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[] =
"+-*/%=",specialch[]=",;[]{}",num[]="1234567890",buf[10];
FILE *fp;
int i,j=0,k=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(ch == specialch[i])
{
printf("%c is special character\n", ch);
}}
if(isalpha(ch)){
buffer[j++] = ch;
}
if(isdigit(ch)){
buf[k++]=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 identifier\n", buffer);
printf("%s is constant\n", buf);
}}}
fclose(fp);
return 0;
}
```

Output:

```
int is keyword
, is special character
, is special character
= is operator
abv is identifier
1 is constant
```

Program:

```
%{
	int  op = 0,i;
	float a, b;
%}
dig [0-9]+|([0-9]*)"."([0-9]+)
add "+"
sub "-"
mul "*"
div "/"
pow "^"
ln \n
%%
{dig} {digi();}
{add} {op=1;}
{sub} {op=2;}
{mul} {op=3;}
{div} {op=4;}
{pow} {op=5;}
{ln} {  printf("\n The Answer :%f\n\n", a);     }
%%
digi()
{
	if(op==0)
		a=atof(yytext);
	else
	{	b=atof(yytext);
		switch(op)
		{	case 1:
				a=a+b;
				break;
			case 2:
				a=a-b;
				break;
			case 3:
				a=a*b;
				break;
			case 4:
```

```
                        a=a/b;
                        break;
                case 5:
                        for (i=a; b>1;b--)
                                a=a*i;
                break;
        }
        op=0;
    }
}
main(int argv, char *argc[])
{
    yylex();
}
yywrap()
{
    return 1;
}
```

Output:

```
16+7
 The Answer : 23.000000

12-45
 The Answer : -33.000000

5*4
 The Answer : 20.000000

10/2
 The Answer : 5.000000
```

Program:

```
%{
	int vow_count=0;
	int const_count =0;
%}
%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){}
int main()
{
	printf("Enter the string of vowels and consonants:");
	yylex();
	printf("Number of vowels are: %d\n", vow_count);
	printf("Number of consonants are: %d\n", const_count);
	return 0;
}
```

Output:

Enter the string of vowels and consonants : i am good

Number of vowels are : 4
Number of consonants are : 3

EXP 5 COUNTING NO OF WORDS

Program:

```
%{
#include<stdio.h>
int sc=0,wc=0,lc=0,cc=0;
%}

%%
[\n] { lc++; cc+=yyleng;}
[  \t] { sc++; cc+=yyleng;}
[^\t\n ]+ { wc++;  cc+=yyleng;}
%%

int main(int argc ,char* argv[ ])
{
	printf("Enter the input:\n");
	yylex();
	printf("The number of lines=%d\n",lc);
	printf("The number of spaces=%d\n",sc);
	printf("The number of words=%d\n",wc);
	printf("The number of characters are=%d\n",cc);
}

int yywrap( )
{
	return 1;
}
```

Output:

```
hello world

The number of lines = 1
The number of spaces = 1
The number of words = 2
The number of characters are = 12
```

Program:

```
#include<stdio.h>
#include<string.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp{
int pos;
char op;
}k[15];
void main() {
printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
printf("Enter the Expression :");
scanf("%s",str);
printf("The intermediate code:\n");
findopr();
explore();
}
void findopr(){
for(i=0;str[i]!='\0';i++)
if(str[i]==':'){
k[j].pos=i;
k[j++].op=':';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='/'){
k[j].pos=i;
k[j++].op='/';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='*'){
k[j].pos=i;
k[j++].op='*';
}
for(i=0;str[i]!='\0';i++)
```

```
if(str[i]=='+'){
k[j].pos=i;
k[j++].op='+';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='-') {
k[j].pos=i;
k[j++].op='-';
}}
void explore(){
i=1;
while(k[i].op!='\0'){
fleft(k[i].pos);
fright(k[i].pos);
str[k[i].pos]=tmpch--;
printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
printf("\n");
i++;
}
fright(-1);
if(no==0){
fleft(strlen(str));
printf("\t%s := %s",right,left);
}
printf("\t%s := %c",right,str[k[--i].pos]); }
void fleft(int x){
int w=0,flag=0;
x--;
while(x!= -1 &&str[x]!= '+'
&&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&str[x]!='/'&&str[x]!=':'){
if(str[x]!='$'&& flag==0){
left[w++]=str[x];
left[w]='\0';
str[x]='$';
flag=1; }
x--;
}}
void fright(int x){
int w=0,flag=0;
x++;
```

```
while(x!= -1 && str[x]!=
'+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/'){
if(str[x]!='$'&& flag==0){
right[w++]=str[x];
right[w]='\0';
str[x]='$';
flag=1;}
x++;
}}
```

Output:

INTERMEDIATE CODE GENERATION

Enter the Expression : w:=a*b+c/d-e/f+g*h
The intermediate code :

```
Z := c/d
Y := e/f
X := a*b
W := g*h
V := X+Z
U := Y+W
T := V-U
w := T
w := $
```

Program:

```
#include<stdio.h>
int Fa[10][10][10],states[2][10],row=0,col=0,sr=0,sc=0,th=0,
in,stat,new_state[10][10],max_inp=-1,no_stat;
FILE *fp;
int search(int search_var)
{ int i;
for(i=0;i<no_stat;i++)
if(search_var == states[1][i])
return 1;
return 0; }
int sort(int *arr,int count) {
int temp,i,j;
for(i=0;i<count-1;i++) {
for(j=i+1;j<count;j++) {
if(arr[i]>=arr[j]) {
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
} } }
return 0; }
int checkcon(int *arr,int *count) {
int i,temp,j,k,c,t,m;
for(i=0;i<*count;i++) {
if(arr[i]>row) {
temp =arr[i];
c=0;
t=0;
while(new_state[arr[i]][t]!=-1) {
t++;
c++; }
for(k=0;k<=c-2;k++) {
for(j=9;j>=i+1+k;j--) {
arr[j]=arr[j-1]; } }
t=0;
for(j=i;j<c;j++) {
arr[j]=new_state[temp][t];
t++;
```

```
} } }
c=0;
for(i=0;arr[i]!=-1;i++)
c++;
*count=c;
return 0; }
int remove_duplicate(int *arr,int *count) {
int i,j=0;
for(i=1;i<*count;i++) {
if(arr[i]!=arr[j]) {
j++;
arr[j]=arr[i]; } }
*count=j+1;
return 0; }
int check(int i ,int j,int c,int *name) {
int t,l,f;
for(l=0;l<=stat;l++) {
t=0; f=0;
while(Fa[i][j][t]!=-1) {
if(Fa[i][j][t]==new_state[l][t])
t++;
else {
f=1;
break; } }
if((t==c)&&!f) {
*name=l;
return 1; }   }
return 0; }
int trans(int i ,int j,int t,int c,int *count,int *arr) {
int k=0,co,temp;
*count=0;
for(k=0;k<c;k++) {
temp=Fa[i][j][k];
co=0;
while(Fa[temp][t][co]!=-1) {
arr[*count]=Fa[temp][t][co++];
(*count)++; } }
return 0; }
int nfa2dfa(int start,int end) {
int j,t,c,i,k,count,arr[10],name,l;
for(i=start;i<=end;i++) {
```

```
for(j=0;j<=max_inp;j++) {
c=0;t=0;
while(Fa[i][j][t]>=0) {
t++;
c++; }
if(c>1) {
if(check(i,j,c,&name)==0) {
for(k=0;k<c;k++) {
new_state[stat][k]=Fa[i][j][k];
for(l=0;states[1][l]!=-1;l++)
if(new_state[stat][k] == states[1][l]&& !search(stat))
states[1][no_stat++]=stat; }
for(t=0;t<=max_inp;t++) {
count=0;
for(k=0;k<10;k++)
arr[k]=-1;
trans(i,j,t,c,&count,arr);
checkcon(arr,&count);
sort(arr,count);
remove_duplicate(arr,&count);
for(k=0;k<count;k++)
Fa[stat][t][k]=arr[k]; }
Fa[i][j][0]=stat++;
for(t=1;t<c;t++)
Fa[i][j][t]=-1; }
else {
Fa[i][j][0]=name ;
for(t=1;t<c;t++)
Fa[i][j][t]=-1;
 } } } }
return 0; }
int main() {
int i,j,k,flag=0,start,end;
char c,ch;
fp=fopen("Nfa_ip.txt","r+");
for(i=0;i<2;i++)
for(j=0;j<10;j++)
states[i][j]=-1;
for(i=0;i<10;i++)
for(j=0;j<10;j++)
new_state[i][j]=-1;
```

```
for(i=0;i<10;i++)
for(j=0;j<10;j++)
for(k=0;k<10;k++)
Fa[i][j][k]=-1;
while(fscanf(fp,"%d",&in)!=EOF)  {
fscanf(fp,"%c",&c);
if(flag) {
states[sr][sc++]=in;
if(c=='\n') {
sr++;
sc=0; }  }
else if(c=='#') {
flag=1;
Fa[row][col][th]=in; }
else if(!flag)  {
Fa[row][col][th]=in;
if(c==',')
{  th++;  }
else if(c=='\n') {
if(max_inp<col)
max_inp=col;
col=0;
row++;
th=0; }
else if(c!=',') {
th=0;
 } } }
no_stat=0;
i=0;
while(states[1][i++]!=-1)
no_stat++;
stat=row+1;
start=0;end=row;
while(1) {
nfa2dfa(start,end);
start=end+1;
end=row;
if(start>end)
break; }
printf("\n\nDFA IS : \n\n\n");
for(i=0;i<=max_inp;i++)
```

```
printf("\t%d",i);
printf("\n");
printf("--------------------------\n");
for(i=0;i<stat;i++) {
printf("%d-> |",i);
for(j=0;j<=max_inp;j++) {
printf("%2d        ",Fa[i][j][0]); }
printf("\n"); }
printf("\n\n");
printf("Total Number Of State Is : %d \n\n",stat);
printf("Final States Are : ");
for(i=0;states[1][i]!=-1;i++)
printf("%d ",states[1][i]);
printf("\n\n");
return 0; }
```

Nfa_ip.txt

```
1,2 1
-1 2
-1 -1#
0
2
```

Output:

```
DFA IS :

        0         1
---------------------------
0-> | 3         1
1-> |-1         2
2-> |-1        -1
3-> |-1         2

Total Number Of State Is : 4
Final States Are : 2 3
```

Program:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
void input();
void output();
void change(int p,char *res);
void constant();
struct expr{
char op[2],op1[5],op2[5],res[5];
int flag;
}arr[10];
int n;
void main(){
input();
constant();
output();
}
void input(){
int i;
printf("\n\nEnter the maximum number of  expressions : ");
scanf("%d",&n);
printf("\nEnter the input : \n");
for(i=0;i<n;i++){
scanf("%s",arr[i].op);
scanf("%s",arr[i].op1);
scanf("%s",arr[i].op2);
scanf("%s",arr[i].res);
arr[i].flag=0;
}
}
void constant()
{
int i;
int op1,op2,res;
char op,res1[5];
for(i=0;i<n;i++)
{
```

```
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op,"=")==0)
{
op1=atoi(arr[i].op1);
op2=atoi(arr[i].op2);
op=arr[i].op[0];
switch(op)
{
case '+':
res=op1+op2;
break;
case '-':
res=op1-op2;
break;
case '*':
res=op1*op2;
break;
case '/':
res=op1/op2;
break;
case '=':
res=op1;
break;
}
sprintf(res1,"%d",res);
arr[i].flag=1;
change(i,res1);
}
}
}
void output()
{
int i=0;
printf("\nOptimized code is : ");
for(i=0;i<n;i++)
{
if(!arr[i].flag)
{
printf("\n%s %s %s %s",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
}
}
}
```

```
void change(int p,char *res)
{
int i;
for(i=p+1;i<n;i++)
{
if(strcmp(arr[p].res,arr[i].op1)==0)
strcpy(arr[i].op1,res);
else if(strcmp(arr[p].res,arr[i].op2)==0)
strcpy(arr[i].op2,res);
}
}
```

Output:

```
Enter the maximum number of  expressions : 4

Enter the input :
= 3 - a
+ a b t1
+ a c t2
+ t1 t2 t3

Optimized code is :
+ 3 b t1
+ 3 c t2
+ t1 t2 t3
```

Program:

[Code for YACC]

```
%{
  #include <stdio.h>%}

%token NUMBER ID%left '+' '-'
%left '*' '/'
%%


E : T       {
              printf("Result = %d\n", $$);
              return 0;
            }
T :
    T '+' T { $$ = $1 + $3; }
    | T '-' T { $$ = $1 - $3; }
    | T '*' T { $$ = $1 * $3; }
    | T '/' T { $$ = $1 / $3; }
    | '-' NUMBER { $$ = -$2; }
    | '-' ID { $$ = -$2; }
    | '(' T ')' { $$ = $2; }
    | NUMBER { $$ = $1; }
    | ID { $$ = $1; };
%%
int main() {
    printf("Enter the expression\n");
    yyparse();
}
int yyerror(char* s) {
    printf("\nExpression is invalid\n");
}
```

[Code for LEX]

```
%{
   #include "y.tab.h"
   extern yylval;
%}

%%
[0-9]+  {
            yylval = atoi(yytext);
            return NUMBER;
        }
[a-zA-Z]+ { return ID; }
[\t]+;
\n  { return 0; }
. { return yytext[0]; }

%%
```

Output:

[Output 1]:

```
Enter the expression
7*(5-3)/2
Result = 7
```

[Output 2]:

```
Enter the expression
6/((3-2)*(-5+2))
Result = -2
```

Program:

```
#include<stdio.h>
#include<stdio.h>
#include<string.h>
void main()
{
	char icode[10][30],str[20],opr[10];
	int i = 0;
	printf("\n Enter the set of intermediate code (terminated by exit):\n");
	do{
		scanf("%s",icode[i]);
	}
	while (strcmp(icode[i++],"exit") != 0);
	printf("\n target code generation");
	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], 1);
		}
		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):
a=a*b
c=f*h
g=a*h
f=Q+w
t=q-j
exit

 target code generation
        MULb,R0
        Mov R0, a
        MULh,R1
        Mov R1, c
        MULh,R2
        Mov R2, g
        ADDw,R3
        Mov R3, f
        SUBj,R4
```

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);
int count, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;
int main(int argc, char **argv)
{   int jm = 0;
    int km = 0;
    int i, choice;
    char c, ch;
    count = 8;
    strcpy(production[0], "E=TR");
    strcpy(production[1], "R=+TR");
    strcpy(production[2], "R=#");
    strcpy(production[3], "T=FY");
    strcpy(production[4], "Y=*FY");
    strcpy(production[5], "Y=#");
    strcpy(production[6], "F=(E)");
    strcpy(production[7], "F=i");
    int kay;
    char done[count];
    int ptr = -1;
     for(k = 0; k < count; k++) {
        for(kay = 0; kay < 100; kay++) {
            calc_first[k][kay] = '!';
        }   }
    int point1 = 0, point2, xxx;
```

```
for(k = 0; k < count; k++)
{   c = production[k][0];
    point2 = 0;
    xxx = 0;
    for(kay = 0; kay <= ptr; kay++)
        if(c == done[kay])
            xxx = 1;
    if (xxx == 1)
        continue;
    findfirst(c, 0, 0);
    ptr += 1;
    done[ptr] = c;
    printf("\n First(%c) = { ", c);
    calc_first[point1][point2++] = c;
    for(i = 0 + jm; i < n; i++) {
        int lark = 0, chk = 0;
        for(lark = 0; lark < point2; lark++) {
            if (first[i] == calc_first[point1][lark])
            {  chk = 1;
                break;
            } }
        if(chk == 0)
        {
            printf("%c, ", first[i]);
            calc_first[point1][point2++] = first[i];
       }    }
    printf("}\n");
    jm = n;
    point1++;
}
printf("\n");
printf("-----------------------------------------------\n\n");
char donee[count];
ptr = -1;
for(k = 0; k < count; k++) {
    for(kay = 0; kay < 100; kay++) {
        calc_follow[k][kay] = '!';
    }    }
point1 = 0;
int land = 0;
for(e = 0; e < count; e++)
```

```
    {
        ck = production[e][0];
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
            if(ck == donee[kay])
                xxx = 1;
        if (xxx == 1)
            continue;
        land += 1;
        follow(ck);
        ptr += 1;
        donee[ptr] = ck;
        printf(" Follow(%c) = { ", ck);
        calc_follow[point1][point2++] = ck;
        for(i = 0 + km; i < m; i++) {
            int lark = 0, chk = 0;
            for(lark = 0; lark < point2; lark++)
            {  if (f[i] == calc_follow[point1][lark])
                {
                    chk = 1;
                    break;
                }   }
            if(chk == 0)
            {
                printf("%c, ", f[i]);
                calc_follow[point1][point2++] = f[i];
            }   }
        printf(" }\n\n");
        km = m;
        point1++;
    } }
void follow(char c)
{
    int i, j;
    if(production[0][0] == c) {
        f[m++] = '$';
    }
    for(i = 0; i < 10; i++)
    {
        for(j = 2;j < 10; j++)
```

```
        {
            if(production[i][j] == c)
            {
                if(production[i][j+1] != '\0')
                {
                    followfirst(production[i][j+1], i, (j+2));
                }
                if(production[i][j+1]=='\0' && c!=production[i][0])
                {
                    follow(production[i][0]);
                }   }   }    }
}
void findfirst(char c, int q1, int q2)
{
    int j;
    if(!(isupper(c))) {
        first[n++] = c;
    }
    for(j = 0; j < count; j++)
    {
        if(production[j][0] == c)
        {
            if(production[j][2] == '#')
            {
                if(production[q1][q2] == '\0')
                    first[n++] = '#';
                else if(production[q1][q2] != '\0'
                        && (q1 != 0 || q2 != 0))
                {
                    findfirst(production[q1][q2], q1, (q2+1));
                }
                else
                    first[n++] = '#';
            }
            else if(!isupper(production[j][2]))
            {
                first[n++] = production[j][2];
            }
            else
            {   findfirst(production[j][2], j, 3);
            }   }   }  }
```

```
void followfirst(char c, int c1, int c2)
{
    int k;
    if(!(isupper(c)))
   f[m++] = c;
    else
    {   int i = 0, j = 1;
        for(i = 0; i < count; i++)
        {       if(calc_first[i][0] == c)
                break;
        }
        while(calc_first[i][j] != '!')
        {    if(calc_first[i][j] != '#')
            {   f[m++] = calc_first[i][j];
            }
            else
            {       if(production[c1][c2] == '\0')
                {   follow(production[c1][0]);     }
                else
                {  followfirst(production[c1][c2], c1, c2+1);  }     }
            j++;
        }   }  }
```

Output:

```
First(E) = { (, i, }
First(R) = { +, #, }
First(T) = { (, i, }
First(Y) = { *, #, }
First(F) = { (, i, }

-----------------------------------------

 Follow(E) = { $, ),  }
 Follow(R) = { $, ),  }
 Follow(T) = { +, $, ),  }
 Follow(Y) = { +, $, ),  }
 Follow(F) = { *, +, $, ),  }
```

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();
int 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->E+E
E->E*E
E->(E)
E->id

enter input string :
id+id*id+id

stack	input	action
$id	+id*id+id$	SHIFT->id
$E	+id*id+id$	REDUCE TO E
$E+	id*id+id$	SHIFT->symbols
$E+id	*id+id$	SHIFT->id
$E+E	*id+id$	REDUCE TO E
$E	*id+id$	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