Lab Experiments

Name: Jai Sahni

Registration Number: RA1711003011094

Batch: 2

Department: Computer Science & Engineering

Subject:15CS314J - Compiler Design

List of Experiments

1. Implementation of a text editor
2. Converting a regular expression to NFA
3. Conversion of an NFA to DFA
4. Conversion of an NFA to DFA
5. Computation of FIRST and FOLLOW sets
6. Computation of Leading and Trailing Sets
7. Construction of Predictive Parsing Table
8. Construction of Recursive Descent Parsing
9. Implementation of Shift Reduce Parsing
10. Computation of LR(0) items
11. Construction of DAG
12. Intermediate code generation- Three address code, Postfix, Prefix

EXPERIMENT-1 REGULAR EXPRESSION TO NFA

Aim: A program to convert Regular Expression to NFA

Algorithm:-

1. Start

2. Get the input from the user

3. Initialize separate variables and functions for Postfix , Display and NFA

4. Create separate methods for different operators like +,*, .

5. By using Switch case Initialize different cases for the input

6. For ' . ' operator Initialize a separate method by using various stack functions do the same for the other operators like ' * ' and ' + '.

7. Regular expression is in the form like a.b (or) a+b

8. Display the output

9. Stop

Program:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
char m[20],t[10][10];
int n,i,j,r=0,c=0;
clrscr();
printf("\n\t\t\t\tSIMULATION OF NFA"); printf("\n\t\t\t\t**********");
for(i=0;i<10;i++)
{for(j=0;j<10;j++)
{
t[i][j]=' ';
}
}
printf("\n\nEnter a regular expression:");
```

```
scanf("%s",m);
n=strlen(m);
for(i=0;i<n;i++)
{
switch(m[i])
{
case '|' :
{
t[r][r+1]='E';
t[r+1][r+2]=m[i-1];
t[r+2][r+5]='E';
t[r][r+3]='E';
t[r+4][r+5]='E';
t[r+3][r+4]=m[i+1];
r=r+5;
break;
}
case '*':
{
t[r-1][r]='E';
t[r][r+1]='E';
t[r][r+3]='E';
t[r+1][r+2]=m[i-1];
t[r+2][r+1]='E';
t[r+2][r+3]='E';
r=r+3;
break;
}
case '+':
{
```

```
t[r][r+1]=m[i-1];
t[r+1][r]='E';
r=r+1;
break;
}
default:
{
if(c==0)
{
if((isalpha(m[i]))&&(isalpha(m[i+1])))
{
t[r][r+1]=m[i];
t[r+1][r+2]=m[i+1];
r=r+2;
c=1;
}
c=1;
}
else if(c==1)
{
if(isalpha(m[i+1]))
{
t[r][r+1]=m[i+1];
r=r+1;
c=2;
}
}
else
{
if(isalpha(m[i+1]))
```

```
{
t[r][r+1]=m[i+1];
r=r+1;
c=3;
}
}
}
break;
}
}
printf("\n");
for(j=0;j<=r;j++)
printf(" %d",j);
printf("\n_____\n");
printf("\n");
for(i=0;i<=r;i++)
{
for(j=0;j<=r;j++)
{
printf(" %c",t[i][j]);
}
printf(" | %d",i);
printf("\n");
}
printf("\nStart state: 0\nFinal state: %d",i-1);
getch();
}
```

Output:-

Simulation of Data

****

Enter Regular expression (a+b)*

0 1 2 3 4

E |0

E E E |1

) | 2

E E |3

|4

Start state:0 Final state:4

Result:-

The program was successfully compiled and run.

EXPERIMENT-2 LEXICAL ANALYSER

Aim: A program to implement Lexical Analyser

Algorithm:-

1.Start.

2.Give the input in a file containing the program and save it in the desired location.

3.Open the file and start reading the file line by line.

4.Scan each line to see if it is an operator, keyword, identifier or a constant.

5.If not an operator, store the complete word in a buffer variable and check if it is a keyword.

6.If it is not a keyword then it is an identifier.

7.the above steps are repeated until the end of the file.

8.Close the file and end the program.

Program:-

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
int isKeyword(char buffer[]){
        char keywords[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"};
        int i, flag = 0;

        for(i = 0; i < 32; ++i){
                if(strcmp(keywords[i], buffer) == 0){
                        flag = 1;
                        break;
                }
        }

        return flag;
}
int main(){
        char ch, buffer[15], operators[] = "+-*/%=";
        FILE *fp;
        int i,j=0;

        fp = fopen("program.txt","r");

        if(fp == NULL){
                printf("error while opening the file\n");
                exit(0);
        }

        while((ch = fgetc(fp)) != EOF){
                for(i = 0; i < 6; ++i){
                        if(ch == operators[i])
                                printf("%c is operator\n", ch);
                }
```

```
                if(isalnum(ch)){
                        buffer[j++] = ch;
                }
                else if((ch == ' ' || ch == '\n') && (j != 0)){
                                buffer[j] = '\0';
                                j = 0;

                                if(isKeyword(buffer) == 1)
                                        printf("%s is keyword\n", buffer);
                                else
                                        printf("%s is indentifier\n", buffer);
                }
        }

        fclose(fp);
        return 0;
}
```

FILE: program.txt

```
void main(){
int a,b,c;
c=a+b;
}
```

Output:

```
void is keyword
main is identifier
int is keyword
a is identifier
b is identifier
c is identifier
```

= is operator

a is identifier

+ is operator

b is identifier

Result:-

The program was successfully compiled and run.

EXPERIMENT-3 NFA TO DFA

Aim: A program to convert NFA to DFA

Algorithm:-

1. Start

2. Get the input from the user

3. Implement the following sudo code:

Set the only state in SDFA to "unmarked"

while SDFA contains an unmarked state do

Let T be that unmarked state

for each a in % do

S = #-Closure(MoveNFA(T,a))

if S is not in SDFA already then

Add S to SDFA (as an "unmarked" state)

endIf

Set MoveDFA(T,a) to S

endFor

endWhile

for each S in SDFA do

if any s&S is a final state in the NFA then

Mark S an a final state in the DFA

endIf

endFor

4. Print the result.

Stop the program

Program:-

```
#include<stdio.h>

#include<string.h>

#define STATES 50

structDstate
```

```
{
        char name;
        charStateString[STATES+1];
        char trans[10];
        intis_final;
}Dstates[50];
structtran
{
        charsym;
        inttostates[50];
        intnotran;
};
struct state
{
        int no;
        structtrantranlist[50];
};
intstackA[100],stackB[100],c[100],Cptr=-1,Aptr=-1,Bptr=-1;
struct state States[10];
char temp[STATES+1],inp[10];
intnos,noi,nof,j,k,nods=-1;



voidpushA(int z)
{
stackA[++Aptr]=z;
```

```
}
voidpushB(int z)
{
stackB[++Bptr]=z;
}


intpopA()
{
returnstackA[Aptr--];
}
void copy(inti)
{
        char temp[STATES+1]=" ";
        int k=0;
        Bptr=-1;
        strcpy(temp,Dstates[i].StateString);
        while(temp[k]!='\0')
{
        pushB(temp[k]-'0');
        k++;
}
}
intpopB()
{
returnstackB[Bptr--];
```

```
}
intpeekA()
{
returnstackA[Aptr];
}
intpeekB()
{
returnstackA[Bptr];
}
int seek(intarr[],intptr,int s)
{
        inti;
        for(i=0;i<=ptr;i++)
        {


                if(s==arr[i])
                return 1;
        }
return 0;
}
void sort()
{
        inti,j,temp;
        for(i=0;i<Bptr;i++)
        {
```

```
		for(j=0;j<(Bptr-i);j++)
		{
			if(stackB[j]>stackB[j+1])
			{
					temp=stackB[j];
					stackB[j]=stackB[j+1];
					stackB[j+1]=temp;
				}
		}
}
}
voidtostring()
{
		inti=0;
		sort();
		for(i=0;i<=Bptr;i++)
		{
		temp[i]=stackB[i]+'0';
}
temp[i]='\0';
}
voiddisplay_DTran()
{
		inti,j;
		printf("\n\t\t DFA transition table");
		printf("\n\t\t---------------------------------------------- ");
```

```
printf("\n States \tString \tInputs\n");
for(i=0;i<noi;i++)
{
printf("\t %c",inp[i]);
}
printf("\n\t ------------------------------------------------- ");
for(i=0;i<nods;i++)
{
if(Dstates[i].is_final==0)
printf("\n%c",Dstates[i].name);
else
printf("\n*%c",Dstates[i].name);
printf("\t%s",Dstates[i].StateString);
for(j=0;j<noi;j++)
{
printf("\t%c",Dstates[i].trans[j]);
}
}
printf("\n");
}
void move(intst,int j)
{
intctr=0;
while(ctr<States[st].tranlist[j].notran)
{
pushA(States[st].tranlist[j].tostates[ctr++]);
```

```
        }
}
voidlambda_closure(intst)
{
        intctr=0,in_state=st,curst=st,chk;
        while(Aptr!=-1)
        {
        curst=popA();
                ctr=0;
                in_state=curst;
        while(ctr<=States[curst].tranlist[noi].notran)
        {
                        chk=seek(stackB,Bptr,in_state);
                        if(chk==0)
                        pushB(in_state);
                        in_state=States[curst].tranlist[noi].tostates[ctr++];
                        chk=seek(stackA,Aptr,in_state);
                        if(chk==0 &&ctr<=States[curst].tranlist[noi].notran)
                        pushA(in_state);
        }
        }
        }
Void main()
{
        inti,final[20],start,fin=0;
        charc,ans,st[20];
```

```
printf("\n Enter no of states in NFA:");
scanf("%d",&nos);
for(i=0;i<nos;i++)
{
States[i].no=i;
}
printf("\n Enter the start states:");
scanf("%d",&start);
printf("Enter the no of final states:");
scanf("%d",&nof);
printf("Enter the final states:\n");
for(i=0;i<nof;i++)
scanf("%d",&final[i]);
printf("\n Enter the no of input symbols:");
scanf("%d",&noi);
c=getchar();
printf("Enter the input symbols:\n");
for(i=0;i<noi;i++)
{
        scanf("%c",&inp[i]);
        c=getchar();
}
g1inp[i]='e';
printf("\n Enter the transitions:(-1 to stop)\n");
for(i=0;i<nos;i++)
{
```

```
for(j=0;j<=noi;j++)
{
        States[i].tranlist[j].sym=inp[j];
        k=0;
        ans='y';
        while(ans=='y')
        {
                printf("move(%d,%c);",i,inp[j]);
                scanf("%d",&States[i].tranlist[j].tostates[k++]);
                if((States[i].tranlist[j].tostates[k-1]==-1))
                {
                        k--;
                        ans='n';
                        break;
}
}
                States[i].tranlist[j].notran=k;
}
}
i=0;nods=0,fin=0;
pushA(start);
lambda_closure(peekA());
tostring();
Dstates[nods].name='A';
nods++;
strcpy(Dstates[0].StateString,temp);
```

```
while(i<nods)
{
for(j=0;j<noi;j++)
{
        fin=0;
        copy(i);
        while(Bptr!=-1)
        {
        move(popB(),j);
        }


while(Aptr!=-1)
lambda_closure(peekA());
tostring();
for(k=0;k<nods;k++)
{
        if((strcmp(temp,Dstates[k].StateString)==0))
        {
                Dstates[i].trans[j]=Dstates[k].name;
                break;
}
}
if(k==nods)
{
        nods++;
```

```
                for(k=0;k<nof;k++)
        {
                fin=seek(stackB,Bptr,final[k]);
                if(fin==1)
                {
                        Dstates[nods-1].is_final=1;
                        break;
                }
        }
                strcpy(Dstates[nods-1].StateString,temp);
                Dstates[nods-1].name='A'+nods-1;
                Dstates[i].trans[j]=Dstates[nods-1].name;
        }
        }
        i++;
        }
        display_DTran();
        }
```

Output:

```
Enter the no of input symbols:2
Enter the input symbols:
a ,b
Enter the transitions:(-1 to stop)
move(0,a);-1
move(0,b);-1
move(0,e);1
move(     0,e);7
```

move(0,e);-1

move(1,a);-1

move(1,b);-1

move(1,e);2

move(1,e);4

move(1,e);-1

move(2,a);3

move(2,a);3

move(2,a);-1

move(2,b);-1

move(2,e);-1

move(3,a);-1

move(3,b);-1

move(3,e);6

move(3,e);-1

move(4,a);-1

move(4,b);-1

move(4,e);-1

move(5,a);-1

move(5,b);-1

move(5,e);6

move(5,e);1

move(5,e);-1

move(6,a);-1

move(6,b);-1

move(6,e);-1

move(7,a);-1

move(7,b);-1

move(7,e);-1

DFA transition table

States String Inputs

a b

A 01247 B C

B 36 C C

C C C

Result:-

The program was successfully compiled and run.

EXPERIMENT-4 ELIMINATION OF LEFT RECURSION

Aim: A program for Elimination Of Left Recursion

Algorithm:

1. Start the program.

2. Initialize the arrays for taking input from the user.

3. Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.

4. Prompt the user to input the right production for non-terminals.

5. Eliminate left recursion using the following rules:-

A->Aα1| Aα2 | .....|Aαm

A->β1| β2|.....| βn

Then replace it by

A'-> βi A' i=1,2,3,..... m

A'-> αj A' j=1,2,3,..... n

A'-> ε

6. After eliminating the left recursion by applying these rules, display the productions without left recursion.

7. Stop.

Program:-

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

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
	char a[10],b[50][10]={""},d[50][10]={""},ch;
	int i,n,c[10]={0},j,k,t,n1;
	clrscr();
	printf("\nEnter the left production(s) (NON TERMINALS) : ");
	scanf("%s",a);
	n=strlen(a);
	for(i=0;i<n;i++)
	{
		printf("\nEnter the number of productions for %c : ",a[i]);
		scanf("%d",&c[i]);
	}
	t=0;
	for(i=0;i<n;i++)
	{
		printf("\nEnter the right productions for %c",a[i]);
		k=t;
		for(j=0;j<c[i];j++)
		{
			printf("\n%c->",a[i]);
			do
			{
				scanf("%s",b[k]);
				k++;
			}while(k<j);
		}
```

```
		t=t+10;
}
t=0;
for(i=0;i<n;i++)
{
		if(a[i]==b[t][0])
		{
			n1=strlen(b[t]);
			for(k=1;k<n1;k++)
			{
				d[t][k-1]=b[t][k];
			}
		}
		t=t+10;
}
t=0;
printf("\n\nThe resulting productions after eliminating Left Recursion are : \n");
for(i=0;i<n;i++)
{
		if(a[i]==b[t][0])
		{
			for(j=1;j<c[i];j++)
			{
				printf("\n%c -> %s%c'",a[i],b[t+j],a[i]);
			}
		}
		t=t+10;
}
t=0;
for(i=0;i<n;i++)
```

```
        {
                if(a[i]==b[t][0])
                        printf("\n%c' -> %s%c'|%c",a[i],d[t],a[i],(char)238);
                else
                        for(j=0;j<c[i];j++)
                                printf("\n%c -> %s",a[i],b[t+j]);
                t=t+10;
        }
        getch();
}
```

Output

```
Enter the left production(s) (NON TERMINALS) : ETF
Enter the number of productions for E : 2
Enter the number of productions for T : 2
Enter the number of productions for F : 2
Enter the right productions for E
E->E+T
E->T
Enter the right productions for T
T->T*F
T->F
Enter the right productions for F
F->(E)
F->i
The resulting productions after eliminating Left Recursion are :
E -> TE'
T -> FT'
E' -> +TE'|ε
```

T' -> *FT'|ε

F -> (E)

F -> i

Result:-

The program was successfully compiled and run.

EXPERIMENT-5 ELIMINATION OF LEFT FACTORING

Aim: A program for Elimination Of Left Factoring

Algorithm:-

1. Start

2. Ask the user to enter the set of productions from which the left factoring is to be removed.

3. Check for left factoring in the given set of productions by comparing with: A->aB1|aB2

4. If found, replace the particular productions with:

A->aA'

A'->B1 | B2|ε

5. Display the output

6. Exit

Program:

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
        char ch,lhs[20][20],rhs[20][20][20],temp[20],temp1[20];
```

```
int n,n1,count[20],x,y,i,j,k,c[20];
clrscr();
printf("\nEnter the no. of productions : ");
scanf("%d",&n);
n1=n;
for(i=0;i<n;i++)
{
	printf("\nProduction %d \nEnter the no. of productions : ",i+1);
	scanf("%d",&c[i]);
	printf("\nEnter LHS : ");
	scanf("%s",lhs[i]);
	for(j=0;j<c[i];j++)
	{
		printf("%s->",lhs[i]);
		scanf("%s",rhs[i][j]);
	}
}
for(i=0;i<n;i++)
{
	count[i]=1;
	while(memcmp(rhs[i][0],rhs[i][1],count[i])==0)
		count[i]++;
}
for(i=0;i<n;i++)
{
	count[i]--;
	if(count[i]>0)
	{
		strcpy(lhs[n1],lhs[i]);
		strcat(lhs[i],"'");
```

```
                for(k=0;k<count[i];k++)
                        temp1[k] = rhs[i][0][k];
                temp1[k++] = '\0';
                for(j=0;j<c[i];j++)
                {
                        for(k=count[i],x=0;k<strlen(rhs[i][j]);x++,k++)
                                temp[x] = rhs[i][j][k];
                        temp[x++] = '\0';
                        if(strlen(rhs[i][j])==1)
                                strcpy(rhs[n1][1],rhs[i][j]);
                        strcpy(rhs[i][j],temp);
                }
                c[n1]=2;
                strcpy(rhs[n1][0],temp1);
                strcat(rhs[n1][0],lhs[n1]);
                strcat(rhs[n1][0],"'");
                n1++;
        }
}
printf("\n\nThe resulting productions are : \n");
for(i=0;i<n1;i++)
{
        if(i==0)
                printf("\n %s -> %c|",lhs[i],(char)238);
        else
                printf("\n %s -> ",lhs[i]);
        for(j=0;j<c[i];j++)
        {
                printf(" %s ",rhs[i][j]);
                if((j+1)!=c[i])
```

```
                                        printf("|");
                }
                printf("\b\b\b\n");
        }
        getch();
}
```

Output

```
Enter the no. of productions : 2
Production 1
Enter the no. of productions : 3
Enter LHS : S
S->iCtSeS
S->iCtS
S->a
Production 2
Enter the no. of productions : 1
Enter LHS : C
C->b
The resulting productions are :
S' -> ε| eS | |
C -> b
S -> iCtSS' | a
```

Result:-

The program was successfully compiled and run.

EXPERIMENT-6 SHIFT REDUCE PARSING

Aim: Program to implement Shift Reduce Parsing

Algorithm:-

1. Start the program.
2. Initialize the required variables.
3. Enter the input symbol.

4. Perform the following:

for top-of-stack symbol, s, and next input symbol, a

shift x: (x is a STATE number)

push a, then x on the top of the stack and

advance ip to point to the next input symbol.

reduce y: (y is a PRODUCTION number)

Assume that the production is of the form

A ==> beta

*pop 2 * |beta| symbols of the stack. At this*

point the top of the stack should be a state number,

say s'. push A, then goto of T[s',A] (a state number)

on the top of the stack. Output the production

A ==> beta.

5. Print if string is accepted or not.

6. Stop the program.

Program:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
int novar=0,sttop=1,intop=1,j=0,i=0,handlelength=0;
char ipstr1[100],ipstr[100],popped,var;
char prod[20][20],handle[100],stack[100]="#",input[100]="#";



struct grammar
{
        char lhs,rhs[20][20];
        int noprod;
```

```
}g[20];



int checkhandle()
{
	int i,m,k;
	char temp[2]={ ' ' , '\0'};
	for(i=0;i<sttop;i++)
	{
		strcpy(handle,"");
		for(m=i;m<=sttop-1;temp[0]=stack[m],strcat(handle,temp),m++);
		for(m=0;m<novar;m++)
		{
			for(k=0;k<g[m].noprod && strcmp(handle,g[m].rhs[k])!=0;k++);
			if(k!=g[m].noprod)
			{
				var=g[m].lhs;
				return strlen(handle);
			}
		}
	}
	return 0;
}



void print(char *text,int textlen)
{
	int i;
	for(i=0;i<textlen;i++)
		printf("%c",text[i]);
```

```
		printf("\t\t\t");
}
void printi(char *text,int textlen)
{
		int i;
		for(i=textlen-1;i>=0;i--)
			printf("%c",text[i]);
		printf("\t\t\t");
}
void main()
{
		int n,m,k,len,j=0,v;
		clrscr();
		printf("\n Enter the productions of the grammar(END to end):\n");
		do
		{
			scanf("%s",prod[i++]);
		}while(strcmp(prod[i-1],"END")!=0);
		for(n=0;n<i-1;n++)
		{
			m=0,k=0;
			for(j=0;j<novar;j++)
				if(g[j].lhs==prod[n][0])
					break;
			if(j==novar)
				g[novar++].lhs=prod[n][0];
			for(k=3;k<strlen(prod[n])+1;k++)
			{
				if(prod[n][k]!='|' && prod[n][k]!='\0')
					g[j].rhs[g[j].noprod][m++]=prod[n][k];
```

```
                        if(prod[n][k]=='|' || prod[n][k]=='\0')
                        {
                                g[j].rhs[g[j].noprod++][m]='\0';
                                m=0;
                        }
                }
}
printf("\nENTER THE INPUT STRING:");
scanf("%s",ipstr);
printf("\n\n\n\n");
for(i=strlen(ipstr)-1;i>=0;i--)
                input[intop++]=ipstr[i];
printf(" ------------------------------------------------------------\n");
printf(" STACK\t\t\tINPUT\t\t\tACTION\n");
printf(" ------------------------------------------------------------\n");
print(stack,sttop);
printi(input,intop);
while(1)
{
                int count=0;
                while((handlelength=checkhandle())>0)
                {
                        if(input[intop-1]=='*' && count++==2)
                                break;
                        else if(input[intop-1]=='=' && count++==3)
                                break;
                        else if(input[intop-1]=='e' && count++==1)
                                break;
                        for(i=0;i<handlelength;i++,--sttop);
                                stack[sttop++]=var;
```

```
                printf("REDUCE BY %c -> %s\n",var,handle);
                print(stack,sttop);
                printi(input,intop);
            }
            popped=input[--intop];
            if(popped!='#')
                stack[sttop++]=popped;
            handlelength=checkhandle();
            if(popped=='#' && (handlelength=checkhandle())==0)
                break;
            printf("SHIFT ' %c '\n",popped);
            print(stack,sttop);
            printi(input,intop);
        }
        if(sttop==2 && stack[1]==g[0].lhs)
            printf("ACCEPT\n");
        else
            printf("ERROR\n");
        printf(" ------------------------------------------------------------");
        getch();
}
```

Output:

Enter the productions of the grammar(END to end):

S->iCtSeS|iCtS|a

C->b

END

ENTER THE INPUT STRING: ibtaea

STACK INPUT ACTION

---

ibtaea# SHIFT ' i '

#i btaea# SHIFT ' b'

#ib taea# REDUCE BY C -> b

#iC taea# SHIFT ' t '

#iCt aea# SHIFT ' a '

#iCta ea# REDUCE BY S -> a

#iCtS ea# SHIFT ' e '

#iCtSe a# SHIFT ' a '

#iCtSea # REDUCE BY S -> a

#iCtSeS # REDUCE BY S -> iCtSeS

#S # ACCEPT

Result:-

The program was successfully compiled and run.

EXPERIMENT-7 PREDICTIVE PARSING

Aim: A program for Predictive Parsing

Algorithm:-

1. Start the program.

2. Initialize the required variables.

3. Get the number of coordinates and productions from the user.

4. Perform the following

for (each production A → α in G) {

for (each terminal a in FIRST(α))

add A → α to M[A, a];

if (ε is in FIRST(α))

for (each symbol b in FOLLOW(A))

add A → α to M[A, b];

5. Print the resulting stack.

6. Print if the grammar is accepted or not.

7. Exit the program.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char fin[10][20],st[10][20],ft[20][20],fol[20][20];
int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;
clrscr();
printf("enter the no. of coordinates\n");
scanf("%d",&n);
printf("enter the productions in a grammar\n");
for(i=0;i<n;i++)
scanf("%s",st[i]);
for(i=0;i<n;i++)
fol[i][0]='\0';
for(s=0;s<n;s++)
{
for(i=0;i<n;i++)
{
j=3;
l=0;
a=0;
l1:if(!((st[i][j]>64)&&(st[i][j]<91)))
{
```

```
for(m=0;m<l;m++)
{
if(ft[i][m]==st[i][j])
goto s1;
}
ft[i][l]=st[i][j];
l=l+1;
s1:j=j+1;
}
else
{
if(s>0)
{
while(st[i][j]!=st[a][0])
{
a++;
}
b=0;
while(ft[a][b]!='\0')
{
for(m=0;m<l;m++)
{
if(ft[i][m]==ft[a][b])
goto s2;
}
ft[i][l]=ft[a][b];
l=l+1;
s2:b=b+1;
}
}
```

```
}
while(st[i][j]!='\0')
{
if(st[i][j]=='|')
{
j=j+1;
goto l1;
}
j=j+1;
}
ft[i][l]='\0';
}
}
printf("first pos\n");
for(i=0;i<n;i++)
printf("FIRS[%c]=%s\n",st[i][0],ft[i]);
fol[0][0]='$';
for(i=0;i<n;i++)
{
k=0;
j=3;
if(i==0)
l=1;
else
l=0;
k1:while((st[i][0]!=st[k][j])&&(k<n))
{
if(st[k][j]=='\0')
{
k++;
```

```
j=2;
}
j++;
}
j=j+1;
if(st[i][0]==st[k][j-1])
{
if((st[k][j]!='|')&&(st[k][j]!='\0'))
{
a=0;
if(!((st[k][j]>64)&&(st[k][j]<91)))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==st[k][j])
goto q3;
}
fol[i][l]=st[k][j];
l++;
q3:
}
else
{
while(st[k][j]!=st[a][0])
{
a++;
}
p=0;
while(ft[a][p]!='\0')
{
```

```
if(ft[a][p]!='@')
{
for(m=0;m<l;m++)
{
if(fol[i][m]==ft[a][p])
goto q2;
}
fol[i][l]=ft[a][p];
l=l+1;
}
else
e=1;
q2:p++;
}
if(e==1)
{
e=0;
goto a1;
}
}
}
else
{
a1:c=0;
a=0;
while(st[k][0]!=st[a][0])
{
a++;
}
while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))
```

```
{
for(m=0;m<l;m++)
{
if(fol[i][m]==fol[a][c])
goto q1;
}
fol[i][l]=fol[a][c];
l++;
q1:c++;
}
}
goto k1;
}
fol[i][l]='\0';
}
printf("follow pos\n");
for(i=0;i<n;i++)
printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);
printf("\n");
s=0;
for(i=0;i<n;i++)
{
j=3;
while(st[i][j]!='\0')
{
if((st[i][j-1]=='|')||(j==3))
{
for(p=0;p<=2;p++)
{
fin[s][p]=st[i][p];
```

```
}
t=j;
for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++)
{
fin[s][p]=st[i][j];
j++;
}
fin[s][p]='\0';
if(st[i][k]=='@')
{
b=0;
a=0;
while(st[a][0]!=st[i][0])
{
a++;
}
while(fol[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);
b++;
}
}
else if(!((st[i][t]>64)&&(st[i][t]<91)))
printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);
else
{
b=0;
a=0;
while(st[a][0]!=st[i][3])
{
```

```
a++;
}
while(ft[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);
b++;
}
}
s++;
}
if(st[i][j]=='|')
j++;
}
}
getch();
}
```

Output:

```
Enter the no. of co-ordinates
2
Enter the productions in a grammar
S->CC
C->eC | d
First pos
FIRS[S] = ed
FIRS[C] = ed
Follow pos
FOLLOW[S] =$
FOLLOW[C] =ed$
M [S , e] =S->CC
M [S , d] =S->CC
```

M [C , e] =C->eC

M [C , d] =C->d

Result:-

The program was successfully compiled and run.

EXPERIMENT-8 FIRST AND FOLLOW

Aim: A program to implement First and Follow

Algorithm:-

For computing the first:

1. If X is a terminal then FIRST(X) = {X}

Example: F -> (E) | id

We can write it as FIRST(F) -> { (, id }

2. If X is a non terminal like E -> T then to get

FIRST(E) substitute T with other productions until you get a terminal as the first symbol

3. If X -> ε then add ε to FIRST(X).

For computing the follow:

1. Always check the right side of the productions for a non-terminal, whose FOLLOW set is

being found. (never see the left side).

2. (a) If that non-terminal (S,A,B…) is followed by any terminal (a,b…,*,+,(,)…) , then add

that “terminal” into FOLLOW set.

(b) If that non-terminal is followed by any other non-terminal then add “FIRST of other nonterminal” into FOLLOW set.

Program:

```
#include<stdio.h>
#include<string.h>
        #include<conio.h>
        #define max 20
        char prod[max][10];
        char ter[10],nt[10];
        char first[10][10],follow[10][10];
        int eps[10];
        int count=0;
```

```
int findpos(char ch)

{

int n;

for(n=0;nt[n]!='\0';n++)

if(nt[n]==ch)

break;

if(nt[n]=='\0')

return 1;

return n;

}


int IsCap(char c)

{

if(c >= 'A' && c<= 'Z')

return 1;

return 0;

}

void add(char *arr,char c)

{

int i,flag=0;

for(i=0;arr[i]!='\0';i++)

{

if(arr[i] == c)

{

flag=1;

break;

}

}

if(flag!=1)

arr[strlen(arr)] = c;
```

```
}
void addarr(char *s1,char *s2)
{
int i,j,flag=99;
for(i=0;s2[i]!='\0';i++)
{
flag=0;
for(j=0;;j++)
{
if(s2[i]==s1[j])
{
flag=1;
break;
}
if(j==strlen(s1) && flag!=1)
{
s1[strlen(s1)] = s2[i];
break;
}
}
}
}
}
void addprod(char *s)
{
int i;
prod[count][0] = s[0];
for(i=3;s[i]!='\0';i++)
{
if(!IsCap(s[i]))
add(ter,s[i]);
```

```
prod[count][i-2] = s[i];
}
prod[count][i-2] = '\0';
add(nt,s[0]);
count++;
}
void findfirst()
{
int i,j,n,k,e,n1;
for(i=0;i<count;i++)
{
for(j=0;j<count;j++)
{
n = findpos(prod[j][0]);
if(prod[j][1] == (char)238)
eps[n] = 1;
else
{
for(k=1,e=1;prod[j][k]!='\0' && e==1;k++)
{
if(!IsCap(prod[j][k]))
{
e=0;
add(first[n],prod[j][k]);
}
else
{
n1 = findpos(prod[j][k]);
addarr(first[n],first[n1]);
if(eps[n1] == 0)
```

```
e=0;
}
The                                     }
if(e==1)
eps[n]=1;
}
}
}
}
void findfollow()
{
int i,j,k,n,e,n1;
n = findpos(prod[0][0]);
add(follow[n],'#');
for(i=0;i<count;i++)
{
for(j=0;j<count;j++)
{
k = strlen(prod[j])-1;
for(;k>0;k--)
{
if(IsCap(prod[j][k]))
{
n=findpos(prod[j][k]);
if(prod[j][k+1] == '\0')          // A -> aB
{
n1 = findpos(prod[j][0]);
addarr(follow[n],follow[n1]);
}
if(IsCap(prod[j][k+1]))                    // A -> aBb
```

```
{
n1 = findpos(prod[j][k+1]);
addarr(follow[n],first[n1]);
if(eps[n1]==1)
{
n1=findpos(prod[j][0]);
addarr(follow[n],follow[n1]);
}
}
else if(prod[j][k+1] != '\0')
add(follow[n],prod[j][k+1]);
}
}
}
}
}
void main()
{
char s[max],i;
printf("\nEnter the productions(type 'end' at the last of the production)\n");
scanf("%s",s);
while(strcmp("end",s))
{
addprod(s);
scanf("%s",s);
}
findfirst();
findfollow();
for(i=0;i<strlen(nt);i++)
{
```

```
printf("%c\t",nt[i]);
printf("%s",first[i]);
if(eps[i]==1)
printf("%c\t",(char)238);
else
printf("\t");
printf("%s\n",follow[i]);
}
getch();
}
}
```

Output:-

```
Enter the productions(type 'end' at the last of the production)
E->TA
A->+TA
A->ε
T->FB
B->*FB
B->ε
F->(E)
F->i
end

NT First Follow
_____________________

E          (i          #)
A          +ε          #)
```

T	(i	+#)
B	*ε	+#)
F	(i	*+#)

Result:-

The program was successfully compiled and run.

EXPERIMENT-9 LEADING AND TRAILING

Aim: A program to implement Leading and Trailing

Algorithm:-

1. For Leading, check for the first non-terminal.

2. If found, print it.

3. Look for next production for the same non-terminal.

4. If not found, recursively call the procedure for the single non-terminal present before the comma or End Of Production String.

5. Include it's results in the result of this non-terminal.

6. For trailing, we compute same as leading but we start from the end of the production to the beginning.

7. Stop

Program:

```
#include<iostream.h>

#include<conio.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>
```

```
int vars,terms,i,j,k,m,rep,count,temp=-1;
char var[10],term[10],lead[10][10],trail[10][10];
struct grammar
{
	int prodno;
	char lhs,rhs[20][20];
}gram[50];
void get()
{
	cout<<"\n---------- LEADING AND TRAILING ---------- \n";
	cout<<"\nEnter the no. of variables : ";
	cin>>vars;
	cout<<"\nEnter the variables : \n";
	for(i=0;i<vars;i++)
	{
		cin>>gram[i].lhs;
		var[i]=gram[i].lhs;
	}
	cout<<"\nEnter the no. of terminals : ";
	cin>>terms;
	cout<<"\nEnter the terminals : ";
	for(j=0;j<terms;j++)
		cin>>term[j];
	cout<<"\n---------- PRODUCTION DETAILS ---------- \n";
	for(i=0;i<vars;i++)
	{
		cout<<"\nEnter the no. of production of "<<gram[i].lhs<<":";
		cin>>gram[i].prodno;
		for(j=0;j<gram[i].prodno;j++)
		{
```

```
                cout<<gram[i].lhs<<"->";
                cin>>gram[i].rhs[j];
            }
        }
}
void leading()
{
        for(i=0;i<vars;i++)
        {
            for(j=0;j<gram[i].prodno;j++)
            {
                for(k=0;k<terms;k++)
                {
                    if(gram[i].rhs[j][0]==term[k])
                        lead[i][k]=1;
                    else
                    {
                        if(gram[i].rhs[j][1]==term[k])
                            lead[i][k]=1;
                    }
                }
            }
        }
        for(rep=0;rep<vars;rep++)
        {
            for(i=0;i<vars;i++)
            {
                for(j=0;j<gram[i].prodno;j++)
                {
                    for(m=1;m<vars;m++)
```

```
                {
                        if(gram[i].rhs[j][0]==var[m])
                        {
                                temp=m;
                                goto out;
                        }
                }
                out:
                for(k=0;k<terms;k++)
                {
                        if(lead[temp][k]==1)
                                lead[i][k]=1;
                }
        }
    }
}
void trailing()
{
    for(i=0;i<vars;i++)
    {
        for(j=0;j<gram[i].prodno;j++)
        {
            count=0;
            while(gram[i].rhs[j][count]!='\x0')
                count++;
            for(k=0;k<terms;k++)
            {
                if(gram[i].rhs[j][count-1]==term[k])
                    trail[i][k]=1;
```

```
                else
                {
                        if(gram[i].rhs[j][count-2]==term[k])
                                trail[i][k]=1;
                }
        }
    }
    for(rep=0;rep<vars;rep++)
    {
        for(i=0;i<vars;i++)
        {
                for(j=0;j<gram[i].prodno;j++)
                {
                        count=0;
                        while(gram[i].rhs[j][count]!='\x0')
                                count++;
                        for(m=1;m<vars;m++)
                        {
                                if(gram[i].rhs[j][count-1]==var[m])
                                        temp=m;
                        }
                        for(k=0;k<terms;k++)
                        {
                                if(trail[temp][k]==1)
                                        trail[i][k]=1;
                        }
                }
        }
    }
```

```
}
void display()
{
	for(i=0;i<vars;i++)
	{
		cout<<"\nLEADING("<<gram[i].lhs<<") = ";
		for(j=0;j<terms;j++)
		{
			if(lead[i][j]==1)
				cout<<term[j]<<",";
		}
	}
	cout<<endl;
	for(i=0;i<vars;i++)
	{
		cout<<"\nTRAILING("<<gram[i].lhs<<") = ";
		for(j=0;j<terms;j++)
		{
			if(trail[i][j]==1)
				cout<<term[j]<<",";
		}
	}
}
void main()
{
	clrscr();
	get();
	leading();
	trailing();
	display();
```

```
        getch();
}
```

Output:

```
---------- LEADING AND TRAILING ----------

Enter the no. of variables : 3

Enter the variables :
E
T
F
Enter the no. of terminals : 5

Enter the terminals :
(
)
*
+
i

---------- PRODUCTION DETAILS ----------
Enter the no. of production of E:2
E->E+T
E->T
```

Enter the no. of production of T:2

T->T*F

T->F

Enter the no. of production of F:2

F->(E)

F->i

LEADING(E) = (,*,+,i,

LEADING(T) = (,*,i,

LEADING(F) = (,i,

TRAILING(E) =),*,+,i,

TRAILING(T) =),*,i,

TRAILING(F) =),i,

Result:-

The program was successfully compiled and run.

EXPERIMENT-10 LRF0

Aim: A program to implement LRF0

Algorithm:-

1. Start.

2. Create structure for production with LHS and RHS.

3. Open file and read input from file.

4. Build state 0 from extra grammar Law S' -> S $ that is all start symbol of grammar and one Dot (.) before S symbol.

5. If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left Hand Side of that Law and set Dot in before of first part of Right Hand Side.

6. If state exists (a state with this Laws and same Dot position), use that instead.

7. Now find set of terminals and non-terminals in which Dot exist in before.

8. If step 7 Set is non-empty go to 9, else go to 10.

9. For each terminal/non-terminal in set step 7 create new state by using all grammar law that Dot position is before of that terminal/non-terminal in reference state by increasing Dot point to next part in Right Hand Side of that laws.

10. Go to step 5.

11. End of state building.

12. Display the output.

13. End.

Program:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR";
int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
int noitem=0;
struct Grammar
{
        char lhs;
        char rhs[8];
}g[20],item[20],clos[20][10];
```

```
int isvariable(char variable)
{
	for(int i=0;i<novar;i++)
		if(g[i].lhs==variable)
			return i+1;
	return 0;
}
void findclosure(int z, char a)
{
	int n=0,i=0,j=0,k=0,l=0;
	for(i=0;i<arr[z];i++)
	{
		for(j=0;j<strlen(clos[z][i].rhs);j++)
		{
			if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a)
			{
				clos[noitem][n].lhs=clos[z][i].lhs;
				strcpy(clos[noitem][n].rhs,clos[z][i].rhs);
				char temp=clos[noitem][n].rhs[j];
				clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];
				clos[noitem][n].rhs[j+1]=temp;
				n=n+1;
			}
		}
	}
	for(i=0;i<n;i++)
	{
		for(j=0;j<strlen(clos[noitem][i].rhs);j++)
```

```
		{
			if(clos[noitem][i].rhs[j]=='.' && isvariable(clos[noitem][i].rhs[j+1])>0)
			{
				for(k=0;k<novar;k++)
				{
					if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)
					{
						for(l=0;l<n;l++)
							if(clos[noitem][l].lhs==clos[0][k].lhs &&
strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)
								break;
						if(l==n)
						{
							clos[noitem][n].lhs=clos[0][k].lhs;
						strcpy(clos[noitem][n].rhs,clos[0][k].rhs);
							n=n+1;
						}
					}
				}
			}
		}
	}
	arr[noitem]=n;
	int flag=0;
	for(i=0;i<noitem;i++)
	{
		if(arr[i]==n)
		{
			for(j=0;j<arr[i];j++)
			{
				int c=0;
```

```
                                        for(k=0;k<arr[i];k++)
                                                if(clos[noitem][k].lhs==clos[i][k].lhs &&
strcmp(clos[noitem][k].rhs,clos[i][k].rhs)==0)
                                                        c=c+1;
                                        if(c==arr[i])
                                        {
                                                flag=1;
                                                goto exit;
                                        }
                                }
                        }
                }
                exit:;
                if(flag==0)
                        arr[noitem++]=n;
}



void main()
{
                clrscr();
                cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";
                do
                {
                        cin>>prod[i++];
                }while(strcmp(prod[i-1],"0")!=0);
                for(n=0;n<i-1;n++)
                {
                        m=0;
                        j=novar;
                        g[novar++].lhs=prod[n][0];
```

```
		for(k=3;k<strlen(prod[n]);k++)
		{
			if(prod[n][k] != '|')
			g[j].rhs[m++]=prod[n][k];
			if(prod[n][k]=='|')
			{
				g[j].rhs[m]='\0';
				m=0;
				j=novar;
				g[novar++].lhs=prod[n][0];
			}
		}
}
for(i=0;i<26;i++)
		if(!isvariable(listofvar[i]))
			break;
g[0].lhs=listofvar[i];
char temp[2]={g[1].lhs,'\0'};
strcat(g[0].rhs,temp);
cout<<"\n\n augumented grammar \n";
for(i=0;i<novar;i++)
		cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";
getch();
for(i=0;i<novar;i++)
{
		clos[noitem][i].lhs=g[i].lhs;
		strcpy(clos[noitem][i].rhs,g[i].rhs);
		if(strcmp(clos[noitem][i].rhs,"ε")==0)
			strcpy(clos[noitem][i].rhs,".");
		else
```

```
		{
			for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)
				clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];
			clos[noitem][i].rhs[0]='.';
		}
}
arr[noitem++]=novar;
for(int z=0;z<noitem;z++)
{
		char list[10];
		int l=0;
		for(j=0;j<arr[z];j++)
		{
			for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
			{
				if(clos[z][j].rhs[k]=='.')
				{
					for(m=0;m<l;m++)
						if(list[m]==clos[z][j].rhs[k+1])
							break;
					if(m==l)
						list[l++]=clos[z][j].rhs[k+1];
				}
			}
		}
		for(int x=0;x<l;x++)
			findclosure(z,list[x]);
}
cout<<"\n THE SET OF ITEMS ARE \n\n";
for(z=0;z<noitem;z++)
```

```
        {
                cout<<"\n I"<<z<<"\n\n";
                for(j=0;j<arr[z];j++)
                        cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";
                getch();
        }
        getch();
}
```

Output:-

```
ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
0

augumented grammar

A->E
E->E+T
E->T
T->T*F
T->F
F->(E)
```

F->i

THE SET OF ITEMS ARE

I0

A->.E

E->.E+T

E->.T

T->.T*F

T->.F

F->.(E)

F->.i

I1

A->E.

E->E.+T

I2

E->T.

T->T.*F

I3

T->F.

I4

F->(.E)

E->.E+T

E->.T

T->.T*F

T->.F

F->.(E)

F->.i

I5

F->i.

I6

E->E+.T

T->.T*F

T->.F

F->.(E)

F->.i

I7

T->T*.F

F->.(E)

F->.i

I8

F->(E.)

E->E.+T

I9

E->E+T.

T->T.*F

I10

T->T*F.

I11

F->(E).

Result:-

The program was successfully compiled and run.

EXPERIMENT-11 INTERMEDIATE CODE GENERATION

Aim: A program to implement Intermediate Code Generation

Algorithm:-

1.Start.

2.Initialize required variables.

3.A quadruple has four fields op,arg1,arg2,result.

4.Triple has only three fields op,arg1,arg2.

5.Assign a value to the variables given.

6.Assign a register to variables and perform arithmetic operations.

7.Move the final result to assign variable.

8. Display the output.

9. End.

Program:

```
#include<iostream.h>
#include<string.h>
#include<conio.h>
char reg[10][3]={"R0","R1","R2","R3","R4","R5"};
char stmt[10][10],code[15];
int nostmt=0,i=0,output[15];
void icode(char source[10],char dest[10],int out)
{
        strcat(code,source);
        strcat(code," ");
        strcat(code,dest);
        output[i]=out;
        cout<<code<<endl;
        getch();
}

void main()
{
        clrscr();
        cout<<" Enter the statements(END to end): \n";
        do
        {
                cin>>stmt[nostmt++];
        }while(strcmp(stmt[nostmt-1],"END")!=0);
        nostmt=nostmt-1;
        cout<<"\n THE INTERMEDIATE CODE IS\n\n";
```

```
for(i=0;i<nostmt;i++)
{
	strcpy(code,"");
	int rd=-1,rs=-1,k;
	for(int j=0;j<i;j++)
	{
		if(stmt[j][0]==stmt[i][2])
			rs=output[j];
		if(stmt[j][0]==stmt[1][4])
			rd=output[j];
	}
	if(rs==-1)
	{
		strcpy(code,"MOV ");
		char temp[2]={stmt[i][2],'\0'};
		icode(temp,reg[i],i);
	}
	if(stmt[i][3]=='+')
		strcpy(code,"ADD ");
	if(stmt[i][3]=='-')
		strcpy(code,"SUB ");
	if(stmt[i][3]=='*')
		strcpy(code,"MUL ");
	if(stmt[i][3]=='/')
		strcpy(code,"DIV ");
	if(rd==-1)
	{
		char temp[2]={stmt[i][4],'\0'};
		if(rs!=-1)
			k=output[rs];
```

```
                else
                        k=i;
                icode(temp,reg[k],k);
        }
        if(rs!=-1 && rd!=-1)
        {
                int flag=0;
                for(j=i;j<nostmt;j++)
                        if(stmt[j][2]==stmt[i][2] || stmt[j][2]==stmt[i][4])
                flag=1;
                if(flag!=1)
                        icode(reg[output[rs]],reg[output[rd]],output[rd]);
                if(flag==1)
                        icode(reg[output[rd]],reg[output[rs]],output[rs]);
        }
    }
    strcpy(code,"MOV ");
    char temp[2]={stmt[i-1][0],'\0'};
    icode(reg[output[i-1]],temp,0);
}
```

Output:-

```
Enter the statements(END to end):
t=a+b
t=a*b
END
THE INTERMEDIATE CODE IS
MOV a R0
ADD b R0
MOV a R1
MUL b R1
```

MOV R1 t

Result:-

The program was successfully compiled and run.

****