Scanner for C- language

Course Project - 1 (July 2020 - Dec 2020)
CS305 Compiler Design Lab
National Institute of Technology, Karnataka

SUBMITTED TO:
Prof. P. Santhi Thilagam
CSE Dept, NITK

TEAM
Kesana Jahnavi (181CO127)
Shreeya Sanjay Sand (181CO150)
Shumbul Arifa (181CO152)
Keerti Chaudhary (181CO226)

Abstract

A compiler is computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language). The name compiler is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language, object code, or machine code) to create an executable program. Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

INDEX

List of Figures:

Overview

FEATURES

The project objective is to construct a compiler that studies the C programming language. It will have the following features:

- The compiler is going to support the following cases:
    - Keywords: eg: int, char, float
    - Identifiers: eg: maximum, avg
    - Constants: eg. 1, 2, 20
    - Operators: eg: +, -, *
    - Strings: eg: “cars”, “cs”
    - Special symbols: eg: [],*, ()
- Support int and char data types and short, long, signed, unsigned subtypes.
- Detection of arrays with specified datatype (eg: int arr[10])
- Detection of looping constructs such as while, nested while.
- Detection conditional statements such as if-else and nested if-else.
- Identification of user-defined functions with one argument with return types int, char, void.
- Hashing techniques used to maintain symbol and constant tables.
- Support for single-line as well as multiline comments and return appropriate error messages.
- Appropriate error messages for comments and strings that don't end until the end of the file.

RESULTS

- Details of the identified tokens for the source program taken as input.
- Errors in the source program along with appropriate error messages
- Symbol table will be designed using hashing organization techniques.

TOOLS USED

- Flex

Introduction

The Lexical Analyzer is the first phase of the Analysis (front end) stage of a compiler. In layman’s terms, the Lexical Analyzer (or Scanner) scans through the input source program character by character and identifies ‘Lexemes’ and categorizes them into ‘Tokens’. These ‘tokens’ are represented as a symbol table and is given as input to the Parser (second phase of the front end of a compiler).

Definitions

What's a lexeme?

A lexeme is a sequence of characters that are included in the source program according to the matching pattern of a token. It is nothing but an instance of a token.

What's a token?

The token is a sequence of characters which represents a unit of information in the source program.

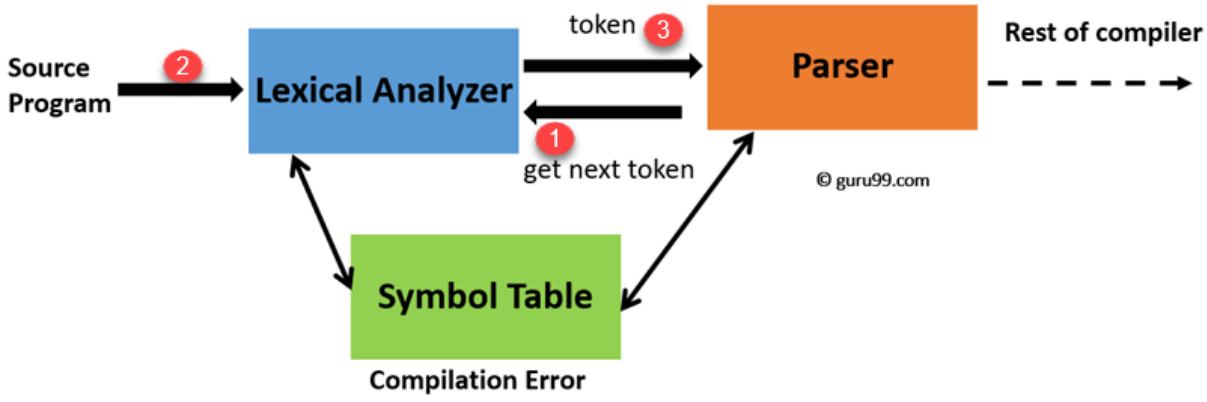
What is Pattern?

A pattern is a description which is used by the token. In the case of a keyword which uses as a token, the pattern is a sequence of characters.

Lexical Analyzer Architecture: How are tokens recognized?

The main task of lexical analysis is to read input characters in the code and produce tokens.

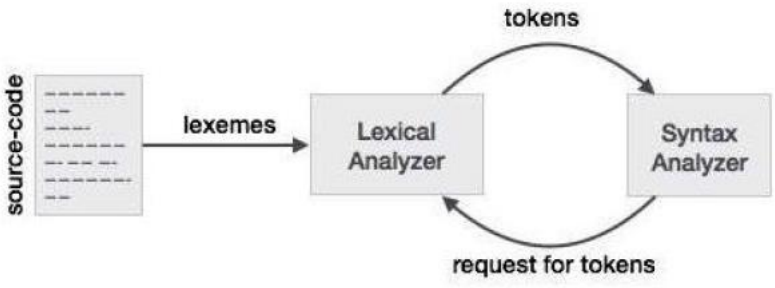
Lexical analyzer scans the entire source code of the program. It identifies each token one by one. Scanners are usually implemented to produce tokens only when requested by a parser. Here is how this works-



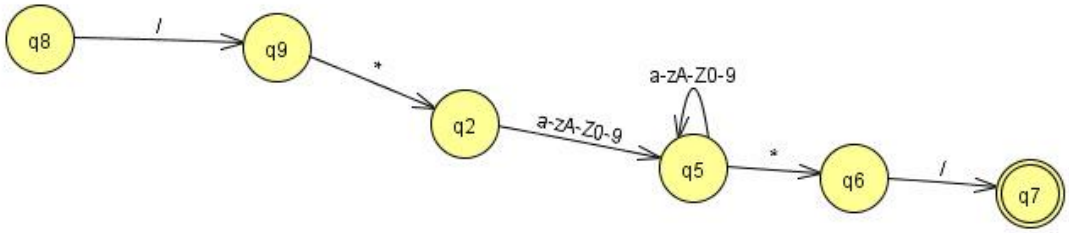
1. "Get next token" is a command which is sent from the parser to the lexical analyzer.
2. On receiving this command, the lexical analyzer scans the input until it finds the next token.
3. It returns the token to Parser.

Lexical Analyzer skips whitespaces and comments while creating these tokens. If any error is present, then Lexical analyzer will correlate that error with the source file and line number.

DFAs

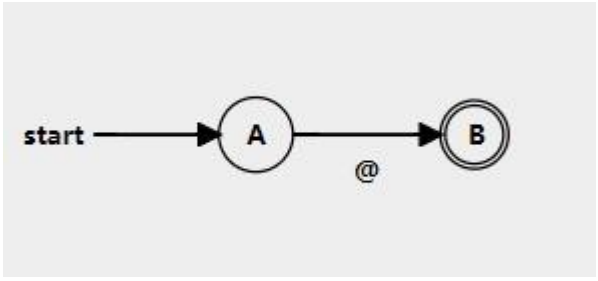


Overall Working of a Scanner

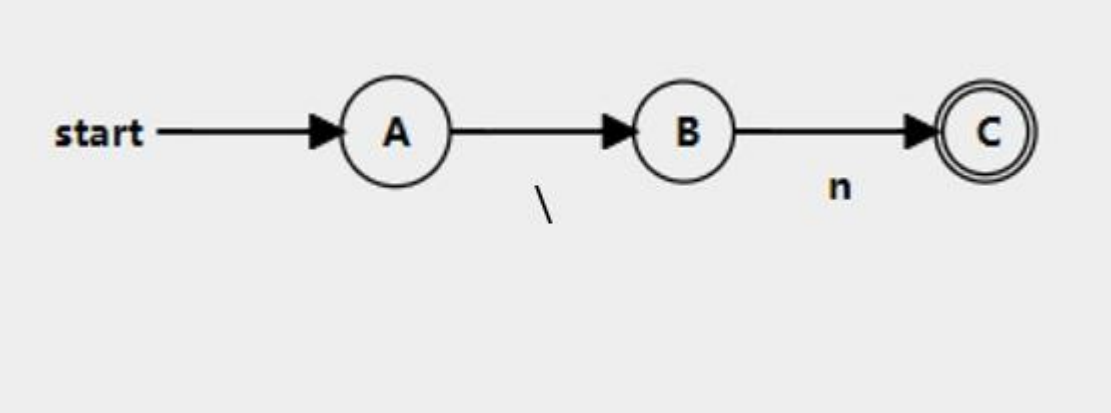


Multi-line Comments

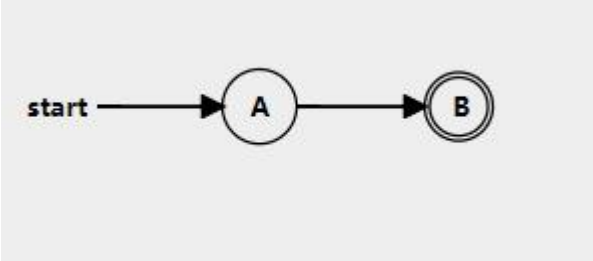
Single-line Comments



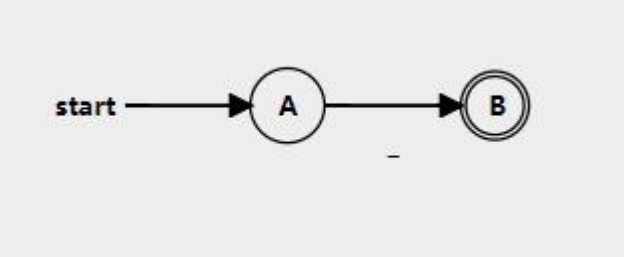
@ symbol



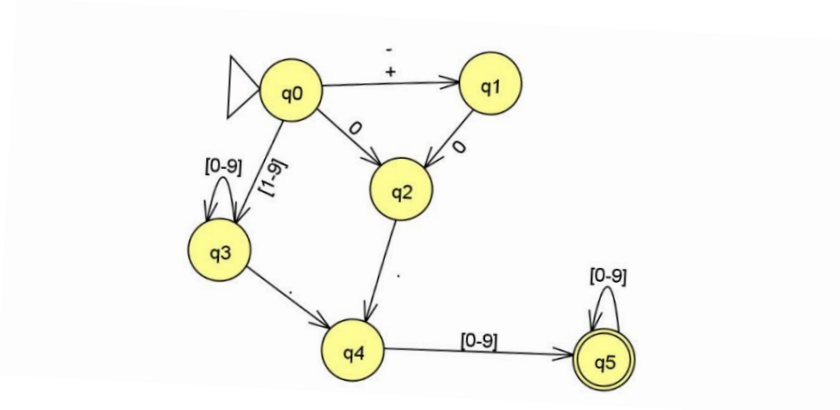
Nextline



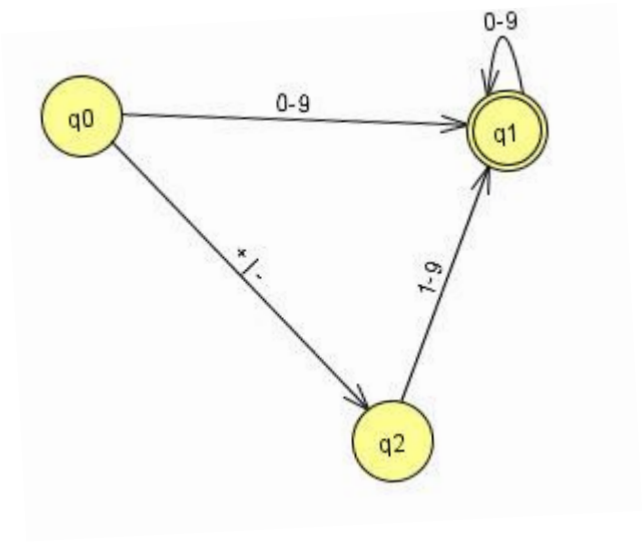
Space



Underscore



Float



Signed Integer

Code

```
/*Program identifies tokens to be returned by the scanner*/
/*Also stores the values in Symbol Table*/
/*Program also identifies errors in C file*/

%{
int lineno = 1;
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define AUTO 1
#define BREAK 2
#define CASE 3
#define CHAR 4
#define CONST 5
#define CONTINUE 6
#define DEFAULT 7
#define DO 8
#define DOUBLE 9
#define ELSE 10
#define ENUM 11
#define EXTERN 12
#define FLOAT 13
#define FOR 14
#define GOTO 15
#define IF 16
#define INT 17
```

```
#define LONG 18
#define REGISTER 19
#define RETURN 20
#define SHORT 21
#define SIGNED 22
#define SIZEOF 23
#define STATIC 24
#define STRUCT 25
#define SWITCH 26
#define TYPEDEF 27
#define UNION 28
#define UNSIGNED 29
#define VOID 30
#define VOLATILE 31
#define WHILE 32

#define IDENTIFIER 33
#define SLC 34
#define MLCS 35
#define MLCE 36

#define LEQ 37
#define GEQ 38
#define EQEQ 39
#define NEQ 40
#define LOR 41
#define LAND 42
#define ASSIGN 43
#define PLUS 44
#define SUB 45
#define MULT 46
#define DIV 47
#define MOD 48
#define LESSER 49
#define GREATER 50
#define INCR 51
#define DECR 52

#define COMMA 53
#define SEMI 54

#define HEADER 55
#define MAIN 56

#define PRINTF 57
#define SCANF 58
```

```
#define DEFINE 59

#define INT_CONST 60
#define FLOAT_CONST 61

#define TYPE_SPEC 62

#define DQ 63

#define OBO 64
#define OBC 65
#define CBO 66
#define CBC 67
#define HASH 68

#define ARR 69
#define FUNC 70

#define NUM_ERR 71
#define UNKNOWN 72

#define CHAR_CONST 73
#define SIGNED_CONST 74
#define STRING_CONST 75
%}

alpha [A-Za-z]
digit [0-9]
und [_]
space [ ]
tab [   ]
line [\n]
char \'.\'
at [@]
string \"(.^([%d]|[%f]|[%s]|[%c]))\"
%%
{space}* {}
{tab}* {}
{string} return STRING_CONST;
{char} return CHAR_CONST;
{line} {lineno++;}
auto return AUTO;
break return BREAK;
case return CASE;
char return CHAR;
const return CONST;
```

```
continue return CONTINUE;
default return DEFAULT;
do return DO;
double return DOUBLE;
else return ELSE;
enum return ENUM;
extern return EXTERN;
float return FLOAT;
for return FOR;
goto return GOTO;
if return IF;
int return INT;
long return LONG;
register return REGISTER;
return return RETURN;
short return SHORT;
signed return SIGNED;
sizeof return SIZEOF;
static return STATIC;
struct return STRUCT;
switch return SWITCH;
typedef return TYPEDEF;
union return UNION;
unsigned return UNSIGNED;
void return VOID;
volatile return VOLATILE;
while return WHILE;
printf return PRINTF;
scanf return SCANF;
{alpha}({alpha}|{digit}|{und})* return IDENTIFIER;
[+-][0-9]{digit}*(\.{digit}+)? return SIGNED_CONST;
"//" return SLC;
"/*" return MLCS;
"*/" return MLCE;
"<=" return LEQ;
">=" return GEQ;
"==" return EQEQ;
"!=" return NEQ;
"||" return LOR;
"&&" return LAND;
"=" return ASSIGN;
"+" return PLUS;
"-" return SUB;
"*" return MULT;
"/" return DIV;
"%" return MOD;
```

```
"<" return LESSER;
">" return GREATER;
"++" return INCR;
"--" return DECR;
"," return COMMA;
";" return SEMI;
"#include<stdio.h>" return HEADER;
"#include <stdio.h>" return HEADER;
"main()" return MAIN;
{digit}+ return INT_CONST;
({digit}+)\.({digit}+) return FLOAT_CONST;
"%d"|"%f"|"%u"|"%s" return TYPE_SPEC;
"\"" return DQ;
"(" return OBO;
")" return OBC;
"{" return CBO;
"}" return CBC;
"#" return HASH;
{alpha}({alpha}|{digit}|{und})*\[{digit}*\] return ARR;
{alpha}({alpha}|{digit}|{und})*\(({alpha}|{digit}|{und}|{space})*\) return FUNC;
({digit}+)\.({digit}+)\.({digit}|\.)* return NUM_ERR;
({digit}|{at})+({alpha}|{digit}|{und}|{at})* return UNKNOWN;
%%
struct node
{
        char token[100];
        char attr[100];
        struct node *next;
};
struct hash
{
        struct node *head;
        int count;
};
struct hash hashTable[1000];
int eleCount = 1000;
struct node * createNode(char *token, char *attr)
{
        struct node *newnode;
        newnode = (struct node *) malloc(sizeof(struct node));
        strcpy(newnode->token, token);
        strcpy(newnode->attr, attr);
        newnode->next = NULL;
        return newnode;
}
int hashIndex(char *token)
```

```
{
        int hi=0;
        int l,i;
        for(i=0;token[i]!='\0';i++)
        {
                hi = hi + (int)token[i];
        }
        hi = hi%eleCount;
        return hi;
}

void hashInsert(char *token, char *attr)
{
        int flag=0;
        int hi;
        hi = hashIndex(token);
        struct node *newnode = createNode(token, attr);
        /* head of list for the bucket with index "hashIndex" */
        if (hashTable[hi].head==NULL)
        {
                hashTable[hi].head = newnode;
                 hashTable[hi].count = 1;
                 return;
        }
        struct node *myNode;
        myNode = hashTable[hi].head;
        while (myNode != NULL)
        {
                if (strcmp(myNode->token, token)==0)
                {
                        flag = 1;
                        break;
                }
                myNode = myNode->next;
        }
        if(!flag)
        {
                //adding new node to the list
                newnode->next = (hashTable[hi].head);
                //update the head of the list and no of nodes in the current bucket
                hashTable[hi].head = newnode;
                hashTable[hi].count++;
        }
        return;
}
```

```
void display()
{
	struct node *myNode;
	int i,j, k=1;
	printf("----------------------------------------------------------------------
");
	printf("\nSNo \t|\tToken \t\t|\tToken Type \t\n");
	 printf("----------------------------------------------------------------------
\n");
	for (i = 0; i < eleCount; i++)
	{
		if (hashTable[i].count == 0)
			continue;
		myNode = hashTable[i].head;
		if (!myNode)
			continue;
		while (myNode != NULL)
		{
			printf("%d\t\t", k++);
			printf("%s\t\t\t", myNode->token);
			printf("%s\t\n", myNode->attr);
			myNode = myNode->next;
		}
	}
	return;
}

int main()
{
	int scan, slcline=0, mlc=0, mlcline=0, dq=0, dqline=0;
	yyin = fopen("test6.c","r");
	printf("\n\n");
	scan = yylex();
	while(scan)
	{
		if(lineno == slcline)
		{
			scan = yylex();
			continue;
		}
		if(lineno!=dqline && dqline!=0)
		{
			if(dq%2!=0)
				printf("\n<<<<<<<<<< ERROR! >>>>>>>>>>
\n<<<<<<<<<<INCOMPLETE STRING at Line %d >>>>>>>>>>\n\n", dqline);
			dq=0;
```

```
		}
		if((scan>=1 && scan<=32) && mlc==0)
		{
			printf("%s\t\t\tKEYWORD\t\t\t\tLine %d\n", yytext, lineno);
			hashInsert(yytext, "KEYWORD");
		}
		if(scan==33 && mlc==0)
		{
			printf("%s\t\t\tIDENTIFIER\t\t\tLine %d\n", yytext, lineno);
			hashInsert(yytext, "IDENTIFIER");
		}
		if(scan==34)
		{
			printf("%s\t\t\tSingleline Comment\t\tLine %d\n", yytext,
lineno);
			slcline = lineno;
		}
		if(scan==35 && mlc==0)
		{
			printf("%s\t\t\tMultiline Comment Start\t\tLine %d\n", yytext,
lineno);
			mlcline = lineno;
			mlc = 1;
		}
		if(scan==36 && mlc==0)
		{
			printf("\n<<<<<<<<<< ERROR! >>>>>>>>>> \n<<<<<<<<<<UNMATCHED
MULTILINE COMMENT END %s at Line %d >>>>>>>>>>\n\n", yytext, lineno);
		}
		if(scan==36 && mlc==1)
		{
			mlc = 0;
			printf("%s\t\t\tMultiline Comment End\t\tLine %d\n", yytext,
lineno);
		}
		if((scan>=37 && scan<=52) && mlc==0)
		{
			printf("%s\t\t\tOPERATOR\t\t\tLine %d\n", yytext, lineno);
			hashInsert(yytext, "OPERATOR");
		}
		if((scan==53||scan==54||scan==63||(scan>=64 && scan<=68)) && mlc==0)
		{
			printf("%s\t\t\tSPECIAL SYMBOL\t\t\tLine %d\n", yytext,
lineno);
			if(scan==63)
			{
```

```
                        dq++;
                        dqline = lineno;
                }
                hashInsert(yytext, "SPECIAL SYMBOL");
        }
        if(scan==55 && mlc==0)
        {
                printf("%s\tHEADER\t\t\t\tLine %d\n",yytext, lineno);
        }
        if(scan==56 && mlc==0)
        {
                printf("%s\t\t\tMAIN FUNCTION\t\t\tLine %d\n", yytext, lineno);
                hashInsert(yytext, "IDENTIFIER");
        }
        if((scan==57 || scan==58) && mlc==0)
        {
                printf("%s\t\t\tPRE DEFINED FUNCTION\t\tLine %d\n", yytext,
lineno);
                hashInsert(yytext, "PRE DEFINED FUNCTION");
        }
        if(scan==59 && mlc==0)
        {
                printf("%s\t\t\tPRE PROCESSOR DIRECTIVE\t\tLine %d\n", yytext,
lineno);
        }
        if(scan==60 && mlc==0)
        {
                printf("%s\t\t\tINTEGER CONSTANT\t\tLine %d\n", yytext,
lineno);
                hashInsert(yytext, "INTEGER CONSTANT");
        }
        if(scan==61 && mlc==0)
        {
                printf("%s\t\t\tFLOATING POINT CONSTANT\t\tLine %d\n", yytext,
lineno);
                hashInsert(yytext, "FLOATING POINT CONSTANT");
        }
        if(scan==62 && mlc==0)
        {
                printf("%s\t\t\tTYPE SPECIFIER\t\t\tLine %d\n", yytext,
lineno);
        }
        if(scan==69 && mlc==0)
        {
                printf("%s\t\t\tARRAY\t\t\t\tLine %d\n", yytext, lineno);
                hashInsert(yytext, "ARRAY");
```

```
		}
		if(scan==70 && mlc==0)
		{
			printf("%s\t\t\tUSER DEFINED FUNCTION\t\tLine %d\n", yytext,
lineno);
			hashInsert(yytext, "USER DEFINED FUNCTION");
		}
		if(scan==71 && mlc==0)
		{
			printf("\n<<<<<<<<<< ERROR! >>>>>>>>>> \n<<<<<<<<<<CONSTANT
ERROR %s at Line %d >>>>>>>>>>\n\n", yytext, lineno);
		}
		if(scan==72 && mlc==0)
		{
			printf("\n<<<<<<<<<< ERROR! >>>>>>>>>> \n<<<<<<<<<<UNKNOWN
TOKEN %s at Line %d >>>>>>>>>>\n\n", yytext, lineno);
		}
		if(scan==73 && mlc==0)
		{
			printf("%s\t\t\tCHARACTER CONSTANT\t\t\tLine %d\n", yytext,
lineno);
			hashInsert(yytext, "CHARACTER CONSTANT");
		}
		if(scan==74 && mlc==0)
		{
			printf("%s\t\t\tSIGNED CONSTANT\t\t\tLine %d\n", yytext,
lineno);
			hashInsert(yytext, "SIGNED CONSTANT");
		}
		if(scan==75 && mlc==0)
		{
			printf("%s\t\t\tSTRING CONSTANT\t\t\tLine %d\n", yytext,
lineno);
			hashInsert(yytext, "STRING CONSTANT");
		}
		scan = yylex();
	}
	if(mlc==1)
		printf("\n<<<<<<<<<< ERROR! >>>>>>>>>> \n<<<<<<<<<<UNMATCHED COMMENT
STARTING at Line %d >>>>>>>>>>\n\n",mlcline);
	printf("\n");
	printf("\n\t############### SYMBOL TABLE ###############\t\t\n");
	display();
	 printf("------------------------------------------------------------------
\n\n");
}
```

```
int yywrap()
{
        return 1;
}
```

Explanation

- Regular expression for identifiers: The lexer must correctly recognize all valid identifiers in C, including the ones having one or more underscores. [a-z|A-Z]([a-z|A-Z]|[0-9])*
- Multiline comments: This has been supported by checking the occurrence of '/*' and '*/' in the code. The statements between them has been excluded.
- Errors for unmatched and nested comments have also been displayed.
- Error Handling for Incomplete String: Open and close quote missing, both kind of errors have been handled in the rules written in the script.
- Error Handling for Nested Comments: This use-case has been handled by checking for occurrence of multiple successive '/*' or '*/' in the C code, and by omitting the text in between them.
- At the end of the token recognition, the lexical analyzer prints a list of all the tokens present in the program. As and when successive tokens are encountered, their respective values are stored in the symbol table structure and then later displayed.

Test Cases Implementation

➢ With errors

TEST CASE 1: output displays error in incomplete string

```
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ cc lex.yy.c
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ .a/.out
bash: .a/.out: No such file or directory
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ ./a.out


int                     KEYWORD                         Line 2
main()                  MAIN FUNCTION                   Line 2
{                       SPECIAL SYMBOL                  Line 3
char                    KEYWORD                         Line 4
A[]                     ARRAY                           Line 4
=                       OPERATOR                        Line 4
"                       SPECIAL SYMBOL                  Line 4
#                       SPECIAL SYMBOL                  Line 4
define                  IDENTIFIER                      Line 4
min                     IDENTIFIER                      Line 4
11                      INTEGER CONSTANT                Line 4
"                       SPECIAL SYMBOL                  Line 4
;                       SPECIAL SYMBOL                  Line 4
char                    KEYWORD                         Line 5
B[]                     ARRAY                           Line 5
=                       OPERATOR                        Line 5
"                       SPECIAL SYMBOL                  Line 5
Almighty                        IDENTIFIER                              Line 5
;                       SPECIAL SYMBOL                  Line 5
```

```
<<<<<<<<<< ERROR! >>>>>>>>>>
<<<<<<<<<<INCOMPLETE STRING at Line 5 >>>>>>>>>>

unsigned                        KEYWORD                         Line 6
int                     KEYWORD                         Line 6
a                       IDENTIFIER                      Line 6
=                       OPERATOR                        Line 6
1                       INTEGER CONSTANT                Line 6
;                       SPECIAL SYMBOL                  Line 6
printf                  PRE DEFINED FUNCTION            Line 7
(                       SPECIAL SYMBOL                  Line 7
"                       SPECIAL SYMBOL                  Line 7
string                  IDENTIFIER                      Line 7
=                       OPERATOR                        Line 7
%s                      TYPE SPECIFIER                  Line 7
value                   IDENTIFIER                      Line 7
of                      IDENTIFIER                      Line 7
E                       IDENTIFIER                      Line 7
=                       OPERATOR                        Line 7
%f                      TYPE SPECIFIER                  Line 7
"                       SPECIAL SYMBOL                  Line 7
,                       SPECIAL SYMBOL                  Line 7
A                       IDENTIFIER                      Line 7
,                       SPECIAL SYMBOL                  Line 7
2.7                     FLOATING POINT CONSTANT         Line 7
)                       SPECIAL SYMBOL                  Line 7
;                       SPECIAL SYMBOL                  Line 7
return                  KEYWORD                         Line 8
0                       INTEGER CONSTANT                Line 8
;                       SPECIAL SYMBOL                  Line 8
}                       SPECIAL SYMBOL                  Line 10
```

```
        ******** SYMBOL TABLE ********
-----------------------------------------------------------------
SNo     |       Token           |       Token Type
-----------------------------------------------------------------
1               "                       SPECIAL SYMBOL
2               #                       SPECIAL SYMBOL
3               (                       SPECIAL SYMBOL
4               )                       SPECIAL SYMBOL
5               ,                       SPECIAL SYMBOL
6               0                       INTEGER CONSTANT
7               1                       INTEGER CONSTANT
8               ;                       SPECIAL SYMBOL
9               =                       OPERATOR
10              A                       IDENTIFIER
11              E                       IDENTIFIER
12              a                       IDENTIFIER
13              11                      INTEGER CONSTANT
14              {                       SPECIAL SYMBOL
15              }                       SPECIAL SYMBOL
16              2.7                     FLOATING POINT CONSTANT
17              of                      IDENTIFIER
18              A[]                     ARRAY
19              B[]                     ARRAY
20              min                     IDENTIFIER
21              int                     KEYWORD
22              char                    KEYWORD
23              main()                  IDENTIFIER
24              value                   IDENTIFIER
25              define                  IDENTIFIER
26              printf                  PRE DEFINED FUNCTION
27              string                  IDENTIFIER
28              return                  KEYWORD
29              Almighty                        IDENTIFIER
30              unsigned                        KEYWORD
```

TEST CASE 2: output displays error in floating point number

```
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ flex lex.l
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ cc lex.yy.c
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ ./a.out


/*                      Multiline Comment Start         Line 1
*/                      Multiline Comment End           Line 1
//                      Singleline Comment              Line 2
//                      Singleline Comment              Line 5
double                  KEYWORD                         Line 6
findSQRT(double N)                      USER DEFINED FUNCTION           Line 6
{                       SPECIAL SYMBOL                  Line 7
return                  KEYWORD                         Line 8
sqrt(N)                 USER DEFINED FUNCTION           Line 8
;                       SPECIAL SYMBOL                  Line 8
}                       SPECIAL SYMBOL                  Line 9
//                      Singleline Comment              Line 11
int                     KEYWORD                         Line 12
main()                  MAIN FUNCTION                   Line 12
{                       SPECIAL SYMBOL                  Line 13
//                      Singleline Comment              Line 15
int                     KEYWORD                         Line 16
N                       IDENTIFIER                      Line 16
=                       OPERATOR                        Line 16

<<<<<<<<<< ERROR! >>>>>>>>>>
<<<<<<<<<<CONSTANT ERROR 12.6. at Line 17 >>>>>>>>>>

f                       IDENTIFIER                      Line 17
;                       SPECIAL SYMBOL                  Line 17
//                      Singleline Comment              Line 19
printf                  PRE DEFINED FUNCTION            Line 20
(                       SPECIAL SYMBOL                  Line 20
"                       SPECIAL SYMBOL                  Line 20
%f                      TYPE SPECIFIER                  Line 20
"                       SPECIAL SYMBOL                  Line 20
,                       SPECIAL SYMBOL                  Line 20
findSQRT(N)                     USER DEFINED FUNCTION           Line 20
)                       SPECIAL SYMBOL                  Line 20
;                       SPECIAL SYMBOL                  Line 20
return                  KEYWORD                         Line 21
0                       INTEGER CONSTANT                Line 21
;                       SPECIAL SYMBOL                  Line 21
}                       SPECIAL SYMBOL                  Line 22


        ############## SYMBOL TABLE ##############
-----------------------------------------------------------------
SNo     |       Token           |       Token Type
-----------------------------------------------------------------
1               "                       SPECIAL SYMBOL
2               #                       SPECIAL SYMBOL
3               (                       SPECIAL SYMBOL
4               )                       SPECIAL SYMBOL
5               ,                       SPECIAL SYMBOL
6               0                       INTEGER CONSTANT
7               ;                       SPECIAL SYMBOL
8               <                       OPERATOR
9               =                       OPERATOR
10              >                       OPERATOR
11              N                       IDENTIFIER
12              f                       IDENTIFIER
13              h                       IDENTIFIER
14              {                       SPECIAL SYMBOL
15              }                       SPECIAL SYMBOL
16              int                     KEYWORD
17              math                    IDENTIFIER
18              main()                  IDENTIFIER
19              findSQRT(double N)                      USER DEFINED FUNCTION
20              sqrt(N)                 USER DEFINED FUNCTION
21              double                  KEYWORD
22              printf                  PRE DEFINED FUNCTION
23              return                  KEYWORD
24              include                 IDENTIFIER
25              findSQRT(N)                     USER DEFINED FUNCTION
-----------------------------------------------------------------
```

TEST CASE 3: output displays the error in incomplete string, parentheses

```
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ flex lex.l
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ cc lex.yy.c
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ ./a.out


/*                      Multiline Comment Start         Line 1
*/                      Multiline Comment End           Line 1
int                     KEYWORD                         Line 3
main()                  MAIN FUNCTION                   Line 3
{                       SPECIAL SYMBOL                  Line 3
char                    KEYWORD                         Line 4
s1[100]                 ARRAY                           Line 4
=                       OPERATOR                        Line 4
"                       SPECIAL SYMBOL                  Line 4
programming                     IDENTIFIER                      Line 4
"                       SPECIAL SYMBOL                  Line 4
,                       SPECIAL SYMBOL                  Line 4
s2[]                    ARRAY                           Line 4
=                       OPERATOR                        Line 4
"                       SPECIAL SYMBOL                  Line 4
is                      IDENTIFIER                      Line 4
awesome                 IDENTIFIER                      Line 4
';                      SPECIAL SYMBOL                  Line 4
<<<<<<<<<< ERROR! >>>>>>>>>>
<<<<<<<<<<INCOMPLETE STRING at Line 4 >>>>>>>>>>

int                     KEYWORD                         Line 5
length                  IDENTIFIER                      Line 5
,                       SPECIAL SYMBOL                  Line 5
j                       IDENTIFIER                      Line 5
;                       SPECIAL SYMBOL                  Line 5
//                      Singleline Comment              Line 7
length                  IDENTIFIER                      Line 8
=                       OPERATOR                        Line 8
0                       INTEGER CONSTANT                Line 8
;                       SPECIAL SYMBOL                  Line 8
while                   KEYWORD                         Line 9
(                       SPECIAL SYMBOL                  Line 9
s1                      IDENTIFIER                      Line 9
[length                 IDENTIFIER                      Line 9
]!=                     OPERATOR                        Line 9
'\0                     INTEGER CONSTANT                Line 9
')                      SPECIAL SYMBOL                  Line 9
{                       SPECIAL SYMBOL                  Line 9
++                      OPERATOR                        Line 10
length                  IDENTIFIER                      Line 10
;                       SPECIAL SYMBOL                  Line 10
}                       SPECIAL SYMBOL                  Line 11
//                      Singleline Comment              Line 13
for                     KEYWORD                         Line 14
(                       SPECIAL SYMBOL                  Line 14
j                       IDENTIFIER                      Line 14
=                       OPERATOR                        Line 14
0                       INTEGER CONSTANT                Line 14
;                       SPECIAL SYMBOL                  Line 14
s2                      IDENTIFIER                      Line 14
[j                      IDENTIFIER                      Line 14
]!=                     OPERATOR                        Line 14
'\0                     INTEGER CONSTANT                Line 14
';                      SPECIAL SYMBOL                  Line 14
++                      OPERATOR                        Line 14
j                       IDENTIFIER                      Line 14
,                       SPECIAL SYMBOL                  Line 14
++                      OPERATOR                        Line 14
length                  IDENTIFIER                      Line 14
)                       SPECIAL SYMBOL                  Line 14
{                       SPECIAL SYMBOL                  Line 14
s1                      IDENTIFIER                      Line 15
```

```
[length                 IDENTIFIER                      Line 15
]=                      OPERATOR                        Line 15
s2                      IDENTIFIER                      Line 15
[j                      IDENTIFIER                      Line 15
];                      SPECIAL SYMBOL                  Line 15
}                       SPECIAL SYMBOL                  Line 16
//                      Singleline Comment              Line 18
s1                      IDENTIFIER                      Line 19
[length                 IDENTIFIER                      Line 19
]=                      OPERATOR                        Line 19
'\0                     INTEGER CONSTANT                Line 19
';                      SPECIAL SYMBOL                  Line 19
printf                  PRE DEFINED FUNCTION            Line 21
(                       SPECIAL SYMBOL                  Line 21
"                       SPECIAL SYMBOL                  Line 21
After                   IDENTIFIER                      Line 21
concatenation                   IDENTIFIER                      Line 21
:"                      SPECIAL SYMBOL                  Line 21
)                       SPECIAL SYMBOL                  Line 21
;                       SPECIAL SYMBOL                  Line 21
puts(s1)                        USER DEFINED FUNCTION           Line 22
;                       SPECIAL SYMBOL                  Line 22
return                  KEYWORD                         Line 24
0                       INTEGER CONSTANT                Line 24
;                       SPECIAL SYMBOL                  Line 24
}                       SPECIAL SYMBOL                  Line 25
```

```
        ############## SYMBOL TABLE ##############
-----------------------------------------------------------------
SNo     |       Token           |       Token Type
-----------------------------------------------------------------
1               "                       SPECIAL SYMBOL
2               (                       SPECIAL SYMBOL
3               )                       SPECIAL SYMBOL
4               ,                       SPECIAL SYMBOL
5               0                       INTEGER CONSTANT
6               ;                       SPECIAL SYMBOL
7               =                       OPERATOR
8               ++                      OPERATOR
9               !=                      OPERATOR
10              j                       IDENTIFIER
11              {                       SPECIAL SYMBOL
12              }                       SPECIAL SYMBOL
13              s1                      IDENTIFIER
14              s2                      IDENTIFIER
15              programming                     IDENTIFIER
16              is                      IDENTIFIER
17              for                     KEYWORD
18              int                     KEYWORD
19              s2[]                    ARRAY
20              concatenation                   IDENTIFIER
21              char                    KEYWORD
22              s1[100]                 ARRAY
23              After                   IDENTIFIER
24              main()                  IDENTIFIER
25              while                   KEYWORD
26              length                  IDENTIFIER
27              printf                  PRE DEFINED FUNCTION
28              return                  KEYWORD
29              puts(s1)                        USER DEFINED FUNCTION
30              awesome                 IDENTIFIER
-----------------------------------------------------------------
```

- Without errors

TEST CASE 1: output for testcase containing function and print statement

```
{                       SPECIAL SYMBOL          Line 19
while                   KEYWORD                 Line 20
(                       SPECIAL SYMBOL          Line 20
a                       IDENTIFIER              Line 20
!=                      OPERATOR                Line 20
b                       IDENTIFIER              Line 20
)                       SPECIAL SYMBOL          Line 20
{                       SPECIAL SYMBOL          Line 21
if                      KEYWORD                 Line 22
(                       SPECIAL SYMBOL          Line 22
a                       IDENTIFIER              Line 22
>                       OPERATOR                Line 22
b                       IDENTIFIER              Line 22
)                       SPECIAL SYMBOL          Line 22
{                       SPECIAL SYMBOL          Line 23
return                  KEYWORD                 Line 24
gcd                     IDENTIFIER              Line 24
(                       SPECIAL SYMBOL          Line 24
a                       IDENTIFIER              Line 24
-                       OPERATOR                Line 24
b                       IDENTIFIER              Line 24
,                       SPECIAL SYMBOL          Line 24
b                       IDENTIFIER              Line 24
)                       SPECIAL SYMBOL          Line 24
;                       SPECIAL SYMBOL          Line 24
}                       SPECIAL SYMBOL          Line 25
else                    KEYWORD                 Line 26
{                       SPECIAL SYMBOL          Line 27
return                  KEYWORD                 Line 28
gcd                     IDENTIFIER              Line 28
(                       SPECIAL SYMBOL          Line 28
a                       IDENTIFIER              Line 28
,                       SPECIAL SYMBOL          Line 28
b                       IDENTIFIER              Line 28
-                       OPERATOR                Line 28
a                       IDENTIFIER              Line 28
)                       SPECIAL SYMBOL          Line 28
;                       SPECIAL SYMBOL          Line 28
}                       SPECIAL SYMBOL          Line 29
}                       SPECIAL SYMBOL          Line 30
return                  KEYWORD                 Line 31
a                       IDENTIFIER              Line 31
;                       SPECIAL SYMBOL          Line 31
}                       SPECIAL SYMBOL          Line 32
```

```
        ############## SYMBOL TABLE ##############
-------------------------------------------------------------------
SNo     |       Token           |       Token Type
-------------------------------------------------------------------
1               "                       SPECIAL SYMBOL
2               (                       SPECIAL SYMBOL
3               )                       SPECIAL SYMBOL
4               ,                       SPECIAL SYMBOL
5               -                       OPERATOR
6               ;                       SPECIAL SYMBOL
7               =                       OPERATOR
8               >                       OPERATOR
9               !=                      OPERATOR
10              a                       IDENTIFIER
11              b                       IDENTIFIER
12              n                       IDENTIFIER
13              {                       SPECIAL SYMBOL
14              }                       SPECIAL SYMBOL
15              GCD                     IDENTIFIER
16              if                      KEYWORD
17              of                      IDENTIFIER
18              is                      IDENTIFIER
19              to                      IDENTIFIER
20              The                     IDENTIFIER
21              gcd                     IDENTIFIER
22              and                     IDENTIFIER
23              the                     IDENTIFIER
24              int                     KEYWORD
25              two                     IDENTIFIER
26              find                    IDENTIFIER
27              else                    KEYWORD
28              main()                  IDENTIFIER
29              Enter                   IDENTIFIER
30              scanf                   PRE DEFINED FUNCTION
31              while                   KEYWORD
32              their                   IDENTIFIER
33              printf                  PRE DEFINED FUNCTION
34              result                  IDENTIFIER
35              return                  KEYWORD
36              numbers                 IDENTIFIER
-------------------------------------------------------------------
```

TEST CASE 2: output for testcase containing while statement and print statement function

```
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ flex lex.l
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ cc lex.yy.c
hp@hp-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/Lexical Analysis$ ./a.out


int                     KEYWORD                         Line 2
sum_n                   IDENTIFIER                      Line 2
(                       SPECIAL SYMBOL                  Line 2
int                     KEYWORD                         Line 2
n                       IDENTIFIER                      Line 2
)                       SPECIAL SYMBOL                  Line 2
{                       SPECIAL SYMBOL                  Line 2
int                     KEYWORD                         Line 3
result                  IDENTIFIER                      Line 3
=                       OPERATOR                        Line 3
0                       INTEGER CONSTANT                Line 3
;                       SPECIAL SYMBOL                  Line 3
while                   KEYWORD                         Line 5
(                       SPECIAL SYMBOL                  Line 5
n                       IDENTIFIER                      Line 5
)                       SPECIAL SYMBOL                  Line 5
{                       SPECIAL SYMBOL                  Line 5
result                  IDENTIFIER                      Line 6
=                       OPERATOR                        Line 6
result                  IDENTIFIER                      Line 6
+                       OPERATOR                        Line 6
n                       IDENTIFIER                      Line 6
;                       SPECIAL SYMBOL                  Line 6
n                       IDENTIFIER                      Line 7
--                      OPERATOR                        Line 7
;                       SPECIAL SYMBOL                  Line 7
}                       SPECIAL SYMBOL                  Line 8
return                  KEYWORD                         Line 10
result                  IDENTIFIER                      Line 10
;                       SPECIAL SYMBOL                  Line 10
}                       SPECIAL SYMBOL                  Line 11
int                     KEYWORD                         Line 13
main                    IDENTIFIER                      Line 13
(                       SPECIAL SYMBOL                  Line 13
)                       SPECIAL SYMBOL                  Line 13
{                       SPECIAL SYMBOL                  Line 13
return                  KEYWORD                         Line 14
sum_n(10)                       USER DEFINED FUNCTION           Line 14
;                       SPECIAL SYMBOL                  Line 14
}                       SPECIAL SYMBOL                  Line 15

        ############## SYMBOL TABLE ##############
--------------------------------------------------------------------
SNo     |       Token           |       Token Type
--------------------------------------------------------------------
1               (                       SPECIAL SYMBOL
2               )                       SPECIAL SYMBOL
3               +                       OPERATOR
4               0                       INTEGER CONSTANT
5               ;                       SPECIAL SYMBOL
6               =                       OPERATOR
7               --                      OPERATOR
8               n                       IDENTIFIER
9               {                       SPECIAL SYMBOL
10              }                       SPECIAL SYMBOL
11              int                     KEYWORD
12              main                    IDENTIFIER
13              while                   KEYWORD
14              sum_n                   IDENTIFIER
15              result                  IDENTIFIER
16              return                  KEYWORD
17              sum_n(10)                       USER DEFINED FUNCTION
--------------------------------------------------------------------
```

Conclusion

The scanner that was created in this project helps in breaking source program into tokens defined by the C programming language.
In the next phase, parser will be designed which will call upon the Flex program to give it tokens and the lexical analyzer will return to the parser the integer value associated with the tokens as and when required by the parser. Together with the symbol table, the parser will prepare a syntax tree with the help of a grammar that we provide it with. The parser can then logically group the tokens to form meaningful statements and can detect C programming constructs such as arrays, loops, and functions. The parser will also help us identify errors that could not be detected in the lexical analysis phase such as unbalanced parentheses, unterminated statements, missing operators, two operators in a row, etc.

Future Work

The flex script presented in this report takes care of all the rules of C language, but is not fully exhaustive in nature. Our future work would include making the script even more robust in order to handle all aspects of C language and making it more efficient.

References

- Compilers Principles, Techniques and Tool by Alfred V.Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman
- http://dinosaur.compilertools.net/lex/index.html
- http://www.csd.uwo.ca/~moreno/CS447/Lectures/Lexical.html/node11.html
- https://www.geeksforgeeks.org/cc-tokens/
- http://www.isi.edu/~pedro/Teaching/CSCI565-Spring11/Practice/SDT-Sample.pdf
- StackOverflow for regex
- https://www.guru99.com/compiler-design-lexical-analysis.html

---

****