G22.2210

Programming Languages: PL

B. Mishra
New York University.

Lecture # 8
October 30 1995

—Slide 1—

Exception Mechanisms

- *Exceptions and Exception Handlers*
  Normal program execution has to be interrupted to perform some special processing.
- **Error Conditions**
  Examples: Arithmetic Overflow, out-of-bound array references.
- **Unpredictable Conditions**
- **Tracing and Monitoring**
  (For post-mortem analysis. *Example*: Real-time systems.)
- **Two Sources**
  - Condition detected by the virtual computer.
  - Condition generated by the underlying semantics

—Slide 2—

Exception Handling Mechanism

- **Unusual Return Value**
- **Forced Branch**—'Skip return' from a subroutine.
- **Non-local Goto**—A '`goto`' with a destination outside the procedure.. The handler must appear in a lexically enclosing block.
- **Procedure Variables**—Each procedure variable is associated with a particular exception. Algol 60.
- **PL/I ON condition**—Conditions/Events are explicitly defined. Exceptions propagate over dynamic program levels.

- Standard Mechanism (Ada, ML, C++)
  Subprogram is invoked when a particular condition occurs (*NOT by explicit call.*)

$$\left.\begin{array}{c}\text{Exception}\\ \text{Signal}\end{array}\right\} \Rightarrow \text{Exception Handler}$$

Raising the Exception

—Slide 3—

Exception Handler

- Does not need a name or parameters (no explicit call).
- Contains
  a) Set of Declarations of local variables.
  b) Sequence of executable statements.
- A class of exceptions is given a name.

Example in Ada:

```
procedure SUB is
   BAD_DATA_VALUE: exception;
   --Other declarations
begin
   --Statements for normal execution
exception
   when BAD_DATA_VALUE =>     --handler1
   when CONSTRAINT_ERROR =>   --handler2
   when others =>
      --handler for all other exceptions
end;
```

—Slide 4—

Exception Handler in `C++`

- **Try:** Exceptions are processed via `try` clause.
- **Throw and Catch:**
  Exception is raised by *throwing* `<exception_name>`
  and
  is handled by *catching* `<exception_name>`

Example in `C++`:

```
try {
   statement1;
   statement2;
    . . .
   if BadCondition { throw ExceptionName };
   }
catch ExceptionName { //Handle
   } //End Exception.
```

—Slide 5—

Raising An Exception

- *By language defined primitive operation*
  (e.g. Addition may raise `Constraint_Error`)
- *By explicit invocation* (e.g. `raise` in Ada)
  `raise BAD_DATA_VALUE`
- 'Raise' statement explicitly transfers control to the associated handler.

ML example

```
exception BadDenominator of int;

fun InnerDivide(a:int, b:int): real =
   if b = 0 then raise BadDenominator(b)
   else real(a)/real(b);

fun Divide(a, b) = InnerDivide(a, b)
   handle
   BadDenominator(b) =>
      (print(b); "is bad denominator, 0 used" ; 0.0);
```

—Slide 6—

Propagating an Exception

When an exception is handled in a subprogram other than the subprogram in which it is raised, the exception is said to be **propagated** from its point of origin to the point of handling.

How does the system determine the handler?

- *Dynamic Chain* of subprogram activations leading from the subprogram that raises the exception
  - C raises exception P
  - P is handled by a handler defined in C, if there is one
  - If none then C terminates.
  - If B called C, then P is propagated to B and raised in B at the calling point in B
    . . . So on . . .
  - If no handler is found at any level, then the entire program is terminated and a standard (language-defined) handler is invoked.

—Slide 7—

After an Exception is Handled ...

Where should the control be transferred to?
Recall: There was no explicit call to the handler.

Choices: Control should be returned to—

- The point where exception was raised.
- The statement in the subprogram containing the handler where the exception was raised after being propagated.
- The caller of the subprogram containing the handler (with the callee being terminated normally). [Ada]

—Slide 8—

Implementation Details

Handling of an exception is different depending on its source.

- **Operating-System-Provided Handler**
- **Language-Provided Handler**

- Operating system exceptions are raised by hardware interrupts or traps or by support software.
- Such exceptions are handled directly by the operating systems.
- `Signals` in `C` are handled with `sigaction` function in `Unix`.

—Slide 9—

Language Specific Exceptions

- Raised by insertion of additional instructions into the executable code.
- Transfer of control to a handler in the same program is implemented by a direct jump (`goto`).
- Propagation up the dynamic chain is handled by the dynamic chain formed by the return points of the activation records in the stack.
- A special form of `return` is used to return control to the handler and raise exception in the caller again.
- Handler is invoked as an ordinary subprogram call.

—Slide 10—

Language Defined Exceptions in Ada

- `NUMERIC_ERROR`:
  Result of a predefined numeric operation is outside the implemented range or accuracy of the numeric type.
- `CONSTRAINT_ERROR`:
  Violation of range-, index- or discriminant-constraints.
  Attempt to dereference a null-valued access.
- `PROGRAM_ERROR`:
  Various implementation dependent situations
  (E.g. invoking a program unit whose body is not yet elaborated)
- `STORAGE_ERROR`:
  Dynamic storage allocated to a task is exceeded.
  Available space for an access object is exhausted.
- `TASKING_ERROR`:
  Exception is raised during a rendezvous.

—Slide 11—

Example from Ada

```
with I_O_PACKAGE;
 procedure INVERT_MATRIX is
     use I_O_PACKAGE;

     SINGULAR: exception;
     MATRIX_SIZE: INTEGER;
     type MATRIX is array (INTEGER range <>, INTEGER range <>) of FLOAT;


     procedure INVERT(M: in out MATRIX) is
         DETERMINANT: FLOAT;
         EPSILON    : constant FLOAT := 1.0E-10;
     begin
         -- Compute the determinant(M)

         if (abs DETERMINANT) < EPSILON then
              raise SINGULAR;
         end if;

         -- Compute inverse(M)
     end;

     .
     .
     .
```

—Last Slide—

Example (Contd.)

```
procedure TREAT_ONE_MATRIX(SIZE: INTEGER) is
    M: MATRIX (1 .. SIZE, 1 .. SIZE);
begin
    OBTAIN(M); INVERT(M); PRINT(M);
exception
    when SINGULAR => PUT("MATRIX IS SINGULAR");
    when others => PUT("UNEXPECTED ERROR");
end;
```

- Computation of too small a determinant raises the exception "SINGULAR," in which case the procedure INVERT is abandoned and the exception is propagated to the caller TREAT_ONE_MATRIX.
- TREAT_ONE_MATRIX is then suspended and the handler prints the error message. At this point the call is completed and the control returns to the caller.
- For any other exception the second handler is used.

```
begin
  for I in 1 .. 20 loop;
     GET (MATRIX_SIZE);
     TREAT_ONE_MATRIX (MATRIX_SIZE);
  end loop;
end;
```

[End of Lecture #8]