Compiler Practical 2013

Exceptions

Berthold Hoffmann (B. Gersdorf, T. Röfer)
hof@informatik.uni-bremen.de
Cartesium 2.48

Structure

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

1. Exception Handling
2. Exceptions in LOOP
3. Task: TRY CATCH and THROW statement
4. Bonus Task: Advanced exception handling

Exception Handling

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- Run time events (occurring rarely) that interrupt the normal control flow, but …
- May be caught, and handled so that the program can continue.

- JAVA, C#: try … catch … finally … throw

Example for Exceptions(LOOP)

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

Input	Output
a	Yes
b	b
c	c

(actually, the example is bad)

```
CLASS Main IS
  METHOD main IS x: Integer; BEGIN
    TRY
      x := helper * 1 + 0;
      WRITE x;
    CATCH 'b' DO
      WRITE 'Y'; WRITE 'e'; WRITE 's'; WRITE '\n';
    END TRY
  END METHOD
  METHOD helper : Integer IS x, y, z: Integer;
  BEGIN
    READ x;
    IF x='a' THEN THROW x+1;
    ELSE RETURN x; END IF
  END METHOD
END CLASS
```

Implementing Exceptions

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- http://en.wikipedia.org/wiki/Exception_handling

- Method 1: Dynamic Registration
  - Maintains exception frame on the stack at runtime that contains information about stack unravelling and exception handling.
  - Recommended for reading: *longjmp* und *setjmp* in (http://en.wikipedia.org/wiki/Setjmp.h)
  - Proposal for LOOP

- Method 2: Table-driven Approach
  - Generate tables at compile-time, which are indexed by the instruction counter at run time in order to do stack unravelling and exception handling.
  - State of the art for C++ compilers.

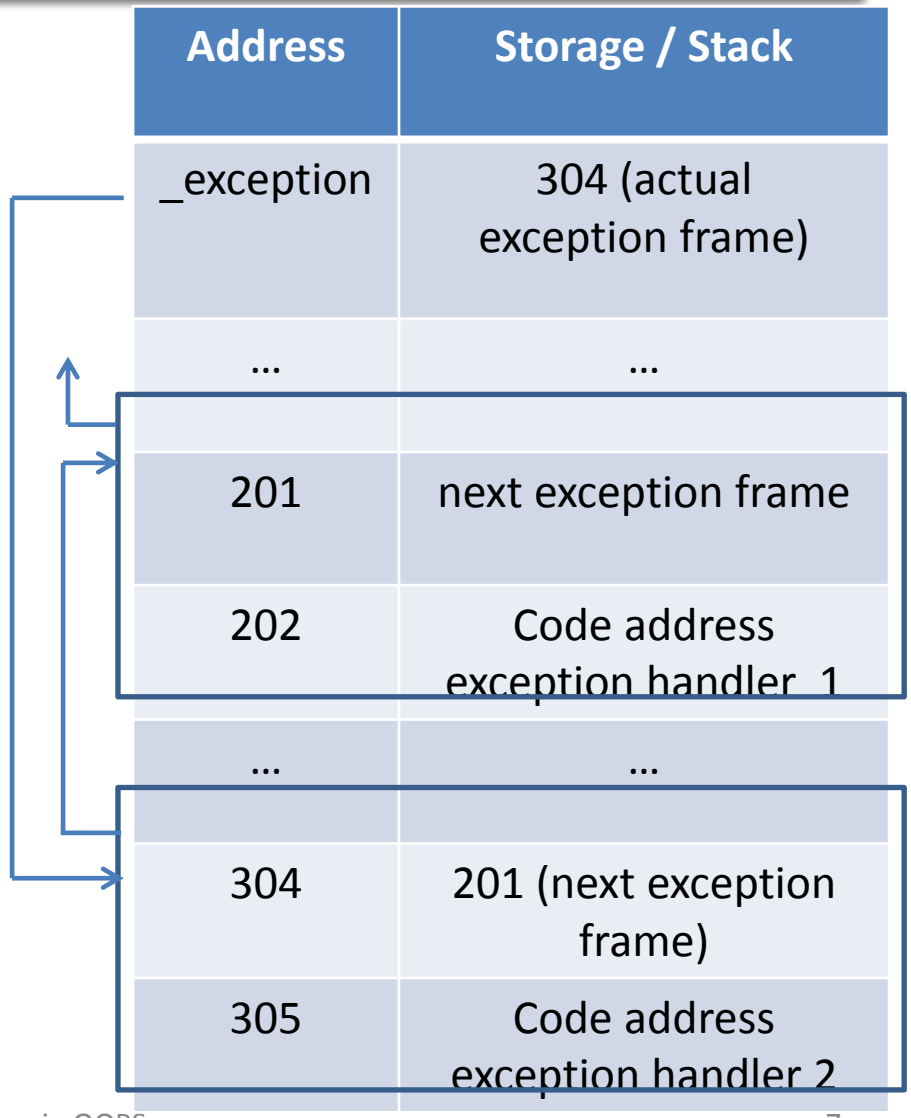
TRY CATCH Syntax (LOOP)

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

```
statements    ::={ statement }
statement     ::= READ memberaccess ';'
                | WRITE expression ';'
                | THROW expression ';'
                | TRY
                      statements
                  CATCH (number | character) DO
                    statements
                  END TRY
                | ...
```

Exception Frames in LOOP

- Global storage address *_exception* points to the actual exception frame
- The frame contains the code address of the exception handler, and a reference to the next exception frame



Using Exception Frames in LOOP

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

Principle:

- Reconstructing a previous state
  - TRY stores the state
  - In case of an exception, it is reconstructed
  - Stack contents is reconstructed to the state when state was stored (reduction).
- After reconstruction, control flow is continued
  - At another place (CATCH)

Task: TRY CATCH THROW

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- Extend lexical, syntax, context analysis
- Initial exception frame refers to final exception handler
  - Output: ABORT <character>
- Storage cell _exception points to initial exception handler
- THROW computes exception number and follows the innermost exception frame
- Entry / exit of TRY block generates / removes an exception frame
- CATCH bock checks the exception thrown in order to handle or propagate it
- RETURN needs to be adapted

10%

Bonus Task

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- Error lists
- CATCH alternatives
- Predefined exceptions
  - Division by zero throws 0
  - NULL pointer access throws 1
- Context analysis

5%

```
statements ::= { statement }
statement  ::= …
             |  THROW expression ';'
             |  TRY
                    statements
                CATCH ( number | character)
                  {',' (number | character)} DO
                  statements
                {CATCH ( number | character)
                  {',' (number | character)} DO
                  statements}
                END TRY
             |  …
```