

Lexical Analyser for Tureasy in Ocaml

- Design Overview

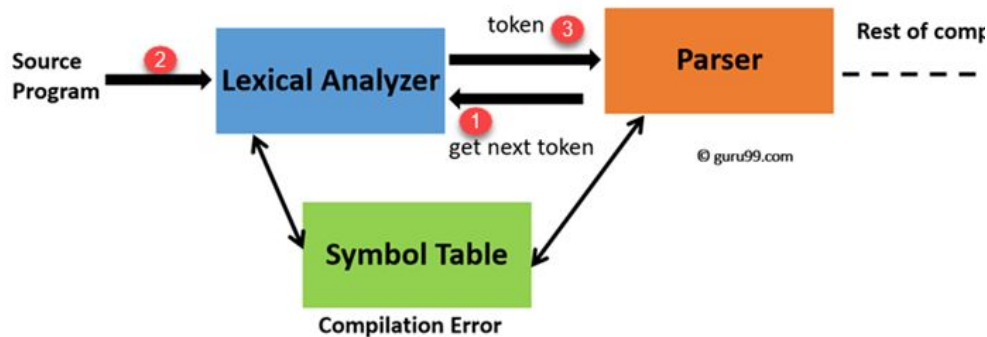


By Team 8

Jatin Kumar - CS19BTECH11036
Manoj Gayala - CS19BTECH11011
Nisha M - CS19BTECH11012
Sharanya Gupta - CS19bTECH11020
Sravanthi Reddy - ES19BTECH11007
Suraj Gubbala - CS19BTECH11042
Vedika Verma - CS19BTECH11057

The Role of lexical Analyzer

- It is the first Phase of Compiler Design.
- Basically it reads user input and produces a sequence tokens that is used in Syntax analysis by parser.



Ocaml for lexical analysis

- We used Ocaml language to write our lexer. The main advantage of using it is, we would be able to parse our complex data structure formats and hence being efficient in providing useful error messages which becomes a complex task otherwise.
- We have written the code for lexer in Ocaml in the file named `lexer.mll` . We execute the file with the command **`ocamllex lexer.mll`** that produces a lexical analyzer from a set of regular expressions with attached semantic actions, in the style of lex in the file **`lexer.ml`**.
- This file defines one lexing function per entry point in the lexer definition. These functions have the same names as the entry points. Lexing functions take as argument a lexer buffer, and return the semantic attribute of the corresponding entry point.

Overview of lexer.ml file

- This file contains the code written in Ocaml to parse the tokens (syntax , keywords, datatypes etc..)
- To give a brief idea of the non primitive datatypes that we added specific to our language include
 - `| "matrix" { MATRIX }`
 - `| "graph" { GRAPH }`
 - `| "numset" { NUMSET }`
 - `| "strset" { STRSET }`
- The tags implementation in our language as mentioned in the whitepaper requires additional tokens as below:
 - `| "#" (letter(letter|digit)* as tag) { TAG_BEGIN(tag) }`
 - `| "#!" (letter(letter|digit)* as tag) { TAG_END(tag) }`
- This was the tokenisation part of our lexical analyser where the output briefly corresponds to set of strings(Identifiers, keywords, whitespace,Integer).

Working of lexer

Correct Syntax



A screenshot of a text editor window titled 'testing.tz' with the path '~/compilers-2-project-team-8-aug21/de...'. The code is as follows:

```
1 struct abc {  
2   matrix M;  
3   $ graph c;  
4 };  
5  
6 int a = 0, b = 1+4;  
7 $*  
8 string message = "hello!";  
9 *$
```

The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 9, Col 3', and 'INS'.

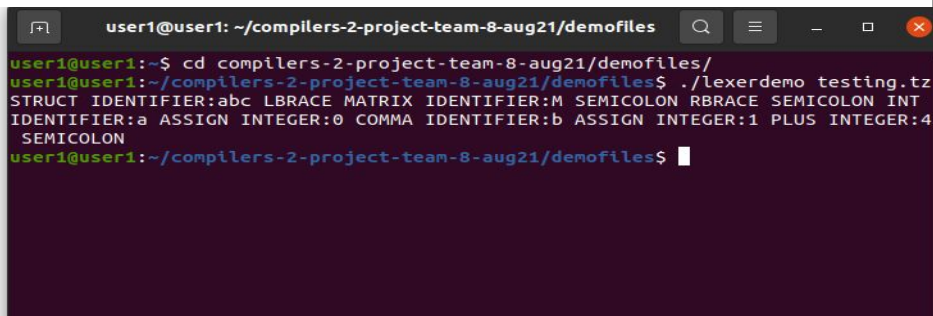
Incorrect Syntax



A screenshot of a text editor window titled 'testing.tz' with the path '~/compilers-2-project-team-8-aug21/de...'. The code is as follows:

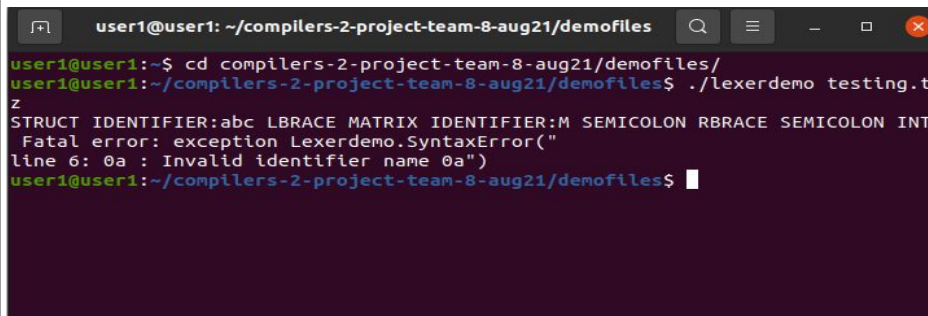
```
1 struct abc {  
2   matrix M;  
3   $ graph c;  
4 };  
5  
6 int 0a = 0, b = 1+4;  
7 $*  
8 string message = "hello!";  
9 *$
```

The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 6, Col 6', and 'INS'.



A screenshot of a terminal window with the prompt 'user1@user1: ~/compilers-2-project-team-8-aug21/demofiles'. The user runs the command './lexerdemo testing.tz'. The output is:

```
STRUCT IDENTIFIER:abc LBRACE MATRIX IDENTIFIER:M SEMICOLON RBRACE SEMICOLON INT  
IDENTIFIER:a ASSIGN INTEGER:0 COMMA IDENTIFIER:b ASSIGN INTEGER:1 PLUS INTEGER:4  
SEMICOLON  
user1@user1:~/compilers-2-project-team-8-aug21/demofiles$
```



A screenshot of a terminal window with the prompt 'user1@user1: ~/compilers-2-project-team-8-aug21/demofiles'. The user runs the command './lexerdemo testing.tz'. The output is:

```
STRUCT IDENTIFIER:abc LBRACE MATRIX IDENTIFIER:M SEMICOLON RBRACE SEMICOLON INT  
Fatal error: exception Lexerdemo.SyntaxError("line 6: 0a : Invalid identifier name 0a")  
user1@user1:~/compilers-2-project-team-8-aug21/demofiles$
```