Term Project in PL

Shool of Software, Chung-Ang University
Programming Language

1. Project Goal

- You should design and implement an interpreter for tiny language explained in the following Lexical Structure & Grammar. Recommended tool is lex/yacc. If you do not use lex/yacc, you can implement own hand-coded lexical analyzer & parser.

2. Lexical Structure.

Token ::= ID | **int** | **float** | ReservedWord | Operator | Delimiter
ID ::= Letter (Letter | Digit)*
Letter ::= **a** | ... | **z** | **A** | ... | **Z**
Digit ::= **0** | ... | **9**
Integer ::= Digit+
Float ::= *** You need to define this ***
ReservedWord ::=**mainprog** | **var** | **array** | **of** | **function** | **procedure** | **begin** | **end** | **if** | **then** | **else** | **nop** | **while** | **return** | **print**
Operator ::= **+** | **-** | ***** | **/** | **<** | **<=** | **>=** | **>** | **==** | **!=** | **!**
Delimiter ::= **;** | **.** | **,** | **=** | **(** | **)** | **[** | **]** | **:**
Whitespace ::= <space> | <tab> | <newline> | Comment

3. Grammar

<program> ::= "mainprog" id ";" <declarations> <subprogram_declarations> <compound_statement>
<declarations> ::= "var" <identifier_list> ":" <type> ";" <declarations> | epsilon
<identifier_list> ::= id | id "," <identifier_list>
<type> = <standard_type> | "array" "[" num "]" "of" <standard_type>
<standard_type> ::= "int" | "float"
<subprogram_declarations> ::= <subprogram_declaration> <subprogram_declarations> | epsilon
<subprogram_declaration> ::= <subprogram_head> <declarations> <compound_statement>
<subprogram_head> ::= "function" id <arguments> ":" <standard_type> ";" | "procedure" id <arguments> ";"
<arguments> ::= "(" <parameter_list> ")" | epsilon
<parameter_list> ::= <identifier_list> ":" <type> | <identifier_list> ":" <type> ";" <parameter_list>

```
<compound_statement> ::= "begin" <statement_list> "end"
<statement_list> ::= <statement> | <statement> ";" <statement_list>
<statement> ::= <variable> "=" <expression> | <print_statement> |
<procedure_statement> |  <compound_statement> | "if" <expression> "then"
<statement> "else" <statement> | "while" "(" <expression> ")" <statement> | "return"
<expression> | "nop"
<print_statement> ::= "print" | "print" "(" <expression> ")"
<variable> ::= id | id "[" <expression> "]"
<procedure_statement> ::= id "(" <actual_parameter_expression> ")"
<actual_parameter_expression> ::= epsilon | <expression_list>
<expression_list> ::= <expression> | expression ","  <expression_list>
<expression> ::= <simple_expression> | <simple_expression> <relop>
<simple_expression>
<simple_expression> ::= <term> | <term> <addop> <simple_expression>
/* NOTE THE ABOVE GRAMMAR HAS BEEN CHANGED */

<term>  ::= <factor> | <factor> <multop> <term>
<factor> :: = Integer | Float | <variable> | <procedure_statement> | "!" <factor> |
<sign> <factor>
/* NOTE THE ABOVE GRAMMAR HAS BEEN CHANGED */

<sign> ::= "+" | "-"
<relop> ::= ">" | ">=" | "<" | "<=" | "==" | "!="
<addop> ::= "+" | "-"
<multop> ::= "*" | "/"
```

4. Requirements and Assumptions

- Your program should read a source file of which name is indicated by the first argument of the command invoking your program. The source file is assumed to be a plain-text-formatted file.
- You should provide reasonable error handling mechanism so that your program does not run an incorrect program. For example, if your program finds that a variable is not defined before its use, then it should display "Undefined variable : <the variable name> at line <line number in the source code>".
- You can assume that programs are defined in a single input file.
- For the variable references, the static scoping rule is used.

5. Evaluation Criteria :

- Implementation & Demo : 50%
- Documentation : 30%
- Team work : 20%

6. Milestone for project

- Presentation with your plan : 11/16
- Final Demo with Reports :12/12 or 12/14