МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Основы проектирования машин»

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ПРОЕКТИРОВАНИИ

Методические рекомендации к лабораторным работам для студентов направления подготовки 15.03.03 «Прикладная механика» очной формы обучения

Могилев 2024

УДК 621.01
ББК 36.4
И87

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Основы проектирования машин» «29» мая 2024 г., протокол № 11

Составитель канд. техн. наук, доц. А. П. Прудников

Рецензент ст. преподаватель Ю. С. Романович

Изложены цели, содержание и порядок выполнения лабораторных работ.

Учебное издание

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ПРОЕКТИРОВАНИИ

Ответственный за выпуск	А. П. Прудников
Корректор	А. Т. Червинская
Компьютерная верстка	Е. В. Ковалевская

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 26 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Методические рекомендации составлены в соответствии с рабочей программой по курсу «Информационные технологии в проектировании» для студентов направления подготовки 15.03.03 «Прикладная механика» очной формы обучения.

Информационные технологии играют важную роль при создании новых продуктов и услуг, улучшении коммуникации и доступа к информации. Обладание навыками программирования позволяет решать сложные задачи и проводить научные исследования, автоматизировать и оптимизировать различные процессы, упрощая работу и повышая ее эффективность. С ростом компьютеризированности всех областей человеческой деятельности востребованность навыков создания собственных программ увеличивается и вместе с тем будет повышаться и актуальность проблемы обучения и самообучения навыкам программирования.

Целью изучения дисциплины «Информационные технологии в проектировании» является формирование у студентов базовых знаний языка C#, умений и навыков программирования.

Задачи изучения дисциплины «Информационные технологии в проектировании»:

– обучиться программировать на языке высокого уровня C#;

– научиться использовать современные методы и средства разработки алгоритмов и программ;

– освоить приемы визуального программирования, изучить способы отладки, испытания программ.

Объектом изучения дисциплины «Информационные технологии в проектировании» является язык программирования C#, предметом изучения – структура, принципы и синтаксис языка.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в то же время гибких, масштабируемых и расширяемых приложений [1, 2].

В методических рекомендациях в краткой форме изложены цель, содержание и порядок выполнения лабораторных работ.

Лабораторная работа выполняется в среде Visual Studio.

Отчет по лабораторной работе оформляется в текстовом редакторе Word и должен содержать:

– цель лабораторной работы;

– описание задания;

– код приложения с комментариями, описывающими код.

Методические рекомендации предназначены для самостоятельной подготовки студентов к лабораторным занятиям по дисциплине «Информационные технологии в проектировании».

1 Лабораторная работа № 1. Консольное приложение

Цель работы: научиться работать с переменными и константами простых типов в C# и создавать консольные приложения.

Теоретические основы

Проект содержит все исходные материалы для приложения, такие как файлы исходного кода, ресурсов, значки, ссылки на внешние файлы, на которые опирается программа, и данные конфигурации, такие как параметры компилятора. В файле проекта находится информация о модулях, составляющих данный проект, входящих в него ресурсах, а также параметрах построения программы. Файл проекта автоматически создается и изменяется средой Visual Studio.

Кроме понятия проект часто используется более глобальное понятие – решение (solution). Решение содержит один или несколько проектов, один из которых может быть указан как стартовый проект.

Весь код C# должен содержаться внутри класса. Объявление класса состоит из ключевого слова class, за которым следует имя класса и пара фигурных скобок. Весь код, ассоциированный с этим классом, размещается между этими скобками.

В C#, как и в других C-подобных языках, большинство операторов завершаются точкой с запятой (;) и могут продолжаться в нескольких строках без необходимости указания знака переноса. Операторы могут быть объединены в блоки с помощью фигурных скобок ({ }). Однострочные комментарии начинаются с двойного слеша (//), а многострочные – со слеша со звездочкой (/*) и заканчиваются противоположной комбинацией (*/).

Следует также помнить о том, что язык C# чувствителен к регистру символов. Это значит, что переменные с именами myVar и MyVar являются разными.

В разделе подключения пространств имен (каждая строка которого располагается в начале файла и начинается ключевым словом using) описываются используемые пространства имен. Каждое пространство имен включает в себя классы, выполняющие определенную работу, например, классы для работы с сетью располагаются в пространстве System.Net, а для работы с файлами – в System.IO. Большая часть пространств, которые используются в обычных проектах, уже подключена при создании нового проекта, но при необходимости можно дописать дополнительные пространства имен.

Для того чтобы не происходили конфликты имен классов и переменных, классы нашего проекта также помещаются в отдельное пространство имен. Определяется оно ключевым словом namespace, после которого следует имя пространства (обычно оно совпадает с именем проекта).

Когда компилируется консольное или Windows-приложение C#, по умолчанию компилятор ищет в точности один метод Main () с описанной выше сигнатурой в любом классе и делает его точкой входа программы. Если существует более одного метода Main ( ), компилятор возвращает сообщение об ошибке.

Функцию Main называют «точкой входа», т. е. началом выполнения программы.

Структура консольного приложения на языке C# приведена на рисунке 1.1.

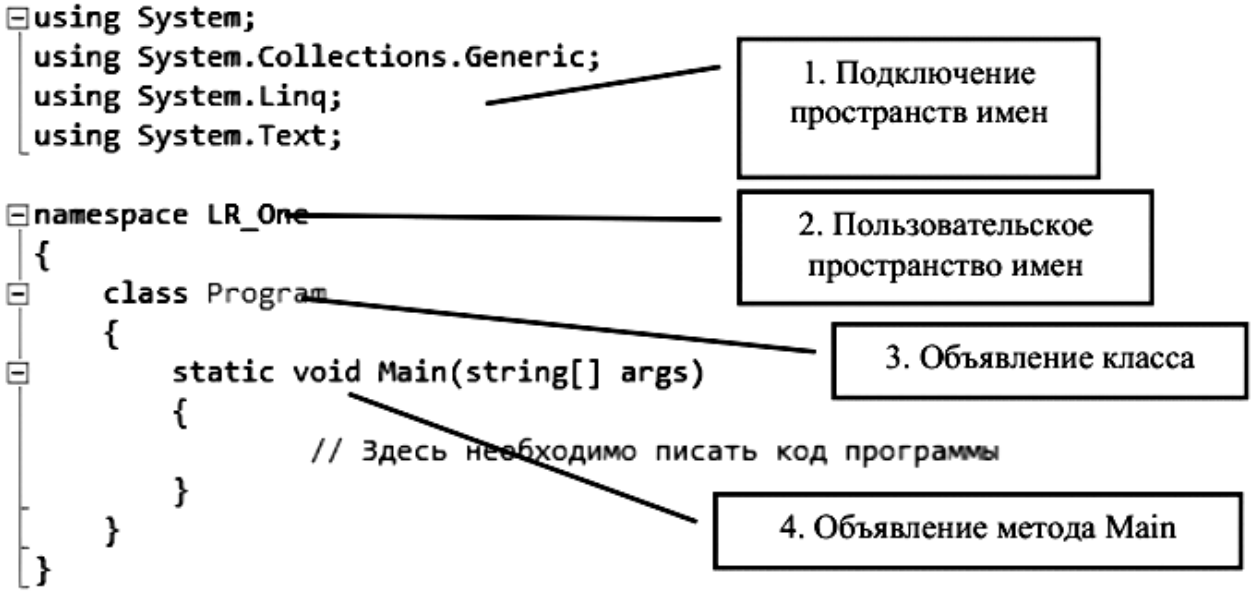

Рисунок 1.1 – Консольное приложение на языке C#

Порядок выполнения работы

1 Для создания консольного приложения выполните следующие действия: File → New → Project...

2 В открывшемся диалоговом окне выберите необходимые настройки для создаваемого проекта: язык Visual C#; фреймворк: .NET Framework 4; шаблон: Console Application.

3 Создайте приложение, которое будет выводить на экран следующую информацию (каждый студент должен использовать информацию о себе):

– название и номер лабораторной работы;
– ФИО студента;
– группа студента и шифр специальности;
– дата рождения студента;
– населенный пункт постоянного места жительства студента;
– краткое описание увлечений, хобби, интересов.

Вопросы для самоконтроля

1 Какая функция имеет особенное значение при выполнении программы на языках C, C++, C#?

2 Что такое «точка входа» в программе?

3 Для чего используется оператор using?

2 Лабораторная работа № 2. Типы данных

Цель работы: научиться работать с переменными и константами простых типов в C#.

Теоретические основы

Язык C# поддерживает предопределенные типы данных, приведенных в таблице 2.1.

Синтаксис объявления переменных в C# выглядит следующим образом: *тип данных идентификатор переменной.*

Компилятор не позволит использовать эту переменную до тех пор, пока она не будет инициализирована (т. е. пока ей не будет присвоено значение).

Константа – это переменная, значение которой не меняется за время выполнения программы. Для объявления константы необходимо воспользоваться ключевым словом const.

Таблица 2.1 – Типы данных

Имя типа	Диапазон (минимум : максимум)
sbyte	–128 : 127
short	–32 768 : 32 767
int	–2 147 483 648 : 2 147 483 647
long	$-2^{63} : 2^{63}$
byte	0 : 255
ushort	0 : 65 535
uint	$0 : 2^{32}$
ulong	$0 : 2^{64}$
float	$\pm 1{,}5 \times 10^{-45} : \pm 3{,}4 \times 10^{38}$
double	$\pm 5{,}0 \times 10^{-324} : \pm 1{,}7 \times 10^{308}$
decimal	$\pm 1{,}0 \times 10^{-28} : \pm 7{,}9 \times 10^{28}$
bool	true или false
char	16-битный (Unicode) символ

Порядок выполнения работы

Объявите требуемые переменные, присвойте им начальные значения (определите самостоятельно значения какого типа могут принимать переменные), выведите на экран с использованием форматной строки значения переменных и результат вычисления выражения

$$U = x \cdot z + z \cdot y - \frac{y + z}{x}.$$

Вопросы для самоконтроля

1 Что такое переменная? Как объявляется переменная?
2 Как объявляется константа? Чем константа отличается от переменной?
3 Какие типы значений применяются C#?
4 Как производится инициализация переменных? Как производится инициализация констант?

3 Лабораторная работа № 3. Арифметические операции

Цель работы: научиться составлять простейшие программы линейного алгоритма в среде Visual Studio, используя арифметические операции.

Теоретические основы

Следующие операторы выполняют арифметические операции с операндами числовых типов:
1) унарные: ++ (приращение), -- (уменьшение), + (плюс) и – (минус);
2) бинарные: * (умножение), / (деление), % (остаток от деления), + (сложение) и – (вычитание).

Эти операторы поддерживаются всеми целочисленными типами и типами с плавающей запятой.

Для задания приоритетов операций могут использоваться круглые скобки (). Также могут использоваться стандартные математические функции, представленные методами класса Math:
– Math.Sin(*a*) – синус (аргумент задается в радианах);
– Math.Cos(*a*) – косинус (аргумент задается в радианах);
– Math.Atan(*a*) – арктангенс (аргумент задается в радианах);
– Math.Log(*a*) – натуральный логарифм;
– Math.Exp(*a*) – экспонента;
– Math.Pow(*a*, *a*) – возводит переменную x в степень y;
– Math.Sqrt(*a*) – квадратный корень;
– Math.Abs(*a*) – модуль числа;
– Math.Truncate(*a*) – целая часть числа;
– Math.Round(*a*) – округление числа.

Порядок выполнения работы

Необходимо вычислить значение функции при $x = 2{,}5$:

$$y = \frac{\cos(\pi x)}{1 + x^2}.$$

Код программы приведен на рисунке 3.1.

Составьте программу для выполнения расчетов функции при x_1 = 2,5; x_2 = 1,3, неизвестные значения задаются путем ввода значений в консоли.

$$y = (x_1 + 3,2)^{1,2} + \sin(4x_2) - \sqrt{12x_1}.$$

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   namespace ConsoleApplication1
6   {
7       class Example1
8       {
9           static void Main()
10          {
11              double p = 3.14159;
12              double x = 2.5;
13              double y = Math.Cos(p * x) / (1 + x * x);
14              Console.WriteLine();
15              Console.WriteLine(" x = {0} \t y = {1} ",
16              x, y);
17          }
18      }
19  }
```

Рисунок 3.1 – Арифметические операции в C#

Вопросы для самоконтроля

1 Назовите стандартные математические функции класса Math.
2 Каков порядок выполнения математических операций?
3 Какие операторы являются унарными?

4 Лабораторная работа № 4. Операторы ветвлений и циклов

Цель работы: изучить операторы, позволяющие организовывать непоследовательное и циклическое выполнение программного кода.

Теоретические основы

В языке C# используются условия, циклы, ветвления.

Условный оператор в C# позволяет организовать условие типа «если …, то …, иначе …».

Условный оператор содержит только одно обязательное зарезервированное слово – if. Все остальные конструкции не являются обязательными.

Пример использования условного оператора:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2)
{
   Console.WriteLine($"Число {num1} больше числа {num2}");
}
else if (num1 < num2)
{
   Console.WriteLine($"Число {num1} меньше числа {num2}");
}
else
{
   Console.WriteLine("Число num1 равно числу num2");
}
```

В представленном примере сначала проверяется является ли значение переменной num1 больше num2. Если «Да», то выводится соответствующее сообщение. На этом выполнение всего условного оператора завершено и программа переходит к выполнению операторов, следующих за ним.

Если первое условие не выполнилось, проверяется второе – является ли num1 меньше num2 (конструкция else if). Если это утверждение истинно, то выводится соответствующее сообщение.

Если не выполнилось ни одно условие с оператором if, то выполняется оператор, указанный после зарезервированного слова else.

Во всей этой конструкции только оператор if является обязательным. Конструкций else if может быть любое количество, а if и else – только по одному.

Следует понимать, что условное выражение, стоящее в конструкции if, должно возвращать булево значение.

Существуют задачи, когда необходимо сделать выбор из нескольких вариантов значения некоторого выражения. Такой выбор можно описать и оператором if, но тогда выражение необходимо повторять в каждом условии. Более наглядным такой выбор можно сделать с помощью оператора switch.

Пример использования оператора switch … case:

```
switch (выражение)
{
   case значение1:
      код,выполняемый если выражение имеет значение1
      break;
   case значение2:
      код,выполняемый если выражение имеет значение2
      break;
   //.............
   case значениеN:
```

```
        код, выполняемый если выражение имеет значениеN
        break;
    default:
        код, выполняемый если выражение не имеет ни одно из выше указанных значений
        break;
}
```

Вначале вычисляется значение switch выражения. Затем оно поочередно в порядке следования case сравнивается на совпадение с константами. Как только значения совпали, выполняется соответствующая последовательность операторов case-ветви. Поскольку последний оператор этой последовательности – оператор перехода (чаще всего это оператор break), обычно он завершает выполнение оператора switch.

В некоторых случаях case-ветвь может быть пустой последовательностью операторов. Тогда при совпадении константы этой ветви со значением switch-выражения будет выполняться первая непустая последовательность очередной case-ветви. Если значение switch-выражения не совпадает ни с одной константой, то выполняется последовательность операторов ветви default, если же таковой ветви нет, то оператор switch эквивалентен пустому оператору.

Циклы позволяют выполнять определенную последовательность операторов до тех пор, пока выполняется некоторое условие. Каждый «круг» выполнения этого блока называется итерацией.

Синтаксис оператора: for (*инициализатор*; *условие*; *итератор*) *оператор.*

Инициализатор – выражение, выполняемое перед первой итерацией цикла.

Условие – выражение булевого типа, которое проверяется перед каждой итерацией. Если оно истинно, то итерация выполняется, иначе – цикл завершается. Итератор – выражение, вычисляемое после каждой итерации.

Пример применения оператора цикла for:

```
for (int i = 1; i < 4; i++)
{
   Console.WriteLine(i);
}
```

Работа оператора цикла for начинается с инициализации управляющей переменной цикла (счетчика). Затем проверяется условие, т. е. значение счетчика сравнивается с конечным значением. Если условие ложно, то управление передается оператору, следующему за оператором for. Если условие истинно, то выполняется тело цикла. Затем изменяется значение счетчика на величину шага (модификация) и снова проверяется условие.

С помощью оператора цикла while реализуется циклический вычислительный процесс с предусловием – условие выхода из цикла проверяется до выполнения тела цикла (цикл может не выполниться ни разу).

Синтаксис оператора цикла while: while (*выражение*) *оператор*.

Оператор – это один оператор или блок операторов, выражение – любое логическое выражение, принимающее значение true или false.

В этом цикле оператор выполняется до тех пор, пока значение выражения равно true (истина). Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла.

Пример применения оператора цикла while:

```
int n = 0;
while (n < 5)
{
   Console.Write(n);
   n++;
}
// Output:
// 01234
```

Оператор break часто применяется внутри циклов for, while и do … while. Данный оператор прерывает выполнение цикла и передает управление оператору, следующему за циклом.

Оператор continue также применяется внутри цикла – он останавливает выполнение текущей итерации и немедленно переходит к выполнению следующей итерации.

Цикл do … while полностью повторяет функциональные возможности while, но предполагает проверку условия окончания цикла после выполнения тела цикла. То есть блок операторов тела цикла выполнится хотя бы один раз.

Синтаксис оператора: do {*оператор*} while (*условие*).

Пример применения оператора цикла do…while:

```
int n = 0;
do
{
   Console.Write(n);
   n++;
} while (n < 5);
// Output:
// 01234
```

В этом цикле оператор выполняется до тех пор, пока значение выражения равно true (истина). Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла. Но в отличие от оператора while, этот цикл выполнится хотя бы один раз.

Порядок выполнения работы

1 Разработайте консольное приложение, вычисляющее значение функции:

$$S = \begin{cases} \sqrt[3]{ab^2}, & \text{если } a \le 3; \\ a - b, & \text{если } 3 < a \le 8; \\ a + b, & \text{если } a > 8. \end{cases}$$

Код программы приведен на рисунке 4.1.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.SetWindowSize(40, 20);
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            Console.Write("Введите a: ");
            double a = Convert.ToDouble(Console.ReadLine());
            Console.Write("Введите b: ");
            double b = Convert.ToDouble(Console.ReadLine());
            double S;
            // Вычисление значения функции y
            if (a <= 3)
            {
                S = Math.Pow(a * Math.Pow(b, 2), 1.0 / 3);
            }
            else if (a <= 8)
            {
                S = a - b;
            }
            else
            {
                S = a + b;
            }
            Console.WriteLine("S={0}", S);
            Console.ReadKey();
        }
    }
}
```

Рисунок 4.1 – Код программы

2 Разработайте консольное приложение, вычисляющее значение функции:

$$y=\begin{cases}2+x, & \text{если } x=1;\\ |x-3|, & \text{если } x=2;\\ 10, & \text{если } x=3.\end{cases}$$

Код программы приведен на рисунке 4.2.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.SetWindowSize(40, 20);
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            Console.Write("Введите x: ");
            string inp_x = Console.ReadLine();
            int x = Convert.ToInt32(inp_x);
            double y = 0;
            switch (x)
            {
                case 1:
                    y = 2 + x;
                    break;
                case 2:
                    y = Math.Abs(x - 3);
                    break;
                case 3:
                    y = 10;
                    break;
            }
            Console.WriteLine("y={0}", y);
            Console.ReadKey();
        }
    }
}
```

Рисунок 4.2 – Код программы

3 Разработайте консольное приложение, вычисляющее элементы последовательности с заданной точностью 0,1:

$$a_n = \frac{1}{n+2}.$$

Код программы приведен на рисунке 4.3.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.SetWindowSize(40, 20);
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            Console.Write("Введите e: ");
            double e = Convert.ToDouble(Console.ReadLine());
            double n = 1;
            double a = 1.0 / 3;
            Console.WriteLine("n={0}\ta={1}", n, a);
            while (a > e)
            {
                n = n + 1;
                a = 1 / (n + 2);
                Console.WriteLine("n={0}\ta={1}", n, a);
            }
            Console.ReadKey();
        }
    }
}
```

Рисунок 4.3 – Код программы

Вопросы для самоконтроля

1 Опишите синтаксис условного оператора. Какие части данного оператора являются обязательными?

2 Опишите синтаксис условного оператора. Какие части данного оператора являются обязательными?

3 Опишите синтаксис оператора цикла for.

4 Можно ли с помощью for реализовать бесконечный цикл? Поясните ответ на примерах.

5 Опишите синтаксис оператора while.

6 Какой цикл лучше while или for?

7 Поясните назначение операторов break и continue.

8 Опишите синтаксис оператора do…while.

9 Какой цикл лучше do…while или while?

5 Лабораторная работа № 5. Работа с массивами

Цель работы: научиться реализовывать алгоритмы обработки массивов на языке программирования C#.

Теоретические основы

Массив – набор элементов одного и того же типа, объединенных общим именем. Массивы в C# можно использовать по аналогии с тем, как они используются в других языках программирования. Однако C#-массивы имеют существенные отличия: они относятся к ссылочным типам данных, более того – реализованы как объекты. Фактически имя массива является ссылкой на область динамической памяти, в которой последовательно размещается набор элементов определенного типа. Выделение памяти под элементы происходит на этапе инициализации массива. А за освобождением памяти следит система сборки мусора – неиспользуемые массивы автоматически утилизируются данной системой.

Одномерный массив – это фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент имеет свой номер. Нумерация элементов массива в C# начинается с нуля.

Одномерный массив в C# реализуется как объект, поэтому его создание представляет собой двухступенчатый процесс. Сначала объявляется ссылочная переменная на массив, затем выделяется память под требуемое количество элементов базового типа, и ссылочной переменной присваивается адрес нулевого элемента в массиве. Базовый тип определяет тип данных каждого элемента массива. Количество элементов, которые будут храниться в массиве, определяет размер массива.

В общем случае процесс объявления переменной типа массив, и выделение необходимого объема памяти может быть разделено. Кроме того, на этапе объявления массива можно произвести его инициализацию. Поэтому для объявления одномерного массива может использоваться одна из следующих форм записи: базовый_тип [] имя__массива.

Например: int [] a={0, 1, 2, 3};

Так как массив представляет собой набор элементов, объединенных общим именем, то обработка массива обычно производится в цикле.

Одним из способов задания массива является определение элементов через случайные числа. Для работы со случайными числами используются в основном два метода класса Random:Random и Next. Метод Random подготавливает работу со случайными числами, обеспечивая надежный способ создания непредсказуемой последовательности чисел.

Многомерные массивы имеют более одного измерения. Чаще всего используются двумерные массивы, которые представляют собой таблицы. Каждый элемент массива имеет два индекса, первый определяет номер строки, второй – номер столбца, на пересечении которых находится элемент. Нумерация строк и столбцов начинается с нуля. Объявить двумерный массив можно следующим способом: тип [,] имя__массива = new тип [размер1, размер2].

Порядок выполнения работы

1 Разработайте консольное приложение для следующей задачи: даны массивы X и Y, состоящие из 10 вещественных чисел каждый. Образовать массив Z, каждый элемент которого вычисляется по правилу

$$Z_i = \sqrt[3]{X_i^2 + Y_i^2}.$$

Код программы приведен на рисунке 5.1.

2 Дан массив, состоящий из 20 вещественных чисел. Найдите значение и номер максимального элемента массива. Код программы приведен на рисунке 5.2.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.SetWindowSize(60, 50);
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            // Объявление массивов X, Y, Z
            double[] X = new double[10];
            double[] Y = new double[10];
            double[] Z = new double[10];
            // Ввод элементов массива X(10)
            Console.WriteLine("\nВведите элементы массива X");
            for (int i = 0; i <= 9; i++)
            {
                X[i] = double.Parse(Console.ReadLine());
            }
            // Ввод элементов массива Y(10)
            Console.WriteLine("\nВведите элементы массива Y");
            for (int i = 0; i <= 9; i++)
            {
                Y[i] = double.Parse(Console.ReadLine());
            }
            // Вычисление и вывод на экран элементов массива Z
            for (int i = 0; i <= 9; i++)
            {
                Z[i] = X[i] + Y[i];
                Console.WriteLine();
                Console.WriteLine("Z[{0}]={1}", i, Z[i]);
            }
        }
    }
}
```

Рисунок 5.1 – Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.SetWindowSize(50, 40);
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Black;
            // Объявление массива K
            double[] K = new double[20];
            // Ввод элементов массива K
            Console.WriteLine("\nВведите элементы массива K");
            for (int i = 0; i <= 19; i++)
            {
                K[i] = double.Parse(Console.ReadLine());
            }
            double max = K[0];
            int N = 0;
            // Поиск максимального элемента массива K
            for (int i = 1; i <= 19; i++)
            {
                if (K[i] > max)
                {
                    max = K[i];
                    N = i;
                }
            }
            Console.WriteLine();
            Console.WriteLine("max={0}\tN={1}", max, N);
        }
    }
}
```

Рисунок 5.2 – Код программы

Вопросы для самоконтроля

1 Дайте определение массива.
2 Что называется размерностью массива?
3 Что такое индекс массива?
4 Напишите способы объявления и инициализации массивов.

6 Лабораторная работа № 6. Методы

Цель работы: изучить структуру и принципы работы с методами (функциями).

Теоретические основы

Метод – это элемент класса, который содержит программный код.

Метод имеет следующую структуру:

[атрибуты] [спецификаторы] тип имя ([параметры])

{

Тело метода;

}

Атрибуты – это особые указания компилятору на свойства метода.

Атрибуты используются редко.

Спецификаторы – это ключевые слова, предназначенные для разных целей, например:

– определяющие доступность метода для других классов:

– private – метод будет доступен только внутри этого класса;

– protected – метод будет доступен также дочерним классам;

– public – метод будет доступен любому другому классу, который может получить доступ к данному классу;

– указывающие доступность метода без создания класса;

– задающие тип.

Тип определяет результат, который возвращает метод: это может быть любой тип, доступный в C#, а также ключевое слово void, если результат не требуется.

Имя метода – это идентификатор, который будет использоваться для вызова метода. К идентификатору применяются те же требования, что и к именам переменных: он может состоять из букв, цифр и знака подчеркивания, но не может начинаться с цифры.

Параметры – это список переменных, которые можно передавать в метод при вызове. Каждый параметр состоит из типа и названия переменной. Параметры разделяются запятой.

Тело метода – это обычный программный код, за исключением того, что он не может содержать определения других методов, классов, пространств имен и т. д. Если метод должен возвращать какой-то результат, то обязательно в конце должно присутствовать ключевое слово return с возвращаемым значением. Если возвращение результатов не нужно, то использование ключевого слова return не обязательно, хотя и допускается.

Пример использования метода класса:

// Объявление класса public class MyFirstClass

{

```
// Данные члены класса
// Доступные на уровне экземпляра
public int a; public float b;
public string fio;
// Доступные только на уровне класса
private bool IsOK;
private double precision;
// Метод для инициализации некоторых полей класса
public void InitClassMembers (int pA, float pB, string pFio)
{
a = pA;
b = pB;
fio = pFio;
}
// Метод, возвращающий значение, вычисленное на основе полей класса
public int GetAbsA()
{
return Math.Abs(a);
}
}
```

В данном примере метод InitClassMembers не возвращает никаких данных, но требует передачи ему фактических параметров. В свою очередь, метод GetAbsA возвращает значение типа int и не предполагает никаких параметров.

В общем случае параметры могут передаваться методу либо по значению, либо по ссылке. Когда переменная передается по ссылке, вызываемый метод получает саму переменную, поэтому любые изменения, которым она подвергнется внутри метода, останутся в силе после его завершения. Но если переменная передается по значению, вызываемый метод получает копию этой переменной, а это значит, что все изменения в ней по завершении метода будут утеряны. Для сложных типов данных передача по ссылке более эффективна вследствие большого объема данных, который приходится копировать при передаче по значению.

Если не указано обратное, то в C# все параметры передаются по значению. Тем не менее, можно принудительно передавать значения по ссылке, для чего используется ключевое слово ref. Если параметр передается в метод, и входной аргумент этого метода снабжен префиксом ref, то любые изменения этой переменной, которые сделает метод, отразятся на исходном объекте.

В C-подобных языках функции часто возвращают более одного значения. Это обеспечивается применением выходных параметров, за счет присваивания значений переменным, переданным в метод по ссылке. Часто первоначальное значение таких переменных не важно. Эти значения перезаписываются в функции.

В C# требуется, чтобы переменные были инициализированы каким-то начальным значением перед тем, как к ним будет выполнено обращение. Хотя можно инициализировать входные переменные какими-то бессмысленными значениями до передачи их в функцию, которая наполнит их осмысленными значениями. Этот прием выглядит в лучшем случае излишним, а в худшем – сбивающим с толку. Тем не менее, существует способ обойти требование компилятора C# относительно начальной инициализации переменных.

Это достигается ключевым словом out. Когда входной аргумент снабжен префиксом out, этому методу можно передать неинициализированную переменную. Переменная передается по ссылке, поэтому любые изменения, выполненные методом в переменной, сохраняются после того, как он вернет управление. Ключевое слово out также должно указываться при вызове метода – так же, как при его определении.

Язык C# позволяет задавать некоторым параметрам значения по умолчанию – так, чтобы при вызове метода можно было опускать часть параметров. Для этого при реализации метода нужным параметрам следует присвоить значение прямо в списке параметров. Параметры по умолчанию можно ставить только в правой части списка параметров. Например, в случае, приведенном на рисунке 6.1, вызывать метод можно следующим образом:

GetData(10, 20);

GetData(10);

```
private void GetData(int Number, int Optional = 5)
{
    Console.WriteLine("Number: {0}", Number);
    Console.WriteLine("Optional: {0}", Optional);
}
```

Рисунок 6.1 – Метод с параметрами по умолчанию

Язык C# позволяет создавать несколько методов с одинаковыми именами, но разными параметрами. Компилятор автоматически подберет наиболее подходящий метод при построении программы. Например, можно написать два отдельных метода возведения числа в степень: для целых чисел будет применяться один алгоритм, а для вещественных – другой (рисунок 6.2). Вызывается такой код одинаково, разница лишь в параметрах – в первом случае компилятор вызовет метод Pow с целочисленными параметрами, а во втором – с вещественными.

```
/// <summary>
/// Вычисление X в степени Y для целых чисел
/// </summary>
private int Pow(int X, int Y)
{
    int b = 1;
    while (Y != 0)
        if (Y % 2 == 0)
        {
            Y /= 2;
            X *= X;
        }
        else
        {
            Y--;
            b *= X;
        }
    return b;
}
/// <summary>
/// Вычисление X в степени Y для вещественных чисел
/// </summary>
private double Pow(double X, double Y)
{
    if (X != 0)
        return Math.Exp(Y * Math.Log(Math.Abs(X)));
    else if (Y == 0)
        return 1;
    else
        return 0;
}
```

Рисунок 6.2 – Перегрузка метода

Порядок выполнения работы

Задание 1

Написать метод, возвращающий наибольшее из двух чисел. Входные параметры метода – два целых числа.

Задание 2

Написать метод, который меняет местами значения двух передаваемых параметров. Параметры передавать по ссылке.

Задание 3

Написать метод вычисления факториала числа, результат вычислений передавать в выходном параметре.

Задание 4

Дана строка символов, состоящая из произвольных десятичных цифр, разделенных пробелами. Вывести количество четных чисел в этой строке.

Задание 5

Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Сформировать новую строку, состоящую из чисел длин слов в исходной строке.

Задание 6

Написать функцию, заменяющую в целочисленном массиве все элементы, кратные пяти, на число 20.

Задание 7

Даны три целых числа. Определить максимальное число, используя функцию нахождения максимума среди двух чисел.

Задание 8

Написать функцию, которая будет вычислять факториал введенного с клавиатуры числа.

Задание 9

Даны основания и высоты двух трапеций. Определить, у какой трапеции площадь больше.

Задание 10

Даны координаты вершин треугольника и радиус круга. Определить площади фигур.

Вопросы для самоконтроля

1 Что такое метод?
2 Параметры и аргументы метода.
3 Доступ к методу.
4 Что такое перегрузка методов?

7 Лабораторная работа № 7. Классы

Цель работы: изучить структуру и принципы объявления классов, освоить технологию создания экземпляров классов (объектов).

Теоретические основы

Класс – это тип данных, объединяющий данные и методы их обработки. Класс – это пользовательский шаблон, в соответствии с которым можно создавать объекты. То есть класс – это правило, по которому будет строиться объект. Сам класс не содержит данных.

Объект класса (экземпляр класса) – переменная типа класс. Объект содержит данные и методы, манипулирующие этими данными. Класс определяет, какие данные содержит объект и каким образом он ими манипулирует.

Все что справедливо для классов можно распространить и на структуры. Отличие состоит в методе хранения объектов данных типов в оперативной памяти: структуры – это типы по значению, они размещаются в стеке; классы – это ссылочные типы, объекты классов размещаются в куче. Структуры не поддерживают наследование.

Структуры применяются для представления небольших объемов данных. Объявление структур происходит с использованием ключевого слова struct, объявление классов – с помощью ключевого слова class. При создании как классов, так и структур, используется ключевое слово new.

Пример объявления класса:

```
class Person
{
   public string name = "Undefined";   // имя
   public int age;                      // возраст
   public void Print()
   {
      Console.WriteLine($"Имя: {name}  Возраст: {age}");
   }
```

Данные и функции, объявленные внутри класса, называются членами класса (class members). Доступность членов класса может быть описана как public, private, protected, internal или internal protected.

Данные-члены – это те структуры внутри класса, которые содержат данные класса – поля, константы события.

Поля – это любые переменные, ассоциированные с классом. После создания экземпляра класса к полям можно обращаться с использованием синтаксиса, например: *имя поля имя объекта*.

Функции-члены – это члены, которые обеспечивают некоторую функциональность для манипулирования данными классов. Они делятся наследующие виды: методы, свойства, конструкторы, финализаторы, операции и индексаторы.

Методы (method) – это функции, ассоциированные с определенным классом. Как и данные-члены, по умолчанию они являются членами экземпляра. Они могут быть объявлены статическими с помощью модификатора static.

Свойства (property) – это наборы функций, которые могут быть доступны клиенту таким же способом, как общедоступные поля класса. В C# предусмотрен специальный синтаксис для реализации чтения и записи свойств для классов, поэтому писать собственные методы с именами, начинающимися на Set и Get, не понадобится. Поскольку не существует какого-то отдельного синтаксиса для свойств, который отличал бы их от нормальных функций, создается иллюзия объектов как реальных сущностей, предоставляемых клиентскому коду.

Конструкторы (constructor) – это специальные функции, вызываемые автоматически при инициализации объекта. Их имена совпадают с именами клас-

сов, которым они принадлежат, и они не имеют типа возврата. Конструкторы полезны для инициализации полей класса.

Финализаторы (finalizer) похожи на конструкторы, но вызываются, когда среда CLR определяет, что объект больше не нужен. Они имеют то же имя, что и класс, но с предшествующим символом тильды (~). Предсказать точно, когда будет вызван финализатор, невозможно.

Операции (operator) – это простейшие действия вроде + или –. Когда вы складываете два целых числа, то, строго говоря, применяете операцию + к целым. Однако C# позволяет указать, как существующие операции будут работать с пользовательскими классами (так называемая перегрузка операций).

Индексаторы (indexer) позволяют индексировать объекты таким же способом, как массив или коллекцию.

Ключевое слово partial (частичный) позволяет определить класс, структуру или интерфейс, распределенный по нескольким файлам. Но ситуации, когда множеству разработчиков требуется доступ к одному и тому же классу, или же в ситуации, когда некоторый генератор кода генерирует часть класса, такое разделение класса на несколько файлов может оказаться полезным. Ключевое слово partial просто помещается перед классом, структурой или интерфейсом.

Статический класс функционально представляет собой то же самое, что и класс с приватным статическим конструктором. Создать экземпляр такого класса невозможно. Если указать ключевое слово static в объявлении класса, компилятор будет гарантировать, что к этому классу никогда не будут добавлены нестатические члены.

Классы .NET изначально унаследованы от System.Object. Фактически, если при определении нового класса базовый класс не указан, компилятор автоматически предполагает, что он наследуется от Object. Практическое значение этого в том, что помимо методов и свойств, которые программист определяет самостоятельно, также появляется доступ к множеству общедоступных и защищенных методов-членов, которые определены в классе Object. Эти методы присутствуют во всех определяемых классах.

Конструктор – это метод класса, который не возвращает значения и имеет то же самое имя, что и класс. Если конструктор класса не определен программистом явно, то компилятор создаст конструктор по умолчанию. Конструкторы подчиняются тем же правилам перегрузки, что и все методы.

В C# поддерживается перегрузка методов – то есть может существовать несколько версий одного метода, но с разными сигнатурами (методы отличаются количеством и / или типом параметров). Чтобы перегрузить метод, просто объявляются методы с одинаковыми именами, но разными сигнатурами.

Порядок выполнения работы

1 Разработать класс для представления объекта «Прямоугольный параллелепипед». Реализуйте все необходимые поля данных (закрытые) и методы, позволяющие:

– устанавливать и считывать значения полей данных;

– вычислять объем прямоугольного параллелепипеда;

– вычислять площадь поверхности прямоугольного параллелепипеда;

– выводить полную информацию об объекте в консоль.

Решение данной задачи состоит из этапов: объявление класса Parallelepiped и демонстрация использования объекта данного класса.

В данном примере необходимо обратить внимание на тот факт, что все вычисления выполняются внутри класса. Метод Main содержит только вызовы методов класса, т. е. вся реализация скрыта.

Пример кода решения поставленной задачи приведен ниже:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace LR_Three
{
class Parallelepiped
{
// Поля данных представляют стороны параллелепипеда
private double a, b, c;
// Методы для установки и считывания значений полей
public void Set_a(double pa) { apa; }
public double Get_a() { return a; }
public void Set_b(double pb) { bpb; }
public double Get_b() { return b; }
public void Set_c(double pc) {cpc; }
public double Get_c() { return c; }
// Метод для вычисления объема прямоугольного параллелепипеда
public double GetV()
{
return a*b*c;
}
// Метод для вычисления площади поверхности прямоугольного параллелепипеда
public double GetS() {
}
return 2* (a*b+b*c+a*c);
}
// Метод для вывода полной информации об объекте в консоль
public void PrintFullInformation()
{
Console.WriteLine("Стороны прямоугольного параллелепипеда: \n" +
"высота= {0}\n "+ "длина (1)"+
"ширина (2)", a, b, c);
```

```
Console.WriteLine("Объем параллелепипеда равен (0)", GetV());
Console.Write("Площадь поверхности равна (0)", GetS());
}
}
class Program
{
static void Main(string[] args)
{
Console.ForegroundColor = ConsoleColor.Yellow;
Console. Title = "Прямоугольный параллелепипед";
Console.Clear();
Console.BackgroundColor = ConsoleColor.Red;
Parallelepiped p;
p=new Parallelepiped();
p.Set_a(10.5);
p.Set_b(25.67);
p.Set_c(40);
p.PrintFullInformation();
Console.ReadKey();
}
}
}
```

2 Класс «Цилиндр». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.

3 Класс «Шар». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.

4 Класс «Точка в пространстве». Реализовать ввод и вывод полей данных, вычисление расстояния до введенной пользователем точки, расстояния от начала координат, а также вывод информации об объекте.

5 Класс «Отрезок». Реализовать ввод и вывод полей данных (координаты начала и координаты конца отрезка), вычисление длины, расстояний начала и конца отрезка от начала координат, а также вывод информации об объекте.

Вопросы для самоконтроля

1 Что такое класс?
2 Что такое структура? Чем структура отличается от класса?
3 Что такое члены класса? Какие группы членов класса вы знаете?
4 Какие модификаторы доступности членов класса вы знаете?
5 Какое ключевое слово используется для создания объекта класса?
6 Что такое конструктор класса?

8 Лабораторная работа № 8. Работа с формами

Цель работы: получить навыки использования диалоговых окон и меню в Windows Form.

Теоретические основы

Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft.NET Framework.

Классы из пространства имен System.Windows.Forms позволяют писать программы, похожие на привычные приложения Windows. Они позволяют использовать кнопки, списки, текстовые поля, меню, окна сообщений и множество других «элементов управления».

Элементы управления нужны для вывода информации, например, текстовой (элемент управления Label) или графической (элемент управления PictureBox), либо для выполнения определенных действий, например, выбора значения или перехода к другой форме после нажатия кнопки. Все элементы управления помещаются на форму.

В Windows Forms форма является видимой поверхностью, на которой отображается информация для пользователя. Обычно приложение Windows Forms строится путем помещения элементов управления на форму и написанием кода для реагирования на действия пользователя, такие как щелчки мыши или нажатия клавиш.

При выполнении пользователем какого-либо действия с формой или одним из ее элементов управления создается событие. Приложение реагирует на эти события с помощью кода и обрабатывает события при их возникновении.

С помощью конструктора Windows Forms Visual Studio, поддерживающего перетаскивание, можно легко создавать приложения Windows Forms: достаточно выделить элемент управления курсором и поместить его на нужное место на форме. Конструктор предоставляет такие средства, как линии сетки и «привязка линий» для преодоления трудностей выравнивания элементов управления.

Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть. Например, при делении двух чисел необходимо предвидеть ситуацию деления на ноль. Язык C# предоставляет возможности для обработки таких ситуаций.

Основу обработки исключительных ситуаций в C# составляет пара ключевых слов try и catch. Эти ключевые слова действуют совместно и не могут быть использованы порознь. Ниже приведена общая форма определения блоков try/catch для обработки исключительных ситуаций:

```
try
{
// Блок кода, проверяемый на наличие ошибок.
}
catch (ExcepType1 exOb)
```

```
{
// Обработчик исключения типа ExcepType1.
}
catch (ExcepType2 exOb)
{
// Обработчик исключения типа ExcepType2.
}
```

ExcepType – это тип возникающей исключительной ситуации.

Когда исключение генерируется оператором try, оно перехватывается составляющим ему пару оператором catch, который затем обрабатывает это исключение. В зависимости от типа исключения выполняется и соответствующий оператор catch. Так, если типы генерируемого исключения и того, что указывается в операторе catch, совпадают, то выполняется именно этот оператор, а все остальные пропускаются.

Когда исключение перехватывается, переменная исключения exOb получает свое значение. На самом деле указывать переменную exOb необязательно. Так, ее необязательно указывать, если обработчику исключений не требуется доступ к объекту исключения, что бывает довольно часто. Для обработки исключения достаточно и его типа.

Следует, однако, иметь в виду, что если исключение не генерируется, то блок оператора try завершается как обычно, и все его операторы catch пропускаются. Выполнение программы возобновляется с первого оператора, следующего после завершающего оператора catch. Таким образом, оператор catch выполняется лишь в том случае, если генерируется исключение.

Исключения в C# представлены классами. Все классы исключений могут быть унаследованы от встроенного класса исключений Exception, который является частью пространства имен System. Именно поэтому все исключения представляют собой подклассы класса Exception.

Порядок выполнения работы

1 Создайте форму и измените ее свойства.

Запустите программу Visual Studio. Выполните команду Файл – Создать – Проект. В появившемся диалоговом окне в списке приложений выберите Windows Forms, введите имя проекта Form6_1 и укажите место его сохранения. В результате на экране появится окно проекта Form6_1. В рабочей области находится форма в режиме конструктора.

Измените заголовок формы Form1 на текст Калькулятор. Для этого выполните щелчок правой кнопкой мышки на форме в режиме конструктора и в контекстном меню выберите команду Свойства. В правом нижнем углу экрана появится окно Свойства. Измените значение свойства Text с Form1 на Калькулятор.

С помощью свойства Name измените имя формы на MainForm.

В окне Свойства измените значения свойства Size: ширина – 470, длина – 500.

Установите расположение формы после загрузки в центре экрана с помощью значения CenterOwner свойства StartPosition.

Свойство StartPosition – расположение формы при загрузке. Значения:

– Manual – положение формы на экране, задается свойствами Left и Top;

– CenterScreen – в центре экрана;

– WindowsDefaultLocation – положение задается системой, исходя из количества открытых окон и расположения их на экране;

– WindowsDefaultBounds – форма отображается в месте и с размерами окна, заданными по умолчанию в Windows;

– CenterParent – расположение формы в центре родительской.

Измените цвет фона формы на белый с помощью свойства BackColor.

2 Добавьте на форму проекта элементы управления, которые позволят выполнять операции сложения, вычитания, умножения и деления двух чисел.

Добавьте обработчик событий загрузки формы. Для этого щелкните по форме и в окне Свойства переключитесь на вкладку работы с событиями элементов управления, нажав на кнопку События, и дважды кликните по событию Load – справа появится надпись MainForm_Load, и в код класса MainForm будет добавлен метод-обработчик события загрузки формы с тем же именем.

Отредактируйте метод MainForm_Load в соответствии с приведенным ниже кодом:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace Form
{
public partial class MainForm Form
{
public MainForm()
{
InitializeComponent();
}
private void MainForm_Load(object sender, EventArgs e)
{
// Добавляем элементы в список математических операций
comboBox1 Mop.Items.Add("+");
comboBox1 Mop.Items.Add("-");
comboBox1 Mop.Items.Add("*");
comboBox1 Mop.Items.Add("/");
// Устанавливаем выбранный элемент с индексом 0
}
comboBox1 Mop.SelectedIndex = 0;
```

```
}
}
```

Дважды щелкните по кнопке Вычислить. Автоматически будет добавлен обработчик события по умолчанию для данного элемента управления (для кнопки это щелчок) – метод button_Calculate_Click. Добавьте в него код, приведенный ниже:

```
private void button Calculate Click(object sender, EventArgs e)
// Определяем текст, введенный пользователем
// в поле "Число 1"
string sx1 = textBox Arg1.Text;
// Преобразуем текст в число int x1= Convert.ToInt32(sx1);
// Определяем текст, введенный пользователем
// в поле "Число 2"
string sx2 = textBox Arg2.Text;
// Преобразуем текст в число int x2 = Convert.ToInt32(sx2);
// Определяем, какую математическую операцию выбрал пользователь
string mop = comboBox1 Mop.SelectedItem.ToString();
double result = 0;
if (mop == "+")
{
result = x1 + x2;
}
else if (mop == "-")
{
result = x1 - x2;
}
else if (mop == "*")
{
result = x1* x2;
}
else
{
result = x1 / x2;
}
// Выводим результат в текстовое поле
textBox Result.Text = "= " + result;
}
```

3 Создайте меню с командами Input, Calc и Exit.

При выборе команды Input открывается диалоговое окно, содержащее:

– три поля типа TextBox для ввода длин трех сторон треугольника;

– группу из двух флажков (Периметр и Площадь) типа CheckBox;

– кнопку типа Button.

Обеспечьте возможность:

– ввода длин трех сторон треугольника;

– выбора режима с помощью флажков: подсчет периметра и/или площади треугольника. При выборе команды Calc открывается диалоговое окно с результатами. При выборе команды Exit приложение завершается.

Вопросы для самоконтроля

1 Что такое Windows Forms?

2 Перечислите элементы управления Windows Forms и укажите их основные свойства.

3 Как изменить свойства элемента управления?

4 Как создать обработчик события элемента управления?

5 Где описывается код обработчика события?

6 Какие ключевые слова используются для обработки исключений?

Список литературы

1 **Гагарина, Л. Г.** Технология разработки программного обеспечения : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул ; под ред. Л. Г. Гагариной. – Москва : ФОРУМ ; ИНФРА-М, 2019. – 400 с.

2 **Васильев, А.** Программирование на C# для начинающих. Основные сведения / А. Васильев. – Москва : Эксмо, 2018. – 592 с.