v2ch3: More on C++ <string>

4 Feb 2009
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review: namespaces

- The static keyword: two uses
  - For local vars: persistent storage
  - For global names: file scope
- Namespaces and using
- Class variables (static member variables)
  - Application: shared data tables
- Class methods (static member functions)
  - Application: singleton classes

Quiz2: 10 min, 20 pts

- Define method overloading and method overriding. What is the difference? [4]
- What is an abstract superclass, and why might one be useful? How do you make one? [4]
- Write C++ declarations for these relationships:
  - "fido is a Dog."
  - "A Dog is a kind of Mammal."
  - "Every Mammal has a Heart."
  - "Any Dog knows how to bark."

Quiz2 answers: #1-2

- Define method overloading and method overriding. What is the difference? [4]
  - Overloading: different versions of method, depending on type of parameters
  - Overriding: a method defined in the superclass, overridden by subclass
- What is an abstract superclass, and why might one be useful? [4]
  - A superclass that cannot be instantiated; write a pure virtual method (=0)

Quiz2 answers: #3

- Write C++ declarations for these relationships:
  - "fido is a Dog."
  - "A Dog is a kind of Mammal."
  - "Every Mammal has a Heart."
  - "Any Dog knows how to bark()."
  - `class Heart {};`
  - `class Mammal { Heart h; };`
  - `class Dog : public Mammal { void bark(); }`
  - `Dog fido;`

Initializing a C++ <string>

- In C, strings are just arrays of char. Limitations?
- In C++, <string> is better: declare/initialize:
  - **#include <string>**
  - **string myName = “vonWilliamson”;**
- Initialize using parameter to constructor:
  - **string yrName(“Billy”);**
- Initialize as a copy (copy constructor):
  - **string hisName(myName);**
- Initialize from a substring (slice):
  - **string herName(myName, 0, 4); // first four chars**

String operations

- **string myName = “vonWilliamson”;**

- Substring: .substr(start, length)
  - **myName.substr(3, 4) // “Will”**
- Concatenate: +
- Insert into string: .insert(where, str)
  - **myName.insert(0, “Count ”) // “Count vonW...”**
- Append to end: .append(str)
  - **myName.append(“son”) // “vonWilliamsonson”**
- Remove characters: .erase(start, len)
  - **myName.erase(7, 3) // “vonWilson”**

String ops: search/replace

- **string myName = “vonWilliamson”;**

- Search for a substring: .find(str, start)
  - Returns location in string
  - **#include <cstddef>** **// for size_t**
  - **size_t idx = myName.find(“son”, 0);** **// 10**
- If not found, find() returns string::npos (nonexistent character position):
  - **if (idx == string::npos) {** **// couldn't find it!**
- Replace: .replace(start, len, replacement)
  - **myName.replace(3, 7, “Ander”)**

String operations

- Length of string: .length() or .size()
- Internal array dynamically resizes as needed
- Current size of internal array: .capacity()
- Force a resize to a larger size: .reserve(size)

- No functions to change case of a whole string, but you can use Standard C++ Library functions on individual characters:
  - **toupper('a')**
  - **tolower('A')**

Lexicographic sorting

- Comparing strings lexically:
  Character-by-character, using ASCII order
  - **“Hello World!” < “Yallo World!”**
  - **“Hello World!” > “Hello Class!”**
  - **“Hello World!” < “Hello world!”**
  - **“Hello World!” > “H”**
  - **“Hello World!” > “ Hello World!”**
- More in the .compare() method