PROGRAM

```
#include<stdio.h>
#include<string.h>
typedef struct DFA{
        int nos;
        int noi;
        int nof;
        int delta[10][10];
        int final[10];
        char inputSymbols[10];
}DFA;

int checkSymbol(char ch,DFA d)
{
        for(int i=0;i<d.noi;i++)
        {
                if(ch==d.inputSymbols[i])
                {
                        return i;
                }
        }
        return -1;
}

int checkFinalState(int st,DFA d)
{
        for(int i=0;i<d.nof;i++)
        {
                if(st==d.final[i]){
                        return 1;
                }
        }
        return 0;
}

int main(){
        DFA d;
        printf("Enter number of states: ");
        scanf("%d",&d.nos);
        printf("Enter number of final states: ");
        scanf("%d",&d.nof);
        printf("Enter number of input symbols: ");
        scanf("%d",&d.noi);

        for(int i=0;i<d.noi;i++){
                printf("Enter input symbol: ");
                scanf(" %c",&d.inputSymbols[i]);
        }

        for(int i=0;i<d.nof;i++)
        {
                printf("Enter final state no %d: ",i+1);
                scanf("%d",&d.final[i]);
        }

        printf("\nEnter transitions: ");
        for(int i=0;i<d.nos;i++){
                for(int j=0;j<d.noi;j++){
                        printf("\nd(q%d,%c): ",i,d.inputSymbols[j]);
                        scanf("%d",&d.delta[i][j]);

                }
        }
         for(int i=0;i<d.noi;i++){
 printf("\t %c",d.inputSymbols[i]);}
printf("\n");


        for(int i=0;i<d.nos;i++){
                printf("\nq%d",i);
                for(int j=0;j<d.noi;j++){
                        printf("\t%d",d.delta[i][j]);
                }
                printf("\n");
        }

        do{
                char string[10];
                printf("\nEnter a string: ");
                scanf("%s",string);
                int statecounter=0;
                int flag=1;

                for(int i=0;i<strlen(string);i++){
                        int sympos=checkSymbol(string[i],d);
                        if(sympos==-1){
                                flag=0;
                                break;
                        }

        statecounter=d.delta[statecounter][sympos];

                }
                if(flag==1 && checkFinalState(statecounter,d)==1){
                        printf("%s is accepted.\n",string);
                }
                else{
                        printf("%s is not accepted",string);
                }
        }while(1);

        return 0;
}
```

OUTPUT

```
Enter number of states: 3
Enter number of final states: 1
Enter number of input symbols: 2
Enter input symbol: a
Enter input symbol: b
Enter final state no 1: 2

Enter transitions:
d(q0,a): 1

d(q0,b): 0

d(q1,a): 1

d(q1,b): 2

d(q2,a): 2

d(q2,b): 2
        a       b

q0      1       0

q1      1       2

q2      2       2

Enter a string: aab
aab is accepted.
```

PROGRAM

```
#include<stdio.h>
#include<string.h>
char result[20][20], copy[3], states[20][20];
void add_state(char a[3],int i){
        strcpy(result[i],a);
}

void display(int n){
        int k=0;
        printf("nnn Epsilon closure of %s ={ ",copy);
        while(k<n){
                printf(" %s",result[k]);
                k++;
        }
        printf("} nnn");
}

int main(){
        FILE *INPUT;
        INPUT=fopen("input.dat","r");
        char state[3];
        int end,i=0,n,k=0;
        char state1[3],input[3],state2[3];
        printf("n Enter the no of states: ");
        scanf("%d", &n);
        printf("n Enter the states n");
        for(k=0;k<3;k++){
                scanf("%s",states[k]);
        }

        for(k=0;k<n;k++){
                i=0;
                strcpy(state,states[k]);
                strcpy(copy,state);
                add_state(state,i++);
                while(1){
                        end=fscanf(INPUT,"%s%s
%s",state1,input,state2);
                        if(end==EOF){
                        break;
                        }
                        if(strcmp(state,state1)==0){

        if(strcmp(input,"e")==0){

        add_state(state2,i++);

        strcpy(state,state2);
                                }
                        }
                }
                display(i);
                rewind(INPUT);
        }
        return 0;
}
```

OUTPUT

```
n Enter the no of states: 3
n Enter the states n
qo
q1
q2
nnn Epsilon closure of qo ={  qo} nnn
nnn Epsilon closure of q1 ={  q1 q2} nnn
nnn Epsilon closure of q2 ={  q2} nnn
```

PROGRAM

```
#include<stdio.h>
char input[100];
int i,error;

int main(){
	printf("Recursive descent parsing for grammar\n");
	printf("E->TE'\nE'->+TE'/@\nT->FT'\nT'->*FT'/@\nF->(E)/id\n");
	gets(input);

	E();
	if(input[i]=='\0' && error==0){
		printf("String is accepted");
	}
	else{
		printf("String rejected");
	}
}

void E(){
	T();
	Eds();
}

void T(){
	F();
	Tds();
}

void Eds(){
	if(input[i]=='+')
	{
		i++;
		T();
		Eds();
	}
}

void Tds(){
	if(input[i]=='*'){
		i++;
		F();
		Tds();

	}
}

void F(){
	if(input[i]=='('){
		i++;
		E();
		i++;
	}
	else if(input[i]=='i')
	{
		i++;
	}
	else{
		error=1;
	}
}
```

OUTPUT

```
Recursive descent parsing for grammar
E->TE'
E'->+TE'/@
T->FT'
T'->*FT'/@
F->(E)/id
i+i*i
String is accepted
```

PROGRAM

```
#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\nE->E*E\nE-
>(E)\nE->id");
	puts("Enter input string: ");
	gets(a);
	c=strlen(a);
	strcpy(act,"SHIFT->");
	puts("Stack\tinput\taction");
	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->E+E
E->E*E
E->(E)
E->id
Enter input string:
id+id*id
Stack   input   action

$id       +id*id$        SHIFT->id
$E        +id*id$|tREDUCE TO E
$E+        id*id$        SHIFT->symbols
$E+id        *id$ SHIFT->id
$E+E         *id$|tREDUCE TO E
$E           *id$ REDUCE TO E
$E*           id$  SHIFT->symbols
$E*id          $   SHIFT->id
$E*E           $|tREDUCE TO E
$E             $   REDUCE TO E
```

PROGRAM

```
#include<stdio.h>
#include<string.h>
struct operation{
        char left;
        char right[20];
};

struct operation operations[10],optimized[10];
int main(){
        int numOperations,optimizedCount=0;
        printf("Enter the number of values: ");
        scanf("%d",&numOperations);

        for(int i=0;i<numOperations;i++){
                printf("left: ");
                scanf(" %c",&operations[i].left);
                printf("right: ");
                scanf("%s",operations[i].right);
        }

        printf("Intermediate code: \n");
        for(int i=0;i<numOperations;i++){
                printf("%c= %s\
n",operations[i].left,operations[i].right);

        }

        for(int i=0;i<numOperations-1;i++){
                char temp=operations[i].left;
                for(int j=0;j<numOperations;j++){

        if(strchr(operations[j].right,temp)){

optimized[optimizedCount].left=operations[i].left;

strcpy(optimized[optimizedCount].right,operations[i].
right);
                                optimizedCount++;
                        }
                }

        }

optimized[optimizedCount].left=operations[numOper
ations-1].left;

strcpy(optimized[optimizedCount].right,operations[nu
mOperations-1].right);
        optimizedCount++;

        printf("\nAfter dead code elimination: \n");
        for(int i=0;i<optimizedCount;i++){
                printf("%c=%s\
n",optimized[i].left,optimized[i].right);
        }

        for(int i=0;i<optimizedCount;i++){
                for(int j=i+1;j<optimizedCount;j++){


if(strcmp(optimized[i].right,optimized[j].right)==0){
                                optimized[j].left='\0';
                        }
                }
        }

        printf("Optimised code\n");
        for(int i=0;i<optimizedCount;i++){
                if(optimized[i].left!='\0'){
                        printf("%c=%s\
n",optimized[i].left,optimized[i].right);
                }

        }

        return 0;
}
```

OUTPUT

```
Enter the number of values: 5
left: a
right: id
left: b
right: id+1
left: c
right: a+b
left: d
right: c
left: e
right: id+2
Intermediate code:
a= id
b= id+1
c= a+b
d= c
e= id+2

After dead code elimination:
a=id
b=id+1
c=a+b
d=c
d=c
d=c
e=id+2
Optimised code
a=id
b=id+1
c=a+b
d=c
e=id+2
```

PROGRAM

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

#define MAX 100

struct Stack {
    int top;
    char items[MAX];
};

void push(struct Stack* s, char item) {
    if (s->top < MAX - 1) {
        s->items[++(s->top)] = item;
    }
}

char pop(struct Stack* s) {
    if (s->top >= 0) {
        return s->items[(s->top)--];
    }
    return '\0';
}

char peek(struct Stack* s) {
    if (s->top >= 0) {
        return s->items[s->top];
    }
    return '\0';
}

int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        default:
            return 0;
    }
}

void infixToPostfix(char* infix, char* postfix) {
    struct Stack s;
    s.top = -1;
    int k = 0;

    for (int i = 0; infix[i]; i++) {
        if (isalnum(infix[i])) {
            postfix[k++] = infix[i];
        } else {
            while (s.top != -1 && precedence(peek(&s))
>= precedence(infix[i])) {
                postfix[k++] = pop(&s);
            }
            push(&s, infix[i]);
        }
    }
    while (s.top != -1) {
        postfix[k++] = pop(&s);
    }
    postfix[k] = '\0';
}

void generateIntermediateCode(char* postfix) {
    struct Stack s;
    s.top = -1;
    char tmpch = 't';

    printf("The intermediate code:\n");
    for (int i = 0; postfix[i]; i++) {
        if (isalnum(postfix[i])) {
            push(&s, postfix[i]);
        } else {
            char rightOp = pop(&s);
            char leftOp = pop(&s);


            printf("\t%c := %c %c %c\n", tmpch, leftOp,
postfix[i], rightOp);
            push(&s, tmpch++);
        }
    }
}

int main() {
    char infix[MAX], postfix[MAX];

    printf("Enter the Expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    generateIntermediateCode(postfix);

    return 0;
}
```

OUTPUT

```
Enter the Expression: a+b*c
The intermediate code:
        t := b * c
        u := a + t
```

PROGRAM

```
#include<stdio.h>
#include<string.h>
void main(){
        char icode[10][30],str[20],opr[10];
        int i=0;
        printf("Enter the set of intermediate code(terminated by exit):\n");
        do{
                scanf("%s",icode[i]);

        }while(strcmp(icode[i++],"exit")!=0);
        printf("\nTarget code generation");
        printf("\n*****************************");
        i=0;
        do{
                strcpy(str,icode[i]);
                switch(str[3]){
                        case '+':
                        {
                        strcpy(opr,"ADD");
                        break;
                        }
                        case '-':
                        {
                        strcpy(opr,"SUB");
                        break;
                        }
                        case '*':
                        {
                        strcpy(opr,"MUL");
                        break;
                        }
                        case '/':
                        {
                        strcpy(opr,"DIV");
                        break;
                        }

                }
                printf("\n\tMov %c , R%d",str[2],i);
                printf("\n\t%s%c,R%d",opr,str[4],i);
                printf("\n\tMov R%d,%c",i,str[0]);

        }while(strcmp(icode[i++],"exit")!=0);

}
```

OUTPUT

```
Enter the set of intermediate code(terminated by exit):
x=a+b
y=x-c
z=y*d
p=z/e
exit

Target code generation
*****************************
        Mov a , R0
        ADD b,R0
        Mov R0,x
        Mov x , R1
        SUB c,R1
        Mov R1,y
        Mov y , R2
        MUL d,R2
        Mov R2,z
        Mov z , R3
        DIV e,R3
        Mov R3,p
```