VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

JAGADEESH A LATTI(1BM21CS079)

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 **JAGADEESH A LATTI(1BM21CS079)** , 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, Jagadeesh A Latti(1BM21CS079), 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 : Jagadeesh. A. L Class : 5-B

Section : Roll No. : Subject : CD-LAB

Sl. No.	Date	Title	Page No.	Teacher Sign. / Remarks
		Lex	9/10	
1		Write a lex program to count no of words in input		
2		Write a lex program to count no. of vowels & consonants.		
3		Write a lex program to identify alphabets & characters & numbers		
4		Write a program in LEX to recognize float point numbers		
5		Write a lex program that copies a file, replacing each non-empty sequence of white space		
6		The set of all strings ending with 00		
7		Write a program to design Lexical Analyser		

Sl. No.	Date	Title	Page No.	Teacher Sign. / Remarks
8		Write a program to perform Recursive DESCENT Parsing on grammar.		
9		Design a grammar for evaluation of arithmetic exp with left associativity.		
10		write yacc program string program $a^n b^n$ $n \geq 5$		
11		Write a yacc program to generate syntax for a given arithmetic expression		
12		Use YACC to convert Infix to Postfix expression		
13		YACC to generate 3-address code.		

PROGRAM-1

4) Write a Lex program to count no. of words in a input sentence.

```
%%
[a-zA-Z]* {count++;}
\n {printf("No of words: %d\n", count);}
%%
```

output:

Jaggu is here

No. of words: 3

5) Write a Lex program to count no of vowels & consonants in a string.

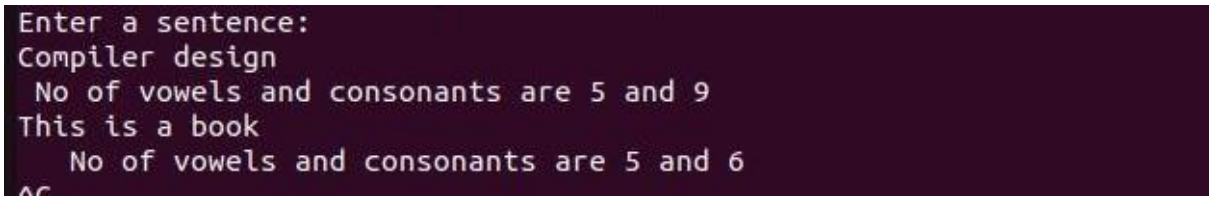
```
%%
a|e|i|o|u|A|E|I|O|U {countV++;}
[a-zA-Z] {countC++;}
\n {printf("No of vowels: %d\n", countV);
    printf("No of consonants: %d\n", countC);}
%%
```

output:

Good morning

No of vowels: 4 No of consonants: 7

OUTPUT SCREENSHOT:



PROGRAM-2

3) Write a Lex program to identify alphabets as characters & numbers as digits.

```
%option noyywrap
%{
#include<stdio.h>
%}
%%
[0-9]* {printf("%s: digit\n", yytext);}
[a-zA-Z]* {printf("%s: alphabets\n", yytext);}
%%
void main()
{
  yylex();
}
```

output:

Good evening 17

17: digits

Good evening: alphabets

4) Write a Lex program to count no. of words in a input sentence.

```
%%
[a-zA-Z]* {count++;}
\n {printf("No of words: %d\n", count);}
%%
```

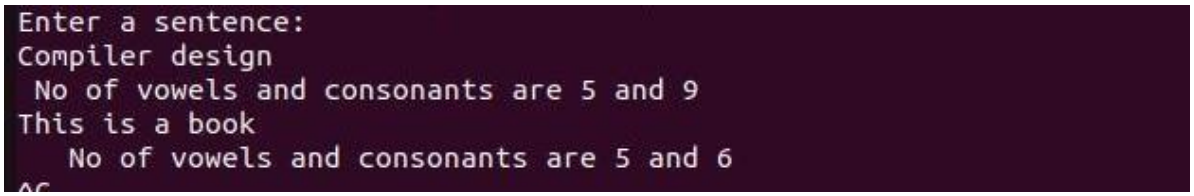
output:
Jaggu is here.
No. of words: 3

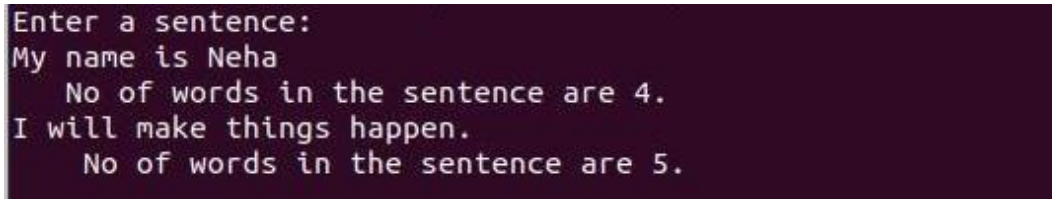
5) Write a Lex program to count no of vowels & consonants in a string.

```
%%
a|e|i|o|u|A|E|I|O|U {countV++;}
[a-zA-Z] {countC++;}
\n {printf("No of vowels: %d\n", countV);
    printf("No of consonants: %d\n", countC);}
%%
```

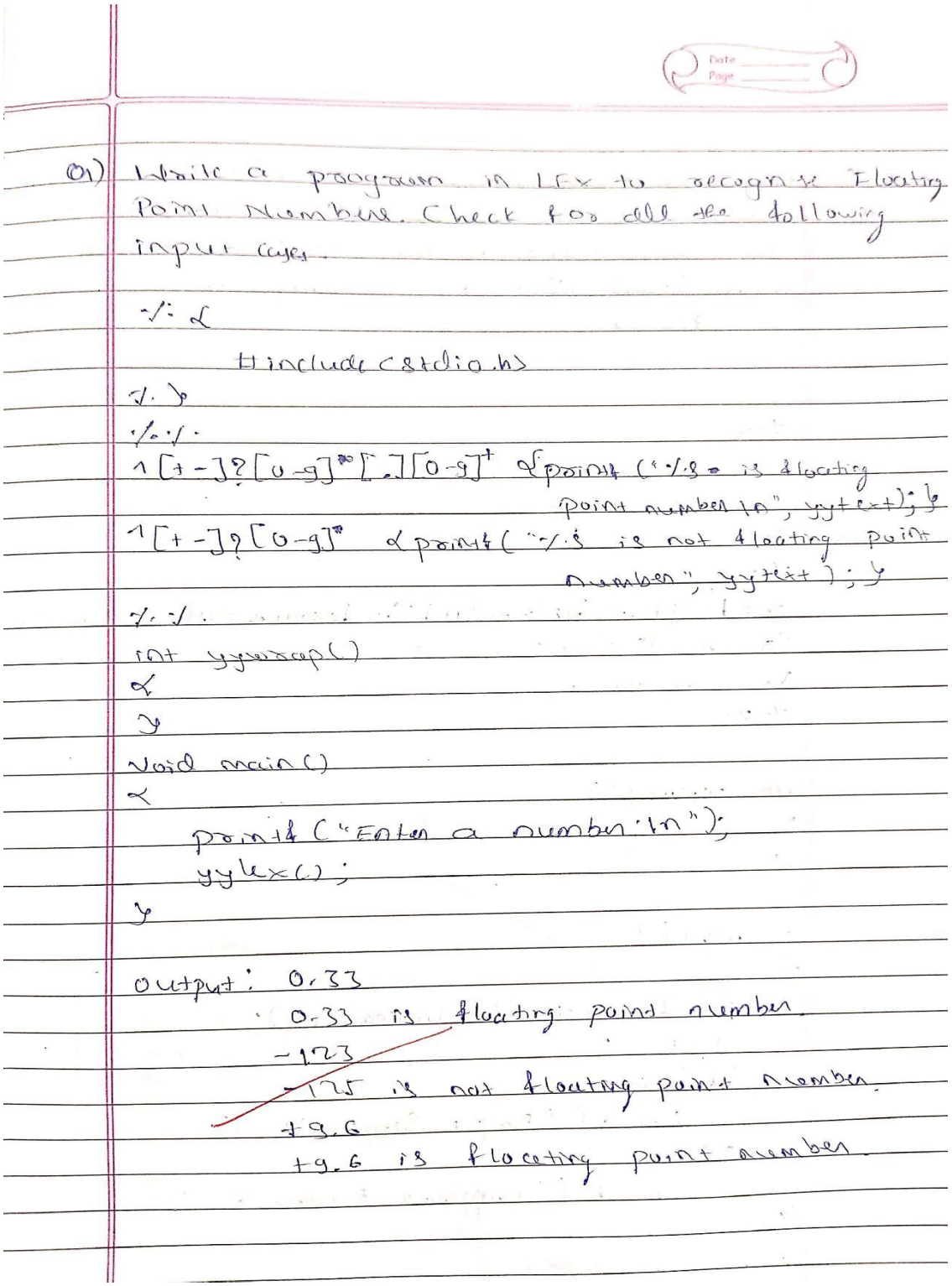
output:
Good morning
No of vowels: 4 No of consonants: 7

OUTPUT SCREENSHOT:





PROGRAM-3:

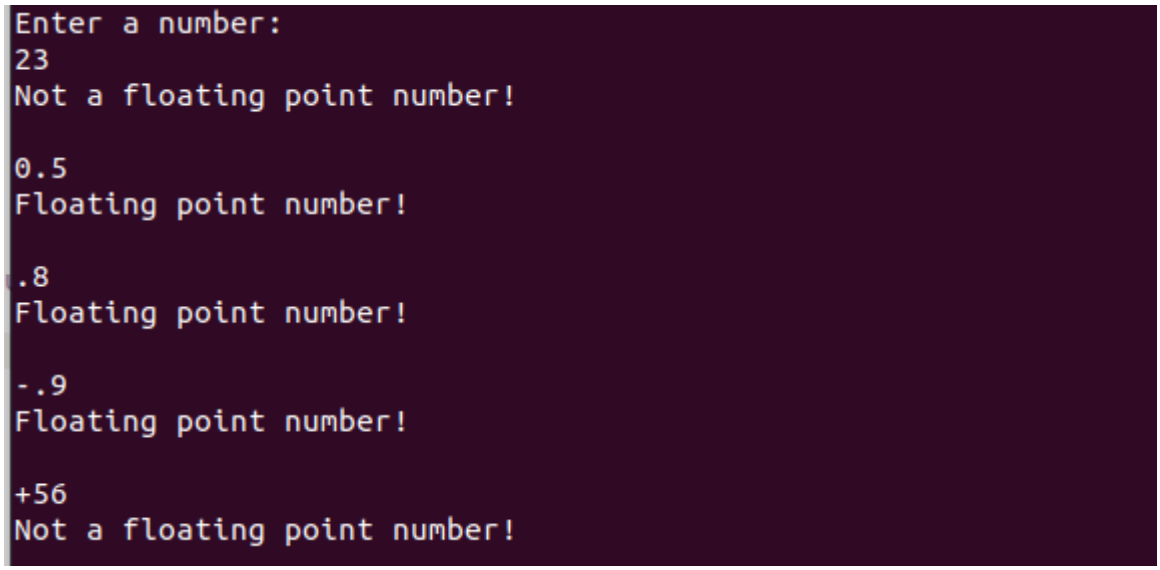


01) Write a program in LEX to recognize Floating Point Numbers. Check for all the following input cases.

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

Output: 0.33
0.33 is floating point number.
-123
-123 is not floating point number.
+9.6
+9.6 is floating point number.

OUTPUT SCREENSHOT:



PROGRAM-4

Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.

```
%{
    #include<stdio.h>
    #include<string.h>
    #include<stdlib.h>
    char string[200];
%}
%%
[\n] {fprintf(yyout,"%s\n",string); string[0]='\0';}
[ ]* {fprintf(yyout,"%s",string); string='\0';
      fprintf(yyout," ");}
      strcat(string,yytext);
<<EOF>> {fprintf(yyout,"%s",string); 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)
    {
        printf("Can't open file %s\n",filename);
        exit(1);
    }
    yylex();
}
```

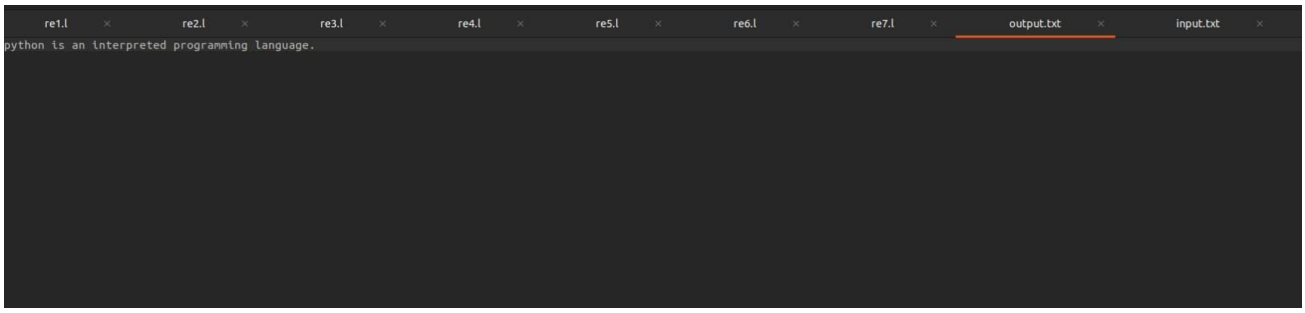
Sun 11/12/23

int yywrap(void)
{
    return 1;
}

Output:
Enter the name of the file to copy: in
Enter the name of the file to open for writing: out

[0-9][0-9][0-9][0...?
[0-9a-zA-Z]* { printf("Invalid\n"); }

OUTPUT SCREENSHOT:



PROGRAM-5

a) the set of all string ending in 00

```
%option noyywrap
%{
#include<stdio.h>
%}
%%
[0-9]*00 {printf("string accepted");}
[0-9]* {printf("string not accepted");}
%%

void main()
{
    printf("Enter string:");
    yylex();
}
```

output:

001

string rejected

100

string accepted.

b) The set of all strings with three consecutive 2's.

```
%option noyywrap
%{
#include<stdio.h>
%}
%%
[0-9]* 222[0-9]* { printf("String accepted."); }
[0-9]* { printf("String not Accepted."); }
%%

void main()
{
    printf("Enter String: ");
    yylex();
}
```

Output:

1239

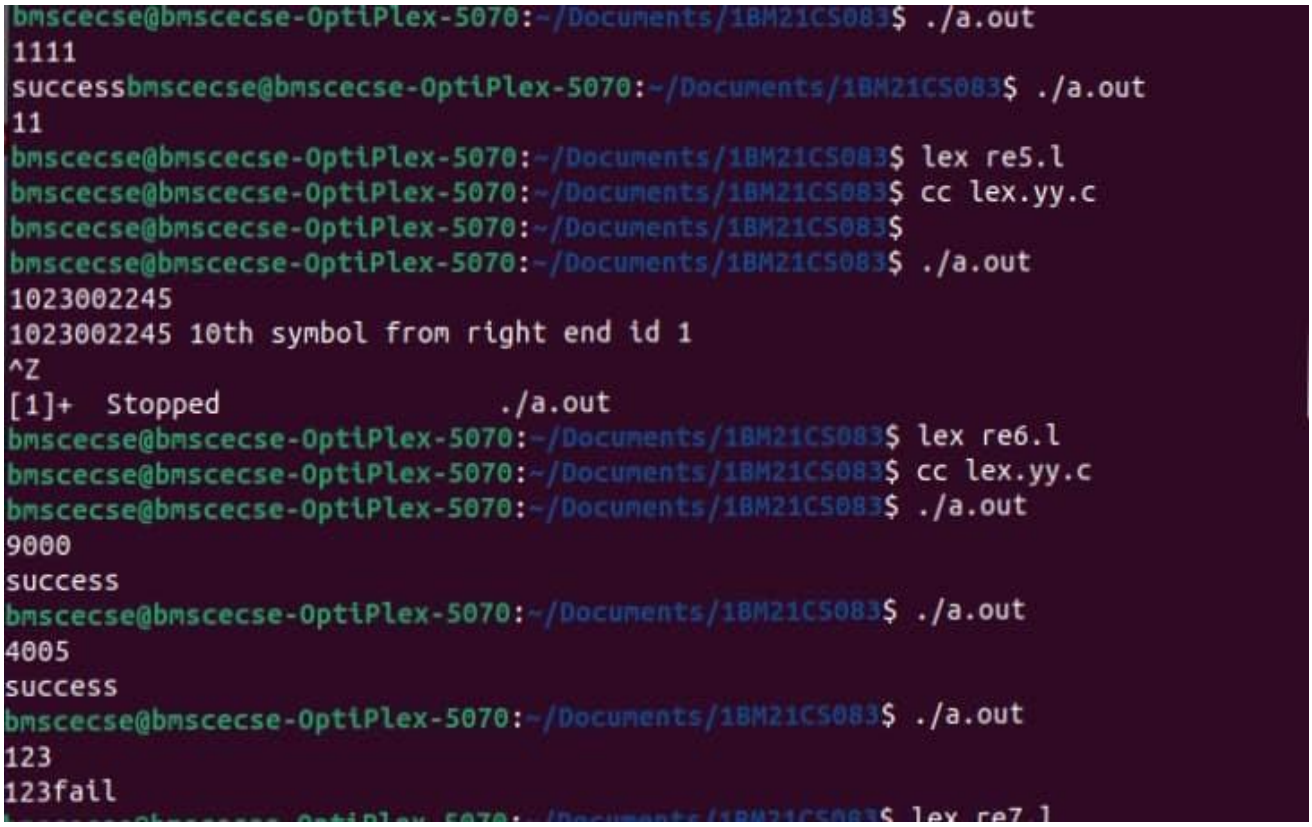
String rejected

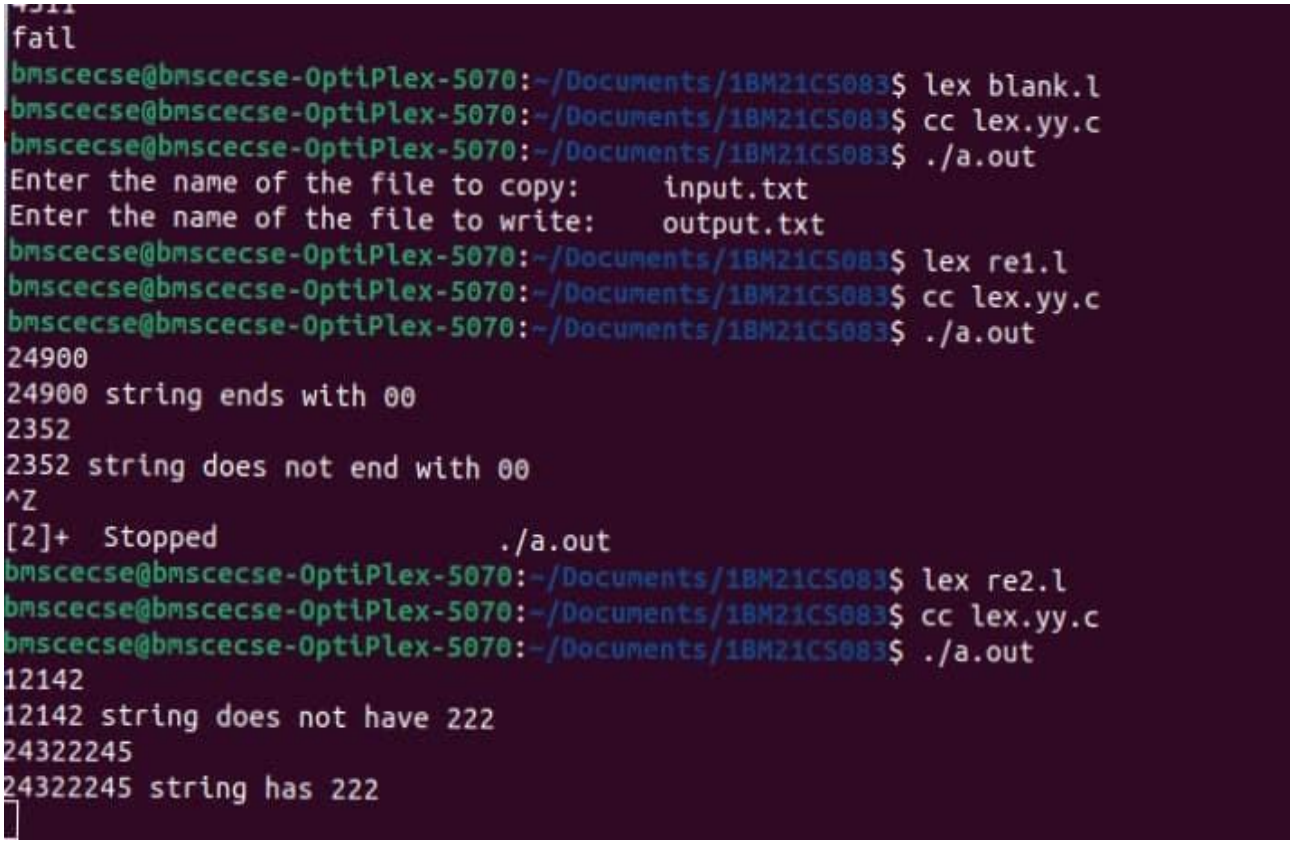
01222

String accepted

4/12/2023

OUTPUT SCREENSHOT:





PROGRAM-6

Write a program to design lexical Analyzer in C.

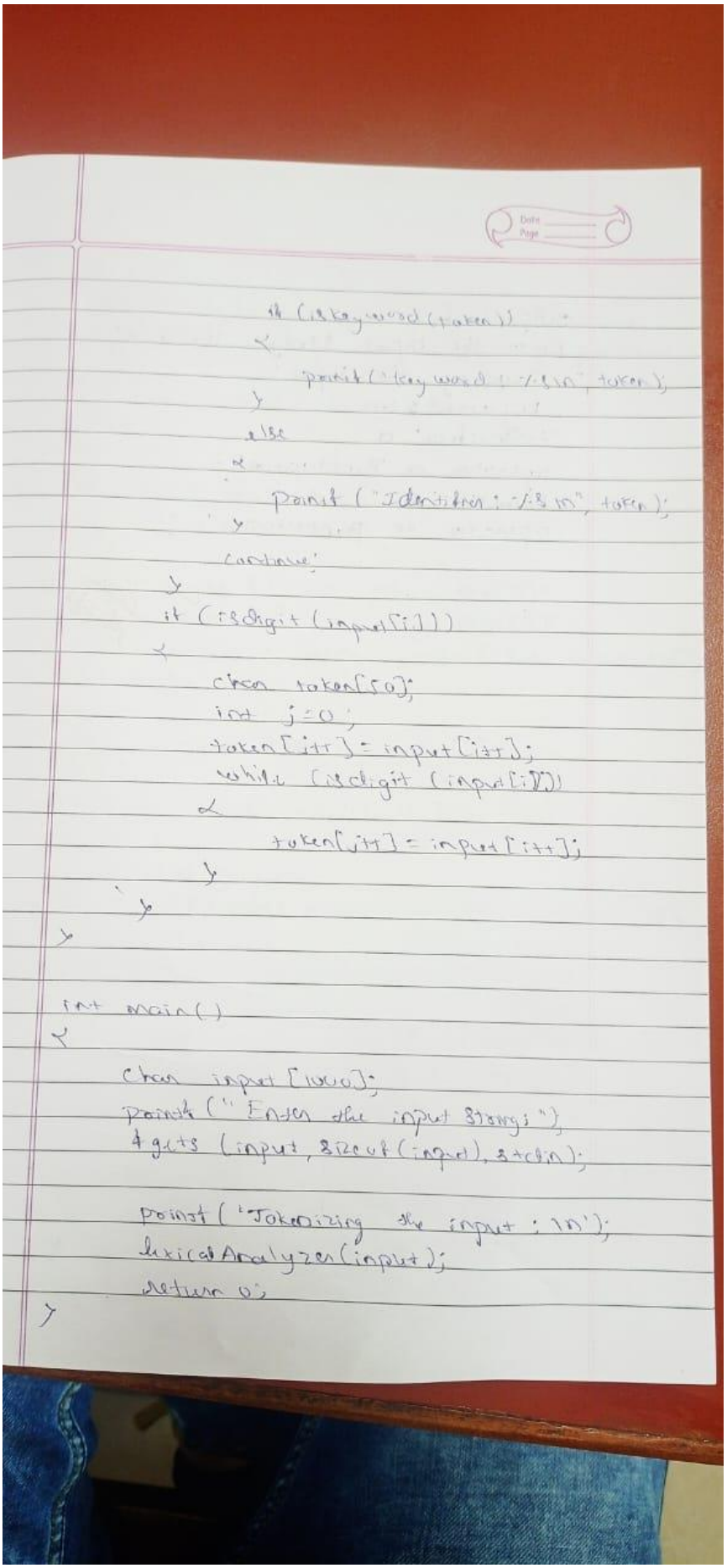
```
#include<stdio.h>
#include<string.h>
#include<ctype.h>

int isKeyword(char *str)
{
    char keywords[5][10]={"int","float","if",
                          "else","while"};
    int i;
    for(i=0;i<5;++i)
    {
        if(strcmp(keywords[i],str)==0)
            return 1;
    }
    return 0;
}

int isOperator or Punctuation(char ch)
{
    char operators[]="+-*/%=";
    char punctuations[]="();,{}[]";
    int i;
    for(i=0;i<strlen(operators);++i)
    {
        if(operators[i]==ch)
            return 1;
    }
```

```
for (i=0; i<strlen(punctuations); ++i)
{
    if (punctuations[i] == ch)
        return 1;
}
return 0;
}

void lexicalAnalyzer(char *input)
{
    int i=0;
    int len = strlen(input);
    while (i<len)
    {
        if (isspace(input[i]))
        {
            i++;
            continue;
        }
        if (isalpha(input[i]) || input[i]=='
        {
            char token[50];
            int j=0;
            token[j++] = input[i++];
            while (isalnum(input[i]) ||
                   input[i] == '_')
            {
                token[j++] = input[i++];
            }
            token[j]='\0';
```



```
        if (isKeyword(token))
        {
            printf("keyword : %s\n", token);
        }
        else
        {
            printf("Identifier : %s\n", token);
        }
        continue;
    }
    if (isdigit(input[i]))
    {
        char token[50];
        int j=0;
        token[j++] = input[i++];
        while (isdigit(input[i]))
        {
            token[j++] = input[i++];
        }
    }
}

int main()
{
    char input[1000];
    printf("Enter the input string: ");
    fgets(input, sizeof(input), stdin);

    printf("Tokenizing the input : \n");
    lexicalAnalyzer(input);
    return 0;
}
```

Output:

Enter the input string: int a=0;

Tokenizing the input:

Keyword: int

Identifier: a

operator or Punctuation: =

Number: 0

operator or punctuation: ;

18/12/23.

OUTPUT SCREENSHOT:

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

PROGRAM-7

Write a Program to perform Recursive Descent parsing on the following grammar

S → cAd, A → ab | a

```
#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 == 'd')
        {
            index++;
            return true;
        }
        else
            return false;
    }
    else
        return false;
}
```

```
bool parse_A (char input_str[])
{
    if (input[index] == 'a')
    {
        if (input[index] == 'b')
        {
            index++;
        }
        return true;
    }
    else
        return false;
}

bool recursive_descent (char input[])
{
    if (parse_S && input == '\0')
    {
        return true;
    }
    else
        return false;
}

int main()
{
    char user_input[100];
    printf ("Enter a string to parse: ");
    scanf ("%s", user_input);
    printf ("S -> (A) \n");
    printf ("A -> (b|a) \n");
```

```
if (recursive_descent(&str_input))
{
    printf("The given string is accepted\n");
}
else
{
    printf("The given string is not accepted\n");
}
}
```

Output:

Enter a string to parse: cdd

S -> cAd

A -> abla

The given string is not accepted by grammar.

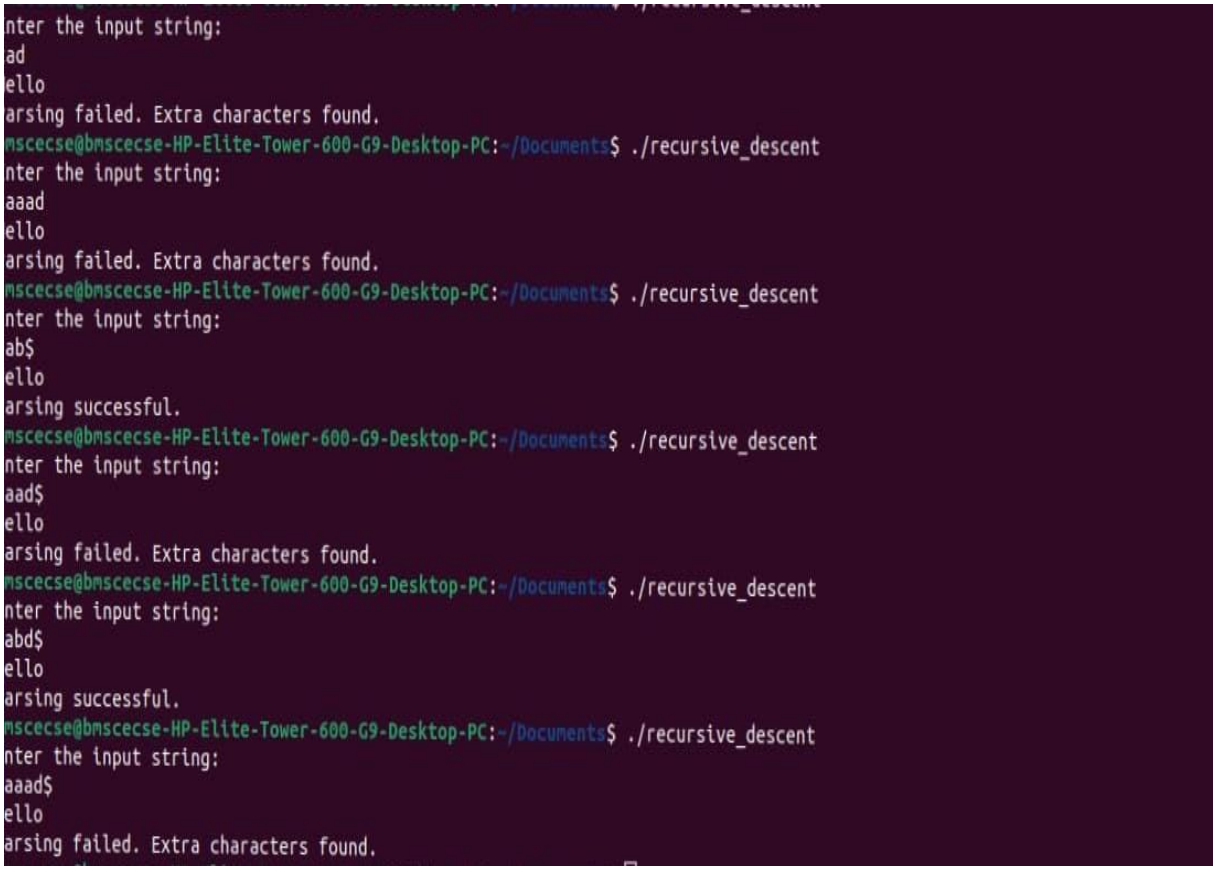
Enter a string to parse: cabd

S -> cAd

A -> abla.

The given string is accepted by grammar.

OUTPUT SCREENSHOT:



Design a suitable grammar for evaluation of arithmatic exps. having + & - operators. + has least & left associative.

Code:

```
%.option noyywrap
%.{
#include "y.tab.h"
%.}
[0-9]+ { yylval = atoi(yytext); return NUM; }
[\t];
\n return 0;
. return yytext[0];
%-1.
```

```
%.{
#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;
}
```

Output:

Enter an expression

5+10-2

Valid expression

Result: 13.

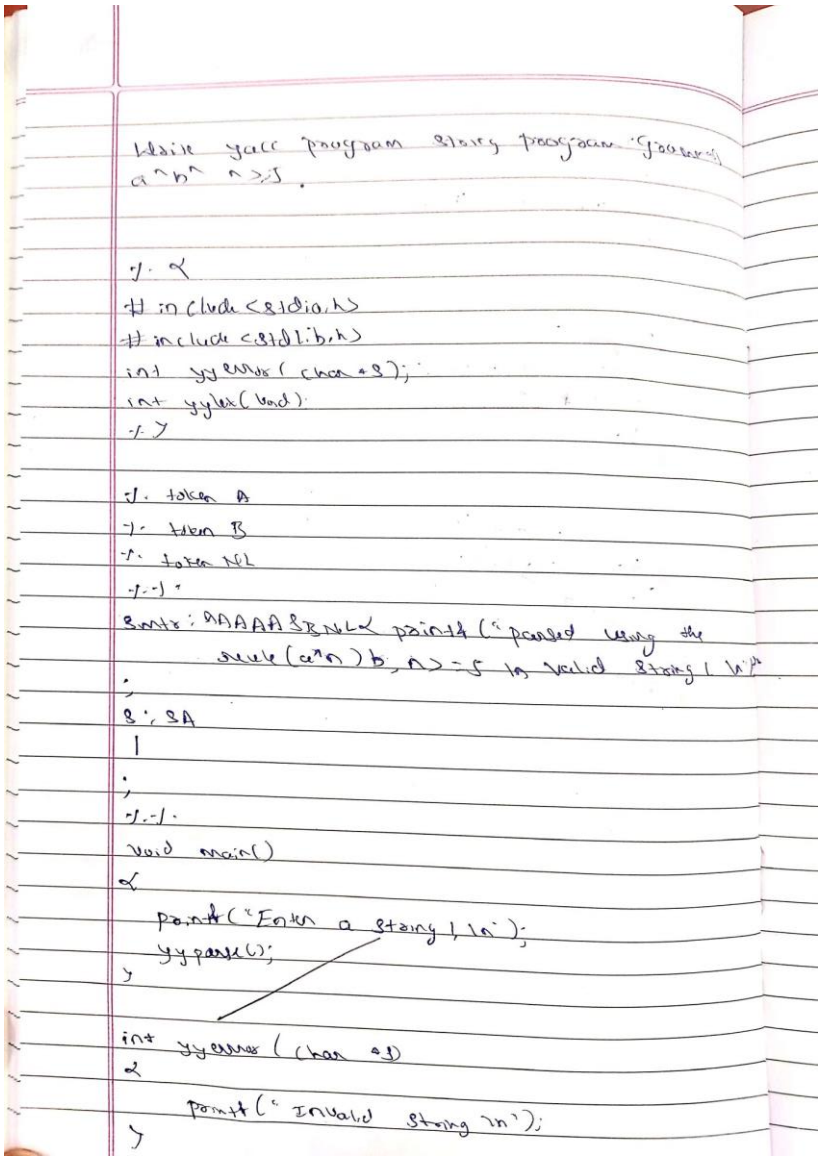
Jan
8/1/24

OUTPUT SCREENSHOT:

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

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

PROGRAM-9



While yacc program string program given
a^b^ n>=5.

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

%token A
%token B
%token NL
%%
Stmt: SAAAAA$ NL { printf("parsed using the
        rule (a^n) b, n>=5 is valid string \n"); }
;
S: SA
|
;
%%
void main()
{
    printf("Enter a string \n");
    yyparse();
}

int yyerror(char *s)
{
    printf("Invalid string \n");
}
```

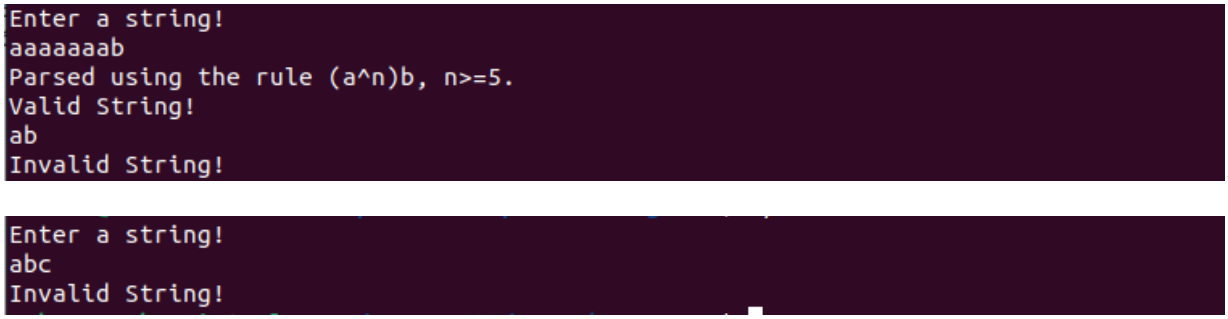
```
%{
#include<stdio.h>
#include<stdlib.h>
int yylval;
%}
%%
[aA] { yylval = yytext[0]; return A; }
[bB] { yylval = yytext[0]; return B; }
\n { return NL; }
. { return yytext[0]; }
%%
int yywrap()
{
    return 1;
}
```

Output:

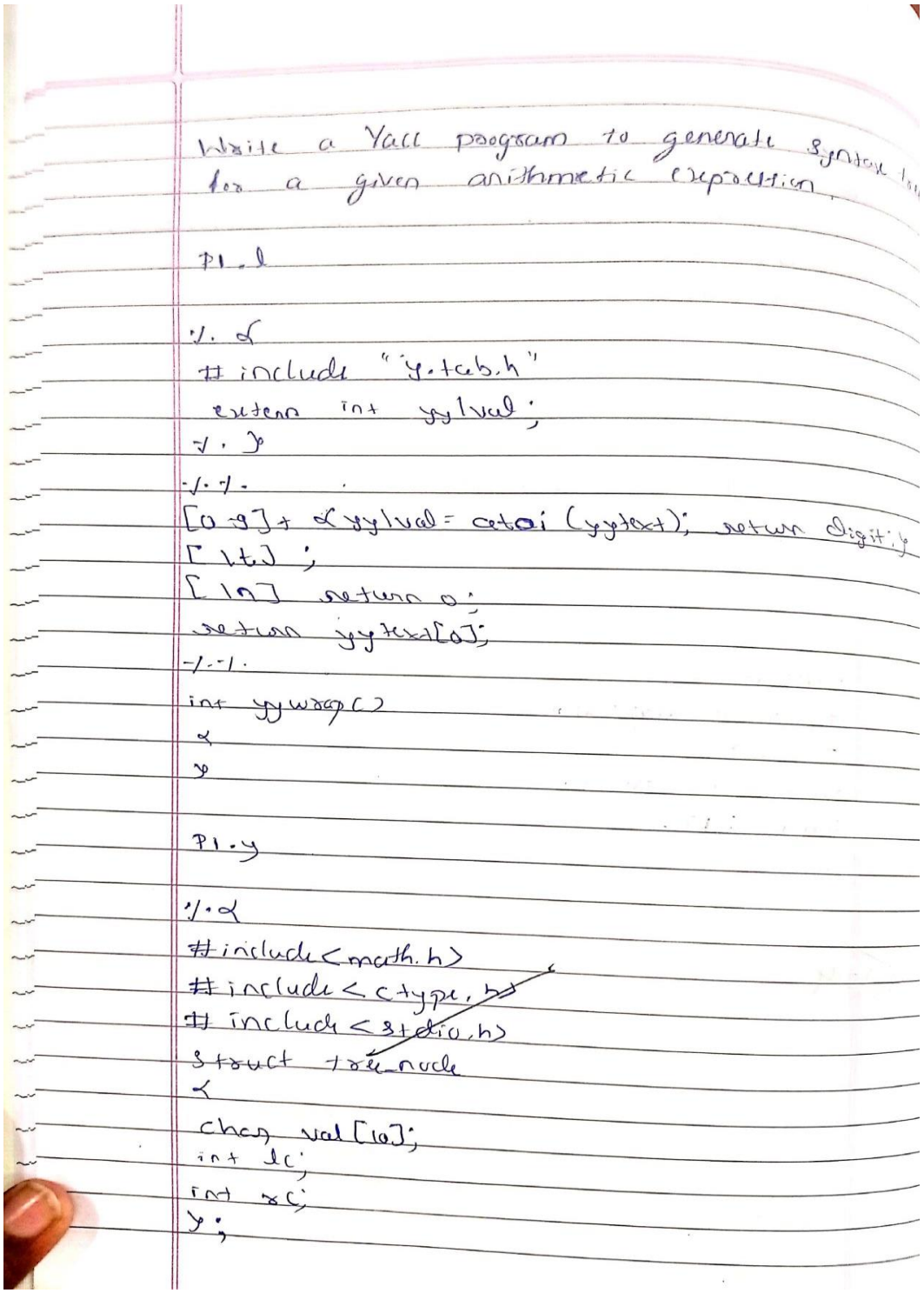
Enter a string:
aaaaab
Parsed using the rule (a^n)b, n>=5
valid string.
aaaby
invalid string.

29/11/2024

OUTPUT SCREENSHOT:



PROGRAM-10



Write a Yacc program to generate syntax tree for a given arithmetic expression.

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>
struct tree_node
{
char val[10];
int lc;
int rc;
};
```

```
int ind;

struct tree_node syn_tree[100];
void mknode (int lc, int rc, char val[10]);
%}
%token digit
%%
S: E { my_print_tree ($1); }
;
E: E '+' T { $$ = mknode ($1, $3, "+"); }
| T { $$ = $1; }
;
%%

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

int yyerror()
{
printf("NEW Error\n");
}
int mknode (int lc, int rc, char val[10])
{
   strcpy (syn_tree[ind].val, val);
   syn_tree[ind].lc = lc;
   ind++;
}
```

```
void my_post_tree (int cur_ind)
{
if (cur_ind == -1) return;
if (syn_tree[cur_ind].l == -1 && syn_tree[cur_ind].r == -1)
printf ("Digit Node -> Index : %d, value: %g\n",
        cur_ind, syn_tree[cur_ind].val);
else
printf ("Operator Node -> Index: %d, Value: %c\n");
my_post_tree (syn_tree[cur_ind].l);
my_post_tree (syn_tree[cur_ind].r);
}
```

Output:

Enter an expression

2+3*5

Operator Node -> Index: 4, value: +, left child: 0, Right child: 3

Leaf Node -> index: 0, value: 2

operator Node -> Index: 3, value: *, left child: 1, Right: 2

Leaf Node -> Index: 1, value: 3

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 to Postfix expression

p9.l

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

p9.y

```
%{
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
%}
%token digit
%%
S: E {printf("\n");}
;
E: E'+'T {printf("+");}
```

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

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

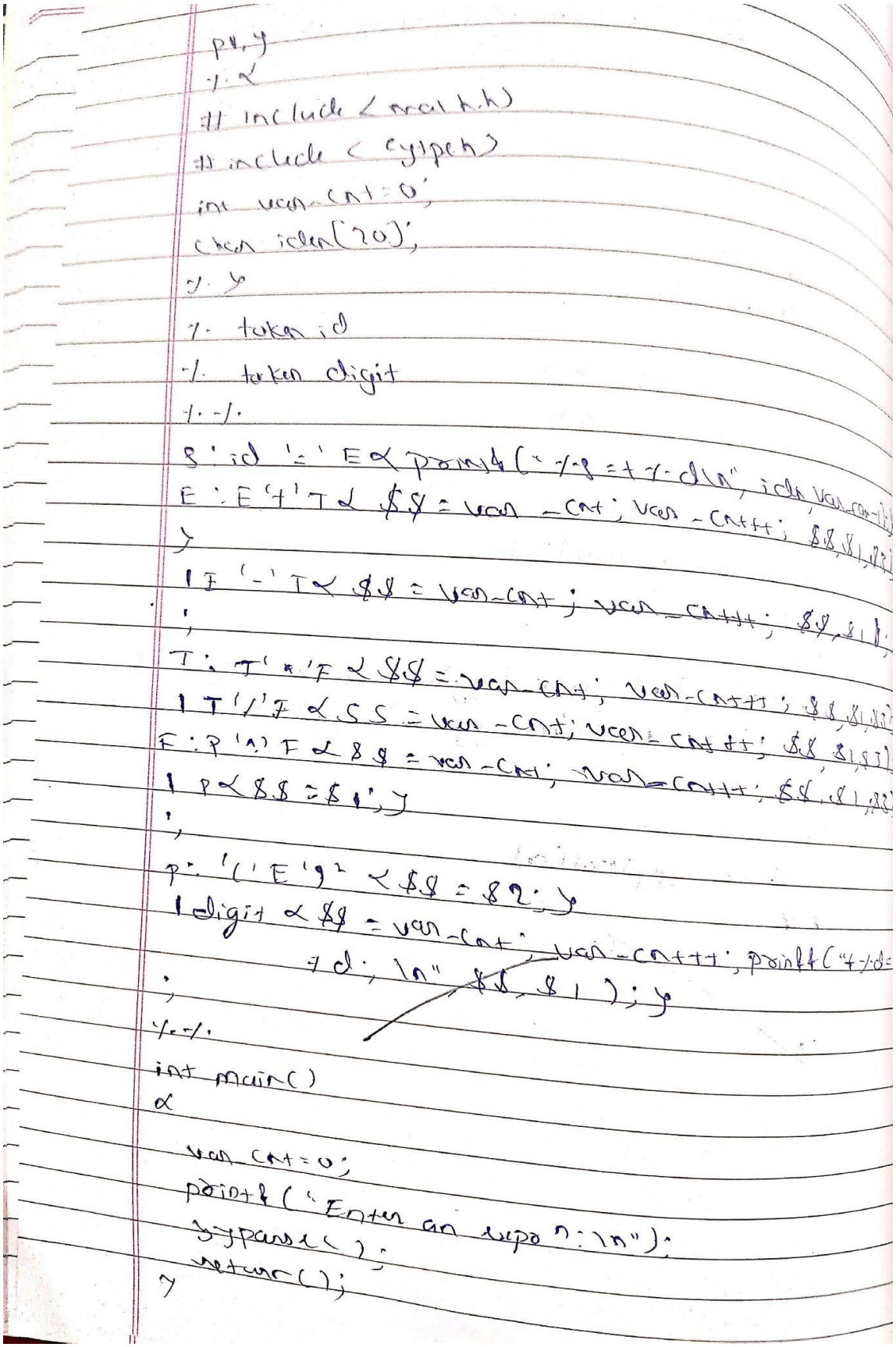
Output:

Enter infix expression: 2+6*3+4

2 6 3 * + 4 +

OUTPUT SCREENSHOT:

PROGRAM-12



```
p4.y
%{
#include<math.h>
#include<ctype.h>
int var_cnt=0;
char idstr[20];
%}
%token id
%token digit
%%
S: id '=' E{ printf("%d=%d\n", idstr, var[var_cnt-1]);
E: E'+'T{ $$=var_cnt; var_cnt++; $$,$1,$3}
| E'-'T{ $$=var_cnt; var_cnt++; $$,$1,$3}
| T;
T: T'*'F{ $$=var_cnt; var_cnt++; $$,$1,$3}
| T'/'F{ $$=var_cnt; var_cnt++; $$,$1,$3}
F: P'^'F{ $$=var_cnt; var_cnt++; $$,$1,$3}
| P{ $$=$1;}
;
P: '('E')'{ $$=$2;}
| digit{ $$=var_cnt; var_cnt++; printf("%d=%d\n", $$,$1);}
;
%%
int main()
{
  var_cnt=0;
  printf("Enter an expn:\n");
  yyparse();
  return();
}
```

yy error()

{

printf("error");

}

Output:

Enter an expression:

a = 2 * 3 ^ 6

t0 = 2;

t1 = 3;

t2 = 6;

t3 = t1 ^ t2;

t4 = t0 + t3;

a = t4

29/8/2024

OUTPUT SCREENSHOT: