Objectives

• In this chapter, you will learn about:
  – SKIP 9.1 & 9.2
  – The `string` Class
  – Character manipulation functions
  – Input data validation
  – SKIP 9.6 (Namespaces and creating a personal library)
  – Common programming errors
The `std::string` Class

- **The `std::string` class** permits string literal values
- **String literal**: Any sequence of characters enclosed in quotation marks
- By convention, first character in string is always designated as position **zero**
- This position value is also referred to as both the characters **index value** and its **offset value**
The **string** Class (continued)

**Figure 9.3** The storage of a string as a sequence of characters
string Class Functions

- **string class** provides a number of functions for declaring, creating, and initializing a string
- In earlier versions of C++, process of creating a new object is referred to as **instantiating an object**
- The methods that perform the tasks of creating and initializing are called **constructor methods**, or **constructors** for short
# Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>string objectName = value</td>
<td>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</td>
<td>string str1 = &quot;Good Morning&quot;; string str2 = str1; string str3 = str1 + str2;</td>
</tr>
<tr>
<td>string objectName(stringValue)</td>
<td>Produces the same initialization as the preceding item</td>
<td>string str1(&quot;Hot&quot;); string str1(str1 + &quot;Dog&quot;);</td>
</tr>
<tr>
<td>string objectName(str, n)</td>
<td>Creates and initializes a string object with a substring of string object str, starting at index position n of str</td>
<td>string str1(str2, 5) If str2 contains the string Good Morning, then str1 becomes the string Morning</td>
</tr>
<tr>
<td>string objectName(str, n, p)</td>
<td>Creates and initializes a string object with a substring of string object str, starting at index position n of str and containing p characters</td>
<td>string str1(str2, 5, 2) If str2 contains the string Good Morning, then str1 becomes the string Mo</td>
</tr>
<tr>
<td>string objectName(n, char)</td>
<td>Creates and initializes a string object with n copies of char</td>
<td>string str1(5,'*') This makes str1 = &quot;*****&quot;</td>
</tr>
</tbody>
</table>
**string Class Functions (continued)**

*Figure 9.4* The character positions of the string *Hot Dog*
String Input and Output

• In addition to string being initialized with constructors listed in Table 9.2, strings can be input from the keyboard and displayed on the screen
String Input and Output (continued)

<table>
<thead>
<tr>
<th>C++ Object or Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cout</code></td>
<td>General-purpose screen output object</td>
</tr>
<tr>
<td><code>cin</code></td>
<td>General-purpose keyboard input object that stops reading string input when white space is encountered</td>
</tr>
<tr>
<td><code>getline(cin, strObj)</code></td>
<td>General-purpose keyboard input function that inputs all characters entered, stores them in the string <code>strObj</code>, and stops accepting characters when it receives a newline character (\n)</td>
</tr>
</tbody>
</table>

**Table 9.3** string Class Input and Output
String Input and Output (continued)

Figure 9.5 Inputting a string with getline()
String Input and Output (continued)

Program 9.7

```cpp
#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;
}
```
Caution: The Phantom Newline Character

• Seemingly strange results can happen when \texttt{cin} input stream and \texttt{getline()} function are used together to accept data, or when \texttt{cin} is used by itself to accept characters

• \underline{Don’t mix} \texttt{cin} with \texttt{getline()} inputs in same program (preferred solution)

• Follow \texttt{cin} input with a call to \texttt{cin.ignore()}

• Accept Enter key in a character variable and then ignore it
Caution: The Phantom Newline Character (continued)

Program 9.8

#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;
}
Caution: The Phantom Newline Character (continued)

Each character is sent to a buffer as it's typed

Figure 9.6 Typed characters are first stored in a buffer
String Processing

• Strings can be manipulated using the `string` class functions or the character-at-a-time functions described in Section 9.4

• Most commonly used function in Table 9.4 is `length()`, which returns number of characters in the string

• Two string expressions can be compared for equality using the standard relational operators
# Table 9.4: String class functions

<table>
<thead>
<tr>
<th>Function/Operation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int length()</td>
<td>Returns the length of the string</td>
<td>string.length()</td>
</tr>
<tr>
<td>int size()</td>
<td>Same as the preceding item</td>
<td>string.size()</td>
</tr>
<tr>
<td>at(int index)</td>
<td>Returns the character at the specified index and throws an exception if the index is nonexistent</td>
<td>string.at(4)</td>
</tr>
<tr>
<td>int compare(str)</td>
<td>Compares the given string to str; returns a negative value if the given string is less than str, a 0 if they are equal, and a positive value if the given string is greater than str</td>
<td>string1.compare(string2)</td>
</tr>
<tr>
<td>c_str()</td>
<td>Returns the string as a null-terminated C-string</td>
<td>string1.c_str()</td>
</tr>
<tr>
<td>bool empty</td>
<td>Returns true if the string is empty; otherwise, returns false</td>
<td>string1.empty()</td>
</tr>
<tr>
<td>Function/Operation</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>erase(ind,n)</td>
<td>Removes n characters from the string, starting at index ind</td>
<td>string1.erase(2,3)</td>
</tr>
<tr>
<td>erase(ind)</td>
<td>Removes all characters from the string, starting from index ind until the end of the string, and the length of the remaining string becomes ind</td>
<td>string1.erase(4)</td>
</tr>
<tr>
<td>int find(str)</td>
<td>Returns the index of the first occurrence of str in the complete string</td>
<td>string1.find(&quot;the&quot;)</td>
</tr>
</tbody>
</table>
**Table 9.4: String class functions**

<table>
<thead>
<tr>
<th>Function/Operation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>void insert(ind, str)</td>
<td>Inserts the string str into the complete string, starting at index ind</td>
<td>string.insert(4, &quot;there&quot;)</td>
</tr>
<tr>
<td>void replace(ind, n, str)</td>
<td>Removes n characters in the string object, starting at index position ind, and inserts the string str at index position ind</td>
<td>string1.replace(2,4,&quot;okay&quot;)</td>
</tr>
<tr>
<td>string substr(ind,n)</td>
<td>Returns a string consisting of n characters extracted from the string, starting at index ind; if n is greater than the remaining number of characters, the rest of the string is used</td>
<td>string2 = string1.substr(0,10)</td>
</tr>
</tbody>
</table>
String Processing (continued)

**Figure 9.7** The initial strings used in Program 9.9
String Processing (continued)

Figure 9.8 Initial storage of a string object

Figure 9.9 The string after the insertion
String Processing (continued)

**Figure 9.10** The string after the replacement

**Figure 9.11** The string after the append
CSc 10200
Introduction to Computing

Lecture 17
Edgardo Molina
Fall 2011 – City College of New York
NOTE ABOUT BOOK

• Some example show function prototypes inside of the main() function

• THIS IS WRONG; DON’T DO IT!
Character Manipulation Functions

- In addition to string functions provided by `std::string` class, the C++ language provides several useful character class functions.
- Function declaration (prototype) for each function is contained in the header file `string` or `cctype`, which must be included in any program using these functions.
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int i;
    string str;

    cout << "Type in any sequence of characters: ";
    getline(cin, str);

    // cycle through all elements of the string
    for (i = 0; i < int(str.length()); i++)
        str[i] = toupper(str[i]);

    cout << "The characters just entered, in uppercase, are: "
        << str << endl;

    cin.ignore();
    return 0;
}
### Character Manipulation Functions (continued)

<table>
<thead>
<tr>
<th>Function Prototype</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int isalpha(charExp)</td>
<td>Returns a true (non-zero integer) if charExp evaluates to a letter; otherwise, it returns a false (zero integer)</td>
<td>isalpha('a')</td>
</tr>
<tr>
<td>int isalnum(charExp)</td>
<td>Returns a true (non-zero integer) if charExp evaluates to a letter or a digit; otherwise, it returns a false (zero integer)</td>
<td>char key; cin &gt;&gt; key; isalnum(key);</td>
</tr>
<tr>
<td>int isupper(charExp)</td>
<td>Returns a true (non-zero integer) if charExp evaluates to an uppercase letter; otherwise, it returns a false (zero integer)</td>
<td>isupper('a')</td>
</tr>
<tr>
<td>int islower(charExp)</td>
<td>Returns a true (non-zero integer) if charExp evaluates to a lowercase letter; otherwise, it returns a false (zero integer)</td>
<td>islower('a')</td>
</tr>
<tr>
<td>int isdigit(charExp)</td>
<td>Returns a true (non-zero integer) if charExp evaluates to a digit (0 through 9); otherwise, it returns a false (zero integer)</td>
<td>isdigit('a')</td>
</tr>
</tbody>
</table>
Character Manipulation Functions (continued)

- int isascii(charExp)
- int isspace(charExp)
- int isprint(charExp)
- int isctrl(charExp)
- int ispunct(charExp)
- int isgraph(charExp)

- More documentation:
```cpp
#include <iostream>
using namespace std;

int main() {
    string str = "This -123/ is 567 A ?<6245> Test!";
    char nextChar;
    int i;
    int numLetters = 0, numDigits = 0, numOthers = 0;

    cout << "The original string is : " << str
    << "\nThis string contains " << int(str.length())
    << " characters, which consist of" << endl;

    //check each character in the string
    for (i = 0; i < str.length(); i++)
    {
        nextChar = str.at(i);
        if (isalpha(nextChar))
            numLetters++;
        else if (isdigit(nextChar))
            numDigits++;
        else
            numOthers++;
    }

    cout << "\t" << numLetters << " letters" << endl;
    cout << "\t" << numDigits << " digits" << endl;
    cout << "\t" << numOthers << " other characters." << endl;

    return 0;
}
```
# Character I/O

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cout.put(charExp)</code></td>
<td>Places the character value of <code>charExp</code> on the output stream.</td>
<td><code>cout.put('A');</code></td>
</tr>
<tr>
<td><code>cin.get(charVar)</code></td>
<td>Extracts the next character from the input stream and assigns it to the variable <code>charVar</code>.</td>
<td><code>cin.get(key);</code></td>
</tr>
<tr>
<td><code>cin.peek(charVar)</code></td>
<td>Assigns the next character from the input stream to the variable `charVar without extracting the character from the stream.</td>
<td><code>cin.peek(nextKey);</code></td>
</tr>
<tr>
<td><code>cin.putback(charExp)</code></td>
<td>Pushes a character value of <code>charExp</code> back onto the input stream.</td>
<td><code>cin.putback(cKey);</code></td>
</tr>
</tbody>
</table>

**Table 9.6** Basic Character I/O Functions (Require the header file `cctype`)
Character I/O (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cin.ignore(n, char)</code></td>
<td>Ignores a maximum of the next $n$ input characters, up to and including the detection of <code>char</code>. If no arguments are specified, ignores the next single character on the input stream.</td>
<td><code>cin.ignore(80, '\n');</code>&lt;br&gt;<code>cin.ignore();</code></td>
</tr>
</tbody>
</table>
Character I/O (continued)

Stream of data

'H' 'e' 'l' 'l' 'o' <Enter>

int value

get()

Is it '\n'?

Yes Stop reading

No

Assemble into a string

Hello

Replace with getline()
Phantom Newline Character Revisited

Program 9.15

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

int main()
{
    char fkey;

    cout << "Type in a character: ";
    cin.get(fkey);
    cout << "The key just accepted is " << int(fkey) << endl;

    return 0;
}
```
Phantom Newline Character Revisited (continued)

Program 9.16

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

int main()
{
    char fkey, skey;

    cout << "Type in a character: ";
    cin.get(fkey);
    cout << "The key just accepted is " << int(fkey) << endl;

    cout << "Type in another character: ";
    cin.get(skey);
    cout << "The key just accepted is " << int(skey) << endl;

    return 0;
}
```
Phantom Newline Character Revisited (continued)

Program 9.17

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

int main()
{
    char fkey, skey;

    cout << "Type in a character: ";
    cin.get(fkey);
    cout << "The key just accepted is " << int(fkey) << endl;
    cin.ignore();

    cout << "Type in another character: ";
    cin.get(skey);
    cout << "The key just accepted is " << int(skey) << endl;

    cin.ignore();
    return 0;
}
```
A Second Look at User-Input Validation

• Sign of well-constructed, robust program:
  – *Code that validates user input* and ensures that the program doesn’t produce unintended results caused by unexpected input

• **User-input validation**: Basic technique for handling invalid data input and preventing seemingly innocuous code from producing unintended results
  – Validates entered data during or after data entry and gives the user a way of reentering data if it is invalid
Input Data Validation

• Validating user input is essential
• Successful programs anticipate invalid data and prevent it from being accepted and processed
• A common method for validating numerical input data is accepting all numbers as strings
• After string is validated it can be converted to the correct type
## Input Data Validation (continued)

Table 9.7  C-String Conversion Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int atoi(stringExp)</td>
<td>Converts stringExp to an integer. Conversion stops at the first non-integer character.</td>
<td>atoi(&quot;1234&quot;)</td>
</tr>
<tr>
<td>double atof(stringExp)</td>
<td>Converts stringExp to a double-precision number. Conversion stops at the first character that can’t be interpreted as a double.</td>
<td>atof(&quot;12.34&quot;)</td>
</tr>
<tr>
<td>char[] itoa(integerExp)</td>
<td>Converts integerExp to a character array. The space allocated for the returned characters must be large enough for the converted value.</td>
<td>itoa(1234)</td>
</tr>
</tbody>
</table>
Program 9.18

• ...

C++ for Engineers and Scientists, Third Edition
Common Programming Errors

- Forgetting to include `string` header file when using `string` class object
- Forgetting that newline character, `\n`, is a valid input character
- Forgetting to convert `string` class object by using `c_str()` function when converting `string` class objects to numerical data types
Chapter Summary

- String literal is any sequence of characters enclosed in quotation marks
  - Referred to as a string value, a string constant, and, more conventionally, a string
- String can be constructed as an object of the string class
- string class is commonly used for constructing strings for input and output purposes
- Strings can be manipulated by using functions of the class they’re objects of or by using the general purpose string and character functions
Chapter Summary (continued)

• `cin` object 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()` function
• `cout` object can be used to display `string` class strings