LABORATORY MANUAL

SREEPATHY INSTITUTE OF MANAGEMENT & TECHNOLOGY

VAVANOOR, PALAKKAD DIST

DEPT. OF COMPUTER SCIENCE & ENGINEERING

SUBJECT: CS431 COMPILER DESIGN LAB

FACULTY: SANEESH P. S./ RAJITHA M. R.

SEMESTER VII

AUGUST 2020

LIST OF EXPERIMENTS

EXP 1

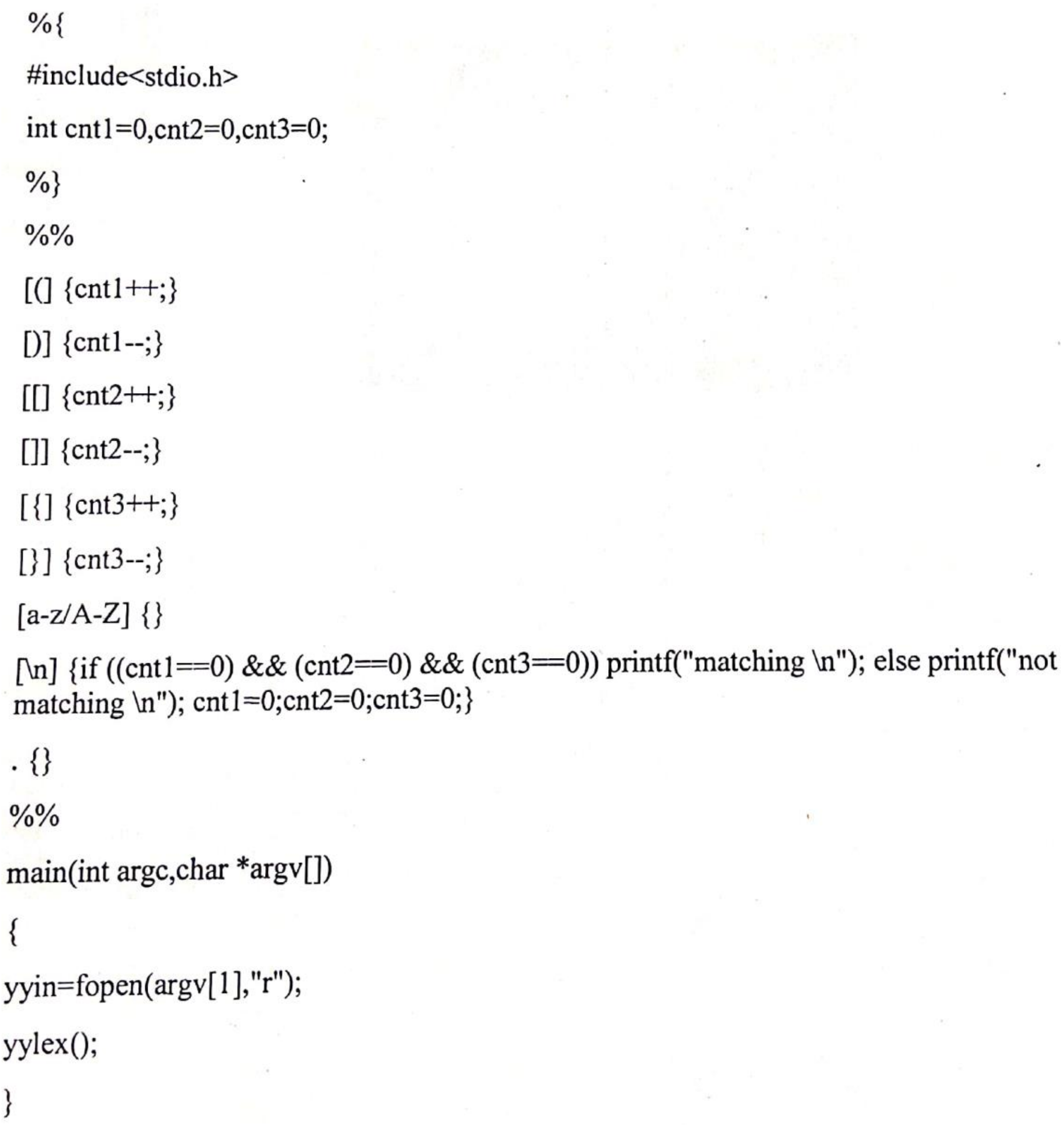
IMPLEMENTATION OF LEXICAL ANALYZER USING LEX TOOL

AIM

To implement lexical analyzer using LEX tool

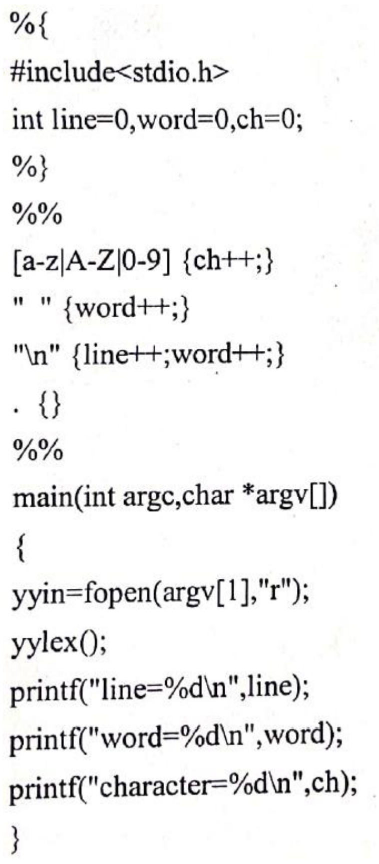
PROGRAMS

Program to verify parenthesis matching by LEX tool



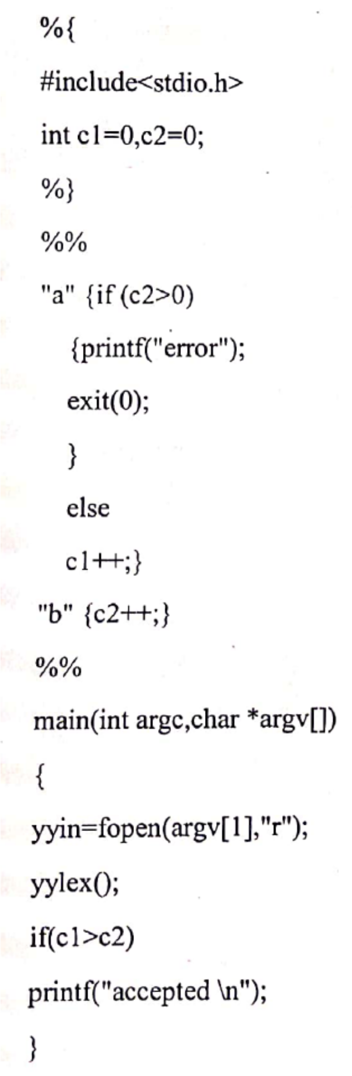
```
%{
#include<stdio.h>
int cnt1=0,cnt2=0,cnt3=0;
%}
%%
[(] {cnt1++;}
[)] {cnt1--;}
[[] {cnt2++;}
[]] {cnt2--;}
[{] {cnt3++;}
[}] {cnt3--;}
[a-z/A-Z] {}
[\n] {if ((cnt1==0) && (cnt2==0) && (cnt3==0)) printf("matching \n"); else printf("not
matching \n"); cnt1=0;cnt2=0;cnt3=0;}
. {}
%%
main(int argc,char *argv[])
{
yyin=fopen(argv[1],"r");
yylex();
}
```

Program to count lines, words and characters in a LEX file



```
%{
#include<stdio.h>
int line=0,word=0,ch=0;
%}
%%
[a-z|A-Z|0-9] {ch++;}
"  " {word++;}
"\n" {line++;word++;}
.  {}
%%
main(int argc,char *argv[])
{
yyin=fopen(argv[1],"r");
yylex();
printf("line=%d\n",line);
printf("word=%d\n",word);
printf("character=%d\n",ch);
}
```

Program to implement a^m b^n



```
%{
#include<stdio.h>
int c1=0,c2=0;
%}
%%
"a" {if (c2>0)
    {printf("error");
    exit(0);
    }
    else
    c1++;}
"b" {c2++;}
%%
main(int argc,char *argv[])
{
yyin=fopen(argv[1],"r");
yylex();
if(c1>c2)
printf("accepted \n");
}
```

RESULT

The programs executed successfully and output obtained

EXP 2

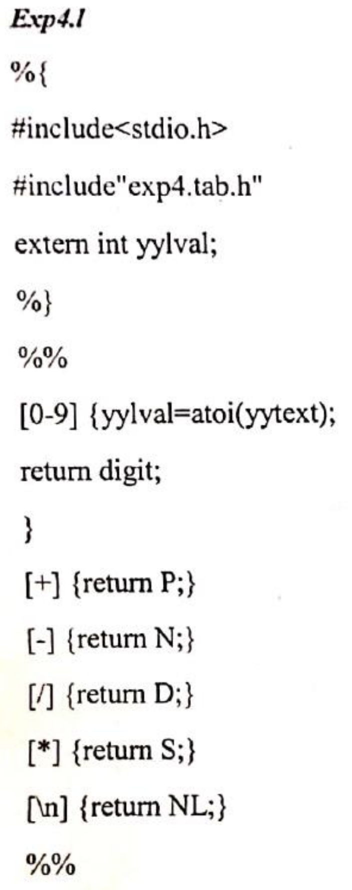
IMPLEMENTATION OF SYNTAX ANALYZER USING YACC

AIM

To implement syntax analyzer using YACC

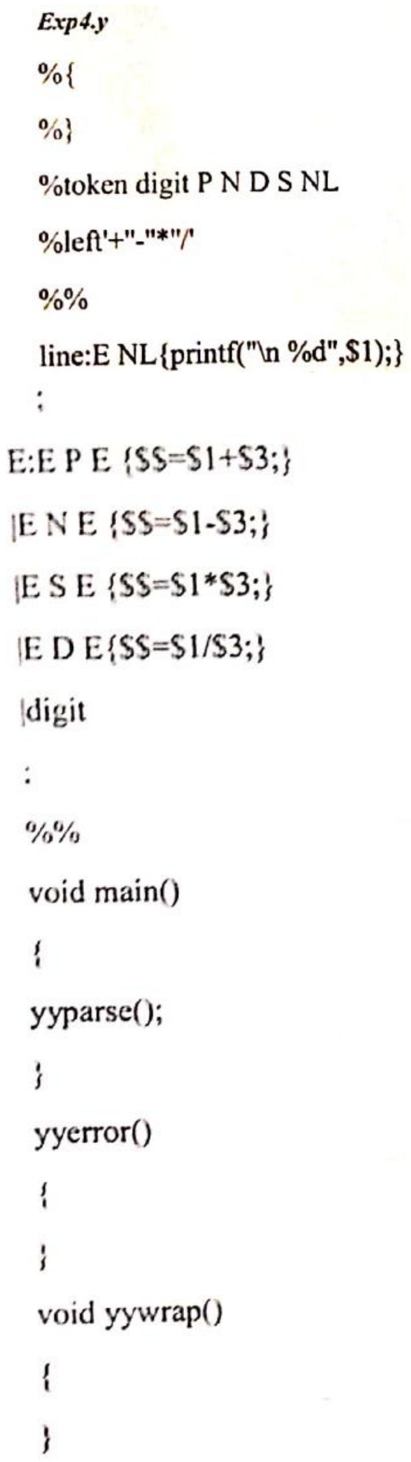
PROGRAMS

Program to implement Desktop Calculator



Exp4.1

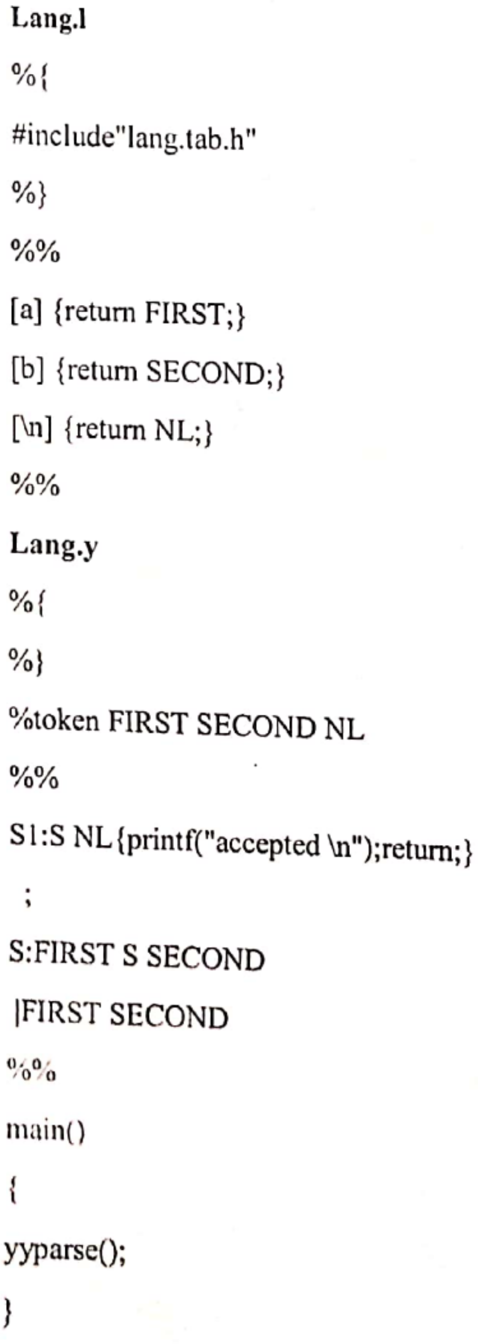
```
%{
#include<stdio.h>
#include"exp4.tab.h"
extern int yylval;
%}
%%
[0-9] {yylval=atoi(yytext);
return digit;
}
[+] {return P;}
[-] {return N;}
[/] {return D;}
[*] {return S;}
[\n] {return NL;}
%%
```



Exp4.y

```
 %{
 %}
 %token digit P N D S NL
 %left'+"-"*"/'
 %%
 line:E NL{printf("\n %d",$1);}
 ;
E:E P E {$$=$1+$3;}
|E N E {$$=$1-$3;}
|E S E {$$=$1*$3;}
|E D E{$$=$1/$3;}
|digit
 ;
 %%
 void main()
 {
 yyparse();
 }
 yyerror()
 {
 }
 void yywrap()
 {
 }
```

Program to implement a^n b^n where n>1



Lang.l

```
%{
#include"lang.tab.h"
%}
%%
[a] {return FIRST;}
[b] {return SECOND;}
[\n] {return NL;}
%%
```

Lang.y

```
%{
%}
%token FIRST SECOND NL
%%
S1:S NL{printf("accepted \n");return;}
  ;
S:FIRST S SECOND
 |FIRST SECOND
%%
main()
{
yyparse();
}
```

RESULT

The program executed successfully and output obtained

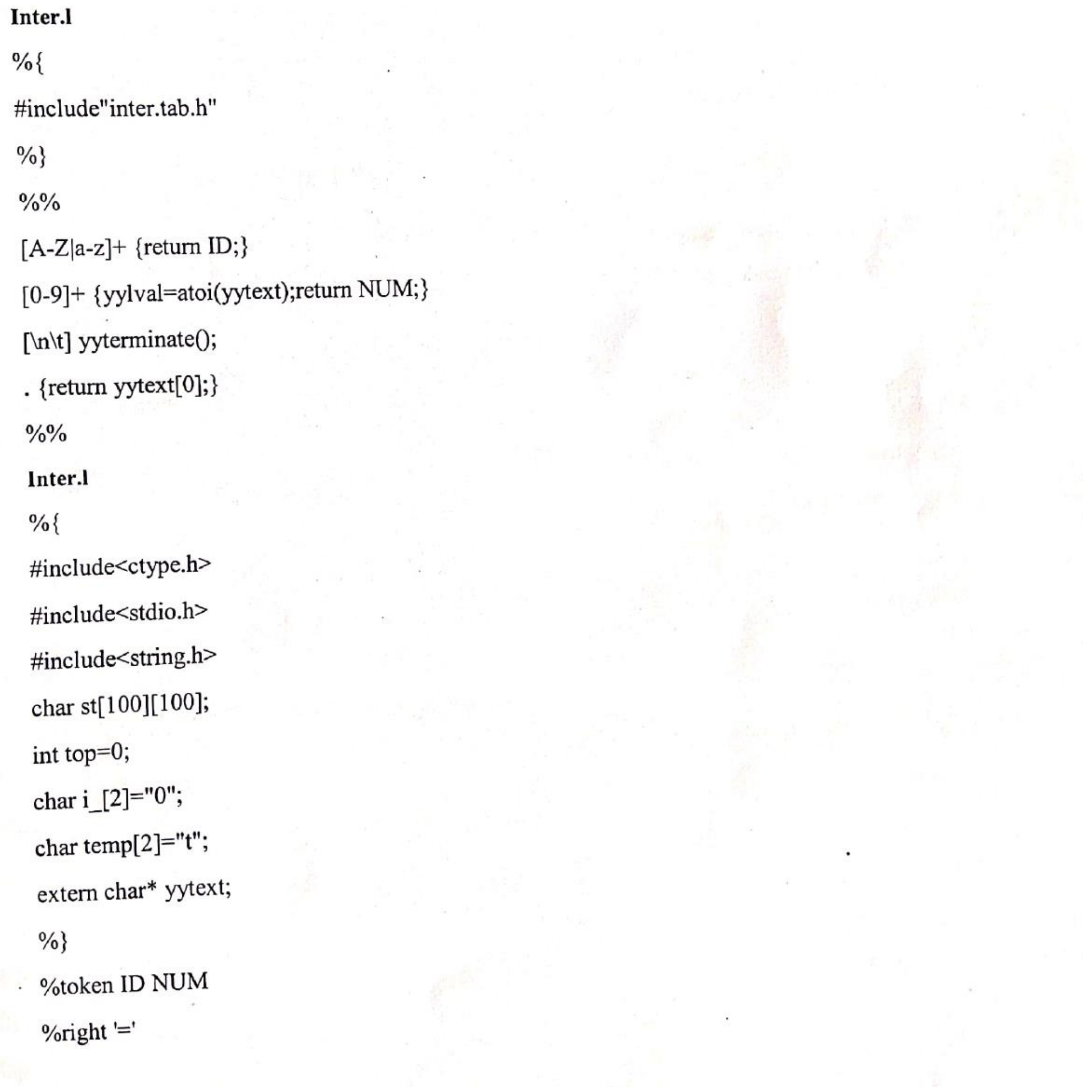
EXP 3

INTERMEDIATE CODE GENERATION

AIM

To implement intermediate code generation

PROGRAM

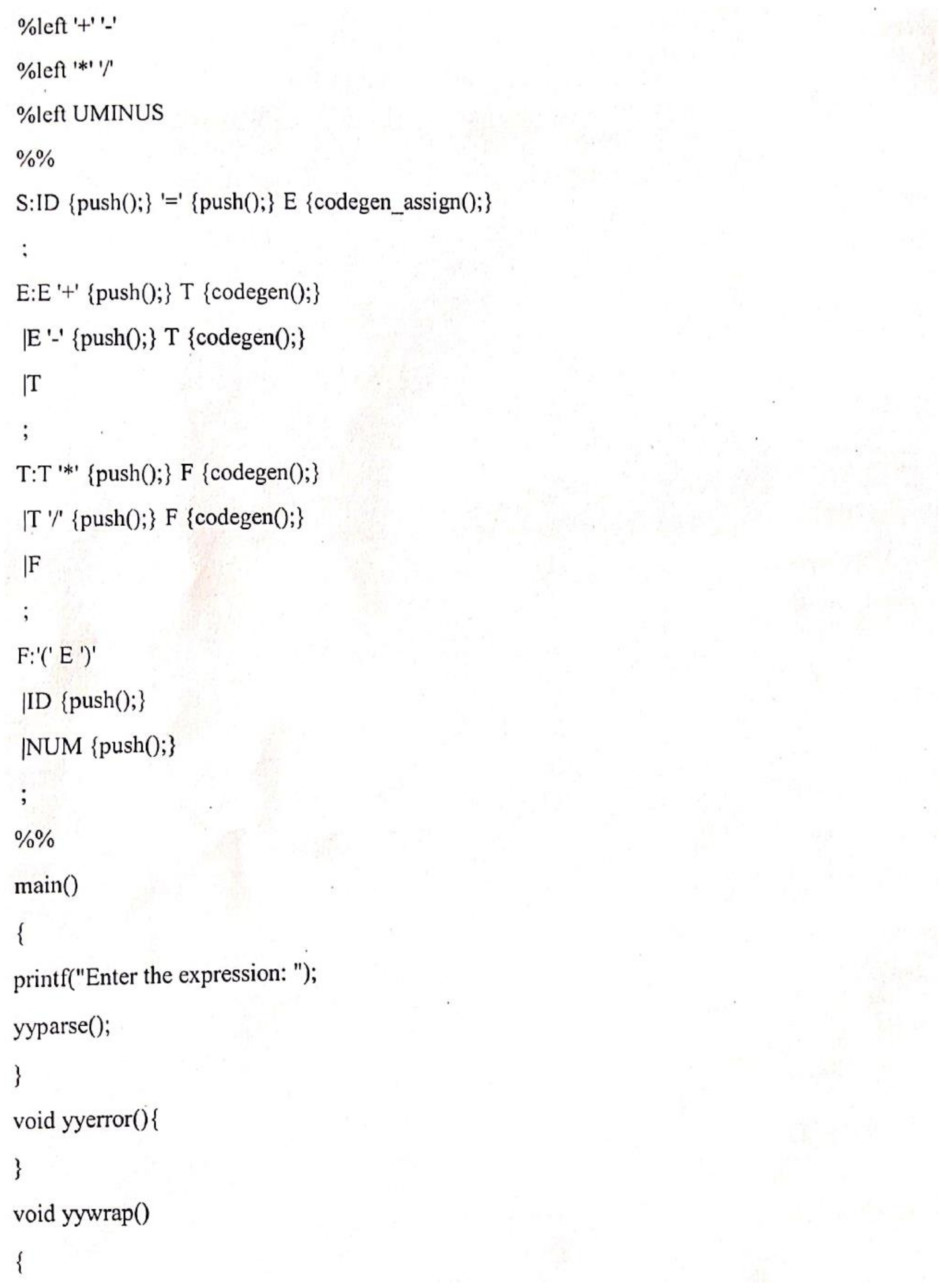


Inter.l

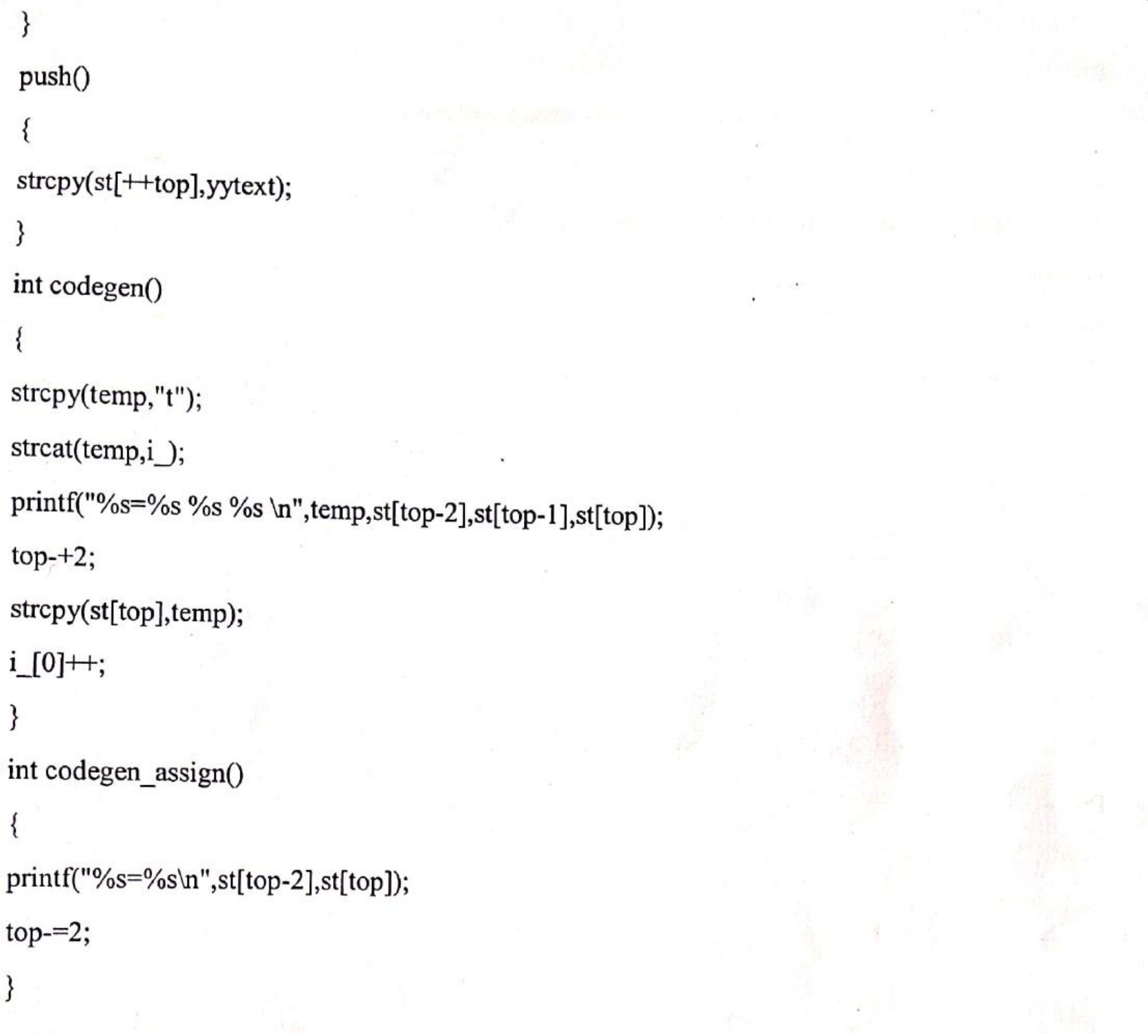
```
%{
#include"inter.tab.h"
%}
%%
[A-Z|a-z]+ {return ID;}
[0-9]+ {yylval=atoi(yytext);return NUM;}
[\n\t] yyterminate();
. {return yytext[0];}
%%
```

Inter.l

```
%{
#include<ctype.h>
#include<stdio.h>
#include<string.h>
char st[100][100];
int top=0;
char i_[2]="0";
char temp[2]="t";
extern char* yytext;
%}
%token ID NUM
%right '='
```



```
%left '+' '-'
%left '*' '/'
%left UMINUS
%%
S:ID {push();} '=' {push();} E {codegen_assign();}
 ;
E:E '+' {push();} T {codegen();}
 |E '-' {push();} T {codegen();}
 |T
 ;
T:T '*' {push();} F {codegen();}
 |T '/' {push();} F {codegen();}
 |F
 ;
F:'(' E ')'
 |ID {push();}
 |NUM {push();}
 ;
%%
main()
{
printf("Enter the expression: ");
yyparse();
}
void yyerror(){
}
void yywrap()
{
```



```
}
push()
{
strcpy(st[++top],yytext);
}
int codegen()
{
strcpy(temp,"t");
strcat(temp,i_);
printf("%s=%s %s %s \n",temp,st[top-2],st[top-1],st[top]);
top-+2;
strcpy(st[top],temp);
i_[0]++;
}
int codegen_assign()
{
printf("%s=%s\n",st[top-2],st[top]);
top-=2;
}
```

RESULT

The program executed successfully and output obtained

EXP 4

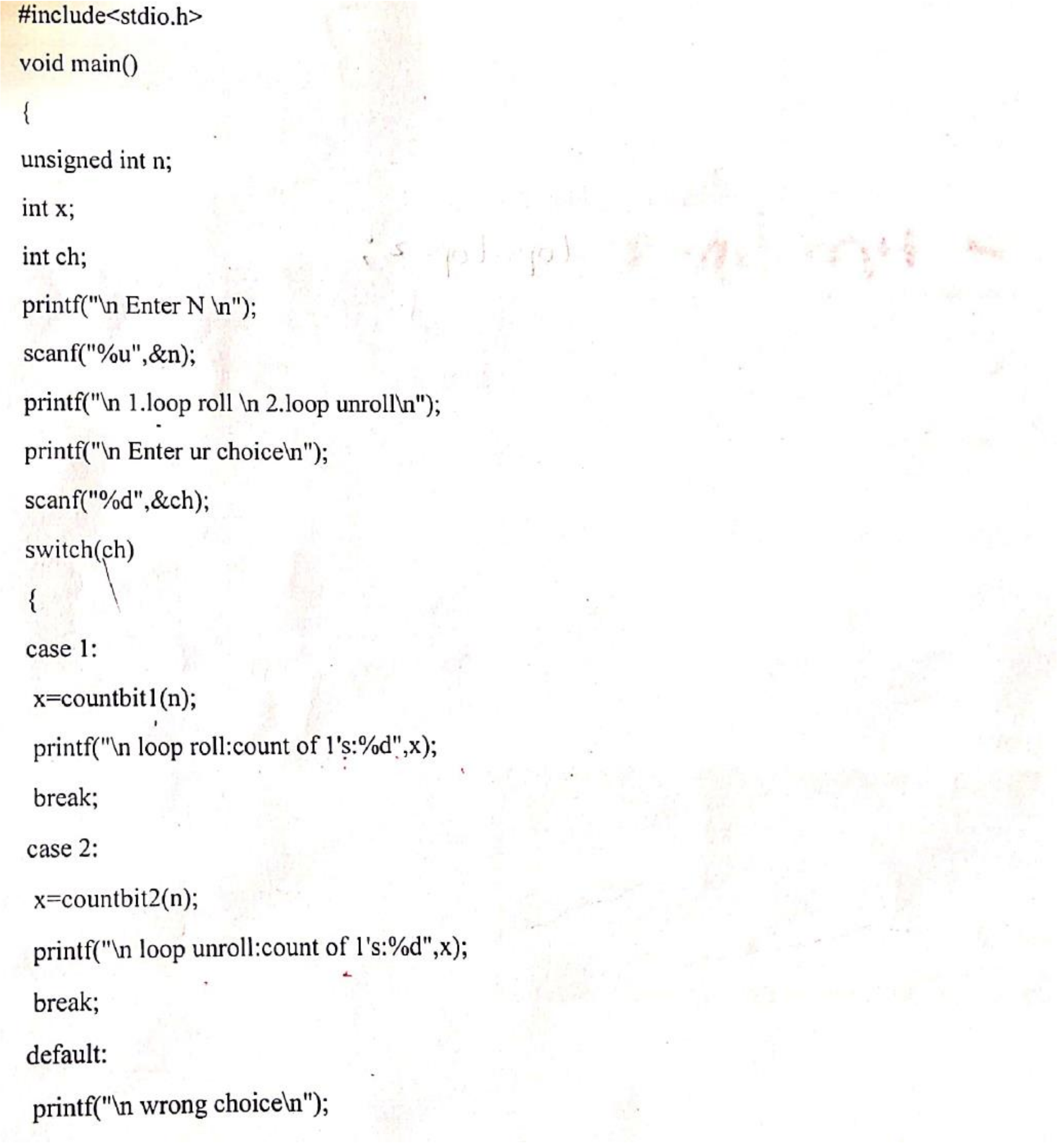
CODE OPTIMIZATION

AIM

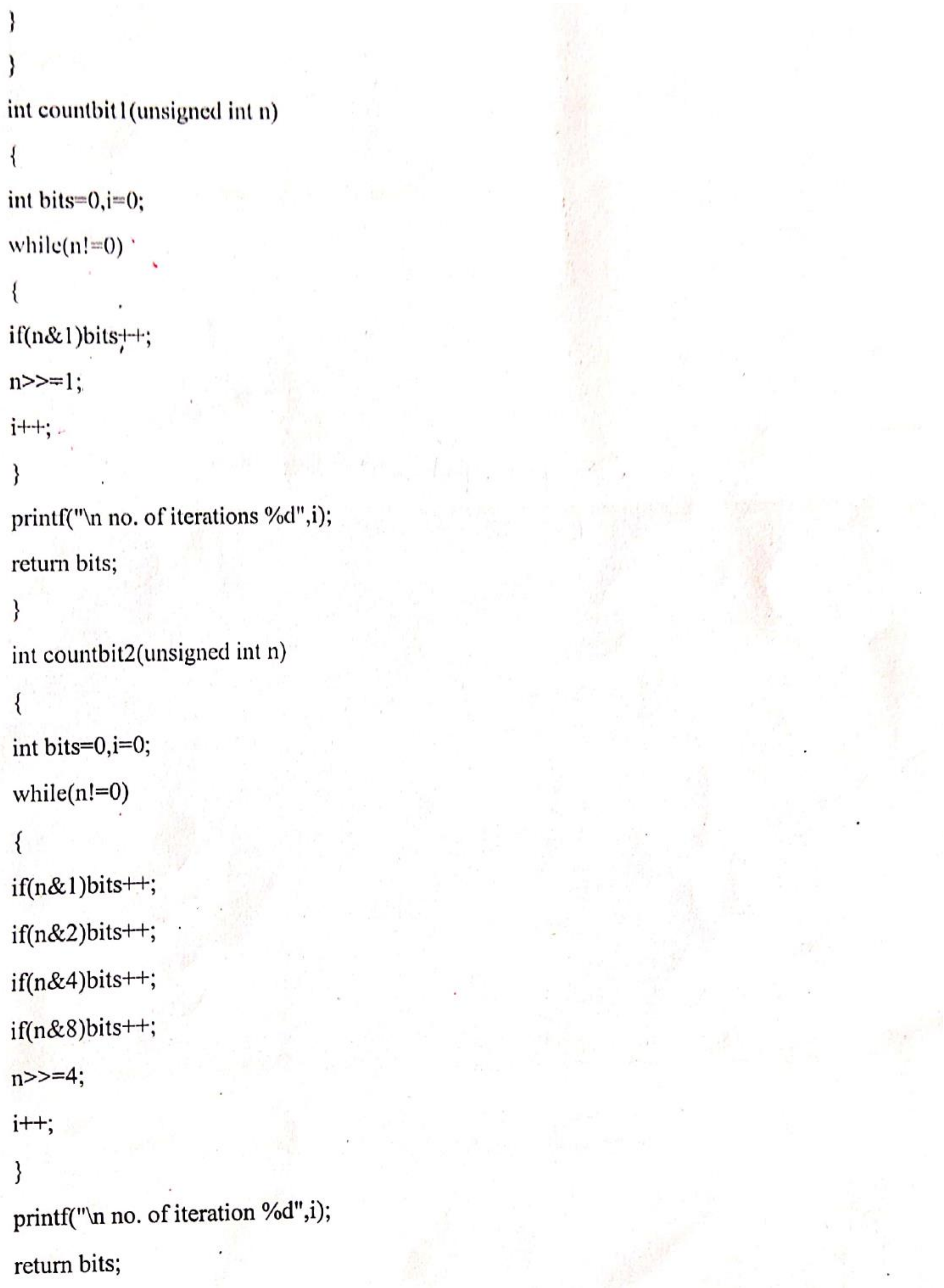
To perform code optimization

PROGRAM

Program to perform loop rolling and loop unrolling



```
#include<stdio.h>
void main()
{
unsigned int n;
int x;
int ch;
printf("\n Enter N \n");
scanf("%u",&n);
printf("\n 1.loop roll \n 2.loop unroll\n");
printf("\n Enter ur choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
 x=countbit1(n);
 printf("\n loop roll:count of 1's:%d",x);
 break;
case 2:
 x=countbit2(n);
 printf("\n loop unroll:count of 1's:%d",x);
 break;
default:
 printf("\n wrong choice\n");
```



```
}
}
int countbit1(unsigned int n)
{
int bits=0,i=0;
while(n!=0)
{
if(n&1)bits++;
n>>=1;
i++;
}
printf("\n no. of iterations %d",i);
return bits;
}
int countbit2(unsigned int n)
{
int bits=0,i=0;
while(n!=0)
{
if(n&1)bits++;
if(n&2)bits++;
if(n&4)bits++;
if(n&8)bits++;
n>>=4;
i++;
}
printf("\n no. of iteration %d",i);
return bits;
```

RESULT

The program executed successfully and output obtained

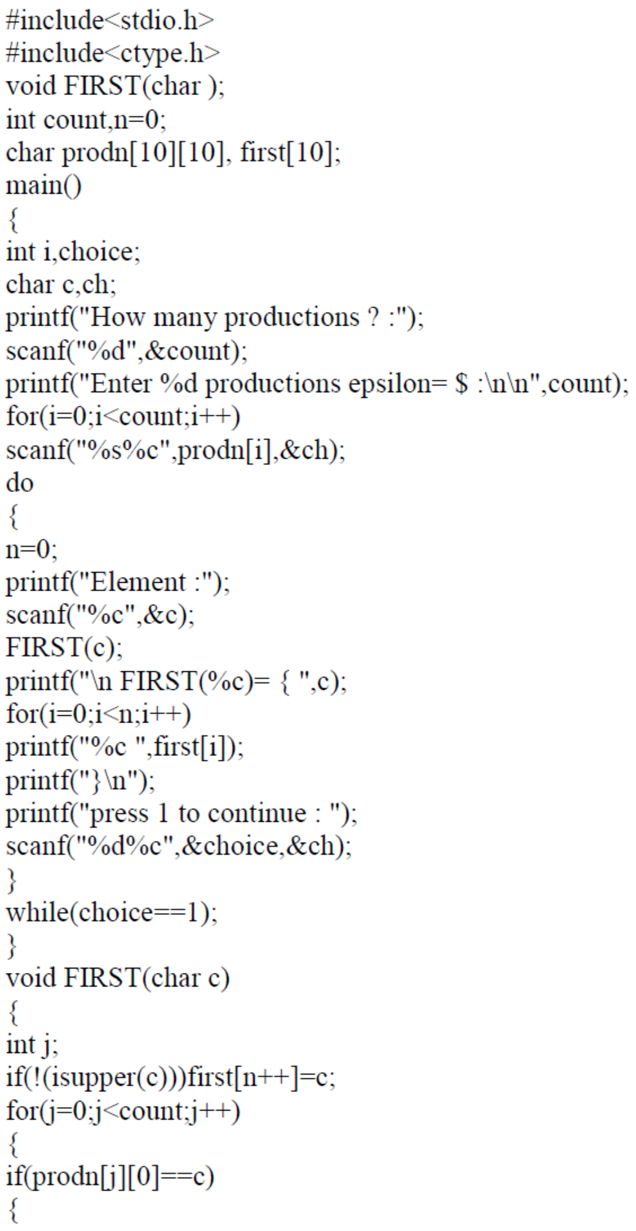
EXP 5

FINDING FIRST OF A GRAMMAR

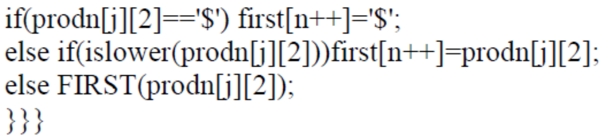
AIM

To find the first of a grammar

PROGRAM



```
#include<stdio.h>
#include<ctype.h>
void FIRST(char );
int count,n=0;
char prodn[10][10], first[10];
main()
{
int i,choice;
char c,ch;
printf("How many productions ? :");
scanf("%d",&count);
printf("Enter %d productions epsilon= $ :\n\n",count);
for(i=0;i<count;i++)
scanf("%s%c",prodn[i],&ch);
do
{
n=0;
printf("Element :");
scanf("%c",&c);
FIRST(c);
printf("\n FIRST(%c)= { ",c);
for(i=0;i<n;i++)
printf("%c ",first[i]);
printf("}\n");
printf("press 1 to continue : ");
scanf("%d%c",&choice,&ch);
}
while(choice==1);
}
void FIRST(char c)
{
int j;
if(!(isupper(c)))first[n++]=c;
for(j=0;j<count;j++)
{
if(prodn[j][0]==c)
{
```



```
if(prodn[j][2]=='$') first[n++]='$';
else if(islower(prodn[j][2]))first[n++]=prodn[j][2];
else FIRST(prodn[j][2]);
}}}
```

RESULT

The program executed successfully and output obtained

EXP 6

FINDING FOLLOW OF A CONTEXT FREE GRAMMAR

AIM

To find the follow of a context free grammar

PROGRAM

```
#include<stdio.h>
#include<string.h>
int n,m=0,p,i=0,j=0;
char a[10][10],f[10];
void follow(char c);
void first(char c);
int main()
{
 int i,z;
 char c,ch;
 printf("Enter the no.of productions:");
 scanf("%d",&n);
 printf("Enter the productions(epsilon=$):\n");
 for(i=0;i<n;i++)
  scanf("%s%c",a[i],&ch);

 do
 {
  m=0;
  printf("Enter the element whose FOLLOW is to be found:");

  scanf("%c",&c);
  follow(c);
  printf("FOLLOW(%c) = { ",c);
  for(i=0;i<m;i++)
   printf("%c ",f[i]);
  printf(" }\n");
  printf("Do you want to continue(0/1)?");
  scanf("%d%c",&z,&ch);
 }
 while(z==1);
}
void follow(char c)
```

```
{

 if(a[0][0]==c)f[m++]='$';
 for(i=0;i<n;i++)
 {
  for(j=2;j<strlen(a[i]);j++)
  {
   if(a[i][j]==c)
   {
    if(a[i][j+1]!='\0')first(a[i][j+1]);

    if(a[i][j+1]=='\0'&&c!=a[i][0])
     follow(a[i][0]);

   }
  }
 }
}
void first(char c)
{
      int k;
                 if(!(isupper(c)))f[m++]=c;
                 for(k=0;k<n;k++)
                 {
                 if(a[k][0]==c)
                 {
                 if(a[k][2]=='$') follow(a[i][0]);
                 else if(islower(a[k][2]))f[m++]=a[k][2];
                 else first(a[k][2]);
                 }
                 }

}
```

RESULT

The program executed successfully and output obtained

EXP 7

SYMBOL TABLE IMPLEMENTATION

AIM

To implement symbol table

PROGRAM

```
%{

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

#include"name.tab.h"

extern char* yylval;

int x=0;

%}

%%

"int" {x++;return INT;}

"float" {x++;return FLOAT;}

"double" {x++;return DOUBLE;}

"char" {x++;return CHAR;;}

[a-z]+ {yylval=yytext;if(x>0)return ID;return O;}

[\n] {return NL;}

"," {return C;}

";" {x--;return SE;}

. {}
%%


%{

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


int fl=0,i=0,type[100],j=0,error_flag=0;

char symbol[100][100],temp[100];

%}

%token INT FLOAT C DOUBLE CHAR ID NL SE O

%%

START:S1 NL {return;}

;

S1:S NL S1

|S NL

;

S:INT L1 E

|FLOAT L2 E

|DOUBLE L3 E

|CHAR L4 E

|INT L1 E S

|FLOAT L2 E S

|DOUBLE L3 E S

|CHAR L4 E S

|O

;
L1:L1 C ID {strcpy(temp,(char *)$3);insert(0);}

|ID {strcpy(temp,(char *)$1);insert(0);}

;

L2:L2 C ID {strcpy(temp,(char *)$3);insert(1);}
```

```
|ID {strcpy(temp,(char *)$1);insert(1);}

;

L3:L3 C ID {strcpy(temp,(char *)$3);insert(2);}

|ID {strcpy(temp,(char *)$1);insert(2);}

;

L4:L4 C ID {strcpy(temp,(char *)$3);insert(3);}

|ID {strcpy(temp,(char *)$1);insert(3);}

;

E:SE

;

%%

main()

{

yyparse();

if(error_flag==0)

for(j=0;j<i;j++)
{

            if(type[j]==0)

            printf(" INT - ");

            if(type[j]==1)

            printf(" FLOAT - ");

            if(type[j]==2)

            printf(" DOUBLE - ");

            if(type[j]==3)

            printf(" CHAR - ");
```

```
        printf(" %s\n",symbol[j]);

}

}

void yyerror()

{ printf("SYNTAX ERROR\n");

error_flag=1;

}

void insert(int type1)

{


for(j=0;j<i;j++)
{
        if(strcmp(temp,symbol[j])==0)

        {

                if(type[i]==type1)

                        printf("REDECLARATION OF %s\n",temp);
                else
                        printf("MULTIPLE DECLARATION OF %s\n",temp);

                error_flag=1;
        }

}

if(error_flag==0)
{

        strcpy(symbol[i],temp);
        type[i]=type1;
        i++;
}
}
```

RESULT

The program executed successfully and output obtained

EXP 8

TYPE CHECKING

AIM

To perform type checking

PROGRAM

```
%{

#define YYSTYPE char*

#include<stdio.h>

#include "typecheck.tab.h"


%}

%%

"int" {return INT;}

"char" {return CHAR;}

[a-z]+ { yylval =yytext;return ID;}

"," {return C;}

"\n" {return NL;}

";" {return SE;}

"+" {return P;}

"-" {return M;}

"/" {return D;}

"*" {return ST;}

"=" {return EQ;}

. {}

%%
```

```
%{
#define YYSTYPE char*
#include<stdio.h>
#include<string.h>
void yyerror(char*);
int i,j,type[100],error_flag=0,sym_entries=0,char_flag=0,cnt=0;
char symbol[100][100],t[100][100],temp[100];
%}
%token INT CHAR C ID NL SE P M D ST EQ
%left '+' '-' '/' '*'
%%
START:S{return;};
S:      INT L1 E NL S
        |CHAR L2 E NL S
        |EXPR E NL;
L1:     L1 C ID{strcpy(temp,(char*)$3);insert(0);}
        | ID {strcpy(temp,(char*)$1);insert(0);};
L2:     L2 C ID{strcpy(temp,(char*)$3);insert(1);}
        | ID {strcpy(temp,(char*)$1);insert(1);};
EXPR    :EXP EQ EXP P ID {strcpy(t[cnt++],$5);     check();}
        |EXP EQ EXP M ID {strcpy(t[cnt++],$5); check();}
    |EXP EQ EXP D ID {strcpy(t[cnt++],$5); check();}
        |EXP EQ EXP ST ID  {strcpy(t[cnt++],$5); check();};
EXP     :ID {strcpy(t[cnt++],$1);};
```

```
E:SE;

%%

main()

{

        yyparse();

        if(error_flag)

                printf("\ntype error\n");

        else

                printf("\nNo type error\n");

}

yyerror(char *s)

{

        printf("Error : %s",s);

}

yywrap(){};

insert(int type1)

{

        type[sym_entries]=type1;

        strcpy(symbol[sym_entries],temp);

        sym_entries++;

}

check()

{

        int t1,t2,t3;

        for(i=0;i<sym_entries;i++)

        {
                if(strcmp(t[0],symbol[i])==0)

                        t1=type[i];

                if(strcmp(t[1],symbol[i])==0)

                        t2=type[i];

                if(strcmp(t[2],symbol[i])==0)

                        t3=type[i];

        }

        if(t2==1 || t3==1 || t1==1)

                error_flag==1;

        if(t1!=t2 || t1!=t3 || t2!=t3)

                error_flag=1;

}
```

RESULT

The program executed successfully and output obtained

EXP 9

NFA TO DFA CONVERSION USING JFLAP

AIM

To convert NFA to DFA using JFLAP

THEORETICAL BACKGROUND

Introduction

JFLAP program makes it possible to create and simulate automata. Learning about automata with pen and paper can be difficult, time consuming and error-prone. With JFLAP we can create automata of different types and it is easy to change them as we want. JFLAP supports creation of DFA and NFA, Regular Expressions, PDA, Turing Machines, Grammars and more.

Setup

JFLAP is available from the homepage: (www.JFLAP.org). From there press “Get FLAP” and follow the instructions. You will notice that JFLAP have a .JAR extension. This means that you need Java to run JFLAP. With Java correctly installed you can simply select the program to run it. You can also use a command console run it from the files current directory with, *Java –jar JFLAP.jar*.

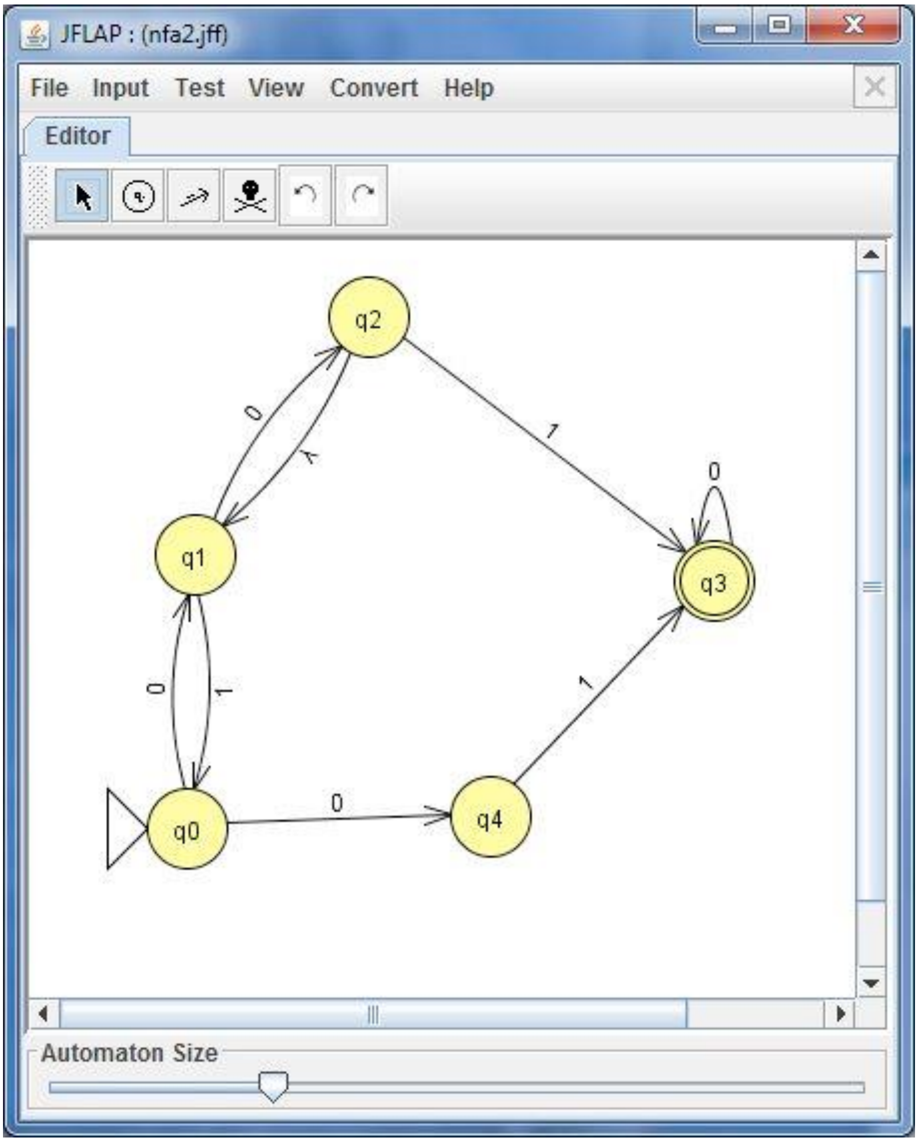
Using JFLAP

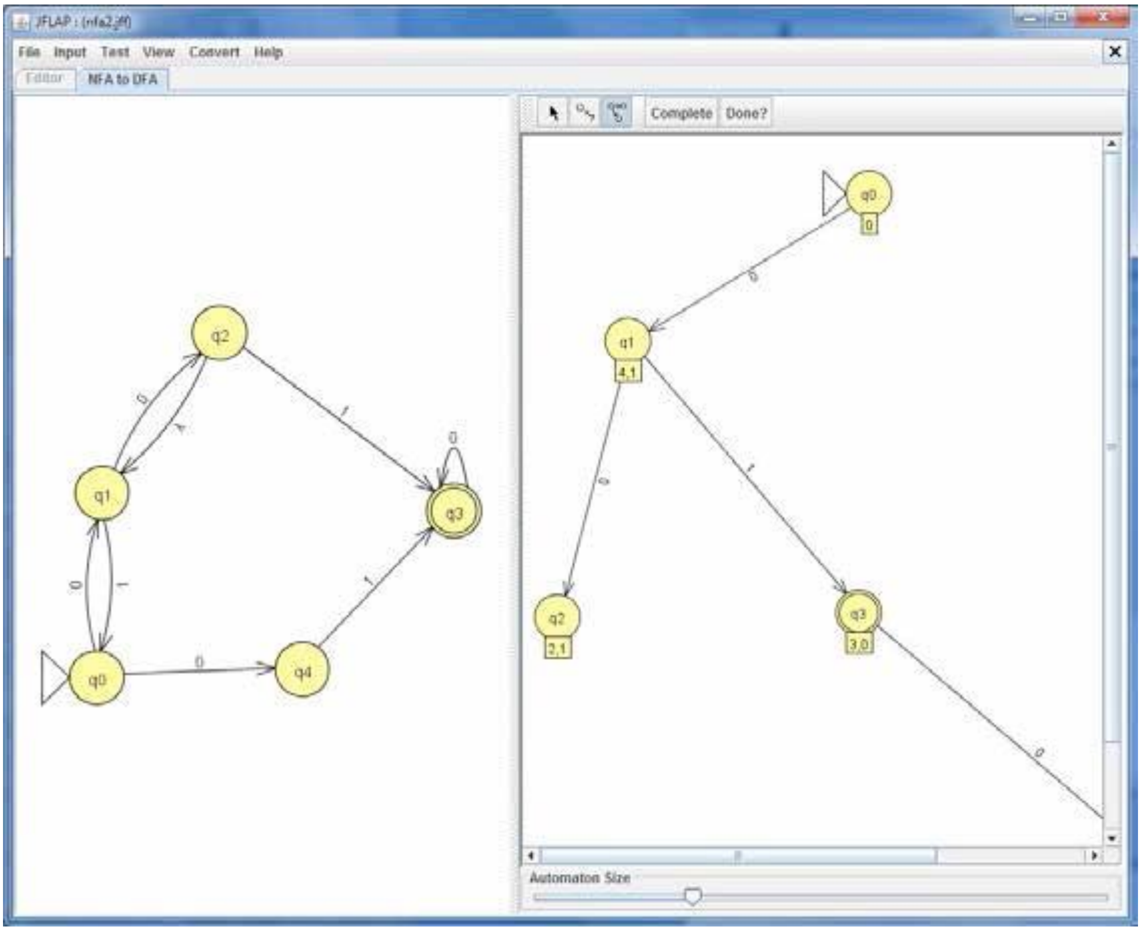
When you first start JFLAP you will see a small menu with a selection of eleven different automata and rule sets. Choosing one of them will open the editor where you create chosen type of automata. Usually you can create automata containing states and transitions but there is also creation of Grammar and Regular Expression which is made with a text editor.

Converting NFA to DFA

Procedure

Create a NFA and then choose Convert → Convert to DFA. This will open the conversion view where you either let JFLAP do the work or try yourself to convert it. The left view is the original automaton and the right one is the new DFA. Use the state expander tool to expand the states until the DFA is complete. Using the *Complete* button will automatically create the whole DFA for you. The *Done?* Button will tell if the DFA is done or not. Once the DFA is complete it will be exported to a new JFLAP window with your converted DFA.





RESULT

The program executed successfully and output obtained

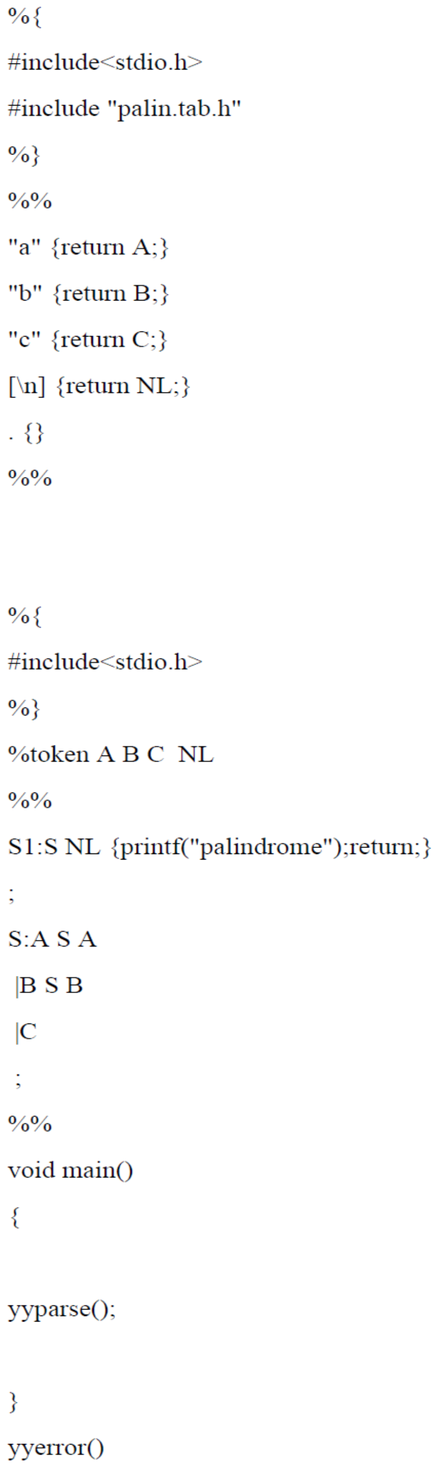
EXP 10

PALINDROME CHECKING

AIM

To check whether a string is palindrome or not

PROGRAM



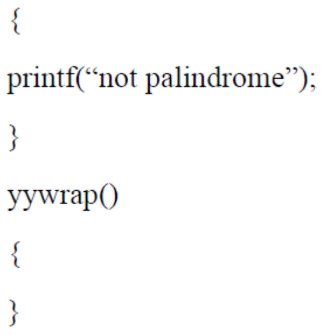
```
%{
#include<stdio.h>
#include "palin.tab.h"
%}
%%
"a" {return A;}
"b" {return B;}
"c" {return C;}
[\n] {return NL;}
. {}
%%



%{
#include<stdio.h>
%}
%token A B C  NL
%%
S1:S NL {printf("palindrome");return;}
;
S:A S A
 |B S B
 |C
 ;
%%
void main()
{

yyparse();

}
yyerror()
```



```
{
printf("not palindrome");
}
yywrap()
{
}
```

RESULT

The program executed successfully and output obtained

EXP 11

RECURSIVE DESCENT PARSER

AIM

To implement a recursive descent parser

PROGRAM

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

char input[10];
int i,error;
void E();
void T();
void Eprime();
void Tprime();
void F();
        main()
        {
i=0;
error=0;
             printf("Enter an arithmetic expression   :  "); // Eg: a+a*a
             gets(input);
             E();
             if(strlen(input)==i&&error==0)
                    printf("\nAccepted..!!!\n");
             else printf("\nRejected..!!!\n");
                      }



void E()
{
    T();
    Eprime();
}
void Eprime()
{
    if(input[i]=='+')
    {
    i++;
    T();
    Eprime();
    }
    }
void T()
{
    F();
    Tprime();
}
void Tprime()
{
    if(input[i]=='*')
    {
                i++;
                F();
                Tprime();
                }
                }
    void F()
    {
        if(isalnum(input[i]))i++;
        else if(input[i]=='(')
        {
        i++;
        E();
        if(input[i]==')')
        i++;

        else error=1;
          }

        else error=1;
        }
```

RESULT

Program executed successfully and output obtained

EXP 12

IMPLEMENTATION OF BACKEND OF A COMPILER

AIM

To implement backend of a compiler

PROGRAM

Back.l

```
%{
#include “back tab.h”
%}
%%
[a-z]+          {return ID;}
[\n]            {return NL;}
%%
back.y
%{
char m[10],n[10],y[10];
%}
%%
S:      ID {yy.text} ”=”
        ID {n=yy.text;} ‘+’
        ID {y=yy.text} NL
{
printf(“mov n,R0);
printf(“mov y,R1);
printf(“add R0,R1);
printf( “mov R1,m);
}
main( )
{
yy.parse( );
}
```

RESULT

Program executed successfully and output obtained