



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по домашнему заданию № 1

Дисциплина: Разработка приложений на языке C#

Вариант 14

Студент

ИУ6-73Б

(Группа)

(Подпись, дата)

К.А. Логачев

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

А.М. Минитаева

(И.О. Фамилия)

Москва, 2024

Введение

Задание:

Разработать на языке C# в среде разработки Visual Studio консольное приложение «калькулятор», позволяющее вычислять арифметическое выражение, подаваемое на STDIN. Результат округлять до целого значения.

Требуется реализовать:

- сложение;
- вычитание;
- умножение;
- деление;
- поддержка скобок.

Нужно написать тесты, которые покрывают все операции.

Пример:

"(2+3)-4" => 1

"4-(2*3)" => -2

Задание по варианту: четные по списку – методом Бауэра-Земельзона.

Описание кода программы:

В этой программе реализован консольный калькулятор, использующий метод Бауэра-Земельзона для вычисления арифметических выражений. Программа поддерживает стандартные математические операции: сложение, вычитание, умножение, деление, а также скобки для задания порядка выполнения операций.

Основные части программы:

Main метод:

Основной метод программы, где происходит тестирование нескольких математических выражений.

Программа принимает выражение от пользователя и вычисляет его результат.

Метод Calculate:

Этот метод отвечает за преобразование инфиксного выражения в обратную польскую запись (ОПЗ) с последующим вычислением результата.

В этом методе сначала вызывается функция ConvertToRPN, которая преобразует выражение в ОПЗ, а затем результат передаётся в функцию EvaluateRPN, которая вычисляет итог.

Метод ConvertToRPN:

Преобразует инфиксное выражение (например, $(2+3)*5$) в обратную польскую запись.

Для этого используется стек, куда записываются операторы. Операнды сразу помещаются в выходную строку.

Метод EvaluateRPN:

Получает выражение в обратной польской записи и вычисляет его с использованием стека операндов.

Когда встречается оператор, извлекаются два последних операнда, и над ними выполняется операция.

Дополнительные методы:

IsOperator: проверяет, является ли символ оператором.

GetPrecedence: определяет приоритет оператора (например, умножение имеет более высокий приоритет, чем сложение).

PerformOperation: выполняет операции сложения, вычитания, умножения и деления.



Рисунок 1 – Схема алгоритма

Листинг программы приведен ниже:

```
using System;
using System.Collections.Generic;
```

```

namespace BauerZamelsonCalculator {
    class Program {
        static void Main(string[] args) {
            if (args.Length > 0 && args[0] == "test") {
                RunTests();
            } else {
                RunConsoleInput();
            }
        }

        static void RunTests() {
            // Тестовые данные (входные выражения и ожидаемые результаты)
            string[] inputs = { "1+2", " (234-11)*34", "6*6/6",
                                "(2+3)-4", "4-(2*3)", "10+((3*5)+2)",
                                "100/(2+3)", "5+(6*7)-(8/4)", "15-(3*2)",
                                "((3+5)*2)" };
            string[] outputs = { "3", "7582", "6", "1", "-2",
                                 "27", "20", "45", "9", "16" };

            int countTestsSuccess = 0;
            int countTestsFailed = 0;
            // Цикл по тестовым выражениям
            for (int i = 0; i < inputs.Length; i++) {
                Console.WriteLine($"Expression: {inputs[i]}");
                try {
                    // Вычисляем результат выражения
                    int result = Calculate(inputs[i]);
                    Console.WriteLine($"Result: {result}, Expected: {outputs[i]}");

                    // Сравнение результата с ожидаемым
                    if (result.ToString() == outputs[i]) {
                        Console.WriteLine("Тест пройден успешно.\n");
                        countTestsSuccess++;
                    } else {
                        Console.WriteLine("Тест не пройден.\n");
                        countTestsFailed++;
                    }
                } catch (Exception ex) {
                    // Обработка ошибок (например, некорректное выражение)
                    Console.WriteLine($"Тест не пройден: {ex.Message}\n");
                }
            }

            if (countTestsSuccess == inputs.Length) {
                Console.WriteLine("Все тесты пройдены успешно.\n");
            } else {
                Console.WriteLine(
                    $"Тесты пройдены с ошибками: {countTestsFailed} из {inputs.Length}.\n");
            }
        }

        static void RunConsoleInput() {
            Console.WriteLine(
                "Введите выражение для вычисления (или 'exit' для выхода):");
            while (true) {
                Console.Write(">> ");
                string input = Console.ReadLine();
                if (input.ToLower() == "exit") {
                    break;
                }

                try {
                    int result = Calculate(input);
                    Console.WriteLine($"Результат: {result}");
                } catch (Exception ex) {
                    Console.WriteLine($"Ошибка: {ex.Message}");
                }
            }
        }
    }
}

```

```

}

// Метод для вычисления выражения по методу Бауэра-Земельзона
static int Calculate(string expression) {
    // Шаг 1: Преобразование выражения в обратную польскую запись (ОПЗ)
    string rpn = ConvertToRPN(expression);

    // Шаг 2: Вычисление значения выражения, представленного в ОПЗ
    return EvaluateRPN(rpn);
}

// Преобразование инфиксного выражения в обратную польскую запись (ОПЗ)
static string ConvertToRPN(string expression) {
    // Стек для хранения операторов
    Stack<char> stack = new Stack<char>();
    // Строка для результата в ОПЗ
    string output = "";
    // Переменная для накопления многозначных чисел
    string number = "";

    // Проход по каждому символу выражения
    foreach (char token in expression) {
        // Если символ – цифра (накапливаем число)
        if (char.IsDigit(token)) {
            number += token;
        } else {
            // Если число накоплено, добавляем его в результат и сбрасываем
            // накопление
            if (number != "") {
                output += number + " ";
                number = "";
            }

            // Обработка скобок и операторов
            if (token == '(') {
                stack.Push(token); // Открывающая скобка помещается в стек
            } else if (token == ')') {
                // Закрывающая скобка: извлекаем операторы до открывающей скобки
                while (stack.Count > 0 && stack.Peek() != '(') {
                    output += stack.Pop() + " ";
                }
                stack.Pop(); // Убираем открывающую скобку
            } else if (IsOperator(token)) {
                // Обработка операторов: поддержка приоритетов операций
                while (stack.Count > 0 &&
                    GetPrecedence(token) <= GetPrecedence(stack.Peek())) {
                    output += stack.Pop() + " ";
                }
                stack.Push(token); // Добавляем текущий оператор в стек
            }
        }
    }

    // Если осталось накопленное число, добавляем его в результат
    if (number != "") {
        output += number + " ";
    }

    // Извлекаем все оставшиеся операторы из стека
    while (stack.Count > 0) {
        output += stack.Pop() + " ";
    }

    // Возвращаем окончательный результат в виде строки ОПЗ
    return output.Trim();
}

// Вычисление выражения, представленного в ОПЗ

```

```

static int EvaluateRPN(string rpn) {
    // Стек для операндов
    Stack<int> stack = new Stack<int>();
    // Разделяем строку ОПЗ на токены (операнды и операторы)
    string[] tokens = rpn.Split(' ');

    // Обрабатываем каждый токен
    foreach (string token in tokens) {
        // Если токен – число, помещаем его в стек
        if (int.TryParse(token, out int number)) {
            stack.Push(number);
        }
        // Если токен – оператор, выполняем операцию
        else if (IsOperator(token[0])) {
            // Извлекаем два операнда
            int operand2 = stack.Pop();
            int operand1 = stack.Pop();

            // Выполняем операцию и результат кладём обратно в стек
            int result = PerformOperation(operand1, operand2, token[0]);
            stack.Push(result);
        }
    }

    // В стеке остаётся окончательный результат
    return stack.Pop();
}

// Проверка, является ли символ оператором
static bool IsOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

// Определение приоритета операций (чем выше значение, тем выше приоритет)
static int GetPrecedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1; // Низкий приоритет
        case '*':
        case '/':
            return 2; // Высокий приоритет
        default:
            return 0; // Неизвестный оператор
    }
}

// Выполнение арифметической операции
static int PerformOperation(int operand1, int operand2, char op) {
    switch (op) {
        case '+':
            return operand1 + operand2; // Сложение
        case '-':
            return operand1 - operand2; // Вычитание
        case '*':
            return operand1 * operand2; // Умножение
        case '/':
            return operand1 / operand2; // Деление
        default:
            throw new ArgumentException("Неверный оператор");
    }
}
}
}
}

```

Демонстрация работы приложения представлена на рисунке 1.

```
dotnet run
/Users/lt/Documents/bmsu/7 семестр/dotnet/dz1/dz1/Program.cs(73,24): warning CS8600: Преобразование литерала, допускающего значение NULL или возможного значения NULL в тип, не допускающий значение NULL. [/Users/lt/Documents/bmsu/7 семестр/dotnet/dz1/dz1/dz1.csproj]
/Users/lt/Documents/bmsu/7 семестр/dotnet/dz1/dz1/Program.cs(74,13): warning CS8602: Разыменованное вероятной пустой ссылки. [/Users/lt/Documents/bmsu/7 семестр/dotnet/dz1/dz1/dz1.csproj]
Введите выражение для вычисления (или 'exit' для выхода):
>> 1+2
Результат: 3
>> 5*5
Результат: 25
>> 5/5
Результат: 1
>> 1-2
Результат: -1
>> (2+3)*5
Результат: 25
>> exit
```

Рисунок 2 – Результат работы программы

```
dotnet run -- test
Expression: 1+2
Result: 3, Expected: 3
Тест пройден успешно.

Expression: (234-11)*34
Result: 7582, Expected: 7582
Тест пройден успешно.

Expression: 6*6/6
Result: 6, Expected: 6
Тест пройден успешно.

Expression: (2+3)-4
Result: 1, Expected: 1
Тест пройден успешно.

Expression: 4-(2*3)
Result: -2, Expected: -2
Тест пройден успешно.

Expression: 10+((3+5)+2)
Result: 27, Expected: 27
Тест пройден успешно.

Expression: 100/(2+3)
Result: 20, Expected: 20
Тест пройден успешно.

Expression: 5+(6*7)-(8/4)
Result: 45, Expected: 45
Тест пройден успешно.

Expression: 15-(3*2)
Result: 9, Expected: 9
Тест пройден успешно.

Expression: ((3+5)*2)
Result: 16, Expected: 16
Тест пройден успешно.

Все тесты пройдены успешно.
```

Рисунок 3 – Результат тестирования программы

Вывод:

В ходе выполнения лабораторной работы было разработано приложение консольного калькулятора, работающего по методу Бауэра-Замельзона. Приложение работает корректно. Изучены основы разработки приложений на языке C#.