Zac Foteff

Dr. Bowers

CPSC 326: 01

5 May 2021

Final Project Summary

Implementing Try-Catch blocks was a very interesting challenge and provided me a lot of insight into the operation of these statements in other languages. Try-Catch blocks allow programmers to control the flow of execution in their programs, as well as manage possible exception cases. For this implementation of MyPL, users are given the capability of checking for two major errors (divide by zero and index out of bounds) as well as the ability to raise expressions to be caught by the Try-Catch block. The construct is implemented into each different component of the MyPL language.

Beginning in token.h, three new tokens were added to the MyPL language: TRY for indicating the start of a Try-Catch block, THROW for raising expressions to be handled by catch statements, and CATCH for operating after expressions or exceptions. In mypl_exception.h, I added two new errors. The first was ZERODIVISION for catching divide by zero errors at runtime, and the second is INDEXOUTOFBOUNDS for catching index overflow errors.

Moving on, in ast.h I implemented TryStmt, CatchStmt, and ThrowStmt nodes. TryStmt nodes contain a null pointer to a CatchStmt and a list of statement node pointers that make up the body of the statement. CatchStmt nodes contain a Token that stores the ID of an error (if one exists), a pointer to an Expr node for catching thrown expressions (if one exists). Finally, ThrowStmt nodes only contain an expression that will be thrown the the catch statement. Each node was also given visitor access and cleanup methods.

Semantic analysis methods were added to parser.h to build the nodes required to implement Try-Catch blocks. For TryStmt nodes, the parser ensures that a try token exists, then it iterates through all statements in the body of the Try block until a catch token is found. Once a catch token is found, a new CatchStmt is created and parsed, then a pointer to that CatchStmt is added into the TryStmt node. For CatchStmt nodes, the parser checks that a catch token exists, ensures that either an expression or ID exists between parenthesis followed by a then token, and then all statements in the catch body get parsed.

Next with type_checker.h, type-checking rules for Try-Catch blocks mainly pertain to catch and throw statements. For TryStmts, simply iterate through all statements in the node body, then call the type checker on its CatchStmt node. For the CatchStmt, if the statement is supposed to catch an expression, then ensure the expression is an int, bool, or double. For the ThrowStmt, the expression should evaluate to an int, bool, or double.

Finally, Try-Catch blocks are put into action in interpreter.h. Try-Catch blocks are implemented mainly in the TryStmt visitor, as most comparisons between values are available in the TryStmt node. First, if the CatchStmt node in the TryStmt node is non-null, then get the value

of the catch expression and store it for later comparison. Next, push a temporary environment while iterating through all statements in the TryStmt body. Using a try statement to check, if a expression is triggered by a program, the interpreter checks if the CatchStmt ID is equal to the type of error thrown. If they are the same, the try statement should stop evaluating its body, pop the environment, and allow it's CatchStmt to handle the error case. Otherwise, the interpreter should examine curr_value to ensure that the expression to be caught is never thrown. If the expression is reached in the Try body, then the program should indicate that the value has been reached and the interpreter should stop evaluating body statements in the Try body and transfer evaluation control to its CatchStmt. The CatchStmt is implemented by creating a new environment, then evaluating all the statements that exist within the CatchStmt body. ThrowStmts simply have their expressions evaluated and put into the curr_value when called by a visitor

Some future additions I'd like to include in Try-Catch blocks is the ability to raise and catch multiple values. Additonally, futher exception handling with continue and break statements would be very interesting to add.