Lexical Analyzer for the C Language

National Institute of Technology Karnataka Surathkal

Date: 25/01/2017

Submitted To: Uma Priya

Group Members:

Sheetal Shalini (14CO142)

Navya R S (14CO126)

Abstract

A compiler is a computer program that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as object code. When executing, the compiler first parses all of the language statements syntactically one after the other and then, in one or more successive stages or passes, builds the output code, making sure that statements that refer to other statements are referred to correctly in the final code. 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. Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler. We have designed a lexical analyzer for the C language using lex. It takes as input a C code and outputs a stream of tokens. The tokens displayed as part of output include Keyword, Identifier, Constant, Operator, Special Character, Header, Format Specifier, Array, Single Line Comment, Multi Line Comment, Preprocessor Directive, Pre Defined Function, User Defined Function and Main Function. The token, the type of token and the line number of the token in the C code are being displayed. The line number is displayed so that it is easier to debug the code for errors. Errors in single line comments, multi line comments, nested comments, unmatched parenthesis and incomplete strings are displayed along with line numbers. We have also coded the symbol table, which has the columns Serial Number, Token, <Attribute, Attribute Number> and Memory Location.

Contents:

- Introduction -- 1
  - Lexical Analyzer -- 1
  - Flex Script -- 1
  - C Program -- 1
- Design of Programs -- 2
  - Code -- 2
  - Explanation -- 18
- Test Cases --- 20
  - Without Errors -- 20
  - With Errors -- 22
- Implementation -- 25
- Results -- 27
- Future work -- 27
- References --- 27

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.

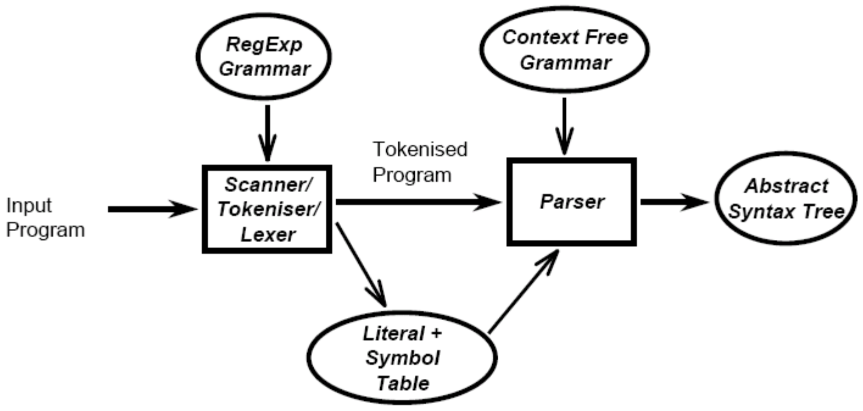
C Program

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.

Lexical analysis only takes care of parsing the tokens and identifying their type. For this reason, we have assumed the C program to be syntactically correct and we generate the stream of tokens as well as the symbol table from it.

Design of Programs

Flow :



Lexical Analyzer generates the tokenized program and symbol table for the input C program.

Code:

Lex Code : (parser.l file)

```
alpha [A-Za-z]

digit [0-9]

und [_]

space [ ]


%{

int yylineno;


#define IF 1

#define ELSE 2

#define INC 3

#define WHILE 4

#define INT 5

#define FLOAT 6
```

```
#define VOID 9
#define RETURN 10
#define MAIN 11
#define BREAK 56

#define ID 12

#define NUM 13

#define LE 14
#define GE 15
#define EQ 16
#define NE 17
#define OR 18
#define AND 19
#define ASS 20
#define PLUS 21
#define SUB 22
#define MULT 23
#define DIV 24
#define MOD 25
#define BA 26
#define BO 27
#define BN 28
#define L 29
#define G 30
#define PP 31
#define MM 32

#define SEMICOLON 33
#define B1 34
#define B2 35
```

```
#define C1 36
#define C2 37
#define D1 38
#define D2 39
#define COMMA 40
#define Q 41
#define COLON 57
#define FULLSTOP 58

#define HEAD 42

#define D 43
#define F 44

#define CA 45
#define CSA 46

#define ARR 47

#define SLC 48
#define MLCO 49
#define MLCC 50
#define DEF 51

#define PRINTF 52
#define SCANF 53
#define FUNC 54
#define MAINFUNC 55

%}

%%
```

```
\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;
else    return ELSE;
printf      return PRINTF;
scanf       return SCANF;
#include<stdio.h> return HEAD;
#include<string.h> return HEAD;
"#define {alpha}({alpha}|{digit}|{und})* {digit}+" return DEF;
"#define {alpha}({alpha}|{digit}|{und})* ({digit}+)\.({digit}+)" return DEF;
"#define {alpha}({alpha}|{digit}|{und})* {alpha}({alpha}|{digit}|{und})*" return DEF;
"//" return SLC;
"/*" return MLCO;
"*/" return MLCC;
{digit}+    return NUM;
({digit}+)\.({digit}+) return NUM;
"<="    return LE;
">="    return GE;
"=="    return EQ;
"!="    return NE;
"||"    return OR;
"&&"    return AND;
while       return WHILE;
";"         return SEMICOLON;
":"         return COLON;
"."         return FULLSTOP;
```

```
int       return INT;
float     return FLOAT;
"("       return B1;
")"       return B2;
"{"       return C1;
"}"       return C2;
"="       return ASS;
"+"       return PLUS;
"-"       return SUB;
"*"       return MULT;
"/"       return DIV;
"%"       return MOD;
"%d"      return D;
"%f"      return F;
"\""      return Q;
","       return COMMA;
",&"      return CA;
", &"     return CSA;
"&"       return BA;
"|"       return BO;
"!"       return BN;
"<"       return L;
">"       return G;
"++"      return PP;
"--"      return MM;
[ \t\n] ;
{alpha}({alpha}|{digit}|{und})*    return ID;
{alpha}({alpha}|{digit}|{und})*\[{digit}*\] return ARR;
{alpha}({alpha}|{digit}|{und})*\(({alpha}|{digit}|{und}|{space})*\) return FUNC;
"["       return D1;
"]"       return D2;
. printf("NA\n");
```

```
%%

struct row
{
        int sno;
        char token[100];
        char attr[100];
        int attrno;
        int addr;
}r[100000];
int main(void)
{
        int
i=0,ptoken,h[20]={0},qline=0,qcnt=0,x=1000,j,k,y=0,b_c=0,b_o=0,c_o=0,c_c=0,d_o=0,d_c=0,ntoken,
vtoken,rep=0;
        ntoken = yylex();
        printf("\n");
        int mlc=0,slcline=0,mlcline;
        while(ntoken) {
                rep = 0;
                if(yylineno==slcline)
                {
                        ntoken=yylex();
                        continue;
                }
                y=0;
                for(k=0;k<i;k++)
                {
                        if(!(strcmp(r[k].token,yytext)))
                        {
                                y = 1;
                                break;
                        }
}
                    if(y==1)
                              rep=1;
                    if(yylineno!=qline && qline!=0)
                    {
                              if(qcnt%2!=0)
                                        printf("\n-------------ERROR ! INCOMPLETE STRING at Line
%d------------------\n",qline);
                              qcnt=0;
                    }
                    if(b_c > b_o)
                              printf("ERROR ! UNMATCHED ')'\n");
                    if(c_c>c_o)
                              printf("ERROR ! UNMATCHED '}'\n");
                    if(d_c>d_o)
                              printf("ERROR ! UNMATCHED ']'\n");
                    if(rep==0 && ntoken!=48 && ntoken!=49 && ntoken!=50 && mlc==0 && qcnt%2==0)
                    {
                              r[i].sno=i+1;
                              strcpy(r[i].token,yytext);
                              r[i].addr = x;
                    }
                    if(((ntoken>=1 && ntoken<=11) || ntoken==56) && mlc==0)
                    {
                              if(rep==0 && qcnt%2==0)
                              {
                                        ptoken=0;
                                        strcpy(r[i].attr,"KEYWORD");
                              }
                              printf("%s\t\tKeyword----------Line %d\n",yytext,yylineno);
                    }
                    else if(ntoken==12 && mlc==0)
```

```
{
	if(rep==0 && qcnt%2==0)
	{
		ptoken=1;
		strcpy(r[i].attr,"ID");
	}
	printf("%s\t\tIdentifier----------Line %d\n",yytext,yylineno);
}
else if(ntoken==13 && mlc==0)
{
	if(rep==0 && qcnt%2==0)
	{
		ptoken=2;
		strcpy(r[i].attr,"CONSTANT");
	}
	printf("%s\t\tConstant----------Line %d\n",yytext,yylineno);
}
else if(ntoken>=14 && ntoken<=32 && mlc==0)
{
	if(rep==0 && qcnt%2==0)
	{
		ptoken=3;
		strcpy(r[i].attr,"OPERATOR");
	}
	printf("%s\t\tOperator----------Line %d\n",yytext,yylineno);
}
else if(ntoken>=33 && ntoken<=41 && mlc==0)
{
	if(ntoken==41)
	{
		qcnt++;
		qline=yylineno;
}
        if(ntoken==34)
                b_o++;
        if(ntoken==35)
                b_c++;
        if(ntoken==36)
                c_o++;
        if(ntoken==37)
                c_c++;
        if(ntoken==38)
                d_o++;
        if(ntoken==39)
                d_c++;
        if(rep==0 && qcnt%2==0)
        {
                ptoken=4;
                strcpy(r[i].attr,"SPECIAL CHAR");
        }
        printf("%s\t\tSpecial Character----------Line %d\n",yytext,yylineno);
}
else if(ntoken==42 && mlc==0)
{
        if(rep==0 && qcnt%2==0)
        {
                ptoken=5;
                strcpy(r[i].attr,"HEADER");
        }
        printf("%s\t\tHeader----------Line %d\n",yytext,yylineno);
}
else if(ntoken>=43 && ntoken<=44 && mlc==0)
{
        if(rep==0 && qcnt%2==0)
```

```
		{
			ptoken=6;
			strcpy(r[i].attr,"FORMAT SPECIFIER");
		}
		printf("%s\t\tFormat Specifier----------Line %d\n",yytext,yylineno);
	}
	else if(ntoken>=45 && ntoken<=46 && mlc==0)
	{
		if(rep==0 && qcnt%2==0)
		{
			ptoken=12;
			strcpy(r[i].attr,"VARIABLE LOCATION");
		}
		printf("%s\t\tVariable Location----------Line %d\n",yytext,yylineno);
	}
	else if(ntoken==47 && mlc==0)
	{
		if(rep==0 && qcnt%2==0)
		{
			ptoken=7;
			strcpy(r[i].attr,"ARRAY");
		}
		printf("%s\t\tArray----------Line %d\n",yytext,yylineno);
	}
	else if(ntoken==48 && mlc==0)
	{
		ptoken=15;
		printf("%s\t\tSingle Line Comment----------Line %d\n",yytext,yylineno);
		slcline=yylineno;
	}
	else if(ntoken==49)
	{
```

```
                mlc=1;

                printf("%s\t\tMulti Line Comment Start----------Line %d\n",yytext,yylineno);

                mlcline = yylineno;


        }

        else if(ntoken==50 && mlc==1)

        {

                mlc=0;

                printf("%s\t\tMulti Line Comment End----------Line %d\n",yytext,yylineno);

                mlcline=0;

        }

        else if(ntoken==50 && mlc==0)

                printf("\n---------------ERROR! UNMATCHED NESTED END
COMMENT-------------\n");

        else if(ntoken==51 && mlc==0)

        {

                if(rep==0 && qcnt%2==0)

                {

                        ptoken=8;

                        strcpy(r[i].attr,"Preprocessor Directive");

                }

                printf("%s\t\tPreprocessor Directive----------Line %d\n",yytext,yylineno);

        }

        else if(ntoken>=52 && ntoken<=53 && mlc==0)

        {

                if(rep==0 && qcnt%2==0)

                {

                        ptoken=9;

                        strcpy(r[i].attr,"Pre Defined Function");

                }

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

        }
```

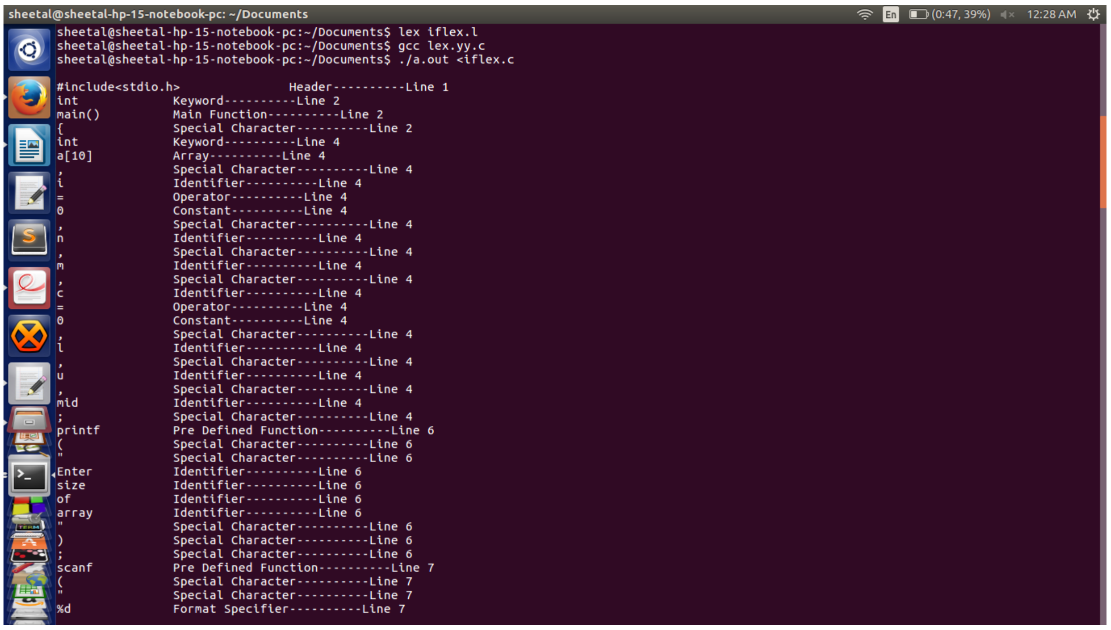
```
		else if(ntoken==54 && mlc==0)
		{
			if(rep==0 && qcnt%2==0)
			{
				ptoken=10;
				strcpy(r[i].attr,"User Defined Function");
			}
			printf("%s\t\tUser Defined Function----------Line %d\n",yytext,yylineno);
		}
		else if(ntoken==55 && mlc==0)
		{
			if(rep==0 && qcnt%2==0)
			{
				ptoken=11;
				strcpy(r[i].attr,"Main Function");
			}
			printf("%s\t\tMain Function----------Line %d\n",yytext,yylineno);
		}
		if(rep==0 && ntoken!=48 && ntoken!=49 && ntoken!=50 && mlc==0 && qcnt%2==0)
		{
			h[ptoken]++;
			r[i].attrno = h[ptoken];
			i++;
			x=x+4;
		}
		ntoken=yylex();
	}
	if(mlc==1)
		printf("\n--------------------ERROR! UNMATCHED COMMENT starting at Line
%d------------\n",mlcline);
	if(b_c!=b_o)
		printf("\n--------------------ERROR ! UNMATCHED '('---------------------\n");
            if(c_c!=c_o)

                        printf("\n-------------------------ERROR ! UNMATCHED '{'----------------\n");

            if(d_c!=d_o)

                        printf("\n--------------------ERROR ! UNMATCHED '['----------------------\n");

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


            for(j=0;j<i;j++)

                        printf("%d\t%s\t\t< %s , %d >\t\t%d\n",r[j].sno,r[j].token,r[j].attr,r[j].attrno,r[j].addr);

            return 0;

}


int yywrap(void)

{

            return 1;

}
```

Input C program (binary_search.c file)

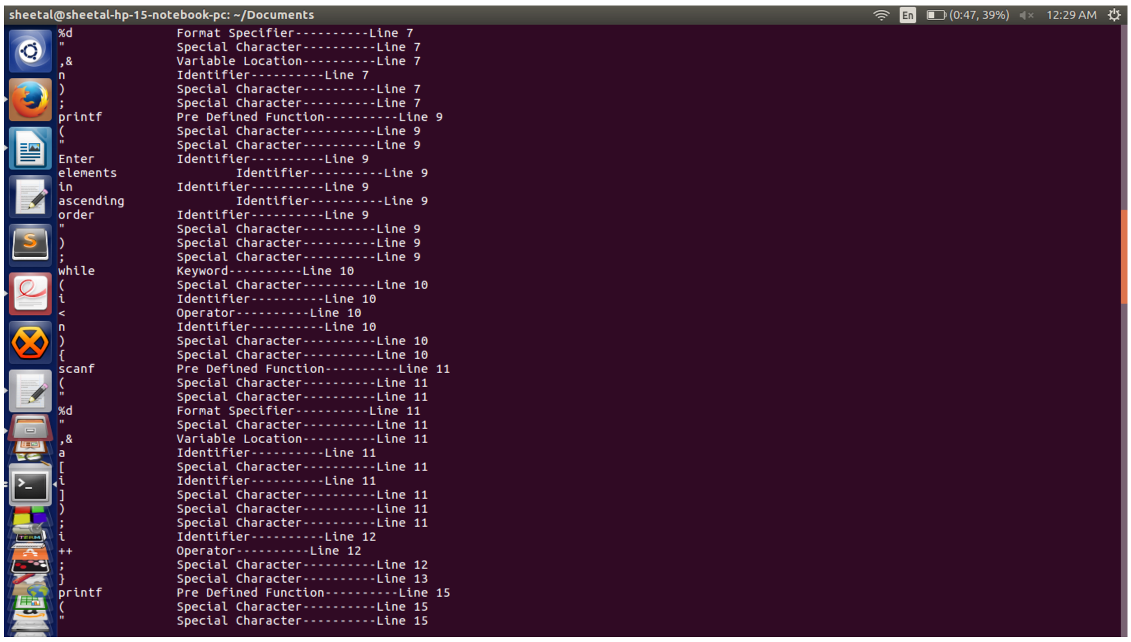
```
#include<stdio.h>

int main(){


   int a[10],i=0,n,m,c=0,l,u,mid;


   printf("Enter size of array");

   scanf("%d",&n);


   printf("Enter elements in ascending order:");

   while(i<n){

       scanf("%d",&a[i]);

            i++;

   }

   printf("Enter key");
```

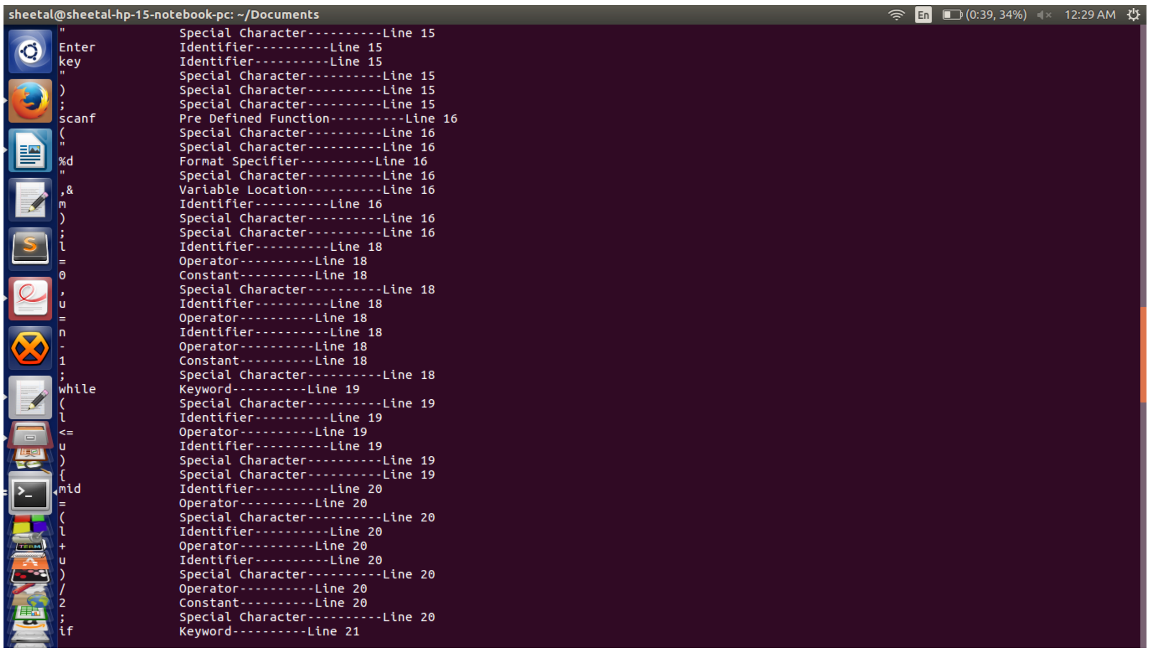
```
    scanf("%d",&m);
    l=0,u=n-1;
    while(l<=u){
        mid=(l+u)/2;
        if(m==a[mid]){
            c=1;
            break;
        }
        else if(m<a[mid])
            u=mid-1;
        else
            l=mid+1;
    }
    if(c==0)
        printf("Number not found");
    else
        printf("Number found");
    return 0;
}
```

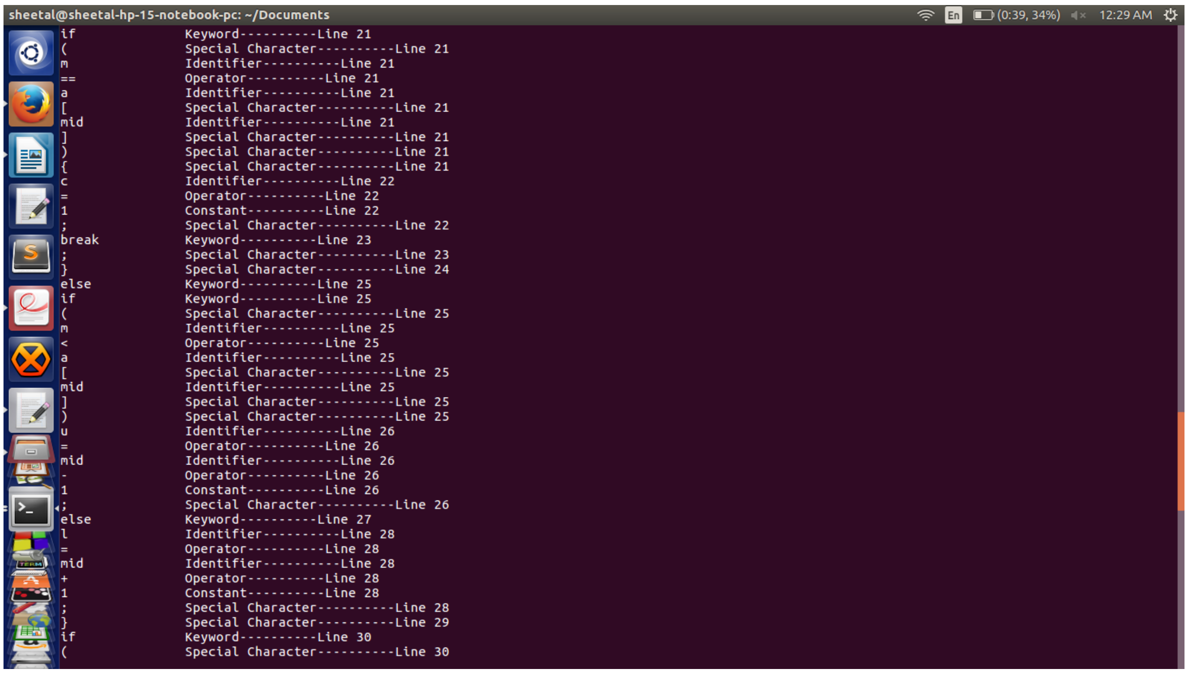
Output :



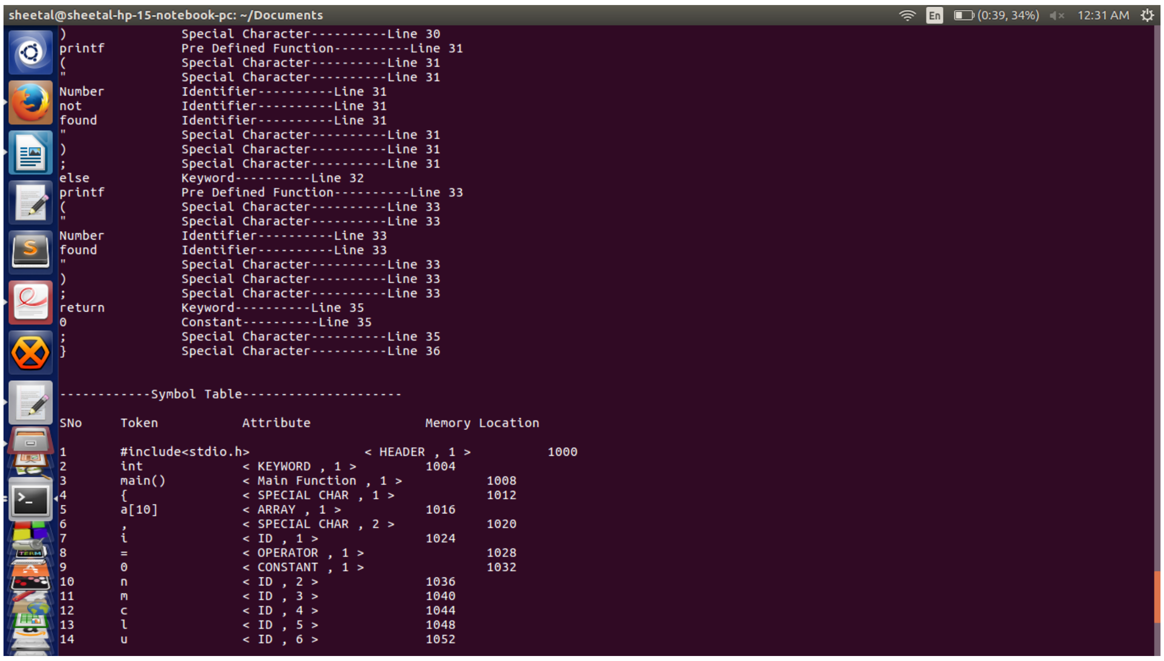
```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ lex iflex.l
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ gcc lex.yy.c
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ ./a.out <iflex.c

#include<stdio.h>                Header---------Line 1
int             Keyword---------Line 2
main()          Main Function---------Line 2
{               Special Character---------Line 2
int             Keyword---------Line 4
a[10]           Array---------Line 4
,               Special Character---------Line 4
i               Identifier---------Line 4
=               Operator---------Line 4
0               Constant---------Line 4
,               Special Character---------Line 4
n               Identifier---------Line 4
,               Special Character---------Line 4
m               Identifier---------Line 4
,               Special Character---------Line 4
c               Identifier---------Line 4
=               Operator---------Line 4
0               Constant---------Line 4
,               Special Character---------Line 4
l               Identifier---------Line 4
,               Special Character---------Line 4
u               Identifier---------Line 4
,               Special Character---------Line 4
mid             Identifier---------Line 4
;               Special Character---------Line 4
printf          Pre Defined Function---------Line 6
(               Special Character---------Line 6
"               Special Character---------Line 6
Enter           Identifier---------Line 6
size            Identifier---------Line 6
of              Identifier---------Line 6
array           Identifier---------Line 6
"               Special Character---------Line 6
)               Special Character---------Line 6
;               Special Character---------Line 6
scanf           Pre Defined Function---------Line 7
(               Special Character---------Line 7
"               Special Character---------Line 7
%d              Format Specifier---------Line 7
```


```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents
%d              Format Specifier----------Line 7
"               Special Character----------Line 7
,&              Variable Location----------Line 7
n               Identifier----------Line 7
)               Special Character----------Line 7
;               Special Character----------Line 7
printf          Pre Defined Function----------Line 9
(               Special Character----------Line 9
"               Special Character----------Line 9
Enter           Identifier----------Line 9
elements                Identifier----------Line 9
in              Identifier----------Line 9
ascending               Identifier----------Line 9
order           Identifier----------Line 9
"               Special Character----------Line 9
)               Special Character----------Line 9
;               Special Character----------Line 9
while           Keyword----------Line 10
(               Special Character----------Line 10
i               Identifier----------Line 10
<               Operator----------Line 10
n               Identifier----------Line 10
)               Special Character----------Line 10
{               Special Character----------Line 10
scanf           Pre Defined Function----------Line 11
(               Special Character----------Line 11
"               Special Character----------Line 11
%d              Format Specifier----------Line 11
"               Special Character----------Line 11
,&              Variable Location----------Line 11
a               Identifier----------Line 11
[               Special Character----------Line 11
i               Identifier----------Line 11
]               Special Character----------Line 11
)               Special Character----------Line 11
;               Special Character----------Line 11
i               Identifier----------Line 12
++              Operator----------Line 12
;               Special Character----------Line 12
}               Special Character----------Line 13
printf          Pre Defined Function----------Line 15
(               Special Character----------Line 15
"               Special Character----------Line 15
```

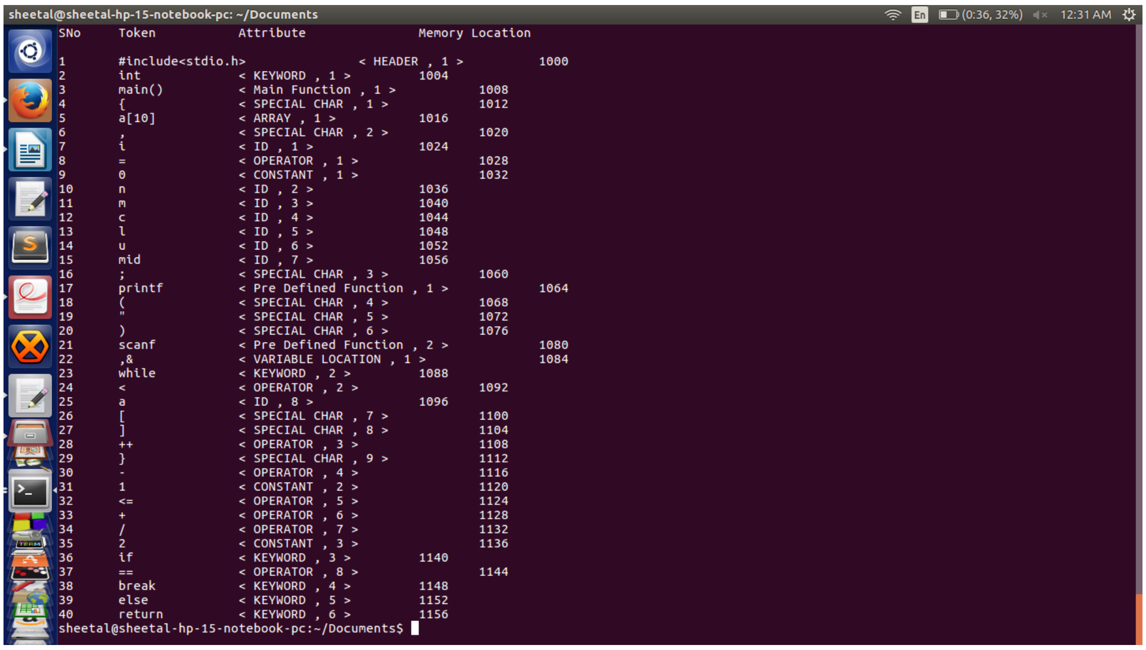

```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents
"               Special Character----------Line 15
Enter           Identifier----------Line 15
key             Identifier----------Line 15
"               Special Character----------Line 15
)               Special Character----------Line 15
;               Special Character----------Line 15
scanf           Pre Defined Function----------Line 16
(               Special Character----------Line 16
"               Special Character----------Line 16
%d              Format Specifier----------Line 16
"               Special Character----------Line 16
,&              Variable Location----------Line 16
m               Identifier----------Line 16
)               Special Character----------Line 16
;               Special Character----------Line 16
l               Identifier----------Line 18
=               Operator----------Line 18
0               Constant----------Line 18
,               Special Character----------Line 18
u               Identifier----------Line 18
=               Operator----------Line 18
n               Identifier----------Line 18
-               Operator----------Line 18
1               Constant----------Line 18
;               Special Character----------Line 18
while           Keyword----------Line 19
(               Special Character----------Line 19
l               Identifier----------Line 19
<=              Operator----------Line 19
u               Identifier----------Line 19
)               Special Character----------Line 19
{               Special Character----------Line 19
mid             Identifier----------Line 20
=               Operator----------Line 20
(               Special Character----------Line 20
l               Identifier----------Line 20
+               Operator----------Line 20
u               Identifier----------Line 20
)               Special Character----------Line 20
/               Operator----------Line 20
2               Constant----------Line 20
;               Special Character----------Line 20
if              Keyword----------Line 21
```



```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents                12:29 AM
if                Keyword---------Line 21
(                 Special Character---------Line 21
m                 Identifier---------Line 21
==                Operator---------Line 21
a                 Identifier---------Line 21
[                 Special Character---------Line 21
mid               Identifier---------Line 21
]                 Special Character---------Line 21
)                 Special Character---------Line 21
{                 Special Character---------Line 21
c                 Identifier---------Line 22
=                 Operator---------Line 22
1                 Constant---------Line 22
;                 Special Character---------Line 22
break             Keyword---------Line 23
;                 Special Character---------Line 23
}                 Special Character---------Line 24
else              Keyword---------Line 25
if                Keyword---------Line 25
(                 Special Character---------Line 25
m                 Identifier---------Line 25
<                 Operator---------Line 25
a                 Identifier---------Line 25
[                 Special Character---------Line 25
mid               Identifier---------Line 25
]                 Special Character---------Line 25
)                 Special Character---------Line 25
u                 Identifier---------Line 26
=                 Operator---------Line 26
mid               Identifier---------Line 26
-                 Operator---------Line 26
1                 Constant---------Line 26
;                 Special Character---------Line 26
else              Keyword---------Line 27
l                 Identifier---------Line 28
=                 Operator---------Line 28
mid               Identifier---------Line 28
+                 Operator---------Line 28
1                 Constant---------Line 28
;                 Special Character---------Line 28
}                 Special Character---------Line 29
if                Keyword---------Line 30
(                 Special Character---------Line 30
```



```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents                12:31 AM
)                 Special Character---------Line 30
printf            Pre Defined Function---------Line 31
(                 Special Character---------Line 31
"                 Special Character---------Line 31
Number            Identifier---------Line 31
not               Identifier---------Line 31
found             Identifier---------Line 31
"                 Special Character---------Line 31
)                 Special Character---------Line 31
;                 Special Character---------Line 31
else              Keyword---------Line 32
printf            Pre Defined Function---------Line 33
(                 Special Character---------Line 33
"                 Special Character---------Line 33
Number            Identifier---------Line 33
found             Identifier---------Line 33
"                 Special Character---------Line 33
)                 Special Character---------Line 33
;                 Special Character---------Line 33
return            Keyword---------Line 35
0                 Constant---------Line 35
;                 Special Character---------Line 35
}                 Special Character---------Line 36


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

SNo     Token           Attribute               Memory Location

1       #include<stdio.h>               < HEADER , 1 >          1000
2       int             < KEYWORD , 1 >         1004
3       main()          < Main Function , 1 >           1008
4       {               < SPECIAL CHAR , 1 >            1012
5       a[10]           < ARRAY , 1 >           1016
6       ,               < SPECIAL CHAR , 2 >            1020
7       i               < ID , 1 >              1024
8       =               < OPERATOR , 1 >                1028
9       0               < CONSTANT , 1 >                1032
10      n               < ID , 2 >              1036
11      m               < ID , 3 >              1040
12      c               < ID , 4 >              1044
13      l               < ID , 5 >              1048
14      u               < ID , 6 >              1052
```



```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents
SNo     Token           Attribute               Memory Location

1       #include<stdio.h>               < HEADER , 1 >          1000
2       int             < KEYWORD , 1 >         1004
3       main()          < Main Function , 1 >           1008
4       {               < SPECIAL CHAR , 1 >            1012
5       a[10]           < ARRAY , 1 >           1016
6       ,               < SPECIAL CHAR , 2 >            1020
7       i               < ID , 1 >              1024
8       =               < OPERATOR , 1 >                1028
9       0               < CONSTANT , 1 >                1032
10      n               < ID , 2 >              1036
11      m               < ID , 3 >              1040
12      c               < ID , 4 >              1044
13      l               < ID , 5 >              1048
14      u               < ID , 6 >              1052
15      mid             < ID , 7 >              1056
16      ;               < SPECIAL CHAR , 3 >            1060
17      printf          < Pre Defined Function , 1 >            1064
18      (               < SPECIAL CHAR , 4 >            1068
19      "               < SPECIAL CHAR , 5 >            1072
20      )               < SPECIAL CHAR , 6 >            1076
21      scanf           < Pre Defined Function , 2 >            1080
22      ,&              < VARIABLE LOCATION , 1 >               1084
23      while           < KEYWORD , 2 >         1088
24      <               < OPERATOR , 2 >                1092
25      a               < ID , 8 >              1096
26      [               < SPECIAL CHAR , 7 >            1100
27      ]               < SPECIAL CHAR , 8 >            1104
28      ++              < OPERATOR , 3 >                1108
29      }               < SPECIAL CHAR , 9 >            1112
30      -               < OPERATOR , 4 >                1116
31      1               < CONSTANT , 2 >                1120
32      <=              < OPERATOR , 5 >                1124
33      +               < OPERATOR , 6 >                1128
34      /               < OPERATOR , 7 >                1132
35      2               < CONSTANT , 3 >                1136
36      if              < KEYWORD , 3 >         1140
37      ==              < OPERATOR , 8 >                1144
38      break           < KEYWORD , 4 >         1148
39      else            < KEYWORD , 5 >         1152
40      return          < KEYWORD , 6 >         1156
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ 
```

Explanation :

Files :

1. **Parser.l** : Lex file which generates the stream of tokens and symbol table.
2. **Binary_search.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

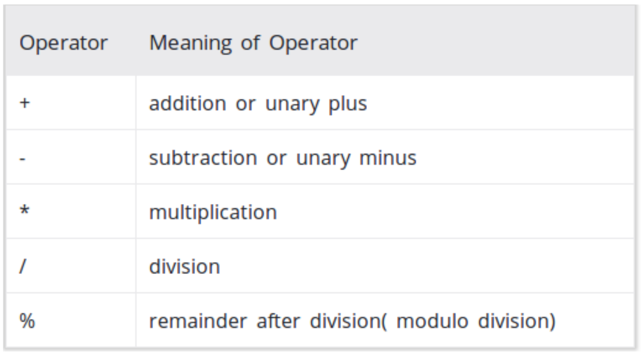
- Parentheses (all types)

  Statements processed : *(..), {..}, [..]   (without errors)*

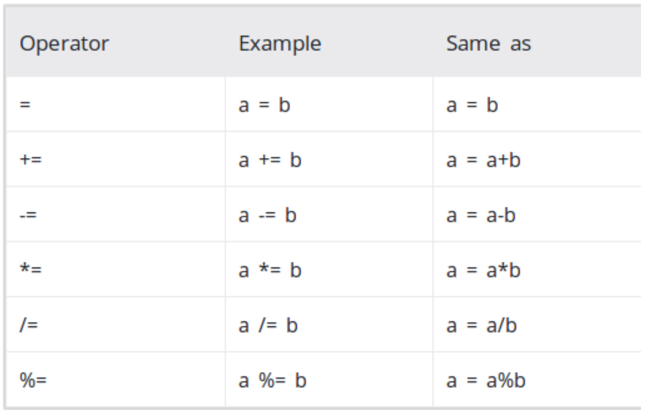
  (..)..), {..}..}, [..]..], (...., {...., [...   (with errors)

  Tokens generated : Parenthesis (without error) / Error with line number (with error)

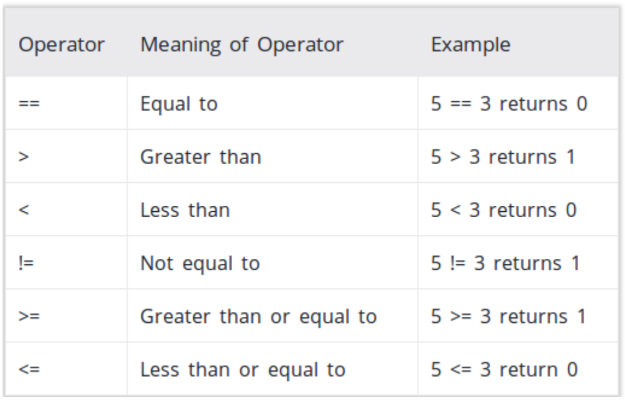
- Operators



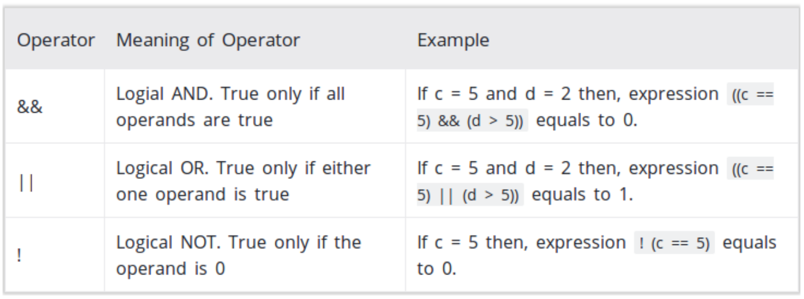
Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)



Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b



Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0



Operator	Meaning of Operator	Example
&&	Logial AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c == 5) && (d > 5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c == 5) || (d > 5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression ! (c == 5) equals to 0.

- Literals

  Statements processed : *int, float*

  Tokens generated : Keyword
- 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

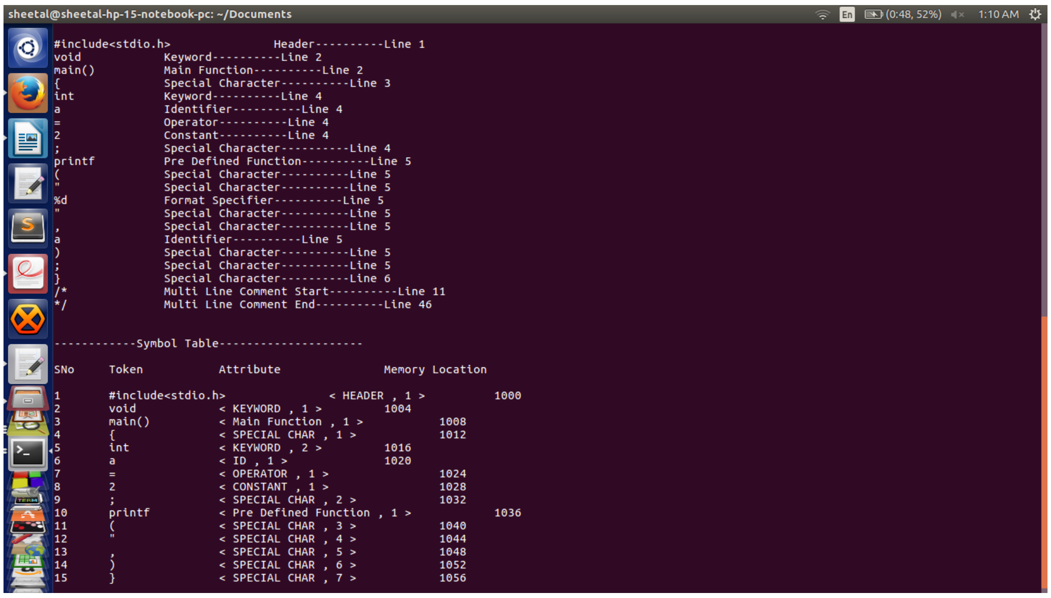
Test Cases:

Without Errors:

1. Code :

```
#include<stdio.h>
void main()
{
        int a=2;
        printf("%d",a);
}
```

Output :

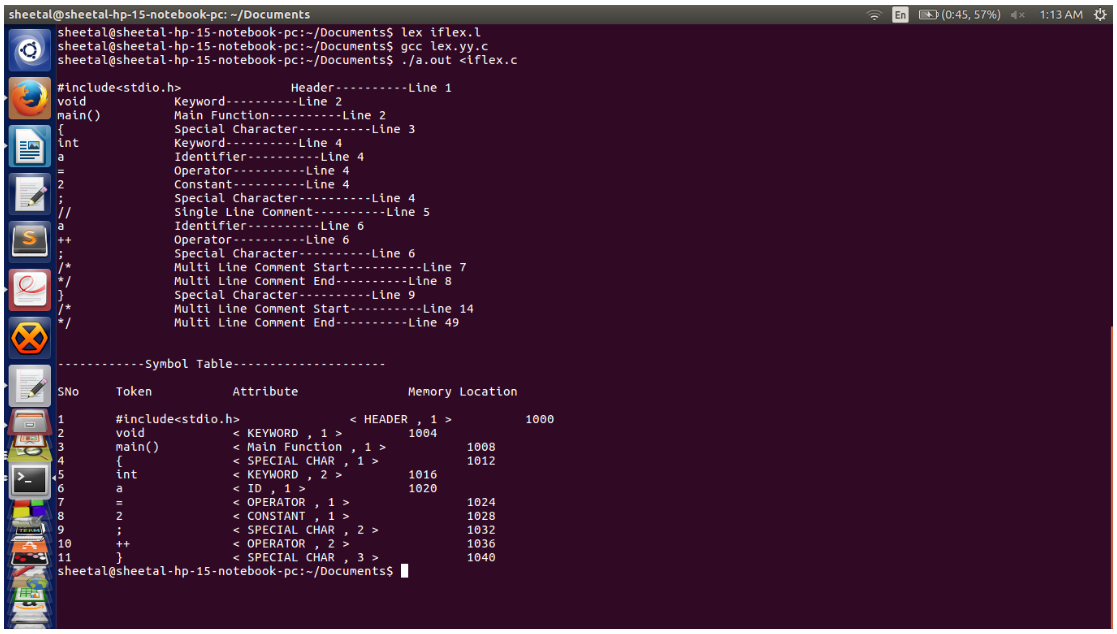


```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents

#include<stdio.h>               Header---------Line 1
void            Keyword---------Line 2
main()          Main Function---------Line 2
{               Special Character---------Line 3
int             Keyword---------Line 4
a               Identifier---------Line 4
=               Operator---------Line 4
2               Constant---------Line 4
;               Special Character---------Line 4
printf          Pre Defined Function---------Line 5
(               Special Character---------Line 5
"               Special Character---------Line 5
%d              Format Specifier---------Line 5
"               Special Character---------Line 5
,               Special Character---------Line 5
a               Identifier---------Line 5
)               Special Character---------Line 5
;               Special Character---------Line 5
}               Special Character---------Line 6
/*              Multi Line Comment Start---------Line 11
*/              Multi Line Comment End---------Line 46


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

SNo     Token           Attribute               Memory Location

1       #include<stdio.h>               < HEADER , 1 >          1000
2       void            < KEYWORD , 1 >         1004
3       main()          < Main Function , 1 >           1008
4       {               < SPECIAL CHAR , 1 >            1012
5       int             < KEYWORD , 2 >         1016
6       a               < ID , 1 >              1020
7       =               < OPERATOR , 1 >                1024
8       2               < CONSTANT , 1 >                1028
9       ;               < SPECIAL CHAR , 2 >            1032
10      printf          < Pre Defined Function , 1 >            1036
11      (               < SPECIAL CHAR , 3 >            1040
12      "               < SPECIAL CHAR , 4 >            1044
13      ,               < SPECIAL CHAR , 5 >            1048
14      )               < SPECIAL CHAR , 6 >            1052
15      }               < SPECIAL CHAR , 7 >            1056
```

2. Code (with single & multi line comments)

```
#include<stdio.h>

void main()

{

        int a=2;

        //printf("%d",a);

        a++;

        /* b=2;

        c=4; */

}
```

Output :



```
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ lex iflex.l
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ gcc lex.yy.c
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ ./a.out <iflex.c

#include<stdio.h>               Header----------Line 1
void            Keyword----------Line 2
main()          Main Function----------Line 2
{               Special Character----------Line 3
int             Keyword----------Line 4
a               Identifier----------Line 4
=               Operator----------Line 4
2               Constant----------Line 4
;               Special Character----------Line 4
//              Single Line Comment----------Line 5
a               Identifier----------Line 6
++              Operator----------Line 6
;               Special Character----------Line 6
/*              Multi Line Comment Start----------Line 7
*/              Multi Line Comment End----------Line 8
}               Special Character----------Line 9
/*              Multi Line Comment Start----------Line 14
*/              Multi Line Comment End----------Line 49


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

SNo     Token           Attribute               Memory Location

1       #include<stdio.h>               < HEADER , 1 >          1000
2       void            < KEYWORD , 1 >         1004
3       main()          < Main Function , 1 >           1008
4       {               < SPECIAL CHAR , 1 >            1012
5       int             < KEYWORD , 2 >         1016
6       a               < ID , 1 >              1020
7       =               < OPERATOR , 1 >                1024
8       2               < CONSTANT , 1 >                1028
9       ;               < SPECIAL CHAR , 2 >            1032
10      ++              < OPERATOR , 2 >                1036
11      }               < SPECIAL CHAR , 3 >            1040
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ █
```

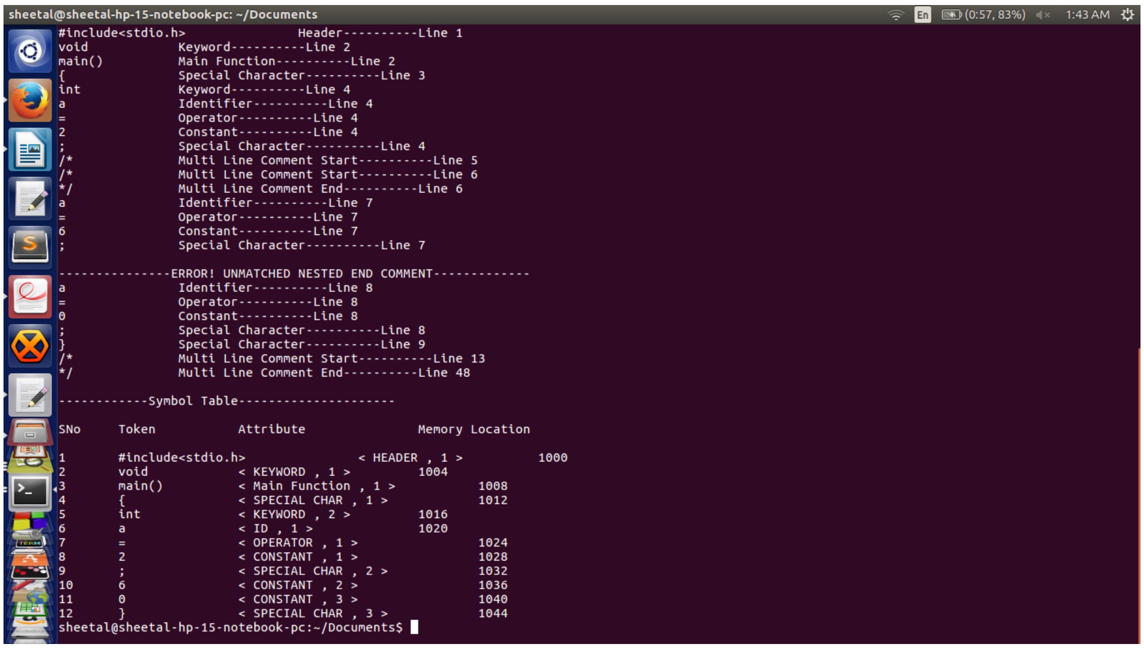
Statement	Output	Status
Int a =2;	Int : Keyword a : Identifier = : Assignment 2 : Constant ; : Semicolon	PASS
// printf(“%d”,a);	// : Single Line Comment	PASS
/* b=2; c=4; */	/* : Multi Line Comment start */ : Multi Line comment end	PASS

With Errors :

1. Code (with nested multiline comment errors)

```
#include<stdio.h>
void main()
{
        int a=2;
        /*printf("%d",a);
        /*a++;*/
        a = 6;*/
        a = 0;
}
```

Output:

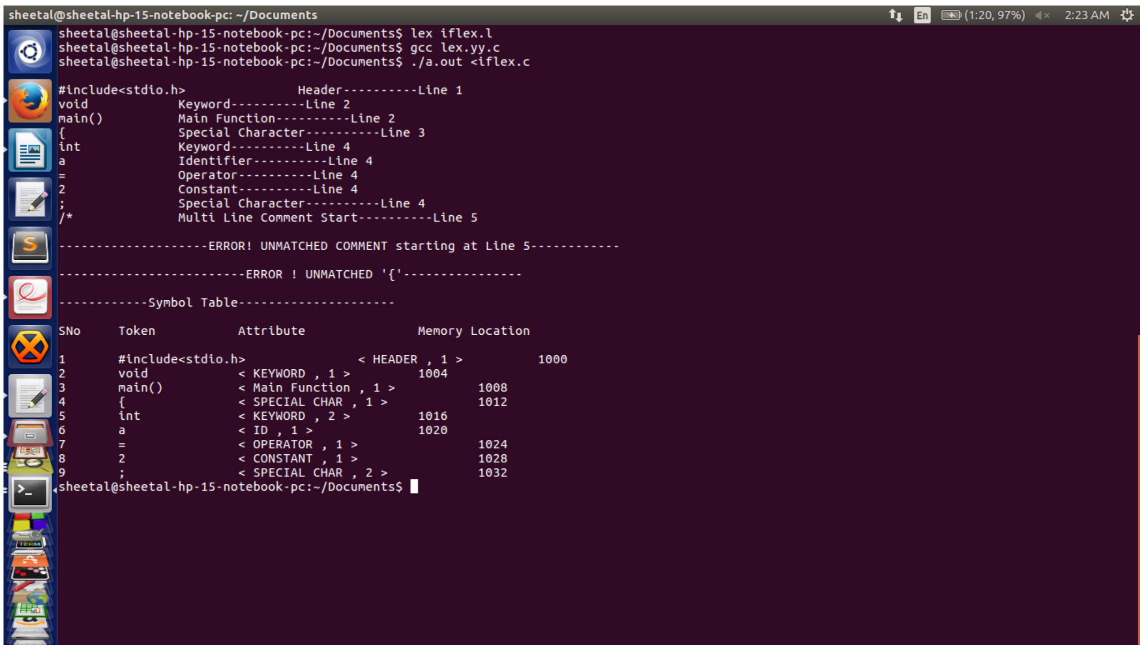

sheetal@sheetal-hp-15-notebook-pc: ~/Documents

```
#include<stdio.h>               Header---------Line 1
void            Keyword---------Line 2
main()          Main Function---------Line 2
{               Special Character---------Line 3
int             Keyword---------Line 4
a               Identifier---------Line 4
=               Operator---------Line 4
2               Constant---------Line 4
;               Special Character---------Line 4
/*              Multi Line Comment Start---------Line 5
/*              Multi Line Comment Start---------Line 6
*/              Multi Line Comment End---------Line 6
a               Identifier---------Line 7
=               Operator---------Line 7
6               Constant---------Line 7
;               Special Character---------Line 7

---------------ERROR! UNMATCHED NESTED END COMMENT-------------
a               Identifier---------Line 8
=               Operator---------Line 8
0               Constant---------Line 8
;               Special Character---------Line 8
}               Special Character---------Line 9
/*              Multi Line Comment Start---------Line 13
*/              Multi Line Comment End---------Line 48

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

SNo     Token           Attribute               Memory Location

1       #include<stdio.h>               < HEADER , 1 >          1000
2       void            < KEYWORD , 1 >         1004
3       main()          < Main Function , 1 >           1008
4       {               < SPECIAL CHAR , 1 >            1012
5       int             < KEYWORD , 2 >         1016
6       a               < ID , 1 >              1020
7       =               < OPERATOR , 1 >                1024
8       2               < CONSTANT , 1 >                1028
9       ;               < SPECIAL CHAR , 2 >            1032
10      6               < CONSTANT , 2 >                1036
11      0               < CONSTANT , 3 >                1040
12      }               < SPECIAL CHAR , 3 >            1044
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ 
```

2. Code (with unmatched multi line comment errors)

```
#include<stdio.h>

void main()

{

        int a=2;

        /*printf("%d",a);

        a++;

}
```

Output :

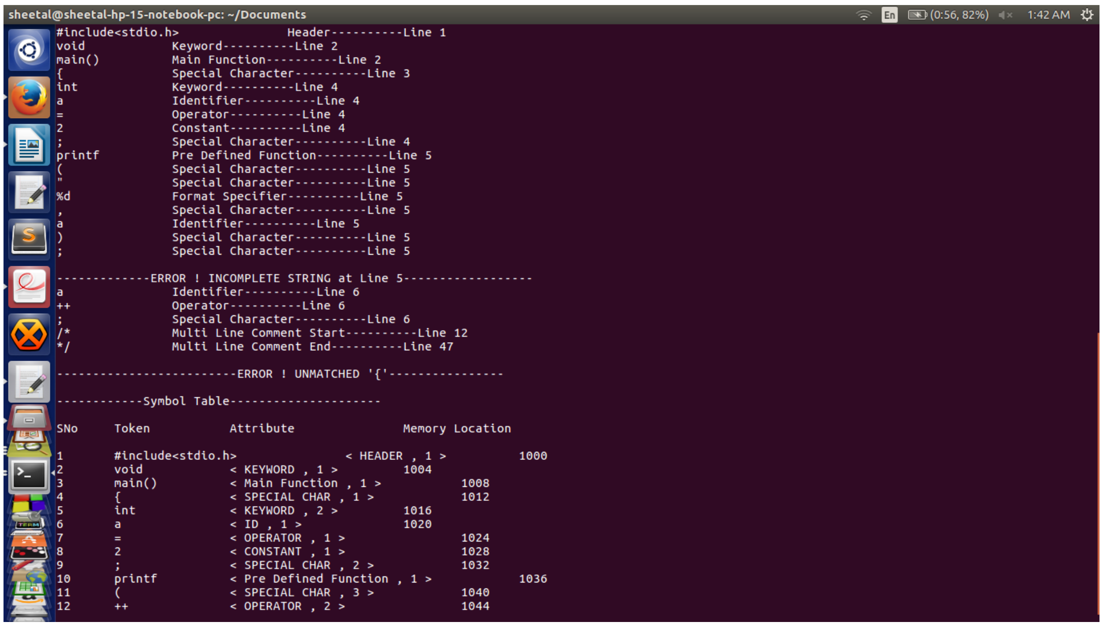


```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ lex iflex.l
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ gcc lex.yy.c
sheetal@sheetal-hp-15-notebook-pc:~/Documents$ ./a.out <iflex.c

#include<stdio.h>               Header---------Line 1
void            Keyword---------Line 2
main()          Main Function---------Line 2
{               Special Character---------Line 3
int             Keyword---------Line 4
a               Identifier---------Line 4
=               Operator---------Line 4
2               Constant---------Line 4
;               Special Character---------Line 4
/*              Multi Line Comment Start---------Line 5

--------------------ERROR! UNMATCHED COMMENT starting at Line 5-----------

------------------------ERROR ! UNMATCHED '{'---------------

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

SNo     Token           Attribute               Memory Location

1       #include<stdio.h>               < HEADER , 1 >          1000
2       void            < KEYWORD , 1 >         1004
3       main()          < Main Function , 1 >           1008
4       {               < SPECIAL CHAR , 1 >            1012
5       int             < KEYWORD , 2 >         1016
6       a               < ID , 1 >              1020
7       =               < OPERATOR , 1 >                1024
8       2               < CONSTANT , 1 >                1028
9       ;               < SPECIAL CHAR , 2 >            1032
sheetal@sheetal-hp-15-notebook-pc:~/Documents$
```

3.Code (with unmatched quotes and unmatched paranthesis errors)

```
#include<stdio.h>

void main()

{

        int a=2;

        printf("%d,a);

        a++;
```

Output:



```
sheetal@sheetal-hp-15-notebook-pc: ~/Documents
#include<stdio.h>                Header---------Line 1
void            Keyword---------Line 2
main()          Main Function---------Line 2
{               Special Character---------Line 3
int             Keyword---------Line 4
a               Identifier---------Line 4
=               Operator---------Line 4
2               Constant---------Line 4
;               Special Character---------Line 4
printf          Pre Defined Function---------Line 5
(               Special Character---------Line 5
"               Special Character---------Line 5
%d              Format Specifier---------Line 5
,               Special Character---------Line 5
a               Identifier---------Line 5
)               Special Character---------Line 5
;               Special Character---------Line 5

------------ERROR ! INCOMPLETE STRING at Line 5-----------------
a               Identifier---------Line 6
++              Operator---------Line 6
;               Special Character---------Line 6
/*              Multi Line Comment Start---------Line 12
*/              Multi Line Comment End---------Line 47

-------------------------ERROR ! UNMATCHED '{'---------------

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

SNo     Token           Attribute               Memory Location

1       #include<stdio.h>               < HEADER , 1 >          1000
2       void            < KEYWORD , 1 >         1004
3       main()          < Main Function , 1 >           1008
4       {               < SPECIAL CHAR , 1 >            1012
5       int             < KEYWORD , 2 >         1016
6       a               < ID , 1 >              1020
7       =               < OPERATOR , 1 >                1024
8       2               < CONSTANT , 1 >                1028
9       ;               < SPECIAL CHAR , 2 >            1032
10      printf          < Pre Defined Function , 1 >            1036
11      (               < SPECIAL CHAR , 3 >            1040
12      ++              < OPERATOR , 2 >                1044
```

Statement	Output	Status

/*printf("%d",a);
/*a++;*/
a = 6;*/
	ERROR ! Unmatched nested end comment	PASS

/*printf("%d",a);	ERROR ! Unmatched comment starting at line 5	PASS
printf("%d,a);	ERROR ! Incomplete string at line 5	PASS

{
int a;
	ERROR ! Unmatched ‘{‘	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 are supported:** This has been supported by checking the occurence of ‘/*’ and ‘*/’ in the code. 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, etc.

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

- **User Defined Functions :**
  {alpha}({alpha}|{digit}|{und})*\(({alpha}|{digit}|{und}|{space})*\)
- **Arrays:**
  {alpha}({alpha}|{digit}|{und})*\[{digit}*\]
  Where,

  ```
  alpha [A-Za-z]
        digit [0-9]
  und [_]
  space [ ]
  ```

- **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 lexer prints a list of all the tokens present in the program. We use the following technique to implement this:

- We have assigned unique integers to all different kinds of tokens present in the C code.
- Based on these integers, we have displayed the type of the token.
- For storing these tokens and their attributes in the symbol table, we have defined a structure.

```
struct row
{
        int sno;
        char token[100];
        char attr[100];
        int attrno;
        int addr;
}r[100000];
```

- As and when successive tokens are encountered, their respective values are stored in the structure and then later displayed.
- We also have functionalities for checking and accordingly omitting duplicate entries in the symbol table.

- In the end, each token is printed along with its type and line number.
- Errors like unmatched multi line comment, nested multi line comments, incomplete strings and unmatched parenthesis are also displayed along with their line numbers.
- The symbol table is displayed, having columns Serial Number, Token, attribute, attribute number and memory address.

Results

1. Token ---- Token Type ---- Line Number
2. Symbol Table :

   Serial Number ---- Token ---<Attribute, Attribute Number>----Memory Location

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. Nested while loop
3. Structures
4. 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