PROJECT Specification:

Task Booklet 03

COS341 2020

INTRODUCTION

- Similar to your scope analyser, you shall write a program that visits your AST, but this time it shall check the type semantics of a SPL program.
- Your program must verify that a given SPL program has no type errors, or —if it does— you must find all type errors in the program and display them.
- Keep in mind that the same variable name might possibly be appear in different scopes with different types, which are then actually different variables in spite of their same name.

TYPE CHECKING RULES

- The following slides will define the grammar-based rules of your type checker.

CODE

- PROC → proc *userDefinedIdentifier* { PROG }
  - *userDefinedIdentifier* is set to type P
- DECL → TYPE NAME ;
  - NAME is set to type of TYPE
- DECL → TYPE NAME ; DECL
  - NAME is set to type of TYPE

IO/CALLS

- IO → input (VAR)
  - VAR must be of type N, B or S
- IO → input (VAR)
  - VAR must be of type N, B or S
- CALL → *userDefinedName*
  - *userDefinedName* must be of type P

TYPE

- TYPE → num
  - TYPE is set to type N
- TYPE → string
  - TYPE is set to type S
- TYPE → bool
  - TYPE is set to type B

ASSIGN

- NAME → *userDefinedIdentifier*
  - *userDefinedIdentifier* is set to type of NAME
- VAR → *userDefinedIdentifier*
  - *userDefinedIdentifier* is set to type of VAR
- ASSIGN → VAR = *stringLiteral*
  - VAR must be of type S

ASSIGN

- ASSIGN → VAR = VAR
  - both VAR must be of the same type
- ASSIGN → VAR = NUMEXPR
  - VAR and NUMEXPR must be of type N
- ASSIGN → VAR = BOOL
  - VAR and BOOL must be of type B

NUMEXPR

- NUMEXPR → VAR
  - if VAR is type N then NUMEXPR is set to type N
- NUMEXPR → *integerLiteral*
  - NUMEXPR and *integerLiteral* is set to type N
- NUMEXPR → CALC
  - if CALC is type N then NUMEXPR is set to type N

CALC

- CALC → add (NUMEXPR , NUMEXPR)
  - if both NUMEXPR is type N then CALC is set to type N
- CALC → sub (NUMEXPR , NUMEXPR)
  - if both NUMEXPR is type N then CALC is set to type N
- CALC → mult (NUMEXPR , NUMEXPR)
  - if both NUMEXPR is type N then CALC is set to type N

COND BRANCH

- COND_BRANCH → if (BOOL) then { CODE }
  - BOOL must be of type B
- COND_BRANCH → if (BOOL) then { CODE } else { CODE }
  - BOOL must be of type B

BOOL

- BOOL → eq (VAR , VAR)
  - if both VAR are of same type then BOOL is set to type B
- BOOL → (VAR < VAR)
  - if both VAR is type N then BOOL is set to type B
- BOOL → (VAR > VAR)
  - if both VAR is type N then BOOL is set to type B

BOOL

- BOOL → not BOOL*
  - if BOOL* is type B then BOOL is set to type B
- BOOL → and (BOOL* , BOOL*)
  - if both BOOL* is type B then BOOL is set to type B
- BOOL → or (BOOL* , BOOL*)
  - if both BOOL* is type B then BOOL is set to type B

Note:
BOOL and BOOL* are the same terminal symbols for grammar production purposes,
but are different only for the purpose of defining these type rules.

BOOL

- BOOL → T
  - both BOOL and T are set to type B
- BOOL → F
  - both BOOL and F are set to type B
- BOOL → VAR
  - if VAR is type B then BOOL is set to type B

COND LOOP

- COND_LOOP → while (BOOL) { CODE }
  - BOOL must be of type B
- COND_LOOP → for(VAR=0;VAR<VAR;VAR=add(VAR;1){CODE}
  - all VAR must be of type N

NOTES

- If you changed your grammar in the earlier Parser part of this project then you must find a way to adapt the given rules to you in this spec to match your modified grammar!
- **If** you allowed same name variables with different types in the same scope from the previous practical task you must decide how to handle overload cases.
  - For example you must decide whether the following shall be an “error” or valid code:

```
num a;
string a;
bool b;
b = (a < a);
output(a)
```

Should this be allowed?

Which “a”?

YOUR TASK

- You must write a program that does a type check pass over your code which has already been through scope analysis.
- Your program must write the types of all nodes of the AST where appropriate into the symbol table.
- If there are any type errors in the code you must output ALL errors. For example:
  - TYPE ERROR [line: 8, col: 3]: expect first argument of add to be of type N
  - TYPE ERROR [line: 4, col: 0]: cannot call non P type identifier

OUTPUT

- In addition to updating your symbol table you must also write, to the standard output, your tree augmented with symbol table information. For example:

```
└─25:PROG
 └─24:CODE
  ├─5:INSTR
  |  └─4:DECL
  |    ├─1:TYPE          N
  |    |  └─0:num        N
  |    └─3:NAME          N
  |      └─2:V0          N
  └─23:CODE
    └──13:INSTR
      └─12:IO
        ├─7:input
        └─10:VAR         N
          └─9:V0         N
```

You need not match the exact style of this tree but ensure you can print a tree like structure with any relevant information. You can find pseudo code for this on the next slide.

TREE PRINTING ALGORITHM

```
class Node {
  var id: Int
  var children: [Node]

  func printed(indentation: String, isLast: Bool, symbolTable: [Int: Info]) {
    print(indentation)
    if isLast {
      print(" └─")
      indentation += "   ";
    } else {
      print(" ├─")
      indentation += "|  ";
    }
    println(id, ":", symbolTable[id])

    for i in 0..<children.count {
      children[i].printed(indentation, i == children.count - 1, symbolTable)
    }
  }
}

tree.root.printed(“”, true, symbolTable)
```

ADDITIONAL NOTES

- Plagiarism is not allowed! You or your group may not use any code written by someone not within your own group.

And now:
HAPPY PROGRAMMING ☺ ☺ ☺