



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 *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ **ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

Студент		Магомадов Шахман Савраниевич
Группа		РК6-64Б
Тип задания		Производственная практика
Название предприятия		Государственная корпорация по атомной энергии «Росатом»

Студент

подпись, дата

Магомадов Ш.С. /
фамилия, и.о.

Наставник

подпись, дата

Демин Е.А. /
фамилия, и.о.

Оценка _____

Москва, 2021 г.

Оглавление

Введение.....	3
Индивидуальное задание.....	4
Процесс выполнения.....	5
Результаты работы программы.....	6
Код программы.....	7
Обзор используемых в разработке средств.....	15
Заключение.....	16
Список литературы.....	17

Введение

Я проходил практику в Росатоме – это государственная корпорация по атомной энергии - российский государственный холдинг, объединяющий более 400 предприятий атомной отрасли.

Перед прохождением практики мне дали выбрать несколько языков программирования, из них я выбрал Python, C#, SQL. Мне было дано задание по C#. Это было индивидуальное задание рассчитанное на все время практики.

В качестве цели практики были поставлены задачи, позволяющие получить новые, а также закрепить имеющиеся знания на примере выполнения задания. Были получены навыки реализации программ на языке C#, а также получен опыт в разработке Web Приложений MVC.

Индивидуальное задание

Задание – реализовать продвинутый калькулятор.

Текст задания:

- Создать на с#
- на главной странице должен быть поле ввода выражения и кнопка «результат»
- на странице с результатом должно отображаться либо результат вычисления либо ошибка (с базовым описанием)

Приложение должно вычислять выражение типа « $2+2$ », « $2+3-5*2$ »
арифметические действия: + (сложить) - (вычесть) * (умножить) / (разделить)
^ (возвести в степень)

Минимум:

реализовать + - сложных вычислений (с учетом скобок «()»)

Процесс выполнения

Выполнение данной задачи было начато с изучения MVC, в этот мне помогли руководства в интернете, в основном от Microsoft с подробным описанием принципов работы и пошаговыми инструкциями, с помощью которых я и сделал изначальный шаблон.

Далее начал реализовывать внешний вид и настраивать цвета, сделал отдельную страницу с результатом или ошибкой и присвоил ему свой http. Сделал поле для ввода и кнопку “результат”, в начале эта кнопка не обладала никаким функционалом.

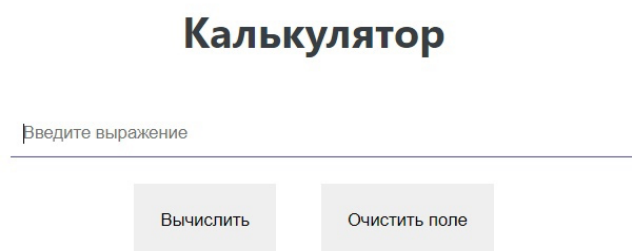
Следующим шагом было запрограммирование кнопки “результат” на вычисление операций, введенных в поле для ввода. На C# сделал программу, в котором были функции для каждого оператора, таких как +,-,/,* и т.д.

Затем реализовал функцию convert для конвертации строки в числа и знаки и размещении их по иерархии выполнения.

И напоследок написал функцию error для вывода ошибки при неправильном вводе и других возможных ошибок.

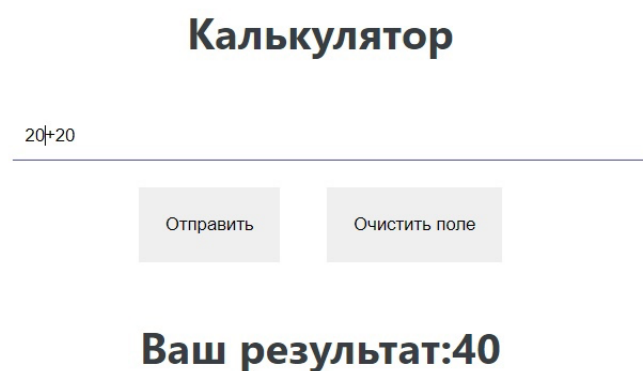
Результат работы программы.

Результаты работы программы приведены на Рис.1, Рис.2 и Рис.3.



The screenshot shows a web-based calculator interface. At the top, the title "Калькулятор" (Calculator) is displayed in a large, bold, black font. Below the title is a text input field with the placeholder text "Введите выражение" (Enter expression). Underneath the input field are two buttons: "Вычислить" (Calculate) on the left and "Очистить поле" (Clear field) on the right. Both buttons are light gray with black text.

Рис. 1. Начальный вид.



The screenshot shows the same calculator interface as in Figure 1, but with data entered. The title "Калькулятор" remains at the top. The input field now contains the expression "20+20". Below the input field, the "Отправить" (Send) button has been replaced by the "Вычислить" (Calculate) button. The "Очистить поле" (Clear field) button remains. Below the buttons, the result "Ваш результат:40" (Your result: 40) is displayed in a large, bold, black font.

Рис. 2. Ввод и результат отработки калькулятора.

Рис. 3. Вывод при введении некорректных данных.

Код программы.

Ниже приведен код программы с подробными комментариями, описывающими каждый этап.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace test
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        class RPN
        {
            //Метод Calculate принимает выражение в виде строки и возвращает
            //результат, в своей работе использует другие методы класса
            static public double Calculate(string input)
            {
                int i = 0;
                while (i < input.Length)
                {
                    if (input[i] >= '0' && input[i] <= '9' || input[i] == '*' ||
                        input[i] == '^' || input[i] == '/' || input[i] == '+' ||
                        input[i] == '-' ||
                        input[i] == '=' || input[i] == '(' || input[i] == ')')
                        i++;
                    else
                        return -1;
                }
                string output = GetExpression(input); //Преобразовываем выражение в
                //постфиксную запись
                double result = Counting(output); //Решаем полученное выражение
                return result; //Возвращаем результат
            }
        }
    }
}
```



```

//Метод, преобразующий входную строку с выражением в постфиксную запись
static private string GetExpression(string input)
{
    string output = string.Empty; //Строка для хранения выражения
    Stack<char> operStack = new Stack<char>(); //Стек для хранения
операторов

    for (int i = 0; i < input.Length; i++) //Для каждого символа в входной
строке
    {
        //Разделители пропускаем
        if (IsDelimiter(input[i]))
            continue; //Переходим к следующему символу

        //Если символ – цифра, то считываем все число
        if (Char.IsDigit(input[i])) //Если цифра
        {
            //Читаем до разделителя или оператора, что бы получить число
            while (!IsDelimiter(input[i]) && !IsOperator(input[i]))
            {
                output += input[i]; //Добавляем каждую цифру числа к нашей
строке

                i++; //Переходим к следующему символу

                if (i == input.Length) break; //Если символ – последний,
то выходим из цикла
            }

            output += " "; //Дописываем после числа пробел в строку с
выражением

            i--; //Возвращаемся на один символ назад, к символу перед
разделителем
        }

        //Если символ – оператор
        if (IsOperator(input[i])) //Если оператор
        {
            if (input[i] == '(') //Если символ – открывающая скобка
                operStack.Push(input[i]); //Записываем её в стек
            else if (input[i] == ')') //Если символ – закрывающая скобка
            {
                //Выписываем все операторы до открывающей скобки в строку
                char s = operStack.Pop();

                while (s != '(')
                {
                    output += s.ToString() + ' ';
                    s = operStack.Pop();
                }
            }
            else //Если любой другой оператор
            {
                if (operStack.Count > 0) //Если в стеке есть элементы

```

```

        if (GetPriority(input[i]) <=
GetPriority(operStack.Peek())) //И если приоритет нашего оператора меньше или равен
приоритету оператора на вершине стека
            output += operStack.Pop().ToString() + " "; //То
добавляем последний оператор из стека в строку с выражением

            operStack.Push(char.Parse(input[i].ToString())); //Если
стек пуст, или же приоритет оператора выше – добавляем операторов на вершину стека
        }
    }
}

//Когда прошли по всем символам, выкидываем из стека все оставшиеся
там операторы в строку
while (operStack.Count > 0)
    output += operStack.Pop() + " ";

return output; //Возвращаем выражение в постфиксной записи
}

//Метод, вычисляющий значение выражения, уже преобразованного в
постфиксную запись
static private double Counting(string input)
{
    double result = 0; //Результат
    Stack<double> temp = new Stack<double>(); //Dhtvtuysq стек для решения

    for (int i = 0; i < input.Length; i++) //Для каждого символа в строке
    {
        //Если символ – цифра, то читаем все число и записываем на вершину
стека

        if (Char.IsDigit(input[i]))
        {
            string a = string.Empty;

            while (!IsDelimiter(input[i]) && !IsOperator(input[i])) //Пока
не разделитель

            {
                a += input[i]; //Добавляем
                i++;
                if (i == input.Length) break;
            }
            temp.Push(double.Parse(a)); //Записываем в стек
            i--;
        }
        else if (IsOperator(input[i])) //Если символ – оператор
        {
            //Берем два последних значения из стека
            double a = temp.Pop();
            double b = temp.Pop();

```

```

        switch (input[i]) //И производим над ними действие, согласно
оператору
        {
            case '+': result = b + a; break;
            case '-': result = b - a; break;
            case '*': result = b * a; break;
            case '/': result = b / a; break;
            case '^': result =
double.Parse(Math.Pow(double.Parse(b.ToString()),
double.Parse(a.ToString())).ToString()); break;
        }
        temp.Push(result); //Результат вычисления записываем обратно в
стек
    }
    }
    return temp.Peek(); //Забираем результат всех вычислений из стека и
возвращаем его
}

public MainWindow()
{
    InitializeComponent();
}

static private bool IsDelimiter(char c)
{
    if ("=".IndexOf(c) != -1)
        return true;
    return false;
}

static private bool IsOperator(char c)
{
    if ("+-/*^()".IndexOf(c) != -1)
        return true;
    return false;
}

static private byte GetPriority(char s)
{
    switch (s)
    {
        case '(': return 0;
        case ')': return 1;
        case '+': return 2;
        case '-': return 3;
        case '*': return 4;
        case '/': return 4;
        case '^': return 5;
        default: return 6;
    }
}
}

```

```

static public double Calculate(string input)
{
    string output = GetExpression(input); //Преобразовываем выражение в
    постфиксную запись
    double result = Counting(output); //Решаем полученное выражение
    return result; //Возвращаем результат
}

static private string GetExpression(string input)
{
    string output = string.Empty; //Строка для хранения выражения
    Stack<char> operStack = new Stack<char>(); //Стек для хранения операторов

    for (int i = 0; i < input.Length; i++) //Для каждого символа в входной
    строке
    {
        //Разделители пропускаем
        if (IsDelimiter(input[i]))
            continue; //Переходим к следующему символу

        //Если символ – цифра, то считываем все число
        if (Char.IsDigit(input[i])) //Если цифра
        {
            //Читаем до разделителя или оператора, что бы получить число
            while (!IsDelimiter(input[i]) && !IsOperator(input[i]))
            {
                output += input[i]; //Добавляем каждую цифру числа к нашей
                строке

                i++; //Переходим к следующему символу

                if (i == input.Length) break; //Если символ – последний, то
                выходим из цикла
            }

            output += " "; //Дописываем после числа пробел в строку с
            выражением

            i--; //Возвращаемся на один символ назад, к символу перед
            разделителем
        }

        //Если символ – оператор
        if (IsOperator(input[i])) //Если оператор
        {
            if (input[i] == '(') //Если символ – открывающая скобка
                operStack.Push(input[i]); //Записываем её в стек
            else if (input[i] == ')') //Если символ – закрывающая скобка
            {
                //Выписываем все операторы до открывающей скобки в строку
                char s = operStack.Pop();

                while (s != '(')
                {
                    output += s.ToString() + ' ';
                }
            }
        }
    }
}

```

```

        s = operStack.Pop();
    }
}
else //Если любой другой оператор
{
    if (operStack.Count > 0) //Если в стеке есть элементы
        if (GetPriority(input[i]) <=
GetPriority(operStack.Peek())) //И если приоритет нашего оператора меньше или равен
приоритету оператора на вершине стека
            output += operStack.Pop().ToString() + " "; //То
добавляем последний оператор из стека в строку с выражением

            operStack.Push(char.Parse(input[i].ToString())); //Если стек
пуст, или же приоритет оператора выше – добавляем операторов на вершину стека
        }
    }
}

//Когда прошли по всем символам, выкидываем из стека все оставшиеся там
операторы в строку
while (operStack.Count > 0)
    output += operStack.Pop() + " ";

return output; //Возвращаем выражение в постфиксной записи
}

static private double Counting(string input)
{
    double result = 0; //Результат
    Stack<double> temp = new Stack<double>(); //стек для решения

    for (int i = 0; i < input.Length; i++) //Для каждого символа в строке
    {
        //Если символ – цифра, то читаем все число и записываем на вершину
стека
        if (Char.IsDigit(input[i]))
        {
            string a = string.Empty;

            while (!IsDelimiter(input[i]) && !IsOperator(input[i])) //Пока не
разделитель
            {
                a += input[i]; //Добавляем
                i++;
                if (i == input.Length) break;
            }
            temp.Push(double.Parse(a)); //Записываем в стек
            i--;
        }
        else if (IsOperator(input[i])) //Если символ – оператор
        {
            //Берем два последних значения из стека

```

```

        double a = temp.Pop();
        double b = temp.Pop();

        switch (input[i]) //И производим над ними действие, согласно
оператору
        {
            case '+': result = b + a; break;
            case '-': result = b - a; break;
            case '*': result = b * a; break;
            case '/': result = b / a; break;
            case '^': result =
double.Parse(Math.Pow(double.Parse(b.ToString()),
double.Parse(a.ToString())).ToString()); break;
        }
        temp.Push(result); //Результат вычисления записываем обратно в
стек
    }
    }
    return temp.Peek(); //Забираем результат всех вычислений из стека и
возвращаем его
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    if ((TextBox.Text = Convert.ToString(RPN.Calculate(TextBox.Text))) != "1")
    { }
    else
        TextBox.Text = "Error, wrong input";
}
}
}

```

Обзор используемых в разработке средств

- Visual Studio - Интегрированная среда разработки (IDE) — это многофункциональная программа, которая поддерживает многие аспекты разработки программного обеспечения. Интегрированная среда разработки Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Помимо стандартного редактора и отладчика, которые есть в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для улучшения процесса разработки.
- C# - современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. C# относится к широко известному семейству языков C
- MVC - Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Заключение

В результате прохождения практики был реализован продвинутый калькулятор в виде Web Приложение MVC на с#. В процессе выполнения задания я периодически созванивался со своим наставником и советовался с ним по тем или иным вопросам.

По окончании этой практики научился делать Web Приложения MVC, лучше разобрался в С#. Получил хороший опыт работы над большим проектом. Еще хорошим опытом практики является выполнение проекта схожего с реальными проектами и в схожих условиях под руководством опытного человека, работающего в крупной компании.

Список литературы

1. Марк Дж. Прайс. С# 7 и .NET Core. Кросс-платформенная разработка для профессионалов
2. Начало работы с ASP.NET MVC 5. <https://docs.microsoft.com/ru-ru/aspnet/mvc/overview/getting-started/introduction/getting-started>
3. Введение в ASP.NET MVC. <https://docs.microsoft.com/ru-ru/teamblog/channel9joinedmicrosoftlearn>
4. Редактор кода Visual Studio Code. Самый подробный гайд по настройке и установке плагинов для начинающих. <https://habr.com/ru/post/490754/>