C++ Strings

Two types of strings in C++ :

- C-strings (already discussed in cmpt130)
- The Standard `string` Class

The Standard `string` Class

- The `string` class allows the programmer to treat strings as a basic data type
  - No need to deal with strings at array level
- The `string` class is defined in the `string` library and the names are in the standard namespace
  - To use the `string` class you need to include it:

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

Assignment and + operator of Strings

- Variables of type `string` can be assigned with the **=** operator and can be concatenated with **+**
  - Example:

```
string s1, s2, s3;
        …
s1 = s2;
s3 = s2 + s3;
```

- Quoted strings are type cast from type C-string to type `string`
  - Example:

```
string s1 = "Hello there!";
```

`string` Constructors

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

```
string phrase; // empty string
string noun(“ants”);
// a C-string “ants” is used to
// construct string object “noun”
```

Mixing `strings` and C-strings

- It is natural to work with strings in the following manner

```
string phrase = "I love" + adjective
                    + " " + noun + "!";
```

  - It is actually not so easy for C++. It must either convert the null-terminated C-strings, such as "I love", to strings, or it must use an overloaded + operator that works with strings and C-strings

I/O With Class `string`

- The insertion operator << is used to output objects of type `string`
  - Example:

```
string s = "Hello there!";
cout << s;
```

- The extraction operator >> can be used to input data for objects of type string
  - Example:

```
string s1;
cin >> s1;
```

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

EXAMPLE	REMARKS
Constructors	
`string str;`	Default constructor; creates empty `string` object `str`.
`string str("string");`	Creates a `string` object with data `"string"`.
`string str(aString);`	Creates a `string` object `str` that is a copy of `aString`. `aString` is an object of the class `string`.
Element access	
`str[i]`	Returns read/write reference to character in `str` at index `i`.
`str.at(i)`	Returns read/write reference to character in `str` at index `i`.
`str.substr(position, length)`	Returns the substring of the calling object starting at `position` and having `length` characters.
Assignment/Modifiers	
`str1 = str2;`	Allocates space and initializes it to `str2`'s data, releases memory allocated for `str1`, and sets `str1`'s size to that of `str2`.
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`; the size is set appropriately.
`str.empty( )`	Returns `true` if `str` is an empty `string`; returns `false` otherwise.

(continued)

Display 9.7 Member Functions of the Standard Class `string`

EXAMPLE	REMARKS
`str1 + str2`	Returns a `string` that has `str2`'s data concatenated to the end of `str1`'s data. The size is set appropriately.
`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.
Comparisons	
`str1 == str2` `str1 != str2`	Compare for equality or inequality; returns a Boolean value.
`str1 < str2` `str1 > str2`	Four comparisons. All are lexicographical comparisons.
`str1 <= str2` `str1 >= str2`	
`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 search at position `pos`.

C-string and `string` Object Conversions

- Automatic type conversions
  - From c-string to string object:

    ```
    char aCString[] = "My C-string";
    string aString;
    aString = aCstring;
    ```

    - **Perfectly legal and appropriate!**
  - `aCString = aString;`
    - ILLEGAL!
    - Cannot auto-convert to c-string
  - Must use explicit conversion:

    ```
    strcpy(aCString, aString.c_str());
    ```