

Lecture Topics

- exception handling (continued)
 - setjmp/longjmp
 - cleanups
 - C++ exceptions

Administrivia

- [delayed: Lab #1 handout & discussion]

Setjmp/LongJump

- philosophically
 - Try to create fault-tolerance boundaries
 - But Not in every function
- common C approach uses setjmp/longjmp
 - basically allows you to discard a bunch of stack frames
 - setjmp
 - stores information about current stack & frame pointers
 - watch out for automatic storage here!
 - typically stored in static data area structure
 - avoid passing pointers through all children
 - children generate exceptions
 - by calling longjmp
 - with an error code
 - intermediate functions are irrelevant
 - longjmp
 - restores the stack pointer
 - does NOT restore the contents of the local stack frame (so changes made to local variables after setjmp are not undone)
 - does NOT magically know about other resources (e.g., dynamically allocated memory)
- several GNU tools
 - gdb, and gcc I think as well
 - use setjmp/longjmp to try to recover from a variety of low-level errors

Cleanups

- What can we do about the pesky intermediate resource problem?
 - In many cases, not much.
 - C++ will provide high-overhead but reasonably powerful help.
 - But still won't solve everything.
- For a limited set of critical activities, we can use cleanups.
 - Some of you may remember these from 391 MP2.
 - Although I think they didn't get integrated into the code with the new board.
 - Do students suffer? Trick question...
- MP2 has two major nuisance factors
 - mode X: you can't see what you're typing
 - terminal mode control: you also can't see what you're typing
- It's beneficial if a student's program
 - crashes (gets a signal),
 - panics (doesn't know how to handle something and terminates), or
 - fails an assertion (debugging check),
 - to have the screen and terminal mode restored! But how?
- generic block-structured cleanup mechanism around a linked list
 - start of block: push cleanup handler onto stack of cleanups
 - end of block: pop cleanup handler and (maybe) execute it
 - exception: pop and execute all cleanups one by one
- easy to build the same mechanism into setjmp/longjmp
- in fact, C++ provides something similar
 - stack walking rather than simply changing the pointer
 - calling destructors on all local instances of classes
 - dynamic storage is still a problem

Exception Handling in C++

- terminology
 - throw an exception
 - and catch it
- objectives [p. 385]
 - transmit arbitrary data in a type-safe manner from throw to catch
 - no added space/time to code that doesn't throw/catch
 - grouping mechanism for exceptions
 - default to correct behavior with multi-threading
 - allow interaction with other languages, especially C
 - easy to use
- additional ideal goals (not met)
 - guarantee that anything thrown is caught
 - easy to implement
- exceptions in C++ organized as class hierarchies
 - root is `std::exception` (`#include <exception>`)
 - multiple-inheritance allowed
 - DAG [directed acyclic graph] structure

```
try {  
    // something usually including function calls that might  
    // throw an exception (throw OneException ());  
} catch (OneException& e) {  
    // do something, possibly involving e  
} catch (AnotherException& f) {  
    // do something else, possibly involving f  
}  
// exceptions not in either class hierarchy are  
// implicitly passed up to this function's caller
```

- pitfall: not using a reference (e.g., “`catch (OneException e)`”)
 - creates a temporary
 - calls copy constructor and destructor (again)
 - if catch type is a superclass
 - slices (copies only part of data)
 - later virtual functions based on superclass, not original class
 - seems like an odd pitfall to leave in C++
 - could use to limit handling by calling functions
 - exception class A derives from both B and C
 - some function casts A to B and rethrows so that ancestor functions differentiate A from C
 - but why not do so explicitly instead? even add a comment?
 - might be instead
 - new people don't understand it
 - experienced people rarely make the mistake in a way that results in errors (usually it's just extra copying, etc.)