**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**COMPILER DESIGN**

***Submitted by***

**POOJASHREE K V (1BM21CS129)**

***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**

**October-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 **POOJASHREE K V(1BM21CS129),** 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 year 2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Compiler Design course (21CS5PCCPD)**work prescribed for the said degree.

## Sunayana S               Dr. Jyothi S Nayak

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
|  | **Part-A: Implementation of Lexical Analyzer, By using C/C++/Java/Python language and using LEX tool.** |  |
| **01** | **Write a program to design Lexical Analyzer in (to recognize any five keywords, identifiers, numbers, operators and punctuations)** | **5-6** |
| **02** | **Write a program in LEX to recognize Floating Point Numbers.** | **7-8** |
| **03** | **Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.** | **9-10** |
| **04** | **Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.** | **11-12** |
| **05** | **Write a LEX program to recognize the following tokens over the alphabets {0,1,..,9}**   1. **The set of all string ending in 00.** 2. **The set of all strings with three consecutive 222’s.** 3. **The set of all string such that every block of five**   **consecutive symbols contains at least two 5’s.**   1. **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. **The set of all strings such that the 10th symbol**   **from the right end is 1.**   1. **The set of all four digits numbers whose sum is 9** 2. **The set of all four digital numbers, whose** 3. **individual digits are in ascending order from left to** 4. **right.** | **13-14** |
|  | **Part-B: Implementation of Parsers (Syntax Analyzers) Using C/C++/Java/Python language)** |  |
| **01** | **Write a program to implement**   1. **Recursive Descent Parsing with back tracking**   **(Brute Force Method). S→ cAd , A →ab /a**   1. **Recursive Descent Parsing with back tracking**   **(Brute Force Method). S→ cAd , A → a / ab** | **15-18** |
| **02** | **Write a program to implement: Recursive Descent Parsing with back tracking (Brute Force Method).**   1. **S→ aaSaa | aa** 2. **S → aaaSaaa | aa** 3. **S → aaaaSaaaa | aa** 4. **S → aaaSaaa |aSa | aa** | **19-31** |
|  | **Part-C: Syntax Directed Translation using YACC tool** |  |
| **01** | **Write a program to design LALR parsing using YACC** | **32-33** |
| **02** | **Use YACC to Convert Binary to Decimal (including fractional numbers)** | **34-35** |
| **03** | **Use YACC to implement, evaluator for arithmetic expressions (Desktop calculator)** | **36-38** |
| **04** | **Use YACC to convert: Infix expression to Postfix expression.** | **39-40** |
| **05** | **Use YACC to generate Syntax tree for a given expression** | **41-44** |
| **06** | **Use YACC to generate 3-Address code for a given expression** | **45-47** |
| **07** | **Use YACC to generate the 3-Address code which contains Arrays.** | **48-53** |

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

**PROGRAM 1**

**Write a program to design Lexical Analyzer in C/C++/Java/Python Language (to recognize any five keywords, identifiers, numbers, operators and punctuations)**

import re

def lexical\_analyzer(input\_text):

keywords = ["if", "else", "for", "while", "return"]

operators = ['+', '-', '\*', '/', '=', '==', '!=', '<', '>', '<=', '>=']

punctuations = [';', ',', '(', ')', '{', '}']

tokens = []

# Tokenize the input\_text

words = re.findall(r'\b\w+\b', input\_text)

for word in words:

if word in keywords:

tokens.append(("Keyword", word))

elif re.match(r'^[a-zA-Z\_]\w\*$', word):

tokens.append(("Identifier", word))

elif re.match(r'^[0-9]+$', word):

tokens.append(("Number", word))

elif word in operators:

tokens.append(("Operator", word))

elif word in punctuations:

tokens.append(("Punctuation", word))

return tokens

if \_\_name\_\_ == "\_\_main\_\_":

input\_text = "if x == 5 for i in range(10): print(i); else: print('Not 5')"

tokens = lexical\_analyzer(input\_text)

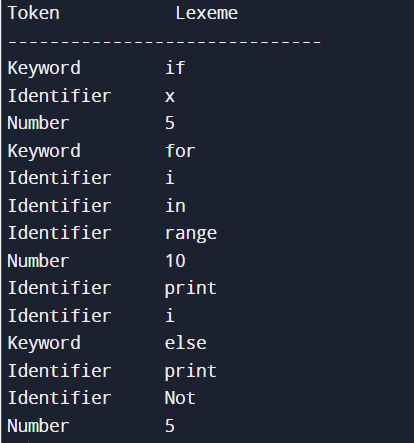
print("Token\t\t\tLexeme")

print("-" \* 30)

for token, lexeme in tokens:

print(f"{token.ljust(15)}{lexeme}")

Output:



**PROGRAM 2**

**Write a program in LEX to recognize Floating Point Numbers.**

%{

#include<stdio.h>

%}

digit [0-9]

num {digit}+

snum [-+]?{num}

%%

({snum}[.]{num})|({num}[.]{num})|([.]{num})|([-+][.]{num}) {printf("%s is a floating number\n", yytext);}

({snum}|{num}) {printf("%s is not a floating number\n", yytext);}

%%

int yywrap() {

return 1;

}

int main() {

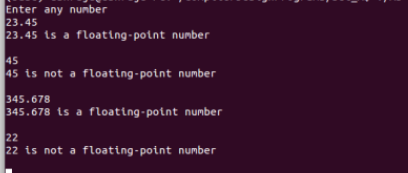
printf("Enter a number: ");

yylex();

return 0;

}

Output:



**PROGRAM 3**

**Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.**

d [0-9]

a [a-zA-Z]

z [a-zA-Z0-9]

x [.]

%%

int|float|char {x1++;}

{a}{z}\* {x2++;}

==|>=|<=|>|< {x3++;}

,|; {x4++;}

[+-]?{d}{d}\*({x}{d}{d}\*)?({x}{d}\*(e[+-]?{d}+)?)? {x5++;}

\n {

printf("Number of keywords:%d\n", x1);

printf("Number of Identifiers:%d\n", x2);

printf("Number of Operators:%d\n", x3);

printf("Number of punctuation:%d\n", x4);

printf("Number of constants:%d\n", x5);

printf("Total number of components:%d\n", x1 + x2 + x3 + x4 + x5);

}

%%

int yywrap() {

return 1;

}

int main() {

x1 = x2 = x3 = x4 = x5 = 0;

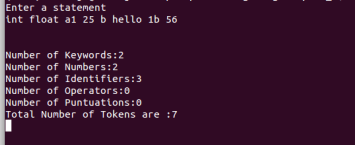
printf("Enter: ");

yylex();

return 0;

}

Output:



**PROGRAM 4**

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

%{

#include<stdio.h>

%}

%%

[ ]([ ])\* {fprintf(yyout," ");}

([ ])\*(\n)([ ])\* {fprintf(yyout," ");}

%%

int yywrap()

{

return 1;

}

int main()

{

yyin=fopen("filename.txt","r");

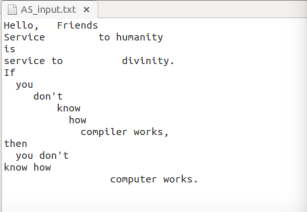
yyout=fopen("filename.txt","w");

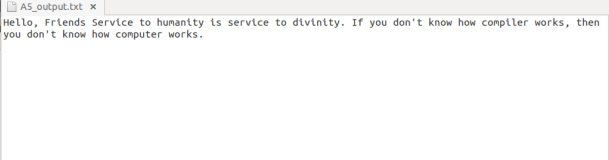
yylex();

return 0;

}

Output:





**PROGRAM 5**

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

**a) The set of all string ending in 00.**

**b) The set of all strings with three consecutive 222’s.**

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

**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.**

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

**f) The set of all four digits numbers whose sum is 9 g) The set of all four digital numbers, whose individual digits are in ascending order from left to right.**

%{

#include <stdio.h>

%}

digit [0-9]

%%

.\*00$ { printf("Token a) String ending in 00: %s\n", yytext); }

.\*222.\* { printf("Token b) String with three consecutive 222's: %s\n", yytext); }

[^5]\*5[^5]\*5[^5]\*5[^5]\*5[^5]\*5[^5]\* { printf("Token c) String with every block of five consecutive symbols containing at least two 5's: %s\n", yytext); }

^1[01]\*0[01]\*$ { printf("Token d) String beginning with a 1 and congruent to zero modulo 5: %s\n", yytext); }

^.{9}1.\*$ { printf("Token e) String with the 10th symbol from the right end being 1: %s\n", yytext); }

^[0-9][0-9][0-9]9$ { printf("Token f) Four-digit numbers whose sum is 9: %s\n", yytext); }

^[0-9][0-9][0-9][0-9]$ { if(yytext[0]<=yytext[1] && yytext[1]<=yytext[2] && yytext[2]<=yytext[3]) printf("Token g) Four-digit numbers with digits in ascending order: %s\n", yytext); }

.|\n

%%

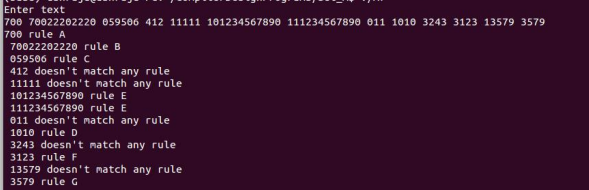
int main() {

yylex();

return 0;

}

Output:



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

**PROGRAM 1**

**Write a program to implement**

1. **Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A →ab /a**

def S(input\_str):

global index

if index < len(input\_str) and input\_str[index] == 'c':

index += 1

if A(input\_str):

if index < len(input\_str) and input\_str[index] == 'd':

index += 1

return True

return False

def A(input\_str):

global index

if index < len(input\_str) and input\_str[index] == 'a':

index += 1

if index < len(input\_str) and input\_str[index] == 'b':

index += 1

return True

elif index < len(input\_str) and input\_str[index] == 'a':

index += 1

return True

return False

def parse(input\_str):

global index

index = 0

if S(input\_str) and index == len(input\_str):

print("Parsing successful!")

else:

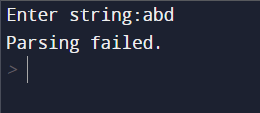
print("Parsing failed.")

# Example usage:

input\_string =input(‘Enter a string:’)

parse(input\_string)

Output:



1. **Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A → a / ab**

def S(input\_str):

global index

if index < len(input\_str) and input\_str[index] == 'c':

index += 1

if A(input\_str):

if index < len(input\_str) and input\_str[index] == 'd':

index += 1

return True

return False

def A(input\_str):

global index

current\_index = index # Backtrack point

if index < len(input\_str) and input\_str[index] == 'a':

index += 1

return True

else:

index = current\_index # Backtrack

if index < len(input\_str) and input\_str[index] == 'a':

index += 1

if index < len(input\_str) and input\_str[index] == 'b':

index += 1

return True

return False

def parse(input\_str):

global index

index = 0

if S(input\_str) and index == len(input\_str):

print("Parsing successful!")

else:

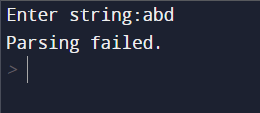
print("Parsing failed.")

# Example usage:

input\_string = input(‘Enter a string:’)

parse(input\_string)

Output:



**PROGRAM 2**

**2. Write a program to implement: Recursive Descent Parsing with back tracking (Brute Force Method).**

**(a) S→ aaSaa | aa**

#include<bits/stdc++.h>

using namespace std;

int curr;

//??

int S(char b[],int l)

{

//match with aa

char prod[20];

int isave=curr;

strcpy(prod,"aaSaa");

if(curr<l && b[curr]=='a')

{

curr++;

if(curr<l && b[curr]=='a')

{

curr++;

//recursive call to match S

if(S(b,l))

{

if(curr<l && b[curr]=='a')

{

curr++;

if(curr<l && b[curr]=='a')

{

curr++;

return 1;

}

}

}

}

}

//match with aa

strcpy(prod,"aa");

curr=isave;

if(curr<l && b[curr]=='a')

2

{

curr++;

if(curr<l && b[curr]=='a')

{

curr++;

return 1;

}

}

return 0;

}

int main()

{

curr=0;

char a[500];

cout<<"Enter the string : ";

cin.getline(a,500,'\n');

int l=strlen(a);

cout<<"length = "<<l<<endl;

if(S(a,l) && curr==l)

{

cout<<"Accepted\n";

}

else

{

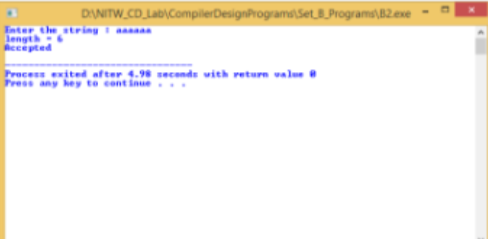
cout<<"Not Accepted\n";

}

return 0;

}

Output:



**(b)S → aaaSaaa | aa**

#include<bits/stdc++.h>

using namespace std;

int i;

//??

//tries all possible centres recursively and try to match the

string

int S(char b[],int l)

{

int isave=i;

//match with aa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

//match with S recursively

if(S(b,l))

{

//match with aa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

return 1;

}

}

}

}

}

i=isave;

//match with middle aa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

return 1;

}

}

return 0;

}

int main()

{

5

i=0;

char a[500];

memset(a,'\0',500);

for(int j=0;j<400;j++)

{

a[j]='a';

i=0;

if(S(a,j+1) && i==j+1)

{

cout<<j+1<<" ";

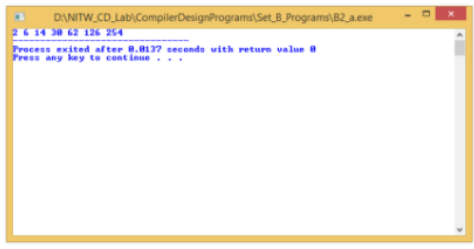
}

}

return 0;

}

Output:



**(c)S → aaaaSaaaa | aa**

#include<bits/stdc++.h>

using namespace std;

int i;

//??

//checks for grammer S->aaaaSaaaa | aa

//tries all possible centres recursively and try to match the

string

int S(char b[],int l)

{

int isave=i;

//match with aaaa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

//match with S recursively

if(S(b,l))

{

//match with aaaa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

if(i<l &&

b[i]=='a')

{

i++;

return 1;

}

}

}

}

}

}

}

}

9

}

i=isave;

//match with middle aa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

return 1;

}

}

return 0;

}

int main()

{

i=0;

char a[500];

memset(a,'\0',500);

for(int j=0;j<400;j++)

{

a[j]='a';

i=0;

if(S(a,j+1) && i==j+1)

{

cout<<j+1<<" ";

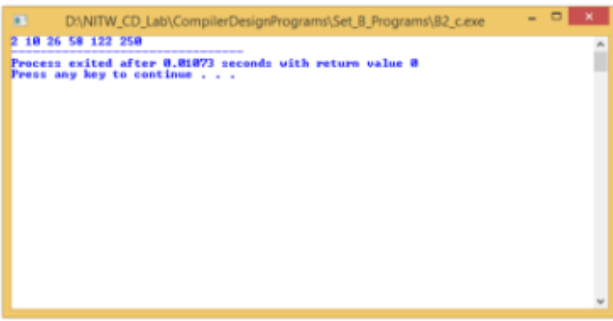
}

}

return 0;

}

Output:



**(d)S → aaaSaaa |aSa | aa**

#include<bits/stdc++.h>

using namespace std;

int i;

//??

//checks for grammer S->aaaSaaa | aSa | aa

//tries all possible centres recursively and try to match the

string

int S(char b[],int l)

{

int isave=i;

//match with aaa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

//match with S recursively

if(S(b,l))

{

//match with aaa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

return 1;

}

}

}

}

}

}

}

i=isave;

//match with a

if(i<l && b[i]=='a')

{

i++;

//match with S recursively

11

if(S(b,l))

{

//match with a

if(i<l && b[i]=='a')

{

i++;

return 1;

}

}

}

i=isave;

//match with middle aa

if(i<l && b[i]=='a')

{

i++;

if(i<l && b[i]=='a')

{

i++;

return 1;

}

}

return 0;

}

int main()

{

i=0;

char a[500];

memset(a,'\0',500);

for(int j=0;j<400;j++)

{

a[j]='a';

i=0;

if(S(a,j+1) && i==j+1)

{

cout<<j+1<<" ";

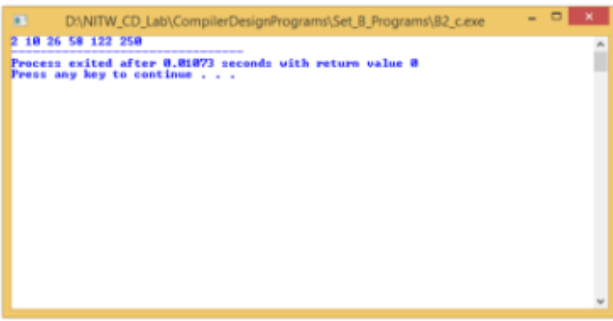
}

}

return 0;

}

Output:



**Part-C: Syntax Directed Translation using YACC tool**

**PROGRAM 1**

**Write a program to design LALR parsing using YACC.**

c1.y

%{

#include <ctype.h>

#include<stdio.h>

#include<stdlib.h>

%}

%token digit

%%

S: E {printf("Reached\n\n");}

;

E: E '+' T

| E '-' T

| T

;

T: T '\*' P

| T '/' P

| P

;

P: F '^' P

| F

;

F: '(' E ')'

| digit

;

%%

int main()

{

printf("Enter infix expression: ");

yyparse();

}

yyerror()

{

printf("NITW Error");

}

C1.l

%{

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext); return digit;}

[\t] ;

[\n] return 0;

. return yytext[0];

%%

Output:



**PROGRAM 2**

**Use YACC to Convert Binary to Decimal (including fractional numbers)**

C2.y

%{

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

void yyerror(char \*s);

float x = 0;

%}

%token ZERO ONE POINT

%%

L: X POINT Y {printf("%f",$1+x);}

| X {printf("%d", $$);}

X: X B {$$=$1\*2+$2;}

| B {$$=$1;}

Y: B Y {x=$1\*0.5+x\*0.5;}

| {;}

B:ZERO {$$=$1;}

|ONE {$$=$1;};

%%

int main()

{

printf("Enter the binary number : ");

// calling yyparse function which execute grammer rules and

lex

while(yyparse());

printf("\n");

}

void yyerror(char \*s)

{

fprintf(stdout,"\n%s",s);

}

C2.l

%{

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

extern int yylval;

%}

%%

0 {yylval=0;return ZERO;}

1 {yylval=1;return ONE;}

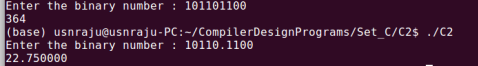
"." {return POINT;}

[ \t] {;}

\n return 0;

%%

Output:



**PROGRAM 3**

**Use YACC to implement, evaluator for arithmetic expressions (Desktop calculator)**

c3.y

%{

#include <stdio.h>

#include <ctype.h>

int x[5],y[5],k,j[5],a[5][10],e,w;

%}

%token digit

%%

S : E { printf("\nAnswer : %d\n",$1); }

;

E : T { x[e]=$1; } E1 { $$=x[e]; }

;

E1 : '+' T { w=x[e]; x[e]=x[e]+$2; printf("Addition Operation %d

and %d : %d\n",w,$2,x[e]); } E1 { $$=x[e]; }

| '-' T { w=x[e]; x[e]=x[e]-$2; printf("Subtraction Operation

%d and %d : %d\n",w,$2,x[e]); } E1 { $$=x[e]; }

| { $$=x[e]; }

;

T : Z { y[e]=$1; } T1 { $$=y[e]; }

;

T1 : '\*' Z { w=y[e]; y[e]=y[e]\*$2; printf("Multiplication

Operation of %d and %d : %d\n",w,$2,y[e]); } T1 { $$=y[e]; }

| { $$=y[e]; }

;

Z : F { a[e][j[e]++]=$1; } Z1 { $$=$3; }

;

Z1 : '^' Z { $$=$2; }

| { for(k=j[e]-1;k>0;k--) { w=a[e][k-1]; a[e][k1]=powr(a[e][k-1],a[e][k]); printf("Power Operation %d ^ %d :

%d\n",w,a[e][k],a[e][k-1]); } $$=a[e][0]; j[e]=0; }

;

F : digit { $$=$1; printf("Digit : %d\n",$1); }

| '(' { e++; } E { e--; } ')' { $$=$3; }

2

;

%%

int main()

{

for(e=0;e<5;e++) { x[e]=y[e]=0; j[e]=0; }

e=0;

printf("Enter an expression\n");

yyparse();

return 0;

}

yyerror()

{

printf("NITW Error");

}

int yywrap()

{

return 1;

}

int powr(int m,int n)

{

int ans=1;

while(n) { ans=ans\*m; n--; }

return ans;

}

C3.l

%{

#include "y.tab.h"

#include <stdlib.h>

extern int yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext);return digit;}

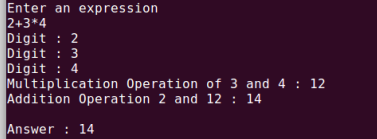
[\t] ;

[\n] return 0;

. return yytext[0];

%%

Output:



**PROGRAM 4**

**Use YACC to convert: Infix expression to Postfix expression.**

File: C4.y

%{

#include <ctype.h>

#include<stdio.h>

#include<stdlib.h>

%}

%token digit

%%

S: E {printf("\n\n");}

;

E: E '+' T { printf ("+");}

| E '-' T { printf ("-");}

| T

;

T: T '\*' P { printf("\*");}

| T '/' P { printf("/");}

| P

;

P: F '^' P { printf ("^");}

| F

;

F: '(' E ')'

| digit {printf("%d", $1);}

;

%%

int main()

{

printf("Enter infix expression: ");

yyparse();

}

yyerror()

{

printf("NITW Error");

}

C3.l

%{

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext); return digit;}

[\t] ;

[\n] return 0;

. return yytext[0];

%%

Output:



**PROGRAM 5**

**Use YACC to generate Syntax tree for a given expression**

C3.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,"+"); ; }

|E'-'T { $$= mknode($1,$3,"-"); ;}

|T { $$=$1; }

;

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

|T'/'F { $$= mknode($1,$3,"/"); ;}

|F {$$=$1 ; }

;

F:P'^'F { $$= mknode($1,$3,"^");}

| P { $$ = $1 ;}

;

P: '('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;

}

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",cur\_ind,syn\_tree[cur\_ind].val, syn\_tree[cur\_ind].lc,

syn\_tree[cur\_ind].rc);

my\_print\_tree(syn\_tree[cur\_ind].lc);

my\_print\_tree(syn\_tree[cur\_ind].rc);

}

C3.l

%{

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext); return digit;}

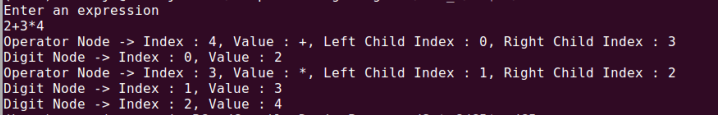
[\t] ;

[\n] return 0;

. return yytext[0];

%%

Output:



**PROGRAM 6**

**Use YACC to generate 3-Address code for a given expression**

C4.y

%{

#include <math.h>

#include<ctype.h>

#include<stdio.h>

int var\_cnt=0;

char iden[20];

%}

%token digit

%token id

%%

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:P '^' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d ^ t%d;\n", $$, $1, $3 );}

| P { $$ = $1;}

;

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

|digit { $$=var\_cnt; var\_cnt++; printf("t%d = %d;\n",$$,$1); }

;

2

%%

int main()

{

var\_cnt=0;

printf("Enter an expression : \n");

yyparse();

return 0;

}

yyerror()

{

printf("NITW Error\n");

}

C5.l

d [0-9]+

a [a-zA-Z]+

%{

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

extern int yylval;

extern char iden[20];

%}

%%

{d} { yylval=atoi(yytext); return digit; }

{a} { strcpy(iden,yytext); yylval=1; return id; }

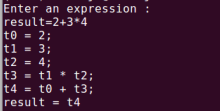
[ \t] {;}

\n return 0;

. return yytext[0];

%%

Output:



**PROGRAM 7**

**Use YACC to generate the 3-Address code which contains Arrays.**

C7.y

%{

#include <stdio.h>

#include <bits/stdc++.h>

#include <ctype.h>

using namespace std;

int yylex(void);

void yyerror(const char \*);

int n,i,j,an,nd[10],dim[10][10],can,r,inter;

int a[20],c[20],rednum,vn;

char b[20],name;

int size\_of\_datatype,sz;

int make\_variable();

%}

%token id

%%

/\* Final reduction printing. Split LHS and RHS and initiate reduction. \*/

S : id '=' E ';' { printf("After reduction number %d\n",rednum++); printf("%c =

t%d\n\n",$1,b[$3]-48); }

;

/\* If a '+' is encountered, split it into two halves and reduce it again. \*/

/\* If it is the last term, reduce it by taking it as T state. \*/

E : E '+' T { printf("After reduction number %d\n",rednum++);

i=make\_variable(); $$=i; c[i]=vn; b[i]=vn+48; vn++; printf("t%d =

",c[i]); if(a[$1]!=-1){printf("t%d + ",c[$1]);}

else { printf("%c + ",b[$1]); } if(a[$3]!=-

1){printf("t%d\n",c[$3]);} else { printf("%c\n",b[$3]); } }

| T { $$=$1; }

;

/\* T can be either a normal variable. id takes care of variables and if it is an

array, it will move to state L. \*/

T : id { printf("After reduction number %d\n",rednum++); i=make\_variable();

a[i]=-1; b[i]=$1; $$=i; }

| L { printf("After reduction number %d\n",rednum++); i=make\_variable(); $$=i;

c[i]=vn; b[i]=vn+48; vn++;

printf("t%d = %c[t%d]\n",c[i],name,c[$1]); can++; }

;

/\* The variable name of the array is received in the token id. \*/

/\* The index of the array can be an expression. Hence, recursively calling E to

reduce the index. \*/

/\* The second term is for multi dimensional arrays. \*/

L : id '[' E ']' { printf("After reduction number %d\n",rednum++);

name=$1; r=0; i=make\_variable(); $$=i; c[i]=vn; b[i]=vn+48;

vn++; printf("t%d = ",c[i]); if(a[$3]!=-1){printf("t%d",c[$3]);}

else { printf("%c",b[$3]); }

2

if(r+1!=nd[can]) { printf(" \*

%d",size\_of\_datatype\*dim[can][nd[can]-1-r]); }

else { printf(" \* %d",size\_of\_datatype); } r++; printf("\n");

}

| L '[' E ']' { printf("After reduction number %d\n",rednum++);

//inter=make\_variable();

inter=vn++; printf("t%d = ",inter); if(a[$3]!=-

1){printf("t%d",c[$3]);} else { printf("%c",b[$3]); }

if(r+1!=nd[can]) { printf(" \*

%d",size\_of\_datatype\*dim[can][nd[can]-1-r]); } else { printf(" \*

%d",size\_of\_datatype); }

r++; printf("\n");

i=make\_variable(); $$=i; c[i]=vn; b[i]=vn+48; vn++;

printf("t%d = t%d + t%d\n",c[i],c[$1],inter);

}

;

%%

int main()

{

rednum=1; vn=1;

printf("Enter size of data type : \n");

scanf("%d",&size\_of\_datatype);

printf("Enter no of arrays : \n");

scanf("%d",&an);

int y,l;

for(y=0;y<an;y++)

{

printf("Enter no of dimension of %d array : \n",y+1);

scanf("%d",&nd[y]);

printf("Enter dimensions of %d array : \n",y+1);

for(l=0;l<nd[y];l++)

{

scanf("%d",&dim[y][l]);

}

}

//an=1; nd[0]=2; dim[0][0]=2; dim[0][1]=3;

can=0;

int x=0;

for(x=0;x<20;x++) { a[i]=0; }

n=1;

printf("Enter Expression ending with Semicolon\n");

cin.ignore();

yyparse();

return 0;

}

int make\_variable()

{

return n++;

}

void yyerror(const char \*str)

3

{

printf("NITW Error occuring\n");

}

int yywrap()

{

return 1;

}

C7.l

%{

#include "y.tab.h"

#include <stdlib.h>

%}

d[0-9]

c[a-z]

extern char yylval;

/\*

Rules:

If an alphabet from a to z is matched, it is sent as a token.

If a tab character is encountered, nothing is done.

If a new line character is encountered, code stops running.

For anything else, the first character of the matched word is

sent as token.

\*/

%%

{c} { yylval=yytext[0]; return(id); }

[\t] ;

[\n] return 0;

. return yytext[0];

%%

Output:

