```c
//V3 - Using Struct and i2p Library
//Author KT (Nov2023)

#include <stdio.h>
#include <stdlib.h>
#include "i2p.h"

// Expression readexpression(void);

void displayExpression(Expression exp);
Fraction expressionValue(Expression exp);
Fraction addFractions(Fraction fr1, Fraction fr2);
Fraction subFractions(Fraction fr1, Fraction fr2);
Fraction mulFractions(Fraction fr1, Fraction fr2);
Fraction divFractions(Fraction fr1, Fraction fr2);
Fraction simplifyFraction(Fraction fr);
void displayFraction(Fraction fr);

int gcd(int a, int b);  //a, b>0  a>=b


int main(int argc, char *argv[]) {
	int numOfExpressions;  // πλήθος
	Expression exp;
	Fraction result; // result.ar, result.par

	int i;

//	printf("Enter the number of expressions to evaluate: ");
//	scanf("%d",&numOfExpressions);

//	for(i=0;i<numOfExpressions;i++)

while (1)
	{
		exp=getExpressionV2();

		if ((exp.oprtr=='q')||(exp.oprtr=='Q')) break;

		displayExpression(exp);

		result = expressionValue(exp);
		displayFraction(result);

		result=	simplifyFraction(result);
		displayFraction(result);
	}
	return 0;
}

void displayExpression(Expression exp){

	printf("%c %d/%d %d/%d \n", exp.oprtr, exp.op1.ar, exp.op1.par, exp.op2.ar,
exp.op2.par);
}
```

```
Fraction expressionValue(Expression exp){
       Fraction res;
//     printf("evaluateExpression\n");
    switch (exp.oprtr) {
          case '+':
             res=addFractions(exp.op1,exp.op2);
              break;
          case '-':
                    res=subFractions(exp.op1,exp.op2);
              break;
          case '*':
             res=mulFractions(exp.op1,exp.op2);
              break;
          case '/':
                    res=divFractions(exp.op1,exp.op2);
              break;
          default:
              printf( "Invalid operand entered.\n");
              break;
       }
       return res;
}


Fraction addFractions(Fraction fr1, Fraction fr2)
{
       Fraction fr;

       if (fr1.par==fr2.par)
       {
          fr.ar=fr1.ar+fr2.ar;
          fr.par=fr1.par;
       }
       else
       {
       fr.ar=fr1.ar*fr2.par+fr2.ar*fr1.par;

       fr.par=fr1.par*fr2.par;
     }

       return fr;
}


Fraction subFractions(Fraction fr1, Fraction fr2){
       Fraction fr;
       if(fr1.par!=fr2.par){
             fr1.ar*=fr2.par;
             fr2.ar*=fr1.par;
             fr1.par*=fr2.par;
       }
       fr.ar = fr1.ar - fr2.ar;
       fr.par=fr1.par;
       return fr;
```

```
}

Fraction mulFractions(Fraction fr1, Fraction fr2){
	Fraction fr;
	fr.ar = fr1.ar * fr2.ar;
	fr.par = fr1.par * fr2.par;
	return fr;
}

Fraction divFractions(Fraction fr1, Fraction fr2){
	Fraction fr;
	fr.ar = fr1.ar * fr2.par;
	fr.par = fr1.par * fr2.ar;
	return fr;
}

Fraction simplifyFraction(Fraction fr){
	Fraction result;

	int gcd_fraction;

	int x, y;
	// x=absolute(fr.ar),  y=absolute(fr.par)
	if (fr.ar >0) x=fr.ar; else x=-fr.ar;
	if (fr.par >0) y=fr.par; else y=-fr.par;

	if (x>y) gcd_fraction=gcd(x,y);
	else gcd_fraction=gcd(y,x);

	fr.ar=fr.ar/gcd_fraction;
	fr.par=fr.par/gcd_fraction;

	return fr;

}


void displayFraction(Fraction fr)
{
	printf("The fraction is %d/%d\n", fr.ar, fr.par);
}




/* Expression readexpression(void)
{

	printf("Give expression:  ");
	Expression value;
	getchar();
	scanf("%c", &value.oprtr);
	scanf("%d", &value.op1.ar);
	scanf("%d", &value.op1.par);
```

```
        scanf("%d", &value.op2.ar);
        scanf("%d", &value.op2.par);

        return value;
}  */

int gcd(int a, int b)  //a, b>0  a>=b
{
        int t;
        while (b!=0)
        {    t= b;
         b= a % t;
         a= t;
        }
    return a;
}
```