Programming for Engineers
Lecture 9: Characters and Strings

Course ID: EE057IU

Instructor: Dr. Hieu Nguyen
Assistant Professor, School of Electrical Engineering
International University – VNU-HCM
Ho Chi Minh City, Fall 2024

Email: nthieu@hcmiu.edu.vn
Office Hours: Wed 1:00 PM – 4:00 PM and Thu 8:30 AM – 11:00 AM

Outline

- Fundamentals of Strings and Characters
- Character-Handling Library (ctype.h)
- String Conversion Functions
- Standard Input/output Functions (stdio.h)
- String Manipulation Function (string.h)

Fundamentals of Strings and Characters

- **Characters**: Building blocks of programs, represented as int values in single quotes.
  - Example: 'z', '\n’

- **Strings**: Series of characters treated as a single unit, enclosed in double quotes.
  - Example: "John Q. Doe“

- **Character Constants**: Each character has an integer value in the system's character set.

String Characteristics

- ❑ Strings are **null-terminated** ('\0') to indicate the end.
- ❑ **Undefined Behavior**: Missing a null terminator may crash programs.
- ❑ Strings are accessed through **pointers** pointing to the first character.
- ❑ Strings can be stored as:
  - **Character Arrays**: Modifiable.
  - **String Literals**: Immutable.

- ❑ **Initializing Strings:**
  - Character Arrays: `char color[] = “blue”;`
  - String Pointer `const char *colorPtr = “blue”;`
  - Array with Explicit Null Character: `char color[] = {‘b’, ‘l’, ‘u’, ‘e’, ‘\0’};`

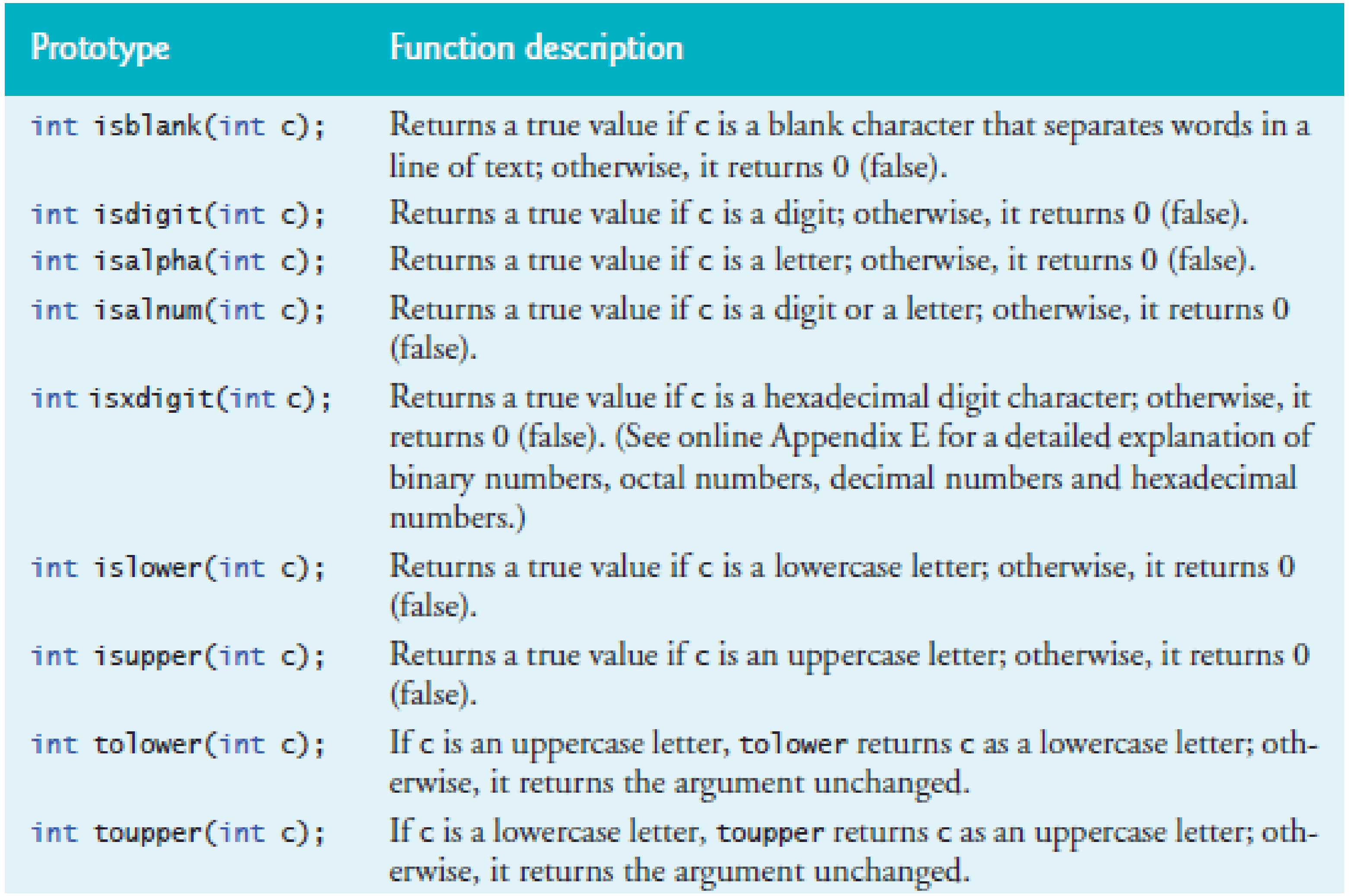
Reading String with scanf

- Function scanf will read characters until a space, tab, newline or end-of-file indicator is encountered.

```
char word[20];
scanf("%19s", word);
```

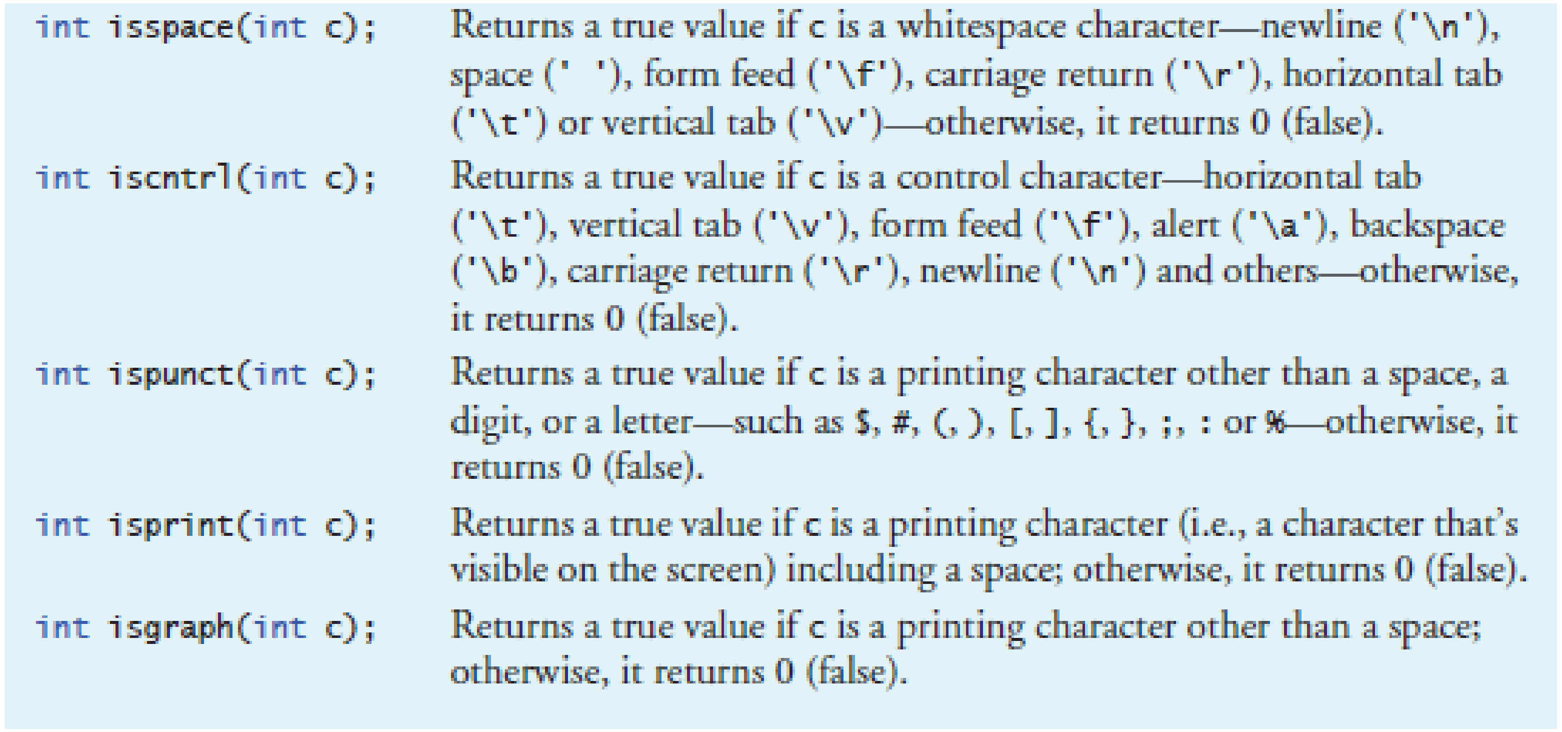
- Ensures safe input by limiting the characters to 19 (leaving space for '\0').
- Prevents overwriting memory or crashing due to excessive input.

Character-Handling Library (ctype.h)



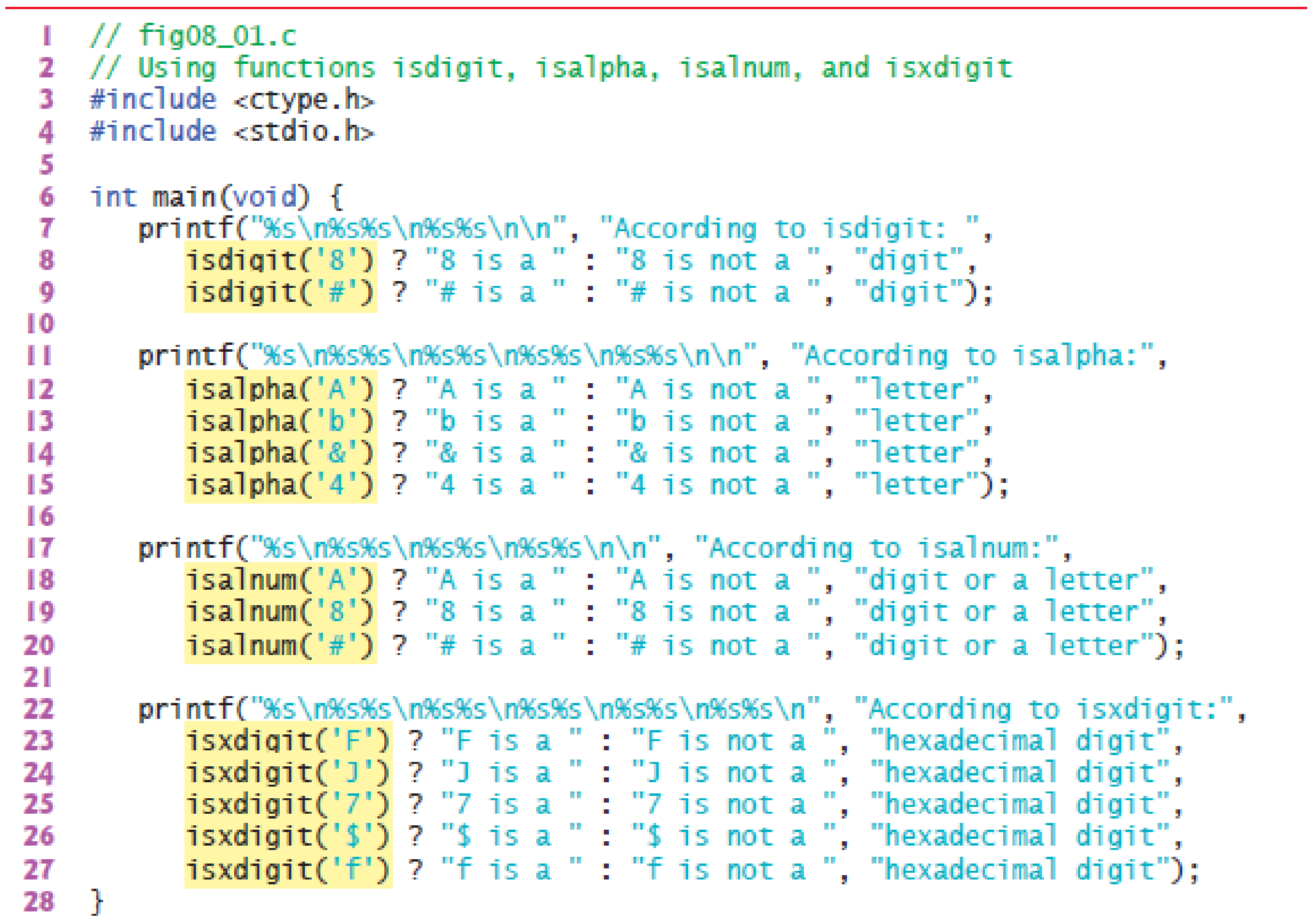
Prototype	Function description
`int isblank(int c);`	Returns a true value if `c` is a blank character that separates words in a line of text; otherwise, it returns 0 (false).
`int isdigit(int c);`	Returns a true value if `c` is a digit; otherwise, it returns 0 (false).
`int isalpha(int c);`	Returns a true value if `c` is a letter; otherwise, it returns 0 (false).
`int isalnum(int c);`	Returns a true value if `c` is a digit or a letter; otherwise, it returns 0 (false).
`int isxdigit(int c);`	Returns a true value if `c` is a hexadecimal digit character; otherwise, it returns 0 (false). (See online Appendix E for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower(int c);`	Returns a true value if `c` is a lowercase letter; otherwise, it returns 0 (false).
`int isupper(int c);`	Returns a true value if `c` is an uppercase letter; otherwise, it returns 0 (false).
`int tolower(int c);`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter; otherwise, it returns the argument unchanged.
`int toupper(int c);`	If `c` is a lowercase letter, `toupper` returns `c` as an uppercase letter; otherwise, it returns the argument unchanged.

Character-Handling Library (ctype.h)



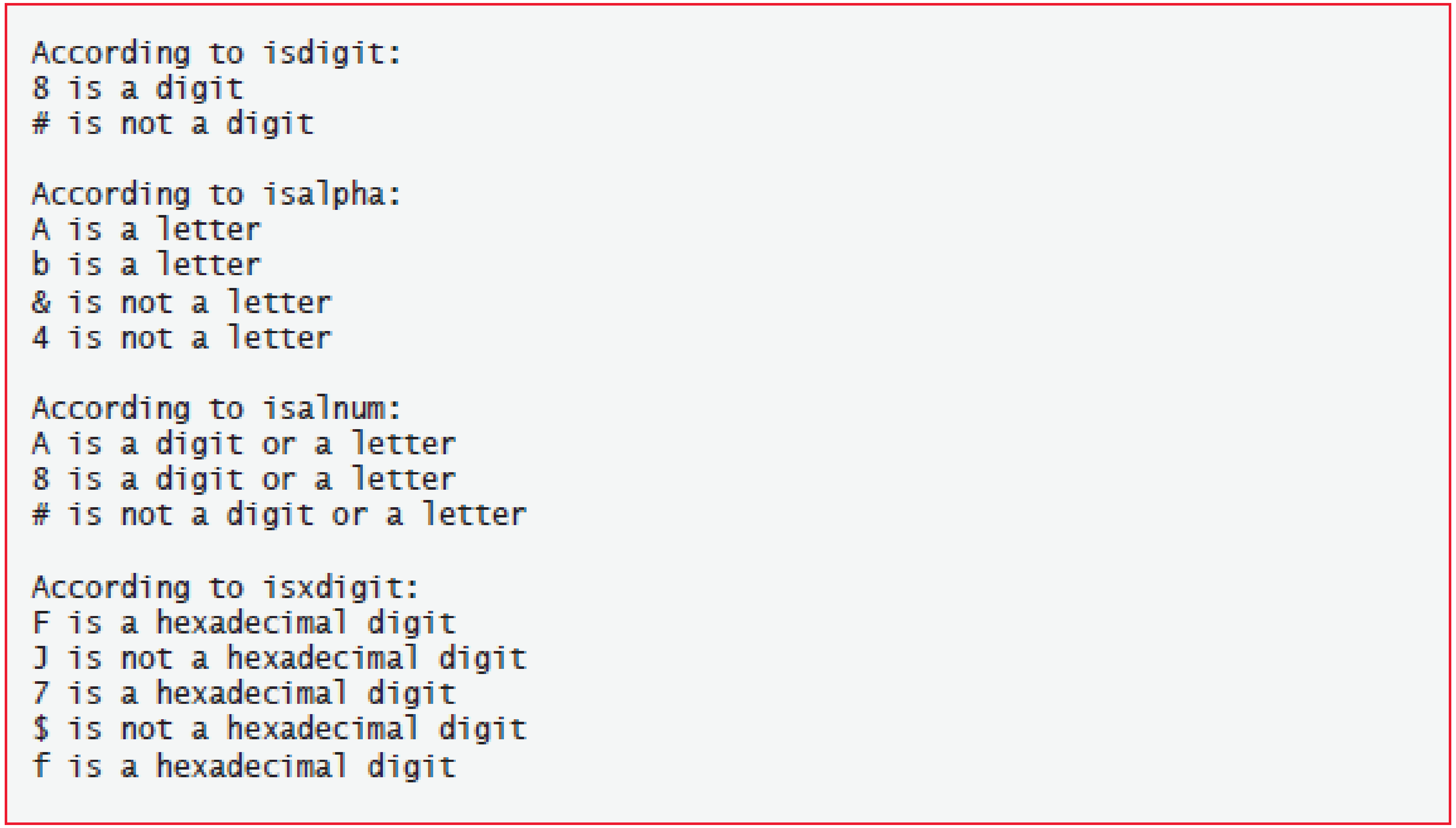
`int isspace(int c);`	Returns a true value if `c` is a whitespace character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—otherwise, it returns 0 (false).
`int iscntrl(int c);`	Returns a true value if `c` is a control character—horizontal tab (`'\t'`), vertical tab (`'\v'`), form feed (`'\f'`), alert (`'\a'`), backspace (`'\b'`), carriage return (`'\r'`), newline (`'\n'`) and others—otherwise, it returns 0 (false).
`int ispunct(int c);`	Returns a true value if `c` is a printing character other than a space, a digit, or a letter—such as `$`, `#`, `(`, `)`, `[`, `]`, `{`, `}`, `;`, `:` or `%`—otherwise, it returns 0 (false).
`int isprint(int c);`	Returns a true value if `c` is a printing character (i.e., a character that's visible on the screen) including a space; otherwise, it returns 0 (false).
`int isgraph(int c);`	Returns a true value if `c` is a printing character other than a space; otherwise, it returns 0 (false).

In-class example



```c
// fig08_01.c
// Using functions isdigit, isalpha, isalnum, and isxdigit
#include <ctype.h>
#include <stdio.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%s%s\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");
}
```

In-class example



```
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```

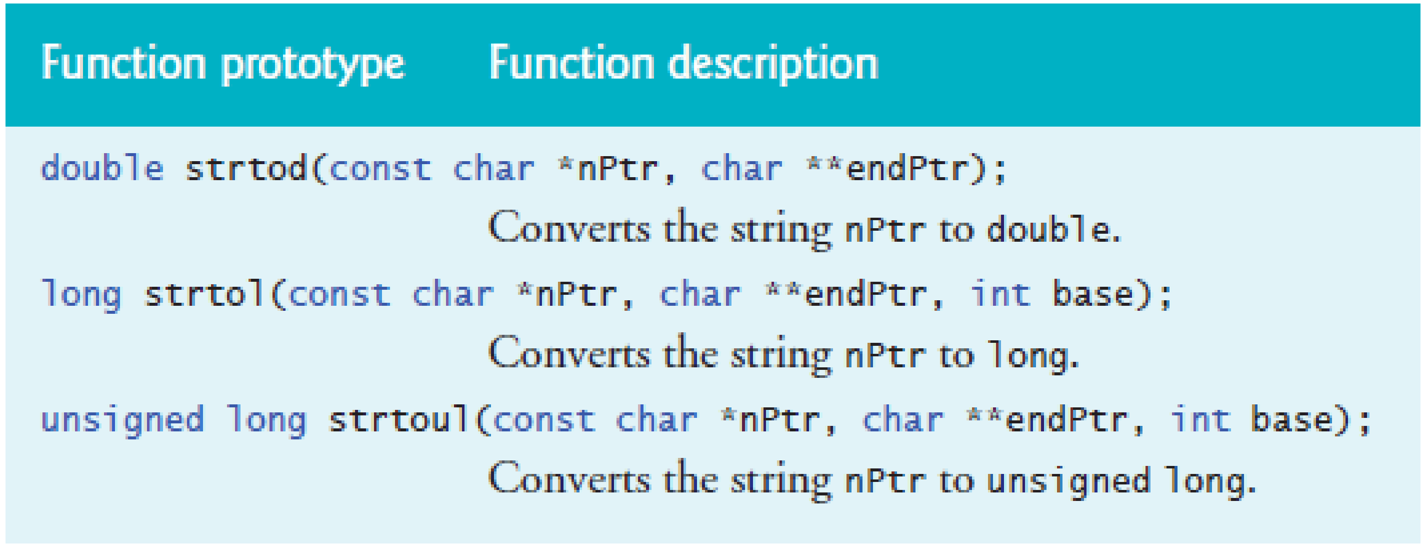
Fig. 8.1 | Using functions `isdigit`, `isalpha`, `isalnum` and `isxdigit`.

In-class example

```c
// fig08_02.c
// Using functions islower, isupper, tolower and toupper
#include <ctype.h>
#include <stdio.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'));
}
```

```
According to islower:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter
```

String-Conversion Functions



Function prototype	Function description
`double strtod(const char *nPtr, char **endPtr);`	Converts the string `nPtr` to `double`.
`long strtol(const char *nPtr, char **endPtr, int base);`	Converts the string `nPtr` to `long`.
`unsigned long strtoul(const char *nPtr, char **endPtr, int base);`	Converts the string `nPtr` to `unsigned long`.

Function $strtod()$

- Converts a string (char *) to a double value
- Takes two arguments:
  1. The input string (char *)
  2. A pointer to a string (char **) for tracking the remaining unprocessed part
- Returns 0 if the conversion fails

Function $strtod()$

```c
// fig08_04.c
// Using function strtod
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   const char *string = "51.2% are admitted";
   char *stringPtr = NULL;

   double d = strtod(string, &stringPtr);

   printf("The string \"%s\" is converted to the\n", string);
   printf("double value %.2f and the string \"%s\"\n", d, stringPtr);
}
```

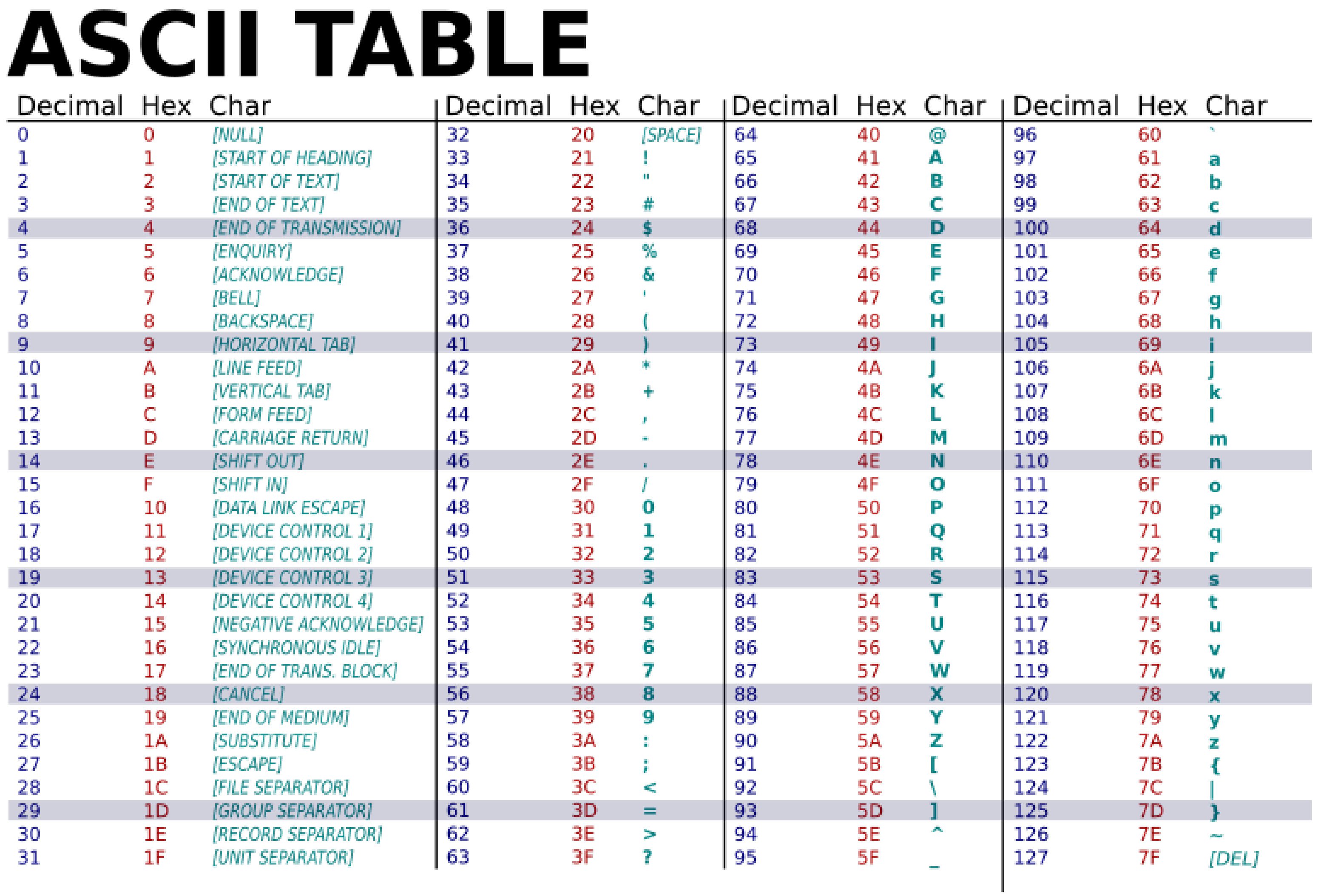
```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

Function *strtol*()

```c
// fig08_05.c
// Using function strtol
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   const char *string = "-1234567abc";
   char *remainderPtr = NULL;

   long x = strtol(string, &remainderPtr, 0);

   printf("%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
      "The original string is ", string,
      "The converted value is ", x,
      "The remainder of the original string is ", remainderPtr,
      "The converted value plus 567 is ", x + 567);
}
```

```
The original string is "-1234567abc"
The converted value is -1234567
The remainder of the original string is "abc"
The converted value plus 567 is -1234000
```

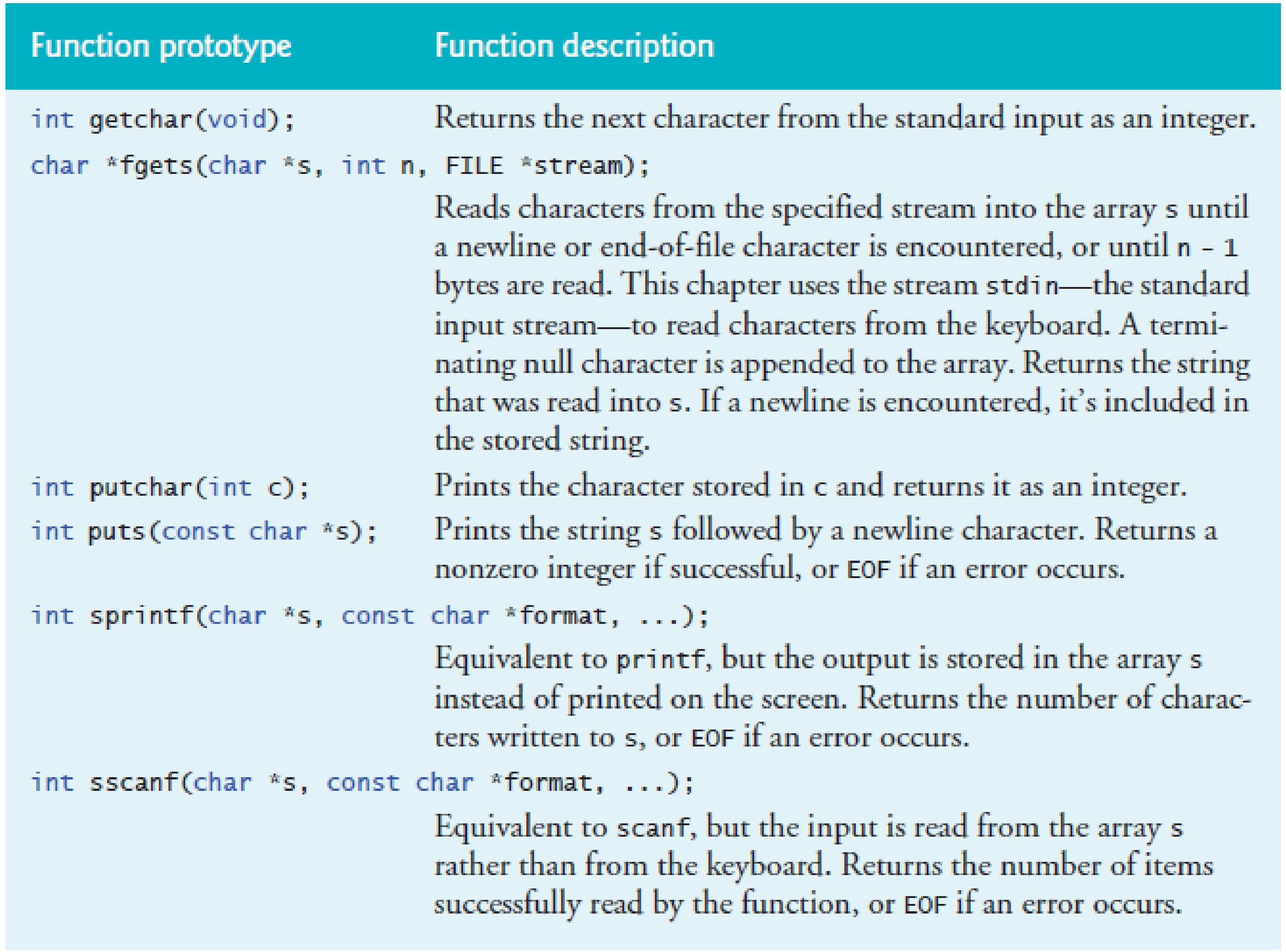
ASCII Character Set



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	[END 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]

Standard Input/Output Library Function (stdio.h)



Function prototype	Function description
`int getchar(void);`	Returns the next character from the standard input as an integer.
`char *fgets(char *s, int n, FILE *stream);`	Reads characters from the specified stream into the array `s` until a newline or end-of-file character is encountered, or until `n - 1` bytes are read. This chapter uses the stream `stdin`—the standard input stream—to read characters from the keyboard. A terminating null character is appended to the array. Returns the string that was read into `s`. If a newline is encountered, it's included in the stored string.
`int putchar(int c);`	Prints the character stored in `c` and returns it as an integer.
`int puts(const char *s);`	Prints the string `s` followed by a newline character. Returns a nonzero integer if successful, or `EOF` if an error occurs.
`int sprintf(char *s, const char *format, ...);`	Equivalent to `printf`, but the output is stored in the array `s` instead of printed on the screen. Returns the number of characters written to `s`, or `EOF` if an error occurs.
`int sscanf(char *s, const char *format, ...);`	Equivalent to `scanf`, but the input is read from the array `s` rather than from the keyboard. Returns the number of items successfully read by the function, or `EOF` if an error occurs.

Function *sprintf*

- Function ***sprintf*** prints formatted data into char array s.

```c
// fig08_09.c
// Using function sprintf
#include <stdio.h>
#define SIZE 80

int main(void) {
   int x = 0;
   double y = 0.0;

   puts("Enter an integer and a double:");
   scanf("%d%lf", &x, &y);

   char s[SIZE] = {'\0'}; // create char array
   sprintf(s, "integer:%6d\ndouble:%7.2f", x, y);

   printf("The formatted output stored in array s is:\n%s\n", s);
}
```

```
Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer:   298
double:  87.38
```

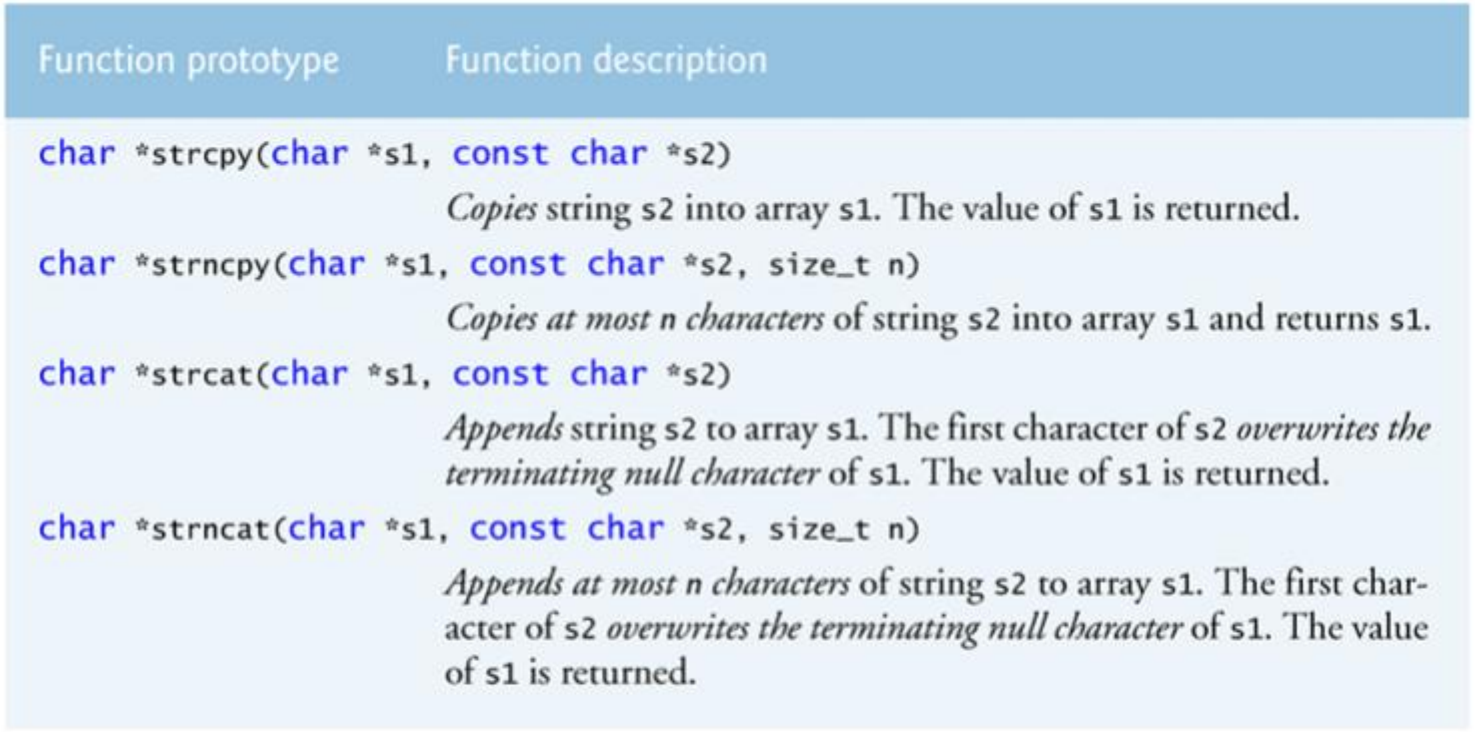
Function *sscanf*

- *sscanf* work like *scanf* but reads formatted data from a string

```c
// fig08_10.c
// Using function sscanf
#include <stdio.h>

int main(void) {
   char s[] = "31298 87.375";
   int x = 0;
   double y = 0;

   sscanf(s, "%d%lf", &x, &y);
   puts("The values stored in character array s are:");
   printf("integer:%6d\ndouble:%8.3f\n", x, y);
}
```

```
The values stored in character array s are:
integer: 31298
double:   87.375
```

String Manipulation in <string.h>



Function prototype	Function description
`char *strcpy(char *s1, const char *s2)`	*Copies* string `s2` into array `s1`. The value of `s1` is returned.
`char *strncpy(char *s1, const char *s2, size_t n)`	*Copies at most `n` characters* of string `s2` into array `s1` and returns `s1`.
`char *strcat(char *s1, const char *s2)`	*Appends* string `s2` to array `s1`. The first character of `s2` *overwrites the terminating null character* of `s1`. The value of `s1` is returned.
`char *strncat(char *s1, const char *s2, size_t n)`	*Appends at most `n` characters* of string `s2` to array `s1`. The first character of `s2` *overwrites the terminating null character* of `s1`. The value of `s1` is returned.

Function *strcpy* and *strncpy*

```c
// fig08_11.c
// Using functions strcpy and strncpy
#include <stdio.h>
#include <string.h>
#define SIZE1 25
#define SIZE2 15

int main(void) {
   char x[] = "Happy Birthday to You"; // initialize char array x
   char y[SIZE1] = ""; // create char array y
   char z[SIZE2] = ""; // create char array z

   // copy contents of x into y
   printf("%s%s\n%s%s\n",
      "The string in array x is: ", x,
      "The string in array y is: ", strcpy(y, x));

   strncpy(z, x, SIZE2 - 1); // copy first 14 characters of x into z
   z[SIZE2 - 1] = '\0'; // terminate string in z, because '\0' not copied
   printf("The string in array z is: %s\n", z);
}
```

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

Function *strcay* and *strncat*

```c
// fig08_12.c
// Using functions strcat and strncat
#include <stdio.h>
#include <string.h>

int main(void) {
   char s1[20] = "Happy "; // initialize char array s1
   char s2[] = "New Year "; // initialize char array s2
   char s3[40] = ""; // initialize char array s3 to empty

   printf("s1 = %s\ns2 = %s\n", s1, s2);

   // concatenate s2 to s1
   printf("strcat(s1, s2) = %s\n", strcat(s1, s2));

   // concatenate first 6 characters of s1 to s3
   printf("strncat(s3, s1, 6) = %s\n", strncat(s3, s1, 6));

   // concatenate s1 to s3
   printf("strcat(s3, s1) = %s\n", strcat(s3, s1));
}
```

```
s1 = Happy
s2 = New Year
strcat(s1, s2) = Happy New Year
strncat(s3, s1, 6) = Happy
strcat(s3, s1) = Happy Happy New Year
```

String Compare

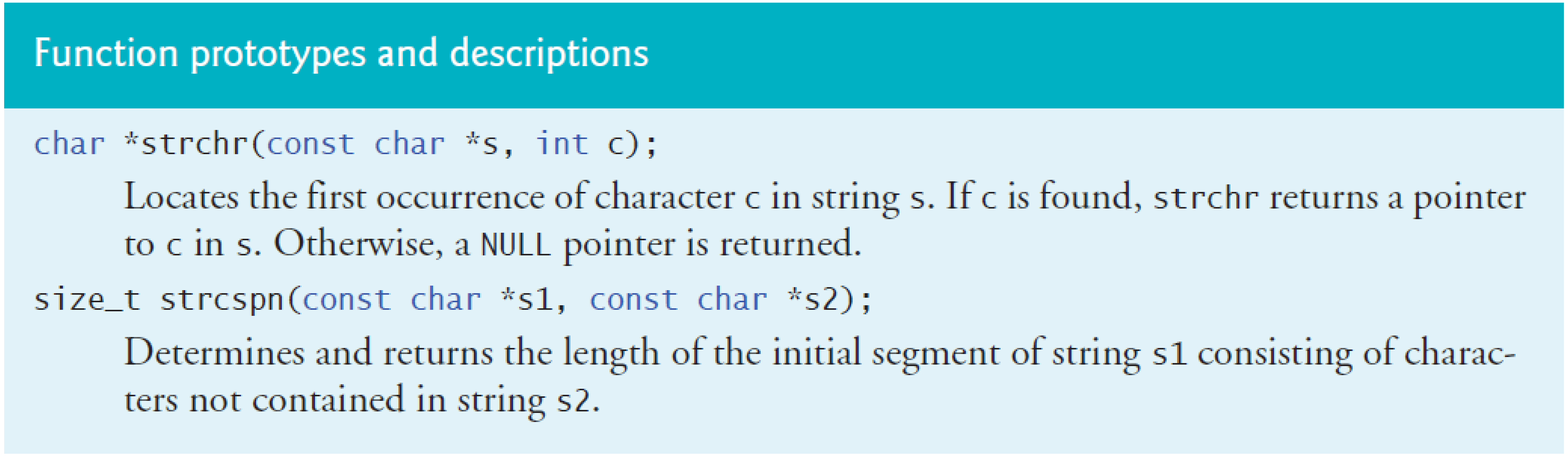
Function prototype	Function description
`int strcmp(const char *s1, const char *s2);`	*Compares* the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.
`int strncmp(const char *s1, const char *s2, size_t n);`	*Compares up to* `n` *characters* of the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.

String Compare

```c
// fig08_13.c
// Using functions strcmp and strncmp
#include <stdio.h>
#include <string.h>

int main(void) {
   const char *s1 = "Happy New Year"; // initialize char pointer
   const char *s2 = "Happy New Year"; // initialize char pointer
   const char *s3 = "Happy Holidays"; // initialize char pointer

   printf("s1 = %s\ns2 = %s\ns3 = %s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
      s1, s2, s3,
      "strcmp(s1, s2) = ", strcmp(s1, s2),
      "strcmp(s1, s3) = ", strcmp(s1, s3),
      "strcmp(s3, s1) = ", strcmp(s3, s1));

   printf("%s%2d\n%s%2d\n%s%2d\n",
      "strncmp(s1, s3, 6) = ", strncmp(s1, s3, 6),
      "strncmp(s1, s3, 7) = ", strncmp(s1, s3, 7),
      "strncmp(s3, s1, 7) = ", strncmp(s3, s1, 7));
}
```

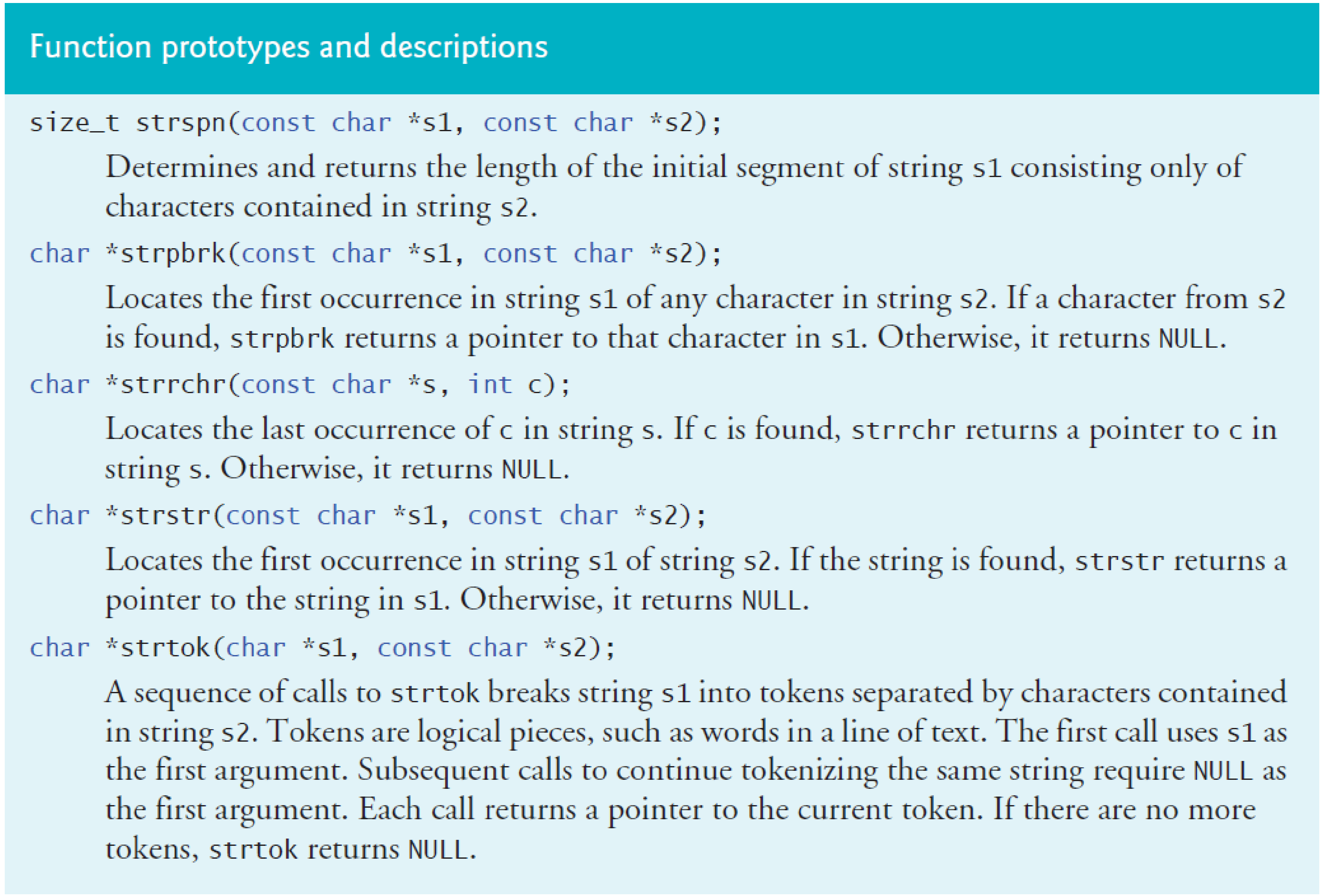
```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

Search Functions (Part I)



Function prototypes and descriptions
`char *strchr(const char *s, int c);`
Locates the first occurrence of character `c` in string `s`. If `c` is found, `strchr` returns a pointer to `c` in `s`. Otherwise, a `NULL` pointer is returned.
`size_t strcspn(const char *s1, const char *s2);`
Determines and returns the length of the initial segment of string `s1` consisting of characters not contained in string `s2`.

Search Functions (Part II)



Function prototypes and descriptions

`size_t strspn(const char *s1, const char *s2);`

Determines and returns the length of the initial segment of string `s1` consisting only of characters contained in string `s2`.

`char *strpbrk(const char *s1, const char *s2);`

Locates the first occurrence in string `s1` of any character in string `s2`. If a character from `s2` is found, `strpbrk` returns a pointer to that character in `s1`. Otherwise, it returns `NULL`.

`char *strrchr(const char *s, int c);`

Locates the last occurrence of `c` in string `s`. If `c` is found, `strrchr` returns a pointer to `c` in string `s`. Otherwise, it returns `NULL`.

`char *strstr(const char *s1, const char *s2);`

Locates the first occurrence in string `s1` of string `s2`. If the string is found, `strstr` returns a pointer to the string in `s1`. Otherwise, it returns `NULL`.

`char *strtok(char *s1, const char *s2);`

A sequence of calls to `strtok` breaks string `s1` into tokens separated by characters contained in string `s2`. Tokens are logical pieces, such as words in a line of text. The first call uses `s1` as the first argument. Subsequent calls to continue tokenizing the same string require `NULL` as the first argument. Each call returns a pointer to the current token. If there are no more tokens, `strtok` returns `NULL`.

Function *strchr*

```c
// fig08_14.c
// Using function strchr
#include <stdio.h>
#include <string.h>

int main(void) {
   const char *string = "This is a test"; // initialize char pointer
   char character1 = 'a';
   char character2 = 'z';

   // if character1 was found in string
   if (strchr(string, character1) != NULL) { // can remove "!= NULL"
      printf("\'%c\' was found in \"%s\".\n", character1, string);
   }
   else { // if character1 was not found
      printf("\'%c\' was not found in \"%s\".\n", character1, string);
   }

   // if character2 was found in string
   if (strchr(string, character2) != NULL) { // can remove "!= NULL"
      printf("\'%c\' was found in \"%s\".\n", character2, string);
   }
   else { // if character2 was not found
      printf("\'%c\' was not found in \"%s\".\n", character2, string);
   }
}
```

```
'a' was found in "This is a test".
'z' was not found in "This is a test".
```

Function *strcspn*

```c
// fig08_15.c
// Using function strcspn
#include <stdio.h>
#include <string.h>

int main(void) {
   // initialize two char pointers
   const char *string1 = "The value is 3.14159";
   const char *string2 = "1234567890";

   printf("string1 = %s\nstring2 = %s\n\n%s\n%s%zu\n", string1, string2,
      "The length of the initial segment of string1",
      "containing no characters from string2 = ",
      strcspn(string1, string2));
}
```

```
string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1
containing no characters from string2 = 13
```

Function *strpbrk*

```c
// fig08_16.c
// Using function strpbrk
#include <stdio.h>
#include <string.h>

int main(void) {
   const char *string1 = "This is a test";
   const char *string2 = "beware";

   printf("%s\"%s\"\n'%c'%s \"%s\"\n",
      "Of the characters in ", string2, *strpbrk(string1, string2),
      " appears earliest in ", string1);
}
```

```
Of the characters in "beware"
'a' appears earliest in "This is a test"
```

Function *strtok*

```c
// fig08_20.c
// Using function strtok
#include <stdio.h>
#include <string.h>

int main(void) {
   char string[] = "This is a sentence with 7 tokens";

   printf("The string to be tokenized is:\n%s\n\n", string);
   puts("The tokens are:");

   char *tokenPtr = strtok(string, " "); // begin tokenizing sentence

   // continue tokenizing sentence until tokenPtr becomes NULL
   while (tokenPtr != NULL) {
      printf("%s\n", tokenPtr);
      tokenPtr = strtok(NULL, " "); // get next token
   }
}
```

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

More functions (Part I)

Function prototype	Function description
`void *memcpy(void *s1, const void *s2, size_t n);`	
	Copies n bytes from the object pointed to by s2 into the object pointed to by s1, then returns a pointer to the resulting object.
`void *memmove(void *s1, const void *s2, size_t n);`	
	Copies n bytes from the object pointed to by s2 into the object pointed to by s1. The copy is performed as if the bytes were first copied from the object pointed to by s2 into a temporary array and then from the temporary array into the object pointed to by s1. A pointer to the resulting object is returned.

More functions (Part II)

Function prototype	Function description
`int memcmp(const void *s1, const void *s2, size_t n);`	Compares the first `n` bytes of the objects pointed to by `s1` and `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`.
`void *memchr(const void *s, int c, size_t n);`	Locates the first occurrence of `c` (converted to `unsigned char`) in the first `n` bytes of the object pointed to by `s`. If `c` is found, `memchr` returns a pointer to `c` in the object; otherwise, it returns `NULL`.
`void *memset(void *s, int c, size_t n);`	Copies `c` (converted to `unsigned char`) into the first `n` bytes of the object pointed to by `s`, then returns a pointer to the result.

More functions (Part III)

Function prototype	Function description
`char *strerror(int errornum);`	
	Maps `errornum` to a full text string in a compiler- and locale-specific manner and returns the string. Error numbers are defined in `errno.h`.
`size_t strlen(const char *s);`	
	Returns the length of string `s`—that is, the number of characters preceding the string's terminating null character.

Function *memcpy*

```c
// fig08_21.c
// Using function memcpy
#include <stdio.h>
#include <string.h>

int main(void) {
   char s1[17] = "";
   char s2[] = "Copy this string";

   memcpy(s1, s2, 17); // 17 so we copy s2's terminating \0
   puts("After s2 is copied into s1 with memcpy, s1 contains:");
   puts(s1);
}
```

```
After s2 is copied into s1 with memcpy, s1 contains:
Copy this string
```

Function *memmove*

```c
// fig08_22.c
// Using function memmove
#include <stdio.h>
#include <string.h>

int main(void) {
   char x[] = "Home Sweet Home"; // initialize char array x

   printf("The string in array x before memmove is: %s\n", x);
   printf("The string in array x after memmove is: %s\n",
      (char *) memmove(x, &x[5], 10));
}
```

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is: Sweet Home Home
```

Function *memcmp*

```c
// fig08_23.c
// Using function memcmp
#include <stdio.h>
#include <string.h>

int main(void) {
   char s1[] = "ABCDEFG";
   char s2[] = "ABCDXYZ";

   printf("s1 = %s\ns2 = %s\n\n%s%2d\n%s%2d\n%s%2d\n", s1, s2,
      "memcmp(s1, s2, 4) = ", memcmp(s1, s2, 4),
      "memcmp(s1, s2, 7) = ", memcmp(s1, s2, 7),
      "memcmp(s2, s1, 7) = ", memcmp(s2, s1, 7));
}
```

```
s1 = ABCDEFG
s2 = ABCDXYZ

memcmp(s1, s2, 4) =  0
memcmp(s1, s2, 7) = -1
memcmp(s2, s1, 7) =  1
```

Function *memchr*

```c
// fig08_24.c
// Using function memchr
#include <stdio.h>
#include <string.h>

int main(void) {
   const char *s = "This is a string";

   printf("The remainder of s after character 'r' is found is \"%s\"\n",
      (char *) memchr(s, 'r', 16));
}
```

```
The remainder of s after character 'r' is found is "ring"
```

Function memset

```c
// fig08_25.c
// Using function memset
#include <stdio.h>
#include <string.h>

int main(void) {
   char string1[15] = "BBBBBBBBBBBBBB";

   printf("string1 = %s\n", string1);
   printf("string1 after memset = %s\n", (char *) memset(string1, 'b', 7));
}
```

```
string1 = BBBBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```

Function *strlen*

```c
// fig08_27.c
// Using function strlen
#include <stdio.h>
#include <string.h>

int main(void) {
   const char *string1 = "abcdefghijklmnopqrstuvwxyz";
   const char *string2 = "four";
   const char *string3 = "Boston";

   printf("%s\"%s\"%s%zu\n%s\"%s\"%s%zu\n%s\"%s\"%s%zu\n",
      "The length of ", string1, " is ", strlen(string1),
      "The length of ", string2, " is ", strlen(string2),
      "The length of ", string3, " is ", strlen(string3));
}
```

```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```