Object-Oriented Programming
(in C++)

Class `string` & String Stream Processing

Professor Yi-Ping You (游逸平)
Department of Computer Science
http://www.cs.nctu.edu.tw/~ypyou/

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Standard Library Class `string`

- `basic_string`
  - A class template that provides typical string-manipulation operations
  - `typedef basic_string<char> string;`
  - `typedef basic_string<wchar_t> wstring;`
  - Defined in `namespace std`
  - Include `<string>` to use `strings`
  - Unlike C-style `char *` strings, `strings` are not necessarily null terminated
    - The length of a `string` can be retrieved with member function `length`
    - Subscript `operator[]` can be used to access and modify individual characters
    - Stream insertion and extraction operators (<< and >>) is overloaded for `string`

Initializing `string` Objects (1/2)

- A `string` object can be initialized with
  - `string text( "Hello" );`
    - creates a `string` from a `const char *`
  - `string name( 8, 'x' );`
    - `string` of 8 `'x'` characters
  - `string month = "March";`
    - same as: `string month( "March" );`
    - It's an implicit call to the string class constructor, not an assignment

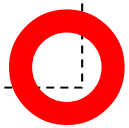
Initializing `string` Objects (2/2)

- Class `string` provides no conversions from `int` or `char` to `string`

```
string error1 = 'c';
string error2( 'u' );
string error3 = 22;
string error4( 8 );
```

- Assigning (rather than initializing) a single character to a `string` object is permitted in an assignment statement

```
string str1;
str1 = 'n';
str1 = 110;
```



Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Assignment and Concatenation

- Assignment
  - `operator=` is overloaded for `string` assignment
  - Member function `assign` is designed for assigning a new value to the `string`, replacing its current contents
- Concatenation
  - `operator+` is overloaded for `string` concatenation
  - Member function `append` is designed for extending the `string` by appending additional characters at the end of its current value

Member Function `assign`

```
string (1)    string& assign (const string& str);
substring (2) string& assign (const string& str, size_t subpos, size_t sublen);
C-string (3)  string& assign (const char* s);
buffer (4)    string& assign (const char* s, size_t n);
fill (5)      string& assign (size_t n, char c);
range (6)     template <class InputIterator>
                 string& assign (InputIterator first, InputIterator last);
```

- string
  - Copies `str`
- substring
  - Copies the portion of `str` that begins at the character position `subpos` and spans `sublen` characters (or until the end of `str`, if either `str` is too short or if `sublen` is `string::npos`)
- c-string
  - Copies the null-terminated character sequence (C-string) pointed by s
- buffer
  - Copies the first n characters from the array of characters pointed by s
- fill
  - Replaces the current value by n consecutive copies of character c
- range
  - Copies the sequence of characters in the range `[first,last)`, in the same order

Member Function append

```
string (1)    string& append (const string& str);
substring (2) string& append (const string& str, size_t subpos, size_t sublen);
C-string (3)  string& append (const char* s);
buffer (4)    string& append (const char* s, size_t n);
fill (5)      string& append (size_t n, char c);
range (6)     template <class InputIterator>
                 string& append (InputIterator first, InputIterator last);
```

- string
  - Appends a copy of `str`
- substring
  - Appends a copy of a substring of `str`. The substring is the portion of `str` that begins at the character position `subpos` and spans `sublen` characters (or until the end of `str`, if either `str` is too short or if `sublen` is `string::npos`)
- c-string
  - Appends a copy of the string formed by the null-terminated character sequence (C-string) pointed by s
- buffer
  - Appends a copy of the first n characters in the array of characters pointed by s
- fill
  - Appends n consecutive copies of character c
- range
  - Appends a copy of the sequence of characters in the range `[first,last)`, in the same order

Assignment and Concatenation: Examples

```
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string string1( "cat" );
10    string string2; // initialized to the empty string
11    string string3; // initialized to the empty string
12
13    string2 = string1; // assign string1 to string2
14    string3.assign( string1 ); // assign string1 to string3
15    cout << "string1: " << string1 << "\nstring2: " << string2
16       << "\nstring3: " << string3 << "\n\n";
17
18    // modify string2 and string3
19    string2[ 0 ] = string3[ 2 ] = 'r';
20
21    cout << "After modification of string2 and string3:\n" << "string1: "
22       << string1 << "\nstring2: " << string2 << "\nstring3: ";
23
24    // demonstrating member function at
25    for ( int i = 0; i < string3.length(); i++ )
26       cout << string3.at( i );
27
28    // declare string4 and string5
29    string string4( string1 + "apult" ); // concatenation
30    string string5;
31
32    // overloaded +=
33    string3 += "pet"; // create "carpet"
34    string1.append( "acomb" ); // create "catacomb"
35
36    // append subscript locations 4 through end of string1 to
37    // create string "comb" (string5 was initially empty)
38    string5.append( string1, 4, string1.length() - 4 );
39
40    cout << "\n\nAfter concatenation:\nstring1: " << string1
41       << "\nstring2: " << string2 << "\nstring3: " << string3
42       << "\nstring4: " << string4 << "\nstring5: " << string5 << endl;
43 } // end main
```

```
string1: cat
string2: cat
string3: cat

After modification of string2 and string3:
string1: cat
string2: rat
string3: car

After concatenation:
string1: catacomb
string2: rat
string3: carpet
string4: catapult
string5: comb
```

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Comparison

- `operator==`, `operator!=`, `operator<`, `operator<=`, `operator>`, `operator>=` are overloaded for `string` comparison
  - Lexicographical comparison
- Member function `compare` is designed for comparing the value of the `string` object (or a substring) to the sequence of characters specified by its arguments

Member Function compare

string (1)	`int compare (const string& str) const;`
substrings (2)	`int compare (size_t pos, size_t len, const string& str) const;`
	`int compare (size_t pos, size_t len, const string& str, size_t subpos, size_t sublen) const;`
c-string (3)	`int compare (const char* s) const;`
	`int compare (size_t pos, size_t len, const char* s) const;`
buffer (4)	`int compare (size_t pos, size_t len, const char* s, size_t n) const;`

- `str`
  - Another string object, used entirely (or partially) as the comparing string.
- `pos`
  - Position of the first character in the compared string.
  - If this is greater than the string length, it throws `out_of_range`.
  - Note: The first character is denoted by a value of 0 (not 1).
- `len`
  - Length of compared string (if the string is shorter, as many characters as possible).
  - A value of `string::npos` indicates all characters until the end of the string.
- `subpos`, `sublen`
  - Same as `pos` and `len` above, but for the comparing string.
- `s`
  - Pointer to an array of characters.
  - If argument n is specified (4), the first n characters in the array are used as the comparing string.
  - Otherwise (3), a null-terminated sequence is expected: the length of the sequence with the characters to use as comparing string is determined by the first occurrence of a null character.
- `n`
  - Number of characters to compare.

Comparison: Examples (1/2)

```
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string string1( "Testing the comparison functions." );
10    string string2( "Hello" );
11    string string3( "stinger" );
12    string string4( string2 );
13
14    cout << "string1: " << string1 << "\nstring2: " << string2
15       << "\nstring3: " << string3 << "\nstring4: " << string4 << "\n\n";
16
17    // comparing string1 and string4
18    if ( string1 == string4 )
19       cout << "string1 == string4\n";
20    else // string1 != string4
21    {
22       if ( string1 > string4 )
23          cout << "string1 > string4\n";
24       else // string1 < string4
25          cout << "string1 < string4\n";
26    } // end else
27
28    // comparing string1 and string2
29    int result = string1.compare( string2 );
30
31    if ( result == 0 )
32       cout << "string1.compare( string2 ) == 0\n";
33    else // result != 0
34    {
35       if ( result > 0 )
36          cout << "string1.compare( string2 ) > 0\n";
37       else // result < 0
38          cout << "string1.compare( string2 ) < 0\n";
39    } // end else
```

```
string1: Testing the comparison functions.
string2: Hello
string3: stinger
string4: Hello

string1 > string4
string1.compare( string2 ) > 0
```

Comparison: Examples (2/2)

```
41    // comparing string1 (elements 2-6) and string3 (elements 0-4)
42    result = string1.compare( 2, 5, string3, 0, 5 );
43
44    if ( result == 0 )
45       cout << "string1.compare( 2, 5, string3, 0, 5 ) == 0\n";
46    else // result != 0
47    {
48       if ( result > 0 )
49          cout << "string1.compare( 2, 5, string3, 0, 5 ) > 0\n";
50       else // result < 0
51          cout << "string1.compare( 2, 5, string3, 0, 5 ) < 0\n";
52    } // end else
53
54    // comparing string2 and string4
55    result = string4.compare( 0, string2.length(), string2 );
56
57    if ( result == 0 )
58       cout << "string4.compare( 0, string2.length(), "
59          << "string2 ) == 0" << endl;
60    else // result != 0
61    {
62       if ( result > 0 )
63          cout << "string4.compare( 0, string2.length(), "
64             << "string2 ) > 0" << endl;
65       else // result < 0
66          cout << "string4.compare( 0, string2.length(), "
67             << "string2 ) < 0" << endl;
68    } // end else
69
70    // comparing string2 and string4
71    result = string2.compare( 0, 3, string4 );
72
73    if ( result == 0 )
74       cout << "string2.compare( 0, 3, string4 ) == 0" << endl;
75    else // result != 0
76    {
77       if ( result > 0 )
78          cout << "string2.compare( 0, 3, string4 ) > 0" << endl;
79       else // result < 0
80          cout << "string2.compare( 0, 3, string4 ) < 0" << endl;
81    } // end else
82 } // end main
```

```
string1: Testing the comparison functions.
string2: Hello
string3: stinger
string4: Hello
```

```
string1.compare( 2, 5, string3, 0, 5 ) == 0
string4.compare( 0, string2.length(), string2 ) == 0
string2.compare( 0, 3, string4 ) < 0
```

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Member Function substr

```
string substr (size_t pos = 0, size_t len = npos) const;
```

- Returns a newly constructed string object with its value initialized to a copy of a substring of this object
- The substring is the portion of the object that starts at character position `pos` and spans `len` characters

Substrings: Examples

```
1  // Fig. 18.3: Fig18_03.cpp
2  // Demonstrating string member function substr.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string string1( "The airplane landed on time." );
10
11    // retrieve substring "plane" which
12    // begins at subscript 7 and consists of 5 characters
13    cout << string1.substr( 7, 5 ) << endl;
14 } // end main
```

```
plain
```

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Member Function swap

```
void swap (string& str);
```

- Exchanges the content of the container by the content of `str`, which is another string object. Lengths may differ.
- After the call to this member function, the value of this object is the value `str` had before the call, and the value of `str` is the value this object had before the call
- Notice that a non-member function exists with the same name, `swap`, overloading that algorithm with an optimization that behaves like this member function

swap: Examples

```
1  // Fig. 18.4: Fig18_04.cpp
2  // Using the swap function to swap two strings.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string first( "one" );
10    string second( "two" );
11
12    // output strings
13    cout << "Before swap:\n first: " << first << "\nsecond: " << second;
14
15    first.swap( second ); // swap strings
16
17    cout << "\n\nAfter swap:\n first: " << first
18       << "\nsecond: " << second << endl;
19 } // end main
```

```
Before swap:
 first: one
second: two

After swap:
 first: two
second: one
```

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Finding Substrings and Characters: Examples

```
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string string1( "noon is 12 pm; midnight is not." );
10    int location;
11
12    // find "is" at location 5 and 24
13    cout << "Original string:\n" << string1
14       << "\n\n(find) \"is\" was found at: " << string1.find( "is" )
15       << "\n(rfind) \"is\" was found at: " << string1.rfind( "is" );
16
17    // find 'o' at location 1
18    location = string1.find_first_of( "misop" );
19    cout << "\n\n(find_first_of) found '" << string1[ location ]
20       << "' from the group \"misop\" at: " << location;
21
22    // find 'o' at location 29
23    location = string1.find_last_of( "misop" );
24    cout << "\n\n(find_last_of) found '" << string1[ location ]
25       << "' from the group \"misop\" at: " << location;
26
27    // find '1' at location 8
28    location = string1.find_first_not_of( "noi spm" );
29    cout << "\n\n(find_first_not_of) '" << string1[ location ]
30       << "' is not contained in \"noi spm\" and was found at: "
31       << location;
32
33    // find ';' at location 13
34    location = string1.find_first_not_of( "12noi spm" );
35    cout << "\n\n(find_first_not_of) '" << string1[ location ]
36       << "' is not contained in \"12noi spm\" and was "
37       << "found at: " << location << endl;
38
39    // search for characters not in string1
40    location = string1.find_first_not_of(
41       "noon is 12 pm; midnight is not." );
42    cout << "\nfind_first_not_of(\"noon is 12 pm; midnight is not.\")"
43       << " returned: " << location << endl;
44 } // end main
```

- `find`
  - Searches the string for the first occurrence of the sequence specified by its arguments
  - Returns the subscript of the starting location of that string
- `find_first_of`
  - Searches the string for the first character that matches any of the characters specified in its arguments.

```
Original string:
noon is 12 pm; midnight is not.

(find) "is" was found at: 5
(rfind) "is" was found at: 24

(find_first_of) found 'o' from the group "misop" at: 1

(find_last_of) found 'o' from the group "misop" at: 29

(find_first_not_of) '1' is not contained in "noi spm" and was found at: 8

(find_first_not_of) ';' is not contained in "12noi spm" and was found at: 13

find_first_not_of("noon is 12 pm; midnight is not.") returned: -1
```

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Replacing Characters: Examples

```
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     // compiler concatenates all parts into one string
10    string string1( "The values in any left subtree"
11       "\nare less than the value in the"
12       "\nparent node and the values in"
13       "\nany right subtree are greater"
14       "\nthan the value in the parent node" );
15
16    cout << "Original string:\n" << string1 << endl << endl;
17
18    // remove all characters from (and including) location 62
19    // through the end of string1
20    string1.erase( 62 );
21
22    // output new string
23    cout << "Original string after erase:\n" << string1
24       << "\n\nAfter first replacement:\n";
25
26    int position = string1.find( " " ); // find first space
27
28    // replace all spaces with period
29    while ( position != string::npos )
30    {
31       string1.replace( position, 1, "." );
32       position = string1.find( " ", position + 1 );
33    } // end while
34
35    cout << string1 << "\n\nAfter second replacement:\n";
36
37    position = string1.find( "." ); // find first period
38
39    // replace all periods with two semicolons
40    // NOTE: this will overwrite characters
41    while ( position != string::npos )
42    {
43       string1.replace( position, 2, "xxxxx;;yyy", 5, 2 );
44       position = string1.find( ".", position + 1 );
45    } // end while
46
47    cout << string1 << endl;
48 } // end main
```

location 62

`find` returns `string::npos` when the search character is not found

```
Original string:
The values in any left subtree
are less than the value in the
parent node and the values in
any right subtree are greater
than the value in the parent node

Original string after erase:
The values in any left subtree
are less than the value in the


After first replacement:
The.values.in.any.left.subtree
are.less.than.the.value.in.the


After second replacement:
The;;alues;;n;;ny;;eft;;ubtree
are;;ess;;han;;he;;alue;;n;;he
```

replaces two characters at `position` with `substr(5, 2)` of “xxxxx;;yyy”

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Inserting Characters: Examples

```
 3  #include <iostream>
 4  #include <string>
 5  using namespace std;
 6
 7  int main()
 8  {
 9     string string1( "beginning end" );
10     string string2( "middle " );
11     string string3( "12345678" );
12     string string4( "xx" );
13
14     cout << "Initial strings:\nstring1: " << string1
15        << "\nstring2: " << string2 << "\nstring3: " << string3
16        << "\nstring4: " << string4 << "\n\n";
17
18     // insert "middle" at location 10 in string1
19     string1.insert( 10, string2 );
20
21     // insert "xx" at location 3 in string3
22     string3.insert( 3, string4, 0, string::npos );
23
24     cout << "Strings after insert:\nstring1: " << string1
25        << "\nstring2: " << string2 << "\nstring3: " << string3
26        << "\nstring4: " << string4 << endl;
27  } // end main
```

```
Initial strings:
string1: beginning end
string2: middle
string3: 12345678
string4: xx

Strings after insert:
string1: beginning middle end
string2: middle
string3: 123xx45678
string4: xx
```

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Conversion to char *: Examples

```
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string string1( "STRINGS" ); // string constructor with char* arg
10    const char *ptr1 = 0; // initialize *ptr1
11    int length = string1.length();
12    char *ptr2 = new char[ length + 1 ]; // including null
13
14    // copy characters from string1 into allocated memory
15    string1.copy( ptr2, length, 0 ); // copy
16    ptr2[ length ] = '\0'; // add null termi
17
18    cout << "string string1 is " << string1
19       << "\nstring1 converted to a C-Style string is "
20       << string1.c_str()   << "\nptr1 is ";
21
22    // Assign to pointer ptr1 the const char * returned by
23    // function data(). NOTE: this is a potentially dangerous
24    // assignment. If string1 is modified, pointer ptr1 can
25    // become invalid.
26    ptr1 = string1.data();
27
28    // output each character using pointer
29    for ( int i = 0; i < length; i++ )
30       cout << *( ptr1 + i ); // use pointer arithmetic
31
32    cout << "\nptr2 is " << ptr2 << endl;
33    delete [] ptr2; // reclaim dynamically allocated memory
34 } // end main
```

Returns a pointer to an array that contains a null-terminated sequence of characters (i.e., a C-string) representing the current value of the string object

Returns a pointer to an array that contains the same sequence of characters as the characters that make up the value of the string object

```
string string1 is STRINGS
string1 converted to a C-Style string is STRINGS
ptr1 is STRINGS
ptr2 is STRINGS
```

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

Iterators for `string`

- Class `string` provides iterators for forward and backward traversal of `strings`
- Iterators provide access to individual characters with syntax that is similar to pointer operations
- Iterators are not range checked

Iterators: Examples

```
 3  #include <iostream>
 4  #include <string>
 5  using namespace std;
 6
 7  int main()
 8  {
 9     string string1( "Testing iterators" );
10     string::const_iterator iterator1 = string1.begin();
11
12     cout << "string1 = " << string1
13        << "\n(Using iterator iterator1) string1 is: ";
14
15     // iterate through string
16     while ( iterator1 != string1.end() )
17     {
18        cout << *iterator1; // dereference iterator to get char
19        iterator1++; // advance iterator to next char
20     } // end while
21
22     cout << endl;
23  } // end main
```

```
string1 = Testing iterators
(Using iterator iterator1) string1 is: Testing iterators
```

Outline

- Introduction to `string`
- Overloaded Operators and Member Functions of `string`
  - Assignment and Concatenation
  - Comparison
  - Substrings
  - Swapping
  - Finding Substrings and Characters
  - Replacing Characters
  - Inserting Characters
  - Conversion to `char *`
  - Iterators
- String Stream Processing

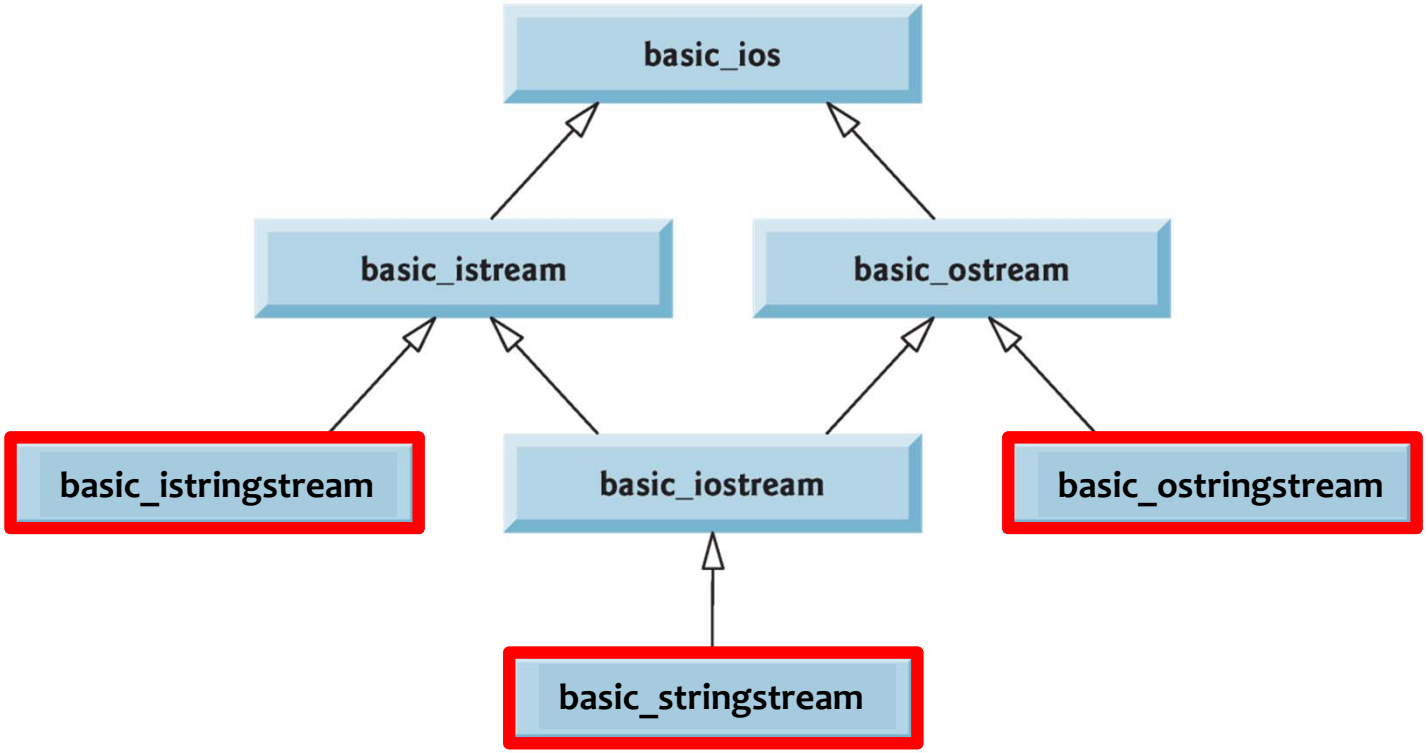
`string` Stream Processing (1/2)

- In addition to standard stream I/O and file stream I/O, C++ stream I/O includes capabilities for inputting from, and outputting to, `strings` in memory
- These capabilities often are referred to as in-memory I/O or string stream processing
- Input from a `string` is supported by class `istringstream`
- Output to a `string` is supported by class `ostringstream`

string Stream Processing (2/2)

- The class names istringstream and ostringstream are actually aliases defined by the typedefs
  - typedef basic_istringstream<char> istringstream;
  - typedef basic_ostringstream<char> ostringstream;
- Class templates basic_istringstream and basic_ostringstream provide the same functionality as classes istream and ostream plus other member functions specific to in-memory formatting
- Programs that use in-memory formatting must include the <sstream> and <iostream> header files

Stream-I/O Template Hierarchy



string Stream Processing: Examples

```
3  #include <iostream>
4  #include <string>
5  #include <sstream> // header file for string stream processing
6  using namespace std;
7
8  int main()
9  {
10    ostringstream outputString; // create ostringstream instance
11
12    string string1( "Output of several data types " );
13    string string2( "to an ostringstream object:" );
14    string string3( "\n        double: " );
15    string string4( "\n           int: " );
16    string string5( "\naddress of int: " );
17
18    double double1 = 123.4567;
19    int integer = 22;
20
21    // output strings, double and int to ostringstream outputString
22    outputString << string1 << string2 << string3 << double1
23       << string4 << integer << string5 << &integer;
24
25    // call str to obtain string contents of the ostringstream
26    cout << "outputString contains:\n" << outputString.str();
27
28    // add additional characters and call str to output string
29    outputString << "\nmore characters added";
30    cout << "\n\nafter additional stream insertions,\n"
31       << "outputString contains:\n" << outputString.str() << endl;
32 } // end main
```

```
outputString contains:
Output of several data types to an ostringstream object:
        double: 123.457
           int: 22
address of int: 0012F540

after additional stream insertions,
outputString contains:
Output of several data types to an ostringstream object:
        double: 123.457
           int: 22
address of int: 0012F540
more characters added
```

string Stream Processing: Another Example

```
3  #include <iostream>
4  #include <string>
5  #include <sstream>
6  using namespace std;
7
8  int main()
9  {
10    string input( "Input test 123 4.7 A" );
11    istringstream inputString( input );
12    string string1;
13    string string2;
14    int integer;
15    double double1;
16    char character;
17
18    inputString >> string1 >> string2 >> integer >> double1 >> character;
19
20    cout << "The following items were extracted\n"
21       << "from the istringstream object:" << "\nstring: " << string1
22       << "\nstring: " << string2 << "\n   int: " << integer
23       << "\ndouble: " << double1 << "\n  char: " << character;
24
25    // attempt to read from empty stream
26    long value;
27    inputString >> value;
28
29    // test stream results
30    if ( inputString.good() )
31       cout << "\n\nlong value is: " << value << endl;
32    else
33       cout << "\n\ninputString is empty" << endl;
34 } // end main
```

```
The following items were extracted
from the istringstream object:
string: Input
string: test
   int: 123
double: 4.7
  char: A

inputString is empty
```