Linden Beemer
CSPC 326
Final Project Writeup

The Extension

For this project, I have implemented try-catch and throw syntax to be used with a UDT object, allowing for a MyPL program to handle its own exceptions. In order to do this, I added new Try-Catch, and Throw syntax to allow MyPL programs to do this. In practice, this can be implemented in a MyPL program like so:

```
type MyPLError {
    var string msg = nil
}
fun void f(int i) {
    var x = new MyPLError
    x.msg = "positive int"
    if i > 0 {
        throw x
    }
}
fun void main() {
    try {
        f(1)
    } catch MyPLError x {
        print("x.msg should be positive int: " + x.msg)
    }
}
```

This syntax also allows for multiple catch statements, in order to check for different types of errors. The grammar for this extension is as follows:

```
     <throw_stmt> ::= THROW ID
     <try_stmt> ::= TRY LBRACE <stmts> RBRACE <catch_stmt>
     <catch_stmt> ::= CATCH ID ID LBRACE <stmts> RBRACE
(catch_stmt)*
     <stmt> ::= <vdecl_stmt> | <assign_stmt> | <cond_stmt> |
                <while_stmt> | <for_stmt> | <call_expr> | <ret_stmt>
          |
                <delete_stmt> | <throw_stmt> | <try_stmt>
```

It is important to note that for both throw and catch, these can only be used with user-defined-types. Using this syntax with any other type will result in an error, and was accounted for in my tests. Also important to note that if try-catch is implemented in a program, both statements must be used. Having only one or the other will result in a parser error, and this was also accounted for in my tests.

What was accomplished, what wasn't

In order to accomplish this extension, the try-catch and throw syntax needs to be recognized by the lexer, added to the abstract syntax tree in the parser, type checked by the static-checker, and finally VM instructions generated by the code generator. For the VM, I added two new commands to accomplish this extension, named "throw" and "catch." Throw stores the ID of the UDT object and its type, and catch will pull this stored information and compare it to the type named in the catch statement to see if it is the thrown exception.

I did slightly edit this project from my initial proposal, but from that edit I have accomplished everything I wanted to. This being said though, there is one small thing in my VM implementation that if I had more time, I would want to fix. In the code generator, the catch statements are implemented very similarly to if statements, with the exception of there not being a JMP command after a catch statement has successfully completed all of the statements within it. This means that every catch statement is being evaluated if the UDT matches the thrown exception, which is not very efficient. This is largely due to how I implemented the catch statement object in the AST parser. If I had made the first catch statement include a list of possible catch statements that came after it, it probably would've been easier to implement JMP commands.

The tests

In total I created 22 total tests, including four testing files segmented into the lexer, parser, static checker, and code generator, as well as four separate mypl files. This files included as follows:

- Lexer Tests (3 in total):
    - Three basic tests for each statement, ensuring that the lexer recognizes the new keywords
- Parser Tests (7 in total):
    - Three valid test cases, one for a throw statement and two for try-catch statements (testing one catch statement and two catch statements).
    - Testing for an invalid throw type (in this case, an integer)
    - Testing for a try statement with no catch statement succeeding it
    - Testing for a catch statement with no try statement preceding it
    - Testing for a catch statement with an invalid throw statement (an integer, in this case)
- Static Checker Tests (4 in total):
    - Testing for an invalid throw type of throwing a defined variable that is not a UDT (in this case, a string).
    - Testing for an invalid throw type of an undefined variable
    - Testing for if a catch statement is attempting to reuse a variable name already in use.

Linden Beemer
CSPC 326
Final Project Writeup

        - A valid test case for two catch statements of differing UDTs.
- Code Generator Tests (4 in total):
    - A valid throw statement test (with no try-catch statements).
    - A valid try-catch statement test
    - A valid double catch statement test
    - A valid try-catch statement that does not feature a throw before it (since it should be possible that no exceptions should be thrown)
- print-1.mypl
    - A file meant to test the ASTParser/pretty printer.
- code-generator-1.mypl
    - This is the second test in my code generator tests, and was created so I could run --ir on it to ensure it was printing the right VM instructions
- code-generator-2.mypl
    - Similarly to the above, this is the third test in my code generator tests, created for the same reason as the first file.
- nested-try-catch.mypl
    - This file tests that nesting try-catch statements within other try-catch statements works. Upon printing it should print (without the dashes):
        - “lexer error found”
        - “static error found”

To run the tests and MyPL files:
- For the tests:
    - bazel test --test_output=errors //:error-handling-lexer-test
    - bazel test --test_output=errors //:error-handling-parser-test
    - bazel test --test_output=errors //:error-handling-static-checker-test
    - bazel test --test_output=errors //:error-handling-code-generator-test
- For the files:
    - First run “bazel build //:mypl”
    - Then run “bazel-bin/mypl examples/[name of file here]”

In addition, on the next page you will find notes I used to ensure I was thinking through the VM commands and generating them correctly.

Linden Beemer
CSPC 326
Final Project Writeup

Working out VM commands for my implementation

```
type MyPLError {
    var string msg = nil
}
fun void f(int i) {
    var x = new MyPLError
    x.msg = "positive int"
    if i > 0 {
        throw x
    }
}
fun void main() {
    try {
        f(1)
    } catch MyPLError x {
        print("x.msg should be positive int: " + x.msg)
    }
}
```

Frame ‘f’:	Frame ‘main’:
STORE 0	PUSH 1
ALLOC [msg]	CALL f
DUP	CATCH
PUSH nil_obj	PUSH MyPLError
SETFLD msg	CMPEQ
STORE 1	JMPF 12
PUSH positive int	STORE 0
LOAD 1	PUSH x.msg should be positive int:
SWAP	LOAD 0
SETFLD msg	GETFLD msg
LOAD 0	ADD
PUSH 0	WRITE
CMGT	NOP
JMPF 16	
PUSH MyPLError	
LOAD 1	
THROW	
PUSH nil_obj	
VRET	

Linden Beemer
CSPC 326
Final Project Writeup