

Assignment 2

Deepshikha CS21BTECH11016

September 22, 2023

0.1 Directory structure:

```
CS21BTECH11016
├── P2
│   ├── TP2
│   │   ├── overview.pdf
│   │   ├── parser_1.parsed
│   │   ├── parser_2.parsed
│   │   ├── parser_3.parsed
│   │   ├── seq_tokens_1.txt
│   │   ├── seq_tokens_2.txt
│   │   └── seq_tokens_3.txt
│   ├── TPP
│   │   ├── TPP0
│   │   │   ├── parser_1.parsed
│   │   │   ├── parser_2.parsed
│   │   │   ├── parser_3.parsed
│   │   │   ├── seq_tokens_1.txt
│   │   │   ├── seq_tokens_2.txt
│   │   │   └── seq_tokens_3.txt
│   │   ├── public_test_1.clike
│   │   ├── public_test_2.clike
│   │   └── public_test_3.clike
│   ├── input.clike
│   ├── lex_src.l
│   ├── lex.yy.c
│   ├── y.tab.c
│   └── yacc_src.y
```

0.2 Compilation steps :

1. Open the CS21BTECH11016.tar.gz file and see the directory structure.
2. Steps:

```
lex lex_src.l
yacc -d yacc_src.y
gcc -o yacc_src y.tab.c lex.yy.c -ll
./yacc_src < input.clike
```

3. An input.clike file is provided in the directory itself for testing your test cases.
4. The tokens file generated by these steps will be "seq.tokens.txt" and parsed file will be "parsed.parser"

5. lex_src.l contains lex program responsible for breaking the source code into a sequence of tokens.
6. lex will generate a C source code file having function yylex() which returns token.
7. Parser generated by yacc will take tokens from lex via yylex() function.
8. The tokens are identified in the grammar code using a file "y.tab.h" generated by yacc, which has constants associated with each token, basically declarations of all the tokens in the yacc program. It is a way of telling the lex program that that token is valid.
9. Grammar rules are written in yacc program which defines the syntax rules.
10. On this command:

```
yacc -v yacc_src.y
```

It will generate an output file of the parsing table

11. error handled by yacc error handling. More information given in section 0.4.

0.3 My implementation

1. I found only one issue (not really issue with my code) was that for example this input:

```
local int bar [1]( int x )
{
```

which doesn't have a closing curly brace at the end, it will terminate but won't give invalid statement as output. It will give output as:

```
local int bar [1]( int x ) : function definition
{
```

This is a syntax error not an invalid statement, so it's correct.

2. I am considering curly braces to be in next line so as to get better printing format in parsed file.
3. Error:
 - I am printing invalid statement in the parsed file after that input line is finished or can say before newline occurs.
 - I have a counter of new lines in lex program and a flag of error in yacc initialised to 0.
 - Whenever error is encountered, flag is assigned 1. And checking in the lex file, whenever newline is encountered and flag is 1, print invalid statement.
 - Also printing Syntax error in stdout (not in parsed file)
 - For the point(1) (curly braces one), printing syntax error in stdout.

4. Return statement :

- (a) Taking atleast one return statement in a function into account.
- (b) Used a int variable initialised to 0 initially.
- (c) Whenever using the return statement rule, incrementing the variable.
- (d) If value of that variable is still 0 , then no return staement. THis will give error in the stdout and wil terminate. Example :

```
local int bar [1]( int x )  
{  
  
}
```

Will give output in the stdout :-

```
Error at line 3 : syntax error  
Error:No return statement
```

- 5. Not taking number of arguments in a function, number of functions inside a class because they come under semantic checks.
- 6. No known problems from my previous assignment lexer except just adding some new key-words.