Complier Lab(CSE4112)

Assignment#2: Writing a Parser using (F)LEX and YACC/BISON

Assignment details

This is the first assignment of a series of assignments on constructing a compiler for a programming language. We will call out language SubC. Given the grammar of SubC you are to write the F(LEX) and YACC/BISON specification file that parses code written in SubC. In this assignment, you are to generate a parser for only assignment statements. Following is the grammar for assignment statements for SubC.

Stmt_list → Stmt | Stmt_list **;** Stmt
Stmt → Variable **assignop** Expression
Variable → **id** | **id [**Expression**]**
Expression → Simple_expression | Simple_expression **relop** Simple_expressi**on**
Simple_expression → Term | Simple_expression **addop** Term
Term → Factor | Term **mulop** Factor
Factor → **id** | **num** | **(** Expression**)** | **id [**Expression**]** | **not** Factor

Tokens/Terminal symbols of this grammar are in BOLD font. Below are the specifications for the tokens

Token name	Pattern/description
Id	starts with a letter followed by letters, digits, or underscore. -- max length 5 characters - cannot be any of the keywords: "if", "else", "while", "for", "repeat", "until", "return" , "main"
Num	- Integers - real numbers that include a decimal point and at least one digit to the left and right of the point
Relop	"<", ">", "<=", ">=", "==", "!="
Addop	"+","-", "||"
Mulop	"*", "/", "%", "&&"
Assignop	"="
Not	"!"

If successful in parsing a list of statements the parser prints “success” otherwise prints appropriate error message

- You code will be evaluated based on the appropriateness & correctness of the error messages that it produces

Implementation

You are to write (F)LEX and YACC/BISON specification files(e.g., subc.l and subc.y). The parser that yacc/bison generates should be able to call the lexical analyzer generated by lex/flex and perform parsing of the code (i.e., list of assignment statements) generated by the given grammar. Your LEX specification should be able to remove any white space from the code. All **id**s should be stored in a symbol table and printed out as output.

Sample input#1

```
b= b + 1;
input[b+2]      =      b; a = input [i+2]
```

Sample output#1

```
b
input
a
Success
```

Sample input#2

```
a[]=0.2
```

Sample output#2

```
a
missing expression
```