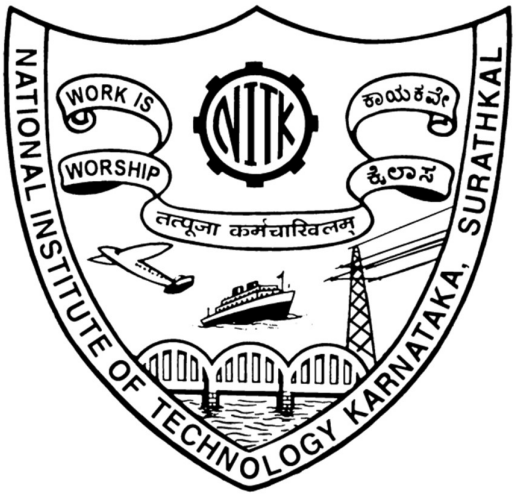
Lexical Analyzer for the C Language



National Institute of Technology Karnataka Surathkal

Date:

29/01/2020

Submitted To:

Mr. Saran Chaitanya

Group Members:

Vasudev B M (17CO150)

S Sai Girish (16CO244)

Gauri V Nair (17CO116)

Abstract:

Aim

To design and implement a lexical analyzer using lex for a subset of the C language.

Features supported

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.

Nature of output

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

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

        #define ID 20 //Identifier
        #define CONST 21      //Constant

      //Relational Operators
        #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 Operators
        #define OR 26
        #define AND 27
        #define NOT 28

      //Assignment Operators
        #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 Operators
        #define PLUS 35
        #define SUB 36
        #define MULT 37
```

```
  #define DIV 38
  #define MOD 39
  #define PP 40          // ++
  #define MM 41          // --

//Bitwise Operators
  #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

// Other tokens such as special characters and parenthesis
  #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        // Multiline 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;
for return FOR;
struct return STRUCT;
define return DEF;

#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}|{und})({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);
}

// Checking whether token already exists in symbol table
int check_symbol_table_present(struct symbol table[], char* yytext_string)
{
        int rep = 0,k;
        for(k=0;k<i;k++)
    {
```

```
      if(!(strcmp(table[k].token,yytext_string)))
      {
        rep = 1;
        break;
      }
    }
    return rep;
}

// Checking whether token already exists in constant table
int check_constant_table_present(struct symbol table[], char* yytext_string)
{
        int rep = 0,k;
        for(k=0;k<c;k++)
    {
      if(!(strcmp(table[k].token,yytext_string)))
      {
        rep = 1;
        break;
      }
    }
    return rep;
}

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 parenthesis, last line where the open
parenthesis was used
      bb_o=0,bb_c=0,bb_l, // Number of open and close square braces,  last line where the
open square 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 test case
  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;
    }

    if(rep == 0)
        rep = check_symbol_table_present(symbolTable,yytext);

    if(rep == 0)
        rep = check_constant_table_present(constantTable,yytext);

    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\tComparison 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];
  }

  if(rep == 0)
    rep = check_symbol_table_present(symbolTable,id);

  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];
  }
```

```
      if (rep == 0)
        rep = check_symbol_table_present(symbolTable,id);

      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(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(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);

  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:

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

- Pre-processor instructions
- Single-line comments
- Multi-line comments
- Errors for unmatched comments
- Errors for nested comments
- Parentheses (all types)
- Operators
- Literals (integer, float, string)
- Errors for unclean integers and floating point numbers
- Errors for incomplete strings
- Keywords
- Identifiers

Keywords accounted for: if, else, void, while, do, int, float, break, return and so on.

Test Cases:

Without Errors:

Sl. No.	Test Case	Expected Output	Status
1.	int x = 3, y=0;	int Keyword----------Line 4 x Identifier----------Line 4 = Assignment Operator----------Line 4 3 Integer Constant---------Line 4 , Special Character--------Line 4 y Identifier----------Line 4 = Assignment Operator----------Line 4 0 Integer Constant---------Line 4 ; Special Character--------Line 4	PASS
2.	for(int i=0; i<n; i++)	for For Loop----------Line 5 (Special Character--------Line 5 int Keyword----------Line 5 i Identifier----------Line 5 = Assignment Operator----------Line 5 0 Integer Constant---------Line 5 ; Special Character--------Line 5 i Identifier----------Line 5 < Comparison Operator----------Line 5 n Identifier----------Line 5 ; Special Character--------Line 5 i Identifier----------Line 5 ++ Arithmetic Operator----------Line 5) Special Character--------Line 5	PASS
3.	y += x*6;	y Identifier----------Line 6 += Assignment Operator----------Line 6 x Identifier----------Line 6 * Arithmetic Operator----------Line 6 6 Integer Constant---------Line 6 ; Special Character--------Line 6	PASS

4.	printf("%d is the final value\n", c);	printf Pre Defined Function----------Line 7 (Special Character----------Line 7 "%d is the final value\n" String literal----------Line 7 , Special Character----------Line 7 c Identifier----------Line 7) Special Character----------Line 7 ; Special Character----------Line 7	PASS
5.	return c ;	return Keyword----------Line 8 c Identifier----------Line 8 ; Special Character----------Line 8	PASS

With Errors:

Sl. No.	Test Case	Expected Output	Status
1.	atmeg = “dfsfds	atmeg Identifier----------Line 11 = Assignment Operator----------Line 11 ----------ERROR : INCOMPLETE STRING starting at Line 11----------	PASS
2.	printf("%d hello);	printf Pre Defined Function----------Line 11 (Special Character--------Line 11 ----------ERROR : INCOMPLETE STRING starting at Line 11---------- ----------ERROR : UNMATCHED '(' at Line 11 ----------	PASS
3.	/* dead meat	/* Multi Line Comment Start----------Line 10 ----------ERROR : UNMATCHED COMMENT starting at Line 10----------	PASS
4.	/* int b = 4; int c = 3 */ a--; */	/* Multi Line Comment Start----------Line 10 */ Multi Line Comment End----------Line 12 a Identifier----------Line 14 -- Arithmetic Operator------Line14 ; Special Character--------Line 14	PASS

		----------ERROR : UNMATCHED NESTED END COMMENT----------	
5.	{ b--; /* } }	{ Special Character--------Line 10 b Identifier----------Line 10 -- Arithmetic Operator-------Line10 ; Special Character--------Line 10 /* Multi Line Comment Start--------- -Line 11 ----------ERROR : UNMATCHED COMMENT starting at Line 11---------- ----------ERROR ! UNMATCHED '{' at Line 10 ----------	PASS

```c
#include<stdio.h>

int main()
{
        //For loop with valid and invalid strings
        char str2;
        int i, s = 0;
        char str[5] = "CDP";
        char str2[5] = "CD_P";
        for(i=1; i <= 6; i++)
                printf("%d",i);

        return 0;
}
```

Figure 1: Input for: For loop with valid and invalid strings

```
File  Edit  View  Search  Terminal  Help
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ lex scanner.l
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ gcc lex.yy.c
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ ./a.out

#include<stdio.h>               Header----------Line 1
int             Keyword----------Line 3
 main()         Main function----------Line 3
{               Special Character----------Line 4
//              Single Line Comment----------Line 5
      char              Keyword----------Line 6
 str2           Identifier----------Line 6
;               Special Character----------Line 6
int             Keyword----------Line 7
 i              Identifier----------Line 7
,               Special Character----------Line 7
 s              Identifier----------Line 7
 =              Assignment Operator----------Line 7
 0              Integer Constant----------Line 7
;               Special Character----------Line 7
char            Keyword----------Line 8
 str[5]         Array Identfier----------Line 8
 =              Assignment Operator----------Line 8
 "CDP"          String literal----------Line 8
;               Special Character----------Line 8
char            Keyword----------Line 9
 str2[5]                Array Identfier----------Line 9
 =              Assignment Operator----------Line 9
 "CD_P"         String literal----------Line 9
;               Special Character----------Line 9
for             For Loop----------Line 10
(               Special Character----------Line 10
i               Identifier----------Line 10
=               Assignment Operator----------Line 10
1               Integer Constant----------Line 10
;               Special Character----------Line 10
 i              Identifier----------Line 10
 <=             Comparision Operator----------Line 10
 6              Integer Constant----------Line 10
;               Special Character----------Line 10
 i              Identifier----------Line 10
++              Arithmetic Operator----------Line 10
)               Special Character----------Line 10
printf          Pre Defined Function----------Line 11
(               Special Character----------Line 11
"%d"            String literal----------Line 11
,               Special Character----------Line 11
i               Identifier----------Line 11
)               Special Character----------Line 11
;               Special Character----------Line 11
return          Keyword----------Line 13
 0              Integer Constant----------Line 13
;               Special Character----------Line 13
}               Special Character----------Line 14

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

SNo     Token           Attribute

1       str2            < ID >
2       i               < ID >
3       s               < ID >
4       str             < ID >

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

SNo     Token           Attribute

1       0               < int >
2       "CDP"           < string >
3       "CD_P"          < string >
4       1               < int >
5       6               < int >
6       "%d"            < string >
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$
```

Figure 2: Output for: For loop with valid and invalid strings

5

```c
#include<stdio.h>

int main()
{
            //Hello
            //Hi
            /*Hi
                        Hello */
            /* Hi HI */ Hello   */
            /* hey !! */ //
            /* return 0;
}
```

Figure 3: Input for: Various forms of multi-line comment

```
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ lex scanner.l
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ gcc lex.yy.c
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ ./a.out

#include<stdio.h>              Header----------Line 1
int             Keyword----------Line 3
 main()         Main function----------Line 3
{               Special Character----------Line 4
//              Single Line Comment----------Line 5
//              Single Line Comment----------Line 6
/*              Multi Line Comment Start----------Line 7
 */             Multi Line Comment End----------Line 8
/*              Multi Line Comment Start----------Line 9
   */           Multi Line Comment End----------Line 9
 Hello          Identifier----------Line 9

---------------ERROR : UNMATCHED NESTED END COMMENT-------------
/*              Multi Line Comment Start----------Line 10
   */           Multi Line Comment End----------Line 10
 //             Single Line Comment----------Line 10
/*              Multi Line Comment Start----------Line 11

--------------------ERROR : UNMATCHED COMMENT starting at Line 11------------

--------------------ERROR ! UNMATCHED '{' at Line 4 -----------------------

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

SNo     Token           Attribute

1       Hello           < ID >

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

SNo     Token           Attribute

(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$
```

Figure 4: Output for: Various forms of multi-line comment

```c
#include<stdio.h>
int main()
{
   //C Program for Binary Search
   int c, first, last, middle, n, search, array[100];

   printf("Enter number of elements\n");
   scanf("%d", &n);

   printf("Enter %d integers\n", n);

   for (c = 0; c < n; c++)
      scanf("%d", &array[c]);

   printf("Enter value to find\n");
   scanf("%d", &search);

   first = 0;
   last = n - 1;
   middle = (first+last)/2;

   while (first <= last) {
      if (array[middle] < search)
         first = middle + 1;
      else if (array[middle] == search) {
         printf("%d found at location %d.\n", search, middle+1);
         break;
      }
      else
         last = middle - 1;

      middle = (first + last)/2;
   }
   if (first > last)
      printf("Not found! %d isn't present in the list.\n", search);

   return 0;
}
```

Figure 5: Input for: Sample C program for binary search.

```
File  Edit  View  Search  Terminal  Help
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ lex scanner.l
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ gcc lex.yy.c
(base) sai-girish@saigirish-Inspiron-5559:~/Documents/Comps/CD-Lab/Compiler-Design-Projects-master/Lexical Analyser$ ./a.out

#include<stdio.h>               Header----------Line 1
int             Keyword----------Line 2
 main()         Main function----------Line 2
{               Special Character----------Line 3
  //            Single Line Comment----------Line 4
     int                Keyword----------Line 5
 c              Identifier----------Line 5
,               Special Character----------Line 5
 first          Identifier----------Line 5
,               Special Character----------Line 5
 last           Identifier----------Line 5
,               Special Character----------Line 5
 middle         Identifier----------Line 5
,               Special Character----------Line 5
 n              Identifier----------Line 5
,               Special Character----------Line 5
 search         Identifier----------Line 5
,               Special Character----------Line 5
 array[100]             Array Identfier----------Line 5
;               Special Character----------Line 5
  printf                Pre Defined Function----------Line 7
(               Special Character----------Line 7
"Enter number of elements\n"            String literal----------Line 7
)               Special Character----------Line 7
;               Special Character----------Line 7
  scanf         Pre Defined Function----------Line 8
(               Special Character----------Line 8
"%d"            String literal----------Line 8
,               Special Character----------Line 8
 &              Bitwise Operator----------Line 8
n               Identifier----------Line 8
)               Special Character----------Line 8
;               Special Character----------Line 8
  printf                Pre Defined Function----------Line 10
(               Special Character----------Line 10
"Enter %d integers\n"           String literal----------Line 10
,               Special Character----------Line 10
 n              Identifier----------Line 10
)               Special Character----------Line 10
;               Special Character----------Line 10
  for           For Loop----------Line 12
 (              Special Character----------Line 12
c               Identifier----------Line 12
 =              Assignment Operator----------Line 12
 0              Integer Constant----------Line 12
;               Special Character----------Line 12
 c              Identifier----------Line 12
 <              Comparision Operator----------Line 12
 n              Identifier----------Line 12
;               Special Character----------Line 12
 c              Identifier----------Line 12
++              Arithmetic Operator----------Line 12
)               Special Character----------Line 12
    scanf               Pre Defined Function----------Line 13
(               Special Character----------Line 13
"%d"            String literal----------Line 13
,               Special Character----------Line 13
 &              Bitwise Operator----------Line 13
array           Identifier----------Line 13
[               Special Character----------Line 13
c               Identifier----------Line 13
]               Special Character----------Line 13
)               Special Character----------Line 13
;               Special Character----------Line 13
  printf                Pre Defined Function----------Line 15
(               Special Character----------Line 15
"Enter value to find\n"         String literal----------Line 15
)               Special Character----------Line 15
;               Special Character----------Line 15
  scanf         Pre Defined Function----------Line 16
(               Special Character----------Line 16
"%d"            String literal----------Line 16
,               Special Character----------Line 16
 &              Bitwise Operator----------Line 16
search          Identifier----------Line 16
search          Identifier----------Line 16
)               Special Character----------Line 16
;               Special Character----------Line 16
  first         Identifier----------Line 18
 =              Assignment Operator----------Line 18
 0              Integer Constant----------Line 18
;               Special Character----------Line 18
  last          Identifier----------Line 19
 =              Assignment Operator----------Line 19
 n              Identifier----------Line 19
 -              Arithmetic Operator----------Line 19
 1              Integer Constant----------Line 19
;               Special Character----------Line 19
  middle                Identifier----------Line 20
 =              Assignment Operator----------Line 20
 (              Special Character----------Line 20
first           Identifier----------Line 20
+               Arithmetic Operator----------Line 20
last            Identifier----------Line 20
)               Special Character----------Line 20
/               Arithmetic Operator----------Line 20
2               Integer Constant----------Line 20
;               Special Character----------Line 20
  while         While Loop----------Line 22
 (              Special Character----------Line 22
first           Identifier----------Line 22
 <=             Comparision Operator----------Line 22
 last           Identifier----------Line 22
)               Special Character----------Line 22
 {              Special Character----------Line 22
    if          If statement----------Line 23
 (              Special Character----------Line 23
array           Identifier----------Line 23
[               Special Character----------Line 23
middle          Identifier----------Line 23
]               Special Character----------Line 23
 <              Comparision Operator----------Line 23
 search         Identifier----------Line 23
)               Special Character----------Line 23
      first             Identifier----------Line 24
 =              Assignment Operator----------Line 24
 middle         Identifier----------Line 24
```

5

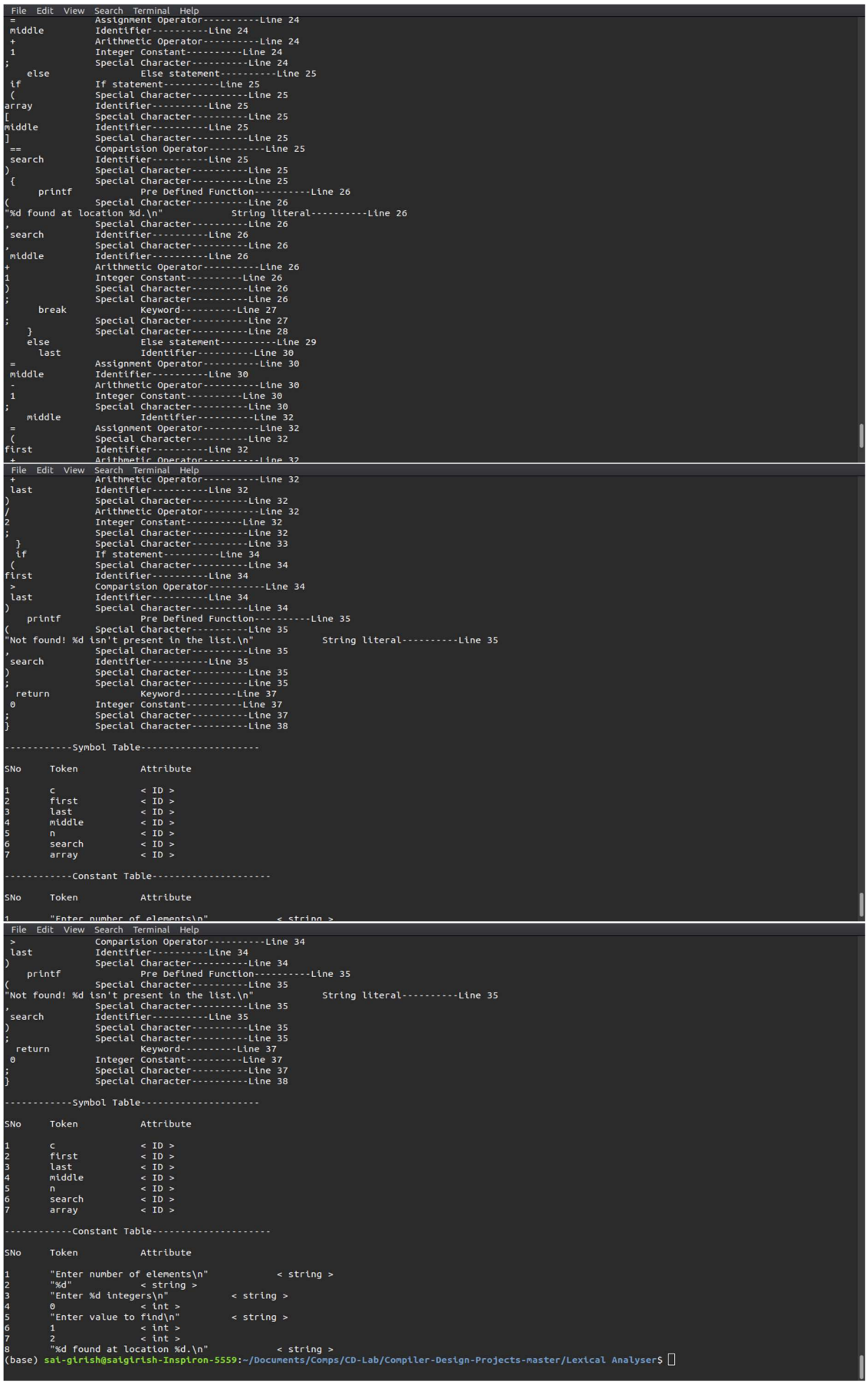


Figure 6: Output for: Sample C program for binary search.

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.
- **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.
- **Literals:** Different regular expressions have been implemented in the code to support all kinds of literals, i.e. integers, floats, strings, etc.
- **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 structures of words, one corresponding to identifiers and other to constants.
- Three functions have been implemented, namely symbolTable(), check_symbol_table_present() and check_constant_table_present() which is used for adding a new identifier/constant to the structure and for checking if the identifier/constant is already present in the structure, respectively.
- Whenever we encounter an identifier/constant, we call the check_symbol_table_present() / constant_table_present() function which returns the value determining presence in Symbol/Constant Table and is used to call symbolTable() to add it to the corresponding structure.
- In the end, in main() function, we print 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:
   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.

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