Bilgisayar Programlama I

C Programlama Dili'nin Temelleri

C Dilinin Tarihçesi

C dilinden önceki tarihsel gelişmeler dikkate alındığında C dilinin UNIX işletim sisteminin geliştirilmesinin bir yan ürünü olduğunu söylemek yanlış olmayacaktır. Bu gelişim aşamalarına şöyle bir bakalım.

1965 yılında Massachusetts Institute of Technology (MIT)'de MAC isimli bir proje gerçekleştirildi. MAC bir bilgisayar sisteminin zaman paylaşımını sağlayan ilk projelerden biriydi. Bu proje ile aynı bilgisayar 30'a kadar kullanıcı tarafından paylaşılabiliyordu 160 ayrı yazıcıyı kullanmak da mümkündü. Bu projenin başarısından cesaret alan MIT, General Elektrik ve Bell Laboratuvarları ile bir ortak girişim oluşturarak zaman paylaşımlı yeni bir sistem oluşturma çalışmasına başladı. Bu projeye MULTICS (*Multiplexed Information and Computing Service* - Çok Katmanlı Bilgi ve Hesaplama Hizmeti) ismi verildi. Bell Laboratuvarları projenin yazılım kısmından sorumluydu. Ancak 1969 yılında Bell Laboratuvarları proje süresinin uzaması ve proje maliyetinin yüksek olması nedeniyle projeden ayrıldı.

1969 yılında Bell Laboratuvarları'nda Ken Thompson öncülüğünde bir grup yeni bir alternatif arayışına girdi. *Thompson* ve ekibi, insan ve makine arasındaki iletişimi kolaylaştıracak bir işletim sistemi tasarımına girişti. İşletim sisteminin omurgası yazıldığında MULTICS'e gönderme yapılarak işletim sistemine **UNIX** ismi verildi. Thompson DEC firmasının ana belleği yalnızca 8K olan PDP 7 isimli bilgisayarı üzerinde çalışıyordu ve kendi dilini tasarlarken, Martin Richards tarafından 1960 yıllarının ortalarında geliştirilmiş olan BCPL (Business Common Programming Language) dilinden yola çıktı. BCPL ise CPL (Cambridge Programming Language)'den türetilmiştir. CPL'in kaynağı da tüm zamanların en eski ve en etkili dillerinden biri olan ALGOL 60 (Algorithmic Language 1960) 'dır. ALGOL dilinde temel amaç, donanıma özgü özelliklere başvurmadan algoritmaları ifade etmek için evrensel bir dil oluşturmaktı. Bu nedenle bu dilde hiçbir giriş/çıkış fonksiyonu tanımlanmamıştır. Bu dilde yazılmış kod örnekleri alttaki resimde gösterilmiştir. ALGOL 60, Pascal, ADA, Modula2 dillerinin de atasıdır, bu dillere bu yüzden C dilinin "kuzenleri" de denebilir.

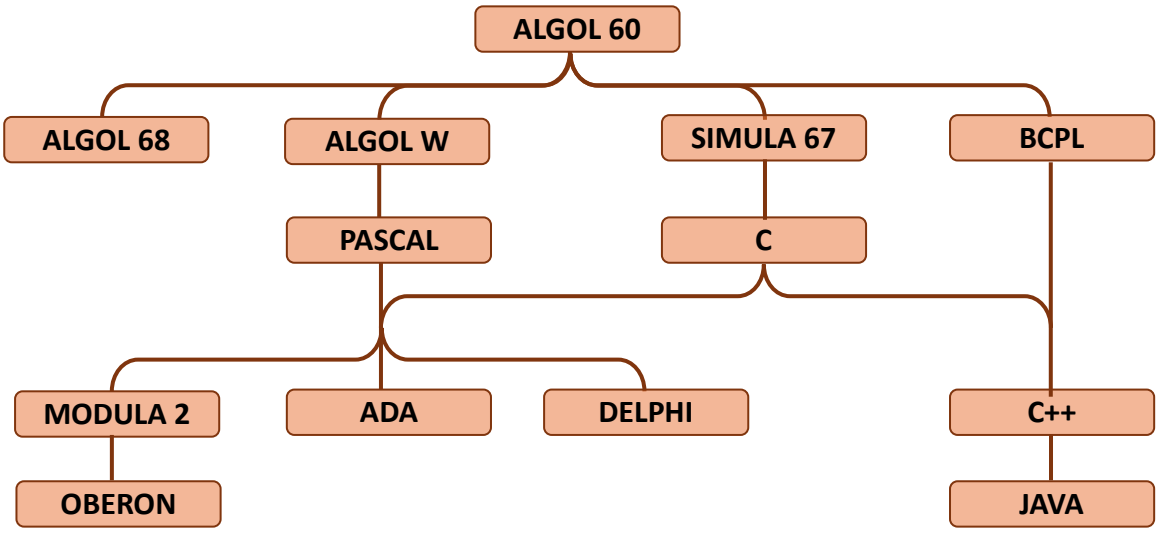
```
'BEGIN'
    'PROCEDURE' INC(A, B);
    'INTEGER' A, B;
    'BEGIN'
        A := A + 1;
        B := B + 1;
    'END'

    'INTEGER' 'ARRAY' X(/10/);
    'INTEGER' J;

    X(/1/) := 10;
    X(/2/) := 20;

    J := 1;
    INC(J, X(/J/));

    OUTINTEGER(1, J);
    OUTINTEGER(1, X(/2/));
'END'
```

```
'BEGIN'
    'INTEGER' I, J;
    I := 3;
    OUTINTEGER(1, I);
    'BEGIN'
        'INTEGER' I;
        'BOOLEAN' STATE;
        I := 4;
        OUTINTEGER(1, I);
    'END'
    OUTINTEGER(1, I);
'END'
```

```
'COMMENT' Block example;
'BEGIN'
    X: 'BEGIN'
        FN1(10);
        FN1(20);
    'END' this is the end of the X block
'END' and this the unnamed block
```

ALGOL programlama dilinde yazılmış kod örnekleri

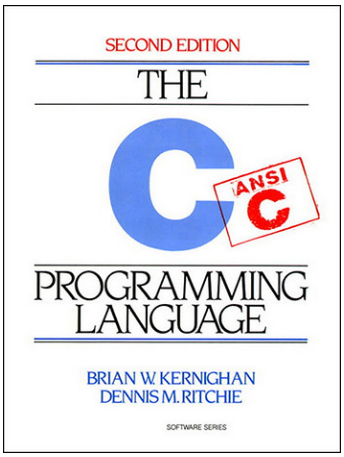
ALGOL 60 ailesi:



Thompson geliştirdiği programlama diline ilk başta B ismini vermiştir. Dennis Ritchie UNIX projesine katılınca, B dilinde programlama yapmaya başlamıştır. B dili daha da gelişerek daha yeni bir teknoloji olan PDP-11 bilgisayarlarında da kullanılmaya başlanmıştır. Thompson, UNIX işletim sisteminin bir kısmını B dilinde tekrar yazmıştır. 1971 yılına gelindiğinde, B dilinin PDP-11 bilgisayarları ve UNIX işletim sisteminin geliştirilmesi için çok uygun olmadığı iyice ortaya çıkmıştır. Bu yüzden Ritchie, B programlama dilinin daha ileri bir versiyonunu geliştirmeye koyulmuştur. Oluşturduğu dili ilk önce NB (new B) olarak isimlendirmiştir ama geliştirdiği dil, B dilinden iyice kopmaya ve ayrı bir karakter göstermeye başlayınca dilin ismini de C olarak değiştirmiştir. 1973 yılında UNIX işletim sisteminin büyük bir kısmı C dili ile tekrar yazılmıştır.

C'nin evrimi ve gelişmesi 70'li yıllarda da devam ederek, geniş kitleler tarafından tanınması ve kullanılmaya başlaması, 1978 yılında Dennis Ritchie ve Brian Kernighan tarafından yazılan "The C Programming Language" kitabı ile olmuştur. Bu kitap aynı zamanda yazılım konusunda yazılan en iyi eserlerden biri olarak değerlendirilmektedir. C'nin standardize edilmesine kadar olan dönemde bu kitap, çoğunluğun benimsediği ve genel kabul gören gayri resmi bir standart vazifesi de görmüştür.

Ken Thompson ve Dennis Ritchie Unix İşletim Sistemi üzerinde çalışırken (Yıl: 1972)



1978 yılında yazılan "The C Programming Language" kitabı

C Programlama Dili Tanıtım

Bir programlama dilinin seviyesi (level) o programlama dilinin insanın konuşma diline ya da makine diline yakınlığının bir ölçüsüdür. İnsanın konuşma diline ne kadar yakınsa dil o kadar yüksek seviyeli (high level), bilgisayarın elektronik yapısına ve çalışma biçimine ne kadar yakınsa da dil o kadar düşük seviyeli (low level) olarak adlandırılmaktadır. C dili bu ölçüye göre orta seviyeli bir programlama dilidir. Diğer yapısal programlama dillerine göre C dilinin seviyesi daha düşüktür. C dili hem yüksek seviyeli dillerin kontrol deyimleri, veri yapıları gibi avantajlarına sahipken hem de ikili sayı operatörleri gibi makine kodu deyimlerine benzer operatörleri içermektedir. Bu açıdan C dili hem makine diline hem de insan diline yakın bir dildir. Diğer bir deyişle C makineye yeterince yakındır ama programcıya da çok uzak değildir. Diğer dillere kıyasla C dilinin daha çok tercih edilmesinin esas sebeplerinde biri de budur.

Bir programı C dili kullanarak yazmak, aynı programı makine dilinde yazmaya kıyasla daha kolaydır. C dili performans ve verimlilik açısından makine dilinden daha aşağı değildir, onun yerine tercih edilebilir. Bu bakımdan C dilinin esas uygulama alanı “Sistem programlama”dır. Yani donanımın yönetilmesi, yönlendirilmesi ve denetimi için yazılan, doğrudan makinenin donanımla ilişkiye giren programların hazırlanmasına yönelik kullanılmaktadır. Örneğin, işletim sistemleri, derleyiciler, yorumlayıcılar, aygıt sürücüleri (device drivers), bilgisayarların iletişimine ilişkin programlar, otomasyon programları birer sistem programıdır.

C dilinden önce sistem programları assembly türü dillerle hazırlanıyordu. Günümüzde ise bu sistem programları çoğunlukla C dili kullanılarak hazırlanmaktadır. Bugün cep telefonlarından uçaklara kadar birçok yerde C kodları kullanılmaktadır. C dilinin bu kadar çok tercih edilmesinde ana sebep C dilinin sahip olduğu özelliklerdir. Bunlar şu şekilde özetlenebilir:

- C algoritmik bir dildir. C dilinde program yazmak için yalnızca dilin sözdizimini ve anlamsal yapısını bilmek yetmez genel bir algoritma bilgisi de gerekir.
- C diğer dillerle kıyaslandığında taşınabilirliği çok yüksek olan bir dildir. Çünkü 1989 yılından bu yana genel kabul görmüş standartlara sahiptir. İfade gücü yüksek, okunabilirlik özelliği güçlü bir dildir.
- C çok esnek bir dildir. Diğer dillerde olduğu gibi programcıya kısıtlamalar getirmez. Makinanın olanaklarını programcıya daha iyi yansıtır.
- C güçlü bir dildir, çok iyi bir biçimde tasarlanmıştır. C'ye ilişkin operatörlerin ve yapıların birçoğu daha sonra başka programlama dilleri tarafından da benimsenmiştir.
- C verimli bir dildir ve düzeyinin düşük olması nedeniyle hızlı çalışır. Verimlilik konusunda assembly diller ile rekabet edebilir.
- C doğal bir dildir ve bilgisayar sistemleriyle uyum içindedir.
- C küçük bir dildir. Bu ise yeni sistemler için derleyici yazmayı kolaylaştırır.

- C dilinin standart bir kütüphanesi vardır. Bu kütüphane ile sık yapılan işlemlerin daha verimli kullanılmasını sağlanmıştır.
- C dilinin öğrenilmesi diğer programlama dillerine göre biraz zordur.

Birçok avantajına rağmen diğer programlama dillerinde olduğu gibi C dilinin de bazı zayıf yönleri bulunmaktadır. Esnek ve güçlü bir dil olması dikkatsiz bir programcının hata yapma riskini artırır. Ayrıca C dilinde yazılan kodlarda yapılan yanlışlıkların bulunması diğer dillere göre daha zor olabilmektedir.

C Derleyicileri

Derleyiciler, bir programlama dilinde yazılmış olan kodları, bilgisayarın işlemcisi tarafından doğrudan çalıştırılabilen makine diline çeviren özel yazılımlar olarak tanımlanır. Bu süreçte, derleyici, yüksek seviyeli programlama dili komutlarını alır ve bunları işlemci tarafından anlaşılabilir olan düşük seviyeli talimatlara dönüştürür. Bir derleyici bu işlemleri şu adımlarla gerçekleştirir:

1. Kodun Analizi (Lexical Analysis): İlk olarak, derleyici, yazdığınız kodu küçük parçalara böler. Bu parçalar, "kelime" olarak da adlandırılır ve değişkenler, anahtar kelimeler (örneğin `if`, `for`) gibi dilin temel yapı taşlarından oluşur.

2. Sözdizimi Analizi (Syntax Analysis): Daha sonra, derleyici bu kelimeleri bir araya getirerek anlamlı cümleler oluşturur. Bu cümleler, programın nasıl çalışacağını belirleyen komutlardır. Eğer kodunuzda bir yazım hatası varsa, derleyici bu aşamada hatayı fark eder ve size bildirir.

3. Anlam Analizi (Semantic Analysis): Derleyici, cümlelerin mantıksal olarak doğru olup olmadığını kontrol eder. Örneğin, bir sayıyı bir metinle toplamak gibi mantıksız bir işlem yapılmaya çalışıldığında, derleyici bunu fark eder.

4. Optimizasyon (Optimization): Derleyici, kodun daha hızlı çalışması veya daha az yer kaplaması için bazı iyileştirmeler yapabilir. Bu, kodun aynı işlevi koruyarak daha verimli hale getirilmesi anlamına gelir.

5. Makine Diline Çeviri (Code Generation): Son adımda, derleyici, anlamlı cümleleri makine diline çevirir. Artık bilgisayar, bu komutları doğrudan çalıştırabilir. Bu aşamada ortaya çıkan dosya, çalıştırılabilir bir program dosyasıdır.

C programlarını derleyebilmek için, kullanılan işletim sistemine uygun bir C derleyicisinin temin edilmesi zorunludur. İşletim sistemleri, derleyicilerin nasıl çalıştığını ve nasıl derlenmiş çıktılar oluşturduğunu belirler, bu nedenle derleyiciler farklı işletim sistemleri için farklılık gösterir. C derleyicileri, kullanılan işletim sistemine ve platforma göre çeşitli biçimlerde karşımıza çıkar. Örneğin, UNIX tabanlı sistemler için farklı, Windows tabanlı kişisel bilgisayarlar için ise farklı C derleyicileri bulunmaktadır. Bu nedenle, her bir platformda C programlarını derlemek için uygun derleyiciyi seçmek önemlidir.

Ayrıca, her derleyicinin kendine özgü bazı özellikleri ve komutları olabilir. Bu özellikleri ve komutları öğrenmek için, derleyici ile birlikte gelen belgeleri incelemek faydalı olacaktır. Bununla birlikte, C programlarında yaygın olarak kullanılan standart kütüphaneler, farklı derleyicilerde genellikle sorunsuz çalışır ve bu da C dilinin platformlar arası uyumluluğunu artırır.

Bir C Programının Çalıştırılması

Bir C programının bilgisayarda çalıştırılabilmesi için birkaç temel aşamadan geçmesi gerekir. Bu aşamalar, yazdığınız programın ham metin dosyasından, bilgisayarın anlayabileceği çalıştırılabilir bir dosya haline gelmesine kadar olan süreci kapsar. İşte bu aşamalar

1. Yazılım Aşaması (Coding)

İlk olarak, C programınızı bir metin düzenleyici kullanarak yazarsınız. Bu aşamada, programınızın tüm kodlarını "**.c**" uzantılı bir dosyada saklarsınız. Bu dosya, programınızın "kaynak kodu" olarak adlandırılır ve bu noktada henüz sadece bir metin dosyasıdır. Bu aşamada yazılan kodlar diğer kişiler tarafından okunabilir ve anlaşılabilir bir yapıya sahiptir.

2. Ön İşlem Aşaması (Preprocessing)

Yazdığınız kaynak kod, derleme sürecinin başlamasından önce ön işlemci adı verilen bir program tarafından işlenir. Ön işlemci, kaynak kodunuzdaki "#include" ve "#define" gibi direktifleri işler, makroları genişletir, dosyaları birleştirir ve gerekli yerlerde düzenlemeler yapar. Bu aşamada yapılan işlemler, kodun derleyiciye daha hazır bir hale getirilmesini sağlar.

3. Derleme Aşaması (Compilation)

Ön işlemden geçen kod, derleyiciye gönderilir. Derleyici, bu kodu makine diline çevirmeye başlar. Ancak bu aşamada ortaya çıkan dosya henüz çalıştırılabilir bir dosya değildir; "amaç kodu" (*object code*) adı verilen bir ara dosyadır. Bu dosya, işlemci tarafından doğrudan çalıştırılamaz ama programın farklı bölümlerinin bir araya getirilmesi için gerekli bir aşamadır.

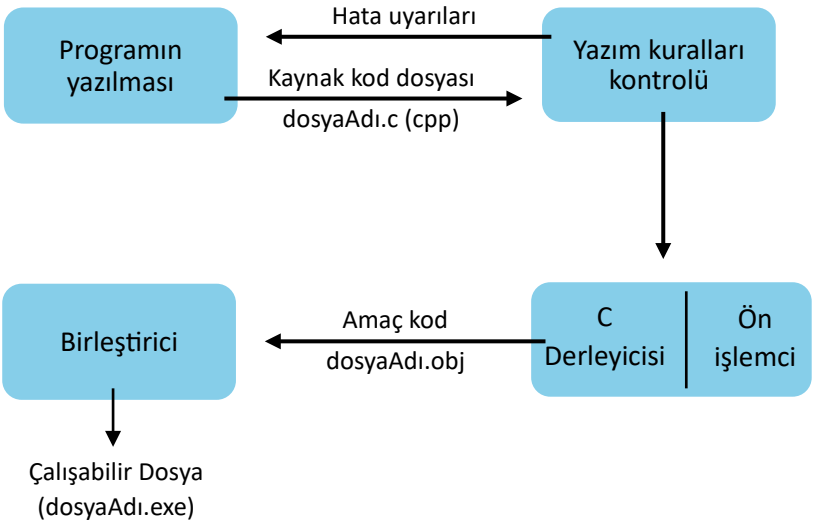
4. Bağlama Aşaması (Linking)

Derleme aşamasında elde edilen amaç kodu, tek başına çalışmaya uygun değildir çünkü bir program genellikle birçok farklı dosyadan ve kütüphaneden gelen kodlara ihtiyaç duyar. **Bağlayıcı** (*linker*) adı verilen program, bu dosyaları ve kütüphaneleri bir araya getirerek tek bir çalıştırılabilir dosya oluşturur. Bu aşama, programın dışarıdan ihtiyaç duyduğu tüm parçaların bir araya getirildiği ve çalışmaya hazır hale getirildiği adımdır.

5. Çalıştırma Aşaması (Execution)

Son olarak, bağlama aşamasından çıkan çalıştırılabilir dosya, bilgisayarınızda doğrudan çalıştırılabilir hale gelir. Bu dosya, artık bilgisayarın işlemcisi tarafından çalıştırılabilir ve programınızın gerçek işlevlerini yerine getirir.

Bu aşamalar alttaki şekilde toplu olarak gösterilmiştir. Özetlemek gerekirse programın kaynak kodunun yazılması, bir editör yardımı ile komutların doğru bir sırada yazılması anlamına gelir. Bu amaçla herhangi bir editör (Visual Studio Code - VS Code, Sublime Text, JetBrains IntelliJ IDEA v.b.) kullanılabileceği gibi, C derleyicisi ile birlikti gelen editör de kullanılabilir.



C dilinde yazılan bir programın çalışması için gereken aşamalar

Programın derlenmesi, doğrudan işletim sisteminden derleyicinin çalıştırılması ile ya da derleyici ile birlikte gelen arayüz aracılığı ile gerçekleştirilebilir. Program derlenirken, derleyici tarafından anlaşılamayan hatalı durumlar programcıya geri bildirilir. Bu durumda programcı, derleyicinin işaretlemiş olduğu hatalı satırları inceleyerek düzeltmeli ve programı tekrar hatasız olana kadar derleme işlemine devam edilmelidir. Derleme sonrası hatasız olan kod, birleştiricinin gerekli kütüphaneleri koda eklemesi sonrasında çalışmaya hazır hale gelmiş olur.

C DİLİNİN TEMELLERİ

Genel Kavramlar

Bir programlama dilini öğrenirken, komutların yanı sıra, bunların nasıl kullanılacağını ve dikkat edilmesi gereken kuralları da öğrenmek gerekir.

Örneğin program satırları hangi düzende oluşturulacak, satırlar nasıl sonlandırılacak, program noktalama işaretleri ne için kullanılacak, program başlangıcı ve program sonu nasıl belirlenecek gibi. Bu bölümde bu konulara önce genel olarak değinilecek, daha sonra ayrıntılı olarak bilgi verilecektir.

Alttaki örnekte görülen program, **kaynak kod** olarak adlandırılır. Kaynak kod herhangi bir metin editörü kullanarak yazılır. Satırların başlarında bulunan satır numaraları C'de kullanılmaz, sadece açıklama yaparken takip kolaylığı olması açısından burada yer almıştır. Kodun 1 numaralı satırında ok işaretinden sonra yer alan ifadedeki kelimeler C komutları değildir.

```
1:  #include <stdio.h>  ⟶  Kullanılan işlevler ile ilgili başlık dosyası
2:
3:  void main()
4:  {
5:  printf ("Merhaba Dünya!!!");
6:  return 0;
7:  }
```

"**#include**" komutu C'nin içinde gelen birtakım kütüphanelerin (library) başlık (header) dosyalarını programa ekleyerek, bu kütüphanelerde yer alan bilgileri kullanmamızı sağlar. Bu dosyalar **.h** uzantısına sahiptir, "**stdio.h**", "**conio.h**", "**math.h**" gibi C'nin içerisinde gelen birtakım hazır kütüphane başlık dosyaları mevcuttur. Bunun yanı sıra, C programcısı kendi kütüphanesini kendisi de yaratabilir, ya da büyük projelerde programın uzunluğundan dolayı tanımlamalar için başlık dosyalarını kullanmayı tercih edebilir. Dosya isminin iki yanına " < " ve " > " işaretleri koyarsak derleyici yalnızca kendi "**include**" klasöründeki başlık dosyalarım arayacaktır. Eğer [< >] işaretleri yerine [" "] tırnak işareti kullanacak olursak, derleyici öncelikle o anda bulunduğumuz projemize ait klasörün içinde belirttiğimiz dosyayı arayacak, bulamazsa o zaman kendi "**include**" klasörüne bakacaktır. Kısaca "....." şeklindeki kullanım kendi yarattığımız başlık dosyalan içindir.

Örnek:

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

içinde, programın tamamlandığı ve işinin bittiği noktada, **return()** komutu kullanılarak işletim sistemine geri dönülmesi gerektiği belirtilmelidir.

Bazı Noktalama ve Yazım Kuralları

Noktalı Virgül " ; "

C dilinde her komutun bittiği yer noktalı virgül " ; " işareti ile belirtilir. Bağımsız komutlar noktalı virgül kullanılarak birbirinden ayrılır, böylece bir satıra birden fazla komutun yazılması mümkün olur. Örneğin,

```
x = y;
y = y + 1;
y = x + y;
```

burada üç farklı satırda yazdığımız üç farklı komutu tek bir satırda

```
x = y; y = y + 1; y = x + y;
```

şeklinde de yazabiliriz. Ancak bu yazım kodun kolaylıkla anlaşılmasında zorluklara sebep olacaktır.

Aç – Kapa Parantez " { " ve " } " Karakterleri

Bir C programı, birbiri ile ilişkili komutların bir araya getirildiği komut bloklarından meydana gelir. Bir bloku oluşturan komutlar " { " ve " } " parantez işaretleri arasına yerleştirilir. Böylece belirli bir amaca yönelik komutların bir arada tutulması ve bloğun anlaşılmasını ve kontrol edilmesini kolaylaştırır.

Açıklama Satırları

Programın içine gelişi güzel istediğimizi yazarsak, bu programın çalışmasını engelleyecektir. Derleyici yazılan açıklama ve yorumlan da komut olarak algılamaya çalışacaktır. Programa açıklama satırı yazmanın iki yolu vardır.

1) İki tane bölü (slash) karakteri " // " kullandığımız takdirde bunların sağ tarafında satır sonuna kadar kalan alan derleyici tarafından atlanır. Bu durum yalnızca bulunduğu satır için geçerlidir.

2) Bölü karakterinden sonra çarpı (asterisk - yıldız) karakteri kullanırsak " /* " derleyici çarpıdan sonra bölü karakteri " */ " görünceye kadar bloğun tamamını atlar. Bu durum sadece tek satır için değil, bütün program alanı için geçerlidir.

Program kodunun içine, kod ile ilgili açıklama yazmak istediğimiz zaman iki farklı şekilde yapmak mümkündür. İlki tek satırlık açıklama şeklinde çift bölü karakteriyle yapılır. İkincisi blok halinde yazım şeklidir ve bu blok " /* " işaretiyle başlar, " */ " işaretiyle biter. Bu iki kullanım şeklini aşağıdaki örnekte gösterilmiştir.

```
#include <stdio.h>

void main()

{

/* Bu program bilgisayarınızın ekranına Merhaba Dünya kelimelerini yazan bir C programıdır. */

printf("Merhaba Dunya!!!"); // Merhaba Dünya yazılır.

return 0;

}
```

Derleyicinin karşısına " /* " karaterleri çıkana kadar program derlenir. Karşılaşılan ilk " /* " karakter çiftine kadar bütün yazılanlar atlanır ve derleme devam eder. Aynı şekilde, programda karşılaşılan " // " karakterlerinin sol tarafında kalan kısımlar derlenip sağ taraflarındakiler derlemeye dahil edilmez.

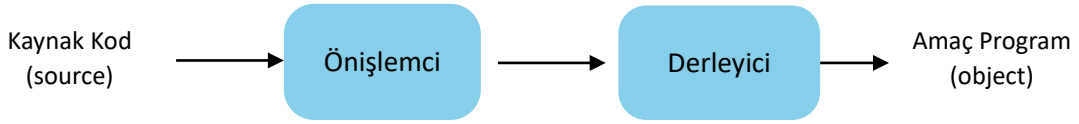
Program Kodunda Girintiler ve Boş Satırlar

C dilinde program yazarken, kodun daha düzenli ve okunabilir olmasını sağlamak amacıyla bazı komutlar içeriye doğru girintili olarak yazılabilir. Aslında, C dilinde komutların fiziksel olarak satırın neresine yazıldığı bilgisayar açısından fark etmez; çünkü bilgisayar tüm komutları aynı şekilde işler. Ancak, programlar büyüyüp yüzlerce hatta binlerce satırdan oluşmaya başladıkça, bu kodların okunması ve anlaşılması zorlaşır. Bu nedenle, kodun yapısal düzenini korumak ve programı daha anlaşılır hale getirmek için belirli bir düzenleme yapmak önemlidir.

C dilinde, bir blok içinde yer alan tüm komutlar ve açıklama satırları, bu bloğun başlangıcını ve sonunu belirten " { " ve " } " sembollerine göre daha içeride yazılır. Bu girintiler, kodun yapısını görsel olarak belirginleştirir ve hangi komutların hangi bloklara ait olduğunu daha kolay anlamanızı sağlar. Ayrıca, kodun daha düzenli görünmesi ve okunabilirliğinin artması için bloklar arasında boş satırlar kullanmak da yaygındır. Boş satırlar, farklı kod bölümlerini birbirinden ayırarak programın daha anlaşılır olmasına katkıda bulunur. Bu basit düzenleme teknikleri, karmaşık programların bile kolayca okunabilir ve sürdürülebilir olmasını sağlar.

Diyez (Kare) " # " Karakteri

Önişlemci (preprocessor) kaynak kodu üzerinde birtakım düzenlemeler ve değişiklikler yapan bir ön programdır. Bu önişlemci, programın derlenme sürecinin ilk aşamasında devreye girer ve **kaynak kodu** alarak belirli işlemler gerçekleştirir. Önişlemcinin girdisi, doğrudan yazdığınız kaynak programdır; yani ***.c** uzantılı dosyanızdır. Önişlemci, bu kaynak kodu üzerinde çalışarak belirli komutları işler, makroları genişletir, dosyaları birleştirir ve bazı kod bölümlerini kaldırır veya ekler. Bu işlemler sonucunda ortaya çıkan çıktıya, önişlemci tarafından *işlenmiş kod* diyebiliriz. Bu çıktı, derleyiciye girdi olarak verilir ve bir sonraki adımda işleme alınır.



Kaynak Kod'un Amaç Program'a dönüştürülmesi

Önişlemci tarafından işlenmiş olan bu kod, derleme modülü tarafından ele alınarak makine diline çevrilecek olan **amaç koda** (*object code*) dönüştürülür. Önişlemci, derleyicinin işini kolaylaştırmak ve kodun daha verimli bir şekilde işlenmesini sağlamak amacıyla çalışır. Bu aşamada yapılan düzenlemeler ve değişiklikler, programın derlenme sürecini etkileyerek nihai çalıştırılabilir dosyanın oluşmasında önemli bir rol oynar. Önişlemci, kaynak kodu üzerinde yapılan bu ön işlemler sayesinde, daha temiz ve derleyici için hazır bir kod oluşturur.

C programlama dilinde " # " ile başlayan bütün satırlar önişlemci komutları (*preprocessor directives*) olarak kabul edilir ve bunlar önişlemciye ne yapması gerektiğini anlatır. Örneğin:

#include	**#define**	**#if**
#else	**#elif**	**#ifdef**
#ifndef	**#endif**	**#undef**
#line	**#error**	**#pragma**

komutları birer önişlemci komutudur.

Önişlemci komutları derleyici açısından birer C komutu değildir. Bunlar " # " karakteriyle birlikte anlam kazanırlar. İstenirse örneğin **include** isimli bir değişken tanımlanabilir ancak bunun programın okunabilirliği açısından sakıncası vardır ve genellikle tanımlanmaması önerilir.

Önişlemci amaç kod oluşturmaya yönelik hiçbir iş yapmaz. O sadece kaynak programını " # " içeren satırlardan arındırır. Derleme modülüne girecek programda artık " # " içeren satırlar yer almayacaktır.

Başlık Dosyaları

Her standart kütüphanede, burada bulunan fonksiyonlara ait fonksiyon prototipleri içeren başlık dosyaları (*header file*) bulunur. Bu dosyalar, programların belirli fonksiyonları, veri türlerini, makroları ve değişkenleri diğer dosyalarla paylaşmasına olanak tanıyan dosyalardır. Genellikle " **.h** " uzantısı ile tanınır ve programın farklı bölümlerinde kullanılacak olan fonksiyonların, değişkenlerin ve veri yapılarının deklarasyonlarını içerir. Bu dosyalar, kaynak kod dosyalarına ("**.c**" dosyalarına) dahil edilerek, bu fonksiyonlar ve veri türleri farklı dosyalar arasında paylaşılabilir hale gelir.

Başlık dosyalarının kullanımı, programların modülerliğini ve bakımını kolaylaştırır. Bir projede, fonksiyon tanımları ve veri yapılarını bir başlık dosyasında toplamak, bu tanımlamaların birden fazla dosyada tekrar edilmesini önler ve kodun yeniden kullanılabilirliğini artırır. Aynı zamanda, başlık dosyaları, bir projenin derlenme sürecinde önemli bir rol oynar. Derleyici, bir kaynak dosyada kullanılan ancak başka bir dosyada tanımlanmış olan fonksiyonları, başlık dosyasında belirtilen deklarasyonlar sayesinde tanır ve programın düzgün bir şekilde çalışmasını sağlar. Başlık dosyalarının doğru ve etkin kullanımı, büyük projelerde kodun daha anlaşılır, bakımı kolay ve hata olasılığını azaltan bir yapıda olmasına yardımcı olur.

C programlarına dâhil edebilecek bazı çok kullanılan C standart kitaplık başlık dosyaları aşağıdaki tabloda listelenmiştir. Fonksiyonlar bölümünde fonksiyon prototipleri daha ayrıntılı anlatılacaktır.

C programlarına dâhil edebilecek bazı çok kullanılan C standart kitaplık başlık dosyaları

AÇIKLAMA	BAŞLIK DOSYA ADI
Hata ayıklama ile ilgili makroları İçerir	**<assert.h>**
Karakterleri test etmeyle ilgili fonksiyonlar	**<ctype.h>**
Kesirli sayı değerleri büyüklük limitlerini içerir	**<float.h>**
İntegral büyüklük limitlerini içerir	**<limits.h>**
Matematik fonksiyonları içerir	**<math.h>**
Standart giriş/çıkış kitaplık fonksiyonları ve bunlar tarafından kullanılan bilgileri içerir	**<stdio.h>**
Sayıları metine, metni sayılara dönüştürme, rastgele sayı üretme, bellek tahsis etmeyle ilgili fonksiyonları içerir	**<stdlib.h>**
Karakter işleme fonksiyonları içerir	**<string.h>**
Zaman ve tarih ile ilgili fonksiyonları içerir	**<time.h>**

C Dilindeki Sözcükler

C dilinde, (A-Z, a-z, 0-9 ve _) gibi karakterler bazı sistematik kurallar çerçevesinde bir araya gelerek, sözcükleri oluştururlar. C dilindeki sözcükler **anahtar sözcükler (keywords)** ve **tanıtıcılar/isimler (identifier)** olarak ikiye ayrılır.

Anahtar sözcükler (keywords): C dilindeki anahtar (*özel amaçlı*) sözcükler derleyici tarafından kullanılan özel kelimelerdir. Bunlara programlamada kullanılan haliyle komut denir. Bunlar program içinde doğru yerde ve doğru şekilde kullanılmak zorundadır. Aşağıdaki tabloda bu 32 özel amaçlı sözcükler yani komutlar, alfabetik sıraya göre verilmiştir.

C dilindeki anahtar sözcükler

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

Örnek bir kod:

```
#include <stdio.h>   // standart input/output
#include <conio.h>  // console input/output
int main(void)
{
int a=4;
if(a%2 == 0)
printf("a değişkeni ciftir.");
else printf("a değişkeni tektir.");
getch();
return(0);
```

Tanıtıcılar/isimler (identifier): Bir program yazılırken anahtar sözcüklerin dışında, değişken, fonksiyon isimleri ve farklı parametre isimleri şeklinde birçok tanımlamanın yapılması gerekmektedir. Bu tanımlamalar *tanıtıcılar/isimler* olarak adlandırılır. Örneğin **printf()** fonksiyonu ve diğer standart fonksiyon isimleri, kütüphaneler içinde yer alan tanıtıcılardır. C dilinde kullanılan tanıtıcılara ait isimlerin geçerli olabilmesi için aşağıdaki kurallara uygun olarak oluşturulması gereklidir.

- Tanıtıcı içinde harf (a...z, A....Z) , sayı (0....9) veya alt çizgi (_) bulunabilir.
- Bir tanıtıcı, bir harf ya da alt çizgi " _ " işareti ile başlamalıdır.
- Tanıtıcı içinde #, $, &, ö, ş gibi özel karakterler bulunamaz.
- Tanıtıcı C dilinde kullanılan anahtar sözcüklerden biri olamaz.
- Tanıtıcıların uzunluğu teorik olarak sınırsızdır, ancak pratikte daha kısa ve anlamlı isimler tercih edilmelidir.
- C dili **büyük-küçük harfe duyarlı** bir dildir.

Geçerli İsimler:

ogrenci_yasi

_06_yili_ucreti

Short : geçerli olsa da kullanımı tavsiye edilmez, anahtar sözcüklerdendir

Geçersiz İsimler:

06_yili_ucreti : Tanıtıcı ismi bir sayı ile başlamaz.

short : Tanıtıcı ismi anahtar bir sözcük olamaz.

Öğrenci yasi : Tanıtıcı ismi boşluk içermez.

Meriç : Tanıtıcı ismi Türkçe karakter içermez.

Ayse?Veli : Tanıtıcı ismi özel bir karakter içermez.

Tanıtıcılar, programcılar tarafından oluşturulur ve programdaki çeşitli öğelere isim vermek için kullanılır. Tanıtıcılar, bir programın okunabilirliğini ve organize edilmesini sağlar çünkü bunlar aracılığıyla programın farklı parçaları adlandırılır ve birbirinden ayırt edilir. Tanıtıcılar, programın yapısının ve mantığının anlaşılabilir olmasını sağlar. İyi seçilmiş tanıtıcılar, kodun okunabilirliğini artırır ve bakımını kolaylaştırır. Örneğin, "**sayi**" veya "**yas**" gibi anlamlı isimler, "**x**" veya "**y**" gibi rastgele harflerden daha anlaşılırdır. Ayrıca, programcılar arasında bir anlaşma (naming convention - adlandırma kuralı) oluşturulması, büyük projelerde tanıtıcıların düzenli ve tutarlı olmasını sağlar.

Değer Sabitleri (Constants)

C dilinde kullanılan diğer bir eleman **değer sabitleri**'dir. Bir program boyunca değerleri değişmeyen sabit değerlerdir. Sabitler, bir değişkene atandıklarında veya doğrudan kullanıldıklarında, bu değerin program çalıştığı sürece sabit kalacağını garanti ederler. Sabitler, kodun daha anlaşılır ve güvenli olmasını sağlar, çünkü yanlışlıkla değiştirilmesi gereken değerlerin sabitlenmesine yardımcı olurlar. C dilinde tamsayı, reel sayı, karakter ve dizi sabitleri gibi birçok farklı değer sabiti türü bulunmaktadır.

Tamsayılar

Tamsayı sabitleri 0-9 rakamlarından oluşan ve ondalık değerleri olmayan sayılardır ve ondalık (decimal), sekizlik (octal) veya onaltılık (hexadecimal) sayı sistemlerinde ifade edilebilirler. Tamsayı sabitleri sıfır ile başlayamaz. Bu sabitler pozitif veya negatif olarak tanımlanabilirler. Bu durumda sayının başına " + " ya da " - " işareti konulur. İşaret konulmaması durumunda tamsayı sabiti pozitif olarak kabul edilir.

Geçerli tamsayı sabitleri : 90 +222 -16

Geçersiz tamsayı sabitleri

090 : Tamsayı sabitleri sıfır ile başlamaz

23,5 : Tamsayı sabitleri özel karakter içermez

76.2 : Tamsayı sabitlerinin ondalık değeri yoktur

Reel/Ondalık Sayılar

Reel/Ondalık sayı sabitleri, tam ve ondalık kısımları olan sabitlerdir. **Kayan nokta sabitleri** (floating point constants) olarak da adlandırılır. Örneğin +5.7 bir reel sayı sabitidir. Buradaki 5 reel sayının tam kısmını, 7 ise ondalık kısmını temsil eder. Reel sayı sabitlerinde, sayının tam ve ondalık kısmı nokta " . " ile birbirinden ayrılır. Reel sayı sabitleri de pozitif ya da negatif olarak tanımlanabilirler. Bu durumda sayının başına " + " ya da " - " işareti konulur. İşaret kullanılmaması sayının pozitif olduğunu gösterir.

C dilinde 10^3 gibi sayı gösterimlerini tanımlamak için bilimsel gösterim şekli kullanılır. Buna göre, 10 üzeri tanımlaması " e " ya da " E " sembolü kullanılarak gerçekleştirilir. Bu tür gösterimlere daha ziyade çok büyük ya da çok küçük sayılarla çalışan bilimsel alanlarda (Astronomi, Atom ve Molekül Fiziği gibi) kullanılan bilgisayar programlarında başvurulur.

Örnek:

0.23898×10^2 : 0.23898e2

238.98×10^{-1} : 238.98e-1

Aşağıdaki örnekte ise aynı sayı farklı biçimlerde gösterilmiştir.

0.72e01 = 0.72e+01 = 0.72e1 = 7.2e-1

Karakterler

C dilinde tek tırnak işaretleri (' ve ') arasında bulunan tek karaktere, **karakter sabiti** adı verilir. Burada karakter olarak tanımlanabilecek olan değerler bir ASCII kod tablosunda verilmiş olan gösterimlerden biri olabilir. Bu sabitler, karakterlerin ASCII değerlerini saklar ve program içinde bu karakterlerle işlem yapmayı sağlar. Kaçış dizileri (*escape sequences*), özel karakterlerin temsil edilmesi için kullanılır ve bir ters eğik çizgi (' \ ') ile başlar. Programlama sırasında sıkça ihtiyaç duyulan bu karakterlerin kodlama içinde düzgün bir şekilde kullanılmasına yardımcı olur.

Geçerli karakter sabitleri

'A', 'z', '3', '%'

'\n' (Yeni satır) '\t' (Sekme -tab) '\r' (Satır başı -carriage return)

Geçersiz karakter sabitleri

'abc' : Karakter sabiti sadece tek bir karakter değeri için tanımlanabilir

“3” : Karakter sabiti tek tırnak işaretleri ile tanımlanabilir.

C : Karakter sabitlerinin tek tırnak işaretleri ile tanımlanması gerekir.

Dizgeler (Karakter Grubu)

Bu sabitler çift tırnak işareti (") ile tanımlanırlar ve birden fazla karakterin bir araya gelmesinden oluşurlar. Geçerli dizgi sabitleri örnekleri aşağıda belirtilmiştir:

"dd4ff", "5", "C dili\n", ""

"Merhaba", "C dilini öğreniyorum!"

Bir dizge sabiti birden çok karakter içerebileceği gibi, tek bir karakterden de oluşabilir. Bir dizge sabitinin içi **boş (NULL)** olabileceği gibi, boşluk karakterini ' [] ' de içerebilir. Ancak dizge sabitinin kendisi birden fazla satıra yayılmamalıdır ve tek bir satırda tanımlanmalıdır.

Aşağıda geçersiz dizgi sabitleri verilmiştir.

'İstanbul' : Dizgi sabitleri çift tırnak ile tanımlanmalıdır.

İstanbul : Dizgi sabitleri çift tırnak ile tanımlanmalıdır.

"örnek" : Dizgi sabitleri **ASCII** tablosunda yer almayan karakterleri içeremez (burada "ö").

İsim/Tanımlı Sabitler

Programın bir parçası, basamak sayısı fazla olan bir değer veya bir bağıntı programın birçok yerinde kullanılacaksa buna bir isim vererek program içinde bu ismi kullanırız. Örneğin trigonometrik fonksiyonları içeren bir program yazılacaksa **π** sayısı birçok kez kullanılacaktır. Her seferinde 3.1415926535 yazmak hem uğraştırıcıdır hem de programı daha karışık bir hale sokar. Bu zorluğu aşmak için şu şekilde bir tanımlama yapabiliriz:

```
#define Pl 3.1415926535
```

Artık her **π** kullanacağımız yerde PI yazmamız yeterli olacaktır. Burada akla şu tür bir soru gelebilir. #define PI 3.1415926535 şeklinde **π**'yi tanımlamak yerine programın başında PI = 3.1415926535 yazsak ne fark eder? Programın çalışması açısından çok büyük bir fark olmasa da (hız farkı hariç, değişken kullanmak çok az da olsa yavaşlatır) **#define** komutunun görevi değişkenlerden çok farklıdır. Başta bir eşitlikle tanımladığımız PI'ye daha sonra farklı değerler vermemiz mümkündür, ancak **#define** ile tanımladığımız PI her zaman sabittir. Değişkenlerin tipleri vardır (tam sayı, ondalıklı sayı, karakter, dizi v.s.). Buna karşılık **#define**'da bu tür bir 'tip' yoktur, **#define**'ın yaptığı tek şey yanında tanım olarak verilen ismi program boyunca arar ve bulduğu yerlerde yerine tanımın yanındaki kalıbı koyar. Bu kalıbın bir tipi yoktur, yani harf, sayı, sembol ne olduğu fark etmez. Gerekli tanımlamaların değerleri yerlerine koyulduktan sonra program çalıştırılır.

#define'ın kullanım biçimine bir örnek olarak aşağıdaki isim sabitinin tanımlandığını kabul edelim:

```
#define ORAN 8
```

Buna göre aşağıdaki program parçasını inceleyelim.

```
maas = saat * 30 + ORAN * 10;
pirim = maas * ORAN;
```

Bu komutlardaki tüm ORAN isimleri önişlemci tarafından **#define** komutunda belirtilen 8 değeriyle değiştirilecektir ve program aşağıdaki hali alacaktır.

```
maas = saat * 30 + 8 * 10;
pirim = maas * 8;
```

Aşağıda verilen atama komutu, bir tanımlı sabit **değiştirilemeyeceği** için hatalıdır.

```
ORAN = 10;                    /* yanlış kullanım */
```

Değişkenler (Variables)

Değişkenler, programda kullanılan verileri depolamak için kullanılan adlandırılmış alanlardır. Bir değişken, belirli bir veri tipine (örneğin, '**int**' tam sayı, '**float**' ondalıklı sayı, '**char**' karakter gibi) sahip olur ve bu veri tipine uygun bir değeri saklar. Değişkenler, programın farklı noktalarında kullanılabilecek ve manipüle edilebilecek değerleri saklar. C dilinde bir değişken tanımlandığında, bu değişken için bellekte belirli bir alan ayrılır ve bu alan, değişkenin tipine bağlı olarak belirli bir boyutta olur.

int yas;	Tamsayı tipinde bir değişken tanımlandı.
float maas;	Ondalıklı sayı tipinde bir değişken tanımlandı.
char harf;	Karakter tipinde bir değişken tanımlandı.
int ogrenci_sayisi = 30;	Tamsayı tipinde bir değişken tanımlandı ve başlangıç değeri 30 olarak atandı.

Yukarıdaki örneklerde, **'yas'**, **'maas'**, **'harf'**, ve **'ogrenci_sayisi'** adında değişkenler tanımlanmıştır. **'ogrenci_sayisi'** değişkenine bir başlangıç değeri atanmıştır, diğer değişkenler ise başlangıç değeri atanmamıştır ve bu nedenle belirsiz bir değere sahip olabilirler. Değişkenler program içinde kullanım amaçlarına göre farklı isimler alabilmektedir. Bu durumda değişkenler **'yerel'**, **'global'** ve **'statik'** değişkenler olarak adlandırabilmektedir.

Veri Tipleri (Data Types)

Veri tipleri, değişkenlerin saklayabileceği veri türünü ve bellekte kaplayacağı alanı belirtir. C dilinde en yaygın kullanılan veri tipleri şunlardır:

int	: Tamsayı değerleri saklamak için kullanılır. Genellikle 2 veya 4 bayt yer kaplar.
float	: Ondalıklı (kayan noktalı) sayı değerleri saklamak için kullanılır. 4 bayt yer kaplar.
char	: Tek bir karakteri saklar. 1 bayt yer kaplar.
double	: **'float'** tipine göre daha yüksek hassasiyetli ondalıklı sayılar saklar. 8 bayt yer kaplar.

Bazı durumlarda **'void'** veri tipi olarak kullanılmaktadır. Ancak bu bir fonksiyonun bir değer döndürmediği belirtmek için kullanılır. Her veri tipi, bellek üzerinde belirli bir miktarda yer kaplar ve farklı türde verilerin işlenmesine olanak tanır. Doğru veri tipini seçmek, bellek kullanımını optimize etmek ve programın daha verimli çalışmasını sağlamak açısından önemlidir.

Operatörler (Operators)

Operatörler, değişkenler, sabitler ve diğer veri yapıları üzerinde işlemler gerçekleştirmek için kullanılan sembollerdir. Operatörler, aritmetik işlemlerden mantıksal karşılaştırmalara, bit düzeyinde işlemlerden atama işlemlerine kadar geniş bir yelpazede kullanılır.

C Dilinde Temel Operatörler:

1. Aritmetik Operatörler : ' + ', ' - ', ' * ', ' / ', ' % '
2. Atama Operatörleri : ' = ' (Atama), ' += ', ' -= ', ' *= ', ' /= ', ' %= '
3. Karşılaştırma Operatörleri : ' == ', ' != ', ' > ', ' < ', ' >= ', ' <= '
4. Mantıksal Operatörler : ' && ' (ve), ' || ' (veya), ' ! ' (değil)
5. Artırma ve Azaltma Operatörleri : ' ++ ' (Artırma), ' -- ' (Azaltma)
6. Bit Düzeyinde Operatörler : ' & ' (ve), ' | ' (veya), ' ^ ' (özel ve), ' ~ ' (inverse)
7. Koşul Operatörü (Ternary Operator) : ' ? / : ' (Satır içi koşul operatörü)
8. Virgül Operatörü : ' , ' (Virgül, birden fazla değişkeni ifade etmek)

Örnekler:

int a = 1, b = 2, c = 3;	Virgül operatörü çoklu değer ve veri tipi belirleme.
toplam = 10 + 20;	Aritmetik operatör kullanarak iki sayıyı toplar.
a += 5;	Atama operatörü a = a + 5 ile aynıdır.
if (a > 0) && (b < 10)	Mantıksal operatör iki değerin doğruluğunu kontrol eder.
a++ veya ++a	Artırma/Azaltma operatörü değişkenin değerini bir arttırır.
int x = (a > b) ? a : b;	Satır içi koşul operatörü x değeri koşula göre değer alır.
a & b, a | b, ~a	İki sayıyı bit düzeyinde 0, 1 değerlerini karşılaştırır.

a = 12 (01100), b = 25 (11001)

a & b = 0 & 1 = 0
1 & 1 = 1
1 & 0 = 0
0 & 0 = 0
0 & 1 = 0

} **8** (Onluk değer)

Kontrol Yapıları (Control Structures)

Kontrol yapıları, bir programın akışını kontrol etmek ve belirli koşullara veya durumlara göre farklı işlemler gerçekleştirmek için kullanılır. Bu yapılar, programın belirli bir kısmının ne zaman çalışacağını veya tekrar edeceğini belirlemeye yardımcı olur. C dilindeki temel kontrol yapıları şunlardır:

Koşullu İfadeler (Conditional Statements)

"if" ve "else" Yapıları:

"**if**" yapısı, belirli bir koşul doğru olduğunda bir kod bloğunun çalıştırılmasını sağlar.

"**else**" yapısı, "**if**" koşulu doğru değilse alternatif bir kod bloğunun çalıştırılmasını sağlar.

"**else if**" yapısı, birden fazla koşulun sıralı olarak kontrol edilmesine olanak tanır.

Örnek:

```
int x = 10;
if (x > 0) {
   printf("x pozitiftir.");
} else if (x == 0) {
   printf("x sıfırdır.");
} else {
   printf("x negatiftir.");
}
```

"switch" Yapısı: Bir değişkenin belirli bir değere sahip olup olmadığını kontrol etmek ve bu değere göre farklı kod bloklarını çalıştırmak için kullanılır.

Örnek:

```
int num = 2;
switch (num) {
   case 1:
      printf("Bir"); break;
   case 2:
      printf("İki"); break;
   case 3:
      printf("Üç"); break;
   default:
      printf("Bilinmeyen sayı");
}
```

Döngüler (Loops)

"for" Döngüsü: Belirli bir sayıda tekrarlanması gereken işlemler için kullanılır. Başlangıç, koşul ve artırma/değiştirme adımlarını içerir.

"while" Döngüsü: Koşul doğru olduğu sürece belirli bir kod bloğunu tekrarlar. Koşul, döngüye girilmeden önce kontrol edilir.

"do-while" Döngüsü: "while" döngüsüne benzer, ancak koşul kontrol edilmeden önce kod bloğu en az bir kez çalıştırılır.

Örnekler:

```
for (int i = 0; i < 5; i++) {
    printf("i = %d\n", i);
}
```

```
int i = 0;
while (i < 5) {
    printf("i = %d\n", i);
    i++;
}
```

```
int i = 0;
do {
    printf("i = %d\n", i);
    i++;
} while (i < 5);
```

Kontrol Deyimleri (Control Statements)

"break": Kullanıldığı yerden itibaren döngüyü veya "switch" yapısını hemen sonlandırır.

"continue": Döngüde sağlanan şartta hiçbir işlem yapmadan bir sonraki adıma geçer.

"return": Bir fonksiyonun çalışmasını sonlandırır ve isteğe bağlı olarak bir değer döndürür.

Örnekler:

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;
    }
    printf("i = %d\n", i);
}
```

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        continue;
    }
    printf("i = %d\n", i);
}
```

```
int sum(int a, int b) {
    return a + b;
}
```

Fonksiyonlar (Functions)

Fonksiyonlar, belirli bir görevi yerine getirmek için yazılmış, bağımsız, yeniden kullanılabilir kod bloklarıdır. Fonksiyonlar, programların daha modüler, okunabilir ve bakımı kolay olmasını sağlar. Ayrıca, karmaşık işlemleri daha küçük, yönetilebilir parçalara ayırarak programcıların işini kolaylaştırır. Bir C fonksiyonunun temel yapısı şu şekildedir:

```
geri_dönüş_tipi fonksiyon_adı(parametre_listesi) {
   // Fonksiyon gövdesi
   // İşlemler
   return geri_dönüş_değeri;
}
```

Geri Dönüş Tipi (Return Type): gerçekleştirdiği işlemlerden sonra döndüreceği (“int”, “float”, “char”, “void”) veri tipini belirtir. Örneğin, bir fonksiyon bir tamsayı döndürecekse “int”, bir değer döndürmeyecekse “void” kullanılır.

Fonksiyon Adı (Function Name): Fonksiyonun ismini belirtir. Programcı tarafından belirlenir ve fonksiyonun ne iş yaptığını yansıtacak şekilde (“topla”, “hesapla”, “yazdir” gibi.) anlamlı bir isim verilmesi tavsiye edilir.

Parametre Listesi (Parameter List): Fonksiyonun işleyebilmesi için gereken girdilerin (parametrelerin) veri tiplerini ve isimlerini içerir. Parametreler, fonksiyonun içinde kullanılmak üzere geçici değişkenler (“int a”, “int b” gibi) olarak tanımlanır. Eğer parametre olmayacaksa parantezler boş bırakılır: “**void fonksiyon_adı(void)**”.

Fonksiyon Gövdesi (Function Body): Fonksiyonun gerçekleştireceği işlemleri içeren kod bloğudur. Bu blok içinde hesaplamalar, diğer fonksiyon çağrıları, kontrol yapıları ve döngüler olabilir.

Return Deyimi: Fonksiyonun işlemi tamamladıktan sonra geri döndüreceği değeri belirtir. Eğer fonksiyon bir değer döndürmüyorsa (“**void**”), “return” ifadesi kullanılmaz veya boş “return” kullanılabilir.

iki sayıyı toplayan basit bir C fonksiyonu örneği

```
#include <stdio.h>
int topla(int a, int b) {
   return a + b
}
void main() {
   int x = 5, y = 10;
   int sonuc = topla(x, y);
   printf("Toplam: %d\n", sonuc);
}
```

Diziler (Arrays)

Diziler, aynı türden birden fazla değeri tek bir değişken altında saklamaya olanak tanıyan veri yapılarıdır. Bir dizi, birden fazla elemanı tek bir isim altında depolar ve bu elemanlara indeksleme yoluyla erişilir. Dizi tanımlama şu şekilde yapılmaktadır:

veri_tipi dizi_adı[dizi_boyutu] = {virgülle ayrılmış eleman listesi};

Veri tipi dizide kullanılacak değerlere (“int”, “float”, “char”, “double”) uygun olarak seçilir. Dizi elemanlarına sıfırdan başlayan indekslerle erişilir. İlk elemanın indeksi **“0”**, son elemanın indeksi ise “**boyut-1**” şeklindedir.

```
int sayilar[5] = {1, 2, 3, 4, 5};                  // 5 elemanlı bir tam sayı dizisi
char isim[] = "Ahmet";                             // karakter dizisi toplu atama
char isim[6] = {'A', 'h', 'm', 'e', 't', '\0'};    // karakter dizi tek tek tanımlama
printf("3. Eleman = %d", sayilar[2]);              // sayilar dizisinin 3. Elemanı yazılır
```

İşaretçiler (Pointers)

İşaretçiler, bellekte bir adresi tutan değişkenlerdir. Yani bir işaretçi, başka bir değişkenin veya bellekteki bir veri bloğunun adresini saklar. İşaretçiler, C dilinin güçlü özelliklerinden biridir ve bellek yönetimi, dinamik veri yapıları, diziler, fonksiyonlar gibi birçok alanda etkin bir şekilde kullanılır. Bir işaretçi değişken isminin başına yıldız sembolü “ * ” konulur. Aşağıda bir işaretçinin nasıl tanımlandığı ve kullanıldığına dair basit bir örnek verilmiştir:

```
#include <stdio.h>
int main() {
   int x = 10;
   int *p = &x;  // x'in adresini p işaretçisine atar

   printf("x'in değeri: %d\n", x);
   printf("p'nin gösterdiği değer: %d\n", *p);  // p'nin gösterdiği adresteki değeri (yani x'in
                                                   değerini) yazdırır
   *p = 20;  // p'nin gösterdiği adresteki değeri değiştirir, bu da x'in değerini değiştirir
   printf("x'in yeni değeri: %d\n", x);
   return 0;
}
```