Jason Lunder
April 9, 2021
CPSC 326
Semester Project
Step 2

My project request was to implement modules in MyPl. Firstly, I will define the syntax for “importing” a module, which is simply another mypl file, into the current mypl program. This will be defined in the following way:

import_stmt ::= IMPORT <module><tail><redefinition> | FROM <module><tail> IMPORT <member_name> (, <member_name>)*
member_name = STRING_VAL
module ::= STRING_VAL
tail ::= DOT “mypl” | empty
redefinition ::= AS STRING_VAL | empty

This will work similarly to the way that python implements module imports. It will work by lexing, parsing, and type checking the module when the import statement is encountered during interpretation, and the function it will preform during interpretation will be to add the functions to the symbol table in a list of functions. To do this, the maps of function names and type names in the symbol table will have to be modified such that a module name is needed to specify where to look for a function or type name, with the current file, no referencing is needed and “main” will be used as a default to specify which set of functions or type names to look in, otherwise functions and type names being used in the current file will need to be denoted as per the following grammer:

call_expr ::= <module_reference> ID LPAREN <args> RPAREN
module_reference ::= <module> DOT | empty

However, types and functions do not need to be referenced with the dot operator and module name if imported directly by using the from keyword and referencing one or more functions or types, these types and functions will be added directly to the main module’s maps, but only those functions and types will be imported.

Examples:

```
import module.mypl
#import module
#these two lines by themselves are fine, but would error if done in the same program, as there will already be a module named module in the list of maps

from math import sqrt
#note that the function id is being used, no parentheses or parameters

fun int main()
  var x = sqrt(4)
  x = module.increment(x)
end
```

Initial tests cases would include ensuring that the new grammer rules are being followed and ensuring that these changes don’t cause other grammer rules to not be followed, by adding the required keywords (import, from, as) and testing for syntax errors surrounding that. I would also have to check that the modifications to the symbol table do not cause problems.

I will need to finish the modifications to all the parts required (token, lexing, parsing, type checking, symbol table, and interpretation) by April 20.\

Game plan for completion:

- add keywords to Token.h, including IMPORT, FROM, AS
- add support for Lexer.h to recognize the parts of an import statement correctly
- implement support for Parser.h to recognize import statements as a new type of stmt
- modify symbol table to accommodate, and update all programs that use the symbol table to support this change.