C PROGRAMLAMA DİLİ İLE YAPISAL PROGRAMLAMA

Elektronik Yüksek Mühendisi İLHAN ÖZKAN
Ankara, Ocak 2025

ÖNSÖZ

Ülkemizde her alanda olduğu gibi yazılım alanında da bir terminoloji karmaşasının yaşandığı yazılan kitaplardan anlaşılmaktadır. Son dönemde yazılan kitaplar, adeta çeviri yapan programların bir gecede ürettiği çıktılar birleştirilerek hazırlanmaktadır. Bu kitaplar incelendiğinde, bilindiği varsayılan kavramların bile tanımlanırken birbirinin içerisine girdiği görülmektedir. 1989 yılından beri birçok programlama dili kullanarak yazılım geliştiriyor olmama rağmen, bu kitapların içeriğini anlama konusunda oldukça zorlanıyorum.

Yapısal programlamanın en çok uygulandığı Pascal dilidir. Pascal diline yakın zamanda ortaya çıkan C dili, kısa sürede sistem dili olarak tarihte yerini almıştır. Günümüzde sistem programlama ve yüksek hız gerektiren gömülü sistemlerde vazgeçilmez bir dil olmuştur.

Daha sonra C dilinden etkilenecek olan C++, Java, C# gibi birçok nesne yönelimli programlama dili ve yapısal programlama üzerine kurulmuş dillerdir. Birçok modern programlama diline esin kaynağı olan bu dilin özellikle bilgisayar mühendisleri tarafından bilinmesi gerekir.

Kitapta verilen konular çok sade olarak hazırlanmıştır. İşlenen kavramlar her bölümde üst üste konularak ilerlediğinden bir bölüm özümsenmeden bir sonrakine geçilmemesi önerilir.

Yapısal programlama mantığını oluşturan kavramların programcıların kafasında aynı ışığı yakması için bu kitap hazırlanmıştır. Bu dilin eğitimini verecek tüm kişiler için serbestçe kullanılabilir.

Bu kitabın güncel sürümüne https://github.com/HoydaBre/clanguage adresinden erişilebilir. Amacım Türkçe düşünen e konuşan öğrencilerimize kavramları doğru bir şekilde aktarmaktır. Katkılarınızı bekler iyi çalışmalar dilerim.

İlhan ÖZKAN

ilhanozkan[at]outlook.com

İçindekiler

TEMEL BİLGİSAYAR KAVRAMLARI

Mühendis ve Teknisyen

Mühendis (engineer), ihtiyaç duyulan ürünü; ekonomik (cost), güvenli (safe) ve kullanışlı (functional) olarak zamanında (on time) ortaya koyan kişilerdir. Bunu yapmak için; keşif (invent), tasarım (design), analiz (analysis), uygulama (build) ve sınama (test) yaparlar[1].

Teknisyenler (technician) ise genellikle laboratuvarda teknik ekipmanlarla ilgilenen veya bu ekipmanlar ile pratik çalışmalar yapan kişilerdir. Genel olarak amacı ekipmanın bakım ve kullanımına ilişkindir.

Konumuz yazılım olduğundan, yazılım mühendisleri ise istenilen bilgisayar yazılımını ekonomik, güvenli ve kullanışlı olarak zamanında ortaya koyan kişilerdir. Programcılar ise teknisyenlere karşılık gelir. Yani geliştirme aşamasında bazı yazılım kısımlarının kodlanmasını veya geliştirilen yazılımın çalışır tutulmasını sağlarlar.

Bilgisayara Niçin İhtiyaç Duyulur?

Bir çiftçi niçin traktöre ihtiyaç duyar? Bir berber niçin elektrikli tıraş makinesi kullanır? Benzer şekilde bir mühendis niçin bilgisayara ihtiyaç duyar?

İşte bütün bu soruların cevabı yapılacak işleri daha kısa sürede yapmaktır. Yani bütün araç gereç ve makinalara olan ihtiyacımız işlerimizi daha hızlı (speed) yapabilmek içindir. Bütün makineler gibi bilgisayar da veriden hızlıca bilgi elde etmemizi ve tekrarlanan hesaplamaları hatasız yapmamızı hızlıca sağlayan makinelerdir.

Veri sorumlusu, veri hazırlama kontrol işletmeni gibi unvanlar bu sebeple ortaya çıkmıştır. Bu unvanlara ihtiyaç duyulmasının sebebi verinin belli bir şekle uygun olarak bilgisayara girilmesi ve işlenen verinin saklanması ve raporlanmasının bilgisayar dünyasında önemli bir yer tutmasıdır.

Bilgisayarlar ilk zamanlarda daha çok cebirsel ifadeleri hatasız ve hızlı bir şekilde yapmak için kullanılmışlardır. Bunu daha sonradan ALGOL (ALGOritmic Language) adını alacak IAL (International Algebraic Language) ya da FORTRAN (FORmula TRANslation) dilinden de anlayabiliriz.

Veri ve Bilgi

Veriler (data), genellikle bir ölçüm ya da sayım sonucu elde edilen ve hakkında az şey bilinen varlıklardır. Örneğin ölçülen hava sıcaklığı, derse giren öğrenci sayısı, tartılan sebzenin ağırlığı gibi şeyler.

Bilgi (information) ise verinin işlenerek otaya çıkan anlamlı verilerdir. Örneğin ortalama hava sıcaklığı, ölçülen yerde bir ölçüm yapmaya gerek kalmadan sıcaklığı yaklaşık olarak bilmeye yarayan bir bilgidir. Bir sınıftaki not ortalaması, o sınıftan rastgele seçilen bir öğrencinin notunu yaklaşık belirleyen bir bilgidir.

Kısacası bilgi, işlenmiş veridir.

En Basit Bilgisayar

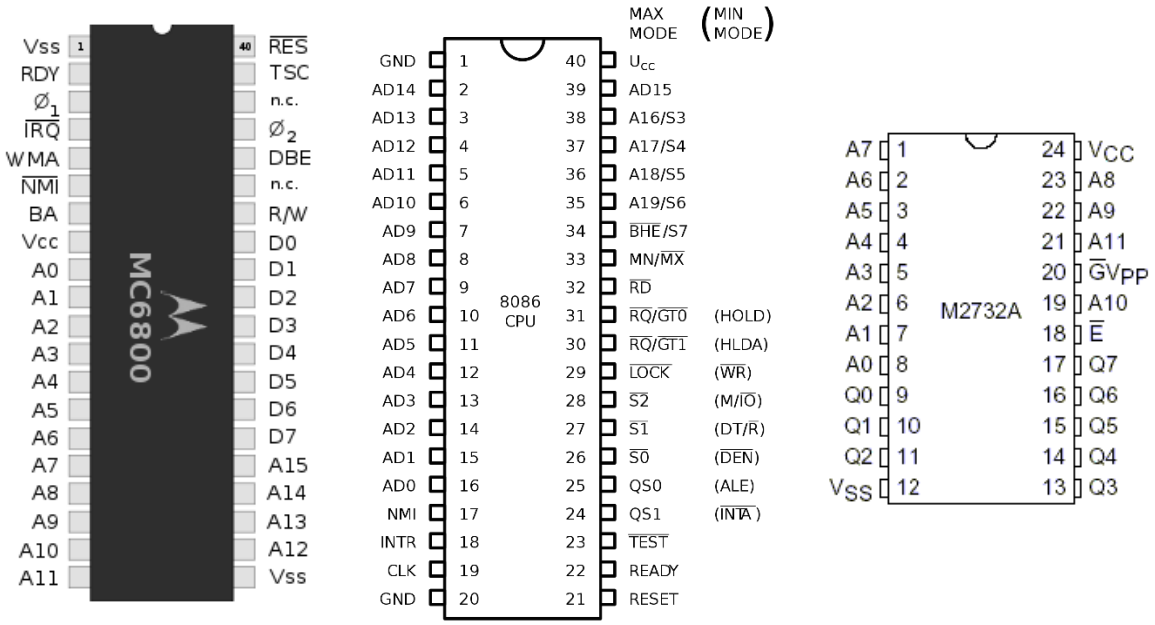
En basit bilgisayar;

- Merkezi işlem birimi (Central Processing Unit-CPU) kısaca işlemci,
- Bellek (memory) ve
- Bunları birbirine bağlayan elektrik yollarından (bus) oluşur.

[1] https://www.bls.gov/

Bellek ise işlemcinin işleyeceği emirleri barındırır. Belleğin hem adres (address) hem de veri (data bus) bacakları bulunur. Benzer şekilde işlemcinin de bellek ile iletişimini sağlayan adres ve veri bacakları bulunur. Belleğin bu bacakları elektrik yolları ile işlemcinin adres ve veri bacaklarına bağlıdır. Bunların yanında hem işlemcinin hem de belleğin okuma (read), yazma (write), kesme (interrupt) veya kristal bağlanan bacakları gibi kontrol (control) bacakları da bulunur.

–



Şekil 1. Motorola 6802, Intel 8086 İşlemciler ile 2732 EPROM Belleği

Şekil 1'de görüldüğü üzere 6800 işlemcisinin 15 adet adres bacağı, 8 adet veri bacağı bulunmaktadır. 8086 işlemcisinin ise 20 adet adres bacağı bulunakta olup bu bacakların 16 adedi aynı zamanda veri bacağı olarak kullanılmaktadır. 2732 belleğin ise 12 adet adres bacağı, Q ile gösterilen 8 adet veri bacağı bulunaktadır.

İşlemci ile bellek arasında bağlantı sağlayan elektrik yolları (bus) adres veya veri taşır. Bu yolların veri taşıyanlarına veri yolu (data bus), adres taşıyanına ise adres yolu (address bus) adı verilir. Veri yolu; hem işlemci tarafından belleğe veri yazmak veya bellekten işlemciye taşınacak veriler için kullanıldığından çift yönlüdür. Adres yolu ise belleğin hangi bölgesindeki veriye ulaşılacağını belirleyen hatlar olup tek yönlüdür.

Bilgisayarlar elektrikle çalışan aletlerdir. İşlemci ile bellek arasında bir hatta elektrik varsa "1" yoksa "0" olarak anlamlandırılır. Bunlar ikili sayı sisteminin rakamlarıdır (BInary DigiT) ve kısaca bit olarak adlandırılır. Dolayısıyla;

– 1 elektrik hattının taşıyacağı veri 0 ya da 1 olabilir.
– 2 elektrik hattının taşıyacağı veri 00, 01, 10 ve 11 olabilir. Yani 2^2 kadar farklı veri taşıyabilir.
– N adet elektrik hattının taşıyacağı veri 2^N farklı alternatifte bilgi olacaktır.

İşlemcinin veri yolunun (data bus) genişliği kelime uzunluğudur ve işlemcinin aynı anda işlem yaptığı bit sayısını da verir. 8 Bitlik veriye bayt (byte) adı verilir ve bellek miktarı genellikle bayt olarak ölçülür. Veri yolu, 8 bit ise 8 bitlik-BYTE işlemci, 16 bit ise 16 bitlik-WORD işlemci, 32 bit ise 32 bitlik-DWORD (Double WORD) işlemci, 64 bit ise 64 bitlik-QWORD (Quad WORD) işlemci olarak adlandırılır. Bu nedenle veri yolu genişliği ile işlemcinin akümülatör ve kaydedicilerin bit sayısı da aynı olur.

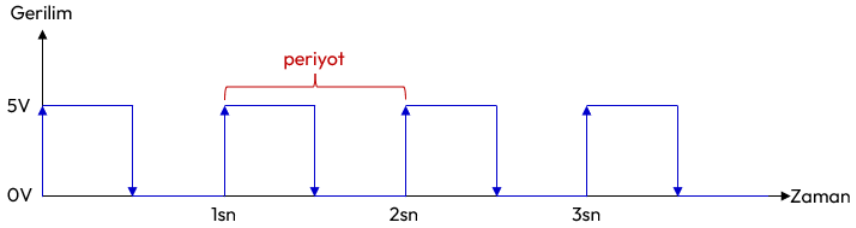
Adreslenebilir bellek miktarı ise işlemcinin adres hatlarının sayısı yani adres yolu (address bus) ile belirlenir. Adres yolu ne kadar geniş ise adreslenebilir bellek miktarı da o kadar artar.

İşlemci	Veri Yolu	Adres Yolu	Adreslenebilir Bellek: bayt	kilobayt	megabayt	gigabayt
Motorola 6802	8 Bit	16 Bit	2^{16}=65.536	64		
Intel 8086	16 Bit	20 Bit	2^{20}=1.048.576	1024	1	

Intel 80286	16 Bit	24 Bit	2^{24}=16.777.216		16	
Intel Pentium	32 Bit	32 Bit	2^{32}=4.294.967.296		4096	4
Intel i7	64 Bit	36 Bit	2^{36}=68.719.476.736			64

Tablo 1. Bazı İşlemcilerin Adres ve Veri Yolu Genişlikleri

Bilgisayarlarda kablodaki gerilimin değeri genelde 5V, 3.3V, 2.8V, 2V, 1.5V veya 1V gibi değerler alır. Kullanıcılar için gerilimin ne olduğu değil varlığı önemlidir. Gerilim değerin yüksek olması fazla akım çekme ve ısınma anlamına gelir. Bu nedenle zamanla daha düşük gerilimle çalışan işlemciler tasarlanmıştır.



Şekil 2. Frekansı 1 olan Kare Dalga İşareti

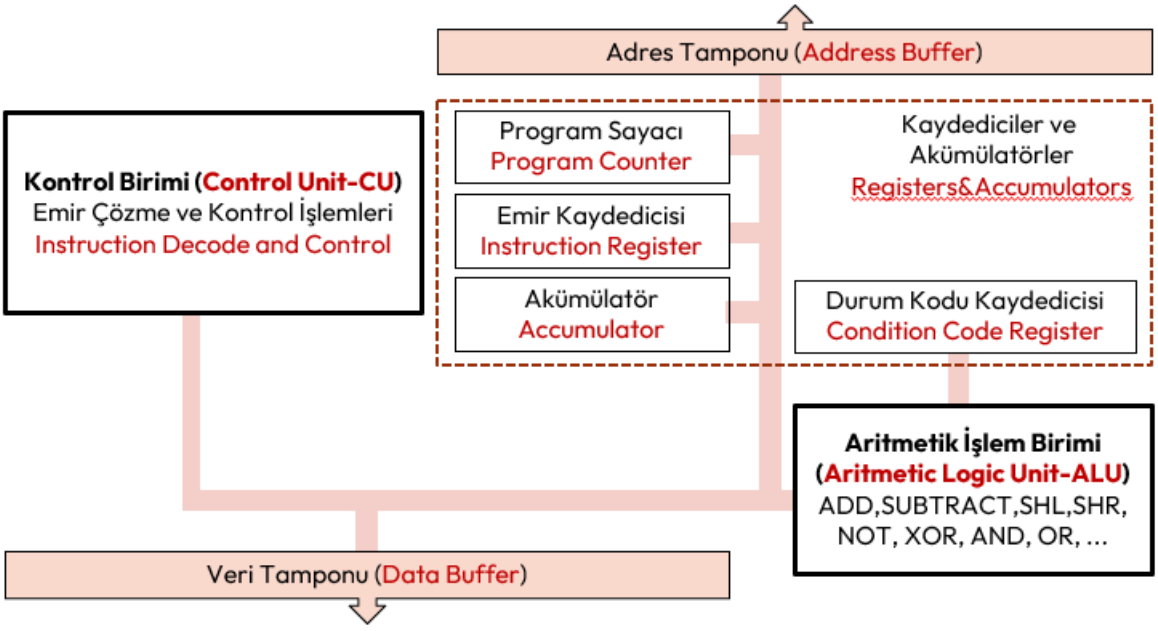
İşlemciler kare dalga olarak oluşturulmuş elektriksel bir işareti saat olarak kullanır. Bu işaret işlemciye Ø ya da CLK bacaklarından uygulanır. Saniyede 1 kez bir dalga üreten işaretin frekansı 1/1sn = 1 Hertz= 1 Hz. olarak hesaplanır. Bir mikro saniyede bir kez dalga üreten işaretin frekansı; 1/1µs = $1/10^{-6}$s = 10^{6} Hertz = 1.000.000 Hertz = 1 MHz olarak hesaplanır.

İşlemciye uygulanan saatin frekansı ne kadar yüksek olursa işlemci o kadar hızlanır. Ancak işlemcilerin de tasarımından ötürü karşılayabileceği belli bir frekans değeri bulunmaktadır. Günümüzde 5 GHz frekansını karşılayan işlemciler bulunmaktadır.

İşlemcinin Emri Alma, Çözme ve İcra Döngüsü

Basit bir işlemcide;

- Kaydediciler (register), işlemci içerisinde adres ve veri saklayan mini belleklerdir.
- Akümülatörler ise bellekten alınan verileri saklamak için en sık kullanılan kaydedicilerdir.
- Program sayacı (program counter), işlemcinin icra edeceği emri (instruction) takip için kullanılır. Getirilecek bir sonraki emrin bellek adresini içerir.
- Emir kaydedicisi (instruction register) işlemcinin o an icra ettiği emirdir.
- Koşul kodu kaydedicisi (condition code register), herhangi bir işlemin durumunu gösteren farklı bayraklar (flag) içerir.



Şekil 3. Basit bir İşlemcinin Yapısı

Bu işlemci tasarımı aynı zamanda depolanmış program (stored program) kavramı olarak adlandırılır ve *Von Neumann Mimarisi* olarak da bilinir[2]. *John Von Neumann* tarafından 1946-1949 yılları arasında tasarlanan bu mimari günümüzde modern bilgisayarlarda hala kullanılmaktadır.

Basit bir bilgisayara elektrik verildiğinde aşağıdaki üç adımlık çevrim elektrik kesilene kadar sürekli tekrarlanır;

1. Emri Alma (fetch): Program sayacının (program counter) tuttuğu adresteki emir (instruction) bellekten okunur.
2. Çözme (decode): İşlemci içerisindeki kontrol birimi (control unit) tarafından emrin ne anlama geldiği ve buna bağlı nelerin yapılacağını (örneğin ulaşılacak bellek adresi değiştirilir) belirler ve program sayacı bir sonraki emir için bir artırılır.
3. İcra (execute): Çözülen emir koduna bağlı olarak gerekirse yine işlemci içinde bulunan aritmetik işlem birimini (Aritmetic Logic Unit-ALU) çalıştırılır ve emrin işlenmesi sonucunda ortaya çıkan değerler ise işlemci içinde bulunan akümülatörlere (accumulator) veya kaydedicilerde (register) saklanır.

Burada çözülecek emirlerin (instruction) her biri bir tamsayıya karşılık gelir. Bu saylar emir kodu (instruction code) veya makine kodu (opcode) olarak da bilinirler. İşlemciler tüm sayıları emir kodu olarak anlamaz. Yani sınırlı sayıda sayı emri tanırlar. Yukarıda verilen Motorola 6802 işlemcisi 72 farklı emri, Intel 8086 işlemcisi ise 116 farklı emri tanır. Çözülebilecek tüm emirlerin oluşturduğu kümeye emir seti (instruction set) adı verilir.

İşlemci Tasarlama Stratejileri

İşlemciler, aşağıda verilen üç strateji ile tasarlanırlar;

1. İndirgenmiş Emir Setli İşlemciler (Reduced Instruction Set Computing-RISC): Az sayıda emir alan yüksek hızlı işlemci tasarımı. RISC işlemciler az sayıda emre sahip olduğundan emrin çözülmesi ve icrası da kısa zaman almaktadır. Bu da çalıştırılacak kodu büyütür.
2. Karmaşık Emir Setli İşlemciler (Complex Instruction Set Computing-CISC): Çok fazla emri anlayan karmaşık nispeten yavaş işlemci tasarımı. CISC işlemciler, çok sayıda emir tanıyabildiğinden emrin çözülmesi ve icra edilmesi nispeten daha uzun sürer. Ancak yüksek düzey dillerin daha kolay makine koduna çevrilmesini kolaylaştırır.
3. Özel İşlemciler: Bunlar grafik kartları ve yapay zekâ gibi belli amaçlar için tasarlanmış işlemcilerdir.

Genel olarak işlemcilerde *"fetch-decode-execute"* çevrimi 1 saat periyodunda tamamlanmaz. RISC işlemciler için bu çevrim, birçok emir için 1 saat periyodunda tamamlanır. CISC işlemciler için ise birden fazla saat periyodu gerekir. Çünkü CISC işlemcilerin emir seti daha geniş olduğundan emrin çözülmesi daha fazla vakit alır. Bu nedenle RISC işlemciler CISC işlemcilere göreceli olarak daha hızlıdır.

Günümüzde ticari olarak satılan işlemcilerin çoğu ortak emir seti kullanmaya başlamışlardır;

– Masaüstü bilgisayarların birçoğu Intel ve AMD Marka işlemciler kullanır ve bunlar IA-16, IA-32, x86-16, x86-64, AMD64, ... emir setleri ve eklentilerini desteklemektedir.
– Bilinen birçok telefon işlemcisi Acorn Risc Machine-ARM tabanlı işlemciler kullanır. Bu işlemcilerde ARMv7, ARMv8, ... emir setleri ve eklentileri kullanılmaktadır.

Program ve Programlama

Belli bir işlemi gerçekleştirmek üzere, işlemciye verilen bir dizi emre (instruction), bilgisayar programı (computer program) denir[3]. Bu bir dizi emrin, işlemcinin çalıştıracağı şekilde hazırlaması işlemine programlama (programming) denir. Programlama emir kodları (instruction code/opcode) ile yapıldığından bu programın dili, makine dili (machine language) olarak adlandırılır.

[2] https://www.youtube.com/watch?v=ByIlwN8q2ss
[3] "ISO/IEC 2382:2015". ISO. 2020-09-03. [Software includes] all or part of the programs, procedures, rules, and associated documentation of an information processing system.

Her işlemci üreticisi, her bir emre farklı makine kodu verdiğinden programlama oldukça zorlu ve teknik bilgi gerektirmekteydi. Burada daha önce yazılmış bir programın ne yaptığını anlamak için her defasında bu listelere defalarca bakmak gerekiyordu.

Sembolik İsim (Mnemonic)	16 Tabanında Emir Kodu (Hexadecimal Opcode)	Tanım
ABA	1B	ADD A to B → A
INCA	4C	INCREMENT A
INCB	5C	INCREMENT B
SBA	10	SUBTRACT B from A→ A
LDA	96	LOAD M to A, M: Memory
LDB	D6	LOAD M to B, M: Memory

Tablo 2. 6802 Emir Seti Örneği ve Sembolik İsim Listesi

Programın okunurluğunu artırmak için her bir emre sembolik isimler (mnemonic) verilmeye başlandı. Bu sembolik isimlerle yazılan program montaj kodu (assembly code) olarak adlandırılır. Montaj kodları bir metin dosyasına yazılarak bir dosyaya saklanır. Bu dosyalar montaj dosyaları olarak adlandırılır. Montaj dosyalarını makine diline çeviren programlar ilk derleyicilerdir (compiler). Bu derleyiciler, sembolik isimlerle yazılmış program kodlarını makine koduna çeviren programlardır.

Aşağıda x86 emir seti kullanan Linux işletim sisteminde örnek montaj (assembly) kodu örneği verilmiştir; Program genel amaçlı CL kaydedicisine koyulan sayıya kadar 2 ve 3'e bölünebilen sayıların toplamını bulup yine genel amaçlı DX kaydedicisine koyar[4].

```
section  .text          ; programı oluşturan emir kodları buradan başlar
         global   _start
_start:
         mov   dx, 0    ; dx 0 olsun. Toplamı tutatcak.
         mov   cl, 99   ; 99 kez tekrarlanacak.
sum:
         mov   ax, cx   ; cx, ax e kopyalanır. Bölme işlemi için.
         mov   bl, 3    ; bl 3 olsun.
         div   bl       ; ax/bl.
         cmp   ah, 0    ; Kalan olup olmadığı kontrol ediliyor.
         jne   cont     ; Kalan yoksa, 'cont' etiketine git.
         mov   bl, 2    ; Sonra bl 2 olsun.
         div   bl       ; ax/bl
         cmp   ah, 0    ; Kalan olup olmadığı kontrol ediliyor.
         jne   cont     ; Kalan yoksa, 'cont' etiketine git.
         add   dx, c    ; Değilse, cx deki sayı 2 ve 3'>e bölünür. Toplama ekle.
cont:
         loop  sum      ; Bir sonraki sayı için 'sum' etiketine dön.
exit:
         mov   eax, 1   ; Artık programdan çıkıyoruz.
         mov   ebx, 0   ; Return 0.
         int   80h      ; linux çekirdeğini çağır. (soft interrupt 80h!).
```

Genel Amaçlı Bilgisayarlar ve İşletim Sistemleri

Genel amaçlı yani herkesin kendi amacına göre kullanacağı bilgisayarlar; Veri girişini sağlayan klavye, verileri görüntüleneceği monitör, işlenen verilerin elektrik kesildiğinde saklanacağı disk ve benzeri donanımlara sahip olmalıdır.

Genel amaçlı bilgisayarın kullanıcı isteklerine cevap verebilmesi için;

1. Çeşitli programlara sahiptirler. İşte bu programlar bütününe işletim sistemi (operating system) adı verilir. DOS, Windows, Unix, Linux, MacOs, ChromeOS, OS/2 bunların en çok bilinenleridir.

[4] https://github.com/armut/x86-Linux-Assembly-Examples

2. Elektrik verildiğinde klavye, monitör ve disk olup olmadığı kontrol eden Basic Input Output System-BIOS programı koşturulur. BIOS gerekli kontrollerden sonra bilgisayara işletim sistemini yükler. İşletim sistemi yüklendikten sonra, işletim sistemi kullanıcının vereceği komutları bekler.
3. Kullanıcı işletim sitemine vereceği konutlarla yeni bir program derleyebilir, derlenmiş bir programı çalıştırabilir.
4. Kullanıcının vereceği komutlar grafik olmayan işletim sistemlerinde (DOS, Unix, Linux gibi) komut yazılarak verilir. Komut yazılan bu ekrana konsol, terminal ya da komut satırı adı verilir. Sunucu işletim sistemlerinde hala tercih edilmektedir.
5. Grafik ara yüzlerine sahip işlettim sistemlerinde (Windows ve MacOs İşletim Sistemleri ile KDE, Mate, GNome, Cinnamon, LXQt, XFce ve Deepin gibi grafik ara yüzleri içeren Linux İşletim Sistemleri) fare tıklamasıyla çoğu komut verilebilmektedir.

İşletim sistemlerinin yaygınlaşması ile kullanıcının bilgisayarı donanım bilmeden yapabilmesi sağlanmıştır. Günümüzdeki işletim sistemlerinin temel programlama dili olarak C dili kullanılmaktadır.

TEMEL YAZILIM KAVRAMLARI

Tarihçe

1642 Yılında *Blaise Pascal* tarafından icat edilen Pascalline Hesap Makinesi, eldeli toplama, ödünç almalı çıkarma yapıyordu.

1671 Yılında *Gottfried Wilhelm Leibniz* tarafından icat edilen Leibniz Çarkı, toplama, ödünç almalı çıkarmanın yanında bölme, çarpma ve karekök işlemlerini yapıyordu.

1801 Yılında *Joseph Maria Jacquard* dokuma tezgahlarında desenleri oluşturmak için delikli kartlar (punch card) kullanmayı icat etmiştir. Sonradan bilgisayara veri girdisi sağlamak ve sonuçların çıktısını göstermek için kullanılacaktır.

1830 Yılında *Charles Babbage* fark makinesi olan analitik makineyi icat etmiştir. Buhar gücü kullanan bu makinenin mantıksal işlem birimi, veri depolama birimi, giriş çıkış üniteleri bulunuyordu. *Augusta Ada Byron*, Babbage ile beraber çalışmıştır. Byron, analitik makinenin bir dizi matematiksel işlemi gerçekleştirebildiğini fark etti ve bu makineyi *Bernoulli* sayılarını üretmek için kullandı. Bu sayıların hesaplanması için yazdığı algoritma tarihteki ilk bilgisayar programı olarak kabul edilir. Bu sebeple *Byron*, ilk programcı olarak kabul görür.

1854 Yılında *George Boole* tarafından ikili sayı sistemini geliştirmiş bugün bilgisayarlarımızın kullandığı bool cebiri üzerine çalışmalar yapmıştır.

1890 Yılında *Herman Hollerith* tarafından delikli kartlarla bilgilerin yüklenebildiği ve bu bilgiler üzerinde toplama işlemlerinin yapılabildiği bir elektro mekanik araç geliştirdi. ABD'nin 1890 nüfus sayımında başarılı biçimde kullanıldı. *Hollerith* başarılı olunca bir şirket kurdu ve daha sonra üç firma işle birleşerek 1924 yılında adını IBM olarak değiştirdi.

1941 Yılında *Konrad Zuze* Z3 isimli elektrik motorları ile çalıştırılan mekanik bir bilgisayar yaptı. Bu (Z1, Z2, Z3 ve Z4 serisi) program kontrollü ilk bilgisayardır.

1944 Yılında *Howard Aitken*, IBM ile iş birliği yaparak MARK I'i yaptı. Bilgiler, MARK I'e delikli kartlarla veriliyor ve sonuçlar yine delikli kartlarla alınıyordu. Saniyede 5 işlem yapabiliyordu. Boyutları,18 m uzunluğunda ve 2,5 m yüksekliğinde idi. İnsan müdahalesi olmadan sürekli olarak, hazırlanan programı yürüten ilk bilgisayar idi ve tam olarak elektronik bir bilgisayar değildi.

1943-1945 Yılları arasında *Konrad Zuze*, ilk yüksek düzey programlama dili olan Plankalkül'ü, Z1 bilgisayarı için geliştirmiştir. Emir kodlarını (instruction code) ve sembolik isimleri (mnemonic) içermeyen programlama dilleri yüksek düzey programlama dili (high level programming language) olarak adlandırılır. Yüksek düzey programlama dilleri çoğunlukla İngilizce sözcükleri kullanır.

1946 Yılında *John Von Neumann* önderliğinde Pensilvanya Üniversitesinde ENIAC (Elektronik Sayısal Hesaplayıcı ve Doğrulayıcı) isimli sayısal elektronik bilgisayar tamamlandı. Askeri amaçla üretildi ve top mermilerinin menzillerini hesaplamak için kullanıldı. *Neumann* tarafından geliştirilmiş bu mimari (Stored-program computer and Universal Turing machine § Stored-program computer), günümüzdeki bilgisayarlarda hala kullanılmaktadır. 18,000 adet elektronik tüp kullanılan ENIAC; 150 kW gücünde idi ve 50 ton ağırlığıyla 167 m^2 yer kaplıyordu. Saniyede 5.000 toplama işlemi yapabiliyordu. Mark-I'den 1000 kat daha hızlıydı.

Bir önceki bölümde anlatıldığı üzere emir kodu veya sembolik isimlerle yazılan programlar düşük düzey programlama (low level programming) olarak adlandırılır. Bu programlamanın yapılabilmesi için hem işlemcinin hem de hafıza ve işlemciye bağlanan diğer bileşenlerin nasıl çalıştığı ve tasarlandığını çok iyi bilinmesi gerekir. Daha çok elektronik ve bilgisayar mühendisleri ya da sistemi tasarlayan matematikçiler bu programlama türünü kullanır.

Genel Amaçlı Yüksek Düzey Diller

Sembolik isimler veya makine kodu yerine bildiğimiz konuşma diline yakın talimatlarla metin olarak yazılan programlara dilleri, yüksek düzey programlama dilleri (high level programming language) olarak adlandırılır. Yani emir kodu (instruction code) ve sembolik isim (mnemonic) kullanmayan programlama dilleri yüksek düzey programlama dilleridir.

Yüksek düzey dillerde talimatlar (statement) olarak yazılan programlar yine kendine has derleyiciler tarafından makine diline çevrilir. Her talimat, genellikle birden fazla emirden (instruction) oluşur.

1954 Yılında FORTRAN (FORmula TRANslation), *John Backus* liderliğinde IBM 704 için geliştirilirdi. 32 farklı talimata (statement) sahiptir. İlk genel amaçlı, yüksek düzey bir programlama dilidir.

Yüksek seviyeli bir dilde yazılmış bir talimat (statement), bilgisayara belirtilen bir eylemi gerçekleştirmesini söyler. Yüksek seviyeli bir dildeki tek bir talimat, birkaç emri (instruction) içerebilir. Talimatlar, yapılaması gerekeni kısa ve net olarak belirten mantıksal satırlardır (logical sequence).

Aşağıda üç kenarı teyp ünitesinden alınan bir üçgenin alanını hesaplayan FORTRAN II programı verilmiştir[5]. Görüldüğü üzere talimatlar kısa öz ve anlaşılırdır.

```
C AREA OF A TRIANGLE - HERON'S FORMULA
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT -
C INTEGER VARIABLES START WITH I,J,K,L,M OR N
      READ(5,501) IA,IB,IC
  501 FORMAT(3I5)
      IF (IA) 701, 777, 701
  701 IF (IB) 702, 777, 702
  702 IF (IC) 703, 777, 703
  777 STOP 1
  703 S = (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - IA) * (S - IB) * (S - IC) )
      WRITE(6,801) IA,IB,IC,AREA
  801 FORMAT(4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     $13H SQUARE UNITS)
      STOP
      END
```

1956 Yılında ticari amaçlarla kullanılabilen ve seri halde üretimi yapılan ilk bilgisayar UNIVAC I oldu. UNIVAC, giriş-çıkış birimleri manyetik bant idi ve bir yazıcıya sahipti. UNIVAC 1951-1959 arasında üretilen bilgisayarlarda vakum tüpleri kullanıldı. Bu tüpler bir ampul büyüklüğünde, çok fazla enerji harcamakta ve çok fazla ısı yaymakta idiler. Veri ve programlar manyetik teyp ve tambur gibi bilgi saklama araçlarıyla saklandı. Veriler ve programlar bilgisayara delgi kartları ile yükleniyordu. 1959 yılı sonrasında üretilen bilgisayarlarda transistörler (yaklaşık 10 bin adet) kullanıldı. Artık transistor içeren ikinci kuşak bilgisayarlar hayatımıza girmiştir.

Arapsaçı Kod

1955-1959 Yılları arasında FLOW-MATIC B0 (Business Language Version 0) Dili, *Grace Hopper* tarafından UNIVAC I için geliştirildi. İngilizceye benzeyen ilk yüksek düzey programlama dilidir. Aşağıda B0 Dilinde örnek bir program verilmiştir[6].

```
(0)  INPUT INVENTORY FILE-A PRICE FILE-B ; OUTPUT PRICED-INV FILE-C UNPRICED-INV
    FILE-D ; HSP D .
(1)  COMPARE PRODUCT-NO (A) WITH PRODUCT-NO (B) ; IF GREATER GO TO OPERATION 10 ;
    IF EQUAL GO TO OPERATION 5 ; OTHERWISE GO TO OPERATION 2 .
(2)  TRANSFER A TO D .
(3)  WRITE-ITEM D .
```

[5] https://github.com/scivision/fortran-II-examples/blob/main/herons_formula.f
[6] https://gist.github.com/marcgg/f9f2da31a973ba319809

```
 (4)  JUMP TO OPERATION 8 .
 (5)  TRANSFER A TO C .
 (6)  MOVE UNIT-PRICE (B) TO UNIT-PRICE (C) .
 (7)  WRITE-ITEM C .
 (8)  READ-ITEM A ; IF END OF DATA GO TO OPERATION 14 .
 (9)  JUMP TO OPERATION 1 .
(10)  READ-ITEM B ; IF END OF DATA GO TO OPERATION 12 .
(11)  JUMP TO OPERATION 1 .
(12)  SET OPERATION 9 TO GO TO OPERATION 2 .
(13)  JUMP TO OPERATION 2 .
(14)  TEST PRODUCT-NO (B) AGAINST ZZZZZZZZZZZZ ; IF EQUAL GO TO OPERATION 16 ;
     OTHERWISE GO TO OPERATION 15 .
(15)  REWIND B .
(16)  CLOSE-OUT FILES C ; D .
(17)  STOP . (END)
```

Görüldüğü üzere bu tarihlere kadar yazılan programlarda bir sürü GO TO ya da JUMP TO talimatı (statement) bulunmaktadır. Bu durumda programın ne yaptığı konusunda fikir yürütmeye çalıştığımızda okunaklılık oldukça azdır. İşte okunaklılığın az olduğu bu program kodlarına arapsaçı kod (spaghetti code) adı verilir. Günümüzde ne yaptığı kolay anlaşılamayan kodlar için bu kavram çokça kullanılmaktadır.

1958 Yılında LISP programlama dili, *John McCarthy* önderliğinde *Steve Russell* tarafından geliştirilmiş ikinci yüksek düzey programlama dilidir.

1959 Yılında Common Business-Oriented Language-COBOL programlama dili, *CODASYL* komitesi tarafından İngilizceye benzer ikinci yüksek düzey bir dil olarak geliştirilmiştir.

1958 Yılında daha sonra ALGOritmic Language-ALGOL adını alan International Algebraic Language-IAL Programlama Dili geliştirilmiştir. Bu dil daha sonra ortaya çıkan birçok dile önderlik etmiştir. Bu dil ile birlikte kod blokları (code block), iç içe fonksiyon (nested function), değişken kapsamı (scope) ve dizi (array) kavramları programcının hayatına girmiştir. ALGOL dili, barındırdığı özellikler sebebiyle, kendisinden sonra ortaya çıkan bütün programlama dilleri için bir esin kaynağı olmuştur.

İşin doğası gereği makine diline çevrilecek tüm metinler gözle okunabilmektedir. Yani bütün kaynak kodlar gözle okunabilir.

Değişken

Değişkenler (variable) aslında cebirden bildiğimiz değişkenlerdir. Cebir (algebra) işlemlerinde doğrudan problemde verilen sayıları değil, onları temsil eden değişkenleri kullanırız.

$$3x^2 + 2x + 10$$

$$c^2 = a^2 + b^2$$

$$c = 2\pi r$$

Yukarıdaki cebirsel ifadelerin ilkinde bir polinom, ikincisinde ise bir dik üçgenin kenar uzunlukları arasındaki ilişki verilmiştir. Bu ifadelerde `x`, `a`, `b` ve `r` bağımsız değişken, `c` ise bağımlı değişkendir. Aslında problemi çözerken bunlar sayılara karşılık gelir. Örneklerdeki `3`,`2`,`10`,`2` ve `π` ise sabit sayılardır.

Bilgisayarda ise sayıları tutacak değişkenler, bir miktar bellek bölgesini işgal eder ve bu bellek bölgesinde çeşitli biçimlerde tutulurlar. Tamsayılar, ikili (binary) sayı olarak bellekte tutulur. Bu ikili sayının en anlamlı bitinin işaret biti olarak kullanılıp kullanılmayacağına göre bellekte biçimlendirilir;

Bit Sayısı	Bellek Miktarı	Sayı Sınırları
8 Bit	bayt	-127 ile +128 arasında veya 0 ile 255 arasında
16 bit	2 bayt	-32.767 ile +32.768 arasında veya 0 ile 65.535 arasında
32 bit	4 bayt	-2.147.483.648 ile +2.147.483.647 arasında veya 0 ile 4.294.967.295 arasında
64 bit	8 bayt	-9.223.372.036.854.775.808 ile +9.223.372.036.854.775.807 arasında veya

		0 ile 18,446,744,073,709,551,615 arasında

Tablo 3. Tamsayı Değişkenler ve Sayı Sınırları

Gerçek sayılar kayan noktalı sayı (floating point number) olarak biçimlendirilir; 1914 yılında *Leonardo Torres Quevedo* tarafından Charles Babage'ın analitik makinesine dayalı kayan nokta analizi yayınladı.

$$12345 = 1234{,}5x10^1 = 123{,}45x10^2 = 12{,}345x10^3 = 1{,}2345x10^4 = 0{,}12345x10^5$$

Yukarıda ondalık tabanda verilen gerçek bir sayıya ilişkin kesir noktasının kaydırılması ile aynı sayı ifade edildiği bir örnek verilmiştir. Kesir noktası sola kaydırıldığında 10 ile çarpılmış, sağa kaydırıldığında ise 10 ile bölünmüş olur. Kayan noktalı sayının üs kısmı dışında kalan kısmı taban (base) veya kesir (fraction) olarak adlandırılır. Tabanın kesir noktası sağında kalan hanelerine ise mantis (mantissa) adı verilir.

1938 yılında *Konrad Zuse*, ilk ikili (binary) programlanabilir mekanik bilgisayar olan Z1'i yapmıştı. Bu bilgisayar, 7 bitlik işaretli üs, 17 bitlik anlamlı değer ve bir işaret biti içeren 24 bitlik ikili tabanda kayan noktalı sayı kullanmıştır. Bilgisayarlarda tamsayılarda olduğu gibi kayan noktalı sayılar da ikili sayı tabanında tutulur[7]. Örneğin ondalık tabanda 50,25 sayısı ikili tabanda 1.1001001 x2^5 olarak ifade edilir.

Bit Sayısı	Bellek Miktarı	Sayı Sınırları
32 bit	4 bayt	$1{,}2x10^{-38}$ ile $3{,}4x10^{+38}$ arasında tek hassasiyetli gerçek sayılar; en soldaki 1 bit işaret biti, sonraki 8 bit üs ve geri kalan 23 bit ise kesir olarak kullanılan ikilik tabanda saylardır.
64 bit	8 bayt	$2{,}3x10^{-308}$ ila $1{,}7x10^{+308}$ arasında çift hassasiyetli gerçek sayılar; en soldaki 1 bit işaret biti, sonraki 11 bit üs ve geri kalan 52 bit ise kesir olarak kullanılan ikilik tabanda saylardır.

Tablo 4. Kayan Noktalı Sayılar ve Sınırları

Program yazılırken tanımlanacak değişkenlere ilişkin bellek ihtiyacı programcı tarafından belirlenir. Programcı değişkene tahsis edilecek bellek miktarına ve tutacağı içeriğe göre veri tipi (data type) belirler. Programcı tarafından kullanılacağı amaca göre belirleyeceği veri tipinde istediği kadar değişken tanımlayabilir.

Fonksiyon

Matematikte fonksiyon (function), bir veya daha fazla kümenin elemanını tek bir kümenin elemanına eşleyen ilişkidir (relationship).

Girdi	İlişki	Çıktı
1	x2	2
2	x2	4
3	x2	6
4	x2	8
...		

Tablo 5. İki ile çarpma fonksiyonu.

Girdi olan bağımsız değişkenler ile çıktı olan bağımlı değişken arasındaki bu ilişki, matematiksel olarak aşağıdaki şekilde ifade (expression) edilir.

$$f(x) = 2x$$

Burada f, fonksiyon anlamında x ise girdi anlamında olup parametre olarak adlandırılır. Böylece tabloda verilen fonksiyon tablosuna her girdiyi yazmak zorunda kalmayız. Bazen fonksiyona aşağıdaki örneği verilen şekilde bir ad verilmez;

$$y = 2x$$

Bazı durumlarda ise girdi daha detaylı tanımlanır;

[7] IEEE Computer Society (2019-07-22). IEEE Standard for Floating-Point Arithmetic. IEEE STD 754-2019. IEEE. pp. 1–84.

$$x \in R \text{ Olmak üzere } f(x) = 2x + 3$$

Bu fonksiyon, reel sayı kümesindeki her `x` elemanını, hesaplanan `f(x)` değerine eşleyen bir ifadedir. Yani `x=1.0` argümanını `f(1.0)=5.0` reel sayısına eşler. Kısaca fonksiyon `5.0` reel sayısını hesaplayıp döndürür (return).

$$a \in R \; ve \; b \in Z \text{ Olmak üzere } g(a,b) = 2a + b$$

Bu fonksiyon ise reel sayı kümesindeki her a elemanı ile tamsayı kümesindeki b elemanını, hesaplanan `g(a,b)` değerine eşleyen bir ifadedir. Yani `a=1.0` ve `b=1` argümanlarını `g(1.0,1)` fonksiyonu yine reel sayı kümesinden `3.0` reel sayısına eşler. Kısaca fonksiyon, `3.0` reel sayısını hesaplayıp döndürür (return). Buradaki `x`, `a` ve `b` parametre olarak adlandırılır. `f(1.0)` ifadesindeki `1.0` ise parametrenin o an andığı değeri belirtir ve gerçek parametre (actual parameter) ya da argüman (argument) olarak adlandırılır. Bu fonksiyonlar;

$$f(3.0) + g(3.0,2) + f(2.0) + g(1.0,3)$$

Şeklinde tanımlandıktan sonra tekrar tekrar çağrılabilirler (call).

Yapısal Programlama

1958 Yılındaki FORTRAN-II diline çıktı üretmeyen fonksiyonlar için SUBROUTINE, değer üreten fonksiyonlar için FUNCTION, CALL ve RETURN özellikleri eklenmiştir.

1958 ile 1969 Yılları arasında ALGOL diline çıktı üretmeyen fonksiyonlar için PROCEDURE ve BEGIN-END arasında kod bloğu özellikleri eklenmiştir.

1966 Yılındaki FORTRAN-IV diline INTEGER, REAL, DOUBLE PRECISION, COMPLEX ve LOGICAL veri tipleri (data type) ile Mantıksal IF, aritmetik (üç yollu) IF ve DO döngüsü talimatları (statement) eklenmiştir.

Bu yıllarda her ne kadar çıktı üretmeyen fonksiyonlar için SUBROUTINE, birden fazla çağrı noktası olabilen COROUTINE ve FUNCTION kullanılıyor olmasına rağmen hala GO TO/JUMP TO ifadeleri kullanılmakta ve kafa karışıklığına devam etmekteydi. 1968 Yılında *Edsger W. Dijkstra* önderliğinde GO TO/JUMP TO ifadesini zararlı olarak ilan edilmiştir.

1970 Yılında *Niklaus Wirth* tarafından Pascal dili geliştirildi. Bu dil ilk geliştirilen yapısal programlama dilidir. Yapısal programlamada (structural programming) verileri taşıyan veri yapıları (değişkenler ve diziler) ile bunu işleyen kontrol yapıları (if, for, do, repeat) birbirinden ayrılmıştır. Pascal dili, kendi kendini çağıran özyinelemeli (recursive) fonksiyonlar ile değişkenlerin adreslerini tutan göstericileri (pointer) desteklemektedir[8].

Yapısal programlamanın genel çerçevesi;

1. Programın başladığı yeri belirten bir ana fonksiyon (main function) tanımlanır. Kodlaması yapılacak işi birbirini çağıran fonksiyonlar olarak bu ana fonksiyonda yazarız. Yani yapısal programlamada programlama, fonksiyonların birbirini çağırmasıyla yapılır.
2. Her fonksiyonda ilk önce veri yapıları (data structure) tanımlanır. Veri yapıları;
   a. En basit veri yapısı `char`, `integer`, `float` ve `double` gibi ilkel veri tiplerinden (primitive data type) olabilen değişken (variable) olabileceği gibi,
   b. İlkel veri tiplerinden meydana gelen dizi (array),
   c. İlkel veri tiplerinin birkaçından veya dizilerinden bir araya gelmesiyle oluşan yapılar (structure),
   d. Ya da dinamik olarak yani çalıştırma anında birbirine bağlanmış verilerden oluşan bağlı liste (linked list) olabilir.
3. Her fonksiyonda tanımlanan veri yapılarının ardından bu yapıları işleyecek kontrol yapıları (control structure) tanımlanır. Kontrol yapıları bir veya daha fazla veya iç içe `if`, `if-else`, `switch`, `while`, `do-while`, `for`, `continue`, `break`, `goto` ve `return` talimatlardan (statement) oluşur.

[8] Wirth, Niklaus (1976). Algorithms + Data Structures = Programs. Prentice-Hall. p. 126

Yapısal programlamanın en çok uygulandığı Pascal dilinde ana fonksiyon nokta ile biten BEGIN-END bloğudur. C dilinde ana fonksiyon, `main()` fonksiyonudur.

C Programlama Dili, *Dennis M. Ritchie* tarafından Bell Laboratuvarlarında UNIX işletim sistemini geliştirmek için geliştirilen genel amaçlı, üst düzey bir dildir. C, ilk olarak 1972'de DEC PDP-11 bilgisayarında çalıştırılmıştır. *Ken Thompson* tarafından geliştirilen B adlı daha eski bir dilden gelişmiştir. 1978 yılında *Brian Kernighan* ve *Dennis Ritchie*, günümüzde K&R standardı olarak bilinen C'nin ilk kamuya açık kılavuzunu hazırlamışlardır.

C programlama dili; Öğrenmesi kolaydır, yapısal (structural) programlama dilidir, hızlı ve verimli (efficient) programlar üretir, assembly dili desteği ile düşük düzey programlamayı (low level programming) da destekler ve standart başlık dosyaları sayesinde çeşitli bilgisayar platformlarında derlenebilir. Bu özellikleri nedeniyle neredeyse tüm işletim sistemlerinin çekirdeği C dilinde yazılmıştır ve sistem dili olarak da adlandırılmaktadır.

Yapısal programlamada veri ile bunu işleyen yapılar birbirinden ayrıldığından arapsaçı programlamanın aksine okunaklılık oldukça yüksektir.

DERLEYİCİLER VE ÇALIŞTIRMA ORTAMI

Genel amaçlı bilgisayarların çok kullanılmasıyla birlikte metin editörleri de (Windows Not Defteri, Mac TextEdit, Notepad++, Epsilon, EMACS, nano, vim veya vi) işletim sistemlerinin bir parçası olmuştur. Metin editörlerinde yazılan programlar ise derleyiciler (compiler) tarafından makine koduna çevrilerek icra edilebilir (executable) dosya olarak kaydedilebilmektedir.

C dilinde programlama öğrenmeye başlamak için ilk adım, programı yazabileceğiniz bir metin editörü kullanmak ve yazdığınız programa ilişkin kaynak kodları c uzantısı ile dosya olarak saklamaktır. İkinci adım ise işletim sisteminizde (operating system) çalışabilen bir icra edilebilir dosya oluşturan bir derleyici kurmaktır. Yani bilgisayarınızda iki yazılım aracına ihtiyacınız vardır; birincisi C derleyicisi, ikincisi ise basit bir betin editörüdür.

Derleyici için girdi, kaynak kodu dosyasıdır. Kaynak dosyasında yazılan kaynak kodu, içerisinde C talimatları (statement) olan, insan tarafından okunabilen metin dosyasıdır. Kaynak kodda verilen talimatlara göre işlemcinizin programı çalıştırabilmesi için, bunun makine diline derlenmesi (compile) gerekir. Derleme, bir başka programlama dilinden sembolik isim (mnemonic) ya da makine koduna (instruction code) dönüştürme işlemidir. Bunu yapan programlara derleyici (compiler) adı verilir.

C derleyicileri

Günümüzde birçok C derleyicisi mevcuttur. En çok kullanılanları aşağıda verilmiştir;

- GNU Derleyici Koleksiyonu (GCC) – GCC, popüler bir açık kaynaklı C derleyicisidir[9]. Windows, macOS ve Linux dahil olmak üzere çok çeşitli platformlarda kullanılabilir. GCC, çok çeşitli özellikleri ve çeşitli C standartlarına desteğiyle bilinir.
- Clang – Clang, LLVM projesinin bir parçası olan açık kaynaklı bir C derleyicisidir[10]. Windows, macOS ve Linux dahil olmak üzere çok çeşitli platformlarda kullanılabilir. Clang, hızı ve optimizasyon yetenekleriyle bilinir.
- Microsoft Visual C – Microsoft Visual C, Microsoft tarafından geliştirilen tescilli bir C derleyicisidir[11]. Yalnızca Windows için kullanılabilir. Visual C, Microsoft Visual Studio geliştirme ortamıyla entegrasyonuyla bilinir.

Bilgisayarınızın 64 bitlik Windows olduğu varsayılarak GCC derleyicisinin MINGW sürümü ilgili web sayfasından indirilerek kurulur[12]. Kurulumu tamamlandıktan sonra PATH değişkenine GCC derleyicisinin klasörü eklenir. Böylece kurulum tamamlanmış olur.

Komut satırı (DOS Prompt) açıldığında `gcc -v` komutu yazılarak derleyicinin düzgün kurulup kurulmadığı kontrol edilebilir.

```
C:\Users\ILHANOZKAN>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=D:/msys64/ucrt64/bin/../lib/gcc/x86_64-w64-mingw32/13.2.0/lto-wrapper.exe
Target: x86_64-w64-mingw32
Configured with: ../gcc-13.2.0/configure --prefix=/ucrt64 --with-local-prefix=/ucrt64/local --
build=x86_64-w64-mingw32 --host=x86_64-w64-mingw32 --target=x86_64-w64-mingw32 --with-native-
system-header-dir=/ucrt64/include --libexecdir=/ucrt64/lib --enable-bootstrap --enable-
checking=release --with-arch=nocona --with-tune=generic --enable-
languages=c,lto,c++,fortran,ada,objc,obj-c++,jit --enable-shared --enable-static --enable-
libatomic --enable-threads=posix --enable-graphite --enable-fully-dynamic-string --enable-
libstdcxx-filesystem-ts --enable-libstdcxx-time --disable-libstdcxx-pch --enable-lto --enable-
libgomp --disable-libssp --disable-multilib --disable-rpath --disable-win32-registry --
disable-nls --disable-werror --disable-symvers --with-libiconv --with-system-zlib --with-
```

[9] https://gcc.gnu.org/
[10] https://clang.llvm.org/
[11] https://visualstudio.microsoft.com/tr/vs/features/cplusplus/
[12] https://www.mingw-w64.org/downloads/

```
gmp=/ucrt64 --with-mpfr=/ucrt64 --with-mpc=/ucrt64 --with-isl=/ucrt64 --with-pkgversion='Rev3,
Built by MSYS2 project' --with-bugurl=https://github.com/msys2/MINGW-packages/issues --with-
gnu-as --with-gnu-ld --disable-libstdcxx-debug --with-boot-ldflags=-static-libstdc++ --with-
stage1-ldflags=-static-libstdc++
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 13.2.0 (Rev3, Built by MSYS2 project)

C:\Users\ILHANOZKAN>
```

Daha sonra metin editörü ile ilk kaynak kodumuzu oluşturmak için komut satırında `notepad hello.c` komutu yazılır.

```
C:\Users\ILHANOZKAN>notepad hello.c
```

Eğer böyle bir dosya yoksa editör bu dosyayı oluşturmak için onay ister. Açılan metin editörü penceresine ilk C programı yazılır ve `hello.c` olarak dosyaya kaydedilir. Eğer böyle bir dosya yoksa editör bu dosyayı oluşturmak için onay ister. Açılan metin editörü penceresine ilk C programı yazılır ve `hello.c` olarak dosyaya kaydedilir.

```
#include <stdio.h>

int main() {
   /* Ilk C Programı */
   printf("Hello, World! \n");
   return 0;
}
```

Aşağıdaki komut ile `hello.c` olarak hazırlanan kaynak kodumuz derleyici tarafından icra edilebilir `hello.exe` dosyası olarak derlenir.

```
C:\Users\ILHANOZKAN>gcc hello.c -o hello.exe
```

Derlenen programı çalıştırmak için yine aynı komut satırında `hello.exe` yazarak icra edilebilir programımız işletim sistemi tarafında işlemciye hedef gösterilerek çalıştırılır.

```
C:\Users\ILHANOZKAN>hello.exe
Hello, World!

C:\Users\ILHANOZKAN>
```

Burada programın yazılması `notepad` programında, derlenmesi `gcc` programı ile ve derlenen `hello.exe` programının çalışması tarafımızdan yapılmıştır.

Mac OSX ve Linux işletim sitemlerinde de derleyici kurulumu tamamlandıktan sonra terminal yani konsol uygulaması açılarak aşağıdakine benzer şekilde derlemeler yapılabilir.

Entegre Geliştirme Ortamı

Şimdiye kadar anlatılan kod yazma (implementation), derleme (compile), icra (execute) ya da koşma (run) ve izleme (trace) gibi süreçlerin tamamını tek bir çatı altında yürütmemizi sağlayan programlar, entegre geliştirme ortamı (Integrated Development Environment-IDE) olarak adlandırılır.

C programlama dili için en çok kullanılan entegre geliştirme ortamları aşağıda verilmiştir;

- Code::Blocks – Açık kaynaklı bir C/C++ geliştirme ortamı olup bir grafik kullanıcı ara yüzüne (Graphic User Interface-GUI) sahiptir. Windows, macOSX ve Linux işletim sitemleri üzerine kurulabilmektedir.
- Visual Studio – Microsoft tarafından geliştirilen, C dilinde yazılmış, güçlü, yüksek performanslı uygulamalar oluşturmak için kullanılabilen bir geliştirme ortamıdır. Yalnızca Windows'ta çalışır. Visual Studio, kod tamamlama (intellisense), kullanıcı ara yüzü (user interface-UI) hazırlama

desteği ve hata ayıklama (debugger) ve birçok eklentisi (plug-in) olan muazzam özelliklere sahiptir. Bu geliştirme ortamının kurulumu dipnotta verilmiştir[13].
- Visual Studio Code – Microsoft tarafından geliştirilen açık kaynaklı bir geliştirme ortamıdır. Windows, macOS ve Linux gibi işletim sistemlerinde çalışır. Git sürüm kontrol (version control) sistemleriyle çok iyi çalışır. Ayrıca, akıllı kod tamamlamanın dikkate değer özellikleriyle birlikte gelir.
- CLion – Jetbrains tarafından geliştirilmiştir ve C++ programcıları için en çok önerilen çapraz platformlu (CMake derleme sistemiyle entegre macOS, Linux ve Windows'u destekler) geliştirme ortamıdır. Ayrıca, yerel (local) ve uzaktan (remote) desteğe sahip birkaç IDE'den biri olarak kabul edilir. Bu da yerel bir makinede kod yazmanıza ancak uzak sunucularda derlemenize olanak tanır. Kaynak kodlarımız yönetmemizi sağlayan Concurrent Versions System-CVS ve Team Foundation Server-TFS ile entegre edilebilir.
- Eclipse – Java'da yazılmış ve IBM tarafından geliştirilmiş ücretsiz ve açık kaynaklı bir geliştirme ortamıdır. Yaklaşık otuz programlama dilini desteklediği için geniş topluluk desteğiyle iyi bilinir. C++ için Eclipse IDE, kod tamamlama, otomatik kaydetme, derleme ve hata ayıklama desteği, uzak sistem gezgini, statik kod analizi, profilleme (profiling) ve yeniden düzenleme (refactoring) gibi beklenebilecek tüm özelliklere sahiptir. Ayrıca çeşitli harici eklentileri entegre ederek işlevselliğini genişletebilirsiniz. Çok platformludur ve Windows, Linux ve macOS'ta çalışabilir.
- NetBeans – Apache Yazılım Vakfı – Oracle Corporation tarafından geliştirilen ücretsiz ve açık kaynaklı bir geliştirme ortamıdır. Öğrenciler veya başlangıç seviyesindeki C/C++ geliştiricileri için şiddetle tavsiye edilmesinin sebebi, Eclipse'e benzer şekilde daha iyi sürükle ve bırak işlevlerine sahip olmasıdır. Windows, Linux, MacOSX ve Solaris gibi birden fazla platformda çalışır.
- Xcode – MacOSX işletim sisteminde yazılım geliştirmek için kullanılan bir entegre geliştirme ortamıdır.

Entegre geliştirme ortamları metin dosyası olarak tutulan kaynak kodları programcıya renklendirerek gösterir. Bu da programcının kodu anlamasını daha da kolaylaştırır. Ancak kodu dosyaya yine sade metin olarak kaydeder.

Çevrimiçi Derleyiciler

Bütün bu entegre geliştirme ortamlarının yanında online olarak da derleme yapılan web uygulamaları da bulunmaktadır. Bunlardan birkaçı aşağıda verilmiştir;

- https://www.tutorialspoint.com/compile_c_online.php
- https://www.onlinegdb.com/online_c_compiler
- https://www.programiz.com/c-programming/online-compiler/
- https://onecompiler.com/c
- https://www.online-ide.com/online_c_compiler
- https://onlinecompilers.com/online-c-compiler

Derleme Zamanı

Derleyici programının derleme işlemine başlayıp bitirildiği sürece derleme zamanı (compile time) denir. Bu süreç başarısızlıkla da sonuçlanabilir ve eğer derleme işleminde hata meydana gelirse programcı hata mesajları ile uyarılır.

Bir derleyici program, kaynak dosyayı makine diline çevirme çabasında, kaynak dosyanın C dilinin sözdizimi kurallarına uygunluğunu da denetler. Eğer dilin kurallarına uyulmamışsa, derleyici bu durumu bildiren bir ileti de vermek zorundadır.

```
int main()
{
    return
}
```

[13] https://www.programiz.com/c-programming/getting-started

Yukarıda hatalı yazılmış bir c programı derlendiğinde aşağıdaki hata alınacaktır.

```
main.c: In function ‘main’:
main.c:5:1: error: expected expression before ‘}’ token
    5 | }
      | ^
```

Aşağıda derleme uyarısı alınacak, ancak derlemenin tamamlandığı bir kod örneği verilmiştir;

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

Bu durumda verilen derleme uyarıları ve programın çalışması aşağıda verilmiştir.

```
main.c: In function ‘main’:
main.c:4:5: warning: ‘return’ with no value, in function returning non-void
    4 |     return;
      |     ^~~~~~
main.c:1:5: note: declared here
    1 | int main()
      |     ^~~~

...Program finished with exit code 41
Press ENTER to exit console.
```

Kaynak kod içerisindeki metinlerde Türkçe karakter kullanılması halinde Windows altında derlenen programı çalıştırdığınızda aynı metni göremeyebilirsiniz. Bunu düzeltmek için komut satırında

```
C:\Users\ILHANOZKAN>Chcp 65001
```

Komutu verilerek komut satırının UTF-16 karakter kümesi ile işlem yapmasını sağlayabiliriz.

Hatalar

Programcı tarafından kodlama yapılırken genellikle aşağıdaki üç tip hata yapılır;

1. Derleme zamanı hataları (compile time error): Bu tip hatalar genelde kullanılan dilin yazım kurallarına (syntax) uyulmadığından, talimatların (statement) yanlış yazılmasından ya da programcının kod metninde uygun olmayan karakterlerin kullanılmasından kaynaklanır.
2. Koşma zamanı hataları (run time error): Kaynak kod, kurallara uygun olarak yazılmıştır ve herhangi bir yazım hatası bulunmaz. Bu tip hatalara en iyi örneklerden birisi sıfıra bölme hatasıdır. Taşma hataları da bu hatalardandır. Daha sonra değinilecektir.
3. Mantıksal hatalar (logical error): Programcının çözüm için gerekli adımların oluşturulmasında, çözüm yönteminin yanlış olmasından ya da yanlış işleçler (operator) kullanmasından kaynaklanır. Örneğin bir büyüktür (`>`) işleci yerine küçüktür (`<`) işleci kullanıldığında ne bir yazım hatası ne de bir çalışma zamanı hatası ortaya çıkar. Fakat program kendisinden istenilen davranışları yerine getirmez ve uygun çıktıları üretmez. Faktöriyel hesaplanırken `0! = 1` yerine `0! =0` kabul etmek buna örnek verilebilir.

C Programlama Kullanım Alanları

C dili, başlangıçta sistem geliştirme çalışmaları için, özellikle işletim sistemini oluşturan programlar için kullanıldı. Montaj (assembly) dilinde yazılan kod kadar hızlı çalışan kod ürettiği için sistem geliştirme dili olarak benimsendi. Bilgisayar mühendislerinin bilmesi gereken bu dilin birçok kullanım alanı mevcuttur;

- İşletim Sistemleri
- Dil Derleyicileri
- Assembler Derleyiciler
- Metin Düzenleyicileri
- Yazdırma Biriktiricileri

– Ağ Sürücüleri
– Gömülü Programlar
– Veri tabanları

C Programlama Dilinin Üstünlükleri

Verimlilik (efficiency) ve hız (speed): C, yüksek performanslı ve verimli olmasıyla bilinir. Düşük seviyede bellekle çalışmanıza olanak tanır ve donanıma doğrudan erişim sağlar, bu da onu hız ve ekonomik kaynak kullanımı gerektiren uygulamalar için ideal hale getirir.

Taşınabilir (portable): C programları, minimum veya hiç değişiklik yapılmadan farklı platformlarda derlenebilir ve yürütülebilir. Bu taşınabilirlik, dilin standartlaştırılmış olması ve derleyicilerin küresel olarak çeşitli işletim sistemlerinde kullanılabilmesinden kaynaklanmaktadır.

Donanıma Yakınlık: C, göstericiler ve düşük seviyeli işlemler kullanılarak donanımın doğrudan manipüle edilmesine olanak tanır. Bu, onu sistem programlama ve donanım kaynakları üzerinde ayrıntılı denetim gerektiren uygulamalar geliştirmek için uygun hale getirir.

Standart Kütüphaneler: Giriş/çıkış işlemleri, metin işleme ve matematiksel hesaplamalar gibi yaygın görevler için C, geliştiricilerin önceden oluşturulmuş işlevlerden yararlanarak daha verimli kod yazmalarına yardımcı olan büyük bir standart kütüphaneyle birlikte gelir.

Yapısal Programlama: C, kodu modüler ve anlaşılması kolay veri ve kontrol yapılarına sahiptir. Fonksiyonlar, döngüler ve koşullarla geliştiriciler, bakımı kolay net kod üretebilirler.

Yordamlı ya da Emreden Paradigma: C dilinde programlama fonksiyonların birbirini çağırması ile yapılır. Bu programlama bakış açısı yordamlı programlama (procedural programing) olarak adlandırılır. Prosedür (procedure), bir değer üretmeyen fonksiyonlardır. Fonksiyonlar içinde ise bilgisayarın gerçekleştireceği süreç, basit ve anlaşılır talimatlar (statement) halinde art arda yazılır. Talimatların art arda verilmesi emreden programlama (imperative programming) olarak da bilinir.

C Dilinin Zayıf Yönleri

Manuel Bellek Yönetimi: C dili, bir geliştiricinin belleği açıkça ayırma ve ayırmayı kaldırmasıyla ilgilenmesi gereken manuel bellek yönetimine ihtiyaç duyar. Programcı bunu kodu yazarken yapar.

Nesne Yönelimli Olmaması: Günümüzde, programlama dillerinin çoğu nesne yönelimli programlamayı (object oriented programming) özelliklerini destekler. Ancak C dili bunu desteklemez.

Çöp Toplama Olmaması: C dili, bellekte yer ayrılan ve kullanılmayan bileşenleri bellekten otomatik kaldırmaz. Yani çöp toplama (garbage collection) kavramını desteklemez. Programcının belleği manuel olarak ayırması ve kaldırması gerekir ve bu hataya açık olabilir ve bellek sızıntılarına veya verimsiz bellek kullanımına yol açabilir.

İstisna İşleme Olmaması: C dili, istisnaları (exception) ya da hata ayıklama (error handling) işlemek için herhangi bir kütüphane sağlamaz. Programcının her türlü beklentiyi işlemek için kod yazması gerekir.

C Sürümlerinin Tarihçesi

1971 yılında *Dennis Ritchie,* C üzerinde çalışmaya başladı ve Bell Labs'daki diğer geliştiricilerle birlikte C'yi geliştirmeye devam etti. Geleneksel C'den sonraki C sürümlerinin tarihi aşağıda verilmiştir[14].

K&R C – *Dennis Ritchie*, *Brian Kernighan* ile birlikte "The C Programming Language" adlı kitaplarının ilk baskısını 1978 yılında yayınladı. Halk arasında K&R olarak bilinen kitap, uzun yıllar boyunca dilin gayri resmi bir kılavuzu olarak hizmet etti. Açıkladığı C sürümüne genellikle "K&R C" denir ve ayrıca C78 olarak da adlandırılır.

[14] https://www.tutorialspoint.com/cprogramming/c_history.htm

K&R C'de tanıtılan C dilinin birçok özelliği, 2018'de onaylanan dilin bir parçasıdır. C'nin erken sürümlerinde, yalnızca `int` dışındaki türleri döndüren fonksiyonlar, fonksiyon tanımı öncesinde kullanılıyorsa bildirilmelidir; önceden bildirilmeden kullanılan fonksiyonların `int` türünü döndürdüğü varsayılırdı.

AT&T ve diğer satıcılar tarafından üretilen C derleyicileri, K&R C diline eklenen çeşitli özellikleri destekledi. C popülerlik kazanmaya başlasa da farklılıklar problem yaşatmaya başladı. Bu nedenle, dil özelliklerinin standartlaştırılması gerektiği düşünüldü.

ANSI C – 1980'lerde, Amerikan Ulusal Standartlar Enstitüsü (ANSI), C dili için resmi bir standart üzerinde çalışmaya başladı. Bu, 1989'da standartlaştırılan ANSI C'nin geliştirilmesine yol açtı. ANSI C, birkaç yeni özellik tanıttı ve dilin önceki sürümlerinde bulunan belirsizlikleri giderdi.

C89/C90 – ANSI C standardı uluslararası olarak kabul edildi ve C89 (veya onaylanma yılına bağlı olarak C90) olarak tanındı. Uzun yıllar boyunca derleyiciler ve geliştirme araçları için temel teşkil etti.

C99 – 1999'da ISO/IEC, C99 olarak bilinen C standardının güncellenmiş bir sürümünü onayladı. C standardı 1990'ların sonlarında daha da revize edildi.

C99, satır içi fonksiyonlar (inline function), karmaşık sayıları temsil eden karmaşık bir tür gibi çeşitli yeni veri türleri ve değişken uzunluklu diziler vb. gibi yeni özellikler sundu. Ayrıca // ile başlayan C++ tarzı tek satırlık yorumlar için destek ekledi.

C11 – 2011'de yayınlanan C11, C standardının bir başka önemli revizyonudur. C11 standardı, C'ye ve kütüphaneye yeni özellikler ekler ve çoklu iş parçacığı (multi threading) desteği, anonim yapılar (anonym structure) ve birleşimler ve geliştirilmiş Unicode desteği gibi özellikler sunar.

Tür genel makroları, anonim yapılar, geliştirilmiş Unicode desteği, atomik işlemler, çoklu iş parçacığı ve sınır denetimli işlevler içerir. C++ ile geliştirilmiş bir uyumluluğu vardır.

C17 – C17 standardı Haziran 2018'de yayınlanmıştır. C17, C programlama dili için geçerli standarttır. Bu standart revizyonuyla yeni özellikler getirilmemiştir. Sadece belirli teknik düzeltmeler ve C11'deki kusurlara yönelik açıklamalar yapar.

C18 – C standardının en son sürümü olan C18, 2018'de yayınlandı. C11'e kıyasla küçük revizyonlar ve hata düzeltmeleri içeriyor.

C23 – C23, 2024'te yayınlanması beklenen bir sonraki büyük C dili standardı revizyonunun gayri resmi adıdır. Bu revizyonda 14 yeni anahtar kelimenin tanıtılması bekleniyor.

C, basitliği, verimliliği ve çok yönlülüğü nedeniyle zamanla popülerliğini korudu. İşletim sistemleri, gömülü sistemler, uygulamalar ve oyunlar dahil olmak üzere çeşitli yazılımlar oluşturmak için kullanıldı. C'nin sözdizimi ve semantiği, C++, Java ve Python gibi farklı modern programlama dillerini de etkilemiştir.

C PROGRAMLAMA DİLİNE GİRİŞ

En Basit C Programı

C programlama dili, her programcının ihtiyacını görecek genel amaçlı, emreden paradigmalı (kısa ve öz talimatlar birbiri ardına verilerek yapılan programlama), yapısal programlama dilidir. Yapısal programlamanın çerçevesi *TEMEL YAZILIM KAVRAMLARI* başlığı altında anlatılmıştır.

FORTRAN II dilinde talimatların yedinci sütunda başlaması gerekir. Pascal dili gibi C dilinde de talimatlar gelişigüzel (free format) yazılabilir.

Yapısal programlamada programın başladığı yeri belirten bir ana fonksiyon (main function) tanımlanır. C dilinde bu fonksiyon `main()` fonksiyonudur. Yani içinde main fonksiyonu yazılmamış bir c programı çalışmaz. Aşağıda en basit C programı verilmiştir.

```
int main()
{

   return 0;
}
```

C dilinde her fonksiyon tırnaklı parantez karakteri ile başlayan ve biten bir kod bloğuna sahiptir ve fonksiyona ait bu blok, fonksiyon bloğu (function block) olarak adlandırılır. Ana fonksiyonunun başındaki `int`, ana fonksiyonun bir tamsayı (integer) geri döndüreceğini belirtir. Ana fonksiyon içinde kullanılan `return 0;` talimatı (statement) main fonksiyonundan sıfır geri döndürerek çıkıldığını söyleyen talimattır. Ana fonksiyondan çıkıldığında programın sonlanıp işletim sistemine dönüleceğinden işletim sistemine `0` sayısını gönderir. Sıfır dışında bir değerin işletim sistemine gönderilmesi, programın hata ile sonlandığı olarak kabul edilir. Yukarıdaki program derlendiğinde aşağıdaki çıktı elde edilir.

```
...Program finished with exit code 0
```

Bu durumda işletim sistemine 11 sayısını gönderen program aşağıdaki şekilde yazılır;

```
int main()
{

   return 11;
}
```

Program çalıştırıldığında aşağıdaki çıktı elde edilir;

```
...Program finished with exit code 11
```

C Programlama Dili Söz Dizim Kuralları

C dili için oluşturulacak kaynak kod programcı tarafından metin dosyasına yazılırken belli söz dizim kullarına (syntax) uyar. Bunlar aşağıda başlıklar halinde verilmiştir.

Kaynak Kodumuzu Oluşturacak Karakterler

1. İngiliz alfabesindeki büyük harfler:
   `A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`
2. İngiliz Alfabesindeki küçük harfler:
   `a b c d e f g h i j k l m n o p q r s t u v w x y z`
3. Rakamlar:
   `0 1 2 3 4 5 6 7 8 9`
4. Özel Karakterler:
   `< > . , _ ( ) ; $ : % [ ] # ? ' & { } " ^ ! * / | - \ ~ +`
5. Metinde görünmeyen ancak metni biçimlendiren beyaz boşluk (white space) karakterleri:
   a. Boşluk (space)

b. ↲ Yeni satır (new line)
c. ⇤ Satır başı (carriage return)
d. ⇦ Geri (backspace)
e. ⇥ Sekme (tab)

Talimatlar ve Anahtar Kelimeler

Yüksek seviyeli bir dilde yazılmış bir talimat (statement), işlemciye belirtilen bir eylemi gerçekleştirmesini söyleyen cümledir. Yüksek seviyeli bir dildeki tek bir talimat, birkaç makine dili emir kodunu (instruction code) temsil edebilir. Talimatlar, yapılaması gerekeni kısa ve net olarak belirten mantıksal satırlardır (logical sequence).

Anahtar kelimeler (keywords), programlamada kullanılan ve C derleyicisi tarafından özel anlama sahip, önceden tanımlanmış, ayrılmış (reserved) kelimeler olup programı oluşturan talimatlar (statements) bu kelimeler kullanılarak yazılırlar.

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

Tablo 6. Anahtar Kelimeler

C dilinde;
- Anahtar kelimeler, küçük-büyük harf ayrımı yapıldığından her zaman küçük harflerle yazılırlar.
- Her talimat (statement), anahtar kelimeler içerecek şekilde yazılır ve noktalı virgül karakteri ile biter.

Açıklamalar

Talimatlar (statement), yapılaması gerekeni kısa ve net olarak belirten mantıksal satırlardır (logical sequence). Programcı kodu yazarken açıklama (comment) yapma gereği duyabilir. Bunu iki şekilde yapar;

1. Kod metninde bulunulan yerden sonra satırın sonuna kadar olan bir açıklama yapılmak istendiğinde, açıklama öncesinde iki adet bölü karakteri kullanılır.
2. Eğer birden çok satırı içeren bir açıklama yapılacak ise açıklama `/*` ile `*/` karakterleri arasına alınır.

Aşağıda açıklama eklenmiş bir ana fonksiyon örneği bulunmaktadır.

```
/*
  Bu program, açıklama (comment) içeren basit bir ana fonksiyonu olan C
  programıdır. İlhan ÖZKAN, Aralık 2024
*/
int main() // Ana Fonksiyon
{ //fonksiyon bloğu başlangıcı

    /*
     Ana Fonksiyon, programın başladığı yerdir.
     Ana fonksiyonu olmayan bir program çalışmaz.
    */

    return 0; /*fonksiyondan dönüş talimatı noktalı virgül karakteri ile bitmiştir.*/
} //fonksiyon bloğu bitişi
```

Kodun bir başkası tarafından bakımı yapılacağı göz önüne alınarak okunaklı yazılması çok önemlidir. Bu nedenle kodu yazarken her kod bloğu içindeki açıklama ve talimatları bloktan itibaren girintili olarak yazarız.

```
int main()
{
    //Ana fonksiyon bloğuna ait girinti
    {
        //iç bloğa ait girinti...
        return 0;
    }
    return 0;
}
```

Kodu bu şekilde yazmamızın derleyici açısından hiçbir önemi yoktur. Derlemenin ilk aşamasında bütün bu açıklamalar ve beyaz boşluk karakterleri metinden çıkartılır. Aşağıda en kısa C programı verilmiştir.

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

Değişken Tanımlama

Her yapısal programlama dilinde olduğu gibi C dilinde de veri yapıları (data structure) ve kontrol yapıları (control structure) birbirinden ayrıdır. Dolayısıyla veriyi işleyecek kontrol yapısı kodlanmadan önce veriyi taşıyan veri yapıları tanımlanmalıdır.

Değişkenler veri yapılarının temel öğeleridir. Değişkenin (variable) ne olduğu *Değişken* başlığı altında verilmiştir. Değişkenler, belleğin belli bir bölgesini işgal eder ve fiziki olarak bellek bitlerden oluştuğundan ikili sayı (binary) sisteminde veri tutarlar. C dilinde verinin bellekte kapladığı yer ve biçimi için kullanılan ilkel veri tipleri (primitive data type) için anahtar kelimeler aşağıda verilmiştir;

1. Tamsayı olarak kullanılacak veri tipleri
   a. `char` bellekte 1 bayt yani 8 bit yer kaplayan işaretli tamsayıları tutan veri tipidir.
   b. `short` bellekte 2 bayt yani 16 bit yer kaplayan işaretli tamsayıları tutan veri tipidir.
   c. `int` bellekte 4 bayt yani 32 bit yer kaplayan işaretli tamsayıları tutan veri tipidir.
   d. `long` bellekte 8 bayt yani 64 bit yer kaplayan işaretli tamsayıları tutan tipidir.
2. Gerçek sayı veri tipleri
   a. `float` bellekte 4 bayt yani 32 bit yer kaplayan tek hassasiyetli (single precision) kayan noktalı gerçek sayıları tutan veri tipidir.
   b. `double` bellekte 8 bayt yani 64 bit yer kaplayan çift hassasiyetli (double precision) kayan noktalı gerçek sayıları tutan veri tipidir.

C programlama dilinde hiçbir değeri olmayan ve bellekte yer kaplamayan `void` veri tipi vardır. Bu veri tipini hiçbir değer döndürmeyen fonksiyon tanımlamalarında ileride çokça göreceğiz.

Değişkenlerin veri tipine bağlı olarak kimlik verilerek tanımlanması (definition) gerekir. Değişken tanımlaması yapılabilmesi için ilk önce veri tipinin (data type) belirlenmesi ve değişkenin kullanılacağı yerler göz önüne alınarak benzersiz bir kimlik (identifier) verilmesi gerekir.

```
/* Bu program değişken tanımlamak için oluşturulmuştur.*/
int main()
{
    char yas; /* veri tipi "char" ve kimliği "yas" olan değişken tanımlandı*/
    int kat;  /* veri tipi "int" ve kimliği "kat" olan değişken tanımlandı */
    float kilo; /* veri tipi "float" ve kimliği "kilo" olan değişken tanımlandı */
    return 0;
}
```

Yukarıda tanımlaması yapılan `yas`, `kat` ve `kilo` değişkenleri ana fonksiyon olan `main` fonksiyon bloğu içinde tanımlandığından, tanımlanan değişkenler sadece bu blok içinde geçerlidir. Aynı blok içinde bu değişken kimlikleri bir başka değişkene verilemez.

Bildirim (declaration) derleyiciye böyle bir değişken, fonksiyon veya nesne var demenin adıdır. Tanımlama (definition) derlemenin sorunsuz olabilecek şekilde eksiksiz yapılan bir işlemdir.

İşaretli bir tamsayı kullanmak yerine en anlamlı işaret bitini de sayı kabul ederek işaretsiz olarak tamsayıları kullanabiliriz. Yukarıda verilen `char`, `short`, `int` ve `long` tamsayı veri tipleri aslında işaretli

tamsayılardır. Yani önlerinde `signed` sıfatı olduğu varsayılır. Varsayılan (default) olarak bu sıfat varmış diye kabul edilir. Değişkenleri işaretsiz olarak kullanmak istememiz halinde `unsigned` sıfatını tamsayı veri tipinin önünde kullanırız.

```
/* Bu program işaretsiz tamsayı değişken tanımlamak için oluşturulmuştur.*/
int main()
{
    unsigned yas; /* veri tipi "unsigned char" ve kimliği "yas" değişken tanımlandı.
                     yas değişkeni 0 ile 255 arasında bir değer alabilir. */
    unsigned short kat; /* veri tipi "unsigned short" ve kimliği "kat" olan değişken
                           tanımlandı.kat değişkeni 0 ile 65535 arasında değer alabilen
                           bir değişkendir. */
    unsigned int pozitifTamsayi; /* pozitifTamsayi değişkeni 0 ile 4.294.967.295 arasında
                                    değer alabilen bir değişkendir. */
    return 0;
}
```

Değişkenin kapsamı (variable scope); Değişkenin bildiriminin (variable declaration) yapılabileceği, değişkene erişilebileceği (access), değişkenin üzerinde çalışılabileceği program kodu olarak tanımlanır. Değişkenin kapsamı, tanımlandığı kod bloğu (ve varsa bu blok içindeki iç bloklar) ile sınırlıdır.

Değişken Kimliklendirme Kuralları

C programlama dilinde değişkenlerin kimliklendirilmesinde (identifier definition) aşağıda verilen kurallara uyulur.

1. Kimliklendirmede anahtar kelimeler kullanılamaz!
2. Kimliklendirme rakamla başlayamaz!
3. Kimlikler en fazla 32 karakter olmalıdır! Daha fazlası derleyici tarafından dikkate alınmaz!
4. Kimliklendirmede ancak İngiliz Alfabesindeki Büyük ve Küçük harfler, rakamlar ile altçizgi karakteri kullanılabilir.

```
/* Bu program değişken tanımlama örneklerini içerir. */
int main()
{
    /* Tamsayı değişkenler: */
    char yas;
    int tam_sayi;
    long uzun_tam_sayi;

    /* Gerçek Sayı Değişkenler:*/
    float kilo;
    double reel_sayi;

    /*
       Aynı veri tipinde birden fazla değişkene
       aralarına virgül koyarak kimlik verilebilir.
    */
    int kat1,kat2,kat3;
    float olcek1,olcek2;
    return 0;
}
```

Bütün bu kuralların yanında değişkenlere kimlik verilirken yazılan kodun okunaklılığını artırmak için camel case olarak kimlik verilir. Bu kimliklendirme türünde ilk sözcük tamamıyla küçük, izleyen sözcüklerin ise baş harfi büyük yazılır. Bu kural zorunlu değildir ama koda bakım yapacak sonraki programcıların işini kolaylaştırmak için ahlaki olarak tercih edilir.

```
/* Bu program camelCASE değişken kimliklendirme örneklerini içerir.*/
int main()
{
    int sayac;
```

```
    int ogrenciBoyu;
    int asansorunOlduguKat;
    float asansorAgirligi;
    int ogrenciYasi;
    char ilkHarf;
    char ogrenciAdi[50];
    char ogrenciSoyadi[50];
    float ortalamaNot;
    return 0;
}
```

Sabitler

Programcı, kodlama yaparken bazı sabitleri (constants) ister istemez kullanır. Bunların biri değişmez (literal), diğeri ise değeri değiştirilemeyen sabit değişkenlerdir (const variable). Derleme öncesinde de sonrasında da kaynak koddakinin kelimesi kelimesine aynısı olan değişmezler (literal) kullanılır. Değişkenlere verilen ilk değerler (initial value) ile katsayılar (coefficient) en çok kullanılan değişmezlerdir. Değişkenler tanımlanırken sahip olacağı ilk değerleri belirten sabitler de verilebilir. Aşağıda bu değişmezlere ilişkin örnek verilmiştir.

```
/* Bu program değişkenlere ilk değer (initial value) vermede kullanılan
   için değişmez (literal) örneklerini içerir. */
int main()
{
    int i,j=0; /* "int" veri tipindeki "j" kimlikli değişkene
                  0 tamsayı değişmezi ile ilk değer verildi */

    char c=65; /* "char" veri tipindeki "c" kimlikli deki değişkene
                  65 tamsayı değişmezi ile ilk değer verildi */

    float f=2.5; /* "float" veri tipindeki "f" kimlikli değişkene
                    Türkçe 2,5 olan kayan noktalı değişmezi ile
                    ilk değer verildi.
                    Kod İngilizce yazıldığından, kodun içerisindeki
                    gerçek sayı değişmezleri için ondalık ayracı
                    NOKTA olarak yazılır.*/

    char harf='A'; /* "char" veri tipindeki "harf" kimlikli değişkene
                      'A' karakter değişmezi ile ilk değer verildi.
                      Klavyeden basılan her bir harf de bellekte sayı
                      olarak tutulur.
                      'A' karakterinin kodu 65,
                      'B' karakterinin kodu 66'dır... Bu ANSI Standardıdır.
                      BU nedenle bu talimat aşağıdaki şekilde de yazılabilir;
                      char harf=65;
                      Bellekte 1 bayt yer kaplayan "char" 255 farklı
                      karakteri gösterebilir.
                      Kodu 0 olan karakter ise NULL karakter olarak
                      adlandırılır.
                      Günümüzde ANSI yerine UTF-8 ve UTF-16 karakter
                      kodları kullanılmaktadır. */

    return 0;
}
```

Aşağıdaki kod örneğinde programcı, `3`, `3.14`, `3.14` ve `2.0` olmak üzere dört farklı değişmez kullanılmıştır. Bu değişmezlerin her biri derleyici tarafından farklı olarak değerlendirilir.

```
/* Bu program, değişmez örneğidir. */
int main() {
   int yariCap=3;
   float daireninAlani=3.14*yariCap*yariCap;
```

```
   float daireninCevresi=2.0*3.14*yariCap;
   return 0;
}
```

Değişmezlerin (literal) çokça kullanılması derleme sonrası üretilen icra edilebilir makine kodunu da büyütür. Bunun yerine çok kullanılan değişmezler bellekte 1 kez yer kaplasın diye const sıfatıyla (const qualifier) sabit değişkenler (const variable) tanımlanabilir. Sabit değişkenler etik olarak büyük harfle kimliklendirilir.

```
/* Bu program, const sıfatı örneğidir. */
int main() {
   int yariCap=3;
   const float PI=3.14;
   float daireninAlani=PI*yariCap*yariCap;
   float daireninCevresi=2.0*PI*yariCap;
   return 0;
}
```

Böylece aynı sabit PI değişkeni birden çok yerde kullanılır ve her seferinde aynı bellek bölgesine erişildiğinden bellekten tasarruf edilmiş olunur. Tanımlama (definition) sırasında const sıfatı kullanılan değişkene ilk değer (initial value) değişmez (literal) olarak verilmelidir. Daha sonra bu değişkene bir değer ataması olamaz!

Bir değişkeni sabit tanımlamanın üstünlükleri aşağıda sıralanmıştır;

- Gelişmiş Kod Okunabilirliği: Bir değişkene const sıfatı vermek, diğer programcılara değerinin değiştirilmemesi gerektiğini belirtir; bu da kodun anlaşılmasını ve sürdürülmesini kolaylaştırır.
- Gelişmiş Veri Tipi Güvenliği: const kullanılması, değerlerin yanlışlıkla değiştirilmemesini sağlayabilir ve kodumuzda hata ve eksiklik olma olasılığını azaltabilir.
- Gelişmiş Optimizasyon: Derleyiciler, değerlerinin program yürütme sırasında değişmeyeceğini bildikleri için const değişkenlerini daha etkili bir şekilde optimize edebilirler. Bu, daha hızlı ve daha verimli kodla sonuçlanabilir.
- Daha İyi Bellek Kullanımı: Değişkenlere const sıfatı vermek, değerlerinin bir kopyasını oluşturmayı engeller; bu da bellek kullanımını azaltabilir ve performansı artırabilir.
- Gelişmiş Uyumluluk: Değişkenlere const sıfatı vermek, kodunuzu const değişkenleri kullanan diğer başlıklarla daha uyumlu hale getirebilir.
- Gelişmiş Güvenilirlik: const kullanarak, değerlerin beklenmedik şekilde değiştirilmemesini sağlayarak kodunuzu daha güvenilir hale getirebilir ve kodunuzdaki hata ve eksiklik riskini azaltabilirsiniz.

Kod içerisinde değişmezleri (literal) tekrar tekrar yazmamanın bir yolu da `#define` ön işlemci yönergesi (preprocessor directive) ile makro tanımlamaktır. Ön işlemciler, kaynak kodları derlemeden önce derleyiciyi yönlendirerek kaynak kodlar üzerinde işlem yapmayı sağlar. Aşağıdaki örnekte derleyici, derlemeye geçmeden `PI` görülen yerleri `3.14` olarak değiştirmesi söylenir. Derleme bu ön işlemci işleminden sonra gerçekleşir. Değişmez değerleri değiştiren makro kimlikleri büyük harfle kimliklendirilir. Makrolar sonunda talimatlar gibi noktalı virgül karakteri ile bitmez! Çünkü ön işlemci yönergeleri kodların tasnifi ve derleme ön hazırlığını yaparlar.

```
/* Bu program, #define ön işlemci yönergesi örneğidir. */

#define PI 3.14  /* Bu makro, derleme öncesinde PI görülen yerlere
                    3.14 yazılmasını sağlar. */
int main() {
   int yariCap=3;
   float daireninAlani=PI*yariCap*yariCap;
   float daireninCevresi=2.0*PI*yariCap;
   return 0;
}
```

Ön işlemci yönergesini yerine getiren derleyici aynı kodu aşağıdaki şekle çevirerek derleme işlemine geçer. Ayrıca ön işlemci yönergeleri yerine getirilmeden önce bütün açıklamalar ve beyaz boşluklar (white space) kaynak koddan kaldırılır. Bu durumda ön işlemci yönergeleri sonrasında kaynak kodumuz aşağıdaki hale döner;

```
int main() {
   int yariCap=3;
   float daireninAlani=3.14*yariCap*yariCap;
   float daireninCevresi=2.0*3.14*yariCap;
   return 0;
}
```

`#define` ön işlemci yönergesi çok tekrarlanan değişmezler için kullanılabileceği gibi aşağıdaki şekilde de kullanılabilir;

```
/* Bu program, #define ön işlemci yönergesinin bir başka
   kullanımı örneğidir. */
#define BEGIN int main() {
#define END }
#define PI 3.14
BEGIN
   const char ILKHARF='A';
   const float AVAGADROSAYISI=6.022e23;
   float yarimRadyan=PI/2;
   return 0;
END
```

Yazdığımız Kod Nasıl İcra Edilir?

Derleyici tarafından işlemcinin anlayacağı makine diline çevrilen programımız aslında yazdığımız talimatları (statement) sırasıyla çalıştırır. C dilinde her talimat noktalı virgül ile biter.

		boy	kilo	bki	boyKare
	int main() {				
1	float boy=1.80;	1.80	?	?	?
2	float kilo=100.0;	1.80	100.0	?	?
3	float bki;	1.80	100.0	?	?
4	float boyKare;	1.80	100.0	?	?
5	boyKare=boy*boy;	1.80	100.0	?	3.24
6	bki=kilo/boyKare;	1.80	100.0	30.86419753	3.24
7	boy=1.50;	1.50	100.0	30.86419753	3.24
8	return 0;	1.50	100.0	30.86419753	3.24
	}				

Tablo 7. Örnek bir C Programının İcra Sırası

Yapısal programlamada program, ana fonksiyondan başlar. Bu nedenle icra, `main` fonksiyonu içindeki ilk talimatla başlar ve solda verilen sırayla icra edilir. Sağda ise her icra sonrası değişkenlerin değerleri gösterilmiştir. Her talimatın icrasında neler olduğu aşağıda verilmiştir;

1. `boy` değişkenine `1.80` değeri atanır.
2. `kilo` değişkenine `100.0` atanır.
3. `bki` değişkeni tanımlanır.
4. `boyKare` değişkeni tanımlanır. Daha sonraki birkaç satır açıklama olduğundan icra edilecek bir şey yoktur.
5. `boy` ile `boy` değişkeni çarpılır ve sonuç `boyKare` değişkenine atanır. Artık `boykare` değişkeninin değeri bu noktadan sonra `3.24` olmuştur.
6. `kilo` değişkeni `boyKare` değişkenine bölünür ve sonuç `bki` değişkenine atanır. Artık `bki` değişkeninin değeri bu noktadan sonra `30.86419753` olmuştur.
7. `boy` değişkenine `1.50` atanmıştır. Bu atama sonrası `boyKare` ve `bki` değişkenlerinin değeri değişmemiştir. Çünkü bu değişkenleri değiştiren 5. ve 6. talimatlar icra edilmiştir.
8. Programdan çıkılırken işletim sitemine `0` geri döndürülür.

İşleçler

İşleçler (operator) matematikten bildiğimiz işleçlerdir.

$$y = 3 + 2$$

Yukarıda verilen cebirsel ifadede toplama işleci (`+`) iki tarafına bulunan argümanları toplar. Bu argümanlara işlenen (operand) adı verilir. Yüksek düzey dillerde de aritmetik işleçler ve bunun yanında birçok işleç de bulunur. En çok kullanılan işleç, atama (`=`) işlecidir. Atama işleci, sağdaki işleneni soldakine atar. Bu nedenle kodlama yapılırken `2+2=x` veya `a+b=x` yazılamaz!

Aritmetik İşleçler

C programlama dilinde aritmetik işleçlerin çalışma şekli işlenenlerin (operand) veri tipine göre değişir. Bu konu detaylı olarak *Üstü Kapalı Tip Dönüşümleri* başlığında detaylı anlatılmıştır. Aritmetik işlemler ve atama işleci için kurallar;

1. İşlenenlerden birinin bellekte kapladığı yer küçük ise ikisi aynı olacak şekilde bellekte daha fazla yer kaplayanına dönüştürülür ve işlem sonra yapılır ve sonuç dönüştürülen veri tipinde üretilir. Örneğin `char` veri tipindeki değişken ile `long` veri tipindeki bir değişken toplanmaya çalışıldığı zaman `char` veri tipindeki değişkenin değeri `long` veri tipine otomatik dönüştürülür. Sonuç `long` veri tipinde üretilir.
2. İşlenenlerden biri tamsayı diğeri kayan noktalı sayı ise işlem yapılmadan önce işlenenler kayan noktalı sayıya dönüştürülür. Örneğin `int` veri tipindeki değişken ile `float` veri tipindeki bir değişken toplanmaya çalışıldığı zaman `int` veri tipindeki değişkenin değeri `float` veri tipine otomatik dönüştürülür. Sonuç `float` veri tipinde üretilir.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
`+`	Sol ve sağındaki yer alan işlenenleri toplar.
`-`	Soldaki işlenenden sağdakini çıkarır.
`*`	Sol ve sağındaki yer alan işlenenleri çarpar.
`/`	Soldaki işleneni sağdakine böler. Eğer işlenenlerin her iki de tamsayı ise sadece kalan kısmı atılır. Bu durumda son uç yine tamsayı olarak üretilir.
`%`	Kalan (modulus) işleci, sadece tamsayıları işlenen olarak kabul eder. Soldaki işlenenin sağdakine bölümünden kalanı verir. Kalan yine tamsayıdır.

Tablo 8. Aritmetik İşleçler

Aritmetik işleçlere (aritmetic operator) ilişkin yukarıda belirtilen kuralların çalıştığını aşağıdaki örnekte görebilirsiniz.

```
/* Bu program, aritmetik işleç örneğidir. */
int main() {
   int a = 9, b = 4, res;
   float fa=6.6, fb= 2.2, fres;
   res = a + b; // işlem sonucu res=13
   res = a - b; // işlem sonucu res=5
   res = a * b; // işlem sonucu res=36
   res = a / b; // işlem sonucu res=9/4=2 tam 1/4=2
   /*
    toplama işlecinin işlenenleri tamsayı olduğundan,
    bölme sonucundaki tam kısım bölme sonucunu verir.
   */
   res = a / 4.3 ; /* işlem sonucu res tamsayı olduğundan;
                      res=9/4.3=9.0/4.3=2.0930= 2 tam 0.0930=2 */

   res = a % b;   // işlem sonucu res=9%4= 2 tam 1/4=1
   fres= fa / fb; // işlem sonucu res=6.6/2.2=3.00
   return 0;
}
```

C programlama dilinde tamsayı bölme işlemi işlemcinin en basit yetenekleriyle çözülür. Bu nedenle C ve C++ diğer programlama dillerinden ayrılır. Aşağıda buna ilişkin örnek verilmiştir.

```
/* Bu program, aritmetik işleç bölme örneğidir. */
int main () {
   int a = 19 / 2 ; /* a = 2*9+1 = 9 */
   int b = 18 / 2 ; /* b = 9 */
   int c = 255 / 4; /* c = 4*63+3 = 63 */
   int d = 45 / 4 ; /* d = 4*11+1 = 11 */

   double aa = 19 / 2.0 ; /* aa = 9.5 */
   double bb = 18.0 / 2 ; /* bb = 9.0 */
   double cc = 255 / 2.0; /* cc = 127.5 */
   double dd = 45.0 / 4 ; /* dd = 11.25 */

   double ee = 45 / 4 ;   /* ee = 11 çünkü bölme
                             tamsayılar arasında yapılmıştır */
   return 0;
}
```

Tekli İşleçler

Tekli işleçler (unary operator) sadece bir işlenen üzerinde işlem yapar. Tekli demek tek bir işlenen ile işlem yapması anlamındadır.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
-	Kendisinden sonra gelen işlenenin işaretini değiştirir. İşlenen pozitif ise negatif yapar, negatif ise pozitif yapar.
+	Kendisinden sonra gelen işlenenin işaretini pozitif yapar.
++	Artırım (increment) işleci, işlenenin sağında veya solunda kullanılabilir. İşlenen solda ise işlenenin değeri kullanılmadan önce 1 artırılacaktır (pre-increment). İşlenen sağda ise işlenenin değeri kullanılmadan sonra 1 artırılacaktır (post-increment).
--	Eksiltme (decrement) işleci, işlenenin sağında veya solunda kullanılabilir. İşlenen solda ise işlenenin değeri kullanılmadan önce 1 eksiltilecektir (pre-decrement). İşlenen sağda ise işlenenin değeri kullanılmadan sonra 1 eksiltilecektir (post-decrement).
!	Mantıksal DEĞİL işleci işlenenden önce kullanılır. İşleneninin mantıksal durumunu tersine çevirmek için kullanılır.
sizeof()	`sizeof()` işleci kendisinden sonra gelen işlenenin bellek boyutunu bayt cinsinden döndürür.
&	Adres işleci (address operator), kendisinden sonra gelen işlenenin bellek adresini döndürür.

Tablo 9. Tekli İşleçler

Aşağıda tekli işleçlerin kullanımına ilişkin örnek programı lütfen inceleyiniz.

```
int main() {
   int a = 5, b = 5, res;

   res= -a; // tekli eksi (unary minus): res=-5;

   res = ++a+2; /* ++a işlemi önce-artırım (pre-increment) olarak
                   adlandırılır. a değişkeni işleme girmeden önce
                   artırılır. İşlem sonucu res=8, a=6 */
   res = 2+b++; /* b++ işlemi sonra-artırım (Post-Increment) olarak
                   adlandırılır. b değişkeni işleme girdikten sonra
                   artırılır. İşlem sonucu res=7, b=6 */

   a=5; b=5;    /* iki talimat(statement) yan yana yazılmıştır.
                   Daha fazla da yazılabilir. */
   res = --a+2; /* Pre-Decrement: işlemden sonra res=6, a=4 */

   res = 2+b--; /* Post-Decrement: işlemden sonra res=7, b=4 */
```

```
   a=1; b=0;
   res=!a; // işlem sonucu res=0
   res=!b; // işlem sonucu res=1

   res=sizeof(char); /* char veri tipinin bellek miktarını öğrenme: işlem sonucu res=1 */
   res=sizeof a;     /* a değişkeninin bellek miktarını öğrenme: işlem sonucu res=4 */
   return 0;
}
```

İlişkisel İşleçler

İlişkisel işleçler (relational operator), işlenenlerin arasındaki ilişkiyi verirler. C programlama dilinde, yalnızca doğru (true) ya da yanlış (false) değer tutabilen mantıksal veri tipi (data type) bulunmaz. Programcı kodlama yaparken tamsayı veri tiplerini bunun için kullanır. Tamsayının değeri `0` değilse YANLIŞ, sıfırdan farklı ise DOĞRU olarak anlamlandırılır.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
==	Eşit mi? İşleci: İki işlenenin değeri birbirine eşit ise 1, değilse 0 verir
!=	Farklı mı? İşleci: İki işlenenin değeri birbirinden farklı ise 1, değilse 0 verir
>	Büyük mü? İşleci: soldaki işlenenin değeri soldakinden fazla ise 1, değilse 0 verir
<	Küçük mü? İşleci: soldaki işlenenin değeri soldakinden az ise 1, değilse 0 verir
>=	Büyük veya eşit mi? İşleci: soldaki işlenenin değeri soldakinden fazla ya da iki işlenen eşit ise 1, değilse 0 verir
<=	Küçük veya eşit mi? İşleci: soldaki işlenenin değeri soldakinden az ya da iki işlenen eşit ise 1, değilse 0 verir

Tablo 10. İlişkisel İşleçler

Aşağıda verilen ilişkisel işleçlerin kullanımına ilişkin örnek programı lütfen inceleyiniz.

```
int main() {
   int a = 5, b = 6, res;

   res = a<b;  // işlem sonucu res=1
   res = a<=b; // işlem sonucu res=1
   res = a>b;  // işlem sonucu res=0
   res = a>=b; // işlem sonucu res=0
   res = a==b; // işlem sonucu res=0
   res = a!=b; // işlem sonucu res=1
   return 0;
}
```

Bit Düzeyi İşleçler

Bit düzeyi işleçler (bitwise operator) özellikle ikilik tabandaki tamsayılarda karşılıklı bitler üzerinde yapılan işlemlerdir. Aslında bu işlemler çarpma, toplama, bölme ve çıkarma işlemleri için kullanılır.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
&	Bit düzeyi VE: Tamsayının karşılıklı bitleri VE işlemine tabi tutulur. Sonuç tamsayıda karşılaştırılan iki bit 1 ise sonuç bit 1, değilse 0 olacak şekilde işlem yapar.
|	Bit düzeyi VEYA: Tamsayının karşılıklı bitleri VEYA işlemine tabi tutulur. Sonuç tamsayıda karşılaştırılan iki bitin herhangi biri 1 ise sonuç bit 1, değilse 0 olacak şekilde işlem yapar.
^	Bit düzeyi ÖZEL VEYA: Tamsayının karşılıklı bitleri ÖZEL VEYA işlemine tabi tutulur. Sonuç tamsayıda karşılaştırılan iki bitin her ikisi farklı ise sonuç bit 1, değilse 0 olacak şekilde işlem yapar.
~	Bit düzeyi NOT: Tekli işleç olup sonrasında gelecek sayının bitlerini 1 ise 0, 0 ise 1 olacak olacak şekilde işlem yapar.
<<	Bit düzeyi sola kaydırma: Tekli işleç olup sonrasında gelecek sayının bitlerini bir bit sola kaydırır. En anlamlı bit taşar. Taşan bu bit 1 ise işlemcinin durum kaydedicisinde taşma bayrağı 1 olur.

>>	Bit düzeyi sağa kaydırma: Tekli işleç olup sonrasında gelecek sayının bitlerini bir bit sağa kaydırır. En anlamsız bit taşar. Taşan bu bit 1 ise işlemcinin durum kaydedicisinde taşma bayrağı 1 olur.

Tablo 11. Bit Düzeyi İşleçler

Aşağıda tamsayılar üzerinde yapılan bit düzeyi işlemler gösterilmiştir;

```
int main() {
   int a = 6; //  0110
   int b = 2; //  0010

   int bitwise_and = a & b; // 0010
   int bitwise_or = a | b;  // 0110
   int bitwise_xor = a ^ b; // 0100
   int bitwise_not = ~b;    // 1101
   int left_shift = b << 2; // 0100
   int right_shift = b >> 1;// 0001
   return 0;
}
```

Mantıksal İşleçler

Mantıksal ifadeler doğru (true) ve yanlış (false) olabilen ifadelerdir. C dilinde mantıksal bir veri tipi yoktur ve `int` veri tipi bunun için kullanılır. Sıfırdan farklı bir tamsayı ifade doğru, sıfır olan tamsayı yanlış olarak değerlendirilir.

C Dilinde VE(AND), VEYA(OR), ÖZEL VEYA(XOR) ve DEĞİL (NOT) gibi mantıksal ifadelerin işlenmesi için aşağıdaki işleçler kullanılır. Mantıksal işleçlere (logical operator) işlenen olarak giren ifadeler öncelikle `int` veri tipine dönüştürülür. Yani bu işleçler için beklenen işlenen (operand) bir tamsayı ifadedir.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
&&	Şartlı Mantıksal VE: Eğer iki işlenen sıfırdan farklı ise 1 verir. Soldaki işlenen yanlış ya da sıfır ise sağdakine bakılmaz.
||	Şartlı Mantıksal VEYA: Eğer iki işlenenden biri sıfırdan farklı ise 1 verir. Soldaki işlenen doğru ya da sıfırdan farklı ise sağdakine bakılmaz.
!	Mantıksal DEĞİL: İşlenenden önce kullanılır. Tekli işleçlerde verilmiştir. İşleneninin mantıksal durumunu tersine çevirmek için kullanılır. Eğer işlenen sıfırdan farklı ise 0, değilse 1 verir.

Tablo 12. Mantıksal İşleçler

Aşağıda verilen ilişkisel ve mantıksal işleçlerin kullanımına ilişkin örnek programı lütfen inceleyiniz.

```
int main() {
   int a = 5, b = 6;
   int res;
   res = a&&b; // işlem sonucu res=1
   res = a||b; // işlem sonucu res=1
   res = !a;   // işlem sonucu res=0
   return 0;
}
```

Atama işleçleri

Atama işleçleri (assignment operator) en çok kullanılan işleçlerdir. Atama işleçleri her zaman sağdaki değeri sola atar! Dolayısıyla atama yapılacak uygun veri tipinde bir değişken (variable) olmak zorundadır.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
=	Sağdaki işleneni sola atar.
+=	Soldaki işleneni sağdaki işlenen üzerine toplar ve sonucu sola atar.
-=	Soldaki işlenenden sağdaki işlenenden çıkarır ve sonucu sola atar.
*=	Soldaki işleneni sağdaki işlenenle çarpar ve sonucu sola atar.

`/=`	Soldaki işleneni sağdaki işlenene böler ve sonucu sola atar. Eğer işlenenler tamsayı ise kesirli kısım göz ardı edilir. Bu durumda sonuç tamsayıdır.
`%=`	Bu işleç tamsayılarla çalışır. Soldaki işleneni sağdaki işlenene böler ve kalanı sola atar.

Tablo 13. Atama İşleçleri

Aşağıda verilen atama işleçlerin kullanımına ilişkin örnek programı lütfen inceleyiniz.

```
/* Bu program, atama işleçleri örneğidir. */
int main() {
  int a = 10; /* a değişkenine 10 değişmezi = işleciyle atanıyor.*/
  a += 10;    /* a=a+10; ile eşdeğer: a değişkenine 10 ekleniyor. */
  a -= 10;    /* a=a-10; ile eşdeğer: a değişkeninden 10 çıkarılıyor. */
  a *= 10;    /* a=a*10; ile eşdeğer: a değişkeni 10 kat yapılıyor.*/
  a /= 10;    /* a=a/10; ile eşdeğer: a değişkeni 1/10 kat yapılıyor.*/
  a %= 10;    /* a=a%10; ile eşdeğer: a değişkeninin 10 a kalanı bulunup
                 ona eşitleniyor. */
  return 0;
}
```

İşleç Öncelikleri

Cebirsel ifadelerde nasıl ki önce çarpma ve bölme sonra toplama işlemleri yapılıyor. C programlama dilinde yazılan ifadelerde de işleç öncelikleri (operator precedence) vardır. İfadelerde bir işlemi öncelikli hale getirmek için parantez `()` içerisine alınır. En içteki parantez en öncelikli olarak gerçekleştirilir. Başka işleçler de bulunmaktadır bunlar ilerleyen bölümlerde yeri geldikçe anlatılacaktır.

Öncelik	İşleç Grubu	İşleçler	Gruptaki Öncelik
1	Sonek (postfix)	`() ++ --`	Soldan Sağa
2	Tekli (unary)	`+ - ! ++ -- sizeof()`	Sağdan Sola
3	Çarpım	`* / %`	Soldan Sağa
4	Toplam	`+ -`	Soldan Sağa
5	İlişkisel (relational)	`< <= > >=`	Soldan Sağa
6	Eşitlik (equality)	`== !=`	Soldan Sağa
7	Bit düzeyi VE	`&`	Soldan Sağa
8	Bit düzeyi ÖZEL VEYA	`^`	Soldan Sağa
9	Bit Düzeyi VEYA	`|`	Soldan Sağa
10	Mantıksal VE	`&&`	Soldan Sağa
11	Mantıksal VEYA	`||`	Soldan Sağa
12	Atama (assignment)	`= += -= *= /= %= <<= >>= &= ^= |=`	Sağdan Sola

Tablo 14. İşleçlerin İşlem Öncelikleri

Aşağıda işleçlerin önceliklerine ilişkin örnek programı lütfen inceleyiniz.

```
/* Bu program, işleç öncelikleri örneğidir. */
int main() {
   int a=10, b=5;
   a= a-b+2; // Öncelik Sırası +, -, = İşlem sonunda a=7
   a= a/b+2; // Öncelik Sırası /, +, = İşlem Sonunda a=7/5+2=3
   a= 2*10-20/2+3+(12-10); // Öncelik Sırası: (-) * / - + +
   /*
    İfadenin çözüm sırası:
    > 2*10-20/2+3+2
    > 20-20/2+3+2
    > 20-10+3+2
    > 10+3+2
    > 13+2
    > 15
   */
   return 0;
}
```

Kayan Noktalı Sayı Hesapları

Kayan noktalı sayıların IEEE-754 standardında ikili tabanda tutulduğu *Değişken* başlığında anlatılmıştı. Kayan noktalı sayılar olarak değişkenlere verdiğimiz onluk tabanda değerler aslında arka planda ikilik tabanda tutulur ve bu durum yazdığımız kodlarda garip davranışların ortaya çıkmasına sebep olur. Buna örnek olarak; hemen hemen her programcının yaptığı ilk hata, aşağıdaki kodun amaçlandığı gibi çalışacağını varsaymaktır;

```
float total = 0;
for(float a = 0; a != 2; a += 0.01f) {
    total += a;
}
```

Acemi programcı bunun 0, 0.01, 0.02, 0.03, gibi artarak, 1.97, 1.98, 1.99 aralığındaki her bir sayıyı toplayacağını ve 199 sonucunu, yani matematiksel olarak doğru cevabı vereceğini varsayar. Bu kodda doğru yürümeyen iki şey olur: Birincisi yazıldığı haliyle program asla sonuçlanmaz. `a` asla 2'ye eşit olmaz ve döngü asla sonlanmaz. İkincisi ise döngü mantığını `a < 2` şartını kontrol edecek şekilde yeniden yazarsak, döngü sonlanır, ancak toplam 199'dan farklı bir şey olur. IEEE754 uyumlu işlemlerde, genellikle bunun yerine yaklaşık 201'e ulaşır. Bunun olmasının nedeni, kayan noktalı sayılarının onluk tabanda kendilerine atanan değerlerin, ikilik tabanda yaklaşık değerlerini temsil etmesidir. Bu duruma aşağıdaki klasik örnek verilir;

```
#include <stdio.h>
int main()
{
    double a = 0.1;
    double b = 0.2;
    double c = 0.3;
    if(a + b == c)
        printf("Bu satır İcra Edilmez!\n"); //IEEE754 standardında işlem yaparsak.
    else
        printf("Kayan Noktalı Sayılar İkili Sayı Sisteminde"
               " Tutulduğundan  Yaklaşık Olarak Hesaplanır." );
    return 0;
}
```

Programcı olarak gördüğümüz şey onluk tabanda yazılmış üç sayı olsa da derleyicinin (ve altta yatan donanımın) gördüğü şey ikili sayılardır. 0.1, 0.2 ve 0.3, ona mükemmel bölmeyi gerektirdiğinden (ki bu onluk sayı sisteminde oldukça kolaydır), ancak ikili sayı sisteminde imkansızdır (bu sayılar, onluk sayı sistemindeki 1/3 sayısı gibi 0.333333333333333... şeklinde kesin olmayan biçimde depolanması gerektiğindendir);

```
#include <stdio.h>
int main()
{
    /*64 bitlik kayan noktalı sayılar, tam sayı kısmı dahil olmak üzere
      53 basamaklı hassasiyete sahiptir *
    double a = 0011111110111001100110011001100110011001100110011001100110011010;
    //onluk sistemdeki 0.1 ikili sistemde kusurlu olarak saklanır
    double b = 0011111111001001100110011001100110011001100110011001100110011010;
    //onluk sistemdeki 0.2 ikili sistemde kusurlu olarak saklanır
    double c = 0011111111010011001100110011001100110011001100110011001100110011;
    //onluk sistemdeki 0.3 ikili sistemde kusurlu olarak saklanır
    double a + b = 0011111111010011001100110011001100110011001100110011001100110100;
    //c değişkeni ile a+b nin onluk sistemde 0,3'e tam olarak eşit olmadığını unutmayın!
    return 0;
}
```

GİRİŞ ÇIKIŞ İŞLEMLERİ

Modüler Programlama

C programlama dili, yapısal olmasının yanında aynı zamanda modüler bir programlama dilidir. Modüler programlamada yazılan fonksiyonlar gruplar halinde başka kod dosyalarına konulur ve bu bileşenler gerektikçe projemize `#include` ön işlemci yönergeleriyle (preprocessor directive) dahil edilir.

Modüllere ayırmanın soyutlama (abstraction), değişim yönetimi (change management) ve yeniden kullanım (reusing) gibi üstünlükleri vardır. Modüllere ayrılan bir kodun bakımı da daha kolaydır. Ancak daha fazla fonksiyonun çağrılması, işlemciyi yorar. Çünkü her fonksiyon çağrısında işlemcinin mevcut durumu ve kaydediciler belleğe itilir ve fonksiyondan geri dönüldüğünde eski duruma dönmek içi bu veriler tekrar geri çekilir. İşlemciye yüklenen bu çağırma yükü (calling overhead) günümüz bilgisayarlarında oldukça ihmal edilebilir seviyededir. Ancak milisaniyeler bazında yapılması gereken zaman kritik işler gerçekleştirmek için bu durum göz önüne alınmalıdır.

C programlama dilinde en çok kullanacağımız modüllerden biri de standart giriş çıkış işlemlerinin (input output-IO) tanımlı olduğu `stdio.h` başlık (header) dosyasıdır. Bu modülü kodumuza dahil etmek (include) için kodu yazdığımız metin editörünüzde kaynak kodun başına `#include` ön işlemci yönergesini (preprocessor directive) aşağıdaki şekilde ekleriz.

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

Böylece konsola (konsol kelimesi UNIX/Linux işletim sitemindeki terminal ya da Windows işletim sistemindeki komut satırını ifade eder) bir şey yazmak için `printf` veya klavyeden bir şey okumak ve bir değişkene atamak için ise `scanf` fonksiyonlarını kullanabilir hale geliriz.

C programlama dilinde her işletim siteminde çalışan standart birçok başlık dosyası bulunmaktadır. Ancak bunların dışında, işletim sistemine özel olan başlık dosyaları da bulunmaktadır;

Başlık Dosyası	Açıklama
`stdio.h`	Standart giriş-çıkış komutları
`math.h`	Matematiksel fonksiyonlar
`stdlib.h`	Dönüşüm, sıralama, arama vb. komutları
`string.h`	Alfa sayısal ve bazı bellek yönetim komutları
`curses.h`	curses, metin terminali ekranında imleç (cursor) denetimini destekleyen eski bir Unix/Linux kütüphanesidir.
`conio.h`	DOS/Windows komut satırında imleç (cursor) denetimini destekleyen kütüphanedir

Tablo 15. En çok kullanılan başlık dosyaları

Konsola Biçimlendirilmiş Veri Yazma

C programlama dilinde değişkenlerimizde olan veriyi biçimlendirilmiş (formatted) bir şekilde imlecin bulunduğu noktadan itibaren konsola yazabiliriz (print). Bunu sağlayan fonksiyon `stdio.h` başlık dosyasındaki `printf` fonksiyonudur.

Bir fonksiyonun hangi parametreleri aldığı ve neyi geri döndürdüğünü prototipinden (prototype) anlarız. Aşağıda `printf` fonksiyonunun prototipi verilmiştir;

```
void printf (char *format, ... );
```

Bu fonksiyon bir değişmez metni (string literal) ilk olarak parametre olarak alır. Metinlere ilişkin işlemler ileride *DİZGİLER* başlığında anlatılacaktır. Değişmez metinler C dilinde çift tırnak (`"`) karakterleri arasında yazılır. İlk parametre olan bu biçimlendirme metni aynı zamanda kendisinden sonra gelecek parametrelerin de hangi biçimde konsola yazılacağını belirler. Her bir parametrenin

konsola nasıl yazılacağını biçim belirleyicisi (format specifier) belirler[15]. Biçim belirleyicileri iki karakter olup ilk karakteri yüzde (%) karakteridir.

```
/* Bu program, printf örneğidir. */
#include <stdio.h>

int main() {
   int yas=50;
   float kilo=100.0;
   printf("Merhaba!");  /* Bu printf fonksiyonunu sadece değişmez metin
                           içeren tek argümanla çağırmıştır. İmlecin
                           bulunduğu yerden itibaren
                           "Merhaba!" yazar ve imleci metnin sonuna taşır.*/

   printf("Yaşınız:%d",yas); /* Bu printf fonksiyonunu içinde
                                biçimlendirme karakteri olan
                                değişmez metin sonrasında yas değişkeniyle
                                birlikte iki argümanla çağırmıştır. İmlecin
                                son konumundan başlayarak
                                "Yaşınız:50" yazar ve imleci metnin
                                sonuna taşır. */
   printf("Kilonuz:%f",kilo); /* Bu printf fonksiyonunu içinde biçimlendirme
                                 karakteri olan değişmez metin ve izleyen
                                 kilo değişkeniyle birlikte iki
                                 argümanla çağırmıştır. İmlecin son
                                 konumundan başlayarak
                                 "Kilonuz:100.000000" yazar ve imleci
                                 metnin sonuna taşır.*/
   return 0;
}
/*Program Çıktısı:
Merhaba!Yaşınız:50Kilonuz:100.000000

...Program finished with exit code 0
*/
```

Görüldüğü üzere biçimlendirme metni içindeki `%d` ile ifade edilmiş biçim belirleyicisi izleyen argüman olan `yas` değişkeni ondalık (decimal) olarak konsola yazdırılmıştır. Burada dikkat edilmesi gereken `yas` argümanının, `%d` yani ondalık olarak biçimlendirilecek bir veri olmasıdır. Benzer şekilde en sonraki talimatta biçimlendirme metni içindeki `%f` ile ifade edilmiş biçim belirleyicisi izleyen argüman olan `kilo` değişkeni kayan noktalı sayı (float) olarak konsola yazdırılmıştır.

Biçim Karakterleri	Biçimlendirme metni sonrasındaki parametrelerin biçimi
%d	Konsola ondalık sayı olarak yazılır
%c	Konsola karakter olarak yazılır.
%f	Konsola kayan noktalı sayı olarak yazılır.
%x	Konsola onaltılık (hexadecimal) sayı olarak yazılır.
%s	Konsola metin olarak yazılır.
%%	Konsola % karakteri yazılır.

Tablo 16. Biçim Belirleyiciler

Eğer konsola `%` karakteri yazmak istersek biçimlendirme olarak iki `%` karakteri kullanmalıyız. Buna ilişkin örnek aşağıda verilmiştir.

```
/* Bu program, printf ile konsola % karakteri basma örneğidir. */
#include <stdio.h>

int main() {
   int doluluk=80;
```

[15] https://www.tutorialspoint.com/cprogramming/c_format_specifiers.htm

```
   printf("Doluluk Oranı: %%%d",doluluk);
   return 0;
}
/*Program Çıktısı:
Doluluk Oranı: %80

...Program finished with exit code 0
*/
```

Bu başlık altında şu ana kadar tem bir parametreyi biçimlendirilmiş olarak konsola yazmayı öğrendik. Birden çok parametreye örnek olarak aşağıdaki örnek verilebilir.

```
/* Bu program, çok argümanlı printf örneğidir. */
#include <stdio.h>

int main() {
   char yas=45;
   float kilo=81.0;
   int doluluk=80;
   printf("Yaş: %d, Kilo: %f, Doluluk Oranı: %%%d",yas,kilo,doluluk);
   return 0;
}
/* Program Çıktısı:
Yaş: 45, Kilo: 81.000000, Doluluk Oranı: %80

...Program finished with exit code 0
*/
```

Format metnini imleci hareket ettirecek şekilde biçimlendirebiliriz. Yukarıdaki örneğin çıktısı göz önüne alındığında, biçimlendirme metni içinde imlecin satır başına geçmesini sağlamak için `'\n'` şeklinde kaçış tuşu dizisi (escape sequence) kullanılabilir. Bu diziler aşağıda verilmiştir;

Kaçış tuşu dizisi	Açıklama
\n	Konsolda imleç yeni satırın (new line) başına hareket eder.
\t	İmlece, metin yazarken sekme (tab) tuşuna basında olan hareketi yapar. Yani imleç, konsolda 8 in katları olan sütuna hareket eder.
\r	Konsolda imleç bulunduğu satırın başına (carriage return) hareket eder.
\a	Konsol, uyarı (alert) sesi çıkarır.
\\	Konsola ters bölü karakterini yazar.
\"	Konsola çift tırnak karakterini yazar.
\'	Konsola tek tırnak karakterini yazar.

Tablo 17. Kaçış Tuşu Dizileri

Aşağıda kaçış tuşu dizilerine ilişkin örnek program verilmiştir.

```
/* Bu program, escape sequence printf örneğidir. */
#include <stdio.h>
int main() {
   printf("ILHAN\n"); // ILHAN yazılarak konsolda satır başı yapılır.
   printf("OZKAN\r"); // OZKAN yazılarak aynı satırın başına dönülür.

   printf("ABC\tDEF\n");  /* ABC yazılır ve 8. sütuna ilerlenir ve
                            sonra DEF yazılır, sonrasında satır başı
                            yapılır. TAB karakteri 8 ve katları olan
                            sütunlara imleci ilerletir.*/
   printf("\"İyi\"-\"Kötü\""); // Çift tırnak içeren "İyi"-"Kötü" yazılır.
   return 0;
}
/* Program Çıktısı:
ILHAN
ABCAN   DEF
```

```
"İyi"-"Kötü"

...Program finished with exit code 0
*/
```

Biçim belirleyiciler (format specifier) ile daha fazla biçimlendirme yapılabilir. Bu durumda biçimlendirme için aşağıdaki desen kullanılır;

```
%[önek][genişlik][.ondalık]<biçim karakteri>
```

Burada;

1. Genişlik, konsola kaç karakter genişliğinde yazdırılacağını belirler.
2. Ondalık, konsola kayan noktalı sayı yazılacak ise noktadan sonra kaç rakam yazdırılacağını belirler.
3. Önek içerisinde aşağıdaki karakterler olabilir;
   a. `+` karakteri: Sayının işaretini konsola yazılmasını sağlar.
   b. `-` karakteri: Yazdırılacak parametrenin belirtilen genişliğin soluna yaslanarak yazdırılmasını sağlar.
   c. `0` karakteri: Yazdırılacak sayının soluna belirtilen genişliğe bağlı olarak `0` karakteri konulmasını sağlar.

Aşağıda 10 karakterlik genişlikte PI sayısının biçimlendirme örnekleri verilmiştir.

```
/* Bu program, detaylı biçimlendirme printf örneğidir. */
#include <stdio.h>
int main() {
   const float PI=3.141527;
   printf("1234567890\n");
   printf("%.2f\n",PI);
   printf("%3.f\n",PI);
   printf("%10.2f\n",PI);
   printf("%+10.3f\n",PI);
   printf("%+10.4f\n",PI);
   printf("%010.4f\n",PI);
   printf("%-10.4f\n",PI);
   printf("%-+10.2f\n",PI);
   printf("%-+10.2f\n",-PI);
   printf("%+10.2f\n",-PI);
   printf("%0+10.2f\n",-PI);
   return 0;
}
/*Program Çıktısı:
1234567890
3.14
  3
      3.14
    +3.142
   +3.1415
00003.1415
3.1415
+3.14
-3.14
     -3.14
-000003.14

...Program finished with exit code 0
*/
```

Klavyeden Biçimlendirilmiş Veri Okuma

C programlama dilinde değişkenlerimizde klavyeden biçimlendirilmiş (formatted) olarak girilen veriyi tarayıp (scan), veriyi tutan değişkenlere koyabiliriz. Bunu sağlayan fonksiyon `stdio.h` başlık dosyasındaki `scanf` fonksiyonudur. Bu `scanf` fonksiyonunun prototipi verilmiştir;

```
void scanf (char *format, ... );
```

Bu fonksiyon da bir metni ilk olarak parametre olarak alır. İlk parametre olan bu biçimlendirme metni aynı zamanda kendisinden sonra gelecek parametrelerin de hangi biçimde klavyeden okunacağını belirler. Her bir parametrenin klavyeden nasıl okunacağını `printf` fonksiyonundaki gibi biçim belirleyicisi (format specifier) belirler. Farklı olarak, okunan değerlerin konulacağı değişkenlerin adresleri parametre olarak verilir. Bir değişkenin adresinin `&` işleci ile elde edileceği *Tekli İşleçler* altında anlatılmıştı.

Aşağıda kapasite oranını okuyup `%` olarak ekrana yazan bir program örneği verilmiştir.

```
/* Bu program, detaylı biçimlendirme scanf örneğidir. */
#include <stdio.h>
int main() {
   float kapasiteOrani;
   printf("Kapasite Oranını (0.00-1.00) Giriniz:");
   scanf("%f",&kapasiteOrani);
   printf("Girilen Kapasite Oranı: %.2f\n",kapasiteOrani);
   printf("%% olarak Girilen Kapasite Oranı: %%%2.2f", 100*kapasiteOrani);
   return 0;
}
/*Program Çıktısı:
Kapasite Oranını (0.00-1.00) Giriniz:0.75
Girilen Kapasite Oranı: 0.75
% olarak Girilen Kapasite Oranı: %75.00

...Program finished with exit code 0
*/
```

Benzer şekilde yarıçapı tamsayı olarak klavyeden alınan bir çemberin ve çevresini hesaplayan program aşağıda verilmiştir.

```
/* Bu program, detaylı biçimlendirme scanf örneğidir. */
#include <stdio.h>
int main() {
   int yaricap;
   float cevre;
   printf("Çemberin yarıçapını tamsayı olarak giriniz:");
   scanf("%d",&yaricap);
   cevre=2*3.1415*yaricap;
   printf("Yarıçapı %d olan çemberin çevresi: %.2f",yaricap,cevre);
   return 0;
}
/*Program Çıktısı:
Çemberin yarıçapını tamsayı olarak giriniz:4
Yarıçapı 4 olan çemberin çevresi: 25.13

...Program finished with exit code 0
*/
```

KONTROL İŞLEMLERİ

Ardışık İşlem ve Kontrol İşlemleri

Yapısal Programlama başlığı altında programın ana fonksiyondan başlayacağı ve programlamanın ise birbirini çağıran fonksiyonlarla yapıldığı anlatılmıştı. Yapısal programlamada, ana fonksiyon dahil tüm fonksiyonlarda ilk önce işlenecek verileri taşıyan veri yapıları tanımlanır ve ardından bu verileri işleyen kontrol yapılarına ilişkin talimatlar yazılır.

Şu ana karar örneğini verdiğimiz kodlarda veri yapısı olarak sadece değişkenler kullanılmıştır. Sonrasında ise giriş çıkış işlemleri talimatlar içeren programlar yazılmıştır. Programın icrası ilk talimattan başlar ve sırasıyla program bitene kadar devam eder. İşte programın icrasını değiştirmeyen bu tür talimatlara ardışık işlem (sequential operation) adı verilir.

Ardışık işlemler aşağıdaki üç tür talimattan (statement) oluşur.

1. Değişkenlerin kimliklendirildiği değişken tanımlamaları (identifier definition):

```
int yariCap=3;
const float PI=3.14;
float daireninAlani,daireninCevresi;
```

2. İfadelerden (expression) yani sabit, değişken ve operatör içeren sözdizimleri:

```
daireninAlani=PI*yariCap*yariCap;
float daireninCevresi=2.0*PI*yariCap;
```

3. Klavyeden veri okume ve konsola veri yazma gibi giriş çıkış işlemleri (input output operation):

```
printf("Kapasite Oranını (0.00-1.00) Giriniz:");
scanf("%f",&kapasiteOrani);
```

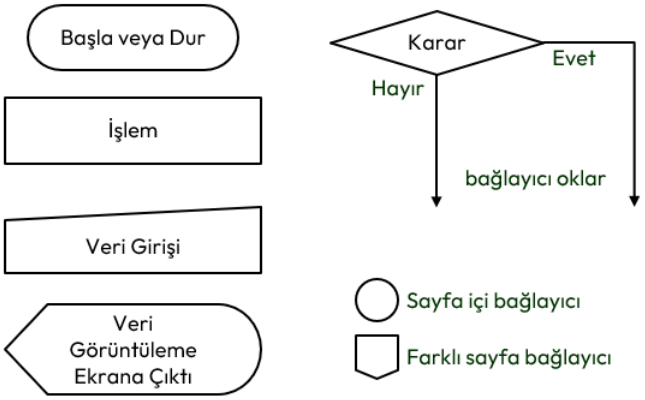
Kontrol işlemleri (control operation) ise programın icra sırasını değiştiren kontrol yapıları olup bu talimatlar (statement) olup üç türü vardır;

1. Duruma göre seçimler (conditional choice): `if`, `if-else` ve `switch` talimatları.
2. İlişkisel döngü (relational loop): `while`, `do-while` ve `for` talimatları.
3. Dallanmalar (jump): `continue`, `break`, `goto` ve `return` talimatları.

Kontrol işlemlerinde; kodlana mantıksal satırlar (logical sequence) ile fiziksel olarak icra edilen satırlar (physical sequence) birbirinden farklıdır.

Akış Diyagramları

Kontrol işlemi içeren programın icrasını anlamak için akış diyagramlarını da anlamak gerekir. Aşağıda akış diyagramlarındaki temel şekiller verilmiştir.



Şekil 4. Akış Diyagramları Genel Şekilleri

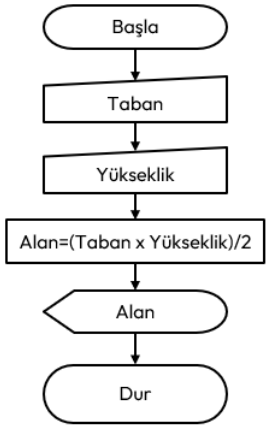
Akış diyagramları (flowchart), adımların grafiksel gösterimleridir. Algoritmaları ve programlama mantığını temsil etmek için bir araç olarak bilgisayar biliminden ortaya çıkmıştır. Ancak diğer tüm işlem türlerinde kullanılmak üzere genişletilmiştir. Günümüzde akış diyagramları, bilgileri göstermede

ve akıl yürütmeye yardımcı olmada son derece önemli bir rol oynamaktadır. Karmaşık süreçleri görselleştirmemize veya sorunların ve görevlerin yapısını açık hale getirmemize yardımcı olurlar. Bir akış şeması, uygulanacak bir süreci veya projeyi tanımlamak için de kullanılabilir.

Akış diyagramı çizilirken aşağıdaki kurallara uyulur;

- Akış şemalarında tek bir başlangıç simgesi olmalıdır
- Bitiş simgesi birden çok olabilir.
- Karar simgesinin haricindeki simgelere her zaman tek giriş ve tek çıkış yolu bulunur.
- Bağlaç simgesi sayfanın dolmasından ötürü parçalanan akış şemasının öğelerini birleştirmede kullanılır.
- Simgeler birbirleri ile tek yönlü okla bağlanırlar.
- Okların yönü algoritmanın mantıksal işlem akışını tanımlar.

Aşağıda taban ve yüksekliği klavyeden girilen bir üçgenin alanını hesaplamak için bir akış diyagramı örneği verilmiştir.



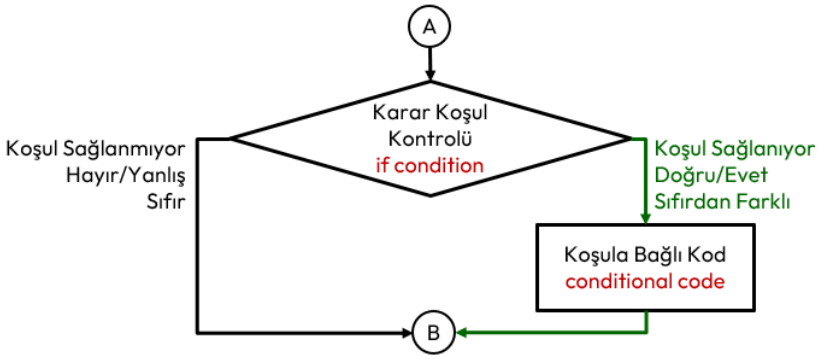
Şekil 5. Örnek Akış Diyagramı

Duruma Göre Seçimler

Duruma göre seçimler (conditional choice) programın akışını bir koşula göre değiştiren talimatlardır.

If Talimatı

If talimatı, karar vermeye (decision making) ilişkin bir talimat olup bir koşula bağlı olarak programın icrasını değiştirir. Koşul (condition) ifadesi DOĞRU ise koşul kodu (conditional code) icra edilir, değilse icra edilmez. Koşul ifadesi (expression) test edilir ve sıfırdan farklı ise DOĞRU, aksi halde YANLIŞ kabul edilir.



Şekil 6. If Talimatı İcra Akışı

Sözde kodu aşağıda verilmiştir.

```
if (koşul)
    koşul-kodu;
```

Aşağıdaki örnek programı inceleyelim;

		yas	cinsiyet
	`#include <stdio.h>`	yas	cinsiyet
	`int main() {`	18	?
1	`int yas=18;`	18	'K'
2	`char cinsiyet='K';`	18	'K'
3	`if (yas<30)`	18	'K'
4	`printf("Genç");`	18	'K'
5	`if (cinsiyet=='E')`	18	'K'
	`printf("Erkek");`	18	'K'
6	`return 0;`	18	'K'
	`}`	18	'K'

Tablo 18. If Talimatı İçeren Bir C Programının İcrası

Programın icrası, `main` fonksiyonu içindeki ilk talimatla başlar ve solda verilen sırayla icra edilir. Sağda ise her icra sonrası değişkenlerin değerleri gösterilmiştir. Her talimatın icrasında neler olduğu aşağıda verilmiştir;

1. `yas` değişkenine `18` değeri atanır.
2. `cinsiyet` değişkenine `'K'` atanır.
3. `if` talimatındaki koşul (condition) test edilir. `yas<30` yani `18<30` testinde küçüktür işleci `1` verir. Test sonucu DOĞRU olduğundan izleyen koşul kodu (conditional code) icra edilir.
4. `printf("Genç\n");` koşul kodu olduğundan icra edilir.
5. `if` talimatındaki koşul test edilir. `cinsiyet=='E'` yani `'E'=='K'` testinde eşit mi işleci `0` verir. Test sonucu YANLIŞ olduğundan izleyen koşul kodu icra edilmez.
6. Programdan çıkılırken işletim sitemine `0` geri döndürülür.

Bu durumda programın çalışması sonucu aşağıdaki çıktı elde edilir.

```
Genç

...Program finished with exit code 0
```

`if` talimatında koşul kodu (conditional code) her zaman ardışık işlem (sequential operation) olmaz. `if` gibi bir kontrol işlemi (control operation) de olabilir. Aşağıda kademeli (compound/cascade) if kullanımına ilişkin kod örneğinde ikinci `if`, birinci `if` talimatının koşul kodudur.

```
    if (yas<30)
        if (yas<7)
            printf("Çoçuk ");
```

Koşul kodu (conditional code) birden fazla talimattan oluşacak ise kod bloğu `{ }` içine alınır. Aşağıda verilen kod örneğinde ilk `if` talimatında `yas<30` ile test edilen koşul doğrulandığında kod bloğunun içindeki tüm talimatlar icra edilir.

```
    if (yas<30) {
        if (yas<7)
           printf("Çocuk ");
        if (yas<18)
           printf("Genç ");
        if (yas>60)
           printf("Yaşlı ");
    }
```

Bahsedilen iki duruma iç içe de olabilir. Buna ilişkin kod örneği de aşağıda verilmiştir.

```
    if (cinsiyet=='E') {
        if (yas<7)
           printf("Erkek Çocuk ");
        if (yas<18)
           printf("Genç Delikanlı ");
        if (yas>60)
           printf("Yaşlı Adam ");
    }
    if (cinsiyet=='K') {
```

```
        if (yas<3)
           printf("Kız Bebek ");
        if (yas<7)
           printf("Kız Çocuk ");
        if (yas<18)
           printf("Genç Kız ");
        if (yas>60)
           printf("Yaşlı Kadın ");
    }
```

If talimatında koşul (condition) her zaman tek bir test ifadesinden oluşmayabilir. Bahsedilen duruma ilişkin kod örneği de aşağıda verilmiştir.

```
    if ((cinsiyet=='E') && (yas<18))
       printf("Genç Delikanlı ");
    if ((cinsiyet=='K') && (yas>60))
       printf("Yaşlı Kadın ");
```

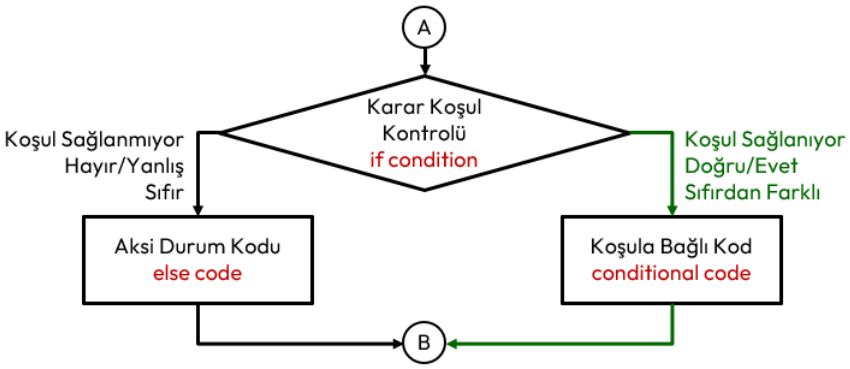
Bazen programcı tarafından aşağıdaki gibi `if` talimatları yazabilir.

```
    if (1)
      printf("Bu metin konsola her zaman yazılır.");
    if (0)
      printf("Bu metin konsola hiçbir zaman yazılmaz!");
```

If-Else Talimatı

If-Else Talimatı, bir başka karar verme (decision making) talimatı olup bir koşula bağlı olarak programın icrasını değiştirir. If talimatına benzer şekilde koşul (condition) ifadesi DOĞRU ise koşul kodu (conditional code) icra edilir, değilse aksi durum kodu (else code) icra edilir.

```
if (koşul)
    koşul-kodu;
else
    aksi-durum-kodu;
```



Şekil 7. If-Else Talimatı Sözde Kodu ve İcra Akışı

Aşağıda verilen örnek programı inceleyelim;

		yas	cinsiyet
	`/* Bu program, if-else talimatı örneğidir. */`		
	`#include <stdio.h>`		
	`int main() {`		?
1	`    int yas=65;`	65	?
2	`    char cinsiyet='K';`	65	'K'
3	`    if (yas<50)`	65	'K'
	`        printf("Genç ");`	65	'K'
4	`    else`	65	'K'
5	`        printf("Yaşlı ");`	65	'K'
6	`    return 0;`	65	'K'
	`}`		

Tablo 19. If-Else Talimatı İçeren Bir C Programının İcrası

Programın icrası, `main` fonksiyonu içindeki ilk talimatla başlar ve solda verilen sırayla icra edilir. Sağda ise her icra sonrası değişkenlerin değerleri gösterilmiştir. Her talimatın icrasında neler olduğu aşağıda verilmiştir;

1. `yas` değişkenine `65` değeri atanır.
2. `cinsiyet` değişkenine `'K'` atanır.
3. `if` talimatındaki koşul (condition) test edilir. `yas<50` yani `65<50` testinde küçüktür işleci `0` verir. Test sonucu YANLIŞ olduğundan aksi durum kodu (else-code) icra edilir.
4. Programın icrası `else` ifadesinden devam eder.
5. `printf("Yaşlı \n");` aksi durum kodu olduğundan icra edilir.
6. Programdan çıkılırken işletim sitemine `0` geri döndürülür.

Bu durumda programın çalışması sonucu aşağıdaki çıktı elde edilir.

```
Yaşlı

...Program finished with exit code 0
```

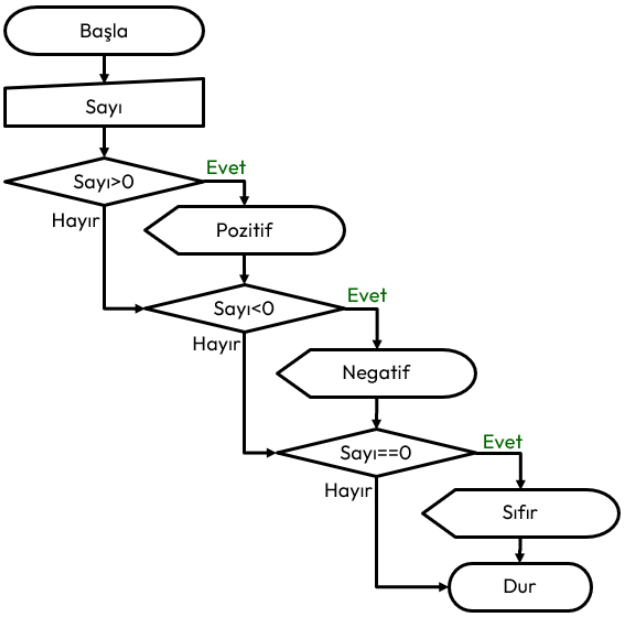
If talimatında olduğu gibi bu talimatta da koşul kodu (conditional code) ya da aksi durum kodu (else-code) birden fazla talimattan oluşacak ise kod bloğu `{ }` içine alınır. Buna ilişkin kod örneği aşağıda verilmiştir.

```
    if (cinsiyet=='E') {
        if (yas<7)
           printf("Erkek Çocuk ");
        if (yas<18)
           printf("Genç Delikanlı ");
        if (yas>60)
           printf("Yaşlı Adam ");
    } else {
        if (yas<3)
           printf("Kız Bebek ");
        if (yas<7)
           printf("Kız Çocuk ");
        if (yas<18)
           printf("Genç Kız ");
        if (yas>60)
           printf("Yaşlı Kadın ");
    }
```

Örnek olarak klavyeden girilen bir sayının sıfırdan küçük mü? Büyük mü? ya da sıfıra eşit mi? Olduğunu bulan bir C programı yazalım;

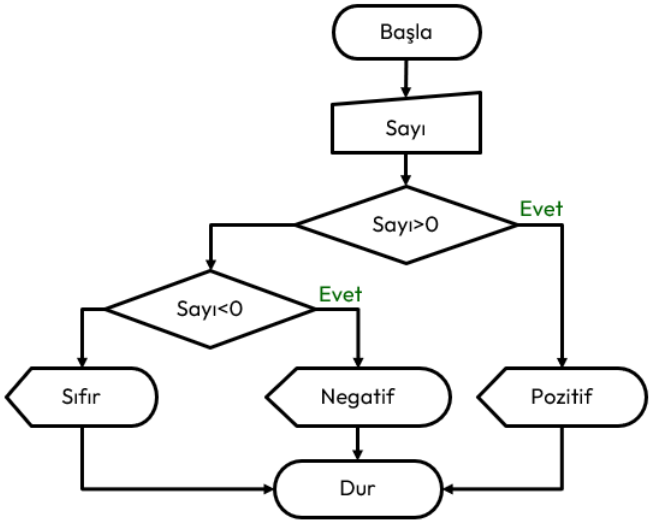
```
/* Sadece if kullanarak programın yazılması */
#include <stdio.h>
int main() {
   int sayi;
   printf("Sayı Giriniz:");
   scanf("%d",&sayi);
   if (sayi>0) // sayının pozitif olup olmadığı test ediliyor.
      printf("Pozitif"); // Bu noktada sayının pozitif olduğu biliniyor.
   if (sayi<0) // sayının negatif olup olmadığı test ediliyor.
      printf("Negatif"); // Bu noktada sayının negatif olduğu biliniyor.
   if (sayi==0) // sayının sıfır olup olmadığı test ediliyor.
      printf("Sıfır"); // Bu noktada sayının sıfır olduğu biliniyor.
   return 0;
}
```

Programın icra akışı aşağıdaki şekilde olacaktır.



Şekil 8. Programın If Talimatlarıyla Yazılması Halinde İcra Akışı

İcra akışı incelendiğinde ilk if talimatında yapılan test doğru çıkarsa geri kalan testlerin yapılmasına gerek kalmaz. Benzer şekilde ilk iki test geçilirse üçüncü if talimatındaki teste gerek yoktur. Bu durumu aşağıdaki akış diyagramında daha net görebiliriz.



Şekil 9. Programın If-Else Talimatlarıyla Yazılması Halinde İcra Akışı

Bu durumda aynı program if-else talimatlarıyla aşağıdaki şekilde yeniden yazabiliriz;

```
#include <stdio.h>
int main() {
    int sayi;
    printf("Sayi Giriniz:");
    scanf("%d",&sayi);
    if (sayi>0) // sayı için pozitif testi yapıldı
        printf("Pozitif"); // Burada sayının pozitif olduğu biliniyor
    else { // Burada sayının pozitif olmadığı biliniyor
        if (sayi<0) // sayı için negatif testi yapıldı
            printf("Negatif"); /* Bu noktada sayının negatif olduğu
                                  biliniyor */
        else /* Bu noktada sayının hem pozitif
                hem de negatif olmadığı biliniyor */
            printf("Sıfır"); // Bu noktada sayının sıfır olduğu biliniyor
    }
    return 0;
```

```
}
```

Program incelendiğinde sayının pozitif girilmesi halinde sadece ilk if talimatındaki test geçecek ve else sonrası aksi durum koduna ilişkin blok icra edilmeyecektir. Sayının negatif girilmesi halinde ilk if talimatının else bloğuna girilecek ve blok içindeki ilk if talimatındaki test geçecek ve bu if talimatının else kısmı çalıştırılmayacaktır. Sayı sıfır girilmiş ise blok içindeki else kodu çalıştırılacaktır.

Bunların dışında blok içinde tek bir if-else talimatı bulunmaktadır. Dolayısıyla blok içine almaya gerek de yoktur. Bu durumda kodun nihai hali aşağıda verilmiştir.

```
    if (sayi>0)
        printf("Pozitif");
    else if (sayi<0)
        printf("Negatif");
    else
        printf("Sıfır");
```

Sarkan Else

Aşağıdaki program örneği incelendiğinde else, hangi if talimatına aittir? Böyle bir kod programcı tarafından yazılabilir ve derleyici buna hiçbir sıkıntı çıkarmaz.

```
    if (cinsiyet=='E') if (yas<18) printf("Delikanlı"); else printf("Adam");
```

Bu durumda kodu aşağıdaki gibi okunaklı hale getirdiğimizde ilk `if` talimatının koşul kodunun (conditional code) ikinci if-else talimatı olduğu görülmektedir. Bu problem sarkan else (dangling else) problemi olarak bilinir. Kod bloğu kullanılmadan yazılan bu tip kodlarda `else` her zaman kendinden bir önceki if talimatına aittir.

```
    if (cinsiyet=='E')
        if (yas<18)
            printf("Delikanlı");
        else
            printf("Adam");
```

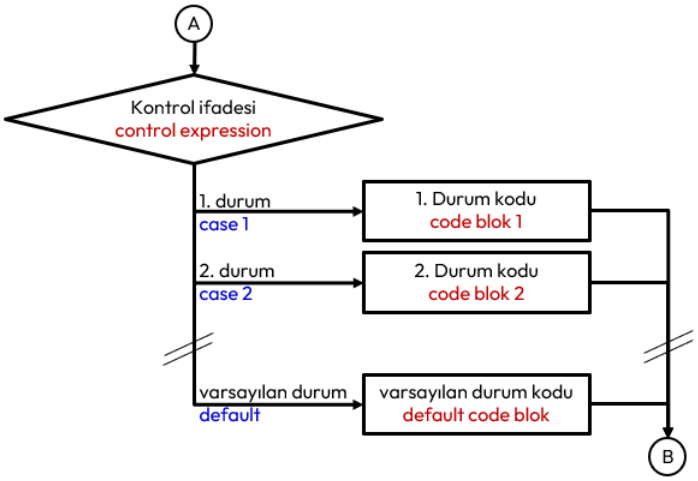
Switch Talimatı

Switch talimatı, bir kontrol ifadesi (expression) sonucunda birden fazla alternatif arasında seçim yapılmasını sağlar. Sözde kodu aşağıda verilmiştir;

```
switch (kontrolifadesi) {
   case alternatif-1:
      alternatif-1-kodu;
      break;
   case alternatif-2:
      alternatif-2-kodu;
      break;
   // ...
   default:
      varsayılan-alternatif-kodu;
}
```

Switch talimatına ilişkin kurallar;

1. Kontrol ifadesi (control expression), sonucu tamsayı olan ifadedir ve bir kez değerlendirilir.
2. Bloğu olmayan bir `switch` talimatı yazılamaz!
3. Her bir alternatif bir tamsayı değişmezi (integer literal) izleyen ve iki nokta ile biten bir etiket (label) olarak tanımlanır. Alternatif bir değişken olamaz!
4. Blok için yazılan kod sonuna `break;` talimatı yazılarak `switch` bloğu dışına çıkılır. Zorunlu değildir, yazılmaz ise bir sonraki alternatif için yazılan kod icra edilir.
5. Tüm tamsayılar içerecek şekilde alternatiflerin tümü yazılmayabilir.
6. Blok sonunda yer alacak şekilde üzerinde yer alan alternatifler dışında kalan tüm alternatifler için `default:` etiketli alternatif yazılabilir. Zorunlu değildir.



Şekil 10.Switch Talimatı Sözde Kodu ve İcra Akışı

Aşağıda menü seçimi için yazılmış bir program örneği verilmiştir.

```
#include <stdio.h>
int main() {
    int menu;
    printf("Menü İçin Rakam Giriniz:");
    scanf("%d",&menu);
    switch(menu) {
    case 1:
        printf("1 Numaralı Menüyü Seçtiniz.\n");
        printf("Hamburger ve Ayran Hazırlanacak.\n");
        break;
    case 2:
        printf("2 Numaralı Menüyü Seçtiniz.\n");
        printf("Patates Kızartması ve Kola Hazırlanacak.\n");
        break;
    case 3:
    case 4:
        printf("3 veya 4 Numaralı Menüyü Seçtiniz.\n");
        break;
    default:
        printf("1,2, 3 veya 4 Dışında Menü Seçtiniz.\n");
        printf("Böyle bir Menümüz Yok!\n");
    }
    return 0;
}
```

Programı aşağıdaki durumlar için çalıştırabiliriz;

1. Programda klavyeden `1` girildiğinde `switch` talimatında kontrol ifadesi test edilir ve sonucu `1` olduğuna karar verilir. Bu durumda `case 1:` etiketine gidilir ve bu alternatife ilişkin kodlar icra edilir. Yani ilk önce `printf("1 Numaralı Menüyü Seçtiniz.\n");` icra edilir ve bir sonraki talimata geçilir. `printf("Hamburger ve Ayran Hazırlanacak.\n");` icra edilir ve bir sonraki talimata geçilir. `break;` talimatı bizi `switch` bloğunun sonundan dışına çıkarır. Son olarak `return 0;` talimatı icra edilerek işletim sistemine dönülür.
2. Programda klavyeden `2` girildiğinde `switch` talimatında kontrol ifadesi test edilir ve sonucu `2` olduğuna karar verilir. Bu durumda `case 2:` etiketine gidilir ve bu alternatife ilişkin kodlar icra edilir. Yani ilk önce `printf("2 Numaralı Menüyü Seçtiniz.\n");` icra edilir ve bir sonraki talimata geçilir. `printf("Patates Kızartması ve Kola Hazırlanacak.\n");` icra edilir ve bir sonraki talimata geçilir. `break;` talimatı bizi `switch` bloğunun sonundan dışına çıkarır. Son olarak `return 0;` talimatı icra edilerek işletim sistemine dönülür.
3. Programda klavyeden `3` girildiğinde `switch` talimatında kontrol ifadesi test edilir ve sonucu `3` olduğuna karar verilir. Bu durumda `case 3:` etiketine gidilir ve bu alternatife ilişkin icra edilecek kod yoktur. Etiket dışında kod bulunana kadar devam edilir. Bulunan ilk kod `printf("3 veya 4 Numaralı`

`Menüyü Seçtiniz.\n");` talimatıdır ve icra edilir ve bir sonraki talimata geçilir. `break;` talimatı bizi `switch` bloğunun sonundan dışına çıkarır. Son olarak `return 0;` talimatı icra edilerek işletim sistemine dönülür.

4. Programda klavyeden `4` girildiğinde `switch` talimatında kontrol ifadesi test edilir ve sonucu `4` olduğuna karar verilir. Bu durumda `case 4:` etiketine gidilir ve bir üst maddede belirtilen icra gerçekleşir.
5. Programda klavyeden `-1` , `0`, `12` ve `300` gibi bir sayı girildiğinde `switch` talimatında kontrol ifadesi test edilir ve sonucu `1`,`2`,`3`,`4` alternatiflerinden biri olmadığına karar verilir. Bu durumda `default:` etiketine gidilir. Bu etiket sonrası ilk önce `printf("1 Numaralı Menüyü Seçtiniz.\n");` icra edilir ve bir sonraki talimata geçilir. `printf("Hamburger ve Ayran Hazırlanacak.\n");` icra edilir. Zaten blok sonuna ulaşılmıştır. Bloktan çıkılır ve son olarak `return 0;` talimatı icra edilerek işletim sistemine dönülür.

Aşağıda klavyeden girilen bir sayının 4 rakamına bölünmesinde kalanı gösteren bir program verilmiştir.

```
#include <stdio.h>
int main() {
    int sayi;
    printf("Bir Sayı Giriniz:");
    scanf("%d",&sayi);
    switch(sayi%4) { //kontrol ifadesi bir işlem içerir
    case 3:
        printf("Sayı 4 rakamına bölündüğünde kalan 3'tür.\n");
        break;
    case 2:
        printf("Sayı 4 rakamına bölündüğünde kalan 2'dir.\n");
        break;
    case 1:
        printf("Sayı 4 rakamına bölündüğünde kalan 1'dir.\n");
        break;
    default: // Başka alternatif yoktur.
        printf("Sayı 4 Rakamına TAM Bölünür\n");
        break;
    }
    return 0;
}
```

Üçlü Koşul İşleci

Daha önce *İşleçler* başlığı altında işlenenlere ek olarak, ardışık işlemlerde (sequential operation) if talimatlarına gerek kalmadan bir koşula göre ifadeler (expression) işlenecek ise üçlü koşul işleci (conditional tenary operator) kullanılır. Aşağıda üçlü işlecin iki sözdizimi verilmiştir;

```
koşul ? doğruifadesi : yanlışifadesi;
değişken = koşul ? doğruifadesi : yanlışifadesi;
```

Aşağıda oy kullanma durumu bu işleçle işlenmiştir;

```
#include <stdio.h>
int main() {
  int yas;
  printf("Yaşınız?:");
  scanf("%d", &yas);
  (yas >= 18) ? printf("Oy Kullanabilirsin.") : printf("Oy Kullanamazsın!");
  return 0;
}
```

Aşağıda başka kullanımlarına ilişkin kod örneği verilmiştir;

```
    int sayi,tek,cift;
    printf("Bir Sayı Giriniz: ");
```

```
    scanf("%d", &sayi);
    tek= (sayi%2) ? 1 : 0;
    cift= tek ? 0 : 1;
    int sonuc1=tek ? sayi+30 : sayi+40;
    int sonuc2=cift ? sayi*30 : sayi*40;
```

İlişkisel Döngüler

İşlemciler iyi bir sayıcıdır. CPU içindeki kaydediciler (registers) sayma işlevini birçok matematiksel işlemi yapmak için de kullanılır. Belli problemlerin çözümünde, gerçekleştirilen belli adımların tekrarlanması ile gidilir. Bu tip problemler şimdiye kadar olan yöntemlerle yapılırsa, tekrar tekrar aynı kodu yazmak zorunda kalırız.

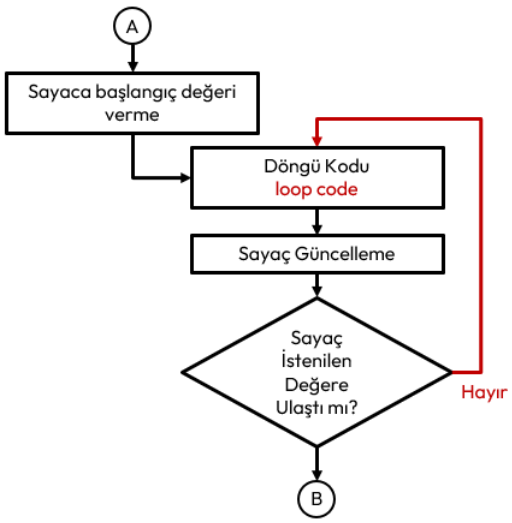
Yazdığımız kodları tekrar tekrar icra edebilmek için ilişkisel döngü (relational loop) talimatlarını kullanırız. Konsola 10 defa ismimizi yazdıran bir program örneğini ele alalım.

```
#include <stdio.h>
int main() {
    printf("ILHAN\n");
    printf("ILHAN\n");
    printf("ILHAN\n");
    printf("ILHAN\n");
    printf("ILHAN\n");
    printf("ILHAN\n");
    printf("ILHAN\n");
    printf("ILHAN\n");
    printf("ILHAN\n");
    printf("ILHAN\n");
    return 0;
}
```

Program incelendiğinde aynı `printf` talimatının tekrar tekrar yazıldığını görürüz. Bu istenmeyen bir durumdur. Bunu yapmamak için sayaç olarak kullanılan bir değişken kullanırız.

Sayaç Kontrollü Döngüler

Belirlenen bir sayaç (counter) değişkeninin istenilen bir değere ulaşıp ulaşmadığı kontrol ederek kodun tekrar tekrar icra edilmesi sağlanır.



Şekil 11. Sayaç Kontrollü Döngü İcra Akışı

Yapısal programlama öncesinde bu döngü `goto` talimatıyla sağlanır. İcra sırasını etiketlenen bir yer olarak değiştirmek için `goto` talimatı kullanılır. Etiketlere kimlik verilirken yine değişken kimliklendirme (identifier definition) kuralları uygulanır.

```
#include <stdio.h>
int main() {
```

```
    int sayac=0;
    Etiket: //Döngü Kodunun Başı ETİKETLENİR!
    printf("ILHAN %d\n",sayac);
    sayac++; //Sayaç Güncelleme
    if (sayac<10) //Sayaç Kontrolü
        goto Etiket; // İstenilen değere ulaşılmamış ise ETİKETE git.
    return 0;
}
```

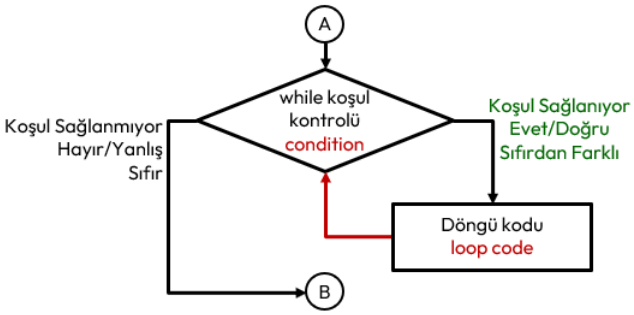
Yukarıdaki örnek incelendiğinde işaretli döngü bloğu üç adet talimat içeren mantıksal satır (logical sequence) olmasına karşın 30 adet fiziksel satır (physical sequence) icra edilmiştir.

C programlama dilinde ara seviye bir dil olduğundan bu talimat desteklenir. 1968 Yılında *Edsger W. Dijkstra* GOTO/JUMP TO ifadelerini zararlı olarak ilan edilmiştir[16]. GOTO kullanmamak için; `while`, `do-while` ve `for` kontrol talimatları yapısal programlamaya eklenmiştir.

While Talimatı

While döngüsü, koşul (condition) DOĞRU olduğu sürece döngü kodu (loop code) icra edilir. Tekrarlanacak kod, tek bir talimat (statement) olabileceği gibi bir kod bloğu da olabilir. Sözde kodu aşağıda verilmiştir;

```
while (koşul)
    döngü-kodu;
```



Şekil 12. While Talimatı İcra Akışı

Sayaç Kontrollü Döngüler başlığında verilen döngü, While talimatı ile aşağıdaki şekilde yazılabilir.

```
#include <stdio.h>
int main() {
    int sayac=0; //sayaca ilk değer verme
    while (sayac<10) { // döngü bloğu
      printf("ILHAN\n");
      sayac=sayac+1; //Sayaç Güncelleme
    }
    return 0;
}
```

Sayaç güncelleme ifadesi koşul ifadesine eklenerek program daha da kısaltılabilir. Her koşul kontrolü sonrası sayaç artırılacaktır.

```
#include <stdio.h>
int main() {
    int sayac=0; //sayaca ilk değer verme
    while (sayac++<10) // hem sayaç güncelleme hem de koşul kontrolü
        printf("ILHAN\n"); // tek talimat olduğundan blok kullanılmadı
    return 0;
}
```

[16] https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf

Yukarıdaki örnek incelendiğinde işaretli döngü bloğu bir adet talimat içeren mantıksal satır (logical sequence) içermesine rağmen 10 adet fiziksel satır (physical sequence) icra edilmiştir.

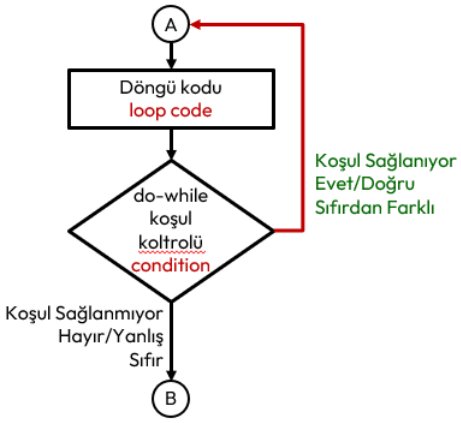
Do-While Talimatı

Do-While döngüsü, `do` ile `while` saklı kelimeleri arasındaki döngü kodu (loop code) koşul (condition) DOĞRU olduğu sürece tekrarlayarak icra eder. Tekrarlanacak kod tek bir talimat (statement) olabileceği gibi bir kod bloğu da olabilir. While döngüsünde koşul kontrolü en başta yapılır, bu döngüde ise döngü bloğu en az bir kez çalıştırıldıktan sonra koşul kontrolü yapılır. Sözde kodu aşağıda verilmiştir;

```
do
    döngü-kodu;
while (koşul);
```

Sayaç Kontrollü Döngüler başlığında verilen döngü, Do-While talimatı ile aşağıdaki şekilde yazılabilir.

```
#include <stdio.h>
int main() {
    int sayac=0; //sayaca ilk değer verme
    do  {  // döngü bloğu
      printf("ILHAN\n");
      sayac=sayac+1; //Sayaç Güncelleme
    } while (sayac<10); // Koşul kontrolü
    return 0;
}
```



Şekil 13. Do-While Talimatı ve İcra Akışı

Sayaç güncelleme ifadesi koşul ifadesine eklenerek program daha da kısaltılabilir.

```
#include <stdio.h>
int main() {
    int sayac=0; //sayaca ilk değer verme
    do
        printf("ILHAN\n"); // tek talimat olduğundan blok kullanıl
    while (sayac++<10); // hem sayaç güncelleme hem de koşul kontrolü
    return 0;
}
```

Yukarıdaki son örnek incelendiğinde işaretli döngü bloğu bir adet talimat içeren mantıksal satır (logical sequence) içermesine rağmen 10 adet fiziksel satır (physical sequence) icra edilmiştir.

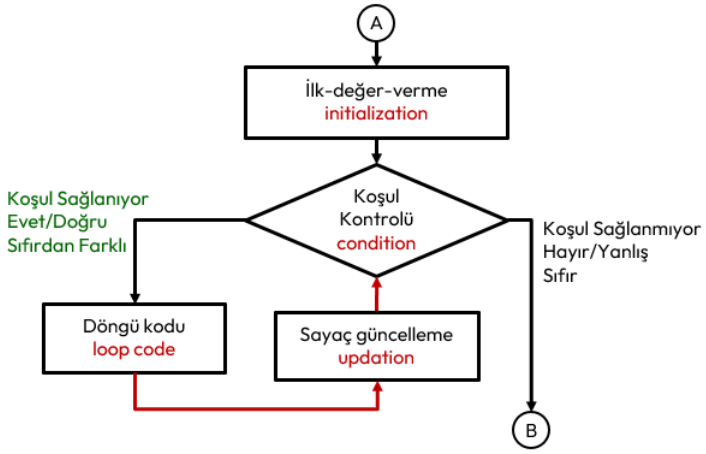
For Talimatı

For döngüsü özellikle tekrar edilen işlemlerin sayısı belli olduğunda kullanılan bir döngü yapısıdır. Sayaç kontrollü döngüler için en iyi seçimdir. Sözde kodu aşağıda verilmiştir.

```
for (ilk-değer-verme; koşul-kontrolü; sayaç-güncelleme)
    döngü-kodu;
```

For talimatı aşağıdaki şekilde icra edilir;

1. İlk değer verme ifadesi bir kez icra edilir.
2. Sonra koşul kontrolü yapılır. Eğer test sonucu DOĞRU ise döngü kodu icrasına geçilir. Aksi halde döngüden çıkılır.
3. Döngü kodu icra edilir.
4. Döngü kodu icrası bitince sayaç güncellemeleri yapılır ve tekrar ikinci adımdaki koşul kontrolüne dönülür.



Şekil 14. For Talimatı İcra Akışı

Sayaç Kontrollü Döngüler başlığında verilen döngü, for talimatı ile aşağıdaki şekilde yazılabilir.

```
#include <stdio.h>
int main() {
    int sayac;
    for (sayac=1; sayac<10; sayac++)
        printf("ILHAN\n");
    return 0;
}
```

Buradaki döngü mantıksal olarak iki satır talimattan oluşmaktadır. Ancak fiziksel olarak 10 satır icra edilmiştir. Aşağıda konsola yazdığımız her bir satırın başına satır numarası koyan bir program gösterilmektedir.

```
#include <stdio.h>
int main() {
    int sayac;
    for (sayac=0; sayac<10; sayac++)
        printf("%02d-ILHAN\n", sayac+1);
    return 0;
}
/*Program Çıktısı:
01-ILHAN
02-ILHAN
03-ILHAN
04-ILHAN
05-ILHAN
06-ILHAN
07-ILHAN
08-ILHAN
09-ILHAN
10-ILHAN

...Program finished with exit code 0
*/
```

Yukarıdaki örnek incelendiğinde işaretli döngü bloğu bir adet talimat içeren mantıksal satır (logical sequence) içermesine rağmen 10 adet fiziksel satır (physical sequence) icra edilmiştir.

For döngüsünde bir sayaç ile çalışılabileceği gibi birden fazla sayaç ile de çalışılabilir. Hem ilk değer verme hem de sayaç güncellemede birden fazla sayaca ilişkin işlem yapılabilir. Bunun için virgül işleci (operator) kullanılır.

```
#include <stdio.h>
int main() {
    int sayac1,sayac2;
    for (sayac1=0,sayac2=10; sayac1<10; sayac1++,sayac2--)
        printf("%02d-%02d-ILHAN\n",sayac1,sayac2);
    return 0;
}
/*Program Çıktısı:
00-10-ILHAN
01-09-ILHAN
02-08-ILHAN
03-07-ILHAN
04-06-ILHAN
05-05-ILHAN
06-04-ILHAN
07-03-ILHAN
08-02-ILHAN
09-01-ILHAN

...Program finished with exit code 0
*/
```

Aşağıda klavyeden girilen iki sayı aralığında tek ve çift sayıların toplamını bulan ve ekrana yazan programı verilmiştir.

```
#include <stdio.h>
int main() {
   int sayac, sayi1,sayi2;
   int tekToplam=0,ciftToplam=0;
   printf("İki Sayı Giriniz (10-25) gibi: ");
   scanf("%d-%d",&sayi1,&sayi2);
   if (sayi2<sayi1) { //sayi2, sayi1 den büyük olmalı.
      int temp=sayi1; //küçük ise yer değiştiriyoruz.
      sayi1=sayi2;
      sayi2=temp;
   }
   for (sayac=sayi1; sayac<=sayi2; sayac=sayac+1) {
      if (sayac%2==1) //sayac tek mi?
         tekToplam+=sayac;
      else
         ciftToplam+=sayac;
   }
   printf("Tektoplam: %d\nCiftToplam: %d\n", tekToplam, ciftToplam);
   return 0;
}
/*Program Çıktısı:
İki Sayı Giriniz (10-25) gibi: 9-17
Tektoplam: 65
CiftToplam: 52

...Program finished with exit code 0
*/
```

İç İçe Döngü Kodu

Döngüler içinde yer alan döngü kodu (loop code) bir başka döngüyü içerebilir. Örneğin ekrana satır ve sütun içeren bir şekil oluşturmak istediğimizde iç içe döngü kullanırız. Aşağıda buna ilişkin bir örnek verilmiştir. İlk for döngüsünün tekrarlanan kodu ikinci for döngüsüdür.

```
#include <stdio.h>
int main() {
  int satir,sutun;
  for (satir=0; satir<5; satir++)
  {
     for (sutun=0; sutun<4; sutun++)
     {
        printf("*");
     }
     printf("\n");
  }
  return 0;
}
/*Program Çıktısı:
****
****
****
****
****

...Program finished with exit code 0
*/
```

Bir başka örnek olarak elemanları, satır ve sütun çarpımları olan 4x5 boyutlarındaki matrisi ekrana yazdıran program;

```
#include <stdio.h>
int main() {
  int satir,sutun;
  printf("   Sütun:\t");//Başlığı Yazdıran Kısım
  for(sutun=0; sutun<3;sutun++)
    printf ("%3d ",sutun+1);
  printf("\n");

  //Matrisi Yazdıran Kısım
  for (satir=0; satir<5; satir++) {
    printf("%2d.Satir:\t",satir+1);
    for(sutun=0; sutun<3;sutun++)
      printf ("%03d ",sutun);
    printf("\n");
  }
  return 0;
}
/*Program Çıktısı:
   Sütun:         1   2   3
 1.Satir:       000 001 002
 2.Satir:       000 001 002
 3.Satir:       000 001 002
 4.Satir:       000 001 002
 5.Satir:       000 001 002

...Program finished with exit code 0
*/
```

Gözcü Kontrollü Döngüler

Döngüler her zaman bir sayaca bağlı çalıştırılmaz. Bir döngüden çıkış koşulu döngü kodu içerisinde üretilen bir değere bağlı ise bu değere gözcü değeri (sentinel value) adı verilir. Buradaki gözcü değeri beklenen bir değer dışındaki bir değerdir. Buna örnek olarak klavyeden `0` girilene kadar girilen rakamları toplayan bir program verilmiştir.

```
#include <stdio.h>
int main()
{
   int sayi; /* okunan tamsayı */
   int toplam=0; /* girilen sayıların toplamı. başlangıçta 0*/
   do
   {
      printf("Bir sayı giriniz (Bitirmek için 0): ");
      scanf("%d",&sayi);
      toplam+=sayi; // sayi 0 girilse bile toplam etkilenmez!
   } while (sayi != 0); //sentinel value 0
   printf("Girilen Sayıların Toplamı: %d",toplam);
   return 0;
}
/*Program Çıktısı:
Bir sayı giriniz (Bitirmek için 0): 10
Bir sayı giriniz (Bitirmek için 0): 20
Bir sayı giriniz (Bitirmek için 0): 30
Bir sayı giriniz (Bitirmek için 0): 0
Girilen Sayıların Toplamı: 60

...Program finished with exit code 0
*/
```

Aşağıda pozitif sayı girilmesini sağlayan bir kod örneği verilmiştir.

```
#include <stdio.h>
int main() {
   int sayi;
   int pozitifSayi=0;
   do /* Pozitif girilmesini zorluyoruz */
   {
      printf("Lütfen pozitif sayı giriniz: ");
      scanf("%d",&sayi);
      if (sayi<=0)
         printf("HATA: Pozitif Sayı GİRMEDİNİZ!\n");
   } while (sayi <= 0);
   pozitifSayi=sayi;
   printf("Girilen pozitif tamsayı: %d \n",pozitifSayi);
   return 0;
}
/*Program Çıktısı:
Lütfen pozitif sayı giriniz: -1
HATA: Pozitif Sayı GİRMEDİNİZ!
Lütfen pozitif sayı giriniz: 10
Girilen pozitif tamsayı: 10

...Program finished with exit code 0
*/
```

Aynı örnek bir başka şekilde aşağıdaki gibi kodlanabilir.

```
#include <stdio.h>
int main() {
   int sayi;
   int pozitifSayi=0;
```

```
    printf("Lütfen pozitif sayı giriniz: ");
    scanf("%d",&sayi);
    while (sayi <= 0) /* Pozitif girilmesini sağlıyoruz */
    {
          printf("HATA: Negatif sayı giriniz!\n");
          printf("Lütfen pozitif sayı giriniz: ");
          scanf("%d",&sayi);
    }
    pozitifSayi=sayi;
    printf("Girilen pozitif tamsayı: %d \n",pozitifSayi);
    return 0;
}
```

Döngülere İlişkin Örnekler

Bir ve kendisinden başka tam böleni olmayan sayıya asal denir. Klavyeden girilen pozitif bir tamsayının asal olup olmadığını ekrana yazdıran c programı aşağıdaki şekilde kodlanabilir;

```
#include <stdio.h>
int main()
{
    int sayi,asal=1;
    printf("Bir Sayı Giriniz:");
    scanf("%d",&sayi);
    for (int sayac=sayi-1; (sayac>1)&&(asal==1); sayac--)
        if (sayi%sayac==0) asal=0;
    if(asal)
        printf("Girilen %d sayısı asaldır.",sayi);
    else
        printf("Girilen %d sayısı asal değildir.",sayi);
    return 0;
}
```

Klavyeden girilen pozitif bir tamsayının asal çarpanlarını ekrana yazdıran c programı aşağıdaki şekilde kodlanabilir;

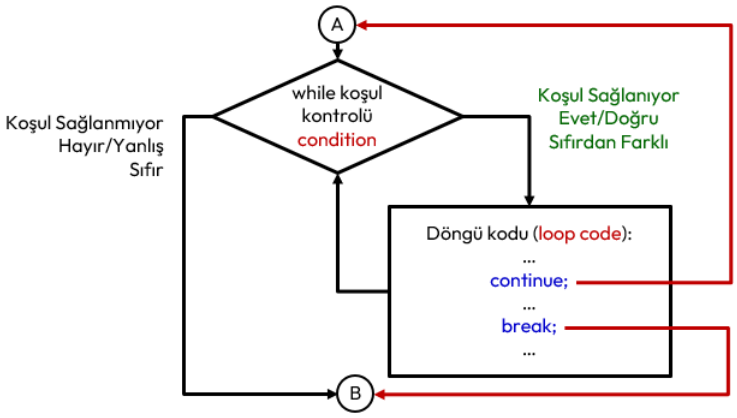
```
#include <stdio.h>
int main()
{
    int sayi;
    printf("Bir Sayı Giriniz:");
    scanf("%d",&sayi);
    for (int sayac=sayi; sayac>0; sayac--)
        if (sayi%sayac==0)
            printf("Asal çarpan:%d\n",sayac);
    return 0;
}
```

Dallanmalar

Dallanmalar (jump), programın akışını bir başka talimata yönlendiren talimatlardır.

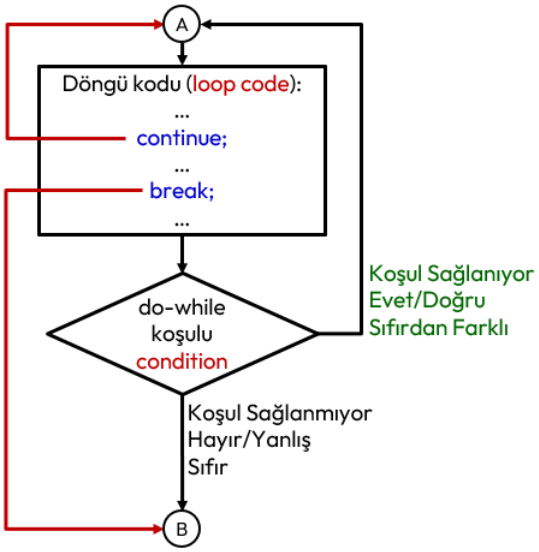
Continue ve Break Talimatları

Continue talimatı yalnızca geçerli yinelemeyi (iteration) sonlandırır ve sonraki yinelemelerle devam eder. Dolayısıyla sadece döngü kodları (loop code) içinde kullanılabilir. `break` Talimatı döngüyü sonlandırır ve program icrası döngü sonrası ilk talimattan devam eder. Bunu daha önce `switch` talimatında görmüştük. Bu talimat, aynı şekilde `while`, `do-while` ve `for` döngüleri ile kullanılabilir. `continue` Talimatı ise bir sonraki yinelemeyi yapmak için kullanılır. Bu talimatlarının döngü kodlarında kullanılması halinde icra akışı aşağıdaki şekilde verilmiştir.

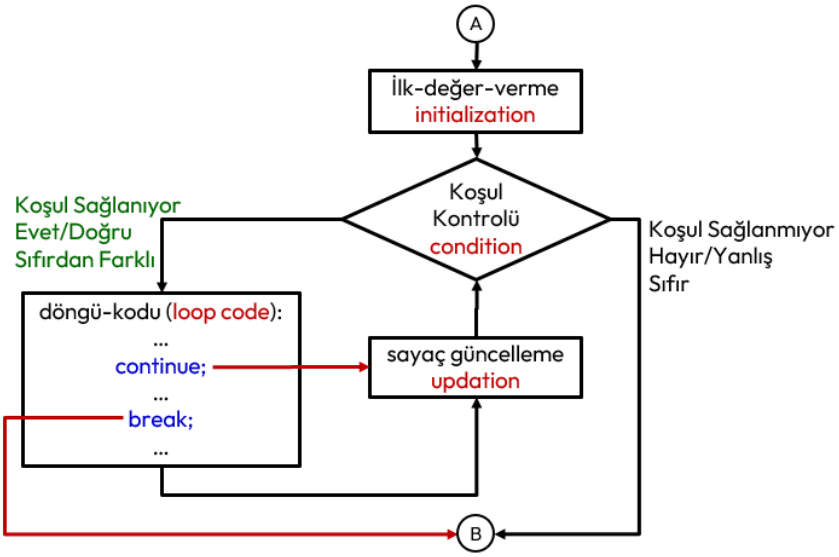


Şekil 15. While Döngü Kodunda Break ve Continue Talimatları İcra Akışı

While döngüsünde, döngü bloğunda `continue` kullanmadan önce koşulu değiştirecek bir talimat vermeliyiz. Aksi halde sonsuz döngüye girmiş oluruz. Do döngüsünde eğer `continue` talimatı bir şarta bağlanmalıdır. Yoksa sonsuz döngüye girilebilir.



Şekil 16. Do-While Döngü Kodunda Break ve Continue Talimatları İcra Akışı



Şekil 17. For Döngü Kodunda Break ve Continue Talimatları İcra Akışı

Benzer durum for döngüsü için de geçerlidir. Ancak for döngüsünde `continue` talimatı sonrasında bir sonraki yineleme için önce sayaç güncelleme ifadeleri icra edilir ve sonra koşul testi yapılır. Bu durum göz önüne alınarak bu talimat kullanılmalıdır. Bu talimatlar döngü kodu içinde amacımıza uygun

olarak birden çok kullanabiliriz. Aşağıda 0 ile 15 arasındaki sayıların beşe bölünenleri ve 10 dışındaki sayılar ekrana yazdıran bir program örneği verilmiştir.

```
#include <stdio.h>
int main() {
   int i;
   for (i=0; i<=15; i=i+1)
   {
      if (i<7) //i'nin değeri 7 den küçük ise
            continue;  //bir sonraki yinelemeye (iteration) geç
      if (i==10) //i'nin değeri 10 ise
         continue;  //bir sonraki yinelemeye (iteration) geç
      if (i%5==0) // i'nin değeri 5 in katı ise
         continue;  //bir sonraki yinelemeye (iteration) geç
      printf ("i = %d \n",i);
   }
   return 0;
}
/*Program Çıktısı:
i = 7
i = 8
i = 9
i = 11
i = 12
i = 13
i = 14

...Program finished with exit code 0
*/
```

Yukarıdaki örnek incelendiğinde işaretli döngü bloğu üç adet talimat içeren mantıksal satır (logical sequence) içermesine rağmen 42 adet fiziksel satır (physical sequence) icra edilmiştir. Bazı `continue` talimatları `printf` talimatının icrasını engellemiştir.

Aşağıda pozitif sayı girilmesini zorlayan sonsuz döngü (infinitive loop) içeren program verilmiştir. Programda döngü, pozitif sayı girildiğinde break talimatı ile kırılmaktadır.

```
#include <stdio.h>
int main() {
    int sayi;
    int pozitifSayi=0;
    while (1) /* Sonsuz döngü */
    {
        printf("Lütfen pozitif sayı giriniz: ");
        scanf("%d",&sayi);
        if (sayi<=0)
            printf("HATA: Pozitif Sayı GİRMEDİNİZ!\n");
        else
            break; //döngüden çık
    }
    pozitifSayi=sayi;
    printf("Girilen pozitif tamsayı: %d \n",pozitifSayi);
    return 0;
}
```

Return Talimatı

Yapısal programlamada ana fonksiyonun (main function) sonunda çokça kullandığımız bu talimat, bir fonksiyonun üreteceği ya da belirlenen değeri geri döndürür. Kullanıldığı yerden fonksiyon bloğu dışına çıkılır.

Aşağıda üç defa negatif sayı girilmesi halinde programı sonlandıran bir program verilmiştir.

```c
#include <stdio.h>
int main() {
   int sayi;
   int pozitifSayi=0;
   int sayac=0; //kaç defa pozitif olmayan sayı girildi.
   do /* Pozitif girilmesini zorluyoruz */
   {
      printf("Lütfen pozitif sayı giriniz: ");
      scanf("%d",&sayi);
      if (sayi<=0) {
         printf("HATA: Pozitif Sayı GİRMEDİNİZ!\n");
         sayac++;
         if (sayac==3) {
             printf("Üç defa negatif sayı girdiniz. ");
             printf("Programdan Çıkılıyor.\n");
             return 1; /* ana fonksiyondan çıkılarak
                          işletim sistemine 1 gönderiliyor. */
         }
      }
   } while (sayi <= 0);
   pozitifSayi=sayi;
   printf("Girilen pozitif tamsayı: %d \n",pozitifSayi);
   return 0;
}
/*Program Çıktısı:
Lütfen pozitif sayı giriniz: -1
HATA: Pozitif Sayı GİRMEDİNİZ!
Lütfen pozitif sayı giriniz: -2
HATA: Pozitif Sayı GİRMEDİNİZ!
Lütfen pozitif sayı giriniz: -3
HATA: Pozitif Sayı GİRMEDİNİZ!
Üç defa negatif sayı girdiniz. Programdan Çıkılıyor.

...Program finished with exit code 1
*/
```

Goto Talimatı

Tanımlı bir etikete program akışını yönlendiren `goto` talimatıdır. *Sayaç Kontrollü Döngüler* başlığında örneği verilmiştir.

FONKSİYONLAR

Fonksiyon Nedir?

Yapısal Programlama mantığında çözülecek problem genel olarak fonksiyonların birbirini çağırmasıyla yapıldığı anlatılmıştı. Yani kodlaması yapılacak işi birbirini çağıran fonksiyonlar yazarak ana fonksiyondan bu fonksiyonları çağırarak yaparız.

Ayrıca yazılacak programda birbiriyle ilgili fonksiyonlar bir araya toplanarak ayrı bir dosyada tutulur. Bu durum *Modüler Programlama* başlığı altında anlatılmıştı. Bu bölüme kadar gördüğümüz kodlarda çözümü oluşturan tüm tasarım parçaları ana fonksiyon olan `main()` fonksiyonu içerisine çözülmüştür. Bir bilgisayar programı, genel olarak, tek başına tüm görevleri yerine getiren tek bir main fonksiyonundan oluşmaz. Bu durumda ana problemin büyüklüğüne göre ana fonksiyon bloğumuz uzar, okunabilirliği azalır, müdahale etmek zorlaşır. Bu tip büyük programları kendi içerisinde, her biri özel bir işi çözmek için tasarlanmış parçacıklarından oluşturmak daha mantıklı olacaktır. Yani böl ve yönet (divide and conquer) tekniği uygulanır. Çünkü büyük bir problemin toptan çözülmesi, problemin parçalar halinde çözülmesinden her zaman daha yorucu ve zordur.

Yazılım geliştirmede problemi büyük ana parçalara ayırırız. Daha sonra bu büyük parçaları daha küçük parçalara ayırarak çözeriz. Buna yukarıdan aşağıya (top down) yaklaşım adı verilir.

C dilinde fonksiyonlar;
- Fonksiyonlar, belirli bir görevi gerçekleştiren bir kod bloğudur.
- Parametreli alabilir, girdilerle ya da parametresiz bir şeyler yapar ve ardından cevabı verebilir. Kısaca bilgiyi fonksiyona argümanlar ile aktarırız.
- Her fonksiyonun bir kimliği vardır. (*Değişken Kimliklendirme Kuralları* burada da geçerlidir.)
- Bir fonksiyon program içerisinde istenildiği kadar farklı yerlerden ulaşılarak çalıştırılabilir ya da çağrılabilir. Bir başka deyişle tekrar kullanılabilir.
- Bir fonksiyon, çağıran koda bir değer döndürebilir.
- Bir fonksiyon, bir başka fonksiyondan çağrılabilir.

C dilinde iki tip fonksiyon bulunmaktadır; başlık dosyalarında bize sunulan hazır fonksiyonlar ve programcının tanımlayacağı kullanıcı tanımlı fonksiyonlar (user defined function).

Hazır Fonksiyonlar

C geliştiricileri, programcıların kullanmaları için, önceden yazılmış olan hazır fonksiyonlar hazırlamışlardır. Hazır fonksiyonlar teknik olarak C dilinin parçası değillerdir. Yalnızca standart hale getirilmişlerdir.

Programcı, kullanmak istediği hazır fonksiyonları; hangi modülde ise o modüle ilişkin başlık (header) dosyasını `#include` ön işlemci yönergesi (preprocessor directive) ile kendi projesine dahil eder. Bunu *C PROGRAMLAMA DİLİNE GİRİŞ* başlığı altında `stdio.h` başlığı için incelemiştik.

Aslında bu başlık dosyalarında;
- Hazır fonksiyonların sadece prototipleri (prototype) bulunur.
- Fonksiyonların kendileri yani derlenmiş halleri her işletim sistemi için ayrı oluşturulmuş kütüphane (library) dosyaları içerisinde bulunur.
- Derleme işlemi sırasında bağlayıcı (linker) tarafından bu fonksiyonlar icra edilebilir dosyaya (executable file) dahil edilir.

Math.h Başlık Dosyası

Matematiksel iş ve işlemleri gerçekleştirmek için standart hale getirilmiş bir başlık dosyasıdır.

Fonksiyon prototipi	Açıklama	Örnek
`double sqrt(double);`	$\sqrt{x}$	sqrt(900.0)=30.0
`double exp(double);`	e^x	exp(1.0)=2.718282

Fonksiyon prototipi	Açıklama	Örnek
`double log(double);`	$ln(x)$	log(2.718282)=1.0
`double log10(double);`	$log(x)$	log10(1.0)=0.0
`double fabs(double);`	$\|x\|$	fabs(-5.0)=5.0
`double ceil(double);`	x'i kendisinden büyük en küçük kesirsiz sayıya yuvarlar.	ceil(-9.8)=-9.0 ceil(9.2)=10.0
`double floor(double);`	x'i kendisinden küçük en büyük kesirsiz sayıya yuvarlar.	floor(-9.8)=-10.0 floor(9.2)=9.0
`double pow(double x,double y);`	x^y	pow(2.0,3.0)=8.0
`double fmod(double x,double y);`	Kayan noktalı sayılarda x'in y'ye bölümünden kalanı verir.	fmod(13.657,2.333)=1.1992
`double sin(double);`	Radyan cinsinden x'in sinüsü	sin(0.0)=0.0
`double cos(double);`	Radyan cinsinden x'in kosinüsü	cos(0.0)=1.0
`double tan(double);`	Radyan cinsinden x'in tanjantı	tan(0.0)=0.0

Tablo 20.Math.h Başlık Dosyası Fonksiyonları

Örnek olarak kenarları verilen bir üçgenin alanını hesaplayan bir program aşağıda verilmiştir.

```
/*AREA OF A TRIANGLE - HERON'S FORMULA */
#include <stdio.h>
#include <math.h>
int main()
{
    int kenar1,kenar2,kenar3;
    double alan,kenarlarToplamininYarisi;
    printf("Üçgenin Kenarlarını Giriniz:");
    scanf("%d%d%d",&kenar1,&kenar2,&kenar3);

    if ((kenar1+kenar2 <= kenar3) ||
       (kenar2+kenar3 <= kenar1) ||
       (kenar1+kenar3 <= kenar2) ) {
        printf("Böyle bir üçgen olamaz.\n");
        return 1;
    }

    kenarlarToplamininYarisi=(kenar1+kenar2+kenar3)/2.0;
    alan=sqrt( kenarlarToplamininYarisi*
               (kenarlarToplamininYarisi-kenar1)*
               (kenarlarToplamininYarisi-kenar2)*
               (kenarlarToplamininYarisi-kenar3) );
    printf("Ucgenin alani: %.2f",alan );
    return 0;
}
/*Program Çıktısı:
Üçgenin Kenarlarını Giriniz:3 10 12
Ucgenin alani: 12.18

...Program finished with exit code 0
*/
```

Stdlib.h Başlık Dosyası

Bu kütüphane içerisinde; Alfa sayısal metinleri ilkel veri tiplerine veya bunun tersini yapan tür dönüşüm fonksiyonları (`atoi`, `itoa`, `atof`, `strtod`, ... ileride anlatılacaktır.), Çalıştırma anında bellek tahsisi ve tahsisli

belleğin serbest bırakılmasını sağlayan hafıza yerleşim fonksiyonları (`malloc`, `free`… ileride anlatılacaktır.) Rastgele sayı üretme fonksiyonları (`rand`, `srand`) bulunur. Şimdilik rastgele sayı üretme üzerinde durulacaktır.

Fonksiyon	Açıklama
`int rand();`	0 ile RAND_MAX (32767) arasında rastgele bir sayı üretir.
`int srand(unsigned int);`	`rand()` fonksiyonu belli bir başlangıç değerinden itibaren bir dizi matematiksel işlem sonucu rastgele bir değer üretir. `srand()` fonksiyonu, `rand()` fonksiyonunun başlangıç değerini farklı belirlemek için kullanılır.

Tablo 21. Stdlib.h Başlık Dosyası Fonksiyonları

Aşağıda bir zar için rastgele sayı üreten bir program verilmiştir.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int rastgeleZar;
   rastgeleZar=1 + rand() %6;
   /*
     rand() fonksiyonunun üreteceği sayıyı kalan işleci ile
     6 sayısına bmleriz ki 0 ile 5 arasında bir sayı elde edelim.
     Buna da 1 ekler isek zarın rakamları ortaya çıkar.
   */
   printf("Atılan Zar:%d",rastgeleZar);
   return 0;
}
```

Fakat programın her çalışmasında `rand()` aynı başlangıç değerini kullandığından aynı rastgele değerleri üretir. Bu nedenle aynı zar rakamı gelmiş olur. Bunu değiştirmek için `srand()` fonksiyonu kullanılır.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int rastgeleZar;
   int randBaslangicDegeri;

   printf("rand() için başlangıç değeri olabilecek bir sayı giriniz:");
   scanf("%d",&randBaslangicDegeri);
   srand(randBaslangicDegeri);

   rastgeleZar=1 + rand() %6;
   printf("Atılan Zar:%d",rastgeleZar);
   return 0;
}
```

Rastgele sayı her seferinde kullanıcıdan alınan sayıya göre hesaplandığından her seferinde farklı bir zar rakamı üretilmiş olacaktır. Ancak başlangıç değerinin her seferinde kullanıcıdan alınması bir başka problemdir. Bunun üstesinden gelmek için bilgisayarın saatini sayı olarak veren `time.h` başlık dosyasındaki `time()` fonksiyonu `NULL` parametresiyle kullanılabilir.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
   int rastgeleZar;
   int randBaslangicDegeri=time(NULL);
   srand(randBaslangicDegeri);   // srand(time(NULL)); olarak da kullanılabilir.
   rastgeleZar=1 + rand() %6;

   printf("Atılan Zar:%d",rastgeleZar);
   return 0;
```

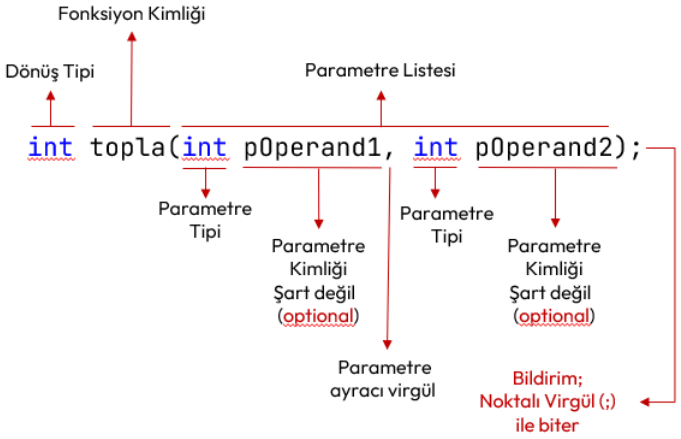
```
}
```

Kullanıcı Tanımlı Fonksiyonlar

Programcılar kendi fonksiyonlarını oluşturarak kodlarını daha kullanışlı hale getirirler. Bu tür fonksiyonlara kullanıcı tanımlı fonksiyonlar (user defined function) denilir. Kullanıcı tanımlı fonksiyonu oluşturmak ve kullanmak için bu üç unsuru bilmemiz gerekir;

- Fonksiyon Bildirimi: Bir fonksiyonu çalıştıran koda, çağıran program adı verilir. Çağıran program kullanılacak fonksiyonu bilmeli. Bunun için çağıran program öncesinde fonksiyon bildirimi (function declaration) yapılmalı veya prototipi (function prototype) tanımlanmalıdır. Bildirim derleyiciye böyle bir fonksiyon var demenin yoludur.
- Fonksiyon Tanımı: Fonksiyon tanımı (function definition), bir fonksiyonun girdileri, yaptığı işlem ve döndüreceği değeri içerecek şekilde başlık ve gövdesinin kodlanmasından oluşur. Bildirimi yapılan fonksiyonun derleme yapılabilmesi için eksiksiz bir şekilde tamamlar. Bildirime benzer başlık ve tırnaklı parantez `{ }` kod bloğu ve arasında yer alan kodlardan oluşur. Bildirim yapılmadan da fonksiyon tanımlanabilir.
- Fonksiyon Çağırma: Fonksiyonu çağırmak (function call/invoke function) için fonksiyonun kimliğini ve ardından parantez içindeki argüman listesini yazmanız yeterlidir.

Fonksiyon Bildirimi Nasıl Yapılır?



Şekil 18. Fonksiyon Bildirimi Örneği

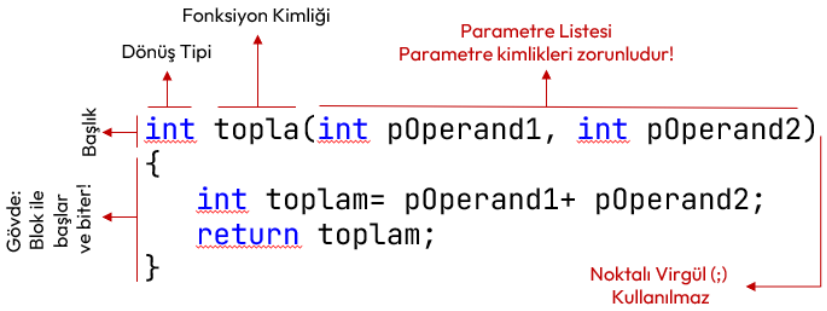
Yukarıda örneği verilen fonksiyon bildiriminde (function declaration); Dönüş tipi, işlevin döndürdüğü değerin veri tipidir. Bazı işlevler, herhangi bir değer döndürmeden istenen işlemleri gerçekleştirir. Bu durumda dönüş tipi `void` anahtar sözcüğüdür ve bu fonksiyonlar yordam (procedure) ya da alt rutin (subroutine) olarak adlandırılır.

```
/* Fonksiyon Bildirimleri */
int ikiSayiyiTopla(int,int); /* ikiSayiyiTopla kimlikli bir fonksiyon olduğu ve
                                bu fonksiyonun iki tamsayı parametre aldığı ve
                                bir tamsayı geri döndürdüğü derleyiciye bildiriliyor. */
void sayiyiKonsolaYaz(int);  /* sayiyiKonsolaYaz kimlikli bir fonksiyon olduğu ve
                                bu fonksiyonun bir tamsayı parametre aldığı ve
                                geri değer döndürmediği derleyiciye bildiriliyor. */
int konsoldanTamsayiOku();   /* konsoldanTamsayiOku kimlikli bir fonksiyon olduğu ve
                                bu fonksiyonun bir parametre almadığı ve
                                bir tamsayı geri döndürdüğü derleyiciye bildiriliyor. */
int main() {
   sayiyiKonsolaYaz(13);
   int toplam= ikiSayiyiTopla (10,20);
   //...
   return 0;
}
```

Fonksiyon adı, fonksiyonun benzersiz kimliğidir ve kimliklendirme, değişken kimliklendirme kuralları burada da geçerlidir. Bağımsız değişkenler parametre olarak adlandırılır. Bir fonksiyon çağrıldığı anda parametreye gerçek parametre ya da argüman adı verilir. Parametre listesi, bir işlevin parametrelerinin veri tipini, sırasını ve sayısını belirtir. Parametreler isteğe bağlıdır; yani bir fonksiyon hiçbir parametre içermeyebilir.

Fonksiyona ilişkin bir bildirim yapıldığında bir yerlerde böyle bir fonksiyon olduğu derleyiciye iletilmiş olur. Bildirim sonrasında artık onu çağırabiliriz. Tanımı daha sonradan yapılabilir. Bildirim yapılmaz ise fonksiyon tanımlanmalıdır.

Fonksiyon Tanımı Nasıl Yapılır?



Şekil 19. Fonksiyon Tanım Örneği

Bir fonksiyon bir değer döndürebilir. Dönüş tipi, işlevin döndürdüğü değerin veri tipidir.

Fonksiyon adı, fonksiyonun benzersiz kimliğidir ve kimliklendirmede, değişken kimliklendirme kuralları geçerlidir. Eğer daha önce fonksiyon bildirimi (function declaration) yapılmış ise aynı kimlik kullanılır.

Bağımsız değişkenler parametre olarak adlandırılır. Parametre listesi, bir işlevin parametrelerinin veri tipini, sırasını ve sayısını belirtir. Parametreler isteğe bağlıdır; yani bir fonksiyon hiçbir parametre içermeyebilir. Bildirimden farklı olarak her bir parametreye kimlik verilmelidir.

Fonksiyon Gövdesi, fonksiyonun ne yaptığını tanımlayan talimatları içerir. Yapısal programlama yapıldığından ilk önce değişkenler tanımlanmalı daha sonra bu değişkenleri işleyecek talimatlar yazılmalıdır. Fonksiyonda istenilen sonuca ulaşmak için kullanacağı adımlara karar verilir yani algoritması belirlenir.

Dönüş tipine uygun olarak bir değer, return talimatı ile geri döndürülmelidir. Return talimatı çoğunlukla fonksiyonundaki son talimat olarak görünür.

Return talimatı, bir fonksiyonun yürütülmesini sonlandırır ve kontrolü çağrıldığı fonksiyona geri verir. Yani programın icrasına devam etmek için çağrıldığı yere dönülür. Bir fonksiyonun tanımında yer alan dönüş tipi ile return talimatına ilişkin ifade sonucu aynı veri tipinde olmalıdır.

Fonksiyonun çağrıldığı yerde, fonksiyonun döndürdüğü değerin tipi ile eşleşen bir değişkene dönüş değeri atanabilir.

C dilinde bir fonksiyon gövdesinde bir başka fonksiyon tanımlanamaz!

Kullanıcı Tanımlı Fonksiyon Örnekleri

Aşağıda topla adlı bir kullanıcı tanımlı fonksiyona ilişkin program verilmiştir.

```
#include <stdio.h>

int topla(int,int); /* kullanıcı tanımlı topla fonksiyonu bildirimi
                       (declaration)/prototipi
                       Bu bildirimde int geri döndüren ve
                       int titpinde iki farklı parametre alan
                       topla fonksiyonu derleyiciye bildirildi.*/
int main () {
   int sonuc;
```

```
   /* ÇAĞRI ORTAMI:
      ana fonksiyon içinde topla fonksiyonu çağrılacak.
   */
   topla(1,2); /* topla fonksiyonu çağrılıyor ama geri
                  döndürdüğü değer kullanılmıyor. */
   printf("toplam: %d\n", topla(2,3)); /* topla fonksiyonu çağrılıyor.
                                          Geri döndürdüğü değer printf
                                          fonksiyonuna
                                          argüman olarak giriyor. */
   sonuc= topla(3,4);  /* topla fonksiyonu çağrılıyor.
                          Geri döndürdüğü değer
                          sonuc değişkenine atanıyor. */
   printf("toplam: %d\n",sonuc);
   sonuc= topla(1,2)+topla(3,4); /* topla fonksiyonu iki kez çağrılıyor.
                                    Her bir çağrıdaki geri dönüş değeri
                                    geçici bir yerde saklanıp toplamı
                                    sonuc değişkenine atanıyor. */
   printf("toplam: %d\n",sonuc);
   sonuc= topla(topla(1,2),3)+topla(4,5); /* ... */
   printf("toplam: %d\n",sonuc);
   return 0;
}
/*
 Kullanıcı tanımlı topla fonksiyonu burada tanımlanıyor (definition):
*/
int topla(int pOperand1,int pOperand2) //Fonksiyon başlığı
{ //Fonksiyon bloğu başlangıcı
   int toplam=pOperand1+ pOperand2;
   return pOperand1+ pOperand2;
} //Fonksiyon bloğu bitişi
/*Program Çıktısı:
toplam: 5
toplam: 7
toplam: 10
toplam: 15

...Program finished with exit code 0
*/
```

Aşağıda pozitif sayı girilmesini garanti ettikten sonra sayının 10 sayısına bölünebilirliğini bulan program verilmiştir.

```
#include <stdio.h>

int pozitifSayiOku(); /* parametre almayan bir fonksiyon bildirimi (declaration).
                         Bu fonksiyon geri int değer döndürüyor. */
int onaBolunurMu(int); /* int veri tipinde parametre alan
                          bir başka fonksiyon bildirimi (declaration)
                          bu fonksiyonda int tipinde veri geri döndürüyor. */
int main () {

    int birPozitifSayi=pozitifSayiOku();
    if (onaBolunurMu(birPozitifSayi))
        printf("Girilen Pozitif Sayı 10 ile bölünebilir.");
    else
        printf("Girilen Pozitif Sayı 10 ile tam bölünmez.");
    return 0;
}
/* Bildirimi yapılan “pozitifSayiOku” fonksiyonunun tanımı (definition): */
int pozitifSayiOku()
{
    int okunan;
```

```
    do {
        printf("Lütfen pozitif sayı giriniz: ");
        scanf("%d",&okunan);
        if (okunan<=0)
            printf("HATA:Pozitif Sayı GİRMEDİNİZ!\n");
    } while (okunan <= 0);
    return okunan;
}
/* Bildirimi yapılan “onaBolunurMu” fonksiyonunun tanımı (definition): */
int onaBolunurMu(int p)
{
    return (p%10)==0;
}
/*Program Çıktısı:
Lütfen pozitif sayı giriniz: 30
Girilen Pozitif Sayı 10 ile bölünebilir.

...Program finished with exit code 0
*/
```

Örnek programda görüldüğü üzere yapısal programa kapsamında bir programın iskeleti oluşmuştur. Ana program içinde problemin çözümü, çeşitli fonksiyonların birbirini çağırması ile yapılmıştır. Her bir fonksiyon içinde işlenecek veriye ilişkin değişkenler (en basit veri yapısı) tanımlanmış ve bu veriler ile bu veri yapılarını işleyen kontrol işlemleri birbirinden ayrılmıştır. Okunaklılık çok yüksek bir kod elde edilmiştir.

Bunların dışında hiçbir değer geri döndürmeyen fonksiyonlar da tanımlayabiliriz. Bunlara diğer dillerde prosedür (procedure) adı da verilir. Bu durumda değeri olmayan veri tipi olan `void` anahtar kelimesi kullanılır. Bu tür fonksiyonların da geri dönüş talimatı yalnızca `return;` şekilde yazılır.

```
#include <stdio.h>

void printMenu(); /* değer geri döndürmeyen bir fonksiyon bildirimi (declaration)
                     aynı zamanda parametre de almıyor. */
int main() {
    int secim;
    do {
        printMenu();
        scanf("%d",&secim);
        switch(secim) {
            case 1:
                printf("Verileri Sakladım.\n");
                break;
            case 2:
                printf("Verileri Okudum.\n");
                break;
            case 0:
                printf("Programdan Çıkılıyor...\n");
                break;
            default:
                printf("Tekrar Seçim Yapınız!\n");
        }
    } while (secim!=0);
    return 0;
}
/* Bildirimi yapılan “printMenu” fonksiyonunun tanımı (definition): */
void printMenu() {
    printf("\n");
    printf("1-Verileri Yaz.\n");
    printf("2-Verileri Oku.\n");
    printf("0-ÇIKIŞ.\n");
    return;
```

```
}
```

Çağrı Kuralı

Çağrı kuralı (calling convention), bir fonksiyon çağrısıyla karşılaşıldığında argümanların fonksiyona hangi sıraya göre iletileceğini belirtir. İki olasılık vardır; Birincisi argümanlar soldan başlanarak sağa doğru fonksiyona geçirilir. İkincisi ise C dilinde kullanılır ve argümanlar sağdan başlanarak sola doğru fonksiyona geçirilir.

```
fonksiyon(a,b,c,d,e);
```

Yukarıdaki örnek incelendiğinde argümanların hangi sırada fonksiyona geçirildiğinin bir önemi yoktur. Ancak aşağıdaki verilen örnekteki durumu inceleyelim;

```
int a = 1;
printf("%d %d %d", a, ++a, a++);
```

Bu kodun çıktısı `1 2 3` olarak beklenir ancak çağrı kuralından ötürü çıktı `3 3 1` olur. Bunun nedeni C'nin çağrı kuralının sağdan sola olmasıdır. Bunu şöyle açıklayabiliriz;

1. Fonksiyona sağdan ilk argüman olarak a değişkeni değeri 1 geçirilir.
2. Ardından a++ ifadesi ile a değişkeni 2'ye yükseltilir.
3. Sonra ++a ifadesi ile a değişkeni 3'e yükseltilir.
4. Fonksiyona ikinci argüman olarak 3 geçirilir.
5. Fonksiyona son olarak a'nın son değeri olan 3 geçirilir.

Depolama Sınıfları

Depolama sınıfları (storage classes) bir değişkenin ömrünü (lifetime), görünürlüğünü (visibility), bellek konumunu (memory location) ve başlangıç değerini (initial value) belirlemek için kullanılır. Bu özellikler temel olarak bir programın çalışma süresi boyunca belirli bir değişkenin varlığını izlememize yardımcı olan kapsam (scope), görünürlüğü ve yaşam süresini içerir.

Yapısal programlama ve fonksiyon kavramının bilgisayarlarda kullanılmasıyla birlikte işlemciler de de değişiklikler olmuştur. *TEMEL BİLGİSAYAR KAVRAMLARI* başlığı altında belirtilen işlemci kaydediciler dışında yeni kaydediciler de eklenmiştir;

– Verilerle icra edilen kod farklı bellek bölgesinde bulunur. Bu nedenle verileri işaret eden veri bellek adreslerini tutan kaydedicisi (data memory register).
– Yığın bellek kaydedicisi (stack register), fonksiyon çağrıları sırasında mevcut işlemci durumu ve yerel değişkenler gibi bazı işlemleri depolamak için kullanılan belleği gösteren (stack pointer) adresi tutan kaydedicidir.
– Bayrak kaydedicisi (flag register), işlemcinin durumunu veya sıfır (zero) bayrağı, taşıma (carry) bayrağı, işaret (sign) bayrağı, taşma (overflow) bayrağı, eşlik (parity) bayrağı, yardımcı taşıma (auxiliary carry) bayrağı ve kesme etkinleştirme (interrupt enable) bayrağı gibi çeşitli işlemlerin sonucunu tutmak için kullanılan özel bir kayıt türüdür.
– Genel amaçla kullanılan kaydediciler (general purpose registers).

Eklenen bu kaydediciler ile derlenen her bir program, dört bellek bölgesinden (segment) oluşur;

1. Kod bellek ya da kaynak koda ithafla metin bellek (code segment / text segment) icra edilecek emirleri içeren bellek.
2. Programın işleyeceği verileri tutan veri belleği (data segment).
3. Fonksiyonların çağırmadan (call) önce mevcut işlemci durumu ve yerel değişkenler gibi bazı işlemleri depolamak için ve fonksiyondan dönülünce (return) işlemciyi ve çağrı ortamını eski hale getirmek için kullanılan yığın bellek (stack segment). Bu bellek bu nedenle son giren veri ilk çıkar (last in first out-LIFO) mantığıyla çalışır.
4. Programın icrası sırasında dinamik olarak eklenip çıkarılan veriler için hurdalık gibi kullanılan öbek bellek (heap segment). İleride *DİNAMİK BELLEK YÖNETİMİ* başlığında anlatılacaktır.

Depolama sınıfları tanımlanan değişkenleri hangi bellek bölgesinde yer alacağını belirler. Dört tür depolama sınıfı vardır; `auto`, `extern`, `static` ve `register`.

Auto Depolama Sınıfı

Otomatik (auto) depolama sınıfı, bir fonksiyon veya blok içinde kimliklendirilmiş tüm değişkenler için varsayılan (default) depolama sınıfıdır. Bu nedenle, C dilinde program yazarken `auto` anahtar sözcüğü nadiren kullanılır. Otomatik değişkenlere;

- Yalnızca blok içinden erişilebilir.
- Değişkenin kapsamı içinde bulunduğu blok ile sınırlıdır.
- Yığın (stack) bellekte tutulur.

```
#include <stdio.h>
int main() {
    auto int a = 32;
    int b=20;
    /*
     int a=32; ile eşdeğerdir.
     a ve b değişkenleri bu fonksiyon bloğu ve iç bloklarda geçerlidir.
     Yani faaliyet alanları (scope) bu fonksiyon ile sınırlıdır.
    */
    if (a==32){
        int b=10; /* Ana fonksiyon bloğunda tanımlı b değişkenine
                     artık ulaşılamaz. */
        int c=50;
        /*
          auto int b; ile eşdeğer
          if bloğunda ve tanımlanabilecek iç bloklarda geçerlidir.
        */
        a=16;// Bu blokta a da geçerlidir.
        printf("a:%d, b:%d c:%d\n",a,b,c); // Çıktı: a:16, b:10 c:50
   }
   /* if bloğu dışında sadece ana fonksiyon bloğunda tanımlı
      değişkenlere ulaşılabilir. */
   printf("a:%d, b:%d\n", a,b); // Çıktı: a:16, b:20
   return 0;
}
```

Static Depolama Sınıfı

Statik (static) depolama sınıfı yaygın olarak kullanılan statik değişkenleri kimliklendirmek için kullanılır. Statik değişkenler, kapsam (scope) dışına çıktıktan sonra bile değerlerini koruma özelliğine sahiptir! Statik değişkenler;

- Kapsamları yani geçerli olduğu yer, tanımlandıkları fonksiyonla ya da blokla sınırlıdır.
- Kapsamlarındaki son kullanımlarının değerini korur.
- Program genelinde geçerli olan evrensel (global) statik değişkenlere programın herhangi bir yerinden erişilebilir.
- Veri bellekte (data segment) yer alır. Derleme sırasında bu bellek hep sıfırla doldurulduğundan ilk değerler (initial value) hep sıfırdır.

Statik (static) olarak tanımlanan değişken, program başladığında veri bellekte oluşturulup program bitene kadar aynı bellek bölgesini kullanan değişkendir.

```
#include <stdio.h>
void func() {
    int sayac = 0;
    for (sayac = 1; sayac < 10; sayac++)
    {
        static int statikOlan = 5;
        int statikOlmayan = 10;
        statikOlan++;
```

```
        statikOlmayan++;
        printf("sayaç: %d, staticOlan: %d\n", sayac,statikOlan);
        printf("sayaç: %d, statikOlmayan: %d\n", sayac, statikOlmayan);
    }
    //Burada statikOlan değişkene erişilemez.
}
int main() {
    func();
    return 0;
}
/*Program Çıktısı:
sayaç: 1, staticOlan: 6
sayaç: 1, statikOlmayan: 11
sayaç: 2, staticOlan: 7
sayaç: 2, statikOlmayan: 11
sayaç: 3, staticOlan: 8
sayaç: 3, statikOlmayan: 11
sayaç: 4, staticOlan: 9
sayaç: 4, statikOlmayan: 11

...Program finished with exit code 0
*/
```

Programı çalıştığında `statikOlan` değişken yalnızca for bloğu içinde geçerli olduğu ve her kullanımda değerini koruduğu görülmektedir.

Harici Depolama Sınıfı

Harici (extern) depolama sınıfı bize basitçe değişkenin kullanıldığı blokta değil başka bir yerde tanımlandığını söyler. Harici değişkenler;

- Bu değişkenlere değerler tanımlandığı yerde değil farklı bir blokta atanır ve üzerinde değişiklik yapılabilir.
- Bir global değişken, `extern` anahtar sözcüğünü kimliklendirme önüne yerleştirerek harici de yapılabilir.
- Bu değişkenler de veri bellekte (data segment) yer alır. Derleme sırasında bu bellek hep sıfırla doldurulduğundan ilk değerler hep sıfırdır.

Bir harici (extern) değişkene bildirim (declaration) yapıldığında bir başka dosyada tanımlanmış (definition) yapıldığı kabul edilir. Bunun için iki farklı kaynak kod dosyası gereklidir. Birinci kaynak kod dosyası (`xtrn.c`) aşağıda verilmiştir.

```
//XTRN.C
int externDegisken; //Diğer dosyalarda kullanılacak değişken tanımlandı.
void funcExtern() //Diğer dosyalarda kullanılacak fonksiyon tanımlandı.
{
    externDegisken++;
}
```

Bu dosyadaki değişken ve fonksiyonları kullanan bir başka kaynak kod (`main.c`) aşağıda verilmiştir;

```
#include <stdio.h>
extern void funcExtern(); // bir başka dosyada tanımlandı. Burada bildirimi yapıldı.
int main()
{
   extern int externDegisken; // bir başka dosyada tanımlandı. Burada bildirimi yapıldı.
   funcExtern();
   printf("extern: %d\n", externDegisken);
   externDegisken = 2;
   printf("extern: %d\n", externDegisken);
   return 0;
}
```

Bu iki dosyayı birlikte derlemek için `gcc xtrc.c main.c -o main.exe` komutu ile derleyiciye iki farklı dosya parametre olarak verilir.

Kaydedici Depolama Sınıfı

Kaydedici (register) depolama, otomatik değişkenlerle aynı işlevselliğe sahiptir. Tek fark, derleyicinin, eğer boş bir işlemci kaydedicisi mevcutsa değişken olarak o kaydedicinin kullanılmasıdır. Genel amaçlı kaydediciler bu amaçla kullanılır. Bu da bu değişkenlerin, bellekte saklanan değişkenlerden çok daha hızlı olmasını sağlar.

Eğer boş bir CPU kaydedicisi mevcut değilse, değişken yalnızca bellekte saklanır. `register` anahtar sözcüğüyle tanımlanan değişken nitelendirilir. `register` Anahtar sözcüğü ile sadece bloklar içinde tanımlanan yerel (local) değişkenler nitelendirilebilir, evrensel (global) değişkenler bu anahtar sözcük ile kullanılamaz.

Aşağıda kaydedici (register) depolama sınıfın ilişkin örnek verilmiştir. Kuradaki sayma ve toplama işlemleri uygun olan işlemci kaydedicilerinde yapılır.

```
#include <stdio.h>
int main() {
    register int k, sum;
    for(k = 1, sum = 0; k < 1000; sum += k, k++);
    //döngü bloğu olmayan for talimatı örneği sonunda ; var!
    printf("%d\n",sum); //Çıktı:499500
}
```

Öncelikle C standardı tek bir değişkenle birden fazla depolama sınıfının kullanılmasını yasaklar. Aynı değişkene hem `static` hem `extern` niteleyicisi kullanılamaz. Depolama sınıfları aşağıdaki tabloda özetlenmiştir; Tablodaki çöp (garbage) kavramı tahsis edilen bellek içeriği o anda ne ise değişkenin o değeri alması anlamındadır.

Depolama sıfatı	Yer aldığı bellek bölgesi	Başlangıç Değeri	Kapsamı (scope)	Ömür (lifetime)
`auto`	Yığın Bellek (Stack Segment)	Çöp	Bulunduğu blok ve bu blok içindeki bloklar	Bulunduğu blok sonunda biter
`extern`	Veri Bellek (Data Segment)	Sıfır	Evrensel (tüm dosyalarda)	Program sonunda biter
`static`	Veri Bellek (Data Segment)	Sıfır	İçinde bulunduğu blok	Program sonunda biter
`register`	İşlemci kaydedicileri (CPU Registers)	Çöp	Bulunduğu blok ve bu blok içindeki bloklar.	Bulunduğu blok sonunda biter

Tablo 22. Depolama Sınıfları Özet Tablosu

Evrensel ve Yerel Değişken

C dilinde bir kaynak koda ait dosya içinde bir değişkene her yerden erişilmek istenirse evrensel (global) değişken olarak tanımlanır. Bu değişkenler herhangi bir fonksiyon bloğu içinde tanımlanmazlar. Tanımlandıkları yerden sonra kaynak kodun her yerinde kullanılabilirler. Bu nedenle genellikle dosyanın başında tanımlama yapılır.

```
#include <stdio.h>

int evrenselDegisken=10; /* evrenselDeğişken bu noktadan sonra
                            her yerde kullanılabilir.*/
void fonksiyon();
int main() {
    int yerelDegisken=20; /* Bu yerel değişken sadece main bloğu
                             içinde kullanılabilir. */

    printf("evrensel:%d, yerel:%d\n",evrenselDegisken,yerelDegisken);
```

```
    evrenselDegisken++; // evrensel değişken değiştiriliyor.
    yerelDegisken--; // yerel değişken değiştiriliyor.
    printf("evrensel:%d, yerel:%d\n",evrenselDegisken,yerelDegisken);
    fonksiyon();

    return 0;
}
void fonksiyon() {
   int fonksiyonYerelDegiskeni=30;
   evrenselDegisken++;
   printf("evrensel:%d, fonsiyonYerel:%d\n",evrenselDegisken,fonksiyonYerelDegiskeni);
   if (fonksiyonYerelDegiskeni==30) {
       int evrenselDegisken=100; /* Artık evrensel değişkene bu blokta
                                    ulaşılamaz. */
       printf("evrensel:%d, fonsiyonYerel:%d\n",
                            evrenselDegisken,fonksiyonYerelDegiskeni);
   }
}
/*Program Çıktısı:
evrensel:10, yerel:20
evrensel:11, yerel:19
evrensel:12, fonsiyonYerel:30
evrensel:100, fonsiyonYerel:30

...Program finished with exit code 0
*/
```

Burada `fonksiyon` içindeki if bloğunda yer alan değişken her ne kadar `evrenselDegişken` olarak kimliklendirilse de yerel değişkendir ve sadece bu if bloğu içinde geçerlidir. Yani tanımlamada mantıksal hata (logical error) yapılmıştır.

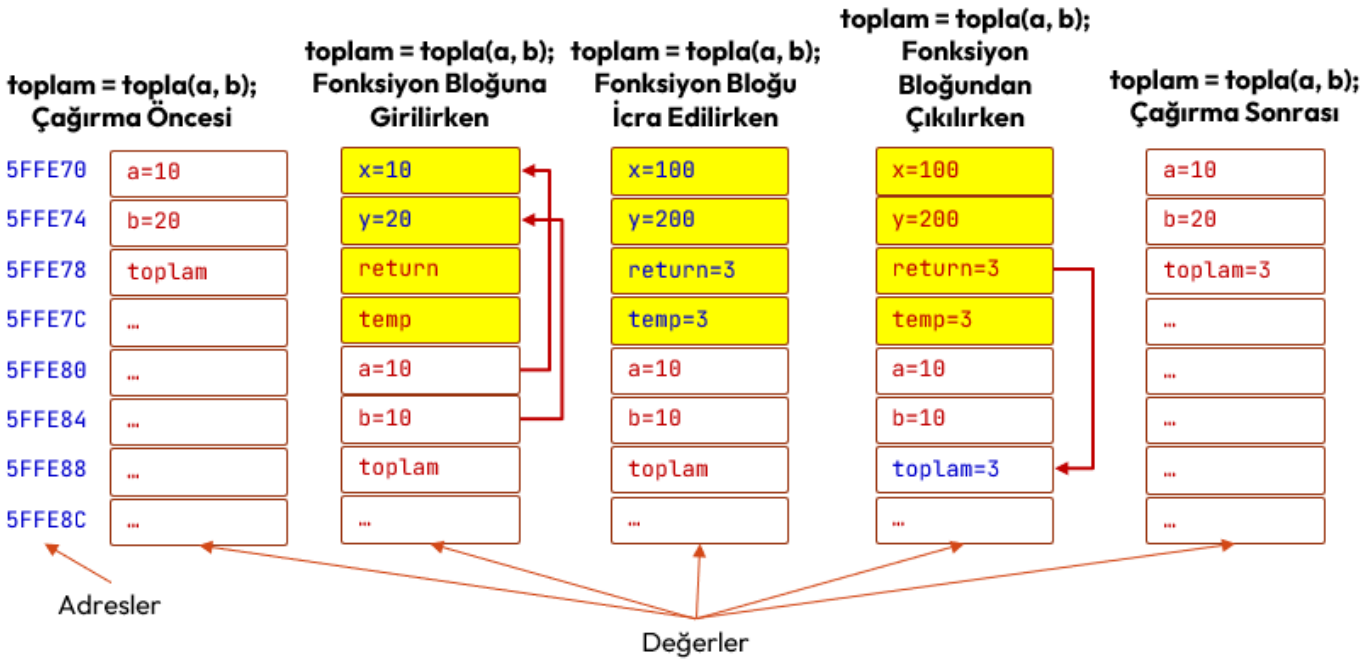
Fonksiyon Çağırma Süreci

Bir fonksiyon çağrıldığında nelerin yapıldığı aşağıdaki programdan anlaşılabilir;

```
#include <stdio.h>

int topla(int, int); /* İki tamsayıyı toplayan fonksiyon bildirimi/prototipi */

int main () {
    /* main() fonksiyonu içinde geçerli yerel (local) değişkenler*/
    int a = 10;
    int b = 20;
    int toplam = 0;
    printf("çağrı öncesi a: %d, b:%d, toplam:%d\n",a,b,toplam);
    toplam = topla(a, b);
    printf("çağrı sonrası a: %d, b:%d, toplam:%d\n",a,b,toplam);
}

int topla(int x, int y) { /* iki tamsayıyı toplayan fonksiyon tanımı*/
    /* Topla fonksiyonu yerel değişkenler */
    int temp= x+y;
    x=100; y=200;
    /*
      Burada değiştirilenler parametre değişkenleridir.
      main() içindeki yerel değişkenler değildir.
    */
    printf("çağrı içinde x: %d, y:%d\n",x,y);
    return temp;
}

/* Program çalıştırıldığında:
```

```
çağrı öncesi a:10, b:20 toplam:0
çağrı içinde x:100, y:200
çağrı sonrası a:10, b:20 toplam:30

...Program finished with exit code 0
*/
```

Aşağıda örneği verilen programda fonksiyon çağırma sürecindeki yığın (stack) bellek değişimi gösterilmektedir.



Tablo 23. Fonksiyon çağırma sürecinde yığın bellek

Yukarıdaki kod incelendiğinde;

- İcra sırası `topla(a,b)` fonksiyonuna geldiğinde fonksiyon çağrılmadan önce main fonksiyonu yerel değişkenleri a, b ve toplam değişkenleri yığın (stack) belleğe itilir (push).
- `topla(a,b)` fonksiyonu çağrıldığı anda fonksiyon bloğuna başlamadan parametrelere sırasıyla a ve b değişkenlerinin değerleri kopyalanarak fonksiyona geçirilir.
- `topla(a,b)` fonksiyonu icra edilirken fonksiyon yerelindeki `temp` ve parametre değişkenleri `x`, ve `y` istenildiği gibi değiştirilebilir. Geri dönüş değeri de belirlenir.
- `toplam=topla(a,b)` fonksiyon bloğundan çıkılırken geri dönüş değeri çağrı ortamındaki toplam değişkenine atanır ve fonksiyon için yığın (stack) belleğe itilen değerler geri çekilir.

Özyinelemeli Fonksiyonlar

Özyineleme (recursion), bir fonksiyonun kendi çağrısını yapma tekniğidir. Bu teknik, karmaşık sorunları çözülmesi daha kolay basit sorunlara ayırmanın bir yolunu sağlar. Özyineleme, yineleme (iteration), döngüler (loop) veya tekrar yerine geçebilecek çok güçlü bir tekniktir.

Özyinelemeli (recursive) algoritmalarda, tekrarlar fonksiyonun kendi kendisini kopyalayarak çağırması ile elde edilir. Bu kopyalar işlerini bitirdikçe kaybolur. Bir problemi özyineleme ile çözmek için problem iki ana parçaya ayrılır.

1. Cevabı kesin olarak bildiğimiz temel durum (base case)
2. Cevabı bilinmeyen ancak cevabı yine problemin kendisi kullanılarak bulunabilecek durum.

Faktöriyel örneğini inceleyelim; Matematikte, negatif olmayan bir tam sayı olan n'nin faktöriyeli, $n!$ ile gösterilir ve n 'den küçük veya ona eşit olan tüm pozitif tam sayıların çarpımıdır.

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

Negatif olmayan bir tam sayı olan n'nin faktöriyeli aynı zamanda n'nin bir sonraki daha küçük faktöriyelle çarpımına eşittir. Yani problemin tanımı yine kendisini içerir:

$$n! = n \cdot (n-1)!$$

Örneğin;

$$5! = 5 \cdot 4! = 5 \cdot 4 \cdot 3! = 5 \cdot 4 \cdot 3 \cdot 2! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$$

Aşağıda faktöriyel için girilen sayıdan sonra tüm sayıların çarpımıyla hesaplayan bir fonksiyon yazılmıştır.

```
#include <stdio.h>

unsigned int faktoriyel(unsigned int n)
{
    int sonuc=1,indis;
    for (indis=n;indis>0;indis--)
        sonuc=sonuc*indis;
    return sonuc;
}

int main() {
    unsigned int sayi,sonuc; //Pozitif tamsayı yanımı yapılmış değişkenler.
    printf("Pozitif Bir Sayı Giriniz:");
    scanf("%d",&sayi);
    sonuc=faktoriyel(sayi);
    printf("%d!=%d\n",sayi,sonuc);
    return 0;
}
```

Sıfır sayısı da pozitif bir sayıdır ama sonucum sıfır çıkmaması için 0 sayısının faktöriyeli 1 kabul edilir. Bu aslında faktöriyel problemi için temel durumdur (base case) ve boş ürün (emty product) kuralı olarak bilinir. Kısaca faktöriyel fonksiyonu aşağıdaki şekilde gösterilir;

$$n \in \mathbb{N} \; olmak \; üzere \; f(n) = n! = \begin{cases} 1, & n = 0 \\ n \cdot f(n-1), & n > 0 \end{cases}$$

Özyinelemeli (recursive) olarak ise bu fonksiyon aşağıdaki şekilde kodlanabilir;

```
unsigned int faktoriyel(unsigned int n)
{
    if (n==0) return 1;
    else return n*faktoriyel(n-1);
}
```

Girilen taban ve üs ile kuvvet hesaplama fonksiyonu özyinelemeli olarak aşağıda verilmiştir;

```
#include <stdio.h>

int kuvvet(int taban, int us) {
    if (taban == 0) { return 0; } //Taban 0 ise 0 dön
    if (us == 0) { return 1; } // Üs 0 ise 1 dön
    return taban*kuvvet(taban, us - 1);
}

int main(){
    int taban,us;
    printf("Tabanı Giriniz:");
    scanf("%d",&taban);
    printf("Üssü Giriniz:");
    scanf("%d",&us);
    printf("%d uzeri %d = %d", taban,us, kuvvet(taban,us) );
    return 0;
}
```

```
/*
Tabanı Giriniz:3
Üssü Giriniz:6
3 uzeri 6 = 729

...Program finished with exit code 0
*/
```

Satır İçi Fonksiyonlar

Fonksiyon Çağırma Süreci başlığında anlatıldığı üzere fonksiyonlar çağrılırken sürekli yığın (stack) belleğe it-çek yapılır. Bazı durumlarda bu işlem performans gereği istenmez. Bu durumda fonksiyonlar, satır içi fonksiyon (inline function) olarak tanımlanır. 2017 C sürümü ile gelen bu özellik, fonksiyona inline sıfatı (inline specifier) kullanılır.

```
#include <stdio.h>
inline int topla (int op1, int op2) {
    int toplam= op1+op2;
    return toplam;
}
int main() {
    int x=10, y=15, sonuc;
    sonuc = topla(x,y);
    printf("%d+%d=%d",x,y,sonuc);
    return 0;
}
```

Bu kod aşağıdaki şekle çevrilmiş gibi derlenir;

```
#include <stdio.h>
int main() {
    int x=10, y=15, sonuc;
    {
        int toplam= op1+op2;
        sonuc = toplam;
    }
    printf("%d+%d=%d",x,y,sonuc);
    return 0;
}
```

Bir fonksiyon aşağıda belirtilen durumlarda satır içi fonksiyon olarak derlenmez;

- Bir döngü içeriyorsa!
- Statik değişkenler içeriyorsa!
- Özyinelemeli ise!
- Dönüş türü `void` haricinde olup return ifadesi işlev gövdesinde mevcut değilse!
- Switch veya `goto` talimatı içeriyorsa!

Bu tip fonksiyon tanımlamadaki amacımız, fonksiyon çağırma yükünün azaltılması için işlemci kaydedicilerinin daha çok kullanılmasıdır. Bu da icra hızını yükseltir.

DİZİLER

Dizi Nedir?

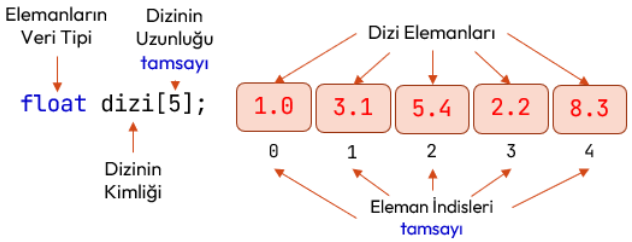
Şu ana kadar en basit veri yapıları (data structure) olan değişkenlerle karşılaştık. Programlamada kullanılan bir başka veri yapısı da dizilerdir (array). Matematikte diziler; bir sıralı elemanlardan oluşan bir listedir. Sıralı elemanların sayısına dizinin uzunluğu (array size) denir. Elemanların sırası indis (index) olarak adlandırılır. Kümenin aksine aynı elemanlar dizide farklı konumlarda birkaç kez bulunabilir. Elemanlara terim adı da verilir.

Örneğin Elemanları 1'den 10'a kadar sayıların karesinden oluşan bir dizi; $(a_k)_{k=1}^{10}, a_k = k^2$ veya $(a_1, a_2, a_3, \ldots, a_{10})$ veya $(1,4,9,\ldots,100)$ olarak gösterilir. Örneğin;

- $(\mathrm{K, İ, T, A, P})$ Dizisi ile $(\mathrm{P, A, T, İ, K})$ dizisi birbirinden farklıdır. Aynı elemanlardan oluşmasına rağmen elemanların sıraları farklıdır.
- $(1,3,5,1,2,1,7)$ Dizisinde birden farklı indiste aynı 1 elemanı vardır. Bu dizinin geçersiz olduğu anlamına gelmez.
- Matematikte sonsuz elemanı olan sonsuz diziler tanımlanabilir. Ama programlamada hep sonlu elemanlı diziler yani uzunluğu (size) belirli diziler kullanılır.

Tek boyutlu Diziler

C Dilinde sonlu elemanlı diziler aşağıdaki gibi tanımlanır;



Şekil 20. Tek Boyutlu Dizi Tanımlama

1. Dizi içinde, benzer veri tipindeki (data type) veri öğelerini bulundurur ve bu öğeler bitişik bellek bölgesini paylaşırlar.
2. Dizi elemanları, ilkel veri tipleri (int, float, char) veya daha sonra göreceğimiz kullanıcı tanımlı tip olan yapı (struct) veya gösterici (pointer) olabilir.
3. Aynı değer birkaç farklı yerde dizi içinde bulunabilir.
4. Dizinin veri tipi, dizideki elemanlarının veri tipini belirler. Bir başka deyişle elemanların veri tipi dizinin veri tipiyle aynı olmalıdır.
5. Dizinin uzunluğu tamsayı değişmez (integer literal) olup dizi kimliklendirmesi sırasında belirtilmelidir. Derleyici buna göre bellekte yer ayırır. Dolayısıyla dizi, bir kez kimliklendirildiğinde dizinin uzunluğu değiştirilemez.
6. Dizinin indisi (index) her zaman sıfırdan başlar ve sıra belirttiğinden her zaman tamsayıdır (integer).

C dilinde beş tamsayı elemanı olan bir dizi aşağıdaki şekilde tanımlanır.

```
int tamsayiDizisi[5];
```

Beş gerçek sayı elemanı olan bir dizi ise aşağıdaki şekilde tanımlanır.

```
float reelsayiDizisi[5];
```

Dizinin elemanlarına tanımlama sırasında başlangıç değeri (initial value) değer verilebilir;

```
int tamsayiDizisi1[5]= {1, 2, 3, 4, 5}; //Elemanlar sırasıyla 1,2,3,4,5
int tamsayiDizisi2[5]= {0}; //Elemanların Hepsi 0
```

Dizinin elemanlarına hepsine başlangıç değeri verilmeyebilir;

```
float a[5] = {1.0, 2.0, [4] = 12.0}; // 1.Eleman 1.0, 2.Elaman 2.0 ve 5.eleman 12.0
```

Dizinin elemanlarına ayrı ayrı erişilip değerleri değiştirilebilir ya da yeni değer ataması yapılabilir;

```
int a[5]={0};
a[0]=1; //Birinci elemana 1 değeri atandı
a[1]++; //İkinci elemanın değeri artırıldı
a[2]--; //Üçüncü elemanın değeri azaltıldı
a[4]=12; //Beşinci elemana 12 değeri atandı.
```

Diziler belleği verimli kullanan ve tek bir değişken ile elemanlara erişim sağlayan bir çözüm sunar. Bir dizideki elemanlar, bellekte bitişik konumda yer aldığından herhangi bir öğeye kolaylıkla erişebiliriz.

Bir dizinin başlıca üstünlükleri şunlardır:

- İndisleri kullanarak dizi öğelere rastgele ve hızlı erişim. Her öğenin bir indisi (index) olduğundan doğrudan erişilebilir ve değiştirilebilir.
- Birden fazla öğeden oluşan tek bir dizi oluşturduğundan daha az kod satırı yazılır.
- Daha az kod satırı yazılarak sıralama yapılabilir.

Dizi kullanmanın zayıf yönleri ise;

- Kimliklendirme sırasında karar verilen değişmez (literal) sayıda elemanın üzerinde işlem yapılır.
- Dizi dinamik değildir. Araya elaman ekleme veya çıkarma yapılamaz.

Örnek olarak klavyeden girilen 5 adet tamsayıyı, giriş sırasının tersinden ekrana yazan C programını verebiliriz;

```
#include<stdio.h>
int main() {
  int dizi[5]; /* 5 elemanı tamsayı olan bir dizi tanımlandı*/
  int indis; /* dizi elemanlarını gezecek ve indis olarak kullanılacak
    tamsayı bir değişken tanımlandı */

  for (indis =0; indis <5; indis ++){
    printf("%d. sayıyı girin:", indis);
    scanf("%d",&dizi[ indis ]); //indis ile belirtilen elemana klavyeden olunan değer atanıyor
  }

  printf("Tersten Sayılar:\n");
  for (indis=4; indis >=0; indis --)
    printf("%d\n", dizi[ indis ]);
  return 0;
}
/*Program Çıktısı:
0. sayıyı girin:12
1. sayıyı girin:11
2. sayıyı girin:4
3. sayıyı girin:6
4. sayıyı girin:0
Tersten Sayılar:
0
6
4
11
12

...Program finished with exit code 0
*/
```

Bir başka örnek; Klavyeden girilen 10 adet sınav notuna göre, ortalamanın üstünde olan notları ekrana yazan C programı;

```
#include<stdio.h>
#define BOYUT 10
int main() {
    unsigned notlar[BOYUT]; /* Her terimi pozitif olan 10 elemanlı dizi */
```

```
    float ortalama, toplam=0;
    int indis; //indis için tamsayı değişken
    for (indis=0; indis<BOYUT; indis++){
        printf("%d. notu girin:",indis);
        scanf("%d",&notlar[indis]);
        toplam+=notlar[indis]; // Her eleman girildiğinde toplanıyor
    }
    ortalama=toplam/BOYUT;
    printf("Ortalamanın (%.2f) Üzerindeki Notlar:\n",ortalama);
    for (indis=0; indis<BOYUT; indis++)
        if (notlar[indis]>ortalama)
            printf("%d,",notlar[indis]);
    return 0;
}
/*Program Çıktısı:
0. notu girin:45
1. notu girin:65
2. notu girin:55
3. notu girin:50
4. notu girin:30
5. notu girin:75
6. notu girin:85
7. notu girin:90
8. notu girin:70
9. notu girin:100
Ortalamanın (66.50) Üzerindeki Notlar:
75,85,90,70,100,

...Program finished with exit code 0
*/
```

Bir başka örnek olarak 10 adet satış miktarlarının, ortalamaya olan uzaklıkları yani sapma (deviation) ve karesi alınmış sapmalar yani varyans (variance) ile standart sapmayı yazdıran C program verilebilir.

```
#include<stdio.h>
#include<math.h>
#define BOYUT 10
int main() {
    float satislar[BOYUT]={100.0,850.0,500.0,600.0,750.0,650.0,450.0,800.0,900.0,110.0};
    float ortalama, toplam=0, varyansToplam=0, standartSapma;
    int indis;

    for (indis=0; indis<BOYUT; indis++)
        toplam+= satislar[indis];
    ortalama=toplam/BOYUT;
    printf("Ortalama:%.2f\n",ortalama);

    printf("Sapma\tVaryans:\n");
    for (indis=0; indis<BOYUT; indis++) {
        float sapma= satislar[indis]-ortalama;
        varyansToplam+=sapma*sapma;
        printf("Sapma: %.2f\tVaryans: %.2f\n", sapma,sapma*sapma);
    }
    standartSapma=sqrt(varyansToplam/BOYUT);
    printf("Standart Sapma:%.2f\n",standartSapma);
    return 0;
}
/*Program Çıktısı:
Ortalama:571.00
Sapma   Varyans:
Sapma: -471.00  Varyans: 221841.00
```

```
Sapma: 279.00   Varyans: 77841.00
Sapma: -71.00   Varyans: 5041.00
Sapma: 29.00    Varyans: 841.00
Sapma: 179.00   Varyans: 32041.00
Sapma: 79.00    Varyans: 6241.00
Sapma: -121.00  Varyans: 14641.00
Sapma: 229.00   Varyans: 52441.00
Sapma: 329.00   Varyans: 108241.00
Sapma: -461.00  Varyans: 212521.00
Standart Sapma:270.50

...Program finished with exit code 0
*/
```

Ortalamaya olan uzaklıklar yani sapma verilerin ortalamaya ne kadar yakın olduğunu gösteren dağılımdır. Sapmaların toplamı sıfır olabileceğinden karelerinin toplamı yani varyans hesaplanır. Varyansın eleman sayısına bağlı karekökü de standart sapmayı verir. Standart sapmanın büyük olması verilerin ortalamadan daha uzak yayıldıklarını; küçük bir standart sapma ise verilerin ortalama etrafında daha çok yakın gruplaştıklarını gösterir.

Bir başka örnek olarak 5 elemanlı diziye girilen elemanlardan tekil (unique) olanlarını bulan program verilebilir.

```
#include <stdio.h>
#define BOYUT 5
int main()
{
  int dizi[BOYUT];
  int i;
  for (i=0; i<BOYUT; i++){
    printf("%d. sayıyı girin:",i);
    scanf("%d",&dizi[i]);
  }
  printf("\nDizi içinde tekil (unique) olan rakamlar: \n");
  for (i=0; i<BOYUT; i++)
  {
    int j, counter = 0;  // Her elemanda sayacı sıfırla
    for (j = 0; j < BOYUT; j++)
      if (i != j) //elemanın kendisini kontrol etmiyoruz
        if (dizi[i] == dizi[j])
          counter++;
    if (counter == 0)
      printf("%d tekil olarak dizide bulundu.\n", dizi[i]);
  }
  return 0;
}
/*Program Çıktısı:
0. sayıyı girin:12
1. sayıyı girin:15
2. sayıyı girin:12
3. sayıyı girin:3
4. sayıyı girin:15

Dizi içinde tekil (unique) olan rakamlar:
3 tekil olarak dizide bulundu.

...Program finished with exit code 0
*/
```

Bir başka örnek olarak 5 elemanlı diziye girilen en büyük elemanına en küçük elemanını ekleyen program verilebilir;

```
#include <stdio.h>
#define BOYUT 5
int main(){
    int dizi[BOYUT]={10,12,3,4,6};
    int enBuyuk, enKucuk, enKucukIndex, enBuyukIndex, i ;
    printf("ÖNCE:"); //Dizinin İlk hali konsola yazdırılıyor.
    for (i = 0; i < BOYUT; i++)
        printf("%02d ",dizi[i]);
    printf("\n");
    enKucuk=dizi[0]; enBuyuk=dizi[0]; /* İlk elemanlar hem en küçük
                                         hem de en büyük olsun. */
    enKucukIndex=0,enBuyukIndex=0; /* En küçük ve en büyük elemanların
                                      indisi 0 olsun. */
    for (i=0; i<BOYUT; i++){ /* En küçük ve en büyük eleman ile
                                indislerini bulan kısım: */
        if (dizi[i]<enKucuk) {
            enKucuk=dizi[i];
            enKucukIndex=i;
        }
        if (dizi[i]>enBuyuk) {
            enBuyuk=dizi[i];
            enBuyukIndex=i;
       }
    }
    printf("En Büyük: dizi[%d]=%02d, En Küçük: dizi[%d]=%02d\n",
        enBuyukIndex,dizi[enBuyukIndex],
        enKucukIndex,dizi[enKucukIndex]);
    dizi[enBuyukIndex]+=dizi[enKucukIndex]; /* En büyük elemana
                                               en küçüğünü ekleme: */
    printf("SONRA:"); //Dizinin son hali konsola yazdırılıyor.
    for (i = 0; i < BOYUT; i++)
        printf("%02d ",dizi[i]);
    return 0;
}
/*Program Çıktısı:
ÖNCE:10 12 03 04 06
En Büyük: dizi[1]=12, En Küçük: dizi[2]=03
SONRA:10 15 03 04 06

...Program finished with exit code 0
*/
```

İki Boyutlu Diziler

Şu ana kadar gördüğümüz tek boyutlu dizilerdi. İki boyutlu diziler matematikten de bildiğimiz üzere matris (matrix) olarak adlandırılır. İki boyutlu dediğimizde en-boy ve satır-sütun gibi kavramlar akla gelmelidir. Matris işlemleri gibi bazı problemlerde; bir dizinin her bir elemanının da dizi olması istenir. Bu tür iki boyutlu dizilerde en içteki dizinin boyutu Kimliklendirmede sağda yer alır.

Aşağıda iki boyutlu matris tanımlamalarına örnek verilmiştir;

```
float matris[2][3];  //Her bir elemanı 3 elemanlı bir dizi olan 2 satır üç sütunlu bir matris
//Aşağıda 2x3 matrise ilk değer verme:
float matris2[2][3]= {
                       {1.0,2.0,3.0}, //Birinci Satır: 3 Elemanlı bir dizi
                       {2.0,4.0,6.0}  //İkinci Satır: 3 Elemanlı bir dizi
                     };
int karematris[2][2];
//2x2 matrise ilk değer verme:
int karematris2[2][2]= {
                         {1,2}, //Birinci Satır: 2 elemanlı bir dizi
                         {5,6} //Birinci Satır: 2 elemanlı bir dizi
```

```
                        };
int karematris3[3][3];
```

Örnek olarak 3x4'lük matris elemanlarını klavyeden girip, tablo halinde ekrana yazdıran programı verebiliriz;

```
#include<stdio.h>
#define SATIR 3
#define SUTUN 4
int main() {
    int matris[SATIR][SUTUN];
    int i,j;
    /*
     for (j=0; j<SUTUN; j++)  //Birinci Satır Okuyalım
         scanf("%d",&matris[0][j]);
     for (j=0; j<SUTUN; j++)  //İkinci Satır Okuyalım
         scanf("%d",&matris[1][j]);
     for (j=0; j<SUTUN; j++)  //Üçüncü Satır Okuyalım
         scanf("%d",&matris[2][j]);
     //Bunun yerine iç içe iki for tanımlanır;
    */
    for (i=0; i<SATIR; i++) { //İkinci Boyut İçin
        for (j=0; j<SUTUN; j++)  //Birinci Boyut İçin
            scanf("%d",&matris[i][j]);
    }
    printf("\nTABLO\n");
    for (i=0; i<SATIR; i++) {  //İkinci Boyut İçin
        for (j=0; j<SUTUN; j++)  //Birinci Boyut İçin
            printf("%d\t",matris[i][j]);
        printf("\n");
    }
    return 0;
}
/*Program Çıktısı:
3 4 5 6 7 8 4 4 5 8 7 9

TABLO
3       4       5       6
7       8       4       4
5       8       7       9

...Program finished with exit code 0
*/
```

Bir başka örnek olarak elemanları klavyeden girilen 3x2'lık matrisin satır ve sütun toplamlarını ekrana yazan programı verilebilir;

```
#include<stdio.h>
#define SATIR 3
#define SUTUN 2

int main() {
    int matris[SATIR][SUTUN]; //Matris Tanımı
    int i,j;  //i sutun için, j satır için tanımlandı
    for (i=0; i<SATIR; i++) {  //İkinci Boyut (SATIR) İçin
        printf("%d. satir:",i);
        for (j=0; j<SUTUN; j++)  //Birinci Boyut (SUTUN) İçin
            scanf("%d",&matris[i][j]);
    }
    int satirToplamlar[SATIR]={0}; //Bütün elemanları 0 olan dizi
    for (i=0; i<SATIR; i++) {  // İkinci Boyut İçin
        for (j=0; j<SUTUN; j++) // Birinci Boyut İçin
            satirToplamlar[i]+=matris[i][j];
```

```
        /* İçdeki for bitince tüm satırdaki elemanlar toplanmış oldu */
    }/* Dışdaki for ile de her bir satır için toplam tekrarlanıyor */
    printf("\nSatır Toplamları:");
    for (i=0; i<SATIR; i++)
        printf("%d\t",satirToplamlar[i]);
    int sutunToplamlar[SUTUN]={0}; //Bütün elemanları 0 olan dizi
    for (j=0; j<SUTUN; j++) // Birinci Boyut İçin
        for (i=0; i<SATIR; i++) {  // İkinci Boyut İçin
            sutunToplamlar[j]+=matris[i][j];
        /* İçdeki for bitince tüm sutundaki elemanlar toplanmış oldu*/
    }/* Dışdaki for ile de her bir sutun için toplam tekrarlanıyor */
    printf("\nSütun Toplamları:");
    for (j=0; j<SUTUN; j++)
        printf("%d\t",sutunToplamlar[j]);
    return 0;
}
/*Program Çıktısı:
0. satir:12 18
1. satir:22 28
2. satir:2 8

Satır Toplamları:30     50      10
Sütun Toplamları:36     54

...Program finished with exit code 0
*/
```

Çok Boyutlu Diziler

Şu ana kadar gördüğümüz tek boyutlu diziler veya iki boyutlu diziler olan matrislerdir. Çok boyutlu dizilere örnek olarak En-Boy-Yükseklik içeren 3 boyutlu diziler verilebilir. Üç boyutlu dizilerde dizinin her bir elemanı bir matristir.

Çok boyutlu dizilerde en içteki dizinin boyutu kimliklendirmede sağda yer alır. Aşağıdaki örnekte; çeşitli boyutlarda diziler tanımlanmıştır,

```
int ucbuyutludizi1[3][2][2];
// ucbuyutludizi1: elemanları 3 adet 2x2 matris olan üç boyutlu bir dizi
int ucbuyutludizi1[2][3][3];
// ucbuyutludizi2: elemanları 2 adet 3x3 matris olan üç boyutlu bir dizi
int ucbuyutludizi1[4][3][2];
// ucbuyutludizi3: elemanları 4 adet 3x2 matris olan üç boyutlu bir dizi
int dortbuyutludizi1[5][2][3][2];
/* dortbuyutludizi1: elemanları 5 adet olan ve her bir elemanı 2 adet 3x2 matris olan dört
boyutlu bir dizi */
```

Aşağıda üç boyutlu dizilerde bir ilk değer verme kod örneği verilmiştir;

```
#define DERINLIK 3
#define SATIR 3
#define SUTUN 3

int ucBoyutluDizi[DERINLIK][SATIR][SUTUN]={
    { {1,2,3},{4,5,6},{7,8,9} },
    { {11,12,13},{14,15,16},{17,18,19} },
    { {21,22,23},{24,25,26},{27,28,29} }
};
```

Çok boyutlu dizi örneği olarak şöyle bir örnek verilebilir; 3 farklı ders alan 10 öğrenci, her bir dersten 4 değişik not almaktadır. Bu notların ağırlıkları %10, %30, %20 ve %40'tır. Notlar rastgele 0 ile 100 arasında verilecektir. Her bir sınavın ortalaması ile her bir öğrencinin ağırlıklı not ortalamalarını bulan C programı yazınız.

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define KACDERS 3
#define KACOGRENCI 10
#define KACDEGISIKNOT 4
int main() {
    int notlar[KACDERS][KACDEGISIKNOT][KACOGRENCI];
    int agirlik[KACDEGISIKNOT]={10,30,20,40};
        //Ödevler:%10, Vize:%30, Hızlı Sınav:%20, Final:%40
    int i,j,k; //indis değişkenleri
    srand(time(NULL)); // rastgele için
    for (k=0; k< KACDERS; k++)
        for (i=0; i< KACOGRENCI; i++)
            for (j=0; j< KACDEGISIKNOT; j++)
                notlar[k][j][i]=rand()%101;
    //Her bir sınavın Ortalaması hesaplanacak:
    for (k=0; k< KACDERS; k++) {
        printf("%d.Ders İçin:\n",k);
        for (j=0; j<KACDEGISIKNOT; j++)  {
            float sinavToplam=0.0;
            for (i=0; i<KACOGRENCI; i++) {
                sinavToplam+=notlar[k][j][i];
            }
            printf("%d. Sınav ortalaması:%.2f\n",j,sinavToplam/KACOGRENCI);
        }
    }
    //Her bir Öğrencinin Ağırlıklı Notu hesaplanacak:
    for (k=0; k< KACDERS; k++) {
        printf("%d.Ders İçin:\n",k);
        for (i=0; i<KACOGRENCI; i++) {
            float agirlikliNot=0.0;
            for (j=0; j<KACDEGISIKNOT; j++)  {
                agirlikliNot+=notlar[k][j][i]*agirlik[j]/100.0;
            }
            printf("%d. Öğrenci ortalaması:%.2f\n",i,agirlikliNot);
        }
    }
    return 0;
}
/*Program Çıktısı:
0.Ders İçin:
0. Sınav ortalaması:48.40
1. Sınav ortalaması:51.40
2. Sınav ortalaması:57.10
3. Sınav ortalaması:54.60
1.Ders İçin:
0. Sınav ortalaması:58.60
1. Sınav ortalaması:40.30
2. Sınav ortalaması:74.00
3. Sınav ortalaması:46.60
2.Ders İçin:
0. Sınav ortalaması:59.40
1. Sınav ortalaması:54.20
2. Sınav ortalaması:49.20
3. Sınav ortalaması:36.00
0.Ders İçin:
0. Öğrenci ortalaması:52.90
1. Öğrenci ortalaması:30.10
2. Öğrenci ortalaması:66.40
3. Öğrenci ortalaması:79.70
```

```
4. Öğrenci ortalaması:57.80
5. Öğrenci ortalaması:46.10
6. Öğrenci ortalaması:27.70
7. Öğrenci ortalaması:70.20
8. Öğrenci ortalaması:53.10
9. Öğrenci ortalaması:51.20
1.Ders İçin:
0. Öğrenci ortalaması:33.50
1. Öğrenci ortalaması:37.30
2. Öğrenci ortalaması:74.90
3. Öğrenci ortalaması:57.30
4. Öğrenci ortalaması:46.40
5. Öğrenci ortalaması:48.10
6. Öğrenci ortalaması:32.40
7. Öğrenci ortalaması:71.90
8. Öğrenci ortalaması:59.00
9. Öğrenci ortalaması:53.10
2.Ders İçin:
0. Öğrenci ortalaması:47.10
1. Öğrenci ortalaması:39.90
2. Öğrenci ortalaması:50.50
3. Öğrenci ortalaması:40.70
4. Öğrenci ortalaması:67.10
5. Öğrenci ortalaması:14.90
6. Öğrenci ortalaması:36.00
7. Öğrenci ortalaması:36.90
8. Öğrenci ortalaması:45.10
9. Öğrenci ortalaması:86.20

...Program finished with exit code 0
*/
```

Dizi Elemanlarını Sıralama

Dizilerle ilgili bazı durumlarda elemanları sıralamak isteriz. Birden çok sıralama yöntemi vardır; Kabarcık sıralama (bubble sort), Seçme sıralama (selection sort), Kabuk sıralama (shell sort) ve Çabuk sıralama (quick sort) gibi.

Kabarcık Sıralama

Kabarcık sıralama (bubble sort) algoritmasında;

1. Her bir eleman kendisinden sonra gelen eleman ile karşılaştırılır.
2. Büyük ya da küçük sorgulaması yapılır. Büyük sorgulaması yapıldığında dizi küçükten büyüğe, küçük sorgulaması yapıldığında dizi büyükten küçüğe sıralanır.
3. Eğer şarta uyuyor ise elemanların yerleri değiştirilir (swap).
4. Bir sonraki elemana geçilerek birinci adımdan bu adıma kadar olan adımlar son dizi elemanına kadar devam edilir.
5. Yukarıdaki dört adım bitirildiğinde yalnızca birinci eleman doğru yerde olur. Böylece algoritmanın birinci aşaması tamamlanmıştır. Dolayısıyla bir sonraki elemana geçilerek dizinin eleman sayısı kadar bu işlemin tekrar edilmesi gerekir. Son elemana için aşamanın tekrar edilmesine gerek yoktur. Dolayısıyla aşama sayısı dizi uzunluğundan bir eksik yapılır.

Bunu bir örnekle açıklayalım;

```
int dizi[5]={14,11,0,3,7}; //Dizisini Küçükten Büyüğe sıralayalım:

if (a[0]<a[1]) { int gecici=a[0]; a[0]=a[1]; a[1]=gecici; } // 14>11 ise yer değiştir
// Dizinin yeni hali: {11,14,0,3,7}
if (a[1]<a[2]) { int gecici=a[1]; a[1]=a[2]; a[2]=gecici; } // 14>0 ise yer değiştir
// Dizinin yeni hali: {11,0,14,3,7}
if (a[2]<a[3]) { int gecici=a[2]; a[2]=a[3]; a[3]=gecici; } // 14>3 ise yer değiştir
```

```
// Dizinin yeni hali: {11,0,3,14,7}
if (a[3]<a[4]) { int gecici=a[3]; a[3]=a[4]; a[4]=gecici; } // 14>7 ise yer değiştir
// Dizinin yeni hali: {11,0,3,7,14}
//Buraya kadar birinci aşama tamamlanmıştır. En son eleman belirlenmiştir.

if (a[0]<a[1]) { int gecici=a[0]; a[0]=a[1]; a[1]=gecici; } // 11>0 ise yer değiştir
// Dizinin yeni hali: {0,11,3,7,14}
if (a[1]<a[2]) { int gecici=a[1]; a[1]=a[2]; a[2]=gecici; } // 11>3 ise yer değiştir
// Dizinin yeni hali: {0,3,11,7,14}
if (a[2]<a[3]) { int gecici=a[2]; a[2]=a[3]; a[3]=gecici; } // 11>7 ise yer değiştir
// Dizinin yeni hali: {0,3,7,11,14}
// Son eleman bir önceki aşamada belirlendiğinden karşılaştırma sayımız eksildi.
//Buraya kadar ikinci aşama tamamlanmıştır. Sondan ikinci eleman belirlenmiştir.

if (a[0]<a[1]) { int gecici=a[0]; a[0]=a[1]; a[1]=gecici; } // 0>3 ise yer değiştir
// Dizinin yeni hali: {0,3,7,11,14}
if (a[1]<a[2]) { int gecici=a[1]; a[1]=a[2]; a[2]=gecici; } // 3>7 ise yer değiştir
// Dizinin yeni hali: {0,3,7,11,14}
// Son eleman bir önceki aşamada belirlendiğinden karşılaştırma sayımız eksildi.
//Buraya kadar üçüncü aşama tamamlanmıştır. Sondan üçüncü eleman belirlenmiştir.

if (a[0]<a[1]) { int gecici=a[0]; a[0]=a[1]; a[1]=gecici; } // 0>3 ise yer değiştir
// Dizinin yeni hali: {0,3,7,11,14}
// Son eleman bir önceki aşamada belirlendiğinden karşılaştırma sayımız eksildi.
//Buraya kadar dördüncü aşama tamamlanmıştır. Sondan dördüncü eleman belirlenmiştir.

// Bir aşamaya daha gerek yoktur. Çünkü kalan 1 elemanın yeri değişmeyecektir.
```

Şimdi de sıralamayı döngü ile yapalım;

```
#include <stdio.h>
#define UZUNLUK 5
int main() {
    int dizi[UZUNLUK]={14,11,0,3,7};
    int indis;
    printf("Dizinin SıralanMAmış hali:\n");
    for (indis=0; indis<UZUNLUK; indis++)
        printf("%d, ",dizi[indis]);
    printf("\n");
    int asama;
    for (asama=0; asama<UZUNLUK-1; asama++) { //asama için
        for (indis=0; indis<UZUNLUK-asama-1;indis++) { /* karşılaştırma
                                                          asama kadar eksik
                                                          yapılır */
            if (dizi[indis] > dizi[indis+1]) { /* birinci eleman ile
                                                  ikincisini karşılaştır */
                int yedek=dizi[indis]; // birinci elemanı yedekle
                dizi[indis]=dizi[indis+1]; /* birinci elemana
                                              ikicici elemanı ata */
                dizi[indis+1]=yedek;  // yedeği ikinci elemana ata
            }
        }
    }
    printf("Dizinin SıralanMIŞ hali:\n");
    for (indis=0; indis<UZUNLUK; indis++)
        printf("%d, ",dizi[indis]);
    return 0;
}
/*Programın Çıktısı:
Dizinin SıralanMAmış hali:
14, 11, 0, 3, 7,
Dizinin SıralanMIŞ hali:
```

```
0, 3, 7, 11, 14,

...Program finished with exit code 0
*/
```

Seçme Sıralama

Seçme sıralama (selection sort) algoritmasında;

Küçükten büyüğe seçme sıralama aşağıdaki adımlar izlenerek yapılır;

- Dizi üzerinde ilk elemandan başlanarak dizi sonuna kadar ilerlenir.
- İlerleme sırasındaki elemanın sonrasındaki elemanlar arasında en küçük eleman bulunur.
- Bulunan en küçük eleman ile ilerleme sırasındaki eleman yer değiştirilir.

Büyükten küçüğe seçme sıralama aşağıdaki adımlar izlenerek yapılır;

- Dizi üzerinde ilk elemandan başlanarak dizi sonuna kadar ilerlenir.
- İlerlemeye sırasındaki elemanın sonrasındaki elemanlar arasında en büyük eleman bulunur.
- Bulunan en küçük eleman ile ilerleme sırasındaki eleman yer değiştirilir.

Aşağıda buna ilişkin örnek program verilmiştir;

```
#include <stdio.h>
#define UZUNLUK 5
void diziYaz(int pDizi[], int pBoyut);
void selectionSortKucuktenBuyuge(int pDizi[], int pBoyut);
void selectionSortBuyuktenKucuge(int pDizi[], int pBoyut);
int main() {
    int dizi[UZUNLUK]={14,11,0,3,7};
    printf("Dizinin SıralanMAMIŞ hali:\n");
    diziYaz(dizi, UZUNLUK);
    selectionSortKucuktenBuyuge(dizi,UZUNLUK);
    printf("Dizinin Küçükten Büyüğe SıralanMIŞ hali:\n");
    diziYaz(dizi, UZUNLUK);
    selectionSortBuyuktenKucuge(dizi,UZUNLUK);
    printf("Dizinin Büyüktem Küçüğe SıralanMIŞ hali:\n");
    diziYaz(dizi, UZUNLUK);
    return 0;
}
void diziYaz(int pDizi[], int pBoyut){
    int sayac;
    for (sayac=0; sayac<pBoyut; sayac++)
        printf("%d,",pDizi[sayac]);
    printf("\n");
}
void selectionSortKucuktenBuyuge(int pDizi[], int pBoyut){
    int ilerleme;
    for (ilerleme=0; ilerleme<pBoyut-1; ilerleme++) {
        int sayac;
        int enKucukIndis=ilerleme;
        for (sayac=ilerleme+1; sayac<pBoyut; sayac++)/* Geri kalanındaki
                                                        en Küçüğün İndisi */
            if (pDizi[sayac] < pDizi[enKucukIndis])
                enKucukIndis =sayac;
        /* Bulunan en küçükindisli eleman ile
           ilerlemedeki eleman yer değiştirilir */
        int yedek=pDizi[ilerleme]; /* yer değiştirme:
                                      yedek, dizi veri tipinde olmalı */
        pDizi[ilerleme]=pDizi[enKucukIndis];
        pDizi[enKucukIndis]=yedek;
   }
}
void selectionSortBuyuktenKucuge(int pDizi[], int pBoyut){
```

```
    int ilerleme;
    for (ilerleme=0; ilerleme<pBoyut-1; ilerleme++) {
        int sayac;
        int enBuyukIndis=ilerleme;
        for (sayac=ilerleme+1; sayac<pBoyut; sayac++)  /*Geri kalanındaki
                                                         en Küçüğün İndisi */
            if (pDizi[sayac] > pDizi[enBuyukIndis])
                enBuyukIndis =sayac;
        /* Bulunan en küçükindisli eleman ile
           ilerlemedeki eleman yer değiştirilir */
        int yedek=pDizi[ilerleme]; //yer değiştirme:
        pDizi[ilerleme]=pDizi[enBuyukIndis];
        pDizi[enBuyukIndis]=yedek;
   }
}
/*Program Çıktısı:
Dizinin SıralanMAMIŞ hali:
14,11,0,3,7,
Dizinin Küçükten Büyüğe SıralanMIŞ hali:
0,3,7,11,14,
Dizinin Büyüktem Küçüğe SıralanMIŞ hali:
14,11,7,3,0,

...Program finished with exit code 0
*/
```

Parametre Olarak Diziler

Parametre olarak gönderilen dizinin boyutu kadar köşeli parantez açılır ve kapatılır, ilk köşeli parantez içerisine eleman sayısını ifade eden değer yazılmaz, fakat diğerlerine eleman sayıları verilmek zorundadır. Bu durumda tek boyutlu diziler parametre olacak ise dizinin boyutu yazılmayabilir.

Aşağıda dizileri parametre olarak alan fonksiyon bildirim örnekleri bulunulmaktadır;

```
void diziIsleyenFonksiyon1(int tekBoyutluDizi[],int boyut);
void diziIsleyenFonksiyon2(int pDizi[][2],int ikinciBoyut,int birinciBoyut);
void diziIsleyenFonksiyon3(int pDizi[][3][2],int ucuncuBoyut,int ikinciBoyut,int
birinciBoyut);
```

Aynı fonksiyonları aşağıdaki şekilde çağırabiliriz;

```
unsigned notlar[10]; //10 öğrenci notu barındırır.
unsigned dersNotlari[2][10]; //2 dersi alan 10 öğrenci için notlar;
unsigned bolumDersNotlari[4][2][10]; /* 4 bölümde 2 dersi alan 10 öğrenci
                                        için notlar;*/
//...
diziIsleyenFonksiyon1(notlar,10);
diziIsleyenFonksiyon2(dersNotlari,2,10);
diziIsleyenFonksiyon3(bolumDersNotlari,4,2,10);
//...
```

Aşağıda bir diziyi okuyan, ekrana yazan ve eleman ortalamalarını hesaplayan fonksiyonlara sahip bir program verilmiştir;

```
#include<stdio.h>
void diziOku(int pDizi[],int pUzunluk);
void diziYaz(int pDizi[],int pUzunluk);
float diziOrtalama(int pDizi[],int pUzunluk);
int main() {
  int dizi[5];
  float ortalama;
  diziOku(dizi,5); /* fonsiyon geri dönüş değeri olmadığından
                      bir değişkene atanmıyor! */
  diziYaz(dizi,5); /* fonsiyon geri dönüş değeri olmadığından
```

```
                        bir değişkene atanmıyor! */
  ortalama=diziOrtalama(dizi,5); /* fonksiyon geri dönüş değeri
                                    bir değişkene atanıyor.
                                    atandığı değişkenin tipi ile
                                    fonksiyonun geri dönüş tipi
                                    aynı olmalı! */
  printf("Dizi Ortalaması: %.2f",ortalama);
  return 0;
}
void diziOku(int pDizi[],int pUzunluk){
  int sayac;
  printf("%d Elemanlı Dizi Okunacaktır:",pUzunluk);
  for (sayac=0; sayac < pUzunluk; sayac++)
     scanf("%d",&pDizi[sayac]);
}
void diziYaz(int pDizi[],int pUzunluk){
  int sayac;
  printf("%d Elemanlı Dizi:\n",pUzunluk);
  for (sayac=0; sayac < pUzunluk; sayac++)
     printf("%4d",pDizi[sayac]);
  printf("\n");
}
float diziOrtalama(int pDizi[],int pUzunluk){
  int sayac;
  float toplam=0;
  for (sayac=0; sayac < pUzunluk; sayac++)
     toplam+=pDizi[sayac];
  return toplam/pUzunluk;
}
/*Program Çıktısı:
5 Elemanlı Dizi Okunacaktır:10 20 30 40 50
5 Elemanlı Dizi:
  10  20  30  40  50
Dizi Ortalaması: 30.00

...Program finished with exit code 0
*/
```

GÖSTERİCİLER

Gösterici nedir? Niçin İhtiyaç Duyarız?

Şu ana kadar gördüğümüz `char`, `int`, `long`, `float`, `double` tipli değişkenler ve bunlara ait diziler, değer tipler (value type) olarak adlandırılır. Çünkü kimliklendirilen değişken, tutulacak değeri içerir. Değer tiplerin yanı sıra değişkenlerin adreslerini tutan değişkenlere de ihtiyaç duyarız. İşte adres tutan bu değişkenlere gösterici tipler (pointer type) adını veririz. Verilerin tipine göre gösterici tanımlandığından göstericiler, türetilmiş tiplerdir (derived type).

Gösterici tipler, gösterdiği yerdeki verileri değiştirmek için de kullanılır. Bu nedenle gösterici tipler tanımlanırken gösterdiği yerdeki verinin tipine göre tanımlanırlar. Bütün gösterici tipler bellekte aynı miktarda yer kaplar. Çünkü hepsi adres tutar. Ancak işaret ettikleri yerdeki veriler, verinin tipine bağlı olarak bellekte farklı miktarda yer kaplar. Bütün bu tanımlamaların ışığında göstericileri, vatandaşların ikametlerini gösteren adres numaraları olarak düşünebiliriz.

Gösterici tiplere ihtiyaç duyulmasının sebebi hız ve esnekliktir. Genel olarak birkaç madde ile sıralayacak olursak;

- Belleğin istenilen bölgesine erişim sağlamak için göstericiler kullanılır. Günümüz modern dillerinde referans tipler kullanılır, ancak referans tiplerde gösterilen adreste bir nesnenin bulunduğu garanti edilir göstericiler gibi her bellek bölgesine erişilemez.
- Bazen çalıştırma anında yeni bellek gölgelerine ihtiyaç duyarız bu durumda göstericiler gereklidir. Dinamik veri yapıları (dynamic data structure) göstericiler yardımıyla oluşturulup yönetilir.
- Fonksiyonlarda yerel değişkenler ve argümanlar yığın belleğe itilir ve fonksiyondan geri dönüldüğünde bu değişkenler eski haline yığın bellekten geri çekilerek çevrilir. Fonksiyonun yerel değişken ya da argümanlardan birini değiştirmesi istendiğinde fonksiyona argüman olarak değişkenin adresi verilir. Böylece fonksiyon içinde gösterici olarak işlem gören argümanlar fonksiyondan geri döndüğünde değerleri değişmiş olur.

Göstericilere olan ihtiyacı bir başka örnekle de açıklayabiliriz; Bir ormandaki ağaçları boylarına göre sıralamaya kalkarsak her birini yer değiştirme yöntemiyle sıralamak oldukça külfetli ve zaman alana bir işlemdir. Halbuki ağaçlara numara vererek ve bu numaraların karşısına uzunluklarını yazacağımız bir liste oluşturulduğunda sadece numaraları sıralamak oldukça kolay ve zahmetsiz bir işlemdir. İşte buradaki numaraları tutan değişkenler göstericilerdir.

Gösterici Tanımlama

Göstericiler, gösterdiği yerdeki verinin tipine göre tanımlanırlar. Gösterici tanımlanırken veri tipi izleyen başvuru kaldırma işleci (dereference operator) olan yıldız (`*`) kullanılır. Başvuru kaldırma terimi, gösterici tarafından tutulan bellek adresinde saklanan değere erişmeyi de ifade eder.

```
veritipi* gostericikimligi;
```

Göstericilere değer atatama aşağıdaki şekilde yapılır;

```
gostericikimligi = &degiskenkimliği;
```

Bir değişkenin adresinin adres işleci (`&`) ile elde edilebileceği *Tekli İşleçler* başlığında anlatılmıştı. Aşağıda göstericilere ilişkin kod örnekleri verilmiştir;

```
char* ptrToChar;  /*tuttuğu adreste char tipinde veri olan ptrToChar kimlikli göstericisi
tanımlandı. */

int i=10;
int* ptrToInt=&i; /* tuttuğu adreste int tipinde veri olan ptrToInt kimlikli göstericisi
tanımlandı ve ilk değer olarak i değişkeninin adresi atanıyor */
*ptrToInt =20; /* ptrToInt göstericisinin gösterdiği adresteki tamsayı değeri 20 yaptık.
ptrToInt göstericisi, i değişkeninin adresini tuttuğundan i değişkeninin değeri de 20 olmuştur
```

```
*/
```

Bazı durumlarda göstericilerin veya gösterdiği değerlerin sabit olması gerekebilir. Üç durum ortaya çıkabilir;

– Gösterilen değerin sabit olması: Bu durumda gösterici tanımlaması aşağıdaki gibi yapılır;

```
int main() {
   int i=10;
   const int* ptrToi=&i; // değerin sabit olması durumunda gösterici tanımı

   *ptrToi=20; /* HATA! pi göstericisinin tuttuğu adresteki tamsayı
    değeri 20 yapılamaz. */

   return 0;
}
```

– Göstericinin işaret ettiği adresin sabit olması: Kısaca göstericinin sabit olması durumunda tanımlama aşağıdaki gibi yapılır;

```
int main() {
   int i=10;
   int* const ptrToi=&i; // göstericinin sabit olması durumunda gösterici tanımı
   /* bu tür tanımlamada adres ilk değer (initialize) olarak mutlaka verilmelidir. */

   int j=30;
   *ptrToi=20; // pi göstericisinin tuttuğu adresteki tamsayı değeri 20 olur.
   ptrToi =&j; //HATA! : pi göstericisinin tuttuğu adres değiştirilemez!

   return 0;
}
```

– Hem göstericinin hem de değerin sabit olması durumu:

```
int main() {
   int j=30;
   const int* const ptr=&j; /* Hem gösterici, hem de gösterilen veri sabit */
   /* bu tür tanımlamada da adres ilk değer (initialize) olarak mutlaka verilmelidir. */
   return 0;
}
```

Atanmış kesin bir adresiniz yoksa, bir gösterici değişkenine `NULL` değeri atamak her zaman iyi bir uygulamadır. İlk değeri olmayan gösterici tanımlamak yerine, ilk değeri sıfır olan göstericiler tanımlanmak doru bir yöntemdir. Bu durumda ilk değer `NULL` olarak atanır. Bu durumda gösterici, NULL gösterici (NULL pointer) adı verilir. Aşağıdaki gibi kullanılır;

```
veritipi* gostericikimligi = NULL;
gostericikimligi = NULL;
```

Örnek kod aşağıda verilebilir;

```
#include <stdio.h>
int main(){

 int* ptr = NULL;
 printf("ptr göstericisinin tuttuğu adres: %p\n", ptr);

 int agirlik=80;
 ptr=&agirlik;
 printf("ptr göstericisinin tuttuğu adres: %p ve değeri: %d\n", ptr, *ptr);

 return 0;
}
/* Program Çıktısı:
ptr göstericisinin tuttuğu adres: (nil)
ptr göstericisinin tuttuğu adres: 0x7ffdedc33dec ve değeri: 80
```

```
...Program finished with exit code 0
*/
```

Bir gösterici bir başka göstericinin adresini tutabilir. Buna çifte gösterici (double pointer) adı verilir. Aşağıda buna ilişkin bir örnek bulunmaktadır;

```
#include <stdio.h>
int main(){
   int var = 10;
   int* intptr = &var;
   int** ptrptr = &intptr;  //double pointer

   printf("var:%d &var:%p \n",var, &var);
   printf("inttptr: %p *inttptr: %p \n\n", intptr, &intptr);

   printf("var: %d *intptr: %d \n", var, *intptr);
   printf("ptrptr: %p &ptrtptr: %p \n", ptrptr, &ptrptr);
   printf("&intptr: %p *ptrptr : %p \n\n", &intptr, *ptrptr);
   printf("var: %d *intptr: %d **ptrptr: %d", var, *intptr, **ptrptr);

   return 0;
}
/*Program Çıktısı:
var:10 &var:0x7ffd52c80524
inttptr: 0x7ffd52c80524 *inttptr: 0x7ffd52c80528

var: 10 *intptr: 10
ptrptr: 0x7ffd52c80528 &ptrtptr: 0x7ffd52c80530
&intptr: 0x7ffd52c80528 *ptrptr : 0x7ffd52c80524

var: 10 *intptr: 10 **ptrptr: 10

...Program finished with exit code 0
*/
```

Bir tamsayının hangi 4 bayt bellek alanından oluştuğu *Değişken Tanımlama* başlığında anlatılmıştı. Aşağıda tanımlanan bir tamsayının hangi baytlardan oluştuğunu gösteren program verilmiştir.

```
#include <stdio.h>
int main() {
    int i=0x10203040;
    char* p= (char*) &i;
    /* int gösteren adres,
       bayt/char içeriyor diye
       (char*) ifadesiyle tip dönüşümü (cast) yapılıyor.
    */
    printf("Sayı  :%10d-%x\n",i,i);
    printf("1.bayt:%10d-%x\n",*p,*p);
    printf("2.bayt:%10d-%x\n",*(p+1),*(p+1));
    printf("3.bayt:%10d-%x\n",*(p+2),*(p+2));
    printf("4.bayt:%10d-%x\n",*(p+3),*(p+3));
    return 0;
}
/*Program Çıktısı:
Sayı  : 270544960-10203040
1.bayt:        64-40
2.bayt:        48-30
3.bayt:        32-20
4.bayt:        16-10

...Program finished with exit code 0
*/
```

Gösterici Aritmetiği

Bilgisayarların tasarımından ötürü işlemciye bağlı belleklerde her bir bayt ayrı adreslenir. Birkaç bayt (`char`) veri içeren bir bellek bölgesini işaret eden bir gösterici tanımlandığında göstericinin işaret ettiği adres, bir sonraki baytı göstermesi istendiğinde gösterici adresi 1 artar.

```
char dizi1[5]={'I','L','H','A','N'}; /* Bellekte Art arda 5 bayt
                                        yer ayrılmış dizi. */
char* pc=&dizi1[0]; /* pc göstericisi dizinin ilk elemanını
                       gösterecek adres (65FDE0) varsayıldı. */
*pc='X';         //dizi {'X','L','H','A','N'} oldu.
pc++;            //pc göstericisi bir sonraki baytı gösterecek (65FDE1) şekilde artırıldı.
*pc='Y';         // dizi {'X','Y','H','A','N'} oldu.
pc+=2;           //pc göstericisi 2 sonraki baytı gösterecek (65FDE2) şekilde artırıldı.

*pc='Z';         // dizi {'X','Y','H','A','Z'} oldu.
```

Benzer şekilde birkaç tamsayı (`int`) veri içeren bir bellek bölgesini işaret eden bir gösterici tanımlandığında göstericinin işaret ettiği adres, bir sonraki tamsayıyı göstermesi istendiğinde gösterici adresi 4 artar. Çünkü tamsayılar bellekte 4 bayt yer kaplar.

```
int dizi2[5]={2,0,3,10,2}; // Bellekte art arda 5 tamsayı (5x4bayt) yer ayrılmış dizi.
int* pi=&dizi2[0];  /*pi göstericisi dizinin ilk elemanını
                      gösterecek adres (65FDE0) varsayıldı. */
*pi=-1;          //dizi {-1,0,3,10,2} şekline döndü.
pi++;            //pi göstericisi ikinci elemanını gösterecek (65FDE4) şekilde artırıldı.
*pi=-2;          //dizi {-1,-2,3,10,2} şekline döndü.
pi+=2;           //pi göstericisi 2 sonraki elemanı gösterecek (65FDEC) şekilde artırıldı.
*pi=-3;          //dizi {-1,-2,3,10,-3} şekline döndü.
```

Görüldüğü üzere göstericilerin üzerinde işleçler (operator) işlem yaparken göstericinin işaret ettiği verinin tipine göre farklı işlem yaparlar. Göstericiler ile aritmetik ve atama işleçleri (`++`, `--`, `+`, `-`, `+=` ve `-=`) ile karşılaştırma işleçleri (`<`, `>` ve `==`) çalışır, diğer işleçler çalışmaz.

Gösterici Dizileri

Göstericiler de dizi olarak tanımlanabilir. Aşağıda buna ilişkin bir örnek verilmiştir;

```
#include <stdio.h>
int main() {
  int i = 10, j=20;
  float f=1.0;
  int k=30, l=40;

  int* ptr[4]; /* int gösteren 4 gösteri içeren bir dizi tanımlandı */

  /* Göstericilere ilk değer veriliyor*/
  ptr[0] = &i; // Birinci eleman i değişkeninin adresini
  ptr[1] = &j; // İkini eleman j değişkeninin adresini
  ptr[2] = &l; // Üçüncü eleman l değişkeninin adresini
  ptr[3] = &k; // Dördüncü eleman k değişkeninin adresini

  // Değerlere Ulaşma
  int indis;
  for (indis = 0; indis < 4; indis++)
    printf("ptr[%d] ile gösterilen değer: %d\n", indis, *ptr[indis]);

  return 0;
}
/*Program Çıktısı:
ptr[0] ile gösterilen değer: 10
ptr[1] ile gösterilen değer: 20
```

```
ptr[2] ile gösterilen değer: 40
ptr[3] ile gösterilen değer: 30

...Program finished with exit code 0
*/
```

Parametre Olarak Göstericiler

Fonksiyonlar çağrılırken argüman olarak giren değerlerin değişmeyeceği *Fonksiyon Çağırma Süreci* başlığında anlatılmıştı. Fonksiyona parametre olarak giren değerler, fonksiyon bloğu içinde değişse bile fonksiyondan geri dönülürken aynı çağrı ortamını sağlamak için kaybolurlar.

```
#include <stdio.h>
void degistir(int,int);
int main(){
   int a = 10, b = 20;
   degistir(a, b);
   printf("1- a:%d, b:%d\n", a, b); //Çıktı: a:10, b:20
}
void degistir(int pX, int pY) {
   int z = pX;
   pX=pY;
   pY=z;
}
```

Fonksiyona giren argümanları gösterici olarak tanımlarsak çağrı ortamına geri dönüldüğünde yerel değişkenler de değişmiş olur. Çünkü fonksiyona değişken adresleri değer olarak girmiştir. Buna ilişkin örnek aşağıda verilmiştir;

```
#include <stdio.h>
void degistir(int *pX, int *pY);
int main(){

   int a = 10, b = 20;
   degistir(&a, &b);
   printf("2- a:%d, b:%d\n", a, b); //Çıktı: a:20, b:10
}
void degistir2(int* pX, int* pY){
   int z = *pX;
   *pX=*pY;
   *pY=z;
}
```

Göstericilerin Dizileri İşaret Etmesi

Çoğu durumda, bir dizi yardımıyla gerçekleştirdiğimiz işleri gösterici ile de gerçekleştirilebiliriz. Genellikle dizilerin kendisi yerine ilk elemanlarını işaret eden göstericiler kullanırız.

```
#include <stdio.h>
int main () {
   int dizi[5] = {10, 20, 30, 40, 50};
   int* ptr = dizi;  // int* ptr=& dizi[0]; olarak da kodlanabilir.

   int indis;
   printf("Dizi elemanlarına gösterici ile erişim: \n");

   for(i = 0; i < 5; i++) {
      printf("dizi[%d]: %d\n", indis, *(ptr+indis));
   }

   return 0;
}
```

Fonksiyonlara dizileri gösteren göstericileri parametre olarak verebiliriz;

– Diziler parametre olarak kullanılırken adres operatörü kullanılmaz. Diziler, gösterici gibi davranırlar. Yani bir dizinin adı, dizinin ilk öğesinin adresi gibi davranır.

```
float notlar[10];
float notlarOrtalamasi(float*); //prototype
//...
float ort=notlarOrtalamasi(notlar); // yada notlarOrtalamasi(&notlar[0]);
//...
float matris[4][3];
float defetminant(float**,int,int); //prototype
//...
float det=defetminant(matris,4,3);
//...
```

– Gösterici gibi davrandıklarından dizi elemanları fonksiyon içinde yine dizi gibi kullanılabilir.

– Gösterici gibi davrandıklarından dizi elemanları fonksiyon içinde değiştirilebilir. Fonksiyondan dönüldüğünde elemanlar değişiklik yapılmış şekliyle işlemler devam eder.

Aşağıda öğrencilerin notlarına ilişkin işlemler yapılan bir program verilmiştir.

```
#include <stdio.h>
#include <math.h>
#define OGRENCISAYISI 10

float notlarOrtalamasi(float*);
void notlariArtir(float*);

int main(){
   float notlar[OGRENCISAYISI]= { 55.0,60.0,70.0,35.0,30.0,
                                  50.0,65.0,90.0,95.0,100.0 };
   float toplam=notlarOrtalamasi(notlar);
   printf("Notlar Ortalaması: %f\n", toplam);
   notlariArtir(notlar);  // Bu fonksiyonla dizi elemanları değiştiriliyor
   int indis;
   for (indis=0; indis<OGRENCISAYISI; indis++)
      printf("Not[%d]=%f\n",indis,notlar[indis]);
   return 0;
}

float notlarOrtalamasi(float* pNotlar){
   int indis;
   float top=0.0;
   for (indis=0; indis<OGRENCISAYISI; indis++)
      top+=pNotlar[indis]; // Gösterici dizi gibi kullanılıyor
   return top/OGRENCISAYISI;
}

void notlariArtir(float* pNotlar) {
   int indis;
   for (indis=0; indis<OGRENCISAYISI; indis++)
      if (pNotlar[indis]<=90) //Göstericinin gösterdiği değerler değiştiriliyor
          pNotlar[indis]+=10.0;

};
/*Program Çıktısı:
Notlar Ortalaması: 65.0
Not[0]=65.0
Not[1]=70.0
Not[2]=80.0
Not[3]=45.0
Not[4]=40.0
Not[5]=60.0
```

```
Not[6]=75.0
Not[7]=100.0
Not[8]=95.0
Not[9]=100.0

...Program finished with exit code 0
*/
```

Fonksiyon Göstericisi

Geri çağırma (callback) fonksiyonları, özellikle olay güdümlü programlamada (event-driven programming) çok kullanışlıdır. Belirli bir olay tetiklendiğinde, bu olaylara yanıt olarak ona eşlenen bir geri çağırma fonksiyonu çalıştırılır. Genellikle grafik ara yüzlü işletim sistemlerinde kullanıcı ara yüzünde bir düğmeye tıklanması gibi bir olay, önceden tanımlanmış bir dizi işlemi başlatabilir. Bu durum, geri çağırma işlevleriyle gerçekleştirilir.

Geri çağırma fonksiyonu, temel olarak, başka bir koda argüman olarak geçirilen ve belirli bir zamanda argümanı geri çağırması beklenen bir icra edilebilir koddur. Geri çağırma mekanizması fonksiyon göstericisine bağlıdır. Fonksiyon göstericisi, bir fonksiyonun bellek adresini depolayan bir değişkendir. Aşağıda buna ilişkin basit bir örnek bulunmaktadır;

```
#include <stdio.h>
void merhaba() {
   printf("Merhaba!\n");
}
void callback(void (*ptr)()) {
   printf("Geri çağırma fonksiyonu çağrılacak!\n");
   (*ptr)();   // Geri çağırma fonksiyonu çağrılıyor
}
main() {
   void (*ptr)() = merhaba;
   callback(ptr);
}
```

Fonksiyonlardan Bir Diziyi Geri Döndürme

C dilinde bir fonksiyonun geri dönüş değeri olarak tüm bir dizi verilemez! Ancak, bir dizinin göstericisini fonksiyondan geri dönüş değeri olarak döndürebilirsiniz. Burada dikkat edilmesi gereken fonksiyondan çıkılınca dizinin bellekten kaldırılmamasıdır. Aşağıda bir tamsayı dizi göstericisini döndüren bir fonksiyon örnekleri verilmiştir;

```
int* tekBoyutluDiziDondur() {
    /*... */
}
int** ikiBoyutluDiziDondur() {
    /*... */
}
```

Yerel değişkenlerin yığın (stack) bellekte tutulduğu ve fonksiyondan geri dönünce fonksiyon yerelindeki değişkenlerin kaybolduğu Depolama Sınıfları başlığında anlatılmıştı. Bu nedenle bir fonksiyon içinde tanımlanan bir dizinin göstericisi geri döndürülecek ise `static` olarak tanımlanır.

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

#define BOYUT 10

int* rastgeleOgrenciNotlari( ) {
    static int  notlar[BOYUT];
    for (int i = 0; i < BOYUT; ++i)
        notlar[i] = rand()%101;
```

```
    return notlar;
}
void notlariYaz(int* pNotlar,int pUzunluk) {
    printf("Öğrenci Notları:\n");
    for (int i = 0; i < pUzunluk; ++i)
        printf("%4d", pNotlar[i]);
    return;
}
int main () {
   int *notlar;
   srand(time(NULL));
   notlar = rastgeleOgrenciNotlari();
   notlariYaz(notlar,BOYUT);
   return 0;
}
/*Program Çıktısı:
Öğrenci Notları:
  30  31  13  40  41  37  34  83  93  76

...Program finished with exit code 0
*/
```

TİP DÖNÜŞÜMLERİ

Üstü Kapalı Tip Dönüşümleri

C dilinde üstü kapalı veri tipi dönüşümü (implicit type casting) ya da standart tip dönüşümüdür (standart type casting), otomatik tip dönüşümü olarak da adlandırılır ve programcının açık talimatlar vermesine gerek kalmadan bir veri tipini otomatik olarak başka bir uyumlu tipe dönüştürmesi işlemidir. Şu durumlarda gerçekleşir;

- Dönüşüm farklı veri tiplerinin değerleri üzerinde gerçekleştirilir.
- Farklı bir veri tipi bekleyen bir fonksiyona bir argüman geçirmeniz halinde gerçekleşir.
- Bir veri tipinin değerini başka bir veri tipinin değişkenine atama durumunda gerçekleşir.

Otomatik tip dönüşümde bellekte az yer kaplayan veri tipinden tanımlanmış değişkenler, bellekte çok yer kaplayan değişkenlere atanmaya (widening casting) çalışıldığında otomatik olarak çok yer kaplayanın veri tipine dönüştürülür.

```
//...
char c;   //1 bayt
int i;    //4 bayt
long l;   //8 bayt
float f;  //4 bayt
double d; //8 bayt
c=65;
i=c; //int veri tipine atanmadan önce; char, int veri tipine dönüştürülür.
l=i; //long veri tipine atanmadan önce; int, long veri tipine dönüştürülür.
f=12.50;
d=f; //double veri tipine atanmadan önce; float, double veri tipine dönüştürülür.
//...
```

Ayrıca bellekte çok yer kaplayan veri tipinden tanımlanmış değişkenler, bellekte az yer kaplayan değişkenlere atanmaya (narrowing casting) çalışıldığında, tip dönüşümü otomatik olarak yapılır ancak veri kaybı söz konusu olur.

```
//...
char c;
int i;
float f;
i=4095; //i=0x00000FFF -> 0x00,0x00,0x0F,0xFF olarak 4 bayt
c=i;    /*c=0xFF -> c değişkenine int veri tipinden char veri tipine en anlamsız
          baytı atanır.*/

f=12.50;
/*
Kayan noktalı bir sayıdan tamsayıya atama yapılıyor ise tam kısmı tamsayıya çevrilir ve
sonrasında atama yapılır.
*/
i=f;    //i=12
//...
```

Bilinçli Tip Dönüşümü

Bazen belli işlemleri daha kontrollü yapmak amacıyla programcı yapılan işlemde işlenen veri tipini bir başka ver tipine kasıtlı olarak değiştirir. Bu tür dönüşümüne bilinçli tür dönüşümü (explicit type casting) denir ve tekli tip dönüştürme (unary cast) işleci (operator) ile yapılır.

Diğer tekli işleçler gibi ifadenin (expression) önünde kullanılır ve parantez içerisinde dönüştürülmek istenen veri tipi yazılarak `int j=(int) 12.8/4;` şeklinde kodlanır. Bu işleç, diğer `+`, `-`, `!`, `--`(pre-decrement), `++`(pre-increment) gibi işleçlerle aynı önceliklidir.

Aşağıda buna ilişkin bir örnek program verilmiştir;

```
#include<stdio.h>
int main() {
   float x = 9.9;
   int y = x+3.3;
   int z= (int)(x)+3.3; // x int veri tipine dönüştürülmüş ve ardından toplama yapılmıştır.

   printf("x: %f\n",x);  // x: 9.900000
   printf("y: %d\n",y);  // y: 13
   printf("z: %d\n",z);  // z: 12
   return 0;
}
```

Üstünlük ve zayıflıklar

Tür dönüşümünün üstünlükleri;

- İşlemlerde Esneklik: Farklı veri tiplerini içeren işlemleri gerçekleştirmede esneklik sağlar. Programcının verileri bir tipten diğerine açıkça dönüştürmesini sağlayarak karışık tip aritmetiğini ve diğer işlemleri kolaylaştırır.
- Uyumluluk: Farklı veri tipleri arasında uyumluluğun sağlanmasına yardımcı olur. Programcının verileri belirli bir bağlamda kullanmadan önce uyumlu bir tipe dönüştürmesini sağlayarak veri uyumsuzluğu hatalarını önler.
- Hassasiyet Kontrolü: Hassasiyet kontrolünün kritik olduğu durumlarda, programcının özellikle sayısal işlemlerde veri tipleri arasında dönüşüm yaparak istenen hassasiyeti açıkça belirtmesini sağlar.
- Açıklık: Tip dönüşümü, programcının veri tipini değiştirme niyetini belirterek kodu daha açık hale getirir. Bu, kod okunabilirliğini artırabilir ve işlenen veri tipiyle ilgili kafa karışıklığını azaltabilir.

Tür dönüşümünün zayıf yönleri;

- Hassasiyet Kaybı: Tip dönüştürmenin en büyük dezavantajlarından biri hassasiyet kaybı potansiyelidir. Örneğin, kayan noktalı bir sayıyı tam sayıya dönüştürürken kesirli kısım kesilir ve bu da bilgi kaybına yol açar.
- Çalışma Zamanı Yükü: Bilinçli tip dönüştürme genellikle program yürütme sırasında dönüştürmenin gerçekleştirilmesi gerektiğinden çalışma zamanı yüküne neden olur. Bu ek işlem, özellikle tip dönüştürmenin sık olduğu durumlarda performansı etkileyebilir.
- Derleyici Uyarıları ve Hataları: Yanlış veya güvenli olmayan tip dönüştürme, derleyici uyarılarına veya hatalarına yol açabilir. Örneğin, uyumsuz tipler arasında dönüştürme yapmaya çalışmak veya geçersiz dönüştürme sözdizimi kullanmak, hata ayıklaması zor olabilecek sorunlara yol açabilir.
- Tanımsız Davranış Potansiyeli: Bazı durumlarda, tip dönüştürme, özellikle uyumsuz tipler arasında dönüştürme yaparken veya dönüştürülen değer hedef tipin aralığının dışında olduğunda tanımsız davranışa yol açabilir. Bu, programda öngörülemez duruma neden olabilir.
- Kod Bakım Zorlukları: Bilinçli tip dönüşümüne kodun bakımı ve anlaşılması, özellikle karmaşık veya iç içe ifadelerde gerçekleştirildiğinde, daha zor hale gelebilir. Bu, artan kod karmaşıklığına ve azalan okunabilirliğe yol açabilir.
- Taşınabilirlik Endişeleri: Bilinçli tip dönüşümüne farklı platformlar veya derleyiciler arasında daha az taşınabilir olabilir. C'de tip dönüşümünün davranışı değişebilir ve belirli veri tipi boyutları hakkındaki varsayımlar tüm ortamlarda geçerli olmayabilir.

NUMARALANDIRMA ve TAKMA İSİMLER

Numaralandırma

C numaralandırması (enumaration), bir grup tamsayılardan oluşan (integral) değişmezden (literal) oluşan numaralandırılmış bir veri türüdür. Bir bütünün parçalarını tamsayı olarak ifade eden değişmezlere kullanıcı tanımlı adlar atamak istediğinizde kullanışlıdır. C dilinde numaralandırılmış sabitler "integral type" olarak adlandırılır ve `enum` anahtar kelimesiyle tanımlanırlar.

```
enum numaralandırmakimliği { DEĞER1, DEĞER2, ...};
```

Sözde kodda `DEĞER1`, `DEĞER2` olarak belirtilenler aslında tamsayılara verilen isimlerdir. Yani tamsayı sabitlerdir. Sabit olduklarından büyük harfle yazılırlar ve belirtilmedikçe ilkinin değeri `0`, ikincisinin değeri `1`, ... şeklinde ilerler. Örnek;

```
#include <stdio.h>
enum cinsiyetKodlari {ERKEK = 1, KADIN = 2, BELIRSIZ = 0 };
/*
Burada tanımlanan tamsayı sabitler aynı konuya (burada cinsiyet) ilişkin bir bütünün parçaları
olan (integral) sabitlerdir.
*/
enum renkKodlari {
   SIYAH=0x000000, // Hiç belirtilmez ise 0
   KIRMIZI=0xFF0000, // Hiç belirtilmez ise 1
   YESIL=0x00FF00, // Hiç belirtilmez ise 2
   MAVI=0x0000FF, // Hiç belirtilmez ise 3
   BEYAZ=0xFFFFFF // Hiç belirtilmez ise 4
};
int main() {
   // Sabitleri konsola yazıyoruz
   printf("ERKEK = %d\n", ERKEK);
   printf("KADIN = %d\n", KADIN);
   printf("BELIRSIZ = %d\n", BELIRSIZ);
   int cinsiyet;
   puts("Cinsiyet Giriniz (0-1-2):");
   scanf("%d",&cinsiyet);
   switch (cinsiyet) {
    case ERKEK: puts("ERKEK Girdiniz"); break;
    case KADIN: puts("KADIN Girdiniz"); break;
    case BELIRSIZ: puts("Cinsiyet Bilinmiyor"); break;
   }
   return 0;
}
```

Takma İsimler

C Dilinde halihazırda mevcut veri tiplerinin adını yeniden tanımlamak ya da takma isim (typedef) verilebilir;

```
typedef mevcutisim takmaisim;
```

Aşağıda takma isim verilmiş yeni veri tipleriyle yazılmış bir kod örneği verilmiştir;

```
#include <stdio.h>

typedef char karakter;
typedef int tamsayi;
typedef long long buyukolceklitamsayi;
typedef unsigned int pozitiftamsayi;

int main() {
```

```
    karakter k='A';
    tamsayi t=12;
    buyukolceklitamsayi bot=0x0FFFFFFF;
    pozitiftamsayi pt=1;

    printf("k:%d veya %c\n",k,k);
    printf("t:%d\n", t);
    printf("bot:%ld\n",bot);
    printf("pt:%u\n",pt);

    return 0;
}
```

Tip tanımlama, daha sonra göreceğimiz yapı (struct) ve birlik (union) veri tiplerinin çokça kullanıldığı kodlarda kodun boyunu oldukça kısaltır. Bunların dışında fonksiyonlara da takma isim verilebilir;

```
typedef int Fonk(int); /* Fonk bir fonksiyona verilen bir takma isimdir;
Fonk, int tipinde parametre alan ve int geri döndüren her fonksiyon olabilir */

int faktoriyel(int n) { //faktoriyel fonksiyonu da Fonk tipindedir.
    return (n==1)? 1: n*faktoriyel(n-1);
}
int onKat(int i) {
    return i*10;
}
int main() {
    Fonk *fn; // fn bir Fonk tipinde olan fonksiyonlara olan göstericidir.
    fn = &faktoriyel; // faktoriyel fonksiyonunu göster
    fn(3); // 3! Hesaplanır
    fn = &onKat; // onKat fonksiyonunu göster
    fn(4); // 4*10 hesaplanır
}
```

DİZGİLER

Dizgi tanımlama

C dilindeki dizgi, dizideki son karakteri '`\0`' yani NULL karakteri (NULL character) olan, `char` veri tipinde tek boyutlu bir dizidir. Bu özel dizilerde metinler tutulur. En yaygın dizgi tanımı aşağıdaki şekilde yapılır;

```
char metin[] = "CLang";
```

Bu tanımda çift tırnak (") karakterleri kullanılarak ilk değer olarak metin kimlikli dizgiye "Clang" metin değişmezi (string literal) atanmıştır. Aslında karakter dizisi olan bu metin aşağıdaki iki şekilde de tanımlanabilir;

```
char metin1[]={'C','L','a','n','g','\0'}; //yada
char metin2[]={'C','L','a','n','g', 0 };
```

Dizgiler karakter göstericisi olarak da tanımlanır. Bu durumda gösterici dizinin ilk elemanını gösterir;

```
char* metin3="CLang";
```

Konsola Metin Yazma

Dizgiler, karakter dizileri olduğunda döngü kullanarak ekrana aşağıdaki şekilde yazabiliriz;

```
for (int i = 0; i < 5; i++) {
    printf("%c", metin[i]);
}
//yada
char* ptrc=&metin;
while (ptrc++!=0)
    printf("%c",*ptrc);
```

Döngü kullanmadan konsola metin yazmak için `printf` fonksiyonunda `%s` yer tutucusu kullanılır.

```
#include <stdio.h>
int main() {
    char metin[]="Clang"; //Metin değişmezler (string literal) çift tırnak arasında verilir.
    char metin1[]={'C','L','a','n','g','\0'}; //ya da
    char metin2[]={'C','L','a','n','g', 0 };
    char* metin3="CLang";

    //for döngüsü ile konsola metin yazma
    for (int i = 0; i < 5; i++)
        printf("%c", metin[i]);
    printf("\n");

    //while döngüsü ile konsola metin yazma
    char* ptrc=&metin[0]; //char *ptr3=metin3;
    while (*ptrc!=0)
        printf("%c",*ptrc++);
    printf("\n");

    //%s yer tutucusu ile konsola metin yazma
    printf("%s\n", metin);
    return 0;
}
```

Klavyeden Metin Okuma

Klavyeden metin okunacak ise ilk önce okunacak karakter sayısına bağlı olarak bir dizgi tanımlanmalıdır;

```
char string[20];
```

Dizgileri klavyeden okumak için `scanf` fonksiyonlarında da `%s` yer tutucusu kullanılır. Belirlenen karakter sayısından fazlası okunduğunda belleğin istenmeyen bölgelerine ulaşılacağından okunacak metnin koyulacağı dizginin boyutuna dikkat edilmelidir.

`scanf` fonksiyonuna okunan değerlerin konulacağı değişkenin adresi `&` karakteri ile verilir. Ancak dizgi olan karakter dizileri `char[]` ile karakter göstericileri `char*` derleyici için aynı olarak kabul edilir. Bu nedenle okunacak dizgi değişkeninin önünde `&` karakteri kullanılmaz. Bu fonksiyonda dizgileri okumak için `%s` yer tutucusu kullanılır.

```
#include <stdio.h>
#include <string.h>
int main (){
   char adi[20];
   char soyadi[20];
   printf("Adınızı Giriniz:");
   scanf("%s", adi);
   printf("Soyadınızı Giriniz:");
   scanf("%s", soyadi);
   printf("Girdiğiniz değerler:");
   printf("%s-%s\n", adi,soyadi);
   return 0;
}
```

Çok Kullanılan Dizgi Fonksiyonları

C dili, dizgiler üzerinde çeşitli işlemler gerçekleştirmek için standart `string.h` başlık dosyasına sahiptir. Bu kütüphanedeki fonksiyonları kullanırken göstericilerin bir karakter dizinini işaret etmesi gerektiği unutulmamalıdır.

Fonksiyon	Açıklama
...	
strcat()	Bir dizgiyi diğerinin sonuna ekler
strchr()	Dizgi içinde aranan bir karakterin ilk bulunanına ait göstericiyi döndürür.
strcmp()	ASCII olarak iki dizgiyi karşılaştırır.
strcpy()	Hafızada yer alan iki dizgiden birinin içeriğini diğerine kopyalar.
strlen()	Dizginin karakter uzunluğunu verir.
strrchr()	Dizgi içinde aranan bir karakterin son bulunanına ait göstericiyi döndürür.
strstr()	Dizgi içinde aranan ikinci bir dizginin ilk bulunanına ait göstericiyi döndürür.
strtok()	Dizgiyi verilen ayraçlara bağlı olarak alt dizgilere böler
...	

Tablo 24. Çok Kullanılan Dizgi Fonksiyonları

strlen() fonksiyonu

Bu fonksiyon dizgideki karakter sayısını verir.

```
#include <stdio.h>
#include <string.h>
int main(){
   char *metin = "Merhaba";
   printf("Dizgi: %s\n", metin);
   printf("Dizgi Uzunluğu: %ld", strlen(metin));
   return 0;
}
```

strcpy() fonksiyonu

Bu fonksiyon, ikinci parametredeki dizgiyi, birinci parametreye kopyalar. Burada dikkat edilmesi gereken husus; hedef gösterilen dizginin karakter sayısının veya göstericinin işaret ettiği bellek bölgesinin büyüklüğünün en az kaynak kadar olması gerektiğidir.

```
#include <stdio.h>
#include <string.h>
int main(){
   char* ptrToDegimmezDizgi = "Ne Haber?";
   char bosDizgi[20]; //Üzerine kopyalanacak metnin karakter sayısı 20 kabul edilmiş!
   char* ptr;
   strcpy(bosDizgi, ptrToDegimmezDizgi);
   printf("%s\n", bosDizgi);
   //strcpy(ptr, ptrToDegimmezDizgi);
   /* HATA: ptr sadece adres tutan değişkendir.
   Şu an için hiçbir dizgiyi göstermiyor! */
   ptr = bosDizgi;
   strcpy(ptr, ptrToDegimmezDizgi);
   printf("%s", ptr);
   return 0;
}
```

Bu fonksiyonu kullanmak yerine kendimiz de dizgi kopyalayan fonksiyon yazabiliriz;

```
#include <stdio.h>
void copyString(char*, char*);
int main(){
    char str1[100] , str2[100]="Deneme Metni";
    printf("str1: %s \nstr2: %s\n", str1,str2);
    copyString(str1, str1);
    printf("str1: %s \nstr2: %s\n", str1,str2);
    return 0;
}
void copyString(char* hedef, char* kaynak){
    int i = 0;
    for(i = 0; kaynak[i]!='\0'; i++)
        hedef[i] = kaynak[i];
    hedef[i] = '\0';
}
```

strcat() fonksiyonu

Bu fonksiyon, parametre olarak girilen iki dizgiyi, birinci parametreye birleştirir.

```
#include <stdio.h>
#include <string.h>
int main() {
    char adi[] = "Ilhan";
    char* soyadi = "OZKAN";
    char adiSoyadi[50]; /* Birleştirme sonrası oluşacak metnin
                           karakter sayısı 50 kabul edilmiş! */
    strcat(adiSoyadi, adi);
    strcat(adiSoyadi, " ");
    strcat(adiSoyadi, soyadi);
    printf("%s", adiSoyadi); // Çıktı: Ilhan OZKAN
    return 0;
}
```

strchr() fonksiyonu

Bu fonksiyon, birinci parametre olarak verilen dizgide, ikinci parametrede verilen karakteri arar. Bulunamaz ise bu fonksiyon NULL gösterici döndürür.

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

int main() {
  char dizgi[] = "Merhaba!";
  char aranan='h';
  char* ptr = strchr(dizgi, aranan);
  if (ptr != NULL) {
    printf("'%c' karakteri \"%s\" dizgisinde %ld indistedir.\n",
           *ptr, dizgi, ptr - dizgi);
  } else {
    printf("'%c' karakteri \"%s\" dizgisinde bulunamadı.\n",
           aranan, dizgi);
  }
  return 0;
}
```

strcmp() fonksiyonu

Bu fonksiyon, parametre olarak girilen iki dizginin, karakterlerini karşılıklı olarak karşılaştırır. Karşılaştırılan iki karakter birbirinden farklı ise ASCII kodu farklarını döndürür. Tüm karakterler aynı ise sıfır döndürür.

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[] = "ARMUT";
    char str2[] = "armut";
    int result = strcmp(str1, str2);

    if (result == 0) {
        puts("İki metin de eşit.");
    } else if (result < 0) {
        puts("Birinci metin ikincisinden küçük");
    } else {
        puts("İkinci metin birincisinden küçük");
    }
    return 0;
}
// Çıktı: Birinci metin ikincisinden küçük
```

Alternatif olarak karşılaştırma fonksiyonunu aşağıdaki gibi yazabiliriz;

```
#include <stdio.h>
int compareString(const char*,const char*);
int main() {
    char str1[] = "Elma";
    char str2[] = "Elma2";
    int result = compareString(str1, str2);
    if (result == 0)
        printf("İki Metin Aynıdır.\n");
    else if (result < 0)
        printf("Birinci Metin İkincisinden Küçüktür:%d\n",result);
    else
        printf("Birinci Metin İkincisinden Büyüktür:%d\n",result);
    return 0;
}
int compareString(const char* metin1,const char* metin2) {
    while (*metin1 == *metin2) {
        if (*metin1 == '\0')  return (0);
        metin1++;
        metin2++;
    }
```

```
    return (*metin1 - *metin2);
}
// Birinci Metin İkincisinden Küçüktür:-50
```

Karakter Fonksiyonları

C dilinde, standart olan bir başka başlık dosyası olan `ctype.h` içinde karakter fonksiyonları bulunur. Bu başlıkta dizgileri oluşturan karakterler üzerinde işlem yapmak için kullanılır. Karakterler aşağıdaki şekilde gruplandırılmıştır;

- Alfasayısal (alfanumeric): hem rakam hem harf barındıran karakter kümesidir.
- Beyaz boşluk: boşluk ‘ ’, yeni satır ‘`\n`’, satır başı ‘`\r`’, alt satır ‘`\v`’, ve tab ‘`\t`’ karakterlerinin oluşturduğu kümedir.
- Onaltılık Sayı Rakamları (hexadecimal digit): `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `a`, `b`, `c`, `d`, `e`, `f`

Aşağıda bu başlık dosyasında yer alan birçok fonksiyon yer almaktadır.

Fonksiyon	Açıklama
`int isalnum(int c)`	Verilen karakter kodunun alfa sayısal olup olmadığını kontrol eder.
`int isalpha(int c)`	Verilen karakter kodunun harf olup olmadığını kontrol eder.
`int iscntrl(int c)`	Verilen karakter kodunun kontrol karakteri olup olmadığını kontrol eder.
`int isdigit(int c)`	Verilen karakter kodunun rakam olup olmadığını kontrol eder.
`int isgraph(int c)`	Verilen karakter kodunun konsola yazdırılabilir karakteri olup olmadığını kontrol eder.
`int islower(int c)`	Verilen karakter kodunun küçük harf olup olmadığını kontrol eder.
`int isprint(int c)`	Verilen karakter kodunun boşluk dahil konsola yazılabilir olup olmadığını kontrol eder.
`int ispunct(int c)`	Verilen karakter kodunun noktalama işareti olan karakterler olup olmadığını kontrol eder.
`int isspace(int c)`	Verilen karakter kodunun beyaz boşluk olup olmadığını kontrol eder.
`int isupper(int c)`	Verilen karakter kodunun büyük harf olup olmadığını kontrol eder.
`int isxdigit(int c)`	Verilen karakter kodunun onaltılık sayı rakamı olup olmadığını kontrol eder.
`int isblank(int c)`	Verilen karakter kodunun boşluk karakteri olup olmadığını kontrol eder.
`int toupper(int c)`	Verilen karakterin büyük harfinin kodunu döner.
`int tolower(int c)`	Verilen karakterin küçük harfinin kodunu döner.

Tablo 25. Ctype.h Karakter Fonksiyonları

Bu fonksiyonlar kullanılarak küçük-büyük harf ayrımı yapmadan metinleri karşılaştıran bir program örneği verilmiştir;

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int compareString(char*, char*);
int main() {
    char str1[] = "Elma";
    char str2[] = "elma";
    int result = compareString(str1, str2);
    if (result == 0)
        printf("İki Metin Aynıdır.\n");
    else if (result < 0)
        printf("Birinci Metin İkincisinden Küçüktür.\n");
    else
        printf("İkinci Metin Birincisinden Küçüktür.\n");
    return 0;
}
int compareString(char* hedef, char* kaynak){
    int i = 0;
    for (i = 0; hedef[i]; i++) //hedefi küçük harfe çevir.
        hedef[i] = tolower(hedef[i]);
```

```
    for (i = 0; kaynak[i]; i++) //kaynağı küçük harfe çevir.
        kaynak[i] = tolower(kaynak[i]);
    return strcmp(hedef, kaynak);
}
```

Metinden İlkel Veri Tiplerine Dönüşüm

Standart başlık dosyalarından olan `stdlib.h` başlık dosyasında daha önce işlediğimiz rastgele fonksiyonlarının yanında metinden ilkel veri tiplerine dönüşüm sağlayan bazı fonksiyonlar da bulunur.

Fonksiyon	Açıklama
`double atof(const char*)`	Dizgi olarak verilen metni `double` tipine dönüştürür ve geri döndürür.
`int atoi(const char*)`	Dizgi olarak verilen metni `int` tipine dönüştürür ve geri döndürür.
`long atol(const char*)`	Dizgi olarak verilen metni `long` tipine dönüştürür ve geri döndürür.
`double strtod(const char*, char**)`	Dizgi olarak verilen metni `double` tipine dönüştürür ve geri döndürür. İkinci parametre ise metinde kayan noktalı sayıya çevrilmeyen ilk karakterin göstericisidir.
`long strtol(const char*, char**, int)`	Dizgi olarak verilen metni `long` tipine dönüştürür ve geri döndürür. İkinci parametre ise metinde kayan noktalı sayıya çevrilmeyen ilk karakterin göstericisidir. Üçüncü parametre ise metnin hangi sayı tabanında sayı içerdiğidir. Bu 2, 8, 10, 16 olabilir.

Tablo 26. Metinden İlkel Veri Tiplerine Dönüşüm Fonksiyonları

Aşağıda bu fonksiyonları içeren bir program örneği verilmiştir.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    char tamsayiMetin[]="-23234";
    char kayannoktalisayiMetin[]="+2.3234e-1";
    char metin[]="-12,3e-2'nin karesi";
    int i = atoi(tamsayiMetin);
    printf("Tamsayı: %d\n",i); // Tamsayı: -23234
    double d= atof(kayannoktalisayiMetin);
    printf("Kayan Noktalı Sayı: %lf\n",d);
                               // Kayan Noktalı Sayı: 0.232340
    double dm;
    char* gerikalanmetin;
    dm= strtod(metin,&gerikalanmetin);
    printf("Metinden Alınan Sayı: %lf\n",dm);
                               // Metinden Alınan Sayı: -12.000000
    printf("Geri Kalan Metin: %s\n",gerikalanmetin);
                               // Geri Kalan Metin: ,3e-2'nin karesi
    return 0;
}
```

Arama Tabloları

Arama tabloları (lookup tables-LUT), önceden hesaplanmış belirli değerlerle doldurulmuş dizilerdir. Uzun iç içe `if`, `if-else` veya `switch` ifadeleri yerine, bir programının verimliliğini artırmak için bu tablolar kullanılabilir.

```
int kareler[10] = {1, 4, 9, 16, 25, 36, 49, 64, 81, 100};
float karekokler[5] = {1.0, 1.41421, 1.7320, 2.0, 2.2360};
```

Bu tablolarda her seferinde hesaplama yapmak yerine önceden hesaplanmış değerleri tutarız. Arama tabloları genellikle dizgilerden oluşur.

```
char* renkler[] ={"Beyaz", "Mavi", "Yeşil", "Kırmızı", "Siyah"};
```

Aşağıda örnek bir program verilmiştir;

```
#include <stdio.h>
enum renlerenum {BEYAZ,MAVI,YESIL,KIRMIZI,SIYAH};
int main() {
    int kareler[10] = {1, 4, 9, 16, 25, 36, 49, 64, 81, 100};
    for (int i = 0; i < 10; i++){
        printf("%d \tKaresi(%d): %d\n", i+1, i+1, kareler[i]);
    }
    float karekokler[5] = {1.0, 1.41421, 1.7320, 2.0, 2.2360};
    for (int i = 0; i < 5; i++){
        printf("%d \tKarekökü(%d): %0.4f\n", i+1, i+1, karekokler[i]);
    }
    char* renkler[] ={"Beyaz", "Mavi", "Yeşil", "Kırmızı", "Siyah"};
    int secim=YESIL;
    printf("Seçtiğiniz renk: %s",renkler[YESIL]);
    return 0;
}
/*Program Çıktısı:
1       Karesi(1): 1
2       Karesi(2): 4
3       Karesi(3): 9
4       Karesi(4): 16
5       Karesi(5): 25
6       Karesi(6): 36
7       Karesi(7): 49
8       Karesi(8): 64
9       Karesi(9): 81
10      Karesi(10): 100
1       Karekökü(1): 1.0000
2       Karekökü(2): 1.4142
3       Karekökü(3): 1.7320
4       Karekökü(4): 2.0000
5       Karekökü(5): 2.2360
Seçtiğiniz renk: Yeşil

...Program finished with exit code 0
*/
```

YAPILAR VE BİRLİKLER

Yapılar

Yapı Tanımlaması

Yapısal programlamaya adını veren yapı (structure), türetilmiş (derived) veya kullanıcı tanımlı (user defined) bir veri tipidir. Farklı tiplerdeki elemanları bir arada gruplandıran özel bir veri tipi tanımlamak için `struct` anahtar kelimesini kullanırız.

Yapı bir veya birden fazla ilkel veri tipinin (`char`, `int`, `long`, `float`, `double`, ... ve bu tiplere ait diziler) bir araya gelmesiyle oluşturulan yeni veri tipleridir. Diziler, aynı tipte elemanlardan oluşmasına rağmen, yapılar farklı dipte elemanların bir araya gelmesiyle oluşabilir;

```
struct yapı-kimliği {
    veri-tipi yapı-elemanı-kimliği1;
    veri-tipi yapı-elemanı-kimliği1;
    ...
} değişken-kimliği1, değişken-kimliği2;
```

Örnek olarak Öğrenci; adı, soyadı, numarası, yaşı, cinsiyeti gibi farklı öğeler ile bir `ogrenci` yapısı tanımlanabilir;

```
struct ogrenci {
    char adi[50];
    char soyadi[0];
    int yas;
    int cinsiyet;
} ogrenci1;
```

Tanımlanan yapı kullanılarak değişkenler aşağıdaki gibi tanımlanabilir;

```
struct yapı-kimliği değişken-kimliği;
```

Yukarıda tanımlanan yapı kimliği kullanılarak birçok değişken kimliklendirilebilir;

```
struct ogrenci ogrenci2,ogrenci3;
```

Yapı Elemanlarına Erişim

Tanımlanan yapı değişkenleri üzerinden her bir alamana nokta işleci (dot operator) erişilir. Bu işleç de öncelik işleci parantez `()` ile aynı önceliktedir. Atama işleci (`=`) bir yapıyı (struct) doğrudan kopyalamak için kullanılabilir. Ayrıca, bir yapının üyesinin değerini başka birine atamak için atama işlecini de kullanabiliriz.

```
#include <stdio.h>
#include <string.h>
int main() {
    struct ogrenci {
        char adi[50];
        char soyadi[50];
        int yas;
        int cinsiyet;
    } ogrenci1;
    strcpy(ogrenci1.adi, "Ilhan");
    strcpy(ogrenci1.soyadi, "OZKAN");
    ogrenci1.yas=50;
    ogrenci1.cinsiyet=1;
    printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d\n",
          ogrenci1.adi, ogrenci1.soyadi,
          ogrenci1.yas, ogrenci1.cinsiyet);
```

```
    struct ogrenci ogrenci2={"Deniz","AK",35,2}; /* yapı değişkenine
                                                    ilk değer veridi */
    printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d\n",
          ogrenci2.adi, ogrenci2.soyadi,
          ogrenci2.yas, ogrenci2.cinsiyet);
    struct ogrenci ogrenci3=ogrenci2;
    printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d\n",
          ogrenci3.adi, ogrenci3.soyadi,
          ogrenci3.yas, ogrenci3.cinsiyet);
    return 0;
}
```

Yapı Göstericileri

Yapı göstericileri, tıpkı diğer değişkenlere gösterici tanımladığımız gibi tanımlayabiliriz. Gösterici üzerinden yapı değişkenlerine dolaylı işleç (indirection operator) yani (`->`) ile erişilir. Bu nokta işleci ile aynı önceliklidir.

Yapılar ve göstericileri; veri tabanları, dosya yönetim uygulamaları ve ağaç ve bağlı listeler gibi karmaşık veri yapılarını işlemek gibi farklı uygulamalarda kullanılır.

```
#include <stdio.h>
#include <string.h>
int main() {
  struct ogrenci {
    char adi[50];
    char soyadi[50];
    int yas;
    int cinsiyet;
  } ogrenci1;
  strcpy(ogrenci1.adi, "Ilhan");
  strcpy(ogrenci1.soyadi, "OZKAN");
  ogrenci1.yas=50;
  ogrenci1.cinsiyet=1;
  struct ogrenci* ogrenciGosterici;
  ogrenciGosterici=&ogrenci1;
  printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d",
    ogrenciGosterici->adi,
    ogrenciGosterici->soyadi,
    ogrenciGosterici->yas,
    ogrenciGosterici->cinsiyet);
  return 0;
}
```

Yapılarla İşlemler

Bir yapı (struct) değişkeni, ilkel tiplerden (`char`, `int`, `float`, ...) tanımlanan bir diziye benzer şekilde bir yapı dizisi tanımlayabiliriz. Ayrıca yapı değişkenini bir fonksiyona parametre olarak gönderebilir ve bir fonksiyondan bir yapı döndürebilirsiniz.

```
#include <stdio.h>
#include <string.h>
enum cinsiyet {BELIRTILMEMIS,KADIN,ERKEK};
struct ogrenci {
    char adi[50];
    char soyadi[50];
    int yas;
    int cinsiyet;
};

struct ogrenci yeniOgrenci();
void ogreciYaz(struct ogrenci);
```

```
int main() {
  struct ogrenci ogrenciler[30];
  struct ogrenci o=yeniOgrenci();
  ogreciYaz(o);
  return 0;
}
struct ogrenci yeniOgrenci() {
    struct ogrenci yeni={"Ilhan","Ozkan",50,ERKEK};
    return yeni;
}
void ogreciYaz(struct ogrenci pOgrenci) {
    printf("Adı:%s\nSoyad:%s\nYaşı:%d\nCinsiyeti:%d",
        pOgrenci.adi,
        pOgrenci.soyadi,
        pOgrenci.yas,
        pOgrenci.cinsiyet);
}
```

Yapılar değer tipler olduğundan, yapılara ait göstericileri parametre olarak kullanmak, olabilecek değişiklikleri aktarmak için üstünlük sağlar.

```
#include <stdio.h>
#include <string.h>
enum cinsiyet {BELIRTILMEMIS,KADIN,ERKEK};
struct ogrenci {
    char adi[50];
    char soyadi[50];
    int yas;
    int cinsiyet; //1:erkek,2:Kadın,0:Belirtmiyor
};

struct ogrenci yeniOgrenci();
void ogrenciOku(struct ogrenci*);

int main() {
  struct ogrenci o=yeniOgrenci();
  ogrenciOku(&o);
  return 0;
}

struct ogrenci yeniOgrenci() {
    struct ogrenci yeni={"Ilhan","Ozkan",50,ERKEK};
    return yeni;
}
void ogrenciOku(struct ogrenci* pOgrenci) {
    printf("Adı Giriniz:");
    scanf("%s",pOgrenci->adi);
    printf("Soyadı Giriniz:");
    scanf("%s",pOgrenci->soyadi);
    printf("Yaş Giriniz:");
    scanf("%d",&(pOgrenci->yas));
    printf("Cinsiyet Giriniz (0-1-2):");
    scanf("%d",&(pOgrenci->cinsiyet));
}
```

Anonim Yapılar

Anonim yapı, kimlik veya takma isim (`typedef`) ile tanımlanmayan bir yapı tanımıdır. Genellikle başka bir yapının içine yerleştirilir. 2011 C sürümü ile kullanılmaya başlanan bu özelliğin aşağıda sıralanan üstünlükleri vardır;

– Esneklik (flexibility): Anonim yapılar, verilerin nasıl temsil edildiği ve erişildiği konusunda esneklik sağlayarak daha dinamik ve çok yönlü veri yapılarına olanak tanır.

- Kolaylık (convenience): Bu özellik, farklı veri tiplerini tutabilen bir değişkenin kompakt bir şekilde temsil edilmesine olanak tanır.
- Başlatma Kolaylığı (easy of initialization): Yapı değişkenine ilişkin ek kimliklendirme yapılmadan ilk değer verilmeleri ve kullanılmaları daha kolay olabilir.

```
#include <stdio.h>
#include <string.h>
struct ogrenci {
    char adi[50];
    char soyadi[50];
    int yas;
    int cinsiyet; //1:erkek,2:Kadın,0:Belirtmiyor
    struct {
      int gun;
      int ay;
      int yil;
    };
};
struct ogrenci yeniOgrenci();
void ogrenciYaz(struct ogrenci);
int main() {
  struct ogrenci o=yeniOgrenci();
  ogrenciYaz(o);
  return 0;
}
struct ogrenci yeniOgrenci() {
    struct ogrenci yeni={"Ilhan","Ozkan",50,1, {1,1,1970}};
    return yeni;
}
void ogrenciYaz(struct ogrenci pOgrenci) {
    puts(pOgrenci.adi);
    puts(pOgrenci.soyadi);
    printf("Yas:%d\n",pOgrenci.yas);
    printf("Cinsiyet:%d\n",pOgrenci.cinsiyet);
    printf("Dogum Tarihi:%d-%d-%d\n",pOgrenci.gun,pOgrenci.ay,pOgrenci.yil);

}
```

Öz Referanslı Yapılar

Kendi kendine yani öz referanslı yapı (self-referetial struct), öğelerinden bir veya daha fazlası kendi türündeki göstericilerden oluşan bir yapıdır. Kendi kendine referanslı kullanıcı tanımlı yapılar, bağlantılı listeler ve ağaçlar gibi karmaşık ve dinamik veri yapıları (dynamic data structure) için yaygın olarak kullanılırlar.

```
#include <stdio.h>
struct ogrenci {
    char adi[10]; char soyadi[10]; int yas;
    struct ogrenci* sonrakiOgrenci;
};
void ogrecileriYaz(struct ogrenci*);
int main() {
  struct ogrenci ilk={"Ilhan","OZKAN",50,NULL};
  struct ogrenci sonraki={"Ayse","YILMAZ",40,NULL};
  ilk.sonrakiOgrenci=&sonraki;
  struct ogrenci birsonraki={"Tahsin","BULUT",35,NULL};
  sonraki.sonrakiOgrenci=&birsonraki;
  ogrecileriYaz(&ilk);
  return 0;
}
void ogrecileriYaz(struct ogrenci* pIlk) {
    int sayac=0;
```

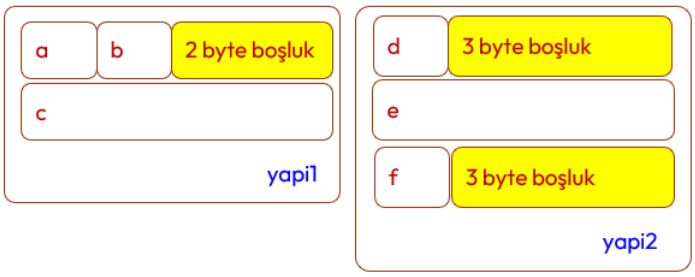
```
    while (pIlk!=NULL) {
        printf("--OGRENCI:%d--\nAdı:%s\nSoyad:%s\nYaşı:%d\n",
        ++sayac, pIlk->adi, pIlk->soyadi, pIlk->yas);
        pIlk=pIlk->sonrakiOgrenci;
    }
}
/*Program Çıktısı:
--OGRENCI:1--
Adı:Ilhan
Soyad:OZKAN
Yaşı:50
--OGRENCI:2--
Adı:Ayse
Soyad:YILMAZ
Yaşı:40
--OGRENCI:3--
Adı:Tahsin
Soyad:BULUT
Yaşı:35

...Program finished with exit code 0
*/
```

Yapı Dolgusu

C dilinde yapı dolgusu (structure padding), işlemci (CPU) mimarisi ile belirlenir. Dolgu (padding) işlemi ile üyelerinin bellekte doğal olarak hizalanması için belirli sayıda boş bayt eklenir. Bunun nedeni 32 veya 64 bitlik bir bilgisayarda işlemcinin tek seferde bellekten 4 bayt okumasından kaynaklanmaktadır.

```
#include <stdio.h>
struct yapi1 {
   char a;
   char b;
   int c;
};
struct yapi2 {
   char d;
   int e;
   char f;
};
int main() {
   printf("char bellek miktarı: %d\n", sizeof(char));
   // char bellek miktarı: 1
   printf("int  bellek miktarı: %d\n", sizeof(int));
   // int  bellek miktarı: 4
   printf("-----------------------\n");
   printf("yapilar için olmasi gereken bellek miktarı: %d\n",
      2*sizeof(char)+sizeof(int));
   // yapilar için olmasi gereken bellek miktarı: 6
   printf("yapi1 için bellekte ayrılan miktar: %d\n", sizeof(struct yapi1));
   // yapi1 için bellekte ayrılan miktar: 8
   printf("yapi2 için bellekte ayrılan miktar: %d\n", sizeof(struct yapi2));
   // yapi2 için bellekte ayrılan miktar: 12
   return 0;
}
```

Yukarıda verilen örnekte aynı bellek miktarına sahip elemanlar için yapının bütününe bakıldığında farklı miktarda bellek ayrılabileceği aşağıdaki şekilden anlaşılmaktadır;



Şekil 21. Yapı Dolgusu

Dolgu (padding) işlemi ile üyelerinin bellekte doğal olarak hizalanması için belirli sayıda boş bayt eklenir. Bunu tüm yapılar için engellemenin yolu;

```
#pragma pack(1)
```

Ön işlemci yönergesini (preprocessor directive) kaynak koda eklemektir. Eğer yalnızca belirlenen yapı için bunun yapılması isteniyorsa yapı tanımına `__attribute__((packed))` özelliği eklenir. Bu durumun engellendiği program aşağıdaki verilmiştir.

```
#include <stdio.h>
#pragma pack(1)
struct yapi1 {
   char a;
   char b;
   int c;
};
struct __attribute__((packed)) yapi2 {
   char a;
   int b;
   char c;
};
int main() {
   printf("char bellek miktarı: %d\n", sizeof(char));
   // char bellek miktarı: 1
   printf("int  bellek miktarı: %d\n", sizeof(int));
   // int  bellek miktarı: 4
   printf("-----------------------\n");
   printf("yapılar için olması gereken bellek miktarı: %d\n",
      2*sizeof(char)+sizeof(int));
   // yapılar için olması gereken bellek miktarı: 6
   printf("yapi1 için bellekte ayrılan miktar: %d\n", sizeof(struct yapi1));
   // yapi1 için bellekte ayrılan miktar: 6
   printf("yapi2 için bellekte ayrılan miktar: %d\n", sizeof(struct yapi2));
   // yapi1 için bellekte ayrılan miktar: 6
   return 0;
}
```

Birlikler

Birlikler (union), Pascal dilindeki `record case` talimatına (statement) benzer. Birlik (union), yapı (struct) gibi tanımlanır. Aralarındaki fark yapı elemanlarının her birine ayrı bellek bölgesi ayrılırken, birlik üyelerinin her biri aynı bellek bölgesini paylaşırlar.

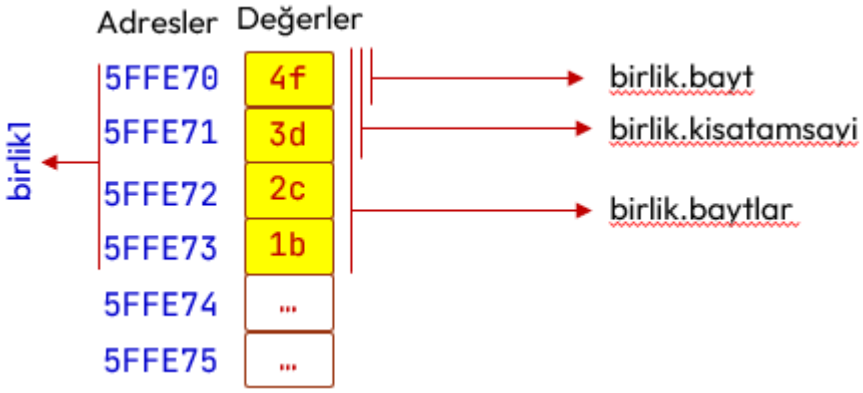
```
union yapı-kimliği {
    veri-tipi yapı-elemanı-kimliği1;
    veri-tipi yapı-elemanı-kimliği1;
    ...
} değişken-kimliği1, değişken-kimliği2;
```

Birliğin belek boyutu, elemanlarından en fazla bellek kaplayanı kadardır. Aşağıda üyelerinin aynı bellek bölgesini paylaştığı görülmektedir.

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

```
int main() {
  union tamsayiBirligi {
    char baytlar[4];
    int  tamsayi;
    char bayt;
    short int kisatamsayi;
  } birlik1,birlik2;
  union tamsayiBirligi birlik3,birlik4;
  birlik1.tamsayi=0x1B2C3D4F;
          //16lık sayılarda her çift rakam bir bayt olur
  printf("sizeof tamsayiBirligi.baytlar: %d\n", sizeof birlik1.baytlar);
          //4
  printf("sizeof tamsayiBirligi.bayt: %d\n", sizeof birlik1.bayt);
          //1
  printf("sizeof tamsayiBirligi.tamsayi: %d\n", sizeof birlik1.tamsayi);
          //4
  printf("sizeof tamsayiBirligi.kisatamsayi: %d\n",
          sizeof birlik1.kisatamsayi);
          //2
  printf("---------------------------------------\n");
  printf("sizeof tamsayiBirligi: %d\n", sizeof birlik1);
          //4
  printf("---------------------------------------\n");
  printf("birlik1.tamsayi:%x\n",birlik1.tamsayi);         //1b2c3d4f
  printf("birlik1.baytlar:%x-%x-%x-%x\n",
         birlik1.baytlar[0],
         birlik1.baytlar[1],
         birlik1.baytlar[2],
         birlik1.baytlar[3]);                             //4f-3d-2c-1b
  printf("birlik1.bayte:%x\n",birlik1.bayt);               //4f
  printf("birlik1.kisatamsayi:%x\n",birlik1.kisatamsayi); //3d4f
  printf("---------------------------------------\n");
  birlik1.bayt=0x00;
  printf("birlik1.tamsayi:%x\n",birlik1.tamsayi);         //1b2c3d00
  return 0;
}
```

Şekil olarak da birlik1 değişkeninin bellek yerleşimi aşağıda verilmiştir;



Şekil 22. Birlik1 Değişkeninin Bellek Yerleşimi

DİNAMİK BELLEK YÖNETİMİ

Göstericilerin (pointer) bellek adreslerini tutan değişkenler olduğu daha önce açıklanmıştı. C yapısal bir dil olduğundan, programlama için bazı sabit kurallara sahiptir. Örneğin bir dizinin boyutunu tanımlandıktan sonra değiştirmezsiniz. Dinamik bellek yönetimi ile çalıştırma anında istenilen boyuta göre dizi oluşturulabilir. Çalıştırma anında istenilen boyutta oluşturulabilen bu dizileri de göstericiler (pointer) sayesinde işleriz.

Dinamik olarak tahsis edilen bellekteki veriler öbek (heap) bellekte tutulur. Yerel değişkenlerin son konulan verinin ilk geri alındığı yığın (stack) bellekte tutulduğu, statik ve global değişkenlerin de veri (data) bellekte tutulduğu daha önce anlatılmıştı.

Çalıştırma anındaki bellek tahsisine dinamik bellek tahsisi (dynamic memory allocation) denir. Bellek tahsisine ilişkin fonksiyonlar `stdlib.h` başlık dosyasında bulunmaktadır. Bu başlıktaki `malloc()`, `calloc()` ve `realloc()` fonksiyonları kullanılarak yapılan dinamik bellek tahsisi yapılır `free()` fonksiyonu ile tahsis edilen belek bölgesi serbest bırakılır.

malloc() fonksiyonu

```
ptr = (cast-type*) malloc(byte-size);
```

Bu fonksiyon, belirtilen boyutta tek bir büyük bellek bloğunu bayt cinsinden dinamik olarak tahsis etmek için kullanılır. Herhangi bir biçimdeki bir göstericiye dönüştürülebilen `void` türünde bir gösterici döndürür.

```
int* ptr1;
ptr1 = (int*) malloc(50 * sizeof(int)); /* 50 adet int içerecek bellek bloğu tahsis edilerek
göstericisi ptr1 değişkenine atandı. Bellek ayrılamasaydı NULL gösterici geri döndürecekti.*/
```

calloc() fonksiyonu

```
ptr = (cast-type*) calloc(n, element-size);
```

Bu fonksiyon, belirtilen veri tipindeki belirtilen sayıda bellek bloğunu birbirine bitişik olarak (contiguous) dinamik olarak tahsis etmek için kullanılır.

```
float* ptr2;
ptr2 = (float*) calloc(20, sizeof(float)); /* 20 adet float içerecek bellek bloğu tahsis
edilerek göstericisi ptr2 değişkenine atandı. Bellek ayrılamasaydı NULL gösterici geri
döndürecekti.*/
```

realloc() fonksiyonu

```
ptr = realloc(ptr, newSize);
```

Bu fonksiyon, daha önce tahsis edilmiş bir belleğin miktarını değiştirmek için kullanılır.

```
ptr1 = (int*) realloc(ptr1, 10 * sizeof(int));
ptr2 = (float*) realloc(ptr2, 40 * sizeof(float));
/* Bellek yeniden ayrılamasaydı NULL gösterici geri döndürecekti.*/
```

free() fonksiyonu

```
free(ptr);
```

Bu fonksiyon, tahsis edilen belleği serbest bırakmak için kullanılır. `malloc()`, `calloc()` ve `realloc()` fonksiyonları kullanılarak tahsis edilen bellek serbest bırakılır.

```
free(ptr1);
ptr1 = NULL;
```

Örnek Program

Aşağıda öğrenci sayısını çalıştırma anında alıp, bu öğrencilere ilişkin notları rastgele belirleyen bir program verilmiştir.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int* notlar=NULL;
    int n, i;
    srand(time(NULL)); //Notlar rastgele atanacak.

    printf("Öğrenci Sayısını Giriniz:");
    scanf("%d",&n);
    printf("%d Elemanlı Notlar Dizisi Oluşturulacak\n", n);

    notlar = (int*) malloc(n * sizeof(int));
    if (notlar == NULL) {
        printf("Hafıza tahsis edilemedi!\n");
        exit(-1); //İşletim sistemine hata ile dönülüyor.
    } else {
        printf("Hafıza başarılı bir şekilde tahsis edildi.\n");
        for (i = 0; i < n; ++i)
            notlar[i] = rand()%101;
        printf("Öğrenci Notları:\n");
        for (i = 0; i < n; ++i)
            printf("%d, ", notlar[i]);
    }
    free(notlar);
    return 0;
}
```

Aşağıda karakter sayısını çalıştırma anında alıp, yığın bellekte bir dizgi (string) oluşturan bir program verilmiştir.

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

int main() {
    char* string;
    int n;
    printf("Girilen Metin En Fazla Kaç Karakter:");
    scanf("%d",&n);

    string = (char*) malloc(n);
    if (string == NULL) {
        printf("Hafıza tahsis edilemedi!\n");
        exit(-1); //İşletim sistemine hata ile dönülüyor.
    } else {
        printf("Hafıza başarılı bir şekilde tahsis edildi.\n");
        puts("Metni Giriniz:");
        scanf("%s",string);
        printf("Girilen Metin:\n%s",string);
    }
    return 0;
}
```

Dinamik Veri Yapıları

Çalıştırma anında bellek tahsis edilerek kullanılan veri yapılarına dinamik veri yapıları (dynamic data structure) denir. Bunlar; Bağlı Listeler (linked list), İstifler (stack), Kuyruklar (queue), İkili Ağaçlar

(binary tree) ve Sözlüktür (dictionary). Buradaki veri yapılarının çoğu bir önceki bölümde anlatılan öz referanslı yapılar (self-referential structure) kullanılarak hayata geçirilir.

Bağlantılı Liste

Bağlantılı liste (linked list), düğüm olarak bilinen her bir öğenin göstericiler kullanılarak bir sonraki düğüme bağlandığı doğrusal bir veri yapısıdır. Diziden farklı olarak, bağlantılı listenin öğelerinin her biri (düğüm olarak adlandırılmaktadır), öbek (heap) bellekte ayrı ayrı oluşturulduğundan, rastgele bellek konumlarında saklanır.

Düğümlerde saklanan veri tek bir değişken olacağı gibi, birden fazla değişken de olabilir. Bağlantılı liste;

- Aşağıda verilen örnekte görüldüğü gibi tek yönlü olabileceği gibi,
- Hem sonraki düğümü hem de önceki düğümü gösteren iki yönlü liste,
- Veya son ve ilk düğümü olmayan halka şekilde liste tanımlanabilir.

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

struct dugumYapi {
  int veri;
  //char veri2; ...
  struct dugumYapi* sonraki;
};
typedef struct dugumYapi Dugum;

void ilkDugumeEkle(Dugum**,int);
void sonaDugumEkle(Dugum*,int);
void listeyiYaz(Dugum*);
void ilkDugumuSil(Dugum**);
void sonDugumuSil(Dugum*);

int main() {
    Dugum* ilk=NULL;  //Başlangıçta hiç düğüm yok.
    ilkDugumeEkle(&ilk,10);
    sonaDugumEkle(ilk,20);
    sonaDugumEkle(ilk,30);
    ilkDugumeEkle(&ilk,-10);
    listeyiYaz(ilk);
    return 0;
}

void ilkDugumeEkle(Dugum** ilkDugumGostericisi, int pVeri) {
    Dugum * yeni;
    yeni = (Dugum *) malloc(sizeof(Dugum));//yeni düğüm için bellek tahsisi
    yeni->veri = pVeri;//tahsis edilen düğüme veri konuluyor
    yeni->sonraki = *ilkDugumGostericisi; // sonraki düğüm ilkDugumGostericisi olsun.
    *ilkDugumGostericisi = yeni;//İlk düğüm göstericisi yeni düğüm olsun
}

void sonaDugumEkle(Dugum* pIlk, int pVeri) {
    Dugum* hangi = pIlk; //hangi ilk düğümü göstersin.
    if (hangi==NULL) {
        printf("Liste olmadığından eklenemedi!\n");
        return;
    }
    while (hangi->sonraki != NULL) {
        hangi = hangi->sonraki; //listenin sonuna kadar ilerle
    }
    hangi->sonraki = (Dugum*) malloc(sizeof(Dugum)); //sonuna yeni düğüm ekle
    hangi->sonraki->veri = pVeri; //yeni düğüme verisi konuluyor
    hangi->sonraki->sonraki = NULL; //son düğümün sonraki düğüm NULL olsun.
```

```
}

void listeyiYaz(Dugum* pIlk) {
    Dugum* hangi = pIlk;  int sayac=0;
    while (hangi != NULL) {
        printf("Dugum (%d): veri=%d\n",
               ++sayac, hangi->veri);
        hangi = hangi->sonraki;
    }
}

void ilkDugumuSil(Dugum** ilkDugumGostericisi) {
    Dugum* sonrakiDugum = NULL;
    if (*ilkDugumGostericisi == NULL) return;
    sonrakiDugum = (*ilkDugumGostericisi)->sonraki;
    free(*ilkDugumGostericisi);
    *ilkDugumGostericisi = sonrakiDugum;
}

void sonDugumuSil(Dugum* pIlk) {
    if (pIlk->sonraki == NULL) {
        free(pIlk);
        return;
    }
    Dugum* hangi = pIlk;
    while (hangi->sonraki->sonraki != NULL)
        hangi = hangi->sonraki;
    free(hangi->sonraki);
    hangi->sonraki = NULL;
}
```

Yığın Veri Yapısı

Yığınlar (stack), bağlantılı listenin (linked list), kısıtlanmış halidir. Haliyle doğrusal bir veri yapısıdır. Yığın veri yapısında;

- Bağlantılı listeye yeni düğüm, ancak listenin başına eklenir. Bu işlem itme (push) olarak adlandırılır.
- Bağlantılı listede silme işlemi yalnızca listenin başındaki düğüme uygulanır. Bu işleme çekme (pop) adı verilir.
- Böylece listeye son giren veri ilk alınır (last in first out-LIFO).

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

struct dugumYapi {
  int veri;
  struct dugumYapi* sonraki;
};
typedef struct dugumYapi Dugum;

void it(Dugum**,int);
int cek(Dugum**);

int main() {
  Dugum* istif=NULL;
  it(&istif,10);
  it(&istif,20);
  it(&istif,30);
  int veri=cek(&istif);
  printf("Çekilen Veri:%d\n",veri);
  veri=cek(&istif);
  printf("Çekilen Veri:%d\n",veri);
  return 0;
```

```
}

void it(Dugum** ilkDugumGostericisi, int pVeri) {
    Dugum* yeni;
    yeni = (Dugum *) malloc(sizeof(Dugum));
    yeni->veri = pVeri;
    yeni->sonraki = *ilkDugumGostericisi;
    *ilkDugumGostericisi = yeni;
}

int cek(Dugum** ilkDugumGostericisi) {
    Dugum * sonrakiDugum = NULL;
    if (*ilkDugumGostericisi == NULL) {
        printf("İstifde düğüm yok!");
        exit(-1);
    }
    int veri=(*ilkDugumGostericisi)->veri;
    sonrakiDugum = (*ilkDugumGostericisi)->sonraki;
    free(*ilkDugumGostericisi);
    *ilkDugumGostericisi = sonrakiDugum;
    return veri;
}
```

Kuyruk Veri Yapısı

Kuyruk (queue) de, bağlantılı listenin (linked list), kısıtlanmış halidir. Haliyle bu veri yapısı da doğrusal bir veri yapısıdır. Kuyruk veri yapısında;

- Bağlantılı listeye yeni düğüm, ancak listenin başına eklenir. Bu işlem, kuyruğa sokma (enqueue) olarak adlandırılır.
- Bağlantılı listede silme işlemi yalnızca listenin sonundaki düğüme uygulanır. Bu işleme kuyruktan çıkarma (dequeue) adı verilir.
- Böylece listeye ilk giren veri ilk alınır (first in first out-FIFO).

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

struct dugumYapi {
  int veri;
  struct dugumYapi* sonraki;
};
typedef struct dugumYapi Dugum;

void enqueue(Dugum**,int);
int dequeue(Dugum*);

int main() {
  Dugum* kuyruk=NULL;
  enqueue(&kuyruk,10);
  enqueue(&kuyruk,20);
  enqueue(&kuyruk,30);
  int veri=dequeue(kuyruk);
  printf("Alınan Veri:%d\n", veri);
  veri=dequeue(kuyruk);
  printf("Alınan Veri: %d\n",veri);
  return 0;
}

void enqueue(Dugum** ilkDugumGostericisi, int pVeri) {
    Dugum* yeni;
    yeni = (Dugum*) malloc(sizeof(Dugum));
    yeni->veri = pVeri;
```

```
    yeni->sonraki = *ilkDugumGostericisi;
    *ilkDugumGostericisi = yeni;
}

int dequeue(Dugum* pIlk) {
    int veri;
    if (pIlk->sonraki == NULL) {
        veri=pIlk->veri;
        free(pIlk);
        return veri;
    }
    Dugum* hangi = pIlk;
    while (hangi->sonraki->sonraki != NULL)
        hangi = hangi->sonraki;

    veri=hangi->sonraki->veri;
    free(hangi->sonraki);
    hangi->sonraki = NULL;
    return veri;
}
```

İkili Ağaç Veri Yapısı

Bu veri yapısı, bir kök ve dallarından oluşan veri yapısıdır. İkili (binary) denmesinin sebebi bir düğümden, genellikle sağ ve sol olarak adlandırılan, yalnıza iki dal çıkabildiğindendir. Haliyle bu veri yapısı da doğrusal olmayan (nonlinear) hiyerarşik bir veri yapısıdır.

```
struct dugumYapi {
  int veri;
  //char veri2;
  //float veri3;
  //...
  struct dugumYapi* sag;
  struct dugumYapi* sol;
};
typedef struct dugumYapi Dugum;
```

Ağacın ilk düğümüne kök düğüm (root node) adı verilir. Kök düğümden dallanan iki düğüm çocuk düğüm (child node) olarak adlandırılır. Bu şekilde her dala eklenen düğüm ile ters bir ağaç oluşur. En uçtaki çocuk düğüme yaprak düğüm (leaf node) adı verilir. İkili bir ağaçta gerçekleştirilebilecek temel işlemler şunlardır:

- Ekleme (insertion)
- Silme (deletion)
- Arama (search) ve
- Gezinme (traversing). Üç şekilde yapılır;
  - Ön Sıralı (pre-order traversal)
  - Son Sıralı (post-order traversal)
  - Sıralı (in-order traversal)

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

struct agacDugum {
    int veri;
    struct agacDugum* sol;
    struct agacDugum* sag;
};
typedef struct agacDugum Dugum;

Dugum* yeniDugum(int pVeri) {
    Dugum* yeni = (Dugum*)malloc(sizeof(Dugum));
```

```
    if (yeni != NULL) {
        yeni->veri = pVeri; // Tahsis yapılan düğüme veri aktarılıyor
        yeni->sol = NULL;
        yeni->sag = NULL;
    }
    return yeni;
}

Dugum* dugumEkle(Dugum* kok, int pVeri) {
    if (kok == NULL)
        return yeniDugum(pVeri); // Ağaç yoksa yeni düğüm ekle
    if (pVeri < kok->veri) // Eklenecek veriyi karşılaştır
        kok->sol = dugumEkle(kok->sol, pVeri);
    else if (pVeri > kok->veri)
        kok->sag = dugumEkle(kok->sag, pVeri);
    return kok; // Değişen kök düğümü döndür.
}

void siraliGezinme(Dugum* kok) {
    //   Sıralı Gezinti: sol altagac, kok, sag altagac
    if (kok != NULL) {
        siraliGezinme(kok->sol);
        printf("%d ", kok->veri);
        siraliGezinme(kok->sag);
    }
}

void onSiraliGezinme(Dugum* kok) {
    //   Ön Sıralı Gezinti: kok, sol altagac, sag altagac
    if (kok != NULL) {
        printf("%d ", kok->veri);
        onSiraliGezinme(kok->sol);
        onSiraliGezinme(kok->sag);
    }
}

void sonSiraliGezinme(Dugum* kok) {
    //   Ön Sıralı Gezinti: kok, sol altagac, sag altagac
    if (kok != NULL) {
        sonSiraliGezinme(kok->sol);
        sonSiraliGezinme(kok->sag);
        printf("%d ", kok->veri);
    }
}

void agaciBelllektenKaldir(Dugum* kok) {
    if (kok != NULL) {
        agaciBelllektenKaldir(kok->sol);
        agaciBelllektenKaldir(kok->sag);
        free(kok);
    }
}

int main() {
    Dugum* kok = NULL;
    int dugumVerisi;
    char secim;
    kok=dugumEkle(kok, 20);
    kok=dugumEkle(kok, 30);
    kok=dugumEkle(kok, 40);
    kok=dugumEkle(kok, 50);
    kok=dugumEkle(kok, 60);
```

```
    kok=dugumEkle(kok, 70);
    kok=dugumEkle(kok, 80);
    printf("\nSıralı Gezinti ile Ağaç: ");
    siraliGezinme(kok);
    printf("\n");
    printf("\nOn Sıralı Gezinti ile Ağaç: ");
    onSiraliGezinme(kok);
    printf("\n");
    printf("\nSon Sıralı Gezinti ile Ağaç: ");
    sonSiraliGezinme(kok);
    printf("\n");
    agaciBelllektenKaldir(kok);
    kok=NULL;
    return 0;
}
```

Sözlük Veri Yapısı

Bu veri yapısı, tekil olarak anahtarları (key) barındıran ve her anahtara bir değer (value) verilebilen bir yapıdır. Tanımından hareketle bu veri yapısına sözlük (dictionary) veya eşleme (map) adı verilir.

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

#define MAX_ESLEME 100 //Sözlükte en fazla 100 eşleşme olacak.

int eslesmeSayisi = 0; // Aktif sözlük Uzunluğu
char keys[MAX_ESLEME][100];
int values[MAX_ESLEME];

int indisiBul(char key[])
{
   int i;
   for (i = 0; i < eslesmeSayisi; i++) {
      if (strcmp(keys[i], key) == 0)
         return i;
   }
   return -1;
}

void sozlugeEkle(char key[], int value)
{
   int index = indisiBul(key);
   if (index == -1) {
      strcpy(keys[eslesmeSayisi], key);
      values[eslesmeSayisi] = value;
      eslesmeSayisi++;
   } else
      values[index] = value;
}

int sozlukteBul(char key[])
{
   int index = indisiBul(key);
   if (index == -1)
      return -1;
   else
      return values[index];
}

void sozluguYazdir()
{
```

```
   for (int i = 0; i<eslesmeSayisi; i++)
      printf("%s: %d\n", keys[i], values[i]);
}

int main()
{
   sozlugeEkle("Elma", 100);
   sozlugeEkle("Armut", 200);
   sozlugeEkle("Muz", 300);

   printf("Sozluk İçeriği: \n");
   sozluguYazdir();

   printf("\nElmanın Sözlükteki Değeri: %d\n",
       sozlukteBul("Elma"));
   printf("Muzun İndisi: %d\n",
       indisiBul("Muz"));
   return 0;
}
```

ÇOK DEĞİŞKENLİ FONKSİYONLAR

Değişken Sayıda Argüman Alabilen Fonksiyonlar

Değişken sayıda argüman alabilen bir fonksiyona çok değişkenli fonksiyon (variadic function) denir. C dilindeki güçlü ancak çok nadiren kullanılan özelliklerden biridir. `printf()` ve `scanf()` fonksiyonları aslında değişken sayıda argümanı biçim dizgisinden (format string) sonra koyabildiğimiz fonksiyonların en bilinen örnekleridir.

```
printf(const char* format,...);

printf("%d %c %s %c",i,c,str,c2); //5 argümanı olan printf
printf("%s %d %c",str,i,c); //4 argümanı olan printf
printf("%s",str); //2 argümanı olan printf
```

C Dilinde, değişken sayıda bağımsız değişkeni olan bir fonksiyon; en az bir sabit bağımsız değişkene sahip olacak şekilde tanımlanır ve ardından derleyicinin değişken sayıda bağımsız değişkeni ayrıştırmasını sağlayan bir üç nokta simgesi (elipsis) (...) eklenir.

```
dönüş-tipi fonksiyonkimliği(veri-tipi birinciargüman, ...);
```

Değişken argümanları işlemek için kodunuza `stdarg.h` başlık dosyasını kodumuza dahil etmeniz gerekir;

Fonksiyon	Açıklama
`va_start(va_list ap, arg)`	Bu fonksiyon, üç nokta ile verilen argümanları `va_list` değişkenine aktarır.
`va_arg(va_list ap, type)`	Her seferinde, üç nokta ile temsil edilen değişken listesindeki bir sonraki argümanı `va_list` üzerinden işler ve listenin sonuna ulaşana kadar onu `type` ile verilen veri tipine dönüştürür.
`va_copy(va_list dest, va_list src)`	`va_list`'teki argümanların bir kopyasını oluşturur.
`va_end(va_list ap)`	Bu, `va_list` değişkenlerine erişimi sonlandırır.

Tablo 27. Stdarg.h Fonksiyonları

Aşağıda ilk parametre ile belirlenmiş argüman sayısı kadar argüman alan bir fonksiyon tanımlanmıştır.

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

int argümanlarinHepsiniTopla(int kacAdet, ...) {
   va_list argumanlar;
   int sayac, toplam = 0;
   va_start(argumanlar, kacAdet);
   for (sayac = 0; sayac < kacAdet; sayac++)
      toplam += va_arg(argumanlar, int);
   va_end(argumanlar);
   return toplam;
}

int main(){
   printf("3 Argüman Toplamı = %d \n",
          argümanlarinHepsiniTopla(3, 10, 20, 30));
   printf("5 Argüman Toplamı = %d \n",
          argümanlarinHepsiniTopla(5, 10, 20, 30,40,50));
   return 0;
}
```

Ana Fonksiyonun Parametreleri

Ana fonksiyon (main function) programın icra edilmeye başladığı fonksiyondur. C Dilinde yazdığımız programlar da çalıştırılırken konsoldan argüman alabilir. Şu ana kadar argümansız ana fonksiyonu gördük. Ana fonksiyon;

- Satır içi (inline) fonksiyon olarak bildirilemez!
- Adresi alınamaz!
- Programın başka hiçbir yerinden çağrılamaz (call)!

Yazdığımız programlar da çalıştırılırken konsoldan argüman alabilir. Şu ana kadar argümansız ana fonksiyonu gördük. Argüman alan ana fonksiyon (main function) aşağıdaki iki şekilde gibi tanımlanır;

```
int main(int argc,
         char* argv[]) {
    // Ana fonksiyon Gövdesi
}
```

Ya da aşağıdaki şekilde tanımlanır;

```
int main(int argc,
         char* argv[],
         char ** envp) {
    // Ana fonksiyon Gövdesi
}
```

Aşağıda konsoldan çalıştırılırken alınan argümanları ve işletim siteminin ortam değişkenleri (environment variable) konsola yazan bir program örneği verilmiştir;

```
#include <stdio.h>
int main( int argc, // argüman sayısı
char* argv[],       // Metin dizisi olarak konsoldan girilen komut ve argümanlar
char** envp)        // Metin dizisi olarak ortam (environment) değişkenleri
{
    int sayac;

    printf_s( "\nKonsoldan Girilen Argümanlar:\n" );
    for( sayac = 0; sayac < argc; sayac++ )
        printf_s( "  argv[%d]   %s\n", sayac, argv[sayac] );

    printf_s( "\nOrtam Değişkenleri:\n" );
    while( *envp != NULL )
        printf_s( "  %s\n", *(envp++) );

    return 0;
}
```

Programın derlenmesi ve çalıştırılması sonrası çıktı aşağıda verilmiştir;

```
C:\Users\ilhan>notepad main.c
C:\Users\ilhan>gcc main.c -o main.exe
C:\Users\ilhan>main 10 20 otuz elli

Konsoldan Girilen Argümanlar:
  argv[0]   main
  argv[1]   10
  argv[2]   20
  argv[3]   otuz
  argv[4]   elli

Ortam Değişkenleri:
  ALLUSERSPROFILE=C:\ProgramData
  APPDATA=C:\Users\ilhan\AppData\Roaming
  CommonProgramFiles=C:\Program Files\Common Files
```

```
  CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
  CommonProgramW6432=C:\Program Files\Common Files
  COMPUTERNAME=DAMLANURO
  ComSpec=C:\WINDOWS\system32\cmd.exe
  configsetroot=C:\WINDOWS\ConfigSetRoot
  DriverData=C:\Windows\System32\Drivers\DriverData
  EFC_8812=1
  FPS_BROWSER_APP_PROFILE_STRING=Internet Explorer
  FPS_BROWSER_USER_PROFILE_STRING=Default
  HOMEDRIVE=C:
  HOMEPATH=\Users\ilhan
  LOCALAPPDATA=C:\Users\ilhan\AppData\Local
  LOGONSERVER=\\DAMLANURO
  NUMBER_OF_PROCESSORS=8
  OneDrive=C:\Users\ilhan\OneDrive
  OneDriveConsumer=C:\Users\ilhan\OneDrive
  OS=Windows_NT Path=C:\WINDOWS\system32;C:\Users\ilhan\Downloads\codeblocks\MinGW\bin;
  PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
  PROCESSOR_ARCHITECTURE=AMD64
  PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
  PROCESSOR_LEVEL=6
  PROCESSOR_REVISION=8e0c
  ProgramData=C:\ProgramData
  ProgramFiles=C:\Program Files
  ProgramFiles(x86)=C:\Program Files (x86)
  ProgramW6432=C:\Program Files
  PROMPT=$P$G
  PSModulePath=C:\Program
Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules
  PUBLIC=C:\Users\Public
  SESSIONNAME=Console
  SystemDrive=C:
  SystemRoot=C:\WINDOWS
  TEMP=C:\Users\ilhan\AppData\Local\Temp
  TMP=C:\Users\ilhan\AppData\Local\Temp
  USERDOMAIN=DAMLANURO
  USERDOMAIN_ROAMINGPROFILE=DAMLANURO
  USERNAME=ilhan
  USERPROFILE=C:\Users\ilhan
  windir=C:\WINDOWS
  ZES_ENABLE_SYSMAN=1

C:\Users\ilhan>
```

DOSYALAR

Dosya Nedir?

Her programının ihtiyaç duyduğu ortak görevler kullanıcıdan girdiyi (input) okumak ve konsolda çıktıyı (output) göstermektir. Bu şekilde girdi ve çıktı işlemleri yaparsak, program çalıştığı sürece veriler var olur, program sonlandırıldığında o verileri tekrar kullanamayız. Bunun için ve elektrik kesildiğinde bellekteki veriler kaybolacağından harici hafıza ortamında (external memory) verileri saklamak gerekir.

Kısaca dosya (file), verilerin bir araya geldiği (collection of data) ikincil saklama ortamıdır (seconder storage). Bu ikincil ortam; Bir bilgisayar hafızası (memory) olabileceği gibi, verilerin elektrikler kesildiğinde kaybolmayacağı sabit disk (hard disc), ya da bir başka ortama/bilgisayara veri gönderen (send) veya alan (receive) modem ve benzeri bir cihaz (device) olabilir.

Standart Dosyalar

C dili programcıya, girdi ve çıktı kısaca IO (input output) için çeşitli işlevleri içeren başlık dosyaları (header file) sağlar ve tüm aygıtları (device) dosyalar olarak ele alınır. Bu nedenle, "ekran" gibi aygıtlar "dosyalar" ile aynı şekilde ele alınır.

Standart Dosya	Dosya	Aygıt
Standart Giriş	`stdin`	Klavye
Standart Çıktı	`stdout`	Ekran
Standart Hata	`stderr`	Kullanıcı Ekranı

Tablo 28. C Dilinde Standart Giriş Çıkış Dosyaları

Dosyalarla işlem yapmak için `stdio.h` başlık dosyası çeşitli fonksiyonlara sahiptir.

Biçimlendirilmemiş Karakter Giriş Çıkış Fonksiyonları

```
int getchar(void);
```

Yeni Satır (`'\n'`) tuşuna basmadan tek bir tuş vuruşunu/karakteri okur.

```
int putchar(int c);
```

Tek bir karakter yazar. Yazdırılacak karakterin ASCII kodu parametre olarak gönderilebilir.

```
#include <stdio.h>
int main() {
   char ch;
   printf("Bir karakter Giriniz: ");
   ch = getchar();
   puts("Girdiğiniz karakter: ");
   putchar(ch);
   return 0;
}
```

Biçimlendirilmemiş Dizgi Giriş Çıkış Fonksiyonları

Okuma ve yazma amacıyla dosyaya erişmek için önceden tanımlanmış bir FILE yapısı (struct) kullanır. Standart dosyalar da bu parametreye argüman olarak verilebilir.

```
char* fgets(char* str,int size,FILE* stream);
```

`stream` dosyasından `str` ile işaret edilen tampon belleğe (`buffer`), yeni satır veya dosya sonu (End of File- EOF) ile karşılaşılıncaya kadar bir satırı okur. Hata yoksa `str` değişkeni, varsa `EOF` ya da `NULL` geri döndürür.

```
fputs(const char* str,FILE* stream);
```

`str` dizgisini ve sonuna yeni satır karakteri koyarak `stream` dosyasına yazar. Hata yoksa pozitif bir değer döner, varsa `EOF` geri döndürür.

```
#include <stdio.h>
int main() {
   char adi[20];
   printf("Adınız: ");
   fgets(adi, sizeof(adi), stdin);
   fputs("Girdiğiniz Adınız", stdout);
   fputs(adi,stdout);
   return 0;
}
```

Biçimlendirilmiş Giriş Çıkış Fonksiyonları

```
int scanf(const char* format, ...);
```

Standart giriş dosyası `stdin` standart dosyasından girişi sağlanan biçime göre okur;

```
int printf(const char format, ...);
```

Çıktıyı standart çıktı dosyası `stdout` standart çıkış dosyasına sağlanan biçime göre üreterek yazar. Şimdiye kadar bu fonksiyonların kullanımını çokça gördük.

```
int fscanf(FILE* stream, const char* format, ...);
```

FILE yapısı ile belirtilen dosyadan girişi sağlanan biçime göre okur. `stdin` standart giriş dosyası olarak kullanılabilir.

```
int fprintf(FILE* stream, const char format, ...);
```

FILE yapısı ile belirtilen dosyaya sağlanan biçime göre çıktıyı üreterek yazar. `stsout` ve `stderr` standart çıkış dosyalarıdır. Bu fonksiyonların kullanımı `scanf` ve `printf` fonksiyonları ile aynıdır. Yalnızca hangi dosyanın kullanılacağına ilişkin ek bir `stream` parametresi eklenmiştir.

Standart Olmayan Dosyalar

Standart olan `stdin`, `stdout` ve `stderr` dosyaları her zaman veri alışverişine açıktır. Bu dosyalarla metinler üzerinden veri alışverişi yapılır. Standart olmayan dosyalar ise her zaman veri alışverişine açık değildir. Bu nedenle dosya ile işlem yapmadan önce dosyayı açmalı ve işimiz bittiğinde de kapatmalıyız.

Standart Olmayan Metin Dosyaları

Standart olmayan dosyaları açıp kapatmayı sağlayan iki fonksiyon vardır;

```
FILE *fopen(const char* file_name, const char* mode_of_operation);
```

Adı `file_name` ile verilen dosyayı; okumak (read), yazmak (write) ya da sonuna eklemek (append) için ya da bunlardan her ikisini yapmak için açar. Açılan dosyanın FILE yapısına (struct) ilişkin göstericiyi geri döndürür. Burada bir metin dosyasını aşağıdaki argümanları bir dizgi (string) olarak `fopen` fonksiyonuna argüman vererek çeşitli şekillerde işlem yapmak için açabiliriz. Bu argümanlar dosya açma modları olarak adlandırılır.

Modlar	Açıklaması
`r`	Dosyayı, çalışılan klasörde/dizinde arar. Dosyayı yalnızca okumak için açar. Dosya başarıyla açılırsa `fopen()` onu belleğe yükler ve içindeki ilk karakteri işaret eden bir gösterici ayarlar. Dosya açılamıyorsa `NULL` değerini döndürür.
`w`	Dosyayı, çalışılan klasörde/dizinde arar. Dosya zaten mevcutsa içeriğinin üzerine yazılır. Dosya mevcut değilse yeni bir dosya oluşturulur. Dosya açılamıyorsa `NULL` değerini döndürür. Yalnızca yazmak için yeni bir dosya oluşturur.
`a`	Dosyayı çalışılan klasörde/dizinde arar. Dosya başarıyla açılırsa `fopen()` onu belleğe yükler ve içindeki son karakteri işaret eden bir gösterici ayarlar. Dosya mevcut değilse yeni bir dosya

Modlar	Açıklaması
	oluşturulur. Dosya açılamıyorsa `NULL` değerini döndürür. Dosya yalnızca ekleme (append) yani dosyanın sonuna yazma amacıyla açılır.
r+	Dosyayı, çalışılan klasörde/dizinde arar. Dosyayı hem okumak hem de yazmak için açar. Başarıyla açılırsa, `fopen()` onu belleğe yükler ve içindeki ilk karakteri işaret eden bir gösterici ayarlar. Dosya açılamıyorsa `NULL` değerini döndürür.
w+	Dosyayı, çalışılan klasörde/dizinde arar. Dosya mevcutsa içeriğinin üzerine yazılır. Dosya mevcut değilse yeni bir dosya oluşturulur. Dosya açılamıyorsa `NULL` değerini döndürür. w ve `w+` arasındaki fark, `w+` kullanılarak oluşturulan dosyayı da okuyabilmemizdir.
a+	Dosyayı, çalışılan klasörde/dizinde arar. Dosya başarılı bir şekilde açılırsa `fopen()` onu belleğe yükler ve içindeki son karakteri işaret eden bir gösterici ayarlar. Dosya mevcut değilse yeni bir dosya oluşturulur. Dosya açılamıyorsa `NULL` değerini döndürür. Dosya okumaya ve eklemeye (dosyanın sonuna yazma) açılır.

Tablo 29. Metin Dosyası Açma Modları

Aşağıdaki fonksiyon ise açık olan dosyayı kapatır.

```
int fclose(FILE* stream);
```

Aşağıda verilen örnekte görüldüğü gibi aynen konsola yazar gibi yazılması gerekenleri `fprintf()` fonksiyonu ile aynen dosyaya yazılmıştır. Kullanımı `printf()` ile aynıdır sadece ilk parametresi dosya göstericisidir. Dosya `w+` modunda açıldığından, program her çalıştığında içeriği silerek yeniden aynı şeyleri yazacaktır.

```
#include <stdio.h>

int main()
{
    FILE* dosyaGostericisi;
    // "metin.txt" dosyası yoksa oluşturulur.
    // Varsa üzerine yazılır. Hem razma hem okuma modunda dosya açılıyor.
    dosyaGostericisi = fopen("metin.txt", "w+");
    fprintf(dosyaGostericisi, "Adi Soyadi;Yaşı;Cinsiyeti\n");
    fprintf(dosyaGostericisi, "%s;%d;%c\n", "Ilhan OZKAN", 50,'E');
    fprintf(dosyaGostericisi, "%s;%d;%c\n", "Yagmur OZKAN", 45,'K');
    fclose(dosyaGostericisi);
    return 0;
}
```

Oluşan metin dosyası olan `metin.txt` içeriği aşağıda verilmiştir;

```
Adi Soyadi;Yaşı;Cinsiyeti
Ilhan OZKAN;50;E
Yagmur OZKAN;45;K
```

Aynı dosyası her bir satırı tam olarak okuyan `fgets()` fonksiyonu ile okuyabiliriz;

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define BIRSATIRDAKIENFAZLAKARAKTER 80

int main()
{
    char *dosyaAdi="metin.txt";
    char satir[BIRSATIRDAKIENFAZLAKARAKTER] = {0};
    unsigned int satirNumarasi = 0;
    FILE *dosyaGostericisi = fopen(dosyaAdi, "r");
    if (!dosyaGostericisi)
    {
        perror(dosyaAdi);
        return EXIT_FAILURE;
    }
```

```
    while (fgets(satir, BIRSATIRDAKIENFAZLAKARAKTER, dosyaGostericisi))
    {
        printf("satir[%06d]: %s", ++satirNumarasi, satir);
        if (satir[strlen(satir) - 1] != '\n')
            printf("\n");
    }
    if (fclose(dosyaGostericisi))
    {
        return EXIT_FAILURE;
        perror(dosyaAdi);
    }
}
/* Program Çıktısı:
satir[000001]: Adi Soyadi;Yasi;Cinsiyeti
satir[000002]: Ilhan OZKAN;50;E
satir[000003]: Yagmur OZKAN;45;K
*/
```

Bunların yanında `perror()` fonksiyonu, programınızda neyin yanlış gittiğini anlamanıza ve hata ayıklamanıza yardımcı olan standart hata akışına (stderr) açıklayıcı bir hata mesajı yazdırmak üzere tasarlanmıştır;

```
#include <stdio.h> /* perror(), fopen(), fputs() and fclose() */
#include <stdlib.h> /* EXIT_* macroları için */

int main(int argc, char **argv)
{
    int e = EXIT_SUCCESS;
    char *path = "output.txt";
    FILE *file = fopen(path, "w");
    if (!file)
    {
        perror(path);
        return EXIT_FAILURE;
    }
    if (fputs("Dosyaya Yazılacak Metin....\n", file) == EOF)
    {
        perror(path);
        e = EXIT_FAILURE;
    }
    if (fclose(file))
    {
        perror(path);
        return EXIT_FAILURE;
    }
    return e;
}
```

Ayrıca aşağıda verilen program `popen()` aracılığıyla bir program ya da servisi çalıştırır ve işlemden gelen tüm standart çıktıyı okur ve bunu standart çıktı olan konsola yansıtır;

```
#include <stdio.h>

void print_all(FILE *stream)
{
    int c;
    while ((c = getc(stream)) != EOF)
    putchar(c);
}

int main(void)
{
    FILE *stream;
```

```
    /* call netstat command. netstat is available for Windows and Linux */
    if ((stream = popen("dir", "r")) == NULL)
        return 1;
    print_all(stream);
    pclose(stream);
    return 0;
}
```

Standart Olmayan İkili Dosyalar

Değişkenlerimizi her zaman insan gözünün okuyacağı metne çevirerek dosyalara kaydetmeyiz. Onun yerine daha az yer kaplaması açısından bellekte saklandığı şekliyle dosyaya hızlıca yazarız. Ancak bu durumda dosyaya, bellekte ikili olarak tutulan verileri ikili olarak yazarız ve insan gözüyle okunamazlar.

Standart olmayan ikili dosyalar da metin dosyaları gibi açılıp kapatılır. İkili dosyaları da açmak için dosya modları farklıdır;

Modlar	Açıklaması
`rb`	Dosyayı çalışılan klasörde/dizinde arar. İkili dosyayı okuma modunda açar. Dosya mevcut değilse, `fopen()` işlevi `NULL` değerini döndürür.
`wb`	İkili dosya yazma modunda açılır. Gösterici dosyanın başlangıcına ayarlanır ve içeriklerin üzerine ikili olarak yazılır. Dosya mevcut değilse yeni bir dosya oluşturulur.
`ab`	İkili dosya, ekleme modunda açılır. Dosya göstericisi, dosyadaki son karakterden sonra ayarlanır. Verilen dosya isminden bir dosya yoksa yeni bir dosya oluşturulur.
`rb+`	İkili dosya okuma ve yazma modunda açılır. Dosya mevcut değilse, `open()` işlevi `NULL` değerini döndürür.
`wb+`	İkili dosya okuma ve yazma modunda açılır. Dosya mevcutsa içeriğin üzerine yazılır. Dosya mevcut değilse oluşturulacaktır.
`ab+`	İkili dosya okuma ve ekleme modunda açılır. Dosya mevcut değilse bir dosya oluşturulacaktır.

Tablo 30. İkili Dosya Açma Modları

İkili dosyalara veri yazmak ve okumak için kullanılan fonksiyonlar aşağıda verilmiştir;

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Bu fonksiyon, `ptr` ile işaret edilen diziden verilen `stream` dosya göstericisi üzerinden dosyadan veri okur.

- `nmemb`, `size` ile verilen uzunluktan kaç adet okunacağını belirtir.
- Genellikle ikili dosyaları okumak için kullanılır ancak metin dosyaları için de kullanılabilir.
- Bir okuma hatası veya dosya sonu (`EOF`) oluşursa `nmemb`'den az olabilen, başarıyla okunan öğelerin sayısını döndürür. `size` veya `nmemb` sıfırsa, `fread` sıfır döndürür ve `ptr` tarafından işaret edilen belleğin içerikleri değişmez.

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Bu fonksiyon, `ptr` ile işaret edilen dizideki verileri belirtilen `stream` dosya göstericisi üzerine yazar.

- `nmemb`, `size` ile verilen uzunluktan kaç adet yazılacağını belirtir.
- Genellikle ikili dosyaları okumak için kullanılır ancak metin dosyaları için de kullanılabilir.
- Bu fonksiyon başarıyla yazılan toplam öğe sayısını döndürür. Bu sayı `nmemb`'den azsa, bir hata oluşmuştur veya dosya sonuna ulaşılmıştır.

Aşağıda verilen örnekte görüldüğü üzere dosyaya yazılan yapılar aynen bellekte olduğu gibi yazılır. Dosya `wb+` modunda açıldığından, program her çalıştığında içeriği silerek yeniden aynı şeyleri yazacaktır.

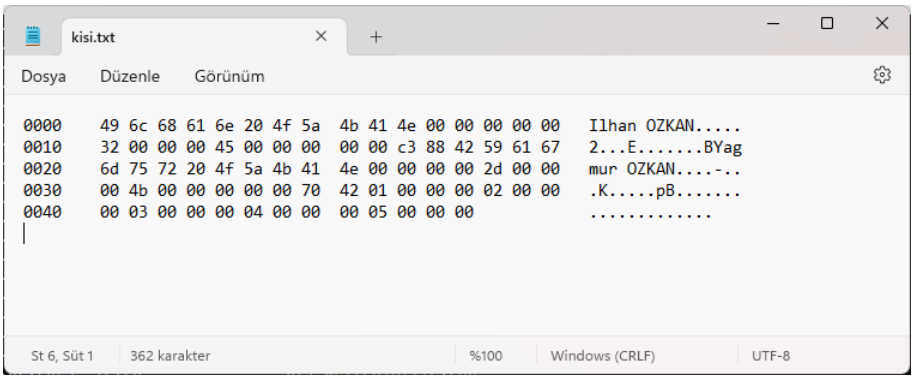
```
#include <stdio.h>

struct kisi {
    char adiSoyadi[16];
    int yas;
    char cinsiyet;
```

```
    float kilo;
};
typedef struct kisi Kisi;

int main() {
    FILE* dosyaGostericisi;
    Kisi kisi1={"Ilhan OZKAN",50,'E',100.0};
    Kisi kisi2={"Yagmur OZKAN",45,'K',60.0};
    int dizi[5]={1,2,3,4,5};
    // "kisi.bin" dosyası yoksa oluşturulur.
    // Varsa üzerine yazılır. Hem yazma hem okuma modunda dosya açılıyor.
    dosyaGostericisi = fopen("kisi.bin", "wb+");
    fwrite(&kisi1, sizeof(Kisi),1,dosyaGostericisi);
    fwrite(&kisi2, sizeof(Kisi),1,dosyaGostericisi);
    fwrite(dizi,sizeof(dizi),1,dosyaGostericisi);
    fwrite(dizi,sizeof(int),5,dosyaGostericisi);
    fclose(dosyaGostericisi);
    return 0;
}
```

Oluşan `kisi.bin` dosyasına ilk önce iki adet kişi yapısı kaydedilmiştir. Sonrasında 5 tamsayıdan oluşan dizi kaydedilmiştir. Dosya bellekte saklandığı şekliyle dosyaya kaydedildiğinden içeriği ikili olarak okunabilir. İkili dosyaları açan görüntüleyiciler ile açılıp görülebilir. Bir ikili dosyayı onaltılık rakamlara çevirip metin olarak görüntülemek için Windows ortamında aşağıdaki komut verilebilir;

```
C:\Users\ILHANOZKAN>certutil -encodehex kisi.bin kisi.txt
Input Length = 77
Output Length = 367
CertUtil: -encodehex command completed successfully.
```

Daha sonrasında dönüştürülen `kisi.txt` dosyası herhangi bir metin editörüyle açılıp okunabilir.



Şekil 24. Programın Oluşturduğu İkili Dosya İçeriği

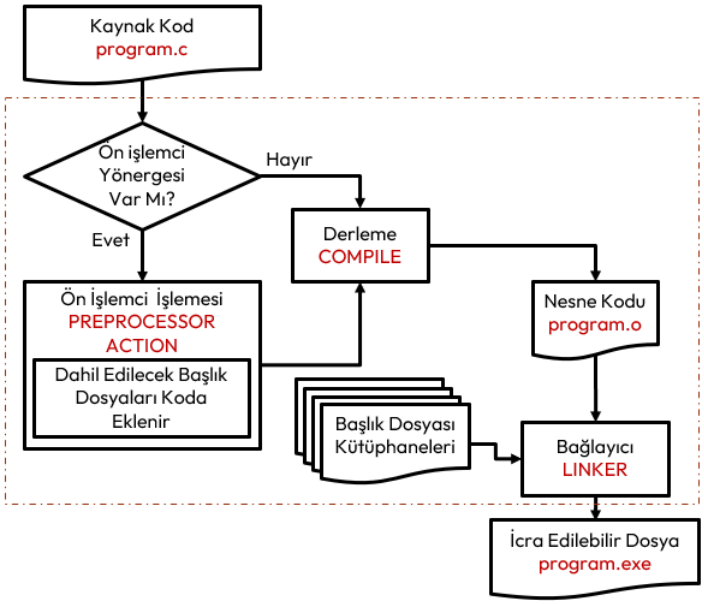
Dosyaya hep aynı veri tipi (data type) yazılacağı gibi, birbirinden farklı olabilen yapı (struct) ve veri tipleri de yazılabilir. Burada önemli olan, yazıldığı sırada bu öğelerin okunmasıdır.

ÖN İŞLEMCİ YÖNERGELERİ

Ön İşlemci Yönergesi Nasıl Çalışır

C dilinde ön işlemci (preprocessor) yönergeleri, derleyicinin bir parçası değildir, ancak derleme sürecinde ayrı bir adımdır. Ön işlemci, yalnızca bir metin değiştirme aracıdır ve gerçek derlemeden önce gerekli ön işlemeyi yapmasını sağlar. Kısaca derleme önce bul-değiştir mantığı ile kodda değişiklikler yaptırır.

- Ön işleme bir C kodunun derlenmesi öncesindeki ilk adımdır.
- Kodu belirteçlere ayırma (tokenization) adımından önce gerçekleşir.
- Ön işlemcinin önemli işlevlerinden biri de programda kullanılan kütüphane işlevlerini içeren başlık dosyalarını koda dahil etmek için kullanılmasıdır.
- Ön işlemci ayrıca değişmezleri (literal) tanımlar ve makro kullanımını sağlar.



Şekil 23. Derleme Süreci

C dilindeki ön işlemci ifadelerine, yönergeler (directive) denir. Programda ön işlemci bölümü her zaman C kodunun en üstünde görünür. Her ön işlemci ifadesi, kare (hash) `#` sembolüyle başlar. Aşağıda çok kullanılan yönergeler bulunmaktadır.

Yönerge	Açıklama
`#define`	Ön işlemci makrosunu değiştirmek ya da ya da ilk defa tanımlamak için kullanılır.
`#include`	Bir başlık dosyasını diğer ile dahil etmek için kullanılır.
`#undef`	Tanımlanmış bir makroyu tanımsız hale getirmek için kullanılır.
`#ifdef`	Bir makro tanımlı ise doğru/true değerini döndürür.
`#ifndef`	Bir makro tanımlı değilse doğru/true değerini döndürür.
`#if`	Derleme zamanında bir durumun kontrolü için kullanılır.
`#else`	`#if` Yönergesinde alternatif durumu ifade eder.
`#elif`	`#else` ve `#if` yönergelerini tek bir şekilde ifade etmek için kullanılır.
`#endif`	Şart ön işlemcilerini (`#if`, `#else`, `#elif`) bitirir.
`#error`	`stderr` standart dosyasına hata mesajını yazar.
`#pragma`	Derleyiciye özel komutlar vermek için kullanılır.

Tablo 31. Ön İşlemci Yönergeleri

Kullanıcı Tanımlı Başlık Dosyası

`#include` yönergesi ile; hazır olan başlık dosyalarını `< >` karakterleri arasında yazarak koda dahil ederiz. Hazır olmayan ve programcı tarafından hazırlanan başlık dosyalarını çift tırnak `" "` karakterleri arasında yazarak koda dahil ederiz.

`#define` yönergesi ile yeni bir makro tanımlanabileceği gibi tanımlanmış bir makro `#undef` ile ortadan kaldırabilir veya yenisini tanımlayabiliriz.

Aşağıdaki örnekte `PI` adlı bir makro tanımlanmış ve `3.1415` reel sayısını vermektedir. Daha önce `PI` adlı bir makro tanımlanmış ise derleyici hata verir.

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

#define PI 3.1415

#ifndef ENFAZLAOGRECISAYISI
   #define ENFAZLAOGRECISAYISI 100
#endif

#ifdef ENFAZLAOGRECISAYISI
   #undef ENFAZLAOGRECISAYISI
   #define ENFAZLAOGRECISAYISI 20
#endif
```

`DEBUG` hazır tanımlanmış bir makrodur ve hata ayıklama modunda kod derlenmesi halinde doğru/true değerini döndürür. Bunun için derleyiciye `-DDEBUG` argümanı verilir. `DEBUG` hazır tanımlanmış bir makro olup hata ayıklama modunda doğru/true olduğundan hata ayıklama modunda daha çok durum ve değer konsola yazılır. Bu durumda hatayı bulmak daha da kolaylaşır.

```
#ifdef DEBUG
  /*
   Hata ayıklama modunda
   derleme yapıldığında
   çalışacak kod buraya yazılır. */
#endif
```

Aşağıda kullanıcı tanımlı bir başlık dosyası örneği verilmiştir; Baslık dosyasının bir koda birden fazla dahil (include) edilmesini önlemek için `_BASLIK_H_` değişmezi, şartlı (conditional)olarak tanımlanmıştır. Bu başlık dosyası bir kod projesinde birden fazla dahil olması halinde, `_BASLIK_H_` tanımlanmaz ise, başlık içindeki değişken ve fonksiyonlar birden fazla aynı kimlikle tanımlanacağından derleme yapılamayacaktı.

```
#ifndef _BASLIK_H_
#define _BASLIK_H_

#define PI 3.1415
#define ENFAZLAOGRECISAYISI 100

float* ogrenciNotlari() {
  static float notlar[ENFAZLAOGRECISAYISI];
  return notlar;
}

float ogrenciNotlarOrtalamasi(float* pNotlar, int pOgranciSayisi) {
  int sayac;
  float ortalama=0;
  for (sayac=0;sayac<pOgranciSayisi;sayac++) {
    ortalama+=pNotlar[sayac];
    #ifdef DEBUG
        printf("%d. Ogrencide Ortalama:%f\n",sayac, ortalama);
```

```
        // Buradaki kod hata ayıklama modunda derlendiğinde çalışır.
        // Bunun için derleyiciye -DDEBUG argümanı verilir
    #endif
  }
  return ortalama/sayac;
}
#endif
```

Kendi hazırlamış olduğumuz başlık dosyasını (`baslik.h`) kodumuza dahil ettiğimizde artık başlık içindeki fonksiyon ve değişkenleri kullanabilir hale geliriz. Aynı klasörde/dizinde bulunacak şekilde bu başlık dosyasını kullanan örnek program aşağıda verilmiştir.

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

int main() {
    printf("PI=%f\n",PI);
    printf("En fazla öğrenci Sayısı=%d\n",ENFAZLAOGRECISAYISI);
    float* notlar=ogrenciNotlari();
    notlar[0]=100;
    notlar[1]=80;
    notlar[2]=60;
    float ortalama=ogrenciNotlarOrtalamasi(notlar,3);
    printf("%d öğrenci için Ortalama %f",3,ortalama);
}
```

Ön Tanımlı Ön İşlemci Makroları

DEBUG gibi her c derleyicisi için standart olarak tanımlı makrolar bulunmaktadır;

Makro	Açıklama
`__DATE__`	Mevcut tarihi "MMM DD YYYY" biçiminde dizgi (string) olarak tanımlıdır.
`__TIME__`	Mevcut saat "HH:MM:SS" biçiminde dizgi (string) olarak tanımlıdır.
`__FILE__`	Dosya adı biçiminde dizgi (string) olarak tanımlıdır.
`__LINE__`	Dosyadaki satır numarası tamsayı (int) olarak tanımlıdır.
`__STDC__`	ANSI standardında derleme yapılıyorsa 1 olarak tanımlıdır.

Tablo 32. Standart Tanımlı Önişlemci Makroları

Bu makroları kullanan örnek program aşağıda verilmiştir;

```
#include <stdio.h>

int main() {
    printf("File: %s\n", __FILE__ );
    printf("Date: %s\n", __DATE__ );
    printf("Time: %s\n", __TIME__ );
    printf("Line: %d\n", __LINE__ );
    printf("ANSI: %d\n", __STDC__ );
}
```

Parametreli Ön işlemci Makroları

Ön işlemci yönergeleriyle (preprocessor directive) tanımlanan makrolar daha derleme yapılmadan işlem görür. Dolayısıyla bu aşamada bazen parametrelerle işlem yapılması gerekebilir.

```
#define kare(x) ((x) * (x))
#define kup(x) ((x) * (x) *(x))
#define buyugu(x,y) ((x) > (y) ? (x) : (y))
```

Burada kullanılacak parametreler veri tipi tanımlanmaz. Dolayısıyla kullanılacağı yerlere buna dikkat edilerek parametreli makro tanımlanmalıdır. Eğer makro birkaç satırdan oluşacak ise ters bölü (back slash) `\` karakteri kullanılır.

```
#include <stdio.h>

#define MAKRO(num, str) { \
            printf("%d", num); \
            printf(" is"); \
            printf(" %s number", str); \
            printf("\n"); \
           }
```

Derleme Sürecinin Detayı

C dilinde derleme (compile) sürecinin iki aşamadan oluştuğu daha önce anlatılmıştı. Aşağıda tüm süreç verilmiştir;

1. Birinci aşama:
   a. Kaynak koddaki tüm açıklamalar silinir.
   b. `#define`, `#if`, `#include`, gibi tüm yönergeler ile verilen işlemler yapılır. Bunlar arasında başlık dosyalarının koda dahil edilmesi de vardır.
   c. Bu aşamaya gelen kod, `gcc -E main.c` komutu yazılarak görülebilir.
2. İkinci aşama:
   a. Kaynak kod montaj koda (assembly code) çevrilir.
   b. Bu duruma gelen kod, `gcc -S main.c` komutu dosya haline getirilebilir.
   c. Assembly kod derlenerek makine diline çevrilir ve amaç dosya (object file) oluşur.
   d. Bu duruma gelen kod, `gcc -c main.c` komutu ile dosya haline getirilebilir.
   e. Amaç dosya kodda belirtilen başlık dosyalarına uygun kütüphaneler (library) ile birbirine bağlanır ve icra edilebilir dosya (executable file) elde edilir. Burada bahsedilen kütüphaneler her işletim sistemi için ayrı olarak oluşturulur ve derleyici kurulumunda bir klasörde tutulur.
   f. Şimdiye kadar yapılan tüm derlemelerde yukarıdaki adımlar atlanarak `gcc main.c -o main.exe` komutuyla icra edilebilir dosya oluşturulmaktadır.

ŞEKİL LİSTESİ

TABLO LİSTESİ

DİZİN