CS 4013 Compiler Construction
Lexical Analysis

Michael Collett

September 2018

INTRODUCTION

This lexical analyzer is written in C for the computer language, Pascal. The make file is included and the lexical analyzer can be compiled by simply typing make in the working directory of a terminal. The code can then be run by typing ./Lex. The lexical analyzer makes 'cuts' in the source program and separates the pascal file into lexemes. Each lexeme has a corresponding token and attribute that are represented as integers. These tokens are stored in a linked list and are outputted as a file to be used for future pieces of the compiler. The lexical analyzer also prints out a listing file, which contains the source program, and any lexical errors that were caught.

METHODOLOGY

Before starting this project, I had very little experience with C. In order to acquire the necessary c programming skills to finish this project, I read "The C Programming Language". I used a barebones IDE, atom with no c packages, and the command line in order to better understand all parts of the language.

In class and in the book, we learned about the necessary machines that are used to parse files. By implementing those machines in c code, I was able to create a lexical analyzer that can recognize a variety of character strings, characters, digits and errors. The machines are: whitespace, idres, catchall, numbers, and relop.

The machines are ordered as to obey two rules. The first being that superstrings must come before substrings. This is exemplified by the fact that the machine checks for long reals before short reals and, likewise, short reals before integers. The second rule that it obeyed is that the machine is ordered for efficiency with whitespace and idres coming first.

If errors are encountered while making cuts, the errors are handled in two ways. The error is tokenized and put into the token list, and the error is printed below the corresponding line in the listing file.

IMPLEMENTATION

The program begins by loading in the reserved words file that contains all of pascals reserved words that we're considering for this project. It proceeds by loading in the source file and defining all the necessary tokens for our project. At this point, a linked list of reserved words and a linked list of tokens are initialized. The program opens two more files to be written to: the listing file and the token file, both of which are text files.

At this point, our machine of machines is ready to read, cut and consume the Pascal source file. The machine reads the file line-by-line and assumes no line is larger than 72 characters. A front and back index of each line are kept in order to track which lexemes have been cut and to backtrack if necessary. The first machine encountered is the whitespace machine which simply moves the front and back index past any whitespace. The second machine encountered is the idres machine. This machine takes in any letter followed by any number of strings and letters. If the resulting lexeme is larger than ten characters, an error is reported. Once that lexeme is cut, we check the reserved word table to see if it is a reserved word. We then check a list of id's. If the id is present then we can tokenize its values, but if the id is not present we add the id to the list. Next we encounter the catchall machine, which simply checks whether the next character is one of a multitude of characters we have defined. These could be mulops, addops, assignops, punctuation, groupings, array and type. After the catchall, comes the number machine. This machine recognizes integers, short reals, and long reals. Throwing errors when: the integer portion of a number leads with a zero or is larger than five characters (ten if an integer), the decimal portion has a trailing zero or is larger than five characters, or the exponent leads with a zero or is larger than two characters. The last machine is the relop machine which recognizes less than, less than or equal to, not equal to, equal to, greater than and greater than or equal to.

If a machine doesn't recognize anything, it blocks and sends the job off to the next machine. If all the machines block, an unrecognized symbol error is thrown.

DISCUSSION AND CONCLUSIONS

The project was wonderful because it made me learn C. I am very glad that I chose to make my compiler front end in this language.

A large mistake I made was implementing the short real, long real and integer machines all at once. By combining them, I thought that it would be more efficient, however I hadn't realized the cost it would have to the ease of programming such a machine as well as the code's readability.

One thing that I'll alter for the parser is the ability to specify the source program via the command line. As well as an option to specify the output file names and the reserved word file.

REFERENCES

The C Programing Language By Brian Kernighan and Dennis Ritchie

Standard Libraries in C
Linked List Description on TutorialsPoint for C

Reserved Word File
```
program 30  0
var 31  0
array 32  0
of  33  0
integer 34  0
real  35  0
function  36  0
procedure 37  0
begin 38  70  0
end 38  71  0
if  39  72  0
then  39  73
else  39  74
while 40  75
do  40  76
or  41  77
div 41  78
mod 41  79
and 41  80
not 42  0
```

Token File (also in appendix 2)
```
#ifndef TOKENS
#define TOKENS
//Block Number
#define BLOCK -1
//Tokens
#define ID 1
#define GROUPING 2
#define ASSIGNOP 3
#define PUNCTUATION 4
#define ARRAY 5
#define TYPE 6
#define RELOP 7
#define MULOP 8
#define ADDOP 9
#define INT 10
#define SREAL 11
#define LREAL 12
//Attributes
#define LPAR 81
```

```
#define RPAR 82
#define LBRACK 83
#define RBRACK 84
#define COMMA 85
#define SEMICOLON 86
#define PERIOD 87
#define LTEQ 88
#define NEQ 89
#define LT 90
#define EQ 91
#define GTEQ 92
#define GT 93
#define MULT 94
#define DIV 95
#define ADD 96
#define SUB 97
//Errors
#define LEXERR 99
#define IDTOOLONG 100
#define UNRECOGSYMB 101
#define LEADZERO 102
#define DECTRAILZERO 103
#define EXPLEADZERO 104
#define INTTOOLONG 105
#define SREAL1TOOLONG 106
#define SREAL2TOOLONG 107
#define LREAL1TOOLONG 108
#define LREAL2TOOLONG 109
#define LREAL3TOOLONG 110
//eof
#define EOFTOKEN 20
#endif
```

APPENDIX I: SAMPLE INPUTS AND OUTPUTS

ERROR PASCAL FILE

```
program errors();
abcdefghijk
12345678901 0123
123456.1 1.123456 1.10 01.1
123456.1E2 1.123456E1 1.1E123
1.10E1 01.1E1 1.1E01
#!@$%^
```

RESULTING LISTING FILE

```
1.   program errors();
2.   abcdefghijk
LEXERR:     IDTOOLONG                   abcdefghijk
3.   12345678901 0123
LEXERR:     INTTOOLONG                  12345678901
LEXERR:     LEADZERO                    0123
4.   123456.1 1.123456 1.10 01.1
LEXERR:     SREAL1TOOLONG               123456.1
LEXERR:     SREAL2TOOLONG               1.123456
LEXERR:     DECTRAILZERO                1.10
LEXERR:     LEADZERO                    01.1
5.   123456.1E2 1.123456E1 1.1E123
LEXERR:     LREAL1TOOLONG               123456.1E2
LEXERR:     LREAL2TOOLONG               1.123456E1
LEXERR:     LREAL3TOOLONG               1.1E123
6.   1.10E1 01.1E1 1.1E01
LEXERR:     DECTRAILZERO                1.10E1
LEXERR:     LEADZERO                    01.1E1
LEXERR:     EXPLEADZERO                 1.1E01
7.   #!@$%^
LEXERR:     UnrecognizedSymbol          #
LEXERR:     UnrecognizedSymbol          !
LEXERR:     UnrecognizedSymbol          @
LEXERR:     UnrecognizedSymbol          $
LEXERR:     UnrecognizedSymbol          %
LEXERR:     UnrecognizedSymbol          ^
```

RESULTING TOKEN FILE

Line No.	Lexeme	Token Type	Attribute
1	program	30	0
1	errors	1	0
1	(	2	81
1	)	2	82
1	;	4	86
2	abcdefghijk	99	100
3	12345678901	99	105
3	0123	99	102
4	123456.1	99	106
4	1.123456	99	107
4	1.10	99	103
4	01.1	99	102
5	123456.1E2	99	108
5	1.123456E1	99	109
5	1.1E123	99	110
6	1.10E1	99	103
6	01.1E1	99	102
6	1.1E01	99	104
7	#	99	101
7	!	99	101
7	@	99	101
7	$	99	101
7	%	99	101
7	^	99	101
-1	EOF	20	0

WORKING PASCAL FILE

```
program example(input, output);
var x, y: integer;
function gcd(a, b: integer): integer;
begin
  if b = 0 then gcd := a
  else gcd := gcd(b, a mod b)
end;

begin
  read(x,y);
  write(gcd(x,y))
end.
```

RESULTING LISTING FILE

```
1.   program example(input, output);
2.   var x, y: integer;
3.   function gcd(a, b: integer): integer;
4.   begin
5.     if b = 0 then gcd := a
6.     else gcd := gcd(b, a mod b)
7.   end;
8.
9.   begin
10.    read(x,y);
11.    write(gcd(x,y))
12.  end.
```

RESULTING TOKEN FILE

Line No.	Lexeme	Token Type	Attribute
1	program	30	0
1	example	1	0
1	(	2	81
1	input	1	1
1	,	4	85
1	output	1	2
1	)	2	82
1	;	4	86
2	var	31	0
2	x	1	3
2	,	4	85
2	y	1	4
2	:	6	0
2	integer	34	0
2	;	4	86
3	function	36	0
3	gcd	1	5
3	(	2	81
3	a	1	6
3	,	4	85
3	b	1	7
3	:	6	0
3	integer	34	0
3	)	2	82
3	:	6	0
3	integer	34	0
3	;	4	86
4	begin	38	70
5	if	39	72

```
5               b                       1               7
5               =                       7               91
5               0                       10              0
5               then                    39              73
5               gcd                     1               5
5               :=                      3               0
5               a                       1               6
6               else                    39              74
6               gcd                     1               5
6               :=                      3               0
6               gcd                     1               5
6               (                       2               81
6               b                       1               7
6               ,                       4               85
6               a                       1               6
6               mod                     41              79
6               b                       1               7
6               )                       2               82
7               end                     38              71
7               ;                       4               86
9               begin                   38              70
10              read                    1               8
10              (                       2               81
10              x                       1               3
10              ,                       4               85
10              y                       1               4
10              )                       2               82
10              ;                       4               86
11              write                   1               9
11              (                       2               81
11              gcd                     1               5
11              (                       2               81
11              x                       1               3
11              ,                       4               85
11              y                       1               4
11              )                       2               82
11              )                       2               82
12              end                     38              71
12              .                       4               87
-1              EOF                     20              0
```

APPENDIX II: PROGRAM LISTINGS

(NEXT PAGE)