Lexical Analyzer for the C Language

National Institute of Technology Karnataka Surathkal

Date:
21/01/2020

Submitted To:
Dr. P. Santhi Thilagam
CSE Dept

Group Members:
17CO227: Niwedita
17CO139: Sanjana P

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.

Phases of Compiler

Conceptually, a compiler operates in phases, each of which transforms the source program from one representation to another.

The phases are as below:

Analysis

1. Lexical Analysis:
2. Parsing:
3. Semantic Analysis:
4. Intermediate Code Generation:

Synthesis

1. Code Optimization:

2. CodeGeneration:

Objectives:

This project aims to undertake a sequence of experiments to design and implement various phases of a compiler for the C programming language. Following constructs will be handled by the mini-compiler :

1. Variable data types - int, char along with its sub types - short, long, signed, unsigned.
2. Looping constructs - while loops along with nested while loops.
3. Identification and classification of tokens.
4. Identification of functions accepting a single parameter.
5. Maintenance of a symbol table and a constant table using hashing techniques.
6. Error detection for multi-line comments and nested comments that are not

terminated before the end of the program.

7. Checking for strings that does not end before the end of a statement and displaying corresponding error message.

Results:

1. Error messages for the errors handled.
2. The token will be displayed along with the type:
    - ❖ Keyword
    - ❖ Identifier
    - ❖ Literal
    - ❖ Operator
    - ❖ Punctuator
3. Symbol table
4. Constant table

Tools used:

- Flex

Contents:

List of Figures and Tables:

Introduction

Lexical Analysis

In computer science, lexical analysis is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings
with an identified "meaning"). A program that performs lexical analysis may be called a lexer, tokenizer, or scanner (though "scanner" is also used to refer to the first stage of a lexer). Such a lexer is generally combined with a parser, which together analyze the syntax
of programming languages, web pages, and so forth.

Flex Script

The script written by us is a program that generates lexical analyzers ("scanners" or "lexers"). Lex reads an input stream specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language.

The structure of our flex script is intentionally similar to that of a yacc file; files are divided into three sections, separated by lines that contain only two percent signs, as follows:

Definition section

%%

Rules section

%%

C code section

The definition section defines macros and imports header files written in C. It is also possible to write any C code here, which will be copied verbatim into the generated source file.

The rules section associates regular expression patterns with C statements. When the lexer sees text in the input matching a given pattern, it will execute the associated C

code.

The C code section contains C statements and functions that are copied verbatim to the generated source file. These statements presumably contain code called by the rules in the rules section. In large programs it is more convenient to place this code in a separate file linked in at compile time.

C Program

This section describes the input C program which is fed to the flex script in order to generate the lex file after taking all the rules mentioned in account. Finally, a file called lex.yy.c is generated, which when executed recognizes the tokens present in the C program which was given as an input.

The script also has an option to take standard input instead of taking input from a file.

Design of Programs

Code:
Lex Code: (scanner.l file)

```
%{
  int yylineno;

//Keywords
  #define WHILE 1
  #define VOID 2
  #define RETURN 3
  #define MAINFUNC 4
  #define BREAK 5
  #define CONTINUE 7
  #define IF 8
  #define INT 10
  #define CHAR 11
  #define UNSIGNED 12
  #define SIGNED 13
  #define LONG 14
  #define SHORT 15
  #define ELSE 16
  #define FOR 17
  #define STRUCT 18

//Identifier and Constant
```

```
  #define ID 20
  #define CONST 21

//Operators
//Comparators
  #define LE 22
          // Less than equal to
  #define GE 23
          // Greater than equal to
  #define EQ 24
          // Check for equality
  #define NE 25
          // Not equal to check
  #define L 77
          // Less than
  #define G 78
          // Greater than

//Logical
  #define OR 26
  #define AND 27
  #define NOT 28

//Assignment
  #define ASS 29
          //   =      Simple assignment operator.
  #define ADDASS 30
          // +=      Add AND assignment operator.
  #define SUBASS 31
          // -=      Subtract AND assignment operator.
  #define MULASS 32
          // *=      Multiply AND assignment operator.
  #define DIVASS 33
          // /=      Divide AND assignment operator.
  #define MODASS 34
          //          %=        Modulus AND assignment operator.

//Arithmetic
  #define PLUS 35
  #define SUB 36
  #define MULT 37
  #define DIV 38
  #define MOD 39
  #define PP 40
          // ++
  #define MM 41
          // --
```

```
//Bitwise Ops
  #define BA 42
            // Bitwise and
  #define BO 43
            // Bitwise or
  #define BC 44
            // Bitwise complement
  #define OC 45
            //one's complement
  #define LS 46
    // left shift
  #define RS 47
     //right shift

// Miscelaneous tokens
  #define SEMICOLON 53
  #define BA1 54
            // '(' bracket
  #define BA2 55
            // ')' bracket
  #define BB1 56
            // '[' bracket
  #define BB2 57
            // ']' bracket
  #define BC1 58
            // '{' bracket
  #define BC2 59
            //'}' bracket
  #define COMMA 60
            // ','
  #define Q 61
            // Quote "
  #define SQ 62
            // Single Quote '
  #define HEAD 63
            // Header file
  #define ARR 64
            // Array
  #define SLC 65
            // Single comment '/'
  #define MLCO 66
            // Multiline Comment Open '/*'
  #define MLCC 67
            // Multilien Comment Close '*/'
  #define DEF 68
    // Macro
  #define PRINTF 69
  #define SCANF 70
```

```
  #define FUNC 71
  #define STRING 72
  #define INTCONST 73
  #define FLOATCONST 74
  #define CHARCONST 75
  #define INVALIDSTRING 76
  #define DOT 80
%}

alpha [A-Z||a-z]
digit [0-9]
und [_]
space [ ]

%%
\n          {yylineno++;}
"main(void)" return MAINFUNC;
"main()" return MAINFUNC;
"main(int argc, char **argv)" return MAINFUNC;
"main(int argc, char *argv[])" return MAINFUNC;
"return" return RETURN;
void return VOID;
break    return BREAK;
if return IF;
while return WHILE;
printf return PRINTF;
continue return CONTINUE;
scanf return SCANF;
int return INT;
char return CHAR;
signed return SIGNED;
unsigned return UNSIGNED;
long return LONG;
short return SHORT;
const return CONST;
else return ELSE;
struct return STRUCT;

#include<{alpha}{alpha}*\.h> return HEAD;

#define{space}+{alpha}({alpha}|{digit}|{und})*{space}+{digit}+ return DEF;
#define{space}+{alpha}({alpha}|{digit}|{und})*{space}+({digit}+)\.({digit}+) return DEF;
#define{space}+{alpha}({alpha}|{digit}|{und})*{space}+{alpha}({alpha}|{digit}|{und})* return DEF;

{alpha}({alpha}|{digit}|{und})*    return ID;
{alpha}({alpha}|{digit}|{und})*\[{digit}*\] return ARR;
{digit}+  return INTCONST;
({digit}+)\.({digit}+) return FLOATCONST;
```

```
\"[^\n|^\"]*[\n] return INVALIDSTRING;

{alpha}({alpha}|{digit}|{und})*\({alpha}({alpha}|{digit}|{und}|{space})*\) return FUNC;

\"[^\n]*\" return STRING;
\'{alpha}\' return CHARCONST;

"<=" return LE;
">=" return GE;
"==" return EQ;
"!=" return NE;
">" return G;
"<" return L;

"[|][|]" return OR;
"&&" return AND;
"!" return NOT;

"=" return ASS;
"+=" return ADDASS;
"-=" return SUBASS;
"*=" return MULASS;
"/=" return DIVASS;
"%=" return MODASS;

"+" return PLUS;
"-" return SUB;
"*" return MULT;
"/" return DIV;
"%" return MOD;
"++" return PP;
"--" return MM;

"&" return BA;
"[|]" return BO;
"^" return OC;
"<<" return LS;
">>" return RS;

"//" return SLC;
"/*" return MLCO;
"*/" return MLCC;

";" return SEMICOLON;
"(" return BA1;
")" return BA2;
"[" return BB1;
```

```
"]" return BB2;
"{" return BC1;
"}" return BC2;
"," return COMMA;
"\"" return Q;
"'" return SQ;
\t ;
"." return DOT;
%%

//Data Structure for the symbol and constant table
struct symbol
{
        char token[100]; // Name of the token
        char type[100];          // Token type: Identifier, string constant, floating point constant etc
}symbolTable[100000], constantTable[100000];

int i=0; // Number of symbols in the symbol table
int c=0; // Number of constants in the constant table

//Insert function for symbol/constant table
void symbolInsert(struct symbol table[], int index, char* tokenName, char* tokenType)
{
  strcpy(table[index].token, tokenName);
  strcpy(table[index].type, tokenType);
}

int main(void)
{
  int newToken, // The current token being processed
     j,k, // Iterators
     ba_c=0,ba_o=0,ba_l, // Number of open and close paranthesis, last line where the open parantesis
was used
     bb_o=0,bb_c=0,bb_l, // Number of open and close square braces,  last line where the open sqaure
brace was used
     bc_o=0,bc_c=0,bc_l, // Number of open and close curly braces,  last line where the open curly brace
was used
     rep=0; // Flag to denote whether the current token is already in symbol table

  //Taking the input program
  yyin= fopen("test.c","r");

  //Reading a single token from the program
  newToken = yylex();
  printf("\n");

  int mlc=0, // Flag to denote whether current token is part of a multiline comment
     slcline=0, // Line number of the single line comment
```

```
    mlcline; // Starting line number of multi line comment

 while(newToken)
 {
  rep = 0;

  if(yylineno==slcline) // If token belongs to a single line comment, ignore all the tokens
  {
   newToken=yylex();
   continue;
  }

  for(k=0;k<i;k++) // Checking whether token already exists in symbol table
  {
   if(!(strcmp(symbolTable[k].token,yytext)))
   {
    rep = 1;
    break;
   }
  }

  for(k=0;k<c;k++) // Checking whether token already exists in constant table
  {
   if(!(strcmp(constantTable[k].token,yytext)))
   {
    rep = 1;
    break;
   }
  }

  if(ba_c > ba_o)
   printf("\n-------------ERROR : UNMATCHED ')' at Line %d-----------------\n", yylineno);

  if(bb_c>bb_o)
   printf("\n-------------ERROR : UNMATCHED ']' at Line %d-----------------\n", yylineno);

  if(bc_c>bc_o)
   printf("\n-------------ERROR : UNMATCHED '}' at Line %d-----------------\n", yylineno);

  if(rep==0 && newToken!=65 && newToken!=66 && newToken!=67 && mlc==0)
  {
   strcpy(symbolTable[i].token,yytext);
  }

  if(newToken ==1 && mlc==0)
  {
   printf("%s\t\tWhile Loop----------Line %d\n",yytext,yylineno);
  }
```

```
else if(newToken ==4 && mlc==0)
{
  printf("%s\t\tMain function----------Line %d\n",yytext,yylineno);
}

else if(newToken ==8 && mlc==0)
{
  printf("%s\t\tIf statement----------Line %d\n",yytext,yylineno);
}

else if(newToken ==16 && mlc==0)
{
  printf("%s\t\tElse statement----------Line %d\n",yytext,yylineno);
}

else if(newToken ==17 && mlc==0)
{
  printf("%s\t\tFor Loop----------Line %d\n",yytext,yylineno);
}

else if(newToken ==18 && mlc==0)
{
  printf("%s\t\tStruct definition/declaration----------Line %d\n",yytext,yylineno);
}

else if(((newToken>=1 && newToken<=15)) && mlc==0) // Keywords
{
  printf("%s\t\tKeyword----------Line %d\n",yytext,yylineno);
}

else if(newToken==20 && mlc==0) // Identifiers
{
  if(rep == 0)
  {       symbolInsert(symbolTable, i, yytext, "ID");
          i++;
  }
  printf("%s\t\tIdentifier----------Line %d\n",yytext,yylineno);
}

else if(newToken==73 && mlc==0)
{
  if(rep==0)
  {
   symbolInsert(constantTable, c, yytext, "int");
   c++;
  }
  printf("%s\t\tInteger Constant----------Line %d\n",yytext,yylineno);
```

```
}

else if(newToken==74 && mlc==0)
{
  if(rep==0)
  {
   symbolInsert(constantTable, c, yytext, "float");
   c++;
  }
  printf("%s\t\tFloating Point Constant----------Line %d\n",yytext,yylineno);
}

else if(((newToken>=22 && newToken<=25)||(newToken>=77 && newToken<=78)) && mlc==0)
{
  printf("%s\t\tComparision Operator----------Line %d\n",yytext,yylineno);
}

else if(newToken>=26 && newToken<=28 && mlc==0)
{
  printf("%s\t\tLogical Operator----------Line %d\n",yytext,yylineno);
}

else if(newToken>=29 && newToken<=34 && mlc==0)
{
  printf("%s\t\tAssignment Operator----------Line %d\n",yytext,yylineno);
}

else if(newToken>=35 && newToken<=41 && mlc==0)
{
  printf("%s\t\tArithmetic Operator----------Line %d\n",yytext,yylineno);
}

else if(newToken>=42 && newToken<=47 && mlc==0)
{
  printf("%s\t\tBitwise Operator----------Line %d\n",yytext,yylineno);
}

else if(((newToken>=53 && newToken<=62)||newToken==80) && mlc==0)
{
  if(newToken==54)
  {
    ba_o++;
            ba_l = yylineno;
  }
  if(newToken==55)
    ba_c++;
  if(newToken==56)
  {
```

```
    bb_o++;
    bb_l = yylineno;
  }
  if(newToken==57)
    bb_c++;
  if(newToken==58)
  {
    bc_o++;
    bc_l = yylineno;
  }
  if(newToken==59)
    bc_c++;
  printf("%s\t\tSpecial Character----------Line %d\n",yytext,yylineno);
}

else if(newToken==63 && mlc==0)
{
  printf("%s\t\tHeader----------Line %d\n",yytext,yylineno);
}

else if(newToken==64 && mlc==0)
{
  char id[100] = "";
  for(int t = 0; ; t++)
  {
    if(yytext[t] == '[')
      break;
    id[t] = yytext[t];
  }

  for(k=0;k<i;k++) // Checking whether token already exists in symbol table
  {
        if(!(strcmp(symbolTable[k].token,id)))
        {
        rep = 1;
                break;
        }
  }

  if(rep == 0)
  {
    symbolInsert(symbolTable, i, id, "ID");
        i++;
  }
  printf("%s\t\tArray Identfier----------Line %d\n",yytext,yylineno);
}

else if(newToken==65 && mlc==0)
```

```
{
  printf("%s\t\tSingle Line Comment----------Line %d\n",yytext,yylineno);
  slcline=yylineno;
}

else if(newToken==66)
{
  mlc=1;
  printf("%s\t\tMulti Line Comment Start----------Line %d\n",yytext,yylineno);
  mlcline = yylineno;

}

else if(newToken==66 && mlc==1)
{
  printf("%s\t\tNested multi Line Comment Start----------Line %d\n",yytext,yylineno);
}

else if(newToken==67 && mlc==1)
{
  mlc=0;
  printf("%s\t\tMulti Line Comment End----------Line %d\n",yytext,yylineno);
  mlcline=0;
}

else if(newToken==67 && mlc==0)
  printf("\n---------------ERROR : UNMATCHED NESTED END COMMENT-------------\n");

else if(newToken==68 && mlc==0)
{
  printf("%s\t\tPreprocessor Directive----------Line %d\n",yytext,yylineno);
  newToken=yylex();
  continue;
}

else if(newToken>=69 && newToken<=70 && mlc==0)
{
  printf("%s\t\tPre Defined Function----------Line %d\n",yytext,yylineno);
}

else if(newToken==71 && mlc==0)
{
  char id[100] = "";
  for(int t = 0; ; t++)
  {
    if(yytext[t] == '('
      break;
    id[t] = yytext[t];
```

```
    }

    for(k=0;k<i;k++) // Checking whether token already exists in symbol table
    {
            if(!(strcmp(symbolTable[k].token,id)))
            {
            rep = 1;
                        break;
            }
    }

    if(rep == 0)
    {
      symbolInsert(symbolTable, i, id, "ID");
            i++;
    }

    printf("%s\t\tUser Defined Function----------Line %d\n",yytext,yylineno);
  }

  else if(newToken==72 && mlc==0)
  {
    if(rep==0)
    {
     symbolInsert(constantTable, c, yytext, "string");
     c++;
    }
    printf("%s\t\tString literal----------Line %d\n",yytext, yylineno);
  }

  else if(newToken==75 && mlc==0)
  {
    if(rep==0)
    {
     symbolInsert(constantTable, c, yytext, "char");
     c++;
    }
    printf("%s\t\tCharacter Constant----------Line %d\n",yytext,yylineno);
  }

  else if(newToken==76 && mlc==0)
  {
            printf("\n--------------------ERROR : INCOMPLETE STRING starting at Line %d------------\n",yylineno);
  }

  newToken=yylex();
}
```

```
  if(mlc==1)
    printf("\n--------------------ERROR : UNMATCHED COMMENT starting at Line %d-----------\n",mlcline);

  if(ba_c<ba_o)
    printf("\n--------------------ERROR : UNMATCHED '(' at Line %d ----------------------\n",ba_l);

  if(bb_c<bb_o)
    printf("\n--------------------ERROR : UNMATCHED '[' at Line %d ----------------------\n",bb_l);

  if(bc_c<bc_o)
    printf("\n--------------------ERROR ! UNMATCHED '{' at Line %d ----------------------\n",bc_l);

  printf("\n------------Symbol Table---------------------\n\nSNo\tToken\t\tAttribute\n\n");

  for(j=0;j<i;j++)
    printf("%d\t%s\t\t< %s >\t\t\n",j+1,symbolTable[j].token,symbolTable[j].type);

  printf("\n------------Constant Table---------------------\n\nSNo\tToken\t\tAttribute\n\n");

  for(j=0;j<c;j++)
    printf("%d\t%s\t\t< %s >\t\t\n",j+1,constantTable[j].token,constantTable[j].type);

  return 0;
}

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

Explanation:

Files:

1. Scanner.l: Lex file which generates the stream of tokens and symbol table.
2. Test.c: The input C program

The flex script recognises the following classes of tokens from the input:

• ***Pre-processor instructions***
Statements processed : #include<stdio.h>, #define var1 var2
Token generated : Header / Preprocessor Directive

• ***Single-line comments***
Statements processed : //.....
Token generated : Single Line Comment

• ***Multi-line comments***
Statements processed : /*.....*/, /*.../*...*/
Token generated : Multi Line Comment

• ***Errors for unmatched comments***
Statements processed : /*.....
Token generated : Error with line number

• ***Errors for nested comments***
Statements processed : /*...../*....*/....*/
Token generated : Error with line number

• ***Parentheses (all types)***
Statements processed : (..), {..}, [..] (without errors)
(..)..), {..}..}, [..]..], (..., {..., [... (with errors)
Tokens generated : Parenthesis (without error) / Error with line number (with error)

• ***Operators***

• ***Literals (integer, float, string)***
Statements processed : int, float
Tokens generated : Keyword

• ***Errors for unclean integers and floating point numbers***

• ***Errors for incomplete strings***
Statements processed : char a[]= ”abcd
Tokens generated : Error Incomplete string and line number

• ***Keywords***
Statements processed : if, else, void, while, do, int, float, break, return and so on.
Tokens generated : Keyword

• ***Identifiers***
Statements processed : a, abc, a_b, a12b4
Tokens generated : Identifier

Test Cases:

Without Errors:

Test Case 1:

- Identification of array identifiers

Code:

```
#include<stdio.h>

int main()
{
   int a[3] = { 1, 2 };
  a[2] = a[1] + a[2];
  a[2]++;
  printf("%d", a[2]);
  return 0;
}
```

Output:

```
int             Keyword----------Line 2
 main()         Main function----------Line 2
{               Special Character----------Line 3
   int          Keyword----------Line 4
 a[3]           Array Identfier----------Line 4
 =              Assignment Operator----------Line 4
 {              Special Character----------Line 4
 1              Integer Constant----------Line 4
,               Special Character----------Line 4
 2              Integer Constant----------Line 4
 }              Special Character----------Line 4
;               Special Character----------Line 4
  a[2]          Array Identfier----------Line 5
 =              Assignment Operator----------Line 5
 a[1]           Array Identfier----------Line 5
 +              Arithmetic Operator----------Line 5
 a[2]           Array Identfier----------Line 5
```

```
;               Special Character----------Line 5
  a[2]          Array Identfier----------Line 6
++              Arithmetic Operator----------Line 6
;               Special Character----------Line 6
  printf        Pre Defined Function----------Line 7
(               Special Character----------Line 7
"%d"            String literal----------Line 7
,               Special Character----------Line 7
 a[2]           Array Identfier----------Line 7
)               Special Character----------Line 7
;               Special Character----------Line 7
  return        Keyword----------Line 8
 0              Integer Constant----------Line 8
;               Special Character----------Line 8
}               Special Character----------Line 9

----------Symbol Table----------

SNo     Token           Attribute

1       a               < ID >

----------Constant Table----------

SNo     Token           Attribute

1       1               < int >
2       2               < int >
3       "%d"            < string >
4       0               < int >
```

Test Case 2:

- Single Line Comment
- Unary operator matching

Code:

```
#include<stdio.h>

int main()
{
  //Program to add 2 numbers and increment by 1
  int a[3] = { 1, 2 };
  a[2] = a[1] + a[2];
  a[2]++;
  printf("%d", a[2]);
  return 0;
}
```

Output:

```
int             Keyword----------Line 2
 main()         Main function----------Line 2
{               Special Character----------Line 3
  //            Single Line Comment----------Line 4
        int             Keyword----------Line 5
 a[3]           Array Identfier----------Line 5
 =              Assignment Operator----------Line 5
 {              Special Character----------Line 5
 1              Integer Constant----------Line 5
,               Special Character----------Line 5
 2              Integer Constant----------Line 5
 }              Special Character----------Line 5
;               Special Character----------Line 5
  a[2]          Array Identfier----------Line 6
 =              Assignment Operator----------Line 6
 a[1]           Array Identfier----------Line 6
 +              Arithmetic Operator----------Line 6
 a[2]           Array Identfier----------Line 6
;               Special Character----------Line 6
  a[2]          Array Identfier----------Line 7
++              Arithmetic Operator----------Line 7
;               Special Character----------Line 7
```

```
  printf          Pre Defined Function----------Line 8
(                 Special Character----------Line 8
"%d"              String literal----------Line 8
,                 Special Character----------Line 8
 a[2]             Array Identfier----------Line 8
)                 Special Character----------Line 8
;                 Special Character----------Line 8
  return          Keyword----------Line 9
 0                Integer Constant----------Line 9
;                 Special Character----------Line 9
}                 Special Character----------Line 10

------------Symbol Table---------------------

SNo     Token           Attribute

1       a               < ID >

------------Constant Table---------------------

SNo     Token           Attribute

1       1               < int >
2       2               < int >
3       "%d"            < string >
4       0               < int >
```

Test Case 3:

- Identification of loops

Code:

```
#include<stdio.h>

int main()
{
	int a = 5;
	while(a>0)
	{
		printf("Hello world");
		a--;
	}

}
```

Output:

```
int		Keyword----------Line 2
 main()		Main function----------Line 2
{		Special Character----------Line 3
int		Keyword----------Line 4
 a		Identifier----------Line 4
 =		Assignment Operator----------Line 4
 5		Integer Constant----------Line 4
;		Special Character----------Line 4
while		While Loop----------Line 5
(		Special Character----------Line 5
a		Identifier----------Line 5
>		Comparision Operator----------Line 5
0		Integer Constant----------Line 5
)		Special Character----------Line 5
{		Special Character----------Line 6
printf		Pre Defined Function----------Line 7
(		Special Character----------Line 7
"Hello world"		String literal----------Line 7
)		Special Character----------Line 7
;		Special Character----------Line 7
a		Identifier----------Line 8
--		Arithmetic Operator----------Line 8
```

```
;                   Special Character----------Line 8
}                   Special Character----------Line 9
}                   Special Character----------Line 11

------------Symbol Table---------------------

SNo     Token           Attribute

1       a               < ID >

------------Constant Table---------------------

SNo     Token           Attribute

1       5               < int >
2       0               < int >
3       "Hello world"           < string >
```

Test Case 4:

- Verifying validity of correctly balanced brackets

Code:

```
#include<stdio.h>

int main()
{
        int a = 5;
        while(a>0)
        {
                printf("%d",a);
                a--;
                int b = 4;
                while(b>0)
                {
                        printf("%d", a*b);
                        b--;
                }
        }
}
```

Output:

```
int             Keyword----------Line 2
 main()         Main function----------Line 2
{               Special Character----------Line 3
int             Keyword----------Line 4
 a              Identifier----------Line 4
 =              Assignment Operator----------Line 4
 5              Integer Constant----------Line 4
;               Special Character----------Line 4
while           While Loop----------Line 5
(               Special Character----------Line 5
a               Identifier----------Line 5
>               Comparision Operator----------Line 5
0               Integer Constant----------Line 5
)               Special Character----------Line 5
{               Special Character----------Line 6
printf          Pre Defined Function----------Line 7
(               Special Character----------Line 7
"%d"            String literal----------Line 7
,               Special Character----------Line 7
a               Identifier----------Line 7
)               Special Character----------Line 7
;               Special Character----------Line 7
a               Identifier----------Line 8
--              Arithmetic Operator----------Line 8
;               Special Character----------Line 8
int             Keyword----------Line 9
 b              Identifier----------Line 9
 =              Assignment Operator----------Line 9
 4              Integer Constant----------Line 9
;               Special Character----------Line 9
while           While Loop----------Line 10
(               Special Character----------Line 10
b               Identifier----------Line 10
>               Comparision Operator----------Line 10
0               Integer Constant----------Line 10
)               Special Character----------Line 10
{               Special Character----------Line 11
printf          Pre Defined Function----------Line 12
(               Special Character----------Line 12
"%d"            String literal----------Line 12
,               Special Character----------Line 12
```

```
 a              Identifier----------Line 12
*               Arithmetic Operator----------Line 12
b               Identifier----------Line 12
)               Special Character----------Line 12
;               Special Character----------Line 12
b               Identifier----------Line 13
--              Arithmetic Operator----------Line 13
;               Special Character----------Line 13
}               Special Character----------Line 14
}               Special Character----------Line 15
}               Special Character----------Line 16

------------Symbol Table---------------------

SNo     Token           Attribute

1       a               < ID >
2       b               < ID >

------------Constant Table---------------------

SNo     Token           Attribute

1       5               < int >
2       0               < int >
3       "%d"            < string >
4       4               < int >
```

Test Case 5:

- Identification of user defined functions
- Identification of string literals

Code:

```
#include<stdio.h>

int square(int a)
{
  return(a*a);
}

int main()
{
  int num=2;
  int num2 = square(num);

  printf("Square of %d is %d", num, num2);

  return 0;
}
```

Output:

```
int                 Keyword----------Line 2
 square(int a)              User Defined Function----------Line 2
{                   Special Character----------Line 3
  return            Keyword----------Line 4
(                   Special Character----------Line 4
a                   Identifier----------Line 4
*                   Arithmetic Operator----------Line 4
a                   Identifier----------Line 4
)                   Special Character----------Line 4
;                   Special Character----------Line 4
}                   Special Character----------Line 5
int                 Keyword----------Line 7
 main()             Main function----------Line 7
{                   Special Character----------Line 8
  int               Keyword----------Line 9
 num                Identifier----------Line 9
=                   Assignment Operator----------Line 9
```

```
2                   Integer Constant----------Line 9
;                   Special Character----------Line 9
  int               Keyword----------Line 10
 num2               Identifier----------Line 10
 =                  Assignment Operator----------Line 10
 square(num)              User Defined Function----------Line 10
;                   Special Character----------Line 10
  printf            Pre Defined Function----------Line 12
(                   Special Character----------Line 12
"Square of %d is %d"              String literal----------Line 12
,                   Special Character----------Line 12
 num                Identifier----------Line 12
,                   Special Character----------Line 12
 num2               Identifier----------Line 12
)                   Special Character----------Line 12
;                   Special Character----------Line 12
  return            Keyword----------Line 14
 0                  Integer Constant----------Line 14
;                   Special Character----------Line 14
}                   Special Character----------Line 15

------------Symbol Table---------------------

SNo     Token           Attribute

1       square          < ID >
2       a               < ID >
3       num             < ID >
4       num2            < ID >

------------Constant Table---------------------

SNo     Token           Attribute

1       2               < int >
2       "Square of %d is %d"            < string >
3       0               < int >
```

PASS

With Errors:

***Test Case 6*:**

- Identification of multiline comments
- Error identification on unclosed nested comments

Code:

```
#include<stdio.h>

int main()
{
        int a = 2;
        // printf(“%d”,a);
        a++;
        /* int b = 4;
        int c = 3 */

        int b = 8;
        int c = 3;
        int d = c*(a+b;

        /* printf(“%d”,a);
        a++;
        /* int b = 4;
        int c = 3 */
        a--; */
}
```

Output:

```
int             Keyword----------Line 2
 main()         Main function----------Line 2
{               Special Character----------Line 3
int             Keyword----------Line 4
 a              Identifier----------Line 4
 =              Assignment Operator----------Line 4
 2              Integer Constant----------Line 4
;               Special Character----------Line 4
```

```
//              Single Line Comment----------Line 5
 “”a            Identifier----------Line 6
++              Arithmetic Operator----------Line 6
;               Special Character----------Line 6
/*              Multi Line Comment Start----------Line 7
      */        Multi Line Comment End----------Line 8
int             Keyword----------Line 10
 b              Identifier----------Line 10
 =              Assignment Operator----------Line 10
 8              Integer Constant----------Line 10
;               Special Character----------Line 10
int             Keyword----------Line 11
 c              Identifier----------Line 11
 =              Assignment Operator----------Line 11
 3              Integer Constant----------Line 11
;               Special Character----------Line 11
int             Keyword----------Line 12
 d              Identifier----------Line 12
 =              Assignment Operator----------Line 12
 c              Identifier----------Line 12
*               Arithmetic Operator----------Line 12
(               Special Character----------Line 12
a               Identifier----------Line 12
+               Arithmetic Operator----------Line 12
b               Identifier----------Line 12
;               Special Character----------Line 12
/*              Multi Line Comment Start----------Line 14
 “”/*           Multi Line Comment Start----------Line 16
      */        Multi Line Comment End----------Line 17
a               Identifier----------Line 18
--              Arithmetic Operator----------Line 18
;               Special Character----------Line 18

---------------ERROR : UNMATCHED NESTED END COMMENT-------------
 }              Special Character----------Line 19

--------------------ERROR : UNMATCHED '(' at Line 12 ----------------------

------------Symbol Table---------------------

SNo     Token           Attribute

1       a               < ID >
```

2	b	< ID >
3	c	< ID >
4	d	< ID >

----------Constant Table----------

SNo	Token	Attribute
1	2	< int >
2	8	< int >
3	3	< int >

***Test Case 7*:**

- Error identification on unbalanced brackets

Code:

```
#include<stdio.h>

int main()
{
	int a = 2;
	// printf(“%d”,a);
	a++;
	/* int b = 4;
	int c = 3 */

	int b = 8;
	int c = 3;
	int d = c*(a+b;

	/* printf(“%d”,a);
	a++;
	/* int b = 4;
	int c = 3 */
	a--; */
}
```

Output:

```
int                 Keyword----------Line 2
 main()             Main function----------Line 2
{                   Special Character----------Line 3
int                 Keyword----------Line 4
 a                  Identifier----------Line 4
 =                  Assignment Operator----------Line 4
 2                  Integer Constant----------Line 4
;                   Special Character----------Line 4
//                  Single Line Comment----------Line 5
 “”a                Identifier----------Line 6
++                  Arithmetic Operator----------Line 6
;                   Special Character----------Line 6
/*                  Multi Line Comment Start----------Line 7
       */           Multi Line Comment End----------Line 8
int                 Keyword----------Line 10
 b                  Identifier----------Line 10
 =                  Assignment Operator----------Line 10
 8                  Integer Constant----------Line 10
;                   Special Character----------Line 10
int                 Keyword----------Line 11
 c                  Identifier----------Line 11
 =                  Assignment Operator----------Line 11
 3                  Integer Constant----------Line 11
;                   Special Character----------Line 11
int                 Keyword----------Line 12
 d                  Identifier----------Line 12
 =                  Assignment Operator----------Line 12
 c                  Identifier----------Line 12
*                   Arithmetic Operator----------Line 12
(                   Special Character----------Line 12
a                   Identifier----------Line 12
+                   Arithmetic Operator----------Line 12
b                   Identifier----------Line 12
;                   Special Character----------Line 12
/*                  Multi Line Comment Start----------Line 14
 “”/*               Multi Line Comment Start----------Line 16
       */           Multi Line Comment End----------Line 17
a                   Identifier----------Line 18
--                  Arithmetic Operator----------Line 18
;                   Special Character----------Line 18

----------ERROR : UNMATCHED NESTED END COMMENT----------
 }                  Special Character----------Line 19
```

----------ERROR : UNMATCHED '(' at Line 12 ----------

----------Symbol Table----------

SNo	Token	Attribute
1	a	< ID >
2	b	< ID >
3	c	< ID >
4	d	< ID >

----------Constant Table----------

SNo	Token	Attribute
1	2	< int >
2	8	< int >
3	3	< int >

Test Case 8:

- Identification of preprocessor directives
- Differentiating between a preprocessor directive and a string literal having same
- pattern
- Error detection for incomplete string along with line number corresponding to the
- error
- Identification of floating point constants

Code:

```
#include<stdio.h>
#define NUM 5

int main()
{
char A[] = "#define MAX 10";
char B[ ] = "Hello;
char ch  = 'B';
unsigned int a = 1;
printf("String = %s Value of Pi = %f", A, 3.14);

        return 0;
}
```

Output:

```
#define NUM 5                   Preprocessor Directive----------Line 1
int             Keyword----------Line 3
 main()         Main function----------Line 3
{               Special Character----------Line 4
char            Keyword----------Line 5
 A[]            Array Identfier----------Line 5
 =              Assignment Operator----------Line 5
 "#define MAX 10"              String literal----------Line 5
;               Special Character----------Line 5
char            Keyword----------Line 6
 B              Identifier----------Line 6
[               Special Character----------Line 6
 ]              Special Character----------Line 6
 =              Assignment Operator----------Line 6

--------------------ERROR : INCOMPLETE STRING starting at Line 6------------
char            Keyword----------Line 6
 ch             Identifier----------Line 6
```

```
  =                 Assignment Operator----------Line 6
 'B'                Character Constant----------Line 6
;                   Special Character----------Line 6
unsigned                    Keyword----------Line 7
 int                Keyword----------Line 7
 a                  Identifier----------Line 7
 =                  Assignment Operator----------Line 7
 1                  Integer Constant----------Line 7
;                   Special Character----------Line 7
printf              Pre Defined Function----------Line 8
(                   Special Character----------Line 8
"String = %s Value of Pi = %f"              String literal----------Line 8
,                   Special Character----------Line 8
 A                  Identifier----------Line 8
,                   Special Character----------Line 8
 3.14               Floating Point Constant----------Line 8
)                   Special Character----------Line 8
;                   Special Character----------Line 8
 return             Keyword----------Line 10
 0                  Integer Constant----------Line 10
;                   Special Character----------Line 10
}                   Special Character----------Line 11

----------Symbol Table----------

SNo     Token           Attribute

1       A               < ID >
2       B               < ID >
3       ch              < ID >
4       a               < ID >

----------Constant Table----------

SNo     Token           Attribute

1       "#define MAX 10"                 < string >
2       'B'             < char >
3       1               < int >
4       "String = %s Value of Pi = %f"              < string >
5       3.14            < float >
6       0               < int >
```

Test Case 9:

- Detection of unmatched multiline comments along with line number corresponding to the error

Code:

```
#include<stdio.h>

struct student
{
  int rollNum;
  int marks;
}student1;

int main()
{
 int a = 1, b=0;

 student1.rollNum = 1;
 student1.marks = 90;

 if(a >= 1 && a <= 10)
          b++;

 else
      {  b--;
       /* }
}
```

Output:

```
struct          Struct definition/declaration----------Line 2
 student                Identifier----------Line 2
{               Special Character----------Line 3
  int           Keyword----------Line 4
 rollNum                Identifier----------Line 4
;               Special Character----------Line 4
  int           Keyword----------Line 5
 marks          Identifier----------Line 5
;               Special Character----------Line 5
}               Special Character----------Line 6
```

```
student1            Identifier----------Line 6
;                   Special Character----------Line 6
int                 Keyword----------Line 8
 main()             Main function----------Line 8
{                   Special Character----------Line 9
 int                Keyword----------Line 10
 a                  Identifier----------Line 10
 =                  Assignment Operator----------Line 10
 1                  Integer Constant----------Line 10
,                   Special Character----------Line 10
 b                  Identifier----------Line 10
=                   Assignment Operator----------Line 10
0                   Integer Constant----------Line 10
;                   Special Character----------Line 10
 student1           Identifier----------Line 12
.                   Special Character----------Line 12
rollNum             Identifier----------Line 12
 =                  Assignment Operator----------Line 12
 1                  Integer Constant----------Line 12
;                   Special Character----------Line 12
 student1           Identifier----------Line 13
.                   Special Character----------Line 13
marks               Identifier----------Line 13
 =                  Assignment Operator----------Line 13
 90                 Integer Constant----------Line 13
;                   Special Character----------Line 13
 if                 If statement----------Line 15
(                   Special Character----------Line 15
a                   Identifier----------Line 15
 >=                 Comparision Operator----------Line 15
 1                  Integer Constant----------Line 15
 &&                 Logical Operator----------Line 15
 a                  Identifier----------Line 15
 <=                 Comparision Operator----------Line 15
 10                 Integer Constant----------Line 15
)                   Special Character----------Line 15
   b                Identifier----------Line 16
++                  Arithmetic Operator----------Line 16
;                   Special Character----------Line 16
 else               Else statement----------Line 18
      {             Special Character----------Line 19
  b                 Identifier----------Line 19
--                  Arithmetic Operator----------Line 19
```

; Special Character----------Line 19
/* Multi Line Comment Start----------Line 20

----------ERROR : UNMATCHED COMMENT starting at Line 20----------

----------ERROR ! UNMATCHED '{' at Line 19 ----------

----------Symbol Table----------

SNo	Token	Attribute
1	student	< ID >
2	rollNum	< ID >
3	marks	< ID >
4	student1	< ID >
5	a	< ID >
6	b	< ID >

----------Constant Table----------

SNo	Token	Attribute
1	1	< int >
2	0	< int >
3	90	< int >
4	10	< int >

Test Case 10:

- Identification of structure definitions and structure variable declarations
- Identification of if and else conditional statements
- Detection of unmatched multiline comments along with line number corresponding
- to the error
- Detection of unmatched brackets along with line number corresponding to the error

Code:

```
#include<stdio.h>

struct student
{
  int rollNum;
  int marks;
}student1;

int main()
{
 int a = 1, b=0;

 student1.rollNum = 1;
 student1.marks = 90;

 if(a >= 1 && a <= 10)
            b++;

 else
       {  b--;
        }
}
```

Output:

```
struct          Struct definition/declaration----------Line 2
 student                Identifier----------Line 2
{               Special Character----------Line 3
  int           Keyword----------Line 4
 rollNum                Identifier----------Line 4
;               Special Character----------Line 4
  int           Keyword----------Line 5
```

```
 marks          Identifier----------Line 5
;               Special Character----------Line 5
}               Special Character----------Line 6
student1                Identifier----------Line 6
;               Special Character----------Line 6
int             Keyword----------Line 8
 main()         Main function----------Line 8
{               Special Character----------Line 9
 int            Keyword----------Line 10
 a              Identifier----------Line 10
 =              Assignment Operator----------Line 10
 1              Integer Constant----------Line 10
,               Special Character----------Line 10
 b              Identifier----------Line 10
=               Assignment Operator----------Line 10
0               Integer Constant----------Line 10
;               Special Character----------Line 10
 student1               Identifier----------Line 12
.               Special Character----------Line 12
rollNum                 Identifier----------Line 12
 =              Assignment Operator----------Line 12
 1              Integer Constant----------Line 12
;               Special Character----------Line 12
 student1               Identifier----------Line 13
.               Special Character----------Line 13
marks           Identifier----------Line 13
 =              Assignment Operator----------Line 13
 90             Integer Constant----------Line 13
;               Special Character----------Line 13
 if             If statement----------Line 15
(               Special Character----------Line 15
a               Identifier----------Line 15
 >=             Comparision Operator----------Line 15
 1              Integer Constant----------Line 15
 &&             Logical Operator----------Line 15
 a              Identifier----------Line 15
 <=             Comparision Operator----------Line 15
 10             Integer Constant----------Line 15
)               Special Character----------Line 15
   b            Identifier----------Line 16
++              Arithmetic Operator----------Line 16
;               Special Character----------Line 16
 else           Else statement----------Line 18
```

```
    {           Special Character----------Line 19
  b             Identifier----------Line 19
--              Arithmetic Operator----------Line 19
;               Special Character----------Line 19
     }          Special Character----------Line 20
}               Special Character----------Line 21

----------Symbol Table----------

SNo     Token           Attribute

1       student                 < ID >
2       rollNum                 < ID >
3       marks           < ID >
4       student1                < ID >
5       a               < ID >
6       b               < ID >

----------Constant Table----------

SNo     Token           Attribute

1       1               < int >
2       0               < int >
3       90              < int >
4       10              < int >
```

PASS

Implementation

The Regular Expressions for most of the features of C are fairly straightforward. However, a few features require a significant amount of thought, such as:

- **The Regex for Identifiers:** The lexer must correctly recognize all valid identifiers in C, including the ones having one or more underscores.

    {alpha}({alpha}|{digit}|{und})*

  Where,

    alpha [A-Za-z]

    digit [0-9]

    und [_]

    space [ ]

- **Multiline comments should be supported:** This has been supported by using custom regular algorithm especially robust in cases where tricky characters like * or / are used within the comments. The statements between them has been excluded. Errors for unmatched and nested comments have also been displayed.

- **Literals:** Different regular expressions have been implemented in the code to support all kinds of literals, i.e integers, floats, strings, etc.

    Float : ({digit}+)\.({digit}+)

- **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 the custom defined regular expressions which help throw errors when comment opening or closing is missing.

At the end of the token recognition, the lexer prints a list of all the identifiers and constants present in the program. We use the following technique to implement this:

- ➢ We maintain two linked lists of words, one corresponding to identifiers and other to Constants.
- ➢ Two functions have been implemented, namely add_to_table() and check_present() which is used for adding a new identifier/constant to the linked list and for checking if the identifier/constant is already present in the linked list, respectively.
- ➢ Whenever we encounter an identifier/constant, we call the add_to_table() function which in turns call check_present() and adds it to the corresponding liked list.
- ➢ In the end, in main() function, after yylex returns, we call print_symbol_table(), which in turn prints the list of identifier and constants in a proper format.

Results

1. Token ---- Token Type ---- Line Number
2. Symbol Table : Serial Number ---- Token ---Attribute
3. Constant table
4. Serial Number ---- Token ----Attribute

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.
Features to be added :

1. Nested if else statement
2. enum

References

1. Compilers – Principles, Techniques and Tools By Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman
2. Lex and Yacc By John R. Levine, Tony Mason, Doug Brown
3. http://www.slideshare.net/Tech_MX/symbol-table-design-compiler-construction
4. https://en.wikipedia.org/wiki/Symbol_table
5. http://www.isi.edu/~pedro/Teaching/CSCI565-Spring11/Practice/SDT-Sample.pdf