VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

IMRAN WADRALI(1BM21CS077)

Under the Guidance of
Prof. Prameetha Pai
Assistant Professor, BMSCE

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

November 2023-February 2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum) Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Compiler Design**" carried out by **IMRAN WADRALI(1BM21CS077)** , who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24.

The Lab report has been approved as it satisfies the academic requirements in respect of **Compiler Design- (22CS5PCCPD)** work prescribed for the said degree.

Prof. Prameetha Pai	Dr. Jyothi Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Imran Wadrali (1BM21CS077), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Prof. Sunayana S, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

TABLE OF CONTENTS

INDEX

Name Imran Wadiali

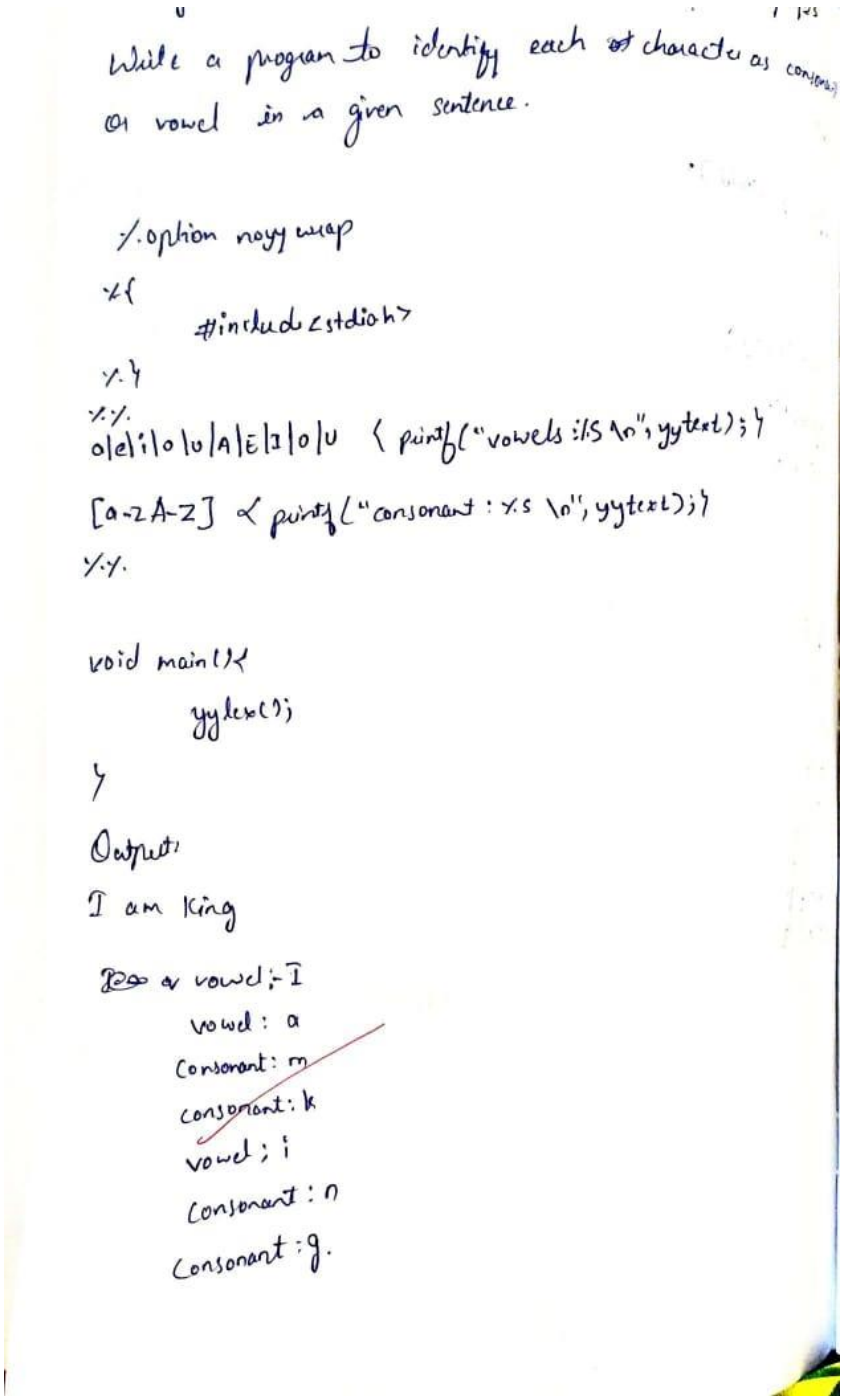
Standard 5 Section B Roll No. 077

Subject Compiler Design Lab.

Sl.No.	Date	Title	Page No.	Teacher's Sign
			9/15	
1]	20/11/23	Program to count the no. of words and consonants (lex)		
2]	20/11/23	Identify tokens, keywords & Separator		
3]	27/11/23	Program to demostriate floting point number		
4]	4/12/23	Program- Replacing Sequence of non-empty space with single space		
5]	11/12/23	Program to recognize tokens over Alphabets Σ 0... 9		
6]	18/12/23	C-Program to design lexical Analyser		
7]	11/1/24	Program Recursin Descent		
8]	11/1/24	Program to demostrate Desk calculator		
9]	11/1/24	Program to demostrate String Parser		
10]	25/1/24	Program to show syntax tree -generator		

S No.	Date	Title	Page No	Teacher's Sign
11	29/1/24	Program to show infix to postfix. [YACC]		
12	29/1/24	Three address code generator [using YACC]		

PROGRAM-1



Write a program to identify each character as consonant or vowel in a given sentence.

```
%.option noyywrap
%{
    #include<stdio.h>
%.}
%%.
a|e|i|o|u|A|E|I|O|U  { printf("vowels :%s \n", yytext); }
[a-zA-Z]  { printf("consonant : %s \n", yytext); }
%%.

void main(){
    yylex();
}
```

Output:

I am king

vowel : I
vowel : a
Consonant : m
consonant : k
vowel : i
Consonant : n
Consonant : g.

OUTPUT SCREENSHOT:

```
Enter a sentence:
Compiler design
 No of vowels and consonants are 5 and 9
This is a book
   No of vowels and consonants are 5 and 6
^C
```

PROGRAM-2

Write a lex program to read the following input from a file and print the valid token on the terminal.

Note : create two files one as Pg.l and other one input.txt

```
%%
int|float|char { printf("Datatype -> %s\n",yytext);}
[a-zA-Z]* { printf("Variable -> %s\n",yytext);}
=  { printf("assignment operator -> %s\n",yytext);}
[(|)|;] { printf("separator -> %s\n",yytext);}
%%

void main(){
  printf("Enter the file name: \n");
  scanf("%s",fname)
  yyin = fopen(fname,"r");
  yylex();
  fclose(yyin);
}
```

Output: Enter the file name : input.txt

Datatype -> int

Variable -> a

assignment operator -> =

digit -> 5

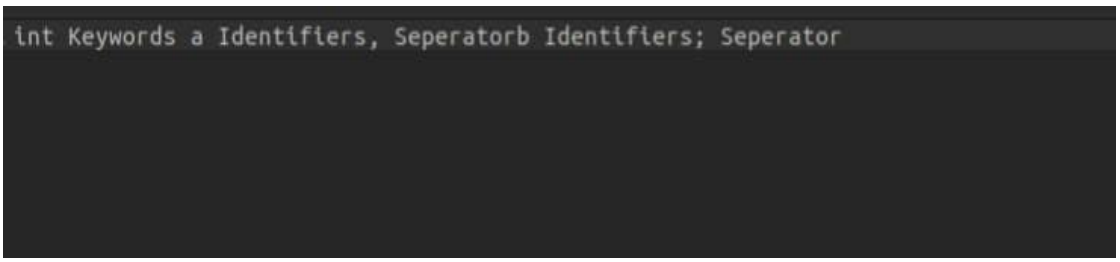
Separator ->)

Variable -> sum

separator -> ;

20/11/23

OUTPUT SCREENSHOT:



PROGRAM-3:

Write a lex program to recognize floating Point Numbers. Check for all the input cases.

```
%{
    #include <stdio.h>
%}
%%
^[+-]?[0-9]*[.][0-9]+ { printf("%s ==> floating point number", yytext); }
^[+-]?[0-9]* { printf("%s ==> Not a floating point number", yytext); }
%%
int yywrap()
{
}
void main()
{
    printf("Enter a number : \n");
    yylex();
}
```

Output: 0.33

0.33 ==> floating point numbe.

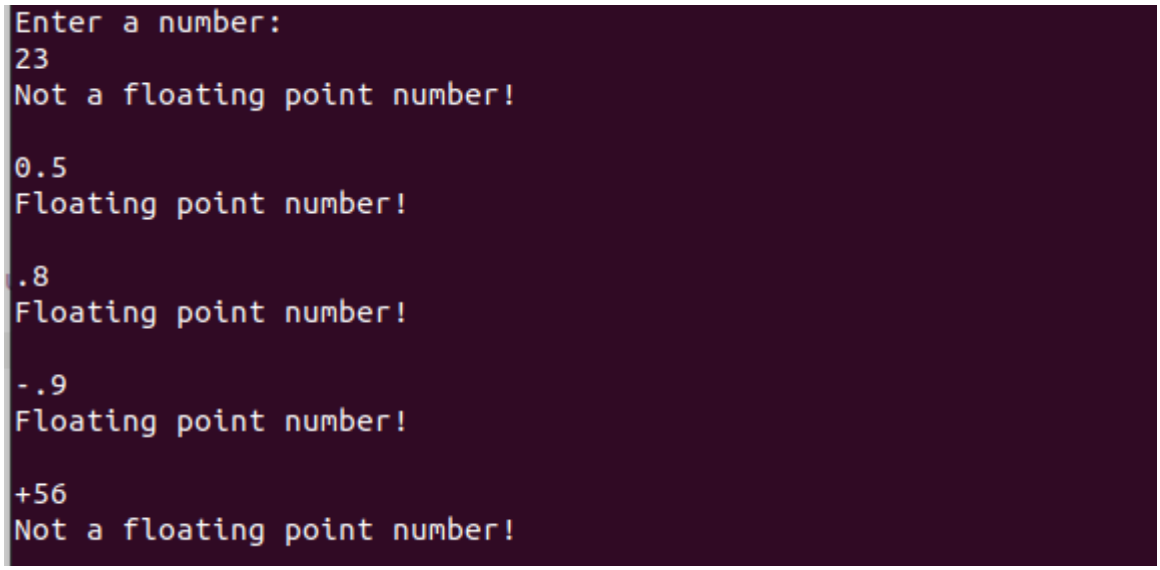
-334

-334 ==> Not a floating point numbe.

+6.3

+6.3 ==> floating point number.

OUTPUT SCREENSHOT:



PROGRAM-4

Write a lex Program that copies a file, replacing each non-empty sequence of white space by a single blank.

Code:

```
%{
    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>
    char str1[200];
%}
%%
[\n]    { fprintf(yyout, "%s\n", str1); str1[0] = '\0'; }
[ ]*    { fprintf(yyout, "%s", str1); str1[0] = '\0';
          fprintf(yyout, "%s", " ");
          strcat(str1, yytext);
<<EOF>> { printf(yyout, "%s", str1); return 0; }
%%
int main()
{
    char filename[100];
    printf("Enter the name of the file to copy: \t");
    scanf("%s", filename);
    yyin = fopen(filename, "r");
    if (yyin == NULL) {
        exit(0);
    }
```

Sen 11/12/23

```
printf("Enter the name of the file to write : \t");
scanf("%s", filename);
yyout = fopen(filename, "w");
if (yyout == NULL)
{
    exit(1);
}
yylex();
}

int yywrap(void)
{
}
```

Output:

Enter the name of file to copy: inpuspace
Enter the name of the file to open for writing: outspace

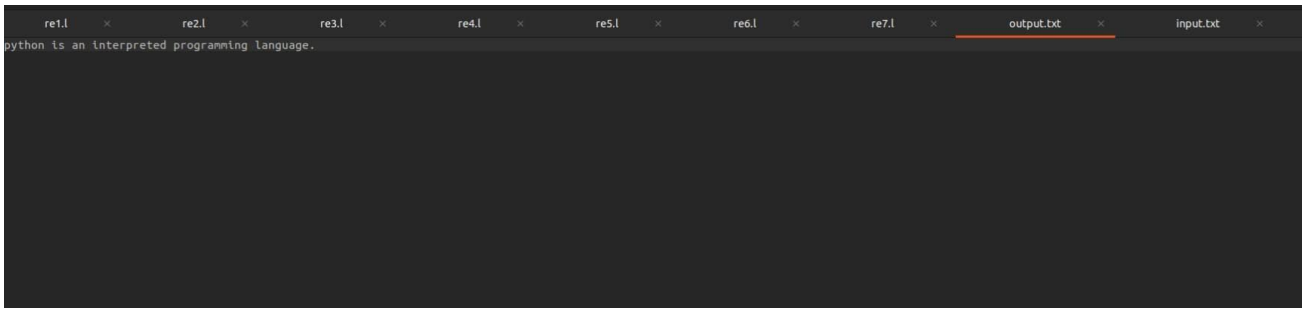
Inpuspace.txt

hii I am Imran.

Outspace.txt

hii I am Imran.

OUTPUT SCREENSHOT:



PROGRAM-5

Program 13:

Write a Lex program to recognize the following tokens over the alphabets {0, 1, ..., 9}

a) The set of all string ending in 00.

Code:

```
%{
    #include <stdio.h>
%}

%%
[0-9]*00  { printf("string accepted"); }
[0-9]*    { printf("string rejected"); }
%%

int main(){
  yylex()
  return 0;
}

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

Output:

```
001
String rejected
100
String accepted

000
String accepted.
```

b] The set of all strings with three consecutive 222's.

Code

```
%{
    #include<stdio.h>
%}

%%
[0-9]*222[0-9]* { printf("string accepted"); }
[0-9]* { printf("string rejected"); }
%%

int main(){
    yylex();
    return 0;
}

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

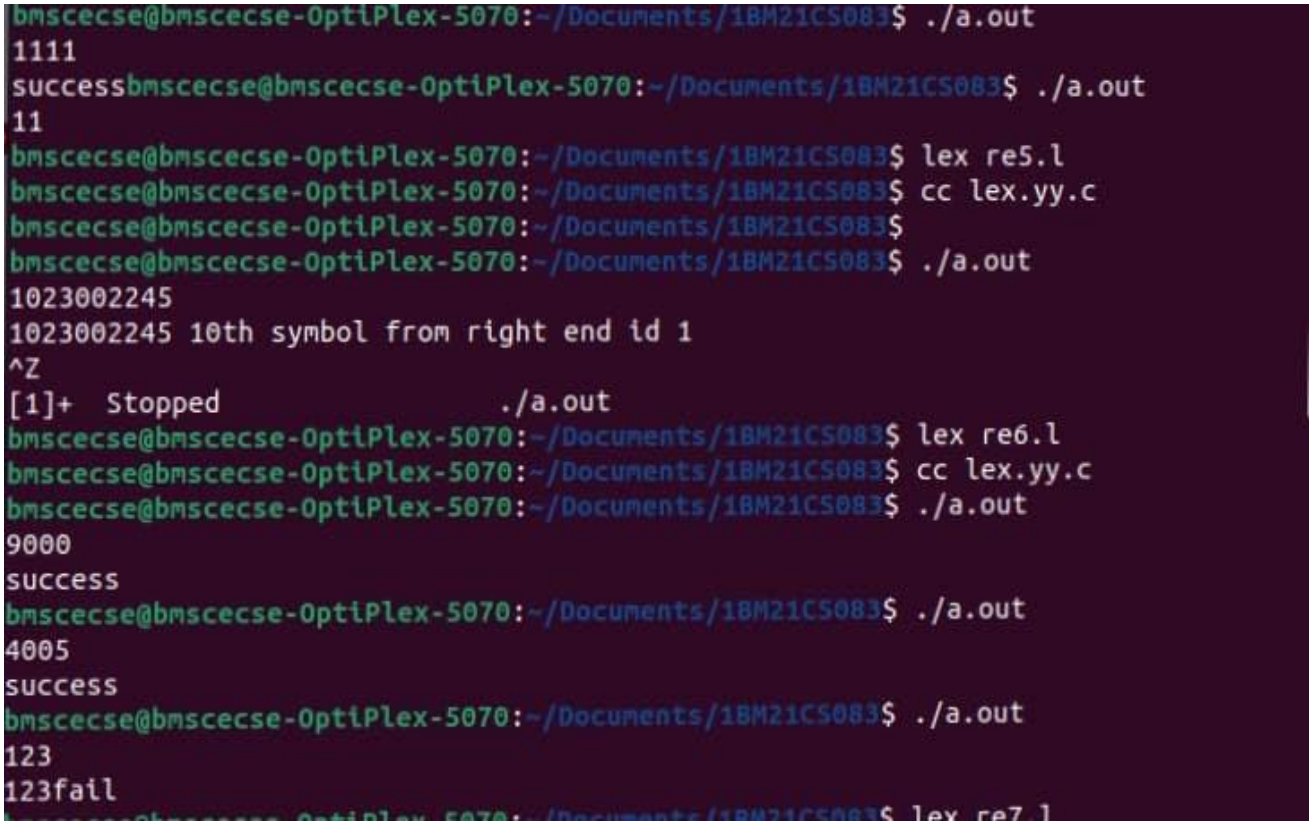
Output:

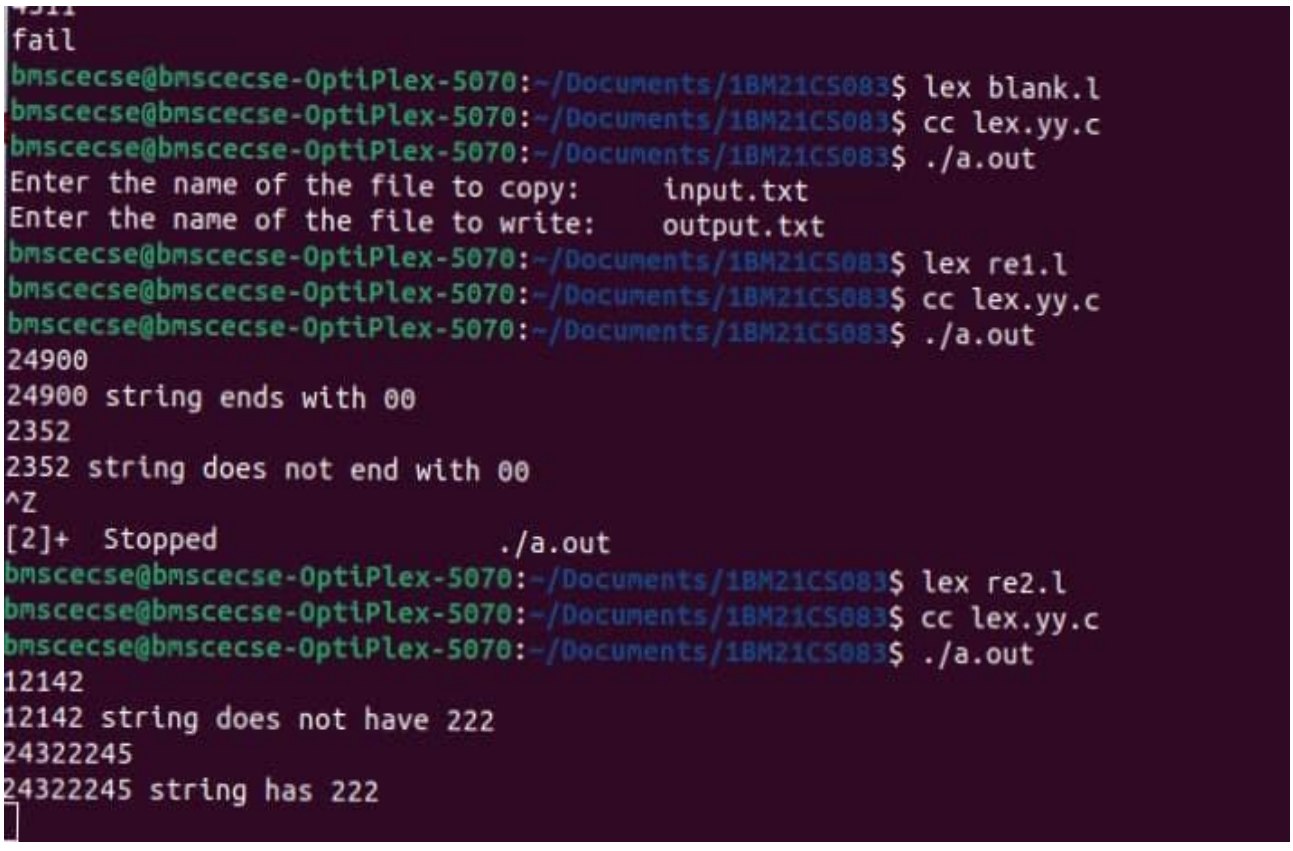
1234
String rejected

01222567
String accepted.

4/12/2023

OUTPUT SCREENSHOT:





PROGRAM-6

18/12/23

Write a C program to a lexical analyzer to recognise any five Keywords, identifiers, numbers, operators and punctuations.

=> Code:

```
#include<stdio.h>
#include <ctype.h>
# include <string.h>

int isKeyword(char *word ){
    char *keywords[] = {"int", "float","char","if","else"};
    for (int i=0 ;i<5; ++i){
        if( strcmp (keywords[i], word ) == 0){
            return 1;
        }
    }
    return 0;
}
int isOperator(char ch){
    char operators[] = "+-*/%=><&|^";
    return strchr (operators, ch) !=NULL;
}
int isPunctuation (char ch){
    char punctuation[]=",;.!";
    return strchr(punctuation,ch) != NULL;
}
```

```
void classify (char *str){
    for(int i=0; str[i]!='\0' ;++i){
        if (isalpha(str[i]) || str[i]=='_'){
            int j=i+1;
            while (isalnum(str[j]) || str[j]=='_'){
                j++;
            }
            char temp[j-i+1];
            strncpy(temp,&str[i],j-i);
            temp[j-i]='\0';
            if(isKeyword(temp)){
                Printf("Keyword : %s\n",temp);
            }
            else{
                printf("Identifier: %s\n", temp);
            }
            i=j-1;
    }
    else if(isdigit(str[i])){
        int j=i+1;
        while(isdigit(str[j])){
            j++;
        }
        printf("Number: %.*s\n",j-i, &str[i]);
        i= j-1;
    }
    else if(isOperator(str[i])){
        printf("Operator: %c\n", str[i]);
    }
}
```

```
        else if (isPunctuation(str[i])) {
            printf("Punctuation: %c\n", str[i]);
        }
    }
}

int main() {
    char input[100];
    printf("Enter input string: ");
    fgets(input, sizeof(input), stdin);
    printf("\n classifying input: \n");
    classify(input);

    return 0;
}
```

18/10/23

Output:

input String: int a = 0; b; float sum = 0.0;

classifying the input:

Keyword : int
Identifier : a
Operator : =
Number = 0
Punctuation : ;
Identifier : b
Punctuation : ;
Keyword : float
Identifier : sum
Operator : =
Number = 0
punctuation : .
Number : 0
Punctuation : ;

OUTPUT SCREENSHOT:

```
enter c code
int a = 1234 ;
Keyword: int
Identifier: a
Punctuation/Operator: =
Number: 1234
Punctuation/Operator: ;
```

PROGRAM-7

8/1/2024

Write a program to perform Recursive Descent Parsing on the following grammar.

$S \rightarrow cAd$

$A \rightarrow ab|a$

⇒ Code

```
#include <stdio.h>
#include <stdbool.h>

bool parse_S(char input_str[]);
bool parse_A(char input_str[]);
bool recursive_descent_parser.(char input_str[]);

int index;

bool parse_S( char input_str[]){
    if (input_str[index]=='c'){
        index++;
        if (parse_A(input_str) && input_str[index]=='d'){
            index++;
            return true;
        }
        else{
            return false;
        }
    }
    else{
        return false;
    }
}
```

```
if (recursive_descent_parser(user_input)){
    printf("The given string is accepted by the grammar.\n");
}
else{
    printf("The given string is not accepted by the
        grammar.\n");
}
}
```

Output:

Enter the string to parse: cad

S → cAd
A → ab|a

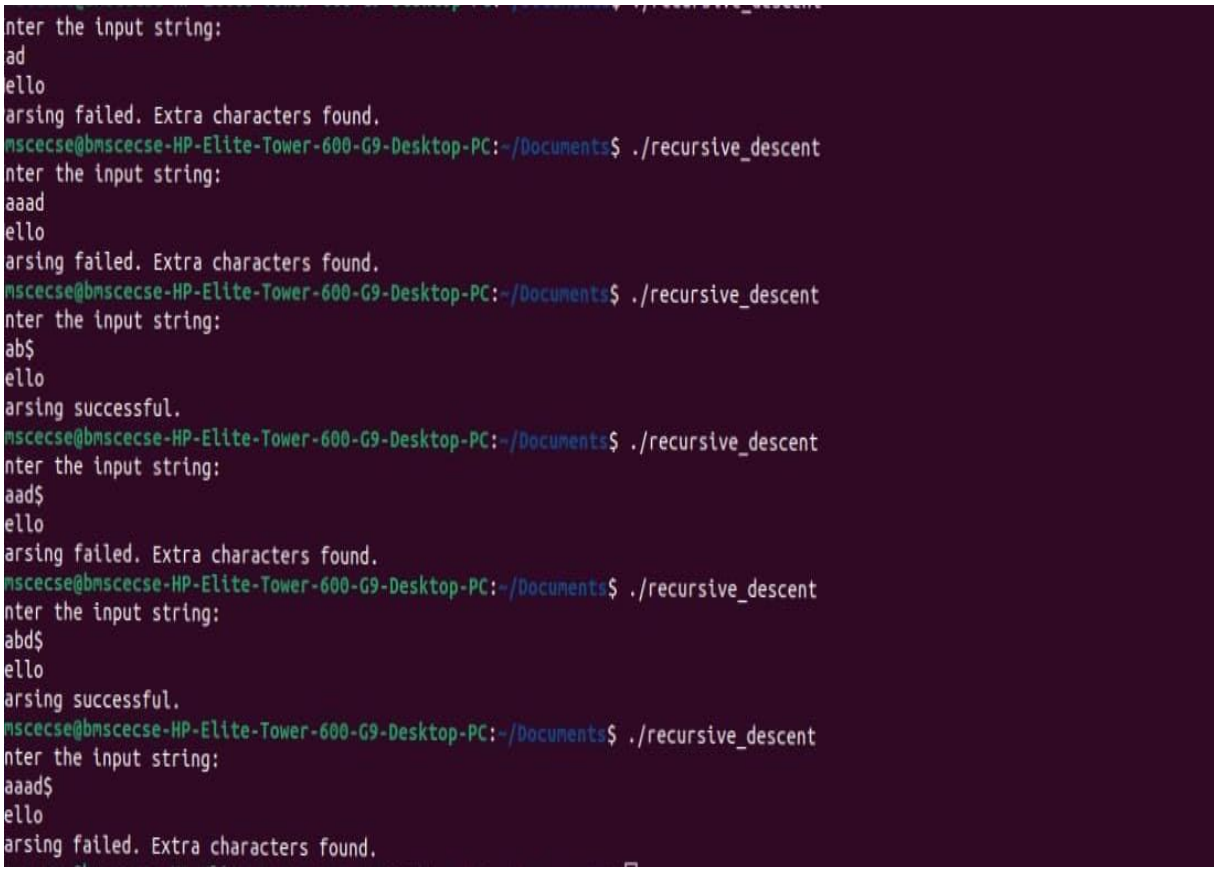
The given string is accepted by the grammar

Enter the string to parse: cadd

S → cAdd
A → ab|a

The given string is not accepted by the grammar.

OUTPUT SCREENSHOT:



PROGRAM-8

8/1/24

Design a suitable grammar for evaluation of Arithmetic expression having + & - operators.

+ has least priority & is left associative

- has high priority and is right associative

Code:

lex file:

```
%option noyywrap
%{
  #include "y.tab.h"
%}
%%
[0-9]+  { yylval = atoi(yytext); return NUM; }
[\t]    ;
\n      return 0;
.       return yytext[0];
%%
```

yacc file:

```
%{
    #include<stdio.h>
%}
%token NUM
%left '+'
%right '-'
%%
expr: e { printf("Valid expression\n"); printf("Result: %d\n", $$);
          return 0; }

e: e'+'e    { $$ = $1 + $3; }
 | e'-'e    { $$ = $1 - $3; }
 | NUM      { $$ = $1; }
 ;
%%
int main() {
  printf("\n Enter an arithmetic expression\n");
  yyparse();
 return 0;
}

int yyerror() {
   printf("\n Invalid expression\n");
   return 0;
}
```

Run
8/1/24

Output : Enter an arithmetic Expression

5+10+10

Valid Expression

Result = 25

(

5+

Invalid Expression

;

:

[

OUTPUT SCREENSHOT:

```
Enter an arithmetic expression:
2+3*4
Valid expression!
Result:14
```

```
Enter an arithmetic expression:
2++3-
Invalid expression!
```

PROGRAM-9

Write a yacc program string grammar. $a^n b^n$. $n \geq 5$;

anbn.y

```
%{
#include<stdio.h>
#include<stdlib.h>
int yyerror(char *s);
int yylex(void);
%}

%token A
%token B
%token NL

%%
Start : AAAAA S B NL { printf("Parsed using the
        rule (a^n)b, n>=5\n valid string.\n"); }
;
S : S A
|
;
%%
void main()
{
  printf("Enter a string!\n");
  yyparse();
}
```

```
int yyerror(char *s)
{
    printf("Invalid String !\n");
    return 0;
}
```

anbn.l

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern int yylval
%}
%%
[aA]  { yylval = yytext[0]; return A; }
[bB]  { yylval = yytext[0]; return B; }
\n    { return NL; }
.     { return yytext[0]; }
%%
int yywrap()
{
    return 1;
}
```

Output

Enter a string

aaaaab

Parsed using rule $(a^n b)$, $n >= 5$

valid string

aaabb

invalid string!

29/1/2024

OUTPUT SCREENSHOT:

PROGRAM-10

Write an YACC program to generate syntax tree for a given arithmetic expression

→ Code

p1.l

```
%{
#include "y.tab.h"
extern int yylval;
%}

%%
[0-9]+  { yylval = atoi(yytext); return digit; }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

p1.y

```
%{
# include <math.h>
# include <ctype.h>
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
```

```
struct tree_node
{
  char val[10];
   int lc;
   int rc;
};
int ind;
struct tree_node syn_tree[100];
void my_print_tree(int cur_ind);
int mknode(int lc, int rc, char val[10]);
%}

%token digit

%%
S:E { my_print_tree($1); }
;
E:E'+'T { $$ = mknode($1, $3, "+"); }
|T { $$ = $1; }
;
T:T'*'F { $$=mknode($1, $3, "*"); }
|F { $$ = $1; }
;

F:'('E')' {$$ = $2;}

|digit { char buf[10]; sprintf(buf,"%d", yylval ); $$ = mknode(-1,-1,
                                                    buf); }

%%
```

```
int main()
{
  ind=0;
  printf("Enter an Expression \n");
  yyparse();
  return 0;
}

int yyerror()
{
  printf("NITW Error\n");
}

int mknode(int lc, int rc, char val[10])
{
  strcpy(syn_tree[ind].val,val);
  syn_tree[ind].lc=lc;
  syn_tree[ind].rc=rc;
  ind++;
  return ind-1;
}

void my_print_tree(int cur_ind)
{
  if (cur_ind==-1) return;
  if (syn_tree[cur_ind].lc==-1 && syn_tree[cur_ind].rc==-1)
  printf("Digit Node->Index: %d, Value: %s\n", cur_ind, syn_tree
        [cur_ind].val);
  else
  printf("Operator Node->Index=%d, Value: %s, left child
        Index=%d, Right child Index: %d\n",
  my_print_tree(syn_tree[cur_ind].lc);
  my_print_tree(syn_tree[cur_ind].rc); }
```

Output:

Enter an Expression:

3+2*5

Operator Node → index: 4 value +, LCI = 0 , RCI = 3

Leaf Node → Index: 0, Value: 3

Operator Node → index = 3, Value: * LCI = 1, RCI = 2

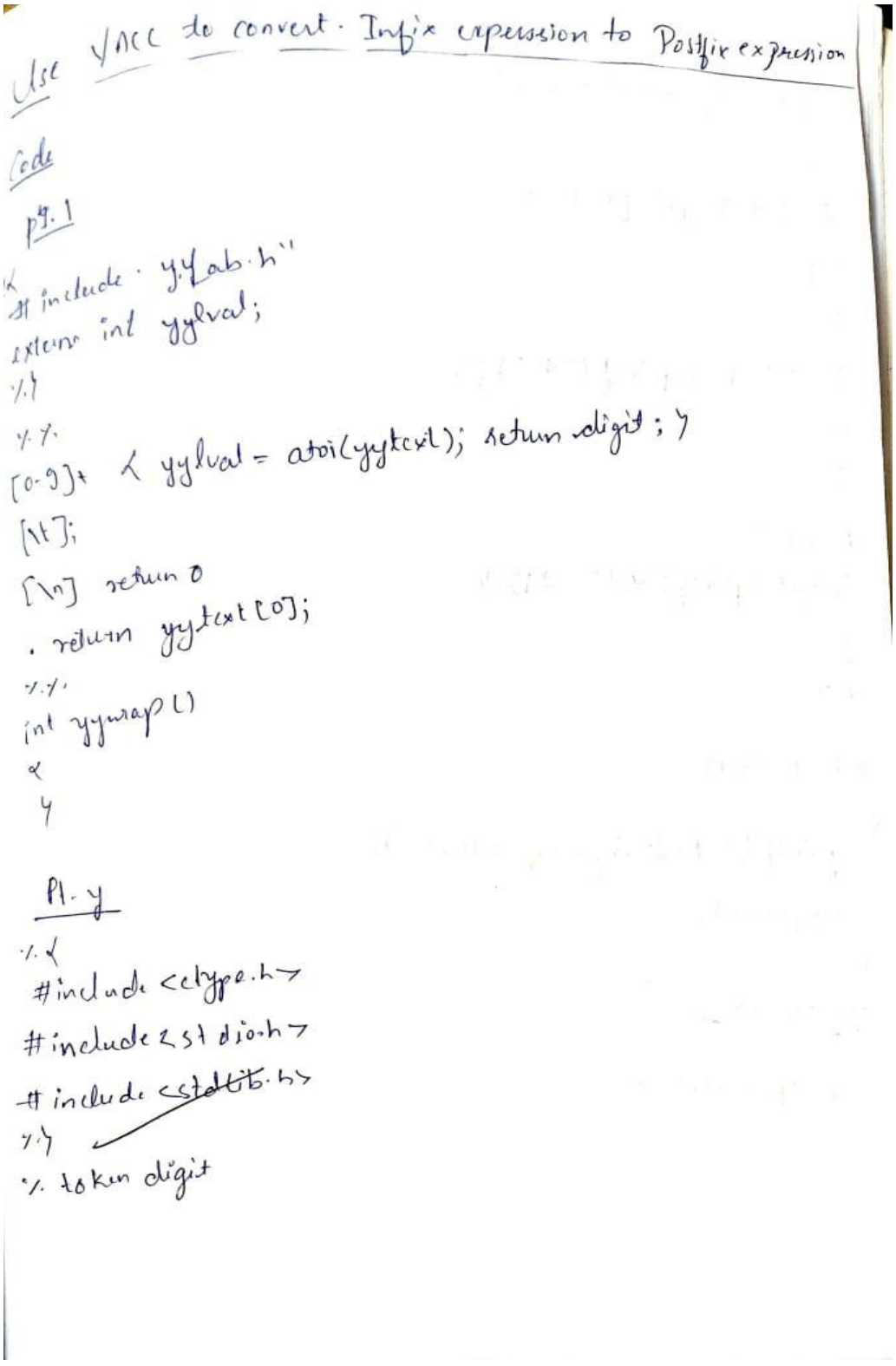
Leaf Node → Index: 1, value: 2

Leaf Node → Index: 2, value: 5

OUTPUT SCREENSHOT:

```
Enter an expression:
2*3+5*4
Operator Node -> Index : 6, Value : +, Left Child Index : 2,Right Child Index : 5
Operator Node -> Index : 2, Value : *, Left Child Index : 0,Right Child Index : 1
Digit Node -> Index : 0, Value : 2
Digit Node -> Index : 1, Value : 3
Operator Node -> Index : 5, Value : *, Left Child Index : 3,Right Child Index : 4
Digit Node -> Index : 3, Value : 5
Digit Node -> Index : 4, Value : 4
```

PROGRAM-11



Use YACC to convert Infix expression to Postfix expression

Code

p1.l

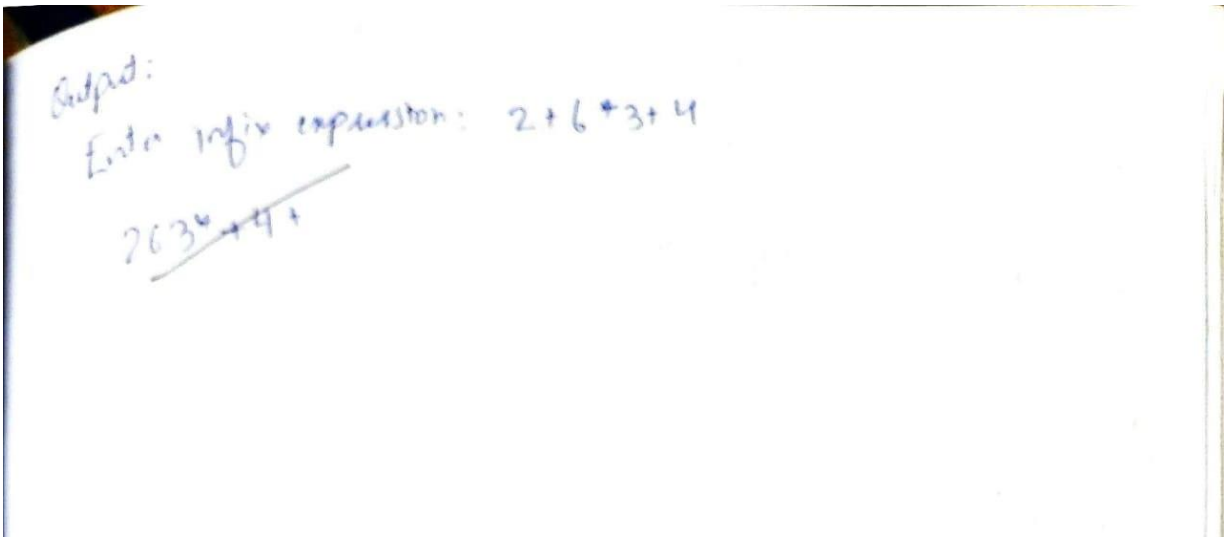
```
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ { yylval = atoi(yytext); return digit; }
[\t];
[\n] return 0
. return yytext[0];
%%
int yywrap()
{
}
```

P1.y

```
%{
#include <ctype.h>
#include<stdio.h>
#include <stdlib.h>
%}
%token digit
```

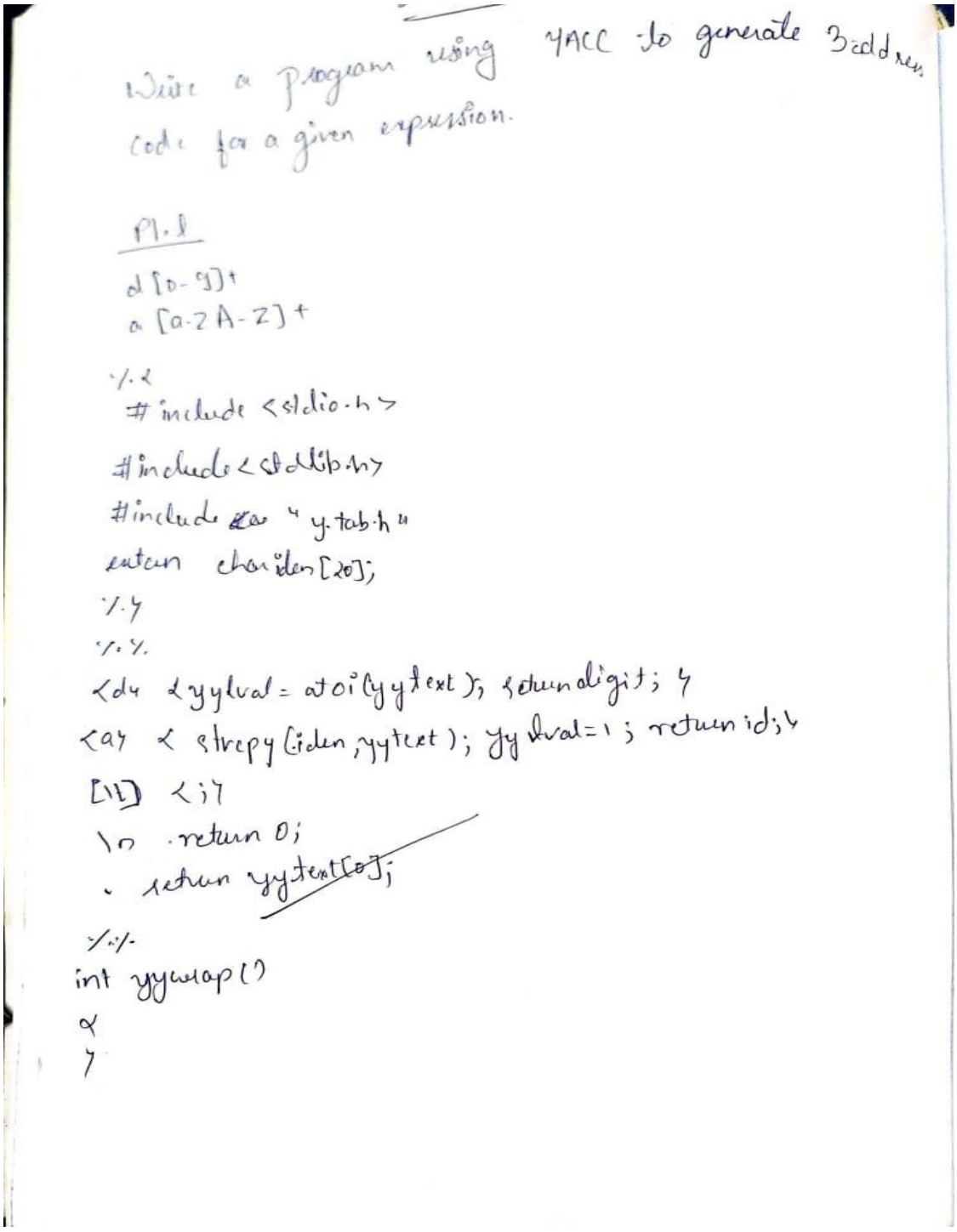
```
%%
S:E {printf("\n\n");}
;
E: E'+' T {printf("+");}
| T
;
T: T'*' F {printf("*");}
| F
;
F: '('E')'
| digit {printf("%d", $1);}
;
%%

int main()
{
  printf("Enter infix expression: ");
  yyparse();
}
yyerror()
{
  printf("Error");
}
```



OUTPUT SCREENSHOT:

PROGRAM-12



Write a program using YACC to generate 3 address code for a given expression.

P1.l

```
d [0-9]+
a [a-zA-Z]+
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
extern char iden[20];
%}
%%
[0-9]+ { yylval = atoi(yytext); return digit; }
[a-zA-Z]+ { strcpy(iden, yytext); yylval=1; return id; }
[\t] ;
\n .return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

p1.y

```
%{
#include<math.h>
#include<ctype.h>
#include<stdio.h>
int var_cnt = 0;
char iden[20];
%}
%token id
%token digit
%%
S:id '=' E { printf("%s = t%d\n", iden, var_cnt-1); }
E:E '+' T { $$ = var_cnt; var_cnt++; printf("t%d = t%d + t%d\n", $$, $1, $3); }
|E '-' T { $$ = var_cnt; var_cnt++; printf("t%d = t%d - t%d\n", $$, $1, $3); }
|T { $$ = $1; }
;
T:T '*' F { $$ = var_cnt; var_cnt++; printf("t%d = t%d * t%d\n", $$, $1, $3); }
|T '/' F { $$ = var_cnt; var_cnt++; printf("t%d = t%d / t%d\n", $$, $1, $3); }
|F { $$ = $1; }
;
F:'(' E ')' { $$ = var_cnt; var_cnt++; printf("t%d = t%d\n", $$, $2); }
|digit { $$ = $1; }
;
```

```
{: "[0-9]+" { yylval = atoi(yytext);
  [a-z] { yylval = yytext[0] - 'a';
  return VARIABLE; }
  \n { return 0; }
  . { return yytext[0]; }
%%
int main()
{
  var_cnt = 0;
  printf("Enter an expression: \n");
  yyparse();
  return 0;
}
yyerror()
{
  printf("error");
}
```

Output

a = 2+3*6

t0 = 2;

t1 = 3

t2 = 6;

t3 = t1 * t2;

t4 = t0 + t3;

a = t4;

OUTPUT SCREENSHOT: