Group Number: 4

Members: Alan Zhang, Ryan Arnouk, Yingkai Zhao, Mike Wang, Sunny Nie

Project Name: FileScript

DSL Purpose
The primary purpose of our Domain-Specific Language (DSL) is to assist those with an interest in databases and programming with file searching tasks. It is designed to enable granular and conditional searches through file metadata and return aggregated results.

Part of the inspiration for the syntax comes from SQL with a combination of scripting languages like Bash and Lua.

DSL Users:
The expected user-base is someone with some sort of programming experience, especially in SQL, Bash, or Lua, and has a need for a language to perform granular file searches using metadata. This affects the design by allowing us to lean into the format and syntax of existing languages (SQL, Bash, Lua) as our users will be familiar with them.

Rich Features:

1. Loops/Conditionals
    a. WHILE loops: This allows users to efficiently loop over files and subdirectories. In addition, it can interact with mutable variables and complex data structures to control which files and subdirectories are expanded in a file tree.
    b. IF/ELSE IF/ELSE statements: These allow the user to perform filtering and aggregation of file statistics. For example, we can check if a filename matches a string in an if block, then execute another block of code after.
    c. Wildcard support for matching variables
2. Mutable variables:
    a. This allows users to perform more complex actions such as declaring a stack as an empty associative array, in which a user can push or pop elements like files to be read, which may also impact control flow.
3. Complex data structures
    a. Associative array (hashmap) that can be used to implement lists and structs: This provides the user with a convenient method for accessing various attributes of the file (name, creation date, size). Lists can be combined with loops to create complex algorithms that are capable of recursively exploring subdirectories.
    b. We will implement a form of dynamic type checking, to ensure that whenever a mismatch in types (i.e. list vs. string), an appropriate error is raise

Code Examples:

Print the names of files that exceed a certain limit:

```
files = open(“/home/foo”) // open the "foo" directory, returns a list of files
i = 0                     // initialize index for iterating

print("Files that are greater than 4096 KB in size:")

while i < len(files) do // loop over all files in the directory
       cur_file = files[i] // get the current file

       // check if the file is actually a file, with size > 4096 KB

       if cur_file["files"] == nil and cur_file["size"] > 4096 then
              print(“Name:”, cur_file["name"], “Size:”, cur_file["size"], “KB”)
       end
end
```

Example output:

```
Files that are greater than 4096 KB in size:
Name: photos.zip Size: 4877246 KB
Name: Never_Gonna_Give_You_Up.mp4 Size: 69420 KB
Name: foobar9001.txt Size: 5192 KB
```

Count the number of .txt files in the specified directory, and all subdirectories:

```
stack = {} // initialize an empty "list" to be used as a stack
push(stack, open("/home/foo")) // add a dir to begin searching from
num_files = 0
while len(stack) > 0 do
       cur = pop(stack) // pops the last file from the stack and returns it
       if cur["name"] == "*.txt" then // compare filename using wildcard operator
              num_files = num_files + 1 // increment file count
       end

       subs = cur["files"] // get the list of sub-files
       if subs != nil then // check if the file is a directory
              i = 0
              while i < len(subs) do // loop over all children
                     push(stack, subs[i]) // push each file object on the stack
              end
       end
end

print("Number of .txt files found:", num_files) // print the result
```

Example output:

```
Number of .txt files found: 42
```