Звіт

# Виконання лабораторної роботи №7

Студента 2ПІ-25Б Заболотного Олександра Івановича

«Структури та об’єднання»

Тема: Структури та об’єднання

Мета: Опанувати cтворення та використання структур та об'єднань, робота зі складними структурами даних.

Задача 8. Увести структуру FRACTION (з полями чисельник і знаменник)   
для опису поняття раціональне число.   
Реалізувати функції: FRACTION add(FRACTION a, FRACTION b) FRACTION sub(FRACTION a, FRACTION b) FRACTION mul(FRACTION a, FRACTION b) FRACTION div(FRACTION a, FRACTION b) FRACTION pow(FRACTION base, FRACTION exp)   
Результат кожної операції має бути спрощено, наприклад 3/9   
спрощено до 1/3.

Текст програми:

#include <stdio.h>

#include <assert.h>

struct FRACTION {

    int numerator;

    int denominator;

};

int gcd(int a, int b) {

    if (a < 0) a = -a;

    if (b < 0) b = -b;

    while (b != 0) {

        int temp = b;

        b = a % b;

        a = temp;

    }

    return a == 0 ? 1 : a;

}

void simplify(struct FRACTION\* frac) {

    int sign = 1;

    if (frac->denominator < 0) {

        sign = -1;

        frac->denominator = -frac->denominator;

        frac->numerator = -frac->numerator;

    }

    int d = gcd(frac->numerator, frac->denominator);

    frac->numerator /= d;

    frac->denominator /= d;

}

struct FRACTION add(struct FRACTION a, struct FRACTION b) {

    struct FRACTION result;

    result.numerator = a.numerator \* b.denominator + b.numerator \* a.denominator;

    result.denominator = a.denominator \* b.denominator;

    simplify(&result);

    return result;

}

struct FRACTION sub(struct FRACTION a, struct FRACTION b) {

    struct FRACTION result;

    result.numerator = a.numerator \* b.denominator - b.numerator \* a.denominator;

    result.denominator = a.denominator \* b.denominator;

    simplify(&result);

    return result;

}

struct FRACTION mul(struct FRACTION a, struct FRACTION b) {

    struct FRACTION result;

    result.numerator = a.numerator \* b.numerator;

    result.denominator = a.denominator \* b.denominator;

    simplify(&result);

    return result;

}

struct FRACTION div(struct FRACTION a, struct FRACTION b) {

    struct FRACTION result;

    result.numerator = a.numerator \* b.denominator;

    result.denominator = a.denominator \* b.numerator;

    simplify(&result);

    return result;

}

struct FRACTION frac\_pow(struct FRACTION base, int exp) {// pow- standard function in math.h

    struct FRACTION result = {1, 1};

    while (exp > 0) {

        if (exp % 2 == 1) {

            result = mul(result, base);

        }

        base = mul(base, base);

        exp /= 2;

    }

    return result;

}

void test\_fractions() {

    struct FRACTION a = {1, 2};

    struct FRACTION b = {3, 4};

    assert(add(a, b).numerator == 5 && add(a, b).denominator == 4);

    assert(sub(a, b).numerator == -1 && sub(a, b).denominator == 4);

    assert(mul(a, b).numerator == 3 && mul(a, b).denominator == 8);

    assert(div(a, b).numerator == 2 && div(a, b).denominator == 3);

    assert(frac\_pow(a, 2).numerator == 1 && frac\_pow(a, 2).denominator == 4);

    assert(frac\_pow(b, 3).numerator == 27 && frac\_pow(b, 3).denominator == 64);

    printf("All fraction tests passed\n");

}

void demo() {

    struct FRACTION a = {1, 2};

    struct FRACTION b = {3, 4};

    struct FRACTION res\_add = add(a, b);

    printf("Addition: %d/%d + %d/%d = %d/%d\n", a.numerator, a.denominator, b.numerator, b.denominator, res\_add.numerator, res\_add.denominator);

    struct FRACTION res\_sub = sub(a, b);

    printf("Subtraction: %d/%d - %d/%d = %d/%d\n", a.numerator, a.denominator, b.numerator, b.denominator, res\_sub.numerator, res\_sub.denominator);

    struct FRACTION res\_mul = mul(a, b);

    printf("Multiplication: %d/%d \* %d/%d = %d/%d\n", a.numerator, a.denominator, b.numerator, b.denominator, res\_mul.numerator, res\_mul.denominator);

    struct FRACTION res\_div = div(a, b);

    printf("Division: %d/%d / %d/%d = %d/%d\n", a.numerator, a.denominator, b.numerator, b.denominator, res\_div.numerator, res\_div.denominator);

    int exp = 3;

    struct FRACTION res\_pow = frac\_pow(a, exp);

    printf("Exponentiation: (%d/%d)^%d = %d/%d\n", a.numerator, a.denominator, exp, res\_pow.numerator, res\_pow.denominator);

}

int main() {

    test\_fractions();

    printf("All tests completed successfully.\n");

    demo();

    return 0;

}

1. Призначення програми:

Програма призначена для роботи з раціональними числами, представленими у вигляді структури FRACTION (з полями чисельник і знаменник). Реалізовано функції для додавання, віднімання, множення, ділення та піднесення дробу до степеня. Результат кожної операції автоматично спрощується.

2. Вхідні дані:

Вхідні дані — два дроби (структури FRACTION) для виконання арифметичних операцій, а також ціле число для піднесення до степеня.

3. Алгоритм роботи програми:

• Оголошення структури FRACTION з полями numerator і denominator.

• Реалізація функції gcd для знаходження найбільшого спільного дільника.

• Реалізація функції simplify для скорочення дробу.

• Реалізація функцій add, sub, mul, div для арифметичних операцій над дробами.

• Реалізація функції frac\_pow для піднесення дробу до степеня.

• Проведення тестів для перевірки коректності роботи функцій.

• Демонстрація роботи функцій на прикладах.

4. Опис основних частин коду:

• #include <stdio.h> — для операцій введення/виведення.

• #include <assert.h> — для написання тестів.

• Оголошення структури FRACTION.

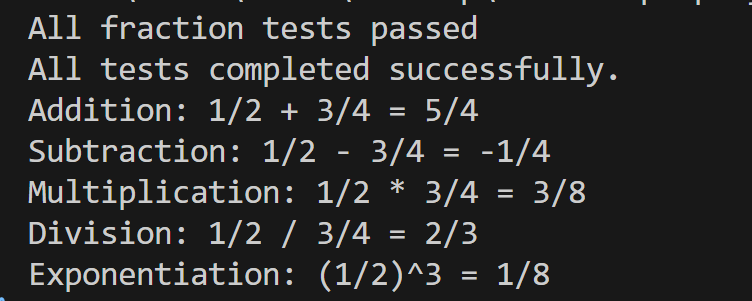
• Реалізація функцій gcd, simplify, add, sub, mul, div, frac\_pow.

• Тестова функція test\_fractions з різними випадками через assert.

• Функція demo для демонстрації роботи.

• Головна функція для запуску тестів і демонстрації.

Результат роботи:



Висновок: У ході виконання лабораторної роботи я навчився оголошувати та використовувати структури у мові C, що дозволяє створювати власні складні типи даних. На прикладі роботи з дробами я реалізував функції для виконання арифметичних операцій над раціональними числами, використовуючи структуру FRACTION.

Використання структур значно спрощує організацію та обробку даних у програмі, а отримані знання стануть основою для подальшого вивчення об'єктно-орієнтованого програмування та розробки більш складних програмних рішень.