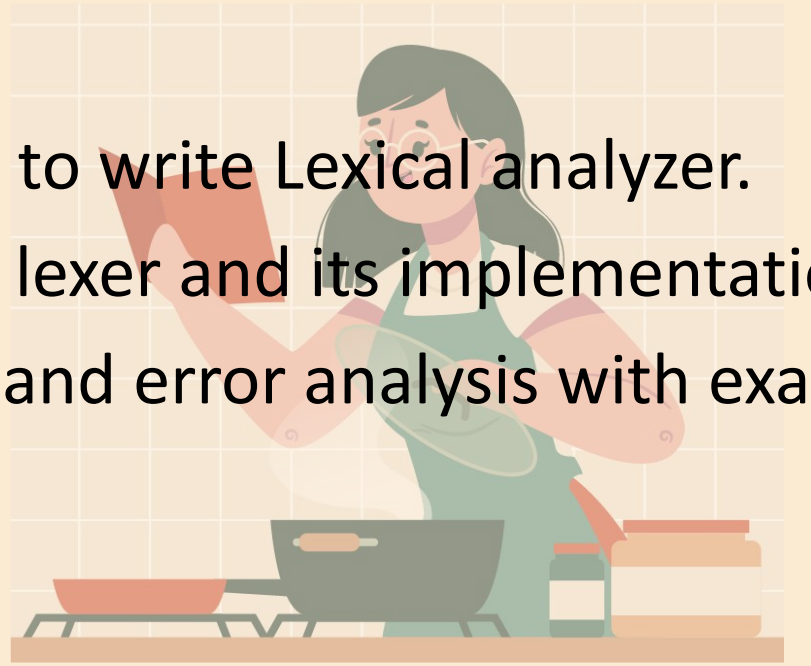
MICHELIN

A Code which Cooks

Michelin’s Lexical Analyzer

Contents

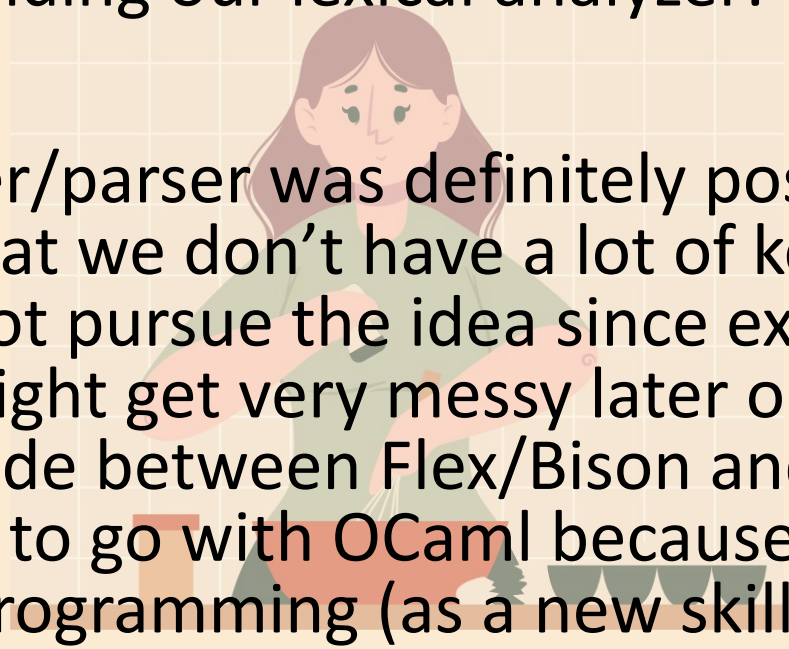
The key points to take a look at, in this phase are:



1. Usage of OCaml to write Lexical analyzer.
2. Structure of the lexer and its implementation
3. Lexer's working and error analysis with examples.

Adoption from OCaml

The following are some of the reasons why we chose to go ahead with OCamllex for building our lexical analyzer:-



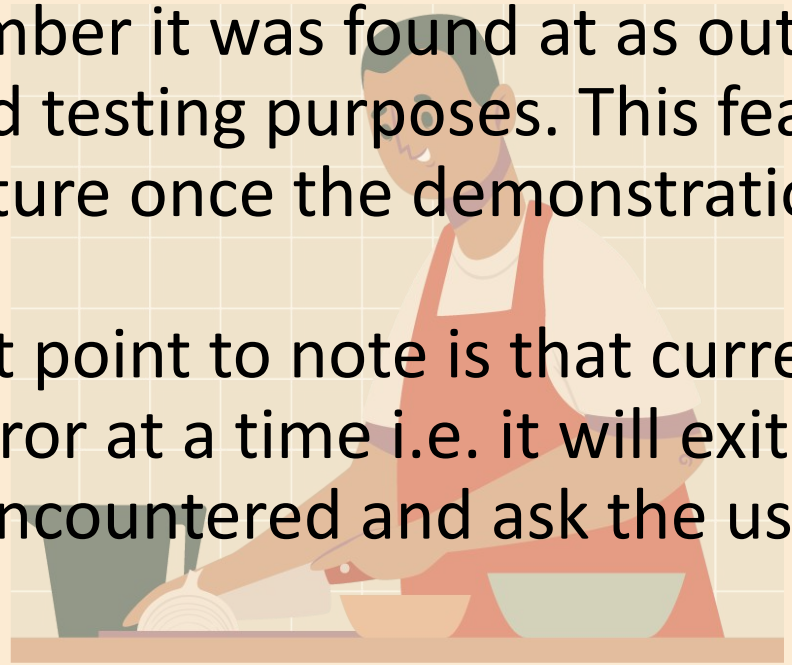
- A hand-coded lexer/parser was definitely possible to build, especially given that we don’t have a lot of keywords right now; however we did not pursue the idea since extending the features of the language might get very messy later on.
- It was hard to decide between Flex/Bison and OCamllex/yacc but we finally decided to go with OCaml because we wanted to try some functional programming (as a new skill).

Lexing the Lexer!

- The lexer.mll file has the information regarding the type of each token which our language recognizes.
- Tokens such as digits,characters,integer constants,decimal constants and string literals are represented using regular expressions which is a functionality provided by ocamllex.
- Then we go on about doing pattern matching of the tokens and return the corresponding token name and line number the token was found at for each valid lexeme of our language.
- The main.ml then sends the input code to the lexer and then reads the lexbuf which is a buffer provided by ocamllex for tokens.

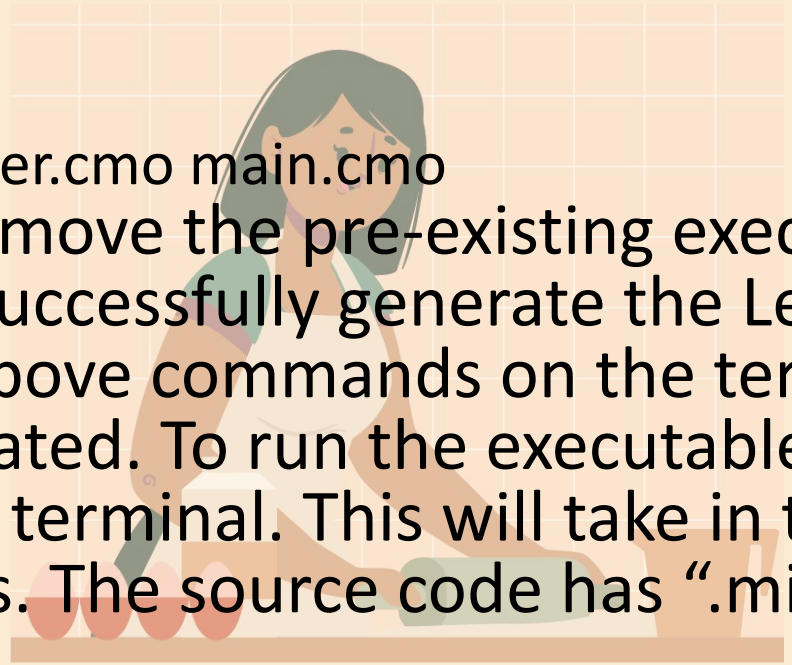
Key Points to Notice

- Currently the main.ml file reads the token and prints the token name and line number it was found at as output to stdout just for demonstration and testing purposes. This feature will be removed in the future once the demonstration of the lexer is completed.
- Another important point to note is that currently our lexer can detect only one error at a time i.e. it will exit out of the code when an error is encountered and ask the user to fix it.



Running the Lexer

- To generate the Lexical Analyser follow the below instructions:
  - ocamllex Lexer.mll
  - ocamlc -c Lexer.ml
  - ocamlc -c main.ml
  - ocamlc -o main Lexer.cmo main.cmo
- It is important to remove the pre-existing executables and object files(.cmo files) to successfully generate the Lexical Analyser.
- After running the above commands on the terminal, the “main” executable is generated. To run the executable enter “./main <filename>” on the terminal. This will take in the source code and generate the tokens. The source code has “.miche” extension.



Example codes

Input-1

main.ml | test.miche | out.txt | Lexer.mll

Michelin-Compiler > test.miche

```
1   Glass g.
2   Ingredient ing1("Water", 100, volume, g).
3   Bowl b.
4   b~>add(ing1).
5   b~>boil(5, 60).
6   g~>transfer(b).
7   serve g.
8   ready.
9
```

Example codes

Output-1

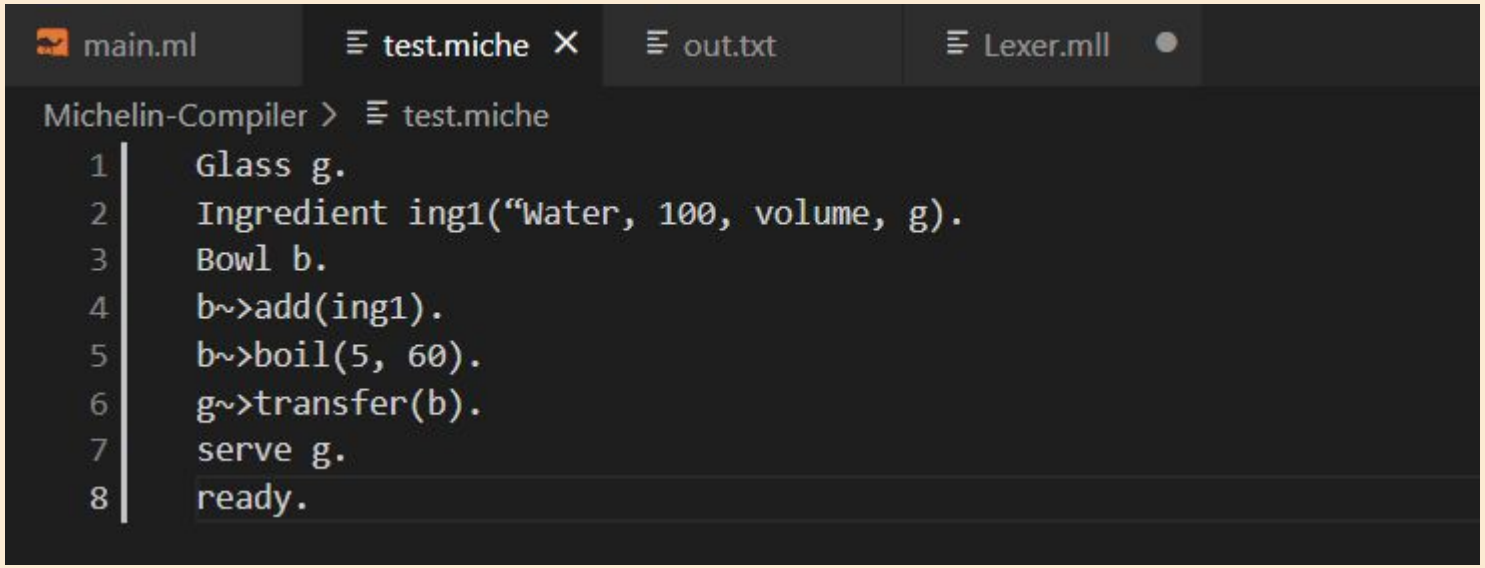
main.ml | test.miche | out.txt | Lexer.mll

Michelin-Compiler > out.txt

```
The token Glass which is a reserved keyword is found at line 1
The token g which is an IDENTIFIER is found at line 1
The token . is found at line 1
The token Ingredient which is a reserved keyword is found at line 2
The token ing1 which is an IDENTIFIER is found at line 2
The token ( is found at line 2
The token "Water" which is a STRING_LITERAL is found at line 2
The token , is found at line 2
The token 100 which is an INTEGER_CONSTANT is found at line 2
The token , is found at line 2
The token volume which is a reserved keyword is found at line 2
The token , is found at line 2
The token g which is an IDENTIFIER is found at line 2
The token ) is found at line 2
The token . is found at line 2
The token Bowl which is a reserved keyword is found at line 3
The token b which is an IDENTIFIER is found at line 3
The token . is found at line 3
The token b which is an IDENTIFIER is found at line 4
The token ACCESS(~>) which is an operator is found at line 4
The token add which is an IDENTIFIER is found at line 4
The token ( is found at line 4
The token ing1 which is an IDENTIFIER is found at line 4
The token ) is found at line 4
The token . is found at line 4
The token b which is an IDENTIFIER is found at line 5
The token ACCESS(~>) which is an operator is found at line 5
The token boil which is an IDENTIFIER is found at line 5
The token ( is found at line 5
The token 5 which is an INTEGER_CONSTANT is found at line 5
The token , is found at line 5
The token 60 which is an INTEGER_CONSTANT is found at line 5
```

Example codes

Input-2



Example codes

Output-2

main.ml | test.miche | out.txt | Lexer.mll

Michelin-Compiler > out.txt

```
1  The token Glass which is a reserved keyword is found at line 1
2  The token g which is an IDENTIFIER is found at line 1
3  The token . is found at line 1
4  The token Ingredient which is a reserved keyword is found at line 2
5  The token ing1 which is an IDENTIFIER is found at line 2
6  The token ( is found at line 2
7  Invalid string literal found at line number 2.
8
```

Team Details

Sai Sidhardha Grandhi
CS19BTECH11050
Project Manager
GitHub ID: G-Sidhardha

Krishn Vishwas Kher
ES19BTECH11015
Language Guru
GitHub ID: KrishnKher

Mukkavalli Bharat Chandra
ES19BTECH11016
System Architect
GitHub ID: chandra3000

Sujeeth Reddy
ES19BTECH11022
System Integrator
GitHub ID: Sujeeth13

Sree Prathyush Chinta
CS19BTECH11043
Tester
GitHub ID: Prathyush-1886

Vemulapalli Aditya
CS19BTECH11025
System Integrator
GitHub ID: VEMULAPALLI-ADITYA

Praneeth Nistala
CS19BTECH11054
Tester
Github ID: Praneeth-Nistala

Thank You