Computer Programming

Exception handling. Review

Marius Minea

marius@cs.upt.ro

10 December 2013

Why exceptions ?

Error handling is absolutely needed for any environment interaction
but it can complicate code
and obscure the main functionality

Error situations can happen anywhere in the "normal" control flow
end-of-file, read error, insufficient memory
or user-level errors (input does not match format)

Functions must be designed to return error conditions
complicates their interface

User code has to check for errors *at all points*
and propagate recovery up from from deep within processing

Exceptions as a programming language feature

Exceptions are a control flow mechanism
- different from function call/return, breaking from loops
- can transfer control across functions

Exceptions are *raised* and *caught* (handled)
- can be raised by a library function
- or by the user

Imagine a statement that says:
`setup` *exception-name* `in` *protected-code* `with` *handler-code*

When this is executed, the runtime system sets up things so if that particular exception appears (is *raised*/*thrown*) when executing *protected-code*, control is transferred to the handling code.

If nothing happens, execution proceeds with the next statement.

Syntax varies:

Java: `try` *protected-code* `catch (` *exception* `)` *handler-code*

ML: `try` *protected-code* `with` *exception* -> *handler-code*

Exceptions in C: setjmp/longjmp

```
#include <setjmp.h>
jmp_buf myexc;
...
int val;  // value transmitted with exception
if ((val = setjmp(myexc))) {
  // exception was thrown, handle here
} else {
  // protected code where exception is caught
}
...
// somewhere else, usually in another function
longjmp(myexc, val); // throws myexc with param val
```