ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ
Государственное бюджетное профессиональное образовательное учреждение города Москвы «Колледж малого бизнеса № 4»
(ГБПОУ КМБ № 4)

Практическая работа №1

Специальность: 09.02.07 Информационные системы и программирование

Форма обучения: очная

Студент(ка): Уливанова София Олеговна

Группа: ИПО-21.24

Руководитель: Рыбаков Александр Сергеевич

Отчётная работа защищена с оценкой «___» _____

Москва, 2025 г.

Оглавление

ТЕМА: Ручная отладка программного обеспечения

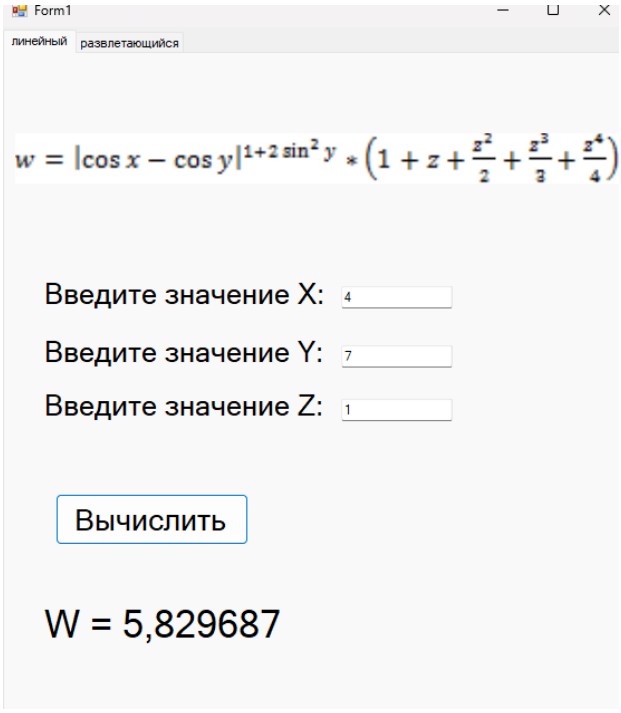
ЦЕЛЬ: Изучить процесс отладки программного обеспечения ручным методом

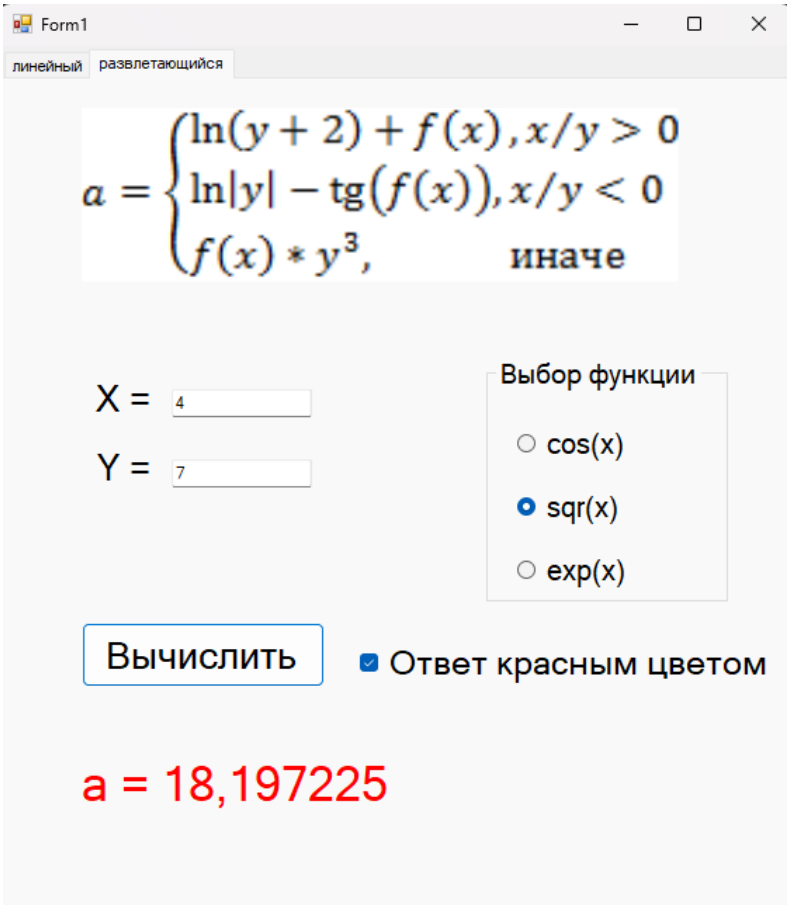
ОБОРУДОВАНИЕ:

- Персональный компьютер
- ОС Windows
- Visual Studio
- Язык программирования C#
- .NET Framework
- Windows Forms приложение

РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

Создано приложение Windows Forms, выполняющее вычисления по линейному и разветвляющемуся алгоритму.


Form1
линейный
развлетающийся
$w = |\cos x - \cos y|^{1+2\sin^2 y} * \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right)$
Введите значение X:
4
Введите значение Y:
7
Введите значение Z:
1
Вычислить
W = 5,829687


Form1
линейный
развлетающийся
$a = \begin{cases} \ln(y+2) + f(x), x/y > 0 \\ \ln|y| - \mathrm{tg}(f(x)), x/y < 0 \\ f(x) * y^3, \quad \text{иначе} \end{cases}$
X =
4
Y =
7
Выбор функции
cos(x)
sqr(x)
exp(x)
Вычислить
Ответ красным цветом
a = 18,197225

ПРАКТИЧЕСКАЯ ЧАСТЬ

Задание 1:

1. 4 Вариант
2. 1 Вариант

Задание 2: Ручная отладка

1. Математическая модель

Линейный алгоритм:

$$w = |\cos x - \cos y|^{1+2\sin^2 y} * \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right).$$

Переменные линейного алгоритма				
Название переменной	Тип данных	Описание переменной	Значение которое хранит	Ограничения
x	double	Входные данные(ввод пользователем)	input	Любое действительное число
y	double	Входные данные(ввод пользователем)	input	Любое действительное число
z	double	Входные данные(ввод пользователем)	input	Любое действительное число
w	double	Вычисляемая переменная (результат)	Output	Результат вычислений по формуле

Разветвляющийся алгоритм:

$$a = \begin{cases} \ln(y+2) + f(x), x/y > 0 \\ \ln|y| - \mathrm{tg}(f(x)), x/y < 0 \\ f(x) * y^3, \quad \text{иначе} \end{cases}$$

Где функция f(x) выбирается пользователем:

- cos(x)
- sqr(x)
- exp(x)

Название переменной	Тип данных	Описание переменной	Значение которое хранит	Ограничения
x	double	Входные данные(ввод пользователем)	input	Любое действительное число
y	double	Входные данные(ввод пользователем)	input	y ≠ 0 (деление на ноль), y > -2 (для ln(y+2))
F(x)	double	Выбор функции пользователем	Выбор из 3	Выбор из: cos(x), sqr(x), exp(x)
x/y	double	Деление вводимых пользователем переменных x и y, для определения ветви решения	X/y	Y ≠ 0

a	double	Результат	Вся формула	Для ветви с tan(f(x)): f(x) ≠ π/2 + π·k (k - целое)

2. Спецификация программы

Линейный алгоритм:

Назначение:
Программа принимает три входных значения x, y, z, выполняет вычисление по линейному алгоритму и выводит результаты.

Входные данные:
Вещественные числа x, y, z.

Выходные данные:
Результат вычисления линейного алгоритма W.

Ограничения:
Делитель не должен равняться нулю.

Разветвляющийся алгоритм:

Назначение:
Программа принимает два входных значения x, y, выполняет вычисление по разветвляющемуся алгоритму и выводит результаты.

Входные данные:
Вещественные числа x, y.

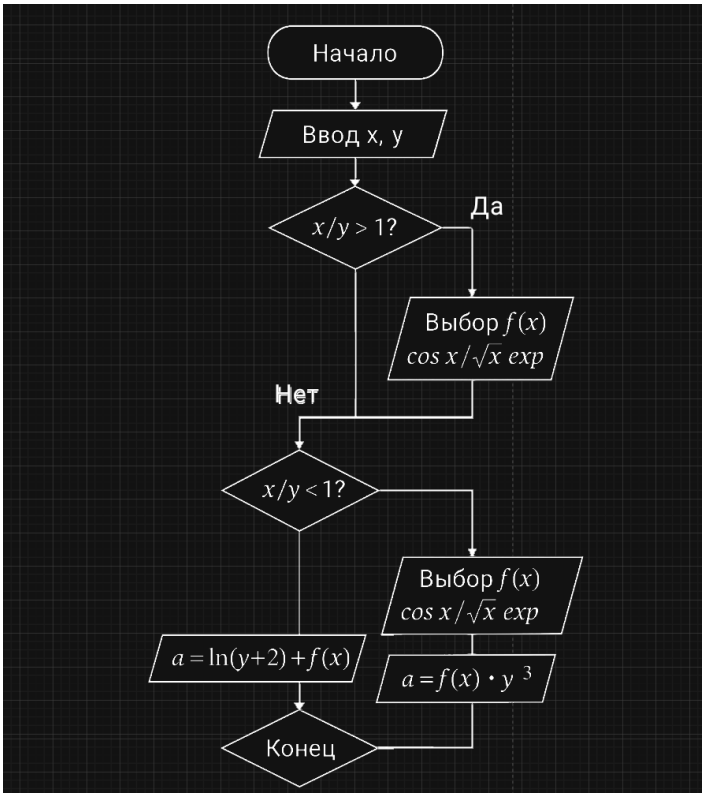
Выходные данные:
Результат вычисления разветвляющегося алгоритма a.

Ограничения:
Ветвление зависит от x, y.

3. Алгоритм программы

Линейный алгоритм:

Разветвляющийся алгоритм:



4. Отладка логики методом грубой силы

x	**y**	**z**	**Ожидаемый результат**	**Комментарий**
5	2	3	14,6	-
-8	1	4	39,5	Проверка отрицательного x
2.4	3	2	2,8	Проверка не целых чисел
0	1	1	0,5	Деление на 0
Проверка кода				
Устранение избыточных вычислений			1. Вынести повторяющиеся математические операции (например, Math.Sin(y) и Math.Cos(x))	

	в отдельные переменные для повышения производительности
Стандартизация обработки ввода	Заменить ручной парсинг на использование TryParse для предотвращения исключений при некорректном вводе
Упрощение логики проверки	Объединить условия проверки входных данных в единый блок валидации перед основными вычислениями
Выделение бизнес-логики	Создать отдельный класс или модуль для математических вычислений, отделив его от кода пользовательского интерфейса
Улучшение обработки ошибок	Заменить общие блоки catch (Exception) на конкретные обработчики для разных типов исключений (FormatException, DivideByZeroException и т.д.)

Линейный алгоритм:

x	y	z	ошибка	решение
0.5	3	3	«Введите число число»	Заменить точку на запятую при вводе
0,5	2	3	Нет ошибок	-
1	0,2	1	Нет ошибок	-
-0,5	1	4	Нет ошибок	-
2	7	3	«acos недопустим»	x должен быть в диапазоне [-1, 1] для корректного вычисления cos(x)

Разветвляющийся алгоритм:

x	y	Выбранная функция	ошибка	как решить

2	4	cos(x)	Нет ошибок	-
-3	2	sqr(x)	Нет ошибок	-
0	5	exp(x)	Нет ошибок	Проверка второй ветви
4	0	cos(x)	Деление на 0	Заменить данные Y
1	-3	sqr(x)	ln(y+2) при y ≤ -2 неопределён	Добавить проверку: если f(x) близко к π/2 + π·k
5	2	cos(x)	Если f(x) = π/2 + π·k, то tan(f(x)) неопределён	Добавить условие: если y ≤ -2, выводить ошибку "y должен быть > -2"
0.5	0.5	exp(x)	Нет ошибок	-

Задание 3:

Листинг кода:

```
using System;

using System.Windows.Forms;


namespace WindowsFormsApp1

{

    public partial class Form1 : Form

    {

        public Form1()
```

```
        {
            InitializeComponent();

            // Привязываем обработчики событий
            button1.Click += new EventHandler(button1_Click);
            button2.Click += new EventHandler(button2_Click);

            radioButton1.Checked = true;
        }

        // линейный алгоритм
        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                double x = Convert.ToDouble(textBox1.Text);
                double y = Convert.ToDouble(textBox2.Text);
                double z = Convert.ToDouble(textBox3.Text);

                // Вычисляем первую часть: |cos x - cos y|^(1+2sin²y)
                double cosDiff = Math.Abs(Math.Cos(x) - Math.Cos(y));
                double exponent = 1 + 2 * Math.Pow(Math.Sin(y), 2);
                double part1 = Math.Pow(cosDiff, exponent);

                // Вычисляем вторую часть: (1 + z + z²/2 + z³/3 + z⁴/4)
                double part2 = 1 + z + (Math.Pow(z, 2) / 2) + (Math.Pow(z,
3) / 3) + (Math.Pow(z, 4) / 4);

                double result = part1 * part2;
```

```
            label4.Text = $"W = {result:F6}";
        }
        catch (FormatException)
        {
            label4.Text = "Ошибка: введите числа!";
        }
        catch (Exception ex)
        {
            label4.Text = $"Ошибка: {ex.Message}";
        }
    }

    // разветвляющийся алгоритм
    private void button2_Click(object sender, EventArgs e)
    {
        try
        {
            double x = Convert.ToDouble(textBox4.Text);
            double y = Convert.ToDouble(textBox5.Text);

            // функция
            double f_x = 0;
            if (radioButton1.Checked) // cos(x)
            {
                f_x = Math.Cos(x);
            }
            else if (radioButton2.Checked) // sqr(x)
            {
```

```
    f_x = Math.Pow(x, 2);
}
else if (radioButton3.Checked) // exp(x)
{
    f_x = Math.Exp(x);
}
else
{
    label7.Text = "Выберите функцию!";
    return;
}

double result = 0;

// проверка деления на 0
if (y == 0)
{
    label7.Text = "Ошибка: Y не может быть 0!";
    return;
}

double ratio = x / y;

// Разветвление по условиям
if (ratio > 0)
{
    result = Math.Log(y + 2) + f_x;
}
else if (ratio < 0)
```

```
        {
            result = Math.Log(Math.Abs(y)) - Math.Tan(f_x);
        }
        else // x/y = 0 (когда x=0)
        {
            result = f_x * Math.Pow(y, 3);
        }

        // резы
        label7.Text = $"a = {result:F6}";

        // красный
        if (checkBox1.Checked)
        {
            label7.ForeColor = System.Drawing.Color.Red;
        }
        else
        {
            label7.ForeColor = System.Drawing.Color.Black;
        }
    }
    catch (FormatException)
    {
        label7.Text = "Ошибка: введите числа!";
        label7.ForeColor = System.Drawing.Color.Black;
    }
    catch (Exception ex)
    {
        label7.Text = $"Ошибка: {ex.Message}";
```

```
            label7.ForeColor = System.Drawing.Color.Black;
        }
    }

    // обработчик событий
    private void Form1_Load(object sender, EventArgs e)
    {
    }

    private void textBox1_TextChanged(object sender, EventArgs e)
    {
    }

    private void textBox2_TextChanged(object sender, EventArgs e)
    {
    }

    private void tabPage1_Click(object sender, EventArgs e)
    {
    }

    private void tabPage2_Click(object sender, EventArgs e)
    {
    }

    private void pictureBox2_Click(object sender, EventArgs e)
    {
    }
```

```
        private void label5_Click(object sender, EventArgs e)
        {
        }

        private void label5_Click_1(object sender, EventArgs e)
        {
        }

        private void tabControl1_SelectedIndexChanged(object sender,
EventArgs e)
        {
        }

        private void radioButton2_CheckedChanged(object sender, EventArgs
e)
        {
        }
    }
}
```

Результаты выполнения практического задания

Программа успешно реализована.

Интерфейс позволяет вводить переменные, выбирать функцию f(x), выполнять вычисления и выводить результат пользователю.

Тестирование показало корректность работы всех ветвей алгоритма и правильность обработки ошибок.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое ручная отладка ПО?

Ручная отладка — это процесс поиска и устранения ошибок в программном коде, выполняемый разработчиком без использования сложных автоматизированных средств. Программист вручную воспроизводит проблему, анализирует код, добавляет временные операторы вывода (например, вывод значений переменных) для отслеживания состояния программы, находит ошибочный фрагмент кода, исправляет его и проверяет результат.

2. На каком этапе выполняется ручная отладка?

Ручная отладка проводится на этапе тестирования программы после обнаружения ошибки. Она является частью этапа отладки, который следует за этапом тестирования. Разработчик может выполнять ручную отладку как при проверке отдельных модулей (модульное тестирование), так и после получения отчетов об ошибках от тестировщиков на более поздних стадиях (системное или приемочное тестирование).

3. Какие методы отладки существуют?

1. Ручная отладка — пошаговая проверка вычислений и подстановка значений в формулы вручную.

2. Отладка «грубой силой» — тестирование программы на большом наборе входных данных, включая пограничные случаи.

3. Пошаговая отладка (debugging) — использование инструментов среды разработки (IDE): установка точек останова, просмотр значений переменных, анализ стека вызовов.

4. Логирование — вывод промежуточных данных (сообщений, значений переменных) на экран или в файл для отслеживания хода выполнения программы.

5. Модульное тестирование — написание и запуск тестов для проверки корректности отдельных компонентов программы.

ВЫВОД

В ходе выполнения практической работы была создана программа на C#, реализующая линейный и разветвляющийся алгоритмы. Были изучены и применены методы ручной отладки, построены математические модели, проведено тестирование и анализ возможных ошибок. Работа позволила закрепить навыки разработки в Windows Forms, работы с типами данных, математическими функциями и условными конструкциями языка C#. Программа успешно выполняет поставленные задачи, демонстрируя корректную работу обоих алгоритмов.