişken register olarak ayarlanabilir. Ayrıca tüm türlerin register olabilmesi mümkün değildir. Uygun olmayan tanımlar için register ifadesi kullanılsa bile derleyici tarafından görülmez. Ayrıca bu kısıtlamalar makinadan makinaya değişebilecektir.

Blok Yapısı

C Pascal, PL/I ve algolde olduğu gibi iç içe blokların tanımlanabildiği yapıda bir dil değildir. C dilinde tüm fonksiyonlar aynı düzeyde tanımlanırlar ve iç içe olamazlar. Ancak bu tarz bir tanım değişkenler için söz konusudur. Yani değişkenler iç içe bloklarda tanımlanabilirler. Ve bu bloklar küme parantezi ile belirlenir. Eğer bir blok içinde tanımlanan bir değişken var ise o değişken o bloğun sonuna kadar geçerlidir. Bir değişken bir blok içinde tanımlanmış ise ve bir üst blokta da tanımlı ise alttaki bloğa girildiğinde yeni tanım geçerli olur diğer tanım bu blok bitinceye kadar bekletilir.

İlk değer atama

Bu bölüm tanımladığınız değişkenlere nasıl ilk değer atayabileceğinizi veya ilk değer atamazsanız nelerin varsayılacağını anlatır.

extern ve static değişkenler sizin tarafınızdan ayrıca bir tanım yapılmadı ise 0 değeri ile otomatik olarak başlarlar. Register ve diğer değişkenlerde ilk değer atam işlemi yapılmadı ise değişkenin değeri rast gele bir değer olur.

Basit değişkenlere tanımlandıkları anda ilk değer ataması yapabilirsiniz. Register değişkenler ve normal değişkenlere ilk değer atanırken sabit olma zorunluluğu bulunmamaktadır. Tanımlı değerlerle oluşmuş olan herhangi bir ifade de değer atama için kullanılabilir.

Özyineleme (Recursion)

C fonksiyonları özyineli olarak kullanılabilirler. Yani bir C fonksiyonu kendini doğrudan ya da dolaylı olarak çağırabilir. En klasik örneklerinden biri faktöriyel hesaplama olarak gösterilebilir. Çünkü faktöryel tanım gereği şu şekilde anlatılabilmektedir

Örnek 13

N!=N*(N-1)! Bu işlem n 1 oluncaya kadar devam edecek şekilde açılabilir. N bir olduğunda faktöryel değeri zaten 1 olacaktır.

```
#include <stdio.h>

int fak(int n){
    if (n==1) return 1;
     else   return n*fak(n-1) ;
}

main(){
   int n;

   printf("Faktöryeli hesaplanacak değeri giriniz");
   scanf("%d",&n);
   printf("N değerinin faktöryeni=%d\n",fak(n));
}
```

Bu programda n değeri faktöryeli hesaplanmak üzere fak fonksiyonuna gönderiliyor. Ancak fak fonksiyonu da kendisini tekrar çağırarak faktöryeli hesaplamaya çalıştığını görebilirsiniz.

Alıştırma 22

Klavyeden girilen herhangi bir tam sayının ikilik sayı sistemindeki karşılığını ekrana yazabilecek fonksiyonu özyineli olarak tanımlayınız.

C dili önişlemcisi

C dili dilin özelliklerini genişletmek için bir takım değimler kullanır. Bunlara önişlemci komutları denir.

Dosya Dahil etme

Bir takım fonksiyonların prototip tanımlarını ve define ile yapılmış bir takım tanımları programa dahil edebilmek için include adındaki komuttur.

```
#include "dosya"
```

şeklinde kullanılır ve adı dosya olan dosyanın içindeki tüm bilgileri bu programa dahil eder. Hemen hemen her C programında 1 yada belki 2 satır bu şekildeki tanımları görmek mümkündür. Büyük programlarda belli bir takım tanımları tüm yardımcı dosyalarda görebilmek için include ile tanımlamış olabileceğiniz genel bir tanım dosyasını tüm dosyalara dahil ederek tanımların ortaklığını garantilemiş olabilirsiniz.

Makro veya sembolik sabit tanımlama

```
#define EVET 1
```

şeklindeki bir tanım en basit sembolik sabit sayıyı tanımlar. Bu tanım ile programımızda kullanılmak üzere EVET adında sembolik bir sabit tanımlamış oluruz. Derleyici programı derlerken define ifadesi yanında tanımlamış olabileceğiniz sabiti programda yerleştirir ve o ifade yerine sabiti koyarak programın derlenmesini sağlar.

Bu şartlar altında şu şekilde yapabileceğiniz tanımlarda ilgili açılımın yapılmasını sağlarlar.

```
#define kare(x) x * x
```

şeklindeki tanım program derlenince kare(x) yerine x*x konmasını sağlayacaktır.

Alıştırma 23

Kendisine gönderilen n değeri için f(x)= $\sum_{x=1}^{n} x$ fonksiyonunu hesaplayıp çağırana döndüren fonksiyonu C dili ile yazınız.

Diziler

Bazen basit değişkenler bir takım sorunların çözülmesi için yeterli olmazlar. Örneğin iki matrisin toplamını bulabilecek bir program basit değişkenler ile yazılamaz. Ya da bir dizi değer üzerinde sıralama yapabilecek program, bir dizi sayı içerisinde istenilen değeri bulabilecek program yine basit değişkenler ile yazılamaz.

Bu tip programlar için hatta bir çok benzeri program için dizi veya matris tanımları yapabilmeniz gerekmektedir.

Dizi Nedir?

Aynı özellikte olan ve aynı amaçla kullanılan bir grup sayının bilgisayar ortamında tutulduğu veri yapısıdır.Kavramsal olarak bir dizi kutuyu düşünerek benzetim yapabilirsiniz.

Genel olarak şöyle tanımlanır.

```
degisken_türü ad[eleman sayısı][ikinci_boyut]...;
```

Bu tanımda değişkene bir ad ve tür belirlemesi yapmak gerekecektir. Tanımlayacağınız dizi tek boyutlu ise sadece eleman_sayısı ile belirtilen tanım yapılacak iki boyutlu bir matris isteniyor ise ikinci_boyut denilen alanı da tanımlamak gerekecektir.

Sonraki Bölümde örneklerle bu daha geniş anlatılacaktır.

Bir Vektörün veya matrisin Tanımlanması

Bir vektör yani tek boyutlu bir dizi tanımlayabilmek için şu ifadenin yazılması yeterli olacaktır(tür olarak tamsayı düşünülmüştür):

```
int ts[100];
```

Bu tanımda adı ts olan ve tamsayı değerleri tutabilen 100 elemanlı bir dizi tanımlanmıştır. Yani bilgisayar belleğinde adı ts olan bir değişkenler zinciri açılmıştır. Bu zincir halkasının ilk öğesinin indis değeri 0 son öğesinin indis değeri 99'dur.

Bu dizinin herhangi bir öğesine program içerisinde erişirken şu şekilde bir kullanım uygulamak gerekir:

```
ts[12]+=12; //veya
ts[x]=23;
ts[x]=23;
```

Göründüğü gibi indis değerleri [] ifadeleri arasına yazılmıştır. Daha da önemlisi bu indis değerleri değişken olabilmektedir. İşte dizileri kullanmayı zorunlu hale getiren bu özelliğidir. Tek bir komut satırı ile tüm öğelere erişmeyi sağlayabilen bir yapıdır bu yapı. Tüm dizi öğelerini tek bir satırla ekrana bu özelliği sayesinde yazabilirsiniz.

Matris tanımlamada aynı özellikleri içerir. Yine tamsayı türünde bir 100 X 100 elemanlı bir matris tanımlamak için şu komut satırını kullanabilirsiniz:

```
int mtr[100][100];
```

Bu satır ile tanımlanan matris yine aynı şekilde iki indisi birden belirtilmek şartıyla kullanılabilir. Satır ya da sütun kavramları tamamen programcının düşüncesine bırakılmıştır. Ancak bir program için ilk boyut satır ikinci boyut sütun olarak düşünülmüşse o program sonuna kadar yöntem hep öyle uygulanmalıdır. Bir matrisi okuyup yazabilmek için iç içe iki for döngüsü kullanılmalıdır. Ama yine de olabilecek başka yöntemlere oranla son derece sağlıklı bir yöntemdir.

Alıştırma 24

3X3'lük bir birim matrisi matris düzeninde ekrana yazan C programını geliştiriniz. Köşegen üzerindeki değerleri 1 diğerlerinin 0 olduğu matris birim matris olup bu matrisi sadece ekrana yazmanız yeterli değildir. Hafızada tutulacak şekilde ayarlamanız gerekmektedir.

Alıştırma 25

Bir matrisin her bir elemanı bulunduğu satır ile sütun değerlerinin toplamı değere sahiptir. Bu şekilde 5X5'lik bir matrisi oluşturup ekrana matris biçiminde yazacak C programını geliştiriniz.

Diziler ve Matrisler üzerinde uygulanabilecek işlemler

Diziler ve matrislerin temel kullanım nedeni bazı işlemleri basit değişkenlere oranla çok daha basit çözmeyi sağlamalarındandır. Bu amaçla diziler veya matrisler üzerinde şu işlemler kolaylıkla yapılabilmektedir.

Dizi sıralama

Bir sayı dizisinin sıralanması sağlanabilir. Sıralanacak sayılar dizi üzerinde değil de başka bir değişken üzerinde tutuluyor ise işlem karmaşıklaşacak hatta belki imkansız duruma dönüşecektir. Ancak bir dizi üzerinde dizinin ne kadar çok değerden oluştuğu önemli değildir ve sıralama aynı sadelikte yapılacaktır.

Örnek 14

Max 100 elemanlı bir dizinin elemanlarını ekrandan okuyacak, artan sırada sıraladıktan sonra tekrar ekrana yazacaktır.

```
#include <stdio.h>

int temp, dsay, k, l, dizi[100];
main(){
printf("Girilecek eleman sayısını Veriniz?\n");
scanf("%d",&dsay);
for(k=0;k<dsay;k++){
printf("%d. sayıyı giriniz",k);
scanf("%d",&dizi[k]);
}
for(k=0;k<dsay-1;k++)
for(l=k+1;l<dsay;l++)
 if (dizi[k]<dizi[l]){
  temp=dizi[k];
  dizi]k]=dizi[l];
  dizi[l]=temp;}
for(k=0;k<dsay;k++)
printf("%d",dizi[k]);
}
```

Bu amaçla değişik sıralama algoritmaları bulunabilir.

Dizi içerisinde istenilen bir şeyler bulma

Bir grup eleman içerisinde bir takım şeyler aramak zorunda kalacağımız durumlar olacaktır. Bu durumlarda Aşağıdakine benzer Algoritmalar kullanılabilir.

Örnek 15

Bu programda 100 elemanlı dizinin dolu olduğu varsayılmıştır.

```
#include <stdio.h>

int deger, k, dizi[100];
main(){
```

```
   printf("Aranacak Değeri Giriniz?\n");
   scanf("%d",&deger);
   for(k=0;k<=100;k++)
      if(dizi[k]==deger) printf("%d değeri %d. Sırada
bulundu",deger,dizi[k]);
   if (k>100) printf("Değer Bulunamadı\n");
}
```

Bu amaçla değişik arama algoritmaları bulunabilir.

Matrisin devriğini alma

Aslında Tüm matrisli işlemlerde tabi ki matris tanımlarına ihtiyacınız olacaktır. İki matrisi toplayıp çarpmak için veya çıkarmak ya da devriğini almak için mutlaka matris tanımına ihtiyacınız var demektir.

Örnek 16

```
#include <stdio.h>

int k, l, matris[100][100];
main(){
// Bu bölüm bir matrisin elemanlarının tek tek ekrandan
okunmasını sağlar
   for(k=0; k<100; k++)
      for(l=0; l<100; l++)
        scanf("%d", &matris[k][l]);
// Bu bölümde matrisin transpozu yani devriği alınıyor
   for(k=0; k<100; k++)
      for(l=k+1; l<100; l++){
        tmp=matris[k][l];
        matris[k][l]=matris[l][k];
        matris[l][k]=tmp;
      }
// Bu bölümde matris matris biçiminde ekrana yazılıyor.
   for(k=0; k<100; k++){
      for(l=0; l<100; l++)
        printf("%d\t", &matris[k][l]);
      printf("\n");
   }
}
```

Örnek 17

Bu örneklerde matris boyutunun daha anlaşılır olması için include değiminden sonra define değimi ile matrisin boyutları sabit olarak tanımlanabilir ve programda belili bir takım yerlerde bu sabitler kullanılarak program yeniden düzenlenebilir. Şöyle ki:

```
#include <stdio.h>
#define MAXSAT 100
```

```
#define MAXSUT 100

int k, l, matris[MAXSAT][MAXSUT];
main(){
// Bu bölüm bir matrisin elemanlarının tek tek ekrandan
okunmasını sağlar
   for(k=0; k<MAXSAT; k++)
      for(l=0; l<MAXSUT; l++)
        scanf("%d", &matris[k][l]);
// Bu bölümde matrisin transpozu yani devriği alınıyor
   for(k=0; k<MAXSAT; k++)
      for(l=k+1; l<MAXSUT; l++){
        tmp=matris[k][l];
        matris[k][l]=matris[l][k];
        matris[l][k]=tmp;
      }
// Bu bölümde matris matris biçiminde ekrana yazılıyor.
   for(k=0; k<MAXSAT; k++){
      for(l=0; l<MAXSUT; l++)
        printf("%d\t", &matris[k][l]);
      printf("\n");
   }
}
```

Bu ikinci örnekte kullanılan define satırları MAXSAT ve MAXSUT tanımları sembolik sabitlerdir. Bu sabitler kullanıldıkları yerlerde derleme aşamasında karşılıklarındaki sayısal değerleri derlenen yerlere korlar. Bu sabitler sadece programda kullanılacak matrisin boyutunu çok pratik şekilde değiştirmenizi sağlayacaktır.

Alıştırma 26

Klavyeden girilen maksimum 100 tane değeri bubble sort (Kabarcık Sırala) algoritmasına göre küçükten büyüğe sıralayıp ekrana yazan C Programını geliştiriniz.

Karakter dizeleri (Dizge)

Bir programda karakterlerin oluşturduğu diziye karakter dizesi veya dizge (string) denir. Alfabetik değerleri ad, soy ad, açıklama vb gibi değerleri saklamak için bu tip değişkenlere ihtiyacınız vardır. Bu tip sabitleri de yine çift tırnaklar içerisinde yazarak ifade edebilirsiniz.

Karakter dizesi tanımlama ve kullanma

Diziler sadece sayısal değişkenlerde ve sayısal işlemlerde kullanılmazlar. Bir çok kez Cümle vb. yapılarda aslında karakter dizileri kullanırız. Karakter dizileri normal bir dizi tanımlanıyormuş gibi tanımlanır sadece tür değimi char olarak seçilir. Bu şekilde içerisinde saklanacak toplam harf sayısını belirtmiş olursunuz.

```
char cumle[100];
```

ifadesi toplam 100 harfli bir ifade tanımlar. Bu ifadenin tüm 100 harfi kullanılmak zorunda değildir. Sayısal dizilerden farklı olarak bu karakter dizileri üzerinde yapılabilecek işlemler ya da kullanım amaçları farklı olabileceği için bize sunulan destek ve fonksiyonlar da yer yer farklılıklar gösterir. Zaten biz bu tür dizilere "karakter dizesi" deriz.

Bu şekilde tanımlanmış bir dize içinde yer alan harflerin en sonuncusu karakter dizesinin sonunu belirleyen '\0' karakteridir. Bu karakter bir karakter dizesi içinde önemli özelliğe sahiptir. Gerek C dili fonksiyonlarının kendisi gerekse de sizin yazacağınız programlar bu karakterden faydalanacaktır. Bölüm sonuna doğru örnekleri inceleyerek '\0' karakterinin nasıl kullanılacağını görebilirsiniz.

Bu karakter dizeleri ile ilgili bir takım işlemleri gerçekleştirmek için gerekli olabilecek fonksiyonlar **"string.h"** kütüphanesi içerisindedir. Bu Fonksiyonlar ve kullanım amaçları şunlardır.

strlen(s):s ile belirtilen karakter dizesinin uzunluğunu bulur.

strcpy(s1,s2):s2 ile belirtilen karakter dizesini s2 ile belirtilen karakter dizesine kopyalar. Bu fonksiyon oldukça önemlidir çünkü C dilinde bir karakter dizesine normal değer atama ifadesi kullanılarak bir dizeye yerleştirilemez. Çünkü aslında C dilinde bir karakter dizesi adı bir değişkene göstergedir ve ona yapacağınız bir değer atama yanlış sonuç verecektir.

strcmp(s1,s2):s1 ile belirtilen karakter dizesi ile s2 karakter dizesinin karşılaştırılmasını sağlar. Bu karşılaştırma sonucunda iki değer eşit ise

sonuç 0, birincisi büyük ise sonuç pozitif, ikincisi büyük ise sonuç negatif olarak döndürülür.

strcat(s1,s2): s1 ile belirtilen karakter dizesinin sonuna s2 ile belirtilen karakter dizesini ekler. Sonuç s1 ile belirtilen dize içinde saklanır.

Örnek 18

Klavyeden enter tuşuna basılıncaya girilen bir ifadenin bir değişkende saklanmasını sağlayan bir program yazınız.

```
#include <stdio.h>
char cumle[1024],ch;
int s=0;

main(){
   ch=getc(stdin);
   while (ch!='\n'){
      cumle[s]=ch;
      s++;
      ch=getc(stdin);
   }
   cumle[s]='\0'; // girilen ifadenin sonuna karakter dizesi
sonu işaretini yerleştiriyoruz.
}
```

Örnek 19

Klavyeden girilmiş bir ifadenin uzunluğunu strlen fonksiyonu kullanmadan bulan programı yazınız.

```
#include <stdio.h>
char cumle[1024];
int s=0;

main(){
   gets(cumle);
   while (cumle[s]!='\0')s++;
   printf("Uzunluk:%d\n",s);
   // girilen ifadenin sonuna kadar harfler sayılır.karakter
dizesi sonu işaretini bulmaya çalışıyoruz
}
```

Örnek 20

Klavyeden istenilen uzunlukta veya enter tuşuna basılıncaya kadar ifadelerin okunmasını sağlayan programı yazınız.

```
#include <stdio.h>
#define SAY 15
```

```
char kelime[SAY];
int l;

main(){
   l=0;
   while((kelime[l]=getc(stdin))!='\n' && l<SAY)
      l++;
   if (l>=SAY) l=SAY-1;

   kelime[l]='\0';
}
```

Göstergeler-İşaretçiler

İşaretçi herhangi bir değişkenin adresini saklayan bir değişkendir. İşaretçiler C programlama dilinde oldukça çok kullanılırlar. Bunun nedeni bazen çok karmaşık işlemleri kolay hale getirmelerindendir. Bazen de başka yollarla çok zor yapacağınız işlemleri daha kolay ifade etmenizi sağladığındandır.

İşaretçiler ve Adresler

Bir işaretçi bir değişkenin adresini sakladığı için bir değer dolaylı yoldan erişmek mümkündür.x'in bir tam sayı değişken olduğunu kabul edelim. px değişkeni ise bu x değişkenine işaret eden bir işaretçi değişken olsun. & operatörü bir nesnenin adresini bulmamızı sağlar.

```
Px = &x;
```

İşlemi x'in adresini px değişkenine atar. Artık px x değerine bir işaretçidir. & operatörü sadece değişkenlere, dizilere ve veri yapılarına uygulanabilirler. &(x+1) ve &3 şeklindeki kullanımlar geçersizdir. Register niteleyicisi ile tanımlanmış bir değişkeninde adresini almak geçersiz işlemdir.

* operatörü de yanındaki ifadenin bir işaretçi olduğunu anlatır ve bu adresin ifade ettiği yerdeki değişkeni anlatır. Eğer adı y olan bir int değişken olsun

```
y= *px;
```

ifadesi y değerine px'in gösterdiği yerdeki değeri atar. Yani:

```
px = &x;
y = *px
```

ifadesi

```
y = x;
```

ifadesine denk bir işlem yapar.

Yukarıdaki işlemi yapabilmek için tabii ki ilgili değişken tanımlarını da aşağıdaki gibi yapmak gerekecektir.

```
int x,y;
int *px;
```

x ve y ile ilgili tanımlar daha önceden gördüğümüz gibidir. *px tanımı ise yeni bir tanım olarak karşımıza çıkıyor. Bu tanıma göre *px'in bir tam sayı olduğunu anlatır. Her hangi bir ifadede px ile karşılaşılırsa bunun bir int tür olduğunu anlatır. Bu da çoğunlukla derleyici düzeyinde hataların giderilmesini sağlar.

İşaretçiler aritmetik ifadeler içerisinde kullanılırlar. Örneğin px bir tam sayı x'e işaretçi ise x'in kullanılabileceği tüm ifadelerde *px kullanılabilir.

```
y = *px +1;
```

ifadesi y'nin değerini x'in bir fazla olarak ayarlar.

```
Printf("%d\n", *px);
```

X'in değerini ekrana yazacaktır.

```
d=sqrt((double) *px);
```

x'in karekökünü hesaplar. Burada x'in değeri önce double türe çevrilir sonra sqrt fonksiyonuna iletilir.

```
y = *px +1
```

şeklindeki ifadelerde * ve & diğer aritmetik operatörlere göre önceliklidir. Ve önce px'in gösterdiği değer bulunur bunun üzerine 1 eklenerek elde edilen sonuç y'ye aktarılır.

```
y = *(px+1)
```

ifadesi öncelikle px ile gösterilen değişkenden sonraki değişkeni bulur ve bu değişkenin içeriğindeki değeri y'ye aktarır.

İşaretçiler eşitliklerin sol tarafında da bulunabilirler. Örneğin

```
*px = 0;
```

ifadesinde x'in değeri 0 olarak değiştirilir.

```
*px += 1;
```

ifadesi x'in değerini bir arttırır. `(*px)++` ifadesi de aynı işlemi yapar. Bu son ifade de parantezler zorunludur. Parantez kullanmadan ifade edilseydi öncelikle px'in değeri öncelikli olarak 1 arttırılır. Çünkü ++, *, & gibi tekil operatörler sağdan sola öncelikte hesaplanır.

Son olarak işaretçiler de birer değişken olduğundan diğer değişkenlerde olduğu gibi işlemlere girebiliyorlar. Bu durumda py'in bir başka int'e

işaretçi olduğunu kabul edersek py = px ifadesi px'in içindeki değeri py'e kopyalar. Bu da py'nin px'in gösterdiği değeri göstermeyi sağlar.

İşaretçiler ve Fonksiyon parametreleri

Normal şartlarda C fonksiyonları "call by value" değer ile çağırma yöntemine göre çalıştığı için çağıran fonksiyondan gönderilen parametreler değişse bile çağırana geri yollamanın doğrudan bir yolu bulunmamaktadır. Öyleyse normal bir parametreyi fonksiyonda değiştikten sonra geri çağırana nasıl döndüreceğiz. Örneğin bir sıralama algoritmasında iki elemanın sırasının değiştirilmesi gerekirse swap adı altında yazılacak bir fonksiyon maalesef aşağıdaki gibi basit bir şekilde çağrılamayacaktır.

```
swap(a,b);
```

bu fonksiyonun tanımı aşağıdaki gibi ise

```
swap(x,y)
int x,y;
{
    int temp;
    temp = x;
    x = y
    y = temp;
}
```

call by value yöntemiyle çağrılan bu fonksiyonda yerleri değiştirilen x ve y değişkenleri çağırana döndürülemeyecektir.

Tabii ki beklenilen sonucu elde etmek için bir yol bulunmaktadır. Çağıran program fonksiyona işaretçileri geçirirse bu işaretçilerin gösterdiği değerler değiştirilebilecektir.

```
swap(&a,&b);
```

& operatörü bir değişkenin adresini bulduğu için &a a değişkeni için bir adres değeri dolayısı ile bir işaretçidir. swap fonksiyonunun kendisi de parametreleri işaretçi olarak tanımlar. Gerekli olan parametreler bu işaretçiler kanalıyla fonksiyona aktarılır.

```
swap(px,py)
int *px, *py;
{
    int temp;
    temp=*px;
    *px = *py;
    *py = temp;
}
```

İşaretçi cinsinden parametre kullanan fonksiyonların genel kullanım nedenlerinden biri bir değerden fazla değer döndürme ihtiyacından kaynaklanır. swap fonksiyonu da iki değer döndürüyor olduğunu dikkatinizi çekmiştir.

Alıştırma 27

Kendisine gönderilen bir dizi sayının içerisindeki en büyük elemanı bulup çağırana döndüren C programını geliştiriniz.

İşaretçiler ve diziler

C programlama dilinde işaretçiler ve diziler arasında oldukça güçlü bir ilişki vardır. Öyle ki bu ikisi neredeyse aynı şeylermiş gibi anlatılabilirler. Herhangi bir dizi indisleme işlemi ile yapılabilen tüm işlemler işaretçiler ile de yapılabilir. İşaretçiler ile yazılmış modeli diğerine göre daha hızlı çalışabilecektir.

`int a[10]` tanımı 20 elemanlı bir dizi tanımlıyordu. Yani adı aynı olan birbirine benzer on eleman tanımlanmış oluyordu. `a[i]` ifadesi de a dizisinin i. Konumundaki elemanı anlatıyordu. Eğer pa bir tamsayıya işaretçi ise ve `int *pa` olarak tanımlanmış ise `pa=&a[0]` ifadesi dizinin sıfırıncı elemanına işaret eder. Bu da pa a[0]'ın adresini içerir anlamına gelir.

x = *pa ifadesi a[0]'ın içindeki değeri x'in içine atar.

Eğer pa bir dizinin belirli bir elemanına işaret ediyorsa pa+1 bir sonraki elemanına işaret eder. Ve tanım gereği pa-i ise pa'nın gösterdiği elemandan i tane öncekini gösterecektir. Bu nedenle pa a[0] elemanına işaret ediyorsa *(pa+1) ifadesi a[1] elemanına işaret eder. pa+i i.elemanın adresini *(pa+i) ise i.elemanın içeriğini anlatır.

Bu tanımların tamamı a dizisinin türüne bağlı kalmaksızın her zaman için doğrudur. Bir işaretçiye bir değerin eklenmesi ve benzeri tüm işaretçi aritmetikleri o dizinin içinde işaretçinin türüne bağlı olarak hesaplanır. İşaretçinin gösterdiği türe göre bir sonraki saklama ifadesini ya da bir önceki adresi bulur. Böyle olunca pa+i şeklindeki bir ifadede i değeri değişkenin boyutu ile çarpılır sonra pa değerine eklenerek yeni adres değeri bulunur. Bu şekilde yeni hesaplanan adres değeri değişken türüne bağlı olmuş olur.

İndisler ve işaretçiler ilgili benzerlik oldukça yakındır. Gerçekte bir dizinin herhangi bir elemanı derleyici tarafından işaretçiye derleme sırasında çevrilir. Aslında dizinin adı bu diziye bir işaretçi değeridir. Bu nedenle pa=&a[0] şeklinde yazdığımız ifadeyi aslında pa=a şeklinde de yazabilirdik.

Aslında kısaca şu şekilde söyleyebiliriz. A[i]ifadesi derleyiciniz tarafından hemen *(pa+i) şekline çevrilmektedir. Her ikisinin birebir eşit olduğunu bilmeniz yeterli olacaktır.

Bir dizi adının bir işaretçiden farklı bir konunun olduğunun bilinmesi gerekir. İşaretçiler normalde birer değişkendir. Pa=a veya pa++ gibi tanımlar geçerlidir. Fakat bir dizi adı sabit bir işaretçi değeridir. Bu nedenle a=pa, a++ veya p=&a gibi tanımlar geçersizdir.

Bir fonksiyona bir dizinin adını geçirdiğiniz zaman aslında dizinin başlangıç adresi fonksiyona geçirilmiş olur. Çağrılan bu fonksiyonun içinde bu parametre diğer değişkenler gibi kullanılabilen bir değişkendir. Ve bu değişken bir diziye işaret eden işaretçidir. Bu gerçeği göz önünde bulundurarak fonksiyonlara dizi gönderip bu dizinin adresi ile ne yapılacağını şu örnekle görelim

```
int strlen(s)
char s;
{
    int n;
    for(n=0; *s != '\0';s++) n++;
    return (n);
}
```

s değerinin arttırılması gayet normal bir davranıştır. Çünkü s++ işleminin çağıran fonksiyondaki karakter dizesi üzerinde herhangi bir işlem yapmamaktadır. Ancak yerel s işaretçisinin değerini arttırarak karakter dizesi içindeki sonraki karakterler ulaşmaktadır. Fonksiyonu tanımlarken `char s[];` şeklindeki tanım ile `char *s;` şeklindeki tanım tamamen aynıdır. Fonksiyondaki kullanımın nasıl olacağı hakkında bilgi verse de bu konuda bir kural belirlemez. İstediğinizi kullanabileceğiniz gibi aynı fonksiyon içinde karışıkta kullanabilirsiniz.

Adres Aritmetiği

p bir işaretçi ise p++ aynı türdeki bir sonraki değere işaret eder. p+=i ise p'nin değerini i kadar arttırarak i sıra sonraki değere işaret eder. Bu ve buna benzer işlemler en basit işaretçi aritmetiğini anlatır.

C dilinin işaretçi, dizi ve işaretçi aritmetiği ile ilgili entegrasyonu ve bütünlüğü dilin çok güçlü olmasını sağlamıştır. Şimdi bellekten yer ayıran ve bu yeri geri serbest bırakabilen program satırlarını inceleyeceğiz. Programda malloc(n) ve free(p) şeklindeki fonksiyonlar kullanılmıştır. Bu fonksiyonlardan malloc(n) n byte'lık bir alan ayırır ve bu alana bir işaret.i değer döndürürken free(p) p ile işaret edilen alanın bellekte serbest bırakılmasını sağlar. Bu işlemlerin sıra bakımından ters olarak yapılması gerektiği açıktır. Yani bir bellek sahası önce malloc ile ayrılmış olmalıdır ki free ile serbest bırakılabilsin. Bu şekilde özellikle istenilen zamanlarda istenilen kadar bellek ayrılabilir ve dizi ya da ona benzer kullanımlar statik olmaktan çıkar ve dinamik kullanıma geçerler.bu uygulamalarda boş bellek sahaları heap-yığın dediğimiz yerden kullanılırlar.

Örnek 21

Bu örnek malloc kullanarak heap denilen sahadan bellek tahsis eder ve bu belleği kullandıktan sonra geri serbest bırakır.

```
#include <stdio.h>
int p;
main(){
   p=(int *) malloc(sizeof(int)*20);
   // 20 elemanlı bir int dizisi tanımlamakla eşdeğer
   if (p==NULL){
      printf("Yeterli Bellek Yok\n");
      return;
   }
   for(k=0;k<20;k++) scanf("%d",p+k);
   for(k=0;k<20;k++) printf("%d\n",*(p+k));
   free(p);
}
```

Karakter İşaretçileri ve Fonksiyonları

"Ben bir karakter dizesiyim" şeklinde yazılmış olan bir ifade aslında karakterlerden oluşmuş bir dizidir. Ancak gerek C fonksiyonları gerekse derleyici bu diziyi ifade ederken '\0' ile sonlandırır. Böylelikle dizenin sonunu bulabilmek mümkün olur. Bu tür karakter dizesi sabitleri çoğunlukla fonksiyon çağrılarında parametre olarak kullanılırlar. Genellikle bu tip durumlarda fonksiyonlar bu karakter dizesinin başlangıcını bir işaretçi olarak alır ve kullanırlar.

Eğer bir tanım şu şekilde yapılmış ise bu da bir karakter dizesi tanımlamış olabilir. `char * mesaj;` ve peşindeki satır `mesaj = "Bu bir Cümledir";` komutu ise aslında gerek bir kopyalama işlemi yapılmaz onun yerine karakter dizesinin adresi mesaj işaretçi değişkenine atanır.

Konunun daha iyi anlaşılması için strcpy fonksiyonunu biz kendi imkanlarımızla yazalım. Şöyle ki bu fonksiyon strcpy(s1,s2) şeklinde kullanılıyordu ve s2 içindeki değeri s1 içine kopyalıyordu.

```
strcpy(s,t)
char s[],t[]
{
   int i;

   i=0;
   while((s[i] = t[i])!='\0')
      i++;
}
```

Yukarıdaki dizi kullanımı şeklinde olan fonksiyon tanımı idi. Şimdi de aynı fonksiyonu işaretçileri kullanarak yazmaya çalışalım.

```
strcpy(s,t)
char *s,*t
{

   while((*s = *t)!='\0'){
      s++;
      t++;
   }
}
```

s ve t işaretçi değerleri fonksiyona değer olarak iletildiğine göre fonksiyon içinde bu değerler istenildiği şekilde değiştirilebilecektir. Hatta yukarıdaki örnek pratikte daha kısa bile ifade edilebilecektir. Şöyle ki:

```
strcpy(s,t)
char *s,*t
{
   while((*s++ = *t++)!='\0');
}
```

Bu örnek s ve t'yi while komutunun koşul sınama bölümünde arttıracaktır. *t++ ifadesinde ise t arttırılmadan önceki t adresinde duran değeri anlatmaktadır. Sonda yer alan ++ işlemi t'nin gösterdiği değere ulaşmadan t'yi değiştirmez. Aynı şekilde s içinde aynı şeyleri söyleyebiliriz.

Benzer algoritmaları kullanarak strcmp ve benzer diğer string fonksiyonlarını yeniden yazabilirsiniz.

Alıştırma 28

Klavyeden girilen bir kelimenin kaç harften oluştuğunu bulan fonksiyonu C programlama dili ile geliştiriniz. Bu fonksiyonun bir program içerisinde nasıl çağrılacağını gösteriniz.

Dosyalarla ilgili işlemler

Yazdığınız program verileri klavyeden okumak yerine sabit bir ortamdan alacaksa veya ekrana yazmak yerine sabit bir ortama yazacaksa programınızda dosya işlemleri yapmak zorunda kalacaksınız. Dosyayı açmak okuma-yazma yapıp geri kapatmak zorunda kalacaksınız. O halde bu konuya bir göz atmanız gerekecektir.

Dosya Türleri ve erişim

Hemen her programlama dilinde olduğu C Programlama dilinde de genel olarak iki tür dosya bulunmaktadır. Bunlardan birincisi Sıradan erişimli (Sequential), diğeri de doğrudan ya da rast gele erişimli (Random) olarak bilinirler. Her programlama dili ve işletim sistemi için şu genel kurallar bu iki dosya türü için geçerlidir.

Sıradan Erişimli Dosyalar Sequential ACCESS	**Doğrudan ya da rastgele erişimli dosyalar Random ACCESS**
Bir dosya sadece okumak ya da yazmak için açılabilir. Aynı anda bir dosya hem okunup hem de yazılamaz	Bir dosya aynı anda hem okunup hem yazılabilir.
Bu dosyalarda kayıt uzunlukları sabit değildir. Dosyadaki her bir kayıt bir satırdır ve satır sonu işareti ile sonlandırılmıştır. Dosya da dosya sonu (EOF) işareti ile sonlandırılmıştır.	Bu dosyalarda ise kayıtlar sabit uzunlukludur. Yani her bir kaydın kaç byte yer işgal ettiği programcı tarafından bilinir ve her okumada ve yazmada o kadar byte okunup-yazılır. Bu nedenle kayıt sonu işareti bulunmaz ve Dosya sonu işaretinin de fazla bir önemi yoktur. Esas olan dosyanın toplam uzunluğudur.
Bir kayda doğrudan erişim söz konusu değildir. Örneğin 8. kaydı okumak isterseniz ilk 7 kaydı da okumanız gerekecektir okumalar tek yönlü ve ileriye doğrudur. Yani önceki kayda erişip okumaya çalışamazsınız.	Bir kayda istenildiği anda doğrudan erişebilirsiniz. Bu kaydın sırası bilindiği takdirde okuma yapılabilir. Bu nedenle geriye doğru okuma yapmak mümkündür.
Dosyaya sadece yazabilir veya sadece okuyabilirsiniz.	Dosyaya hem yazabilir hem de okuyabilirsiniz

Genellikle metin Dosyaları, Belgeler, Program dosyaları ve benzeri dosyalar sıradan erişimli dosyalar iken, Bilgi kayıt dosyaları gibi dosyalar doğrudan erişimli dosyalardan seçilirler.

Metin Dosyaları

Öncelikli olarak bu şekilde bir dosyanın tanıtılması gerekir bu amaçla da FILE adında bir tür ile tanımlanmış bir değişken olmalıdır.

Dosya açma işlemi

Bir dosyanın üzerinde işlem yapabilmek için mutlaka dosyanın açılarak program ile ilişkilendirilmesi gerekir.

Bir dosyayı açabilmek için öncelikle bu dosyayı sembolize edecek bir değişkene ihtiyacımız bulunmaktadır. Bu değişken FILE değimi ile tanımlanmaktadır.

```
FILE *dosya;
```

ifadesi adı dosya olan ve disk üzerinde bir dosyayı belirtecek olan mantıksal bir dosya tanımlar. Daha sonra bu tanıma göre dosya şu şekilde açılır:

```
fp=fopen("deneme.dat","r");
```

Burada dosyamızın adının deneme.dat olduğu varsayılmıştır. Bu dosya okuma amaçlı olarak açılmış olup bundan sonra dosyanın program içinde fp adıyla anılacağı öngörülmüştür.

Eğer dosya açılabilmiş ise fp o dosyaya bir gösterge değeri tutar. Eğer dosya herhangi bir nedenle açılamamış ise fp'nin değeri 'NULL' dir (Yani BOŞ).

```
dosya_değişkeni=fopen("dosya_adı","dosyanın_modu");
```

Bu biçim kullanımda dosya_değişkeni FILE değimi ile tanımlanmış olan değişken olmalıdır. "dosya_adı" açmak istediğimiz dosyanın disk üzerindeki tam konumunu ve adını belirtmelidir. "dosyanın modu ise aşağıdakilerden biri olabilir:

r: Dosyanın sadece okumak için açılacağını belirtir. Bu şekilde açılacak dosya mutlaka sistem üzerinde bulunmalıdır aksi halde hatalı bir sonuç alınır.
w: Dosyanın sadece yazmak için açılacağını belirtir. Bu şekilde açılacak bir dosya sistem üzerinde oluşturulacaktır. Buna dikkat edilmelidir. Eğer bu dosya var ve "w" modu ile açılıyorsa dosya silinerek yerine yenisi konulacaktır.
a: Dosyanın sona ekleme için açılacağını belirtir. Dosya yoksa yeni oluşturulacak var ise öncekinin sonundan yazmak üzere hazır hale gelecektir.

Dosyadan okuma

getc `c=getc(fp)` şeklinde kullanılır ve fp ile temsil edilen dosyadan 1 karakter okuyup c değişkenine getirir.

fscanf `fscanf(fp, "biçim tanımları", değişkenler)` şeklinde kullanılır. Bildiğimiz scanf gibi çalışmaktadır. Ancak klavyeden değil de bilgileri fp ile temsil edilen dosyadan okumaktadır. Biçim tanımları kısmı normal scanf komutu ile aynıdır.

Dosyaya yazma

putc `putc(fp, c)` şeklinde kullanılır ve fp ile temsil edilen dosyaya c değişkenindeki 1 karakteri yazar.

fprintf `fprintf(fp, "biçim tanımları", değişkenler)` şeklinde kullanılır. Bildiğimiz printf gibi çalışmaktadır. Ancak sonucu ekrana değil de fp ile temsil edilen dosyaya yazmaktadır. Biçim tanımları kısmı normal printf komutu ile aynıdır.

Dosya kapama

fclose dosyaların program tarafından işleri bitmiş ise kapatılmaları gerekir. Kapatma komutu için `fclose(fp);` kullanılmalıdır

Alıştırma 29

Klavyeden girilen değerlerden pozitif olanlarını p.dat, negatif olanlarını n.dat isimli dosyaya yazan ve sıfır girildiğinde işlemi sonlandıran programı C dili yazınız.

Alıştırma 30

Kendisine gönderilen iki dosya ismine göre birinden diğerine kopyalama yapıp toplam kopyalanan karakter sayısını çağıran programa döndüren fonksiyonu yazınız.

Alıştırma 31

Bir dosyadan okunan 3X3'lük bir matrisi ekrana matris biçiminde yazan C Programını geliştiriniz.

Alıştırma 32

Bir dosyadan okunacak sayıların içinden en büyük ve en küçüklerini bulup ekrana yazacak C programını geliştiriniz.

Doğrudan Erişimli Dosyalar

Doğrudan erişimli dosyalarda da program içerisinde bu dosyayı sembolize edebilecek bir değişkene ihtiyaç duyulur. Ancak bu tip dosyalarda bu değişken int tipindedir.

Dosya açma işlemi

```
int dosya;
```

şeklinde tanımlanır.

```
dosya=open("deneme.dat");
```

disk üzerinde adı deneme.dat olan bir dosyayı doğrudan erişimli olmak üzere açacaktır.

`dosya_değişkeni=open("dosyanın_fiziksel_adı");` şekliyle genel tanım yazılabilir.

Bu şekilde bir dosya açılınca dosyanın program içinde temsil edilebilmesi için bir değer programa döndürülür ve bu değer dosya_değişkeninde saklanır. Eğer bu değer -1 ise bu dosya bir takım nedenlerden dolayı açılamamıştır. Sonraki Okuma, Yazma vb. işlemlerde dosyamızın bu numarasını kullanarak işlem yapmamız gerekecektir.

Bu dosyaların açma komutlarının biraz farklı olduğunu görmüş olmalısınız. Bu nedenle okuma yazma komutları da biraz farklı olacaktır.

Dosyadan okuma

read `read(dosya_no,okunanlar,okunacak_byte);` şeklinde kullanılır. Bu fonksiyon ile doğrudan erişimli dosyadan okuma yazma kafasının bulunduğu noktadan itibaren okunacak_byte kadar karakter okunarak okunanlar dizisine yerleştirir. Sonuç olarak bu fonksiyon okuduğu karakter sayısını döndürür.

Dosyaya yazma

Yazma işlemi için **`write`** komutu kullanılmaktadır. Bu Komut `write(dosya_no, yazılacaklar, yazılacak_byte);` biçiminde kullanılır. Bu tanımda dosya_no bilgilerin yazılacağı dosyayı temsil eder. Yazılacaklar içerisinde yazılacak bilgilerin bulunduğu diziyi ifade eder. yazılacak_byte ise dosyaya kaç byte bilgi yazılacağını belirtir. Bu fonksiyon yazılanların

toplam kaç byte olduğunu geri döndürür. Eğer -1 değeri dönmüş ise yazmada bir sorun ile karşılaşılmış demektir.

Dosya kapama işlemleri

Bu şekilde açılan bir komutun kapatılması için **`close(fp);`** komutu kullanılır.

Doğrudan erişimli dosyalara özel durumlar.

Bu tip bir dosyada okuma yazma kafasının konumunu okuma ya da yazma yapmadan değiştirmek mümkündür. Bu amaçla kullanılan bir **lseek** fonksiyonu bulunmaktadır. **`lseek(dosya_no, mesafe, nereye_gore);`** şeklinde kullanılır. Burada dosya_no kafa hareketinin hangi dosyada yapılacağını belirtir. mesafe kafa hareketinin kaç byte olacağını belirtirken (sayısal bilgi) nereye_gore bilgisi bu hareket miktarının nereye göre yapılacağını belirler (Dosya başına göre, dosya sonuna göre ve bulunduğu yere göre).

Veri Yapıları (structure)

Veri yapısı içinde birden fazla değişken saklayan bir bilgi topluluğudur. Genellikle bu veriler farklı türdedir. Ancak aynı nesne için kullanılan verileri sembolize ederler ve tek bir nesne adı altında anlatılırlar.

Basit bir örnek olarak bir öğrenci kaydı verilebilir. Bir öğrenci nesnesinin sahip olduğu bir takım özellikler; adı, soyadı, vize, final ve bütünleme notu gibi veriler tek bir isim altında toplanabilir.

Bu tarzdaki veri yapıları karmaşık verilerin düzenli tutulmasını sağlar. Bu özellikle büyük programlarda düzenli bir yapılanma sağlayacaktır.

Temel Tanımlar

Bir tarih bilgisini ele alalım. Tarih birkaç bölümden oluşan bir bilgidir. Bu bölümler gün, ay, yıl, haftanın günü, ayın adı gibi değerlerden oluşur. Bu değerlerin tümü tek bir veri yapısı içinde şu şekilde anlatılabilir.

```
struct date {
   int day;
   int month;
   int year;
   int yearday;
   char mon_name[4];
};
```

struct kelimesi bir veri yapısı tanımının yapıldığını belirtir. Bu tanım küme parantezleri içinde bu veri yapısının detaylarının anlatıldığı bir tanımdır. Bu söz dizimde küme parantezinden sonra noktalı virgülden önce bu veri yapısı cinsinden tanımlarınızı yapmak üzere değişken türlerinizi anlatabilirsiniz.

Bir veri yapısı tanımı içinde bahsi geçen her bir değişken o veri yapısının elemanı veya üyesidir. Bir veri yapısı içinde geçen bir elemana verdiğiniz bir ismi c programının içinde bir başka değişkene verebilirsiniz. Bunlar bulundukları ve hükmettikleri yerler itibari ile biri birlerine karıştırılmazlar.

`struct {...} x,y,z;` şeklindeki bir tanım görsellik olarak int x,y,z ile aynıdır. Çünkü her ikisi de x,y,z adında değişik türlerde de olsa değişkenler tanımlar ve bu değişkenler uygun miktarda yer ayırır.

Değişken adı kullanılmaksızın yapılan bir veri yapısı tanımı bellekte yer ayrılmasına neden olmaz. Böyle bir tanım sadece değişkenin genel görünüşü ve şekli hakkında bir şablon bilgisi sunar. Eğer veri yapısı tanımında bir isimlendirme yapılmış ise bu isimler ile programın diğer

bölümlerinde değişkenler tanımlanabilir. Örneğin biraz önce tanımladığımız date adlı veri yapısını şu şekilde kullanarak gerçek değişkenlerin tanımlanmasını sağlayabilirsiniz.

```
struct date tarih;
```

şeklindeki bir satır adı tarih ve türü date olan bir değişken tanımlar. Bu şekilde yapılacak bir tanımlama ile ilk değer ataması da yapılabilir. Şöyle ki:

```
struct date tarih={3, 3, 2004, 61, "Mar"};
```

Bir veri yapısındaki tek bir elemana değer atamak için aşağıdaki şekli kullanabilirsiniz.

```
veriyapısıadı.elemanadı
```

Bu tanımdaki "." Veri yapısı ile bu veri yapısının içindeki elemanın birbiriyle ilişkilendirilmesini sağlar.

Yukarıdaki date örneğinde tanımlanan tarih adlı değişkenin içindeki herhangi bir elemana erişebilmek için programda şu şekilde kullanımlar bulunmalıdır.

```
tarih.year=1994;
tarih.month=12;
tarih.day=22;
```

Veri yapıları içi içe tanımlardan da oluşabilir. Örneğin bir öğrenci veri yapısının şu şekilde tanımlandığını varsayalım:

```
struct ogrenci {
   char adsoy[25];
   char adres[40];
   unsigned short vize, final, butunleme;
   struct date dtarihi;
   struct date kaytarihi;
}
```

Bu veri yapısı tanımında öğrenci hem doğum tarihi hem de kayıt tarihi olmak üzere iki tarih bilgisine sahip olacaktır. Bu tarih bilgilerinin her biri de yukarıda bahsedilen tüm elemanlara sahip olacaktır. Bu şekildeki bir yapının değişkenle ifade edilmesi ve kullanılması ise şöyle olmaktadır:

```
struct ogrenci ogr;
ogr.dtarihi.ay=8;
```

şeklindeki kullanımda öğrencinin doğum tarihinin ay bilgisi 8 olarak verilmiştir.

Alıştırma 33

Bir dairenin parametrelerinin struct değimi ile ifade edilmesi gerekirse nasıl tanımlanması gerektiğini gösteriniz.

Veri Yapılarının fonksiyonlarla kullanılması

C Veri yapıları üzerinde tanımlı işlemler pek fazla değildir. Genel kural olarak şu söylenebilir: yapılabilecek tek işlem bir veri yapısının & operatörü ile adres bilgisini elde etmektir. Bu, veri yapılarının bir yerden bir yere kopyalanamayacağı ve fonksiyonlarda parametre olarak geçirilemeyeceği anlamına gelir. Ancak yeni kurallara göre bu kısıtlamaların çoğu kaldırılmıştır. Ancak veri yapılarına işaretçiler bu kısıtlamalardan etkilenmezler. O halde fonksiyonlarla veri yapılar beraberce sorunsuz şekilde çalışır.

Bunlar konusundaki tanımları daha fazla açıklayabilmek için örnekleri yazmaya çalışalım. Kendisine gönderilen bir tarih bilgisinin yıl içinde kaçıncı güne geldiğini bulan bir fonksiyon yazalım. Bu fonksiyon şu şekilde yazılabileceği gibi;

```
Yilin_gunu=yil_gun(tarih.yil, tarih.ay, tarih.gun);
```

Bu fonksiyon tarih isimli veri yapısı ile doğrudan çağrılacak şekilde de yazılabilir;

```
Yilin_gunu=yil_gun(&tarih);
```

Bu tanım fonksiyona tarih isimli veri yapısının adresini gönderir ve bir tam sayı döndürür. Bu fonksiyonun yazılmış şekli şu şekildedir.

Örnek 22

```
int yil_gun(struct date *ti){
   int i, day, leap;
   day=ti->day;
   for (i=1;i<ti->month;i++){
      switch(i){
         case 1:
         case 3:
         case 5:
         case 7:
         case 8:
         case 10:
         case 12:day+=31;break
         case 4:
         case 6:
         case 9:
         case 11: day+=30;break
```

```
         case 2:if(ti->year %4 ==0) day+=29;
                              else       day+=28;
      }
   return (day);
}
```

struct date *ti; ifadesi ti adında date veri yapısına bir işaretçi tanımlar. Bu şekilde tanımlanmış bir veri yapısı işaretçisinin öğelerin erişirken **ti->day** şeklinde ifade etmek gerekir. Eğer adı **p** olan bir veri yapısı işaretçisi var ise bu veri yapısının elemanlarına **p->eleman** şeklinde erişilebilir. (-> operatörü bir tire işareti ile yanına yazılmış büyüktür işaretinden oluşmuştur.)

Bu gösterimde p bir işaretçi olduğuna göre (*p).day gösterimi de doğru bir gösterimdir. Ancak alışılagelmiş gösterim -> şeklinde olanıdır.

Örnek 23

Aşağıdaki örnekte Karmaşık bir sayının üzerinde toplama işlemi yapan bir fonksiyon tanımlanmıştır. Sonra bu fonksiyon çağrılmıştır.

```
struct karma{
   double gercek,sanal;
};

void ktopla(struct karma a, struct karma b, struct karma *c){
   c->sanal=a.sanal+b.sanal;
   c->gercek=a.gercek+b.gercek;
}


// Program içerisinde bu fonksiyon şu şekilde çağrılır.
ktopla(a,b,&c);
```

TypeDef

C'de varolan değişken türlerine yenilerini eklemek için kullanılan bir komuttur. Örneğin:

typedef int LENGTH;

kullanımı int adlı türün aynından LENGTH ismiyle oluşturur. Bu smi kullanarak yeni değişkenler tanımlayabilir tür dönüşümlerinden faydalanabilirsiniz.

LENGTH len,maxlen;

Benzer şeilde

typedef char *STRING;

şeklindeki bir tanım karakter işaretçisinin yeniden tanımlanmasını ve adının STRING olmasını sağlar.

STRING p,lineptr[LINES]; şeklindeki kullanımlar p isminde bir karaktere işaretçi tanımlar.

Dikkat edilirse typedef sözcüğü ile tanımlanan yeni tür değişken tanımlanan konumda bulunur. Ve kullanım olarak static veya extern ifadelerinde olduğu gibi görülür. Genellikle oluşturulan yeni türleri var olanlarından ayırt etmek için büyük harf kullanırız.

Daha karmaşık bir örnek olarak önceden verdiğimiz öğrenci örneğini burada kullanalım.

```
typedef struct ogrenci {
   char adsoy[25];
   char adres[40];
   unsigned short vize, final, butunleme;
   struct date dtarihi;
   struct date kaytarihi;
} OGRENCI;
```

yazımında adı OGRENCI olan yeni bir tür tanımlanmıştır.

Aslında typedef komutu yeni bir tür yaratmaz. Sadece bir kısım tanımlara bir yenisini ekler. Gerçekte de define gibi çalışır ve programın derlenmesi sırasında typedef ile oluşturulmuş tanımları ilgili yerlerde değiştirerek elde ettiği yeni kaynak kodunu derler.

Hazır Fonksiyonlar

Diyagnostik ile ilgili fonksiyonlar

void assert(int ifade);

Bu fonksiyon sorunlar ile ilgili nedenlerin öğrenilmesini sağlayacak satırları program içine yerleştirir. İfade hesaplanır ve eğer yanlış değer dönerse assert fonksiyonu çağrı ile ilgili olarak dosya adı, hatalı satırın numarası ifadenin kendisini standart hata çıktısına yazılır. Karşılaşacağınız hata mesajı şekilde bir şey olacaktır:

`Assertion failed:` ifade, **`file`** dosya_adı, **`line`** satır_no

Bu mesajdan sonra assert fonksiyonu abort fonksiyonunu çağırır. Eğer derleyici bildirimlerinden #define NDEBUG var ise assert fonksiyonu işlevini yerine getirmez.

Karakter Fonksiyonları

Bu bölümdeki fonksiyonlar kedilerine parametre olarak gönderilen c değerinin fonksiyon adı ile doğru ifade edilebiliyorsa sıfırdan farklı bir değer aksi halde sıfır değerini döndürürler.

int isalnum(int c)

c karakterinin rakam veya harf olması durumunda 0 den farklı değilse sıfır döndüren fonksiyondur.

int isalpha(int c)

c karakterinin harf olup olmadığını kontrol eder ve harf ise bir döndürür.

int iscntrl(int c)

c karakterinin kontrol karakteri olup olmadığını kontrol eder.

int isdigit(int c)

c karakterinin rakam olup olmadığını kontrol eder

int islower(int c)

c karakterinin küçük harf olup olmadığının kontrol eder.

int isspace(int c)

c karakterinin boşluk olup olmadığını kontrol eder. Bu boşluk karakteri ' ', '\f', '\n', '\r', '\t' ve '\v' karakterlerinden biridir.

int isupper(int c)

c karakterinin büyük harf olup olmadığını kontrol eder.

int tolower(int c)

Kendisine gönderilen c karakterinin büyük harf olması durumunda küçük harfe çeviren ve döndüren büyük harf olmaması durumunda olduğu gibi döndüren fonksiyondur.

int toupper(int c)

Kendisine gönderilen c karakterinin küçük harf olması durumunda büyük harfe çeviren ve döndüren küçük harf olmaması durumunda olduğu gibi döndüren fonksiyondur.

Matematik Fonksiyonları

Bu fonksiyonları kullanabilmek için math.h kütüphanesini programınıza include değimi ile dahil etmeniz gerekecektir.

double acos(double x)

x değerinin Arc Cosinüsünü hesaplar. X değeri –1 ile 1 arasında olmalıdır ve dönen değer 0 ile 3.14 olur.

double asin(double x)

x değerinin Arc Sinüsünü hesaplar. X değeri –1 ile 1 arasında olmalıdır ve dönen değer 0 ile 3.14 olur.

double atan(double x)

x değerinin Arc Tanjantını hesaplar.

double cos(double x)

x değerinin Cosinüsünü hesaplar. X değeri radyan cinsinden bir açı değeri olmalıdır.

double sin(double x)

x değerinin sinüsünü hesaplar. X değeri radyan cinsinden bir açı değeri olmalıdır.

double tan(double x)

x değerinin tanjantını hesaplar. X değeri radyan cinsinden bir açı değeri olmalıdır.

double log(double x)

x değerinin tabi logaritmasını hesaplar. X değeri pozitif bir sayı olmalıdır.

double log10(double x)

x değerinin 10 tabanına göre logaritmasını hesaplar. X değeri pozitif bir sayı olmalıdır.

double pow(double x, double y)

x üzeri y değerini hesaplar.

double sqrt(double x)

x değerinin karekökünü hesaplar. X değeri pozitif bir sayı olmalıdır.

double log(double x)

x değerinin tabi logaritmasını hesaplar. X değeri pozitif bir sayı olmalıdır.

double ceil(double x)

x'den küçük olmayan en küçük tam sayıyı bulan fonksiyondur. Yani x değerini yukarı yuvarlayan fonksiyondur.

double floor(double x)

x'den büyük olmayan en büyük tam sayıyı bulan fonksiyondur. Yani x değerini aşağı yuvarlayan fonksiyondur.

double fabs(double x)

x değerinin mutlak değerini hesaplar.

Genel Amaçlı Fonksiyonlar

double atof(const char *nptr)

nptr ile gösterilen ifadenin ilk bölümündeki anlamlı sayısal değerleri kesirli değere döndürür.

int atoi(const char *nptr)

nptr ile gösterilen ifadenin ilk bölümündeki anlamlı sayısal değerleri tam değere döndürür.

long int atol(const char *nptr)

nptr ile gösterilen ifadenin ilk bölümündeki anlamlı sayısal değerleri uzun tamsayıya değere döndürür.

int rand(void)

0 ile RAND_MAX aralığında rasgele bir sayı üretir.

void srand(unsigned int sed)

sed değeri ile rasgele sayı üretecini başlatır. Bu fonksiyona gönderilecek sed değeri üretilecek rasgele değerlerin her seferinde farklı diziler şeklinde üretilmesini sağlayacaktır.

Alıştırma 34

Srand() ve rand() fonksiyonlarını kullanarak 0..1000 aralığında gerçek rasgele sayılar üreten ve ürettiği bu sayıları "rast.txt" isimli bir dosyaya yazan C programını geliştiriniz.

void *malloc (size_t miktar)

miktar ile belirtilen (işaretsiz tam sayı) miktarda bellek sahasını heap dediğimiz yerden ayırarak çağrılan programa bu bellek sahasının adresini döndürür.

void *realloc(void *ptr, size_t miktar)

ptr ile gösterilen bellek sahası için ayrılmış miktarın miktar kadar bir alana değiştirilmesini sağlayan ve ayrılan yeni alan için bellek sahasını döndürür

void free(void *ptr)

ptr ile gösterilen bellek sahası için ayrılmış alanın serbest bırakılmasını sağlar.

void exit(int status)

Programın normal şekilde sonlandırılmasını sağlar.

int abs(int j)

j ile gösterine değerin mutlak değerini döndürür.

Karakter Dizisi Fonksiyonları

Bu fonksiyonlar karakter dizeleri üzerinde işlemler yapan fonksiyonlardır. Programlarınızda kullanılabilmeleri için string.h kütüphanesini dahil etmemiz gerekecektir.

char *strcpy(char *s1, const char *s2)

s2 ile gösterilen dize içindeki ifadeyi s1 ile gösterilen dizeye kopyalar.

char *strncpy(char *s1, const char *s2, size_t n)

s2 ile gösterilen dize içindeki ifadeyi n harften fazla olmamak üzere s1 ile gösterilen dizeye kopyalar.

int strcmp(char *s1, const char *s2)

s2 ile gösterilen dize içindeki ifadeyi s1 ile gösterilen dize ile karşılaştırır. İki dize eşit ise 0, birincisi büyük ise pozitif ikincisi büyük ise negatif sonuç üretilir

int strncmp(char *s1, const char *s2, size_t n)

s2 ile gösterilen dize içindeki ifadeyi n harften fazla olmamak üzere s1 ile gösterilen ifade ile karşılaştırır.

char *strcat(char *s1, const char *s2)

s2 ile gösterilen dize içindeki ifadeyi s1 ile gösterilen dizenin peşine ekler.

char *strncat(char *s1, const char *s2, size_t n)

s2 ile gösterilen dize içindeki ifadeyi n harften fazla olmamak üzere s1 ile gösterilen dizenin peşine ekler.

size_t strlen(const char *s)

s ile gösterilen dize içindeki ifadeni kaç harften oluştuğunu bulan fonksiyondur.

Tarih ve Saat Fonksiyonları

Bu fonksiyonlar sistem tarih ve saati ile ilgili işlemleri yerine getiren fonksiyonlardır. Kullanılabilmeleri için time.h kütüphanesini programınıza dahil etmeniz gerekecektir. Bu fonksiyonları kullanırken aşağıdaki veri yapısını kullanmanız da gereklidir.

```
struct tm {
   int tm_sec;
   int tm_min;
   int tm_hour;
   int tm_mday;
   int tm_mon;
   int tm_year;
   int tm_wday;
   int tm_yday;
   int tm_isdst
}
```

Bu arada bu kütüphane içinde tanımlı şu yeni türleri de fonksiyon tanımlarında göreceksiniz.

clock_t zamanı göstermeyi sağlayan aritmetik bir tür.

time_t zamanı göstermeyi sağlayan aritmetik bir tür.

size_t unsigned int türünde yeni bir değişken.

Double difftime(time_t zaman1, time_t zaman0)

İki zaman ölçüsü arasındaki farkı hesaplar fark saniye cinsinden ve double türünde döndürülür. (zaman1-zaman0)

time_t time(time_t timer)

Bu fonksiyon geçerli zamanı bulur.

char *asctime(const struct tm *timeptr)

struct tm yapısındaki bir zaman bilgisini Sun Sep 16 01:03:52 1973\n\0 şeklindeki bir karakter dizesine çevirir.

struct tm *gmtime(const time_t *timer)

Güncel zaman bilgisini struct tm şeklindeki bir zamana dönüştürür.

struct tm *localtime(const time_t *timer)

Güncel zaman bilgisini struct tm şeklindeki bir zamana dönüştürür. Bulunan bu zaman bilgisi yerel zaman bilgisidir.

Örnek 24

Sistem üzerindeki saat bilgisini ekrana yazan bir C Programı geliştiriniz.

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

int main(){
   struct tm zaman;
   time_t zz;

   zz=time();
   zaman=localtime(zz);
   printf("Gün:%d\nAy :%d\nYıl:%d\n",
zaman.tm_mday,zaman.tm_mon+1,zaman.tm_year);
}
```

Derleyici Bildirimleri

Bu bölümde yer alan konular büyük programlar yazabilmek için #include ifadesinin nasıl kullanılacağını; makro ve sembolik sabitler tanımlamak için #define ifadesinin nasıl kullanılacağını ve Koşullu derlemenin ne demek olduğunu anlatacaktır. Bu konular C ön derleyicisinin ne olduğunu anlamanızı sağlayacaktır. Ön derleme işlemi C derleyicisi programınızı derlemeye başlamadan önce gerçekleşen işlemdir. Bu adımda şu işlemler yapılır.

- Başka dosya ve kütüphanelerin programa dahil edilmesi,
- Sembolik sabit ve makroların tanımlanması,
- Program kodlarının koşullu derlenmesinin sağlanması,
- Ön derleyici kodlarının koşullu olarak derlenmesi,

Tüm ön derleyici bildirimleri # (diyez) işareti ile başlar.

#include bildirimi

Bu bildirim ile belirtilen dosyanın bir kopyasını bildirimin bulunduğu yere kopyaladığımızı söyleyebiliriz. Bu bildirim aşağıda görüldüğü gibi iki farklı şekilde kullanılabilir.

```
#include<dosyaadı>
#include "dosyaadı"
```

Bu iki kullanım arasındaki fark derleyicinin dosyayı arayacağı yeri belirtmekten kaynaklanır. Tırnak arasına yazılmış bir dosya derleyici tarafından dosyanın bulunduğu klasörün içinde aranır. Bu tarz genellikle programcının kendi yazdığı dosyaları kaynak koda yerleştirmek istediği zaman kullanılır. Diğer gösterimde eklenecek dosya sistemin standart include klasörünün içinde aranır. Bu genellikle işletim sisteminden ve derleyiciden bağımsız noktaları işaret eder. Çoğunlukla da Derleyiciye ait kütüphane başlık dosyalarını programınıza eklemenizi sağlar.

Bu bildirim genellikle programınızda kullandığınız kütüphanelere ait başlık dosyalarının programınıza dahil edilmesini sağlar. Bir başlık dosyasının içinde ise çoğunlukla o konu ile ilgili sembolik sabitlerin, fonksiyon prototip tanımlarının, veri yapılarının, union ve enum türü tanımların bulunduğu bilinir.

#define bildirimi

Bu bildirim programınızda sembolik bir sabit oluşturmanızı sağlar. Genel kullanımı şu şekildedir:

```
#define sembolik_ifade yerine_konacak_ifade
```

Derleyici böyle bir satırla karşılaştığında programın geri kalan kısmında sembolik_ifade ile gösterilen yere yerine_konacak_ifade denilen değerleri yerleştirir ve sonra derlemeye devam eder.

```
#define PI 3.14159
```

şeklindeki bir ifade PI adında bir sembolik sabit tanımlar. Ön derleyici derleme işleminden önce program içerisinde PI gördüğü yere 3.14159 değerini yerleştirir. Bu tarz bir işlem programcılara belirli değerler için sabitler tanımlamayı ve programda bu sabitleri kullanmayı sağlar. Bu işlem ise kullanıcının bir takım değerlere isim vermesini ve gerektiğinde çok kolay bir şekilde değiştirmesini sağlar. Örneğin yukarıdaki PI örneğinde 3.14159 değerinin değişmesi gerekirse sadece #define tanımındaki sayısal değerin değişmesi yeterlidir. Programın yeniden derlenip çalıştırılması durumunda tüm hesaplamalar PI'nin yeni değeri için yapılır. Bu tür sabitler için anlamlı isimlerin kullanılmış olması programın anlaşılabilirliği ve belgelendirilmesi noktasında yardımcı olacaktır.

Ancak #define ifadesi sadece sembolik sabitler tanımlamaz. Aynı zamanda makro tanımları yapmanızı da sağlar. Bu işlem için de yukarıdaki kullanım şekli aynıdır. Makrolar hem parametreleri hem de parametresiz tanımlanabilirler. Parametresiz bir Makro sembolik sabit gibi işlem görür. Parametreli kullanımda ise parametreler yerine konan ifadesinde uygun yerlere yerleştirilerek açılırlar. Şu şekilde bir örnek düşünelim.

`#define CIRCLE_AREA(x) (PI * (x) * (x) )` böyle bir tanımın bulunduğu programda şu satır yazılırsa,

`alan=CIRCLE_AREA(4);` ifadesi ön derleyici tarafından şu şekle açılır

```
alan=(3.14159 * (4) * (4) );
```

Bu örnekte parantezlerin gereksiz olduğu düşünülebilir. Ancak makro tanımı daha farklı çağrılmış olsaydı parantezlerin son derece gerekli olduğu ortaya çıkacaktı.

`alan=CIRCLE_AREA(c+2);` ifadesi ön derleyici tarafından şu şekle açılır

```
alan=(3.14159 * (c+2) * (c+2) );
```

Burada parantezlerin gerekliliği ortaya çıkmaktadır. Aksi halde işlem önceliğinden dolayı işlem yanlış hesaplanacaktı. Şöyle ki

```
alan=(3.14159 * c+2 * c+2 );
```

işlemi tamamen hatalı bir sonuca bizi götürecektir.

#if...#endif bildirimi

#if ve #endif değimleri derleyicinin ve ön derleyicinin derleme sürecini kontrol edecektir. Tüm #if değimleri bir sabit tamsayı hesaplar. Genel görünüşü şu şekildedir:

```
#if !defined(NULL)
   #define NULL 0
#endif
```

Bu tümce NULL diye bir sabitin tanımlı olup olmadığına bakar ve tanımlı değil ise NULL sabitini 0 olarak tanımlar. Aksi halde NULL zaten tanımlıdır. Her **#if** bir **#endif** ile bitmek zorundadır. **#ifdef** ve **#ifndef** şeklinde iki kısa bildirim daha tanımlıdır ve bunlar **#if define(SABIT)** ve **#if ! define(SABIT)** şeklindeki tanımların kısa biçimleridir. Bunun haricinde **#elif** ve **#else** şeklinde koşulun olumsuz durumlarını değerlendirmek üzere iki bildirim daha tanımlıdır.

Program geliştirme sürecinde bazen programcılar bazı bölümlerin derlenmesini istemeyebilirler. Bu durumda ön derleme bildirimlerinde şu şekilde faydalanılabilir.

```
#if 0
    derlenmesini istemediğiniz kod bölümleri
#endif
```

Bu şekildeki bir bölümün derlenmesini sağlamak amacıyla 0 yerine 1 konulabilir.

Bu şekildeki koşullu derleme işlemleri çoğunlukla hata ayıklama maksadıyla kullanılır. Aslında bu tür bir işlem için debugger adı verilen yazılımlar vardır ama bunları anlayıp kullanabilmek her zaman kolay olmayabilir. Onun yerine programcılar zaman zaman printf komutunu istedikleri yerlerde kullanarak kendileri için yeterli olacak bir takım değerlerin yazılmasını sağlayabilirler.

```
#if DEBUG
   printf("Değer:%d\n", x);
#endif
```

Şeklindeki bir bölümde eğer DEBUG adında sembolik sabit tanımlanmışsa printf satırı derlenip koda katılacaktır. Aksi halde bu satır derlenmeyecektir. Bu programın dağıtım sürümünü elde ettiğimizde DEBUG sembolik sabitini tanımlayan satırlar kaldırılarak derlenmelidir. Bu da elde edilen kodda printf satırının olmamasını sağlayacaktır.

#error ve #pragma bildirimleri

#error mesaj kelimeleri

gerçekleştirime bağlı olan bir hata mesajı görüntülemeyi sağlar. Mesaj kelimeleri istediğiniz kadar kelimeden oluşabilir. Örneğin

```
#error 1 - Değer Taşması
```

şeklindeki bir ifade derleyici tarafından işlenirken**#error** ifadesi yanındaki kelimeler hata mesajı olarak ekrana yazılacaktır. Derleme işlemi duracaktır. Program derlenmeyecektir.

#pragma bildirimi de derleyiciye bağlı davranışlar sergiler. Eğer kullandığınız derleyici gerçekleştirimi **#pragma** bildirimini desteklemiyor ise bildirim dikkate alınmaz.

Parametre Tanımlama

C dilini destekleyen bir çok ortamlarda komut satırındaki parametre veya argümanların programa aktarılmasını ve değerlendirilmesini sağlayan yapılar mevcuttur. main fonksiyonu ilk çalıştırıldığı zaman iki parametre ile kendisine değer aktarılır. Bunlardan ilki (genellikle argc olarak adlandırılır) komut satırındaki parametrelerin sayısı verir. İkinci parametre ise (argv olarak adlandırılır) bir karakter dizesine işaretçi değeridir. Bu karakter dizesinin her bir elemanı parametre olarak verilen ifadeleri saklayan değerledir. Bu karakter dizesi ve işaretçiler üzerinde oynamak işaretçilerle ilgili derste gördüğümüz işlemleri yapabilmenizi gerektirir.

Bu özelliğin nasıl kullanıldığını gösterebilecek en basit program echo olarak adlandırılabilir. Bu program kendisine parametre olarak gönderilen ifadeleri ekrana yazan bir program olsun. Şu şekilde kullanıldığında;

```
Echo merhaba dünya
```

Çıktı merhaba dünya biçiminde olacaktır.

Alışılagelmiş olarak argv[0] programın çalıştırılmasını sağlayan çalıştırılabilir komut dosyasının adını bildirir. Bu şartlar altında argc en az 1 değerine sahiptir. Yukarıdaki örnekte argc 3 olmalıdır. argv[0] echo; argv[1] merhaba; argv[2] ise dünya değerlerine sahip olur. Birinci gerçek parametre argv[1] olarak elde edilirken son parametre ise argv[argc-1] şeklinde elde edilir. Bu durumda argc 1 ise program gerçek bir parametre ile çalıştırılmamıştır. Yukarıdaki echo programının şu şekilde yazıldığını düşünebiliriz.

Örnek 25

```
main(argc, argv)
int argc;
char *argv[];
{
   int i;

   for (i=1; i<argc; i++)
      printf("%s,%c",argv[i],(i<argc-1) ? ' ' : '\n')
}
```

argv bir işaretçi olduğu için bu programı yazmak için farklı algoritmalar düşünülerek yazılabilir. İşaretçi aritmetiği ve mantığı ile ilgili olarak geliştirilecek bu programlardan biri şu şekilde olabilir:

```
main(argc, argv)
int argc;
```

```
char argv[];
{
   while (--argc>0)
      printf("%s%c",*++argv,(argc>1) ? ' ' : '\n');
}
```

argv bir karakter dizisine işaretçi olduğu için ++argv bu dizi içinde sıradaki diğer elemanı gösterir. İşaretçi değerinin her artımında argc bir azaltılacağı için argc>0 olduğu sürece geçerli bir dizi elemanına bu tanım sağlanmış olacaktır.

Alıştırma 35

Klavyeden "islem 8 X 3" şeklinde belirtilen bir dört işlemi yapabilen sonucu ekrana yazan ve eksik parametre girildiğinde kullanıcıyı ekran vasıtasıyla uyaran bir C programı geliştiriniz.

Alıştırma 36

Dosya adı klavyeden parametre olarak girilen bir dosyayı ekrana listeleyen programı C Programlama dili ile geliştiriniz.