

Ryan Reynolds: 2693018

Mina Kamel: 2667766

Loggin:

For this project we were asked to write an interpreter which uses the top-down recursive descent method and inherited/synthesized attributes to parse and evaluate a very simple programing language. The code was implemented using Python. The interpreter reads the program file and prints out the only if it is a valid sentence in the language else it prints “Syntax Errors.”

We used three dictionaries; one to store the id and value of the variable, one to store the id and type of a variable, and the third stores the result from a valid expression. Parsing was implemented through a lexan function with a declared global iterator. We then created a simple match function that checked the passed argument with our parse value lookahed that points to one value ahead at all times. Then, we started to defined each grammar rule one by one. <prog> the program function, <dec-list> this function was to store the declared variables and their values, the <type> grammar rule was to declare the two type of values allowed int and real. The type function stores the type and value of the token and it gets stored into the value list and the type list dictionary’s. Then defined statement list <stmt-list> rule which was used for to determine if the statement is valid within the language or not. The declared the different statements and the different ways to print ints, printi, and to print reals, printr. Then we moved into expression function and we checked for syntax error in the expression first if there were any syntax error the program would print syntax error and exit but if there was not any it would keep going down the list of the functions and star printing.

We used three outside predefined functions within this interpreter. The first is the isdigit() function which checks if a string contains only digits then returns a Boolean true if only digits are contained, and false if not. Instead of creating a global variable to sum, subtract, divide, multiply, or apply an exponent at each stage, we used the eval() function. The eval() function takes in an expression as a string argument. eval() parses the expression checking that it is a valid argument and then evaluates the string argument (based on python’s defined math rules) and returns the result. In our case eval() was only used to evaluate expressions of numerical values after they had been checked for syntax errors. For example: int x ; x = 1 ; printi x + 1 Our parser would check the syntax of the expression replace the x with its declared value 1 then pass exp string, 1 + 1, to the eval() function to perform the addition. Lastly we used the isinstance() function to check if the object returned from the eval expression is the correct type for the current variable or print statement. The isinstance() function takes in two arguments an object and a type. The function checks if the passed object is of the specified type (returns true) or not (returns false. This was used because our type identifier function ty() only works for string objects.

When it was time to debug our code. There were multiple issues we faced. But we took them one at the time to make it easy to fix. Some issues required lots of critical thinking and some were very easy to fix. One of the main issues we faced was how difficult it was to get the correct sequence to follow top-down recursive descent. In order to trace the errors we downloaded an IDE pycharm to visually track the descent. Instead of iterating with the match() function we elected to iterate manually to ensure we knew exactly where we were in the descent. Also, another helpful tool was going through and drawing the parse trees for all of the examples given on the project sheet. The largest source of debugging came in implementing the expression rules. Coming up with ways that checked the syntax order in the same way as the rules was difficult to maintain. Furthermore, we often thought we had implemented the correct syntax then found exceptions like forgetting a closing parenthesis was not caught. Some of these exceptions caused significant refactoring within our code.

Over all it was a fun experience going over the project. We were able to apply all the concepts we learned in class such as parsing tress, top-down recursion and grammar rules. This project tested the abilities of our programming skills and how well can we apply the concept we learned in class. The main lesson we got from working in this project was to divide your work into sections. Makes it a lot easier this way to get your work done on time. We are looking forward to more challenging tasks in class to apply all the knowledge we obtained.