VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Compiler Design

Submitted by

KEERTHI P REDDY
(1BM21CS090)

in partial fulfillment 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
Nov-2023 to Feb-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 **KEERTHI P REDDY (1BM21CS090),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to Sep-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Compiler Design(22CS5PCCPD)** work prescribed for the said degree.

Sunayna S	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Part-A: Implementation of Lexical Analyzer, By using C/C++/Java/Python language and using LEX tool.

Program 1:

```
Lex program to count no. of vowels & consonants
in a ~~set~~ sentence.

[aeiouAEIOU] { vow_count ++; }
[a-zA-Z] { const_count ++; }
\n { printf("vow_count = %d , const_count = %d ",
            vow_count, const_count); }

OUTPUT:

Enter the string of vowels & consonants : Keerthi
vow_count = 3 , const_count = 4
```

Output:

```
Enter the string of vowels and consonents: KEERTHI
vow_count:  3 , const_count: 4
```

Program 2:

An program to read the following input from a file & print the valid token on the terminal.

```
%option noyywrap
%{
#include <stdio.h>
char fname[20];
%}
%%
int|float|char {printf("keywords : %s\n", yytext);}
[0-9]* {printf("number: %s\n", yytext);}
[a-zA-Z]* {printf("character: %s\n", yytext);}
%%
```

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

OUTPUT:

```
Enter the file name : p.txt
keywords: float
number: 0978
character: abc.
```

Output:

```
enter the input file name
input.txt
enter the output file name
output.txt
```

```
1 int a,b;
```

```
int Keywords a Identifiers, Seperatorb Identifiers; Seperator
```

Program 3:

Lex program to recognize floating point no.'s.

```
^[+-]?[0-9]*[.][0-9]+ { printf("Floating point no."); }
^[+-]?[0-9]* { printf("Not a valid floating point no."); }
```

OUTPUT:

```
4.5
floating point no.
4
Not a valid floating point no.
+3.89
Floating point no.
-3.89
Floating point no.
.44
Floating point no
44.
Not a valid floating point no.
```

Output:

```
Enter the number: 5
Not Floating point no.
.6
Floating point no.
7.8
Floating point no.
```

Program 4:

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

```
%{
#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", str1);
       strcat (str1, yytext);
<<EOF>> { fprint (yyout, "%s", str1);
       return 0;
%%
```

```
int main ()
{
    extern FILE *yyin,*yyout
    char filename [100];
    printf(" Enter name of file to copy : \t");
    scanf(" %s", filename);
    yyin = fopen (filename , " r");
    if (yyin == NULL)
    {
        exit(0);
    }
    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:

Open

input.txt
~/Desktop/CD LAB/Lab Programs

```
1 this is      a sample      text for        the program
```

Open

output.txt
~/Desktop/CD LAB/Lab Programs

```
1 this is a sample text for the program
```

Program 5:

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

a. The set of all strings ending in 00

```
%%
[0-9]*00 { printf("string accepted"); }
[0-9]* { printf("string rejected"); }
%%
int yywrap()
{
}
int main()
{
    yylex();
    return 0;
}
```

OUTPUT:

```
lex string_end_00.l
cc lex.yy.c
./a.out
10100
string accepted
34560
string rejected.
```

b. The set of all strings with 3 consecutive 222's

```
%%
[0-9]*222[0-9]* { printf("string contains 222"); }
[0-9]* { printf("string does not contain 222"); }
%%
```

OUTPUT:

12223

String contains 222.

1234

String does not contain 222.

c. The set of all string such that every block of five consecutive symbols contains at least two 5's.

```
%%
[0-9]*
{
    int i, c=0;
    if (yyleng < 5)
    {
        printf("%s doesn't match any such \n", yytext
    }
    else
    {
        for (i=0; i<5; i++)
        {
            if (yytext[i] == '5')
            {
                c++;
            }
            if (c==2)
            {
                for (i < yyleng ; i++)
            {
```

```
        if (yytext[i-5] == '5') { c--; }
        if (yytext[i] == '5') { c++; }
    }
    if (c < 2)
    {
        printf("%s doesn't match any rule\n", yytext);
        break;
    }
    }
    else
    {
        printf("%s doesn't match any rule\n", yytext);
    }
    }
%%
```

d. The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer is congruent to zero modulo 5.

```
%%
(1 (0)*(11|01)(01*01|01*10(0)*(11|1)*0)
 (1|10(0)*(11|01)(01*01|00*10(0)*(11|1)*10)
printf("The set of all strings beginning with
        a 1 is interpolated as the binary
        representation of an integer");
}
%%
```

e. The set of all strings such that the 10th symbol from ~~from~~ the right end is 1.

```
%%
({a})* 1 {a}{a}{a}
    printf("The set of all strings such that
            the 10th symbol from the right
            end is 1");
}
%%
```

OUTPUT

1011100010

The set of all string such that the 10th symbol from right end is 1.

f. The set of all 4 digit no.'s whose individual digits are in ascending order from left to right.

```
%%
{d}{a}
int sum = 0, i;
for (i= 0; i<4; i++)
{
    sum = sum + yytext [i] - 48;
}
if (sum == 9)
{
  printf("The sum of digits whose sum is 9");
}
```

```
}
}
% %
```

OUTPUT:

0 2 3 4

The sum is 9.

9. The set of all 4 digit no.'s whose individual digits are in ascending order from left to right:

```
% %
{d}{U}{
    for (i=0; i<3; i++)
    {
        if (gytext[i] > gytext[i+1])
        {
            sum = 0;
            break;
        }
    }
    if (sum==1)
    {
        printf("ascending order from left to right
    }
    else
    {
        printf("%s doesnt match any rule\n",
    }
}
% %
```

OUTPUT:

4 5 6 7

in ascending order.

11/12/2023

Output:

```
Enter text
1200
1200 -> string ending in 00

122299
122299 -> string with three consecutive 222's

10
10 doesn't match any rule

157495
157495 doesn't match any rule
```

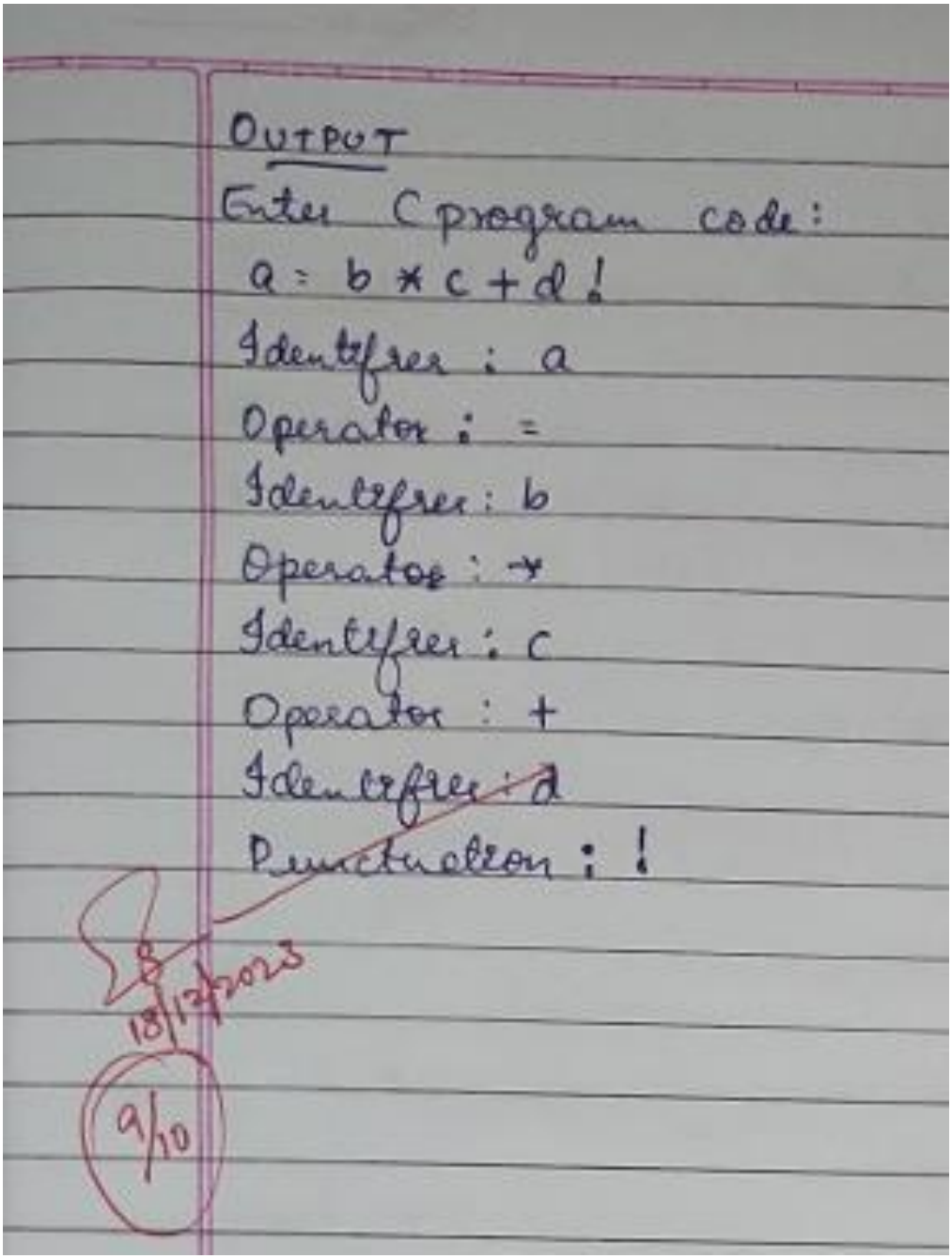
Program 6:

WAP to design a Lexical Analyzer to recognize any 5 keywords, identifiers, no's, operators & punctuations.

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

#define KEYWORD 1
#define IDENTIFIER 2
#define NUMBER 3
#define OPERATOR 4
#define PUNCTUATION 5
#define INVALID 6

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

```
void analyzeToken(char *token)
{
    if (isKeyword(token) == KEYWORD)
    {
        printf("KEYWORD: %s \n", token);
    }
    else if (isdigit(token[0]))
    {
        printf("Number: %s \n", token);
    }
    else if (isalpha(token[0]) || token[0] == '_')
    {
        printf("Op Identifier: %s \n", token);
    }
    else if (strch("+-*/=", token[0]) != NULL)
    {
        printf("Operator: %s \n", token);
    }
    else
    {
        printf("Invalid Token: %s \n", token);
    }
}

int main()
{
    char input[500];
    char token[100];
    int tokenIndex = 0;
    int i;
```

```
printf("Enter the C program code: \n");
fget(input, size(input), stdin);
for (i=0; i<=strlen(input); ++i)
{
    if(isalnum(input)|| ~~strlen~~; input[i]=='_')
    {
        token[tokenIndex++] = input[i];
    }
    else
    {
        if(tokenIndex>0)
        {
            token[tokenIndex] = '\0';
            analyzeToken(token);
            tokenIndex=0;
        }
        if(input[i]!=' ' && input[i]!='\n'
                && input[i]!='\t')
        {
            token[0] = input[i];
            token[1] = '\0';
            analyzeToken(token);
        }
    }
}
return 0;
}
```



OUTPUT

Enter C program code:

a = b * c + d !

Identifier : a

Operator : =

Identifier : b

Operator : *

Identifier : c

Operator : +

Identifier : d

Punctuation : !

18/12/2023

9/10

Output:

```
Enter the sentence: int
Keyword
abc
Identifiers
+
Operator
!
Punctuator
123
Constants
```

Part-B: Part-B: Implementation of Parsers (Syntax Analyzers) Using C/C++/Java/Python language)

Program 1:

PROGRAM 18: Recursive Decent

```
#include <stdio.h>
#include < stdlib.h>
char input [100];
int ind =0;
void match (char expected)
{
        if (input [ind ] = = expected )
        {
            ind ++;
        }
}
void A();
void S()
{
        match( 'c');
        A();
        match ( 'd');
}
void A()
{
        if (input [ind]== 'a')
        {
            printf (" Hello \n");
            match ('a'),
            match( 'b');
        }
```

```
        else
        {
            printf("Parsing Failed.\n", end);
            exit(1);
        }
}
int main()
{
    printf("Enter the input string:\n");
    scanf("%s", input);
    s();
    if (input[end] == '$')
    {
        printf("Parsing successful.\n");
    }  ~~scanf("~~
    else
    {
        printf("Parsing Failed. Extra character
                                    found");
    }
    return 0;
}
```

OUTPUT:

Enter the input string: cabd$
Parsing Successful.

Enter the input string: caac$
Parsing Failed. Extra characters found.

Output:

```
Enter the input string: cabd
Parsing successful! Input belongs to the given grammar.
```

```
Enter the input string: caab
Parsing failed! Input does not belong to the given grammar.
```

Part-C: Syntax Directed Translation using YACC tool

Program 1:

String match $a^n b$ where $n > 5$

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

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

%token A
%token B
%token NL
```

```
%%
Sinta: AAAAA SBB    NL {printf(" Parsed using
                        the rule (a^n b) 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;
}
```

Output

```
Enter a string
a a a a a b
Parsed using the rule (a^n)b, n >=5
valid string

a a b b b
Invalid String!
```

Output:

```
Enter the string : aaaaab$
Parsed using the rule(a^n)b, n>5
Valid string!
```

```
Enter the string : aabb$
Invalid string!
```

Program 2:

PROGRAM -19 : Desk Calculator

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


%{
#include<stdio.h>
%}
%token NUM
%left '+'
%right '-'
%%
exp : 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");
```

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

OUTPUT

```
>lex prog1.l
>yacc -d prog1.y
>gcc lex.yy.c y.tab.c
> ./a.out
Enter an arithmetic expression
5+6-3-6
Valid expression
Result: 14
```

8/1/24

Output:

```
Enter an arithmetic expression:
1+2*3%1^2
Valid expression
Result: 1
```

Program 3:

Date :
Page No :

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

p1.y

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

struct tree_node
{
   char val[10];
   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 : '+' T  { $$ = mknode ($1, $3, "+"); ; }
| T  { $$ = $1; }
;
```

```
T: T '*' F   { $$ = mknode ($1, $3, "+"); ; }
| F          { $$ = $1; }
;

F: '(' E ')'  { $$ = $2; }
| digit  { char buf[10]; sprintf(buf, "%d", yyle
           $$ = mknode(-1, -1, buf); }
%%

int main()
{
    ind = 0;
    printf("Enter an expression");
    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 = ;
    syn_tree[ind].rc = rc;
    ind ++;
    return ind - 1;
}
```

```
void my-print-tree (int cur-ind)
{
        if (cur-ind == -1) return;
        if (sym-tree[cur-ind].lc == -1 && sym-tree
                                [cur-ind].rc == -1)
        printf (" Digit Node -> Index : %d, Value : %d",
                cur-ind, sym-tree [cur-ind].val);
        else
        printf (" Operator Node -> Index : %d, Value : %c
                Left Child Index : %d, Right Child Index : %d
                \n sym-tree[cur-ind].lc. sym-tree
                                [cur-ind]. );
    my-print-tree (sym-tree[cur-ind].lc);
    my-print-tree( sym-tree( cur-ind].rc);
}
```

P.l

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

Output

Enter an expression

2 +4

Operation Node -> Index:2, Value: +, left Child Index:0

Right Child Index:1

Digit Node -> Index:0, Value:2

Digit Node -> Index:1, Value:4

Output:

```
Enter an expression
4+6*9
Operator Node -> Index : 4, Value : +, Left Child Index : 0,Right Child Index : 3
Digit Node -> Index : 0, Value : 4
Operator Node -> Index : 3, Value : *, Left Child Index : 1,Right Child Index : 2
Digit Node -> Index : 1, Value : 6
Digit Node -> Index : 2, Value : 9
```

Program 4:

```
Infix Postfix
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern int yylval;
%}

%%
[0-9]+   {yylval = atoi(yytext); return num;
[\t];
\n       {return 0;}
         {return yytext[0];}
%%

int yywrap()
{
}


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

%token num
%left '+' '-'
%left '*' '/'
```

```
%left ')'
%left '('
%right '^'
%%

S: e   {printf("\n");}
;
e: e '+' t   {printf(" +");}
| e '-' t   {printf(" - ");}
| t
;
t: t '*' h   {printf(" * ");}
| t '/' h   {printf(" / ");}
| h
;
h: f '^' h   {printf(" ^");}
| f
;

f: '(' e ')'
| num   {printf("%d, $t");}
;

%%
void main()
{
        printf(" Enter an infix expression: \n");
        yyparse();
}
int yyerror(const char *s)
{
```

```
    printf("Invalid infix expression!\n");
    return 0;
}
```

Output:

```
Enter an infix expression
3+9*8
398*+

Enter an infix expression
3+9*6/5+*+
396*5/+ Invalid expression!
```

Output:

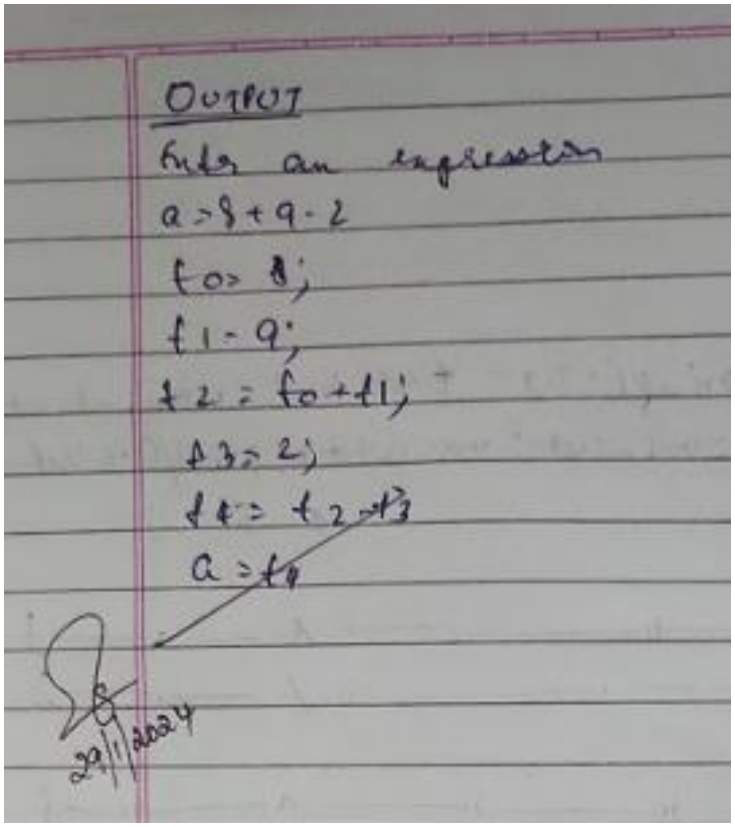
```
Enter an infix expression:
3+6*2-1/3
362*+13/-
```

```
Enter infix expression:  2+3*4
234*+
```

Program 5:

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

%%
int yywrap()
{
    return 1;
}

Address & Copy.y

%{
#include<math.h>
#include <ctype.h>
#include <stdio.h>

int yyerror(char *s);
```

```
char Iden[20];
%
%token id
%token digit
%%
S: id '=' E {printf("%s = %d \n", Iden, val_cut);}
E: E '+' T {$$ = val_cut; val_cut++; printf("%d
                                    , $$);}

T: T '*' F {  ———— " ———— " ———— * ———— " ———— }
| T '/' F {  ———— " ———— " ———— / ———— " ———— };
| F { $$ = $1; }
F: P '^' F { ———— " ———— " ———— ^ ———— " ———— };

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


int yyerror(char *s)
{
   return 0;
}
```



Output:

```
Enter an expression:
a=8+9-2
t0 = 8;
t1 = 9;
t2 = t0 + t1;
t3 = 2
t4 = t2 - t3;
a = t4;
```