Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский государственный технологический университет»

(ФГБОУ ВО «КубГТУ»)

Институт компьютерных систем и информационной безопасности

Кафедра информационных систем и программирования

**ОТЧЕТ**

по лабораторной работе №2.

Тема работы: «Отладка программ в интегрированных средах разработки».

Наименование дисциплины: «Тестирование и отладка программного обеспечения».

Подготовил:

Студент группы 23-КБ-ПР1

Булгаков В. В.

Краснодар

2025

**Цель работы**: изучить средства и возможности программной отладки и трассировки программ.

**Задание**

**Задание 1.**

Написать программу для приближенного вычисления значения интеграла ∫ с заранее заданной точностью a продолжаются до тех пор, пока ε Δ n2 > ε , > 0 , т.е. вычисления и прекращаются в тот момент, когда значение погрешности n2 Δ из формулы Рунге становится меньше либо равноε . В лабораторной работе читать ε = 0 . 0001 . Функцию выбрать из вариантов в разделе 5 в соответствии с номером в журнале. Для вычислений использовать формулу приближенного вычисления в соответствии с номером в журнале по модулю 3 (№%3): 0) Обобщенная формула прямоугольников. 1) Обобщенная формула Симпсона. 2) Обобщенная формула трапеций. Оценить погрешность вычислений с помощью правила Рунге. На каждой итерации производить трассировку всех значений изменяющихся переменных. Вовремя работы алгоритма проверять выход за границы интегрирования (интервал [a;b]) переменной x при помощи Assert. При вычислении значения погрешности и интеграла выводить информацию в лог трассировки на каждой итерации вычислений. При возникновении ошибок также выводить их в лог трассировки. Пусть FN – номер первой буквы имени, а LN – номер первой буквы фамилии студента. Найти значение интеграла на FN-ом шаге и вывести его 35 36 в вывод отладчика. Найти значение интеграла на LN-ом шаге и вывести его в вывод трассировщика.

**Задание 2.**

Написать программу для вычисления суммы элементов последовательности. Последовательность выбрать из вариантов в разделе 5.2 в соответствии с номером в журнале.

В программе проверить ситуацию арифметического переполнения при помощи Assert и при возникновении ошибки выводить в лог трассировки сообщение.

**Ход работы**:

Часть первая

Пример 1:

class Trace {

    static indentLevel = 0;

    static indentSize = 2;

static enabled = true;

    static getIndent() {

        return ' '.repeat(this.indentLevel \* this.indentSize);

    }

    static Indent() {

        this.indentLevel++;

    }

    static Unindent() {

        if (this.indentLevel > 0) this.indentLevel--;

    }

    static WriteLine(message) {

        if (this.enabled) {

            console.log(this.getIndent() + message);

        }

    }

    static WriteLineIf(condition, message) {

        if (this.enabled && condition) {

            console.log(this.getIndent() + message);

        }

    }

    static Assert(condition, message = "Assertion failed") {

        if (this.enabled && !condition) {

            console.error(this.getIndent() + `[ASSERTION FAILED] ${message}`);

        }

    }

    static TraceInformation(message) {

        if (this.enabled) {

            console.info(this.getIndent() + `[INFO] ${message}`);

        }

    }

}

const e = 0.0001;

const a = 0.2;

const b = 0.5;

let count = 0;

const FN = 3;

const LN = 2;

function f(x) {

    count++;

    const result = Math.atan(0.7 \* x) / (x + 1.48);

    // Проверка границ

    Trace.Assert(x >= a && x <= b, `x=${x} выходит за границы [${a}, ${b}]`);

    // Трассировка вызова

    Trace.Indent();

    Trace.WriteLine(`Вызов № ${count} x = ${x.toFixed(6)} result = ${result.toFixed(8)}---`);

    Trace.Indent();

    if (count === FN) {

        Trace.WriteLine(`Вывод на FN-м шаге = ${result.toFixed(8)}`);

    }

    if (count === LN) {

        Trace.WriteLine(`Вывод на LN-м шаге = ${result.toFixed(8)}`);

    }

    Trace.Unindent();

    Trace.Unindent();

    return result;

}

// Вычисление интеграла по формуле обобщенных трапеций

function CalculateIntegral(func, a, b) {

    let n = 1;

    let integralN = 0;

    let inacur = Infinity;

    Trace.Indent();

    Trace.TraceInformation("Начало вычисления интеграла по формуле");

    while (inacur > e) {

        n \*= 2;

        const h = (b - a) / n;

        let sum = 0;

        for (let k = 1; k <= n; k++) {

            const xPrev = a + (k - 1) \* h;

            const xCurr = a + k \* h;

            const trapezoidArea = (func(xPrev) + func(xCurr)) / 2;

            sum += trapezoidArea;

        }

        const integral2N = h \* sum;

        inacur = Math.abs(integral2N - integralN) / 3;

        Trace.WriteLine(`n = ${n} h = ${h.toFixed(6)} integralN = ${integralN.toFixed(8)} integral2N = ${integral2N.toFixed(8)} inacur = ${inacur.toFixed(8)}`);

        integralN = integral2N;

    }

    Trace.TraceInformation("Конец вычисления интеграла");

    Trace.Unindent();

    return integralN;

}

function main() {

    Trace.Indent();

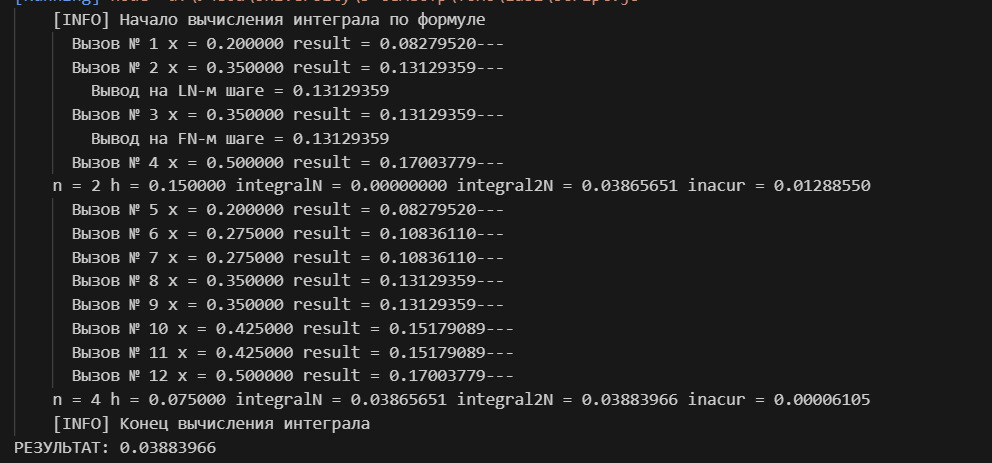
    const result = CalculateIntegral(f, a, b);

    console.log("РЕЗУЛЬТАТ: " + result.toFixed(8));

    Trace.Unindent();

}

main();



Часть вторая:

class Trace {

    static enabled = true;

    static indentLevel = 0;

    static indentSize = 2;

    static getIndent() {

        return ' '.repeat(this.indentLevel \* this.indentSize);

    }

    static Indent() {

        this.indentLevel++;

    }

    static Unindent() {

        if (this.indentLevel > 0) this.indentLevel--;

    }

    static WriteLine(message) {

        if (this.enabled) {

            console.log(this.getIndent() + message);

        }

    }

    static WriteLineIf(condition, message) {

        if (this.enabled && condition) {

            console.log(this.getIndent() + message);

        }

    }

    static Assert(condition, message = "Assertion failed") {

        if (this.enabled && !condition) {

            console.error(this.getIndent() + `[ASSERTION FAILED] ${message}`);

        }

    }

    static TraceInformation(message) {

        if (this.enabled) {

            console.info(this.getIndent() + `[INFO] ${message}`);

        }

    }

}

function calculateSequenceSum(p, q, x0, x1, n) {

    if (n < 0) {

        throw new Error("n должно быть неотрицательным");

    }

    let sum = 0;

    let xPrev2 = x0;

    let xPrev1 = x1;

    Trace.Indent();

    Trace.TraceInformation(`Вычисление суммы первых ${n} элементов последовательности`);

    Trace.WriteLine(`Начальные значения: x0 = ${x0}, x1 = ${x1}, p = ${p}, q = ${q}`);

    if (n === 0) {

        sum = x0;

        Trace.WriteLine(`Сумма = ${sum}`);

        Trace.Unindent();

        return sum;

    }

    if (n === 1) {

        sum = x0 + x1;

        Trace.WriteLine(`Сумма = ${sum}`);

        Trace.Unindent();

        return sum;

    }

    sum = x0 + x1;

    for (let i = 2; i <= n; i++) {

        const xi = p \* xPrev1 + q \* xPrev2;

        // Проверка арифметического переполнения

        Trace.Assert(

            !isNaN(xi) && isFinite(xi),

            `Арифметическое переполнение при вычислении x${i} = ${xi}`

        );

        sum += xi;

        Trace.WriteLine(`x${i} = ${xi} | Сумма = ${sum}`);

        xPrev2 = xPrev1;

        xPrev1 = xi;

    }

    Trace.WriteLine(`Итоговая сумма = ${sum}`);

    Trace.Unindent();

    return sum;

}

function main() {

    Trace.Indent();

    try {

        // p=2, q=1, x0=1, x1=1, n=5

        const result = calculateSequenceSum(2, 1, 1, 1, 5);

        console.log("РЕЗУЛЬТАТ: " + result);

    } catch (error) {

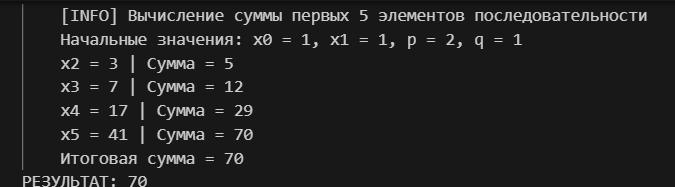
        console.error("Ошибка:", error.message);

    }

    Trace.Unindent();

}

main();



Вывод: изучены средства и возможности программной отладки и трассировки программ.