v1ch8: Constructor Initializer List
v2ch1: Exceptions

6 Feb 2009
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review last time: <string>

- Ways to initialize a new string
- .substr(), +, .insert(), .append(), .erase()
- .find(), string::npos, .replace()
- .length(), .capacity(), .reserve()
- toupper(), tolower()
- Lexicographic sorting of strings

Addendum: initializers (v1ch8)

- Consider a simple class with attributes:
  - **class Student {**
    - **string name;**
    - **int ID;**
- Initialize the attributes in the constructor:
  - **public:**
    - **Student(string n="", int i=0) {**
      - **name = n;**
      - **ID = i;**
    - **} }**
- Then we can create a new instance easily:
  - **Student bob("Bob", 28920);**

The constructor initializer list

- This is such a common idiom that there is a special way to initialize instance attributes:
  - **Student(string n="", int i=0) :**
    name(n), ID(i) {}
    - Attrib. name is initialized by param n
    - Attrib. ID is initialized by param i
    - Constructor body can be empty now!
- More precisely, the constructor for string name is called, with n passed as an argument: like
  - **string name(n);**

Constructor initializer and const

- The constructor initializer list is useful to set an instance attribute that's declared const:

```
class Student {
    const int ID;          // can't change ID
  public:
    Student(int i=0) : ID(i) {}
}
```

- ID is set at instantiation (via initializer list), but can't be changed after that

Calling superclass constructor

- The same syntax is used when a constructor wants to call its superclass constructor:
  - First setup superclass stuff,
    then setup our subclass-specific stuff

```
class Student : TWUPerson {
    const int ID;
public:
    Student(string name="", int i=0) :
      TWUPerson(name), ID(i) {}
}
```

  - (Assume TWUPerson has a constructor that takes a string)

Exceptions for error handling

- Recall that exceptions are used for indicating runtime errors
  - Incorrect user input or parameters
    - No memory, disk space, permissions, etc.
- When an exception is thrown:
    - Execution of the current block is terminated
    - Search for the nearest exception handler
      - **Search enclosing blocks ({})**
      - **Search down the call-stack (what code invoked the current function)**

C++ exception syntax

- In C++, any object may serve as an exception:

```
class Error {        // small object to throw
    const char* const txt;
public:
    Error(const char* const t = 0) : txt(t) {}
};
```

- Exceptions are thrown:

```
void fun() {
    Error newErr("oops!");
    throw newErr;          // throw an object
    throw Error("oops!");      // equivalent
    throw 42;              // can throw an int
}
```

Handling exceptions

- Exceptions are handled with try/catch blocks:
  - int main() {
    - try {
      - fun(); // throws an Error
- Specify the class of exception to handle:
    - } catch(Error) {
      - cout << “caught an Error!” << endl;
- Or catch(...) to handle all exceptions:
    - } catch(...) {
- First block that matches is used

Accessing the exception object

- The exception handler may take a reference to the actual object that was thrown:

```
try {
    fun();      // throws an Error()
} catch(Error& e) {
    cout << e.txt << endl;
}
```

- This is how we send auxiliary information along with the exception

Re-throwing an exception

- Inside an exception handler, just call throw; to re-raise the current exception
- Good with the catch-all handler: catch(...)
  - Clean-up our program (free used memory, close open file handles, etc.)
  - Then pass the exception on to the caller

Caution: exception class hierarchies

- Say classes Big and TooMuch are subclasses of class Trouble
- A catch(Trouble) clause will catch any exceptions that are instances of Big or TooMuch, too
- => Always put the more general exception handlers (those that catch superclasses) later in the list
- catch(...) should go last (to catch any unhandled exceptions)
- See exceptiontest example on our website

Standard exception classes

- Any object in C++ may be thrown, but the Standard C++ Library does include some standard exception classes for you to subclass:
  - **#include <stdexcept>**
- The superclass is exception; two subclasses include runtime_error and logic_error
  - **class Error : public runtime_error { ....**
- Constructors can take a string argument
  - Read it using the .what() method