Caige Middaugh
12/10/21
Programming Languages

Report Part 3

Changes:

- The main struggle with this project was having enough time to make changes. I change some of the grammars to better represent the language.
- If I had more time, I would add exponents using python's representation of exponents since I am hijacking pythons eval function to maintain order of operations. However, this change would become a major change because I would have to change a complicated list of grammars, Lexical Analyzer program, Parser, and then add support in the interpreter which could easily become a multi-hour adventure.
- Due to the complexity and early ambition of this project I did not have many changes that I can add that would be reasonable to add.
- Added nested for/while/if, etc statements. However, I still choose to not have functions defined anywhere but a global scope.
- I added a stack to implement function calls similar to how we discussed them in class, except instead of having a memory address I used a index location in the list on commands from the parser.
- Added support to cast the same type. For example, if I wanted to cast a string type to a string then it will not error out, it will simply execute the casting with no issues. Similarly, with int casting.
- Added bounds checking
- Supports return type checking.
- Function and loop symbols now maintain a body integer. The body integer represents the index of which its statement body will begin. Then end will differentiate when the function call or loop, etc will end.
- Add value field in the variable symbol to hold the value that that variable represents. For an array it will be a list in python.
- Add scoping support for if, elif, and else statements as if they are functions to replicated similar work done with the for and while loops.

Issues:

- Does not support math equations in cast statement. Casts should be single arguments only. A way to do "math" in a cast is to make a variable and do the math in that variable then cast the variable. Difficulties telling where the cast statement started and ended with how I executed the command augmentations in the parser.
- Does not support multi-dimensional arrays. I think this is a rather obvious reason why my languages do not support multi-dimensional array. However, its due to complexity and time consumption.
- Cannot nest function calls of the same function name. Example: fun(a, b, fun(1,2,3)) Due to how I have set up the symbol table it will not support dynamic access of the parameter values. That is a and b in the first fun call will be get established, but once you do the 3rd parameter for the same function call, the values of a, and b will no longer be passed instead 1 and 2 will be passed to the first fun also. A solution is to have a "General map" that stores the meaning of the function and its parameters, then when we get a function call make a copy of that general map and manipulate the variables that way. It is still

possible to implement this even with the way I have done scope in this project. However, it would become very difficult to debug, thus there are higher priority things to add such as the actual language specifications.

- No overloading due to scoping only working based off the unique naming of functions.
- The type checking when passing parameters in a function call is a bit vague. Parameters can have a 3rd type in two typed languages called “neutral” If a parameter is such a type then when passing a variable it may cause a python run time error for type issues. This is a purposeful feature to rely on the python error checking.

Symbol Table

The symbol table had some changes as mentioned above. There are 3 symbols used, Function symbol, variable symbol, and a loop symbol. The loop symbol and function symbol have the body integer associated with them. This will allow me to jump to the index where the commands should be executed. Each end is separated into its own end type such as “functionend”, “ifend” etc all behind the scenes in the parser. This will make it easy to tell when to jump back to the return index that is stored in the stack. That is all this is done sequentially.

The defining parts of symbols is done in the parsers, all the interpreter must handle is the assigning of values and querying of values. Which arguably can be much more difficult with the implementation used. Hence, why I added a value field I the variable symbol.

Summary

The general idea is to separate out the commands into a 2-dimensional list. Iterate through that list in a sequential manor and each array entry in a single command to be executed, or to represent some information to move the program to another state such as “ifend”.

I originated a bunch of keyword:

- 1regID – a regular variable
- ‘”’ – raw string if it starts with “
- 1ArrayID – a array access call
- 1functionID – a function call
- 1strCast – casting to an string
- 1intCast – casting to an int
- For loop uses assignend and condition end both to separate the assignment and condition portion of the forloop.

These keywords were the main ones used to represent the next entry in the command list is that description since this is done sequentially. The sequential process is the reason these keywords must be inserted before the actual value.