## CODE

### Lex file

```
%{
    #include<stdio.h>
    #include<string.h>
    #include<stdlib.h>
    #include "ifelse.tab.h"
%}
st [a-z0-9]*
%%
if return IF;
else return ELSE;
{st} return STATEMENT;
[<,>,<=,>=,==,!=] return RELOP;
[(,),{,},;] return yytext[0];
[\t] ;
[\n] return NL;
. yyerror();
%%
int yywrap(){
    return 1;
}
```

### Yacc file

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int yylex();
extern char* yytext;
int yyerror();
```

```
int flag = 0;
extern FILE* yyin;
%}
%token IF ELSE STATEMENT CONDITION RELOP NL
%%
E: S E
| S {
    return 0;
};
S: B
| B ELSE'{'NL S NL'}'
| A
;
B: IF'(' C ')''{'NL S NL'}'
;
C: STATEMENT RELOP STATEMENT
;
A: STATEMENT';'
;
%%
//driver code
void main()
{
    FILE* f1 = fopen("try.txt", "r");
    if(f1==NULL){
        printf("Error while openeing file\n");
        exit(0);
    }
    yyin = f1;
    yyparse();
    if(flag==0)
    printf("\nEntered syntax is Valid\n\n");
```

```
}

int yyerror()
{
printf("\nEntered syntax is Invalid %s\n\n", yytext);
flag = 1;
return 0;
}
```

## INPUT FILE – try.txt

```
if(c<d){
stmt;
}
else{
stmt;
}
```

## OUTPUT

```
PS C:\Users\user\Desktop\C> bison -d ifelse.y
PS C:\Users\user\Desktop\C> flex ifelse.l
PS C:\Users\user\Desktop\C> gcc ifelse.tab.c lex.yy.c
ifelse.l: In function 'yylex':
ifelse.l:16:1: warning: implicit declaration of function 'yyerror';
   16 | . yyerror();
      | ^~~~~~~
      | perror
PS C:\Users\user\Desktop\C> ./a

Entered syntax is Valid

PS C:\Users\user\Desktop\C>
```

## CODE

### Lex file

```
%{
#include "ast2.tab.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int LineNo=1;
%}

identifier [a-zA-Z][_a-zA-Z]*
number [0-9]+
%%

{identifier} {return VARIABLE;}
{number} {return NUM;}
"="  {return ASSIGNOP;}
"+" {return ADD;};
"*" {return MUL;};
"-" {return SUB;};
"/" {return DIV;};

\t  ;
\n  LineNo++;
.   yytext[0];
%%
int yywrap(void){
    return 1;
}
```

### Yacc file

```
%{
```

```
    #include<string.h>
    #include<stdio.h>
    #include <stdlib.h>
    int yylex();
    extern char* yytext;
    int yyerror();
    extern FILE *yyin;
    struct quad{
        char op[5];
        char arg1[10];
        char arg2[10];
        char result[10];
    }QUAD[30];

    int Index=0,tIndex=0;

    extern int LineNo;
    void AddQuadruple(char op[5],char arg1[10],char arg2[10],char
result[10]) {
        // Add code here to add a new quadruple to the QUAD array
        strcpy(QUAD[Index].op,op);
        strcpy(QUAD[Index].arg1,arg1);
        strcpy(QUAD[Index].arg2,arg2);
        sprintf(QUAD[Index].result,"t%d",tIndex++);
        strcpy(result,QUAD[Index++].result);
    }
%}
%union
{
    char var[10];
}
%token <var> NUM VARIABLE RELOP ARTHOP ASSIGNOP ADD MUL SUB DIV
```

```
%type <var> START EXPR TERM PRIMARY

%left '-' '+'

%left '*' '/'

%%
START: PRIMARY ASSIGNOP EXPR {strcpy(QUAD[Index].op,"=");
    strcpy(QUAD[Index].arg1,$3);
    strcpy(QUAD[Index].arg2," ");
    strcpy(QUAD[Index].result,$1);
    strcpy($$,QUAD[Index++].result);
        }
;
EXPR: EXPR ADD TERM {AddQuadruple("+",$1,$3,$$);}
| EXPR SUB TERM {AddQuadruple("-",$1,$3,$$);}
| TERM
;
TERM: TERM MUL PRIMARY {AddQuadruple("*",$1,$3,$$);}
| TERM DIV PRIMARY {AddQuadruple("*",$1,$3,$$);}
| PRIMARY
;
PRIMARY: VARIABLE {strcpy($$, yytext);}
| NUM  {strcpy($$, yytext);}
;

%%

int main(){
    FILE *fp;
    int i;
```

```
    fp=fopen("arth.txt","r");
    if(!fp){
        printf("\n File not found");
        exit(0);
    }
    yyin=fp;
    yyparse();
    printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator
\tArg1 \tArg2 \tResult" "\n\t\t--------------------");
    for(i=0;i<Index;i++){
        printf("\n\t\t %d\t %s\t %s\t
%s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
    }
    printf("\n\t\t ----------------------");
    printf("\n\n");
    return 0;
}


int yyerror(){
    printf("\n Error on line no:%d %s",LineNo, yytext);
}
```

## INPUT FILE – arth.txt

```
X=b+c-d*e+1
```

## OUTPUT

```
PS C:\Users\user\Desktop\C> bison -d ast2.y
PS C:\Users\user\Desktop\C> flex ast2.l
PS C:\Users\user\Desktop\C> gcc ast2.tab.c lex.yy.c
PS C:\Users\user\Desktop\C> ./a


                ----------------------------
                Pos Operator   Arg1    Arg2    Result
                --------------------
                0       +       b       c       t0
                1       *       d       e       t1
                2       -       t0      t1      t2
                3       +       t2      1       t3
                4       =       t3              X
                ----------------------

PS C:\Users\user\Desktop\C>
```

## CODE

```
//shift-reduce parser
//The chosen grammar is
//(1)E' → E
//(2)E → aEa
//(3)E → b
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int i, c, flag=1,k=0;
char a[16], stk[15];
char symbols[] = {'a', 'b', '$', 'E'};

struct parsing_table{
    char action[10];
    int state;

};
struct parsing_table table[10][10];

//function to find the index of the character
int findindex(char c){
    for(int i=0;i<4;i++){
        if(c == symbols[i]){
            return i;
        }
    }
    return -1;
}

//function for goto
```

```
void gotofunc(){
    int table_y = findindex(stk[k]);
    int table_x = stk[k-1] - '0';
    if(strcmp(table[table_x][table_y].action, "goto") == 0){
        stk[++k] = table[table_x][table_y].state + '0';
    }
    else{
        printf("Error in function!");
        exit(0);
    }
}

//function to determine what action to perform
void take_action(int index){
    printf("\n%s\t\t%s\t\t", stk, a);
    int table_y = findindex(a[index]);
    int table_x = stk[k]-'0';
    if(strcmp(table[table_x][table_y].action, "accept") == 0){
        printf("Accept");
        flag = 1;
        return;
    }
    else if(strcmp(table[table_x][table_y].action, "shift") == 0){
        stk[++k] = a[index];
        stk[++k] = table[table_x][table_y].state + '0';
        a[index] = ' ';
        printf("Shift to state %d", table[table_x][table_y].state);
    }
    else if(strcmp(table[table_x][table_y].action, "reduce") == 0){
        if(table[table_x][table_y].state == 2){
            for(int l=0;l<5;l++){
                stk[k] = ' ';
```

```
                k--;
            }
            stk[k] = 'E';
            printf("REDUCE E --> aEa\n");
            gotofunc();
            take_action(index);
        }
        else if(table[table_x][table_y].state == 3){
            stk[k]=' ';
            k = k-1;
            stk[k] = 'E';
            printf("REDUCE E --> b\n");
            gotofunc();
            take_action(index);
        }
    }
    else{
        printf("ERROR!\n");
        flag = 0;
        return;
    }
}

int main()
{
    char* in_str;
    printf("GRAMMAR is -\nE->aEa \nE->b\n");
    //     (0)     (1)     (2)     (3)
    //      a       b       $   |   E
    //0     s2      s3          |   1
    //1                     ac  |
    //2     s2      s3          |   4
```

```
//3     r3              r3  |
//4     s5                  |
//5     r2              r2  |
//Initialize parsing table with values
strcpy(table[0][0].action, "shift");
table[0][0].state = 2;
strcpy(table[0][1].action, "shift");
table[0][1].state = 3;
strcpy(table[0][3].action, "goto");
table[0][3].state = 1;
strcpy(table[1][2].action, "accept");
table[1][2].state = 0;
strcpy(table[2][0].action, "shift");
table[2][0].state = 2;
strcpy(table[2][1].action, "shift");
table[2][1].state = 3;
strcpy(table[2][3].action, "goto");
table[2][3].state = 4;
strcpy(table[3][0].action, "reduce");
table[3][0].state = 3;
strcpy(table[3][2].action, "reduce");
table[3][2].state = 3;
strcpy(table[4][0].action, "shift");
table[4][0].state = 5;
strcpy(table[5][0].action, "reduce");
table[5][0].state = 2;
strcpy(table[5][2].action, "reduce");
table[5][2].state = 2;

printf("Enter input string:\t");
scanf("%s", in_str);
strcpy(a,strcat(in_str,"$"));
```

```
    c=strlen(a);
    stk[k] = '0';
    printf("\nstack \t\t input \t\t action");

    for(i = 0; i < c; i++){
        // print the values of stack and input
        take_action(i);
        if(flag==0){
            break;
        }
    }

    // if top of the stack is E(starting symbol), then it will accept the
input
    if(flag == 1){
        printf("\nString accepted successfully\n");
    }
    else{
        printf("\nString rejected\n");
    }
}
```

## OUTPUT

```
PS C:\Users\user\Desktop\C> ./a
GRAMMAR is -
E->aEa
E->b
Enter input string:     aabaa

stack           input          action
0              aabaa$         Shift to state 2
0a2             abaa$         Shift to state 2
0a2a2            baa$         Shift to state 3
0a2a2b3           aa$         REDUCE E --> b

0a2a2E4           aa$         Shift to state 5
0a2a2E4a5                 a$         REDUCE E --> aEa

0a2E4                     a$         Shift to state 5
0a2E4a5                    $         REDUCE E --> aEa

0E1                        $         Accept
String accepted successfully
PS C:\Users\user\Desktop\C>
```

## CODE

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

// E=TR
// R=+TR
// R=$
// T=FY
// Y=*FY
// Y=$
// F=(E)
// F=i
int numOfProductions;
char productionSet[10][10];
//function to add a character to result
void addToResultSet(char Result[],char val)
{
    int k;
    for(k=0 ;Result[k]!='\0';k++)
        if(Result[k]==val)
            return;
    Result[k]=val;
    Result[k+1]='\0';
}

//function to find first of a symbol
void FIRST(char* Result,char c)
{
    int i,j,k;
    char subResult[20];
    int foundEpsilon;
    subResult[0]='\0';
```

```
Result[0]='\0';
//If X is terminal, FIRST(X) = {X}.
if(!(isupper(c)))
{
    addToResultSet(Result,c);
           return ;
}
//If X is non terminal
//Read each production
for(i=0;i<numOfProductions;i++)
{
//Find production with X as LHS
    if(productionSet[i][0]==c)
    {
        //If X → ε is a production, then add ε to FIRST(X).
        if(productionSet[i][2]=='#')
        addToResultSet(Result,'#');
        //If X is a non-terminal, and X → Y1 Y2 … Yk
        //is a production, then add a to FIRST(X)
        //if for some i, a is in FIRST(Yi),
        //and ε is in all of FIRST(Y1), …, FIRST(Yi-1).
        else{
            j=2;
            while(productionSet[i][j]!='\0')
            {
            foundEpsilon=0;
            FIRST(subResult,productionSet[i][j]);
            for(k=0;subResult[k]!='\0';k++)
                addToResultSet(Result,subResult[k]);
            for(k=0;subResult[k]!='\0';k++)
                if(subResult[k]=='#')
                {
```

```
                        foundEpsilon=1;
                        break;
                    }
                //No ε found, no need to check next element
                if(!foundEpsilon)
                    break;
                j++;
                }
            }
        }
    }
    return ;
}

//function to find follow of an element
void follow(char* Result,char c){
    int i,j,k;
    char subResult[20];
    int foundEpsilon;
    subResult[0]='\0';
    Result[0]='\0';
    //if start symbol add $ to result set
    if(productionSet[0][0]== c){
        addToResultSet(Result,'$');
    }
    for(i=0;i<numOfProductions;i++){
        j=2;
        //search every symbol in the RHS for a match
        while(productionSet[i][j]!='\0'){
            if(productionSet[i][j]==c){
                int limit = 1;
                //A --> pBqrs, then FOLLOW(B) = FIRST(q)
```

```
                //if FIRST(q) contains epsilon, then add to FOLLOW(B)
FIRST(r)...and so on
                while(productionSet[i][j+limit]!='\0'){
                    foundEpsilon=0;
                    FIRST(subResult, productionSet[i][j+limit]);
                    for(k=0;subResult[k]!='\0';k++){
                        if(subResult[k] != '#')
                            addToResultSet(Result,subResult[k]);
                        else{
                            limit++;
                            foundEpsilon=1;
                        }
                    }
                    if(foundEpsilon==0)
                    break;
                }
                //A --> pB or A --> pBqrs and first of q, r, s contains
epsilon, then add FOLLOW(A) to FOLLOW(B)
                //provided A!=B
                if(productionSet[i][j+limit]=='\0'){
                    subResult[0]='\0';
                    if(productionSet[i][0] != productionSet[i][j]){
                        follow(subResult, productionSet[i][0]);
                    }
                    for(k=0;subResult[k]!='\0';k++)
                    addToResultSet(Result,subResult[k]);
                }
            }
            j++;
        }
    }


}
```

```
void main()
{
    int i, k;
    char choice;
    char c;
    char result[20];
    printf("How many number of productions ? :");
    scanf(" %d",&numOfProductions);
    for(i=0;i<numOfProductions;i++)
    {
        printf("Enter productions Number %d : ",i+1);
        scanf(" %s",productionSet[i]);
    }
    do
    {
        printf("\nFIRST or FOLLOW:\n1. FIRST\n2. FOLLOW\n");
        scanf("%d", &k);
        switch(k){
            case 1:
            printf("\nFind the FIRST of  :");
            scanf(" %c",&c);
            FIRST(result,c);
            printf("\nFIRST(%c)= { ",c);
            for(i=0;result[i]!='\0';i++)
            printf(" %c ",result[i]);
            printf("}\n");
            break;

            case 2:
            printf("\nFind the FOLLOW of  :");
            scanf(" %c",&c);
            follow(result,c);
```

```
            printf("\nFOLLOW(%c)= { ",c);
            for(i=0;result[i]!='\0';i++)
            printf(" %c ",result[i]);
            printf("}\n");
            break;
        }
        printf("press 'y' to continue : ");
        scanf(" %c",&choice);
    }
    while(choice=='y'||choice =='Y');
}
```

## CODE

```
PS C:\Users\user\Desktop\C> gcc fisrtfollow.c
PS C:\Users\user\Desktop\C> ./a
How many number of productions ? :8
Enter productions Number 1 : E=TR
Enter productions Number 2 : R=+TR
Enter productions Number 3 : R=#
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=#
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=i

FIRST or FOLLOW:
1. FIRST
2. FOLLOW
1

Find the FIRST of  :E

FIRST(E)= {  (  i }
press 'y' to continue : y

FIRST or FOLLOW:
1. FIRST
2. FOLLOW
2

Find the FOLLOW of  :F

FOLLOW(F)= {  *  +  $  ) }
press 'y' to continue :
```