Strings and Vectors

Modified from Sections 8.2, 8.3

The Standard **string** Class

Modified from Section 8.2

The Standard string Class

- The string class allows the programmer to treat strings as a data type
- The string class is defined in the string library and the names are in the standard namespace
  - To use the string class you need these lines:

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

Assignment of Strings

- Variables of type string can be assigned with the = operator
  - Example:

```
string s1, s2, s3;
…
s3 = s2;
```

- A string literal is a C-style string, which is an array of characters that uses a null terminator.
- String literals can be type cast to type string
  - Example: string s1 = "Hello Mom!";

Using + With strings

- Variables of type string can be concatenated with the + operator
  - Example:

```
string s1, s2, s3;
s1 = “Hello ”;
s2 = “world”;
s3 = s1 + s2;
// s3 is “Hello world”
```

string Constructors

- The default string constructor initializes the string to the empty string
- Another string constructor takes a C-string argument
  - Example:

```
string phrase; // empty string
string noun("ants"); // string “ants”
```

String Processing

- The string class allows the same operations we used with C-strings…and more
  - Characters in a string object can be accessed as if they are in an array
    - last_name[i] provides access to a single character as in an array
    - Index values are not checked for validity!

Member Function length

- The string class member function length returns the length of the string, in terms of bytes.
- One ASCII-encoded character takes 1 byte, so we can use the returned value as the number of characters in a string.
  - Example:

```
int n = string_var.length( );
```

Member Function at

- at is an alternative to using [ ]'s to access characters in a string.
    - at checks for valid index values
    - Example:

```
string str("Mary");
```

Equivalent

```
cout << str[6] << endl;
cout << str.at(6) << endl;
```

Equivalent

```
str[2] = 'X';
str.at(2) = 'X';
```

Comparison of strings

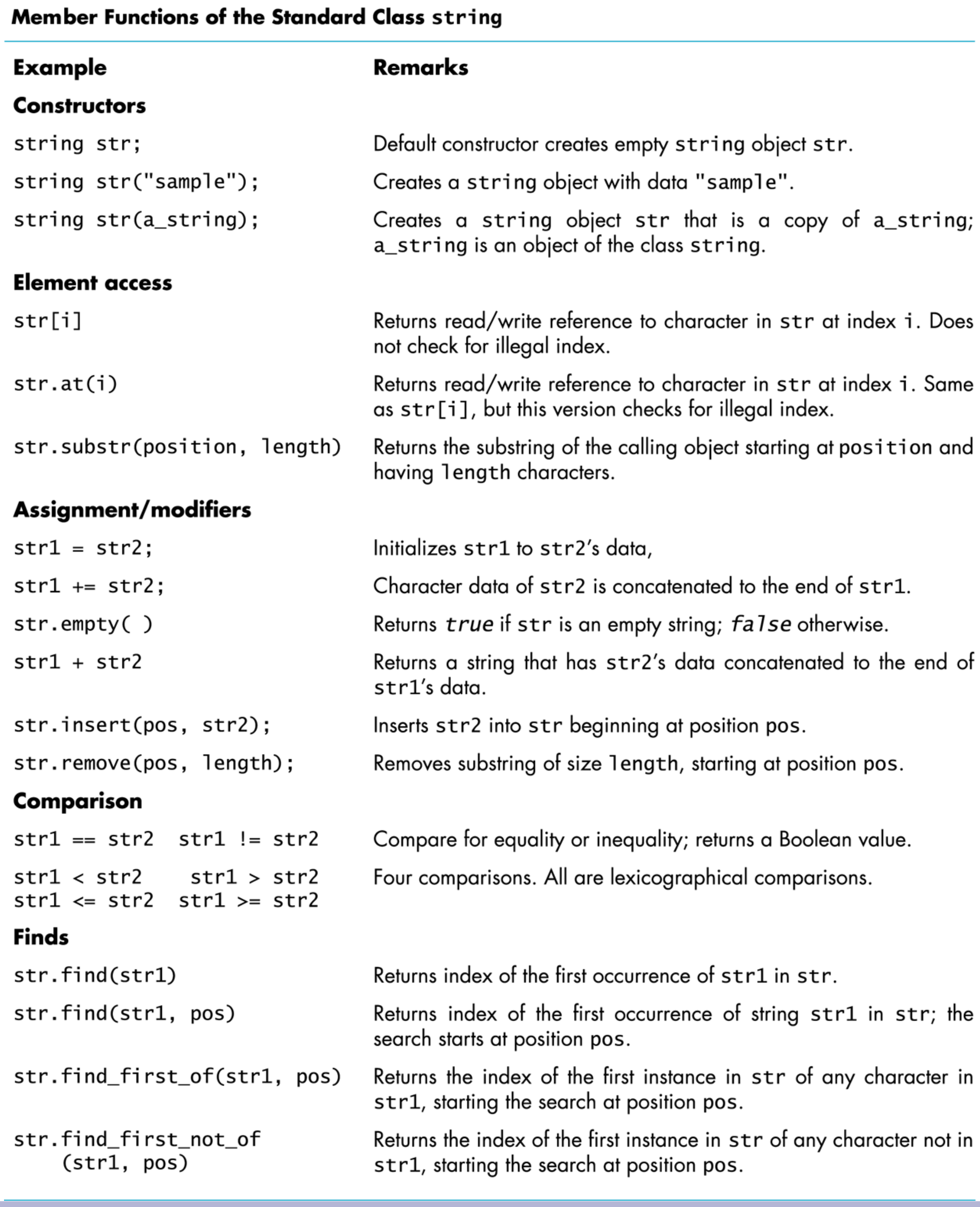
- Comparison operators work with string objects
  - Objects are compared using lexicographic order (Alphabetical ordering using the order of symbols in the ASCII character set.)
  - == returns true if two string objects contain the same characters in the same order
  - <, >, <=, >= can be used to compare string objects

Other string class functions are found in **Display 8.7**

Display 8.7

Back

Next



Member Functions of the Standard Class `string`

Example	Remarks
Constructors	
`string str;`	Default constructor creates empty `string` object `str`.
`string str("sample");`	Creates a `string` object with data `"sample"`.
`string str(a_string);`	Creates a `string` object `str` that is a copy of `a_string`; `a_string` is an object of the class `string`.
Element access	
`str[i]`	Returns read/write reference to character in `str` at index `i`. Does not check for illegal index.
`str.at(i)`	Returns read/write reference to character in `str` at index `i`. Same as `str[i]`, but this version checks for illegal index.
`str.substr(position, length)`	Returns the substring of the calling object starting at `position` and having `length` characters.
Assignment/modifiers	
`str1 = str2;`	Initializes `str1` to `str2`'s data,
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`.
`str.empty( )`	Returns *`true`* if `str` is an empty string; *`false`* otherwise.
`str1 + str2`	Returns a string that has `str2`'s data concatenated to the end of `str1`'s data.
`str.insert(pos, str2);`	Inserts `str2` into `str` beginning at position `pos`.
`str.remove(pos, length);`	Removes substring of size `length`, starting at position `pos`.
Comparison	
`str1 == str2   str1 != str2`	Compare for equality or inequality; returns a Boolean value.

`str1 < str2    str1 > str2`
`str1 <= str2   str1 >= str2`
	Four comparisons. All are lexicographical comparisons.

Finds	
`str.find(str1)`	Returns index of the first occurrence of `str1` in `str`.
`str.find(str1, pos)`	Returns index of the first occurrence of string `str1` in `str`; the search starts at position `pos`.
`str.find_first_of(str1, pos)`	Returns the index of the first instance in `str` of any character in `str1`, starting the search at position `pos`.

`str.find_first_not_of`
`(str1, pos)`
	Returns the index of the first instance in `str` of any character not in `str1`, starting the search at position `pos`.

Vectors

Modified from Section 8.3

The vector Library

- To use the vector class
  - Include the vector library

    #include <vector>
  - Vector names are placed in the standard namespace so the usual using directive is needed:

    using namespace std;

Vectors

- Vectors are like arrays that can change size as your program runs
- Vectors, like arrays, have a base type
- To declare an empty vector with base type int:

  vector<int> v;

  - <int> identifies vector as a template class
  - You can use any base type in a template class:

    vector<string> v;

Accessing vector Elements

- Vectors elements are indexed starting with 0
- [ ]'s are used to read or change the value of an element:
  - Example:

```
v[i] = 42;
cout << v[i];
```

Initializing vector Elements

- Elements can be added to a vector using the member function push_back
  - push_back adds an element in the next available position
  - Example:

    ```
    vector<double> sample;
    sample.push_back(0.0);
    sample.push_back(1.1);
    sample.push_back(2.2);

    // sample will be {0.0, 1.1, 2.2}
    ```

The size Of A vector

- The member function size returns the number of elements in a vector
  - Example: To print each element of a vector given the previous vector initialization:

```
          for (int i= 0; i < sample.size( ); i++)
               cout << sample[i] << endl;
```

The Type unsigned int

- The vector class member function size returns an unsigned int
  - Unsigned int's are nonnegative integers
  - Some compilers will give a warning if the previous for-loop is not changed to:

```
   for (unsigned int i= 0; i < sample.size( ); i++)
         cout << sample[i] << endl;
```

Alternate vector Initialization

- A vector constructor exists that takes an integer argument and initializes that number of elements
  - Example: vector<int> v(10);
    initializes the first 10 elements to 0
    v.size( ) would return 10
  - [ ]'s can now be used to assign elements 0 through 9
  - push_back is used to assign elements with indices greater than 9
  - Initializes elements of basic number types to zero
  - Initializes elements of class types using the default constructor for the class

vector Issues

- Attempting to use [ ] to set a value beyond the size of a vector may not generate a syntax error
  - The program will probably misbehave
  - vector<int> v; // declared as an empty vector
    v[0] = 1;   // run-time error
- The assignment operator with vectors does an element by element copy of the right-hand vector
  - For class types, the assignment operator must make independent copies

vector Efficiency

- A vector's capacity is the number of elements allocated in memory
  - Accessible using the capacity( ) member function
- Size is the number of elements initialized
- When a vector runs out of space, the capacity is automatically increased
  - A common scheme is to double the size of a vector
  - More efficient than allocating smaller chunks of memory

Controlling vector Capacity

- When efficiency is an issue
  - Member function reserve can increase the capacity of a vector
    - v.reserve(32); // at least 32 elements
      v.reserve(v.size( ) + 10); // at least 10 more

  - resize can be used to shrink or expand a vector

    v.resize(24); //elements beyond 24 are lost
    v.resize(100); //76 elements added to the end