

C PROGRAMLAMA

Bilgisayar Mühendisliği

Karamanoğlu Mehmetbey Üniversitesi

İSMET ARSLAN

12.05.2020

ÖN SÖZ

Bu döküman herhangi bir kar amacı gütmeyen bilginin paylaştıkça çoğaldığı inancıyla yayınlanmıştır.

Dökümanın içerisinde verilen örnek kodlara

https://github.com/4rslanismet/C_Rapor_Kodlar

bana

arslanismet@protonmail.com adresinden ulaşabilirsiniz.

İÇİNDEKİLER

BÖLÜM 0 : C PROGRAMLAMA DİLİNE GİRİŞ.....	5
BÖLÜM 1 : C PROGRAMLAMA.....	5
1.0 : C DİLİNİN GENEL YAPISI.....	6
1.1 : DERLEYİCİ VE KOD DERLEME İŞLEMİ.....	7
1.2 : DEĞİŞKEN KAVRAMI VE İLCEL VERİ TİPLERİ.....	8
1.2.0 : İŞARETLİ (SIGNED) VE İŞARETSİZ (UNSIGNED)	
DEĞİŞKENLER.....	8
1.4 : SABİTLER.....	10
1.5 : OPERATÖRLER.....	12
1.5.0 : ARİTMETİK İŞLEM OPERATÖRLERİ.....	12
1.5.1 : KARŞILAŞTIRMA OPERATÖRLERİ.....	13
1.5.2 : MANTIKSAL OPERATÖRLER.....	14
1.5.3 : ATAMA OPERATÖRLERİ.....	14
1.5.4 : ARTTIRMA VE AZATMA OPERATÖRLERİ	16
BÖLÜM 2 : PROGRAM DENETİM YAPILARI.....	17
2.0 : DEYİM NEDİR ?.....	17
2.1 : İF DEYİMİ.....	19
2.2 : İF-ELSE DEYİMİ.....	21
2.3 : ELSE İF DEYİMİ.....	23
2.4 : SWITCH-CASE YAPISI.....	27
BÖLÜM 3 : DÖNGÜLER.....	31
3.0 : WHILE DÖNGÜSÜ.....	31
3.1 : DO-WHILE DÖNGÜSÜ.....	34
3.2 : FOR DÖNGÜSÜ.....	35
3.3 : BREAK DEYİMİ.....	38
BÖLÜM 4 : ÖN İŞLEMCİ KOMUTLARI.....	39
4.0 : #include DİREKTİFİ.....	39
4.1 : #define DİREKTİFİ.....	40
4.2 : #undef DİREKTİFİ.....	40
BÖLÜM 5 : DİZİLER.....	41
5.0 : TEK BOYUTLU DİZİLER.....	41
5.1 : KARAKTER DİZİLERİ.....	48
5.2 : İKİ VE ÇOK BOYUTLU DİZİLER.....	48
BÖLÜM 6 : FONKSİYONLAR.....	51
6.0 : FONKSİYON TANIMLAMA	52
6.1 : DEĞER İLE FONKSİYON ÇAĞIRMA (CALL BY VALUE)	56
6.2 : REFERANS İLE FONKSİYON ÇAĞIRMA (CALL BY REFERENCE).....	59
6.3 : REKÜRSİF (ÖZYİNELİ) FONKSİYONLAR.....	61
BÖLÜM 7 : İŞARETÇİLER (POINTERS).....	66
7.0 : DİZİLER VE POINTERLAR.....	68

BÖLÜM 8 : DİNAMİK BELLEK YÖNETİMİ (DYNAMIC MEMORY MANAGEMENT).....	70
8.0 : MALLOC FONKSİYONU.....	71
8.1 : CALLOC FONKSİYONU.....	73
8.2 : REALLOC FONKSİYONU.....	74
8.3 : FREE FONKSİYONU.....	75
BÖLÜM 9 : STRINGLER.....	77
9.0 : GETS FONKSİYONU	79
9.1 : FGETS FONKSİYONU.....	80
9.2 : FPUTS FONKSİYONU.....	81
9.3 : PUTS FONKSİYONU.....	82
9.4 : string.h KÜTÜPHANESİ.....	83
9.4.0 : STRLEN FONKSİYONU.....	83
9.4.1 : STRNLEN FONKSİYONU.....	84
9.4.2 : STRCMP FONKSİYONU.....	85
9.4.3 : STRNCMP FONKSİYONU.....	86
9.4.4 : STRCAT FONKSİYONU	87
9.4.5 : STRNCAT FONKSİYONU.....	88
9.4.6 : STRCPY FONKSİYONU.....	89
9.4.7 :STRNCPYFONKSİYONU.....	90
9.4.8 : STRCHR FONKSİYONU.....	91
9.4.9 : STRSTR FONKSİYONU.....	93
BÖLÜM 10 : MATH.H KÜTÜPHANESİ	94
10.0 : ACOS FONKSİYONU.....	94
10.1 : ASİN FONSKİYONU.....	95
10.2 :ATAN FONSKİYONU.....	95
10.3 : COS FONSKİYONU.....	96
10.4 : COSH FONSKİYONU.....	97
10.5 : SİN FONSKİYONU.....	98
10.6 : SİNH FONSKİYONU.....	99
10.7 : TANH FONSKİYONU.....	100
10.8 : EXP FONSKİYONU.....	100
10.9 : FREXP FONKSİYONU.....	101
10.10 : LDEXP FONKSİYONU.....	102
10.11 : MODF FONKSİYONU.....	103
10.12 : LOG FONKSİYONU.....	104
10.13 : LOG10 FONKSİYONU.....	104
10.14 : POW FONKSİYONU.....	105
10.15 : SQRT FONKSİYONU.....	106
10.16 : CEIL FONKSİYONU.....	107
10.17 : FABS FONKSİYONU.....	108
10.18 : FLOOR FONKSİYONU.....	110
10.19 : FMOD FONKSİYONU.....	109
BÖLÜM 11 : YAPILAR (STRUCTURES).....	120

11.0 : İÇ İÇE YAPILAR.....	112
11.0.0 : TYPE KULLANIMI.....	114
11.0.1 : STURCTLARA FARKLI YÖNTEMLE ERİŞİM.....	114
BÖLÜM 12 : DOSYA İŞLEMLERİ.....	115
12.0 : DOSYA İŞLEME FONKSİYONLARI.....	116
12.1 : DOSYAYA VERİ YAZMA.....	119
12.1.0 : PUTC FONKSİYONU.....	119
12.1.1 : PUTS FONKSİYONU.....	120
12.1.2 : PRINTF FONKSİYONU.....	121
12.2 : DOSYADAN VERİ OKUMA	122
12.3 : GETCH VE PUTC İLE ETKİLEŞİMLİ DOSYA OKUMA VE YAZMA.....	124
KAYNAKÇA	125

0. C PROGRAMLAMA DİLİNE GİRİŞ

Genel amaçlı bir programla dili olan C 1972 yılında Bell Labarautarında Dennis Ritchie tarafından Unix işletim sistemini yazmak amacıyla geliştirilmiştir. Esnek yapısı sayesinde mikro denetleyici programlamadan işletim sistemi yazımına , paket programlamadan bilimsel programlara kadar geniş bir yelpazede kullanım olanağı sunar. Popülaritesinin nedenlerinden birisi de bugün kullanılan yüksek seviye programlama dillerinin çoğunun (python , java , php , ...) C syntax'ine (söz dizimi) benzer olmasıdır.

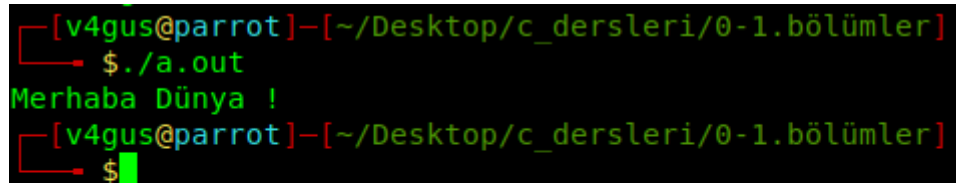
1. C PROGRAMLAMA

1.0. C DİLİNİN GENEL YAPISI

ÖRNEK 1.1:

```
1. // C İLE İLK PROGRAM
2.
3. #include <stdio.h>
4.
5.     int main () //main fonksiyonu başlangıcı
6.     {
7.
8.         printf ("Merhaba Dünya !\n"); // Merhaba Dünya ! yazdırır
9.
10.        return 0;
11.
12.    } //main fonksiyonu sonu
13.
```

ÇIKTI:



```
[v4gus@parrot]~/.
└─$ ./a.out
Merhaba Dünya !
[v4gus@parrot]~/.
└─$
```

3. satırda

gördüğümüz '# include ' bir önışlemci (preprocessor) komutudur , 'dahil et' anlamına gelir. Ön işlemci ve komutlarını ilerleyen bölümlerde daha detaylı bir şekilde anlatacağız bunun için kodumuza geri dönelim. '<stdio.h>' bir kütüphane dosyasıdır (header) .Bu kütüphane bizim standart giriş-çıkış (standart input-output) birimlerimizi ayarlamamızı sağlar. Çünkü C kaynak kodu derledikten sonra nereden giriş yapılacağını ya da nereye çıktı vereceğini bilmez .

Yine bu satırda gördüğümüz iki tane slash '/' derleyici tarafından yorum satırı olarak algılanır ve kod gibi çalıştırılmaya çalışılmadan atlanır. Eğer bir satırda yorum belirteceksek bu yöntem kullanışlı iken daha uzun açıklamalar için " /* yorum */ " yapısını kullanmak daha uygundur.

5.satırda gördüğümüz 'int main ()' ana fonksiyondur. Programımız buradan derlenmeye başlar. Fonksiyonları ilerleyen aşamalarda anlatacağımız için şimdilik kodlarımızın olmazsa olmaz yapısı olarak düşünebilirsiniz.

6ve 12. satırda '{ }' kıvrıcık parantez (curly bracket) fonksiyonların , kontrol ifadelerinin ve değişkenlerin etki alanını belirler. Genellikle 'scope' yani 'alan' da denir.

8.satırda ' printf ' fonksiyonu satandart kütüphane içerisinde bulunan standart çıkışa formatlı veri yazdırma fonksiyonudur. Kodumuzda verisi ise 'Merhaba Dünya !' olarak yazısı standart çıkışa gönderilmiştir. '\n' ise yeni satıra geç (new line) anlamına gelir. En önemli yazım özelliklerinden birisi de her komutumuzun noktalı virgül ';' ile bitmesidir.

10.satırda 'return 0 ;' komutu kodumuzun başarılı bir şekilde yürütüldüğünü işletim sistemine bildirir.

Yani genel olarak bir C programı

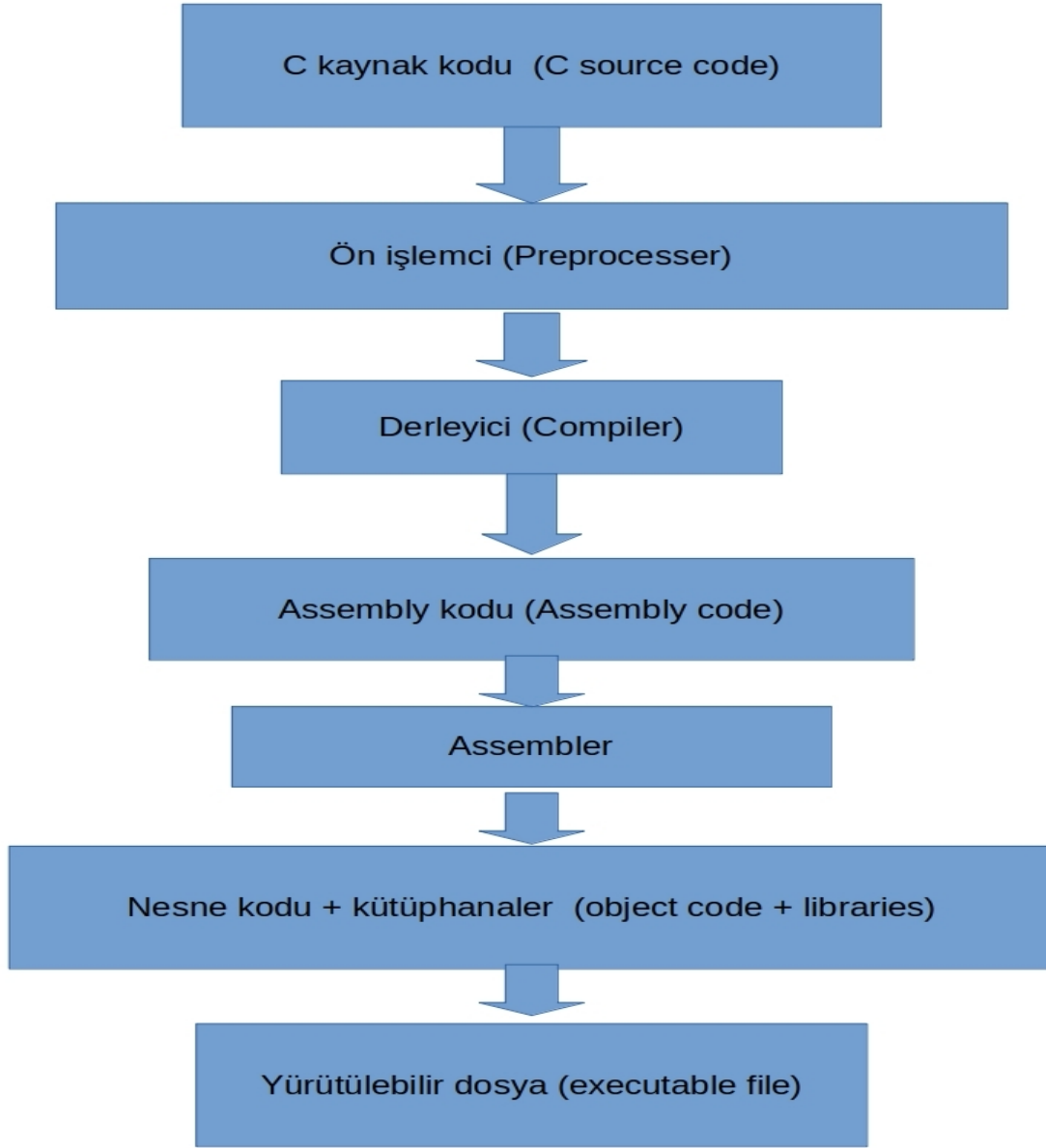
```
#include <stdio.h>
```

```
int main () {  
    kodlar  
}
```

şeklinde yazılır

1.1 DERLEYİCİ VE KOD DERLEME İŞLEMİ

C kaynak kodları (source code) '.c ' uzantılı dosyalarda tutulur. Kaynak kod C derleyicisi (C compiler) ile önce nesne koduna (object code) daha sonra da linker denilen bağlayıcı program ile yürütülebilir (executable) hale getirilir. Bu derleme işlemi çeşitli yöntemler ile yapılabilir ise de anlatım sırasında GNU/Linux dağıtımı olan Parrot OS üzerindeki terminalden GCC version 9.3.0 (Debian 9.3.0-10) üzerinden yapacağız.Her ne kadar işletim sistemi ve programlama ortamı farkı olsa da gerçekleşen işlemler genel olarak tablodaki gibidir.



1.2 DEĞİŞKEN KAVRAMI VE İLKEL VERİ TİPLERİ

Bellekte geçici olarak bilgilerin saklandığı bölümlere genel olarak "değişken" (variable) denir.

Değişken Tanımlama Kuralları

- Değişken adları İngiliz alfabesinde yer almayan bir harf (İ,ı,Ç,ç,Ş,ş,Ğ,ğ,ö,Ö,Ü,ü) içermemelidir.
- Değişken adları rakamla başlamamalıdır.
- Değişken isimleri bir harf veya "_" işareti ile başlayabilir.
- Değişken ismi içerisinde boşluk bırakılamaz.

1.2.0. İşaretli (Signed) ve İşaretsiz (Unsigned) Değişkenler

Bir veri türünün negatif değer almayacaksa ve çok daha fazla değer saklayabilmesi gerekiyorsa ya da benzeri durumlar varsa işaretsiz (unsigned) olarak tanımlanır. Unsigned olarak tanımlanan değişkenler negatif değer alamaz. Herhangi bir belirtme yapılmadığı zaman ise değişken signed (işaretli) olarak tanımlanır. Genellikle işaretli değişkenleri kullanacağımız için tanımlamalarımız işaretli değişkenler için geçerlidir.

```
unsigned int degiskenIsmi=0; // işaretsiz integer değişken tanımlama  
int degisken=-10; // işaretli integer değişken tanımlama
```

Char

Tek karakter saklayabilen veri türüdür (1- byte). "%c" ile okuma ve yazma işlemi yapılır.

Bir char tanımlaması :

```
char karakter='k' ;
```

şeklinde dir.

Integer

Tam sayı türünde minimum -32.768 maksimum 32.768 değer tutabilen veri türüdür (donanıma bağımlı olarak 2 yada 4 byte) . "%d" ile okuma ve yazma işlemi yapılır.

Bir integer tanımlaması :

```
int tamSayi= 70;
```

şeklinde dir.

Float

Kayan noktalı sayıları minimum 3.4×10^{-38} maksimum $3.4 \times 10^{+38}$ değer alabilen veri türüdür (4-byte) . "%f" ile okuma ve yazma işlemi yapılır.

Bir float tanımlaması:

```
float kayanNokataliSayi= 0.7;
```

şeklinde yapılır.

Double

Kayan noktalı sayıları minimum 1.7×10^{-308} maksimum $1.7 \times 10^{+308}$ değer alabilen veri türüdür (8-byte) . "%lf" ile okuma ve yazma işlemi yapılır.

Double tanımlaması :

```
double katanNoktaliSayi=0.34;
```

şeklinde yapılır.

Genel olarak ilkel veri tipleri ve içerisinde bulundurabildikleri değerler tablodaki gibidir.

DEĞİŞKEN TÜRÜ	DEĞİŞKEN İSMİ	RAM'DE TUTULDUĞU ALAN	DEĞİŞKEN DEĞERİ
char	karakter	0x7fff5ed98c4c	a
integer	sayi	0x7fff5ed98c51	70
float	kayanNoktali	0x7fff5ed98c52	0.5
double	kayanNoktali_2	0x7fff5ed98c53	1.2343

Basit olarak kullanıcıdan aldığı iki tam sayının toplamını yazdıran bir program yazalım.

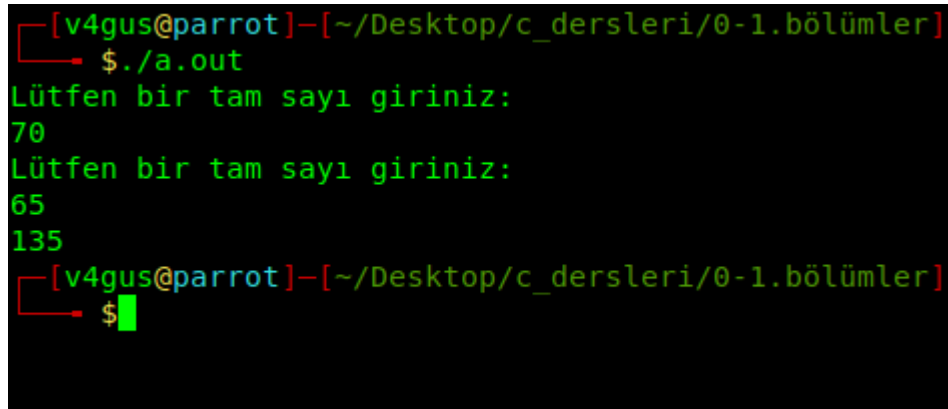
ÖRNEK1.2:

```
1. // KULLANICININ GİRDİĞİ İKİ SAYIYI TOPLAYAMA
2.
3. #include <stdio.h>
4.
5. int main() //main fonksiyonunun başlangıcı
6. {
7.     int sayi1 ;
8.     int sayi2 ;
9.     int toplam;
10.
11.     printf("%s\n","Lütfen bir tam sayı giriniz:" );
12.
13.     scanf("%d",&sayi1); // kullanıcıdan aldığı değeri sayi1 değişkenine atar
```

```
14.  
15.  
16.     printf("%s\n","Lütfen bir tam sayı giriniz:" );  
17.  
18.     scanf("%d",&sayi2); // kullanıcıdan aldığı değeri sayi2 değişkenine atar  
19.  
20.     toplam = sayi1 + sayi2; // sayi1 ve sayi2 yi topla toplam değişkenine ata  
21.  
22.     printf("%d\n",toplam ); //toplam değişkeninin değerini yazdır.  
23.     return 0;  
24.  
25. } //main fonksiyonu sonu
```

Programımızın 11 ve 15. satırlarda printf içerisinde "%s" karakter katarı veya karakter dizisi (string) okunmasını sağlar ve kullanmamızın amacı printf fonksiyonun yalın kullanılması durumunda oluşabilecek olan güvenlik sorunlarını önlemektir. 13. satırda yer alan "scanf" fonksiyonu ise standart girişten girilen verinin okunmasını sağlar. "%d" integer türünü okumak ve yazmak için kullanılan ifadedir.

ÇIKTI:



```
[v4gus@parrot]~/. Desktop/c_dersleri/0-1.bölümler  
$ ./a.out  
Lütfen bir tam sayı giriniz:  
70  
Lütfen bir tam sayı giriniz:  
65  
135  
[v4gus@parrot]~/. Desktop/c_dersleri/0-1.bölümler  
$
```

1.4. SABİTLER

Program içerisinde sabit değerli değişkenlere ihtiyaç duyarız. Tanımlamamızı 'const' deyimi veya '#define' deyimi ile kullanılır. '#define' ile oluşturulan sabitler '#undef' ile sabit tanımlamasını ön işlemcinin unutmamasını sağlar. Sabitler tanımlandığı anda kullanılabilir fakat içeriği değiştirilemez.

#define PI=3.141 // define ile sabit tanımlaması

#undef PI //tanımlanan sabitin etkisiz hale getirilmesi

```
const char []= "İsim";  
const float PI=3.141  
const double E=2.71828182845905;
```

Bir kürenin yarıçapını girdi olarak alıp yüzey alanını ve hacmini hesaplayan program yazalım. Kürenin yüzey alanı $4\pi r^2$, hacmi $\frac{4}{3}\pi r^3$ 'tür. Burada π için PI sabiti tanımlayacağız ve r değerini kullanıcıdan girdi olarak alacağız.

ÖRNEK 1.3 :

```
1. // YARI ÇAPINI KULLANICIDAN ALINAN KÜRENİN YÜZEY ALANI VE HACMİNİ  
   HESAPLAYAN PROGRAM  
2.  
3. #include <stdio.h>  
4.  
5. #define PI 3.14 //define deyimi ile sabit tanımlama  
6.  
7. int main ()  
8. {  
9.     float yaricap ;  
10.    float alan , hacim;  
11.  
12.    printf("%s\n","Kürenin alan ve hacmini hesaplamak için yarıçap giriniz:");  
13.    scanf("%f",&yaricap);  
14.  
15.    alan = 4 * PI * yaricap * yaricap; //yarıçapa göre alan hesaplar  
16.    hacim= 4/3 * PI * yaricap * yaricap * yaricap; // yarıçapa göre hacim  
    hesaplanır  
17.  
18.    printf("Girilen yarıçapa göre kürenin yüzey alanı:%.2f\n",alan); //kürenin  
    yüzey alanını yadırır  
19.    printf("Girilen yarıçapa göre kürenin hacmi:%.2f\n",hacim); //kürenin hacmini  
    yazdırır  
20.  
21.    return 0;  
22.  
23. }    //main fonksiyonu sonu
```

ÇIKTI:

```
[v4gus@parrot]-[~/Desktop/c_dersleri/0-1.bölümler]
→ ./a.out
Kürenin alan ve hacmini hesaplamak için yarıçap giriniz:
4.56
Girilen yarıçapa göre kürenin yüzey alanı:261.17
Girilen yarıçapa göre kürenin hacmi:297.73
[v4gus@parrot]-[~/Desktop/c_dersleri/0-1.bölümler]
→ $
```

1.5 OPERATÖRLER

1.5.0 ARİTMETİKSEL OPERATÖRLER

toplama	+
çıkarma	-
çarpma	*
bölme	/
mod alma	%

5 kişilik bir sınıfın not ortalamasını hesaplayıp monitöre yazdıran bir C programı yazalım.

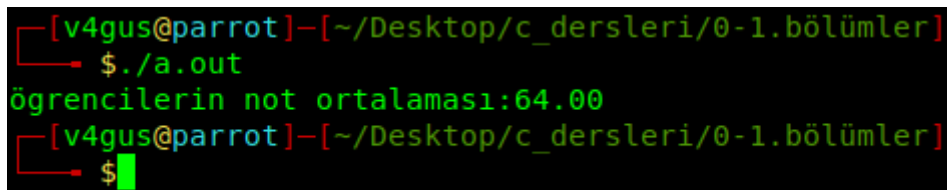
ÖRNEK 1.4:

1. //BEŞ KİŞİLİK SINIFIN NOT ORTALAMASINI BULMA
- 2.
3. #include <stdio.h>
- 4.
5. int main()//main fonksiyonunun başlangıcı
6. {
7. int ogrenci1=100;
8. int ogrenci2=20;

```
9.      int ogrenci3=90;
10.     int ogrenci4=65;
11.     int ogrenci5=45;
12.
13.     float ortalama = (ogrenci1+ogrenci2+ogrenci3+ogrenci4+ogrenci5)/5;
        //ortalama deęerine 5 ğrencinin notları toplamının 5'e bln ata
14.
15.     printf("ğrencilerin not ortalaması:%.2f\n",ortalama );//ortalama deęerini
        yazdır
16.
17.     return 0;
18. }    //main fonksiyonu sonu
19.
```

Kodumuzda yine farklı bir deyim olan "%f" ile float deęişkenleri okumak iin kullanılır. ncesinde kullandığımız ".2" ise noktadan sonra iki basamak yazdır demektir ve opsiyoneldir.

IKTI:



```
[v4gus@parrot]~/. Desktop/c_dersleri/0-1.blmler]
- ./a.out
ğrencilerin not ortalaması:64.00
[v4gus@parrot]~/. Desktop/c_dersleri/0-1.blmler]
- $
```

1.5.1 KARŞILAŞTIRMA OPERATÖRLERİ

Karşılaştırma operatörleri karşılaştırma işlemi yaparak doğru (true) veya yanlış (false) deęeri döndürür. Genellikle koşullu ifadelerle birlikte kullanılır.

byk m ?	>
kk m ?	<
byk veya eřit mi?	>=
kk veya eřit mi?	<=

bu operatrler ařaęıdakilerden ncelik sırasında daha nceliklidir.

eşit mi?	= =
farklı mı ?	! =

1.5.2 MANTIKSAL OPERATÖRLER

Bu operatörler de karşılaştırma operatörleri gibi koşullu ifadelerde veya döğülerde kullanılır.

VE işlemi (AND)	&&
VEYA işlemi (OR)	
DEĞİL işlemi (NOT)	!

1.5.3 ATAMA OPERATÖRLERİ

Bir değişkenin içerisine herhangi bir değeri yüklemek için atama operatörleri kullanılır. Çeşitli türleri vardır.

OPERATÖR	ANLAMI
=	sağdaki değeri soldaki değişkene ata
- =	sağdaki değeri soldakinden çıkar ve soldaki değişkene ata
+ =	sağdaki değeri soldakinden topla ve soldaki değişkene ata
* =	sağdaki değeri soldaki ile çarp ve soldaki değişkene ata
/ =	sağdaki değeri soldakine böl ve soldaki değişkene ata
% =	sağdaki değerin soldaki değeri kadar modunu al ve sağdaki değişkene ata

Tüm bu değişkenleri kullanarak iki değişken ile farklı türde işlemler yapan bir program yazalım.

ÖRNEK 1.5:

1. //ATAMA İŞLEMLERİ
- 2.
3. #include <stdio.h>

```
4.
5. int main() //main fonksiyonu başlangıcı
6. {
7.     int a=25 , b=25 ;
8.
9.     printf("a değişkeninin ilk değeri :%d\nb değişkeninin ilk değeri :%d\n",a,b); //
    a ve b değişkenini yazdır
10.
11.     a -=5; // a değişkeninden 5 çıkar ve a değişkenine sonucu ata
12.
13.     printf("a -=5 atamasından sonra a değişkeni :%d\n",a ); // a değerini yazdır
14.
15.     a +=10; // a değişkenine 10 ekle ve a değişkenine sonucu ata
16.
17.     printf("a +=10 atamasından sonra a değişkeni:%d\n",a ); // a değerini yazdır
18.
19.     a *=2; // a değişkenini 2 ile çarp ve a değişkenine sonucu ata
20.
21.     printf("a *=2 atamasından sonra a değişkeni:%d\n",a ); // a değerini yazdır
22.
23.     a /=b; //a değişkenini b değişkenine böl ve sonucu a değişkenine ata
24.
25.     printf("a /=b atamasından sonra a değişkeni:%d\n",a ); // a değerini yazdır
26.
27.     a %=b; //a değişkeninin b değişkeni kadar modunu al ve sonucu a
    değişkenine ata
28.
29.     printf("a %=b atamasından sonra a değişkeni:%d\n",a ); // a değerini
    yazdır
30.
31.
32.     return 0;
33. } //main fonksiyonu sonu
```

Kodumuzda tüm atama operatörlerini kullandık (5.satır) ve ilk değeri 25 olan integer türünde a ve b değişkenlerini ilk önce değerlerini yazdırdık (7. satır) . 9. satırda a değişkeninin değerini 5 azaltıp oluşan değeri tekrar a değişkenine atadık ve 11. satırda yazdırdık. 13. satırda a değişkeninin değerine 10 ekleyip tekrar oluşan değeri a tekrar a değişkenine atadık ve 15. satırda a

değişkenini yazdırdık. 17. satırda a değişkeninin değerini 2 ile çarpıp tekrar a değişkenine atadık ve 19. satırda yazdırdık. 21. satırda a değişkeninin değerini b değişkeninin değerine bölüp a değişkenine atadık ve 23. satırda a değişkenini yazdırdık. 25. satırda a değişkeninin değerinin b değişkeninin değerine göre modunu aldık ve sonucu a değişkenine atadık. 27. satırda a değişkenini yazdırdık ve işletim sistemine programımızın başarıyla yürütüldüğünü söyleyip programımızı sonlandırdık.

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/0-1.bölümler]
└─$ ./a.out
a değişkeninin ilk değeri :25
b değişkeninin ilk değeri :25
a -=5 atamasından sonra a değişkeni :20
a +=10 atamasından sonra a değişkeni:30
a *=2 atamasından sonra a değişkeni:60
a /=b atamasından sonra a değişkeni:2
a %=b atamasından sonra a değişkeni:2
[v4gus@parrot]--[~/Desktop/c_dersleri/0-1.bölümler]
└─$
```

1.5.4 ARTTIRMA VE AZALTMA OPERATÖRLERİ

C dilinde değişkenlerin artırılması ya da azaltılması için çeşitli yöntemler kullanılsa da sıkça olarak karşımıza çıkacak olan artırma (increment) ve azaltma (decrement) operatörleri vardır.

++ / -- variable : Önceden artırma / eksiltme operatörüdür. Variable değişkeninin 1 artırır / eksiltir . Yani variable = variable + / - 1 ; işlemini yapar.

variable ++ / -- : Sonradan artırma / azaltma operatörüdür. Variable değişkeninin 1 artırır / azaltır. Yani variable = variable + / - 1 ; işlemini yapar.

İlk bakışta iki operatör de aynı işlemi yapıyormuş gibi görünse de aralarında önemli bir fark vardır ; önceden artırma ve azaltma operatörleri kullanıldığı değişken herhangi bir işlemde kullanılıyorsa (yazdırma , toplama ... vb) önce değişkenin değeri artırılır ya da azaltılır sonrasında işlem yapılır. Sonradan artırma ve azaltma operatörleri ise önce değişkenin değeri ile işlem (yazdırma , toplama ... vb) yapılır sonrasında artırılır ya da azaltılır.

Integer türünde sayi1 ve sayi2 değerlerini 5 olarak tanımlayıp önce sayi1 değişkeninin

değerini önceden artırıp , sayi2 değişkeninin değerini sonradan artırıp yazdıralım.Sonrasında değişkenlerimizin değerlerini tekrar yazdıralım.

ÖRNEK 1.6:

```
1. // ARTTIRMA OPERATÖRLERİ ARASINDAKİ FARK
2.
3. #include <stdio.h>
4.
5. int main (void) //main fonksiyonu başlangıcı
6. {
7.     int sayi1=5;
8.     int sayi2=5;
9.
10.    printf("sayi1:%d\nsayi2:%d",sayi1++,++sayi2);
11.    printf("sayi1:%d\nsayi2:%d",sayi1,sayi2);
12.
13.    return 0;
14. } // main fonksiyonu sonu
```

Bu program bize açıkça iki durum arasındaki farkı gösteriyor. Programımız başlangıçta integer türünde sayi1 ve sayi2 isimli iki değişkene 5 değerini atıyor.Sonrasında sayi1 değişkenini yazdırıyor (sayi1=5) . Sayi1 değişkenini önceden artırıp sayi1 değişkeninde meydana gelen değişikliği görebilmemiz için tekrar yazdırıyoruz(sayi1=6). Son olarak sayi1'in son halini yazdırıyoruz(sayi1=5). Sayi2 değişkeninin ilk durumunu yazdırıyoruz (sayi2=5).Sayi2 değişkenini sonradan artırıp sayi2 değişkeninde meydana gelen değişikliği görebilmemiz için tekrar yazdırıyoruz(sayi2=5). Son olarak sayi2'nin son halini yazdırıyoruz (sayi2=6).

2. PROGRAM DENETİM YAPILARI

Programlama yaparken genellikle koşullar ve karşılaştırmalar yapmamız gerekcek denetim yapıları bize bu işlemleri yapma imkanı verir.

2.0 DEYİM (STATEMENT) NEDİR ?

C programlama dilinde ifadeler birbirinden noktalı virgül (terminator / sonlandırıcı) ile ayrılırlar.

Deyim (statement) ifade ve terminatordan oluşan gurubun adıdır.

mesela:

$x = y / z - q$ bir ifadedir fakat ;

$x = y / z - q$; bir deyimdir.

Genel olarak 4 farklı deyim türü vardır.

1- yalın deyim : bir ifadenin sonuna noktalı virgül getirilmesiyle oluşturulan deyimlerdir.

```
num=10+20;
```

```
value++;
```

gibi

2- bileşik deyimler (compound statement) : birden fazla deyim bir blok içerisinde toplanması ile oluşan deyimlerdir.

Bu açıdan her blok aslına birleşik deyimdir.

```
{  
    num=10+70 ;  
    value --;  
}
```

gibi

3- bildirim deyimleri : bildirim için oluşturulan deyimlerdir.

```
int number;
```

```
float temp;
```

```
char c;
```

gibi

4- kontrol deyimleri : Program akışının kontrolünü sağlayan deyimlerdir.

```
if (ifade) {
```

```
        deyimler
    }
else
    {
        deyimler
    }
```

veya

```
switch (number) {
    case1:
        deyimler;
        break;

    case2:
        deyimler;
        break;
default
    deyimler;
}
```

gibi. Şimdi kontrol deyimlerini daha detaylı inceleyelim.

2.1 İF DEYİMİ

"if" eğer demektir. Kullanımı sırasında bir koşulla karşılaştırma yapar.

Genel kullanım biçimi :

```
if(ifade)
    deyim1
```

....

şeklindedir.

If deyimi çalıştırılma aşamasında , compiler önce parantez içindeki ifadeyi sayısal olarak hesaplar. Bu sayısal değeri doğru (true) veya yanlış (false) olarak yorumlar.

Basit bir şekilde kullanıcının aldığı iki sayının eşit olup olmadığını karşılaştıran ve sonucu yazdıran bir program yazalım.

ÖRNEK 2.1:

```
1.
2. //KULLANICININ GİRDİĞİ İKİ SAYININ EŞİTLİĞİNİ KARŞILAŞTIRAN PROGRAM
3.
4.
5. #include <stdio.h>
6.
7. int main() //main fonksiyonu başlangıcı
8. {
9.     int sayi1;
10.    int sayi2;
11.
12.    printf("%s\n","Lütfen karşılaştırmak istediğiniz iki tam sayı giriniz:" );
13.
14.    scanf("%d%d",&sayi1,&sayi2); // sayi1 ve sayi2 değişkenlerini kullanıcıdan
    girilmesi
15.
16.    if(sayi1==sayi2) // iki değerin eşitliğinin kontrolünü yap
17.    {
18.        printf("%s\n","Girilen değerler eşit");
19.    }
20.
21.    else{
22.        printf("%s\n","Girilen değerler eşit değil");
23.    }
24.
25.    return 0;
26.
27.} //main fonksiyonu sonu
28.
```

13. satırda "scanf" fonksiyonunu arka arkaya iki değer okuması için "scanf("%d%d",&

deger1,°er2)" şeklinde kullandık.

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
└─ $./a.out
Lütfen karşılaştırmak istediğiniz iki tam sayı giriniz:
70
99
Girilen değerler eşit değil
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
└─ $
```

2.2. İF - ELSE DEYİMİ

"if" koşul ifadesi , koşul istenilen durumda olduğunda işlemleri yaparken ; "else" ifadesi ise koşulun istenmediği durumda olmadığında yapılacak işlemleri tanımlayabilmemiz açısından yalnız "if" ifadesine göre daha kullanışlıdır. Genel kullanım biçimi:

if(kontrol ifadesi)

```
{
    işlemler
    deyimler
}
```

else

```
{
    işlemler
    deyimler
}
```

şeklinde dir. Daha iyi kavrayabilmek amacıyla kullanıcıdan girilen vizenin %20'si ile finalin %80'ini ortalama 49.000 'a eşit veya küçük ise kaldınız , büyükse geçtiniz yazdıran programı yazalım.

ÖRNEK 2.2:

1. //GİRİLEN VİZE VE FINAL DEĞERİNDEN HARF NOTU HESAPLAYAN PROGRAM
- 2.
3. #include <stdio.h>

```
4.
5. int main ()
6. {
7.     int vize;
8.     int final;
9.     float not;
10.
11.     printf("%s\n","Vize notunu giriniz:"); //kullanıcıdan vize değerini al
12.     scanf("%d",&vize);
13.
14.     printf("%s\n","Final notunu giriniz:"); //kullanıcıdan final değerini al
15.     scanf("%d",&final);
16.
17.     not =(vize * 0.2) + (final * 0.8); //vizenin %20'si ile finalin %80'ni topla
18.
19.     if(not <= 49.999)
20.         printf("Not ortalamanız : %.3f kaldınız.\n",not);
21.
22.     else
23.         printf("Not ortalamanız : %.3f geçtiniz.\n",not);
24.
25.     return 0;
26.
27. } //main fonksiyonu sonu
28.
```

Dikkat ederseniz 19. satırda karşılaştırdığımız değişken float olduğu için hata ile karşılaşmamak için hassasiyeti virgülden sonra 3 basamağa çıkardık. Eğer 49.000 ile karşılaştırma yapsaydık 49.001 ortalamaya sahip bir öğrenci bu programdan "geçtiniz" sonucu alacaktı. Float ve double karşılaştırması yapılırken basamak hassasiyetine dikkat edilmelidir.

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$ ./a.out
Vize notunu giriniz:
40
Final notunu giriniz:
90
Not ortalamanız : 80.000 geçtiniz.
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$
```

2.3. ELSE İFDEYİMİ

Bazen birden fazla ihtimalin değerlendirilmesi gerekir. Bu durumlarda sürekli "if" kullanmak programımızın çalışma hızını azaltacak ve çalışma hızının azalmasına neden olacaktır. Bu durumun önüne geçmek için elseif deyimini kullanabiliriz.

Genel kullanım yapısı

```
if(kontrol ifadesi)
{
    işlemler
    deyimler
}
```

```
else if(kontrol ifadesi)
{
    işlemler
    deyimler
}
```

```
else if(kontrol ifadesi)
{
    işlemler
    deyimler
}
```

```
else
{
    işlemler
    deyimler
}
```

şeklindedir.

Else if deyimini bir durum üzerinden inceleyelim. Mesela bir X sayısının 0 ile olan karşılaştırılması sırasında oluşabilecek kontrol edelim:

1. durum 0 'a eşittir ;

2. durum 0 'dan büyüktür ;

3.durum 0'dan küçüktür .

Bu durumu önce if kullanarak sonra da elseif kullanarak değerlendiren bir program yazalım.

ÖRNEK 2.3:

1. //X DEĞERİNİN 0 'A GÖRE OLASI İHTİMALLERİNİN DEĞERLENDİRİLMESİ

2.

3. #include <stdio.h>

4.

5. int main() //main fonksiyonu başlangıcı

6. {

7. int X ;

8.

9. printf("%s\n"," Lütfen karşılaştırma yapılacak olan X tam sayı değerini
giriniz:");//X değerini kullanıcıdan al

10.

11. scanf("%d",&X);

12.

13. if(X==0)// X = 0 olma durumu

14. printf("girdiğiniz X %d değeri 0 'a eşittir\n", X);

15.

16. if(X>0) // X > 0 olma durumu

17. printf("girdiğiniz X %d değeri 0 'dan büyüktür\n", X);

18.

19. if(X<0) // X < 0 olma durumu

20. printf("girdiğiniz X %d değeri 0'dan küçüktür\n", X);

21.

22. return 0;

23.

24. }//main fonksiyonu sonu

25.

19. satırda yani son kontrol kısmında else yerine if kullanmamızın nedeni burada da durum kontrolü yapmaktır. Eğer else ile yazmış olsaydık X=0 için hem X==0 hem kontrolü de else çalışacaktı.

ÇIKTI:

```
[v4gus@parrot]-[~/Desktop/c_dersleri/2-3. bölümler]
→ ./a.out
    Lütfen karşılaştırma yapılacak olan X tam sayı değerini giriniz:
70
girdiğiniz X 70 değeri 0 'dan büyüktür
[v4gus@parrot]-[~/Desktop/c_dersleri/2-3. bölümler]
→ $
```

Şimdi aynı işlemleri elseif yapısını kullanarak yapalım.

ÖRNEK 2.4:

```
1.
2. // X DEĞERİNİ 0 İLE KARŞILAŞTIRAN PROGRAM
3.
4. #include <stdio.h>
5.
6. int main () //main fonksiyonu başlangıcı
7. {
8.
9.     int X;
10.
11.     printf("%s\n","Lütfen karşılaştırma yapılacak olan X tam sayısını giriniz:");
12.
13.     scanf("%d",&X); // X değerini oku
14.
15.     if(X==0)// X=0 durumu
16.         printf("X in değeri %d , 0'a eşittir \n", X);
17.
18.     else if(X>0) // X > 0
19.         printf("X in değeri %d , 0'dan büyüktür \n", X);
20.
21.     else
22.         printf("X in değeri %d , 0'dan küçüktür\n", X);
23.
24.     return 0;
25.
26.} //main fonksiyonu sonu
```

27.

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$ ./a.out
Lütfen karşılaştırma yapılacak olan X tam sayısını giriniz:
70
X in değeri 70 , 0'dan büyüktür
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$
```

Else if yapısının avantajı tüm koşulların dentlenmesi yerine koşullardan birisi sağlandığında kontrol etmeyi bırakmasıdır.

Öğrencinin vize notunun %40 , final notunun %60 ile oluşan ortalamaya göre harf notunu :

eğer not 90 'dan büyük veya eşitse A
eğer not 75 'dan büyük veya eşitse B
eğer not 60 'dan büyük veya eşitse C
eğer not 50 'den büyük veya eşitse D
değilse F
olarak yazdıran programı yazalım

ÖRNEK 2.5:

```
1. // ÖĞRENCİNİN HARF NOTUNU HESAPLAYAN PROGRAM
2.
3. #include <stdio.h>
4.
5. int main ()//main fonksiyonu başlangıcı
6. {
7.     int vize ;
8.     int final;
9.     float not;
10.
11.     printf("%s\n","lütfen vize notunuzu giriniz:");
12.
13.     scanf("%d",&vize);
14.
15.     printf("%s\n","lütfen final notunuzu giriniz:");
16.
17.     scanf("%d",&final);
18.
19.     not=(vize*0.4) + (final*0.6);
```

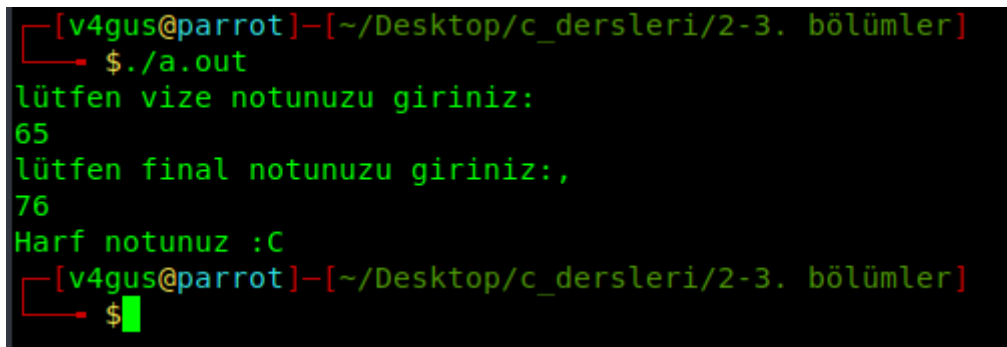
```

20.
21.     if(not >=90.00)
22.         printf("%s\n","Harf notunuz :A" );
23.
24.     else if(not >= 75)
25.         printf("%s\n","Harf notunuz :B" );
26.
27.     else if(not >= 60)
28.         printf("%s\n","Harf notunuz :C" );
29.
30.     else if(not >= 50)
31.         printf("%s\n","Harf notunuz :D" );
32.
33.     else
34.         printf("%s\n","Harf notunuz :F KALDINIZ !" );
35.
36.     return 0;
37.
38. } // main fonksiyonu sonu
39.

```

Programımızda float karşılaştırması yaptığımız için yine hassasiyeti virgülden sonraki üç basamak olarak ayarladık.

ÇIKTI:



```

[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$ ./a.out
lütfen vize notunuzu giriniz:
65
lütfen final notunuzu giriniz:,
76
Harf notunuz :C
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$

```

2.4 SWITCH - CASE DEYİMİ

İf deyimi true veya false olarak iki farklı seçeneğe sahiptir. Switch deyimi ise ifadenin çeşitli şekilde sayısal değerlerine göre farklı işlemlerin yapılması için kullanılır. Genel kullanım biçimi:

```

switch(ifade)
{
    case<sabit ifade>:
        işlemler;
        deyimler;

    break;

    case<sabit ifade>:
        işlemler;

```

```

        deyimler;
break;

        case<sabit ifade>:
            işlemler;
            deyimler;
break;

default:
    hiçbirini karşılamazsa yapılacak olan işlemler;
    deyimler;
}

```

şeklindedir.

Switch , case ve default anahtar sözcüklerdir. Compiler switch parantezi içerisindeki ifadeyi sayısal olarak hesaplayıp eşit olan case ifadesi bulmaya çalışır.Eğer bulursa akış oraya yönelir , bulamazsa default ile belirtilen yerden akış devam eder.

```

switch(3)
{
    case 1:
        printf("Edward Snowden\n");
        break;

    case 2:
        printf("Julian Assange\n");
        break;

    case 3:
        printf("Kevin Mitnick\n");
        break;

    default:
        printf("Kevin Poulsen\n");
}

```

Bu program Kevin Mitnick çıktısını verir.

Switch yapısını kullanarak dört işlemli basit bir hesap makinesi programı yazalım.

ÖRNEK 2.6:

```

1. // HESAP MAKİNESİ
2.
3. #include <stdio.h>
4.
5. int main ()//main fonksiyonu başlangıcı
6. {
7.     int sayi1;

```

```

8.      int sayi2;
9.      int secim;
10.
11.
12.      printf("%s\n","1-TOPLAMA\n2-ÇIKARMA\n3-ÇARPMA\n4-BÖLME\n0-ÇIKIŞ\
n\nLütfen yapmak istediğiniz işlemi giriniz:");
13.
14.      scanf("%d",&secim);
15.
16.
17.
18.      switch(secim)
19.      {
20.          case(0): // seçim 0 ise
21.              {
22.                  printf("%s\n","Çıkış yapılıyor...");
23.
24.                  return 0;
25.              }
26.
27.              break;
28.
29.          case(1): // seçim 1 ise
30.          {
31.              printf("%s\n","Lütfen işle yapmak istediğiniz iki tam sayı değerini
giriniz:" );
32.
33.              scanf("%d%d",&sayi1,&sayi2);
34.
35.              printf("Sonuc:%d",sayi1+sayi2 );
36.          }
37.
38.          break;
39.
40.          case(2): // seçim 2 ise
41.          {
42.              printf("%s\n","Lütfen işle yapmak istediğiniz iki tam sayı değerini
giriniz:" );
43.
44.              scanf("%d%d",&sayi1,&sayi2);
45.
46.              printf("Sonuc:%d",sayi1-sayi2 );
47.          }
48.
49.          break;
50.
51.          case(3): // seçim 3 ise
52.          {
53.              printf("%s\n","Lütfen işle yapmak istediğiniz iki tam sayı değerini
giriniz:" );
54.

```

```

55.             scanf("%d%d",&sayi1,&sayi2);
56.
57.             printf("Sonuc:%d",sayi1*sayi2 );
58.         }
59.
60.         break;
61.
62.         case(4):// seçim 4 ise
63.         {
64.             printf("%s\n","Lütfen işle yapmak istediğiniz iki tam sayı değerini
    giriniz:" );
65.
66.             scanf("%d%d",&sayi1,&sayi2);
67.
68.             printf("Sonuc:%.3f",sayi1/sayi2 );
69.         }
70.
71.         break;
72.
73.         default: // geçersiz bir değer girilirse
74.             printf("%s\n","Geçersiz değer girildi\nÇıkış yapılıyor" );
75.             return 0;
76.
77.     }
78.
79.     return 0;
80.
81. } // main fonksiyonu sonu
82.

```

Yazdığımız program basit bir menü oluşturup

1-Toplama
 2-Çıkarma
 3-Çarpma
 4-Bölme
 0-Çıkış

kullanıcıdan yapmak istediği işlemi girmesini istiyor ve gelen değere göre ;

0 değeri için "Çıkış yapılıyor..." yazdırıp return 0 komutu ile programı sonlandırıyor.

1 değeri için toplama işlemi yapmak için tanımladığımız değerleri istiyor ve toplamalarını yazdırıyor;

2 değeri için çıkarma işlemi yapmak için tanımladığımız değerleri istiyor ve farklarını yazdırıyor;

3 değeri için çarpma işlemi yapmak için tanımladığımız değerleri istiyor ve çarpımlarını yazdırıyor;

4 değeri için bölme işlemi yapmak için tanımladığımız değerleri istiyor ve sayi1/sayi2 işleminin sonucunu yazdırıyor;

Eğer girilen değer tanımlı bir değer değilse "Geçersiz bir değer girildi.Çıkış yapılıyor" yazdırıp return 0 komutu ile programı sonlandırıyor.

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$ ./a.out
1-TOPLAMA
2-ÇIKARMA
3-ÇARPMA
4-BÖLME
0-ÇIKIŞ

Lütfen yapmak istediğiniz işlemi giriniz:
3
Lütfen işle yapmak istediğiniz iki tam sayı değerini giriniz:
7
10
Sonuc:70 [v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$
```

3. DÖNGÜLER

Programın belirli bölümlerinin yinelemeli olarak çalıştırılmasını sağlayan deyimlere döngü denir. Sürekli tekrarlanan bir işlemi döngü ile yapmak kod hamallığından kurtarır ve daha okunabilir kodlar yazmamıza olanak sağlar.

Temel Olarak 2 tür döngü vardır ;

1) For döngüleri

2) While döngüleri

For döngüleri belirli sayıda yineleme oluşturmak için kullanılır. While döngüleri de aynı işlevi koşulun doğru olmasına bağlı olarak yapar. Bu açıdan istisnai durumları gözardı ederek for ile yazılan döngüleri while ile yazmak mümkündür diyebiliriz.

3.0 While Döngüsü

Çalışma prensibi parantez içerisindeki ifade doğru (true) ise dön şeklindedir. İçerisinde istenilen döngü tekrarlandığında çıkış yapabilmek için if kontrolü ile birlikte kullanılabilir.

Kullanımı :

```
while(koşul)
{
    işlemler;
    ifadeler;
}
```

şeklindedir.

Eğer birisi bizden ismimizi 100 defa yazdıran bir program yazmamızı isteseydi , çözüm için

iki yolumuz olacaktı;

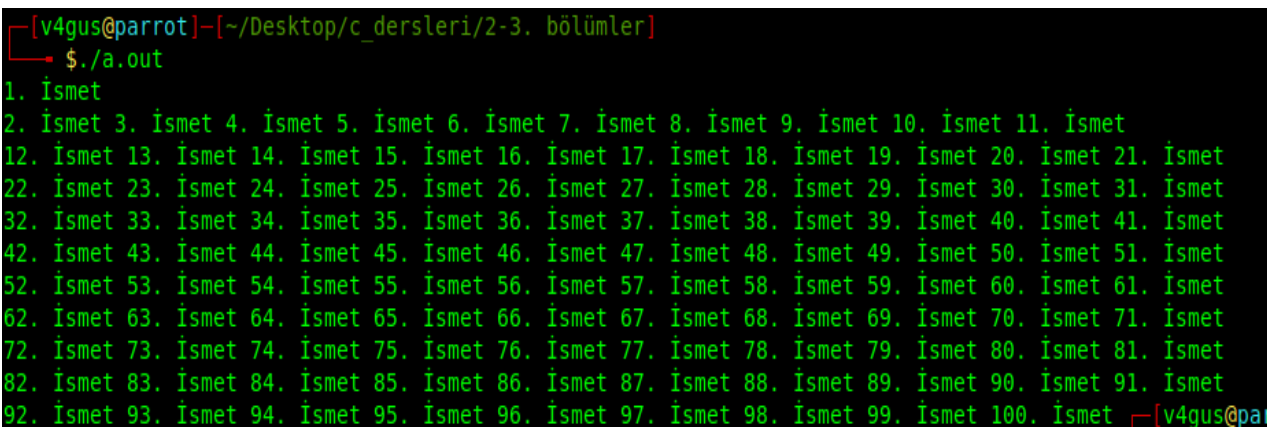
1-100 ayrı printf ile yazmak , bir printf içerisinde 100 kere yazmak (bu durumları okurken bile can yakıyor).

2-döngü kullanarak sadece 1 printf ile işlemi yapmak.

Döngülerde sayaç dediğimiz sadece dönu sayısını saymaya yarayan değişkenler vardır. Bunlar integer türünde herhangi bir değişken gibi tanımlanırlar. Kullanımları da arttırma ya da azaltma operatörleri ile olur. Hem döngü kavramının hem de sayaç kavramının daha iyi anlaşılabilmesi için ismimizi 100 kere yazan bir program yazalım.

ÖRNEK 3.1:

```
1. //WHILE DÖNGÜSÜ İLE 100 KERE İSİM YAZDIRAN PROGRAM
2.
3. #include <stdio.h>
4.
5. int main ()//main fonksiyonu başlangıcı
6. {
7.     int sayac = 0; //sayaç tanımlaması
8.
9.     while(sayac!=100)//sayac 100 'e eşit değilse döngüyü sürdür
10.    {
11.        printf("%d. İsmet ",sayac+1); // sayac değişkenini ve İsmet yazdır
12.
13.        if(sayac%10==0)
14.            printf("\n");
15.
16.        sayac++;
17.    }
18.
19.    return 0;
20.
21. } //main fonksiyonu sonu
```



```
[v4gus@parrot]~[~/Desktop/c_dersleri/2-3. bölümler]
$ ./a.out
1. İsmet
2. İsmet 3. İsmet 4. İsmet 5. İsmet 6. İsmet 7. İsmet 8. İsmet 9. İsmet 10. İsmet 11. İsmet
12. İsmet 13. İsmet 14. İsmet 15. İsmet 16. İsmet 17. İsmet 18. İsmet 19. İsmet 20. İsmet 21. İsmet
22. İsmet 23. İsmet 24. İsmet 25. İsmet 26. İsmet 27. İsmet 28. İsmet 29. İsmet 30. İsmet 31. İsmet
32. İsmet 33. İsmet 34. İsmet 35. İsmet 36. İsmet 37. İsmet 38. İsmet 39. İsmet 40. İsmet 41. İsmet
42. İsmet 43. İsmet 44. İsmet 45. İsmet 46. İsmet 47. İsmet 48. İsmet 49. İsmet 50. İsmet 51. İsmet
52. İsmet 53. İsmet 54. İsmet 55. İsmet 56. İsmet 57. İsmet 58. İsmet 59. İsmet 60. İsmet 61. İsmet
62. İsmet 63. İsmet 64. İsmet 65. İsmet 66. İsmet 67. İsmet 68. İsmet 69. İsmet 70. İsmet 71. İsmet
72. İsmet 73. İsmet 74. İsmet 75. İsmet 76. İsmet 77. İsmet 78. İsmet 79. İsmet 80. İsmet 81. İsmet
82. İsmet 83. İsmet 84. İsmet 85. İsmet 86. İsmet 87. İsmet 88. İsmet 89. İsmet 90. İsmet 91. İsmet
92. İsmet 93. İsmet 94. İsmet 95. İsmet 96. İsmet 97. İsmet 98. İsmet 99. İsmet 100. İsmet [v4gus@pa]
```

Programda döngümüz zaten 0-99 arasında 100 kere dönmüş oluyor fakat yazdırırken daha iyi görülebilmesi amacıyla sayac değişkenini yazdırırken 1 ekleyip yazdırdık.

Bunun yerine

```
sayac=1;
while(sayac<=100)
{
    işlemler;
    ifadeler;
}
```

şeklinde de kullanabilirdik.Öğrenci sayısını ve notlarını kullanıcıdan alarak sınıfın not ortalamasını bulan pogramı yazalım.

ÖRNEK 3.2:

```
1. //ÖĞRENCİLERİN NOTLARINI KULLANICIDAN ALARAK SINIFIN NOT
   ORTALAMASINI BULAN PROGRAM
2.
3. #include <stdio.h>
4.
5. int main()//main fonksiyonu başlangıcı
6. {
7.     int ogrenci_sayisi;
8.     int toplam_not = 0;
9.     int not;
10.    float ortalama;
11.    int i = 0; // sayaç
12.
13.    printf("%s\n","Lütfen öğrenci sayısını giriniz:");
14.    scanf("%d",&ogrenci_sayisi);
15.
16.    if(ogrenci_sayisi<1) //öğrenci sayısını kontrol et
17.    {
18.        printf("Öğrenci sayısı 1'den küçük olamaz\n");
19.        return 0; //programı sonlandır
20.    }
21.
22.    while(i<ogrenci_sayisi)
23.    {
24.        printf("%d. öğrencinin notunu giriniz:\n",i+1 );
25.        scanf("%d",&not);
26.        toplam_not+=not; //toplam notun üzerine girilen not değerini ekle
27.        i++; // sayacı arttır
28.    }
29.
30.    ortalama=toplam_not/ogrenci_sayisi;
31.
32.    printf("%d. kişilik sınıfın ortalaması %.3f'dir\n",ogrenci_sayisi,ortalama );
33.
34.    return 0;
35.
```

```
36.} // main fonksiyonu sonu
37.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$ ./a.out
Lütfen öğrenci sayısını giriniz:
5
1. öğrencinin notunu giriniz:
65
2. öğrencinin notunu giriniz:
45
3. öğrencinin notunu giriniz:
76
4. öğrencinin notunu giriniz:
45
5. öğrencinin notunu giriniz:
87
5. kişilik sınıfın ortalaması 63.000'dir
[v4gus@parrot]--[~/Desktop/c_dersleri/2-3. bölümler]
$
```

3.1 DO-WHILE DÖNGÜSÜ

While döngüsünden farklı olarak "önce bir kere döngüyü çalıştır sonrasında döngü şartını kontrol et" mantığında çalışır.Kullanımı :

```
do
{
    deyim
}while (kontrol ifadesi);
```

şeklindedir.

Klavyeden girilen tam sayının kaç basamaklı olduğunu do-while döngüsü ile bulan program.

ÖRNEK 3.3:

```
1. //KLAVYEDEN GİRİLEN SAYININ BASAMAK SAYISINI BULAN PROGRAM
2.
3. #include <stdio.h>
4.
5. int main()//main fonksiyonu başlangıcı
6. {
7.     int sayi;
8.     int sayac = 0;
9.
```

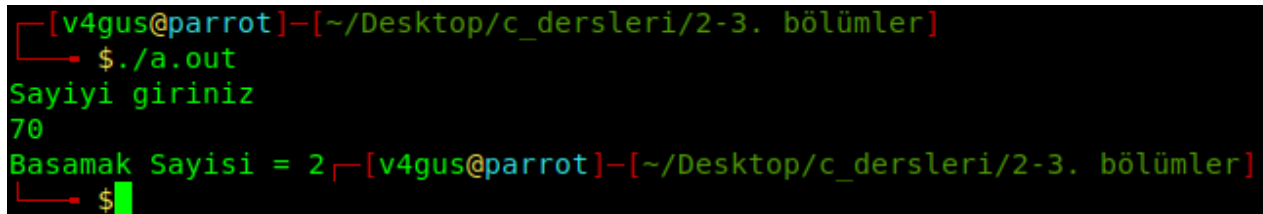
```

10. printf("Sayiyi giriniz\n");
11. scanf("%d", &sayi);
12.
13. do
14. {
15.     sayi /= 10;
16.     sayac++;
17. } while(sayi > 0);
18.
19. printf("Basamak Sayisi = %d", sayac);
20.
21.
22. return 0;
23.
24. } //main fonksiyonu sonu
25.

```

Programımız kullanıcının girdiği tam sayıyı sayi değişkeninde tutuyor , sayac değişkeni ise sayaç olması için 0 'dan başlatıyoruz.Do içerisine geldiğimizde sayıyı 10'a bölüp tekrar sayi değişkenine atıyor ve sayacı 1 arttırıyoruz.Sonrasında while içerisinde sayi değişkeninin 0'dan büyük olup olmadığını kontrol ediyoruz. Döngü sayi 0'dan küçük olana kadar devam ediyor ve girilen sayının basamak sayısını yazdırıyor.

ÇIKTI:



```

[v4gus@parrot]~[~/Desktop/c_dersleri/2-3. bölümler]
└─$ ./a.out
Sayiyi giriniz
70
Basamak Sayisi = 2
[v4gus@parrot]~[~/Desktop/c_dersleri/2-3. bölümler]
└─$

```

3.2 FOR DÖNGÜSÜ

For döngüsü özel bir döngüdür.C++'dan gelen bir özellik sayesinde for döngüsü içerisinde değişken tanımlama yapılabilir de biz C ile programlama yaptığımız için bu özelliği kullanmayacağız.

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

ifade1 : Genellikle döngü değişkenine (sayaç) ilk değer atama yapmak için kullanılır.

ifade2: Döngü kontrolü burada yazılır.

ifade3: Döngü değişkeninin azaltma veya arttırma işlemi yapılır.

```

for ( sayac=0 ; sayac<10 ; sayac++)
{
    işlemler;
    ifadeler;
}

```

şeklinde kullanılabilir.

For döngüsünde ilk kısmı opsiyoneldir.Çünkü döngünün girişinde bir defa işlem görür. Bu atamayı döngünün başında ayrıca deyim olarak yazılabilir.İlk örneğimizi bu şekilde yazalım.

```
sayac=0;
```

```
for ( ; sayac<10 ; sayac++)  
{  
    işlemler;  
    ifadeler;  
}
```

Yine for döngüsünde hiçbir atama , kontrol ve arttırma azaltma işlemi olmacak şekilde tanımlanabilir.

```
for ( ; ; )  
{  
    ifadeler;  
  
    işlemler;  
}
```

Bu kullanım ile sonsuz döngü dediğimiz program devam ettikçe döngünün sürdüğü yapılar oluşturulabilir.

While ile yaptığımız 100 kere isim yazdırma programı for ile yapalım.

ÖRNEK 3.4:

```
1. //FOR DÖNGÜSÜ İLE 100 KERE İSİM YAZDIRAN PROGRAM  
2.  
3. #include <stdio.h>  
4.  
5. int main()//main fonksiyonunun başlangıcı  
6. {  
7.     int i; //sayaç-döngü değişkeni  
8.  
9.     for(i=0;i<100;i++)  
10.    {  
11.        printf("%d. İsmet\t",i+1);  
12.  
13.        if(i%10==0)  
14.            printf("\n");  
15.    }  
16.    return 0;  
17. } //main fonksiyonunun sonu  
18.
```

Programımızda "i" döngü değişkeni ya da sayaçtır. "i" değişkeni 0'dan başlayarak 99 olana kadar isim yazdıran ve sayacı 1 arttırarak döngü kontrolüne gider.

ÇIKTI:

```
[v4gus@parrot]~/Desktop/c_dersleri/2-3. bölümler
$ ./a.out
1. İsmet
2. İsmet      3. İsmet      4. İsmet      5. İsmet      6. İsmet      7. İsmet 8. İsmet      9. İsmet      10. İsmet      11. İsmet
12. İsmet     13. İsmet     14. İsmet     15. İsmet     16. İsmet     17. İsmet     18. İsmet     19. İsmet     20. İsmet     21. İsmet
22. İsmet     23. İsmet     24. İsmet     25. İsmet     26. İsmet     27. İsmet     28. İsmet     29. İsmet     30. İsmet     31. İsmet
32. İsmet     33. İsmet     34. İsmet     35. İsmet     36. İsmet     37. İsmet     38. İsmet     39. İsmet     40. İsmet     41. İsmet
42. İsmet     43. İsmet     44. İsmet     45. İsmet     46. İsmet     47. İsmet     48. İsmet     49. İsmet     50. İsmet     51. İsmet
52. İsmet     53. İsmet     54. İsmet     55. İsmet     56. İsmet     57. İsmet     58. İsmet     59. İsmet     60. İsmet     61. İsmet
62. İsmet     63. İsmet     64. İsmet     65. İsmet     66. İsmet     67. İsmet     68. İsmet     69. İsmet     70. İsmet     71. İsmet
72. İsmet     73. İsmet     74. İsmet     75. İsmet     76. İsmet     77. İsmet     78. İsmet     79. İsmet     80. İsmet     81. İsmet
82. İsmet     83. İsmet     84. İsmet     85. İsmet     86. İsmet     87. İsmet     88. İsmet     89. İsmet     90. İsmet     91. İsmet
92. İsmet     93. İsmet     94. İsmet     95. İsmet     96. İsmet     97. İsmet     98. İsmet     99. İsmet     100. İsmet
[v4gus@parrot]~/Desktop/c_dersleri/2-3. bölümler
$
```

200'den 900'e kadar olan çift sayıların toplamını for dönüğü ile bulup yazdıran programı yazalım.

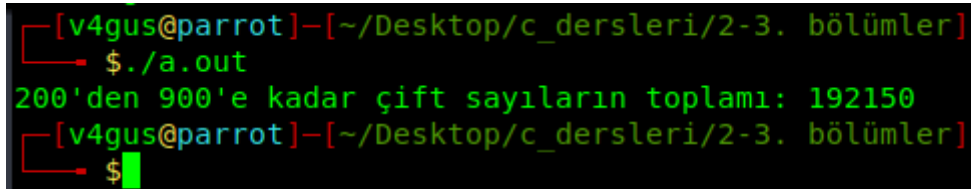
ÖRNEK 3.5:

```
1. //200'DEN 900'E KADAR OLAN ÇİFT SAYILARIN TOPLAMINI BULAN PROGRAM
2.
3. #include <stdio.h>
4.
5. int main() //main fonksiyonu başlangıcı
6. {
7.     int i; // sayaç
8.     int toplam = 0; // toplamı saklayan değişken
9.
10.    for(i=200;i<900;i++)
11.    {
12.        if(i%2==0)
13.            toplam +=i;
14.
15.        else
16.            continue;
17.
18.
19.    }
20.
21.    printf("200'den 900'e kadar çift sayıların toplamı: %d\n",toplam );
22.
23.
24.    return 0;
```

25.} //main fonksiyonu başlangıcı
26.

Programımızda çift sayıları noluabilmek için sayaç değerinin 2'ye göre modunu 0 ise toplama ekledik.16. satırdaki continue deyimi ise o anki değerin atlanmasını sağlar.

ÇIKTI:



```
[v4gus@parrot]~/. Desktop/c_dersleri/2-3. bölümler
$ ./a.out
200'den 900'e kadar çift sayıların toplamı: 192150
[v4gus@parrot]~/. Desktop/c_dersleri/2-3. bölümler
$
```

3.3 BREAK DEYİMİ

Break deyimi döngüleri sonlandırmak için kullanılan bir deyimdir.Kullanımı:

```
for( ; ;)
{
    if(c=='q')
        break;
}
```

```
while(1)
{
    if(c=='q')
        break;
}
```

şeklindedir.

Kullanıcı klavyeden 'q' girilene kadar kullancıdan sürekli karakter girmesini isteyen programı yazalım.

ÖRNEK 3.6:

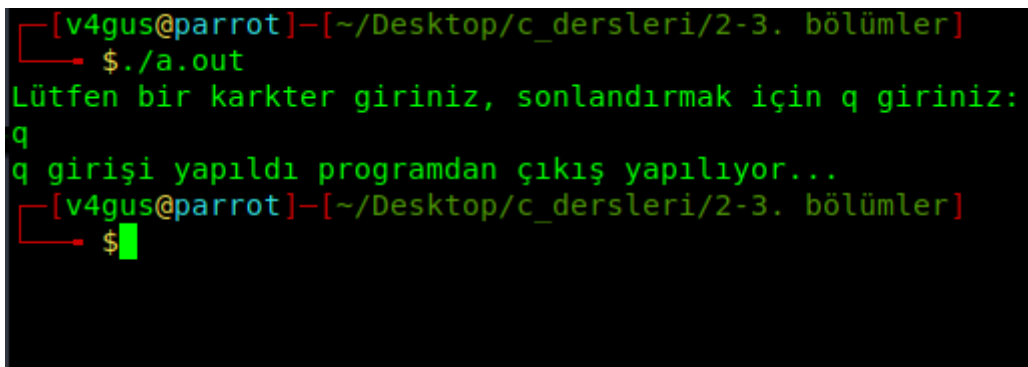
1. //KULLANICI KLAVYEDEN 'q' GİRENE KADAR SÜREKLİ KARAKTER GİRİŞİ İSTEYEN PROGRAM
- 2.
3. #include <stdio.h>
- 4.
5. int main()//main fonksiyonu başlangıcı
6. {
- 7.

```

8.      char c;
9.
10.     for(;;)
11.     {
12.         puts("Lütfen bir karkter giriniz, sonlandırmak için q giriniz:");
13.
14.         scanf("%c",&c);
15.
16.         if(c=='q')
17.         {
18.             printf("%s\n","q girişi yapıldı programdan çıkış yapılıyor..." );
19.             break;
20.         }
21.
22.     }
23.
24.     return 0;
25. } // main fonksiyonu sonu
26.

```

ÇIKTI:



```

[v4gus@parrot]~[~/Desktop/c_dersleri/2-3. bölümler]
└─$ ./a.out
Lütfen bir karkter giriniz, sonlandırmak için q giriniz:
q
q girişi yapıldı programdan çıkış yapılıyor...
[v4gus@parrot]~[~/Desktop/c_dersleri/2-3. bölümler]
└─$

```

4. ÖN İŞLEMÇİ KOMUTLARI

C ön işlemcisi derlenecek olan dosyada diğer dosyaların dahil edilmesi , makro ve sabitlerin tanımlanması programın koşullu derlenmesi gibi görevleri yerine getirir. Tipik olarak kaynak kodları değiştirmek, programı farklı donanımlarda kolaylıkla çalıştırabilmek, programın parçalarını kaynak programda birleştirmek ve derleme sırasındaki bazı uyarı mesajlarını aktif veya pasif hale getirmek için kullanılırlar.

4.0 #include direktifi

Bir başka dosyadan belirli başlık (header) dosyası ekler. İki farklı kullanımı vardır:

#include <dosya_adi> : Standart kütüphane başlıklarını eklemek için kullanılır. Aramasını

önceden tanımlanmış dizinlerde yapar.

#include "dosya_adi": Genellikle programcılar kendilerinin yazdığı header dosyalarını eklemek için kullanılır. Aramasını kaynak kodun bulunduğu dizinde gerçekleştirir.

4.1 #define direktifi (sembolik sabitler ve makrolar)

#define direktifi ile makro ve sembolik sabit tanımlaması yapılır. Ve kaynak kodumuzdaki bir isminin diğer isimle yer değiştirmesini sağlar. Genel olarak kullanımı:

#define tanımlayıcı ifade

şeklindedir.

Hatırlarsanız sabitleri anlatırken kürenin alanı ve hacmini hesaplayan bir programda pi değerini **#define PI 3.14** şeklinde tanımlamıştık.

makro tanımlaması

#define KARE(a) ((a*a))

şeklinde parametre alan bir makro tanımlı olsun. Biz makromuza

b=KARE(2) ;

şeklinde tanımlama ile yer değiştirme yaptığımızda ön işlemci bunu

b=(2*2) ;

şeklinde açar.

4.2 #undef direktifi

Tanımladığımız makro ve sabitleri ön işlemcide geçersiz kılar. kullanımı :

#undef makro_adı

şeklindedir.

#define EKLE(A,B) ((A)+(B))

.

.

.

#undef EKLE(A,B) ((A)+(B))

gibi kullanılabilir.

5. DİZİLER

5.0 TEK BOYUTLU DİZİLER

Dizileri aynı veri türüne sahip olan ve hafızada sıralı biçimde tutulan veri grubu olarak tanımlayabiliriz. Konun pekişmesi için basit olarak bizden yüz kişilik bir sınıfın notlarını tutmamız istenirse ; her öğrenci için teker teker bir tanımlama yapmaktansa bu tanımlamayı bir kere yapıp döngüler yardımı ile dizinin indislerini gezerek notları almak daha mantıklı olacaktır. Dizi tanımlaması

dizinin_türü dizi_adı [eleman_sayısı] ;

şeklinde yapılır.

Dizinin isimlendirilmesinde değişken isimlendirme kuralları geçerlidir. Eğer dizinin elemanları tanımlama sırasında atanmıyorsa eleman sayısı belirtilmelidir. Fakat elemanlar dizi tanımlanırken atanıyorsa eleman sayısı belirtilmesi zorunlu değildir.

dizi [] ;

tanımlaması hatalı bir tanımlamayken

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

tanımlaması doğru bir tanımlamadır.

Dizilerin hafızada görünümünü şematize edelim:

Dizi şeması

burada bir dizinin 0 indisinden başladığını ve bellekteki değerine erişmek için indis değerlerinin kullanılabileceğini anlıyoruz.

ÖRNEKLER

1.) 5 elemanlı bir diziye öncelikle for döngüsü ile her elemanına 10 değeri atayıp sonra kullanıcıdan aldığı değerleri diziye yerleştiren ve dizinin son halini yazdıran programı yazalım

ÖRNEK 5.1:

1. // 5 ELEMANLI BİR DİZİ TANIMLAYIP ÖNCE TÜM ELEMANLARINI 0 ATAYAN SONRA KULLANICIDAN ALAN PROGRAM
- 2.
3. #include <stdio.h>
4. #include <stdlib.h>
5. int main()//main fonksiyonu başlangıcı
6. {
- 7.
8. int dizi[4];

```

9.     int i;//sayaç ve indis
10.
11.     for(i=0;i<5;i++)
12.         printf("%d. eleman : %d\n",i+1,dizi[i]);
13.
14.     printf("%s\n","Tüm elemanlara 0 değeri ataması yapılıyor");
15.
16.     for(i=0;i<5;i++){
17.
18.         dizi[i]=0;
19.
20.         if(i==4)
21.             printf("%s\n","Atma tamamlandı" );
22.
23.     }
24.
25.
26.
27.     for(i=0;i<5;i++)
28.         printf("%d. eleman : %d\n",i+1,dizi[i]);
29.
30.     printf("%s\n","Lütfen 5 tane tam sayı giriniz:" );
31.
32.     for(i=0;i<5;i++)
33.         scanf("%d",&dizi[i]);
34.
35.     printf("%s\n","Girdiğiniz tam sayı değerleri" );
36.
37.     for(i=0;i<5;i++)
38.         printf("%d. eleman : %d\n",i+1,dizi[i]);
39.
40. return 0;
41.
42. } //main fonksiyonu sonu
43.

```

ÇIKTI:

```

1. eleman : -752509792
2. eleman : 32767
3. eleman : 0
4. eleman : 0
5. eleman : 0
Tüm elemanlara 0 değeri ataması yapılıyor
Atma tamamlandı
1. eleman : 0
2. eleman : 0
3. eleman : 0
4. eleman : 0
5. eleman : 0
Lütfen 5 tane tam sayı giriniz:
5
3
4
56
87
Girdiğiniz tam sayı değerleri
1. eleman : 5
2. eleman : 3
3. eleman : 4
4. eleman : 56
5. eleman : 87
...Program finished with exit code

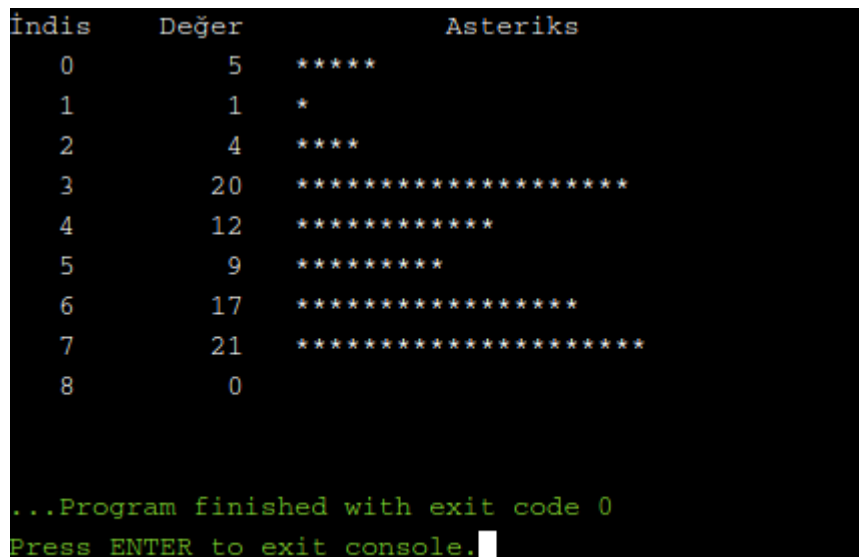
```

2.) Dizinin ilklendirilmiş elemanlarını asteriks olarak yazdıran programı yazalım.

ÖRNEK 5.2:

```
1. //DİZİNİN ELEMAN DEĞERLERİ KADAR ASTERİKS YAZDIRAN PROGRAM
2.
3. #include <stdio.h>
4. #define BOYUT 8
5.
6. int main()//main fonksiyonu başlangıcı
7. {
8.     int dizi[BOYUT]={5,1,4,20,12,9,17,21};// dizi tanımlaması ve değerlerinin
        atanması
9.
10.    int i , j; //döngü elemanları
11.
12.    printf("%s%10s%20s\n","İndis","Değer","Asteriks");
13.
14.    for(i=0;i<=BOYUT;i++)
15.    {
16.        printf("%4d%10d  ",i,dizi[i]);
17.
18.        for(j=1;j<=dizi[i];j++){ //bir satır yazdır
19.            printf("*");
20.        }
21.
22.        printf("\n"); //yeni satıra geç
23.    }
24.
25.
26.    return 0;
27.
28.
29.} // main fonksiyonu sonu
30.
```

ÇIKTI:



```
İndis      Değer      Asteriks
0          5      *****
1          1      *
2          4      ****
3         20      *****
4         12      *****
5          9      *****
6         17      *****
7         21      *****
8          0

...Program finished with exit code 0
Press ENTER to exit console.
```

3.) Rastgele sıralı bir tam sayı dizisini baloncuk sıralama (bubble sort) algoritmasını kullanarak küçükten büyüğe sıralayalım. Algoritmanın mantığı her diziyi taramada dizinin en büyük elemanını bulup sona at şeklinde çalışır. Örneğin `dizi[5]={3,2,4,5,1}`; şeklinde 5 elemanlı bir diziyi baloncuk algoritması kullanarak sıralayalım:

sırasız hali

3,2,4,5,1

1. adım

2,3,1,4,5

2.adım

2,1,3,4,5

3.adım

1,2,3,4,5

şeklinde oluşur.

ÖRNEK 5.3:

```
1. //KARMAŞIK SIRADA VERİLEN 5 ELEMANLI BİR DİZİYİ SIRALAYAN PROGRAM
2.
3. #include <stdio.h>
4.
5. int main()//main fonksiyonu başlangıcı
6. {
7.
8.     int dizi[]={3,2,4,5,1};
9.
10.    int i,j; //döngü değişkenleri
11.    int temp; //geçici değişken
12.
13.    printf("%s\n","sırasız dizi: ");
14.
15.    for(i=0;i<5;i++)
16.        printf("%d ",dizi[i]);
17.
18.    for(i=0;i<5;i++)
19.    {
20.        for(j=0;j<5;j++)
21.        {
22.            if(dizi[j]>dizi[j+1])
23.            {
24.                temp=dizi[j];
25.                dizi[j]=dizi[j+1];
26.                dizi[j+1]=temp;
27.
```

```

28.         }
29.
30.     }
31. }
32.
33.     printf("\n\n%s\n", "sıralı dizi:");
34.     for(i=0;i<5;i++)
35.         printf("%d ",dizi[i]);
36.
37.     return 0;
38. } // main fonksiyonu sonu
39.

```

Programımızda iç içe for döngüsü kullacağız böylece dizimizin tamamının gezildiğinden emin olacağız. Değiştirme işlemi yapmak için bir tane , döngülerimiz için iki tane olmak üzere üç değişken tanımladık. İlk başta dizinin sırasız halini yazdırdık. Sonrasında sıralama işlemini anlattığımız biçimde yaptık ve sıralı diziyi yazdırdık.

ÇIKTI:



```

[v4gus@parrot]~[~/Desktop/c_dersleri/4.bölüm diziler]
└─$ ./a.out
sırasız dizi:
3 2 4 5 1

sıralı dizi:
1 2 3 4 5 [v4gus@parrot]~[~/Desktop/c_dersleri/4.bölüm diziler]
└─$

```

5.1 Karakter dizileri

Buraya kadar yalnızca integer dizileri ile işlem yaptık. Fakat diziler her veri türündeki verileri tutabilirler. String veya karakter dizileri denilen dizilerde veri depolama işlemini anlatacağız. Tanımlama işlemini char türünde yapacağız. Dizi isimlendirmesinin yine değişken isimlendirme kurallarına uygun olması gerekiyor.

```
char karakterDizisi[]="Degerler" ;
```

ya da

```
char karkterDizisi[4]="isim";
```

şeklinde tanımlanırlar. Diğer dizilerden farklı olarak son indisinde NULL karakteri tutarlar. NULL'ın karakter sabiti "\0" şeklinde gösterilir. Tüm string ifadeleri \0 ile sonlandırılır.

```
char string[]="isim";
```

ataması yapıldığında bellekte oluşan görüntü

[0] [1] [2] [3] [4]

'i' 's' 'i' 'm' '\0' şeklindedir. Dolayısıyla 4 karakterlik veri tutmak için 5 karakterlik alan ayrılır. Fakat diziler 0'dan başladığı için bizim fazladan alan tanımlaması yapmamıza gerek yoktur. Sadece atama yapacağımız ya da girdi alacağımız ifadenin karakter sayısını yazmamız yeterlidir.

Farklı bir yöntem ile string içersine "isim" ifadesini atayalım:

```
char string []={'i', 's','i' , 'm'};
```

Bu yöntem sayısal dizilerin ilklendirilmesine benzer bir yöntemle çalışır ve çıktısını sıralı olarak aldığımız zaman "isim" çıktısını alırız.

Şimdilik karakter dizilerine karakter okumak için "scanf("%c",&dizi[indis]); " ifadesini kullanacağız.

Kullanıcıdan ismini girdi olarak alan ve hoş geldiniz yazdıran programı yazalım.

ÖRNEK 5.4:

```
1.
2. //KULLANICIDAN İSMİNİ ALARAK HOŞGELDİNİZ YAZDIRAN PROGRAM
3.
4. #include <stdio.h>
5.
6. #define BOYUT 50
7.
8. int main()//main fonksiyonun başlangıcı
9. {
10.     char isim[BOYUT];
11.
12.     char karakter='i'; //kontrol karakteri
13.     int i=0; // döngü değişkeni
14.
15.     printf("%s\n","isminizi giriniz:" );
16.
17.     while(1)
18.     {
19.         scanf("%c",&karakter);
20.
21.         if(karakter=='\n'){
22.             isim[i]='\0';
23.             break;
24.         }
25.         else
26.             isim[i]=karakter;
27.
28.         i++;
```

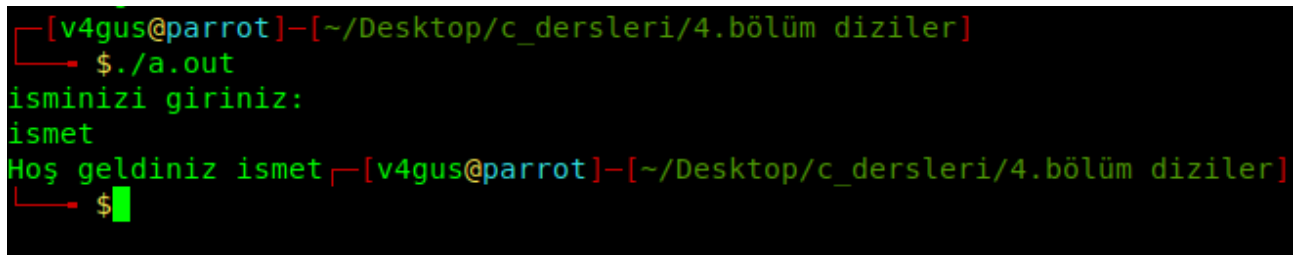
```

29.     }
30.
31.     i=0;
32.
33.     printf("%s","Hoş geldiniz ");
34.
35.     while (isim[i]!='\0')
36.     {
37.         printf("%c",isim[i]);
38.
39.         i++;
40.
41.     }
42.
43.     return 0;
44.
45. } //main fonksiyonu sonu
46.

```

Bu programımızda birçok kontrol kullandık. ilk while içerisinde karakter değişkeni ile kullanıcının girdiği karakterin Enter olup olmadığını kontrol ettik ;
 Enter değilse "isim" değişkenine atadık ve döngü sayacını arttırdık , Enter girilmişse sonlandırma karakterini hata olmaması için kendimiz ekledik ve döngüden çıktık. Bu işlemi ileride hazır fonksiyonlar ile daha kolay yapacağız fakat şimdilik nasıl çalıştığının anlaşılması için bir defa yazdık. Sonrasında "hoş geldiniz" yazdırdık ve yeni satıra geçmedik ki ismi de yazabilelim. Döngü değişkenini 0 olarak atayıp ismi while döngüsünde ismin 'i'. indisi NULL karakterinden farklı ise yazdır komutu ile yazdırdık ve döngü değişkeninin değerini bir arttırdık.

ÇIKTI:



```

[v4gus@parrot]~/. Desktop/c_dersleri/4.bölüm diziler
└─$ ./a.out
isminizi giriniz:
ismet
Hoş geldiniz ismet [v4gus@parrot]~/. Desktop/c_dersleri/4.bölüm diziler
└─$

```

5.2. İKİ VE ÇOK BOYUTLU DİZİLER

Çoğu durumda işlemlerimizi yerine getirebilmek için tek boyutlu diziler yetersiz olur. Mesela vize ve final notlarını aynı dizide tutmak varken neden farklı iki dizi tanımlayalım ki ? Bu örnekten daha karmaşık olarak matris işlemleri yapmak içinde çok boyutlu dizilere ihtiyaç duyarız.

Tanımları:

veri_tipi dizi [boyut1][boyut2];

şeklinde yapılır.

Satır ve sütunların hangisi olacağına programcı karar verir. Biz ilk boyutu satır , ikinci boyutu sütun olarak kabul ederek devam ederiz.

```
int cokBoyutlu [2][5]={    1,2,3,4,5
                          6,7,8,9,10};
```

gibi bir tanımlama kullanılması durumunda ikinci satırı ikinci satır olarak algılar ve okunabilirliği artırırken :

```
int cokBoyutlu [2][5]={1,2,3,4,5,6,7,8,9,10};
```

tanımlaması da kullanılabilir.

bu tanımlamalar hafızada

		[0]	[1]	[2]	[3]	[4]
cokBoyutlu	[0]	1	2	3	4	5
	[1]	6	7	8	9	10

şeklinde tutulur.

Şimdi geçme notu 50 olan ve dizi indislerini numara olarak alan , 5 öğrencinin vize ve final notlarını alarak öğrencilerin dersten kalıp kalmadığını yazan programı kodlayalım

ÖRNEK 5.5:

```
1.
2. //5 ÖĞRENCİNİN VİZE VE FİNAL NOTLARINI ALARAK GEÇME KALMA
   DURUMUNU YAZDIRAN PROGRAM
3.
4. #include <stdio.h>
5.
6. #define SIZE1 2
7. #define SIZE2 5
8.
9. int main()//main fonksiyonu başlangıcı
10. {
11.     int notlar[SIZE1][SIZE2];
12.     float durum[SIZE2];
13.
14.     int i,j;
15.
16.     for(i=0;i<SIZE1;i++)
17.     {
18.         for(j=0;j<SIZE2;j++)
19.         {
20.             if(i==0)
```

```

21.         {
22.             printf("%d. öğrencinin vize notunu giriniz:",j+1);
23.             scanf("%d",&notlar[i][j]);
24.         }
25.
26.     else
27.     {
28.         printf("%d. öğrencinin final notunu giriniz:",j+1);
29.         scanf("%d",&notlar[i][j]);
30.     }
31.
32.     }
33. }
34.
35.
36. for(i=0;i<SIZE1;i++)
37. {
38.     for(j=0;j<SIZE2;j++)
39.     {
40.         if(i==0)
41.             durum[j] +=(notlar[i][j]*0.4);
42.
43.         else
44.             durum[j] +=(notlar[i][j]*0.6);
45.     }
46. }
47.
48. for(j=0;j<SIZE2;j++)
49. {
50.     if(durum[j]<50.000)
51.         printf("%d numaralı öğrenci kaldı\n",j+1);
52.     else
53.         printf("%d numaralı öğrenci geçti\n",j+1);
54. }
55.
56. return 0;
57. } // main fonksiyonu sonu
58.

```

ÇIKTI:

```

[v4gus@parrot]--[~/Desktop/c_dersleri/4.bölüm diziler]
└─ $ ./a.out
1. öğrencinin vize notunu giriniz:9
2. öğrencinin vize notunu giriniz:32
3. öğrencinin vize notunu giriniz:65
4. öğrencinin vize notunu giriniz:87
5. öğrencinin vize notunu giriniz:56
1. öğrencinin final notunu giriniz:87
2. öğrencinin final notunu giriniz:76
3. öğrencinin final notunu giriniz:30
4. öğrencinin final notunu giriniz:60
5. öğrencinin final notunu giriniz:10
1 numaralı öğrenci geçti
2 numaralı öğrenci geçti
3 numaralı öğrenci kaldı
4 numaralı öğrenci geçti
5 numaralı öğrenci kaldı
[v4gus@parrot]--[~/Desktop/c_dersleri/4.bölüm diziler]
└─ $

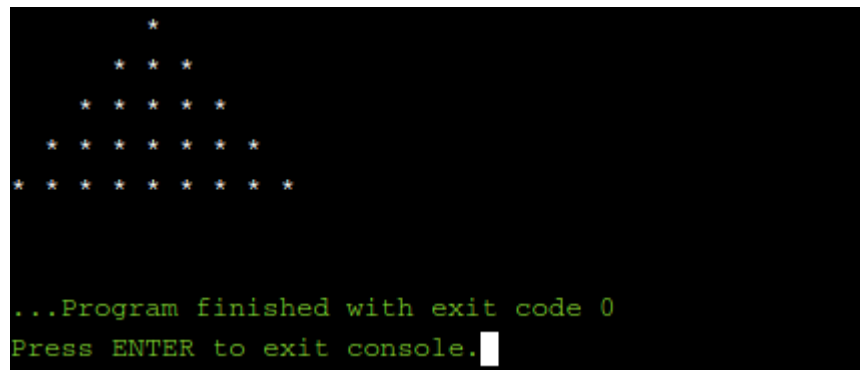
```

İki boyutlu diziler ile çeşitli şekiller de oluşturulabilir. asteriklerden bir pramit oluşturan programı yazalım.

ÖRNEK 5.6:

```
1.
2. //ASTERİKSLERDEN PİRAMİT YAZDIRAN PROGRAM
3.
4. #include <stdio.h>
5.
6. int main()//main fonksiyonu başlangıcı
7. {
8.     int i, j, l; //döngü değişkenleri
9.
10.
11.     for(i=1; i<=5; ++i,l=0)
12.     {
13.         for(j=1; j<=5-i; ++j) // boşlukları yazdırmak için kullandığımız döngü
14.         {
15.             printf(" ");
16.         }
17.
18.         while(l != 2*i-1) // asterikleri yazdırmak için döngümüz
19.         {
20.             printf("* ");
21.             ++l;
22.         }
23.
24.         printf("\n");
25.     }
26.
27.     return 0;
28.
29. } //main fonksiyonu sonu
30.
```

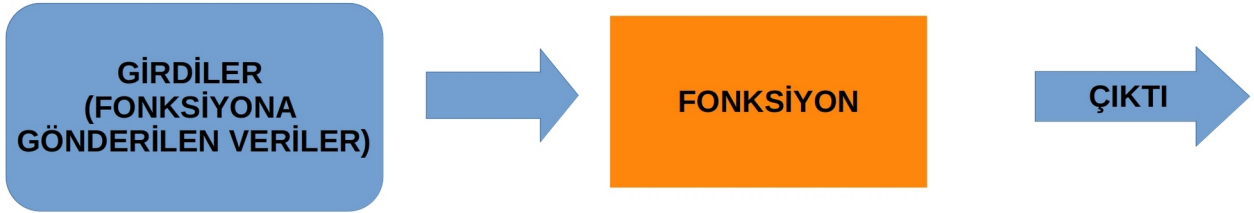
ÇIKTI:



```
      *
     * * *
    * * * * *
   * * * * * * *
  * * * * * * * *
...Program finished with exit code 0
Press ENTER to exit console.
```

6. FONKSİYONLAR

Fonksiyon bir girdi alarak bu girdi üzerinde çeşitli işlemler yaparak bir çıktı üreten yapıdır.



Bizden kullanıcıdan farklı aralıklarla 200 tane sayı çifti alıp bu sayı çiftinin ortalamasını bulan bir program yazmamız istendiğinde ortalama hesabı yapmak için iki seçeneğimiz bulunuyor:

0- ya her sayı çifti için teker teker ortalama alan kodları yazacağız

1- ya da iki sayının ortalamasını bulan bir fonksiyon yazıp ihtiyaç duydukça çağıracağız

Fonksiyon yazmak bize ,

kod tekrarının önüne geçme ve böylece kodumuzun hacmi küçülürken işlevi artması,
kod okunabilirliğinin artması ve tekrar kullanılabilirliğinin artması,
hata ayıklama ve bakım sırasında işlemleri kolaylaşması,

bu sayılan özellikler aynı zamanda iyi bir programın özellikleri olduğunu da göz önünde bulundurarak şu sonuca varabiliriz "iyi bir program fonksiyonlara ayrılabilir veya fonksiyonlar şeklindedir."

Genel anlamda fonksiyonları iki kısma ayırabiliriz:

0)Önceden tanımlanmış standart kütüphane fonksiyonları:

Sıklıkla kullandığımız printf() , scanf() ve puts() fonksiyonları bu gruba girer.

1)Sonradan tanımlanmış fonksiyonlar:

Programcının ihtiyaç duyduğu biçimde oluşturduğu fonksiyonlardır. Örneğimizdeki ortalama hesaplayan fonksiyon gibi.

6.0 FONKSİYON TANIMLAMA

Genel olarak bir fonksiyon

```
fonksiyonun_dönüş_tipi fonksiyon_ismi (argümanlar) //fonksiyon
{
    tanımlamalar
    işlemler
}
```

şeklinde tanımlanır.

fonksiyonun dönüş tipi : fonksiyonun çıktı olarak gönderdiği verinin türüdür (integer,float,char,vb).Bazı fonksiyonlar çıktı olarak veri üretmezler bunlar void fonksiyon olarak isimlendirilir.

fonksiyon ismi : Fonksiyonlar isimlendirilirken değişken tanımlama kurallarına uygun olarak isimlendirilir.Fakat okunabilirliğin artması açısından isminin yaptığı işleme işaret etmesi tavsiye edilir.

argümanlar: argümanlar fonksiyonun işlem yapmak için kullandığı değişkenler olarak düşünülebilir.Yani iki tam sayıyı toplayan bir fonksiyon için argümanlarımız (int sayi1 , int sayi2) olarak tanımlanabilir.Bazı fonksiyonlar argüman almazlar bunların argüman kısımları boştur.

Bir fonksiyon tanımlanırken iki farklı şekilde tanımlanabilir. Ya main fonksiyonundan önce fonksiyon tanımlanır ya da önce fonksiyonun prototipi tanımlanır , main'den sonra fonksiyon tanımlanır.Prototip şablon olarak düşünebiliriz.Eğer prototip bildirmişsek derleyici ilerleyen bloklardan birisinde bu prototipin fonksiyonunu tanımlayacağımızı düşünür ve hata vermez.Fakat tanımlama yapmazsak hata alırız.Tanımlama işlemine aynı zamanda fonksiyon imzası (function signature) da denir.

Fonksiyonun main'den önce tanımlanması :

```
fonksiyonun_dönüş_tipi fonksiyon_ismi (argümanlar) fonksiyon imzası tanımlaması
{
    tanımlamalar
    işlemler
}

main()
{
    tanımlamalar
    işlemler
}
```

Fonksiyonun main'den sonra tanımlanması ve prototip yazımı :

```
fonksiyonun_dönüş_tipi fonksiyon_ismi (argümanlar) ; //fonksiyon prototipi
main()
{
    tanımlamalar
    işlemler
}
```

```
}
```

fonksiyonun_dönüş_tipi fonksiyon_ismi (argümanlar) // fonksiyon imzası/ tanımlaması

```
{
```

```
    tanımlamalar  
    işlemler
```

```
}
```

şeklinde yapılır.

Fonksiyon kavramının daha iyi anlaşılabilmesi için iki tam sayıyı toplayan bir fonksiyon yazalım.İki tam sayı toplayacağımız için fonksiyonumuzun dönüş tipi integer , alacağı parametreler de integer tipinde olacaktır.

```
int topla (int sayi1 , int sayi2)
```

```
{
```

```
    return sayi1+sayi2;
```

```
}
```

return ifadesi herhangi bir veriyi fonksiyonu çağırıldığı yere gönderir.Eğer printf içerisinde çağırırsak direk yazdırabiliriz.Yazdığımız örnek için çağırımızı

```
printf("%d",topla(2,5));
```

şeklinde yaparız.

Fonksiyona argüman olarak 2 ve 5 değerlerini gönderdik fonksiyonumuz da bize toplamalarını geri dönderdi.Bu işlemleri koda dökecek olursak:

ÖRNEK 6.1:

```
1. #include <stdio.h>  
2.  
3. int topla(int sayi1, int sayi2) ; //fonksiyon prototipi  
4.  
5. int main(void) // main fonksiyonu başlangıcı  
6. {  
7.  
8.     printf("2 ve 5 tam sayılarının toplamı :%d'dir\n",topla(2,5));  
9.  
10.    return 0;  
11.  
12. }      //main fonksiyonu sonu  
13.  
14.  
15. int topla ( int sayi1,int sayi2)// fonksiyon imzası  
16. {  
17.     return sayi1+sayi2;  
18. }      //toplam fonksiyonu sonu
```

19.

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
➔ ./a.out
2 ve 5 tam sayılarının toplamı :7'dir
[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
➔ $
```

Programımızda main'in int veri türünde değer döndürdüğünü fakat argüman almadığını görüyoruz. Main'in geri dönüş değeri 0 ise programın başarılı bir şekilde yürütüldüğünü ifade eder. Eğer programın yürütülme sırasında problemi işaret etmesini istersek bu dönüş değerini sıfırdan farklı bir değer olarak değiştirmemiz yeterli olacaktır.

ÖRNEK 6.2:

```
1.
2. //Kullanıcının girdiği üç sayıdan en büyük olanını fonksiyon ile bulan program:
3.
4. #include <stdio.h>
5.
6. #define SIZE 3
7.
8. int enBuyuk (int arr[] , int size ); // enBuyuk fonksiyon prototipi
9.
10. int main (void) // main başlangıcı
11. {
12.     int dizi[SIZE];
13.     int i;
14.
15.
16.
17.     for(i=0;i<SIZE;i++)
18.     {
19.         printf("%d. sayıyı giriniz:",i+1);
20.         scanf("%d",&dizi[i]);
21.
22.     }
23.
24.     printf("Girdiğiniz değerlerin en büyük olanı %d değeridir.\n",enBuyuk(dizi,SIZE));
25.
26.     return 0;
27.
28. } // main sonu
29.
30. // enBuyuk fonksiyonu tanımı
31. //x , y ve z parametre
32.
33. int enBuyuk(int arr[],int size)
```

```

34.{
35.    int buyuk; // en büyük değeri saklayacak olan değişken
36.    int i,j;//döngü değişkenleri
37.
38.    buyuk=arr[0];
39.
40.    for(i=0;i<size;i++)
41.    {
42.        for(j=0;j<size;j++)
43.        {
44.            if(arr[j]>buyuk)
45.                buyuk=arr[j];
46.        }
47.    }
48.
49.    return buyuk;
50.
51.} // enBuyuk fonksiyonunun sonu

```

ÇIKTI:

```

[v4güs@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
$ ./a.out
1. sayıyı giriniz:70
2. sayıyı giriniz:99
3. sayıyı giriniz:08
Girdiğiniz değerlerin en büyük olanı 99 değeridir.

```

Fonksiyonlarla ilgili dikkat edilmesi gereken bazı noktalar:

0-main de bir fonksiyondur.

1-Her C programının main fonksiyonu olmak zorundadır.

2-Fonksiyon tanımlamanın üst sınırı yoktur.

3-Fonksiyonların isimleri yerine getirdiği görevi anlatmalıdır.

4-Bir fonksiyon bir tane görevi yerine getirmeli. Her görev için farklı fonksiyonlar tanımlanmalıdır.

6.1 DEĞER İLE FONKSİYON ÇAĞIRMA (CALL BY VALUE)

C programlama'da varsayılan fonksiyon çağırma yöntemidir.Bu yöntemi tam açıklayabilmek için gerçek ve biçimsel parametreleri açıklamamız gerekir.Gerçek parameteler , fonksiyon çağırıldığında görünen parametlerdir.Biçimsel parametreler ise fonksiyon bildiriminde kullandığımız parametlerdir. Toplama işlemi yapan fonksiyonumuzu hatırlayacak olursak sayi1 ve sayi2 parametreleri biçimsel parametre , topla fonksiyonuna parametre olarak gönderdiğimiz 2 ve 5 gerçek parametredir.

ÖRNEK 6.3:

1. //İKİ TAM SAYIYI TOPLAYAN FONKSİYON


```

2.
3. #include <stdio.h>
4.
5. int topla(int sayi1, int sayi2) ; //fonksiyon prototipi
6.
7. int main(void) // main fonksiyonu başlangıcı
8. {
9.
10.     printf("2 ve 5 tam sayılarının toplamı :%d'dir\n",topla(2,5));
11.
12.     return 0;
13.
14. }//main fonksiyonu sonu
15.
16.
17. int topla ( int sayi1,int sayi2)// fonksiyon imzası
18. {
19.     return sayi1+sayi2;
20. } //toplama fonksiyonu sonu
21.

```

ÇIKTI:

```
[v4gus@parrot]~/.Desktop/c_dersleri/5.bölüm fonksiyonlar
└─ $./a.out
2 ve 5 tam sayılarının toplamı :7'dir
[v4gus@parrot]~/.Desktop/c_dersleri/5.bölüm fonksiyonlar
└─ $
```

Değer ile çağırma (Call by value) Nedir ?

Biz bir fonksiyonu gerçek parametreler ile çağırdığımızda gerçek parametrelerin değerleri biçimsel parametrelere kopyalanır. Bundan dolayı biçimsel parametrelerde yapılan işlemler gerçek parametrelere return edilmediği müddetçe yansımaz.

ÖRNEK 6.4:

```
1. //DEĞER İLE FONKSİYON ÇAĞIRMA (CALL BY VALUE)
2. #include <stdio.h>
3.
4. char degistir(char karakter); // degistir fonksiyonu prototipi
5.
6. int main()//main fonksiyonu başlangıcı
7. {
8.     char karakter1='a';
9.     char karakter2=degistir(karakter1);
10.    printf("karakter1: %c\n", karakter1);
```

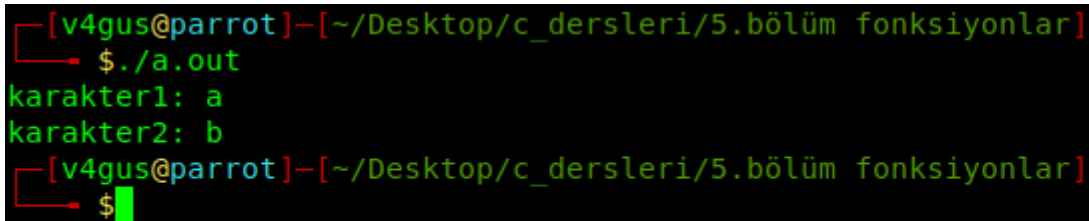
```

11. printf("karakter2: %c\n", karakter2);
12.
13. return 0;
14.}
15.
16.
17.
18.char degistir(char karakter) // degistir fonksiyonu tanımlaması
19.{
20.    karakter ++;
21.    return karakter;
22.} // char degistir fonksiyonu sonu
23.

```

Bu örneğimizde char degistir fonksiyonu char türündeki bir değeri integer türündeymiş gibi arttırma operatörü kullanarak arttırıp char türündeki karakter2 değişkenine atıyor. Burada yaptığımız işlem aslında ascii koduna 1 eklemektir. Sonrasında ise iki değişkenin değerlerini de yazdırıyoruz. Açıkça görülüyor ki değer ile çağırma işleminde gerçek parametrelerimiz değişikliğe uğramıyor.

ÇIKTI:



```

[v4gus@parrot]~/. Desktop/c_dersleri/5.bölüm fonksiyonlar
➔ ./a.out
karakter1: a
karakter2: b
[v4gus@parrot]~/. Desktop/c_dersleri/5.bölüm fonksiyonlar
➔ $

```

Çok daha açık görebilmek için iki değişkenin değerlerini değiştiren bir fonksiyon yazıp çıktısını inceleyelim.

ÖRNEK 6.5:

```

1. //DEĞER İLE FONKSİYON ÇAĞIRIP DEĞERLERİ DEĞİŞTİREN PROGRAM
2. #include <stdio.h>
3.
4. char degistir(char kar1,char kar2); // degistir fonksiyonu prototipi
5.
6. int main()//main fonksiyonu başlangıcı
7. {
8.    char karakter1='i';
9.    char karakter2='a';
10.
11.    printf("ilk durum\nkarakter1: %c\nkarakter2: %c\n",karakter1,karakter2);
12.
13.    degistir(karakter1,karakter2);

```

```

14.
15. printf("\ndeğişme işleminden sonra\nkarakter1: %c\nkarakter2: %c\n",karakter1,karakter2);
16.
17. return 0;
18.}
19.
20.
21.
22.char degistir(char kar1,char kar2) // degistir fonksiyonu tanımlaması
23.{
24.    char gecici=kar1;
25.    kar1=kar2;
26.    kar2=gecici;
27.
28.    printf("\ndeğişim işlemi\nkar1:%c\nkar2:%c\n",kar1,kar2);
29.
30.} // char degistir fonksiyonu sonu
31.

```

ÇIKTI:

```

[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
$ ./a.out
ilk durum
karakter1: i
karakter2: a

değişim işlemi
kar1:a
kar2:i

değişme işleminden sonra
karakter1: i
karakter2: a
[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
$

```

Çıktıda görüldüğü gibi kar1 ve kar2 biçimsel değişkenleri değişse bile karakter1 ve karakter2 gerçek değişkenleri ilk durumlarını koruyorlar.

6.2 REFERANS İLE FONKSİYON ÇAĞIRMA (CALL BY REFERENCE)

Bir fonksiyonu gerçek fonksiyonların adresleri ile çağırma işlemine referas ile çağırma (call by reference) denir.Referansla çağırma işleminde gerçek değişkenlerin adresleri biçimsel değişkenlere kopyalanır. Dolayısıyla biçimsel değişkenlerde yapılan değişiklikler gerçek değişkenlere de yansır.

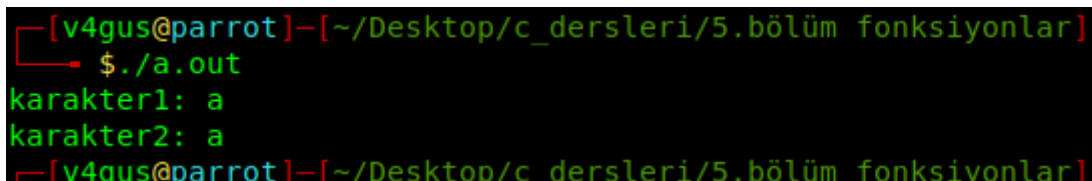
Değer ile çağırdığımız fonksiyonları şimdi referansla çağıralım:

ÖRNEK 6.6:

```
1. //REFERANS İLE FONKSİYON ÇAĞIRMA (CALL BY REFERENCE)
2. #include <stdio.h>
3.
4. char degistir(char *karakter); // degistir fonksiyonu prototipi
5.
6. int main()//main fonksiyonu başlangıcı
7. {
8.     char karakter1='a';
9.     char karakter2=degistir(&karakter1);
10.    printf("karakter1: %c\n", karakter1);
11.    printf("karakter2: %c\n", karakter2);
12.
13.    return 0;
14.}
15.
16.
17.
18.char degistir(char *karakter) // degistir fonksiyonu tanımlaması
19.{
20.    *karakter=*karakter++;
21.    return *karakter;
22.} // degistir fonksiyonu sonu
23.
```

Burada "&" referans operatörüdür bir değişkenin adresine ulaşmak için kullanılır. "*" ise dereferans operatörüdür ve adresi belirtilen değere ulaşmak için kullanılır. Gerçek değişkenlerin adresi biçimsel değişkenlere atandığı için gerçek değişkenler de değişmiş oldu.

ÇIKTI:



```
[v4gus@parrot]~-[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
$ ./a.out
karakter1: a
karakter2: a
[v4gus@parrot]~-[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
```

Daha iyi anlayabilmek için bir örnek daha yazalım

ÖRNEK 6.7:

```
1. //DEĞER İLE FONKSİYON ÇAĞIRMA (CALL BY VALUE)
2. #include <stdio.h>
3.
4. char degistir(char *kar1,char *kar2); // degistir fonksiyonu prototipi
5.
```

```

6. int main()//main fonksiyonu başlangıcı
7. {
8.     char karakter1='i';
9.     char karakter2='a';
10.
11.     printf("ilk durum\nkarakter1: %c\nkarakter2: %c\n",karakter1,karakter2);
12.
13.     degistir(&karakter1,&karakter2);
14.
15.     printf("\ndeğişme işleminden sonra\nkarakter1: %c\nkarakter2: %c\n",karakter1,karakter2);
16.
17.     return 0;
18.}
19.
20.
21.
22.char degistir(char *kar1,char *kar2) // degistir fonksiyonu tanımlaması
23.{
24.    char gecici=*kar1;
25.    *kar1=*kar2;
26.    *kar2=gecici;
27.
28.    printf("\ndeğişim işlemi\nkar1:%c\nkar2:%c\n",*kar1,*kar2);
29.
30.}// degistir fonksiyonu sonu
31.

```

ÇIKTI:

```

[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
└─ $ ./a.out
ilk durum
karakter1: i
karakter2: a

değişim işlemi
kar1:a
kar2:i

değişme işleminden sonra
karakter1: a
karakter2: i
[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
└─ $

```

6.3 Recursive (Özyineli) Fonksiyonlar

En basit anlatımı ile kendi çıktısını kendisine değer olarak çağıran fonksiyon diyebiliriz. Mesela 10 metrelik bir duvar inşa etmem için önce 9 metrelik bir duvar inşa etmem gerekir. Sonrasında bir tuğla daha koyarak amacıma ulaşmış olurum. Kavramsal olarak bu , "duvar inşa etme" fonksiyonunun bir yükseklik aldığı ve bu yükseklik birden büyükse ilk olarak kendinden

daha düşük bir duvar inşa etmek için çağırır ve sonra 1 metre tuğla ekler.

Tanımlanması :

```
void ozyineli()
{
    koşul
    ozyineli() ;// fonksiyon kendini çağırıyor
}
int main()
{
    ozyineli();
    return 0;
}
```

Tanımlamaya basit bir örnek olması için 1 den 100 e kadar olan sayıları recursiv fonksiyon ile yazdıran programı yazalım:

ÖRNEK 6.8:

```
1. //1 DEN 100 E KADAR OLAN SAYILARI YAZDIRAN RECURSİVE FONKSİYON
2.
3. #include <stdio.h>
4. #include <stdlib.h>
5.
6. void sayiYaz(int sayi)
7. {
8.
9.     if(sayi<=100)
10.    {
11.
12.        printf("sayi:%d\t",sayi );
13.        sayiYaz(sayi+1);
14.    }
15.
16.    else
17.        exit;
18.
19.}
20.int main () //main başlangıcı
21.{
22.    sayiYaz(1);
23.
24.    return 0;
25.
26.} // main sonu
27.
```

Programımızda "exit" komutunu koşulların sağlanmadığı durumda programın sonlandırmak amacıyla kullandık.

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
$ ./a.out
sayi:1 sayi:2 sayi:3 sayi:4 sayi:5 sayi:6 sayi:7 sayi:8 sayi:9 sayi:10s
sayi:11 sayi:12 sayi:13 sayi:14 sayi:15 sayi:16 sayi:17 sayi:18 sayi:19 sayi:20s
sayi:21 sayi:22 sayi:23 sayi:24 sayi:25 sayi:26 sayi:27 sayi:28 sayi:29 sayi:30s
sayi:31 sayi:32 sayi:33 sayi:34 sayi:35 sayi:36 sayi:37 sayi:38 sayi:39 sayi:40s
sayi:41 sayi:42 sayi:43 sayi:44 sayi:45 sayi:46 sayi:47 sayi:48 sayi:49 sayi:50s
sayi:51 sayi:52 sayi:53 sayi:54 sayi:55 sayi:56 sayi:57 sayi:58 sayi:59 sayi:60s
sayi:61 sayi:62 sayi:63 sayi:64 sayi:65 sayi:66 sayi:67 sayi:68 sayi:69 sayi:70s
sayi:71 sayi:72 sayi:73 sayi:74 sayi:75 sayi:76 sayi:77 sayi:78 sayi:79 sayi:80s
sayi:81 sayi:82 sayi:83 sayi:84 sayi:85 sayi:86 sayi:87 sayi:88 sayi:89 sayi:90s
sayi:91 sayi:92 sayi:93 sayi:94 sayi:95 sayi:96 sayi:97 sayi:98 sayi:99 sayi:100s
[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
```

Recursive fonksiyonlara en iyi örnek faktöriyel hesabı üzerinden verilebilir :

$5! = 5.4.3.2.1$ şeklinde bir işlem gerçekleşir biraz daha dikkatli bakıldığında yapılan işlemin aslına

$$\begin{aligned} 5! &= 5 * 4! \\ &= 5 * 4 * 3! \\ &= 5 * 4 * 3 * 2! \\ &= 5 * 4 * 3 * 2 * 1 \end{aligned}$$

şeklinde olduğunu görürüz.

Buradan yola çıkarak recursive bir şekilde kullanıcıdan girilen değerin faktöriyelini hesaplayan bir program yazalım.

ÖRNEK 6.9:

```
1. //REKÜRSİF ŞEKİLDE FAKTÖRİYEL HESAPLAYAN PROGRAM
2.
3. #include <stdio.h>
4.
5. int faktoriyel (int x); // faktoriyel fonksiyonunun prototipi
6.
7. int main (void) // main başlangıcı
8. {
9.     int sayi ;
10.    printf("%s","lütfen faktöriyelini hesaplamak istediğiniz sayıyı giriniz:");
11.    scanf("%d",&sayi);
12.    printf("%d!=%d",sayi,faktoriyel(sayi));
13.
14.    return 0;
15.} //main sonu
```

```

16.
17. int faktoriyel(int x) // faktoriyel fonksiyonu tanımlaması
18. {
19.     if(x>=1) // x 1 'e eşit veya büyükse
20.         return x*faktoriyel(x-1);
21.
22.     else
23.         return 1;
24. } // faktoriyel fonksiyonunun sonu
25.

```

Programımız kullanıcının girdiği sayıyı 1 sayısına eşit oluncaya kadar kendisinin bir eksiği ile çarpıyor ve 1 sayısına eşit olunca 1 ile çarpıp "faktoriyel" fonksiyonu görevini yerine getirmiş oluyor.

ÇIKTI:

```

[v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
$ ./a.out
lutfen faktoriyelini hesaplamak istediğiniz sayıyı giriniz:23
23!=862453760 [v4gus@parrot]--[~/Desktop/c_dersleri/5.bölüm fonksiyonlar]
$

```

Özyineli fonksiyonlar bazı durumlar için işi yokuşa süren bir yöntem olmasına karşın, hızlı sıralama (quicksort) algoritması gibi bazı algoritmaların içerisinde yer alarak işimizi kolaylaştırabiliyor. Yani "taş yerinde ağırdır" deyiimi ile özetleyebiliriz kullanımını. Hızlı sıralama algoritması böl ve fethet (divide and conquer) mantığıyla çalışır. Sıralanacak olan dizinin bir tane pivot seçerek büyük sayıları pivotun sağına, küçük olanları pivotun sol tarafına atar. Bu işlem tüm dizi sıralı hale gelene kadar sağ ve sol kısımlar arasında da uygulanır. Dizimiz 11 elemanlı olsun ve sıralacak olan değerleri kullanıcıdan alarak hızlı sıralama algoritmasını kullanarak sıralayıp yazdıralım.

ÖRNEK 6.10:

```

1. //HIZLI SİRLAMA KULLANARAK KULLANICININ GİRDİĞİ DİZİYİ SİRALAYAN
   PROGRAM
2.
3. #include<stdio.h>
4. #define BOYUT 11
5.
6. void hizlisiralama(int dizi[BOYUT],int ilk,int son ); //hizlisiralama fonksiyon prototipi
7.
8. int main() //main fonksiyonu başlangıcı
9. {
10.
11.     int i ; //döngü değişkeni
12.     int dizi[BOYUT];
13.
14.
15.
16.     for(i=0;i<BOYUT;i++) // sıralanacak dizinin elemanlarının girilmesi

```



```

17. {
18.     printf("%d. elemanı giriniz: ", i+1);
19.     scanf("%d", &dizi[i]);
20. }
21.
22.
23. printf("%s\n", "Sıralanmamış dizi:"); //sırasız dizinin yazdırılması
24.
25. for(i=0; i<BOYUT; i++)
26.     printf(" %d", dizi[i]);
27.
28. hizlisiralama(dizi, 0, BOYUT-1); //hizlisirala fonksiyon çağırısı
29.
30. printf("\n%s\n", "Sıralanmış dizi:"); //sıralanmış dizinin yazdırılması
31.
32. for(i=0; i<BOYUT; i++)
33.     printf(" %d", dizi[i]);
34.
35. return 0;
36.
37. } //main fonksiyonu sonu
38.
39. void hizlisiralama(int dizi[BOYUT], int ilk, int son ) { //hizlisiralama fonksiyon tanımı
40.
41.     int i, j; //döngü değişkenleri
42.     int pivot, gecici; // pivot ve değişim için geçici değişken tanımlaması
43.
44.     if(ilk<son){ //pivotun seçilmesi
45.         pivot=ilk;
46.         i=ilk;
47.         j=son;
48.
49.         while(i<j){ //dizinin alt bölümlere ayrılarak sıralanması
50.             while(dizi[i]<= dizi[pivot] && i<son)
51.                 i++;
52.             while(dizi[j]>dizi[pivot])
53.                 j--;
54.             if(i<j){
55.                 gecici=dizi[i];
56.                 dizi[i]=dizi[j];
57.                 dizi[j]=gecici;
58.             }
59.         }
60.
61.         gecici=dizi[pivot];
62.         dizi[pivot]=dizi[j];
63.         dizi[j]=gecici;
64.         hizlisiralama(dizi, ilk, j-1);
65.         hizlisiralama(dizi, j+1, son);
66.
67.     }

```

68.} // hizlisiralama sonu
69.

ÇIKTI:

```
[v4gus@parrot]~/Desktop/c_dersleri/5.bölüm_fonksiyonlar
$ ./a.out
1.elemanı giriniz: 70
2.elemanı giriniz: 99
3.elemanı giriniz: 45
4.elemanı giriniz: 3
5.elemanı giriniz: 67
6.elemanı giriniz: 324
7.elemanı giriniz: 34
8.elemanı giriniz: 56
9.elemanı giriniz: 89
10.elemanı giriniz: 45
11.elemanı giriniz: 23
Sıralanmamış dizi:
70 99 45 3 67 324 34 56 89 45 23
Sıralanmış dizi:
3 23 34 45 45 56 67 70 89 99 324 [v4gus@parrot]~/Desktop/c_dersleri/5.bölüm_fonksiyonlar
$
```

7. İŞARETÇİLER (POINTERS)

İşaretçiler başka değişkenlerin adreslerini tutan değişkenlerdir. Pointerlar isimlendirilirken değişken isimlendirme kuralları kullanılır. Pointer tanımlaması

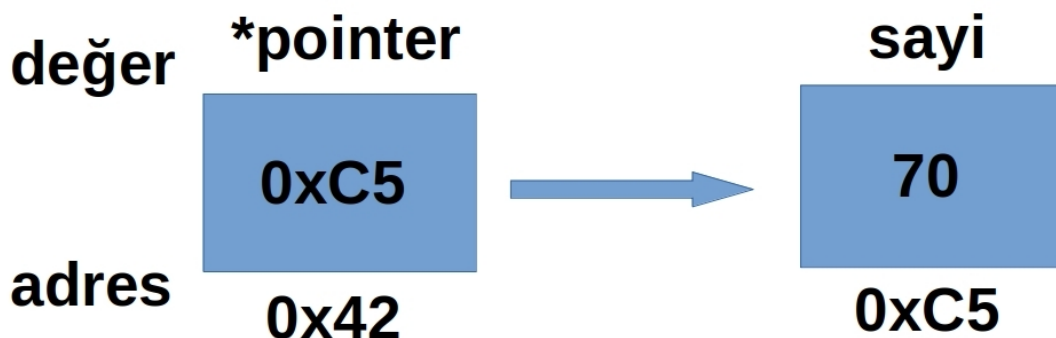
```
pointer_tipi *pointer_ismi;
```

şeklinde yapılır.

Önemli bir nokta da değişkenin türü ile işaretçinin türü aynı olmalı gerektirir. Yani float bir değişkenin adresini float tipinde bir pointera atayabiliriz. Adres gösterme kavramının daha iyi anlaşılabilmesi için

```
int *pointer;
int sayi=70;
pointer=&sayi;
```

atamasını şematize edelim:

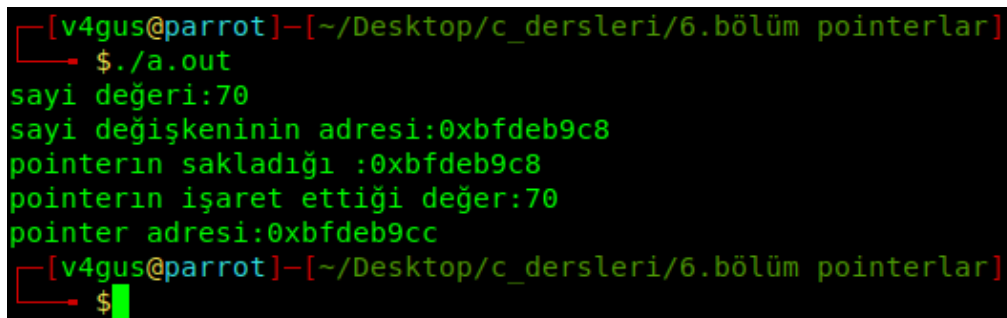


Daha önce referans ve dereferans operatörlerini anlatmıştık."&" adres referans operatörüdür bir değişkenin adresine ulaşmak için kullanılır."*" ise dereferans operatörüdür ve adresi belirtilen değere ulaşmak için kullanılır."%p" pointerın işaret ettiği adresi hexadecimal biçimde görüntülemek için kullanılan format belirtecidir.Pointerların da hafızada bir alan kapladığını bilmeliyiz ve adreslerinin olduğunu unutmamalıyız.Buraya kadar anlattıklarımızı koda dökelim ve bir integer pointera integer değişkenin adresini atayalım. Değişkenin adresini,değerini , pointerın adresini , tuttuğu adresi ve tuttuğu adresin değerini yazdıralım.

ÖRNEK 7.1:

```
1. #include <stdio.h>
2.
3. int main()
4. {
5.     int *pointer;
6.     int sayi=70;
7.     pointer=&sayi;
8.
9.     printf("sayi değeri:%d\n",sayi);
10.    printf("sayi değişkeninin adresi:%p\n",&sayi);
11.    printf("pointerın sakladığı :%p\n",pointer);
12.    printf("pointerın işaret ettiği değer:%d\n",*pointer);
13.    printf("pointer adresi:%p\n",&pointer);
14.
15.
16.    return 0;
17.
18. } //main fonksiyonu sonu
19.
```

ÇIKTI:



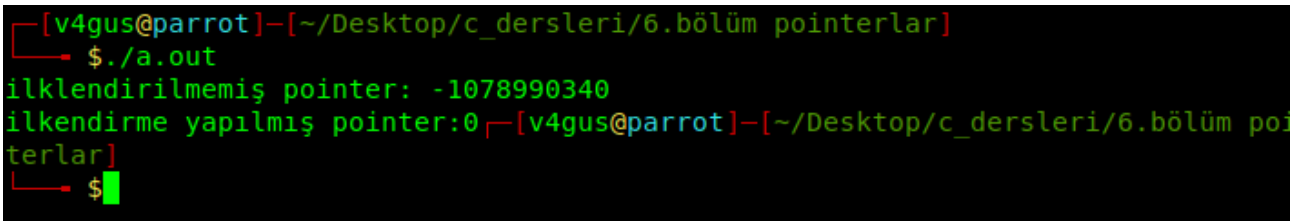
```
[v4gus@parrot]~/. Desktop/c_dersleri/6.bölüm pointerlar
➔ ./a.out
sayi değeri:70
sayi değişkeninin adresi:0xbfdeb9c8
pointerın sakladığı :0xbfdeb9c8
pointerın işaret ettiği değer:70
pointer adresi:0xbfdeb9cc
[v4gus@parrot]~/. Desktop/c_dersleri/6.bölüm pointerlar
➔ $
```

Eğer bir pointer tanımlamışsak mutlaka ilklendirme yapılmalıdır ve bu ilklendirme NULL ataması ile yapılır. İlkendirme yapmamızın amacı pointerımızın tanımlandığı alanın dolu olabileceği ihtimaline karşıdır.Çünkü RAM bize o anda işletim sistemimizin kullanmadığı herhangi bir alan ayırır ve o alanın boş olup olmadığını kontrol etmez.Bu durumu test etmek için bir kod çıktısı inceleyebiliriz.

ÖRNEK 7.2:

```
1. #include <stdio.h>
2.
3. int main ()
4. {
5.     int *p;
6.     int *q=NULL;
7.
8.     printf("ilkendirilmemiş pointer: %d\n",p);
9.     printf("ilkendirme yapılmış pointer:%d",q);
10. } //main fonksiyonu sonu
11.
```

ÇIKTI:



```
[v4gus@parrot]~[~/Desktop/c_dersleri/6.bölüm pointerlar]
└─ $ ./a.out
ilkendirilmemiş pointer: -1078990340
ilkendirme yapılmış pointer:0 [v4gus@parrot]~/Desktop/c_dersleri/6.bölüm pointerlar]
└─ $
```

görüldüğü üzere ilkendirilmemiş pointerların değerlerini 0 olarak varsaymak büyük hatalara neden olabilir.

7.0. Diziler Ve Pointerlar

Dizileri ve pointerları birinin yerine kullanabileceğimiz birçok durum vardır. Diziler sabit boyutlu pointerlar olarak düşünülebilir. Dizileri pointerlara atayabiliriz.

Elimizde dizi[5] integer dizisi ve diziPointer integer pointerı olsun.

```
diziPointer = dizi;
```

ile dizimizin ilke elemanının adresini diziPointer'a atamış oluyoruz. Bu atama yerine

```
diziPointer = &dizi[0];
```

ataması da kullanılabilir. Diziler pointerlara atandıktan sonra , dizi indislerine erişmek için kullandığımız aritmetik operatörleri pointerlarla da kullanabiliriz. Mesela dizimizin 4. elemanına erişmek için:

`*(dizi+4)` kullanımı `dizi[4]` ile aynı işlevi yerine getirir.

Bir diğer yöntem olarak offset'tir, yine dizimizin 4. elemanına erişmeye çalışalım:

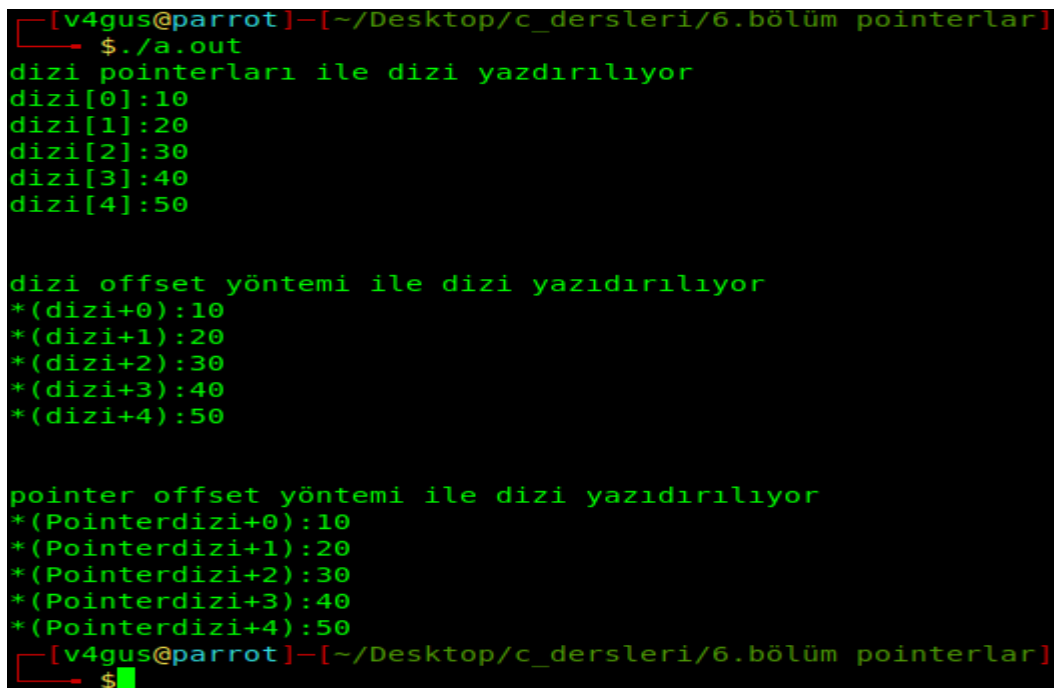
```
*(diziPointer+4);
```

Bu işlemleri

Buraya kadar anlattıklarımızı ve diğer yöntemleri kullandığımız bir program yazalım :

ÖRNEK 7.3:

```
1. //DİZİ ELEMANLARINI POİNERLARA ATAMAK
2.
3. #include <stdio.h>
4.
5. int main ()//main fonksiyonu başlangıcı
6. {
7.     int dizi[]={10,20,30,40,50};
8.     int *diziPointer=dizi; // diziPointer dizi'yi işaret etsin
9.     int i ; //döngü değişkenleri
10.
11.     printf("dizi pointerları ile dizi yazdırılıyor\n");//dizi pointerları ile dizi yazmak
12.
13.     for(i=0;i<5;i++)
14.         printf("dizi[%d]:%d\n",i,dizi[i]);
15.
16.     printf("\n\ndizi offset yöntemi ile dizi yazdırılıyor\n");//dizi offset yöntemi ile
    dizi yazmak
17.
18.     for(i=0;i<5;i++)
19.         printf("*(dizi+%d):%d\n",i,*(dizi+i));
20.
21.
22.     printf("\n\npointer offset yöntemi ile dizi yazdırılıyor\n");//pointer offset
    yöntemi ile dizi yazmak
23.
24.     for(i=0;i<5;i++)
25.         printf("*(Pointerdizi+%d):%d\n",i,*(diziPointer+i));
26.
27.     return 0;
28.
29. } // main fonksiyonu sonu
```



```
[v4gus@parrot]--[~/Desktop/c_dersleri/6.bölüm pointerlar]
$ ./a.out
dizi pointerları ile dizi yazdırılıyor
dizi[0]:10
dizi[1]:20
dizi[2]:30
dizi[3]:40
dizi[4]:50

dizi offset yöntemi ile dizi yazdırılıyor
*(dizi+0):10
*(dizi+1):20
*(dizi+2):30
*(dizi+3):40
*(dizi+4):50

pointer offset yöntemi ile dizi yazdırılıyor
*(Pointerdizi+0):10
*(Pointerdizi+1):20
*(Pointerdizi+2):30
*(Pointerdizi+3):40
*(Pointerdizi+4):50
[v4gus@parrot]--[~/Desktop/c_dersleri/6.bölüm pointerlar]
$
```

ÇIKTI:

Pointerlar yalnızca değişkenlerin değil başka bir pointerın ve fonksiyonları da işaret ederler. Pointerın pointerını tanımlamak ise :

```
int sayi =70;  
int *P=&sayi;  
int **ptrP=&P;
```

şeklinde yapılabilir. Bu tanımlama karmaşık gelse de dinamik dizilerde çok boyutlu dizi tanımlaması kullanırken bu yöntemi kullanacağız. Kısaca P pointerı sayi değişkeninin adresini tutarken ptrP pointerı P nin adresinde bulunan adresi gösterir.

Değişken İsmi	Değişken Adresi	Değişken Değeri
sayi	3051	70
*P	4065	3051
**ptrP	6045	4065

ÖRNEK 7.3:

```
1.  
2. //POINTERİ GÖSTEREN POINTERLAR  
3.  
4. #include <stdio.h>  
5.  
6. int main ()  
7. {  
8.  
9.  
10.     int sayi=70;  
11.     int *ptr=&sayi;  
12.     int **ptrPtr=&ptr;  
13.  
14.  
15.  
16.     printf("sayi:%d\nptr:%d\nptrPtr:%d",sayi,*ptr,**ptrPtr );  
17.  
18.     return 0;  
19. } //main fonksiyonu sonu  
20.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/6.bölüm pointerlar]
→ ./a.out
sayi:70
ptr:70
ptrPtr:70 [v4gus@parrot]--[~/Desktop/c_dersleri/6.bölüm pointerlar]
```

8. DİNAMİK BELLEK YÖNETİMİ

Temel veri tiplerinden herahgi birisini tanımladığımız anda stack (yığın) denilen alanda derleyici tarafından otomatik boyutlu olarak tanımlanır.Bu alanlar genellikle sabit alanlardır ve porgram çalıştırılırken grevlerini yerine getirirler dahi hafızada yer kaplamaya devam ederler.Büyük boyutlu ya da byte'ların bile önemli olduğu mikro işlemci programlama gibi durumlarda bize büyük sorun teşkil eder. Bu sorunu aşmak için kullanacağımız dizileri dinamik olarak tanımlamak bize hem hafızadan tasarruf sağlayacaktır hem de dizinin bütölüp küçültölmesi durumunda sürekli dizi tanımlamamızın önüne geçecektir..İlkel veri tiplerinin sistemimde (Parrot GNU/Linux 4.9 / 32-bit)ne kadar alan kapladığını görmek için sizeof() fonksiyonu ile çıktı alıyorum.

ÖRNEK 8.1:

```
1. //İLKEL VERİ TİPLERİNİN BOYUTUNU ÖĞRENME
2.
3. #include <stdio.h>
4.
5. int main ()//main fonksiyonu başlangıcı
6. {
7.     int integer=70;
8.     float flo=70.000;
9.     char c='A';
10.
11.     printf("integer:%d\nfloat:%d\nchar:%d\n",sizeof(integer),sizeof(flo),sizeof(c));
12.
13.     return 0;
14. } //main fonksiyonu sonu
15.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/7.Bölüm dinamik bellek]
→ ./a.out
integer:4
float:4
char:1
[v4gus@parrot]--[~/Desktop/c_dersleri/7.Bölüm dinamik bellek]
→ $
```

Dinamik bellek alalnını program çalışırken ayırılması işlemine allocation denir.Bellek

alanının gerek duyulmadığı zaman serbest bırakılmasına da deallocation denir. Programın çalışması esnasında belleğin bu yöntemler kullanılarak yönetilmesine , dinamik bellek yönetimi (dynamic memory allocation) denir. Yönetim gerçekleştirilirken standart kütüphane fonksiyonlarından malloc, calloc, realloc ve free fonksiyonları kullanılır.

8.1 MALLOC FONKSİYONU

Malloc fonksiyonu , kendisine byte cinsinden girilen argüman kadar bellek alanı ayırmaya çalışır ve bellek ayrılması başarılı olursa ayrılan alanın başlangıç adresini döndürür. Eğer alan ayırma işlemi başarılı olmazsa adres değeri (NULL) olarak döner. Prototipi:

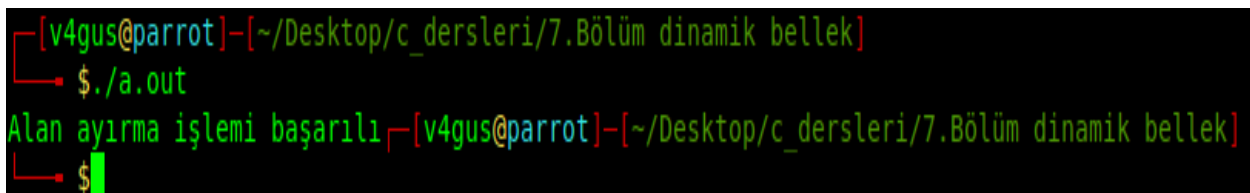
```
void *malloc(boyut);
```

şeklinde kullanılır. Malloc fonksiyonunu kullanarak 1024 byte'lık integer tipinde bir dizi tanımlaya çalışalım , alan ayırma işleminin başarılı olup olmadığını yazdıralım.

ÖRNEK 8.2:

```
1. //MALLOC FONKSİYONU İLE 10 ELEMANLI İNTEGER DİZİSİ OLUŞTURAN
   PROGRAM
2.
3. #include <stdio.h>
4. #include <stdlib.h>
5.
6. int main ()//main fonksiyonu başlangıcı
7. {
8.
9.     int *p;
10.    p=malloc(1024);
11.
12.    if(p==NULL)
13.    {
14.        printf("Bellek yetersiz !...\n");
15.        exit(1);
16.    }
17.
18.    else
19.        printf("Alan ayırma işlemi başarılı");
20.
21.    return 0;
22.
23. } //main fonksiyonu sonu
```

ÇIKTI:



```
[v4gus@parrot]~-[~/Desktop/c_dersleri/7.Bölüm dinamik bellek]
└─$ ./a.out
Alan ayırma işlemi başarılı-[v4gus@parrot]~-[~/Desktop/c_dersleri/7.Bölüm dinamik bellek]
└─$
```


Bellek yönetim çeşitleri işletim sistemleri arasında çeşitlilik gösterdiği için programımızın taşınabilirliği büyük bir tehlike altına girmiş oluyor.Örneğin bizim

```
int *p ;  
p=malloc(100);
```

ile bazı işletim sistemlerinde 50 elemanlı , bazı işletim sistemlerinde ise 25 elemanlı bir integer boyutu tahsis edilir.Bu gibi sıkıntıların önüne geçmek için 50 elemanlık bir integer blok tahsis etmek istersek "sizeof()" fonksiyonunu kullanırız.

```
int *p ;  
p=malloc(sizeof(int)*10);
```

şeklinde malloc ve sizeof fonksiyonları birbirini tamamlayan iki fonksiyon gibi düşünebiliriz.

8.2. CALLOC FONKSİYONU

Calloc fonksiyonu çalışma bakımından malloc fonksiyonuna benzer.İlk parametre ile belirtilen sayının çarpımı kadar bellekten alan tahsis edilir.Fonksiyon prototipi:

```
void * calloc (sayı * boyut);
```

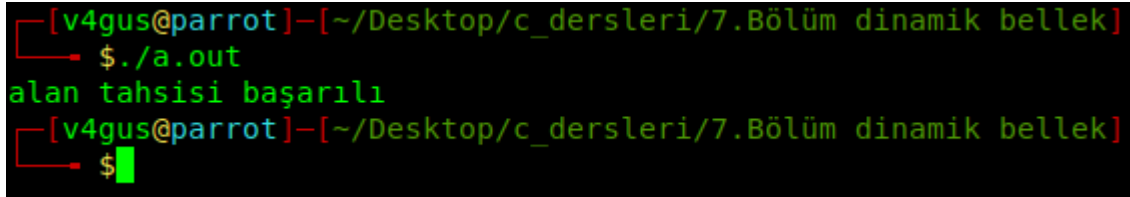
şeklinde dir.Tekrar 50 elemanlı bir integer blok tahsis etmeye çalışalım ve başarı durumunu yazdıralım.

ÖRNEK 8.3:

```
1. //CALLOC FONKSİYONU İLE BELLEKTEN ALAN TAHSİSİ  
2.  
3. #include <stdio.h>  
4. #include <stdlib.h>  
5.  
6. int main () // main fonksiyonu başlangıcı  
7. {  
8.     int *pointer;  
9.  
10.    pointer=calloc(50,sizeof(int));  
11.  
12.    if(pointer==NULL)  
13.    {  
14.        printf("bellek yetersiz!...\n");  
15.        exit(1);  
16.    }  
17.  
18.    else  
19.    {  
20.        printf("alan tahsisi başarılı\n");  
21.    }
```

```
22.  
23.     return 0;  
24.} // main fonksiyonu sonu  
25.
```

ÇIKTI:



```
[v4gus@parrot]~/.  
→ ./a.out  
alan tahsisi başarılı  
[v4gus@parrot]~/.  
→ $
```

8.3 REALLOC FONKSİYONU

Daha önceden malloc ve calloc fonksiyonları ile ayrılmış olan bellek alanını büyütme ya da küçültme amacıyla kullanılan fonksiyondur. Prototipi:

```
void *realloc(void *blok, yeniBoyut);
```

şeklindedir.

İlk parametersi daha önceden ayrılmış bellek bloğunun başlangıcı, ikinci parametresi ise bloğun yeni uzunluğudur. Çalışma şekli; istenilen bellek bloğu için yeterli alan arar bulursa bulduğu alana eski bellek bloğunu ve eklenecek kısmı kesintisiz şekilde yerleştirir böylece tahsisi tamamlanmış olur. Yeterli alanı bulamazsa NULL döndürür.

char tipinde bir pointer üzerine önce malloc fonksiyonunu kullanarak 10 byte alalım sonrasında realloc ile 5 byte'a düşürelim. İşlemimizin başarılı oluşup olmadığını da çıktı olarak alalım.

ÖRNEK 8.4:

```
1.  
2. //REALLOC FONKSİYONU KULLANIMI  
3.  
4. #include <stdio.h>  
5. #include <stdlib.h>  
6.  
7. int main () // main fonksiyonu başlangıcı  
8. {  
9.     int i; //döngü değişkeni  
10.    char *c;  
11.    c=malloc(10);  
12.  
13.    if(!c)  
14.    {  
15.        printf("Bellek yetersiz!..\n");  
16.        exit(1);  
17.    }
```


alalım ve girilen değerleri yazdırıp dizinin alanını free ile iade edelim.

ÖRNEK 8.5:

```
1. //STANDART KÜTÜPHANAE FONKSİYONLARINI KULLANARAK DİNAMİK
   BELLEK YÖNETMİ
2.
3. #include <stdio.h>
4. #include <stdlib.h>
5.
6. int main () //main fonksiyonu başlangıcı
7. {
8.
9.     int * dizi=malloc(sizeof(int)*5); // dizi pointerını 5 elemanlık bir dizi haline
   getiriyoruz
10.    int i; //döngü değişkeni
11.
12.    dizi=realloc(dizi,10); //dizimizi 10 elemanlık olacak şekilde genişletiyoruz
13.
14.    if(!dizi)
15.    {
16.        printf("bellek yeterli değil!..\n");
17.        exit(1);
18.    }
19.
20.    for(i=0;i<10;i++)
21.    {
22.        printf("dizinin %d.elemanını giriniz:",i+1);
23.        scanf("%d",&dizi[i]);
24.    }
25.
26.
27.    printf("\n\n%s\n\n","Girilen dizi yazdırılıyor....");
28.
29.    for(i=0;i<10;i++)
30.        printf("dizinin %d.elemanı:%d\n",i+1,dizi[i]);
31.
32.    printf("%s\n","Dizi siliniyor");
33.
34.    free(dizi); //dizinin alanını geri iade ediyoruz
35.
36.
37.
38.
39.
40.    return 0;
41.
42.
43. } //main fonksiyonu sonu
44.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/7.Bölüm dinamik bellek]
$ ./a.out
dizinin 1.elemanını giriniz:70
dizinin 2.elemanını giriniz:34
dizinin 3.elemanını giriniz:65
dizinin 4.elemanını giriniz:8
dizinin 5.elemanını giriniz:65
dizinin 6.elemanını giriniz:78
dizinin 7.elemanını giriniz:54
dizinin 8.elemanını giriniz:78
dizinin 9.elemanını giriniz:56
dizinin 10.elemanını giriniz:78

Girilen dizi yazdırılıyor....

dizinin 1.elemanı:70
dizinin 2.elemanı:34
dizinin 3.elemanı:65
dizinin 4.elemanı:8
dizinin 5.elemanı:65
dizinin 6.elemanı:78
dizinin 7.elemanı:54
dizinin 8.elemanı:78
dizinin 9.elemanı:1769630052
dizinin 10.elemanı:544106862
Dizi siliniyor
[v4gus@parrot]--[~/Desktop/c_dersleri/7.Bölüm dinamik bellek]
$
```

9. STRINGLER

Char dizilerine string ya da karakter katarı denir.Stringler harfleri , rakamları ve özel karakterleri tutabilirler.String atamalarında atnacak olan kelime çift tırnak içerisinde yazılır. Stringlerin isimlendirilmesinde değişken isimlendirme kuralları geçerlidir.Stringler NULL karakteri yani "\0" ile biterler. Bu durumda 29 karakterlik bir string için 30 karakterlik yer ayırmamız gerekecektir.Stringler karakter dizileri oldukları için dizilerde yaptığımız tüm işlemleri bu yapılarda da yapabiliriz.Bir string tanımı :

```
char string[ 50]="C dilinde string ifadeler";
```

şeklinde yapılır.

Eğer standart veri girişinden string okumak istiyorsak scanf fonksiyonu "%s" parametresi ile kullanılır. Kullanıcıdan ismini ve soy ismini alıp , isminin ve soy isminin ilk harflerini değiştiren bir program yazalım.

ÖRNEK 9.1:

- 1.
2. //KULLANICIDAN İSMİNİ VE SOY İSMİNİ ALIP İLK HARFLERİNİ DEĞİŞTİRİP YAZAN PROGRAM
- 3.
4. #include <stdio.h>
5. #include <string.h>
- 6.

```

7. int main () //main fonksiyonu başlangıcı
8. {
9.     char isim[50];
10.    char soy_isim[50];
11.    char temp;
12.
13.    printf("%s\n","lutfen isminizi giriniz:");
14.    scanf("%s",isim);
15.
16.
17.    printf("%s\n","lutfen soy isminizi giriniz:");
18.    scanf("%s",soy_isim);
19.
20.    //isim ve soy ismin ilk harfleri değiştiriliyor
21.
22.    temp=isim[0];
23.    isim[0]=soy_isim[0];
24.    soy_isim[0]=temp;
25.
26.
27.    printf("\nisim:%s\nsoy isim:%s\n",isim,soy_isim);
28.
29.    return 0;
30.
31.} // main fonksiyonu sonu

```

ÇIKTI:



```

[v4gus@parrot]~-[~/Desktop/c_dersleri/8.bölüm stringler]
$ ./a.out
lutfen isminizi giriniz:
ismet
lutfen soy isminizi giriniz:
arslan

isim:asmet
soy isim:irslan
[v4gus@parrot]~-[~/Desktop/c_dersleri/8.bölüm stringler]
$

```

Dizilerde yapabildiğimiz tüm işlemleri stringler ile de yapabileceğimizi söylemiştik.Örneğin dinamik bir string oluşturabiliriz.

ÖRNEK 9.2:

```

1. // DİNAMİK OLARAK STRING TANIMLAMA
2.
3. #include <stdio.h>
4. #include <stdlib.h>

```

```

5.
6. int main () // main fonksiyonu başlangıcı
7. {
8.     char *str = malloc(sizeof(char)*20); //dinamik olarak 20 karakterlik bir string
        alanı ayırdık
9.
10.    str="Dennis Ritchie";
11.
12.    printf("str:%s\n",str);
13.
14.
15.
16.
17.    return 0;
18. } //main fonksiyonu sonu
19.

```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
$ ./a.out
str:Dennis Ritchie
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
$
```

9.0. GETS FONKSİYONU

Buraya kadar kullandığımız scanf fonksiyonunun boşluk ve yeni sekme gibi karakterleri gönderdiğimiz anda okuma işlemini durdurmasıdır."Dennis Ritchie" olarak bir girdi vermeye çalışırsak scanf fonksiyonu yalnızca "Dennis" i okuyacaktır.Böyle durumlarda gets() fonksiyonunu kullanırız.gets() fonksiyonu enter'a basıldığında veri girişini durdurur.Fonksiyon prototipi :

```
gets(string_ismi);
```

şeklindedir.

Gets kullanarak kullanıcıdan ismini alan ve yazdıran bir program yazalım.

ÖRNEK 9.3:

1. //GETS FONSKİYONU KULLANIMI
- 2.
3. #include <stdio.h>
- 4.
5. int main () //main fonksiyonu başlangıcı

```

6. {
7.     char isim1[50];
8.
9.     printf("isminizi giriniz:");
10.    gets(isim1);
11.    printf("\nbenim ismim %s", isim1);
12.
13.    return 0;
14.
15. } // main fonksiyonu sonu
16.

```

Derleyicimiz bizi gets fonksiyonunun güvenli bir yöntem olmadığı konusunda uyarıyor.

```

[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
$ gcc 45.c
45.c: In function 'main':
45.c:10:2: warning: implicit declaration of function 'gets'; did you mean
'fgets'? [-Wimplicit-function-declaration]
   10 |     gets(isim1);
       |     ^~~~
       |     fgets
/usr/bin/ld: /tmp/ccN5PAuW.o: in function 'main':
45.c:(.text+0x37): uyarı: the 'gets' function is dangerous and should not
be used.

```

Derleme işlemi yapıldığı için çalıştırabiliriz.

```

[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
$ ./a.out
isminizi giriniz:İsmet Arslan

benim ismim İsmet Arslan [v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm s
tingler]
$

```

Görüldüğü gibi giriş yaparken boşluk bıraktık ve boşluk karakteri ile okuma işlemi sonlanmıyor.

9.1. FGETS FONKSİYONU

Gets fonksiyonunun güvenli bir yoludur. Parametre olarak string ismini , kaç karakter girileceği ve girişin nereden yapılacağını alır. Prototipi:

```
fgets(string_ismi , karakter_sayısı , giriş(stdin) ) ;
```

şeklinde dir.

Az önce yaptığımız programı fgets kullanarak tekrar yazalım. Derleyicimizin hata vermediğini görelim.

ÖRNEK 9.4:

```

1.

```

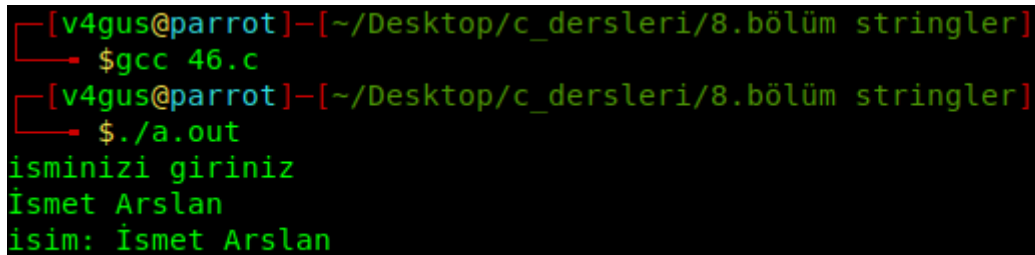


```

2. // FGETS FONKSİYONU KULLANIMI
3.
4. #include <stdio.h>
5. #define BOYUT 15
6.
7. int main()
8. {
9.     char isim[BOYUT];
10.
11.     printf("%s\n","isminizi giriniz");
12.     fgets(isim , BOYUT , stdin);
13.     printf("isim: %s\n", isim);
14.
15.     return 0;
16. } // main fonksiyonu sonu
17.

```

ÇIKTI:



```

[v4gus@parrot]~-[~/Desktop/c_dersleri/8.bölüm stringler]
→ gcc 46.c
[v4gus@parrot]~-[~/Desktop/c_dersleri/8.bölüm stringler]
→ ./a.out
isminizi giriniz
İsmet Arslan
isim: İsmet Arslan

```

9.2. FPUTS FONKSİYONU

fputs fonksiyonu , stringin adını ve içeriğini yazdırmak için pointer kullanan bir fonksiyondur.Fonksiyonun prototipi:

```
fputs(string_ismi , çıkış(stdout));
```

şeklinde.Daha önce yazdığımız programı printf fonksiyonu yerine fputs kullanarak çıktı verecek şekilde yeniden yazalım.

ÖRNEK 9.5:

```

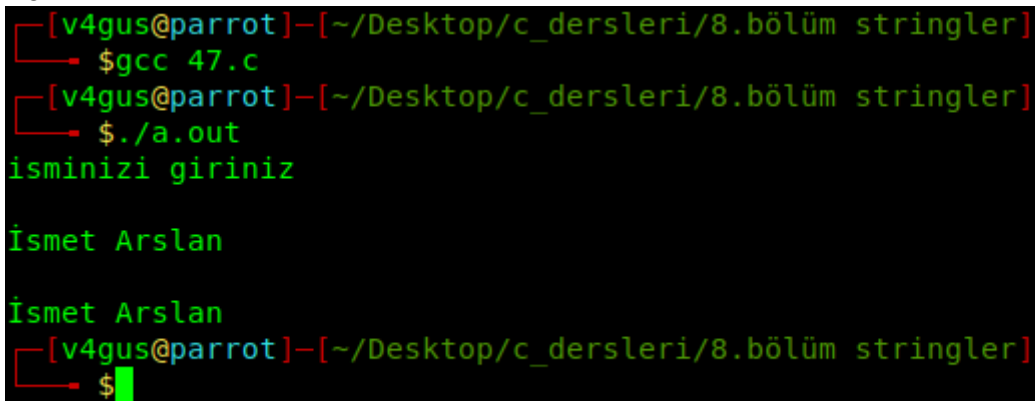
1.
2. // FPUTS FONKSİYONU KULLANIMI
3.
4. #include <stdio.h>
5. #define BOYUT 25
6.
7. int main()
8. {
9.     char isim[BOYUT];
10.

```

```

11. printf("%s\n","isminizi giriniz");
12.
13. printf("\n");
14.
15. fgets(isim , BOYUT , stdin);
16.
17. printf("\n");
18.
19. fputs(isim , stdout);
20.
21. return 0;
22.} // main fonksiyonu sonu
23.

```



```

[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
→ gcc 47.c
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
→ ./a.out
isminizi giriniz

İsmet Arslan

İsmet Arslan
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
→ $

```

9.3. PUTS FONKSİYONU

Puts fonksiyonu , tanımlı olan çıkış birimine stringi yazdırır ve imleci yeni satırın başına getirir. Prototipi :

```
puts(string_ismi);
```

şeklindedir.

Kullanıcıdan isimini alan ve puts ile yazdıran programı yazalım.

ÖRNEK 9.6:

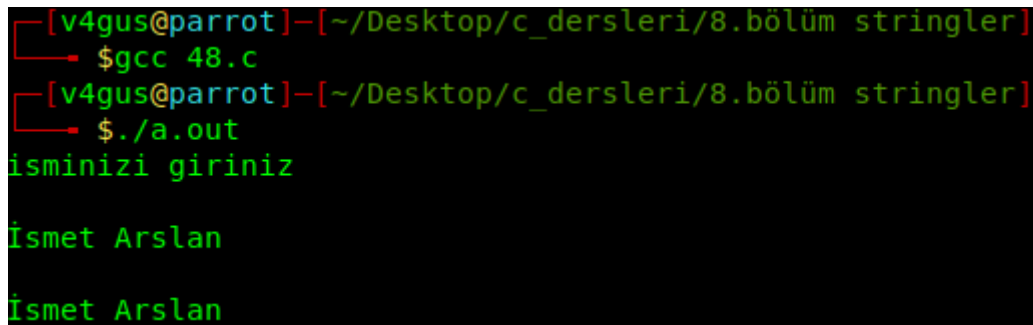
```

1.
2. // PUTS FONKSİYONU KULLANIMI
3.
4. #include <stdio.h>
5. #define BOYUT 25
6.
7. int main()
8. {
9.     char isim[BOYUT];

```

```
10.  
11. printf("%s\n","isminizi giriniz");  
12.  
13. printf("\n");  
14.  
15. fgets(isim , BOYUT , stdin);  
16.  
17. printf("\n");  
18.  
19. puts(isim);  
20.  
21. return 0;  
22.} // main fonksiyonu sonu  
23.
```

ÇIKTI:



```
[v4gus@parrot]~[~/Desktop/c_dersleri/8.bölüm stringler]  
→ $gcc 48.c  
[v4gus@parrot]~[~/Desktop/c_dersleri/8.bölüm stringler]  
→ $./a.out  
isminizi giriniz  
İsmet Arslan  
İsmet Arslan
```

9.3. STRING.H KÜTÜPHANESİ

Stringleri karşılaştırmak , string içerisinde karakter ya da string aramaları yapmak için kullanılabileceğimiz fonksiyonlar sunan bir kütüphanedir. Programımıza

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

ifadesi ile ekleyebilir ve içerisindeki fonksiyonları kullanabiliriz.

9.3.0. STRLEN

Parametre olarak aldığı string ifadenin eleman sayısını döndürür. Her string ifadenin sonunda bulunan "\0" ifadesi ile sonlanır. Daha önce bahsettiğimiz sizeof fonksiyonunun byte türünde boyut döndürdüğü unutulmamalıdır. Fonksiyonun prototipi:

```
size_t strlen(const char *string);
```

şeklinde dir.

Sizeof ve strlen arasındaki farkın daha iyi anlaşılabilmesi için bir string oluşturup iki fonksiyonunun çıktısını inceleyelim.

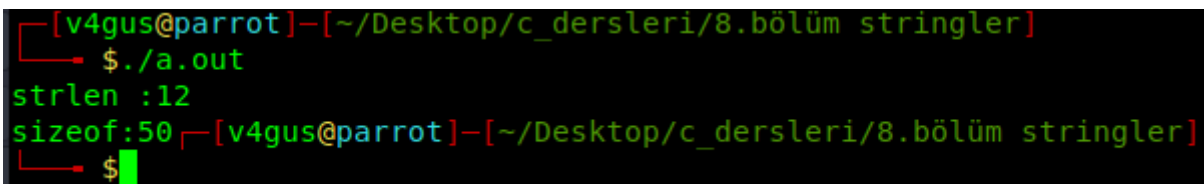
ÖRNEK 9.7:

```

1.
2. //STRLEN FONKSİYONU VE SIZEOF FARKI
3.
4. #include <stdio.h>
5. #include <string.h>
6.
7. #define BOYUT 50
8.
9. int main (void) //main fonksiyonu başlangıcı
10.{
11.     char str[BOYUT]="nervus vagus";
12.     printf("strlen :%d\nsizeof:%d",strlen(str),sizeof(str));
13.
14.     return 0;
15.
16. } //main fonksiyonu sonu
17.

```

ÇIKTI:



```

[v4gus@parrot] - [~/Desktop/c_dersleri/8.bölüm stringler]
→ ./a.out
strlen :12
sizeof:50 [v4gus@parrot] - [~/Desktop/c_dersleri/8.bölüm stringler]
→ $

```

String içerisine 10. kraniyal ve en uzun sinir çifti olan "nervus vagus" ifadesini atıyoruz. Fonksiyonların çıktıları da açıkça gösteriyor ki strlen string içerisindeki dolu olan uzunluğu verirken, sizeof dizinin byte türünden ve dolu olup olmadığının kontrolünü yapmadan boyutu döndürür.

9.3.1. STRNLEN

Stringte parametre olarak verilen değer kadar karakter varsa girilen değeri yoksa stringin boyutunu döndüren fonksiyondur. Prototipi :

```
size_t strnlen(const char *string, size_t maksimum_uzunluk);
```

şeklindedir.

ÖRNEK 9.8:

```

1. //STRNLEN
2.
3. #include <stdio.h>
4. #include <string.h>
5.
6. #define BOYUT 50
7.
8. int main (void) //main fonksiyonu başlangıcı

```

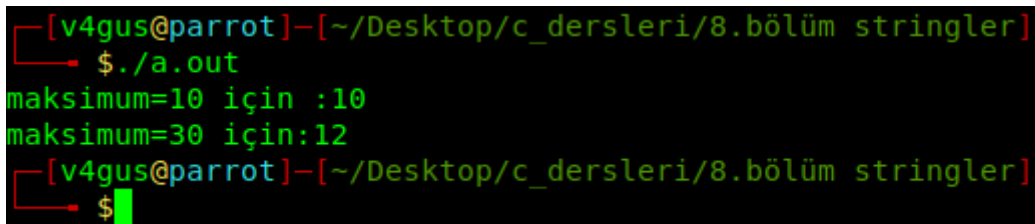
```

9. {
10.     char str[BOYUT]="nervus vagus";
11.     printf("maksimum=10 için :%d\n",strnlen(str,10));
12.     printf("maksimum=30 için:%d\n",strnlen(str,30));
13.
14.     return 0;
15.
16. } //main fonksiyonu sonu
17.

```

12 karakterlik bir ifade olan "nervus vagus" u string'e atıyoruz ve önce maksimum değeri 10 sonra da 30 olarak çıktı vermesini istiyoruz.

ÇIKTI:



```

[v4gus@parrot]~/. Desktop/c_dersleri/8.bölüm stringler
➔ ./a.out
maksimum=10 için :10
maksimum=30 için:12
[v4gus@parrot]~/. Desktop/c_dersleri/8.bölüm stringler
➔ $

```

9.3.2. STRCMP

İsminden de anlaşılacağı gibi iki stringi karşılaştıran fonksiyondur. Parametre olarak iki string alır. Karşılaştırılan iki string eşitse 0, eşit değilse karşılaştırmanın türüne bağlı olarak pozitif veya negatif bir değer döndürür. Prototipi:

```
int strcmp(const char *string1, const char *string2);
```

şeklinde dir.

Şimdi str1'e "nervus vagus", str2'ye "nervus olfactorius" ifadelerini atayıp iki stringi karşılaştıran ve eğer eşitseler stringler eşit, eşit değilse stringler eşit değil yazdıralım.

ÖRNEK 9.9:

```

1.
2. //STRCPM
3.
4. #include <stdio.h>
5. #include <string.h>
6.
7. #define BOYUT 50
8.
9. int main (void) //main fonksiyonu başlangıcı
10. {
11.     char str1[BOYUT]="nervus vagus";
12.     char str2[BOYUT]="nervus olfactorius";

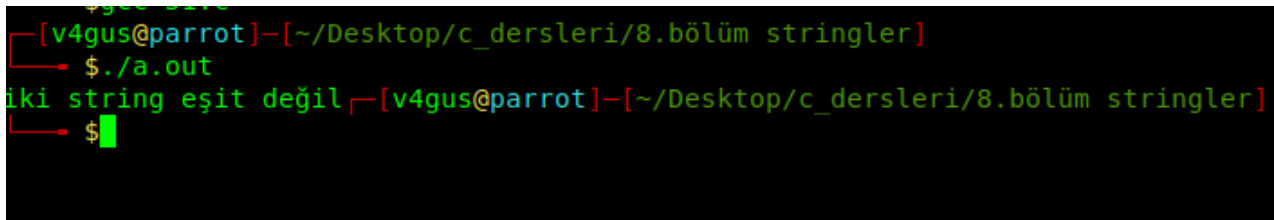
```

```

13.
14.     if(0==strcmp(str1,str2))
15.         printf("iki string eşit");
16.     else
17.         printf("iki string eşit değil");
18.
19.     return 0;
20.
21. } //main fonksiyonu sonu
22.

```

ÇIKTI:



```

[v4gus@parrot]~[~/Desktop/c_dersleri/8.bölüm stringler]
└─$ ./a.out
iki string eşit değil [v4gus@parrot]~[~/Desktop/c_dersleri/8.bölüm stringler]
└─$

```

9.3.3. STRNCMP

İki stringin n tane elemanını karşılaştırmak için kullanılan fonksiyondur. Parametre olarak iki string ve karşılaştırılması istenilen karakter sayısını alır. Prototipi:

```
int strncmp(const char *str1 , const char *str2 , size_t n);
```

şeklindedir.

Bir önceki örneğimizi bu sefer 6 karakteri karşılaştıracak şekilde değiştirip çıktısını inceleyelim.

ÖRNEK 9.10:

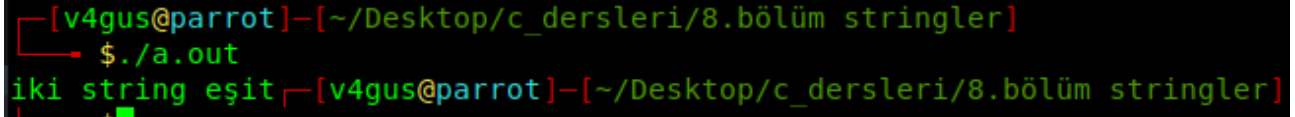
```

1.
2. //STRNCMP
3.
4. #include <stdio.h>
5. #include <string.h>
6.
7. #define BOYUT 50
8.
9. int main (void) //main fonksiyonu başlangıcı
10. {
11.     char str1[BOYUT]="nervus vagus";
12.     char str2[BOYUT]="nervus olfactorius";
13.
14.     if(0==strncmp(str1,str2,6))
15.         printf("iki string eşit");
16.     else
17.         printf("iki string eşit değil");

```

```
18.  
19.     return 0;  
20.  
21. } //main fonksiyonu sonu
```

ÇIKTI:



```
[v4gus@parrot]~/. Desktop/c_dersleri/8.bölüm stringler]  
$ ./a.out  
iki string eşit [v4gus@parrot]~/. Desktop/c_dersleri/8.bölüm stringler]
```

Stringlerin değerleri birbirinden farklı olsa da biz yalnızca 6 elemanı karşılaştırması istedik. Bu 6 eleman eşit olduğu için sonuç olarak "iki string eşittir" çıktısını aldık.

9.3.4. STRCAT

İki stringi birleştiren fonksiyondur. Parametre olarak birleştirilecek olan stringleri alır. İkinci ifadeyi birinci ifadeye ekler. Prototipi:

```
char *strcat(char *string1 , char *string2);
```

şeklindedir.

"nervus" ve "vagus" olarak iki farklı ifadeyi tutan iki stringi birleştirip birinci stringe atayan ve yazdıran bir program yazalım.

ÖRNEK 9.11:

```
1.  
2. //STRCAT  
3.  
4. #include <stdio.h>  
5. #include <string.h>  
6.  
7. #define BOYUT 50  
8.  
9. int main (void) //main fonksiyonu başlangıcı  
10. {  
11.     char str1[BOYUT]="nervus";  
12.     char str2[BOYUT]="vagus";  
13.  
14.  
15.  
16. strcat(str1,str2);  
17.  
18. printf("%s\n",str1);  
19.  
20.  
21.     return 0;  
22.  
23. } //main fonksiyonu sonu
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
→ ./a.out
nervusvagus
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
→ $
```

9.3.5. STRNCAT

İkinci stringin belirtilen n elemanını ilk string ile birleştiren fonksiyondur. Parametre olarak iki string ve eleman sayısını alır. Prototipi:

```
char *strncat(char *str1 , *str2 , int n);
```

şeklindedir.

Önceki örneğimizi ikinci ifadenin 2 elemanı ilk stringe ekleyecek şekilde değiştirelim.

ÖRNEK 9.12:

```
1.
2. //STRNCAT
3.
4. #include <stdio.h>
5. #include <string.h>
6.
7. #define BOYUT 50
8.
9. int main (void) //main fonksiyonu başlangıcı
10. {
11.     char str1[BOYUT]="nervus";
12.     char str2[BOYUT]="vagus";
13.
14.
15.
16. strncat(str1,str2,2);
17.
18. printf("%s\n",str1);
19.
20.
21.     return 0;
22.
23. } //main fonksiyonu sonu
24.
```

ÇIKTI:


```
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
└─ $./a.out
nervusva
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
└─ $
```

9.3.6. STRCPY

İkinci stringi birinci stringe kopyalayan fonksiyondur. Prototipi:

```
char *strcpy(char *str1, char *str2);
```

şeklindedir.

"deneme" stringinin üzerine "GNU/Linux" ifadesini kopyalayıp yazdıran programı kodlayalım.

ÖRNEK 9.13:

```
1. //STRCPY
2.
3. #include <stdio.h>
4. #include <string.h>
5.
6. #define BOYUT 50
7.
8. int main (void) //main fonksiyonu başlangıcı
9. {
10.     char str1[BOYUT]="deneme";
11.     char str2[BOYUT]="GNU/Linux";
12.
13.
14.
15. strcpy(str1,str2);
16.
17. printf("%s\n",str1);
18.
19.
20.     return 0;
21.
22.} //main fonksiyonu sonu
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
→ ./a.out
GNU/Linux
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
```

9.3.7. STRNCPY

İkinci stringin belirtilen kadar elemanını birinci stringe kopyalayan fonksiyondur. Bu fonksiyon için iki farklı durum vardır :

- 0- $\text{str2} > n$ ise ; str2 'nin n karakterini str1 'e kopyalar
- 1- $\text{str2} < n$ ise ; str2 'nin tüm karakterlerini str1 'e kopyalar

Prototipi:

```
char *strncpy(char *str1, char *str2, int n);
```

şeklindedir.

Örneğimizi str1 'in üzerine 4 karakter kopyalacak şekilde yeniden yazalım.

ÖRNEK 9.14:

```
1.
2. //STRNCPY
3.
4. #include <stdio.h>
5. #include <string.h>
6.
7. #define BOYUT 50
8.
9. int main (void) //main fonksiyonu başlangıcı
10. {
11.     char str1[BOYUT]="deneme";
12.     char str2[BOYUT]="GNU/Linux";
13.
14.
15.
16. strncpy(str1, str2, 4);
17.
18. printf("%s\n", str1);
19.
20.
21.     return 0;
22.
23. } //main fonksiyonu sonu
```

24.

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
└─ $ ./a.out
GNU/me
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
└─ $
```

9.3.8. STRCHR

String içerisinde doğrusal şekilde arama yapmak için kullanılan fonksiyondur. Tüm ifade integer'a dönüştürülerek arama gerçekleştirilir. Prototipi:

```
char *strchr(char *str , int karakter);
```

şeklindedir.

"GNU/Linux" stringi içerisinde "x" karakterini arayan programı yazalım.

ÖRNEK 9.15:

```
1. //STRCHR
2.
3. #include <stdio.h>
4. #include <string.h>
5.
6. #define BOYUT 50
7.
8. int main (void) //main fonksiyonu başlangıcı
9. {
10.     char str1[BOYUT]="GNU/Linux";
11.
12.
13.
14. printf("%s\n",strchr(str1,'x'));
15.
16.
17.     return 0;
18.
19. } //main fonksiyonu sonu
20.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
$ ./a.out
x
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
$
```

9.3.9. STRRCHR

String içerisinde tersten arama yapmak için kullanılan fonksiyondur. Tüm ifade integer'a dönüştürülerek arama gerçekleştirilir. Prototipi:

`char *strrchr(char *str , int karakter);`

şeklinde dir.

"GNU/Linux" stringi içerisinde "L" karakterini arayan programı yazalım.

ÖRNEK 9.16:

```
1. //STRRCHR
2.
3. #include <stdio.h>
4. #include <string.h>
5.
6. #define BOYUT 100
7.
8. int main (void) //main fonksiyonu başlangıcı
9. {
10.     char str1[BOYUT]="GNU/Linux kerneli Linus Torvalds tarafından yazıldı";
11.
12.
13.
14. printf("%s\n",strrchr(str1,'L'));
15.
16.
17.     return 0;
18.
19. } //main fonksiyonu sonu
20.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
- ./a.out
Linux Torvalds tarafından yazıldı
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
- $
```

9.4.10. STRSTR

String içerisinde string aramak için kullanılan fonksiyondur. Prototipi:

```
char *strstr(char *str , char *aranan_ifade);
```

şeklindedir.

Örneğimizi "GNU" ifadesini arayacak şekilde düzenleyelim.

ÖRNEK9.17:

```
1. //STRSTR
2.
3. #include <stdio.h>
4. #include <string.h>
5.
6. #define BOYUT 100
7.
8. int main (void) //main fonksiyonu başlangıcı
9. {
10.     char str1[BOYUT]="GNU/Linux kerneli Linus Torvalds tarafından yazıldı";
11.
12.
13.
14. printf("%s\n",strstr(str1,"GNU"));
15.
16.
17.     return 0;
18.
19. } //main fonksiyonu sonu
20.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
- ./a.out
GNU/Linux kerneli Linus Torvalds tarafından yazıldı
[v4gus@parrot]--[~/Desktop/c_dersleri/8.bölüm stringler]
- $
```

10. math.h KÜTÜPHANESİ

Birçok matematiksel fonksiyon bulunduran C kütüphanesidir.

10.0. ACOS FONKSİYONU

Kendisine parametre olarak verilen değerin arc cosinus değerini Radyan cinsinden döndürür.

Prototipi:

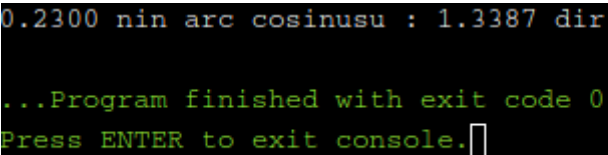
`double acos(double sayi);`

şeklindedir.

ÖRNEK 10.1:

```
1. //ACOS FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6.
7. int main(int argc, char const *argv[]) //main fonksiyonu başlangıcı
8. {
9.     double deger = 0.23;
10.    double radyan;
11.
12.    radyan=acos(deger);
13.
14.    printf("%.4lf nin arc cosinusunu : %.4lf dir",deger,radyan);
15.
16.
17.    return 0;
18.} //main fonksiyonu sonu
19.
```

ÇIKTI:



```
0.2300 nin arc cosinusunu : 1.3387 dir
...Program finished with exit code 0
Press ENTER to exit console.
```

10.1. ASİN FONKSİYONU

Kendisine parametre olarak verilen değerin arc sinus değerini Radyan cinsinden döndürür. Prototipi:

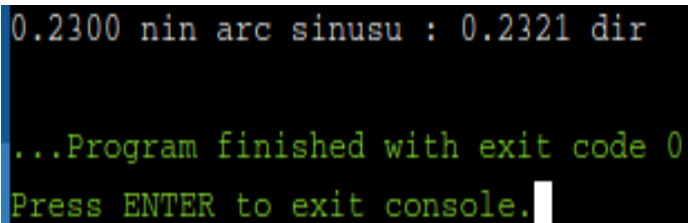
`double asin(double sayi);`

şeklindedir.

ÖRNEK 10.2:

```
1.
2. //ASİN FONKSİYONU
3.
4. #include <stdio.h>
5. #include <math.h>
6.
7.
8. int main(int argc, char const *argv[]) //main fonksiyonu başlangıcı
9. {
10.     double deger = 0.23;
11.     double radyan;
12.
13.     radyan=asin(deger);
14.
15.     printf("%.4lf nin arc sinusu : %.4lf dir",deger,radyan);
16.
17.
18.     return 0;
19. } //main fonksiyonu sonu
20.
```

ÇIKTI:



10.2.ATAN FONKSİYONU

Kendisine parametre olarak verilen değerin arc tanjant değerini Radyan cinsinden döndürür.

Prototipi:

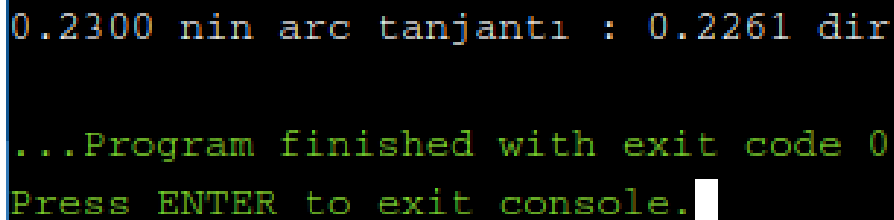
double atan(double sayi);

şeklindedir.

ÖRNEK 10.3:

```
1. //ATAN FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6.
7. int main(int argc, char const *argv[]) //main fonksiyonu başlangıcı
8. {
9.     double deger = 0.23;
10.    double radyan;
11.
12.    radyan=atan(deger);
13.
14.    printf("%.4lf nin arc tanjantı : %.4lf dir",deger,radyan);
15.
16.
17.    return 0;
18.} //main fonksiyonu sonu
19.
```

ÇIKTI:



```
0.2300 nin arc tanjantı : 0.2261 dir
...Program finished with exit code 0
Press ENTER to exit console.
```

10.3.COS FONKSİYONU

Kendisine parametre olarak verilen değerin cosinus değerini Radyan cinsinden döndürür.

Prototipi:

double cos (double sayi);

şeklindedir.

ÖRNEK 10.4:

```
1.
2. //COS FONKSİYONU
3.
```



```

4. #include <stdio.h>
5. #include <math.h>
6.
7.
8. int main(int argc, char const *argv[]) //main fonksiyonu başlangıcı
9. {
10.     double deger = 0.23;
11.     double radyan;
12.
13.     radyan=cos(deger);
14.
15.     printf("%.4lf nin  cosinusunu : %.4lf dir",deger,radyan);
16.
17.
18.     return 0;
19. } //main fonksiyonu sonu
20.

```

ÇIKTI:

```

0.2300 nin  cosinusunu : 0.9737 dir

...Program finished with exit code 0
Press ENTER to exit console.

```

10.4. COSH FONKSİYONU

Kendisine parametre olarak verilen değerin hiperbolik cosinus değerini Radyan cinsinden döndürür.

Prototipi:

```
double cosh(double sayi);
```

şeklindedir.

ÖRNEK 10.5:

```

1. //COSH FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6.
7. int main(int argc, char const *argv[]) //main fonksiyonu başlangıcı
8. {

```

```

9.      double deger = 0.23;
10.     double radyan;
11.
12.     radyan=cosh(deger);
13.
14.     printf("%.4lf nin hiperbolik cosinusu : %.4lf dir",deger,radyan);
15.
16.
17.     return 0;
18. } //main fonksiyonu sonu
19.

```

ÇIKTI:

```

0.2300 nin hiperbolik cosinusu : 1.0266 dir

...Program finished with exit code 0
Press ENTER to exit console.

```

10.5 SİN FONKSİYONU

Kendisine parametre olarak verilen değerin sinus değerini Radyan cinsinden döndürür.

Prototipi:

```
double sin(double sayi);
```

şeklindedir.

ÖRNEK 10.6:

```

1. //SİN FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6.
7. int main(int argc, char const *argv[]) //main fonksiyonu başlangıcı
8. {
9.     double deger = 0.23;
10.    double radyan;
11.
12.    radyan=sin(deger);
13.
14.    printf("%.4lf nin sinusu : %.4lf dir",deger,radyan);
15.
16.

```

```
17.         return 0;
18.    } //main fonksiyonu sonu
19.
```

ÇIKTI:

```
0.2300 nin  sinusu : 0.2280 dir

...Program finished with exit code 0
Press ENTER to exit console.
```

10.6 SİNH FONKSİYONU

Kendisine parametre olarak verilen değerin hiperbolik sinus değerini Radyan cinsinden döndürür.

Prototipi:

```
double sinh(double sayi);
```

şeklindedir.

ÖRNEK 10.7:

```
1.  //SİNH FONKSİYONU
2.
3.  #include <stdio.h>
4.  #include <math.h>
5.
6.
7.  int main(int argc, char const *argv[]) //main fonksiyonu başlangıcı
8.  {
9.      double deger = 0.23;
10.     double radyan;
11.
12.     radyan=sinh(deger);
13.
14.     printf("%.4lf nin hiperbolik sinusu : %.4lf dir",deger,radyan);
15.
16.
17.     return 0;
18. } //main fonksiyonu sonu
19.
```

ÇIKTI:

```
0.2300 nin hiperbolik sinusu : 0.2320 dir

...Program finished with exit code 0
Press ENTER to exit console.
```

10.7. TANH FONKSİYONU

Kendisine parametre olarak verilen değerin hiperbolik tanjant değerini Radyan cinsinden döndürür.

Prototipi:

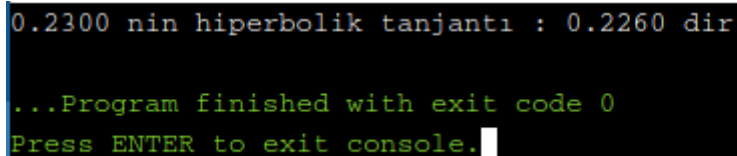
`double tanh(double sayi);`

şeklindedir.

ÖRNEK 10.8:

```
1.
2. //TANH FONKSİYONU
3.
4. #include <stdio.h>
5. #include <math.h>
6.
7.
8. int main(int argc, char const *argv[]) //main fonksiyonu başlangıcı
9. {
10.     double deger = 0.23;
11.     double radyan;
12.
13.     radyan=tanh(deger);
14.
15.     printf("%.4lf nin hiperbolik tanjantı : %.4lf dir",deger,radyan);
16.
17.
18.     return 0;
19. } //main fonksiyonu sonu
20.
```

ÇIKTI:



```
0.2300 nin hiperbolik tanjantı : 0.2260 dir
...Program finished with exit code 0
Press ENTER to exit console.
```

10.8. EXP FONKSİYONU

Girilen parametre değeri n ise en değerini döndürür.

Protitipi:

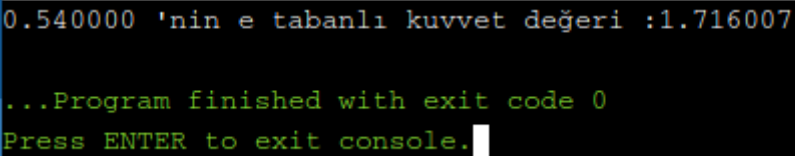
`double exp(double n)`

şeklindedir.

ÖRNEK 10.9:

```
1. // EXP FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6. int main (void) //main fonksiyonu başlangıcı
7. {
8.     double n=0.54;
9.     double deger=exp(n);
10.
11. printf("%lf 'nin e tabanlı kuvvet değeri :%lf",n,deger);
12.
13.     return 0;
14.
15. }//main fonksiyonu sonu
16.
```

ÇIKTI:



```
0.540000 'nin e tabanlı kuvvet değeri :1.716007
...Program finished with exit code 0
Press ENTER to exit console.
```

10.9 FREXP FONKSİYONU

Girilen parametre değeri n ise iki bölüme ayırıp ondalık kısmını döndürür. Yaptığı işlem : $n = \text{ondalık değer} * 2^{\text{kuvvet}}$ 'dir.

Prototipi :

`double frexp(double n, int *kuvvet) ;`

şeklindedir.

ÖRNEK 10.10:

```
1. //FREXP FONKSİYONU
2.
3. #include <stdio.h>
```

```

4. #include <math.h>
5.
6. int main (void) //main fonksiyonu başlangıcı
7. {
8.     double n=0.54;
9.     int eKuv;
10.    double deger=frexp(n,&eKuv);
11.
12. printf("%.2f = :%.2f * 2^%d",n,deger,eKuv);
13.
14.    return 0;
15.
16. }//main fonksiyonu sonu
17.

```

ÇIKTI:

```

0.54 = :0.54 * 2^0

...Program finished with exit code 0
Press ENTER to exit console.

```

10.10.LDEXP FONKSİYONU

Girilen n parametresini $n \cdot 2^{\text{exp}}$ işlemini yaparak sonucu döndürür.

Prototipi:

`double ldexp(double n , int kuvvet);`

biçimindedir.

ÖRNEK 10.11:

```

1.
2. //LDEXP FONKSİYONU
3.
4. #include <stdio.h>
5. #include <math.h>
6.
7. int main (void) //main fonksiyonu başlangıcı
8. {
9.     double n=76.54;
10.    double eKuv=5.00;
11.    double deger=ldexp(n,eKuv);
12.
13. printf("%.4f * 2^%.2f =%.4f",n,eKuv,deger);

```

```
14.
15.     return 0;
16.
17. }//main fonksiyonu sonu
18.
```

ÇIKTI:

```
76.5400 * 2^5.00 =2449.2800

...Program finished with exit code 0
Press ENTER to exit console.□
```

10.11. MODF FONKSİYONU

Girilen kayan noktalı sayının tam kısmını ile kesirli kısmını ayırır.

Prototipi:

```
double modf(double sayi1, double *tamSayi);
```

şeklindedir.

ÖRNEK 10.12:

```
1. //MODF FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6. int main (void) //main fonksiyonu başlangıcı
7. {
8.
9.     double n=564.55;
10.    double ondalik;
11.    double tam;
12.
13. ondalik=modf(n,&tam);
14.
15. printf("%.3f = %.3f + %.3f",n,tam,ondalik);
16.
17.     return 0;
18.
19. }//main fonksiyonu sonu
20.
```

ÇIKTI:

```
564.550 = 564.000 + 0.550

...Program finished with exit code 0
Press ENTER to exit console.□
```

10.12. LOG FONKSİYONU

Parametre olarak girilen değerin doğal (e tabanında) logaritmasını döndürür.

Prototipi:

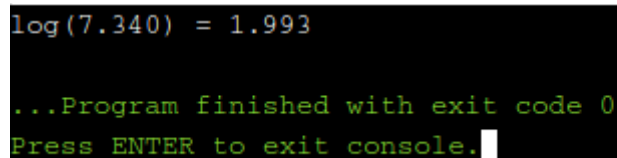
`double (double sayi);`

şeklindedir.

ÖRNEK 10.13:

```
1. //LOG FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6. int main (void)//main fonksiyonu başlangıcı
7. {
8.     double sayi =7.34;
9.     double deger;
10.
11.     deger=log(sayi);
12.
13. printf("log(%f) = %f",sayi,deger);
14.
15.
16.     return 0;
17.
18. } //main fonksiyonu sonu
19.
```

ÇIKTI:



```
log(7.340) = 1.993
...Program finished with exit code 0
Press ENTER to exit console.
```

10.13. LOG10 FONKSİYONU

Parametre olarak girilen değerin genel(10 tabanında) logaritmasını döndürür.

Prototipi:

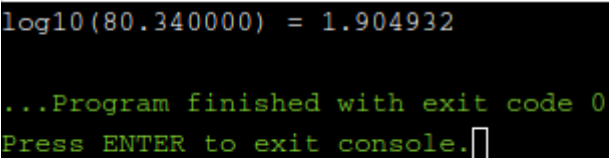
double log10(double sayi) ;

şeklindedir.

ÖRNEK 10.14:

```
1. //LOG FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6. int main (void)//main fonksiyonu başlangıcı
7. {
8.     double sayi =7.34;
9.     double deger;
10.
11.     deger=log(sayi);
12.
13. printf("log(%f) = %f",sayi,deger);
14.
15.
16.     return 0;
17.
18. } //main fonksiyonu sonu
19.
```

ÇIKTI:



```
log10(80.340000) = 1.904932
...Program finished with exit code 0
Press ENTER to exit console.█
```

10.14. POW FONKSİYONU

İlk parametrenin ikinci parametre kuvvetini hesaplar.

Prototipi:

double pow(double sayi , double n);

şeklindedir.

ÖRNEK 10.15:

```
1.
2. //POW FONKSİYONU
3.
```

```

4. #include <stdio.h>
5. #include <math.h>
6.
7. int main (void) //main fonksiyonu başlangıcı
8. {
9.
10.     double deger =2.44;
11.     double yeni;
12.     yeni=pow(deger,4);
13.
14. printf("%3.f^4 = %.3f",deger,yeni);
15.
16.
17.
18.     return 0;
19.
20. } //main fonksiyonu sonu
21.

```

ÇIKTI:

```

2.000000^4 = 16.000000
...Program finished with exit code 0
Press ENTER to exit console.

```

10.15. SQRT FONKSİYONU

İlk parametrenin ikinci parametre kökünü hesaplar.

Prototipi:

```
double sqrt(double sayi , double n);
```

şeklindedir.

ÖRNEK 10.16:

```

1. //SQRT FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6. int main (void) //main fonksiyonu başlangıcı
7. {
8.
9.     int deger =2.44;
10.    double yeni;
11.    yeni=sqrt(deger);
12.
13. printf("%d^(1/2) = %.3f",deger,yeni);
14.

```

```
15.  
16.  
17.     return 0;  
18.  
19. } //main fonksiyonu sonu
```

ÇIKTI:

```
2^(1/2) = 1.414  
  
...Program finished with exit code 0  
Press ENTER to exit console. █
```

10.16. CEİL FONKSİYONU

Girilen kayan noktalı sayının noktalı kısmına bakmaksızın bir üst tam sayıya yuvarlar.

Prototipi:

```
double ceil(double sayi);
```

şeklindedir.

ÖRNEK 10.17:

```
1. //CEİL FONKSİYONU  
2.  
3. #include <stdio.h>  
4. #include <math.h>  
5.  
6. int main (void) //main fonksiyonu başlangıcı  
7. {  
8.  
9.     double deger =2.44;  
10.    double yeni;  
11.    yeni=ceil(deger,4);  
12.  
13. printf("ilk deger: %3.1f\nceil fonksiyonu sonrası: %.3f\n",deger,yeni);  
14.  
15.  
16.  
17.     return 0;  
18.  
19. } //main fonksiyonu sonu
```

ÇIKTI:

```
ilk hali:2.440
ceil fonksiyonundan sonra:3.000

...Program finished with exit code 0
Press ENTER to exit console.
```

10.17. FABS FONKSİYONU

Parametre olarak girilen değerin mutlak değerini döndürür.

Prototipi :

double fabs(double sayi);

şeklindedir.

ÖRNEK 10.18:

```
1. //FABS FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6. int main (void) //main fonksiyonu başlangıcı
7. {
8.
9.     double deger =-2.44;
10.    double yeni;
11.    yeni=fabs(deger);
12.
13. printf("ilk deger: %f\nfabs fonksiyonu sonrası: %f\n",deger,yeni);
14.
15.
16.
17.    return 0;
18.
19.} //main fonksiyonu sonu
```

ÇIKTI:

```
ilk hali:-2.440000
fabs fonksiyonundan sonra:2.440000

...Program finished with exit code 0
Press ENTER to exit console.
```

10.18.FLOOR

Girilen noktalı sayının noktalı kısmına bakmaksızın bir alt tam sayıya yuvarlar.

Prototipi:

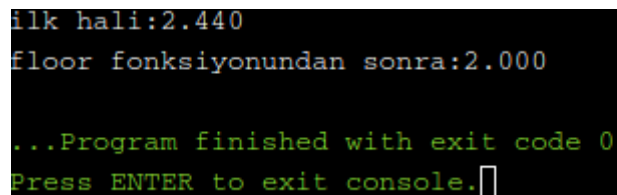
double floor (sayi1);

şeklindedir.

ÖRNEK 10.19:

```
1. //FLOOR FONKSİYONU
2.
3. #include <stdio.h>
4. #include <math.h>
5.
6. int main (void) //main fonksiyonu başlangıcı
7. {
8.
9.     double deger =2.44;
10.    double yeni;
11.    yeni=floor(deger,4);
12.
13.    printf("ilk deger: %3.f\nfloor fonksiyonu sonrası: %.3f\n",deger,yeni);
14.
15.
16.
17.    return 0;
18.
19. } //main fonksiyonu sonu
```

ÇIKTI:



```
ilk hali:2.440
floor fonksiyonundan sonra:2.000

...Program finished with exit code 0
Press ENTER to exit console.█
```

10.19.FMOD FONKSİYONU

Parametre olarak girilen değerlerin bölümünden kalanı verir.

Prototipi:

double fmod (double sayi1 , double sayi2);

şeklindedir.

ÖRNEK 10.20:

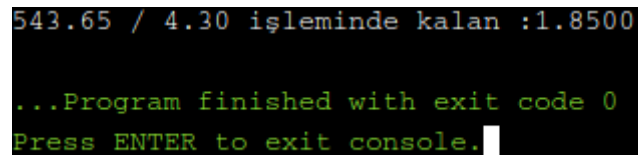
```
1.
2. //FMOD FONKSİYONU
```

```

3.
4. #include <stdio.h>
5. #include <math.h>
6.
7. int main () //main fonksiyonu başlangıcı
8. {
9.     double deger1=543.65;
10.    double deger2=4.3;
11.    double kalan;
12.
13.    kalan=fmod(deger1,deger2);
14.
15.    printf("%.2f / %.2f işleminde kalan :%.4f",deger1,deger2,kalan);
16.
17.    return 0;
18. } //main fonksiyonu sonu

```

ÇIKTI:



```

543.65 / 4.30 işleminde kalan :1.8500

...Program finished with exit code 0
Press ENTER to exit console.

```

11. YAPILAR (STRUCTURES)

Yapılar birden farklı veri türündeki değişkenleri barındıran gruptur. Mesela bizden öğrencilerin isimlerini , soy isimlerini , numaralarını , yaşlarını ve kimlik numaralarını alan ve istenilen eriyi yazdıran programı yazmamız istensin. Önümüzde iki seçenek vardır: ya oturup her öğrenci için tüm istenilenler için ayrı ayrı değişkenler oluşturacağız ya da öğrenci isimli istenilen verileri alan bir struct oluşturacağız. Ve her öğrenci için bir defa çağıracağız. İlk çözüm ne kadar korkutucu olsa da ikinci bir yolun varlığı bize büyük kolaylık sağlıyor. Structlar aynı zamanda nesne yönelimli programlamaya geçiş için bir adım olarak da düşünülebilir.

Struct Tanımlaması

```
struct struct_ismi degisken_ismi ;
```

ya da

```

struct struct_ismi {
    veri_türü degisken_ismi;
    veri_türü degisken_ismi;
    veri_türü degisken_ismi;
}

```

```
.  
.   
.   
};
```

şeklinde yapılır.

Örnekte de bahsettiğimiz gibi tüm veri türlerinde değişken tanımlaması yapılabilir. Tanımladığımız bir yapının elemanlarına erişmek için :

`struct_ismi.degisken_ismi`

kullanılır.

Başlarken verdiğimiz örneği kodlayalım.

ÖRNEK 11.1:

```
1. #include <stdio.h>  
2.  
3. struct ogrenci{  
4.     char *isim;  
5.     char *soy_isim;  
6.     int numara;  
7.     int sıra;  
8. } ogrenci;  
9.  
10. int main (void) //main fonksiyonu başlangıcı;  
11. {  
12.  
13.     ogrenci.isim="İsmet";  
14.     ogrenci.soy_isim="Arslan";  
15.     ogrenci.numara=523133;  
16.     ogrenci.sıra=23;  
17.  
18.     printf("Öğrenci ismi :%s\nÖğrenci soy ismi:%s\nÖğrenci numara:%d\   
nÖğrenci sıra:%d",ogrenci.isim,ogrenci.soy_isim,ogrenci.numara,ogrenci.sıra);  
19.  
20.     return 0;  
21.  
22.  
23.} //main fonksiyonu sonu
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/10.bölüm yapılar]
└─ $ ./a.out
Öğrenci ismi :İsmet
Öğrenci soy ismi:Arslan
Öğrenci numara:523133
Öğrenci sıra:23 [v4gus@parrot]--[~/Desktop/c_dersleri/10.bölüm yapılar]
└─ $
```

11.0. İÇ İÇE YAPILAR

Bir struct içerisinde başka bir struct tanımlayabiliriz.Böyle bir tanımlama:

```
struct struct_ismi {  
  
    veri_türü degisken_ismi;  
    veri_türü degisken_ismi;  
    veri_türü degisken_ismi;  
    .  
    .  
    .  
    struct struct_ismi {  
  
        veri_türü degisken_ismi;  
        veri_türü degisken_ismi;  
        veri_türü degisken_ismi;  
        .  
        .  
        .  
    };  
  
};
```

şeklinde yapılır. Erişim için

struct_ismi1.struct_ismi2.degisken;

kullanılır.

Örneğin öğrencilerin ders notlarını tutan ayrı bir struct ile öğrencilerin genel bilgilerini tutan bir structı birleştirelim:

ÖRNEK 11.2:


```

1. //İÇ İÇE STRUCTLAR
2. #include <stdio.h>
3.
4. struct {
5.     char *isim;
6.     char *soy_isim;
7.     int numara;
8.     int sıra;
9.
10. struct {
11.     int matematik;
12.     int ingilizce;
13.     int programlamaya_giris;
14.     }not;
15.
16. } ogrenci;
17.
18. int main (void) //main fonksiyonu başlangıcı;
19. {
20.
21.     ogrenci.isim="İsmet";
22.     ogrenci.soy_isim="Arslan";
23.     ogrenci.numara=523133;
24.     ogrenci.sıra=23;
25.     ogrenci.not.matematik=78;
26.     ogrenci.not.ingilizce=89;
27.     ogrenci.not.programlamaya_giris=98;
28.
29.     printf("Öğrenci ismi:%s\nÖğrenci soy ismi:%s\nÖğrenci numarası:%d\
nÖğrenci sırası:%d\n\töğrenci notları\t\nmatematik:%d\ningilizce:%d\
nprogramlamaya_giris:%d\
n",ogrenci.isim,ogrenci.soy_isim,ogrenci.numara,ogrenci.sıra,ogrenci.not.matematik
,ogrenci.not.ingilizce,ogrenci.not.programlamaya_giris);
30.
31.     return 0;
32.
33. } //main fonksiyonu sonu
34.

```

ÇIKTI:

```

[v4gus@parrot]--[~/Desktop/c_dersleri/10.bölüm yapılar]
$ ./a.out
Öğrenci ismi :İsmet
Öğrenci soy ismi:Arslan
Öğrenci numara:523133
Öğrenci sıra:23
        öğrenci notları
matematik:78
ingilizce:
programlamaya_giris:89
[v4gus@parrot]--[~/Desktop/c_dersleri/10.bölüm yapılar]
$

```

11.0.1. Typedef Kullanımı

Daha önce kodumuzun okunabilirliğinin artması için değişkenlerin ve fonksiyonların isimlerinin işlevlerini belirtmesinin büyük fayda sağlayacağını anlatmıştık. Bu durum structlar için geçerlidir. Structlar isimlendirilirken uzun isimler vermekten kaçınmak için "typedef" kullanılır. İki kullanımın kodlarda nasıl fark oluşturduğuna bakalım.

```
struct ev_adresi {
    int sokak;
    char *ilce;
    char *sehir;
    char *ulke;
};
.
.
.

struct ev_adresi degisken;
degisken.ilce="Kazımkarabekir";
```

typedef kullanımı

```
typedef struct ev_adresi {
    int sokak;
    char *ilce;
    char *sehir;
    char *ulke;
} adres;
.
.
.

adres degisken1;
degisken.ilce="Kazımkarabekir";
```

11.0.2.Structlara Farklı Yönetimle Erişim

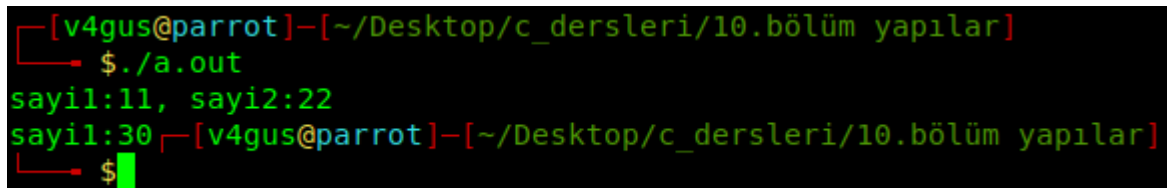
Structlara yalnızca birkaç üyesinin atamasını yapmak için kullanılır.

ÖRNEK 11.3:

1. //YAPILARA FARKLI YOLLA ERİŞMEK
- 2.
3. #include <stdio.h>

```
4.
5. struct sayilar
6. {
7.     int sayi1,sayi2;
8. };
9.
10. int main (void)//main fonksiyonu başlangıcı
11. {
12.
13.     struct sayilar s1 ={.sayi2=22, .sayi1=11};
14.     struct sayilar s2 ={.sayi2=30};
15.
16. printf("sayi1:%d, sayi2:%d\n",s1.sayi1,s1.sayi2);
17. printf("sayi1:%d",s2.sayi2);
18.
19.
20.
21.
22.     return 0;
23. }//main fonksiyonu sonu
```

ÇIKTI:



```
[v4gus@parrot]~[~/Desktop/c_dersleri/10.bölüm yapılar]
└─ $ ./a.out
sayi1:11, sayi2:22
sayi1:30 [v4gus@parrot]~[~/Desktop/c_dersleri/10.bölüm yapılar]
└─ $
```

12.DOSYA İŞLEMLERİ

C programlama dili birçok dil gibi büyük miktarda veri depolamak amacıyla dosyaları kullanır. C dilinde dosya fonksiyonlarını kullanarak :

- bir dosyayı oluşturabilir
- bir dosyayı açabilir
- bir dosyayı okuyabilir
- bir dosyayı kapatabiliriz.

12.0. Dosya Fonksiyonları

C programlama dili bize dosya işlemleri için standart kütüphanede hazır bazı fonksiyonlar sunar.

12.0.0. fopen

Bir dosyayı açmak için dosya pointerı döndüren fopen fonksiyonu kullanılır. Bir dosyayı

açtıktan sonra, derleyicinin dosyadaki giriş ve çıkış işlevlerini gerçekleştirmesine izin vermek için dosya pointerı kullanılır.

Prototipi:

```
FILE *fopen(const char *dosya_ismi, const char *mod);
```

şeklindedir.

Dosyayı açarken okuma, yazma gibi işlemleri mod kısmında belirtiriz.

Modlar :

- r :** dosyayı okuma için açar.
- w :** dosya mevcut değilse yazma için dosyayı oluşturup okuma açar.
- a :** dosya mevcut değilse okuma ve yazma için açar.
- r+ :** dosyayı en başından okuma ve yazma işlemleri için açar.
- w+ :** var olan bir dosyayı okuma ve üzerine yazma için açar.
- a+ :** dosya varsa okuma ve üzerine yazma için açar.

Programımız her ne kadar hatasız çalışsa da fopen fonksiyonunun başarısız olabileceği bir gerçektir. Mesela kullanıcı tarafından belirtilen bir dosyayı açmaya çalıştığımızda dosya yoksa veya yeterli erişim haklarına sahip değilsek fopen NULL pointerı olan 0 değerini döndürür. Bu durumu örneklendirelim ve olmayan bir dosyayı açmaya çalışalım.

ÖRNEK 12.1:

```
1. //VAR OLAYAN DOSYANIN KONTROLÜ
2.
3. #include <stdio.h>
4.
5. int main (void) // main fonksiyonu başlangıcı
6. {
7.     FILE *fp;
8.     fp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya işlemleri/test.c","r");
9.     if(fp==NULL)
10.         printf("erişmek istenilen dosya bulunamadı");
11.
12. return 0;
13.
14. } //main fonksiyonu sonu
15.
```

ÇIKTI:

```
[v4gus@parrot]~[/Desktop/c_dersleri/11.Dosya işlemleri]
$ ./a.out
erişmek istenilen dosya bulunamadı [v4gus@parrot]~[/Desktop/c_dersleri/11.Dosya işlemleri]
$
```

Böyle bir durumda ya programdan çıkış yapmamız gerekir ya da erişmek istenilen dosyayı oluşturarak işlemlere devam etmek. Biz hata mesajı verip çıktık şimdi de istenilen dosyayı oluşturalım. Dosya oluşturmak için yine fopen fonksiyonunu kullanacağız.

```
FILE *fp;
fp=fopen("dosya_ismi","mod") ;
```

şeklinde oluşturulur. Dosya yolu belirtilmemişse dosya programın çalıştığı dizin altında oluşturulacaktır.

Şimdi ulaşmak istediğimiz "/home/v4gus/Desktop/c_dersleri/11.Dosya işlemleri/dosyalar " dizini altında bulunan "dosya1.txt" dosyasına erişemezsek bu isimle yeni bir dosya açan programı yazalım.

ÖRNEK 12.2:

```
1.
2. //ERİŞMEK İSTEDİĞİMİZ DOSYAYA ERİŞİLEMEDĞİ DURUMDA DOSYA
   OLUŞTURAN PROGRAM
3.
4. #include <stdio.h>
5.
6. int main (void)//main fonksiyonu başlangıcı
7. {
8.     FILE *fp;
9.     fp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
   işlemleri/dosyalar/dosya1.txt","r");
10.
11.     if(fp==NULL)
12.     {
13.         printf("dosyaya erişilemedi!\n\nDosya oluşturuluyor...\n");
14.         fp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
   işlemleri/dosyalar/dosya1.txt","w");
15.     }
16.
17.     if(fp==NULL)
18.         printf("dosya oluşturulamadı ! ");
19.
20.     else
21.         printf("dosya başarıyla oluşturuldu");
22.
23. return 0;
24.
25. } //main fonksiyonu sonu
```

ÇIKTI:

Program çalışmadan önce izin:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri/dosyalar]
└─ $ls -l
toplam 0
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri/dosyalar]
└─ $
```

Program çalıştıktan sonra izin:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
└─ $./a.out
dosyaya erişilemedi!

Dosya oluşturuluyor...
dosya başarıyla oluşturuldu [v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
└─ $cd dosyalar/ && ls -l
toplam 0
-rw-r--r-- 1 v4gus v4gus 0 May 11 10:13 dosya1.txt
```

12.0.1. fclose

Bir dosya üzerinde yapılan çalışma bittiğinde fclose fonksiyonu kullanılarak dosya kapatılır. Eğer dosya başarıyla kapatılırsa fclose 0 değerini, hata olursa EOF (dosya sonu) döndürür.

Fonksiyonun prototipi:

```
int fclose (FILE *dosya_ismi);
```

şeklindedir.

Az önce oluşturduğumuz dosyayı okuma modunda açıp , fclose fonksiyonu ile kapatan ve kapatma işleminin başarılı olup olmadığı bilgisini veren programı yazalım.

ÖRNEK 12.3:

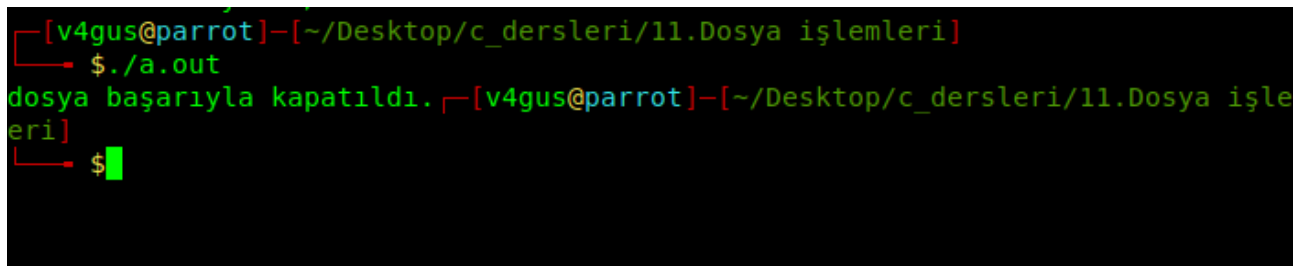
```
1. //FCLOSE FONKSİYONU İLE AÇILAN DOSYAYI KAPATAN PROGRAM
2.
3. #include <stdio.h>
4.
5. int main(void) //main fonksiyonunun başlangıcı
6. {
7.     FILE *fp;
8.     fp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
işlemleri/dosyalar/dosya1.txt","r");
```

```

9.
10.     if(fclose(fp)!=EOF)
11.         printf("dosya başarıyla kapatıldı.");
12.
13.     else
14.         printf("dosya kapatma işlemi başarısız oldu !");
15.
16. return 0;
17.
18. } //main fonksiyonu sonu
19.

```

ÇIKTI:



```

[v4gus@parrot]~$ ./a.out
dosya başarıyla kapatıldı. [v4gus@parrot]~/Desktop/c_dersleri/11.Dosya işlemleri$

```

12.1. Dosyaya Yazma

Bir dosya yazarken yeni satı karakterlerinin '\n' bildirilmesi gerekir. Standart kütüphane bize bir dosyaya yazmak için gerekli fonksiyonları sunar.

fputc(char,dosya_pointer): dosya pointerı ile işaret edilen dosyaya bir karakter yazar.

fputs(str,dosya_pointer): dosya pointerı ile işaret edilen dosyaya bir karakter katarı yazar.

fprintf(dosya_pointer, str , degiskenler): dosya pointerı ile işaret edilen dosyaya bir karakter katarı yazdırır. Katar isteğe bağlı olarak biçim belirteçleri ve değişkenleri içerebilir.

12.2.0.fputc() fonksiyonu

ÖRNEK 12.4:

```

1. //FPUTC FONKSİYONU İLE DOSYAYA YAZMA İŞLEMİ
2.
3. #include <stdio.h>
4. #define BOYUT 50
5.
6. int main (void)//main fonksiyonunun başlangıcı
7. {

```

```

8.      int i;
9.      FILE *dp;
10.     char katar[BOYUT];
11.     char str[]= "Nervus Vagus\n";
12.     dp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
işlemleri/dosyalar/dosya2.txt","w"); // dosyayı oluşturup yazma modunda açıyoruz
13.
14.     for(i=0;str[i] !='\n';i++)
15. {
16.     fputc(str[i],dp); //katar dosyaya yazdırılıyor
17. }
18.
19. fclose(dp); // dosya kapatılıyor
20.
21. return 0;
22.
23.
24. } //main fonksiyonunun sonu
25.

```

Programımız dosya.txt dosyasına, cümlenin başarıyla yazıldığını gösteren bir sonraki '\n' karakterine ulaşana kadar bir tane karakter yazar. Yapılan işlem katarın her karakterini teker teker alıp dosyaya yazmaktır.

ÇIKTI:

```

[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
└─ $ ./a.out
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
└─ $ cd dosyalar/ && cat dosya2.txt
Nervus Vagus [v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri/dosyalar]
└─ $

```

12.2.1.fputs() fonksiyonu

ÖRNEK 12.5:

```

1. //FPUTS FONKSİYONU İLE DOSYAYA YAZMA İŞLEMİ
2.
3. #include <stdio.h>
4.
5.
6. int main (void) //main fonksiyonunun başlangıcı
7. {
8.
9.     FILE *dp;
10.    dp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
işlemleri/dosyalar/dosya3.txt","w"); // dosyayı oluşturup yazma modunda açıyoruz

```



```

11.
12.     fputs("Bu bir deneme yazısıdır, ",dp);//yazma işlemi
13.     fputs("döngü kullanmadık\n",dp);//yazma işlemi
14.     fputs("fputc fonksiyonundan daha kolay yazma işlemi yapıyor\n",dp);//yazma
    işlemi
15.
16. fclose(dp); // dosya kapatılıyor
17.
18. return 0;
19.
20.
21. }//main fonksiyonunun sonu
22.

```

ÇIKTI:

```

[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
└─ $cd dosyalar/ && cat dosya3.txt
Bu bir deneme yazısıdır, döngü kullanmadık
fputc fonksiyonundan daha kolay yazma işlemi yapıyor
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri/dosyalar]
└─ $

```

Programımız dosya3.txt isimli dosyayı yazma modunda oluşturup açıyor ve içerisine fputs kullanarak üç tane katar yazdırıp fclose fonksiyonu ile dosyayı kapatıyor.

12.2.2. fprintf() fonksiyonu

ÖRNEK 12.6:

```

1. //FPRINTF FONKSİYONU İLE DOSYAYA YAZMA İŞLEMİ
2.
3. #include <stdio.h>
4.
5.
6. int main (void)//main fonksiyonunun başlangıcı
7. {
8.
9.     FILE *dp;
10.    dp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
    işlemleri/dosyalar/dosya4.txt","w"); // dosyayı oluşturup yazma modunda açıyoruz
11.
12.    fprintf(dp,"GNU/Linux\n");//yazma işlemi
13.
14. fclose(dp); // dosya kapatılıyor
15.

```

```
16. return 0;
17.
18.
19. } //main fonksiyonunun sonu
20.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
└─ $cd dosyalar/ && cat dosya4.txt
GNU/Linux
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri/dosyalar]
└─ $
```

Programda dosya4.txt dosyasını oluşturup yazma modunda açtık ve fprintf fonksiyonunu kullanarak "GNU/Linux" yazdırıp dosyayı kapattık.

12.2. DOSYADAN VERİ OKUMA

Bir dosyadan veri okumak için üç fonksiyon kullanabiliriz.

fgetc(dosya_pointer): Dosya pointerının gösterdiği dosyadan sonraki karakteri döndürür. Dosya sonuna geldiğinde EOF dönderir.

fgets(buffer,karakter_sayisi,dosya_pointer): dosyadan karakter_sayisi-1 karakterlerini okur ve katarı NULL karakterinin son karakter olarak eklendiği bir arabellekte sakar.

fscanf(dosya_pointer, dönüşüm_belirteci,değişken_adresi) : Verileri ayrıştırmak ve analiz etmek için kullanılır. Dosyadan karakterleri okur ve girdiyi , dönüşüm belirteçleri kullanarak değişken_adresi pointerlarına atar. Scanf ile olduğu gibi, fscanf'in boşluk veya yeni satırla karşılaştığında katar okumayı bırakır.

Şimdi üç fonksiyonu da bir programda kullanalım.

ÖRNEK 12.7:

```
1. //FGETS , FSCANF ve FGETC FONKSİYONLARININ KULLANIMI
2.
3. #include <stdio.h>
4.
5. int main (void)//main fonksiyonu başlangıcı
6. {
7.     FILE *dp;
8.     char buffer[30];
9.     char c;
10.
```

```

11.      dp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
işlemleri/dosyalar/dosya3.txt","r");
12.
13.      printf("-----satır okunuyor-----\n");
14.      fgets(buffer, 50 ,dp);
15.      printf("%s\n",buffer);
16.
17.      printf("-----veri okuma ve ayrıştırma-----\n");
18.      dp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
işlemleri/dosyalar/dosya3.txt","r"); //dosya pointerı yeniden konumlandırılıyor
19.
20.      char katar1[10], katar2[2],katar3[20],katar4[2];
21.      fscanf(dp,"%s %s %s %s",katar1,katar2,katar3,katar4);
22.
23.      printf("1. katar okunuyor|s|\n",katar1);
24.      printf("2. katar okunuyor|s|\n",katar2);
25.      printf("3. katar okunuyor|s|\n",katar3);
26.      printf("4. katar okunuyor|s|\n",katar4);
27.
28.      printf("-----tüm dosyayı oku-----\n");
29.      dp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
işlemleri/dosyalar/dosya3.txt","r"); // pointer yeniden konumlandırılıyor
30.
31.      while((c=getc(dp))!=EOF)
32.          printf("%c",c);
33.
34.      fclose(dp);
35.
36.      return 0;
37.
38. }//main fonksiyonu sonu
39.

```

ÇIKTI:

```

[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
$ ./a.out
-----satır okunuyor-----
Bu bir deneme yazısıdır, döngü kullanmadık

-----veri okuma ve ayrıştırma-----
1. katar okunuyor|r|
2. katar okunuyor|bir|
3. katar okunuyor|zısıdır,|
4. katar okunuyor|yazısıdır,|
-----tüm dosyayı oku-----
Bu bir deneme yazısıdır, döngü kullanmadık
fputc fonksiyonundan daha kolay yazma işlemi yapıyor

```

Daha önce yazdırmış olduğumuz dosya3.txt dosyası açtık ve fgets fonksiyonu ile bir satır okuduk. Ardından dosya pointerının dosyanın başına konumlanması için dosyayı yeniden açtık ve fscanf fonksiyonu ile farklı karakter sayılarındaki katarlar içerisine atayıp yazdırdık. Dosya pointerının

yeniden konumlanması için dosyayı tekrar okuma modunda açtık. Bu defa EOF ile karşılaşana kadar getc fonksiyonu ile dosyadan verileri okuduk ve karakterler halinde yazdırdık. Fclose fonksiyonu ile dosyayı kapattık.

12.3. GETC ve PUTC İle Etkileşimli Dosya Okuma ve Yazma

ÖRNEK 12.8:

```
1. //GETC VE PUTC İLE İNTERAKTİF DOSYA İŞLEMLERİ
2.
3. #include <stdio.h>
4.
5. int main (void)//main fonksiyonu başlangıcı
6. {
7.
8.     FILE *dp;
9.     char k;
10.    printf("Dosyaya veri giriniz\nÇıkış yapmak için tab tuşuna basınız\n");
11.
12. //dosya açılıyor
13.    dp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
işlemleri/dosyalar/dosya5.txt","w");
14.
15. //yazma işlemi
16.    while((k=getchar()) != '\t')
17.        putc(k,dp);
18.
19. //dosya kapatılıyor
20.    fclose(dp);
21.
22.    printf("veri girişi yapıldı\n");
23.
24. //okuma
25.    dp=fopen("/home/v4gus/Desktop/c_dersleri/11.Dosya
işlemleri/dosyalar/dosya5.txt","r");
26.
27.    while((k=getc(dp))!=EOF)
28.        printf("%c",k);
29.
30.    fclose(dp);
31.
32.    return 0;
33.
34. }//main fonksiyonu sonu
35.
```

ÇIKTI:

```
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
./a.out
Dosyaya veri giriniz
Çıkış yapmak için tab tuşuna basınız
bu veriler dosyaya
yazılıp
o
ku
n
a
cak

veri girişi yapıldı
bu veriler dosyaya
yazılıp
o
ku
n
a
cak
[v4gus@parrot]--[~/Desktop/c_dersleri/11.Dosya işlemleri]
$
```

Programımızda dosya5.txt isimli bir dosya oluşturduk ve yazma modunda açtık. Yazma işlemini tab ile sonlandırdıktan sonra fclose fonksiyonu ile kapattık. Sonrasında okuma modunda tekrar açtık ve EOF bulunana kadar while döngüsü ile printf fonksiyonunu kullanarak dosyadaki verileri yazdırdık. Bu işlemten sonra tekrar fclose kullanarak dosyayı kapattık.

KAYNAKÇA

1. İsmet Kocaman ,C programlama dili
2. Gökhan Yalnız , Temel C programlama dili örnekleri
3. Kaan Aslan ,A'dan Z'ye C klavuzu , Pusula Yayıncılık
4. How To C Deitel ,7. baskı ,Palme Yayınları
5. <https://www.cprogramming.com/tutorial/>
6. <https://beginnersbook.com/2014/01/c-tutorial-for-beginners-with-examples/>
7. <https://www.guru99.com/c-programming-tutorial.html>
8. <https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings>
9. <http://man7.org/linux/man-pages/man3/>
10. <https://www.onlinegdb.com/>

