Practical Assignment - Lexical Analysis

Ian Nery Bandeira - 17/0144739

University of Brasília - (UnB), P.O. Box 4466, 70910-900
Brasília-DF, Brazil
iannerybandeira@gmail.com

1 Motivation

The Translators subject taught at the University of Brasilia targets a translator's study and its related elements and implementation. Thus, this subject's primary assignment concerns implementing a translation process for a C language subset combined with complementary methods for handling set theory processes, as the set primitive and operators that cover commonly employed methods for manipulating sets, according to the professor's language description[5]. This paper covers the first phase of a translator implementation for the previously mentioned C language subset, its lexical analysis.

2 Description

To perform the lexical analysis, regular expressions were created that covered the keywords described in the language description, such as digits, valid characters for identifiers, operators (relational, logical, arithmetic, and assignment), comments, and flow control commands. Following this, an analysis was performed regarding the additional tokens that compose the language, such as curly braces, brackets, parentheses, quotation marks, semicolons, and commas.

In order to identify errors with line and column tracking, two variables were also created for this purpose, conducting this verification line by line. Anything that was not described in the regular expressions defined for the keywords above was considered an "unexpected character" error. No other data structure was employed to implement the lexical analysis. The types, identifiers, and delimiter tokens will be used in the upcoming project to delimit scopes and generate the symbol table, an important resource for the project's syntactic analysis.

The lexemes that regulate the language grammar, along with the grammar description, are detailed in Appendix A.

3 Test Files

The test files are stored in the *tests/* folder.

Inside the folder are 2 files containing correct lexemes within the C language subset:

1. t_correct01.c,
2. t_correct02.c;

Along with 2 files containing incorrect lexemes:

1. t_error01.c, which contains unidentified characters in line 2 column 15,
2. t_error02.c, which contains unidentified characters in line 2 columns 16, 17, 18, 20 and 29, and in line 9 column 21.

4 Compilation and Execution Instructions

The lexical analysis algorithm was compiled and executed with the following system specifications:

- OS Version → Ubuntu 20.04.1 LTS
- Flex Version → flex 2.6.4
- GCC Version → gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1 20.04)

To compile the program, write the following on a terminal:

```
$ flex clang_lexical.l && gcc lex.yy.c
```

To run the executable with the test files, write the following on a terminal:

```
$ ./a.out tests/<test_file_name>
```

References

1. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers, principles, techniques. Addison wesley **7**(8), 9 (1986)
2. Bagaria, J.: Set Theory. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, spring 2020 edn. (2020)
3. Heckendorn, R.: A grammar for the c- programming language (version s21). University of Idaho (2021), http://marvin.cs.uidaho.edu/Teaching/CS445/c-Grammar.pdf, visited on 2021-02-16
4. Kakade, S.: C tokens. Savitribai Phule Pune University (2020), http://studymaterial.unipune.ac.in:8080/jspui/bitstream/123456789/5889/1/C%20Tokens.pdf, visited on 2021-02-16
5. Nalon, C.: Trabalho prático - descrição da linguagem (2021), https://aprender3.unb.br/mod/page/view.php?id=294131, visited on 2021-02-16

A Language Grammar

The lexemes, described into Flex Regular Expressions, were created and added to Table 1 according to the valid tokens in the C programming language [4], and the Language Grammar was developed using the C- Grammar [3] as reference, once they are both C language simplified subsets.

1. $program \rightarrow declarationList$ **MAIN {** $localDeclarations\ stmtList$ **}**
2. $declarationList \rightarrow declarationList\ declaration \mid declaration$
3. $declaration \rightarrow varDeclaration \mid funcDeclaration \mid comment$
4. $comment \rightarrow$ **INLINE_COMMENT** $(digit \mid$ **ID**$)^*$
5. $varDeclaration \rightarrow$ **TYPE ID ;**
6. $funcDeclaration \rightarrow$ **TYPE ID (** $params$ **)** $compoundStmt$
7. $params \rightarrow params$ **,** $param \mid param \mid \varepsilon$
8. $param \rightarrow$ **TYPE ID**
9. $compoundStmt \rightarrow$ **{** $localDeclarations\ stmtList$ **}** | **;**
10. $localDeclarations \rightarrow localDeclarations \mid varDeclaration \mid \varepsilon$
11. $stmtList \rightarrow stmtList\ primitiveStmt \mid \varepsilon$
12. $primitiveStmt \rightarrow exprStmt \mid compoundStmt \mid condStmt$
    $\mid iterStmt \mid returnStmt \mid setStmt$
13. $exprStmt \rightarrow expression$ **;**
14. $compoundStmt \rightarrow$ **{** $localDeclarations\ stmtList$ **}**
15. $condStmt \rightarrow$ **if (** $expression$ **)** $compoundStmt$
    $\mid$ **if (** $expression$ **)** $compoundStmt$ **else** $compoundStmt$
16. $iterStmt \rightarrow$ **for (** $expression$ **;** $relationalExp$
    ; $expression$**)** $compoundStmt$
17. $returnStmt \rightarrow$ **return** $expression$ **;**
18. $setStmt \rightarrow pertOP \mid typeOP$**;** $\mid addOP$**;** $\mid remOP$**;** $\mid selectOP$**;** $\mid forallOP$
19. $pertOP \rightarrow expression$ **in set**
20. $typeOP \rightarrow$ **is_set(ID)**

21. $addOP \rightarrow$ **add(** $pertOP$ **)**
22. $remOP \rightarrow$ **remove(** $pertOP$ **)**
23. $selectOP \rightarrow$ **exists(ID** in **set)** | **exists(ID , set)**
24. $forallOP \rightarrow$ **forall(** $pertOP$ **)** $compoundStmt$
25. $expression \rightarrow$ **ID ASSIGN_OP** $expression \mid simpleExp \mid constOP \mid inOP \mid outOP$
26. $simpleExp \rightarrow logicalExp \mid relationalExp$
27. $constOP \rightarrow$ **INT** | **FLOAT** | **EMPTY**
28. $inOP \rightarrow$ **IN (ID)**
29. $outOP \rightarrow$ **OUT (OUTCONST)**
30. $logicalEx \rightarrow$ $simpleExp$ **BIN_LOGICAL_OP** $simpleExp$
    | **UN_LOGICAL_OP** $simpleExp$
31. $relationalExp \rightarrow arithExp$ **RELATIONAL_OP** $sumExp \mid sumExp$
32. $sumExp \rightarrow sumExp\ sumOP\ mulExp \mid mulExp$
33. $sumOP \rightarrow + -$
34. $mulExp \rightarrow mulExp\ mulOP\ factor \mid factor$
35. $mulOP \rightarrow * /$
36. $factor \rightarrow$ **ID** $\mid fCall \mid$ **(**$simpleExp$**)** $\mid constOP \mid typeOP \mid addOP \mid remOP \mid selectOP$
37. $fCall \rightarrow$ **ID (**$params$**)**

Table 1. Labels and regular expressions for the language lexemes

Label	Regular Expression(Flex RegEx)
digit	[0-9]
MAIN	**main()**
ID	[a-zA-Z_][a-z0-9A-Z]*
EMPTY	EMPTY
KEYWORD	**if**|**else**|**for**|**forall**|**is_set**|**return**|**in**|**add**|**remove**|**exists**
ARITHMETIC_OP	[+ ∗ /−]
BIN_LOGICAL_OP	[&]{2}|[|]{2}|[!]
UN_LOGICAL_OP	[!]
RELATIONAL_OP	[=]{2}|(! =)|(>=)|(<=)|[>]|[<]
ASSIGN_OP	[=]{1}
INLINE_COMMENT	[/]{2}.∗
TYPE	**int**|**float**|**set**|**elem**
IN	**read**
OUT	**write**|**writeln**
OUTCONST	**char**|**string**
INT	**DIGIT**+
FLOAT	**DIGIT**+ . **DIGIT**+
STR_DELIM	”
CHAR_DELIM	‘