COMPILER DESIGN LAB-1 TELUGU LEXICAL ANALYZER

Team members:

Karthik Chittoor – 106121033

Vishnu Vardhan P – 106121089

Pranav Prakash - 106121093

A.Components of our Programming Language(Telugu).

```
Syntax for our programming language will be a mix of C++,Python and Javascript
```

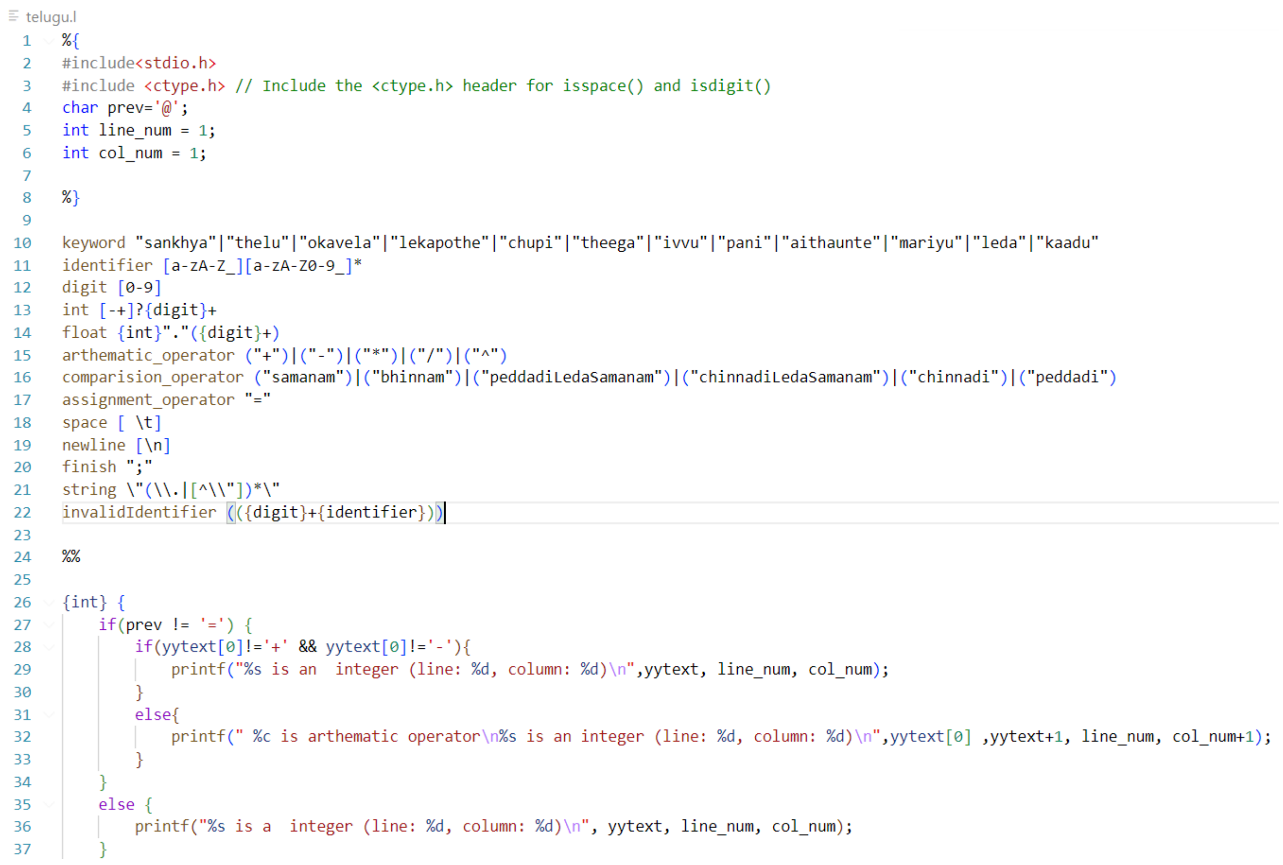
```
// translations
   sankhya - int
    thelu-float
    aksharam-char
    okavela-if
    lekaokavela-elif
    lekapothe-else
    chupi-print
    theega - string
    ivvu - return
    pani - function
    aithaunte - while
    mariyu - and
    leda - or
    kaddu - not
    pratyekam - xor
    samanam - ==
    bhinnam - !=
    peddadiLedaSamanam - >=
    chinnadiLedaSamanam - <=
    chinnadi - <
    peddadi - >
```

```
TELUGU_DATATYPE: sankhya,thelu,aksharam,theega,
TELUGU_COMPARISION_OPERATOR: samanam,bhinnam,peddadiLedaSamanam,chinnadiLedaSamanam,chinnadi,peddadi
TELUGU_IF: okavela
TELUGU_ELIF: lekaokavela
TELUGU_ELSE: lekapothe
TELUGU_WHILE: aithaunte
TELUGU_FUNCTION: pani
TELUGU_RETURN: ivvu
```

B.Regular Expression:

```
keyword "sankhya"|"thelu"|"okavela"|"lekapothe"|"chupi"|"theega"|"ivvu"|"pani"|"aithaunte"|"mariyu"|"leda"|"kaadu"
identifier [a-zA-Z_][a-zA-Z0-9_]*
digit [0-9]
int [-+]?{digit}+
float {int}"."({digit}+)
arthematic_operator ("+")|("-")|("*")|("/")|("^")
comparision_operator ("samanam")|("bhinnam")|("peddadiLedaSamanam")|("chinnadiLedaSamanam")|("chinnadi")|("peddadi")
assignment_operator "="
space [ \t]
newline [\n]
finish ";"
string \"(\\.|[^\\"])*\"
invalidIdentifier (({digit}+{identifier}))
```

C. LEX Code :



telugu.l

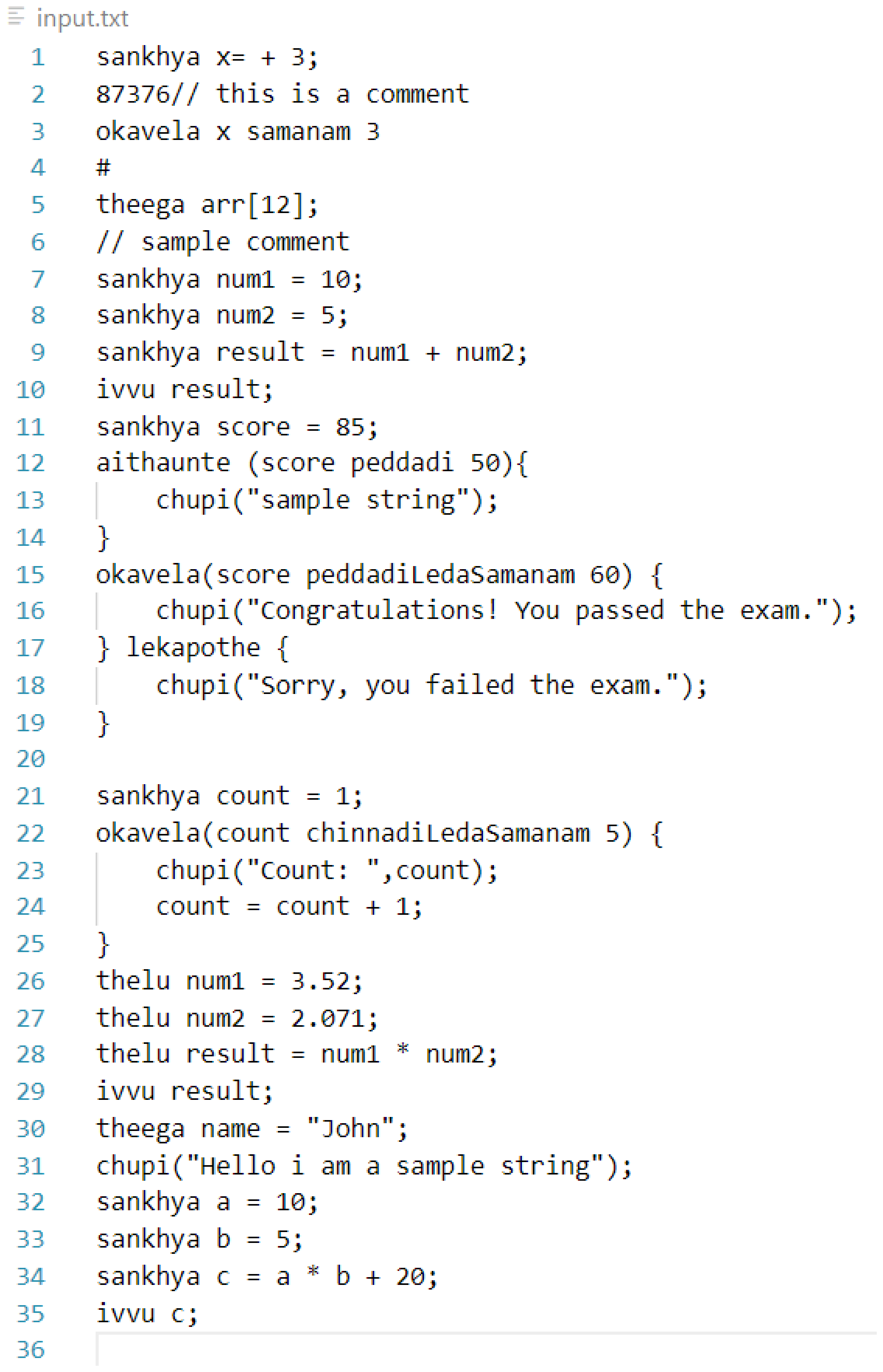
```
%{
#include<stdio.h>
#include <ctype.h> // Include the <ctype.h> header for isspace() and isdigit()
char prev='@';
int line_num = 1;
int col_num = 1;

%}

keyword "sankhya"|"thelu"|"okavela"|"lekapothe"|"chupi"|"theega"|"ivvu"|"pani"|"aithaunte"|"mariyu"|"leda"|"kaadu"
identifier [a-zA-Z_][a-zA-Z0-9_]*
digit [0-9]
int [-+]?{digit}+
float {int}"."({digit}+)
arthematic_operator ("+")|("-")|("*")|("/")|("^")
comparision_operator ("samanam")|("bhinnam")|("peddadiLedaSamanam")|("chinnadiLedaSamanam")|("chinnadi")|("peddadi")
assignment_operator "="
space [ \t]
newline [\n]
finish ";"
string \"(\\.|[^\\"])*\"
invalidIdentifier (({digit}+{identifier}))

%%

{int} {
    if(prev != '=') {
        if(yytext[0]!='+' && yytext[0]!='-'){
            printf("%s is an  integer (line: %d, column: %d)\n",yytext, line_num, col_num);
        }
        else{
            printf(" %c is arthematic operator\n%s is an integer (line: %d, column: %d)\n",yytext[0] ,yytext+1, line_num, col_num+1);
        }
    }
    else {
        printf("%s is a  integer (line: %d, column: %d)\n", yytext, line_num, col_num);
    }
    prev='@';
    col_num += yyleng;
}

{float} {
    if(prev != '=') {
        if(yytext[0]!='+' && yytext[0]!='-'){
            printf("%s is an unsigned float (line: %d, column: %d)\n",yytext, line_num, col_num);
        }
        else{
            printf("- is arthematic operator\n%s is a float (line: %d, column: %d)\n", yytext+1, line_num, col_num+1);
        }
    }
    else {
        printf("%s is a signed float (line: %d, column: %d)\n", yytext, line_num, col_num);
    }
    prev='@';
    col_num += yyleng;
}

{finish} {
    printf("%s is the end of a statement (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}

{arthematic_operator} {
    if(prev != '='){
        printf("%s is an arthematic operator (line: %d, column: %d)\n", yytext, line_num, col_num);
        prev='@';
        col_num += yyleng;
    }
    // Check if the arithematic operator is '+' or '-'
    else if(yytext[0] == '+' || yytext[0] == '-') {
        int c;
        // Skip whitespace characters
        do {
            c = input();
        } while (isspace(c));

        // If the next character is a digit, treat the token as an integer
        if (isdigit(c)) {
            // Print the arithematic operator
            printf("%c", yytext[0]);
            // Print the digits
            do {
                printf("%c", c);
                c = input();
            } while (isdigit(c));
            // Put back the non-digit character
            unput(c);
            printf(" is an integer (line: %d, column: %d)\n", line_num, col_num);
            prev = '@';
            continue;
        } else {
            // If the next character is not a digit, treat the token as an arithematic operator
            unput(c); // Put back the non-digit character
            printf("%s is an arithematic operator (line: %d, column: %d)\n", yytext, line_num, col_num);
            prev = '@';
            col_num += yyleng;
        }
    } else {
        // If the arithematic operator is not '+' or '-', treat it as an arithematic operator
        printf("%s is an arithematic operator (line: %d, column: %d)\n", yytext, line_num, col_num);
        prev = '@';
        col_num += yyleng;
    }
}

{comparision_operator} {
    printf("%s is a comparison operator (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
```

```
}

{assignment_operator} {
    printf("%s is an assignment operator (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='=';
    col_num += yyleng;
}

{keyword} {
    printf("%s is a keyword (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}

{space} {
    col_num += yyleng;
}

{newline} {
    printf("%s is a new line (line: %d)\n", yytext, line_num);
    line_num++;
    col_num = 1;
}

{identifier} {
    printf("%s is an identifier (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='i';
    col_num += yyleng;
}

{invalidIdentifier} {
     printf("%s is an invalid identifier (line: %d, column: %d)\n", yytext, line_num, col_num);
     prev='@';
     col_num += yyleng;
}

```

```
{string} {
    printf("%s is a string (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
"(" {
    printf("%s is an open curly bracket (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
")" {
    printf("%s is an closed curly bracket (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
"[" {
    printf("%s is an open square bracket (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
"]" {
    printf("%s is an closed square bracket (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
"{" {
    printf("%s is an open flower bracket (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
"}" {
    printf("%s is an closed flower bracket (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
"," {
    printf("%s is a punctuation comma (line: %d, column: %d)\n", yytext, line num, col num);
```

```
        prev='@';
        col_num += yyleng;
}

"//"        {
                while (1) {
                    int c = input();
                    if (c == '\n' || c == EOF) {
                        unput(c);
                        break;
                    }
                }
            }

. {
    printf("\nLEXER ERROR:%s is unknown symbol to me (line: %d, column: %d)\n\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}


%%

int main(int argc, char *argv[]) {
    FILE *file;
    if (argc < 2) {
        printf("Usage: %s filename\n", argv[0]);
        return 1;
    }
    file = fopen(argv[1], "r");
    if (!file) {
        perror("Error opening file");
        return 1;
    }
    yyin = file;  // Set yyin to point to the file stream

    yylex();

    fclose(file);
    return 0;
}
int yywrap(){
    return 1;
}
```

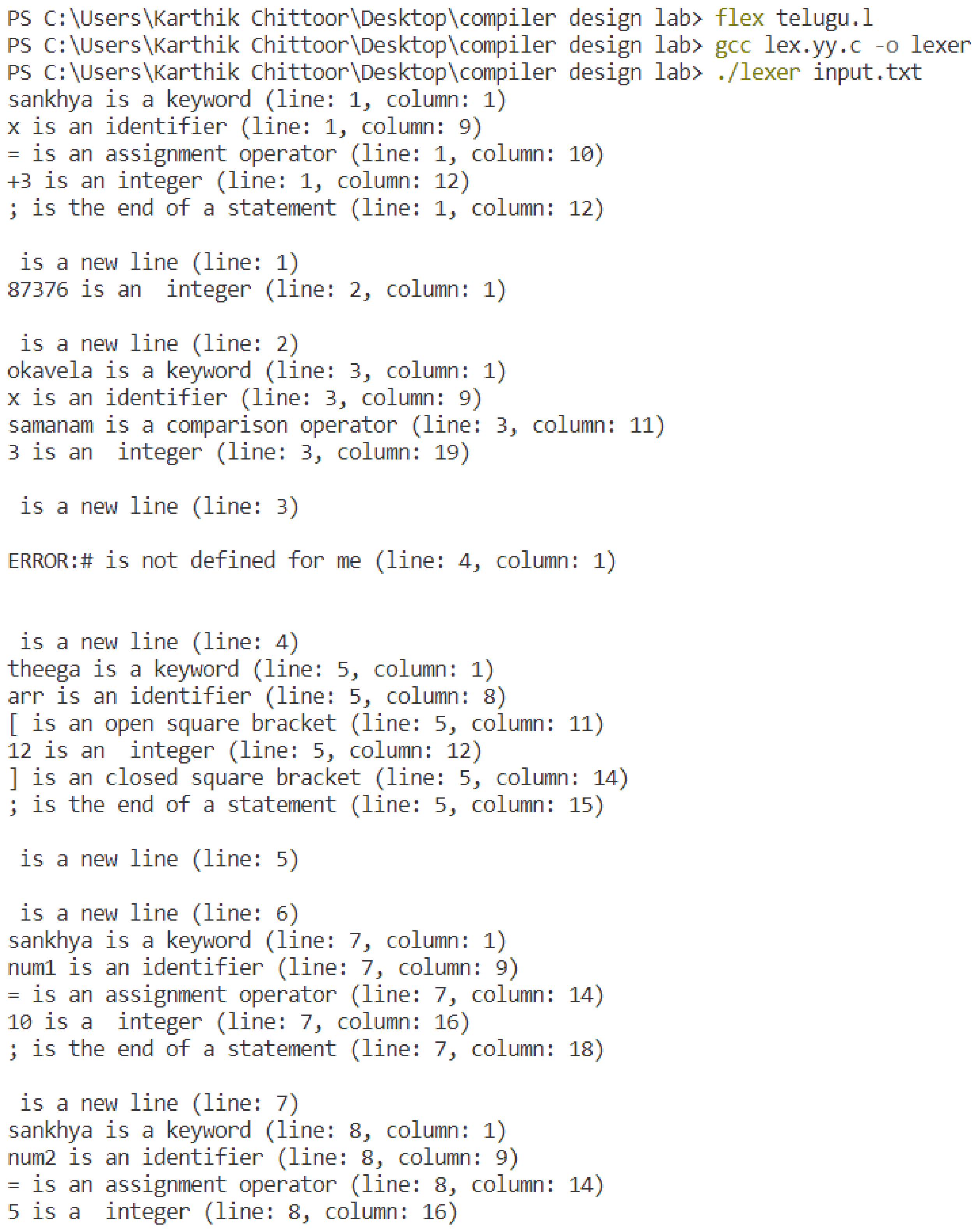
INPUT:



input.txt

```
sankhya x= + 3;
87376// this is a comment
okavela x samanam 3
#
theega arr[12];
// sample comment
sankhya num1 = 10;
sankhya num2 = 5;
sankhya result = num1 + num2;
ivvu result;
sankhya score = 85;
aithaunte (score peddadi 50){
    chupi("sample string");
}
okavela(score peddadiLedaSamanam 60) {
    chupi("Congratulations! You passed the exam.");
} lekapothe {
    chupi("Sorry, you failed the exam.");
}

sankhya count = 1;
okavela(count chinnadiLedaSamanam 5) {
    chupi("Count: ",count);
    count = count + 1;
}
thelu num1 = 3.52;
thelu num2 = 2.071;
thelu result = num1 * num2;
ivvu result;
theega name = "John";
chupi("Hello i am a sample string");
sankhya a = 10;
sankhya b = 5;
sankhya c = a * b + 20;
ivvu c;

```

OUTPUT:



```
PS C:\Users\Karthik Chittoor\Desktop\compiler design lab> flex telugu.l
PS C:\Users\Karthik Chittoor\Desktop\compiler design lab> gcc lex.yy.c -o lexer
PS C:\Users\Karthik Chittoor\Desktop\compiler design lab> ./lexer input.txt
sankhya is a keyword (line: 1, column: 1)
x is an identifier (line: 1, column: 9)
= is an assignment operator (line: 1, column: 10)
+3 is an integer (line: 1, column: 12)
; is the end of a statement (line: 1, column: 12)

 is a new line (line: 1)
87376 is an  integer (line: 2, column: 1)

 is a new line (line: 2)
okavela is a keyword (line: 3, column: 1)
x is an identifier (line: 3, column: 9)
samanam is a comparison operator (line: 3, column: 11)
3 is an  integer (line: 3, column: 19)

 is a new line (line: 3)

ERROR:# is not defined for me (line: 4, column: 1)


 is a new line (line: 4)
theega is a keyword (line: 5, column: 1)
arr is an identifier (line: 5, column: 8)
[ is an open square bracket (line: 5, column: 11)
12 is an  integer (line: 5, column: 12)
] is an closed square bracket (line: 5, column: 14)
; is the end of a statement (line: 5, column: 15)

 is a new line (line: 5)

 is a new line (line: 6)
sankhya is a keyword (line: 7, column: 1)
num1 is an identifier (line: 7, column: 9)
= is an assignment operator (line: 7, column: 14)
10 is a  integer (line: 7, column: 16)
; is the end of a statement (line: 7, column: 18)

 is a new line (line: 7)
sankhya is a keyword (line: 8, column: 1)
num2 is an identifier (line: 8, column: 9)
= is an assignment operator (line: 8, column: 14)
5 is a  integer (line: 8, column: 16)
```

```
; is the end of a statement (line: 8, column: 17)

 is a new line (line: 8)
sankhya is a keyword (line: 9, column: 1)
result is an identifier (line: 9, column: 9)
= is an assignment operator (line: 9, column: 16)
num1 is an identifier (line: 9, column: 18)
+ is an arthematic operator (line: 9, column: 23)
num2 is an identifier (line: 9, column: 25)
; is the end of a statement (line: 9, column: 29)

 is a new line (line: 9)
ivvu is a keyword (line: 10, column: 1)
result is an identifier (line: 10, column: 6)
; is the end of a statement (line: 10, column: 12)

 is a new line (line: 10)
sankhya is a keyword (line: 11, column: 1)
score is an identifier (line: 11, column: 9)
= is an assignment operator (line: 11, column: 15)
85 is a  integer (line: 11, column: 17)
; is the end of a statement (line: 11, column: 19)

 is a new line (line: 11)
aithaunte is a keyword (line: 12, column: 1)
( is an open curly bracket (line: 12, column: 11)
score is an identifier (line: 12, column: 12)
peddadi is a comparison operator (line: 12, column: 18)
50 is an  integer (line: 12, column: 26)
) is an closed curly bracket (line: 12, column: 28)
{ is an open flower bracket (line: 12, column: 29)

 is a new line (line: 12)
chupi is a keyword (line: 13, column: 5)
( is an open curly bracket (line: 13, column: 10)
"sample string" is a string (line: 13, column: 11)
) is an closed curly bracket (line: 13, column: 26)
; is the end of a statement (line: 13, column: 27)

 is a new line (line: 13)
} is an closed flower bracket (line: 14, column: 1)

 is a new line (line: 14)
```

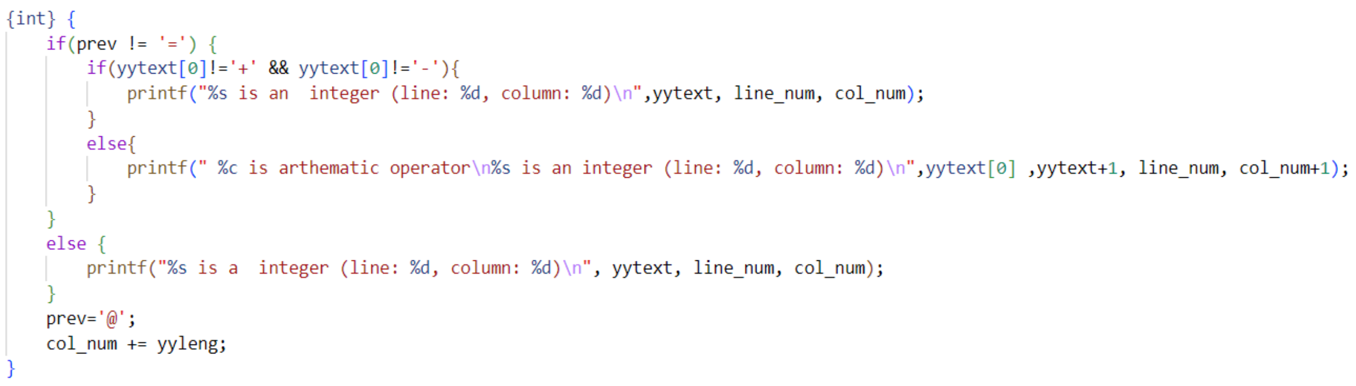
```
okavela is a keyword (line: 15, column: 1)
( is an open curly bracket (line: 15, column: 8)
score is an identifier (line: 15, column: 9)
peddadiLedaSamanam is a comparison operator (line: 15, column: 15)
60 is an  integer (line: 15, column: 34)
) is an closed curly bracket (line: 15, column: 36)
{ is an open flower bracket (line: 15, column: 38)

 is a new line (line: 15)
chupi is a keyword (line: 16, column: 5)
( is an open curly bracket (line: 16, column: 10)
"Congratulations! You passed the exam." is a string (line: 16, column: 11)
) is an closed curly bracket (line: 16, column: 50)
; is the end of a statement (line: 16, column: 51)

 is a new line (line: 16)
} is an closed flower bracket (line: 17, column: 1)
lekapothe is a keyword (line: 17, column: 3)
{ is an open flower bracket (line: 17, column: 13)

 is a new line (line: 17)
chupi is a keyword (line: 18, column: 5)
( is an open curly bracket (line: 18, column: 10)
"Sorry, you failed the exam." is a string (line: 18, column: 11)
) is an closed curly bracket (line: 18, column: 40)
; is the end of a statement (line: 18, column: 41)

 is a new line (line: 18)
} is an closed flower bracket (line: 19, column: 1)

 is a new line (line: 19)

 is a new line (line: 20)
sankhya is a keyword (line: 21, column: 1)
count is an identifier (line: 21, column: 9)
= is an assignment operator (line: 21, column: 15)
1 is a  integer (line: 21, column: 17)
; is the end of a statement (line: 21, column: 18)
```

```
okavela is a keyword (line: 22, column: 1)
( is an open curly bracket (line: 22, column: 8)
count is an identifier (line: 22, column: 9)
chinnadiLedaSamanam is a comparison operator (line: 22, column: 15)
5 is an  integer (line: 22, column: 35)
) is an closed curly bracket (line: 22, column: 36)
{ is an open flower bracket (line: 22, column: 38)

 is a new line (line: 22)
chupi is a keyword (line: 23, column: 5)
( is an open curly bracket (line: 23, column: 10)
"Count: " is a string (line: 23, column: 11)
, is a punctuation comma (line: 23, column: 20)
count is an identifier (line: 23, column: 21)
) is an closed curly bracket (line: 23, column: 26)
; is the end of a statement (line: 23, column: 27)

 is a new line (line: 23)
count is an identifier (line: 24, column: 5)
= is an assignment operator (line: 24, column: 11)
count is an identifier (line: 24, column: 13)
+ is an arthematic operator (line: 24, column: 19)
1 is an  integer (line: 24, column: 21)
; is the end of a statement (line: 24, column: 22)

 is a new line (line: 24)
} is an closed flower bracket (line: 25, column: 1)

 is a new line (line: 25)
thelu is a keyword (line: 26, column: 1)
num1 is an identifier (line: 26, column: 7)
= is an assignment operator (line: 26, column: 12)
3.52 is a signed float (line: 26, column: 14)
; is the end of a statement (line: 26, column: 18)

 is a new line (line: 26)
thelu is a keyword (line: 27, column: 1)
num2 is an identifier (line: 27, column: 7)
= is an assignment operator (line: 27, column: 12)
2.071 is a signed float (line: 27, column: 14)
; is the end of a statement (line: 27, column: 19)

 is a new line (line: 27)
thelu is a keyword (line: 28, column: 1)
result is an identifier (line: 28, column: 7)
```

```
* is an arthematic operator (line: 28, column: 21)
num2 is an identifier (line: 28, column: 23)
; is the end of a statement (line: 28, column: 27)

 is a new line (line: 28)
ivvu is a keyword (line: 29, column: 1)
result is an identifier (line: 29, column: 6)
; is the end of a statement (line: 29, column: 12)

 is a new line (line: 29)
theega is a keyword (line: 30, column: 1)
name is an identifier (line: 30, column: 8)
= is an assignment operator (line: 30, column: 13)
"John" is a string (line: 30, column: 15)
; is the end of a statement (line: 30, column: 21)

 is a new line (line: 30)
chupi is a keyword (line: 31, column: 1)
( is an open curly bracket (line: 31, column: 6)
"Hello i am a sample string" is a string (line: 31, column: 7)
) is an closed curly bracket (line: 31, column: 35)
; is the end of a statement (line: 31, column: 36)

 is a new line (line: 31)
sankhya is a keyword (line: 32, column: 1)
a is an identifier (line: 32, column: 9)
= is an assignment operator (line: 32, column: 11)
10 is a  integer (line: 32, column: 13)
; is the end of a statement (line: 32, column: 15)

 is a new line (line: 32)
sankhya is a keyword (line: 33, column: 1)
b is an identifier (line: 33, column: 9)
= is an assignment operator (line: 33, column: 11)
5 is a  integer (line: 33, column: 13)
; is the end of a statement (line: 33, column: 14)

 is a new line (line: 33)
sankhya is a keyword (line: 34, column: 1)
15)
b is an identifier (line: 34, column: 17)
+ is an arthematic operator (line: 34, column: 19)
20 is an  integer (line: 34, column: 21)
; is the end of a statement (line: 34, column: 23)

  is a new line (line: 34)
 ivvu is a keyword (line: 35, column: 1)
 c is an identifier (line: 35, column: 6)
 ; is the end of a statement (line: 35, column: 7)

  is a new line (line: 35)
```

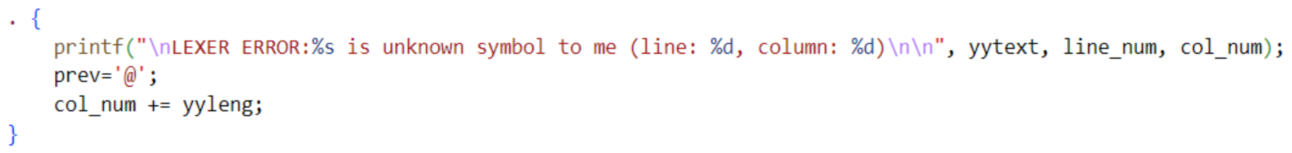
D.Error/Ambiguity Handling:

sankhya a=-3;   // here -3 should be read as a signed integer
a=a-3;   // here '-' should be read as binary arthematic operator and 3 as unsigned integer seperately



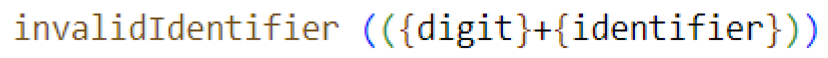
```
{int} {
    if(prev != '=') {
        if(yytext[0]!='+' && yytext[0]!='-'){
            printf("%s is an  integer (line: %d, column: %d)\n",yytext, line_num, col_num);
        }
        else{
            printf(" %c is arthematic operator\n%s is an integer (line: %d, column: %d)\n",yytext[0] ,yytext+1, line_num, col_num+1);
        }
    }
    else {
        printf("%s is a  integer (line: %d, column: %d)\n", yytext, line_num, col_num);
    }
    prev='@';
    col_num += yyleng;
}
```

Here "prev" stores '=' if it lexer saw an assignment operator. Other wise it will store a dummy symbol '@'. Based on prev as a **look-behind,** we differentiate if a + or – is unary/binary operator at that statement.

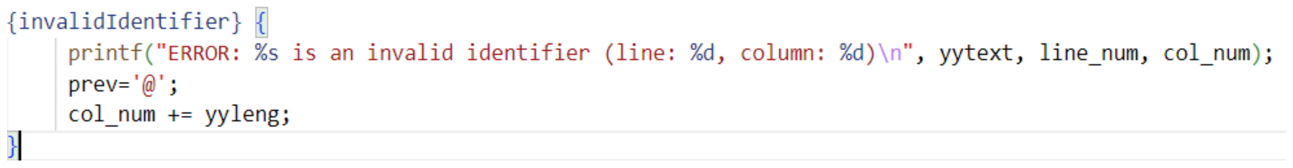


```
. {
    printf("\nLEXER ERROR:%s is unknown symbol to me (line: %d, column: %d)\n\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
```

Any unknown symbol(not defined) the lexer comes across is shown as a lexer error.



```
invalidIdentifier (({digit}+{identifier}))
```

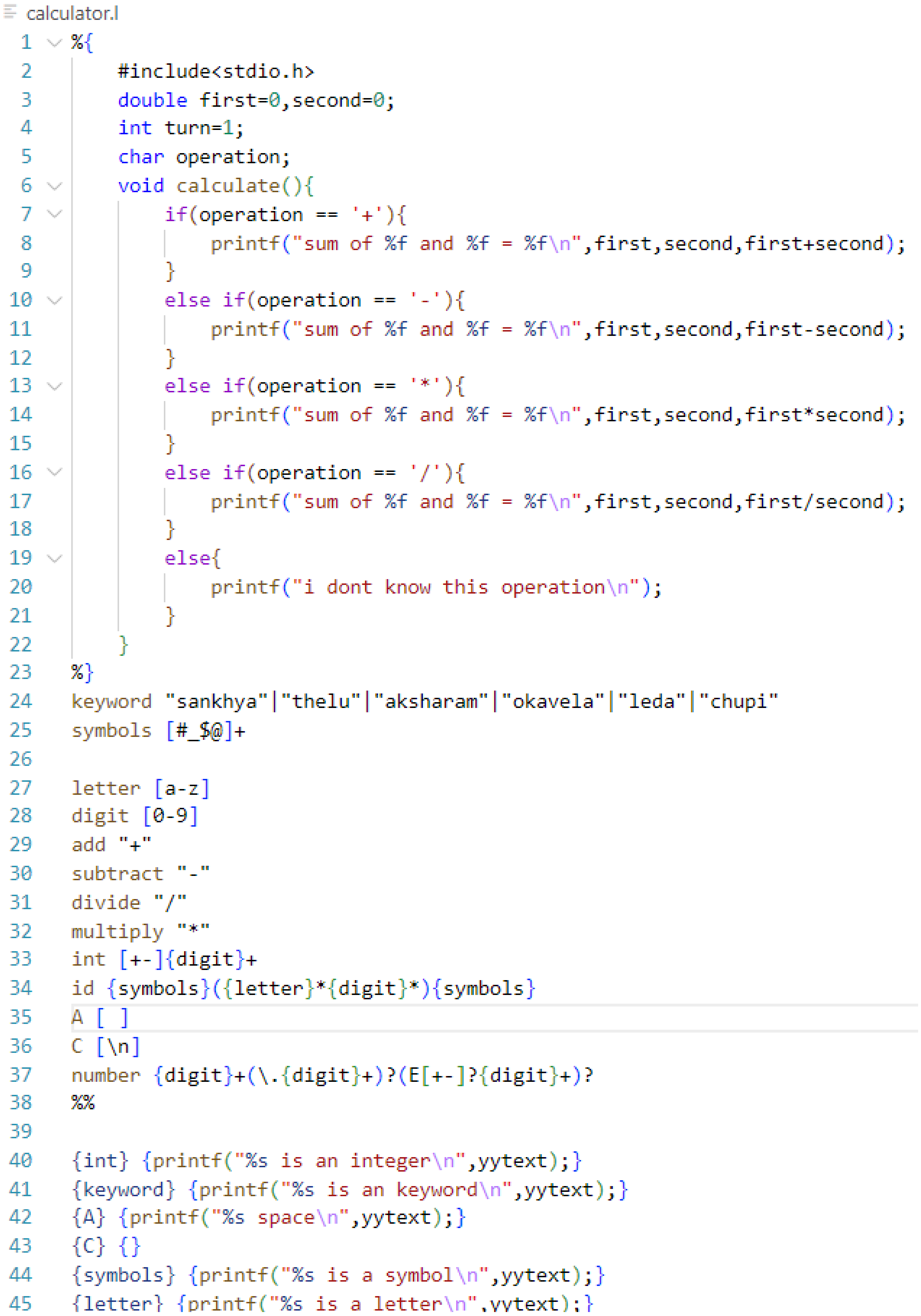


```
{invalidIdentifier} {
    printf("ERROR: %s is an invalid identifier (line: %d, column: %d)\n", yytext, line_num, col_num);
    prev='@';
    col_num += yyleng;
}
```

Identifiers that start with digits are shown as invalid Identifier error.
(note that we could simple shown it as error instead of categorizing it as a separate token, but this is for more information to the coder.)

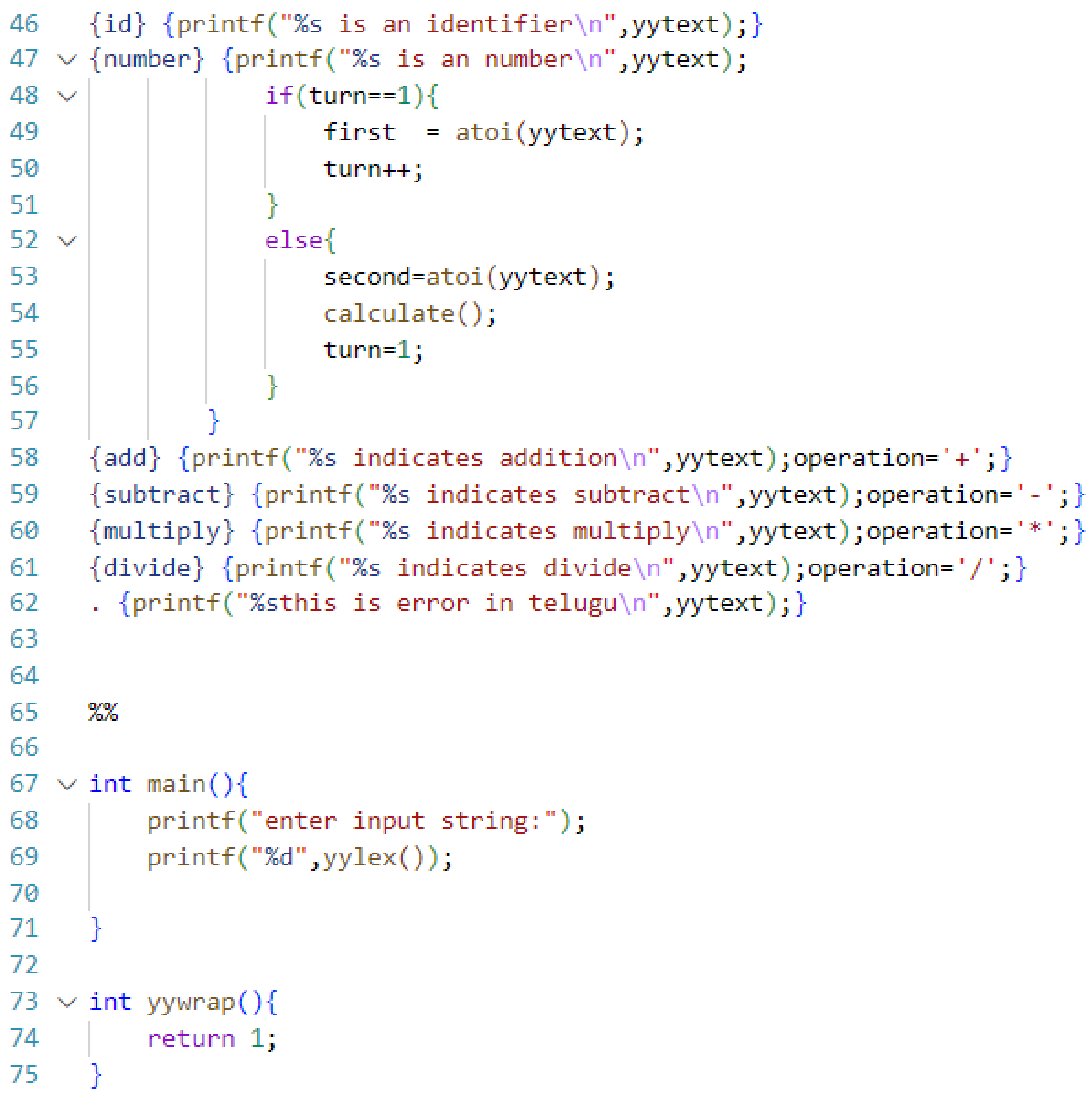
F. Small Calculator program in Telugu

Code:



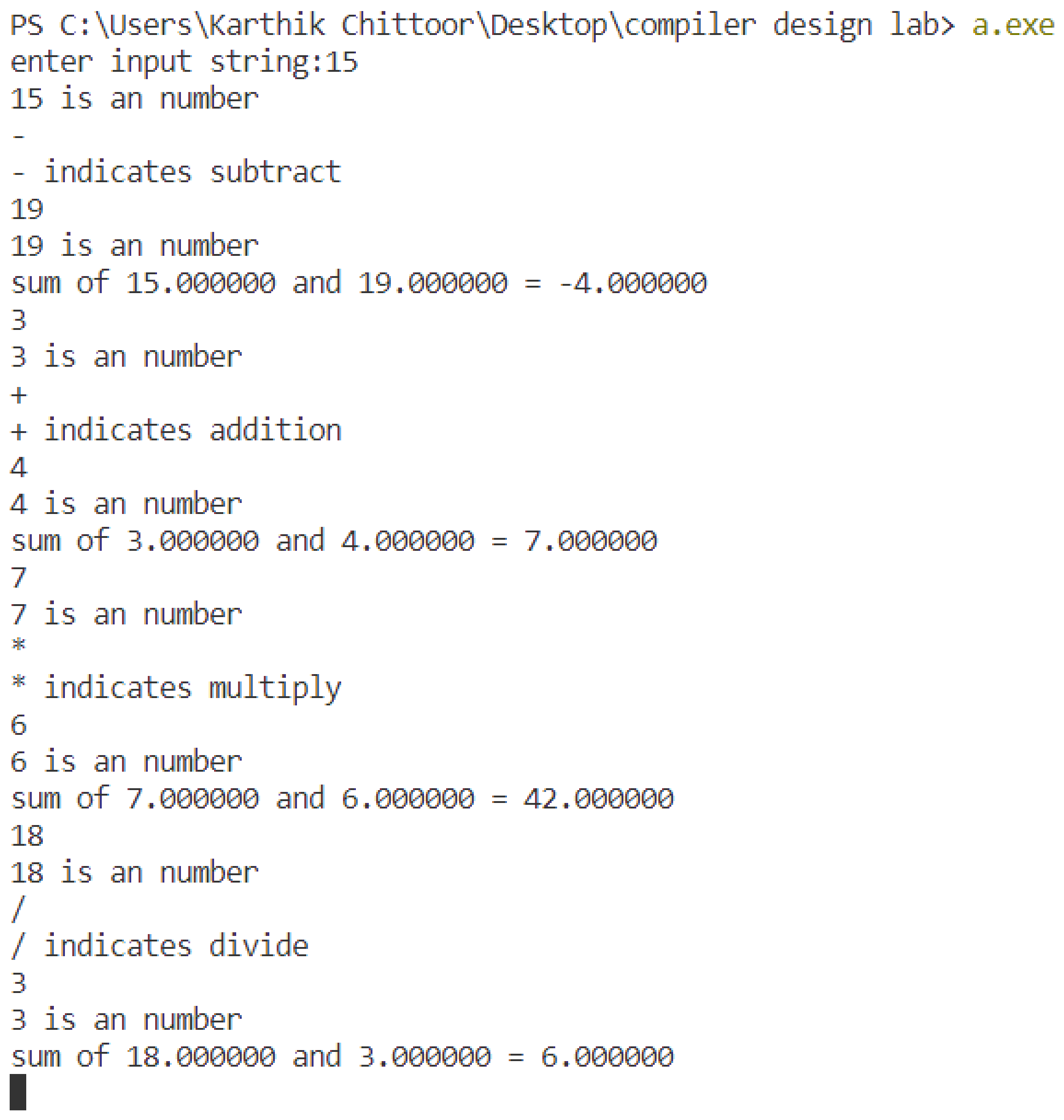
calculator.l

```
%{
    #include<stdio.h>
    double first=0,second=0;
    int turn=1;
    char operation;
    void calculate(){
        if(operation == '+'){
            printf("sum of %f and %f = %f\n",first,second,first+second);
        }
        else if(operation == '-'){
            printf("sum of %f and %f = %f\n",first,second,first-second);
        }
        else if(operation == '*'){
            printf("sum of %f and %f = %f\n",first,second,first*second);
        }
        else if(operation == '/'){
            printf("sum of %f and %f = %f\n",first,second,first/second);
        }
        else{
            printf("i dont know this operation\n");
        }
    }
%}
keyword "sankhya"|"thelu"|"aksharam"|"okavela"|"leda"|"chupi"
symbols [#_$@]+

letter [a-z]
digit [0-9]
add "+"
subtract "-"
divide "/"
multiply "*"
int [+-]{digit}+
id {symbols}({letter}*{digit}*){symbols}
A [ ]
C [\n]
number {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
%%

{int} {printf("%s is an integer\n",yytext);}
{keyword} {printf("%s is an keyword\n",yytext);}
{A} {printf("%s space\n",yytext);}
{C} {}
{symbols} {printf("%s is a symbol\n",yytext);}
{letter} {printf("%s is a letter\n",yytext);}
```



```
{id} {printf("%s is an identifier\n",yytext);}
{number} {printf("%s is an number\n",yytext);
            if(turn==1){
                first  = atoi(yytext);
                turn++;
            }
            else{
                second=atoi(yytext);
                calculate();
                turn=1;
            }
        }
{add} {printf("%s indicates addition\n",yytext);operation='+';}
{subtract} {printf("%s indicates subtract\n",yytext);operation='-';}
{multiply} {printf("%s indicates multiply\n",yytext);operation='*';}
{divide} {printf("%s indicates divide\n",yytext);operation='/';}
. {printf("%sthis is error in telugu\n",yytext);}


%%

int main(){
    printf("enter input string:");
    printf("%d",yylex());

}

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

Ouput:



```
PS C:\Users\Karthik Chittoor\Desktop\compiler design lab> a.exe
enter input string:15
15 is an number
-
- indicates subtract
19
19 is an number
sum of 15.000000 and 19.000000 = -4.000000
3
3 is an number
+
+ indicates addition
4
4 is an number
sum of 3.000000 and 4.000000 = 7.000000
7
7 is an number
*
* indicates multiply
6
6 is an number
sum of 7.000000 and 6.000000 = 42.000000
18
18 is an number
/
/ indicates divide
3
3 is an number
sum of 18.000000 and 3.000000 = 6.000000
```

Result:

The Lexical Analyser is able to generate tokens correctly based on the regular expressions defined.

1.One line comments are skipped

2.Unknown symbols are shown as error.

3.Identifiers starting with a number are shown as invalid identifiers-error().

Conclusion:

The lexical analyser (the first stage of compilation) has been successfully implemented in LEX.