Министерство образования и науки Российской Федерации
ФГАОУ ВО «КФУ им. В.И. Вернадского»
физико-технический институт (структурное подразделение)

Кафедра компьютерной инженерии и моделирования

Шенгелай Всеволод Михайлович

отчет по лабораторной работе №2
по дисциплине **«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Направление подготовки:
09.03.04 "Программная инженерия"

Оценка - 88

Симферополь, 2021

Лабораторная работа №2

Тема: Использование программных конструкций C#

Цель работы: Научиться преобразовывать различные типы данных в C#, познакомиться с типом данных Decimal, научиться грамотно использовать циклы для итерационных вычислений с контролем погрешности, обрабатывать события нажатия клавиш, научиться использовать классы String, StringBuilder, компоненты Grid или DataGridView в Windows Forms или WPF приложениях.

Описание ключевых понятий:

Класс – является описанием объекта, а объект представляет экземпляр этого класса. Класс представляет новый тип, который определяется пользователем. Класс определяется с помощью ключевого слова **class.**

Статический тип – статическими могут быть: поля, методы, свойства. Статические поля, методы, свойства относятся ко всему классу и для обращения к подобным членам класса необязательно создавать экземпляр класса.

Динамический тип – этот тип является статическим, но объект типа dynamic обходит проверку статического типа. В большинстве случаев он работает как тип object. Во время компиляции предполагается, что элемент, типизированный как dynamic, поддерживает любые операции. Это даёт нам некоторые преимущества динамически / слабо типизированных языков, при этом во всех других случаях сохраняя преимущества строго типизированного языка.

Встроенные типы – стандартный набор типов. Они используются для представления целых чисел, значений с плавающей запятой, логических выражений, текстовых символов, десятичных значений и других типов данных.

Таблица 1.1. Встроенные типы данных

Числовые	sbyte, byte, short, ushort, int, uint, long, ulong, float, double, decimal
Текстовые	char, string

Логический (булевый)	bool

Типы-значения – тип значений хранит непосредственно данные; Параметры и переменные метода, которые представляют типы значений, размещают свое значение в стеке. Стек представляет собой структуру данных, которая растет снизу вверх: каждый новый добавляемый элемент помещается поверх предыдущего. Время жизни переменных таких типов ограничено их контекстом. Физически стек - это некоторая область памяти в адресном пространстве.

Ссылочные типы – содержит ссылку на данные, которые именуются каким-либо объектом; Ссылочные типы хранятся в куче или хипе, которую можно представить как неупорядоченный набор разнородных объектов. Физически это остальная часть памяти, которая доступна процессу. При создании объекта ссылочного типа в стеке помещается ссылка на адрес в куче (хипе). Когда объект ссылочного типа перестает использоваться, в дело вступает автоматический сборщик мусора: он видит, что на объект в хипе нету больше ссылок, условно удаляет этот объект и очищает память – фактически помечает, что данный сегмент памяти может быть использован для хранения других данных.

Фундаментальные типы:

- **логический –** «bool», true/false;
- **символьный** – «char», один символ**;**
- **целый –** byte, sbyte, short, ushort, int, uint, long, ulong – содержит целое число;
- **с плавающей точкой –** float, double – экспоненциальная форма представления вещественных (действительных) чисел, в которой число хранится в виде мантиссы и порядка (показателя степени).;
- **void –** ключевое слово void указывает на то, что метод ничего не возвращает; Мы также можем использовать void как ссылочный тип для объявления того, что тип указателя неизвестен. Нельзя использовать void в качестве типа переменной;
- **указатели –** тип + «*», блок кода или метод, в котором используются указатели, помечается ключевым словом unsafe, сами указатели работают как

переменная-адрес в оперативной памяти другой переменной; Указатели позволяют получить доступ к определенной ячейке памяти и произвести определенные манипуляции со значением, хранящимся в этой ячейке. В языке C# указатели очень редко используются, однако в некоторых случаях можно прибегать к ним для оптимизации приложений. Код, применяющий указатели, еще называют небезопасным кодом. Однако это не значит, что он представляет какую-то опасность. Просто при работе с ним все действия по использованию памяти, в том числе по ее очистке, ложится целиком на нас, а не на среду CLR. И с точки зрения CLR такой код не безопасен, так как среда не может проверить данный код, поэтому повышается вероятность различного рода ошибок;

- **ссылки –** ref» + пробел + тип, означает прямое обращение к памяти, содержащейся в какой-то переменной;
- **массивы –** «Array» или тип + «[]», структура для хранения множества переменных одного типа вместе;
- **перечисления –** «enum», структура для хранения множества пользовательских имен без особых значений;
- **структуры –** «struct», пользовательская структура данных;
- **классы –** «class», пользовательский класс, обладает полями и методами;

Преобразования типов:

- **упаковать** – упаковка представляет собой процесс преобразования типа значения в тип object или в любой другой тип интерфейса, реализуемый этим типом значения. Когда тип значения упаковывается общеязыковой средой выполнения (CLR), он инкапсулирует значение внутри экземпляра System.Object и сохраняет его в управляемой куче. Операция распаковки извлекает тип значения из объекта. Упаковка является неявной; распаковка является явной. Понятия упаковки и распаковки лежат в основе единой системы типов C#, в которой значение любого типа можно рассматривать как объект.

- **распаковать –** распаковка ссылочного типа в значимый подразумевает, что это должно быть выполнено явно. При этом, необходимо, во-первых, сначала удостовериться, что тип упакованного объекта по ссылке соответствует исходному, а во-вторых, скопировать поля данных упакованного объекта в новую переменную данного типа. Как правило, проверку соответствия типов осуществляют с помощью механизма генерирования и обработки исключений, после чего копирование переносит внутренние данные (поля) объекта из «кучи» в стек выполняемого приложения, где хранятся его локальные переменные.
- **неявное преобразование** – преобразование без явного указания типа;
- **явное преобразование** – преобразование с явным указанием названия типа**;**
- **класс Convert –** встроенный класс, позволяющий переводить некоторые встроенные типы из одного в другой, весьма нагляден в коде;

Перед выполнением лабораторной работы изучена следующая литература:

1. Презентации лектора курса: «Типы в C#» и «Приведение типов в C#».
2. Видеолекция сотрудника Microsoft С.Байдачного «Основные конструкции C#»
3. Биллиг В.А. Основы объектного программирования на языке C# -М.: Интуит, 2009. (3,4 лекции)
4. Зиборов В. В.Visual C# 2012 на примерах. — СПб.: БХВ-Петербург, 2013. — 480 с.: ил. (главы 7)
5. Климов А. П.C#. Советы программистам. — СПб.: БХВ-Петербург, 2008. 544 с.: ил. (глава 2,6)
6. Сайт MSDN Microsoft.
7. Сайт Metanit.com

Выполнены 3 задания, описанных в методических указания к выполнению лабораторных работ.

Дополнительное задание: Написана программа преобразования из любого тапа в любой.

В этой программе определены универсальные методы, в которых будет производиться попытка явного/неявного приведения одного типа к другому. Также используется тип dynamic – он позволяет избежать проверки типов во время компиляции.

```
Ссылок: 9
static bool MyExplicit <inType> (dynamic from)
{
    try
    {
        inType to = (inType)from;
        return true;
    }
    catch
    {
        return false;
    }
}

Ссылок: 9
static bool MyImplict<inType>(dynamic from)
{
    try
    {
        inType to = from;
        return true;
    }
    catch
    {
        return false;
    }
}
```

Рисунок 1. Методы явного и неявного преобразования

Тренировочное задание: Написана программа преобразования типов, в которой предусмотрены все требования из условия.

```
// is и as
static void SafeFunction_as (object obj)
{
    Point point = obj as Point;

    if (point != null)
    {
        Console.WriteLine("Объект оказался Point");
        point.Print();
    }
    else
    {
        Console.WriteLine("Объект не Point! Его приведение не было успешным!");
    }
}

static void SafeFunction_is (object obj)
{
    if (obj is Point)
    {
        Point point = (Point)obj;
        Console.WriteLine("Объект оказался Point");
        point.Print();
    }
    else
    {
        Console.WriteLine("Объект не Point! Его приведение не было успешным!");
    }
}
```

Рисунок 2. Методы безопасного приведения ссылочных типов с помощью операторов as и is

```
// Подпрограмма сложения двух чисел, введённых через консоль
string str = "5";
int number1, number2;

Console.WriteLine("\nПодпрограмма сложения двух чисел");
Console.WriteLine("Введите число 1");
str = Console.ReadLine();
number1 = Convert.ToInt32(str);

Console.WriteLine("Введите число 2");
str = Console.ReadLine();
number2 = Convert.ToInt32(str);

Console.WriteLine("Сумма чисел = " + (number1 + number2));
Console.WriteLine("Работа подпрограммы завершена.");
```

Рисунок 3. Преобразование string к int с помощью класса Convert

Задание 1: Создано WPF приложение с функционалом, описанным в методических указаниях.

```
// Алгоритм Ньютона
double Calculate_double(double xi, double xi_prev, int iterations)
{
    for (int i = 0; i < iterations; ++i)
    {
        xi = 0.5 * (number_d / xi_prev + xi_prev);
        xi_prev = xi;
    }
    return xi;
}

decimal approx_decimal;
decimal number_decimal;
result_1 = decimal.TryParse(approx_s, out approx_decimal);
result_2 = decimal.TryParse(number_s, out number_decimal);

decimal Calculate_decimal(decimal xi, decimal xi_prev, int iterations)
{
    for (int i = 0; i < iterations; ++i)
    {
        xi = (decimal)0.5 * (number_decimal / xi_prev + xi_prev);
        xi_prev = xi;
    }
    return xi;
}
```

Рисунок 4. Алгоритм Ньютона для итеративного нахождения квадратного корня

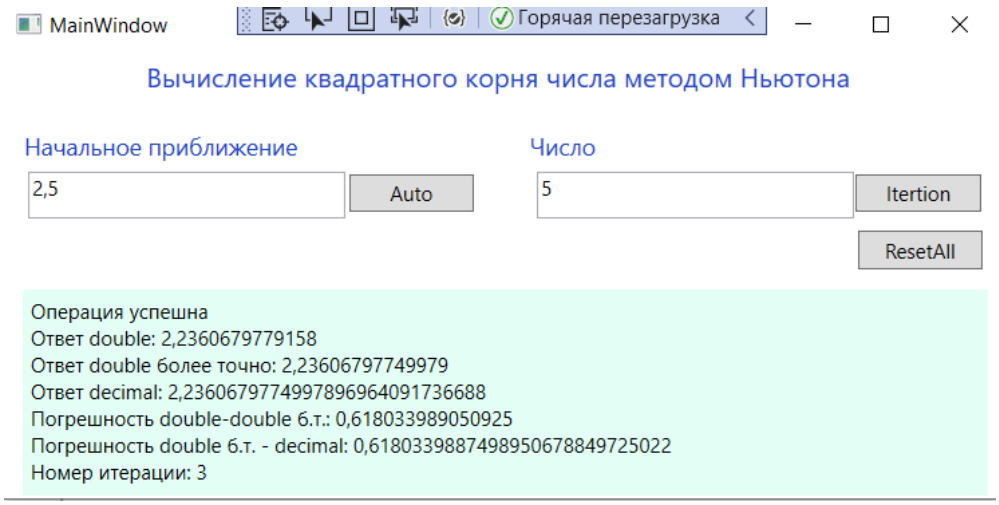

Рисунок 5. Графический интерфейс приложения для задания 1.

Задание 2: Создано WPF приложение с функционалом, описанным в методических указаниях.

В этом упражнении мы создали приложение, которое позволяет пользователю вводить целочисленное значение, генерировать строку, содержащую двоичное представление этого значения, а затем отображает результат.

Задача программы реализуется следующим циклом for:

a. Вычисляем остаток от деления i на 2, а затем сохраняем это значение в переменной остатка.

б. Делим i на 2.

c. Приставляем значение остатка к началу строки, создаваемой двоичной переменной. Завершите цикл for, когда i меньше или равно нулю.

```
int number_i;

bool result = int.TryParse(number_s, out number_i);

if (result)
{
    if (number_i >= 0)
    {
        int i = number_i;
        int remainder = 0;

        StringBuilder binary = new StringBuilder("");

        do
        {
            remainder = i % 2;
            i = i / 2;
            binary.Insert(0, remainder);
        }
        while (i > 0);

        string output = binary.ToString();
        binaryLabel.Content = output;

        return;
    }
    else
    {
        MessageBox.Show("Введите целое неотрицательное число", "Сообщение", MessageBoxButton.OK,
            MessageBoxImage.Information);
        return;
    }
```

Рисунок 4. Цикл деления введённого числа на 2

Задание 3: Создано WPF приложение с функционалом, описанным в методических указаниях.

Это приложение WPF предоставляет пользовательский интерфейс, который позволяет пользователю предоставлять данные для двух матриц и сохранять эти данные в прямоугольных массивах. Приложение вычислит произведение этих двух массивов и отобразит их.

Для отображения матриц используется элемент управления Grid, который перед каждым заполнением матриц числами очищается, и затем в него помещаются textbox`ы со сгенерированными числами, которые уже может изменить пользователь.

```
ссылка: 1
static double[,] MatrixGeneration(double[,] M, double[,] N)
{
    double[,] Rez = new double[N.GetLength(0), M.GetLength(1)];

    for (int i = 0; i < N.GetLength(0); i++)
    {
        for (int j = 0; j < M.GetLength(1); j++)
        {
            Rez[i, j] = 0;

            for (int k = 0; k < M.GetLength(0); k++)
            {
                Rez[i, j] += M[k, j] * N[i, k];
            }
        }
    }

    return Rez;
}
```

Рисунок 5. Метод умножения двух матриц

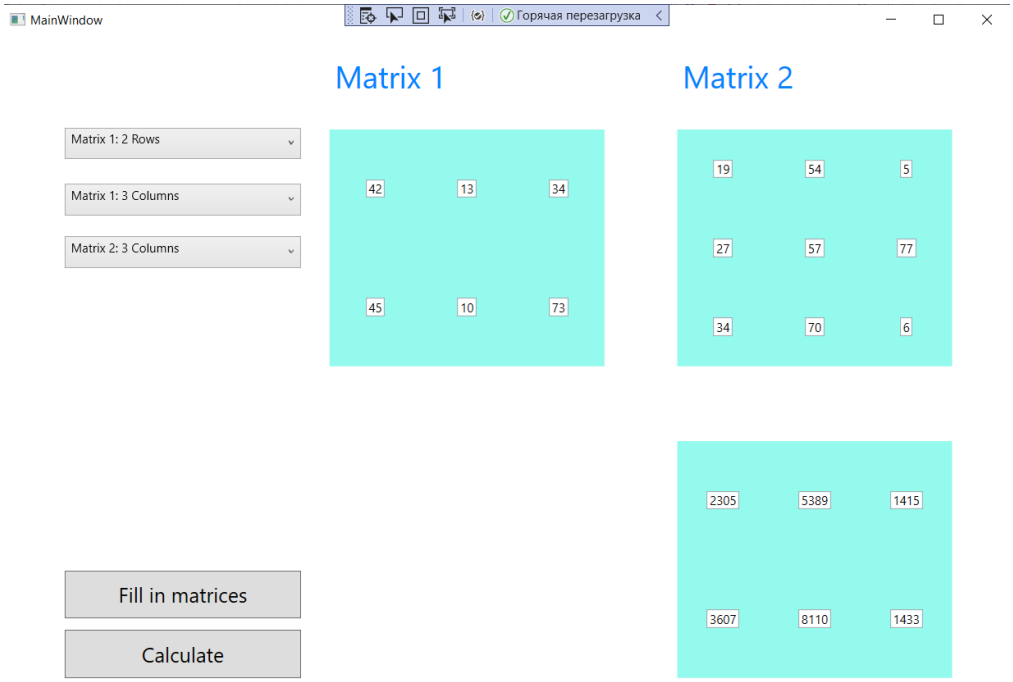

Рисунок 6. Графический интерфейс приложения для задания 3.

Представлены 4 проекта, реализованных в Visual Studio Common Eddition 2019. Проекты представлены преподавателю в электронной форме, продемонстрирована их работоспособность, разъяснены детали программного кода.