

Lecture #27: More Special Effects—Exceptions

- Exception-handling in programming languages is a very limited form of continuation.
- Execution continues after a function call that is still active when exception raised.
- Java provides mechanism to return a value with the exception, but this adds no new complexity.

Approach I: Do Nothing

- Some say keep it simple; don't bother with exceptions.
- Use return code convention:
 - Example: C library functions often return either 0 for OK or non-zero for various degrees of badness.
- Problems:
 - Forgetting to check.
 - Code clutter.
 - Clumsiness: makes value-returning functions less useful.
 - Slight cost in always checking return codes.

Approach II: Non-Standard Return

- First idea is to modify calls so that they look like this:

```
call _f
jmp OK
code to handle exception
OK:
code for normal return
```

- To throw exception:
 - Put type of exception in some standard register or memory location.
 - Return to instruction *after* normal return.
- Awkward for the ia32 (above). Easier on machines that allow returning to a register+constant offset address [why?].
- Exception-handling code decides whether it can handle the exception, and does another exception return if not.
- Problem: Requires small distributed overhead for every function call.

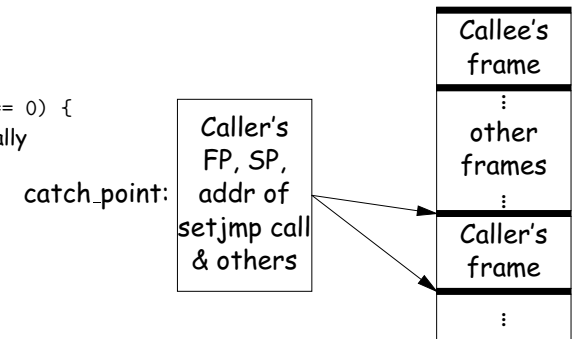
Approach III: Stack manipulation

- C does not have an exception mechanism built into its syntax, but uses library routines:

```
jmp_buf catch_point;

void Caller () {
    if (setjmp (catch_point) == 0) {
        normal case, which eventually
        gets down to Callee
    } else {
        handle exception
    }
}

void Callee () {
    ...
    // Throw exception:
    longjmp (catch_point, 42);
    ...
}
```



Approach III: Discussion

- On exception, call to `setjmp` appears to return twice, with two different values.
- Does not require help from compiler,
- But implementation is architecture-specific.
- Overhead imposed on every `setjmp` call.
- If used to implement `try` and `catch`, therefore, would impose cost on every `try`.
- Subtle problems involving variables that are stored in registers:
 - The `jmp_buf` typically has to store such registers, but
 - That means the value of some local variables may revert unpredictably upon a `longjmp`.

Approach IV: PC tables

- Sun's Java implementation uses a different approach.
- Compiler generates a table mapping instruction addresses (program counter (PC) values) to exception handlers for each function.
- If needed, compiler also leaves behind information necessary to return from a function ("unwind the stack") when exception thrown.
- To throw exception E:
 - while (current PC doesn't map to handler for E)
 - unwind stack to last caller
- Under this approach, a `try-catch` incurs no cost unless there is an exception, but
- Throwing and handling the exception more expensive than other approaches, and
- Tables add space.