Chapter 9

Strings
Learning Objectives

- An Array Type for Strings
  - C-Strings
  - C-String Manipulation Functions
  - C-String input and output
Introduction

- **C-strings**
  - Array with base type char
  - End of string marked with null, ‘\0’
  - ‘Older’ method inherited from C
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-String Variable

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

- A standard array:
  char s[10];
- If s contains string “Hi Mom”, stored as:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>i</td>
<td>M</td>
<td>o</td>
<td>m</td>
<td>!</td>
<td>\0</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
C-String Initialization

- Can initialize c-string:
  ```c
  char myMessage[20] = "Hi there.";
  ```
- Needn’t fill entire array
- Initialization places ‘\0’ at end
- Can omit array-size:
  ```c
  char shortString[] = "abc";
  ```
- Automatically makes size one more than length of quoted string
- NOT same as:
  ```c
  char shortString[] = {'a', 'b', 'c'};
  ```
C-String Indexes

- A c-string IS an array
- Can access indexed variables of:
  ```
  char ourString[5] = “Hi”;
  ```
  - ourString[0] is ‘H’
  - ourString[1] is ‘i’
  - ourString[2] is ‘\0’
  - ourString[3] is unknown
  - ourString[4] is unknown
C-String Index Manipulation

- Can manipulate indexed variables
  ```c
  char happyString[7] = "DoBeDo";
  ```
  - Be careful!
  - Here, '\0' (null) was overwritten by a 'Z'!

- If null overwritten, c-string no longer 'acts' like c-string!
  - Unpredictable results!
Library

- Declaring c-strings
  - Requires no C++ library
  - Built into standard C++
- Manipulations
  - Require library <cstring>
  - Typically included when using c-strings
    - Normally want to do ‘fun’ things with them
= and == with C-strings

- C-strings not like other variables
  - Cannot assign or compare:
    ```
    char aString[10];
    aString = "Hello";    // ILLEGAL!
    ```
  - Can ONLY use `=` at declaration of c-string!

- Must use library function for assignment:
  ```
  strcpy(aString, "Hello");
  ```
  - Built-in function (in `<cstring>`)  
  - Sets value of aString equal to "Hello"
  - NO checks for size!
  - Up to programmer, just like other arrays!
Comparing C-strings

- Also cannot use operator ==
  ```
  char aString[10] = "Hello";
  char anotherString[10] = "Goodbye";
  ```
- `aString == anotherString;   // NOT allowed!`

- Must use library function again:
  ```
  if (strcmp(aString, anotherString))
      cout << "Strings NOT same."
  else
      cout << "Strings are same."
  ```
The `<cstring>` Library

- Full of string manipulation functions

Display 9.1, page 357

<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>
<tr>
<td><code>strcat(Target_String_Var, Src_String, Limit)</code></td>
<td>The same as the two argument <code>strcat</code> except that at most <code>Limit</code> characters are appended.</td>
<td>If <code>Limit</code> is chosen carefully, this is safer than the two-argument version of <code>strcat</code>. Not implemented in all versions of C++.</td>
</tr>
</tbody>
</table>
Full of string manipulation functions

Display 9.1  Some Predefined C-String Functions in `<cstring>` (part 2 of 2)

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>CAUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strlen(Src_String)</code></td>
<td>Returns an integer equal to the length of <code>Src_String</code>. (The null character, '\0', is not counted in the length.)</td>
<td></td>
</tr>
<tr>
<td><code>strcmp(String_1, String_2)</code></td>
<td>Returns 0 if <code>String_1</code> and <code>String_2</code> are the same. Returns a value &lt; 0 if <code>String_1</code> is less than <code>String_2</code>. Returns a value &gt; 0 if <code>String_1</code> is greater than <code>String_2</code> (that is, returns a nonzero value if <code>String_1</code> and <code>String_2</code> are different). The order is lexicographic.</td>
<td>If <code>String_1</code> equals <code>String_2</code>, this function returns 0, which converts to <code>false</code>. Note that this is the reverse of what you might expect it to return when the strings are equal.</td>
</tr>
<tr>
<td><code>strncpy(String_1, String_2, Limit)</code></td>
<td>The same as the two-argument <code>strcat</code> except that at most <code>Limit</code> characters are compared.</td>
<td>If <code>Limit</code> is chosen carefully, this is safer than the two-argument version of <code>strcat</code>. Not implemented in all versions of C++.</td>
</tr>
</tbody>
</table>
C-string Functions: `strlen()`

- ‘String length’
- Often useful to know string length:
  ```c
  char myString[10] = “dobedo”;
  cout << strlen(myString);
  ```
  - Returns number of characters
    - Not including null
  - Result here:
    - 6
C-string Functions: `strcat()`

- `strcat()`
- ‘String concatenate’:
  ```c
  char stringVar[20] = "The rain";
  strcat(stringVar, "in Spain");
  ```
  - Note result:
    ```c
    stringVar now contains "The rain in Spain"
    ```
  - Be careful!
  - Incorporate spaces as needed!
C-string Arguments and Parameters

- Recall: c-string is array
- So c-string parameter is array parameter
  - C-strings passed to functions can be changed by receiving function!
- Like all arrays, typical to send size as well
  - Function ‘could’ also use ‘\0’ to find end
  - So size not necessary if function won’t change c-string parameter
- Use ‘const’ modifier to protect c-string arguments
C-String Output

- Can output with insertion operator, `<<`
- As we’ve been doing already: `cout << news << " Wow.\n";`
  - Where `news` is a c-string variable
- Possible because `<<` operator is overloaded for c-strings!
C-String Input

- Can input with extraction operator, `>>`
  - Issues exist, however
- Whitespace is ‘delimiter’
  - Tab, space, line breaks are ‘skipped’
  - Input reading ‘stops’ at delimiter
- Watch size of c-string
  - Must be large enough to hold entered string!
  - C++ gives no warnings of such issues!
C-String Input Example

- char a[80], b[80];
  cout << "Enter input: ";
  cin >> a >> b;
  cout << a << b << "END OF OUTPUT\n";
- Dialogue offered:
  Enter input:  Do be do to you!
  DobeEND OF OUTPUT
- Note: Underlined portion typed at keyboard
- C-string a receives: “do”
- C-string b receives: “be”
C-String Line Input

- Can receive entire line into c-string
- Use getline(), a predefined member function:
  ```c
  char a[80];
  cout << "Enter input: ";
  cin.getline(a, 80);
  cout << a << "END OF OUTPUT\n";
  ```
- Dialogue:
  Enter input:  Do be do to you!
  Do be do to you!END OF INPUT
More getline()

- Can explicitly tell length to receive:
  ```cpp
  char shortString[5];
  cout << "Enter input: ";
  cin.getline(shortString, 5);
  cout << shortString << "END OF OUTPUT\n";
  ```

- Results:
  Enter input: dobedowap
dobeEND OF OUTPUT

- Forces FOUR characters only be read
  - Recall need for null character!