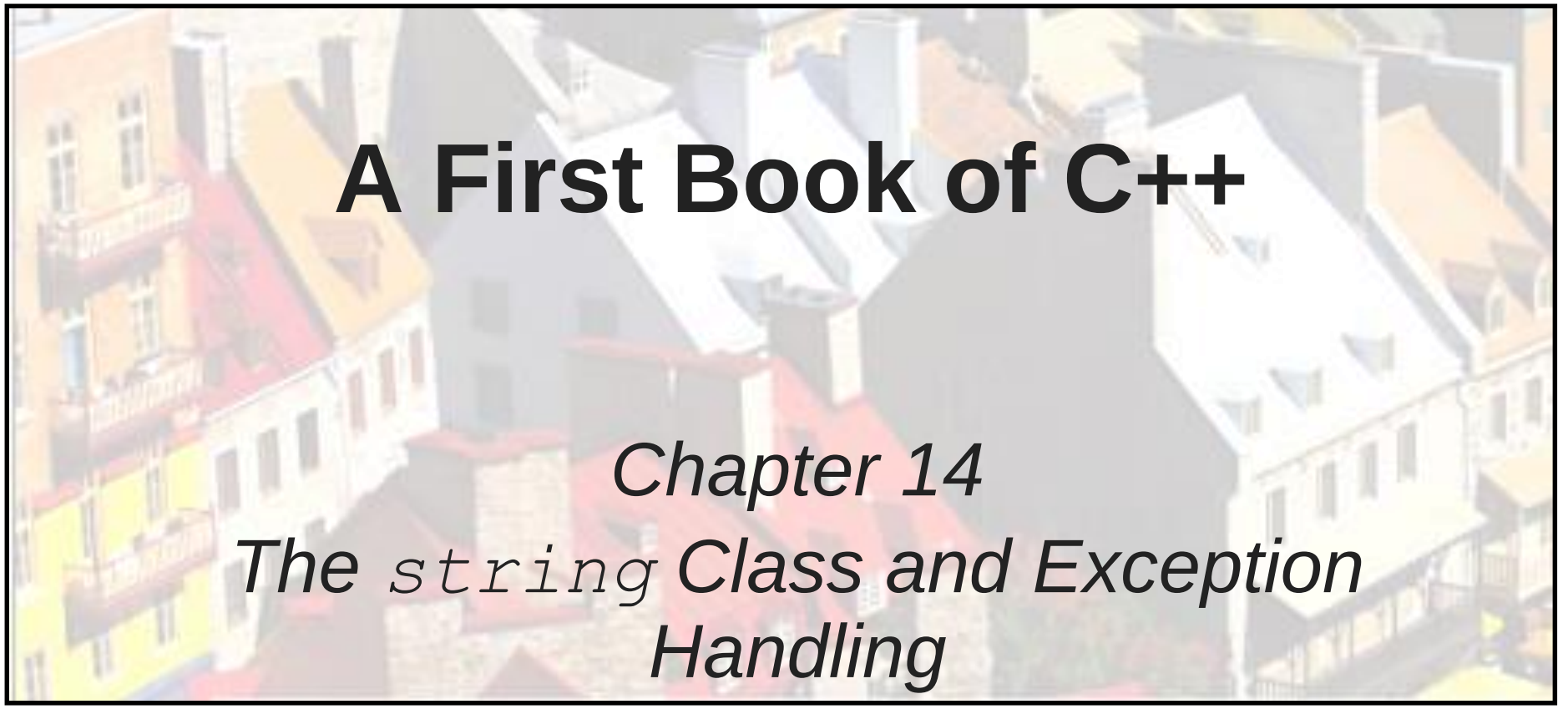

A First Book of C++

Chapter 14

The `string` Class and Exception Handling

Objectives

- In this chapter, you will learn about:
  - The `string` Class
  - Character Manipulation Methods
  - Exception Handling
  - Exceptions and File Checking
  - Input Data Validation
  - Common Programming Errors
  - Namespaces

The `string` Class

- Provides methods for declaring, creating, and initializing a string
- **String literal**: any sequence of characters enclosed in quotation marks
- Examples:
  - `“This is a string”`
  - `“Hello World!”`
- Quotation marks identify the beginning and end of a string
  - Quotation marks are not stored with string

The `string` Class (cont'd.)

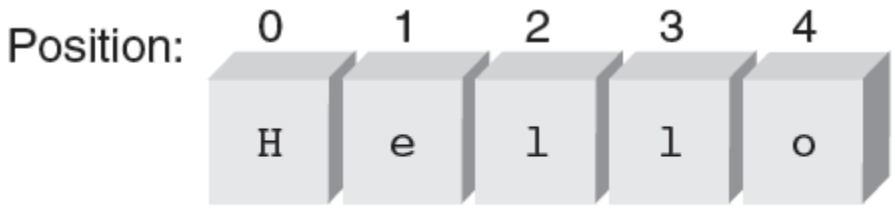

Figure 14.1 Storing a string as a sequence of characters

`string` Class Functions

Table 14.1 `string` Class Constructors (Require the Header File `string`)

Constructor	Description	Examples

`string objectName = value`	Creates and initializes a string object to a value that can be a string literal, a previously declared string object, or an expression containing string literals and string objects	`string str1 = "Good Morning";`
		`string str2 = str1;`
		`string str3 = str1 + str2;`

`string objectName(stringValue)`	Produces the same initialization as the preceding item	`string str1("Hot");` `string str1(str1 + " Dog");`
`string objectName(str, n)`	Creates and initializes a string object with a substring of string object `str`, starting at index position *n* of `str`	`string str1(str2, 5);` If `str2` contains the string `Good Morning`, `str1` becomes the string `Morning`
`string objectName(str, n, p)`	Creates and initializes a string object with a substring of string object `str`, starting at index position *n* of `str` and containing *p* characters	`string str1(str2, 5,2);` If `str2` contains the string `Good Morning`, `str1` becomes the string `Mo`
`string objectName(n, char)`	Creates and initializes a string object with *n* copies of `char`	`string str1(5,'*');` This makes `str1 = "*****"`
`string objectName`	Creates and initializes a string object to represent an empty character sequence (same as `string objectName = "";`, so the string's length is 0)	`string message;`

`string` Class Functions (cont'd.)

- String creation
  - Example: Program 14.1

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string str1; // an empty string
  string str2("Good Morning");
  string str3 = "Hot Dog";
  string str4(str3);
  string str5(str4, 4);
  string str6 = "linear";
  string str7(str6, 3, 3);
```

`string` Class Functions (cont'd.)

– Example: Program 14.1 (cont'd):

```
    cout << "str1 is: " << str1 << endl;
    cout << "str2 is: " << str2 << endl;
    cout << "str3 is: " << str3 << endl;
    cout << "str4 is: " << str4 << endl;
    cout << "str5 is: " << str5 << endl;
    cout << "str6 is: " << str6 << endl;
    cout << "str7 is: " << str7 << endl;

    return 0;
}
```

`string` Class Functions (cont'd.)

- Output created by Program 14.1:

```
str1 is:
str2 is: Good Morning
str3 is: Hot Dog
str4 is: Hot Dog
str5 is: Dog
str6 is: linear
str7 is: ear
```

`string` Input and Output

- In addition to methods listed in Table 14.1, strings can be:
  - Input from the keyboard
  - Displayed on the screen
- Additional methods include:
  - **`cout`**: general-purpose screen output
  - **`cin`**: general-purpose terminal input that stops reading when a whitespace is encountered

`string` Input and Output (cont'd.)

- Additional methods include:
  - **`getline(cin, strObj)`**: general-purpose terminal input that inputs all characters entered into the string named `strObj` and stops accepting characters when it receives a newline character (`\n`)
  - Example: `getline(cin, message)`
    - Continuously accepts and stores characters entered at terminal until Enter key is pressed
      - Pressing Enter key generates newline character, `'\n'`
      - All characters except newline are stored in string named `message`

`string` Input and Output (cont'd.)

Program 14.2

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string message;    // declare a string object

  cout << "Enter a string:\n";

  getline(cin, message);

  cout << "The string just entered is:\n"
       << message << endl;

  return 0;
}
```

`string` Input and Output (cont'd.)

- Sample run of Program 14.2:

```
Enter a string:
This is a test input of a string of
characters.
The string just entered is:
This is a test input of a string of
characters.
```

`string` Input and Output (cont'd.)

- In Program 14.2, the `cin` object cannot be used in place of `getline()`
- `cin` reads a set of characters up to a blank space or a newline character
- Statement `cin >> message` cannot be used to enter the characters `This is a string`
  - Statement results in word `This` assigned to `message`
- `cin`'s usefulness for entering string data is limited; blank space terminates `cin` extraction

`string` Input and Output (cont'd.)

- General form of `getline()` method:

  `getline(cin, strObj, terminatingChar)`

  - `strObj`: a string variable name
  - `terminatingChar`: an optional character constant or variable specifying the terminating character
- Example:
  - `getline(cin, message, '!')`
    - Accepts all characters entered at the keyboard, including newline, until an exclamation point is entered

`string` Input and Output (cont'd.)

- Unexpected results occur when:
  - `cin` input stream and `getline()` method are used together to accept data
  - Or when `cin` input stream is used to accept individual characters
- Example: Program 14.3
  - When `value` is entered and Enter key is pressed, `cin` accepts `value` but leaves the `'\n'` in the buffer
  - `getline()` picks up the code for the Enter key as the next character and terminates further input

`string` Input and Output (cont'd.)

Program 14.3

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  int value;
  string message;

  cout << "Enter a number: ";
  cin  >> value;
  cout << "The number entered is:\n"
       << value << endl;

  cout << "Enter text:\n";
  getline(cin, message);
  cout << "The text entered is:\n"
       << message << endl;
  cout << int(message.length());

  return 0;
}
```

`string` Input and Output (cont'd.)

- Sample run of Program 14.3:

```
Enter a number: 26
The number entered is 26
Enter text:
The string entered is
```

`string` Input and Output (cont'd.)

- Solutions to the “phantom” Enter key problem
  - Do not mix `cin` with `getline()` inputs in the same program
  - Follow the `cin` input with the call to `cin.ignore()`
  - Accept the Enter key into a character variable and then ignore it
- Preferred solution is the first option

String Processing

- Methods for manipulating strings (Table 14.3):
  - Most commonly used `string` class method is `length()`, which returns the number of characters in the string
- Most commonly used methods:
  - Accessor
  - Mutator
  - Additional methods that use standard arithmetic and comparison operators

String Processing (cont'd.)

- String expressions may be compared for equality using standard relational operators
- String characters are stored in binary using ASCII or Unicode code as follows:
  - A blank precedes (is less than) all letters and numbers
  - Letters are stored in order from A to Z
  - Digits are stored in order from 0 to 9
  - Digits come before uppercase characters, which are followed by lowercase characters

String Processing (cont'd.)

- Procedure for comparing strings:
  - Individual characters compared a pair at a time
    - If no differences, the strings are equal
    - Otherwise, the string with the first lower character is considered the smaller string
- Examples:
  - `"Hello"` is greater than `"Good Bye"` because the first `H` in `Hello` is greater than the first `G` in `Good Bye`
  - `"Hello"` is less than `"hello"` because the first `H` in `Hello` is less than the first `h` in `hello`

Character Manipulation Methods

- C++ language provides a variety of character class functions (listed in Table 14.4)
- Function declarations (prototypes) for these functions are contained in header files `string` and `cctype`
- Header file must be included in any program that uses these functions

Character Manipulation Methods (cont'd.)

- Example: If `ch` is a character variable, consider the following code segment

```
if(isdigit(ch))
cout << "The character just entered is a digit"
  << endl;
else if(ispunct(ch))
cout << "The character just entered is a
  punctuation mark" << endl;
```

  - If `ch` contains a digit character, the first `cout` statement is executed
  - If `ch` is a letter, the second statement is executed

Character I/O

- Entry of all data from keyboard, whether a string or a number, is done one character at a time
  - Entry of string `Hello` consists of pressing keys `H`, `e`, `l`, `l`, `o`, and the Enter Key (as in Figure 14.10)
- All of C++'s higher-level I/O methods and streams are based on lower-level character I/O

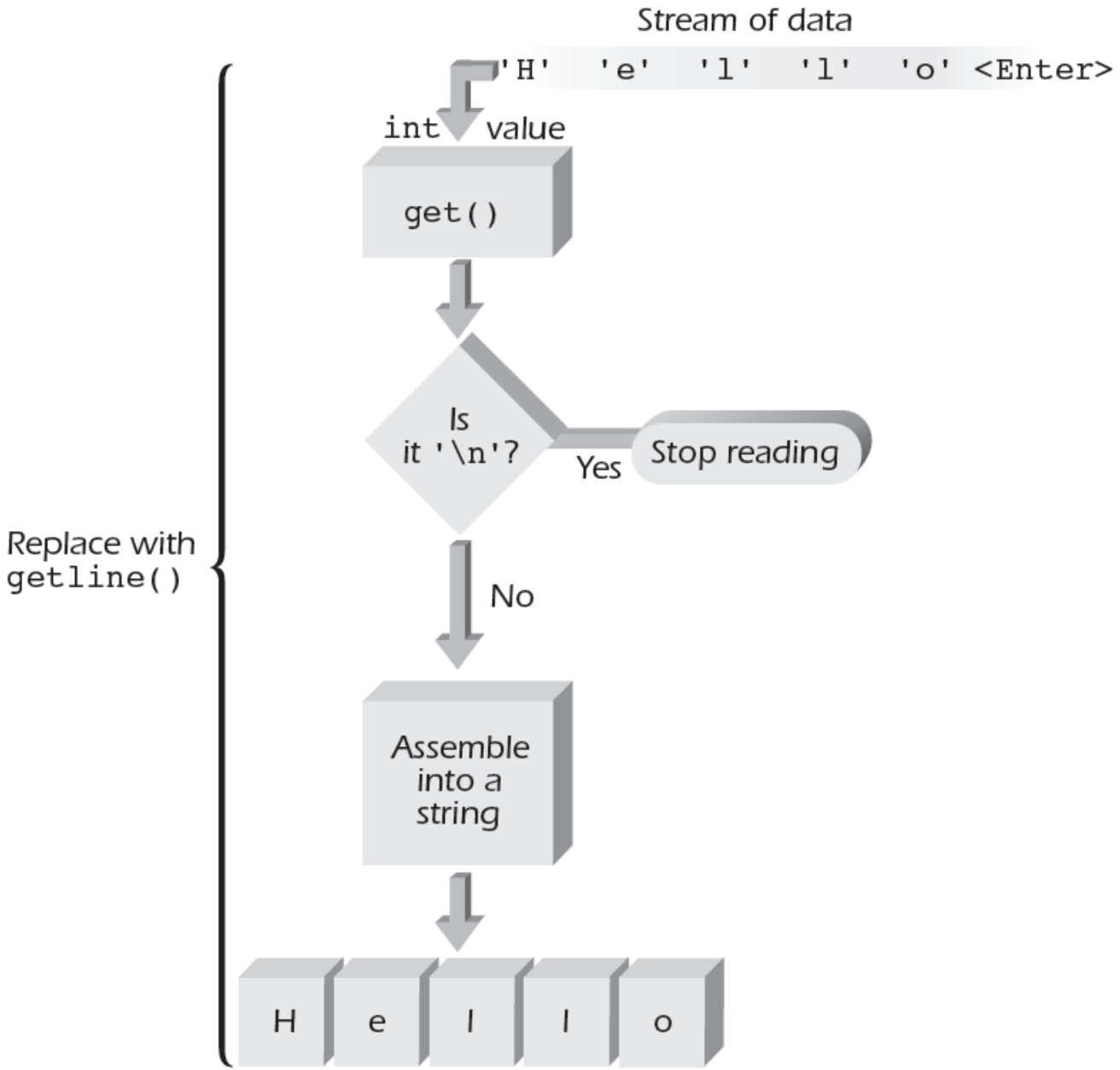

Figure 14.10 Accepting keyboard-entered characters

Character I/O (cont'd.)

- Undesired results can occur when characters are input using the `get()` character method
  - Program 14.10 is an example of this problem
- Two ways to avoid this:
  - Follow `cin.get()` input with the call `cin.ignore()`
  - Accept the Enter key into a character variable and then don't use it further
- Program 14.12 applies the first solution to Program 14.11

A Second Look at User-Input Validation

- Robust (bulletproof) program: responds effectively to unexpected user input
- User-input validation: code incorporated into a well-constructed program that validates user input and avoids unexpected results
  - Must check each entered character to verify that it qualifies as a legitimate character for the expected data type

Exception Handling

- Traditional approach: a function returns a specific value to specific operations
  - Example:
    - Return value of `0` or `1` to indicate successful completion
    - Negative return value indicates error condition
- Problems associated with this approach
  - Programmer must check return value
  - Return value checking becomes confused with normal processing code

Exception Handling (cont'd.)

Table 14.6 Exception-Handling Terminology

Terminology	Description
Exception	A value, a variable, or an object that identifies a specific error that has occurred while a program is running
Throw an exception	Send the exception to a section of code that processes the detected error
Catch or handle an exception	Receive a thrown exception and process it
Catch clause	The section of code that processes the error
Exception handler	The code that throws and catches an exception

Exception Handling (cont'd.)

- The general syntax of the code required to throw and catch an exception:

```
try
{
   // one or more statements,
   // at least one of which should
   // be capable of throwing an exception;
}
catch(exceptionDataType parameterName)
{
    // one or more statements
}
```

Exceptions and File Checking

- Error detection and exception handling are used in C++ programs that access one or more files
- General exception-handling code (section 14.3)

```
try
{
  // one or more statements, at least one
  // of which should throw an exception
}
catch(exceptionDataType parameterName)
{
  // one or more statements
}
```

- Program 14.15 illustrates file opening exception handling

Opening Multiple Files

- Example: Read the data from character-based file named `info.txt`, one character at a time, and write this data to file named `backup.txt`
  - Essentially, this is a file-copy program
- Figure 14.12 illustrates the structure of streams needed to produce file copy
- Program 14.17 creates `info.bak` file as an exact duplicate of `info.txt` file using the procedure described in Figure 15.4

Opening Multiple Files (cont'd.)

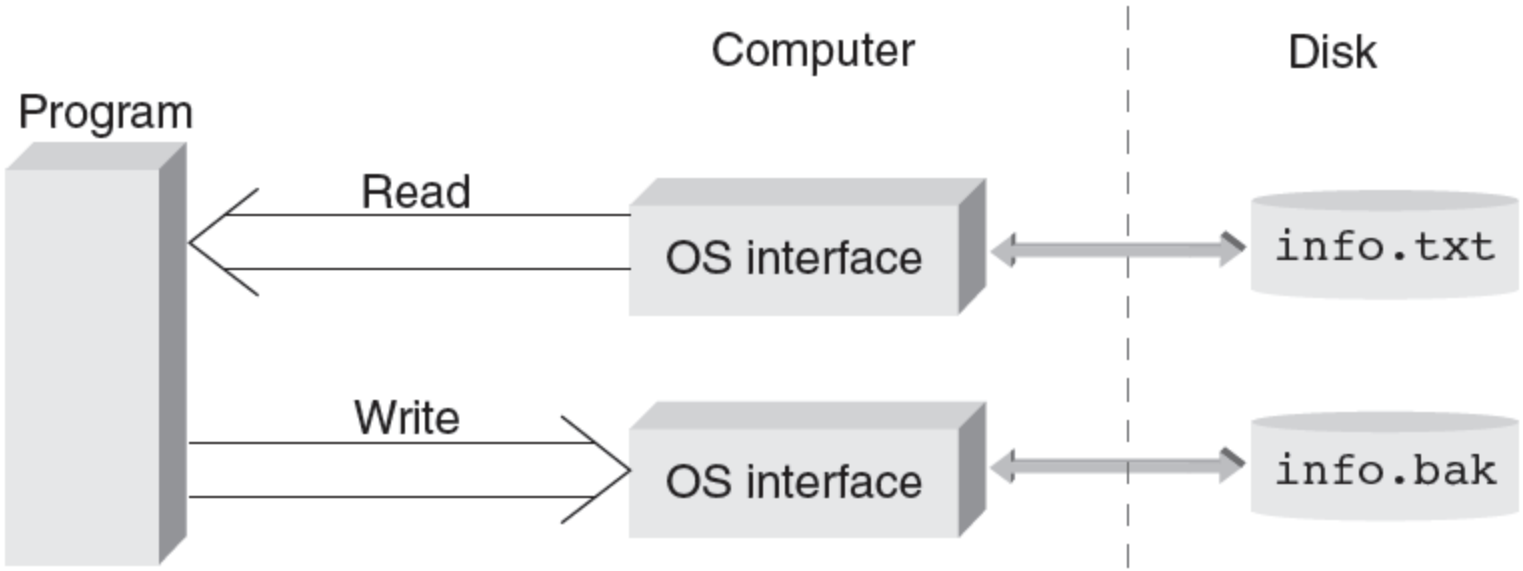

Figure 14.12 The file copy stream structure

Input Data Validation

- Major use of strings is user-input validation
- Common method of validating numerical input is to accept all numbers as strings
  - Each character in string can be checked to ensure it complies with the requested data type
  - After the data is checked and verified for the correct type, the string is converted to either an integer or floating-point value
    - Conversion is accomplished by functions in Table 14.7

Input Data Validation (cont'd.)

Table 14.7 C-String Conversion Functions

Function	Description	Example
`int atoi(stringExp)`	Converts `stringExp` to an integer. Conversion stops at the first non-integer character.	`atoi("1234")`
`double atof(stringExp)`	Converts `stringExp` to a double-precision number. Conversion stops at the first character that can't be interpreted as a `double`.	`atof("12.34")`
`char[] itoa(integerExp)`	Converts `integerExp` to a character array. The space allocated for the returned characters must be large enough for the converted value.	`itoa(1234)`

Common Programming Errors

- The common errors associated with defining and processing strings are:
  - Forgetting to include the `string` header file when using string class objects
  - Forgetting that the newline character, `'\n'`, is a valid data input character
  - Forgetting to convert a `string` class object using the `c_str()` method when converting `string` class objects to numerical data types

Common Programming Errors (cont'd.)

- The common errors associated with defining and processing strings are: (cont'd.)
  - Not defining a `catch` block with the correct parameter data type for each thrown exception
  - Attempting to declare an exception parameter in a `catch` block as a `string` class variable

Summary

- String literal (`string`, `string` value, `string` constant): any sequence of characters enclosed in double quotation marks
- A string can be constructed as an object of the `string` class
- `string` class is commonly used to construct strings for input and output:
  - Prompts and displayed messages

Summary (cont'd.)

- Other `string` class uses:
  - When strings need to be compared or searched or individual characters in a string need to be examined or extracted as a substring
  - When characters in a string need to be replaced, inserted, or deleted on a relatively regular basis
- Strings can be manipulated by:
  - Methods of the class they are objects of
  - General-purpose string and character methods

Summary (cont'd.)

- The `cin` object, by itself, tends to be of limited usefulness for string input because it terminates input when a blank is encountered
- For `string` class data input, use the `getline()` method
- The `cout` object can be used to display `string` class strings
- In exception handling, information about the error that caused the exception is sent to an exception handler

Chapter Supplement: Namespaces and Creating a Personal Library

- C++ provides mechanisms for programmers to build libraries of specialized functions and classes
- Steps in creating a library:
  - Encapsulate all of the specialized functions and classes into one or more *namespace*s
  - Store the complete code in one or more files
  - *namespace* syntax:

```
namespace name
{
  functions and/or classes in here
} // end of namespace
```

Chapter Supplement: Namespaces and Creating a Personal Library (cont'd.)

- After a *namespace* has been created and stored in a file, it can be included within another file
  - Supply a preprocessor directive informing the compiler where the *namespace* is to be found
  - Include a `using` directive instructing the compiler which particular *namespace* in the file to use
- Example:

```
#include <c:\\myLibrary\\dataChecks.cpp>
using namespace dataChecks;
```