8

C Characters and Strings

8.2 Fundamentals of Strings and Characters

- **Characters**
  - **Building blocks of programs**
    - **Every program is a sequence of meaningfully grouped characters**
  - **Character constant**
    - **An `int` value represented as a character in single quotes**
    - **`'z'` represents the integer value (ASCII code) of `z`**
- **Strings**
  - **Series of characters treated as a single unit**
    - **Can include letters, digits and special characters `(*, /, $)`**
  - **String literal (string constant) - written in double quotes**
    - **`"Hello"`**
  - **Strings are arrays of characters**
    - **String a pointer to first character**
    - **Value of string is the address of first character**

8.2 Fundamentals of Strings and Characters

- **String definitions**
  - **Define as a character array or a variable of type `char *`**

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

  - **Remember that strings represented as character arrays end with `'\0'`**
    - **`color` has `5` elements**
- **Inputting strings**
  - **Use `scanf`**

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

    - **Copies input into `word[]`**
    - **Do not need `&` (because a string is a pointer)**
  - **Remember to leave room in the array for `'\0'`**

Error-Prevention Tip 8.1

When storing a string of characters in a character array, be sure that the array is large enough to hold the largest string that will be stored. C allows strings of any length to be stored. If a string is longer than the character array in which it is to be stored, characters beyond the end of the array will overwrite data in memory following the array.

Common Programming Errors

8.4: Passing a character as an argument to a function when a string is expected is a syntax error.

8.5: Passing a string as an argument to a function when a character is expected is a syntax error.

8.3 Character Handling Library

- **Character handling library**
  - **Includes functions to perform useful tests and manipulations of character data**
  - **Each function receives a character (an `int`) or `EOF` as an argument**
- **The following slides contain a table of all the functions in `<ctype.h>`**

Prototype	Function description
`int isdigit( int c );`	Returns a true value if `c` is a digit and 0 (false) otherwise.
`int isalpha( int c );`	Returns a true value if `c` is a letter and 0 otherwise.
`int isalnum( int c );`	Returns a true value if `c` is a digit or a letter and 0 otherwise.
`int isxdigit( int c );`	Returns a true value if `c` is a hexadecimal digit character and 0 otherwise. (See Appendix E, Number Systems, 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 and 0 otherwise.
`int isupper( int c );`	Returns a true value if `c` is an uppercase letter and 0 otherwise.
`int tolower( int c );`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter. Otherwise, `tolower` returns the argument unchanged.

Fig. 8.1 | Character-handling library functions. (Part 1 of 2.)

Prototype	Function description
`int toupper( int c );`	If `c` is a lowercase letter, `toupper` returns `c` as an uppercase letter. Otherwise, `toupper` returns the argument unchanged.
`int isspace( int c );`	Returns a true value if `c` is a white-space character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 otherwise.
`int iscntrl( int c );`	Returns a true value if `c` is a control character and 0 otherwise.
`int ispunct( int c );`	Returns a true value if `c` is a printing character other than a space, a digit, or a letter and returns 0 otherwise.
`int isprint( int c );`	Returns a true value if `c` is a printing character including a space (`' '`) and returns 0 otherwise.
`int isgraph( int c );`	Returns a true value if `c` is a printing character other than a space (`' '`) and returns 0 otherwise.

Fig. 8.1 | Character-handling library functions. (Part 2 of 2.)

Outline

fig08_03.c

(1 of 2)

```
1  /* Fig. 8.3: fig08_03.c
2     Using functions islower, isupper, tolower, toupper */
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main( void )
7  {
8     printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
9             "According to islower:",
10            islower( 'p' ) ? "p is a " : "p is not a ",
11            "lowercase letter",
12            islower( 'P' ) ? "P is a " : "P is not a ",
13            "lowercase letter",
14            islower( '5' ) ? "5 is a " : "5 is not a ",
15            "lowercase letter",
16            islower( '!' ) ? "! is a " : "! is not a ",
17            "lowercase letter" );
18
19    printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
20            "According to isupper:",
21            isupper( 'D' ) ? "D is an " : "D is not an ",
22            "uppercase letter",
23            isupper( 'd' ) ? "d is an " : "d is not an ",
24            "uppercase letter",
25            isupper( '8' ) ? "8 is an " : "8 is not an ",
26            "uppercase letter",
27            isupper( '$' ) ? "$ is an " : "$ is not an ",
28            "uppercase letter" );
29
```

islower tests if a character is a lowercase letter

isupper tests if a character is an uppercase letter

Outline

fig08_03.c

(2 of 2)

```
30    printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
31            "u converted to uppercase is ", toupper( 'u' ),
32            "7 converted to uppercase is ", toupper( '7' ),
33            "$ converted to uppercase is ", toupper( '$' ),
34            "L converted to lowercase is ", tolower( 'L' ) );
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

toupper and **tolower** convert letters to upper or lower case

```
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

According to isupper:
D is an uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
$ is not an uppercase letter

u converted to uppercase is U
7 converted to uppercase is 7
$ converted to uppercase is $
L converted to lowercase is l
```

Outline

fig08_04.c

(1 of 2)

```
1  /* Fig. 8.4: fig08_04.c
2     Using functions isspace, iscntrl, ispunct, isprint, isgraph */
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main( void )
7  {
8     printf( "%s\n%s%s%s\n%s%s%s\n%s%s\n\n",
9         "According to isspace:",
10        "Newline", isspace( '\n' ) ? " is a " : " is not a ",
11        "whitespace character", "Horizontal tab",
12        isspace( '\t' ) ? " is a " : " is not a ",
13        "whitespace character",
14        isspace( '%' ) ? "% is a " : "% is not a ",
15        "whitespace character" );
16
17    printf( "%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
18        "Newline", iscntrl( '\n' ) ? " is a " : " is not a ",
19        "control character", iscntrl( '$' ) ? "$ is a " :
20        "$ is not a ", "control character" );
21
22    printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
23        "According to ispunct:",
24        ispunct( ';' ) ? "; is a " : "; is not a ",
25        "punctuation character",
26        ispunct( 'Y' ) ? "Y is a " : "Y is not a ",
27        "punctuation character",
28        ispunct( '#' ) ? "# is a " : "# is not a ",
29        "punctuation character" );
30
```

isspace tests if a character is a whitespace character

iscntrl tests if a character is a control character

ispunct tests if a character is a punctuation character

Outline

```
31    printf( "%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
32        isprint( '$' ) ? "$ is a " : "$ is not a ",
33        "printing character",
34        "Alert", isprint( '\a' ) ? " is a " : " is not a ",
35        "printing character" );
36
37    printf( "%s\n%s%s\n%s%s%s\n",  "According to isgraph:",
38        isgraph( 'Q' ) ? "Q is a " : "Q is not a ",
39        "printing character other than a space",
40        "Space", isgraph( ' ' ) ? " is a " : " is not a ",
41        "printing character other than a space" );
42
43    return 0; /* indicates successful termination */
44
45 } /* end main */
```

isprint tests if a character is a printing character

isgraph tests if a character is a printing character that is not a space

fig08_04.c

(2 of 2)

```
According to isspace:
Newline is a whitespace character
Horizontal tab is a whitespace character
% is not a whitespace character

According to iscntrl:
Newline is a control character
$ is not a control character

According to ispunct:
; is a punctuation character
Y is not a punctuation character
# is a punctuation character

According to isprint:
$ is a printing character
Alert is not a printing character

According to isgraph:
Q is a printing character other than a space
Space is not a printing character other than a space
```

8.4 String-Conversion Functions

- **Conversion functions**
  - **In `<stdlib.h>` (general utilities library)**
- **Convert strings of digits to integer and floating-point values**

Function prototype	Function description
`double atof( const char *nPtr );`	Converts the string `nPtr` to `double`.
`int atoi( const char *nPtr );`	Converts the string `nPtr` to `int`.
`long atol( const char *nPtr );`	Converts the string `nPtr` to `long int`.
`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`.

Fig. 8.5 | String-conversion functions of the general utilities library.

Outline

fig08_06.c

```
1  /* Fig. 8.6: fig08_06.c
2     Using atof */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main( void )
7  {
8     double d; /* variable to hold converted string */
9
10    d = atof( "99.0" );
11
12    printf( "%s%.3f\n%s%.3f\n",
13            "The string \"99.0\" converted to double is ", d,
14            "The converted value divided by 2 is ",
15            d / 2.0 );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

`atof` converts a string to a **`double`**

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

Outline

fig08_07.c

```
1  /* Fig. 8.7: fig08_07.c
2     Using atoi */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main( void )
7  {
8     int i; /* variable to hold converted string */
9
10    i = atoi( "2593" );
11
12    printf( "%s%d\n%s%d\n",
13            "The string \"2593\" converted to int is ", i,
14            "The converted value minus 593 is ", i - 593 );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
```

atoi converts a string to an **int**

```
The string "2593" converted to int is 2593
The converted value minus 593 is 2000
```

Outline

fig08_09.c

```
1  /* Fig. 8.9: fig08_09.c
2     Using strtod */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main( void )
7  {
8     /* initialize string pointer */
9     const char *string = "51.2% are admitted"; /* initialize string */
10
11    double d;        /* variable to hold converted sequence */
12    char *stringPtr; /* create char pointer */
13
14    d = strtod( string, &stringPtr );
15
16    printf( "The string \"%s\" is converted to the\n", string );
17    printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

strtod converts a piece of a string to a **double**

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

8.5 Standard Input/Output Library Functions

- **Used to manipulate character and string data**
- **Functions in `<stdio.h>`**

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array `s` until a newline or end-of-file character is encountered. A terminating null character is appended to the array. Returns the string inputted into `s`. Note that an error will occur if `s` is not large enough to hold the 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 non-zero integer if successful, or `EOF` if an error occurs.
`int sprintf( char *s, const char *format, ... );`	Equivalent to `printf`, except 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`, except 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.

Outline

fig08_13.c

(1 of 2)

```
1  /* Fig. 8.13: fig08_13.c
2     Using gets and putchar */
3  #include <stdio.h>
4
5  void reverse( const char * const sPtr ); /* prototype */
6
7  int main( void )
8  {
9     char sentence[ 80 ]; /* create char array */
10
11    printf( "Enter a line of text:\n" );
12
13    /* use gets to read line of text */
14    gets( sentence );
15
16    printf( "\nThe line printed backward is:\n" );
17    reverse( sentence );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

`gets` reads a line of text from the user

Outline

fig08_13.c

(2 of 2)

```
22
23 /* recursively outputs characters in string in reverse order */
24 void reverse( const char * const sPtr )
25 {
26    /* if end of the string */
27    if ( sPtr[ 0 ] == '\0' ) { /* base case */
28       return;
29    } /* end if */
30    else { /* if not end of the string */
31       reverse( &sPtr[ 1 ] ); /* recursion step */
32
33       putchar( sPtr[ 0 ] ); /* use putchar to display character */
34    } /* end else */
35
36 } /* end function reverse */
```

putchar prints a single character on the screen

```
Enter a line of text:
Characters and Strings

The line printed backward is:
sgnirtS dna sretcarahC
```

```
Enter a line of text:
able was I ere I saw elba

The line printed backward is:
able was I ere I saw elba
```

Outline

fig08_14.c

```
1  /* Fig. 8.14: fig08_14.c
2     Using getchar and puts */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char c;              /* variable to hold character input by user */
8     char sentence[ 80 ]; /* create char array */
9     int i = 0;           /* initialize counter i */
10
11    /* prompt user to enter line of text */
12    puts( "Enter a line of text:" );
13
14    /* use getchar to read each character */
15    while ( ( c = getchar() ) != '\n') {
16       sentence[ i++ ] = c;
17    } /* end while */
18
19    sentence[ i ] = '\0'; /* terminate string */
20
21    /* use puts to display sentence */
22    puts( "\nThe line entered was:" );
23    puts( sentence );
24
25    return 0; /* indicates successful termination */
26
27 } /* end main */
```

puts prints a line of text on the screen

getchar reads a single character from the user

```
Enter a line of text:
This is a test.

The line entered was:
This is a test.
```

Outline

fig08_15.c

```
1  /* Fig. 8.15: fig08_15.c
2     Using sprintf */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char s[ 80 ]; /* create char array */
8     int x;        /* x value to be input */
9     double y;     /* y value to be input */
10
11    printf( "Enter an integer and a double:\n" );
12    scanf( "%d%lf", &x, &y );
13
14    sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );
15
16    printf( "%s\n%s\n",
17            "The formatted output stored in array s is:", s );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

sprintf prints a line of text into an array like **printf** prints text on the screen

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

Outline

fig08_16.c

```
1  /* Fig. 8.16: fig08_16.c
2     Using sscanf */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char s[] = "31298 87.375"; /* initialize array s */
8     int x;    /* x value to be input */
9     double y; /* y value to be input */
10
11    sscanf( s, "%d%lf", &x, &y );
12
13    printf( "%s\n%s%6d\n%s%8.3f\n",
14            "The values stored in character array s are:",
15            "integer:", x, "double:", y );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

sscanf reads a line of text from an array like **scanf** reads text from the user

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

8.6 String Manipulation Functions of the String Handling Library

- **String handling library has functions to**
  - **Manipulate string data**
  - **Search strings**
  - **Tokenize strings**
  - **Determine string length**

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`. The value of `s1` is returned.
`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.

Fig. 8.17 | String-manipulation functions of the string-handling library.

Outline

fig08_18.c

```
1  /* Fig. 8.18: fig08_18.c
2     Using strcpy and strncpy */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     char x[] = "Happy Birthday to You"; /* initialize char array x */
9     char y[ 25 ]; /* create char array y */
10    char z[ 15 ]; /* create char array z */
11
12    /* copy contents of x into y */
13    printf( "%s%s\n%s%s\n",
14       "The string in array x is: ", x,
15       "The string in array y is: ", strcpy( y, x ) );
16
17    /* copy first 14 characters of x into z. Does not copy null
18       character */
19    strncpy( z, x, 14 );
20
21    z[ 14 ] = '\0'; /* terminate string in z */
22    printf( "The string in array z is: %s\n", z );
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

```
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
```

strcpy copies string **x** into character array **y**

strncpy copies 14 characters of string **x** into character array **z**

Note that **strncpy** does not automatically append a null character

Outline

fig08_19.c

```
1  /* Fig. 8.19: fig08_19.c
2     Using strcat and strncat */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     char s1[ 20 ] = "Happy "; /* initialize char array s1 */
9     char s2[] = "New Year ";  /* initialize char array s2 */
10    char s3[ 40 ] = "";       /* initialize char array s3 to empty */
11
12    printf( "s1 = %s\ns2 = %s\n", s1, s2 );
13
14    /* concatenate s2 to s1 */
15    printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
16
17    /* concatenate first 6 characters of s1 to s3. Place '\0'
18       after last character */
19    printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
20
21    /* concatenate s1 to s3 */
22    printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

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

strcat adds the characters of string **s2** to the end of string **s1**

strncat adds the first 6 characters of string **s1** to the end of string **s3**

8.7 Comparison Functions of the String-Handling Library

- **Comparing strings**
  - **–Computer compares numeric ASCII codes of characters in string**
  - **–Appendix D has a list of character codes**

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.

Fig. 8.20 | String-comparison functions of the string-handling library.

Outline

fig08_21.c

```
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     const char *s1 = "Happy New Year"; /* initialize char pointer */
9     const char *s2 = "Happy New Year"; /* initialize char pointer */
10    const char *s3 = "Happy Holidays"; /* initialize char pointer */
11
12    printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
13           "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
14           "strcmp(s1, s2) = ", strcmp( s1, s2 ),
15           "strcmp(s1, s3) = ", strcmp( s1, s3 ),
16           "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
17
18    printf("%s%2d\n%s%2d\n%s%2d\n",
19           "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
20           "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
21           "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

strcmp compares string **s1** to string **s2**

strncmp compares the first 6 characters of string **s1** to the first 6 characters of string **s3**

```
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
```

Common Programming Error 8.7

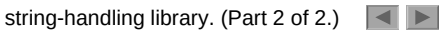
Assuming that `strcmp` and `strncmp` return 1 when their arguments are equal is a logic error. Both functions return 0 (strangely, the equivalent of C's false value) for equality. Therefore, when testing two strings for equality, the result of function `strcmp` or `strncmp` should be compared with 0 to determine if the strings are equal.

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. 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`.
`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 string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.

Fig. 8.22 | String-manipulation functions of the string-handling library. (Part 1 of 2.)

Function prototype	Function description
`char *strrchr( const char *s, int c );`	Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of string `s2`. If the string is found, a pointer to the string in `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to `strtok` breaks string `s1` into "tokens"—logical pieces such as words in a line of text—separated by characters contained in string `s2`. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.
`size_t strlen( const char *s );`	Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

Fig. 8.22 | String-manipulation functions of the string-handling library. (Part 2 of 2.)



Outline

fig08_23.c

```
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     const char *string = "This is a test"; /* initialize char pointer */
9     char character1 = 'a'; /* initialize character1 */
10    char character2 = 'z'; /* initialize character2 */
11
12    /* if character1 was found in string */
13    if ( strchr( string, character1 ) != NULL ) {
14       printf( "\'%c\' was found in \"%s\".\n",
15          character1, string );
16    } /* end if */
17    else { /* if character1 was not found */
18       printf( "\'%c\' was not found in \"%s\".\n",
19          character1, string );
20    } /* end else */
21
22    /* if character2 was found in string */
23    if ( strchr( string, character2 ) != NULL ) {
24       printf( "\'%c\' was found in \"%s\".\n",
25          character2, string );
26    } /* end if */
27    else { /* if character2 was not found */
28       printf( "\'%c\' was not found in \"%s\".\n",
29          character2, string );
30    } /* end else */
31
32    return 0; /* indicates successful termination */
33
34 } /* end main */
```

strchr searches for the first instance of **character1** in **string**

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

Outline

fig08_26.c

```
1  /* Fig. 8.26: fig08_26.c
2     Using strrchr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     /* initialize char pointer */
9     const char *string1 = "A zoo has many animals including zebras";
10
11    int c = 'z'; /* character to search for */
12
13    printf( "%s\n%s'%c'%s\"%s\"\n",
14            "The remainder of string1 beginning with the",
15            "last occurrence of character ", c,
16            " is: ", strrchr( string1, c ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

```
The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"
```

strrchr returns the remainder of **string1** following the last occurrence of the character **c**

Outline

fig08_24.c

```
1  /* Fig. 8.24: fig08_24.c
2     Using strcspn */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     /* initialize two char pointers */
9     const char *string1 = "The value is 3.14159";
10    const char *string2 = "1234567890";
11
12    printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
13       "string1 = ", string1, "string2 = ", string2,
14       "The length of the initial segment of string1",
15       "containing no characters from string2 = ",
16       strcspn( string1, string2 ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

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

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

strcspn returns the length of the initial segment of **string1** that does not contain any characters in **string2**

Outline

fig08_27.c

```
1  /* Fig. 8.27: fig08_27.c
2     Using strspn */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     /* initialize two char pointers */
9     const char *string1 = "The value is 3.14159";
10    const char *string2 = "aehi lsTuv";
11
12    printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
13       "string1 = ", string1, "string2 = ", string2,
14       "The length of the initial segment of string1",
15       "containing only characters from string2 = ",
16       strspn( string1, string2 ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

`strspn` returns the length of the initial segment of **`string1`** that contains only characters from **`string2`**

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

Outline

fig08_25.c

```
1  /* Fig. 8.25: fig08_25.c
2     Using strpbrk */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     const char *string1 = "This is a test"; /* initialize char pointer */
9     const char *string2 = "beware";         /* initialize char pointer */
10
11    printf( "%s\"%s\"\n'%c'%s\n\"%s\"\n",
12       "Of the characters in ", string2,
13       *strpbrk( string1, string2 ),
14       " appears earliest in ", string1 );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
```

`strpbrk` returns a pointer to the first appearance in **`string1`** of any character from **`string2`**

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

Outline

fig08_28.c

```
1  /* Fig. 8.28: fig08_28.c
2     Using strstr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     const char *string1 = "abcdefabcdef"; /* string to search */
9     const char *string2 = "def"; /* string to search for */
10
11    printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
12       "string1 = ", string1, "string2 = ", string2,
13       "The remainder of string1 beginning with the",
14       "first occurrence of string2 is: ",
15       strstr( string1, string2 ) );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

```
string1 = abcdefabcdef
string2 = def

The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

strstr returns the remainder of **string1** following the last occurrence of **string2**

Outline

fig08_29.c

```
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     /* initialize array string */
9     char string[] = "This is a sentence with 7 tokens";
10    char *tokenPtr; /* create char pointer */
11
12    printf( "%s\n%s\n\n%s\n",
13       "The string to be tokenized is:", string,
14       "The tokens are:" );
15
16    tokenPtr = strtok( string, " " ); /* begin tokenizing sentence */
17
18    /* continue tokenizing sentence until tokenPtr becomes NULL */
19    while ( tokenPtr != NULL ) {
20       printf( "%s\n", tokenPtr );
21       tokenPtr = strtok( NULL, " " ); /* get next token */
22    } /* end while */
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

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

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

strtok "tokenizes" **string** by breaking it into tokens at each space

Calling **strtok** again and passing it **NULL** continues the tokenizing of the previous string

Outline

fig08_38.c

```
1  /* Fig. 8.38: fig08_38.c
2     Using strlen */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     /* initialize 3 char pointers */
9     const char *string1 = "abcdefghijklmnopqrstuvwxyz";
10    const char *string2 = "four";
11    const char *string3 = "Boston";
12
13    printf("%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n",
14       "The length of ", string1, " is ",
15       ( unsigned long ) strlen( string1 ),
16       "The length of ", string2, " is ",
17       ( unsigned long ) strlen( string2 ),
18       "The length of ", string3, " is ",
19       ( unsigned long ) strlen( string3 ) );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

strlen returns the length of **string1**

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

22

Operator Overloading; String and Array Objects

22.13 Standard Library Class `string`

- Class `string`
  - Header `<string>`, namespace `std`
  - Can initialize `string s1( "hi" );`
  - Overloaded `<<` (as in `cout << s1`)
  - Overloaded relational operators
    - `==`, `!=`, `>=`, `>`, `<=`, `<`
  - Assignment operator `=`
  - Concatenation (overloaded `+=`)

22.13 Standard Library Class `string` (Cont.)

- Class `string` (Cont.)
  - Substring member function `substr`
    - `s1.substr( 0, 14 );`
      - Starts at location 0, gets 14 characters
    - `s1.substr( 15 );`
      - Substring beginning at location 15, to the end
  - Overloaded `[]`
    - Access one character
    - No range checking (if subscript invalid)
  - Member function `at`
    - Accesses one character
      - `Example`
        - `s1.at( 10 );`
    - Has bounds checking, throws an exception if subscript is invalid
      - Will end program (learn more in Chapter 16)

Outline

fig22_15.cpp

```
1  // Fig. 22.15: fig22_15.cpp
2  // Standard Library string class test program.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 int main()
11 {
12    string s1( "happy" );
13    string s2( " birthday" );
14    string s3;
15
16    // test overloaded equality and relational operators
17    cout << "s1 is \"" << s1 << "\"; s2 is \"" << s2
18       << "\"; s3 is \"" << s3 << '\"'
19       << "\n\nThe results of comparing s2 and s1:"
20       << "\ns2 == s1 yields " << ( s2 == s1 ? "true" : "false" )
21       << "\ns2 != s1 yields " << ( s2 != s1 ? "true" : "false" )
22       << "\ns2 >  s1 yields " << ( s2 > s1 ? "true" : "false" )
23       << "\ns2 <  s1 yields " << ( s2 < s1 ? "true" : "false" )
24       << "\ns2 >= s1 yields " << ( s2 >= s1 ? "true" : "false" )
25       << "\ns2 <= s1 yields " << ( s2 <= s1 ? "true" : "false" );
26
27    // test string member-function empty
28    cout << "\n\nTesting s3.empty():" << endl;
```

Passing strings to the `string` constructor

Create empty **`string`**

fig22_15.cpp

(2 of 4)

```
29
30    if ( s3.empty() )
31    {
32       cout << "s3 is empty; assigning s1 to s3;" << endl;
33       s3 = s1; // assign s1 to s3
34       cout << "s3 is \"" << s3 << "\"";
35    } // end if
36
37    // test overloaded string concatenation operator
38    cout << "\n\ns1 += s2 yields s1 = ";
39    s1 += s2; // test overloaded concatenation
40    cout << s1;
41
42    // test overloaded string concatenation operator with C-style string
43    cout << "\n\ns1 += \" to you\" yields" << endl;
44    s1 += " to you";
45    cout << "s1 = " << s1 << "\n\n";
46
47    // test string member function substr
48    cout << "The substring of s1 starting at location 0 for\n"
49       << "14 characters, s1.substr(0, 14), is:\n"
50       << s1.substr( 0, 14 ) << "\n\n";
51
52    // test substr "to-end-of-string" option
53    cout << "The substring of s1 starting at\n"
54       << "location 15, s1.substr(15), is:\n"
55       << s1.substr( 15 ) << endl;
```

Member function **`empty`** tests if the **`string`** is empty

Member function **`substr`** obtains a substring from the **`string`**

Outline

fig22_15.cpp

(3 of 4)

```
56
57    // test copy constructor
58    string *s4Ptr = new string( s1 );
59    cout << "\n*s4Ptr = " << *s4Ptr << "\n\n";
60
61    // test assignment (=) operator with self-assignment
62    cout << "assigning *s4Ptr to *s4Ptr" << endl;
63    *s4Ptr = *s4Ptr;
64    cout << "*s4Ptr = " << *s4Ptr << endl;
65
66    // test destructor
67    delete s4Ptr;
68
69    // test using subscript operator to create lvalue
70    s1[ 0 ] = 'H';
71    s1[ 6 ] = 'B';
72    cout << "\ns1 after s1[0] = 'H' and s1[6] = 'B' is: "
73       << s1 << "\n\n";
74
75    // test subscript out of range with string member function "at"
76    cout << "Attempt to assign 'd' to s1.at( 30 ) yields:" << endl;
77    s1.at( 30 ) = 'd'; // ERROR: subscript out of range
78    return 0;
79 } // end main
```

Accessing specific character in **`string`**

Member function **`at`** provides range checking

Outline

fig22_15.cpp

(4 of 4)

```
s1 is "happy"; s2 is " birthday"; s3 is ""

The results of comparing s2 and s1:
s2 == s1 yields false
s2 != s1 yields true
s2 >  s1 yields false
s2 <  s1 yields true
s2 >= s1 yields false
s2 <= s1 yields true

Testing s3.empty():
s3 is empty; assigning s1 to s3;
s3 is "happy"

s1 += s2 yields s1 = happy birthday

s1 += " to you" yields
s1 = happy birthday to you

The substring of s1 starting at location 0 for
14 characters, s1.substr(0, 14), is:
happy birthday

The substring of s1 starting at
location 15, s1.substr(15), is:
to you

*s4Ptr = happy birthday to you

assigning *s4Ptr to *s4Ptr
*s4Ptr = happy birthday to you

s1 after s1[0] = 'H' and s1[6] = 'B' is: Happy Birthday to you

Attempt to assign 'd' to s1.at( 30 ) yields:

abnormal program termination
```