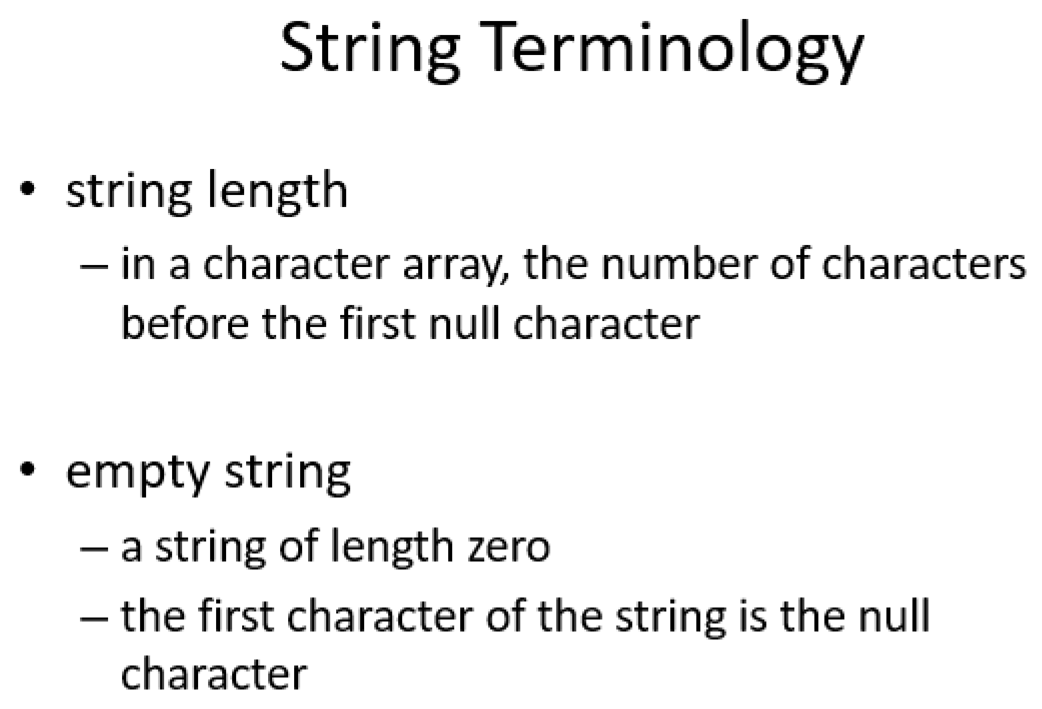


String Terminology

- string length
  - in a character array, the number of characters before the first null character
- empty string
  - a string of length zero
  - the first character of the string is the null character

Concatenation

- strcat
  - appends source to the end of dest
  - assumes that sufficient space is allocated for the first argument to allow addition of the extra characters
    - s1 = “hello”;
    - strcat(s1, “and more”);

h	e	l	l	o	a	n	d		m	o	r	e	\0

Concatenation

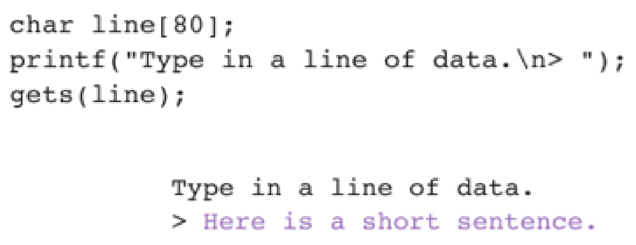
- strncat
  - appends up to n characters of source to the end of dest, adding the null character if necessary
  - assumes that sufficient space is allocated for the first argument to allow addition of the extra characters
    - s1 = “hello”;
    - strncat(s1, “and more”, 5);

h	e	l	l	o	a	n	d		m	\0	?

Scanning a Full Line

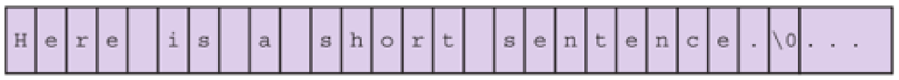
- For interactive input of one complete line of data, use the gets function.
- The \n character representing the <return> or <enter> key pressed at the end of the line is not stored.

Scanning a Full Line



```
char line[80];
printf("Type in a line of data.\n> ");
gets(line);
```

```
Type in a line of data.
> Here is a short sentence.
```



```
#include<stdio.h>
#include<string.h>
int main()
{
    char line[80];
    int i, cnt = 0;
    printf("Enter a line of data... \n");
    gets(line);
    printf("%s\n", line);
    for(i=0;i<sizeof(line);i++)
       if(line[i] == ' ')
          cnt++;
    printf("There are %d spaces in the line.\n", cnt);
}
```

There is also a version of gets for files called `fgets`. The following reads a text file called MyText.txt and prints it to the console using fgets.

```
#define LINELEN 80
#include<stdio.h>
int main()
{
    char line[LINELEN];
    FILE *inp;  //Declare pointers to in
    int err;
    char *status;
    //Open the file for input
    err = fopen_s(&inp, "MyText.txt", "r");
    status = fgets(line, LINELEN, inp);
     while(status != 0)
     {
        printf("%s", line);
       status = fgets(line, LINELEN, inp);
     }
    fclose(inp);
}
```

String Comparison

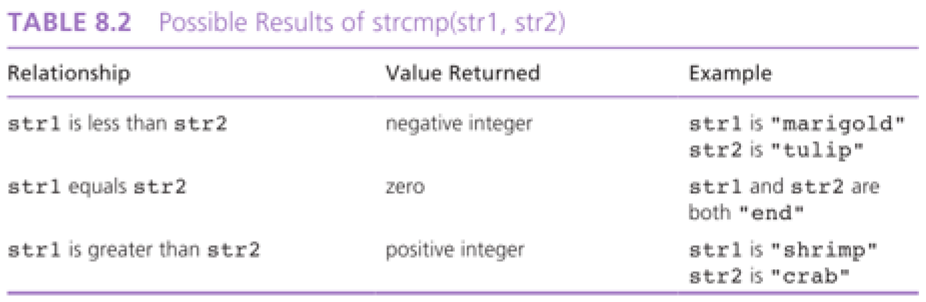
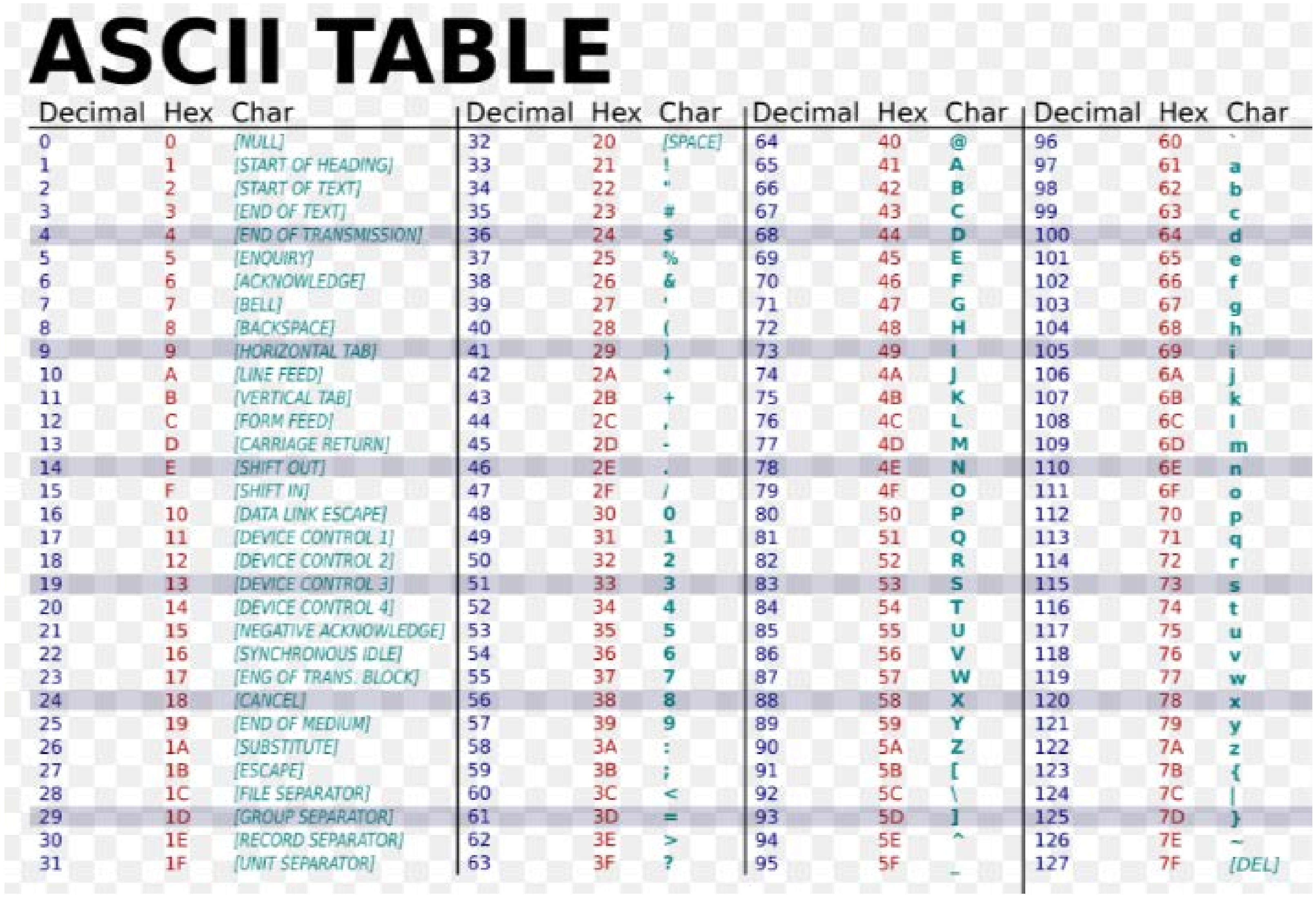


TABLE 8.2 Possible Results of strcmp(str1, str2)

Relationship	Value Returned	Example
`str1` is less than `str2`	negative integer	`str1` is `"marigold"` `str2` is `"tulip"`
`str1` equals `str2`	zero	`str1` and `str2` are both `"end"`
`str1` is greater than `str2`	positive integer	`str1` is `"shrimp"` `str2` is `"crab"`



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

```c
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[] = "abc";
    char s2[] = "bcd";
    char s3[] = "abcd";
    char s4[] = "ABC";
    char s5[] = "123";
    printf("%s to %s = %d\n", s1, s2, strcmp(s1, s2));
    printf("%s to %s = %d\n", s1, s3, strcmp(s1, s3));
    printf("%s to %s = %d\n", s1, s4, strcmp(s1, s4));
    printf("%s to %s = %d\n", s1, s1, strcmp(s1, s1));
    printf("%s to %s = %d\n", s1, s5, strcmp(s1, s5));

}
/*
abc to bcd = -1
abc to abcd = -1
abc to ABC = 1
abc to abc = 0
abc to 123 = 1
Press any key to continue . . .
*/
```

Since strings are char arrays and string names are pointers to those arrays, if we have an array of strings we have an array of pointers where each pointer points to a string.

Suppose we have five color names and we put them in an array which is necessarily two-dimensional.

```c
char color[5][10] = {"red", "orange", "yellow", "green", "blue"};
```

In memory this looks like this:

Address		0	1	2	3	4	5	6	7	8	9
100	0	r	e	d							
110	1	o	r	a	n	g	e				
120	2	y	e	l	l	o	w				
130	3	g	r	e	e	n					
140	5	b	l	u	e						

The addresses are the pointer values of each string: color[0] = 100, color[1] = 110, etc.

If we want to sort this array of colors into alphabetical order we can do it the traditional way using strcpy_s to move the strings around. The following program uses a Bubble sort

```
/*This program sorts an array of strings by sorting
    in the traditional manner using strcpy_s to
    swap strings in a Bubble sort
*/
void BubbleSort(char color[][STRSIZE]);
int main()
{
   char color[STRNUM][STRSIZE] = {"red", "orange", "yellow", "green", "blue"};
   int i;
   for(i=0;i<STRNUM;i++)
      printf("%s\n", color[i]);
   BubbleSort(color);
   printf("Sorted\n");
   for(i=0;i<STRNUM;i++)
      printf("%s\n", color[i]);
}
void BubbleSort(char color[][STRSIZE])
{
    int i, fDone;
    char tmp[STRSIZE];
    fDone = 0;
    while(!fDone)
        {fDone = 1;
         for(i=0;i<STRNUM-1;i++)
             {if(strcmp(color[i],color[i+1]) == 1)
                 {strcpy_s(tmp, STRSIZE, color[i]);
                  strcpy_s(color[i], STRSIZE, color[i+1]);
                  strcpy_s(color[i+1], STRSIZE, tmp);
                  fDone = 0;
                 }
             }
         }
     }
```

The lines:

```
              if(strcmp(color[i],color[i+1]) == 1)
                 {strcpy_s(tmp, STRSIZE, color[i]);
                  strcpy_s(color[i], STRSIZE, color[i+1]);
                  strcpy_s(color[i+1], STRSIZE, tmp);
                  fDone = 0;
                 }
```

do the swap of two strings using `strcpy_s` when two adjacent strings are out of order.

If we run this program the string array will be rearranged to look like this:

Address		0	1	2	3	4	5	6	7	8	9
100	0	b	l	u	e						
110	1	g	r	e	e	n					
120	2	o	r	a	n	g	e				
130	3	r	e	d							
140	5	y	e	l	l	o	w				

Note that in each case the entire color string has been moved to a new address. The color "blue" was at 140 and is now at 100 etc.

Alternatively, we need not move the strings. We could just move the pointers to the strings – that is, create a set of pointers and rearrange the pointers so that they are in order.

We can create an array of pointers and put these addresses (pointers) into the array like this:

```
char color[5][10] = {"red", "orange", "yellow", "green", "blue"};
char *colorPtr[5];
int i;
for(i=0;i<5;i++)
   colorPtr[i] = color[i];
```

In memory we have:

colorPtr[]

Index	Address	Value	Points to
0	550	100	-> red
1	551	110	-> orange
2	552	120	->yellow
3	553	130	->green
4	554	140	->blue

We can rearrange the pointers so that if the pointers are printed by index they point to the colors in the right order. Here is the program.

```
/*This program sorts an array of strings by sorting
    the pointers to those strings. It uses a bubble
    sort.
*/
void BubbleSort(char *cPtr[]);
int main()
{
   char color[STRNUM][STRSIZE] = {"red", "orange", "yellow", "green", "blue"};
   char *colorPtr[STRNUM];
   int i;
   for(i=0;i<STRNUM;i++)
      colorPtr[i] = color[i];
   for(i=0;i<STRNUM;i++)
      printf("%d => %s\n", colorPtr[i], colorPtr[i]);
   BubbleSort(colorPtr);
   printf("Sorted\n");
   for(i=0;i<STRNUM;i++)
      printf("%d => %s\n", colorPtr[i], colorPtr[i]);
}
void BubbleSort(char *cPtr[])
{
    int i, fDone;
    char *tmp;
    fDone = 0;
    while(!fDone)
        {fDone = 1;
         for(i=0;i<STRNUM-1;i++)
             {if(strcmp(cPtr[i],cPtr[i+1]) == 1)
                 {tmp = cPtr[i];
                  cPtr[i] = cPtr[i+1];
                  cPtr[i+1] = tmp;
                  fDone = 0;
                 }
             }
         }
     }
```

Note that the swap given by:

```
{if(strcmp(cPtr[i],cPtr[i+1]) == 1)
     {tmp = cPtr[i];
      cPtr[i] = cPtr[i+1];
      cPtr[i+1] = tmp;
      fDone = 0;
     }
```

is swapping pointers – not strings. After this program runs we have:

In memory we have:
`colorPtr[]`

Index	Address	Value	Points to
0	550	140	->blue
1	551	130	->green
2	552	110	->orange
3	553	120	->yellow
4	554	100	->red

Now when we print the data by pointer index it is sorted but the data has not moved – only the pointers have moved in the pointer array.

Character Input/Output

- getchar
  - get the next character from the standard input source (that scanf uses)
  - does not expect the calling module to pass the address of a variable to store the input character
  - takes no arguments, returns the character as its result

```
ch = getchar()
```

Character Input/Output

- getc
  - used to get a single character from a file
  - comparable to getchar except that the character returned is obtained from the file accessed by a file pointer (ex., inp)

```
getc(inp)
```

Character Input/Output

- putchar
  - single-character output
  - first argument is a type int character code
  - recall that type char can always be converted to type in with no loss of information

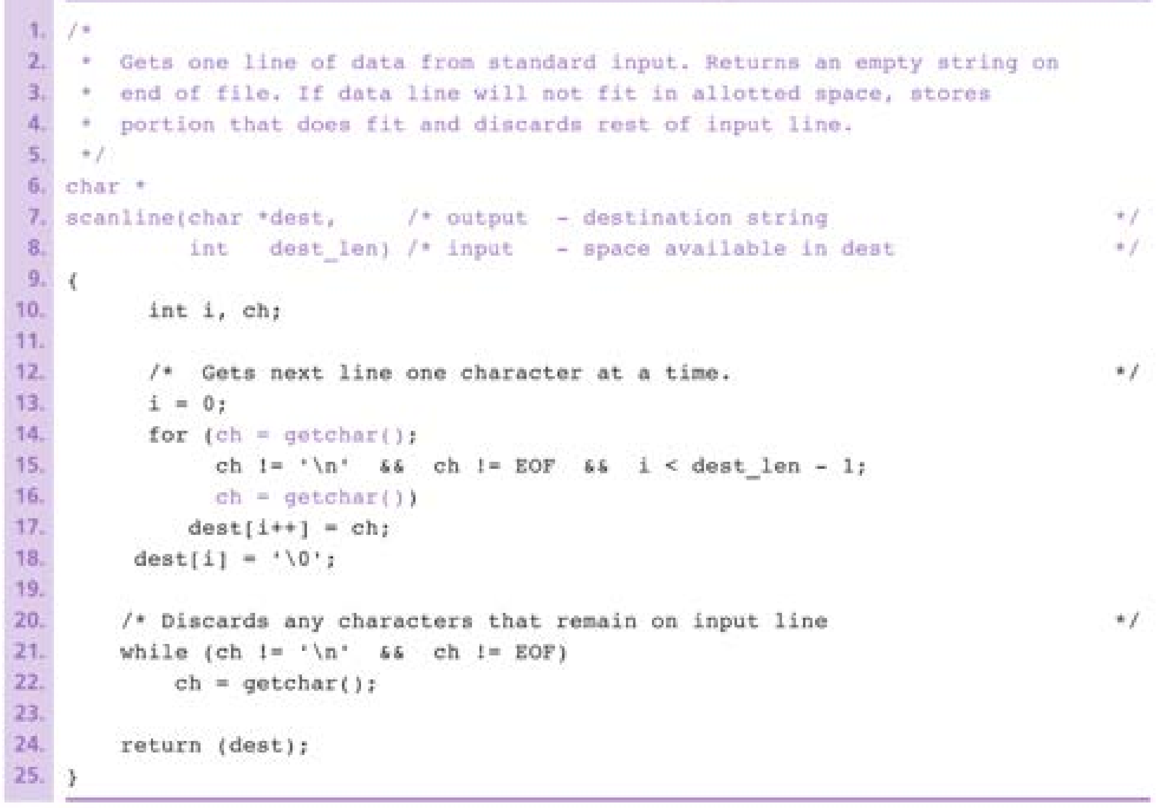
```
putchar('a');
```

Character Input/Output

- putc
  - identical to putchar except it sends the single character/int to a file, ex., outp

putc('a', outp);

FIGURE 8.15 Implementation of scanline Function Using getchar



```c
/*
 *  Gets one line of data from standard input. Returns an empty string on
 *  end of file. If data line will not fit in allotted space, stores
 *  portion that does fit and discards rest of input line.
 */
char *
scanline(char *dest,     /* output  - destination string                 */
         int   dest_len) /* input   - space available in dest            */
{
      int i, ch;

      /*  Gets next line one character at a time.                        */
      i = 0;
      for (ch = getchar();
           ch != '\n'  &&  ch != EOF  &&  i < dest_len - 1;
           ch = getchar())
         dest[i++] = ch;
     dest[i] = '\0';

    /* Discards any characters that remain on input line                 */
    while (ch != '\n'  &&  ch != EOF)
        ch = getchar();

    return (dest);
}
```

These character functions need `#include<ctype.h>`

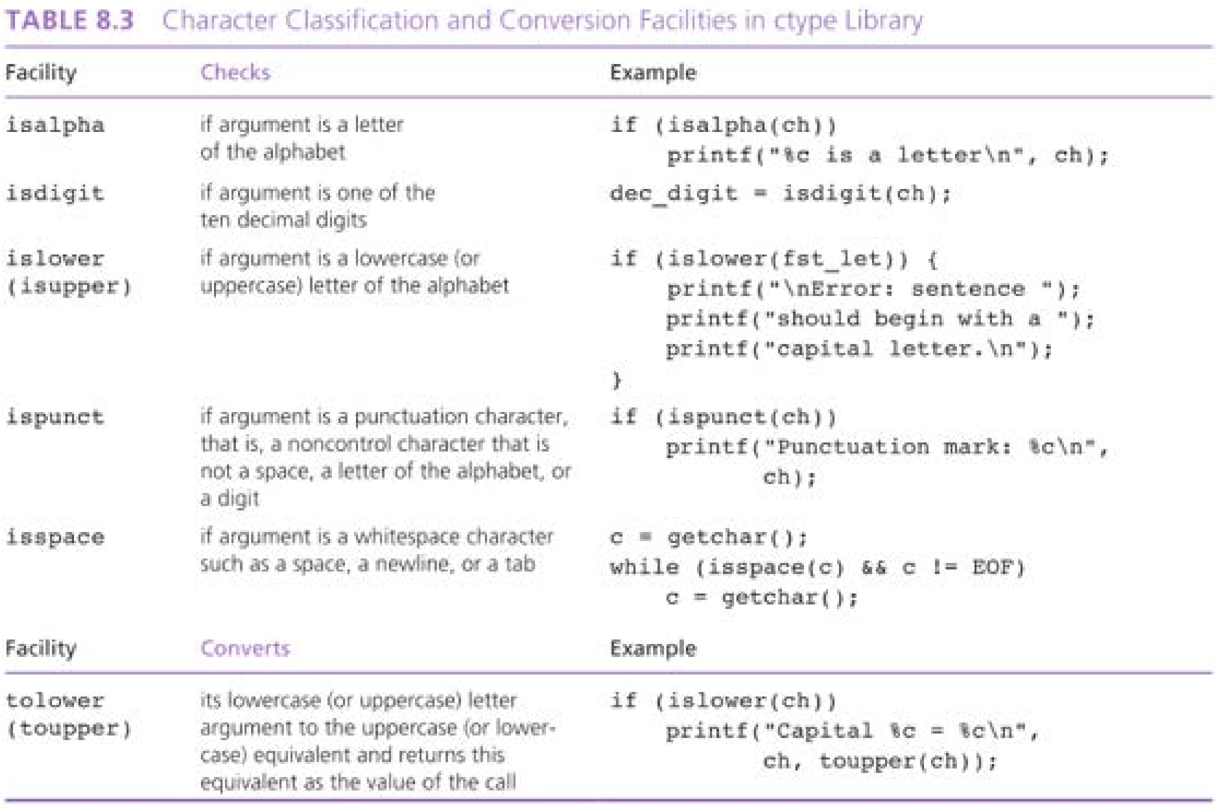


TABLE 8.3 Character Classification and Conversion Facilities in ctype Library

Facility	Checks	Example

`isalpha`	if argument is a letter of the alphabet	`if (isalpha(ch))`
		`    printf("%c is a letter\n", ch);`

`isdigit`	if argument is one of the ten decimal digits	`dec_digit = isdigit(ch);`

`islower` `(isupper)`	if argument is a lowercase (or uppercase) letter of the alphabet	`if (islower(fst_let)) {`
		`    printf("\nError: sentence ");`
		`    printf("should begin with a ");`
		`    printf("capital letter.\n");`
		`}`

`ispunct`	if argument is a punctuation character, that is, a noncontrol character that is not a space, a letter of the alphabet, or a digit	`if (ispunct(ch))`
		`    printf("Punctuation mark: %c\n",`
		`           ch);`

`isspace`	if argument is a whitespace character such as a space, a newline, or a tab	`c = getchar();`
		`while (isspace(c) && c != EOF)`
		`    c = getchar();`

Facility	Converts	Example

`tolower` `(toupper)`	its lowercase (or uppercase) letter argument to the uppercase (or lowercase) equivalent and returns this equivalent as the value of the call	`if (islower(ch))`
		`    printf("Capital %c = %c\n",`
		`           ch, toupper(ch));`

These string functions need `#include<string.h>`

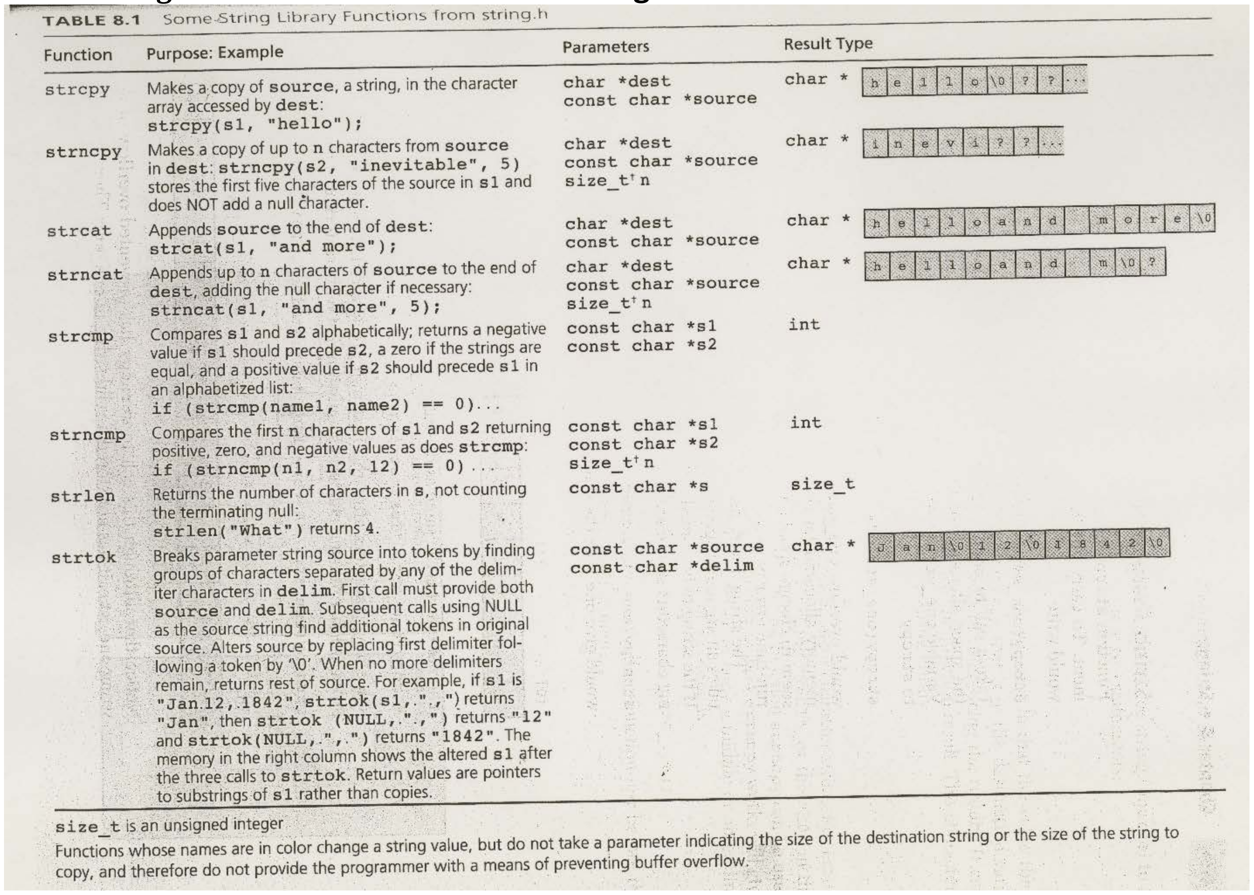


TABLE 8.1 Some String Library Functions from string.h

Function	Purpose: Example	Parameters	Result Type

`strcpy`	Makes a copy of `source`, a string, in the character array accessed by `dest`: `strcpy(s1, "hello");`	`char *dest`
		`const char *source`
			`char *` `h e l l o \0 ? ? ...`

`strncpy`	Makes a copy of up to `n` characters from `source` in `dest`: `strncpy(s2, "inevitable", 5)` stores the first five characters of the source in `s1` and does NOT add a null character.	`char *dest`
		`const char *source`
		`size_t`† `n`
			`char *` `i n e v i ? ? ...`

`strcat`	Appends `source` to the end of `dest`: `strcat(s1, "and more");`	`char *dest`
		`const char *source`
			`char *` `h e l l o a n d   m o r e \0`

`strncat`	Appends up to `n` characters of `source` to the end of `dest`, adding the null character if necessary: `strncat(s1, "and more", 5);`	`char *dest`
		`const char *source`
		`size_t`† `n`
			`char *` `h e l l o a n d   m \0 ?`

`strcmp`	Compares `s1` and `s2` alphabetically; returns a negative value if `s1` should precede `s2`, a zero if the strings are equal, and a positive value if `s2` should precede `s1` in an alphabetized list: `if (strcmp(name1, name2) == 0)...`	`const char *s1`
		`const char *s2`
			`int`

`strncmp`	Compares the first `n` characters of `s1` and `s2` returning positive, zero, and negative values as does `strcmp`: `if (strncmp(n1, n2, 12) == 0) ...`	`const char *s1`
		`const char *s2`
		`size_t`† `n`
			`int`

`strlen`	Returns the number of characters in `s`, not counting the terminating null: `strlen("What")` returns `4`.	`const char *s`	`size_t`

`strtok`	Breaks parameter string source into tokens by finding groups of characters separated by any of the delimiter characters in `delim`. First call must provide both `source` and `delim`. Subsequent calls using NULL as the source string find additional tokens in original source. Alters source by replacing first delimiter following a token by '\0'. When no more delimiters remain, returns rest of source. For example, if `s1` is `"Jan.12,.1842"`, `strtok(s1,".,")` returns `"Jan"`, then `strtok (NULL,".,")` returns `"12"` and `strtok(NULL,".,")` returns `"1842"`. The memory in the right column shows the altered `s1` after the three calls to `strtok`. Return values are pointers to substrings of `s1` rather than copies.	`const char *source`
		`const char *delim`
			`char *` `J a n \0 1 2 \0 1 8 4 2 \0`

† `size_t` is an unsigned integer

Functions whose names are in color change a string value, but do not take a parameter indicating the size of the destination string or the size of the string to copy, and therefore do not provide the programmer with a means of preventing buffer overflow.

CS 210 **October 25, 2016**
Strings

A palindrome is a word or a sentence which reads the same forward or backward. For example, *mom* is a palindrome as are all of the following (from Wikipedia):
redivider, noon, civic, radar, level, rotor, kayak, reviver, racecar, redder, madam, and refer.

Write a program to input a sentence on one line from the user and determine if the sentence is a palindrome. Your program should print whether or not the input sentence is a palindrome.

Turn in a hard copy of your source code.