Metin dizileri - Dizgiler

C dilinde **dizgiler (strings)**, metin veya karakter dizilerini temsil etmek için kullanılan veri yapılarıdır. Bir dizgi, karakterlerden oluşan bir dizinin ardışık olarak bellekte saklanmasıyla oluşturulur. Dizginin sonunda yer alan özel bir **'\0' (null karakter)**, dizginin bittiğini belirtir ve diğer karakterlerden ayrılmasını sağlar. Bu yapı, dizgilerin boyutlarını dinamik olarak belirlemeyi ve işlemleri düzenli bir şekilde yapmayı mümkün kılar. Dizgiler, özellikle kullanıcı girdisi alma, metin tabanlı verilerle işlem yapma ve metinlerin manipüle edilmesi gibi işlemlerde temel bir rol oynar.

Dizgiler, programlarda metinlerin düzenlenmesi, saklanması ve işlenmesi için önemli bir araçtır. Örneğin, kullanıcıdan alınan bir isim veya mesaj, bir dosyadan okunan metin verileri ya da bir uygulama tarafından üretilen bir hata mesajı dizgiler aracılığıyla işlenir. Dizgiler, esneklik sağlayan ve programların dinamik bir şekilde metin tabanlı veriyle çalışmasını kolaylaştıran yapılar sunar. Bu nedenle, dizgiler sadece basit metinlerle çalışmak için değil, aynı zamanda veri işleme, iletişim ve kullanıcı arayüzü oluşturma gibi daha karmaşık uygulamalarda da yaygın olarak kullanılır.

Dizgilerin Özellikleri

1. **Sonlandırıcı karakter ('\0'):** C dilinde dizgiler, null karakterle ('\0') sonlandırılır. Bu karakter, dizginin nerede bittiğini belirtir ve bellekte diğer karakterlerle karışmasını önler. Örneğin, "Selam" dizgisi bellekte 'S', 'e', 'l', 'a', 'm', '\0' olarak saklanır.
2. **Karakter dizisi yapısı:** Dizgiler, aslında char türünde bir dizidir. Örneğin, char str[6] = "Selam"; ifadesi bir dizgi tanımlar. Dizginin boyutu, karakterlerin sayısına artı sonlandırıcı karaktere göre belirlenir.
3. **Statik veya dinamik tanımlama:** Dizgiler statik olarak sabit uzunlukta tanımlanabilir veya bellek yönetimi kullanılarak dinamik şekilde oluşturulabilir.
4. **Standart kütüphane desteği:** C dilinde dizgilerle çalışmak için **string.h** kütüphanesi kullanılabilir. Bu kütüphane, dizgiler üzerinde işlem yapmak için çeşitli fonksiyonlar sağlar (örneğin, strlen(), strcpy(), strcat()).

Dizgilerin tanımlanması

Metin dizileri, dizgiler iki tırnak simgeleri (" ") arasında tanımlanırlar. Bu tanımlama şekli metin dizilerinin 1'den fazla karakterden oluştuğunu belirtmektedir. Daha önceki derslerde anlatıldığı üzere C dilinde tek harf metin değişkenleri tek tırnakla (' ') ayrı bir şekilde tanımlanmaktadır. Bu ise iki farklı şekilde karakter değişken tanımlanmasına neden olmaktadır. Bu açıdan kullanım amacına uygun olarak metin değişkenleri tanımlamak gereklidir. Metin dizileri 1'den fazla karakterden oluştukları için sayısal dizilerde olduğu gibi bir dizi formatından tanımlanmaları lazımdır.

Bir metin dizisinin tanımlama formatı/düzeni şu şekildedir:

```
char  dizgi ismi  [  uzunluk  ];
```

Dizgiler tanımlanırken bellek kullanımına dikkat edilmesi, taşmaların ve bellek hatalarının önlenmesi açısından önemlidir. Gereğinden fazla veya yetersiz bellek tahsisi, programın işleyişini olumsuz etkileyebilir. Kullanıcıdan alınacak girdilerin saklanması ve işlenmesi için ise dizgiler önemli bir araçtır. Tanımlama sırasında dizgi uzunluğunu ve formatını doğru şekilde belirlemek, kullanıcı girdilerinin sorunsuz bir şekilde işlemek için gereklidir.

1. **Sabit Boyutlu Tanımlama:** Sabit boyutlu diziler, belirli bir karakter sayısına uygun olacak şekilde tanımlanır. char dizi[20]; // 20 karakterlik bir yer ayrılır.
2. **Başlangıç Değeri ile Tanımlama:** Tanımlama sırasında bir metin atanır ve '\0' karakteri otomatik olarak eklenir. char dizi[] = "Merhaba"; // 8 karakterlik bir dizi (7 harf + '\0') oluşur.
3. **Karakter Dizisi ile Tanımlama:** Her karakter ayrı ayrı belirtilerek dizi tanımlanabilir. char dizi[5] = {'H', 'e', 'l', 'l', 'o'};
4. **Dinamik Bellek Tahsisi ile Tanımlama:** Daha esnek bir yapı sağlamak için malloc() veya calloc() kullanılarak bellekte yer ayrılır. char *dizi = (char *)malloc(50 * sizeof(char)); // 50 karakterlik bir alan oluşturulur.

Metin dizilerine değer atama

Metin dizilerine değer atama işlemi önceki derste anlatılan sayısal dizilerde yapıldığı şekildedir. Değer atama işlemi dizi tanımlaması sırasında yapılabildiği gibi, programın çalışması sırasında da dinamik olarak da yapılabilir. Tanımlama sırasında başlangıç değerlerini atama işlemi özellikle sabit değerler söz konusu olduğunda ve değişkenlerin başlangıç değerleri bilindiği durumlarda idealdir.

1. Dizgiyi tanımlarken değer atama

Dizgiyi tanımlanırken başlangıç değerleri dizi formatında uzunluk değeri belirtilerek ya da belirtilmeyerek tırnak simgeleri " " içinde belirtilebilir. Bu, genellikle sabit bir metin verisinin önceden bilindiği durumlarda kullanılır.

```
char metin[6] = "Selam";
```

```
char mesaj[] = "Merhaba";
```

2. Elemanlara tek tek değer atama

Dizinin her elemanına tek tek değer atanabilir. Bu yöntem, genellikle döngülerle veya elle manuel atama ile yapılır.

```
char metin[6];
metin[0] = 'S';
metin[1] = 'e';
metin[2] = 'l';
metin[3] = 'a';
metin[4] = 'm';
metin[5] = '\0'; // Sonlandırıcı karakter
```

```
char harf[10];
for (int i = 0; i < 9; i++) {
    dizi[i] = 'A'; // Her eleman 'A' ile doldurulur.
}
dizi[9] = '\0'; // Sonlandırıcı karakter.
```

3. Kullanıcı dirdisi ile değer atama

Dizilere değer atamak için kullanıcıdan veri almak oldukça yaygın bir yöntemdir. Bu, dinamik ve etkileşimli uygulamalar geliştirmek için idealdir.

```c
char isim[50];
printf("Isminizi girin: ");
scanf("%s", &isim); // Kullanıcı girdisi "isim" dizisine atanır.

printf("Hoş geldiniz: %s", isim);
```

4. String kütüphane fonksiyonları ile değer atama

C dilinde **string.h** kütüphanesi, dizilere değer atamak veya dizgileri kopyalamak için çeşitli fonksiyonlar sunar.

```c
char kaynak[] = "Merhaba";
char hedef[20];

strcpy(hedef, kaynak); // 'kaynak' dizisi 'hedef' dizisine kopyalanır
```

5. İşaretçi kullanarak dizgiye değer atama

Bir dizgiyi işaretçi aracılığıyla tanımlamak ve değer atamak mümkündür. Bu durumda, dizgi sabit bir dizi yerine belleğin farklı yerlerine atanabilir.

```c
char *dizgi = "Merhaba Dünya!";
printf("%s\n", dizgi); // "Merhaba Dünya!" yazdırılır.
```

Metin dizisi giriş işlemleri

C dilinde dizgilerle girdi işlemleri, kullanıcıdan metin türünde veri almak ve bu veriler üzerinde işlem yapmak için temel bir mekanizma sağlar. Bu işlemler, kullanıcıdan doğrudan giriş almayı kolaylaştıran çeşitli standart kütüphane fonksiyonlarıyla gerçekleştirilir. Bu fonksiyonlar sayesinde, kullanıcının klavyeden girdiği metinler okunabilir, işlenebilir ve gerektiğinde farklı formatlara dönüştürülebilir. Dizgi girdi işlemleri, etkileşimli programlar geliştirmek, kullanıcı girdilerini doğrulamak, metin manipülasyonu yapmak ve dosyalardan veri okumak gibi geniş bir yelpazede kullanılır.

Bu amaçla kullanılan fonksiyonlar arasında **scanf**, **gets** (modern standartlarda kullanımı kaldırılmıştır), **fgets**, **getchar** ve **sscanf** gibi çeşitli araçlar bulunur. Her bir fonksiyonun özellikleri ve kullanım senaryoları farklılık gösterir. Doğru fonksiyonun seçimi, programın güvenliği ve işlevselliği açısından büyük önem taşır.

1. scanf() fonksiyonu

scanf() ile bir dizgi, boşluk veya satır sonu (enter) gibi ayrıştırıcı karakterlere kadar alınır. Ancak bu fonksiyon, sadece tek bir kelime alabilir; boşluk sonrası karakterleri dikkate almaz. Yazım düzeni şöyledir.

```
scanf(char format, char dizgi_adı);
```

```c
#include <stdio.h>
int main() {
    char isim[50];
    printf("Adınızı giriniz: ");
    scanf("%s", isim); // Boşluk öncesine kadar metin alır.
    printf("Merhaba, %s!\n", isim);
    return 0;
}
```

scanf() örnekleri

scanf("%s",dizgi adı) şeklindeki bir yazımda dizgi adı bir dizi olarak tanımlandığından dizgi adının önünde herhangi bir adres operatörü (&) kullanılmamıştır. Bu dizginin adı dizginin ilk elemanına işaret eden bir gösterge olduğunu belirtmektedir. Örneğin, **scanf("%s", kelime)** şeklindeki bir girdi işlemiyle Bilgisayar, Bilgi Sor, Bil kelimelerini girdiğimizde, kelime dizgisinde saklanacak değerler aşağıdaki şekilde olacaktır.

Girdi	Kelime
Bilgisayar	Bilgisayar\0
Bilgi Sor	Bilgi\0
Bil	Bil\0

Burada **"Bilgi Sor"** cümlesinde **"Bilgi"** kelimesi sonrasında bir boşluk olduğu için geride kalan karakterler okunmamıştır. Her durum için **'\0'** otomatik olarak dizgi sonuna eklenmiştir.

scanf() fonksiyonunda, dizginin uzunluğunu sınırlamak için **"%ns"** şeklinde bir format belirteci kullanılabilir. Bu format, kullanıcıdan alınacak dizginin maksimum karakter sayısını tanımlamayı sağlar. Eğer girilen karakter sayısı bu tanımlanan n değerinden daha küçük veya eşitse ve boşluk karakteri girilmemişse, dizgi değişkeni kullanıcıdan alınan tüm karakterleri tutar. Aksi durumda girilen karakterlerin sayısı bu tanıma uymadığından fazla girilen karakterler ve satır sonu karakteri değişkene aktarılmaz.

```
char kelime[11];
scanf("%7s", kelime);
```

Girdi	Kelime
Programlama	Program\0
Prog.	Prog.\0

scanf() fonksiyonunda okunacak karakter sayısı 7 olduğu için 1. kelimenin son 4 karakteri okunmamıştır. 2. kelime karakter sayısı 7'den az olduğu için tüm girdi kelime değişkenine aktarılmıştır.

2) gets() fonksiyonu (kullanımı önerilmez)

Kullanıcı tarafından girilen tüm satırı okur ve sonlandırıcı karakter olan **"\n"** karakterine kadar olan kısmı dizgiye kopyalar. Boşluklar dahil olmak üzere tüm satırı tam olarak almak için kullanılır. Bellek sınırını kontrol etmez, bu nedenle güvenlik riski taşıdığından modern C standartlarında kullanımı kaldırılmıştır. Yazım düzeni şöyledir.

```
char gets(char dizgi_adı);
```

```
#include <stdio.h>
int main() {
    char adres[100];
    printf("Adresinizi giriniz: ");
    gets(adres); // Yazılan tüm metni boşluk dahil alır.
    printf("Adresiniz: %s\n", adres);
    return 0;
}
```

3) fgets() fonksiyonu

Belirtilen karakter sayısı kadar (veya bir satır) giriş almak için kullanılır. Giriş, **stdin** veya bir dosya verisinden yapılabilir. Bellek taşmasını önlemek için maksimum karakter sınırı belirlenir. Ayrıca yeni satır karakterini dizgide saklar ve boşluk içeren girişleri de okuyabilir. Yazım düzeni şöyledir.

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

```
#include <stdio.h>
int main() {
    char mesaj[100];
    printf("Bir mesaj giriniz: ");
    fgets(mesaj, sizeof(mesaj), stdin); // En fazla 99 karakter alır.
    printf("Girdiğiniz mesaj: %s\n", mesaj);
    return 0;
}
```

4) getchar() ve döngülerle kullanımı

Kullanıcıdan birer karakter şeklinde giriş almak için kullanılır. Döngüyle birlikte bir dizgiye karakterleri tek tek doldurabilir. Böylece her girişte bir karakteri okuyarak bir dizgi oluşturulabilir. Fonksiyon sadece tek bir karakter alır ve ASCII değeri döndürür. Girişin sonlandırıcısı, genellikle **Enter** tuşudur ve dönen değer **int** türündedir. Yazım düzeni şöyledir.

```
int getchar(void);
```

```
#include <stdio.h>
int main() {
    char metin[100];
    int i = 0;
    char c;

    printf("Bir metin giriniz (sonlandırmak için Enter): ");

    while ((c = getchar()) != '\n' && i < 99) {
        metin[i++] = c;
    }
    metin[i] = '\0'; // Dizgiyi sonlandır

    printf("Girdiğiniz metin: %s\n", metin);
    return 0;
}
```

5) sscanf() fonksiyonunu

C dilinde bir karakter dizisinden (string) formatlı veri okumak için kullanılan bir fonksiyonudur. Bu fonksiyon, bir dizi içerisindeki verileri belirtilen format doğrultusunda ayrıştırır ve değişkenlere kopyalar. Fonksiyon, başarılı bir şekilde ayrıştırılan ve atanmış değerlerin sayısını döndürür. Eğer hiçbir veri ayrıştırılamazsa, 0-sıfır döndürür.

```
int sscanf (char dizgi_adı, char format, değişken listesi);
```

sscanf() fonksiyonu scanf() gibi kullanıcıdan aldığı giriş verisini değil, program içinde zaten var olan metin dizileri üzerinde işlem yapar. Bu veriler halühazırda belirli bir formatta oldukların kolaylıkla uygun türlere dönüştürülerek istenilen değişkenlere atanır. Şimdi sscanf() fonksiyonunun çeşitli uygulama şekillerine bakalım.

a) Basit sayısal veri ayrıştırma

Bir metin dizisindeki iki tam sayı, %d format belirteçleri kullanılarak ayrıştırılır ve a ile b değişkenlerine atanır.

```
#include <stdio.h>
int main() {
    char veri[] = "25 50";
    int a, b;

    sscanf(veri, "%d %d" , &a, &b);

    printf("Birinci sayı: %d\n", a);
    printf("İkinci sayı: %d\n", a);

    return 0;
}
```

b) Metin ve sayısal veriyi birlikte işlemek

İlk örnekte, bir metin dizisinden bir isim ve bir yaş bilgisi okunarak ilgili değişkenlere atama gerçekleştirilir. İkinci örnekte ise sadece format içinde verilen değerlere atama yapılır, fazla değerler işleme alınmaz.

```c
#include <stdio.h>
int main() {
    char veri[] = "Ad: Ahmet, Yas: 30";
    char isim[20];
    int yas;

    sscanf(veri, "Ad: %s, Yas: %d" , isim, &yas);

    printf("İsim: %s\n", isim);
    printf("Yaş: %d\n", yas);

    return 0;
}
```

```c
char cumle[] = "Sinifa gelmeyenler 15 idi bugun";
char metin1[20], metin2[20];
int i;

sscanf(cumle, "%s %s %d", metin1, metin2, i);

printf("%s--> %d\n", metin1, i);
```

Sinifa	gelmeyenler	15	idi	bugun
%s	%s	%d		
metin 1	metin 2	i		

c) Kayan noktalı sayı ayrıştırma

Bir metin dizisinde belirtilen koordinat verilerini, "float" veri türüne dönüştürülerek atama yapılabilir.

```c
#include <stdio.h>
int main() {
    char veri[] = "Enlem: 40.7128 Boylam: -74.0060";
    float enlem, boylam;

    sscanf(veri, "Enlem: %f Boylam: %f" , &enlem, &boylam);

    printf("Enlem: %.4f\n", enlem);
    printf("Boylam: %.4f\n", boylam);

    return 0;
}
```

Metin dizisi çıktı işlemleri

C dilinde dizgilerle çıktı işlemleri, programın kullanıcıya bilgi aktarmasının temel yollarından biridir. Dizgi verileri, genellikle kullanıcı ekranına veya bir çıktı cihazına (dosya, yazıcı gibi) iletilir. Bu işlemler program sonucunda elde sonuçların kullanıcıya uygun bir şekilde sunabilmesi için kritik bir öneme sahiptir. Dizgilerle çıktı işlemleri, metin tabanlı kullanıcı arayüzlerinden rapor oluşturmaya kadar birçok farklı uygulama alanında kullanılır. Özellikle metin verileriyle çalışan uygulamalarda, dizgilerin doğru ve biçimlendirilmiş bir şekilde sunulması, kullanıcı deneyimini ve programın etkinliğini artırır.

Bu işlemler için C dilinde standart kütüphane fonksiyonlarından **printf, puts,** gibi fonksiyonlar, dizgi verilerinin ekrana yazdırılmasını kolaylaştırırken **sprintf** fonksiyonu verileri dizgilere yazma imkanı verir. **fprintf** ve **fputs** gibi fonksiyonlar ise çıktının dosyalara veya belirli cihazlara yönlendirilmesine olanak tanır. Ayrıca, çıktının formatlanması veya düzenlenmesi gerektiğinde, bu fonksiyonlar esneklik sunar.

1) printf() fonksiyonu

Formatlı bir şekilde dizgileri ve diğer verileri ekrana yazdırmak için kullanılır. Dizgi içinde özel karakterlerle (%s, %d gibi) yer tutucular kullanılarak çıktı biçimi kontrol edilebilir. Yazım düzeni şöyledir.

```
printf(char format, dizgi_adı);
```

```
char metin1[15] = "Selamlar";
char metin2[] = "iyi";
```

metin1 ve metin2'nin farklı printf() kullanıma ait çıktılar yandaki gibidir.

Girdi	Kelime
printf("%s", metin1);	Selamlar
printf("%s", "Merhaba");	Merhaba
printf("%5s", metin2);	• • i y i sağa dayalı
printf("%-5s", metin2);	i y i • • sola dayalı
printf("%s %s", metin1, metin2);	Selamlar iyi

2) sprintf() fonksiyonu

Bir metin dizisine (string) biçimlendirilmiş veri yazmak için kullanılır. Burada çıktı ekran yerine bir karakter dizisine aktarılmaktadır. Bu işlev, çıktı işlemlerinin daha fazla kontrol edilmesini ve geçici verilerin dizgiler içinde saklanmasını sağlar. Tamsayılar, ondalıklı sayılar, karakterler gibi farklı veri türlerini biçimlendirilmiş bir şekilde ayarlanıp sonradan istenilen her hangi çıktı birimine (terminal ya da dosyaya) yazılmasını kolaylaştırır. Yazım düzeni şöyledir.

```
sprintf(char dizge_adı, format, değişkenler);
```

basit birleştirme işlemi

```
char metin[];
int yas;

sprintf(metin, "Yaşınız: %d", yas);
printf("%s\n", metin);
```

```
Yaşınız: 25
```

birden fazla değeri formatlı yazma

```
char mesaj[100];
char isim[] = "Ali";
int puan = 95;

sprintf(mesaj, "Merhaba %s, sınav puanınız %d.", isim, puan);
printf("%s\n", mesaj);
```

```
Merhaba Ali, sınav puanınız 95.
```

ondalıklı sayı formatlama

```
char sonuc[50];
float oran = 5.678;

sprintf(sonuc, "Sonuc: %.2f", oran);
printf("%s\n", sonuc);
```

```
Sonuç: 5.68
```

```
char yol[80];
float zaman = 4, hiz = 60;

sprintf(yol, "Yol %5.3f km olarak bulunmaktadır.", zaman*hiz);
printf("%s\n", yol);
```

```
Yol 240.000 km olarak bulunmaktadır.
```

3) puts() fonksiyonu

Bir dizgiyi hızlı bir şekilde ekrana yazdırır ve ardından bir yeni satır karakteri **"\n"** ekler. Karmaşık formatlama gerektirmeyen ve sadece dizgiyi ekrana aktarma amacı taşıyan durumlarda kullanılır ve **printf**'e kıyasla daha hızlı çalışabilir. Yazım düzeni şöyledir.

```
char puts(char dizgi_adı);
```

```
char metin1[] = "Merhaba";
char metin2[] = "Nasilsiniz?";

puts(metin1);
puts(metin2);
```

```
Merhaba
Nasilsiniz?
```

4) fprintf() ve fputs() fonksiyonları

fprintf(), biçimlendirilmiş veriyi bir dosyaya veya belirli bir çıktı akışına yazmak için kullanılırken **fputs()** her hangi bir biçimlendirme kullanılmadan metin dizilerini yazmak için kullanılır. fprintf() fonksiyonu, formatlama özellikleri sayesinde tamsayı, ondalıklı sayı, dizgi gibi farklı veri türlerini birleştirerek belirli bir düzenle çıktı sağlayabilir. fopen() ve fclose() fonksiyonları ile dosya açma ve kapa işlemleri gerçekleştirilir. Yazım düzenleri şöyledir.

```
fprintf(FILE *dosya, format, değişkenler);
```

```
fputs(char dizgi_adı, FILE *dosya);
```

```
FILE *dosya = fopen("rapor.txt", "w");
if (dosya == NULL) {
    printf("Dosya açılamadı.\n");
    return 1;
}
```

```
char isim[] = "Ayşe", bilgi[] = "Çıktı sonu";
int puan = 85;
fprintf(dosya, "Öğrenci Adı: %s\n", isim);
fprintf(dosya, "Sınav Puanı: %d\n", puan);
fputs(bilgi, dosya);
fclose(dosya);
```

rapor.txt

```
Öğrenci Adı: Ayşe
Sınav Puanı: 85
Çıktı sonu
```

Metin dizisi fonksiyonları

Dizgi işlemleri, programlamada sıkça karşılaşılan bir ihtiyaçtır ve bu işlemleri kolaylaştırmak için C dilinde birçok hazır fonksiyon sunulmuştur. Bu fonksiyonlar, dizgiler üzerinde kopyalama, birleştirme, karşılaştırma gibi işlemleri daha hızlı ve hatasız bir şekilde gerçekleştirebilmek için geliştirilmiştir. Bu fonksiyonların tamamı, standart bir C kütüphanesi olan **"string.h"** başlık dosyasında tanımlanmıştır. Bu fonksiyonlardan yararlanmak için mutlaka programın başında;

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

şeklinde bir tanımlamanın bulunması gerekir. Bu tanımlama, programın string.h dosyasındaki fonksiyonlara erişilmesini sağlar. string.h kütüphanesinde 22 tane fonksiyon bulunmaktadır. Burada yaygın kullanılan 8 tanesi hakkında bilgi verilecektir.

Fonksiyon	Açıklaması
memchr()	Bellek bloğundaki bir değerin ilk oluşumuna bir işaretçi döndürür
memcmp()	Hangisinin daha büyük bir sayısal değeri temsil ettiğini belirlemek için iki bellek bloğunu karşılaştırır
memcpy()	Verileri bir bellek bloğundan diğerine kopyalar
memmove()	Bellek bloklarının çakışması olasılığını hesaba katarak verileri bir bellek bloğundan diğerine kopyalar
memset()	Bir bellek bloğundaki tüm baytları aynı değere ayarlar
strcat()	Bir dizeyi diğerinin sonuna ekler
strchr()	Dizede bir karakterin ilk geçtiği yere bir işaretçi döndürür
strcmp()	Hangi dizenin daha yüksek değere sahip olduğunu belirlemek için iki dizedeki karakterlerin ASCII değerlerini karşılaştırır
strcoll()	Hangi dizenin daha yüksek değere sahip olduğunu belirlemek için iki dizedeki karakterlerin yerel tabanlı değerlerini karşılaştırır
strcpy()	Bir dizenin karakterlerini başka bir dizenin belleğine kopyalar
strcspn()	Belirtilen karakterlerden birinin ilk geçtiği yere kadar olan dizenin uzunluğunu verir

Fonksiyon	Açıklaması
strerror()	Hata kodunun anlamını açıklayan bir dize döndürür
strlen()	Bir dizenin uzunluğunu döndürür
strncat()	Bir dizeden belirli sayıda karakteri başka bir dizenin sonuna ekler
strncmp()	Hangi dizenin daha yüksek değere sahip olduğunu belirlemek için iki dizede belirtilen sayıda karakterin ASCII değerlerini karşılaştırır
strncpy()	Bir dizeden bir dizi karakteri başka bir dizenin belleğine kopyalar
strpbrk()	Dizede belirtilen karakterlerden birini içeren ilk konuma bir gösterici döndürür
strrchr()	Dizede bir karakterin son geçtiği yere bir işaretçi döndürür
strspn()	Belirtilen karakterlerden biri olmayan ilk karaktere kadar olan dizenin uzunluğunu döndürür
strstr()	Bir dizenin başka bir dize içinde ilk geçtiği yere bir gösterici döndürür
strtok()	Sınırlayıcıları kullanarak bir dizeyi parçalara böler
strxfrm()	Bir dizedeki karakterleri ASCII kodlamasından geçerli yerel ayarın kodlamasına dönüştürür

1) strlen() fonksiyonu

Bir dizginin uzunluğunu (karakter sayısını), sonlandırıcı **"\0"** karakteri hariç, hesaplar. **strlen(dizgi_adı)** şeklinde yazım düzeniyle kullanılır.

```
char metin[] = "Merhaba";
size_t harfsay= strlen(metin);

printf("Metnin harf sayısı: %d\n", harfsay);
```

```
Metnin harf sayısı: 7
```

2) strcpy() fonksiyonu

Bir metin dizisini başka bir metin dizisine kopyalar. Yazım düzeni **strcpy(hedef dizgi_adı, kaynak dizgi_adı)** şeklindedir.

```
char kaynak[] = "Merhaba";
char hedef[20];
strcpy(hedef, kaynak);
printf("Hedef dizi: %s\n", hedef);
```

```
Hedef dizi: Merhaba
```

3) strncpy() fonksiyonu (sınırlı sayıda dizgi kopyalama)

Kaynak dizgiden belirtilen karakter sayısı kadarını hedef dizgiye kopyalar. Yazım düzeni **strncpy(hedef dizgi_adı, kaynak dizgi_adı, sayı n)** şeklindedir.

```
char kaynak[] = "Merhaba";
char hedef[20];
strncpy(hedef, kaynak, 4);
printf("Hedef dizi: %s\n", hedef);
```

```
Hedef dizi: Merh
```

4) strcat() fonksiyonu

Bir metin dizisini başka bir metin dizisinin sonuna ekler. Yazım düzeni **strcat(hedef dizgi_adı, kaynak dizgi_adı)** şeklindedir.

```
char metin1[20] = "Selamlar ";
char metin2[] = "herkese";
strcat(metin1, metin2);
printf("Mesaj: %s\n", metin1);
```

```
Mesaj: Selamlar herkese
```

5) strncat() fonksiyonu (sınırlı sayıda dizgi birleştirme)

Bir metin dizisini başka bir metin dizisinin sonuna belirtilen karakter sayısı kadar ekleme yapar. Yazım düzeni **strncat(hedef dizgi_adı, kaynak dizgi_adı, sayı n)** şeklindedir.

```
char metin1[20] = "Selamlar ";
char metin2[] = "herkese";
strncat(metin1, metin2, 5);
printf("Mesaj: %s\n", metin1);
```

```
Mesaj: Selamlar herke
```

6) strcmp() fonksiyonu

İki metin dizgisini alfabetik sıraya göre karşılaştırır. Karşılaştırma durumları her karakterin ASCII karakter tablosundaki sıralamalarına göre bulunur. Yazım düzeni **strcmp(hedef dizgi_adı, kaynak dizgi_adı)** şeklindedir. İki dizgi eşit ise "0", ilk dizgi, ikinci dizgiden önce geliyorsa "-1", sonra geliyorsa "1" tamsayı değeri döner.

```
char metin1[] = "Elma", metin2[] = "Armut", metin3[] = "Visne";
int sonuc = strcmp(metin1, metin2);
int sonuc1 = strcmp(metin1, metin3);
printf("Mesaj: %d, %d\n", sonuc, sonuc1);
```

```
Mesaj: 1, -1
```

7) strncmp() fonksiyonu (sınırlı karakter dizgisi karşılaştırma)

İki metin dizgisinin sadece ilk n karakterini karşılaştırır. Yazım düzeni **strcmp(hedef dizgi_adı, kaynak dizgi_adı , sayı n)** şeklindedir. İki dizgi eşit ise "0", ilk dizgi, ikinci dizgiden önce geliyorsa "-1", sonra geliyorsa "1" tamsayı değeri döner.

```
char metin1[] = "Elma", metin2[] = "Elmas";
int sonuc = strncmp(metin1, metin2, 4);

printf("Mesaj: %d, %d\n", sonuc);
```

```
Mesaj: 0
```

8) strstr() fonksiyonu

Bir metin dizisi içinde başka bir metin dizisini arar. Yazım düzeni **strstr(samanlık dizgi_adı, iğne dizgi_adı)** şeklindedir. Dönüş değeri **"iğne"** dizgisinin ilk bulunduğu yeri gösteren bir göstergeyi döndürür, bulunamazsa NULL değeri döner.

```
#include <stdio.h>
#include <string.h>
int main() {
    char samanlik[] = "Bugun hava cok guzel, hava oldukca sicak.";
    char igne[] = "hava";

    char *sonuc = strstr(samanlik, igne);
    while (result != NULL) {
        printf("Bulunan alt dizgi: %s\n", sonuc);
        sonuc = strstr(sonuc + 1, igne); // Aramaya bulunduğu yerden sonrasıyla devam et
    }
    return 0;
}
```

```
Bulunan alt dizgi: hava çok güzel, hava oldukça sıcak.
Bulunan alt dizgi: hava oldukça sıcak.
```

Karakter fonksiyonları

C dilinde karakterlerle işlem yapmak için **"ctype.h"** kütüphanesi içinde tanımlanan karakter fonksiyonları kullanılır. Bu fonksiyonlar, karakterlerin türünü belirlemek ve karakter üzerinde işlem yapmak için hızlı ve kullanışlı araçlar sunar. Özellikle karakterlerin alfasayısal (alfanumeric), harf (alphabetic), rakam (digit) gibi özelliklerini kontrol etmek ve dönüşümler yapmak için kullanılır. Kütüphaneyi programda kullanmak aşağıdaki tanımlama yapılmalıdır.

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

Karakter türü kontrol fonksiyonları

Fonksiyon	Açıklama	Örnek Kullanım
isalpha	Karakter bir harf (A-Z veya a-z) ise 1 (doğru), değilse 0 döner.	isalpha('a') → 1
isdigit	Karakter bir rakam (0-9) ise 1, değilse 0 döner.	isdigit('3') → 1
isalnum	Karakter harf veya rakam ise 1, değilse 0 döner.	isalnum('G') → 1
isspace	Karakter bir boşluk, tab, yeni satır vb. ise 1, değilse 0 döner.	isspace(' ') → 1
islower	Karakter küçük harfse 1, değilse 0 döner.	islower('b') → 1
isupper	Karakter büyük harfse 1, değilse 0 döner.	isupper('X') → 1
ispunct	Karakter noktalama işareti ise 1, değilse 0 döner.	ispunct('!') → 1
isprint	Karakter yazdırılabilir (ASCII 32-126) ise 1, değilse 0 döner.	isprint('A') → 1

Karakter dönüşüm fonksiyonları

tolower	Büyük harfi küçük harfe çevirir. Diğer karakterler aynen döner.	tolower('G') → 'g'
toupper	Küçük harfi büyük harfe çevirir. Diğer karakterler aynen döner.	toupper('g') → 'G'
toascii	Verilen ASCII koduna sahip karakteri geri döner.	toascii(65) → 'A'

Karakter fonksiyonlarıyla örnek kodlar

Örnek 1: Karakterin türünü belirleme

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

int main() {
    char harf = 'A';

    if (isalpha(harf)) {
        printf("%c bir harftir.\n", harf);

    } else if (isdigit(harf)) {
        printf("%c bir rakamdır.\n", harf);

    } else {
        printf("%c ne harf ne de rakamdır.\n", harf);
    }

    return 0;
}
```

```
A bir harftir.
```

Bu kodda **"harf"** değişkenine verilen farklı değerlere göre değişkenin harf ya da rakam olduğu mesajı yazılmaktadır. Örnekte harf='A' verildiğinden mesaj "A bir harftir." şeklindedir. Eğer harf='9' şeklinde verilirse mesaj "9 bir rakamdır." olacaktır.

Karakter fonksiyonlarıyla örnek kodlar

Örnek 2: Bir dizgideki harfleri ve rakamları sayma

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

int main() {
    char metin[] = "Merhaba123!";
    int harfSayisi = 0, rakamSayisi = 0;

    for (int i = 0; metin[i] != '\0'; i++) {
        if (isalpha(metin[i])) {
            harfSayisi++;

        } else if (isdigit(metin[i])) {
            rakamSayisi++;
        }
    }

    printf("Harf sayısı: %d\n", harfSayisi);
    printf("Rakam sayısı: %d\n", rakamSayisi);

    return 0;
}
```

```
Harf sayısı: 7
Rakam sayısı: 3
```

Bu kodda **"metin"** değişkenine verilen harf ve rakamlardan oluşan farklı değerler verilerek test yapılabilir. Bu örnekte gösterildiği üzere "!" işareti harf olarak sayılmamıştır.

Çalışma soruları

1) İçinde küçüklü büyüklü harfler bulunan bir metindeki bütün harfleri küçük harfe çevirip terminale yazdıran kodu yazınız.

2) Kullanıcıdan aldığı bir metni tersten yazdıran kodu yazınız.

3) Kullanıcıdan bir döngü ile aldığı 5 metin dizisini "output.txt" dosyasına yazdıran kodu yazınız.