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
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>

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

(continued)

Character Functions

- Found in <cctype> library

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

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

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
</table>
| islower(Char) | Returns true provided Char is a lowercase letter; otherwise, returns false. | char c = 'a';
                 | if (islower(c))
                 |         | cout << c << " is lowercase."
                 | Outputs: a is lowercase. |
| isalpha(Char) | Returns true provided Char 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) | Returns true provided Char 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) | Returns true provided Char 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

**isspace(Char)**  
Returns true provided Char is a whitespace character, such as the blank or newline character; otherwise, returns false.

```cpp
// Skips over one "word" and sets c equal to the first whitespace character after the "word":
do {
    cin.get(c);
} while (! isspace(c));
```

**ispunct(Char)**  
Returns true provided Char is a printing character other than whitespace, a digit, or a letter; otherwise, returns false.

```cpp
if (ispunct('?'))
    cout << "Is punctuation."
else
    cout << "Not punctuation."
```

**isprint(Char)**  
Returns true provided Char is a printing character; otherwise, returns false.

**isgraph(Char)**  
Returns true provided Char is a printing character other than whitespace; otherwise, returns false.

**isctrl(Char)**  
Returns true provided Char is a control character; otherwise, returns false.
C++ Strings

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

- String variables and expressions
  - Treated much like simple types

- Can assign, compare, add:
  ```cpp
  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

```cpp
1  //Demonstrates the standard class string.
2  #include <iostream>
3  #include <string>
4  using namespace std;
5  
6  int main( )
7  {
8       string phrase;
9       string adjective("fried"), noun("ants");
10      string wish = "Bon appetite!";
11      phrase = "I love " + adjective + " " + noun + "!";
12      cout << phrase << endl
13          << wish << endl;
14  }
```

**Sample Dialogue**

I love fried ants!
Bon appetite!
String I/O

• Just like other types!
  
  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):
  
  string line;
  cout << "Enter a line of input: ";
  getline(cin, line);
  cout << line << "END OF OUTPUT";

• Dialogue produced:
  
  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:
  ```cpp
  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
String Manipulation

Display 9.7  Member Functions of the Standard Class string

<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>str1 + str2</td>
<td>Returns a string that has str2’s data concatenated to the end of str1’s data. The size is set appropriately.</td>
</tr>
<tr>
<td>str.insert(pos, str2)</td>
<td>Inserts str2 into str beginning at position pos.</td>
</tr>
<tr>
<td>str.remove(pos, length)</td>
<td>Removes substring of size length, starting at position pos.</td>
</tr>
</tbody>
</table>

Comparisons

<table>
<thead>
<tr>
<th>str1 == str2</th>
<th>str1 != str2</th>
<th>Compare for equality or inequality; returns a Boolean value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str1 &lt; str2</td>
<td>str1 &gt; str2</td>
<td>Four comparisons. All are lexicographical comparisons.</td>
</tr>
<tr>
<td>str1 &lt;= str2</td>
<td>str1 &gt;= str2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>str.find(str1)</th>
<th>Returns index of the first occurrence of str1 in str.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str.find(str1, pos)</td>
<td>Returns index of the first occurrence of string str1 in str; the search starts at position pos.</td>
</tr>
<tr>
<td>str.find_first_of(str1, pos)</td>
<td>Returns the index of the first instance in str of any character in str1, starting the search at position pos.</td>
</tr>
<tr>
<td>str.find_first_not_of (str1, pos)</td>
<td>Returns the index of the first instance in str of any character not in str1, starting search at position pos.</td>
</tr>
</tbody>
</table>

String Conversions

- **Automatic type conversions**
  - From c-string to c++ string object:
    ```
    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:**
    ```
    strcpy(aCString, stringVar.c_str());
    ```