CS 2704

Topic:
C++ Exceptions
Outline

• Normal control flow
• Handling errors in normal control flow
• Exceptions
  – Execution
  – Reporting errors (throwing exceptions)
  – Handling errors (catching exceptions)
• Guidelines
Normal Control Flow

• Program executes as sequence of statements
  – if conditional reached, will branch
  – if loop reached, stay in loop until condition false
  – if function call found, enter function
  – if end of function found, exit function

• If error occurs must break standard flow
Errors in Normal Flow

• Can deal with exceptional conditions by having functions return a “bad” value
• OK, if function ordinarily returns positive value, can just return negative value
• Otherwise, have to have return value for error, plus reference parameter to return value
• Error flags considered bad style
Exceptions

• Based on idea that if error occurs do not want to continue with normal flow
• Two parts to exceptions
  – throw (or raise)
  – catch (or handle)
• Separates error reporting from error handling
Call Chain

• Sequence of procedure/function calls
• In C++ begins with main
• Next function is call from main
• Thrown exceptions cause exit from call chain looking for exception handler
Exceptional Execution

• Throwing exception causes execution to look for an enclosing catch clause
• If catch is not at the level of the statement that caused exception, then exit function
• If catch is not at the level of function call that caused exception, then exit function
• If function is main and exception not caught, program will crash
Exception Values

• Exception is an object that may contain data
• Example:
  – vectors (as in linear algebra) of ints
  – want to add two vectors
  – throw exception if dimensions not same
• Usually declared inside class
  – Example would occur inside “vector” class
Example Exception Class

class DimensionMismatch {
    public:
        DimensionMismatch(int a, int b) : fst(a), scd(b) {}
        int first () { return fst; }
        int second () { return scd; }
    private:
        int fst, scd;
};
Throwing Exception

// Assumes vec has field vector<int> v
vec vec::operator+(const vec& b) {
    if (v.size() != b.v.size())
        throw DimensionMismatch(v.size(), b.v.size());
    else {
        // add code
    }
}

Catching Exception

```cpp
void example(istream& is) {
    vec a = get_vec(is); //read in first vec
    vec b = get_vec(is); //read in second vec
    try {
        vec c = a + b;
    } 
    catch (vec::DimensionMismatch) {
        cerr << "Dimensions don’t match" " endl;
    }
}
```
Catching Exception (2)

```cpp
void example(istream& is) {
    vec a = get_vec(is); // read in first vec
    vec b = get_vec(is); // read in second vec
    try {
        vec c = a + b;
    } catch (vec::DimensionMismatch dmerr) {
        cerr << "Dimensions don’t match: (" << dmerr.first() << " " << dmerr.second() << ")" << endl;
    }
}
```
Exception Handler

- Try block: `try { /* statements */ }`

- Handler: `catch ( /* class name */ ) { /* …*/}`

- Handlers
  - must follow try block, or another handler
  - Can throw “caught” exception - do something and then decide if must be handled further up call chain

- Catch-all: `catch (...) { /* code */}`
Exception Guidelines

• Exceptions deal with non-local problems
  – if can deal with problem locally, do so
• Use exceptions to handle errors
• Have main() catch and report all exceptions
• Beware of memory leaks because of exception handling