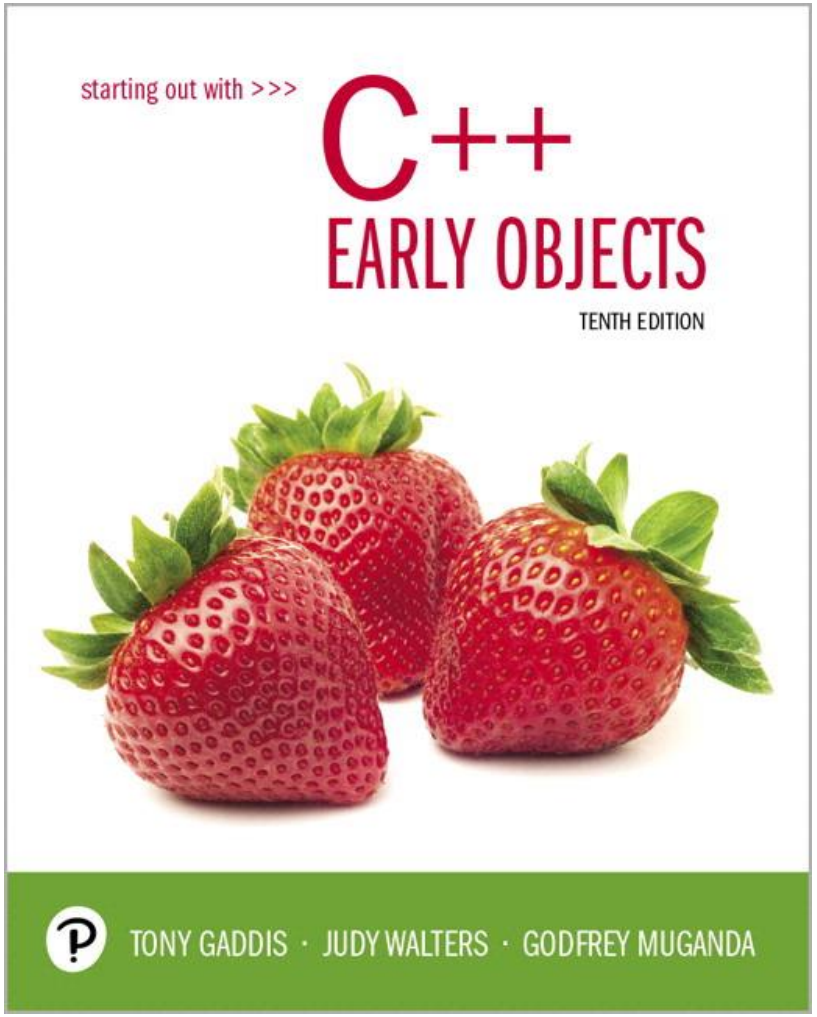
Starting Out with C++ Early Objects

Tenth Edition



Chapter 12

More on C-Strings and the **`string`** Class

Topics

12.1 Character Testing

12.2 Character Case Conversion

12.3 C-Strings

12.4 Library Functions for Working with C-Strings

12.5 Conversions Between Numbers and Strings

12.6 Writing Your Own C-String Handling Functions

12.7 More About the C++ **`string`** Class

12.1 Character Testing

Functions to determine information about the value of a character

Argument: a **`char`**

Return value: a non-zero value or zero, representing **`true`** or **`false`**

Requires: **`<cctype>`** header file

Character Testing Functions 1 of 2

Function	Description
`isalpha`	true if argument is a letter of the alphabet
`isdigit`	true if argument is a digit 0 – 9
`isalnum`	true if argument is a letter or a digit
`islower`	true if argument is a lowercase letter
`isupper`	true if argument is an uppercase letter
`isprint`	true if argument is a printable character
`isspace`	true if argument is a whitespace character: space, horizontal or vertical tab, or newline
`ispunct`	true if argument is a punctuation character: printable, but not a letter, digit, or a space

Character Testing 2 of 2

Functions can be used to validate data format

```
// Part number: digit,uppercase letter,
// digit
bool valid = false;
if ((isdigit(partNum[0])) &&
     (isupper(partNum[1])) &&
     (isdigit(partNum[2])))
          valid = true;
```

Hands-On Pg. 822

- Listing 12-1

Hands-On Pg. 823

- Listing 12-2

12.2 Character Case Conversion

Functions to convert uppercase letters to their lowercase equivalent, and vice versa.

Argument: a **`char`**

Return value: a **`char`**

Requires: **`<cctype>`** header file

Character Conversion Functions 1 of 2

Function	Description
`toupper`	If the argument is a lowercase letter, returns the uppercase equivalent; returns the argument otherwise.
`tolower`	If the argument is an uppercase letter, returns the lowercase equivalent; returns the argument otherwise

Character Conversion Functions 2 of 2

Can be used to simplify data testing

```
// Determine if user wants to continue
char response;
cout << "Continue?  (Y/N) ";
cin >> response;
if (toupper(response) == 'Y')
{  // yes, continue
   . . .
}
```

Hands-On Pg. 826

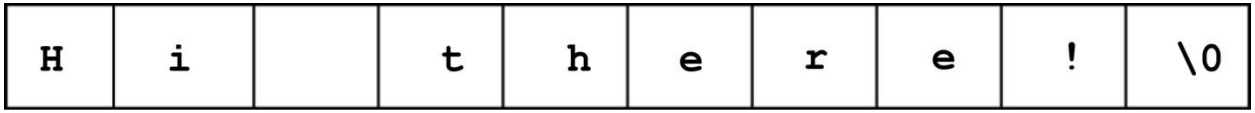
- Listing 12-3

12.3 C-Strings

- C-string: sequence of characters stored in adjacent memory locations and terminated by **`NULL`** character
- The C-string

  `"Hi there!"`

  would be stored in memory as shown:



Hands-On Pg. 829

- Listing 12-4

Hands-On Pg. 831

- Listing 12-5

What is NULL?

- The null character is used to indicate the end of a string
- It can be specified as
  - the character `'\0'`
  - the `int` value 0
  - the named constant **`NULL`**

Representation of C-strings

As a string literal

```
"Hi There!"
```

As a pointer to **`char`**

```
char *p;
```

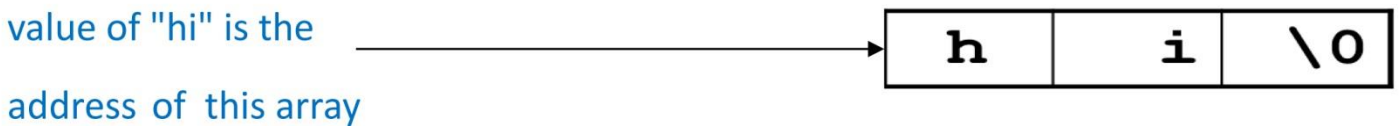
As an array of characters

```
char str[20];
```

All three representations are pointers to char

String Literals

- A string literal (string constant) is stored as a null-terminated array of **`char`**
- The compiler uses the address of the first character of the array as the value of the string
- A string literal is a **`const`** pointer to char



Array of char 1 of 2

- An array of char can be defined and initialized to a C-string

```
char str1[20] = "hi";
char str2[] = "Hello";
```

- An array of char can be defined and later have a string copied into it using **cin.getline**

```
char str3[20];
cout << "Enter your name: ";
cin.getline(str3, 20);
```

Array of `char` 2 of 2

- The name of an array of char is used as a pointer to char
- Unlike a string literal, a C-string defined as an array can be referred to in other parts of the program by using the array name

Pointer to `char` 1 of 2

- Defined as

```
char *pStr;
```

- Definition not allocate string memory
- It is useful in referring to C-strings defined as string literals

```
pStr = "Hi there";
cout << pStr << " "
     << pStr;
```

Pointer to `char` 2 of 2

- Pointer to **`char`** can also refer to C-strings defined as arrays of char

```
char str[20] = "hi";

char *pStr = str;

cout << pStr;  // prints hi
```

- Can dynamically allocate memory to be used for C-string using **`new`**

```
pStr = new char[20];
```

Hands-On Pg. 832

- Listing 12-6

12.4 Library Functions for Working with C-Strings 1 of 2

- Require **`cstring`** header file
- Functions take one or more C-strings as arguments. Argument can be:
  - Name of an array of char
  - pointer to char
  - string literal

Library Functions for Working with C-Strings 2 of 2

```
int strlen(char *str)
```

Returns length of a C-string:

```
cout << strlen("hello");
```

Prints: **5**

Note: This is the number of characters in the string, not including the terminating **`null`**, NOT the size of the array that contains it

strcat

```
strcat(char *dest, char *source)
```

- Takes two C-strings as input. It adds the contents of the second string to the end of the first string:

```
char str1[15] = "Good ";
char str2[30] = "Morning!";
strcat(str1, str2);
cout << str1; // prints: Good Morning!
```

- No automatic bounds checking: programmer must ensure that 1st string has enough room for result

strcpy

```
strcpy(char *dest, char *source)
```

- Copies a string from a source address to a destination address

```
char name[15];
strcpy(name, "Deborah");
cout << name; // prints Deborah
```

- Bounds checking is up to the programmer.

Hands-On Pg. 835

- Listing 12-7

strcmp 1 of 2

```
int strcmp(char *str1, char*str2)
```

- Compares strings stored at two addresses to determine their relative alphabetic order:
- Returns a value:

  less than 0 if `str1` precedes `str2`

  equal to 0 if `str1` equals `str2`

  greater than 0 if `str1` succeeds `str2`

strcmp 2 of 2

- It can be used to test for equality

```
if(strcmp(str1, str2) == 0)
   cout << "They are equal";
else
   cout << "They are not equal";
```

- Also used to determine ordering of C-strings when sorting
- Note:
  - Comparisons are case-sensitive: `"Hi"` is not `"hi"`
  - C-strings cannot be compared using relational operators (compares addresses of C-strings, not contents)

Hands-On Pg. 837

- Listing 12-8

Hands-On Pg. 838

- Listing 12-9

Hands-On Pg. 839

- Listing 12-10

12.5 Conversions Between Numbers and Strings

- **`"4326"`** is a string; **`4326`** without quotes is an **`int`**
- There are classes that can be used to convert between string and numeric representations of numbers
- Need to include **`sstream`** header file

Conversion Classes

- **`istringstream`:**
  - contains a string to be converted to numeric values where necessary
  - Use **`str(s)`** to initialize string to contents of string **`s`**
  - Use the stream extraction operator **>>** to read from the string
- **`ostringstream`:**
  - collects a string in which numeric data is converted as necessary
  - Use the stream insertion operator **<<** to add data onto the string
  - Use **`str()`** to retrieve converted string

Numeric Conversion Functions

- Introduced in C++ 11
- Allow conversion between numeric values and strings
  - **`to_string()`** – convert a numeric value to a string
  - **`stoi()`** – convert a string to an int
  - **`stol()`** – convert a string to a long
  - **`stof()`** – convert a string to a float
  - **`stod()`** – convert a string to a double

Example - `stoi`

- `stoi` converts string to int

```
int stoi(const string& str,
    size_t* pos=0, int base = 10)
```

  - **`str`**: string containing the number, possibly other chars
  - **`pos`**: the position in str where conversion stops
  - **`base`**: the base to use for integral conversion

- Example:

```
int number; size_t where;     //number will
string data = "23 dozen";     //have 23,
number = stoi(data, &where); //where has 2
```

Hands-On Pg. 843

- Listing 12-11

Hands-On Pg. 845

- Listing 12-12

12.6 Writing Your Own C-String Handling Functions

When writing C-String Handling Functions:

- can pass arrays or pointers to **`char`**
- can perform bounds checking to ensure enough space for results
- can anticipate unexpected user input

Hands-On Pg. 847

- Listing 12-13

Hands-On Pg. 848

- Listing 12-14

12.7 More About the C++ `string` Class

- The string class offers several advantages over C-style strings:
  - large body of member functions
  - overloaded operators to simplify expressions
- Need to include the **`string`** header file

`string` class constructors

- Default constructor `string()`
- Copy constructor `string(string&)` initializes string objects with values of other string objects
- Convert constructor `string(char *)` initializes string objects with values of C-strings

Overloaded `string` Operators 1 of 3

OPERATOR	MEANING
>>	reads a whitespace-delimited string into a string object
<<	inserts a string object into a stream
=	assigns string on right to string object on left
+=	appends string on the right to the end of contents of string on left

Overloaded `string` Operators 2 of 3

OPERATOR	MEANING
`+`	Returns concatenation of the two strings
`[]`	references character in string using array notation
`>, >=, <, <=, ==, !=`	relational operators for string comparison. Return **`true`** or **`false`**

Overloaded `string` Operators 3 of 3

```
string word1, phrase;
string word2 = " Dog";
cin >> word1; // user enters "Hot"
              // word1 has "Hot"
phrase = word1 + word2; // phrase has
                        // "Hot Dog"
phrase += " on a bun";
for (int i = 0; i < 16; i++)
   cout << phrase[i];  // displays
                       // "Hot Dog on a bun"
```

`string` Member Functions

Categories:

- conversion to C-strings: `c_str`, `data`
- modification: `append`, `assign`, `clear`, `copy`, `erase`, `insert`, `replace`, `swap`
- space management: `capacity`, `empty`, `length`, `size`
- substrings: `find`, `substr`
- comparison: `compare`

Hands-On Pg. 856

- Listing 12-16

Conversion to C-strings

- `data()` and `c_str()` both return the C-string equivalent of a **`string`** object
- Useful when using a string object with a function that is expecting a C-string

```
char greeting[20] = "Have a ";
string str("nice day");
strcat(greeting, str.data());
```

Modification of `string` objects 1 of 2

- `str.append(string s)`

  appends contents of `s` to end of `str`

- Convert constructor for **`string`** allows a C-string to be passed in place of `s`

```
string str("Have a ");
str.append("nice day");
```

- `append` is overloaded for flexibility

Modification of `string` objects 2 of 2

- `str.insert(int pos, string s)`

  inserts `s` at position `pos` in `str`

- Convert constructor for **`string`** allows a C-string to be passed in place of `s`

```
string str("Have a day");
str.insert(7, "nice ");
```

- `insert` is overloaded for flexibility

Copyright