Chapter 9

Strings
C-Strings vs C++ Strings

- Two string types:
  - C-strings
    - Array with base type char
    - End of string marked with null, ‘\0’
    - Older method inherited from C
  - C++ strings
    - Objects of string class
C-Strings

• Array with base type *char*
  – One character per indexed variable
  – One extra character: ‘\0’
    • Called "null character"
    • End marker

• We’ve used c-strings
  – Literal "Hello" stored as c-string
C-Strings

- Array of characters: `char s[10];`
  - Declares a c-string variable to hold up to 9 characters
  - And one null character
- Typically "partially-filled" array
  - Declare large enough to hold max-size string
  - Indicate end with null
- Only difference from standard character array:
  - Must contain null character

```
  H     i     M     o     m     !     \0     ?     ?
```
Initializing C-Strings

- Initialize c-string: `char myMessage[20] = "Hi there.";`
  - Needn’t fill entire array
  - Initialization places ‘\0’ at end

- Can omit array-size: `char shortString[] = "abc";`
  - Automatically makes size one more than length of quoted string
  - NOT same as: `char shortString[] = {'a', 'b', 'c'};`
C-String Index Manipulation

• Can manipulate indexed variables
  char happyString[] = "DoBeDo";
  – Be careful!
  – Here ‘\0’ (null) is overwritten by a ‘Z’

• If null is overwritten, it no longer acts like c-string
  – Becomes a regular char array
**<cstring> Library**

- Declaring c-strings don’t require a library
- Manipulations require library `<cstring>`

### Display 9.1  Some Predefined C-String Functions in `<cstring>`

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>CAUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcpy(Target_String_Var, Src_String)</code></td>
<td>Copies the C-string value <code>Src_String</code> into the C-string variable <code>Target_String_Var</code>.</td>
<td>Does not check to make sure <code>Target_String_Var</code> is large enough to hold the value <code>Src_String</code>.</td>
</tr>
<tr>
<td><code>strcpy(Target_String_Var, Src_String, Limit)</code></td>
<td>The same as the two-argument <code>strcpy</code> except that at most <code>Limit</code> characters are copied.</td>
<td>If <code>Limit</code> is chosen carefully, this is safer than the two-argument version of <code>strcpy</code>. Not implemented in all versions of C++.</td>
</tr>
<tr>
<td><code>strcat(Target_String_Var, Src_String)</code></td>
<td>Concatenates the C-string value <code>Src_String</code> onto the end of the C-string in the C-string variable <code>Target_String_Var</code>.</td>
<td>Does not check to see that <code>Target_String_Var</code> is large enough to hold the result of the concatenation.</td>
</tr>
</tbody>
</table>

(continued)
Character Functions

- Found in `<cctype>` library

### Display 9.3 Some Functions in `<cctype>`

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toupper(Char_Exp)</code></td>
<td>Returns the uppercase version of <code>Char_Exp</code> (as a value of type <code>int</code>).</td>
<td><code>char c = toupper('a'); cout &lt;&lt; c; Outputs: A</code></td>
</tr>
<tr>
<td><code>tolower(Char_Exp)</code></td>
<td>Returns the lowercase version of <code>Char_Exp</code> (as a value of type <code>int</code>).</td>
<td><code>char c = tolower('A'); cout &lt;&lt; c; Outputs: a</code></td>
</tr>
<tr>
<td><code>isupper(Char_Exp)</code></td>
<td>Returns true provided <code>Char_Exp</code> is an uppercase letter; otherwise, returns false.</td>
<td><code>if (isupper(c)) cout &lt;&lt; &quot;Is uppercase.&quot;; else cout &lt;&lt; &quot;Is not uppercase.&quot;;</code></td>
</tr>
</tbody>
</table>
## Character Functions

### Display 9.3  Some Functions in `<cctype>`

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
</table>
| islower(Char_Exp) | Returns true provided Char_Exp is a lowercase letter; otherwise, returns false. | char c = 'a';  
if (islower(c))  
cout << c << " is lowercase.";  
Outputs: a is lowercase. |
| isalpha(Char_Exp) | Returns true provided Char_Exp is a letter of the alphabet; otherwise, returns false. | char c = '$';  
if (isalpha(c))  
cout << "Is a letter.";  
else  
cout << "Is not a letter.";  
Outputs: Is not a letter. |
| isdigit(Char_Exp) | Returns true provided Char_Exp is one of the digits '0' through '9'; otherwise, returns false. | if (isdigit('3'))  
cout << "It’s a digit.";  
else  
cout << "It’s not a digit.";  
Outputs: It’s a digit. |
| isalnum(Char_Exp) | Returns true provided Char_Exp is either a letter or a digit; otherwise, returns false. | if (isalnum('3') && isalnum('a'))  
cout << "Both alphanumeric.";  
else  
cout << "One or more are not.";  
Outputs: Both alphanumeric. |
# Character Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
</table>
| isspace    | Returns true provided `Char_Exp` is a whitespace character, such as the blank or newline character; otherwise, returns false. | `// Skips over one "word" and sets c equal to the first whitespace character after the "word":`  
`do`  
`{`  
`    cin.get(c);`  
`}`  
`while (! isspace(c));` |
| ispunct    | Returns true provided `Char_Exp` is a printing character other than whitespace, a digit, or a letter; otherwise, returns false. | `if (ispunct('?'))`  
`    cout << "Is punctuation.";`  
`else`  
`    cout << "Not punctuation.";` |
| isprint    | Returns true provided `Char_Exp` is a printing character; otherwise, returns false. |  |
| isgraph    | Returns true provided `Char_Exp` is a printing character other than whitespace; otherwise, returns false. |  |
| isctrl     | Returns true provided `Char_Exp` is a control character; otherwise, returns false. |  |
C++ Strings

• Defined in library:
  ```
  #include <string>
  using namespace std;
  ```

• String variables and expressions
  – Treated much like simple types

• Can assign, compare, add:
  ```
  string s1, s2, s3;
  s3 = s1 + s2;      //Concatenation
  s3 = "Hello Mom!"   //Assignment
  ```
  – Note c-string "Hello Mom!" automatically converted to string type!
String Example

Display 9.4  Program Using the Class string

1  //Demonstrates the standard class string.
2  #include <iostream>
3  #include <string>
4  using namespace std;

5  int main( )
6  {
7      string phrase;
8      string adjective("fried"), noun("ants");
9      string wish = "Bon appetit!";
10     phrase = "I love " + adjective + " " + noun + "!";
11     cout << phrase << endl
12       << wish << endl;
13     return 0;
14  }

SAMPLE DIALOGUE
I love fried ants!
Bon appetit!
String I/O

• Just like other types!
  
  ```cpp
  string s1, s2;
cin >> s1;
cin >> s2;
  ```

• If the user types in:
  “May the hair on your toes grow long and curly!”

• Extraction still ignores whitespace:
  
  s1 receives value "May"
  s2 receives value "the"
String I/O

• To read in entire lines use getline(cin, string):
  
```cpp
string line;
cout << "Enter a line of input: ";
getline(cin, line);
cout << line << "END OF OUTPUT";
```

• Dialogue produced:
  
```plaintext
Enter a line of input: Do be do to you!
Do be do to you!END OF OUTPUT
```
Pitfall: Mixing the extraction operator with getline

- Be careful mixing `cin >> var` and `getline`
  ```cpp
  int n;
  string line;
  cin >> n;
  getline(cin, line);
  ```

- If user enters:
  ```
  42
  Hello hitchhiker.
  ```
  - Variable n set to 42
  - line set to empty string

- `cin >> n` skips whitespace and leaves `\n` in stream for `getline()`
## String Manipulation

- `str.length()` - Returns length of string variable

### Display 9.7  Member Functions of the Standard Class string

<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constructors</strong></td>
<td></td>
</tr>
<tr>
<td><code>string str;</code></td>
<td>Default constructor; creates empty string object <code>str</code>.</td>
</tr>
<tr>
<td><code>string str(&quot;string&quot;);</code></td>
<td>Creates a string object with data &quot;string&quot;.</td>
</tr>
<tr>
<td><code>string str(aString);</code></td>
<td>Creates a string object <code>str</code> that is a copy of <code>aString</code>. <code>aString</code> is an object of the class string.</td>
</tr>
<tr>
<td><strong>Element access</strong></td>
<td></td>
</tr>
<tr>
<td><code>str[i]</code></td>
<td>Returns read/write reference to character in <code>str</code> at index <code>i</code>.</td>
</tr>
<tr>
<td><code>str.at(i)</code></td>
<td>Returns read/write reference to character in <code>str</code> at index <code>i</code>.</td>
</tr>
<tr>
<td><code>str.substr(position, length)</code></td>
<td>Returns the substring of the calling object starting at <code>position</code> and having <code>length</code> characters.</td>
</tr>
<tr>
<td><strong>Assignment/Modifiers</strong></td>
<td></td>
</tr>
<tr>
<td><code>str1 = str2;</code></td>
<td>Allocates space and initializes it to <code>str2</code>’s data, releases memory allocated for <code>str1</code>, and sets <code>str1</code>’s size to that of <code>str2</code>.</td>
</tr>
<tr>
<td><code>str1 += str2;</code></td>
<td>Character data of <code>str2</code> is concatenated to the end of <code>str1</code>; the size is set appropriately.</td>
</tr>
<tr>
<td><code>str.empty()</code></td>
<td>Returns true if <code>str</code> is an empty string; returns false otherwise.</td>
</tr>
</tbody>
</table>

(continued)
# String Manipulation

## Member Functions of the Standard Class `string`

<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str1 + str2</code></td>
<td>Returns a string that has <code>str2</code>'s data concatenated to the end of <code>str1</code>'s data. The size is set appropriately.</td>
</tr>
<tr>
<td><code>str.insert(pos, str2)</code></td>
<td>Inserts <code>str2</code> into <code>str</code> beginning at position <code>pos</code>.</td>
</tr>
<tr>
<td><code>str.remove(pos, length)</code></td>
<td>Removes substring of size <code>length</code>, starting at position <code>pos</code>.</td>
</tr>
<tr>
<td><strong>Comparisons</strong></td>
<td></td>
</tr>
<tr>
<td><code>str1 == str2</code></td>
<td><code>str1 != str2</code></td>
</tr>
<tr>
<td><code>str1 &lt; str2</code></td>
<td><code>str1 &gt; str2</code></td>
</tr>
<tr>
<td><code>str1 &lt;= str2</code></td>
<td><code>str1 &gt;= str2</code></td>
</tr>
<tr>
<td><code>str.find(str1)</code></td>
<td>Returns index of the first occurrence of <code>str1</code> in <code>str</code>.</td>
</tr>
<tr>
<td><code>str.find(str1, pos)</code></td>
<td>Returns index of the first occurrence of string <code>str1</code> in <code>str</code>; the search starts at position <code>pos</code>.</td>
</tr>
<tr>
<td><code>str.find_first_of(str1, pos)</code></td>
<td>Returns the index of the first instance in <code>str</code> of any character in <code>str1</code>, starting the search at position <code>pos</code>.</td>
</tr>
<tr>
<td><code>str.find_first_not_of(str1, pos)</code></td>
<td>Returns the index of the first instance in <code>str</code> of any character <code>not</code> in <code>str1</code>, starting search at position <code>pos</code>.</td>
</tr>
</tbody>
</table>
String Conversions

• Automatic type conversions
  – From c-string to c++ string object:
    ```c
    char aCString[] = "My C-string";
    string stringVar;
    stringVar = aCstring;
    
    • Perfectly legal and appropriate!
  – aCString = stringVar;
    • ILLEGAL!
    • Cannot auto-convert to c-string
  – Must use explicit conversion:
    ```c
    strcpy(aCString, stringVar.c_str());
    ```