

TC.

KARAMANOĞLU MEHMETBEY ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

**C PROGRAMLAMA DİLİ**

Hazırlayan

Beyza KALCI

Bilgisayar Mühendisliği

Ara Sınav Tezi

1. **İÇİNDEKİLER**

* **1. C Programlama Dili**
* **1.5.2.1 FOR**
* **1.5.2.2 WHİLE**
* **1.5.2.3 DO WHİLE**
* **1.6 Önişlemci Komutları**
* **1.7 Diziler**
* **1.8 Fonksiyonlar**
* **1.9 Pointerlar**
* **1.9.1 Pointer Ve Diziler**
* **1.10 Dinamik Bellek Yönetimi**
* **1.10.1 Malloc() Ve Free() Fonksiyonu**
* **1.10.2 Calloc() Fonksiyonu**
* **1.10.3 Realloc() Fonksiyonu**
* **1.11 String.h Kütüphanesi**
* **1.11.1 atoi(  ) ve atof(  ) ile string dönüşümü**
* **1.12 Math.h Kütüphanesi**
* **1.13 Structlar (Yapılar)**
* **1.13.1 Yapı Bildirimi**
* **1.13.2 Yapı Elemanlarına Değer Atanması**
* **1.14 Dosya İşlemleri**
* **1.14.1 Metin Akışı**
* **1.14.2 İkili Sistem Akışı**
* **1.14.3 Aktif Konum**
* **1.14.4 Dosya Sistemi**
* **1.14.5 Standart Akışlar**
* **1.14.6 Dosya Açma**
* **1.14.7 Dosya Kapatma**
* **1.14.8 fgetc() ve fputc() Fonksiyonları**
* **1.14.9 Rastgele Erişim**
* **1.15 KAYNAKÇA**

1. **C PROGRAMLAMA DİLİ**

C dilinin temelleri Martin Richards tarafından geliştirilen BCPL adlı dil ile atılmıştır. Sonra, BCPL dilinden esinlenerek, Ken Thompson, B adlı dili oluşturmuştur. B dili, C dilinin oluşturulmasında bir basamak olarak kullanılmıştır. Daha sonra ise C dili, Brian Kernighan ve Dennis Ritchie tarafından geliştirilmiştir. Yapısal bir dil olduğundan düzenli ve anlaşılır bir kod yazımına olanak sağlar.

Çok yaygın bir kullanıcı kitlesine, dolayısıyla desteğe sahiptir. C dilinde yazdığımız bir program, Assembly dilinin aksine, bütün bilgisayarlarda çalışacak şekilde hazırlanır.C dili öğrendiğimizde nesneye dayalı (object oriented) ve görsel nesneye dayalı (visual object oriented) programlama dilleri daha rahat öğrenebiliriz.

* 1. **VERİTİPLERİ**

C programlama dilinde kullanılan 5 adet veritipi vardır.

**char** : Karakter

**int** : Tamsayı

**float** : Kayan noktalı sayı

**double** : Çift duyarlıklı kayan noktalı sayı

**void** : Değersiz

ifadeler için kullanılır. Bunun dışında daha özelleştirilmiş veritiplerini ve bellekte kapladıkları alanları Program 1.1.1 de görebiliriz.

**Program 1.1.1**

#include <stdio.h>

main()

{

printf( "char : %d bayt\n", sizeof(char));

printf( "short : %d bayt\n", sizeof(short));

printf( "int : %d bayt\n", sizeof(int));

printf( "long : %d bayt\n", sizeof(long));

printf( "unsigned char : %d bayt\n", sizeof(unsigned

char));

printf( "unsigned short : %d bayt\n", sizeof(unsigned

short));

printf( "unsigned int : %d bayt\n", sizeof(unsigned

int));

printf( "unsigned long : %d bayt\n", sizeof(unsigned

long));

printf( "float : %d bayt\n", sizeof(float));

printf( "double : %d bayt\n", sizeof(double));

printf( "long double : %d bayt\n", sizeof(long double));

}

**1.2 VERİTİPİ VE DEĞİŞKEN ADI**

Değişken bildiriminde değişken-adı ifadesi yerine kullanılacak değişken adları, verebileceğimiz herhangi bir ad olabilir. Bu ad harflerden, rakamlardan ve (\_) işaretinden oluşan bir karakter dizisi olmalıdır. C küçük ve büyük harflere farklı işlem yapar. A1 ve a1 olarak verilen değişken adları birbirinden tamamen farklı olarak kabul edilir.

Aşağıdaki işlem satırlarından birincisi a1 adlı int ve ikincisi b1 adlı float bir değişken bildirimi yapar:

**int a1;**

**float b1;**

Değişken bildirimini fonksiyonların ya da kod bloklarının başında yapılırsa, bu değişkenlere **"lokal değişken"**, bütün fonksiyonların dışında ve kaynak dosyamızın başında yapılırsa bu değişkenlere **"global değişken"** adı verilir.

Ayrıca, fonksiyonlara değer aktarma işlemi için kullanılan değişkenlere **"fonksiyon parametre değişkenleri"** adı verilir.

Lokal değişkenler sadece tanımlanmış olduğu fonksiyonlar içinde geçerlidir ve sadece içinde tanımlandığı fonksiyon çalıştığı sürece geçerlidir. Bir fonksiyon programın herhangi bir yerinden çağırıldığında, çağırılan fonksiyonun içindeki lokal değişkenler oluşturulur ve fonksiyonun son işlem satırına işlem yapıldıktan sonra yok edilirler.

Global değişkenler main() fonksiyonu da dahil olmak üzere bütün fonksiyonlar içinde geçerlidir. Bu değişkenlere, main() fonksiyonu veya diğer fonksiyonların işlem satırından herhangi bir işlem yapabiliriz. Global değişkenler kaynak dosyamızın başında tanımlandığından, programın çalışması sona erene kadar varlıklarını sürdürürler.

Global bir değişkene bir ilk değer atamazsak, değişken sıfır (0) değerini alır. Lokal bir değişkene bir ilk değer atamazsak, değişken anlamsız bir değer alır.

**Program 1.2.1**

#include <stdio.h>

int gid1; /\* İlk değer atanmadığından değeri 0 olur. \*/

int gid2 = 128;

main()

{

int id1;

int id2 = 21;

id1 = 5;

printf("Lokal degisken degerleri : %d %d\n", id1, id2);

fonk1(7);

}

fonk1(int id1) /\* Fonksiyon parametre değişkeni \*/

{

/\* Değişkenler global olduğundan fonksiyon içinden erişim sağlanır. \*/

printf("Global degisken degerleri: %d %d\n", gid1, gid2);

printf("Sayinin karesi: %d", id1\*id1);

}

* 1. **OPERATÖRLER**

C programlama dilinde yazdığımız kodda matematiksel işlemler yaptırabiliriz. Bu işlemleri yaptırmak için bazı operatörlere ihtiyacımız var. Operatörler ifade içinde yer alan sabitler ve değişkenlerle işlemler yaparlar. 6 tür operatör çeşidi vardır.

**1.3.1 Aritmetik Operatörler**

**Operatör Kullanma Amacı**

\* Çarpma

/ Bölme

% Bölme işleminde kalanı verme

+ Toplama

- Çıkarma

**1.3.2 İlişkisel Operatörler**

**Operatör Anlamı**

< Küçüktür

<= Küçüktür veya eşittir

> Büyüktür

>= Büyüktür veya eşittir

== Eşittir

!= Eşit değildir

İlişkisel operatörler iki değişkenin arasına konur ve çıktı olarak bize doğruysa (1) yanlışsa (0) çıktılarını verir.

**1.3.3 Mantıksal Operatörler**

**Operatör Yapılan işlem**

! NOT

&& AND

|| OR

**1.3.4 Artırma ve Azaltma Operatörleri**

Artırma ve azaltma operatörleri değişkenin değerini bir sayı artırır veya azaltır.

**değişken\_adı++;**

**değişken\_adı--;**

şeklinde kullanılır. Artırma ve azaltma operatörleri değişken adının başında ya da sonunda kullanabiliriz.

**1.3.5 Bit Operatörleri**

**&** : Eğer, bir sırada her iki bit 1 ise, sonuç bit bölümüne 1 yazılarak sonuç değeri elde edilir.

**|** : Eğer karşılıklı gelen bit'lerden en az bir tanesi 1 ise, sonuç bit bölümüne 1, aksi takdirde 0 yazılarak sonuç değeri elde edilir.

**^** : Eğer karşılıklı gelen bit'lerden her ikisi de aynı değeri taşıyorsa, sonuç bit bölümüne 0, aksi takdirde 1 yazılarak sonuç değeri elde edilir.

**~** : Tek değere işlem yapar. int veya char bir değer içindeki bütün bit değerlerini tersine çevirir (1 ise 0, 0 ise 1 yapar).

**1.3.6 Bit Kaydırma Operatörleri**

C'de kullanılan çok ilginç uygulamalardan biride char ve int değerlerle kullanılan bit kaydırma işlemcileridir. << işareti sola bit kaydırma ve >> işareti sağa bit kaydırma işlemcisini temsil eder. Bu işlemcilerin genel yapısı aşağıda gösterildiği şekildedir:

**değişken << int\_değer**

**değişken >> int\_değer**

Sola kaydırma bit işlemcisini kullandığınızda, değişkenin ikili sayı sistemine göre yazılmış olan değerinde yer alan bitler << işlemcisinin sağında yer alan int değer kadar sola kayar. Bu durumda, en sağda boşalan bitlerin yerine 0 değeri gelir.

Sağa kaydırma bit işlemcisi kullandığınızda ise, değişkenin ikili sayı sistemine göre yazılmış olan değerinde yer alan bitler >> işlemcisinin sağında yer alan int değer kadar sağa kayar. Bu durumda en solda boşalan bitlerin yerine 0 değeri gelir.

**1.3.7 Operatör Öncelikleri**

() [] -> .

! ~ ++ -- - (veri\_türü) \* & sizeof

\* / %

+ -

<< >>

< <= > >=

== !=

<

^

|

<<

||

C programı , yaptığı bütün işlemleri yukarıdaki işlemci önceliği kuralına uygun olarak yapar.

**1.4 Temel Giriş Çıkış İşlemleri**

Bütün bunların dışında bir de ekrana yazdırma komutları var. Bunlara da bakacak olursak;

**1.4.1 Printf/Scanf Fonksiyonları**

Öncelikle kullanıcıya ne istediğimizi söylüyoruz:

printf("Bir sayı giriniz: ");

Şimdi de kullanıcıdan bir tamsayı alıyoruz:

scanf("%d",&sayi);

Son işlem olarak programın doğru çalışıp çalışmadığını kontrol etmek için aldığımız sayıyı ekrana yazıyoruz:

printf("%d alındı. teşekkürler**\n**",sayi);

**Örnek 1.4.1.1**

#include <stdio.h>

main()

{int sayi;

printf("Bir sayi giriniz: ");

scanf("%d",&sayi);

printf("%d alindi\n tesekkurler.\n",sayi);

}

\n komutu sonuna geldiği metni bir aşağı satıra indirir.

**Örnek 1.4.1.2**

#include<stdio.h>

int main( void )

{

printf("Hello World\n");

printf("Merhaba Dunya");

return 0;

}

**Örnek 1.4.1.3**

#include<stdio.h>

int main( void )

{

printf("Hello World");

printf("\nMerhaba Dunya");

return 0;

}

**Örnek 1.4.1.4**

#include<stdio.h>

int main( void )

{

printf("Hello World\nMerhaba Dunya");

return 0;

}

Yukarıdaki örneklerin hepsinde yazım tarzı farklı olmasına rağmen ekran çıktısı aynıdır.

**1.4.2 Puts/Gets Fonksiyonu**

Puts fonksiyonu ekrana yazdırılacak ifade bir karakter topluluğu ise, printf'e alternatif olarak puts fonksiyonu kullanılabilir. Ancak puts, ekrana bu karakter topluluğu yazdıktan sonra, imleci alt satıra geçirir. Klavyeden bir karakter topluluğu okumak için kullanılır. Okuma işlemi yeni satır karakteriyle(\n) karşılaşıncaya kadar sürer. Puts - gets arasındaki ilişki, printf – scanf arasındaki gibidir.

Buna göre:

**printf("Bilgisayar mühendisliği.\n"); ile puts("Bilgisayar mühendisliği.");**

**scanf("%s",str); ile gets(str);**

kullanımları eşdeğerdir.

**Örnek 1.4.1.5**

#include<stdio.h>

int main( void )

{

int a;

a = 25;

printf("a sayisi: %d",a);

return 0;

}

Yukarıdaki programı açıklamak gerekirse;

İlk satıra, int a **-**int, ingilizcede integer'ın kısaltmasıdır**-** dedik. Bunun anlamı, tamsayı tipinde, a isimli bir değişken var demektir. a=25 ise, a değişkenine 25 değerini ata anlamına geliyor. Yani, a artık 25 sayısını içinde taşımaktadır. printf(); fonksiyonunun içerisine yazılan %d ise, ekranda tamsayı bir değişken değeri gözükecek anlamındadır. Çift tırnaktan sonra koyacağımız a değeri ise, görüntülenecek değişkenin a olduğunu belirtir. Yalnız dikkat edilmesi gereken, çift tırnaktan sonra, virgül koyup sonra değişkenin adının yazılmasıdır.

**Örnek 1.4.1.6**

#include<stdio.h>

int main( void )

{ int a; int b; int toplam;

a = 25; b = 18;

toplam = a + b;

printf("a sayisi %d ve b sayisi %d, Toplami %d.\n", a, b, toplam);

return 0;}

**Örnek 1.4.1.7**

#include <stdio.h>

int main()

{ int carpan1; int carpan2; int sonuc;

printf("Ilk carpani giriniz: "); scanf("%d",&carpan1);

printf("Ikinci carpani giriniz: "); scanf("%d", &carpan2);

sonuc= carpan1 \* carpan2;

printf(" %d x %d = %d \n", carpan1, carpan2, sonuc);

return 0;

}

**Örnek 1.4.1.8**

#include<stdio.h>

int main( void )

{

int saat;

float ucret, toplam\_ucret;

char bas\_harf;

printf("Calisanin bas harfini giriniz> ");

scanf("%c",&bas\_harf);

printf("Calisma saatini giriniz> ");

scanf("%d",&saat);

printf("Saat ucretini giriniz> ");

scanf("%f",&ucret);

toplam\_ucret = saat \* ucret;

printf("%c basharfli calisanin alacagi ucret: %f\n",bas\_harf,toplam\_ucret);

return 0;

}

**Örnek 1.4.1.9**

#include<stdio.h>

int main( void )

{ float sayi1,sayi2,ortalama;

printf("Iki sayi giriniz> ");

scanf("%f%f",&sayi1,&sayi2);

ortalama = ( sayi1 + sayi2 ) / 2;

printf("Ortalama sonucu: %f'dir",ortalama);

return 0;

}

Yazılım mühendislerinin en büyük sıkıntısı yazılmış bir kodu okuyup anlamaktır. -hele ki büyük bir projeden söz ediyorsak-. Bundan korunmak için kodlara açıklama koyulabilir.

C programlama dilinde iki şekilde açıklama koymak mümkündür. Bunlardan bir tanesi, satır bazında yapılır. Diğeriyse, belirli bir bloğu yorumlamaya yarar. Compiler her iki şekilde de, açıklama olarak belirlemiş yerleri işlemeyecektir.

**Örnek 1.4.1.10**

/\*Çok satırlı bir açıklama.Yıldızlar arasında kalan bütün alan yorum olarak değerlendirilir

ve derleyici (compiler) tarafından işlenmez.\*/

#include<stdio.h>

int main( void )

{

//Tek satırlık bir açıklama.

printf("Hello World\n");

}

* 1. **Program Denetim Ve Döngü Yapıları**
     1. **Koşul Fonksiyonları**

**1.5.1.1 İF**

Belirli bir koşul olması durumunda eğer anlamına gelen "if" kalıbı kullanılabilir. Koşul if'den sonra parantez içine yazılır. Yapılması istenen komut ise ayraç içine yazılır.

**İf** (koşul)

{

koşulun sağlanması durumunda yapılacak komut

}

Eğer yapılması istenen komut tek satır ise ayraç kullanılmayabilir.

**İf** (koşul) tek komut;

Belirli bir koşul sunulduktan sonra eğer gerekli şartlar sağlanmıyor ise else komutu ile o koşul sağlanmıyorsa yapılacak işlem yazılır.

İnt koşul ;

{

**if** kalıbından sonra koşulun doğru olmaması durumunda kullanılabilir:

}

**else**

{

**if** (koşul)

}

koşulun sağlanması durumunda yapılacak komut

{

**else**

}

koşulun sağlanmaması durumunda yapılacak komut

}

}

}

### **Örnek 1.5.1.1.1**

#include<stdio.h>

int main( void )

{

int girilen\_sayi;

printf("Lutfen bir tam sayi giriniz> ");

scanf("%d",&girilen\_sayi);

if( girilen\_sayi > 100 )

printf("Girilen sayi 100'den buyuktur\n");

return 0;

}

**Örnek 1.5.1.1.2**

#include<stdio.h>

int main( void )

{

int girilen\_sayi;

printf("Lutfen bir tam sayi giriniz> ");

scanf("%d",&girilen\_sayi);

if( girilen\_sayi < 100 )

printf("Girilen sayi 100'den kucuktur.\n");

return 0;

}

### **1.5.1.2 ELSE İF**

else ve if komutları birden fazla koşul yapmak için beraber kullanılabilir.

**if**(birinci koşul)

{

birinci koşulun sağlanması durumunda yapılacak komut

}

**else** **if**(ikinci koşul)

{

1. koşulun sağlanmaması, 2. koşulun sağlanması durumunda yapılacak komut

}

**else**

{

her iki koşulun sağlanmaması durumunda yapılacak komut:

}

**Örnek 1.5.1.2.1**

#include <stdio.h>

int main()

{

int sayi;

printf("Bir sayi girin: ");

scanf("%d",&sayi);

if (sayi % 2 == 0)

printf("sayi cifttir.\n");

else

printf("sayi tektir.\n");

return 0;

}

**Örnek 1.5.1.2.2**

#include <stdio.h>

int main(void)

{

int id1;

printf("Bir int deger giriniz: ");

scanf("%d", &id1);

if(id1 > 0 && id1 <= 100) printf("Girdiginiz sayi 1-100 arasindadir!\n");

if(id1 > 0 && id1 <= 100) printf("Girdiginiz sayi 1-100 arasindadir!\n");

else printf("Girdiginiz sayi 100'den buyuktur!\n");

if(id1 > 0 && id1 <= 100) printf("Girdiginiz sayi 1-100 arasindadir!\n");

else if(id1 > 100 && id1 <= 200) printf("Girdiginiz sayi 100-200 arasindadir!");

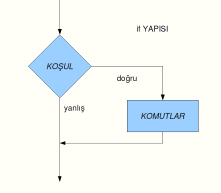
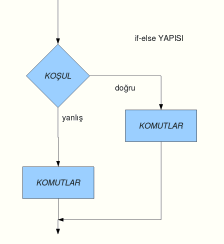
else if(id1 > 200 && id1 <= 300) printf("Girdiginiz sayi 200-300 arasindadir!");

else printf("Girdiginiz sayi 300'den buyuktur!");

return 0;

}

**İf Ve Else İf Arasındaki Fark**

**[](http://www.cagataycebi.com/programming/c_programming/c_operands/if.png) [](http://www.cagataycebi.com/programming/c_programming/c_operands/if_else.png)**

**1.5.1.3 Switch Case**

Birçok koşulun olması durumunda if/else yerine switch/case kalıbı kullanılabilir. Switch döngüsünden çıkmak için her seçeneğin sonuna break fonksiyonu konulur.

**switch** (seçenek){

**case** 1. seçenek:

seçeneğin 1. seçeneğe eşit olması durumunda işlenecek kod;

**break**();

**case** 2. seçenek:

seçeneğin 2. seçeneğe eşit olması durumunda işlenecek kod;

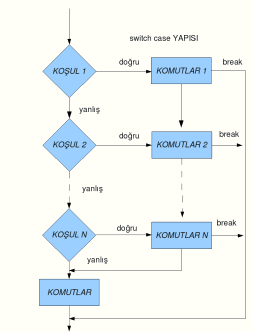
**break**();

defualt:

.......

**break**();

...}

**[](http://www.cagataycebi.com/programming/c_programming/c_operands/switch_case.png)**

**Örnek 1.5.1.3.1**

#include<stdio.h>

int main( void )

{ int not;

printf("Lütfen notu giriniz> ");

scanf("%d",&not);

switch( not / 10 ) {

case 10:

case 9: printf("NOT: A\n"); break;

case 8: printf("NOT: B\n"); break;

case 7: printf("NOT: C\n"); break;

case 6: printf("NOT: D\n"); break;

case 5:

case 4:

case 3:

case 2:

case 1:

case 0: printf("NOT: F\n"); break;

default:

printf("HATA: Bilinmeyen bir değer girdiniz!\n");}

return 0;}

**Örnek 1.5.1.3.2**

#include <stdio.h>

#include <conio.h>

int main(void)

{ int id; char cd;

printf("Yandaki sayilardan birini giriniz: [1] [2] [3]"); scanf("%d", &id);

switch (id) {

case 1 : printf("Yandaki harflerden birini giriniz: [A] [B]\n");

cd = toupper (getch());

switch (cd) {

case 'A': fonk1(); break;

case 'B': fonk2(); break;

default :

printf("Belirtilen harflerden birini girmediniz!\n");} break;

case 2 : printf("2 numarali secenegi sectiniz!\n"); break;

case 3 : printf("3 numarali secenegi sectiniz!\n"); break;

default :

printf("Verilen seceneklerden birini girmediniz!\n");} return 0;}

fonk1()

{printf("1 numarali secenek icinde A harfini girdiniz!\n");}

fonk2()

{printf("1 numarali secenek icinde B harfini girdiniz!\n");}

**Örnek 1.5.1.3.3**

#include <conio.h>

int main(void)

{ char cd;

int id;

printf("Ekrana yazdirmak istediginiz harfi girin: [A] [B] [C]\n");

cd = toupper(getch());

printf("Ekrana kac kez yazdirmak istersiniz?: [1] [3] [5]");

scanf("%d", &id);

switch(id) {

case 1:

printf("%c", cd);

break;

case 3:

printf("%c %c %c", cd, cd, cd);

break;

case 5:

printf("%c %c %c %c %c", cd, cd, cd, cd, cd);

break;

}

return 0;

}

**Örnek 1.5.1.3.4**

#include <stdio.h>

int main(void)

{ int id;

printf("Bir sayi giriniz [1-5]");

scanf("%d", &id);

printf("\n");

switch (id) {

case 1 :

printf("Bir ");

case 2 :

printf("İki ");

case 3 :

printf("Uc ");

case 4 :

printf("Dort ");

case 5 :

printf("Bes ");

}

return 0;

}

**Örnek 1.5.1.3.5**

#include <stdio.h>

int main(void)

{ int id;

printf("Bir sayi giriniz [1-5]");

scanf("%d", &id);

printf("\n");

switch (id) {

case 1 :

printf("Bir "); break;

case 2 :

printf("İki "); break;

case 3 :

printf("Uc "); break;

case 4 :

printf("Dort "); break;

case 5 :

printf("Bes "); break;

}

return 0;

}

* + 1. **Döngü Fonksiyonları**

**1.5.2.1 FOR**

Koşul sağlandığı sürece yapılması istenen komutlar için for kalıbı kullanılabilir. Belirlenen bir başlangıç değeri için değişim uygulanır ve koşul sağlandığı müddetçe kod işlenir.

**For** (başlangıç değeri; koşul; değişim)

{

yapılması istenen kod.}

**Örnek 1.5.2.1.1**

#include<stdio.h>

int main( void )

{int i;

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

printf("%2d: Merhaba Dunya\n",(i+1));}

return 0;}

**Örnek 1.5.2.1.2**

#include<stdio.h>

int main( void )

{int i; i = 0;

for( ; i < 10; ) {

printf("%2d: Merhaba Dunya\n",(i+1));

i = i + 1;}

return 0;}

**Örnek 1.5.2.1.3**

/\*Sadece tek sayıları yazdıran bir program\*/

#include<stdio.h>

int main( void )

{ int i;

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

//i değişkeninin 2'ye göre modu 0 sonucunu veriyorsa, bu onun

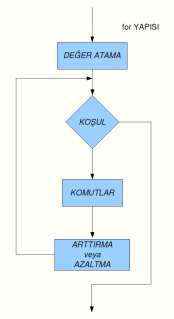
//bir çift sayı olduğunu gösterir. Bu durumda ekrana yazdırılmaması

//için döngü bir sonraki adıma geçer.

if( i%2 == 0 ) continue;

printf("%2d\n",i);}

return 0;}

**[](http://www.cagataycebi.com/programming/c_programming/c_operands/for.png)**

**1.5.2.2 WHİLE**

Belirli bir koşul sağlandığı sürece işlenmesi istenen kodlar için while döngüsü kullanılır. While koşulu ile sonsuz döngüler yaratılabilir. Koşul yerine "1" yazılırsa, mantıksal doğru anlamına geldiği için program sürekli devam eder.

**while**(koşul)

{

yapılması istenen kod;}

**Örnek 1.5.2.2.1**

/\*0 ile 99 arasında tesadüfi sayılar üreten bir programın, kaçıncı seferde 61 sayısını bulacağını yazan program aşağıdadır.\*/

#include<stdio.h>

int main( void )

{ int i,tesadufi\_sayi;

int deneme\_sayisi = 0;

//while içinde 1 olduğundan sonsuza kadar döngü çalışır.

while( 1 ){

//tesadufi\_sayi değişkenine, 0 ile 99 arasında her seferinde farklı bir sayı atanır. Rand( ) //fonksiyonu tesadüfi sayı atamaya yarar. Mod 100 işlemiyse, atanacak sayının 0 ile 99 //arasında olmasını garantiler.

tesadufi\_sayi = rand() % 100;

//Döngünün kaç defa çalıştığını deneme\_sayisi değişkeniyle buluruz.

deneme\_sayisi++;

//Eğer tesadufi sayı 61'e eşit olursa, döngü kırılıp, sonlandırılır.

if( tesadufi\_sayi == 61 ) break;}

printf("Toplam deneme sayisi: %d\n",deneme\_sayisi);

return 0;}

**Örnek 1.5.2.2.2**

#include <stdio.h>

int main(void)

{ int id=1;

while (id<=10) {

printf("%d ", id);

id++; }

return 0;}

**Örnek 1.5.2.2.3**

#include <stdio.h>

int main(void)

{ int id=1;

char cd;

while (id<=5) {

cd = 'A';

while (cd<='K') {

printf("%c ", cd++); }

printf("\n");

id++; }

return 0;

}

**Örnek 1.5.2.2.4**

#include <stdio.h>

#include <conio.h>

int main(void)

{ char cd;

cd = 'a';

while (cd != 'c') {

printf("Bir harf giriniz: ");

cd = getch();

printf("%c\n", cd); }

printf("c harfini girdiniz ve donguden ciktiniz!");

return 0;}

**Örnek 1.5.2.2.5**

#include <stdio.h>

int main(void)

{ int id=1;

while (id<=100) {

printf("Bir int deger giriniz: ");

scanf("%d", &id); }

printf("Girilen sayi 100'den buyuk oldugundan donguden ciktiniz!");

return 0;}

**1.5.2.3 DO WHİLE**

C'de kullanılan döngülerden sonuncusu do while döngüsüdür. Döngünün genel yapısı aşağıda gösterilmiştir.

do {işlem-satırı;

.

.

işlem-satırı;

} while (ifade);

**Örnek 1.5.2.3.1**

#include <stdio.h>

int main(void)

{

int id=1;

do

{

printf("%d ", id++); }

while (id<=10);

return 0;

}

**Örnek 1.5.2.3.2**

#include <stdio.h>

#include <conio.h>

int main(void)

{char cd; cd = 'a';

do { printf("Bir harf giriniz: ");

cd = getch();

printf("%c\n", cd); }

while (cd != 'c');

printf("c harfini girdiniz ve donguden ciktiniz!");

return 0;}

**Örnek 1.5.2.3.3**

#include <stdio.h>

#include <conio.h>

int main(void)

{ char cd;

printf("Bir karakter giriniz:\n");

do {cd = getch();

printf("Girdiginiz karakter: %c ASCII Kodu: %d\n", cd, cd);}

while (cd!='\r');

return 0;}

**Örnek 1.5.2.3.4**

#include <stdio.h>

int main(void)

{int id1, id2, id3=3;

printf("do while dongusu ile yazdirma:\n\n");

id2 = 0;

do { id1=1;

do {

printf("%d ", id1++); }

while (id1<=10); printf("\n"); }

while (++id2<id3); printf("\nwhile dongusu ile yazdirma:\n\n");

id2 = 0;

while (id2++<id3) {

id1=1;

while (id1<=10) {printf("%d ", id1++); }

printf("\n"); }

printf("\nfor dongusu ile yazdirma:\n\n");

for (id2=0; id2<id3; id2++) {

for (id1=1; id1<=10; id1++) { printf("%d ", id1); }

printf("\n");}

return 0;

}

**Örnek 1.5.2.3.5**

#include <stdio.h>

#include <conio.h>

int main(void)

{ int id1, id2, id3;

char cd;

printf("Ekrana sayi yazdirma!\n"); printf("[A] 1-50 [B] 1-100 [C] 1-150 [D] 1-200\n");

do {printf("Yapacaginiz islemi gosteren harfi giriniz: ");

cd = toupper(getche());

printf("\n");} while (cd!='A' && cd!='B' && cd!='C' && cd!='D');

do {printf("Kac sayi aralikla yazilmasini tercih edersiniz? [1-10]: "); scanf("%d", &id1);}

while (id1>10);

switch (cd) {

case 'A': id2 = 50; break;

case 'B': id2 = 100; break;

case 'C': id2 = 150; break;

case 'D': id2 = 200; break; }

id3 = 1;

do {printf("%d ", id3);

id3 = id3+id1; }

while (id3<=id2);

return 0;}

* 1. **Önişlemci Komutları**

### **#include Önişlemci Komutu**

#include oldukça tanıdık bir operatördür. Her programda, #include önişlemci komutunu kullanılır. Şayet kullanılmazsa, printf(  ) veya scanf(  ) gibi fonksiyonları tekrar tekrar yazmak gerekirdi. #include komutu, programda bir başlık dosyasının (header file) dâhil edileceğini belirtir. Bu başlık dosyası, standart giriş çıkış işlemlerini içeren bir kütüphane olabileceği gibi, kendimize ait fonksiyonların bulunduğu bir dosya da olabilir.

Eğer sistem kütüphanelerine ait bir başlık dosyasını programa dâhil edilmek istenirse, küçüktür ( < ) ve büyüktür ( > ) işaretlerini kullanılması gerekir. Örneğin stdio.h sisteme ait bir kütüphane dosyasıdır. Dolayısıyla stdio.h kütüphanesini programımız eklerken, #include<stdio.h> şeklinde yazarız.

* 1. **Diziler**

Diziler aynı veri çeşidinden olan değişkenlerin aynı isim altında tanımlanması olarak ifade edilebilir. Diziler kullanılarak, aynı isimle birden fazla değişkene erişilebilir ve işlem yapılabilir.

Diziler tek ya da çok boyutlu olarak tanımlanabilir ve dizi boyutu ihtiyaca göre ayarlanabilir.

int, char, float ve double olarak tanımlanabilen dizilerde sadece aynı veri türünden değişken değerleri yer alabilir.

Tek ve çok boyutlu dizilerin bildirimi için kullanılan genel yapı aşağıda gösterilmektedir;

**veri-çeşidi** **dizi-adı** [boyut]; /\* Tek boyutlu dizi bildirimi \*/

**veri-çeşidi** **dizi-adı** [boyut1][boyut2]...[boyutn]; /\* Çok boyutlu dizi bildirimi \*/

/\* Tek boyutlu eleman sayısı tanımsız dizi bildirimi

(Atanan eleman sayısı dizi boyut kapasitesini belirler) \*/

**veri-çeşidi** **dizi-adı** [] = { ele1, ele2, ..., elen };

int idizi[10]; /\* Tek boyutlu 10 elemanlı int dizi bildirimi \*/

char cdizi[10]; /\* Tek boyutlu 10 elemanlı char dizi bildirimi \*/

int idizi[3][4]; /\* İki boyutlu 12 elemanlı int dizi bildirimi \*/

char cdizi[3][21]; /\* İki boyutlu char dizi bildirimi \*/

/\* Eleman sayısı tanımsız 5 elemanlı int dizi bildirimi \*/

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

/\* Eleman sayısı tanımsız 6 elemanlı char dizi bildirimi (2 farklı yöntem) \*/

char cdizi[] = { 'R', 'a', 'k', 'a', 'm', '\0' };

char cdizi[] = "Rakam";

/\* Eleman sayısı tanımsız çok boyutlu char dizi bildirimi \*/

char cdizi[][20] = { "Çoklu", "Karakter", "Dizileri" };

**C'de bir dizinin ilk elemanına 0 değeri ile erişim sağlanır.**

int idizi[10]; /\* 10 elemanlı dizi adlı bir dizi oluşturur. \*/

dizi[0] = 38; /\* Dizinin ilk elemanına 38 değerini atar. \*/

dizi[7] = 174; /\* Dizinin 8 nci elemanına 174 değerini atar. \*/

**Örnek 1.7.1**

#include <stdio.h>

main()

{ char tamad[10]; // Bir karakter dizisi tanimlayalim //

tamad[0] = 'B';

tamad[1] = 'E';

tamad[2] = 'Y';

tamad[3] = 'Z';

tamad[4] = 'A';

tamad[5] = 'K';

tamad[6] = 'A';

tamad[7] = 'L';

tamad[8] = 'C';

tamad[9] = 'I'; // Bos karakter - stringin sonu //

printf("tamadi %s dir. \n",tamad);

printf("Icinden bir karakter: %c\n",tamad[2]);

printf("Tamadinin bir parcasi: %s \n",&tamad[3]);

}

**Örnek 1.7.2**

#include<stdio.h>

int main( void )

{// Aylari temsil etmesi icin aylar adinda 12 elemanli bir dizi olusturuyoruz.

int aylar[ 12 ];

int toplam = 0;

int i;

// Birinci dongu, deger atamak icindir

for( i = 0; i < 12; i++ ) {

printf( "%2d.Ay: ", (i+1) );

// aylara deger atıyoruz:

scanf( "%d", &aylar[ i ] );}

// Girilen degerleri gostermek icin ikinci bir dongu kurduk

printf( "\nGIRDIGINIZ DEGERLER\n\n" );

for( i = 0; i < 12; i++ ) {

printf( "%2d.Ay icin %d girdiniz\n", (i+1), aylar[i] );

toplam += aylar[ i ];}

printf( "Toplam gunesli gun sayisi: %d\n", toplam );

return 0;

}

**Örnek 1.7.3**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

int dizi[5] = {4, 12, 9, 152, 124};

int min = dizi[0];

int max = dizi[0];

int i;

printf("Dizi Min Max\n");

for(i = 1; i < 5; i++)

{

if(dizi[i] > max)

max = dizi[i];

if(dizi[i] < min)

min = dizi[i];

printf("%d - %4d - %4d\n", dizi[i], min, max);

}

printf("Minimun = %d\nMaksimum = %d", min, max);

getch();

return 0;

}

**Örnek 1.7.4**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

int dizi[5] = {5, 10, 2, 3, 2};

int min = dizi[0];

int max = dizi[0];

int sayi;

int i;

printf("Sayi giriniz: ");

scanf("%d", &sayi);

for(i = 0; i < 5; i++)

if(sayi == dizi[i])

printf("Sayi = %d dizi[] = %d, i = %d. index", sayi, dizi[i], i + 1);

getch();

return 0;

}

**Örnek 1.7.5**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

int dizi[5] = {2, 5, 8, 9, 5};

int sayi;

int gecici;

int i;

for(i = 0; i < 5 / 2; i++)

{

gecici = dizi[i];

dizi[i] = dizi[4 - i];

dizi[4 - i] = gecici;

}

for(i = 0; i < 5; i++)

printf("%d ", dizi[i]);

getch();

return 0;

}

**Örnek 1.7.6**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

/\*

int dizi[4][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

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

\*/

int dizi[4][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};

int i;

int j;

for(i = 0; i < 4; i++)

{

for(j = 0; j < 3; j++)

printf("%d ", dizi[i][j]);

printf("\n");

}

getch();

return 0;

}

* 1. **Fonksiyonlar**

C'de fonksiyon bir veya daha fazla işlem satırından oluşan kodların bir kod bloğu şeklinde yapılandırılması ile oluşturulur. Fonksiyonlar oluşturulduktan sonra programın herhangi bir yerinden sadece fonksiyon adı kullanılarak çağrılabilir. Bu sayede, çok fazla sayıda işlem satırı tek bir isim kullanılarak çalıştırılmış olur.

**veri-türü** fonksiyon-adı (**parametreler**)

{

işlem satırı

.

.

işlem satırı

return **ifade**;

}

Fonksiyon yapısında 5 temel eleman vardır. Bu elemanlardan fonksiyon-adı ve işlem satırı mutlaka bulunmalıdır. Ancak, veri-türü, parametreler ve return ifadelerinin tanımlanması programcının ihtiyaçları doğrultusunda isteğe bağlıdır.

**veri-türü** : Fonksiyonun geri verdiği veri türünü gösterir.

**fonksiyon-adı :** Fonksiyon adını gösterir.

**parametreler** : Fonksiyona geçirilen verileri gösterir. Sayısı birden fazla olabilir.

**işlem satırı :** Fonksiyon içindeki işlem satırlarını gösterir.

**return :** Verileri geri döndürmeye yarar. Son satırda kullanılması şart değildir.

**ifade** : Değişken, sabit ve işlemciler kullanılarak oluşturulan veridir. Elde edilen veri türü fonksiyonun geri döndürdüğü veri türü ile aynı olmalıdır.

**Örnek 1.8.1**

#include<stdio.h>

int sonuc = 0;

// Verilen sayinin karesini hesaplayip,

// global 'sonuc' degiskenine yazar.

void kare\_hesapla( int sayi )

{

sonuc = sayi \* sayi;

}

int main( void )

{

// main( ) fonksiyonunda

// a degiskeni tanimliyoruz.

int a;

printf( "Sayi giriniz> ");

scanf( "%d",&a );

printf( "Girdiginiz sayi\t: %d\n", a );

kare\_hesapla( a );

printf("Sayinin karesi\t: %d\n", sonuc );

return 0;

}

**Örnek 1.8.2**

#include <stdio.h>

void fonk1 (int id1);

void main (void)

{ int id1;

id1 = 34;

fonk1 (id1);

printf("main() fonksiyonu icinde id1 degisken degeri: %d", id1);}

void fonk1 (int id1)

{id1 = id1 + 27; printf("Fonksiyon icinde id1 degisken degeri: %d\n", id1);}

**Örnek 1.8.3**

#include<stdio.h>

void elemanlari\_goster( int [ 5 ] );

int main( void )

{ int dizi[ 5 ] = { 55, 414, 7, 210, 15 };

elemanlari\_goster( dizi );

return 0;}

void elemanlari\_goster( int gosterilecek\_dizi[ 5 ] )

{ int i;

for( i = 0; i < 5; i++)

printf( "%d\n", gosterilecek\_dizi[ i ] );}

**Örnek 1.8.4**

/\* Sayının tek veya çift olmasını kontrol eder. \*/

#include<stdio.h>

void tek\_mi\_cift\_mi( int sayi )

{if( sayi%2 == 0 ) printf( "%d, cift bir sayidir.\n", sayi );

else printf( "%d, tek bir sayidir.\n", sayi );}

int main( void )

{int girilen\_sayi; printf( "Lutfen bir sayi giriniz> " ); scanf( "%d",&girilen\_sayi );

tek\_mi\_cift\_mi( girilen\_sayi );

return 0;}

**Örnek 1.8.5**

/\* iki sayıyı toplar ve sonucu ekranda gösterir \*/

#include <stdio.h>

int topla( int x, int y ){

return (x+y);}

int main(void)

{ int toplam,a,b;

printf("Iki sayi girin : "); scanf("%d %d",&a,&b);

toplam = topla(a,b);

printf("%d ve %d nin toplami %d dir.\n", a,b,toplam);

return 0;}

**Örnek 1.8.6**

#include<stdio.h>

// Verilen sayinin asal olup olmadigina bakar. Sayi asalsa, geriye 1 aksi hâlde 0 degeri doner.

int sayi\_asal\_mi( int sayi )

{ int i;

for( i = 2; i <= sayi/2; i++ ) {

// Sayi asal degilse, i'ye tam olarak bolunur.

if( sayi%i == 0 ) return 0;}

// Verilen sayi simdiye kadar hicbir sayiya bolunmediyse, asaldir ve geriye 1 doner.

return 1;}

// main fonksiyonu

int main( void )

{ int girilen\_sayi;

int test\_sonucu;

do{ printf( "Lutfen bir sayi giriniz> " );

scanf( "%d",&girilen\_sayi );

test\_sonucu = sayi\_asal\_mi( girilen\_sayi );

if( !test\_sonucu )

printf("Girilen sayi asal degildir!\n");}

while( !test\_sonucu );

printf( "Girilen sayi asaldir!\n" );

return 0;}

**1.9 Pointerlar**

Pointerlar, bilgisayar belleğindeki belli bir adresi gösteren ve gösterdiği bellek adresine erişimi sağlayan değişkenlerdir.

Pointerların diğer değişkenlerden tek farkı başka bir değişkenin bellek adresini içeriyor olmasıdır. Aşağıdaki satır pointer bildiriminin genel yapısını göstermektedir;

**veri-türü** **\*değişken-adı**;

Yukarıdaki satırda yer alan **veri-türü** ifadesi pointerın adresini gösterdiği değerin veri türünü, **\*değişken-adı** ifadesi ise pointerın adını göstermektedir. \* işareti pointer değişken bildiriminde kullanılmaktadır. Aşağıdaki işlem satırı ip adlı int bir pointer değişkeni oluşturmaktadır;

int \*ip;

Pointer tanımlamalarında veri türünü belirlerken, daha sonra adresini pointera atanacak değişkenin veri türünü dikkate almamız gerekir. Başka bir ifade ile, pointer ile pointera bellek adresi atanan değişkenin veri türü aynı olmalıdır. Bir pointerı kullanmadan önce, bir bellek adresinin mutlaka pointera atanması gerekir. Çünkü, pointer bellek adresleri ile işlem yapar. C dili pointera bellek adresi atama işlemini, **&** işlemcisini kullanarak yapar.

**\*** : Hangi işaretçi değişken adından önce kullanılırsa, o işaretçi değişkene adresi atanan değişkenin değerini verir.

**&** : Hangi değişken adından önce kullanılırsa, o değişkenin adresini verir.

Aşağıda yer alan işlem satırlarından 1 ve 2 sayıları ile gösterilen satırlar, C dilinin kurallarına göre geçersizdir:

int \*ip;

\*ip = 357; /\* 1 \*/

printf("%d", \*ip); /\* 2 \*/

**Örnek 1.9.1**

#include<stdio.h>

int main( void )

{

// int tipinde değişken tanımlıyoruz:

int xyz = 10, k;

// int tipinde pointertanımlıyoruz:

int \*p;

// xyz değişkeninin adresini pointer'a atıyoruz. Bir değişken adresini '&' işaretiyle alırız.

p = &xyz;

// k değişkenine xyz'nin değeri atanır. Pointer'lar değer tutmaz.

// değer tutan değişkenleri işaret eder. Başına '\*' koyulduğunda,

// işaret ettiği değişkenin değerini gösterir.

k = \*p;

return 0;

}

Programda ekrana printf() fonksiyonu ile herhangi bir şey yazdırmadığımız için herhangi bir ekran çıktısı olmaz.

**Örnek 1.9.2**

#include<stdio.h>

int main( void )

{ int x, y, z;

int \*int\_addr;

x = 41;

y = 12;

// int\_addr x degiskenini isaret ediyor.

int\_addr = &x;

// int\_addr'in isaret ettigi degiskenin sakladigi deger aliniyor. (yani x'in degeri)

z = \*int\_addr;

printf( "z: %d\n", z );

// int\_addr, artik y degiskenini

// isaret ediyor.

int\_addr = &y;

// int\_addr'in isaret ettigi

// degiskenin sakladigi deger

// aliniyor. (yani y'nin degeri)

z = \*int\_addr;

printf( "z: %d\n" ,z );

return 0;

}

**Örnek 1.9.3**

#include<stdio.h>

int main( void )

{ int x, y; int \*int\_addr;

x = 41; y = 12;

// int\_addr x degiskenini isaret ediyor

int\_addr = &x;

// int\_addr'in isaret ettigi degiskenin degerini degistiriyoruz

\*int\_addr = 479; printf( "x: %d y: %d\n", x, y );

int\_addr = &y;

return 0;}

**Örnek 1.9.4**

#include<stdio.h>

int bolme\_islemi( int bolunen, int bolen, int \*kalan )

{\*kalan = bolunen % bolen;

return bolunen / bolen;}

int main( void )

{ int bolunen, bolen;

int bolum, kalan;

bolunen = 13; bolen = 4;

bolum = bolme\_islemi( bolunen, bolen, &kalan );

printf( "Bolum: %d Kalan: %d\n", bolum, kalan ); return 0;}

**Örnek 1.9.5**

#include <stdio.h>

int main()

{

int \*ptam, tam = 33;

ptam = &tam; /\* ptam -> tam \*/

printf("&tam = %p\n",&tam);

printf("ptam = %p\n",ptam);

printf("\n");

printf("tam = %d\n",tam);

printf("\*ptam = %d\n",\*ptam);

printf("\n");

\*ptam = 44; /\* tam = 44 anlamında \*/

printf("tam = %d\n",tam);

printf("\*ptam = %d\n",\*ptam);

return 0;

}

• \*ptam ve tam, tam adlı değişkenin içeriği ile ilgilidir.

• ptam ve &tam, tam adlı değişkenin adresi ile ilgilidir.

• \* yönlendirme ve & adres operatörüdür.

**1.9.1 Pointer Ve Diziler**

Pointerlar dizileri de işaret edebilirler. Örneğin, *int dizi[6];* şeklinde tanımlanmış bir diziyi, pointer ile işaret etmek istersek, *ptr = dizi;* yazmamız yeterlidir. Değişkenlerde, değişken adının başına '&' işareti getirilmesi gerekir, fakat dizilerde buna gerek yoktur. Çünkü dizilerin kendisi de bir pointer'dır. Dizilerin hepsi hafıza alanında bir başlangıç noktası işaret eder. Örnek olması açısından bu noktaya 6066 dersek siz "*dizi[0]*" dediğiniz zaman 6066 ile 6068 arasında kalan bölgeyi kullanırsınız. Ya da "*dizi[4]*" dediğiniz zaman 6074 ile 6076 hafıza bölgesi işleme alınır.

Bir diziyi işaret eden pointer aynen dizi gibi kullanılabilir. Yani *ptr = dizi;* komutunu verdikten sonra, ptr[0] ile dizi[0] birbirinin aynısıdır. Eğer \*ptr yazarsak, yine dizinin ilk elemanı dizi[0]'ı işaret etmiş oluruz. Ancak dizi işaret eden pointer'lar genellikle, \*( ptr + 0 ) şeklinde kullanılır. Burada 0 yerine ne yazarsak, dizinin o elemanını elde ederiz. Diyelim ki, 5.elemanı ( yani dizi[ 4 ] ) kullanmak istiyoruz, o zaman \*( ptr + 4 ) yazarız.

**Örnek 1.9.1.1**

#include<stdio.h>

int main( void )

{ int i;

// dizi'yi tanimliyoruz.

int dizi[ 6 ] = { 4, 8, 15, 16, 23, 42 };

// ptr adinda bir pointer tanimliyoruz.

int \*ptr;

// ptr'nin dizi'yi isaret etmesini soyluyoruz.

ptr = dizi;

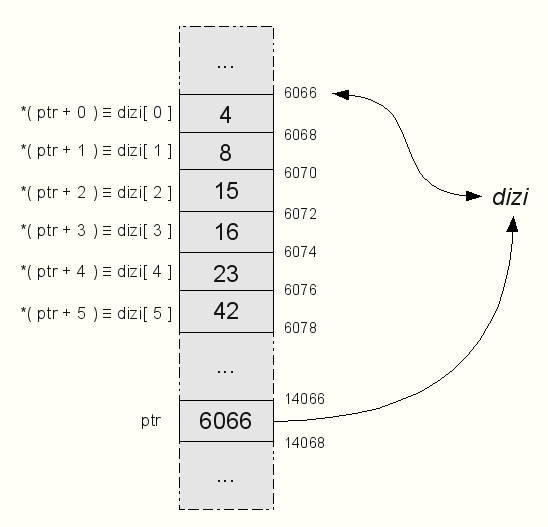
// ptr'in degerini artirip, dizi'nin butun elemanlarini yazdiriyoruz.

for( i = 0; i < 6; i++ )

printf( "%d\n", \*( ptr + i ) );

return 0;}

Yukarıdaki kodu resme dökmek gerekirse;



**Örnek 1.9.1.2**

#include<stdio.h>

int main( void )

{ int elm; int month[ 12 ]; int \*ptr;

ptr = month; // month[0] adresini atar

elm = ptr[ 3 ]; // elm = month[ 3 ] değerini atar

ptr = month + 3; // month[ 3 ] adresini atar

ptr = &month[ 3 ]; // month[ 3 ] adresini atar

elm = ( ptr+2 )[ 2 ]; // elm = ptr[ 4 ] ( = month[ 7 ] )değeri atanır.

elm = \*( month + 3 ); ptr = month;

elm = \*( ptr + 2 ); // elm = month[ 2 ] değerini atar

elm = \*( month + 1 ); // elm = month[ 1 ] değerini atar

return 0;}

Yukarıdaki kodda herhangi bir şeyi printf() fonksiyonu ile ekrana yazdırmadığımız için çıktı olmaz.

**Örnek 1.9.1.3**

#include <stdio.h>

double ortalama(double dizi[], int n);

int main()

{

double a[5] = {1.1, 2.2, 3.3, 4.4, 5.5};

double o;

o = ortalama(a,5);

printf("Dizinin ortalamasi = %lf\n",o);

return 0;

}

double ortalama(double dizi[], int n)

{

double \*p, t=0.0;

int i;

p = dizi; /\* veya p = &dizi[0] \*/

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

t += \*(p+i);

return (t/n);

}

* 1. **Dinamik Bellek Yönetimi**

C dilinde, dizi tanımlaması gibi yapılan bazı işlemler belleğin otomatik olarak tahsis edilmesini sağlarken, bazı işlemler bu olanağı sağlamaz. Bu durumda, programın çalışması esnasında gereksinim duyduğumuzda bellek tahsis edebiliriz. Bu işleme dinamik bellek yönetimi adı verilir.

C standartlarına göre, tüm derleyiciler tarafından sağlanan 4 adet dinamik bellek fonksiyonu vardır;

* malloc()
* calloc()
* realloc()
* free()

## 1.10.1 Malloc() Ve Free() Fonksiyonu

Dinamik bellek kullanımı için genellikle malloc() ve free() fonksiyonları birlikte kullanılır. malloc() fonksiyonu belleği tahsis ederken, free() fonksiyonu ise önceden tahsis edilmiş belleği boşa çıkarır. Bu fonksiyonların yazılış şekilleri aşağıda gösterilmiştir;

void \*malloc (size\_t byte-sayısı);

void free (void \*p);

malloc() fonksiyonunda kullanılan byte-sayısı ifadesi tahsis etmek istediğimiz belleğin byte olarak değerini gösterir. malloc() fonksiyonu tahsis edilmiş belleğin başlangıcını gösteren bir işaretçi geri verir. Tahsis edilmek istenen bellek ihtiyacını karşılayamadığında, NULL bir işaretçi geri verir.

Tahsis edilen belleği boşa çıkarmak için free() fonksiyonu, daha önce malloc() fonksiyonu ile tahsis edilen belleğin başlangıcını gösteren bir işaretçi ile kullanılır.

**Malloc() ve free() fonksiyonları stdlib.h başlık dosyasını kullanır.**

**Örnek 1.10.1.1**

#include <stdio.h>

#include <stdlib.h>

int main(void)

{ int \*ip;

ip = (int\*) malloc(sizeof(int)); /\* 1 \*/

\*ip = 126;

printf("%p\n", ip); printf("%d", \*ip);

free(ip);

return 0;}

**Örnek 1.10.1.2**

#include <stdio.h>

#include <stdlib.h>

int main(void)

{char \*cp; cp = malloc(40);

if (!cp) {

printf("Bellek tahsis hatası!"); exit(1);}

printf("Bir karakter dizisi giriniz: "); gets(cp);

printf(cp);

free(cp);

return 0;}

**Örnek 1.10.1.3**

#include<stdio.h>

#include<stdlib.h>

int main( void )

{// Dinamik bir dizi yaratmak icin pointer kullaniriz.

int \*dizi;

// Dizimizin kac elemanli olacagini eleman\_sayisi isimli degiskende tutuyoruz.

int eleman\_sayisi;

int i;

printf( "Eleman sayisini giriniz> ");

scanf( "%d", &eleman\_sayisi );

// malloc( ) fonksiyonuyla dinamik olarak dizimizi istedigimiz boyutta yaratiyoruz.

dizi = malloc( eleman\_sayisi \* sizeof( int ) );

for( i = 0; i < eleman\_sayisi; i++ )

printf( "%d\n", dizi[i] );

// Dinamik olan diziyi kullandiktan ve isinizi tamamladiktan sonra free fonksiyonunu kullanip hafizadan temizlemelisiniz.

free( dizi );

return 0;

}

**Örnek 1.10.1.4**

#include<stdio.h>

#include<stdlib.h>

int main( void )

{ int \*\*matris;

int satir\_sayisi, sutun\_sayisi;

int i, j;

printf( "Satir sayisi giriniz> " ); scanf( "%d", &satir\_sayisi );

printf( "Sutun sayisi giriniz> " ); scanf( "%d", &sutun\_sayisi );

// Once satir sayisina gore hafizada yer ayiriyoruz. Eger gerekli miktar yoksa, uyari veriliyor.

matris = (int \*\*)malloc( satir\_sayisi \* sizeof(int) );

if( matris == NULL )

printf( "Yetersiz bellek!" );

// Daha sonra her satirda, sutun sayisi kadar hucrenin ayrilmasini sagliyoruz.

for( i = 0; i < satir\_sayisi; i++ ) {

matris[i] = malloc( sutun\_sayisi \* sizeof(int) );

if( matris[i] == NULL )

printf( "Yetersiz bellek!" );

}

// Ornek olmasi acisindan matris degerleri gosteriliyor. Dizilerde yaptiginiz butun islemleri //burada da yapabilirsiniz.

for( i = 0; i < satir\_sayisi; i++ ) {

for( j = 0; j < sutun\_sayisi; j++ )

printf( "%d ", matris[i][j] ); printf( "\n" );}

// Bu noktada matris ile isimiz bittiginden hafizayi bosaltmamiz gerekiyor. Oncelikle

// satirlari bosaltiyoruz.

for( i = 0; i < satir\_sayisi; i++ ) {

free( matris[i] );}

// Satirlar bosaldiktan sonra, matrisin bos oldugunu isaretliyoruz.

free( matris );

return 0;

}

**1.10.2 Calloc() Fonksiyonu**

Bellek tahsislerinde calloc() fonksiyonu da kullanılabilir. Bu fonksiyonun malloc() fonksiyonundan tek farkı ayrılacak bellek miktarının eleman boyutu ve eleman sayısı olarak iki argüman halinde tanımlanmış olmasıdır. calloc() fonksiyonunun genel yapısı aşağıda gösterilmiştir;

void \*calloc (size\_t elem-say, size\_t elem\_boy);

Yukarıdaki satırda, elem\_say ifadesi eleman sayısını, elem\_boy ifadesi ise, eleman boyutunu ifade eder. Aşağıda yer alan her iki işlem satırı aynı işlemi gerçekleştirir:

ip = malloc (10 \* sizeof(int));

ip = calloc (10, sizeof (int));

**Örnek 1.10.2.1**

#include<stdio.h>

#include<stdlib.h>

int main( void )

{// Dinamik bir dizi yaratmak icin pointer kullaniriz.

int \*dizi;

// Dizimizin kac elemanli olacagini eleman\_sayisi isimli degiskende tutuyoruz.

int eleman\_sayisi;

int i;

// Kullanicidan eleman sayisini girmesini istiyoruz.

printf( "Eleman sayisini giriniz> "); scanf( "%d", &eleman\_sayisi );

// calloc( ) fonksiyonuyla dinamik olarak dizimizi istedigimiz boyutta yaratiyoruz.

dizi = calloc( eleman\_sayisi, sizeof( int ) );

// Ornek olmasi acisindan dizinin elemanlarini ekrana yazdiriliyor. Dizilerde yapabildiginiz

// her seyi hicbir fark olmaksizin yapabilirsiniz.

for( i = 0; i < eleman\_sayisi; i++ )

printf( "%d\n", dizi[i] );

// Dinamik olan diziyi kullandiktan ve isinizi tamamladiktan sonra free fonksiyonunu kullanip

// hafizadan temizlemelisiniz.

free( dizi );

return 0;

}

**Örnek 1.10.2.2**

#include <stdio.h>

#include <stdlib.h>

int main(){

int \*p;

p=(int \*)calloc(100,sizeof(int));

if(p==NULL){

printf("Yer ayrilamadi.");}

return 0;}

Yukarıdaki örnekte eğer yer ayrılırsa ekrana bir şey yazdırmadığımız için program çalıştığında ekran çıktısı gözükmeyecektir.

## 1.10.3 Realloc() Fonksiyonu

Daha önce malloc(), calloc() veya realloc() fonksiyonu ile tahsis edilen belleğin boyutunu, boyutu byte olarak ifade edilen size parametre değeri kadar, değiştirir. realloc() fonksiyonunun genel yapısı aşağıda gösterilmiştir;

**void\* realloc(void \*ptr, size\_t size);**

Yeniden bellek tahsis işlemi ptr parametresi ile gösterilen bellek adresi büyütülerek veya küçültülerek yapılabilir. Bellek alanı genişletilirse, tahsis edilen önceki bellek içeriği değişmeden kalır ve eklenen bellek içeriğine herhangi bir değer atanmaz.

Yeterli bellek yoksa, eski bellek bloğu serbest bırakılmaz ve NULL bir işaretçi geri döndürülür.

**Parametreler**

**ptr:** Yeniden tahsis edilecek bellek bölgesini gösterir.

**size:** Tahsis edilecek belleğin yeni boyutunu byte olarak gösterir.

**Örnek 1.10.3.1**

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

int \*ip, id;

ip = (int \*) malloc(5 \* sizeof(int));

for (id=0; id<5; id++) {

\*(ip+id) = id+1;

printf("%p adresindeki deger: %d\n", (ip+id), \*(ip+id));

}

ip = (int \*) realloc(ip, 10 \* sizeof(int));

printf("Genisletilmis bellek degerleri:\n");

for ( ; id<10; id++) { /\* Burada id değişken değeri 5 olarak başlar. \*/

\*(ip+id) = id+1;

printf("%p adresindeki deger: %d\n", (ip+id), \*(ip+id));

}

free(ip);

return 0;

}

**Örnek 1.10.3.2**

#include <stdio.h>

#include <stdlib.h>

int main(){

int \*p,sayac=0,secim=1,i;

p=(int \*)malloc(2\*sizeof(int));

if(p==NULL){

printf("Yer ayrilamadi.");

}

while(secim!=0){

printf("Bir deger giriniz:"); scanf("%d",&p,secim);

p[sayac]=secim;

sayac++;

p=(int \*)realloc(p,(sayac+2)\*sizeof(int));

}

for(i=0;i>sayac;i++);{

printf("%d\t",p[i]);

}

free(p);

return 0;

}

* 1. **String.h Kütüphanesi**

C Programlama Dili standart kütüphanesinde yer alan **string.h** başlık dosyasında karakter

dizileri ile ilgili fonksiyon, veri türü ve makro tanımlamaları içerir.

|  |  |
| --- | --- |
| **[strcat](https://www.bilgigunlugum.net/prog/cprog/c_stdkut/string/strcat)** | **char\* strcat(char \*dest, const char \*src);**  İkinci parametredeki karakter dizisini ilk parametrede gösterilen karakter dizisinin sonuna ekler. |
| [**strcmp**](https://www.bilgigunlugum.net/prog/cprog/c_stdkut/string/strcmp) | **int strcmp(const char \*str1, const char \*str2);**  İlk ve ikinci parametredeki karakter dizilerini eşitlik durumunu belirlemek için birbiriyle karşılaştırır. |
| **[strcpy](https://www.bilgigunlugum.net/prog/cprog/c_stdkut/string/strcpy)** | **char\* strcpy(char \*dest, const char \*src);**  İkinci parametredeki karakter dizisini ilk parametrede gösterilen karakter dizisine kopyalar. |
| **[strlen](https://www.bilgigunlugum.net/prog/cprog/c_stdkut/string/strlen)** | **size\_t strlen(const char \*str);**  Kendisine geçirilen parametredeki karakter dizisinin uzunluğunu geri döndürür. |
| [**strncat**](https://www.bilgigunlugum.net/prog/cprog/c_stdkut/string/strncat) | **char\* strncat(char \*dest, const char \*src, size\_t n);**  İkinci parametredeki karakter dizisinin, üçüncü parametrede gösterilen değer kadar ilk byte  değerini ilk parametrede gösterilen karakter dizisinin sonuna ekler. |
| [**strncmp**](https://www.bilgigunlugum.net/prog/cprog/c_stdkut/string/strncmp) | **int strcmp(const char \*str1, const char \*str2, size\_t n);**  İlk ve ikinci parametredeki karakter dizilerinin, üçüncü parametrede gösterilen değer kadar,  ilk karakterini birbiriyle eşitlik durumunu belirlemek için karşılaştırır. |
| **[strncpy](https://www.bilgigunlugum.net/prog/cprog/c_stdkut/string/strncpy)** | **char\* strncpy(char \*dest, const char \*src, size\_t n);**  İkinci parametredeki karakter dizisinin, üçüncü parametrede gösterilen değer kadar, ilk byte  değerini ilk parametrede gösterilen karakter dizisine kopyalar. |

|  |  |
| --- | --- |
| **strchr** | **char\* strchr(const char \*str, int c);**  İkinci parametredeki unsigned char değerinin ilk parametrede gösterilen karakter  dizisinin içinde bulunduğu ilk yeri arar. |

**Örnek 1.11.1**

#include<stdio.h>

#include<string.h>

int main( void )

{printf( "Kelime Uzunlugu: %d\n", strlen("Merhaba") );

return 0;

}

**Örnek 1.11.2**

#include<stdio.h>

#include<string.h>

int main( void )

{

char kaynak[40]="Merhaba Dünya";

char kopya[30] = "";

strncpy( kopya, kaynak, 9 );

printf( "%s\n", kopya );

return 0;

}

**Örnek 1.11.3**

#include<stdio.h>

#include<string.h>

int main( void )

{

int sonuc;

char ilk\_kelime[40]="Maymun";

char ikinci\_kelime[40]="Maytap";

sonuc = strcmp( ilk\_kelime, ikinci\_kelime );

printf( "%d\n", sonuc );

sonuc = strncmp( ilk\_kelime, ikinci\_kelime, 3 );

printf( "%d\n", sonuc );

return 0;

}

İlk önce çağrılan *strcmp(  )*, null karakterini görene kadar bütün karakterleri karşılaştıracak ve geriye negatif bir değer döndürecektir. Çünkü "*Maymum*" kelimesi alfabede "*Maytap*" kelimesinden önce gelir; dolayısıyla küçüktür. Fakat ikinci olarak çağırdığımız *strncmp(  )* geriye 0 değeri verecektir. Her iki katarın ilk üç harfi aynıdır ve fonksiyonda sadece ilk üç harfin karşılaştırılmasını istediğimizi belirttik. Dolayısıyla karşılaştırmanın sonucunda 0 döndürülmesi normaldir.

**Örnek 1.11.4**

#include<stdio.h>

#include<string.h>

int main( void )

{

char ad[30], soyad[20];

char isim\_soyad[50];

printf( "Ad ve soyadinizi giriniz> " );

scanf( "%s%s", ad, soyad );

// isim\_soyad <-- ad

strcat( isim\_soyad, ad );

// isim\_soyad <-- ad + " "

strcat( isim\_soyad, " " );

// isim\_soyad <-- ad + " " + soyad

strcat( isim\_soyad, soyad );

printf( "Tam Isim: %s\n", isim\_soyad );

return 0;

}

**Örnek 1.11.5**

#include <stdio.h>

#include <string.h>

int main(int argc, char const \*argv[])

{

char ilk[10] = "Bilgi";

char son[10] = "merkezi";

char birlesim[20] = "";

printf("%s %s\n", ilk, son);

strcat(birlesim, ilk);

printf("%s\n", birlesim);

strcat(birlesim, " ");

printf("%s\n", birlesim);

/\* strcat(birlesim, son); \*/

strncat(birlesim, son, 3);

printf("%s\n", birlesim);

getch();

return 0;

}

**Örnek 1.11.6**

#include <stdio.h>

#include <string.h>

int main(int argc, char const \*argv[])

{

char cumle[] = "Karaman Toki Elmasehir Mahallesi";

char \*ilk;

char \*son;

ilk = strchr(cumle, 'a');

son = strrchr(cumle, 'a');

if(ilk != NULL || ilk != NULL)

printf("Ilk Nokta = %d\nIkinci Nokta = %d", ilk - cumle, son - cumle);

else

printf("Eslesme Yok");

getch();

return 0;

}

#### **atoi(  ) ve atof(  ) ile string dönüşümü**

#### Verilen kelimeyi, sayıya çevirmek gerekebilir. Eğer elinizdeki metni, bir tam sayıya ( int ) çevirecekseniz, atoi(  ) fonksiyonunu kullanmanız gerekir. Şayet dönüşüm sonunda elde etmek istediğiniz değişken tipi, virgüllü sayı ise ( float ), atof(  ) fonksiyonu kullanılır. Her iki fonksiyon stdlib.h kütüphanesi içindedir.

#### **Bu fonksiyonları kullanırken, #include<stdlib.h> komutunu program başlangıcına yazmalısınız.**

**Örnek 1.11.1.1**

#include<stdio.h>

#include<stdlib.h>

int main( void )

{

char kok\_iki[] = "1.414213";

char pi[] = "3.14";

char tam\_bir\_sayi[] = "156";

char hayatin\_anlami[] = "42 is the answer";

printf( "%d\n", atoi( tam\_bir\_sayi ) );

printf( "%d\n", atoi( hayatin\_anlami ) );

printf( "%f\n", atof( kok\_iki ) );

printf( "%f\n", atof( pi ) );

return 0;

}

* 1. **Math.h Kütüphanesi**

**Program 1.12.1**

#include <stdio.h>

#include <math.h>

int main(int argc, char const \*argv[])

{

int y = 5; double x = 1.5; double ondalik; double tam; double deger = 0.5; double pi = 3.14;

printf("e uzeri %d = %f\n", y, exp(y)); //üssü hesaplar.

printf("%d'nin karekoku = %f\n", y, sqrt(y)); //karekök hesaplar.

printf("%d'nin karesi = %f\n", y, pow(y, 2)); //karesini hesaplar.

printf("log(%d) = %f\n", y, log(y)); //logaritma hesaplar.

printf("floor(%f) = %f\n", x, floor(x)); // alt onluğa yuvarlar.

printf("ceil(%f) = %f\n", x, ceil(x)); //üst onluğa yuvarlar.

printf("fabs(-5) = %f\n", fabs(-5)); //negatifi pozitife döndürür.

ondalik = modf(x, &tam);

printf("Ondalik = %f, Tam = %f\n", ondalik, tam); //ondalık ve tam kısmı ayırır.

printf("fmod(57.8, 5.5) = %f\n", fmod(57.8, 5.5)); // modunu alır.

printf("Cos(%f) = %f\n", deger, cos(deger) \* 180 / pi); //cosinüs hesaplar.

printf("Sin(%f) = %f\n", deger, sin(deger) \* 180 / pi); //sinüs hesaplar.

printf("Cosh(%f) = %f\n", deger, cosh(deger) \* 180 / pi); // hiperbolik cosinüs hesaplar.

printf("Sinh(%f) = %f\n", deger, sinh(deger) \* 180 / pi); // hiperbolik sinüs hesaplar.

printf("Tan(%f) = %f\n", deger, tan(deger) \* 180 / pi); // tanjant hesaplar.

getch();

return 0;

}

**Örnek 1.12.1**

#include<stdio.h>

#include<math.h>

int main()

{ int sayi;

double x;

printf("sayi girin: "); scanf("%d",&sayi);

x=exp(sayi);

printf("e uzeri %d = %.2f",sayi,x);

getchar(); getchar();

return 0;}

**Örnek 1.12.2**

#include<stdio.h>

#include<math.h>

int main()

{

int x,y;

int sonuc;

printf("X uzeri y icin x ve y degerlerini giriniz:");

scanf("%d%d",&x,&y);

sonuc=pow(x,y);

printf("%d uzeri %d = %d",x,y,sonuc);

getchar();getchar();

return 0;

}

**Örnek 1.12.3**

#include<stdio.h>

#include<math.h>

int main()

{ double x;

printf("Bir ondalik sayi giriniz (ondalik olmasi icin araya Nokta koyunuz):"); scanf("%lf",&x);

printf("Floor icin = %f /n",floor(x));

printf("Ceil icin = %f",ceil(x));

getchar();getchar(); getchar();getchar();

return 0; }

**Örnek 1.12.4**

#include<stdio.h>

#include<math.h>

int main()

{

double x;

printf("Negatif bir sayi giriniz :");

scanf("%lf",&x);

printf("mutlak deger icin = %.2f ",fabs(x));

getchar();getchar();

return 0;

}

* 1. **Structlar (Yapılar)**

Türleri farklı olan verileri tek bir isim altında toplayabilmemiz için C bize farklı ve özel bir veri türü kullanma olanağı sağlar. Bu özel veri türüne **yapı (structure)** adı verilir.

Yapı, farklı veri türlerini bir grup altında toplayan bir veri türüdür. İki veya daha fazla elemandan oluşur. Bir yapı içindeki bütün veri tipleri aynı olabileceği gibi, her veri türü birbirinden farklı da olabilir.

## 1.13.1 Yapı Bildirimi

Yapılar aşağıdaki 3 yöntemin herhangi biriyle tanımlanabilir;

**1.13.1.1 Veri yapısı ve değişkenlerin birlikte tanımlanması**

**struct** **adı** {

veri-türü eleman1;

veri-türü eleman2;

.

.

veri-türü elemanN;

} **değişken-listesi**;

**1.13.1.2 Veri yapısı ve değişkenlerin ayrı ayrı tanımlanması**

**struct** **adı** {

veri-türü eleman1;

veri-türü eleman2;

.

.

veri-türü elemanN;

}

**struct** **adı** **değişken**;

**1.13.1.3 Yapı adı kullanmadan sadece değişken kullanarak yapı tanımlanması**

**struct** {

veri-türü eleman1;

veri-türü eleman2;

.

.

.

veri-türü elemanN;

} **değişken-listesi**;

Yapı bildiriminde, **adı** veya **değişken-listesi** ifadelerinden birisi mutlaka tanımlanmış olmalıdır. Yukarıdaki her üç yöntemde de, bu kural uygulanmaktadır.

Yukarıda görülen struct kelimesi bir yapı tanımlar. **adı** ifadesi ise yapının adını göstermektedir. **veri-türü** ifadesi C'de kullanılan herhangi bir veri türünü gösterir. değişken-listesi ifadesi yapı için tanımlanan değişkenleri göstermektedir.

Yukarıda kullanılan yapı tanımlama yönteminin ilkinde yapı ve yapı değişken değerleri birlikte tanımlanır. İkinci yöntemde, önce yapı daha sonra yapı değişken değeri tanımlanmaktadır. Üçüncü yöntemde ise, yapı adı kullanmadan sadece değişken adı kullanarak yapı tanımlanmaktadır. Her üç yöntem de aynı sonucu vermektedir.

**Eğer sadece “adı” ifadesi kullanılırsa, veri türünün sadece yapısı oluşturulmuş olur. Değişken bildirimi ayrıca yapılmalıdır.**

struct yap {

char cdizi1[15]; /\* İsim \*/

char cdizi2[15]; /\* Soyadı \*/

char cdizi3[15]; /\* Memleketi \*/

char cdizi4[15]; /\* Tahsili \*/

int id; /\* Yaşı \*/

} yd;

**1.13.2 Yapı Elemanlarına Değer Atanması**

Bir yapı ve yapı değişkeni oluşturulduktan sonra, yapı içinde yer alan bir elemana yapı değişkeni yolu ile ulaşmak için aşağıdaki ifadeyi kullanabiliriz;

**yapı-değişkeni**.alan-adı

Örneğin yap yapısının id değişkenine 21 değerini atamak için aşağıdaki ifadeyi kullanabiliriz;

yd.id = 21;

Aynı değeri ekrana yazmak için ise şu ifadeyi kullanabiliriz;

printf("%d", yd.id);

yap yapısının id değişkenine kullanıcının girdiği bir değeri okutmak için şu ifadeyi kullanabiliriz;

printf("Yaşını giriniz: ");

scanf("%d", &yd.id);

yap yapısında yer alan cdizi1 dizisine bir değer girmek ve aynı değeri ekrana yazdırmak için aşağıdaki ifadeleri kullanabiliriz;

printf("Adını giriniz: ");

gets(yd.cdizi1);

printf("%s", yd.cdizi1);

yap yapısında yer alan cdizi1 dizisinin ikinci karakterine erişmek için de aşağıdaki ifadeyi kullanabiliriz;

printf("%c", yd.cdizi1[1]);

**Yapılar sayesinde tek bir değişken yolu ile birden fazla farklı türden veriye ulaşabiliriz.**

**Örnek 1.13.1**

#include <stdio.h>

struct yap {

char cdizi1[15]; /\* İsim \*/

char cdizi2[15]; /\* Soyadı \*/

char cdizi3[15]; /\* Memleketi \*/

char cdizi4[15]; /\* Bölümü \*/

int id; /\* Yaşı \*/

}

yd;

int main(void)

{

printf("Adi: ");

gets(yd.cdizi1);

printf("Soyadi: ");

gets(yd.cdizi2);

printf("Memleketi: ");

gets(yd.cdizi3);

printf("Bolumu: ");

gets(yd.cdizi4);

printf("Yasi: ");

scanf("%d", &yd.id);

printf("%s %s %s %s %d", yd.cdizi1, yd.cdizi2, yd.cdizi3, yd.cdizi4, yd.id);

return 0;

}

**Örnek 1.13.2**

#include <stdio.h>

int main(void)

{struct yap {

int id;

char cd;} yd;

int id = 21;

yd.id = 192;

yd.cd = 'A';

printf("%d %d %c", id, yd.id, yd.cd);

return 0; }

**Örnek 1.13.3**

int main( void )

{

// Degisken tipinin nasil olacagini tanimliyoruz

enum ana\_renkler {

Kirmizi,

Mavi,

Sari};

// Degiskeni tanimliyoruz.

enum ana\_renkler piksel;

// Degisken degerini Mavi olarak belirliyoruz. Dilersek Sari ve Kirmizi da girebiliriz.

piksel = Mavi;

// Degisken degeri karsilastiriliyor.

if( piksel == Kirmizi )

printf( "Kırmızı piksel\n" );

else if( piksel == Mavi )

printf( "Mavi piksel\n" );

else

printf( "Sarı piksel\n" );

return 0;}

**Örnek 1.13.4**

#include<stdio.h>

int main( void )

{// Yeni veri tipini oluşturuyoruz. Ayrica yeni veri tipinden bir degisken tanimliyoruz.

enum boolean {

false = 0,

true = 1

} dogru\_mu;

dogru\_mu = true;

if( dogru\_mu == true )

printf( "Dogru\n" );

return 0;}

Yukarda gördüğünüz yöntem, yeni veri tipini oluşturduğunuz anda, bu veri tipinden bir değişken tanımlamayı sağlar. Her seferinde *enum* yazmaktan kurtaracak diğer yöntemse, **typedef** kullanmaktan geçer. *typedef* kullanımı şu şekildedir;

**typedef veri\_tipi\_eski\_adi\_veri\_tipi\_yeni\_adi**

**Örnek 1.13.5**

#include<stdio.h>

int main( void )

{

// Yeni veri tipini oluşturuyoruz. Ayrica yeni veri tipinden bir degisken tanimliyoruz.

enum boolean {

false = 0,

true = 1};

// Alttaki komut sayesinde, boolean veri tipini tek adimda yaratabiliyoruz.

typedef enum boolean bool;

bool dogru\_mu;

dogru\_mu = true;

if( dogru\_mu == true )

printf( "Dogru\n" );

return 0;}

**Örnek 1.13.6**

#include<stdio.h>

// Ay listesini olusturuyoruz. Ocak ayi 1 olacak sekilde, aylar sirayla numerik degerler aliyor.

enum ay\_listesi {

ocak = 1, subat, mart, nisan,

mayis, haziran, temmuz, agustos,

eylul, ekim, kasim, aralik};

// Degisken tanimlamasini kolaylaştırmak icin typedef kullaniliyoruz. aylar diyerek

// tanimlama yapmak mumkun hale geliyor.

typedef enum ay\_listesi aylar;

void ay\_ismini\_yazdir( aylar );

int main( void )

{// aylar tipinde bir degisken yaratip, 'kasim' degerini atiyoruz.

aylar bu\_ay = kasim;

// kasim, numerik olarak 11'i ifade edecektir.

printf( "%d. ay: ", bu\_ay );

// fonksiyonumuzu cagiriyoruz.

ay\_ismini\_yazdir( bu\_ay );

return 0;

}

// Kendisine verilen aylar tipindeki degiskene gore

// hangi ayin oldugunu ekrana yazmaktadir.

void ay\_ismini\_yazdir( aylar ay\_adi )

{

switch( ay\_adi ) {

case ocak: printf( "Ocak\n" );break;

case subat: printf( "Subat\n" );break;

case mart: printf( "Mart\n" );break;

case nisan: printf( "Nisan\n" );break;

case mayis: printf( "Mayis\n" );break;

case haziran: printf( "Haziran\n" );break;

case temmuz: printf( "Temmuz\n" );break;

case agustos: printf( "Agustos\n" );break;

case eylul: printf( "Eylul\n" );break;

case ekim: printf( "Ekim\n" );break;

case kasim: printf( "Kasim\n" );break;

case aralik: printf( "Aralik\n" );break;}}

**Örnek 1.13.7**

#include<stdio.h>

int main( void )

{

struct

{ int yil;

int ay;

int gun;

} dogum\_gunu;

printf( "Dogum gununuzu " );

printf( "GG-AA-YYYY olarak giriniz> ");

scanf( "%d-%d-%d", &dogum\_gunu.gun,

&dogum\_gunu.ay,

&dogum\_gunu.yil );

printf( "Dogum gununuz: " );

printf( "%d/%d/%d\n", dogum\_gunu.gun,

dogum\_gunu.ay,

dogum\_gunu.yil );

return 0;

}

**Örnek 1.13.8**

#include<stdio.h>

int main( void )

{struct {

char isim[40];

int boy;

struct {

int yil;

int ay;

int gun;

} dogum\_bilgileri;

} kisi;

printf( "Adiniz: " );

scanf( "%s", kisi.isim );

printf( "Boyunuz: " );

scanf( "%d", &kisi.boy );

printf( "Dogum tarihi: ");

scanf( "%d-%d-%d", &kisi.dogum\_bilgileri.gun,

&kisi.dogum\_bilgileri.ay,

&kisi.dogum\_bilgileri.yil );

printf( "Girilen bilgiler:\n" );

printf( "Isim: %s\n", kisi.isim );

printf( "Boy: %d\n", kisi.boy );

printf( "Dogum tarihi: %d/%d/%d\n", kisi.dogum\_bilgileri.gun,

kisi.dogum\_bilgileri.ay,

kisi.dogum\_bilgileri.yil );

return 0;

}

**Örnek 1.13.9**

#include<stdio.h>

int main( void )

{

int i;

// Dogum tarihi tutmak icin 'dogum\_tarihi' adinda bir yapi olusturuyoruz

struct dogum\_tarihi {

int gun;

int ay;

int yil;

};

// Kisiye ait bilgileri tutmak icin 'sahis\_bilgileri' adinda bir yapi kuruluyor.

struct sahis\_bilgileri {

char isim[40];

int boy;

// Yapi icinde bir baska yapiyi kullanmak mumkundur. dogum\_tarihi

// yapisindan 'tarih' adinda bir degisken tanimlaniyor.

struct dogum\_tarihi tarih;

};

// Dizi elemanlarina ilk deger atamasi yapiyoruz. Dilerseniz klavyeden deger girmeyi tercih //edebilirsiniz.

struct sahis\_bilgileri kisi[3] = { "Sezgin", 170, { 17, 2, 1976 },

"Veli", 178, { 14, 4, 1980 },

"Cenk", 176, { 4, 11, 1983 } };

// Yapi dizisi yazdiriliyor:

for( i = 0; i < 3; i++ ) {

printf( "Kayit no.: %d\n", ( i + 1 ) );

printf( "Ad: %s\n", kisi[i].isim );

printf( "Boy: %d\n", kisi[i].boy );

printf( "Dogum Tarihi: %d/%d/%d\n\n", kisi[i].tarih.gun,

kisi[i].tarih.ay,

kisi[i].tarih.yil );}

return 0;

}

**Örnek 1.13.10**

#include<stdio.h>

#include<string.h>

struct sahis\_bilgileri {

char isim[40];

int boy;

};

struct sahis\_bilgileri bilgileri\_al( void );

void bilgileri\_goster( struct sahis\_bilgileri );

int main( void )

{

struct sahis\_bilgileri kisi;

kisi = bilgileri\_al( );

bilgileri\_goster( kisi );

return 0;

}

struct sahis\_bilgileri bilgileri\_al( void )

{

struct sahis\_bilgileri sahis;

printf( "Isim> " );

gets( sahis.isim );

printf( "Boy> " );

scanf( "%d", &sahis.boy );

return sahis;

}

void bilgileri\_goster( struct sahis\_bilgileri sahis )

{

printf( "Ad: %s\n", sahis.isim );

printf( "Boy: %d\n", sahis.boy );

}

* 1. **Dosya İşlemleri**

C'de, dosya giriş çıkış işlemlerinin temelini, bilgisayarı oluşturan aygıtlar için mantıksal bir arabirim görevi yapan, **Akış** adı verilen bir kavram oluşturur. Genel bir ifade ile, akış bir dosyaya ulaşmak için kullanabileceğiniz mantıksal bir arabirimdir.

Akış, **stdio.h** başlık dosyası içinde tanımlı **FILE** adlı yapı veri türünden bildirimi yapılan bir işaretçi değişkeni ile kullanılan bir dosya veya fiziksel bir cihazdır.

Bir akışın dosya (fiziksel aygıtlar dahil) ile bağlantısını sağlamak için açma işlemi, dosyadan bağlantısını kesmek için kapama işlemi uygulanır. Dosya açıldıktan sonra, dosyadan programınıza bilgi aktarabilir veya dosya içeriğini değiştirebiliriz.

## 1.14.1 Metin Akışı

Metin akışı, herhangi bir metin editörü ile oluşturulan DOS metin dosyaları ile kullanılır. Yani, metin akışı ASCII karakterlerle birlikte kullanılır. C dilinde metin akışı, metin dosyalarını satırları tek bir yeni satır karakteri (ASCII 10) ile ayrılmış olarak dikkate alır. Ancak, DOS metin dosyaları, her satırı arasında bir yeni satır (ASCII 10) ve bir satır başı (ASCII 13) karakteri olmak üzere toplam iki karakter ile, diske kaydedilir.

C dili, metin akışına diskten bir dosya okurken yeni satır ve satır başı karakterlerini tek bir yeni satır karakterine çevirir. Metin akışından diske bir dosyayı kaydederken ise, tek bir yeni satır karakterini, yeni satır ve satır başı karakterlerine çevirir. Bu nedenle, metin akışına gönderilen ifadelerle dosyaya yazılan ifadeler tıpatıp aynı olmayabilir.

## 1.14.2 İkili Sistem Akışı

İkili sistem akışında ise, herhangi bir karakter dönüşümü yapılmaz. Yani, ikili akıştan diske bir dosya kaydederken veya diskten ikili akışa bir dosya aktarırken, dosya içinde yer alan karakterlerde herhangi bir değişiklik yapılmaz. ikili sistem akışına gönderilen ifadelerle dosyaya gönderilen ifadeler tamamen birbirinin aynıdır.

## 1.14.3 Aktif Konum

Aktif konum, bir dosyaya yapılacak bir sonraki girişin yapılacağı konumdur. Örneğin, 80 byte uzunluğunda bir dosyanın ilk erişimde 20 byte ının okunduğunu kabul edelim. Bir sonraki dosya okuma işlemi 21 nci byte dan başlayacaktır. Bu durumda, 21 nci byte aktif konum olarak kabul edilir. Başka bir ifade ile, dosya işlemlerinde yapılan bir sonraki işlem dosya başından değil kalınan yerden devam eder.

## 1.14.4 Dosya Sistemi

C dosya sisteminde kullanılan dosya kavramının, normal bir disk dosyası için kullanılabileceği gibi, ekran, klavye gibi bilgisayar parçaları içinde kullanılabileceğinden bahsetmiştik.

Bir dosya açıldığı zaman, aktif konum göstergesi dosya başlangıcını gösterir. Dosyadan her okuma ve yazma işleminde, aktif konum göstergesi otomatik olarak bir sonraki konuma geçer.

## 1.14.5 Standart Akışlar

Bir programı çalıştırdığınız zaman, program aşağıda gösterilen standart metin akışlarını otomatik olarak açar. Bu metin akışları yoluyla, program bilgisayara bağlı aygıtlara (klavye, ekran, yazıcı gibi...) tıpkı bir dosya gibi erişebilir:

**stdin** : Klavyeden veri okur.

**stdout** : Verileri ekrana yollar.

**stderr** : Verileri ekrana yollar.

**stdaux** : Seri port'tan veri alışverişi yapar.

**stdprn** : Verileri yazıcıya yollar.

**Fonksiyon İşlevi**

**fopen( )** Bir dosya açar.

**fclose( )** Bir dosyayı kapatır.

**fputc( ) ve putc()** Dosyaya bir karakter yazar.

**fgetc( ) ve getc()** Dosyadan bir karakter okur.

**fseek( )** Bir dosyadaki belirli bir byte'ı bulur.

**fprintf( )** Yapılandırılmış verileri dosyaya yazar.

**fscanf( )** Yapılandırılmış verileri dosyadan okur.

**feof( )** Dosya sonu geldiğinde doğru bir değer verir.

**ferror( )** Bir hata durumunda doğru bir değer verir.

**rewind( )** Dosya aktif konumunu başa alır.

**remove( )** Dosyayı siler.

**fflush( )** Tampon belleği siler.

## 1.14.6 Dosya Açma

Bir dosyayı açmak ve bu dosyayı ilgili akışla birleştirmek için aşağıda prototipi verilen fopen() fonksiyonunu kullanmak gerekir;

FILE \*fopen (char \*dosya-adı, char \*mod);

**dosya-adı:** açılacak dosya adı

**mod :** dosya açılış şekli

fopen() fonksiyonu **stdio.h** başlık dosyasını kullanır. dosya-adı ifadesi ile gösterilen ve fopen() fonksiyonuna argüman olarak geçirilen dosya adı işletim sisteminde kullanılan geçerli bir dosya adı olmalıdır. fopen() fonksiyonuna argüman olarak geçirilen ve mod ile gösterilen karakter dizisi ise dosyanın ne şekilde açılacağını belirler. Burada, mod ifadesi yerine kullanılabilecek değerler aşağıdaki tabloda yer almaktadır;

**MOD Anlamı**

**r :** Okuma için bir metin dosyası açar.

**w**  : Yazma için bir metin dosyası oluşturur.

**a :**  Bir metin dosyasına ekleme yapar.

**rb :** Okuma için bir dosyayı ikili sistemde açar.

**wb** : Yazma için ikili sistemde bir dosya oluşturur.

**ab :** İkili sistemde bir dosyaya ekleme yapar.

**r+** : Okuma ve yazma için bir metin dosyası açar.

**w+** : Okuma ve yazma için bir metin dosyası oluşturur.

**a+** : Okuma ve yazma için bir metin dosyası oluşturur veya ekleme yapar.

**r+b :** Okuma ve yazma için bir ikili sistem dosyası açar.

**w+b** : Okuma ve yazma için bir ikili sistem dosyası oluşturur.

**a+b** : Okuma ve yazma için bir ikili sistem dosyasına ekleme yapar.

Eğer dosya açma işlemi başarılı olarak sonuçlanırsa, fopen() fonksiyonu geçerli bir dosya işaretçisi geri verir. **FILE**, stdio.h dosyası içinde tanımlanmış olup dosyanın boyutu, aktif konumu ve giriş modları gibi dosya hakkındaki değişik değerleri içeren bir **YAPI**'dır. Yapı tek bir isimle giriş yapılan bir grup değişkenden oluşur. fopen() fonksiyonu dosya açma işlemi ile birlikte dosya ile ilgili yapıya bir işaretçi geri verir. Bu işaretçi diğer fonksiyonlarla dosya üzerinde işlem yapmak için kullanılır.

Eğer fopen() fonksiyonu normal olarak çalışmazsa NULL bir işaretçi geri verir. NULL stdio.h dosyasında null bir işaretçi olarak tanımlanmıştır. fopen() fonksiyonunun geçerli bir dosya işaretçisi geri vermesi gerekir. Geri verilen değerin NULL olmadığından emin olmak için geri verilen değeri kontrol etmek gerekir.

## 1.14.7 Dosya Kapatma

Bir dosyayı kapatmak için aşağıda prototipi verilen fclose() fonksiyonu kullanılır;

int fclose (FILE \*fp);

fp: fopen() ile kullanılan dosya işaretçisi

**fclose() fonksiyonu fp ile bağlantılı dosyayı kapatır ve dosyanın akış ile olan bağlantısını keser. fclose() fonksiyonu normal olarak çalışırsa 0, bir hata meydana gelirse EOF değerini geri verir.**

## 1.14.8 fgetc() ve fputc() Fonksiyonları

Bir dosya açıldıktan sonra, açılış moduna bağlı olarak, aşağıda genel yapısı verilen 2 fonksiyonu kullanarak dosyaya bir karakter yazabilir veya dosyadan bir karakter okuyabiliriz;

int fgetc (FILE \*fp);

int fputc (int id, FILE \*fp);

* fgetc() fonksiyonu, fp ile tanımlanan dosyadaki bir sonraki byte'ı unsigned char olarak okur ve int bir değer olarak geri verir. fgetc() fonksiyonu bir hata durumunda ve dosya sonuna geldiğinde int bir değer olan EOF değerini geri verdiği için, geri verilen değerin kontrolü amacıyla fgetc() fonksiyonu int bir değer geri verir. Ancak, fgetc() fonksiyonunun geri verdiği değeri int bir değişkene atamanız şart değildir. Geri verilen değeri bir karakter değişkene de atayabilirsiniz.
* fputc() fonksiyonu d1 değişken değeri olan byte'ı fp ile gösterilen dosyaya unsigned char olarak yazar. id değişkeni int bir değer olarak tanımlandığı halde, char bir değer olarak çağırabilirsiniz. Eğer dosyaya yazma işlemi başarılı olursa, fputc() fonksiyonu yazılan karakteri aksi takdirde EOF değerini geri verir.

**Örnek 1.14.1**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

FILE \*dosya;

char kr = 'A';

dosya = fopen("./Ornek\_1.txt", "w");

if(dosya == NULL)

{ printf("Dosya acilamadi\n");

return -1; }

printf("Dosya acildi\n");

fputc(kr, dosya);

fclose(dosya);

printf("Yazma tamamlandi\n");

getch();

return 0;}

**Örnek 1.14.2**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

FILE \*dosya;

char kr;

dosya = fopen("./Ornek\_1.txt", "r");

if(dosya == NULL)

{

printf("Dosya acilamadi\n");

return -1; }

printf("Dosya acildi\n");

kr = fgetc(dosya);

printf("Okunan karakter = %c\n", kr);

fclose(dosya);

printf("Okuma tamamlandi\n");

getch();

return 0;

}

**Örnek 1.14.3**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

FILE \*dosya;

char kr;

dosya = fopen("./Ornek\_1.txt", "r");

if(dosya == NULL)

{printf("Dosya acilamadi\n");

return -1;}

printf("Dosya acildi\n");

while (!feof(dosya))

{

kr = fgetc(dosya);

printf("%c", kr);

}

fclose(dosya);

printf("\nOkuma tamamlandi\n");

getch();

return 0;

}

**Örnek 1.14.4**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

FILE \*dosya;

char kr;

int sayi = 0;

dosya = fopen("./Ornek\_1.txt", "r");

if(dosya == NULL)

{ printf("Dosya acilamadi\n");

return -1; }

printf("Dosya acildi\n");

printf("Aradiginiz karakteri giriniz: ");

scanf("%c", &kr);

while (!feof(dosya))

if(fgetc(dosya) == kr)

sayi++;

printf("Karakterden %d tane bulunmaktadir", sayi);

fclose(dosya);

printf("\nOkuma tamamlandi\n");

getch();

return 0;}

**Örnek 1.14.5**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

FILE \*dosya;

char metin[80];

dosya = fopen("./Ornek\_1.txt", "r");

if(dosya == NULL)

{

printf("Dosya acilamadi\n");

return -1;

}

printf("Dosya acildi\n");

fgets(metin, 80, dosya);

printf("%s", metin);

fclose(dosya);

printf("\nOkuma tamamlandi\n");

getch();

return 0;

}

**Örnek 1.14.6**

#include <stdio.h>

int main(int argc, char const \*argv[])

{

FILE \*dosya;

char metin[80] = "Lab Dersi";

dosya = fopen("./Ornek\_1.txt", "a+");

if(dosya == NULL)

{

printf("Dosya acilamadi\n");

return -1;

}

printf("Dosya acildi\n");

fputs("\n", dosya);

fputs(metin, dosya);

fclose(dosya);

printf("%s yazildi\n", metin);

getch();

return 0;

}

**Örnek 1.14.7**

#include <stdio.h>

typedef struct ogr

{int no;

char \*isim;

}ogrenci;

int main(int argc, char const \*argv[])

{ FILE \*dosya;

int secim;

ogrenci ogrencim, aranacakOgrencim;

ogrencim.isim = (char \*) malloc(sizeof(char) \* 20);

aranacakOgrencim.isim = (char \*) malloc(sizeof(char) \* 20);

while (1)

{printf("1- Kayit Girisi\n2- Kayit Arama\n3- Cikis\nSeciminizi Yapin: ");scanf("%d", &secim);

switch (secim)

{case 1:

dosya = fopen("./Ornek\_2.txt", "a+");

printf("Isim giriniz: ");

scanf("%s", ogrencim.isim);

printf("Numara giriniz: ");

scanf("%d", &ogrencim.no);

fprintf(dosya, "%s\n%d\n", ogrencim.isim, ogrencim.no);

fclose(dosya);

printf("Ogrenci Eklendi!\n");

getch();

break;

case 2:

dosya = fopen("./Ornek\_2.txt", "r");

printf("Numara giriniz: ");

scanf("%d", &aranacakOgrencim.no);

while (!feof(dosya))

{ fscanf(dosya, "%s\n%d\n", ogrencim.isim, &ogrencim.no);

if(aranacakOgrencim.no == ogrencim.no)

{ printf("Ogrenci Bulundu!\n"); break; }}

printf("Arama Tamamlandi!\n");

fclose(dosya);

getch(); break; }

if(secim == 3)

break; }

free(ogrencim.isim);

free(aranacakOgrencim.isim);

fclose(dosya);

return 0;}

**1.14.9 Rastgele Erişim**

Şimdiye kadar incelediğimiz bütün örneklerde, dosyadan yaptığımız okuma işlemlerini dosyanın başından sonuna doğru bir sıra dahilinde yaptık. Bunun yanında, aşağıda prototipi verilen fseek() fonksiyonunu kullanarak bir dosyanın herhangi bir yerindeki bir bilgiyi okuyabiliriz;

**int fseek (FILE \*fp, long ara, int yer);**

Burada, fp ifadesi işlem yapılan dosyayı, ara ifadesi yer ifadesinin gösterdiği değerin tanımladığı dosya konumundan,işlem yapılmak istenen yerin byte olarak uzaklığını verir. yer ifadesi dosyada arama işleminin başlayacağı yeri gösterir. yer ifadesinin yerine aşağıdaki makrolardan birini kullanmak gerekir;

**SEEK\_SET (0) Aramayı dosya başından başlatır.**

**SEEK\_CUR (1) Aramayı aktif konumdan başlatır.**

**SEEK\_END (2) Aramayı dosya sonundan başlatır.**

**Örnek 1.14.8**

#include <stdio.h>

int main(int argc, char const \*argv[])

{ FILE \*dosya;

dosya = fopen("./Ornek\_3.txt", "w");

if(dosya == NULL)

{printf("Dosya acilamadi\n");

return -1;}

printf("Dosya acildi\n"); fputs("Bu bir dosyadir", dosya);

fseek(dosya, 3, SEEK\_SET); fputs("Elma", dosya); printf("Islem tamamlandi\n");

fclose(dosya); getch(); return 0;}

**Örnek 1.14.9**

#include <stdio.h>

#define diziBoyutu 10

int main(int argc, char const \*argv[])

{FILE \*dosya;

int dizi[diziBoyutu] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

int okunanDizi[diziBoyutu];

int i;

dosya = fopen("./Ornek.dat", "wb");

if(dosya == NULL)

{ printf("Dosya acilamadi\n"); return -1; }

printf("Dosya acildi\n");

fwrite(dizi, sizeof(int), diziBoyutu, dosya);

printf("Yazma tamamlandi\n");

fclose(dosya);

getch();

dosya = fopen("./Ornek.dat", "rb");

fread(okunanDizi, sizeof(int), diziBoyutu, dosya);

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

printf("%d\n", okunanDizi[i]);

getch();

return 0;}

**Örnek 1.14.10**

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{ long int lid1, lid2;

int id;

FILE \*fp;

if (argc!=2) {

printf("Kullanim : deneme <Dosya adi>\n");

exit(1); }

if ((fp=fopen(argv[1], "rb")) == NULL) {

printf("Dosya açılamadı!\n");

exit(1); }

/\* Dosya sonuna ulaşıp dosya boyutunu kaydeder. \*/

fseek (fp, 0L, SEEK\_END);

lid2 = ftell(fp);

for ( ; ; ) { if (lid1>=lid2) break;

if (fseek(fp, lid1, SEEK\_SET)) {

printf("Arama hatasi!");

exit(1); } }

fclose(fp);

return 0;}

* 1. **KAYNAKÇA**
* https://komhedos.com
* http://www.cagataycebi.com
* http://www.ibrahimbayraktar.net
* https://ckaynak.com
* https://www.bilgigunlugum.net
* https://f1r4t.com/dosyalar/C\_Programlama\_Dili.pdf
* https://tr.wikibooks.org
* https://wmaraci.com
* https://www.yusufsezer.com.tr
* http://kadirselen.blogspot.com
* https://sanalkurs.net
* https://emresupcin.com
* https://github.com/Syntriax/KMU-Bilgisayar-Depo