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

Chapter 9
Strings

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

<table>
<thead>
<tr>
<th></th>
<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></td>
<td>H</td>
<td>i</td>
<td>M</td>
<td>o</td>
<td>m</td>
<td>l</td>
<td>1</td>
<td>0</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

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
### 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'};`

### Character Functions

- Found in `<cctype>` library

### C-String Index Manipulation

- **Can manipulate indexed variables**
    - 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
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

```cpp
//Demonstrates the standard class string.
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    string phrase = "fried", noun="ants";
    string wish = "Bon appetit!";
    phrase = "I love " + adjective + " " + noun + "!";
    cout << phrase << endl;
    cout << wish << endl;
    return 0;
}
```

### Sample Dialogue

I love fried ants!  
Bon appetit!

### Character Functions

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

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>islower(Char_Exp)</td>
<td>Returns true provided Char_Exp is a lowercase letter; otherwise, returns false.</td>
</tr>
<tr>
<td>isalpha(Char_Exp)</td>
<td>Returns true provided Char_Exp is a letter of the alphabet; otherwise, returns false.</td>
</tr>
<tr>
<td>isdigit(Char_Exp)</td>
<td>Returns true provided Char_Exp is one of the digits '0' through '9'; otherwise, returns false.</td>
</tr>
<tr>
<td>isalnum(Char_Exp)</td>
<td>Returns true provided Char_Exp is either a letter or a digit; otherwise, returns false.</td>
</tr>
<tr>
<td>isspace(Char_Exp)</td>
<td>Returns true provided Char_Exp is a whitespace character, such as the blank or newline character; otherwise, returns false.</td>
</tr>
<tr>
<td>ispunct(Char_Exp)</td>
<td>Returns true provided Char_Exp is a punctuation character other than whitespace, a digit, or a letter; otherwise, returns false.</td>
</tr>
<tr>
<td>isprint(Char_Exp)</td>
<td>Returns true provided Char_Exp is a printing character; otherwise, returns false.</td>
</tr>
<tr>
<td>isgraph(Char_Exp)</td>
<td>Returns true provided Char_Exp is a printing character other than whitespace; otherwise, returns false.</td>
</tr>
<tr>
<td>isctli(Char_Exp)</td>
<td>Returns true provided Char_Exp is a control character; otherwise, returns false.</td>
</tr>
</tbody>
</table>

#### Display 9.4 Program Using the Class string

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

int main()
{
    string phrase = "fried", noun = "ants";
    string wish = "Bon appetit!";
    phrase = "I love " + adjective + " " + noun + "!";
    cout << phrase << endl;
    cout << wish << endl;
    return 0;
}
```
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 I/O

- Just like other types!
  ```cpp
  string s1, s2;
  cin >> s1;
  cin >> s2;
  ```
- If the user types in:
  ```cpp
  "May the hair on your toes grow long and curly!"
  ```
- Extraction still ignores whitespace:
  ```cpp
  s1 receives value "May"
s2 receives value "the"
  ```

---

Display 6.7 Member Functions of the Standard Class string

<table>
<thead>
<tr>
<th>MEMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str()</code></td>
<td>Default constructor; creates empty string object str.</td>
</tr>
<tr>
<td><code>str(const string&amp; str)</code></td>
<td>Creates a string object with data &quot;string&quot;.</td>
</tr>
<tr>
<td><code>str(const char* str)</code></td>
<td>Creates a string object str that is a copy of str.</td>
</tr>
<tr>
<td><code>str(char c)</code></td>
<td>Allocates space and initializes it to str’s data, releases memory allocated for str1, and sets str1’s size to that of str2.</td>
</tr>
<tr>
<td><code>str(const string&amp; str)</code></td>
<td>Character data of str2 is concatenated to the end of str1, the size is set appropriately.</td>
</tr>
<tr>
<td><code>str.empty()</code></td>
<td>Returns true if str is an empty string, returns false otherwise.</td>
</tr>
</tbody>
</table>

---

To read in entire lines use `getline(cin, string)`:  
```cpp
string line;
cout << "Enter a line of input: ";
gline(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
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>str1 &lt; str2</th>
<th>str1 &gt; str2</th>
<th>str1 &lt;= str2</th>
<th>str1 &gt;= str2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare for equality or inequality; returns a Boolean value.</td>
<td>Four comparisons. All are lexicographical comparisons.</td>
<td></td>
<td></td>
<td></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>
</tbody>
</table>

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

String Conversions

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