Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(ФГАОУ ВО ПНИПУ)

УДК 004.652.4

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ

По дисциплине «Дискретная математика» на тему:

РАЗРАБОТКА КАЛЬКУЛЯТОРА МНОЖЕСТВ

Студент 2 курса группы РИС-24-3Б,  
специальности   
09.03.04 Программная инженерия \_\_\_\_\_\_\_\_\_\_ Егор Михайлович Носков  
  
  
Старший преподаватель

кафедры ИТАС \_\_\_\_\_\_\_\_\_\_ Рустамханова Гульшат Ильдаровна

Пермь 2025

# Постановка задачи

Разработать калькулятор множеств, который выполняет следующие последовательные действия над множествами целых чисел:

1. Позволяет задать несколько универсальных множеств (минимум 3), универсум считается от -50 до 50 несколькими способами: Ручное заполнение, случайное заполнение и заполнение по условию или группе условий.
2. Позволяет задать и провести операции между ними: объединение, пересечение, разность, симметрическая разность.

# Описание файлов и методов программы

Калькулятор множеств был разработан на языке программирования c#, вывод результатов работы программы оформлен с помощью Windows Forms в приложении Microsoft Visual Studio 2022.

1. Form1.cs - Главная форма (Приложение А).

Задача: Создание графического интерфейса пользователя и организация взаимодействия между компонентами приложения.

Основные функции:

1. Создание и настройка главного окна приложения
2. Организация интерфейса с помощью с тремя вкладками
3. Обработка пользовательского ввода и событий
4. Визуализация данных и результатов операций
5. Координация работы между различными компонентами приложения

2. Program.cs - Точка входа (Приложение Б).

Задача: Инициализация и запуск приложения.

Основные функции:

1. Содержит метод Main() - точку входа в приложение
2. Настройка параметров приложения для Windows Forms
3. Запуск главной формы приложения

3. Set.cs - Класс для работы с множествами (Приложение В).

Задача: Реализация математической структуры "множество" и основных операций над множествами.

Основные функции:

1. Хранение элементов множества в HashSet<int> для обеспечения уникальности
2. Ограничение значений элементов диапазоном от -50 до 50
3. Реализация основных операций над множествами

4. SetCalculator.cs - Логика калькулятора множеств (Приложение Г).

Задача: Управление коллекцией множеств и выполнение сложных операций.

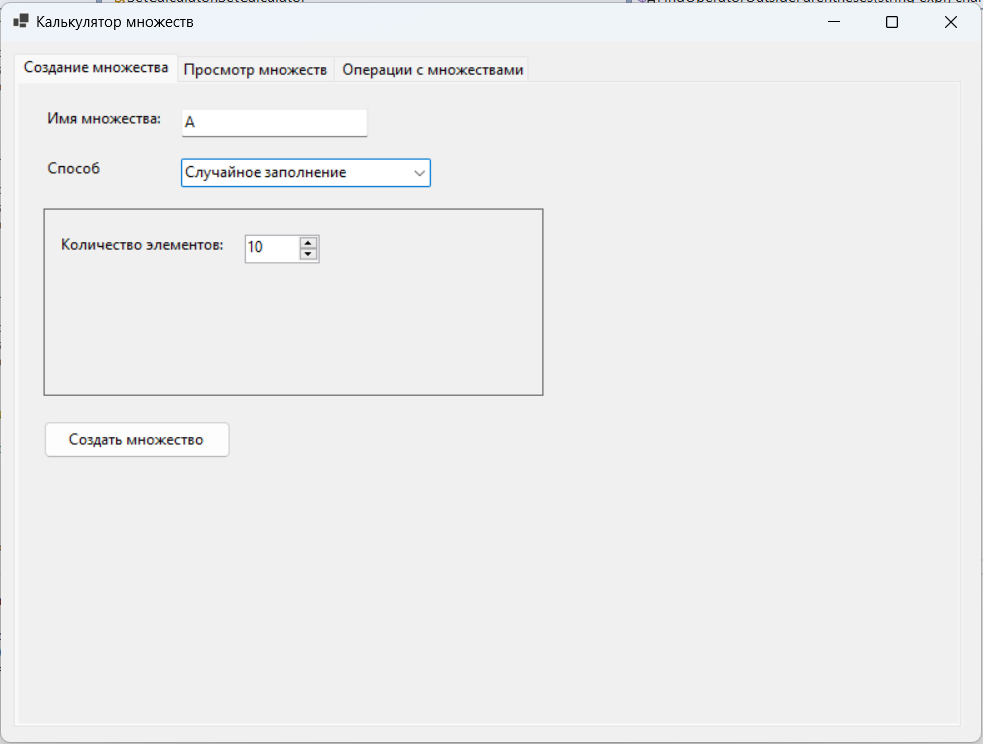
Основные функции:

1. Хранение множеств в словаре с именами в качестве ключей
2. Создание множеств различными способами:
3. Парсинг и вычисление формул с операциями над множествами
4. Управление приоритетом операций и обработка скобок

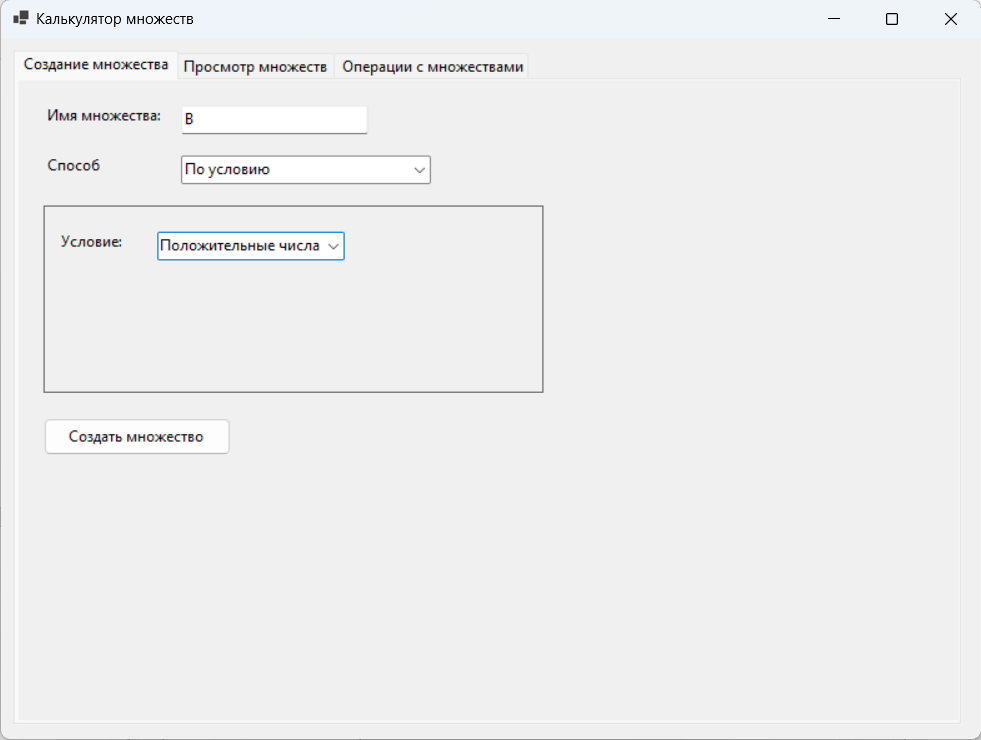
# Результат работы программы

После запуска программы калькулятора появляется окно Windows Forms с тремя вкладками:

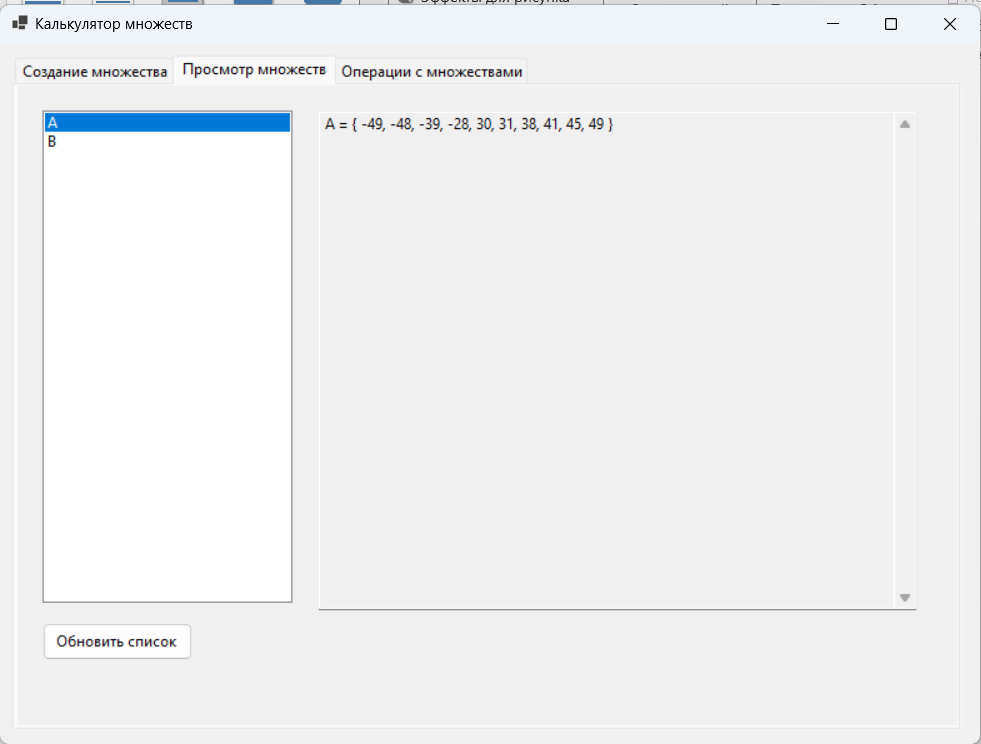
1. Создание множества (Изображения 1.1, 1.2) для создания и заполнения множества разными способами, согласно постановке задачи;
2. Просмотр множеств (Изображение 2) для просмотра и редактирования созданных и полученных в результате операций множеств;
3. Операции с множествами (Изображение 3) для проведения разных операций над созданными множествами, согласно постановке задачи.

****

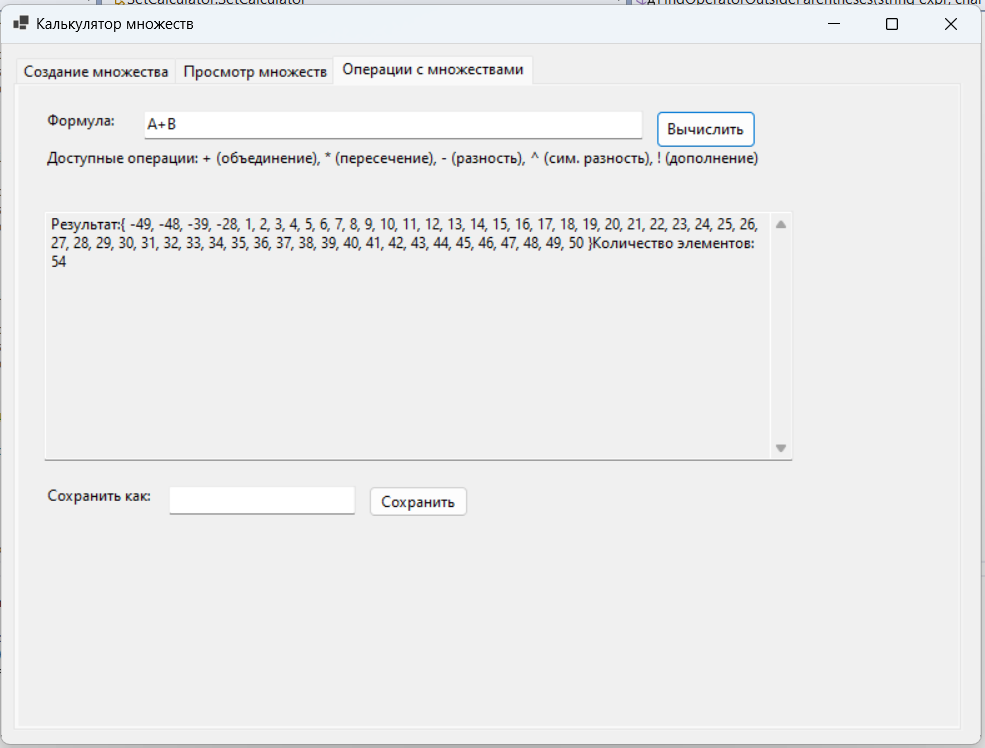
Изображение 1.1 – Создание множества

****

Изображение 1.2 – Создание множества



Изображение 2 – Просмотр множеств



Изображение 3 - Операции с множествами

# Приложение А

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace SetCalculator

{

public partial class MainForm : Form

{

private SetCalculator calculator = new SetCalculator();

public MainForm()

{

InitializeComponent();

}

private void InitializeComponent()

{

this.SuspendLayout();

// Основные настройки формы

this.Text = "Калькулятор множеств";

this.Size = new Size(800, 600);

this.StartPosition = FormStartPosition.CenterScreen;

// Создание элементов управления

CreateControls();

this.ResumeLayout(false);

}

private void CreateControls()

{

// TabControl для организации функционала

TabControl tabControl = new TabControl

{

Location = new Point(10, 10),

Size = new Size(760, 540),

Parent = this

};

// Вкладка создания множеств

TabPage createSetTab = new TabPage("Создание множества");

CreateSetTab(createSetTab);

tabControl.TabPages.Add(createSetTab);

// Вкладка просмотра множеств

TabPage viewSetsTab = new TabPage("Просмотр множеств");

CreateViewSetsTab(viewSetsTab);

tabControl.TabPages.Add(viewSetsTab);

// Вкладка операций

TabPage operationsTab = new TabPage("Операции с множествами");

CreateOperationsTab(operationsTab);

tabControl.TabPages.Add(operationsTab);

}

private void CreateSetTab(TabPage tab)

{

// Название множества

Label nameLabel = new Label

{

Text = "Имя множества:",

Location = new Point(20, 20),

Size = new Size(100, 20),

Parent = tab

};

TextBox nameTextBox = new TextBox

{

Location = new Point(130, 20),

Size = new Size(150, 20),

Parent = tab

};

// Тип создания

Label typeLabel = new Label

{

Text = "Способ создания:",

Location = new Point(20, 60),

Size = new Size(100, 20),

Parent = tab

};

ComboBox typeComboBox = new ComboBox

{

Location = new Point(130, 60),

Size = new Size(200, 20),

Parent = tab

};

typeComboBox.Items.AddRange(new string[] { "Случайное заполнение", "По условию", "Ручной ввод" });

typeComboBox.SelectedIndex = 0;

// Панель для параметров

Panel paramsPanel = new Panel

{

Location = new Point(20, 100),

Size = new Size(400, 150),

Parent = tab,

BorderStyle = BorderStyle.FixedSingle

};

// Кнопка создания

Button createButton = new Button

{

Text = "Создать множество",

Location = new Point(20, 270),

Size = new Size(150, 30),

Parent = tab

};

createButton.Click += (s, e) => CreateSet(nameTextBox.Text, typeComboBox.SelectedIndex, paramsPanel);

UpdateParamsPanel(0, paramsPanel);

typeComboBox.SelectedIndexChanged += (s, e) => UpdateParamsPanel(typeComboBox.SelectedIndex, paramsPanel);

}

private void UpdateParamsPanel(int type, Panel panel)

{

panel.Controls.Clear();

switch (type)

{

case 0: // Случайное

Label countLabel = new Label

{

Text = "Количество элементов:",

Location = new Point(10, 20),

Size = new Size(150, 20),

Parent = panel

};

NumericUpDown countNumeric = new NumericUpDown

{

Location = new Point(160, 20),

Size = new Size(60, 20),

Minimum = 1,

Maximum = Set.MaxValue - Set.MinValue + 1,

Value = 10,

Parent = panel,

Name = "paramControl"

};

break;

case 1: // По условию

Label conditionLabel = new Label

{

Text = "Условие:",

Location = new Point(10, 20),

Size = new Size(80, 20),

Parent = panel

};

ComboBox conditionComboBox = new ComboBox

{

Location = new Point(90, 20),

Size = new Size(150, 20),

Parent = panel,

Name = "paramControl"

};

conditionComboBox.Items.AddRange(new string[] { "Четные числа", "Нечетные числа", "Положительные числа", "Отрицательные числа", "Кратные n" });

conditionComboBox.SelectedIndex = 0;

// Дополнительный параметр для "Кратные n"

Label nLabel = new Label

{

Text = "n:",

Location = new Point(10, 60),

Size = new Size(30, 20),

Parent = panel,

Visible = false,

Name = "nLabel"

};

NumericUpDown nNumeric = new NumericUpDown

{

Location = new Point(40, 60),

Size = new Size(60, 20),

Minimum = -50,

Maximum = 50,

Value = 2,

Parent = panel,

Visible = false,

Name = "nControl"

};

conditionComboBox.SelectedIndexChanged += (s, e) =>

{

bool showN = conditionComboBox.SelectedIndex == 4;

nLabel.Visible = showN;

nNumeric.Visible = showN;

};

break;

case 2: // Ручной ввод

Label manualLabel = new Label

{

Text = "Элементы (через запятую):",

Location = new Point(10, 20),

Size = new Size(180, 20),

Parent = panel

};

TextBox elementsTextBox = new TextBox

{

Location = new Point(10, 50),

Size = new Size(350, 20),

Parent = panel,

Name = "paramControl"

};

break;

}

}

private void CreateViewSetsTab(TabPage tab)

{

// ListBox для отображения множеств

ListBox setsListBox = new ListBox

{

Location = new Point(20, 20),

Size = new Size(200, 400),

Parent = tab

};

// TextBox для детальной информации

TextBox infoTextBox = new TextBox

{

Location = new Point(240, 20),

Size = new Size(480, 400),

Multiline = true,

ScrollBars = ScrollBars.Vertical,

ReadOnly = true,

Parent = tab

};

// Кнопка обновления

Button refreshButton = new Button

{

Text = "Обновить список",

Location = new Point(20, 430),

Size = new Size(120, 30),

Parent = tab

};

refreshButton.Click += (s, e) =>

{

UpdateSetsList(setsListBox);

infoTextBox.Text = calculator.GetAllSetsInfo();

};

setsListBox.SelectedIndexChanged += (s, e) =>

{

if (setsListBox.SelectedItem != null)

{

infoTextBox.Text = calculator.GetSetInfo(setsListBox.SelectedItem.ToString());

}

};

UpdateSetsList(setsListBox);

infoTextBox.Text = calculator.GetAllSetsInfo();

}

private void CreateOperationsTab(TabPage tab)

{

// Формула

Label formulaLabel = new Label

{

Text = "Формула:",

Location = new Point(20, 20),

Size = new Size(80, 20),

Parent = tab

};

TextBox formulaTextBox = new TextBox

{

Location = new Point(100, 20),

Size = new Size(400, 20),

Parent = tab

};

// Info label

Label infoLabel = new Label

{

Text = "Доступные операции: + (объединение), \* (пересечение), - (разность), ^ (сим. разность), ! (дополнение)",

Location = new Point(20, 50),

Size = new Size(600, 40),

Parent = tab

};

// Кнопка вычисления

Button calculateButton = new Button

{

Text = "Вычислить",

Location = new Point(510, 20),

Size = new Size(80, 30),

Parent = tab

};

// Результат

TextBox resultTextBox = new TextBox

{

Location = new Point(20, 100),

Size = new Size(600, 200),

Multiline = true,

ScrollBars = ScrollBars.Vertical,

ReadOnly = true,

Parent = tab

};

// Сохранение результата

Label saveLabel = new Label

{

Text = "Сохранить как:",

Location = new Point(20, 320),

Size = new Size(100, 20),

Parent = tab

};

TextBox saveTextBox = new TextBox

{

Location = new Point(120, 320),

Size = new Size(150, 20),

Parent = tab

};

Button saveButton = new Button

{

Text = "Сохранить",

Location = new Point(280, 320),

Size = new Size(80, 25),

Parent = tab

};

calculateButton.Click += (s, e) =>

{

try

{

Set result = calculator.EvaluateFormula(formulaTextBox.Text);

resultTextBox.Text = $"Результат:\n{result.ToString()}\n\nКоличество элементов: {result.Count}";

}

catch (Exception ex)

{

resultTextBox.Text = $"Ошибка: {ex.Message}";

}

};

saveButton.Click += (s, e) =>

{

if (!string.IsNullOrEmpty(saveTextBox.Text))

{

try

{

Set result = calculator.EvaluateFormula(formulaTextBox.Text);

calculator.CreateSet(saveTextBox.Text, result);

UpdateSetsList();

MessageBox.Show($"Результат сохранен как '{saveTextBox.Text}'", "Успех",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

};

}

private void CreateSet(string name, int type, Panel paramsPanel)

{

if (string.IsNullOrEmpty(name))

{

MessageBox.Show("Введите имя множества", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (calculator.SetExists(name))

{

var result = MessageBox.Show($"Множество '{name}' уже существует. Перезаписать?", "Подтверждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result != DialogResult.Yes) return;

}

try

{

Set newSet;

switch (type)

{

case 0: // Случайное

var countControl = paramsPanel.Controls.Find("paramControl", true).FirstOrDefault() as NumericUpDown;

newSet = calculator.CreateRandomSet((int)countControl.Value);

break;

case 1: // По условию

var conditionComboBox = paramsPanel.Controls.Find("paramControl", true).FirstOrDefault() as ComboBox;

string condition = conditionComboBox.SelectedIndex switch

{

0 => "even",

1 => "odd",

2 => "positive",

3 => "negative",

4 => "multiple",

\_ => "even"

};

int parameter = 0;

if (condition == "multiple")

{

var nControl = paramsPanel.Controls.Find("nControl", true).FirstOrDefault() as NumericUpDown;

parameter = (int)nControl.Value;

}

newSet = calculator.CreateSetByCondition(condition, parameter);

break;

case 2: // Ручной ввод

var elementsTextBox = paramsPanel.Controls.Find("paramControl", true).FirstOrDefault() as TextBox;

var elements = elementsTextBox.Text.Split(',')

.Where(s => !string.IsNullOrWhiteSpace(s))

.Select(s => int.Parse(s.Trim()))

.ToList();

newSet = new Set(elements);

break;

default:

throw new ArgumentException("Неизвестный тип создания множества");

}

calculator.CreateSet(name, newSet);

UpdateSetsList();

MessageBox.Show($"Множество '{name}' успешно создано", "Успех",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при создании множества: {ex.Message}", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void UpdateSetsList()

{

// Обновляет списки во всех вкладках

foreach (Control control in this.Controls)

{

if (control is TabControl tabControl)

{

foreach (TabPage tabPage in tabControl.TabPages)

{

UpdateSetsListInTab(tabPage);

}

}

}

}

private void UpdateSetsList(ListBox listBox = null)

{

var setNames = calculator.GetSetNames();

if (listBox != null)

{

listBox.Items.Clear();

listBox.Items.AddRange(setNames.ToArray());

}

}

private void UpdateSetsListInTab(TabPage tabPage)

{

var listBoxes = tabPage.Controls.OfType<ListBox>();

foreach (var listBox in listBoxes)

{

UpdateSetsList(listBox);

}

}

private void UpdateSetsInfo()

{

// Можно добавить обновление информации о множествах если нужно

}

}

}

**Приложение Б**

using System;

using System.Windows.Forms;

namespace SetCalculator

{

internal static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

**Приложение В**

using System;

using System.Collections.Generic;

using System.Linq;

namespace SetCalculator

{

public class Set

{

private HashSet<int> elements = new HashSet<int>();

public static int MinValue => -50;

public static int MaxValue => 50;

private bool IsValid(int value) => value >= MinValue && value <= MaxValue;

public Set() { }

public Set(IEnumerable<int> values)

{

foreach (int value in values)

{

if (IsValid(value))

{

elements.Add(value);

}

}

}

public void Add(int value)

{

if (IsValid(value))

{

elements.Add(value);

}

}

public void Remove(int value) => elements.Remove(value);

public bool Contains(int value) => elements.Contains(value);

public int Count => elements.Count;

public bool IsEmpty => elements.Count == 0;

public void Clear() => elements.Clear();

public List<int> GetElements() => elements.ToList();

public Set UnionWith(Set other)

{

Set result = new Set();

foreach (int elem in elements) result.Add(elem);

foreach (int elem in other.elements) result.Add(elem);

return result;

}

public Set IntersectionWith(Set other)

{

Set result = new Set();

foreach (int elem in elements)

{

if (other.Contains(elem))

{

result.Add(elem);

}

}

return result;

}

public Set DifferenceWith(Set other)

{

Set result = new Set();

foreach (int elem in elements)

{

if (!other.Contains(elem))

{

result.Add(elem);

}

}

return result;

}

public Set SymmetricDifferenceWith(Set other)

{

Set unionSet = UnionWith(other);

Set intersectionSet = IntersectionWith(other);

return unionSet.DifferenceWith(intersectionSet);

}

public Set Complement()

{

Set result = new Set();

for (int i = MinValue; i <= MaxValue; i++)

{

if (!Contains(i))

{

result.Add(i);

}

}

return result;

}

public string ToString(string name = "")

{

if (elements.Count == 0)

{

return string.IsNullOrEmpty(name) ? "∅" : $"{name} = ∅";

}

string elementsStr = string.Join(", ", elements.OrderBy(x => x));

return string.IsNullOrEmpty(name)

? $"{{ {elementsStr} }}"

: $"{name} = {{ {elementsStr} }}";

}

// Операторы

public static Set operator +(Set a, Set b) => a.UnionWith(b);

public static Set operator \*(Set a, Set b) => a.IntersectionWith(b);

public static Set operator -(Set a, Set b) => a.DifferenceWith(b);

public static Set operator ^(Set a, Set b) => a.SymmetricDifferenceWith(b);

public static Set operator !(Set a) => a.Complement();

}

}

**Приложение Г**

using System;

using System.Collections.Generic;

using System.Linq;

namespace SetCalculator

{

public class SetCalculator

{

private Dictionary<string, Set> sets = new Dictionary<string, Set>();

private Random random = new Random();

public bool SetExists(string name) => sets.ContainsKey(name);

public void CreateSet(string name, Set set)

{

sets[name] = set;

}

public Set GetSet(string name) => sets.ContainsKey(name) ? sets[name] : new Set();

public Set CreateRandomSet(int count)

{

Set result = new Set();

while (result.Count < count && result.Count < (Set.MaxValue - Set.MinValue + 1))

{

result.Add(random.Next(Set.MinValue, Set.MaxValue + 1));

}

return result;

}

public Set CreateSetByCondition(string condition, int parameter = 0)

{

Set result = new Set();

switch (condition.ToLower())

{

case "even":

for (int i = Set.MinValue; i <= Set.MaxValue; i++)

if (i % 2 == 0) result.Add(i);

break;

case "odd":

for (int i = Set.MinValue; i <= Set.MaxValue; i++)

if (i % 2 != 0) result.Add(i);

break;

case "positive":

for (int i = 1; i <= Set.MaxValue; i++)

result.Add(i);

break;

case "negative":

for (int i = Set.MinValue; i < 0; i++)

result.Add(i);

break;

case "multiple":

if (parameter != 0)

for (int i = Set.MinValue; i <= Set.MaxValue; i++)

if (parameter != 0 && i % parameter == 0) result.Add(i);

break;

}

return result;

}

public List<string> GetSetNames() => sets.Keys.ToList();

public string GetSetInfo(string name)

{

return sets.ContainsKey(name) ? sets[name].ToString(name) : $"Множество '{name}' не найдено";

}

public string GetAllSetsInfo()

{

if (sets.Count == 0) return "Нет сохраненных множеств.";

var info = "Существующие множества:\n";

foreach (var pair in sets)

{

info += $" {pair.Key} ({pair.Value.Count} элементов)\n";

}

return info;

}

public Set EvaluateFormula(string formula)

{

string expression = formula.Replace(" ", "");

if (string.IsNullOrEmpty(expression))

throw new ArgumentException("Пустое выражение");

if (!CheckParentheses(expression))

throw new ArgumentException("Несбалансированные скобки в формуле");

return EvaluateExpression(expression);

}

private bool CheckParentheses(string expr)

{

int balance = 0;

foreach (char c in expr)

{

if (c == '(') balance++;

else if (c == ')') balance--;

if (balance < 0) return false;

}

return balance == 0;

}

private Set EvaluateExpression(string expr)

{

if (expr.StartsWith("(") && expr.EndsWith(")") && CheckParentheses(expr.Substring(1, expr.Length - 2)))

{

return EvaluateExpression(expr.Substring(1, expr.Length - 2));

}

// Обработка унарных операций

if (expr.StartsWith("!"))

{

string operand = expr.Substring(1);

if (!SetExists(operand))

throw new ArgumentException($"Множество '{operand}' не существует");

return !GetSet(operand);

}

// Поиск операторов в порядке приоритета

int pos;

// Симметрическая разность

if ((pos = FindOperatorOutsideParentheses(expr, '^')) != -1)

{

string left = expr.Substring(0, pos);

string right = expr.Substring(pos + 1);

return EvaluateExpression(left) ^ EvaluateExpression(right);

}

// Разность

if ((pos = FindOperatorOutsideParentheses(expr, '-')) != -1)

{

string left = expr.Substring(0, pos);

string right = expr.Substring(pos + 1);

return EvaluateExpression(left) - EvaluateExpression(right);

}

// Пересечение

if ((pos = FindOperatorOutsideParentheses(expr, '\*')) != -1)

{

string left = expr.Substring(0, pos);

string right = expr.Substring(pos + 1);

return EvaluateExpression(left) \* EvaluateExpression(right);

}

// Объединение

if ((pos = FindOperatorOutsideParentheses(expr, '+')) != -1)

{

string left = expr.Substring(0, pos);

string right = expr.Substring(pos + 1);

return EvaluateExpression(left) + EvaluateExpression(right);

}

// Если не операция, возвращаем само множество

if (!SetExists(expr))

throw new ArgumentException($"Множество '{expr}' не существует");

return GetSet(expr);

}

private int FindOperatorOutsideParentheses(string expr, char op)

{

int bracketCount = 0;

for (int i = 0; i < expr.Length; i++)

{

if (expr[i] == '(') bracketCount++;

else if (expr[i] == ')') bracketCount--;

else if (bracketCount == 0 && expr[i] == op)

return i;

}

return -1;

}

}

}

**ССЫЛКА НА GITHUB**

