Compilers Assignment - 2

Vishal Vijay Devadiga (CS21BTECH11061)

Instructions

Folder Structure

P2 has the following files/folders:-

- `run.sh` : Bash script to run the entire program
- `lex.y` : Lex Source File
- `parse.y` : Parser source File
- `lex.yy.c` : Generated Lex File
- `y.tab.c` : Generated Parser File
- `y.tab.h` : Generated Parser Header
- `TPP` consists of 3 public test cases with their `seq_tokens` output and `parsed` output.

- `TP2` contains the source files:
  - `seq_token_i.parsed` : Token file genarated on `i.clike` file.
  - `parser_i.parsed` : Token file genarated on `i.clike` file.
  - `overview.pdf` is the report for the Assignment.

Running the lexer

Edit the `run.sh` and set the Input file name as `i.clike` where `i` is any natural number.

The script can now be run by either right-clicking it and clicking `Run as a program` or by running `./run.sh` in a terminal.

All output files are stored in the `TP2` subfolder.

Note:-

- All input files **must** be placed in the same directory as `run.sh`.
- If the bash script is not executable, then run the following command: `chmod +x ./run.sh`
- If input file name does not exist then the program will execute indefinitely.
- If `TP2` directory does not exist, then the program will give a segmentation fault

Some implementation details and issues:

- Most of the lexer is similar to the one in assignment-1 expect for the changes in the language, and also the changes in the variable naming scheme.
- RHS of a expression, return statements and arguments of function calls can take any type of expression.
- The lexer prints the code token by token in another file, and whenever a grammar is recognized, the parser prints the type of statement.
- The parsed output does not take the placement of other tokens into account. If it recognizes a grammar rule, then it immediately prints the type of the statement.
- Currently, on an invalid statement, the program stops printing the `parsed` file at that token and does not print the rest of the line.
- The program does not do any semantic analysis, expect that it only checks whether a return statement is present in a function or not.
- The program does not use `$$` for anything. In case, semantic analysis must be done then it would be easy to use this as it is built in the parser to ease semantic analysis.
- The program does not handle scopes(braces) in inside a function body, since that would lead to a reduce-reduce conflict in my grammar.