C Characters and Strings

Bu derste, dizeleri ve karakterleri işlememize yardımcı olan C standart kütüphane fonksiyonlarını tanıtacağız. Fonksiyonlar, programların karakterleri, dizeleri, metin satırlarını ve bellek bloklarını işlemesini sağlar. Bu bölümde editörler, kelime işlemciler, sayfa düzeni yazılımları, bilgisayarlı dizgi sistemleri ve diğer metin işleme yazılımları geliştirmek için kullanılan teknikler verilecektir. “printf” ve “scanf” gibi biçimlendirilmiş giriş/çıkış işlevleri tarafından gerçekleştirilen metin işlemleri, bu bölümde tartışılan fonksiyonlar kullanılarak gerçekleştirilebilir.

Karakterler, kaynak programların temel yapı taşlarıdır. Bir program karakter sabitleri içerebilir. Bir karakter sabiti, tek tırnak içinde bir karakter olarak temsil edilen bir “int” değeridir. Bir karakter sabitinin değeri, karakter setindeki karakterin tamsayı değeridir. Örneğin, 'z', z harfinin tamsayı değerini ve '\n' yeni satırın tamsayı değerini temsil eder (ASCII'de sırasıyla 122 ve 10)(NOT: b’00000000’ dan b’11111111’ e kadar binary sayılar desimal olarak 0’dan 127’ye kadar sayıları ifade eder)

Bir dize, tek bir birim olarak ele alınan bir dizi karakterdir. Bir dize, harfleri, rakamları ve +, -, *, / ve $ gibi çeşitli özel karakterleri içerebilir. C'deki dize hazır değerleri veya dize sabitleri aşağıdaki gibi çift tırnak içinde yazılır:

"Celaleddin" (isim)
"125 nci sokak" (sokak adresi)
"İskenderun, hatay" (şehir ve bölge)
"(232) 555-1212" (telefon no)

C'deki bir dize, boş karakterle ('\0') biten bir karakter dizisidir. Bir diziye, dizideki ilk karaktere bir işaretçi aracılığıyla erişilir. Bir dizenin değeri, ilk karakterinin adresidir. Bu nedenle, C'de bir dizenin bir işaretçi olduğunu söylemek uygundur - aslında, dizenin ilk karakterini gösteren bir işaretçidir. Bu tıpkı diziler gibidir, çünkü dizeler basitçe karakter dizileridir.

Bir karakter dizisi veya char* türündeki bir değişken, tanımdaki bir dize ile başlatılabilir. Aşağıdaki tanımların her biri, "blue" dizisine bir değişken başlatır. İlk tanım, 'b', 'l', 'u', 'e' ve '\0' karakterlerini içeren 5 öğeli bir color dizisi oluşturur. İkinci tanım, salt okunur bellekte bir yerde "blue" dizesine işaret eden colorPtr işaretçi değişkenini oluşturur.

```
char color[] = "blue";
const char *colorPtr = "blue";
```

Önceki dizi tanımı şu şekilde de yazılabilirdi;

```
char color[] = { 'b', 'l', 'u', 'e', '\0' };
```

Bir karakter dizisini bir dize içerecek şekilde tanımlarken, dizi, diziyi ve onun sonlandırıcı boş karakterini saklayacak kadar büyük olmalıdır. Önceki tanım, başlatıcı listesindeki başlatıcı sayısına (5) dayalı olarak dizinin boyutunu otomatik olarak belirler.

Bir dize, scanf kullanılarak bir dizide saklanabilir. Örneğin, aşağıdaki ifade, word [20] karakter dizisinde bir dize string () depolar:

```
scanf("%19s", word);
```

Kullanıcı tarafından girilen dizi ‘word’ değişkeninde saklanır. Değişken ‘word’ bir işaretçi olan bir dizidir, bu nedenle bağımsız değişken kelimesinde & karakterine gerek yoktur. Bildiğimiz gibi, “scanf” fonksiyonu bir boşluk, sekme, yeni satır veya dosya sonu göstergesiyle karşılaşılıncaya kadar karakterleri okur. Dolayısıyla, %19s dönüştürme belirleyicisinde 19 alan genişliği olmadan, kullanıcı girişi 19 karakteri geçebilir ve programınız çökebilir! Bu nedenle, bir karakter dizisini okumak için “scanf” kullanırken her zaman bir alan genişliği kullanmalısınız. Önceki ifadedeki alan genişliği 19, scanf'in maksimum 19 karakter okumasını ve dizenin sonlandırıcı null karakteri için son karakteri kaydetmesini sağlar. Bu, scanf'in karakterleri karakter dizisinin sonundan sonra belleğe yazmasını engeller. (İsteğe bağlı uzunluktaki giriş satırlarını okumak için, genellikle “stdio.h” içinde yer alan, standart olmayan ancak yaygın olarak desteklenen bir fonksiyon okuma satırı vardır.) Bir karakter dizisinin düzgün bir şekilde dize olarak yazdırılabilmesi için, dizinin sonlandırıcı bir boş karakter (null) içermesi gerekir.

Karakter işleme kütüphanesi (<ctype.h>), karakter verileri üzerinde yararlı testler ve manipülasyonlar gerçekleştiren çeşitli fonksiyonlar içerir. Her fonksiyon, bağımsız değişken olarak işaretsiz bir karakter (int olarak temsil edilir) veya EOF (End of File) (dosya sonu) alır. Bildiğimiz gibi, karakterler genellikle tamsayılar olarak işlenir, çünkü C'deki bir karakter bir baytlık bir tamsayıdır. EOF normalde -1 değerine sahiptir. Şekil 8.1, karakter işleme kütüphanesinin fonksiyonlarını özetlemektedir.

Fonksiyon Prototipi	Fonksiyon tanımı
int isblank(**int** c);	c değişkeni, bir metin satırındaki sözcükleri ayıran boş bir karakterse 1 (true) , aksi halde 0 (false) döndürür.
int isdigit(int c);	c değişkeni, bir rakam ise 1 (true), aksi takdirde 0 (false) değerini döndürür.
int isalpha(int c);	c değişkeni, bir harf ise 1 (true), aksi takdirde 0 (false) değerini döndürür.
int isalnum(int c);	c bir rakam veya harf ise 1 (true), aksi takdirde 0 (false) değerini döndürür.
int isxdigit(int c);	c bir hexadesimal karakteriyse 1 (true), aksi takdirde 0 (false) döndürür.
int islower(int c);	c küçük harf ise 1 (true), aksi takdirde 0 (false) değerini döndürür.
int isupper(int c);	c büyük harf ise 1 (true), aksi takdirde 0 (false) değerini döndürür.
int tolower(int c);	c büyük harf ise, tolower c'yi küçük harf olarak döndürür. Aksi takdirde, tolower bağımsız değişkeni değiştirmeden döndürür.
int toupper(int c);	c küçük harf ise, toupper c'yi büyük harf olarak döndürür. Aksi takdirde, toupper argümanı değiştirmeden döndürür.
int isspace(int c);	c bir boşluk karakteri ise 1 (true) döndürür; [yeni satır ('\n'), boşluk (' '), form girme ('\f'), satır başı ('\r'), yatay sekme ('\t')) veya dikey sekme ('\v')]— aksi halde 0 (false) değerini döndürür.
int iscntrl(int c);	c bir kontrol karakteri ise 1 (true) döndürür; (yatay sekme ('\t'), dikey sekme ('\v'), form besleme ('\f'), uyarı ('\a'), geri al ('\ b'), satır başı ('\r'), yeni satır ('\n') ve diğerleri) aksi halde 0 (false) değerini döndürür.
int ispunct(int c);	c boşluk, rakam veya harften farklı bir yazdırma karakteriyse ($, #, (,), [,], {, }, ;, : veya % gibi) 1 (true) değer döndürür ve aksi takdirde 0 döndürür .
int isprint(int c);	c, boşluk içeren bir yazdırma karakteri (yani ekranda görünen bir karakter) ise 1 (true) değer döndürür ve aksi takdirde 0 (false) döndürür.
int isgraph(int c);	c boşluk dışında bir yazdırma karakteriyse 1 (true) değer döndürür ve aksi takdirde 0 (false) döndürür.

Fig. 8.1 | Character-handling library (<ctype.h>) functions.

Şekil 8.2 isdigit, isalpha, isalnum ve isxdigit fonksiyonlarını göstermektedir. isdigit fonksiyonu, argümanının bir rakam (0-9) olup olmadığını belirler. isalpha fonksiyonu bağımsız değişkeninin büyük harf mi (A–Z) yoksa küçük harf mi (a–z) olduğunu belirler. isalnum fonksiyonu bağımsız değişkeninin büyük harf mi, küçük harf mi yoksa rakam mı olduğunu belirler. isxdigit fonksiyonu, bağımsız değişkeninin onaltılık bir basamak (A–F, a–f, 0–9) olup olmadığını belirler.

```c
// Fig. 8.2: fig08_02.c
// Using functions isdigit, isalpha, isalnum, and isxdigit
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
      isdigit('8')   ? "8 is a " : "8 is not a ", "digit",
      isdigit('#')   ? "# is a " : "# is not a ", "digit");

   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
      "According to isalpha:",
      isalpha('A')   ? "A is a " : "A is not a ", "letter",
      isalpha('b')   ? "b is a " : "b is not a ", "letter",
      isalpha('&')   ? "& is a " : "& is not a ", "letter",
      isalpha('4')   ? "4 is a " : "4 is not a ", "letter");

   printf("%s\n%s%s\n%s%s\n%s%s\n\n",
      "According to isalnum:",
      isalnum('A')   ? "A is a " : "A is not a ",
      "digit or a letter",
      isalnum('8')   ? "8 is a " : "8 is not a ",
      "digit or a letter",
      isalnum('#')   ? "# is a " : "# is not a ",
      "digit or a letter");

   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
      "According to isxdigit:",
      isxdigit('F')   ? "F is a " : "F is not a ",
      "hexadecimal digit",
      isxdigit('J')   ? "J is a " : "J is not a ",
      "hexadecimal digit",
      isxdigit('7')   ? "7 is a " : "7 is not a ",
      "hexadecimal digit",
      isxdigit('$')   ? "$ is a " : "$ is not a ",
      "hexadecimal digit",
      isxdigit('f')   ? "f is a " : "f is not a ",
      "hexadecimal digit");
}
```

Şekil 8.2 “isdigit”, “isalpha”, “isalnum” ve “isxdigit” fonksiyonlarını kullanma.

Şekil 8.2, test edilen her karakter için çıktıda " is a " dizisinin mi yoksa " is not a " dizisinin mi yazdırılacağını belirlemek için koşullu operatörü (?:) kullanır. Örneğin aşağıdaki ifadedeki gibi;

```c
isdigit('8') ? "8 is a " : "8 is not a "
```

Bu ifade; '8' bir rakamsa, "8 is a " dizisinin yazdırıldığını ve '8' bir rakam değilse (isdigit 0 döndürür), "8 is not a" yazdırıldığını gösterir.

Şekil 8.3 “islower”, “isupper”, “tolower” ve “toupper” fonksiyonlarını göstermektedir. “islower” fonksiyonu bağımsız değişkeninin küçük harf (a–z) olup olmadığını belirler. “isupper” fonksiyonu, bağımsız değişkeninin büyük harf (A–Z) olup olmadığını belirler. “tolower” fonksiyonu, büyük harfi küçük harfe dönüştürür ve küçük harfi döndürür. Argüman büyük harf değilse, “tolower” argümanı değiştirmeden döndürür. “toupper” fonksiyonu, küçük harfi büyük harfe dönüştürür ve büyük harfi döndürür. Argüman küçük harf değilse, toupper argümanı değiştirmeden döndürür.

```c
// Fig. 8.3: fig08_03.c
// Using functions islower, isupper, tolower and toupper
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
      "According to islower:",
      islower('p')   ? "p is a " : "p is not a ",
      "lowercase letter",
      islower('P')   ? "P is a " : "P is not a ",
      "lowercase letter",
      islower('5')   ? "5 is a " : "5 is not a ",
      "lowercase letter",
      islower('!')   ? "! is a " : "! is not a ",
      "lowercase letter");

   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
      "According to isupper:",
      isupper('D')   ? "D is an " : "D is not an ",
      "uppercase letter",
      isupper('d')   ? "d is an " : "d is not an ",
      "uppercase letter",
      isupper('8')   ? "8 is an " : "8 is not an ",
      "uppercase letter",
      isupper('$')   ? "$ is an " : "$ is not an ",
      "uppercase letter");

   printf("%s%c\n%s%c\n%s%c\n%s%c\n",
      "u converted to uppercase is ", toupper('u')  ,
      "7 converted to uppercase is ", toupper('7')  ,
      "$ converted to uppercase is ", toupper('$')  ,
      "L converted to lowercase is ", tolower('L')  );
}
```

Şekil 8.3 "Islower", "isupper", "tolower" ve "toupper" fonksiyonlarını kullanma.

Bu bölümde, genel yardımcı programlar kütüphanesinden (<stdlib.h>) dize dönüştürme fonksiyonlarını ele alalım. Bu fonksiyonlar, basamak dizilerini tamsayı ve kayan noktalı değerlere dönüştürür. Şekil 8.5, dizi dönüştürme fonksiyonlarını özetlemektedir. C standardı ayrıca, dizeleri sırasıyla “long”, “long int” ve “unsigned long int”e dönüştürmek için "strtol" ve "strtod" fonksiyonlarını kullanır. Fonksiyon başlıklarında "nPtr" değişkenini bildirmek için “const” kullanımına dikkat edelim; “const”, bağımsız değişken değerinin değiştirilmeyeceğini belirtir.

Fonksiyon prototipi	Fonksiyon tanımı
double strtod(**const char** *nPtr, **char** **endPtr);	nPtr dizesini (string) “double”a dönüştürür.
long strtol(**const char** *nPtr, **char** **endPtr, **int** base);	nPtr dizesini (string) “long”a dönüştürür.
unsigned long strtoul(**const char** *nPtr, **char** **endPtr, **int** base);	nPtr dizesini unsigned long'a dönüştürür.

8.5 Genel yardımcı programlar kütüphanesinden dize dönüştürme işlevleri.

“strtod” fonksiyonu (Şekil 8.6), kayan nokta değerini temsil eden bir karakter dizisini double’a dönüştürür. Fonksiyon, ilk bağımsız değişkeninin herhangi bir bölümünü double’a dönüştüremezse 0 döndürür. Fonksiyon iki bağımsız değişken alır; bir (string) dize (char *) ve bir (string’e) dizeye işaretçi (char**). Dize bağımsız değişkeni, double’a dönüştürülecek karakter dizisini içerir; dizenin başındaki tüm boşluk karakterleri yok sayılır. Fonksiyon, çağıran fonksiyondaki (stringPtr) bir char*'ı değiştirmek için char** bağımsız değişkenini kullanır, böylece dizenin dönüştürülen kısmından sonraki ilk karakterin konumuna veya hiçbir bölüm dönüştürülemezse tüm dizeye işaret eder. 11. satır d'ye dizeden dönüştürülen double değerin atandığını ve stringPtr'nin dizede dönüştürülen değerden (51.2) sonraki ilk karakterin konumuna atandığını gösterir.

```
1  // Fig. 8.6: fig08_06.c
2  // Using function strtod
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main(void)
7  {
8     const char *string = "51.2% are admitted"; // initialize string
9     char *stringPtr; // create char pointer
10
11    double d = strtod(string, &stringPtr);
12
13    printf("The string \"%s\" is converted to the\n", string);
14    printf("double value %.2f and the string \"%s\"\n", d, stringPtr);
15 }
```

Şekil 8.6 “strtod” fonksiyonunu kullanma.

“strtol” fonksiyonu (Şekil 8.7), bir tamsayıyı temsil eden bir karakter dizisini “long int”e dönüştürür. Fonksiyon, ilk bağımsız değişkeninin herhangi bir bölümünü “long int”e dönüştüremezse 0 döndürür. Fonksiyonun üç argümanı bir (string) dize (char*), bir dizeye işaretçi ve bir tam sayıdır. Dize, “long’a dönüştürülecek karakter dizisini içerir; dizenin başındaki boşluk karakterleri yok sayılır. Fonksiyon, çağıran fonksiyondaki (remainderPtr) bir char*'ı değiştirmek için char** bağımsız değişkenini kullanır, böylece dizenin dönüştürülen kısmından sonraki ilk karakterin konumuna veya hiçbir bölüm dönüştürülemezse tüm dizeye işaret eder. Tamsayı, dönüştürülmekte olan değerin tabanını belirtir.

```
1  // Fig. 8.7: fig08_07.c
2  // Using function strtol
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main(void)
7  {
8     const char *string = "-1234567abc"; // initialize string pointer
9     char *remainderPtr; // create char pointer
10
11    long x = strtol(string, &remainderPtr, 0);
12
13    printf("%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
14       "The original string is ", string,
15       "The converted value is ", x,
16       "The remainder of the original string is ",
17       remainderPtr,
18       "The converted value plus 567 is ", x + 567);
19 }
```

11. satır, dizeden (string’den) dönüştürülen long değerin x'e atandığını gösterir. “remainderPtr” adlı ikinci bağımsız değişkene, dönüştürmeden sonra dizenin geri kalanı atanır. İkinci bağımsız değişken için NULL kullanılması, dizenin geri kalanının yok sayılmasına neden olur. Üçüncü bağımsız değişken olan 0, dönüştürülecek değerin sekizlik (8 tabanlı), ondalık (10 tabanlı) veya onaltılı (16 tabanlı) biçimde olabileceğini belirtir.
Taban 0 olarak veya 2 ile 36 arasında herhangi bir değer olarak belirtilebilir. 10 tabanından 36 tabanına kadar olan tam sayıların sayısal gösterimleri A–Z karakterlerini kullanır. 10 ile 35 arasındaki değerleri temsil eder. Örneğin, onaltılık değerler 0–9 rakamlarından ve A–F karakterlerinden oluşabilir. 11 tabanlı bir tamsayı, 0-9 rakamlarından ve A karakterinden oluşabilir. 24 tabanlı bir tam sayı, 0-9 rakamlarından ve A–N karakterlerinden oluşabilir. 36 tabanlı bir tamsayı, 0–9 rakamlarından ve A–Z karakterlerinden oluşabilir.

“strtoul” fonksiyonu (Şekil 8.8), işaretsiz bir “long int” değerini temsil eden bir karakter dizisini işaretsiz “long int”e dönüştürür. Fonksiyon, “strtol” fonksiyonu ile aynı şekilde çalışır. 11. satır, x'e dizeden dönüştürülen işaretsiz long int değerinin atandığını gösterir. İkinci argüman olan &remainderPtr, dönüşümden sonra dizenin geri kalanına atanır. Burada 0 ise , dönüştürülecek değerin sekizlik, ondalık veya onaltılık biçimde olabileceğini belirtir.

```
1  // Fig. 8.8: fig08_08.c
2  // Using function strtoul
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main(void)
7  {
8     const char *string = "1234567abc"; // initialize string pointer
9     char *remainderPtr; // create char pointer
10
11    unsigned long int x = strtoul(string, &remainderPtr, 0);
12
13    printf("%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
14       "The original string is ", string,
15       "The converted value is ", x,
16       "The remainder of the original string is ",
17       remainderPtr,
18       "The converted value minus 567 is ", x - 567);
19 }
```