Topics
Explicit Vs Implicit Control Transfer

Control abstractions studied so far are explicit:
- At the statement involving transfer of control, there is a syntactic indication of the point of transfer.
- Even for procedure calls or goto statements, there is an explicit indication of the target of transfer.

An implicit control abstraction involves:
- Constructs that enable one to set up the transfer point in advance.
- At the statement that transfers control, the target is not explicitly specified.
Explicit Vs Implicit Control Transfer

- Examples:
  - Function Pointers
  - Return Statements
  - Exceptions
Exception: An error, or more generally, an unusual condition.

Raise, Throw, Signal: A statement is said to “raise” (or “throw” or “signal”) an exception if the execution of this statement leads to an exception. ("throw" is the term used in C++/Java language descriptions, “raise” is used in OCAML.)

Catch: A catch statement is used in C++/Java to declare a handler. OCAML uses the “try ... with” statement to handle exceptions.
**Resumption model:** After the execution of the handler, control returns back to the statement that raised the exception.
- Example: signal handling in UNIX/C.

**Termination Model:** Control does not return to that statement after the handler is executed.
- Example: Exception handling in most programming languages (C++, Java and OCAML).
Exceptions are like datatypes in many ways.

- exception BadN;;

They may take arguments, such as:

- exception BadM of string * int * int * int;;

Once defined, they may be raised in functions as follows:

```ocaml
# let rec comb(n, m) = if n<0 then raise BadN
    else if m<0 then raise (BadM("M less than zero", 0, n, m))
    else if m>n then raise (BadM("M > N", 1, n, m))
    else if (m=0) || (m=n) then 1
    else comb(n-1,m) + comb(n-1,m-1);;
val comb : int * int -> int = <fun>

# comb(-1, 2);;
Exception: BadN.

# comb(9, -1);;
Exception: BadM ("M less than zero", 0, 9, -1).
```
Exception Handling in OCAML (Continued)

- Handlers can be setup using the “handle” keyword:

\[
\text{exprWithHandler} := \text{try} \ \text{expr} \ \text{with} \ \text{match} \\
\text{match} := \text{handler} \ | \ \ldots \ | \ \text{handler} \\
\text{handler} := \text{exceptionValue} -> \text{handleexpr} \\
\text{handleexpr} := \text{expr}
\]

- The meaning of expressions:
  - If the \text{expr} evaluates without raising an exception, then its value is returned as the value of \text{exprWithHandler}.
  - If the evaluation of some function \text{f} in \text{expr} returns an exception value \text{EV}, then the rest of \text{expr} is not evaluated.
    - Instead, \text{EV} is matched against the \text{exceptionValue} associated with each of the \text{handler}’s. If it matches an \text{exceptionValue}, then the corresponding \text{handleexpr} is executed.
    - If there is no match, \text{EV} is returned as the value of the expression \text{exprWithHandler}.
Uncaught exceptions are propagated up the call stack.

Example: $f$ calls $g$, which in turn calls $h$

If $h$ raises an exception and there is no handler for this exception in $h$, then $g$ gets that exception.

If there is a handler for the exception in $g$, the handler is executed, and execution continues normally after that.

otherwise, the exception is propagated to $f$. 
Exception Handling in OCAML (Continued)

- The semantics of matching exception handlers is exactly as with function definitions. In particular, when there are multiple matches, the first match is taken.

Example:

```ocaml
# let f n m = 
  try comb(n, m) with 
  | BadN -> 1 
  | BadM(s, 0, x, y) -> (print_string "BadM exception, "; print_string (s^", "); 
    print_string "raised, ignoring\n"; 1);;
val f : int -> int -> int = <fun>
# f 2 (-1);;
BadM exception, M less than zero, raised, ignoring
- : int = 1
# f (-2) 1;;
- : int = 1
# f 1 3;;
Exception: BadM ("M > N", 1, 1, 3).
```
The syntactic constructs for exceptions parallel those of OCAML, and semantics of exceptions remains essentially the same.

Syntax:

```
<blockWithHandler> ::= try <block> <match>
<match> ::= <handler> ... <handler>
<handler> ::= catch (<parameter decl>) { <block> }
```
Exception Handling in C++/Java (Continued)

Example:

```cpp
int fac(int n) {
    if (n <= 0) throw (-1); else if (n > 15) throw ("n too large");
    else return n*fac(n-1); }

void g (int n) {
    int k;
    try { k = fac (n); }
    catch (int i) { cout << "negative value invalid" ; }
    catch (char *s) { cout << s; }
    catch (...) { cout << "unknown exception" ;}
}
```

- use of g(-1) will print “negative value invalid”, g(16) will print “n too large”
- If an unexpected error were to arise in evaluation of fac or g, such as running out of memory, then “unknown exception” will be printed
Exceptions are often used to communicate error values from a callee to its caller. Return values provide alternate means of communicating errors.

Example use of exception handler:

```c
float g (int a, int b, int c) {
    float x = fac(a) + fac(b) + fac(c) ; return x ; }
main() {
    try { g(-1, 3, 25); }
    catch (char *s) { cout << "Exception ‘" << s << "’raised, exiting\n"; }
    catch (...) { cout << "Unknown exception, exiting\n"; }
}
```

We do not need to concern ourselves with every point in the program where an error may arise.
float g(int a, int b, int c) {
    int x1 = fac(a);
    if (x1 > 0) {
        int x2 = fac(b);
        if (x2 > 0) {
            int x3 = fac(c);
            if (x3 > 0)
                return x1 + x2 + x3;
            else return x3;
        }
        else return x2;
    }
    else return x1;
}

main() {
    int x = g(-1, 2, 25);
    if (x < 0) { /* identify where error occurred, print */ }
}
Use of Exceptions in C++ Vs Java

- In C++, exception handling was an after-thought.
  - Earlier versions of C++ did not support exception handling.
  - Exception handling not used in standard libraries
  - Net result: continued use of return codes for error-checking

- In Java, exceptions were included from the beginning.
  - All standard libraries communicate errors via exceptions.
  - Net result: all Java programs use exception handling model for error-checking, as opposed to using return codes.