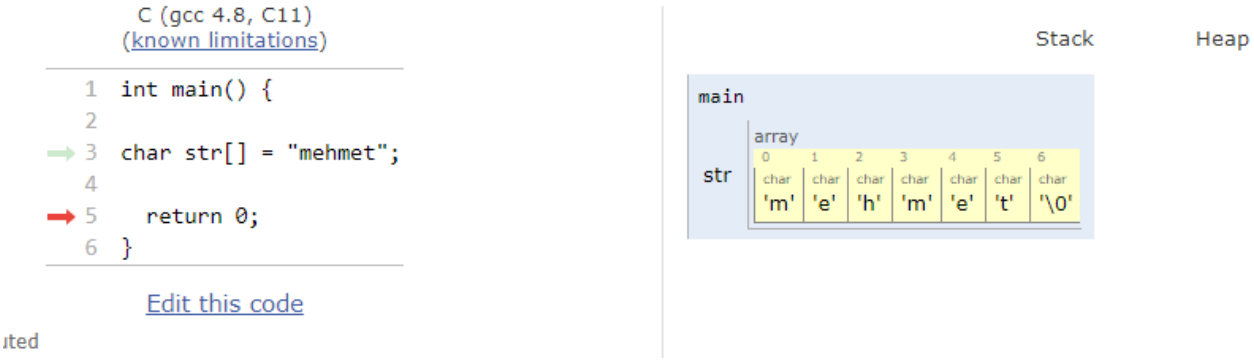
BÖLÜM 7 STRİNGLER

STRİNG NEDİR

Stringler char arraylerdir fakat double veya int arraylerden farklı olarak {} içinde yazılmasına gerek yoktur “ ” çift tırnak içinde tanımlanabilirler.

“mehmet” stringini tanımlayalım



mehmet 6 harfli olmasına rağmen derleyici bize 7 char ayırmış ve "mehmet" yazısının sonuna ‘\0’ charını eklemiş. Bu char başında \ olmasından anlaşılacağı gibi özel bir karakterdir, bir kaçış dizisidir ve “null terminator” olarak isimlendirilir. Null terminator bir stringin bittiğini gösterir ve stringin yazdırılma işlemi sırasında null terminatore gelindiğinde yazdırma işlemi sonlanır. Strinleri yazdırmayı göreceğiz şimdilik bilelim ki yazdığımız stringin sonuna ‘\0’ charı eklenir bu yüzden bir string tanımladığımızda uzunluğu en az yazının uzunluğu + 1 olmalıdır bu yüzden “mehmet” stringinin boyu 6 değil 7 dir

char str[7] = “mehmet”;

STRİNGLER NASIL TANIMLANIR

Temel olarak 2 şekilde string tanımlayabiliriz

1)[ ] NOTASYONU İLE

emre yazan bir string tanımlayalım

char str[] = “emre”;

köşeli parantez içine sayı yazmak istersek emre kelimesinin uzunluğu ve ‘\0’ için toplam 5 char ayırmalıyız

char str[5] = “emre”;

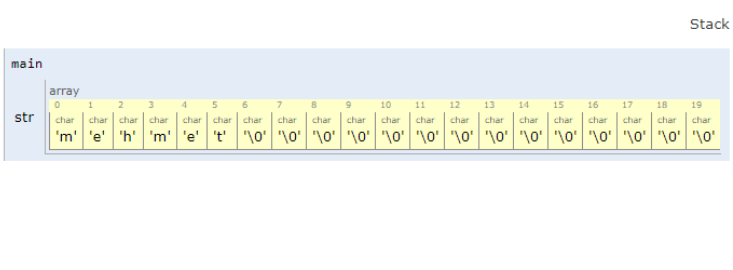
görünmese bile ‘\0’ için yer ayırmalıyız.

Arraylerde yaptığımız gibi bir kısmı dolu bir string tanımlamak istersek

C (gcc 4.8, C11)
(known limitations)

```
1  #define SIZE 20
2
3  int main() {
4
5    char str[SIZE] = "mehmet";
6
7    return 0;
8  }
```

Edit this code



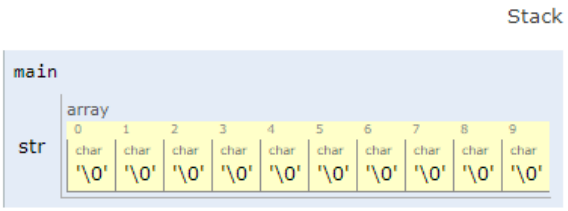
c bizim için verinin anlamsız kısımlarına ‘\0’ yerleştiriyor

Eğer boş bir string tanımlarsak c stringin her indeksine ‘\0’ koyar

C (gcc 4.8, C11)
(known limitations)

```
1  #include <stdio.h>
2
3  int main(){
4
5     char str[10] = "";
6
7
8      return 0;
9  }
```

Edit this code

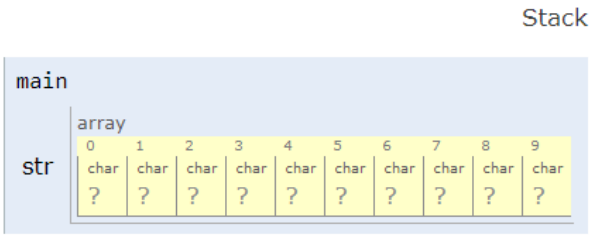


eğer bir string tanımlayıp başlangıç değeri vermezsek stringin elemanı rastgele charlardır.

Char str[10];
rastgele 10 adet char

C (gcc 4.8, C11)
(known limitations)

```
1  int main() {
2
3    char str[10];
4
5    return 0;
6  }
```



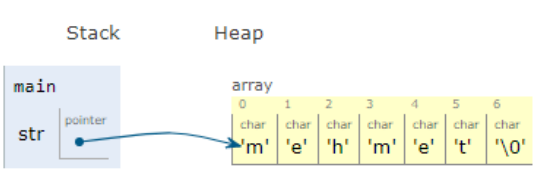
1)* POİNTER NOTASYONU İLE

Bir char * tanımlayarak bir string tanımlayabiliriz.

C (gcc 4.8, C11)
(known limitations)

```
1  #include <stdio.h>
2
3  int main(){
4
5     char *str = "mehmet";
6
7
8      return 0;
9  }
```

Edit this code



char* bir stringi yani stringin ilk elemanını gösteren bir pointerdır.

```
printf("%c",*str);//m
printf("%c",*(str + 1));//e
```

char* ve char[ ] notasyonun farklarına bakalım.

1)char[ ] tipinin herhangi bir indeksi değiştirilebilir ama char* tipi modifiye edilemez

```
char *str1 = "mehmet";
str1[0] = 'h';//OLMAZ

char str2[] = "emre";
str2[0] = 'm';//OLUR
```

2)char[ ] tipiyle atama yapılamaz ama char* ile yapılabilir

```
char *str1 = "mehmet";
str1 = "topdal";//OLUR

char str2[] = "emre";
str2 = "topdal";//OLMAZ
```

eğer char[ ] yapısını atama yapar gibi tamamen değiştirmek istersek arrayin elemanlarını teker teker değiştirmeliyiz yani arrayi kopyalamalıyız. Arrayler

konusunda kopyalamayı görmüştük bu unitede kopyalama işlemini strcpy isimli fonksiyona yaptıracağız

Son olarak stringlerin uzunluğu tutmak için companion variable tanımlamamıza gerek yok çünkü stringin uzunluğunu bulan strlen() isminde bir fonksiyonumuz var ve bu fonksiyonu kullanmak için string.h dosyasını eklememiz gerekiyor

C (gcc 4.8, C11)
(known limitations)

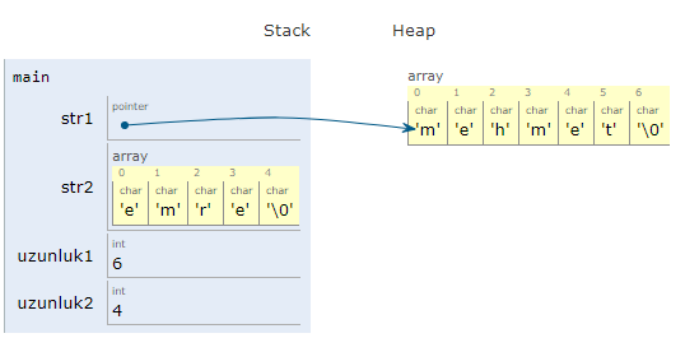
```c
#include <stdio.h>
#include <string.h>

int main(){

   char *str1 = "mehmet";
   char str2[] = "emre";

   int uzunluk1 = strlen(str1);
   int uzunluk2 = strlen(str2);

    return 0;
}
```



gördüğümüz gibi strlen fonksiyonu stringin uzunluğunu '\0' a kadar buluyor

bir stringin elemanlarını alt alta bastıralım

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

int main()
{
    char *str1 = "mehmet";

    for(int i = 0; i < strlen(str1);i++){
        printf("%c\n",str1[i]);
    }

    return 0;
}
```

STRİNGLERLE GİRİŞ ÇIKIŞ

printf ve scanf fonksiyonlarını stringlerle kullanmak istersek %s belirtecini kullanırız

```c
#include <stdio.h>

int main()
{
    char *str1 = "mehmet";

   printf("%s\n",str1);

    return 0;
}
```

char [25] şeklinde kısmen dolu bir stringi bastırmak istersek yazdırma işlemi yaparsak ‘\0’ karakterine gelince yazdırma işlemi sonlanır ve ‘\0’ ve sonrası yazılmaz

```c
#include <stdio.h>
#define SIZE 20

int main()
{

    char str[SIZE] = "mehmet";
    printf("%s\n",str);

    return 0;
}
```

şimdi ise ‘\0’ ı olmayan bir stringi bastıralım

```c
#include <stdio.h>

int main()
{

    char str[3] = "abc";
    printf("%s\n",str);

    return 0;
}
```

Çıktı:

```
PS C:\Users\Mehmet Emre Topdal\Desktop\CC++\EGITIM\bölüm 7 str> .\a
abc
    a
```

string sadece “abc” idiyse diğer a nerden geldi? Demek ki ‘\0’ olmadığı zaman saçma sonuçlar alıyoruz demek ki null terminatoru her zaman korumalıyız.

Şimdi sıra scanf ile string almakta.

char* ile tanımladığımız stringi almak bazı bilgisayarlarda çalışsa bile birçok bilgisayarda çalışmayacaktır çünkü pointer derslerinden bildiğimiz gibi

char *str ifadesi rastgele bir yeri gösteriyor ve bu rastgele değere erişmemiz yasak.

```c
#include <stdio.h>


int main()
{

    char *str;
    printf("str: ");
    scanf("%s",str);
    printf("%s\n",str);

    return 0;
}
```

Eğer başlangıç değeri olan bir string (char *str = “sdhf”;) tanımlayıp scanf kullanırsak segmentation fault alırız. Bu yüzden string almak için char[ ] gösterimini kullanmak en iyi yoldur.

NOT: scanf ile string alırken & koymuyoruz çünkü stringler arraydir. Arrayler pointerdır ve pointerlar adres olduğu için & koymaya gerek yoktur.

Şimdi boş bir string tanımlayıp scanf ile bir değer alalım.

```c
#include <stdio.h>
#define SIZE 20

int main(){

    char str[SIZE];
    printf("str: ");
    scanf("%s",str);
    printf("%s\n",str);

    return 0;
}
```

Çıktı:

```
str: mehmet
mehmet
```

Şimdi ise başlangıç değeri olan bir string tanımlayım ve scanf kullanalım

```c
#include <stdio.h>
#define SIZE 20

int main(){

    char str[SIZE] = "asdfgh";
    printf("str: ");
    scanf("%s",str);
    printf("%s\n",str);

    return 0;
}
```

ÇIKTI:

```
str: abc
abc
```

Bir önceki durum ile aynı sonucu verdi.

Bu örnekleri yaparken kullanıcın 20 den fazla karakter gireceğini düşünmedik şimdi bu durumu inceleyeceğiz.

```c
#include <stdio.h>
#define SIZE 20

int main(){

    char str[SIZE];
    printf("str: ");
    scanf("%s",str);
    printf("%s\n",str);

    return 0;
}
```

Çalıştıralım:

```
PS C:\Users\Mehmet Emre Topdal\Desktop\CC++\EGITIM\bölüm 7 str> .\a
str: vbxzbxcbvbxcvbnxcvbncmvbnxmbnxmvzxzmnxcvb
vbxzbxcbvbxcvbnxcvbncmvbnxmbnxmvzxzmnxcvb
```

Gayet güvenli bir şekilde çalıştı gibi gözükse de ne yazik ki çok büyük bir problem var. 20 karakterden sonra girdiğimiz karakterler ne tuttuğunu bilmediğimiz yerlere yazılıyor. Bu olaya yani ayırdığımız yerin dışına taşmaya “buffer overflow” denir ve çok tehlikeli bir hatadır çünkü veri kaybetmemize ve bazı bilgisayarlarda programımızın çökmesine neden olabilir.

Bu problemi çeşitli şekillerde aşabiliriz şimdi göreceğimiz en basit yöntem: 20 karakterlik alan ayırdık. En sona null terminator yani ‘\0’ koyacağımızı düşünürsek 19 karakter alma hakkımız var bunu belirtmek için scanfin içine “%19s” yazıyoruz ve c 19. karakterden sonra okumayı bırakıyor.

```
char str[SIZE];
printf("str: ");
scanf("%19s",str);

printf("%s\n",str);
```

Çalıştıralım:

```
str: aaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbb
aaaaaaaaaaaaaaaaaa
```

GETS FGETS PUTS

Bir önceki derste scanf ile string almıştık bu derste string almaya yarayan 2, bastırmaya yarayan 1 tane fonksiyon göreceğiz.

```
#include <stdio.h>
#define SIZE 20

int main(){

    char str[SIZE];
    printf("str: ");
    scanf("%s",str);

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

    return 0;
}
```

Scanf varken neden başka başka bir fonksiyon kullanmalıyız çünkü scanf fonksiyonunu boşluk gördüğünde okumayı bitirir aşağıdaki kodu ve çıktıyı inceleyelim.

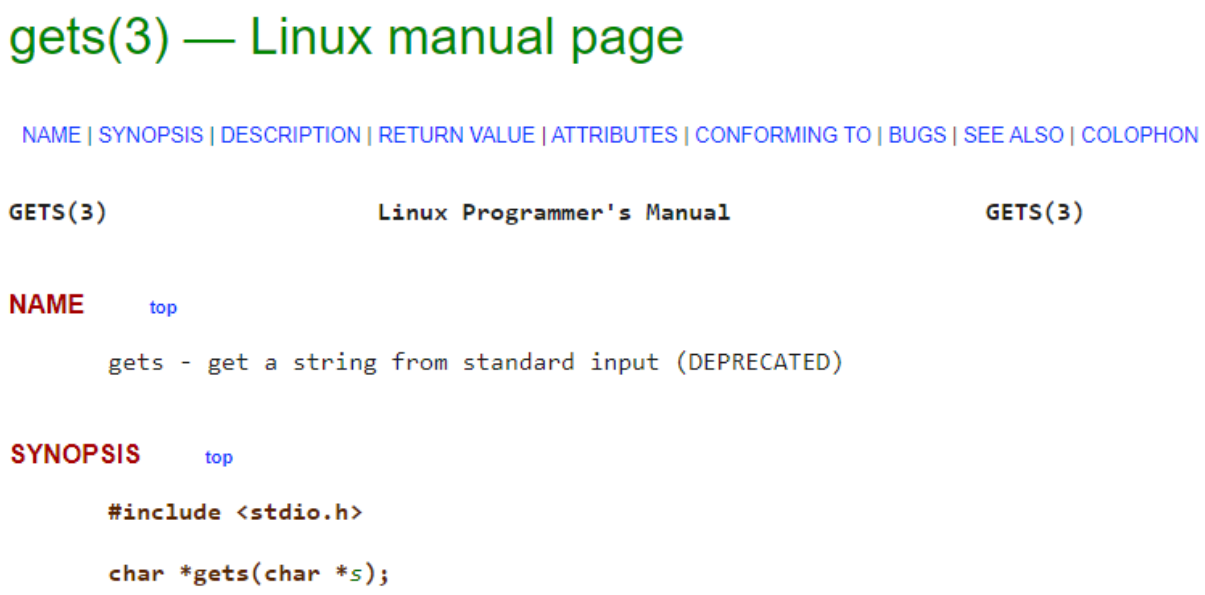
Çıktı:

```
str: mehmet emre
mehmet
```

“mehmet emre” girdiğimizde “ emre” kısmını okumuyor çıktı olarak “mehmet” alıyoruz.

Bu boşluk probleminden kurtulmak için gets() fonksiyonunu kullanacağız fakat başlamadan önce manual page'ten bahsetmek istiyorum.

Manual page mac ve linux sistemlerinde terminal diğer adıyla uç birim ile ulaşılabilen, windowsta ise internetten ulaşabileceğimiz, fonksiyonların prototiplerini ve kullanımlarıyla alakalı ipuçlarını barındıran sayfalardır. gets() fonksiyonunun manuel page sayfasını inceleyelim. Mac veya linux kullananlar terminallerine
man 3 gets veya man 2 gets yazarak bu sayfaya ulaşabilirler



gets(3) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUE | ATTRIBUTES | CONFORMING TO | BUGS | SEE ALSO | COLOPHON

GETS(3) Linux Programmer's Manual GETS(3)

NAME top

gets - get a string from standard input (DEPRECATED)

SYNOPSIS top

```
#include <stdio.h>

char *gets(char *s);
```

name kısmında: standart inputtan (klavye demiştik) string alır diyerek kısaca bir tanımlama yapıyor ve parantez içinde (KALDIRILDI) dese de standartlara ve derleyiciye göre hala kullanılabiliyor.

Synopsis kısmında: kullanmak için stdio.h header file'ını dahil etmemizi, argüan olarak char *(bir string veya char gösteren bir pointer) aldığını söylüyor. Char * döndürdüğünü de söylese de fu fonksiyon için döndürdüğü değer çok d önemli değil.

Gets fonksiyonu ile bir string alıp bastıralım.

```c
#include <stdio.h>
#define SIZE 20

int main(){

    char str[SIZE];
    printf("str: ");
    gets(str);

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

    return 0;
}
```

Gayet güzel görünse de man page'te bir açıklama var

DESCRIPTION top

Never use this function.

gets() reads a line from *stdin* into the buffer pointed to by *s* until either a terminating newline or **EOF**, which it replaces with a null byte ('\0'). No check for buffer overrun is performed (see BUGS below).

yeşil yazı:bu fonksiyonu asla kullanmayın

çok kesin bir uyarı bize "kullanma" diyor çünkü gets fonksiyonu kaç karakter okuyacağını bilemez. 20 karakterlik bir stringe "sknvsdongoıfdgoıdfhıgohfıghdgghdıfhgıodg" gibi uzun bir string girdiğimizde 20 karakteri aşar ve yazmaya devam eder (Biz bu olaya bir önceki derste "buffer error" demiştik.) bu yüzden bu fonksiyonu asla kullanmamalıyız bunun yerine fgets fonksiyonunu kullanacağız.

fgets fonksiyonun man page'ini görelim

NAME top

fgets – get a string from a stream

SYNOPSIS top

```
#include <stdio.h>

char *fgets(char *restrict s, int n, FILE *restrict stream);
```

name kısmında: stream yani bir akıştan veri aldığını söylüyor

synopsis kısmında: üçüncü parametrenin bir “akış” olduğunu söylüyor.

Akış kelimesi hakkında fazla kafa yormayıp şunu düşünelim. Vereceğimiz string fonksiyona nereden veriliyor. Klavyeden yani standart inputtan. Standart input kavramını stdin ile gösteriyoruz.

Birinci parametre bir stringi, ikinci parametre ise kaç karakter okunacağını gösteriyor.

Şimdi fgets kullanarak bir string alalım

a.c > ...

```c
#include <stdio.h>
#define SIZE 20

int main(){

    char str[SIZE];
    printf("str: ");
    fgets(str,SIZE,stdin);

    printf("%s",str);

    return 0;
}
```

input olarak “mehmet emre” stringini girelim. Fgets fonksiyonunun boşluklarla bir problemi yok.

```
PS C:\Users\Mehmet Emre Topdal\Desktop\CC++\EGITIM\bölüm 7 str> .\a
str: mehmet emre
mehmet emre
PS C:\Users\Mehmet Emre Topdal\Desktop\CC++\EGITIM\bölüm 7 str>
```

Farkettiyseniz 10. satırda ‘\n’ olmamasına rağmen alt satıra geçilmiş. Bunu açıklamak için fgets fonksiyonunun çalışmasından bahsedelim.

1)SIZE ifadesinin değeri 20. eğer çok uzun bir string girilirse: fgets 19 karakter okur ve stringin son indeksine ‘\0’ yerleştirir.

2)eğer 19. karakterden önce enter tuşuna basarsak önce ‘\n’ daha sonra da ‘\0’ koyar

3)eğer 19 karakter girip entera basarsak son indekse ‘\n’ değil ‘\0’ yerleştirilir. Bu öncül windows ortamında bundan farklı çalışabilir fakat linux ortamında (üniversitelerimizin çoğu linux kullandığı için bunu belirtmek istiyorum) çalışması bu şekildedir.

bu kadar string almak yeter diyerek ekrana string bastırmamızı sağlayan puts fonksiyonuna geçelim.

Puts fonksiyonu içine aldığı stringi bastırır ve aşağı satıra geçer. İçine isterseniz bir değişken veya çift tırnak içinde bir string bastırabilirsiniz fakat puts fonksiyonunu kullanırken string formatlayamazsınız

```c
#include <stdio.h>

int main(void){

    char *str1 = "string 1";
    char str2[] = "string 2";

    puts(str1);
    puts(str2);
    puts("bu bir string");

    //puts("%d",10);
    //bu geçersizz
}
```

çıktı:

```
string 1
string 2
bu bir string
```

ARRAY OF STRİNG / STRİNG ARRAY

stringlerden oluşan bir array tanımlamak istiyoruz. Stringler de bir array olduğu için string array dediğimiz tip aslında iki boyutlu bir arraydir.

Char str_arr[5][10] -> 5 adet en az 10 chardan oluşan bir string array anlamına gelir. 10 char içine ‘\0’ dahildir.

```c
#include <stdio.h>

int main(){

    char str_arr[5][10] = {"str1","str2","str3","string4"};

}
```

Bu yapıyı c tutorda inceleyelim.

main

str_arr

array

0,0 char 's'	0,1 char 't'	0,2 char 'r'	0,3 char '1'	0,4 char '\0'	0,5 char '\0'	0,6 char '\0'	0,7 char '\0'	0,8 char '\0'	0,9 char '\0'
1,0 char 's'	1,1 char 't'	1,2 char 'r'	1,3 char '2'	1,4 char '\0'	1,5 char '\0'	1,6 char '\0'	1,7 char '\0'	1,8 char '\0'	1,9 char '\0'
2,0 char 's'	2,1 char 't'	2,2 char 'r'	2,3 char '3'	2,4 char '\0'	2,5 char '\0'	2,6 char '\0'	2,7 char '\0'	2,8 char '\0'	2,9 char '\0'
3,0 char 's'	3,1 char 't'	3,2 char 'r'	3,3 char 'i'	3,4 char 'n'	3,5 char 'g'	3,6 char '4'	3,7 char '\0'	3,8 char '\0'	3,9 char '\0'
4,0 char '\0'	4,1 char '\0'	4,2 char '\0'	4,3 char '\0'	4,4 char '\0'	4,5 char '\0'	4,6 char '\0'	4,7 char '\0'	4,8 char '\0'	4,9 char '\0'

Boşluklara ‘\0’ yerleştirilmiş.

Char [][] yapısı char[] lardan array tanımlamak olduğuna göre yazdığımız stringleri modifiye edebiliriz fakat onlarla atama yapamayız.

str_arr[3][2] = ‘a’; ifadesi geçerliyken
str_arr[3] = “baska_str”; ifadesi geçersizdir

Atama yapmak geçersiz olsa da atama yapar gibi yapabiliriz
Örnek olsun diye kullanıcıdanbir string alalım ve kullanıcının seçtiği bir elemanı alınan stringe çevirelim

```c
#include <stdio.h>
#include <string.h>
#define SIZE 10

int main(){

    char str_arr[5][10] = {"str1","str2","str3","string4"};

    char str[SIZE];
    printf("str gir: ");
    fgets(str,SIZE,stdin);

    int indeks;
    printf("kacinci indekse koyalim: ");
    scanf("%d",&indeks);

    int i;
    for(i = 0;i < strlen(str);i++){
        str_arr[indeks][i] = str[i];
    }
    str_arr[indeks][i] = '\0';

    puts(str_arr[indeks]);

    return 0;
}
```

string array tanımlamanın ikinci yoluna geçelim.
Stringlerin char* tipiyle ifade edildiğini biliyoruz o zaman string tanımlamak için char* isim[ ] yapısını kullanabiliriz.

Char *isim[10] demek “ben 10 tane char* tanımlayacağım her biri bir adet stringi gösterecek ” demek
bu gösterimde öncekinden farklı olarak 2d arrayin elemanlarına atama yapabiliriz ama onları modifiye edemeyiz.

```c
char *str_arr[10] = {"str1","str2","str3","string4","string5"};

str_arr[1] = "string2";//gecerli
str_arr[1][2] = 'c';//segmantation fault
```

CTYPE KÜTÜPHANESİ

Bu derste ctype kütüphanesinden 6 fonksiyon göreceğiz ve bu fonksiyonların benzerini tekrar yazacağız. Bu fonksiyonları kullanmak için ctype.h dosyasını dahil etmeliyiz.
Göstereceğimiz fonksiyonlar bir adet char alıyor ve bu charı DEĞİŞTİRMEYİP değer döndürüyor

isdigit

Aldığı char bir rakam belirtiyorsa 1, belirtmiyorsa 0 döndürür

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


int main(){

    printf("%d\n",isdigit('5'));//1 döndürür
    printf("%d\n",isdigit('a'));//0 döndürür

    return 0;
}
```

Bu fonksiyonu tekrar yazalım.

```c
int my_isdigit(char ch){

    if(57 >= ch && ch >= 48){
        return 1;
    }
    else{
        return 0;
    }

}
```

isalpha

Aldığı char bir harf ise sıfırdan farklı bir sayı, değilse 0 döndürür.

```c
printf("%d\n",isalpha('a'));//0 dan farklı bir sayi
printf("%d\n",isalpha('%'));//0
```

Bu fonksiyonu tekrar yazalım.

```c
int my_isalpha(char ch){

    //    buyuk harf                kucuk harf
    if((90 >= ch && ch >= 65) || (122 >= ch && ch >= 97) ){
        return 1;
    }
    else{
        return 0;
    }

}
```

islower

Aldığı char küçük harfse 0 dan farklı bir sayı, değilse (büyük harf veya özel karakter) 0 döndürür

```
printf("%d\n",islower('b'));//0 dan farklı sayi
printf("%d\n",islower('A'));//0
printf("%d\n",islower('%'));//0
```

bu fonksiyonu kendimiz yazalım

```
int my_islower(char ch){

    if((122 >= ch && ch >= 97 ){
        return 1;
    }
    else{
        return 0;
    }

}
```

tolower

Aldığı char büyük harf ise küçük halini döndürür. Bunun dışındaki durumlarda aldığı charı döndürür

```
char ch1 = 'A';
char ch2 = tolower(ch1);

printf("%c\n",ch1);//A
printf("%c\n",ch2);//a
```

Çıktıdan gördüğümüz üzere fonksiyon charı değiştirmiyor

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

void my_tolower(char *ch){

    if(90 >= *ch && *ch >= 65 ){
        *ch += 32;
    }

}

int main(){

    char ch1 = 'A';
    my_tolower(&ch1);

    printf("%c\n",ch1);

    return 0;
}
```

bu fonksiyonu içine aldığı charı değiştirecek şekilde tekrar yazalım. Geçen üniteden hatırladığımız gibi bir değişkeni değişkenin değerini değiştirmek için call by referance yöntemini yani pointerları kullanmalıyız.

Aklınıza “char * ifadesi bir stringi mi yoksa bir charı mı ifade eder” sorusu geldiyse cevap: ikisi de. Şu an my_tolower fonksiyonumuz bir charı gösteren bir pointer alıyor
char *str = “dfosd”; tanımlamasında str isimli pointer stringin ilk charını gösteriyor.
Sonuç olarak char* tipinin ne ifade ettiğini biz kullanıma göre belirliyoruz

isupper

Aldığı char büyük harfse 0 dan farklı bir sayı, değilse (küçük harf veya özel karakter) 0 döndürür

```
printf("%d\n",isupper('A'));//0 dan farklı bir sayi
printf("%d\n",isupper('a'));//a
```

bu fonksiyonu kendimiz yazalım

```
int my_isupper(char ch){

    if(90 >= ch && ch >= 65 ){
        return 1;
    }
    else{
        return 0;
    }

}
```

toupper

Aldığı char küçük harf ise büyük halini döndürür. Küçük harf almazsa aldığı charı aynen döndürür

```
char ch1 = 'a';
char ch2 = toupper(ch1);

printf("%c\n",ch2);//A
```

bu fonksiyonu aldığı charı değiştirecek şekilde tekrar yazalım

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

void my_toupper(char *ch){

    if(122 >= *ch && *ch >= 97){
        *ch -= 32;
    }

}


int main(){

    char ch1 = 'a';
    my_toupper(&ch1);

    printf("%c\n",ch1);//A

    return 0;
}
```

STDLİB.H KÜTÜPHANESİ

atoi

aldığı stringin int halini döndürür.

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


int main(){

    char *str = "12345";
    int sayi = atoi(str);
    printf("%d\n",sayi);//12345

    return 0;
}
```

Atof

aldığı stringin double halini döndürür

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


int main(){

    char *str = "12345.678";
    double sayi = atof(str);
    printf("%f\n",sayi);

    return 0;
}
```

STDİO.H

putchar

içine aldığı charı ekrana bastırır

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


int main(){

    char *str = "mehmet";
    for(int i = 0;i < strlen(str);i++){
        putchar(str[i]);
    }
    putchar('\n');

    return 0;
}
```

getchar

kullanıcıdan aldığı charı döndürür

```c
#include <stdio.h>


int main(){

    char ch;
    ch = getchar();

    putchar(ch);

    return 0;
}
```

input: a
output: a

Eğer yukarıdaki koda bir char yerine string gönderirsek sadece ilk char döndürülür
input: abcd
output: a

yine de getchar kullanarak bir stringin bütün karakterlerini elde edebiliriz. Bir stringte birden fazla char olduğunu düşününce döngü kullanmak işimize yarayabilir.

Aşağıdaki kodda getchar fonksiyonu sırayla alınan stringin elemanları alır. Fazla kullanacağımız bir yöntem olmasa da internette gördüğünüzde bunun ne olduğunu bilmeniz gerekebilir.

Örnek olması açısından bir string alalım ve bastıralım.
Entera basılana kadar yani '\n' gelene kadar char okuyacak ve bastıracağız

```c
#include <stdio.h>

int main(){

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

    putchar('\n');
    return 0;

}
```

sprintf

printf fonksiyonunun ekrana değil de başka bir stringe formatlanmış veya formatlanmamış string yazmayı sağlayan halidir.

Yapısı:
sprintf(yazma işlemi yapılacak string, yazılacak string, varsa format elemanları)

Örnek yapmak açısından kullanıcıdan isim ve yaş alıp ekrana bastıralım

```c
#include <stdio.h>
#define SIZE 25

int main(){

    char isim[SIZE];
    int yas;

    printf("isim girin: ");
    fgets(isim,SIZE,stdin);

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

    char mesaj[50];
    sprintf(mesaj,"isminiz: %syasiniz: %d\n",isim, yas);
    puts(mesaj);

    return 0;
}
```

programın çalışması:
input
isim girin: mehmet emre
yas girin: 19
output:
isminiz mehmet emre
yasiniz 19

sscanfe bakarsanız “isminiz %s” ten sonra \n yok fakat aşağı geçilmiş bunun sebebi fgets fonksiyonunun \n de bir char olarak kabul edip isim stringine koyması. Bunu önlemek için scanf kullanabilirsiniz. Boşluklarda durmamayı ilerideki bölümlerde öğreneceğiz.

Sprintfin bir diğer kullanımı da bir integer veya double ı stringe çevirmek yani sayıyı stringe koymak (yazmak) tır.

```c
char int_str[SIZE];
int tamsayi = 100;
sprintf(int_str,"%d",tamsayi);
//int_str = "100"

char double_str[SIZE];
int ondalik_sayi = 3.14;
sprintf(double_str,"%f",ondalik_sayi);
//double_str = "3.14"
```

Peki başlangıç değeri olan bir string üzerinde sprintf kullanırsak ne olur.

```c
#include <stdio.h>

int main(){

  char str[25] = "aaaaaaaaaaaaaaaa";
  sprintf(str,"bbbb");

  return 0;
}
```

(formatlamaya gerek olmadığı için sprintfin 3. kısmını yazmadım)

main

str (array)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char	char
'b'	'b'	'b'	'b'	'\0'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

var olan karakterlerin üzerine yazıp sonuna \0 koyuyor ve stringi bitiriyor. 5. indeks ve sonrası \0 dan dolayı ekrana bastırılamaz hale geliyor

sscanf

sscanf fonksiyonu bir stringten veri alıp değişkenlere koymamıza yarar.

Yapısı:
sscanf(veri alınacak string, verileri alacak string , değişkenler)

```c
int sayi1;
double sayi2;
char str1[20];
char str2[20];

char *string = "10 3.14 mehmet emre";
scanf(string,"%d %lf %s %s",&sayi1, &sayi2, str1, str2);
```

STRİNG.H

strcpy

strcpy fonksiyonu bir stringi başa bir stringe kopyalar. Önemli bir fonksiyondur çünkü char[ ] gösterimiyle tanımladığımız bir stringe atama yapamayız ama bu tip bir array modifiye edilebileceğinizden dolayı stringin elemanlarını teker teker değiştirerek atama yapar gibi yapabiliriz.

Kullanım:
strcpy(hedef, kaynak);

Strcpy fonksiyonu bu işlemi bizim için yapar ve kaynak stringini \0 İLE BERABER hedef stringe kopyalar.

```c
#include <stdio.h>
#include <string.h>
#define SIZE 25

int main(){

    char kaynak[SIZE];
    char hedef[SIZE];

    printf("isminizi yazin: ");
    scanf("%s",kaynak);
    //mehmet girildiğini varsayalım

    strcpy(hedef,kaynak);
    //hedef = "mehmet\0"

    puts(hedef);//mehmet

    return 0;
}
```

Peki başlangıç değeri olan bir stringe kopyalama yaparsak ne olur mesela “mehmet emre” stringine “topdal” stringini kopyalamak istiyoruz.

Strcpy kullannılmadan önce

C (gcc 4.8, C11)
(known limitations)

```c
#include <string.h>

int main() {

  char str[] = "mehmet emre";

  strcpy(str,"topdal");

  return 0;
}
```

Stack

main

str

array

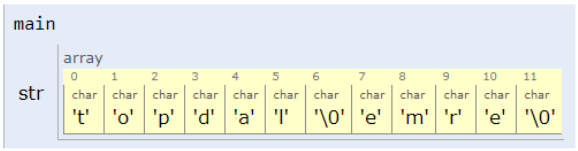
0	1	2	3	4	5	6	7	8	9	10	11
char	char	char	char	char	char	char	char	char	char	char	char
'm'	'e'	'h'	'm'	'e'	't'	' '	'e'	'm'	'r'	'e'	'\0'

strcpy kullanıldıktan sonra

C (gcc 4.8, C11)
(known limitations)

```
1  #include <string.h>
2
3  int main() {
4
5    char str[] = "mehmet emre";
6
7    strcpy(str,"topdal");
8
9    return 0;
10 }
```

Stack



“mehmet emre\0”, “topdal\0emre\0” a dönüşmüş fakat\0 ile stringin bittiğini düşünürsek “mehmet emre” nin “topdal” a dönüştüğünü söyleyebiliriz. İşte atama yapmak da buna benziyor.

Önemli bir nokta var: strcpy fonksiyonu kaynak stringi hedef stringe taşma kontrolu yapmadan kopyalar. Yani

#define SIZE 10

char *kaynak = “aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”;
char hedef[SIZE];

strcpy(hedef,kaynak);

kodunda bütün a harflerini 5 tane kutucuğa ve sonrasına yazmaya çalışır ki biz buna tampon taşma hatası, buffer overflow error demiştik.

Böyle bir durumda kalmamak için bizim belirdeğimiz sayıda kopyalama yapan strncpy fonksiyonunu kullanabiliriz.

strncpy

Kullanım:
strncpy(hedef, kaynak, kopyalanacak karakter sayısı)

karakter sayısını biz belirliyorsak muhtemelen butun stringi kopyalamıyoruz bu yüzden gerektiği zaman \0 koymak bizim görevimiz.

Başlangıç değeri olmayan bir stringe “aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa” stringini taşma olmayacak şekilde kopyalayalım. Bunun için hedef stringin boyutundan 1 eksik sayıda karakter kopyalayalım ki \0 için yer kalsın. Daha sonra da son indekse \0 koyalım

```c
#include <stdio.h>
#include <string.h>
#define SIZE 10

int main(){
    char *kaynak = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
    char hedef[SIZE];

    //kopyalama ve \0 için yer ayırma
    strncpy(hedef,kaynak,SIZE - 1);

    //son indekse \0 koyma
    hedef[SIZE - 1] = '\0';

    puts(hedef);
    return 0;
}
```

son indekse \0 koymayı unutacağınızı düşünüyorsanız hedef stringi char hedef[SIZE] = “ ” şeklinde tanımlayabilirsiniz bu durumda stringin tüm elemanları \0 olacağı için son karaktere \0 koymaya gerek yoktur.

Örnek yapmak için kullanıcıdan alınan sayıda karakteri bir stringe kopyalayalım

```c
#include <stdio.h>
#include <string.h>
#define SIZE 10

int main(){
    char *kaynak = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
    char hedef[SIZE];

    int adet;
    printf("kac karakter kopyalayalim: ");
    scanf("%d",&adet);
    //5 girildiğini varsayalım

    strncpy(hedef,kaynak,adet);
    //5 tane karakteri kopyaladık ve \0 koyacağız

    hedef[adet] = '\0';
    //6. elemana yani 5. indekse \0 koyduk

    puts(hedef);

    return 0;
}
```

Başka bir örnek:
hedef: “AAAAAAAAAAAAAAAAAAAAA” strinine
kaynak: “aaaaa” stringini kopyalayarak hedef stringi “aaaaaAAAAAAAAAAAAAAAA” haline getirmek istiyoruz.

Eğer strcpy kullansaydık. Kaynak stringi \0 ile kopyalanacağından dolayı hedef stringi: “aaaaa\0AAAAAAAAAAAAAAAA” olacaktı. \0 ile string bittiği için büyük A harflerini kaybetmiş olacaktık. Strncpy kullanarak sadece küçük a harflerini kopyalayacağız \0 koymayacağız.

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

int main(){
    char *kaynak = "aaaaa";
    char hedef[] = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";

    strncpy(hedef,kaynak,strlen(kaynak));

    puts(hedef);

    return 0;
}
```

Çıktı:
aaaaaAAAAAAAAAAAAAAAAAAAAAAA

strcat

strcat fonksiyonu aldığı iki stringi birleştirir. İlk stringin \0 karakterini kaldırır ve diğer stringi ekler

“mehmet\0” ve “emre\0” birleştirildiğinde “mehmet\0” stringi “mehmetemre\0” a dönüşür.

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


int main(){

    char str1[20] = "mehmet";
    char *str2 = "emre";

    strcat(str1,str2);

    puts(str1);

    return 0;
}
```

Ne yazık ki strcat fonksiyonu strcpy gibi taşma olup olmadığına bakmaz. Taşmayı engellemek ve belirli sayıda karakter eklemek için strncat fonksiyonunu kullanırız.

Strncat

kullanımı:
strcat(hedef, kaynak, karakter sayısı)

Diyelim ki char hedef[20] = “AAAAA” stringine
char kaynak[ ] = “abcdefg” stringindeki ilk 3 karakteri eklemek istiyoruz.

strncat(hedef, kaynak, 3) koduyla bu eklemeyi yapabiliriz. Şu an hedef stringi “AAAAAabc” durumunda. “AAAAA” şeklinde başlangıç değeri verdiğimizden dolayı harflerden sonra \0 var yine de güvende olmak için stringin sonuna \0 ekleyelim.

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


int main(){

    char hedef[20] = "AAAAA";
    int m = strlen(hedef);
    char kaynak[] = "abcdefg";

    //birleştirme
    int n = 3;
    strncat(hedef,kaynak,n);

    //en son indekse \0 koyma
    //m + n yeni boyut
    //m + n - 1 son indeks
    hedef[m + n - 1];

    puts(hedef);

    return 0;
}
```

18. satırda son indekse ulaşmak için hedef[strlen(hedef) – 1] demedim çünkü strlen(hedef) ifadesi \0 a gelinceye kadar çalışır biz bu örnekte hedef stringinin sonunda \0 olmadığını varsayarak çalıştık.

Hedef stringinin boyutubu bulmak için ilk boy ile eklenen karakter sayısını topladık ve 1 çıkararak son indekse ulaştık.

Son olarak taşma oluncaya kadar iki stringi birleştirelim.
Hedef stringimiz en fazla 20 elemanlı
char hedef[SIZE] = “AAAAAAAAAA”;

kaynak stringimiz ise
char *kaynak = “aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”;

olsun.

Hedef stringinde SIZE – strlen(hedef) kadar boşluk var (20 max kapasite, 10 tane de dolu indeks var olarak düşünün).

Hedef stringin en sonuna \0 koymak istediğimiz için SIZE – strlen(hedef) – 1 kadar karakter kopyalama hakkımız var.

```c
#include <stdio.h>
#include <string.h>
#define SIZE 20

int main(){

    char hedef[SIZE] = "AAAAAAAAAA";
    char *kaynak = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";

    //birleştirme
    strncat(hedef,kaynak,SIZE - strlen(hedef) - 1);

    //en son indekse \0 koyma
    hedef[SIZE - 1];

    puts(hedef);

    return 0;
}
```

Strtok

strtok fonksiyonu bir stringi parçalara(tokenlere) ayırır. Stringi modifiye eder bu yüzden bu fonksiyonu kullanmadan önce orijinal stringi saklamalıyız.

Kullanım:
strtok(parçalanacak_string, sınırlayıcı);
sınırlayıcı: ingilizce delimiter.

Fonksiyon sınırlayıcı stringe kadar olan stringi döndürür.

```c
#include <stdio.h>
#include <string.h>
#define SIZE 20

int main(){

    char str[SIZE] = "AAA BBB CCC DDD EEE";

    char *token = strtok(str," ");
    //sınırlayıcı: boşluk
    //token = "AAA"

    printf("token: |%s|\n",token);

    return 0;
}
```

Br tane alıp bırakmak olmaz. Geri kalan tokenleri de almalıyız.

```c
#include <stdio.h>
#include <string.h>
#define SIZE 20

int main(){

    char str[SIZE] = "AAA BBB CCC DDD EEE";

    char *token = strtok(str," ");

    char *token2 = strtok(NULL," ");
    //token2 = "BBB"

    printf("token: |%s|\n",token2);

    return 0;
}
```

Aynı stringten bir token daha almak istersek fonksiyonun ilk parametresine NULL veriyoruz.

Şimdi ise bir stringin kelimelerini alt alta yazdıralım. Birden fazla token olduğu için döngü kullanmak mantıklı olacaktır.

Döngünün sonlanması için bilmeliyiz: Alınacak token kalmadığında strtok fonksiyonu NULL döndürür. Token, NULL olduğunda döngüyü sonlandıracağız.

```c
#include <stdio.h>
#include <string.h>
#define SIZE 20

int main(){

    char str[SIZE] = "AAA BBB CCC DDD EEE";

    char *token = strtok(str," ");

    while(token != NULL){
        printf("|%s|\n",token);
        token = strtok(NULL," ");
    }

    return 0;
}
```

```
PS D:\EGITIM\bölüm7> gcc a.c -o a
PS D:\EGITIM\bölüm7> .\a
|AAA|
|BBB|
|CCC|
|DDD|
|EEE|
```

İkinci veya üçüncü token almak istediğimizde parçaladığımız stringi yazmıyorsak aşağıdaki durumda çıktı ne olur.

```c
#include <stdio.h>
#include <string.h>
#define SIZE 20

int main(){

    char str1[SIZE] = "AAA BBB CCC DDD EEE";
    char str2[SIZE] = "aaa bbb ccc ddd eee";

    char *ptr1 = strtok(str1," ");
    char *ptr2 = strtok(str2," ");

    char *ptr3 = strtok(NULL," ");
    puts(ptr3);


    return 0;
}
```

Çıktı:bbb

yani en son kullanılan strtoktan devam ediyor.

Strcmp

strcmp fonksiyonu iki stringi karşılaştırır.

Kullanımı:
strcmp(str1, str2)

Eğer iki string aynıysa 0
birinci string alfabede önce geliyorsa negatif bir sayı
ikinci string alfabede önce geliyorsa pozitif bir sayı döndürür.

Daha detaylı anlatmak gerekirse: strcmp fonksiyonları aldığı stringlerin karaktarlerini sırayla karşılaştırır; birinci stringte bakılan karakterin ASCII değeri ikinci stringtekinden büyükse negatif sayı, ikinci stringteki karakterin ASCII değeri büyükse pozitif sayı döndürür. işlemler ASCII tablosu üzerinden gerçekleştiği için alfabetik sıra karşılaştırması yapmak istersek bakılan karakterlerin ikisi birden küçük veya büyük olmalıdır.

Örnek üzerinde inceleyelim.

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

int main(){

    char *str1 = "aaaa";
    char *str2 = "aaaa";
    char *str3 = "bbbb";

    printf("%d\n",strcmp(str1, str2));
    printf("%d\n",strcmp(str2, str3));
    printf("%d\n",strcmp(str3, str2));

    return 0;
}
```

ÇIKTI:

```
PS D:\EGITIM\bölüm7> .\a
0
-1
1
```

strcmp fonksiyonuna dair en önemli şey stringler aynı olduğunda 0 döndürmesidir çünkü if koşulu içinde == kullanmak string karşılaştımak için uygun değildir.

Sıradaki örnekte program bir isim alsın ve eğer stringte "emre" yazıyorsa "merhaba adaşım", başka bir şey yazıyorsa "merhaba (isim)" olarak çıktı versin.

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

int main(){

    char str[20];
    printf("isminiz nedir: ");
    scanf("%s",str);

    if(str == "emre"){
        printf("merhaba adasim\n");
    }
    else{
        printf("merhaba %s\n",str);
    }

    return 0;
}
```

programı çalıştıralım ve "emre" girelim

```
PS D:\EGITIM\bölüm7> .\a
isminiz nedir: emre
merhaba emre
```

emre girdiğim halde adaşım demedi o zaman hatalı karşılaştırma yapmış. bu hatayı düzeltmek için strcmp fonksiyonunu kullancağız. iki string aynıysa 0 döneceğini biliyoruz o zaman

strcmp(str, "emre") ifadesinin 0 olduğu durumda if bloğunu çalıştırmalıyız.

tam program:

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

int main(){

    char str[20];
    printf("isminiz nedir: ");
    scanf("%s",str);

    if(strcmp(str, "emre") == 0){
        printf("merhaba adasim\n");
    }
    else{
        printf("merhaba %s\n",str);
    }

    return 0;
}
```

GİRİŞ ÇIKIŞ İŞLEMLERİ

Bu konu başlığında giriş çıkışlar(input / output) ile ilgili birlkaç bilgi vereceğiz.

1)giriş ve çıkış işlemleri her zaman string ile yapılır.

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

şeklindeki bir kodda kullanıcı 12 değil "12" girer ve bu string %d ifadesi ile bir integera çevrilir.

aynı şekilde

printf("%d\n", 15);

kodunda ekrana yazılan şey integer olan 15 değil string "15" tir

2)ekrana kaçış dizilerini ve çift tırnak işaretini yazdırma

diyelim ki ekrana \n yazdırmak istiyoruz eğer bir printf içine bu ifadeyi koyarsak bunun bir kaçış dizisi olduğunu düşünecek ve aşağı satıra geçecek. \n yazdırmadan önce şunu düşünelim. aşağı satıra geçen \n ifadesindeki n harfi normal bir n harfi değil başına \ aldığı için artık farklı bir anlam taşıyor o zaman \n ifadesinin önüne tekrar \ koyarsak bu "bu ifade alt satıra geçen \n değil" anlamına gelir.

o zaman ekrana "\n" yazdırmak için

printf("\\n");

yazmak gerekiyor.

Şimdi ise ekrana " yani çift tırnak yazdırmak istiyoruz. printf içine direk " koyamayız. diyelim ki

Ali bana "merhaba" dedi.

çıktısını almak istiyoruz

```
printf("Ali bana "merhaba" dedi");
```

fark ettiyseniz merhaba kelimesi turuncu değil çünkü c ilk tırnaktan itibaren yeni bir " gördüğü zaman stringin bittiğini düşünüyor.

"Bu tırnak stringi bitirmiyor, onu bilerek koydum. Bu öyle bir tırnak değil" demek için \ işaretinden faydalanıyoruz ve \" yazarak ekrana tırnak bastırabiliyoruz.

```
printf("Ali bana \"merhaba\" dedi");
```

Bunlardan farklı olarak ekrana % işaretini bastırmak için yan yana iki tane % işareti koyuyoruz.

```
printf("yuzde isareti boyle basilir: %%");
```

3)printf ve scanf fonksiyonu ne döndürür.

printf fonksiyonu ekrana yazılan karakter sayısını döndürür.

aşağıdaki kodda ilk önce mehmet yazılacak, alt satırda ise 7 sonucunu alacağız çünkü \n ifadesi özel tanımlanmış bir char.

```
int i = printf("mehmet\n");

printf("%d\n",i);
```

Çıktı:
mehmet
7

aşağıdaki kodda ekrana 123 basılıp aşağı satıra geçeçek ve 4 basılacak.

```
int i = printf("%d\n",123);

printf("%d\n",i);
```

Çıktı:
123
4

scanf fonksiyonu ise kullanıcıdan aldığı değişken sayısını döndürür.

```
int a, b;
double c;

int i = scanf("%d %d %lf", &a, &b, &c);

printf("%d\n",i);
```

input: 2 5 7.8
çıktı: 3

4)hassaslık (precision)
diyelim ki bir ondalık sayıyı virgülden sonraki iki basamağı ile yazdırmak istiyoruz. bunun için

%.(bas_sayısı)f kalıbını kullanabiliriz. pi sayısını ilk iki basamağı ile yazdırmak istiyoruz. (pi = 3.14159)

```
double pi = 3.14159;
printf("%.2f",pi);
```

bu kodun çıktıssı: 3.14

şimdi de 4,39 sayısını virgülden sonraki ilk basamağı ile yazdıralım

```
double i = 4,39;
printf("%.1f",i);
```

çıktı: 4,4

4.3 çıktısı daha anlamlı olabilirdi. burda olan şey ise şu:

.1 yazdık bundan bir sonraki basamağa yani 2. basamağa göre sayımız yuvarlanacak ve virgülden sonra 1 basamak yazılacak.

sayının tam kısmını yazdırmak içinse

```
double i = 4.9
printf("%.f",i);
```

yazılabilir fakat yuvarlama olacağı için eğer sayının tam kısmını istersek typer casting yapmak

```
printf("%.f", (int) i) ;
```

yazmak daha uygun olacaktır.

5) Bir sayı veya string bastırırken ekranda alan ayırabiliriz.

mesela "mehmet" stringi için ekranda 10 karakter ayıralım

```
printf("|%10s|", "mehmet");
```

çıktı:
| mehmet|

yazı 10 karakterin sağına yaslanmış, sola yaslamak istersek negatif sayı kullanacağız. şimdi 10 karakter ayırıp yazımızı sola yaslayalım

```
 printf("|%-10s|", "mehmet");
```

çıktı:
|mehmet |

10 karakter ayırdık diyelim peki yazdırılacak yazı 10 karakter veya daha fazla olursa ne olur

```
printf("|%10s|", "aaaaaaaaaa");    //10 karakter,
printf("|%10s|", "aaaaaaaaaaaaaaa");      //15 karater
```

çıktı:
|aaaaaaaaaa|
|aaaaaaaaaaaaaaa|

yani boşluk çıkmaz yaazılacak şey aynen yazılır.

Bu olayı bir de sayılarda görelim

```
printf("|%10f|", -3.141);
```

çıktı:
| -3.141000|

buradan çıkaracağımız şey şu: ondalık sayılardaki nokta ve negatif sayılardaki - için de ekrana karakter ayrılıyor.

varsayılan olarak ekrana virgülden sonra 6 sayı yazılıyor eğer sadece -3.141 yazdırmak istersek

printf("|%10.3f|", -3.141); yazmak gerekir
çıktı:
| -3.141|
6)diyelim ki aşağıdaki kod ile kullanıcıdan tarih almak istiyoruz.

```
int gun, ay, yil;
printf("tarih girin: ");
scanf("%d %d %d",&gun, &ay, &yil);

printf("%d %d %d",gun, ay, yıl);
```

kullanıcı tarihi 1/1/2001 şeklinde girerse

Çıktı:
1 3297280 0

saçma bir çıktı alıyoruz çünkü c, / işaretlerinin sayıya ait olduğunu düşünüyor.

Bunu engellemek içinse / geleceğini bildiğimiz yerlere / işaretini koyuyoruz.

scanf("%d/%d/%d",&gun, &ay, &yil);.

tekrar kodu çalıştırdığımızda çalıştığını göreceğiz fakat kullanıcı tarihi 1.1.2001 veya 1 1 2001 şeklinde de girebilirdi. bu üç durumu kapsamak için "char görürsen okuma boş ver, sonraki şeye geç" anlamına gelen "%*c" işaretini kullanıyoruz. % ve c harfi arasındaki * işareti "okuma!" demek gibi düşünebiliriz.

scanf("%d%*c%d%*c%d",&gun, &ay, &yil);

BU durumda sayıların arasına konulan char ne olursa olsun okunmkayacak ve değişkenlere koyulmayacaktır.

7)Son olarak scanf fonksiyonunun boşluklarda durmasını engelleyeceğiz.

diyelim ki 80 elemanlı bir stringe "ali topu at" cümlesini koymak istiyoruz.
bu cümle 79 karakterden az olduğu için eğer fgets fonksiyonu kullanırsak cümlenin sonuna '\n' eklenecek ama scanf kullanarak bunu engelleyebiliriz.
Bu sefer de biliyoruz ki scanf fonksiyonu boşluklarda duruyor bunu engellemek için

scanf("%[^\n]s",str);

satırını kullanacağız.

[ ] içindeki kısma "scan set" adı verilir.

^\n ifadesi "\n gelinceye kadar okumaya devam et." demektir
eğer ^ işaretini kullanmazsak yani [\n] yazarsak bu da "\n leri oku. \n dışında bir şey gelirse okumayı bitir" anlamına gelir. taşma hatası olmaması için maksimum 79 karakter okuyup son indeksi \0 için ayırmalıyız ki bunu da % işaretinden sonra 79 yazarak yapacağız.

 scanf("%79[^\n]s",str);