

COMPILER DESIGN LAB

LAB MANUAL

Prepared By
Mr.G.SAIDULU
Assistant Professor

Department of Computer Science & Engineering
CMR COLLEGE OF ENGINEERING & TECHNOLOGY
Medchal – 501 401, Hyderabad

1.1 OBJECTIVE

AIM: Design a DFA to accept all strings containing a substring(01) .

CODE:

```
#include <stdio.h>
#define max 100
main()
{
    char str[max],f='a'; int i;
    printf("enter the string to be checked: ");
    scanf("%s",str);
    for(i=0;str[i]!='\0';i++)
    {
        switch(f)
        {
            case 'a': if(str[i]=='0') f='b';
            else if(str[i]=='1') f='a'; break;
            case 'b': if(str[i]=='0') f='b';
            else if(str[i]=='1') f='c'; break;
            case 'c': if(str[i]=='0') f='b';
            else if(str[i]=='1') f='a'; break;
        }
    }
    if(f=='c')
        printf("\nString is accepted as it reached the final state %c at the end.",f);
    else
        printf("\nString is not accepted as it reached %c which is not the final state.",f);
    return 0;
}
```

OUTPUT:

Enter the String: 0010
Given string is accepted.

OBJECTIVE

AIM:

Design a DFA to accept all strings containing string starting with (01)

CODE:

```
#include <stdio.h>
#define max 100
main()
{
    char str[max],f='a'; int i;
    printf("enter the string to be checked: "); scanf("%s",str);
    for(i=0;str[i]!='\0';i++)
    {
        switch(f)
        {
            case 'a': if(str[i]=='0') f='b';
            else if(str[i]=='1') f='d'; break;
            case 'b': if(str[i]=='0') f='d';
            else if(str[i]=='1') f='c'; break;
            case 'c': if(str[i]=='0') f='c';
            else if(str[i]=='1') f='c'; break;
            case 'd': if(str[i]=='0') f='d';
            else if(str[i]=='1') f='d'; break;

        }
    }
    if(f=='c')
        printf("\nString is accepted as it reached the final state %c at the end.",f);
    else
        printf("\nString is not accepted as it reached %c which is not the final state.",f);
    return 0;
}
```

OUTPUT:

1).Enter the String: 01000100
Given string is accepted.

2). Enter the string:
11101010
Given string is no accepted.

OBJECTIVE:

Write a LEX Program to scan reserved word & Identifiers of C Language.

CODE:

```
for [f][o][r] letter [A-Za-z]
digit [0-9]
%{
int f;
% }
%%
{for} {return(1);}
{letter}({letter}|{digit})* {return(2);}
{digit}+ {return(3);}
%%
```

```
int main(void)
{
while(f=yylex())
{
if(f==1)
printf("given string keyword\n"); else
if(f==2)
printf("given string identifier\n"); else
if(f==3)
printf("given string number\n");
}
return 0;
}
```

OUTPUT:

```
vi filename.l lex filename.l
cc lex.yy.c -ll
./a.out
```

```
for
given string is keyword.
```

```
abc123@123
given string is identifier.
```

```
520
Given string is number.
```

AIM:

Write a LEX Program to scan integers as Float Numbers in C Language.

CODE:

```
digit [0-9] sign [+ -] e [E]
decimal [.]
%{
int f;
%}
%%
{digit}+ {return(1);} ({digit})+{decimal}({digit})+{E}{sign}?({digit})+ {return(2);}
({digit})+{decimal}({digit})+ {return(3);}
%%
int main(void)
{
while(f=yylex())
{
if(f==1) printf("digit\n"); else if(f==2)
printf("exponent\n"); else printf("float\n");
}
}
```

OUTPUT:

```
vi filename.l lex filename.l cc lex.yy.c
-ll
./a.out

25
digit 23.54
float 2.4E+5
Exponent
```

EXPERIMENT 1:

AIM: Write a C/C++ program to Implement Lexical Analyzer.

CODE:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
void keyw(char *p);
int i=0,id=0,kw=0,num=0,op=0;
char
keys[32][10]={ "auto","break","case","char","const","continue","default","do","double","else","enum","extern","float",
",","for","goto","if","int","long","register","return","short","signed","sizeof","static","struct","switch","typedef","union",
",","unsigned","void","volatile","while"};
main()
{
char ch,str[25],seps[15]=" \t\n,;(){}[]#\"<>\",oper[]="!%^&*-+=~|.<>/?";
int j;
char fname[50];
FILE *f1;
printf("enter file path (drive:\\fold\\filename)\n");
scanf("%s",fname);
f1 = fopen(fname,"r");
if(f1==NULL)
{
printf("file not found");
exit(0);
}
while((ch=fgetc(f1))!=EOF)
{
for(j=0;j<=14;j++)
{
if(ch==oper[j])
{
printf("%c is an operator\n",ch);
op++;
str[i]='\0';
keyw(str);
}
}
}
for(j=0;j<=14;j++)
{
if(i==1)
break;
if(ch==seps[j])
{
if(ch=='#')
{
while(ch!='>')
{
```

```

        printf("%c",ch);
        ch=fgetc(f1);
    }
    printf("%c is a header file\n",ch);
    i=-1;
    break;
}
if(ch=="")
{
    do
    {
        ch=fgetc(f1);
        printf("%c",ch);
    }while(ch!="");
    printf("\b is an argument\n");
    i=-1;
    break;
}
str[i]='\0';
keyw(str);
}
}
if(i!=-1)
{
    str[i]=ch;
    i++;
}
else
i=0;
}
printf("Keywords: %d\nIdentifiers: %d\nOperators: %d\nNumbers: %d\n",kw,id,op,num);
}
void keyw(char *p)
{
    int k,flag=0;
    for(k=0;k<=31;k++)
    {
        if(strcmp(keys[k],p)==0)
        {
            printf("%s is a keyword\n",p);
            kw++;
            flag=1;
            break;
        }
    }
    if(flag==0)
    {
        if(isdigit(p[0])){
            printf("%s is a number\n",p);
            num++;}
        else
        {
            if(p[0]!='\0')
            {
                printf("%s is an identifier\n",p);

```

```

    id++;
}
}
}
    i=-1;}

```

OUTPUT:

The screenshot shows a Cygwin terminal window titled 'C:\cygwin\bin\sh.exe' with the following output:

```

enter file path (drive:\fold\filename)
C:\Users\sid\Documents\NetBeansProjects\id2\input
#include<stdio.h> is a header file
void is a keyword
main is an identifier
int is a keyword
a is an identifier
b is an identifier
c is an identifier
printf is an identifier
Enter is an argument
while is a keyword
< is an operator
a is an identifier
b is an identifier
= is an operator
a is an identifier
+ is an operator
b is an identifier
c is an identifier
= is an operator
c is an identifier
a is an identifier
Keywords: 3
Identifiers: 12
Operators: 4
Numbers: 0
Press [Enter] to close the terminal ...

```

Overlaid on the terminal is a Notepad window titled 'Input - Notepad' containing the following C code:

```

#include<stdio.h>
void main()
{
    int a,b,c;
    printf("Enter");
    while(a<b)
        a=b+c;
    c=a;
}

```


EXPERIMENT 2:

AIM: Write a Program to Implement the Lexical Analyzer Using LEX Tool.

CODE:

```
% {
int COMMENT=0;
% }
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;} {printf("\n\t%s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{ {if(!COMMENT)printf("\n BLOCK BEGINS");}
\} {if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}\([0-9]*\)? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\'.*\' {if(!COMMENT)printf("\n\t%s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}
\(\(:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\ (ECHO;
= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
```

```
int main(int argc, char **argv)
{
FILE *file;
file=fopen("var.c","r");
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

INPUT:

```
//var.c
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

OUTPUT:

```
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ lex exp3_lex.l
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ cc lex.yy.c
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out
```

```
#include<stdio.h> is a preprocessor directive
```

```
#include<conio.h> is a preprocessor directive
```

```
void is a keyword
```

```
FUNCTION
main(
)
```

```
BLOCK BEGINS
```

```
int is a keyword
```

```
a IDENTIFIER,
b IDENTIFIER,
c IDENTIFIER;
```

```
a IDENTIFIER
= is an ASSIGNMENT OPERATOR
1 is a NUMBER ;
```

```
b IDENTIFIER
= is an ASSIGNMENT OPERATOR
2 is a NUMBER ;
```

```
c IDENTIFIER
= is an ASSIGNMENT OPERATOR
a IDENTIFIER+
b IDENTIFIER;
```

```
FUNCTION
printf(
"Sum:%d" is a STRING,
c IDENTIFIER
)
```

```
;
BLOCK ENDS
```

EXPERIMENT 3:

AIM:

Write a Program to Compute FIRST of Non-Terminals.

CODE:

```
#include<stdio.h>
char array[10][20],temp[10];
int c,n;
void fun(int,int[]);
int fun2(int i,int j,int p[],int );
void main()
{
    int p[2],i,j;
    printf("Enter the no. of productions :");
    scanf("%d",&n);
    printf("Enter the productions :\n");
    for(i=0;i<n;i++)
        scanf("%s",array[i]);
    for(i=0;i<n;i++)
    {
        c=-1,p[0]=-1,p[1]=-1;
        fun(i,p);
        printf("First(%c) : [ ",array[i][0]);
        for(j=0;j<=c;j++)
            printf("%c,",temp[j]);
        printf("\b ]\n");
        getch();
    }
}
int fun2(int i,int j,int p[],int key)
{
    int k;
    if(!key)
    {
        for(k=0;k<n;k++)
            if(array[i][j]==array[k][0])
                break;
        p[0]=i;p[1]=j+1;
        fun(k,p);
        return 0;
    }
    else
    {
        for(k=0;k<=c;k++)
        {
            if(array[i][j]==temp[k])
                break;
        }
        if(k>c)return 1;
        else return 0;
    }
}
```


EXPERIMENT 4

AIM:

Write a Program to compute FOLLOW of Non-Terminals.

CODE:

```
#include<stdio.h>
#define max 10
#define MAX 15

void ffun(int,int);
void fun(int,int[]);
void follow(int i);
char array[max][MAX],temp[max][MAX];
int c,n,t;
int fun2(int i,int j,int p[],int key)
{
    int k;
    if(!key){
        for(k=0;k<n;k++){
            if(array[i][j]==array[k][0])
                break;
            p[0]=i;p[1]=j+1;
            fun(k,p);
        }
        return 0;
    }
    else{
        for(k=0;k<=c;k++){
            if(array[i][j]==temp[t][k])
                break;
        }
        if(k>c)return 1;
        else return 0;
    }
}

void fun(int i,int p[])
{
    int j,k,key;
    for(j=2;array[i][j]!=NULL;j++)
    {
        if(array[i][j-1]=='/'){
            if(array[i][j]>='A'&&array[i][j]<='Z'){
                key=0;
                fun2(i,j,p,key);
            }
        }
        else{
            key=1;
            if(fun2(i,j,p,key))
                temp[t][++c]=array[i][j];
            if(array[i][j]=='@'&&p[0]!=-1){ //taking , @, as null symbol.
```



```

printf("Enter the no. of productions :");
scanf("%d",&n);
printf("Enter the productions :\n");
for(i=0;i<n;i++)
    scanf("%s",array[i]);
for(i=0,t=0;i<n;i++,t++){
    c=0,p[0]=-1,p[1]=-1;
    temp[t][0]=array[i][0];
    fun(i,p);
    temp[t][++c]=NULL;
    printf("First(%c) : [ ",temp[t][0]);
    for(j=1;j<c;j++)
        printf("%c,",temp[t][j]);
    printf("\b ]\n");
}
/* Follow Finding */
for(i=0,l=0;i<n;i++,l++)
{
    f=-1;ff0=-1;
    fol[l][++f]=array[i][0];
    follow(i);
    fol[l][++f]=NULL;
}
for(i=0;i<n;i++)
{
    printf("\nFollow[%c] : [ ",fol[i][0]);
    for(j=1;fol[i][j]!=NULL;j++)
        printf("%c,",fol[i][j]);
    printf("\b ]");
}
return 0;
}

```

OUTPUT:

```

Enter the no. of productions :6
Enter the productions :
S/aBDh
B/cC
C/bC/e
D/E/F
E/g/e
F/f/e
First(S) : [ a ].
First(B) : [ c ].
First(C) : [ b,e ].
First(D) : [ g,e,f ].
First(E) : [ g,e ].
First(F) : [ f,e ].

Follow[S] : [ $ ]
Follow[B] : [ g,f,h,$ ]
Follow[C] : [ g,f,h,$ ]
Follow[D] : [ h ]
Follow[E] : [ h ]
Follow[F] : [ h ]
Process returned 0 (0x0)   execution time : 112.581 s
Press any key to continue.
-
-
Press any key to continue.
Process returned 0 (0x0)   execution time : 112.281 s
Follow[S] : [ $ ]
Follow[B] : [ g,f,h,$ ]
Follow[C] : [ g,f,h,$ ]
Follow[D] : [ h ]
Follow[E] : [ h ]
Follow[F] : [ h ]

```


EXPERIMENT 5:

AIM:

Write a program to implement Top Down Parser (or) Recursive Descent Parser for given grammar.

CODE:

```
#include<stdio.h>
#include<string.h> char l;
void E();
void match(char); void T1();
void F();
void T();
void E1(); void main()
{
printf("enter string:\n"); l=getchar();
E();
if(l=='$')
printf("Given string is accepted \n"); else
printf("Given string is not accepted \n");
}
void match(char t)
{
if(l==t) l=getchar(); else printf("error");
}
void T1()
{
if(l=='*')
{
match('*'); F();
T();
}
else return ;
}
void F()
{ if(l=='('){
match('('); E(); if(l==')')
match(')');} else{ if(l=='i')
match('i');
if(l=='d') match('d');
}
}
void T()
{ F();
T1();
}
void E1()
{ if(l=='+')
{
match('+'); T();
E1();
}
else return ;
}
void E()
{ T();
E1();
}
```

OUTPUT:

```
Enter the string:id+id*id$
Given string is accepted
```

(OR)

Write a program to implement Top Down Parser (or) Recursive Descent Parser for given grammar2.

S → AB

A → a/ε

B → b/ε

CODE:

```
#include<stdio.h>
#include<string.h>
char str[10];
int i,error; void A();
void B();
void S(); int main()
{ i=0;
error=0;
printf("enter the string"); scanf("%s",str);
S();
if(strlen(str)==i && error==0) printf("Given grammer is Accepted\n"); else
printf("Given grammer is not accepted\n");
}
void S()
{ A();
B();
}
void A()
{
if(str[i]=='a')
{ i++;
}
}
void B()
{
if(str[i]=='b')
{ i++;
}
}
```

OUTPUT:

vi filename.c cc filename.c

./a.out

ab

Given grammer is accepted

ba

Given grammer is not accepted

EXPERIMENT 6

AIM: Write a program to implement LL(1) Parser for given grammar.

CODE:

```
#include<string.h>
char s[20],stack[20];
void main()
{
char m[5][6]={ "tb"," ","","tb"," "," "," "," "+tb"," "," ","n","n","fc"," "," ","fc"," "," "," ","n","*fc","
","n","n","i"," "," ","(e)"," "," "};
int size[5][6]={ 2,0,0,2,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};
int i,j,k,n,str1,str2;
printf("\n Enter the input string: ");
scanf("%s",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';
stack[1]='e';
i=1;
j=0;
printf("\nStack   Input\n");
printf("_____\n");
while((stack[i]!='$')&&(s[j]!='$')){
if(stack[i]==s[j]){
i--;
j++;}
switch(stack[i]){
case 'e': str1=0;
break;
case 'b': str1=1;
break;
case 't': str1=2;
break;
case 'c': str1=3;
break;
case 'f': str1=4;
break;}
switch(s[j]){
case 'i': str2=0;
break;
case '+': str2=1;
break;
case '*': str2=2;
break;
case '(': str2=3;
break;
case ')': str2=4;
break;
```

```

    case '$': str2=5;
    break;}
if(m[str1][str2][0]=="0"){
    printf("\nERROR");
    exit(0);}
else if(m[str1][str2][0]=='n')
    i--;
else if(m[str1][str2][0]=='i')
    stack[i]='i';
else{
    for(k=size[str1][str2]-1;k>=0;k--){
        stack[i]=m[str1][str2][k];
        i++;}
    i--;}
for(k=0;k<=i;k++)
    printf(" %c",stack[k]);
printf(" ");
for(k=j;k<=n;k++)
    printf("%c",s[k]);
printf(" \n ");}
printf("\n SUCCESS"); }

```

OUTPUT:

Enter the input string:i*i+i

Stack INPUT

```

$bt  i*i+i$
$bcf  i*i+i$
$bci  i*i+i$
$bc   *i+i$
$bcf*  *i+i$
$bcf  i+i$
$bci  i+i$
$bc   +i$
$b    +i$
$bt+  +i$
$bt   i$
$bcf  i$
$ bci  i$
$bc   $
$b    $
$     $

```

success

EXPERIMENT 8:

AIM:

Write a Program to Convert Infix Expression to Post Expression.

CODE:

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top = -1;
void push(char x)
{
    stack[++top] = x;
}
char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}
int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
    return 0;
}
int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;
    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c ",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
                printf("%c ", x);
        }
        else
        {

```

```

        while(priority(stack[top]) >= priority(*e))
            printf("%c ",pop());
        push(*e);
    }
    e++;
}
while(top != -1)
{
    printf("%c ",pop());
}return 0;
}

```

OUTPUT:

```

1.Enter the expression : a+b*c
a b c * +
2.Enter the expression : (a+b)*c+(d-a)
a b + c * d a - +
3.Enter the expression : ((4+8)(6-5))/((3-2)(2+2))
4 8 + 6 5 - 3 2 - 2 2 + /

```

EXPERIMENT 9

AIM: Write a Program to Construct the Predictive Parser for a Given Grammar.

CODE:

```
#include <stdio.h>
#include <string.h>
char prol[7][10] = { "S", "A", "A", "B", "B", "C", "C" };
char pror[7][10] = { "A", "Bb", "Cd", "aB", "@", "Cc", "@" };
char prod[7][10] = { "S->A", "A->Bb", "A->Cd", "B->aB", "B->@", "C->Cc", "C->@" };
char first[7][10] = { "abcd", "ab", "cd", "a@", "@", "c@", "@" };
char follow[7][10] = { "$", "$", "$", "a$", "b$", "c$", "d$" };
char table[5][6][10];
int numr(char c)
{
    switch (c)
    {
        case 'S':
            return 0;
        case 'A':
            return 1;
        case 'B':
            return 2;
        case 'C':
            return 3;
        case 'a':
            return 0;
        case 'b':
            return 1;
        case 'c':
            return 2;
        case 'd':
            return 3;
        case '$':
            return 4;
    }
    return (2);
}
int main()
{
    int i, j, k;
    for (i = 0; i < 5; i++)
        for (j = 0; j < 6; j++)
            strcpy(table[i][j], " ");
    printf("The following grammar is used for Parsing Table:\n");
    for (i = 0; i < 7; i++)
        printf("%s\n", prod[i]);
    printf("\nPredictive parsing table:\n");
    fflush(stdin);
    for (i = 0; i < 7; i++)
```

```

{
    k = strlen(first[i]);
    for (j = 0; j < 10; j++)
        if (first[i][j] != '@')
            strcpy(table[numr(prol[i][0]) + 1][numr(first[i][j]) + 1], prod[i]) }

for (i = 0; i < 7; i++)
{
    if (strlen(pror[i]) == 1)
    {
        if (pror[i][0] == '@')
        {
            k = strlen(follow[i]);
            for (j = 0; j < k; j++)
                strcpy(table[numr(prol[i][0]) + 1][numr(follow[i][j]) + 1], prod[i]);
        }
    }
}

strcpy(table[0][0], " ");
strcpy(table[0][1], "a");
strcpy(table[0][2], "b");
strcpy(table[0][3], "c");
strcpy(table[0][4], "d");
strcpy(table[0][5], "$");
strcpy(table[1][0], "S");
strcpy(table[2][0], "A");
strcpy(table[3][0], "B");
strcpy(table[4][0], "C");
printf("\n-----\n");
for (i = 0; i < 5; i++)
    for (j = 0; j < 6; j++)
    {
        printf("%-10s", table[i][j]);
        if (j == 5)
            printf("\n-----\n");
    }
}

```


OUTPUT:

The following grammar is used for Parsing Table:

- S->A
- A->Bb
- A->Cd
- B->aB
- B->@
- C->Cc
- C->@

Predictive parsing table:

	a	b	c	d	\$
S	S->A	S->A	S->A	S->A	
A	A->Bb	A->Bb	A->Cd	A->Cd	
B	B->aB	B->@	B->@		B->@
C			C->@	C->@	C->@

EXPERIMENT 10

AIM:

Write a Program to Construct the Shift-Reduce Parser for a given Grammar.

CODE:

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
void main()
{
puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
puts("enter input string ");
gets(a);
c=strlen(a);
strcpy(act,"SHIFT->");
puts("stack \t input \t action");
for(k=0,i=0; j<c; k++,i++,j++)
{
if(a[j]=='i' && a[j+1]=='d')
{
stk[i]=a[j];
stk[i+1]=a[j+1];
stk[i+2]='\0';
a[j]=' ';
a[j+1]=' ';
printf("\n$%s\t%s$\t%sid",stk,a,act);
check();
}
else
{
stk[i]=a[j];
stk[i+1]='\0';
a[j]=' ';
printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
check();
}
}

}
void check()
{
strcpy(ac,"REDUCE TO E");
```

```

for(z=0; z<c; z++)
    if(stk[z]=='i' && stk[z+1]=='d')
    {
        stk[z]='E';
        stk[z+1]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        j++;
    }
for(z=0; z<c; z++)
    if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
    }
for(z=0; z<c; z++)
    if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+1]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
    }
for(z=0; z<c; z++)
    if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+1]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
    }
}

```

OUTPUT:

GRAMMAR is $E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow id$

enter input string

id+id*id

stack	input	action
-------	-------	--------

\$id	+id*id\$	SHIFT->id
\$E	+id*id\$	REDUCE TO E
\$E+	id*id\$	SHIFT->symbols
\$E+id	*id\$	SHIFT->id
\$E+E	*id\$	REDUCE TO E
\$E	*id\$	REDUCE TO E
\$E*	id\$	SHIFT->symbols
\$E*id	\$	SHIFT->id
\$E*E	\$	REDUCE TO E
\$E	\$	REDUCE TO E

Given string is Accepted

EXPERIMENT 12 & 13:

AIM: Generate the Three Address Code for a Given Expression.

(OR)

Generate the Optimized Three Address Code for a Given Expression.

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h> int main()
{
char s[100]; char j='A';
printf("Enter the expression\n"); scanf("%s",s);
printf("3address code\n"); int i=2,n,h;
n=strlen(s); for(i=2;i<=n;i++)
{
if(s[i]=='*'||s[i]=='/')
{
printf("%c=%c%c%c\n",j,s[i-1],s[i],s[i+1]); s[i-1]=j++;
for(h=i;h<n-1;h++) s[h]=s[h+2];
n-=2;
i=i-1;
}
}
for(i=2;i<=n;i++)
{
if(s[i]=='+'||s[i]=='-')
{
printf("%c=%c%c%c\n",j,s[i-1],s[i],s[i+1]); s[i-1]=j++;
for(h=i;h<n-1;h++) s[h]=s[h+2];
n-=2;
i=i-1;
}
}
printf("%c=%c",s[0],s[2]); return 0;
}
```

OUTPUT:

$Z = a + b - c + d * e / f / g / h - k * l + m$

$A = d * e$

$B = A / f$

$C = B / g$

$D = C / h$

$E = k * l$

$F = a + b$

$G = F - c$

$H = D + G$

: $Z = H$

