Learning Objectives

• An Array Type for Strings
  – C-Strings

• Character Manipulation Tools
  – Character I/O
  – get, put member functions
  – putback, peek, ignore

• Standard Class string
  – String processing
Introduction

• Two string types:
  • C-strings
    – Array with base type char
    – End of string marked with null, "\0"
    – "Older" method inherited from C
  • String class
    – Uses templates

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:
  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 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
  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:
    ```c
    char aString[10];
    aString = "Hello"; // ILLEGAL!
    
    * Can ONLY use "=" at declaration of c-string!
    ```

- Must use library function for assignment:
  ```c
  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 ==
  ```c
  char aString[10] = "Hello";
  char anotherString[10] = "Goodbye";
  
  aString == anotherString; // NOT allowed!
  ```

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

**Display 9.1** Some Predefined C-String Functions in `<cstring>` (1 of 2)

- Full of string manipulation functions

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>CAUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcpy(Target_String_Var, Snc_String)</code></td>
<td>Copies the C-string value <code>Snc_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>Snc_String</code>.</td>
</tr>
<tr>
<td><code>strcpy(Target_String_Var, Snc_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, Snc_String)</code></td>
<td>Concatenates the C-string value <code>Snc_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)

The `<cstring>` Library:

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

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>CAUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcat(Target_String_Var, Snc_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>
<tr>
<td><code>strlen(Snc_String)</code></td>
<td>Returns an integer equal to the length of <code>Snc_String</code>. (The null character <code>\0</code> 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>strcmp(String_1, String_2, Limit)</code></td>
<td>The same as the two-argument <code>strcmp</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>strcmp</code>. Not implemented in all versions of C++.</td>
</tr>
</tbody>
</table>
C-string Functions: strlen()

- "String length"
- Often useful to know string length:
  ```
  char myString[10] = "d obero";
  cout << strlen(myString);
  ```
  - Returns number of characters
    - Not including null
  - Result here: 6

C-string Functions: strcat()

- strcat()
- "String concatenate":
  ```
  char stringVar[20] = "The rain";
  strcat(stringVar, "in Spain");
  ```
  - Note result:
    - 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:
  
  ```cpp
  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

Example: Command Line Arguments

• Programs invoked from the command line (e.g. a UNIX shell, DOS command prompt) can be sent arguments

  – Example: COPY C:\FOO.TXT  D:\FOO2.TXT

  • This runs the program named “COPY” and sends in two C-String parameters, “C:\FOO.TXT” and “D:\FOO2.TXT”
  
  • It is up to the COPY program to process the inputs presented to it; i.e. actually copy the files

• Arguments are passed as an array of C-Strings to the main function
Example: Command Line Arguments

• Header for main
  – int main(int argc, char *argv[])
  – argc specifies how many arguments are supplied. The name of the program counts, so argc will be at least 1.
  – argv is an array of C-Strings.
    • argv[0] holds the name of the program that is invoked
    • argv[1] holds the name of the first parameter
    • argv[2] holds the name of the second parameter
    • Etc.

Example: Command Line Arguments

```cpp
// Echo back the input arguments
int main(int argc, char *argv[])
{
    for (int i=0; i<argc; i++)
    {
        cout << "Argument " << i << " " << argv[i] << endl;
    }
    return 0;
}
```

Sample Execution

```
> Test  Argument 0  Test
Argument 0 Test
Argument 1 hello
Argument 2 world
```
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!

Character I/O

- Input and output data
  - ALL treated as character data
  - e.g., number 10 outputted as "1" and "0"
  - Conversion done automatically
    - Uses low-level utilities

- Can use same low-level utilities ourselves as well
Member Function get()

- Reads one char at a time
- Member function of cin object:
  char nextSymbol;
  cin.get(nextSymbol);
  - Reads next char & puts in variable nextSymbol
  - Argument must be char type
    - Not "string"!

Member Function put()

- Outputs one character at a time
- Member function of cout object:
- Examples:
  cout.put("a");
  - Outputs letter "a" to screen
  char myString[10] = "Hello"
  cout.put(myString[1]);
  - Outputs letter "e" to screen
More Member Functions

- **putback()**
  - Once read, might need to "put back"
  - `cin.putback(lastChar);`

- **peek()**
  - Returns next char, but leaves it there
  - `peekChar = cin.peek();`

- **ignore()**
  - Skip input, up to designated character
  - `cin.ignore(1000, "\n");`
    - Skips at most 1000 characters until "\n"

### Display 9.3 Some Functions in `<cctype>` (1 of 3)

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

### Display 9.3 Some Functions in `<cctype>` (2 of 3)

<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. | \begin{verbatim}
char c = 'a';
if (islower(c))
    cout << c << " is lowercase."
else
    cout << "is not a letter."
\end{verbatim} |

| isalpha(Char_Exp)   | Returns true provided Char_Exp is a letter of the alphabet, otherwise, returns false. | \begin{verbatim}
char c = 'S';
if (isalpha(c))
    cout << "is a letter."
else
    cout << "is not a letter."
\end{verbatim} |

| isdigit(Char_Exp)   | Returns true provided Char_Exp is one of the digits '0' through '9'; otherwise, returns false. | \begin{verbatim}
if (isdigit('3'))
    cout << "It's a digit."
else
    cout << "It's not a digit."
\end{verbatim} |

| isalnum(Char_Exp)   | Returns true provided Char_Exp is either a letter or a digit; otherwise, returns false. | \begin{verbatim}
if (isalnum('3') && isalnum('a'))
    cout << "Both alphanumeric."
else
    cout << "One or more are not."
\end{verbatim} |

## Character-Manipulating Functions:

### Display 9.3 Some Functions in `<cctype>` (3 of 3)

| isspace(Char_Exp)   | Returns true provided Char_Exp is a whitespace character, such as the blank or newline character; otherwise, returns false. | \begin{verbatim}
//Skips over one "word" and sets c to equal to the first whitespace after the "word"
//character after the "word"
do {
    cin.get(c);
} while (!isspace(c));
\end{verbatim} |

| ispunct(Char_Exp)   | Returns true provided Char_Exp is a printing character other than whitespace, a digit, or a letter; otherwise, returns false. | \begin{verbatim}
if (ispunct('?'))
    cout << "Is punctuation."
else
    cout << "Not punctuation."
\end{verbatim} |

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

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

| isctrl(Char_Exp)    | Returns true provided Char_Exp is a control character; otherwise, returns false. | | |
Standard Class string

- 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!

Display 9.4
Program Using the Class string

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

    return 0;
}
```

**SAMPLE DIALOGUE**
I love fried ants!
Bon appetit!
I/O with Class string

• Just like other types!

• `string s1, s2;`  
  `cin >> s1;`  
  `cin >> s2;`

• Results:  
  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"

getline() with Class string

• For complete lines:  
  `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 INPUT  
  – Similar to c-string’s usage of getline()
Other getline() Versions

• Can specify "delimiter" character:
  string line;
  cout << "Enter input: ";
  getline(cin, line, "?");
  – Receives input until "?" encountered

• getline() actually returns reference
  – string s1, s2;
    getline(cin, s1) >> s2;
  – Results in: (cin) >> s2;

Pitfall: Mixing Input Methods

• Be careful mixing cin >> var and getline
  – int n;
    string line;
    cin >> n;
    getline(cin, line);
  – If input is: 42
    Hello hitchhiker.
    • Variable n set to 42
    • line set to empty string!
  – cin >> n skipped leading whitespace, leaving "\n" on stream for getline()!
Class string Processing

• Same operations available as c-strings
• And more!
  – Over 100 members of standard string class
• Some member functions:
  – .length()
    • Returns length of string variable
  – .at(i)
    • Returns reference to char at position i

Display 9.7 Member Functions of the Standard Class string (1 of 2)

<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constructors</strong></td>
<td></td>
</tr>
<tr>
<td>string str;</td>
<td>Default constructor; creates empty string object str.</td>
</tr>
<tr>
<td>string str(&quot;string&quot;);</td>
<td>Creates a string object with data &quot;string&quot;.</td>
</tr>
<tr>
<td>string str(aString);</td>
<td>Creates a string object str that is a copy of aString. aString is an object of the class string.</td>
</tr>
<tr>
<td><strong>Element access</strong></td>
<td></td>
</tr>
<tr>
<td>str[i]</td>
<td>Returns read/write reference to character in str at index i.</td>
</tr>
<tr>
<td>str.at(i)</td>
<td>Returns read/write reference to character in str at index i.</td>
</tr>
<tr>
<td>str.substr(position, length)</td>
<td>Returns the substring of the calling object starting at position and having length characters.</td>
</tr>
<tr>
<td><strong>Assignment/Modifiers</strong></td>
<td></td>
</tr>
<tr>
<td>str1 = str2;</td>
<td>Allocates space and initializes it to str2's data, releases memory allocated for str1, and sets str1's size to that of str2.</td>
</tr>
<tr>
<td>str1 += str2;</td>
<td>Character data of str2 is concatenated to the end of str1; the size is set appropriately.</td>
</tr>
<tr>
<td>str.empty( )</td>
<td>Returns true if str is an empty string; returns false otherwise.</td>
</tr>
</tbody>
</table>
Display 9.7  Member Functions of the Standard Class string (2 of 2)

C-string and string Object Conversions

• Automatic type conversions
  
  – From c-string to 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());
Converting between \texttt{string} and numbers

• In C++11 it is simply a matter of calling \texttt{stof}, \texttt{stod}, \texttt{stoi}, or \texttt{stol} to convert a string to a float, double, int, or long, respectively.

```cpp
int i;
double d;
string s;
i = stoi("35"); // Converts the string "35" to an integer 35
d = stod("2.5"); // Converts the string "2.5" to the double 2.5
```

Converting between numbers and \texttt{string} objects

• In C++11 use \texttt{to_string} to convert a numeric type to a string

```cpp
string s;
s = to_string(d*2); // Converts the double 5.0 to a // string "5.0000"
```
Summary

- C-string variable is "array of characters"
  - With addition of null character, "\0"
- C-strings act like arrays
  - Cannot assign, compare like simple variables
- Libraries <cctype> & <string> have useful manipulating functions
- cin.get() reads next single character
- getline() versions allow full line reading
- Class string objects are better-behaved than c-strings