МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждениевысшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению практических работ
по дисциплине «Информационные технологии и программирование»
Для студентов направления подготовки 09.03.02 Информационные системы и технологии, направленность (профиль) Цифровые технологии химических производств

(ЭЛЕКТРОННЫЙ ДОКУМЕНТ)

ОГЛАВЛЕНИЕ

Введение 6

Практическая работа 1. Структура консольного приложения в C# 8

Практическая работа 2. Предопределенные типы данных, переменные, константы 17

Практическая работа 3. Использование возможностей консольного ввода-вывода. 27

Практическая работа 4. Управление потоком выполнения с использованием операторов if, switch 34

Практическая работа 5. Управление потоком выполнения с использованием оператора цикла for 42

Практическая работа 6. Управление потоком выполнения с использованием операторов while 47

Практическая работа 7. Управление потоком выполнения с использованием оператора do …while 53

Практическая работа 8. Классы. Структуры 57

Практическая работа 9. Конструктор класса. Перегрузка конструкторов класса. 71

Практическая работа 10. Многомодульные приложения 74

Практическая работа 11. Операции классов. Перегрузка операций 87

Практическая работа 12. Построение иерархии классов 93

Практическая работа 13. Разработка пользовательских интерфейсов 107

Практическая работа 14. Файловый ввод-вывод. Работа с каталогами. Работа с файлами 120

Практическая работа 15. Решение вычислительной задачи с применением файлового ввода-вывода 146

Практическая работа 16. Создание приложения по технологии Windows Forms 151
Практическая работа 17. Применение элементов управления в приложениях Windows..... 161
Заключение 175
Список литературы 176

ВВЕДЕНИЕ

В настоящее время особое значение для всех отраслей экономики принимает автоматизация производственных процессов. Для разработки приложений масштаба предприятия необходимо владеть современными методами создания сложных программных продуктов, а также инструментарием разработки программных средств.

Особое значение при подготовке специалиста в сфере информационных систем и технологий имеет практическая подготовка специалиста: производственные задачи автоматизации связаны с многопоточным программированием, использованием баз данных, применением сложных структур данных.

Сложность освоения технологий программирования, в совокупности с постоянным ростом значимости информационных технологий, указывает на необходимость подготовки специалистов в области программирования.

Основная цель изучения дисциплины «Основы алгоритмизации и программирования»: изучение и получение практических навыков в области современных средств и методов разработки, алгоритмов, основных синтаксических конструкция языка высокого уровня.

Задачами изучения дисциплины «Основы алгоритмизации и программирования» являются:

– усвоение теоретических знаний о структуре и принципах построения современных программных комплексов, а также об архитектуре современных программных платформ для автоматизации проектирования информационных систем;

– получение практических навыков разработки приложений с использованием языка высокого уровня C#;

– изучение основных алгоритмических приемов решения задач с использованием C#;

– освоение принципов разработки программного обеспечения на основе технологий Microsoft .NET Framework;

– освоение принципов использования перспективных инструментальных средств разработки и проектирования приложений.

Знания, полученные студентами в результате выполнения заданий практикума, позволит студентам применять полученные знания при разработке программных средств, как для образовательных целей, так и в профессиональной деятельности.

ПРАКТИЧЕСКАЯ РАБОТА 1. СТРУКТУРА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ В C#

1. Цель и содержание

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

Задачи работы:

– научиться объявлять переменные простых типов в языке C#;

– научиться объявлять константы простых типов в языке C#;

– научиться выполнять простейшие действия с переменными и константами.

2. Формируемые компетенции

Работа направлена на формирование следующих компетенций:

– способность к проектированию базовых и прикладных информационных технологий (ПК-11);

– способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные) (ПК-12).

3. Теоретическая часть

Рассмотрим структуру консольного приложения на языке C#, созданного с использованием средств MS Visual Studio (рис. 1.1).

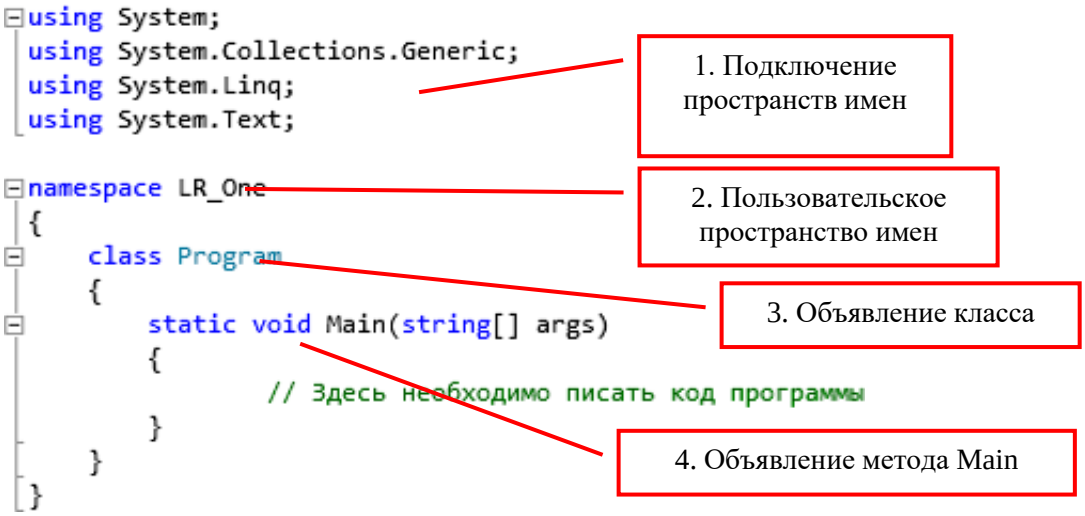


Рисунок 1.1 – Исходный код консольного приложения.

В интерактивном редакторе кода среды разработки код может быть свернут/ развернут с использованием кнопок +/-, внедренных в код. На рис. 1.1 в области 1 показано подключение пространств имен (библиотек, содержащих стандартные инструменты) с использованием зарезервированного слова using.

Программист сам создает свое пространство имен с именем LR_One (2), в котором объявляется класс с именем Program.

В классе Program объявлен один метод – функция Main (параметры функции не рассматриваем).

Функция Main имеет особенное значение в программировании на языках C, C++ и C#.

Функцию Main называют «точкой входа», то есть началом выполнения программы. Далее мы рассмотрим приложения, содержащие множество функций. Операционная система знает с какой именно функции начать выполнение программы – с функции Main. Очевидно, что имя этой функции менять нельзя. Это должен быть статический метод класса (или структуры), возвращающий либо значение типа int, либо void. Хотя нередко модификатор public указывается явно, поскольку по определению этот метод должен быть вызван извне программы, на самом деле неважно, какой уровень доступа вы назначите методу точки входа. Он запустится, даже если вы пометите его как private.

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.

Обратите внимание на вложенность конструкций на рис. 1.1: в пространство имен LR_One вложен класс Program, в класс Program вложена функция Main. В свою очередь, в функции Main содержатся инструкции на языке C#-код, который начнет выполняться при старте программы.

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 – сослеша со звездочкой (/*) и заканчиваются противоположной комбинацией (*/). В этих аспектах язык C# идентичен C++ и Java.

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

Причина присутствия оператора using в файле Program.cs связана с использованием библиотечных классов.

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.

4. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный

(x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; неразрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение, для этого выполните следующие действия:

1.1 Выберите команду главного меню $File \rightarrow New \rightarrow Project...$

1.2 В открывшемся диалоговом окне (рис. 1.2) выберите необходимые настройки для создаваемого проекта: язык Visual C#; фреймворк: .NET Framework 4; шаблон: Console Application.

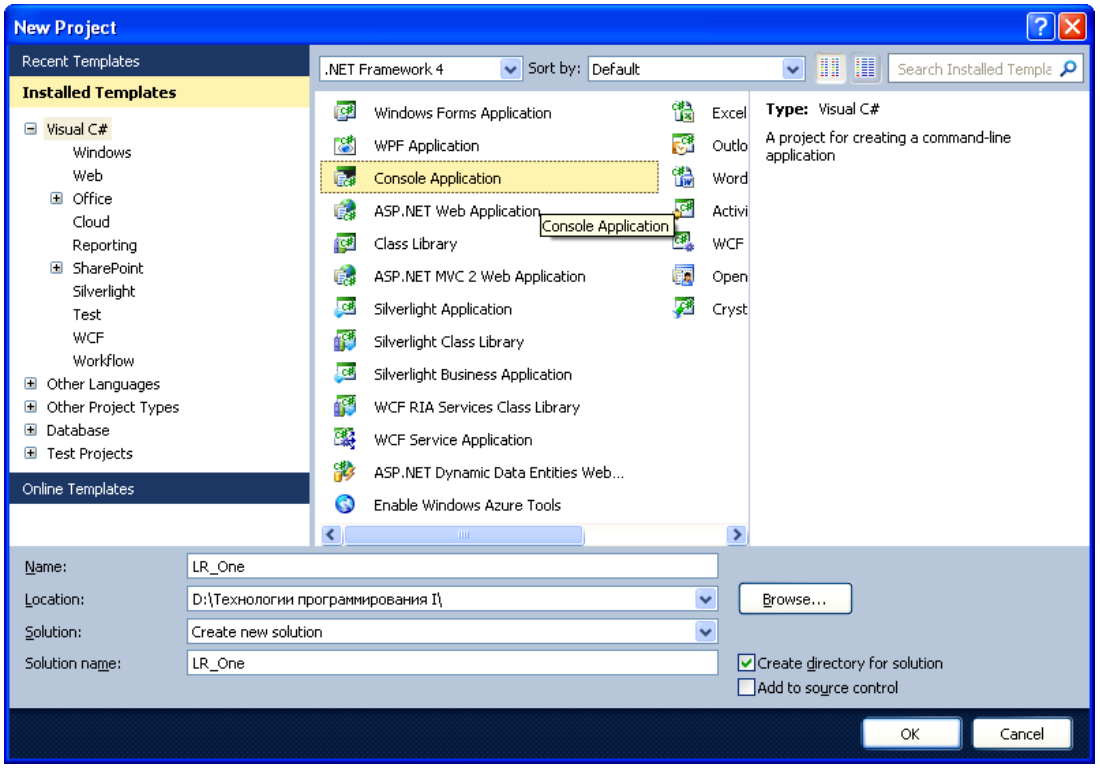


Рисунок 1.2 – Создание нового проекта консольного приложения.

1.3 В текстовом поле Name введите имя проекта (например LR_One).

1.4 В текстовом поле Location выберите место сохранения нового проекта.

1.5 Установите флажок-переключатель «Create directory for solution».

1.6 Нажмите кнопку «OK».

2. После выполнения пункта 1 в среде разработки откроется новый созданный проект (рис. 1.3).

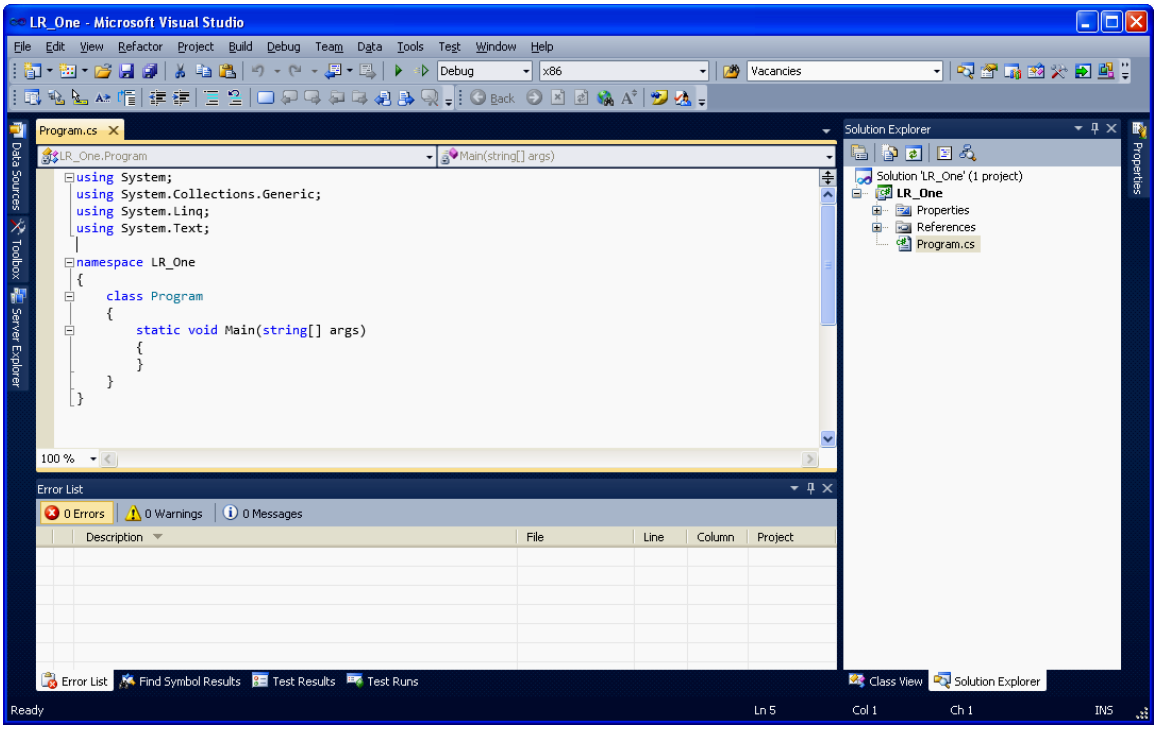


Рисунок 1.3 – Новый проект, загруженный в среду разработки: вкладка «Solution Explorer» отображает состав проекта (нас интересует только файл Program.cs); в левой части окна (сверху) открыт файл Program.cs в редакторе кода.

3. На данном этапе необходимо ознакомиться со структурой исходного файла консольного приложения (рис. 1.4).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
                // Здесь необходимо писать код программы
        }
    }
}
```

Рисунок 1.4 – Исходный файл консольного приложения.

4. Весь код программы необходимо писать внутри функции Main.

5. Для построения сборки (исполняемого exe-файла) выполните команду главного меню *Build → Build Solution* (или использовать горячую клавишу *F6*). После этого сборка создана, но приложение не будет запущено автоматически.

6. Для создания сборки и последующего запуска программы можно воспользоваться командой *Debug → Start Debugging* главного меню среды разработки или нажать кнопку панели инструментов  *Start Debugging* (*F5*). Можно также использовать горячую клавишу *F5*.

7. Запустите приложение на выполнение одним из методов, указанных в пункте 6. Окно консольного приложения появится и исчезнет. Это означает, что приложение выполнило все команды, написанные программистом, и завершило свою работу.

8. Для удержания окна на экране измените исходный файл в соответствии с рисунком 1.5. В функции Main добавлен вызов только одной команды Console.DeadKey ( ) – эта функция останавливает выполнение программы и ждет, когда пользователь нажмет любую клавишу.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.ReadKey();
        }
    }
}
```

Рисунок 1.5 – Исходный файл консольного приложения для предотвращения закрытия окна консольного приложения.

9. Запустите измененное приложение, убедитесь, что окно удерживается на экране.

10. Добавьте несколько строк кода в исходный файл (рис. 1.6).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная работа №1");
            Console.WriteLine("");
            Console.WriteLine("Выполнил: Иванов Иван иванович");
            Console.WriteLine("Группа: ИСТБ-121");
            Console.WriteLine("Наименование ЛР: Структура консольного приложения");
            Console.WriteLine("");
            Console.WriteLine("для завершения работы программы нажмите любую клавишу...");

            Console.ReadKey();
        }
    }
}
```

Рисунок 1.6 – Исходный файл консольного приложения для вывода информации на экран.

11. Внимательно изучите исходный код примера на рис. 1.5. Запустите приложение и убедитесь, что отсутствуют ошибки и информация выводится.

12. Выполните индивидуальное задание.

Индивидуальное задание.

Измените приложение, созданное в ходе выполнения данной работы таким образом, чтобы программа выводила на экран следующую информацию (каждый студент должен использовать персональнуюинформацию о себе):

– Название и номер работы;

– ФИО студента;

– Группа студента и шифр специальности;

– Дата рождения студента;

– Населенный пункт постоянного места жительства студента;

– Любимый предмет в школе;

– Краткое описание увлечений, хобби, интересов.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

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

2. Что такое «точка входа» в программе?

3. Как вы понимаете термины «пространства имен», «класс», «метод», «функция»?

4. Чем отличается структура консольного приложения от приложения, построенного с использованием технологий Windows Forms, WPF?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [1-3].

ПРАКТИЧЕСКАЯ РАБОТА 2. ПРЕДОПРЕДЕЛЕННЫЕ ТИПЫ ДАННЫХ, ПЕРЕМЕННЫЕ, КОНСТАНТЫ

1. Цель и содержание

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

Задачи работы:

– научиться объявлять переменные простых типов в языке C#;

– научиться объявлять константы простых типов в языке C#;

– научиться выполнять простейшие действия с переменными и константами.

2. Теоретическая часть

Перед выполнением работы необходимо изучить материалы лекций. Следует понимать принцип деления типов .NET на типы значений и ссылочные типы.

В данной работе необходимо освоить приемы работы с предопределенными типами значений.

2.1 Типы значений C#

Язык C# поддерживает 8 предопределенных целочисленных типов (таблица 2.1).

Таблица 2.1 – Целочисленные типы C#.

Имя типа	Тип CTS	Описание	Диапазон (минимум : максимум)
sbyte	System.SByte	8-битное целое со знаком	-128 : 127
short	System.Int16	16-битное целое со знаком	-32 768 : 32 767
int	System.Int32	32-битное целое со знаком	-2 147 483 648 : 2 147 483 647
long	System.Int64	64-битное целое со знаком	-2^{63} : $2^{63}-1$
byte	System.Byte	8-битное целое без знака	0 : 255

ushort	System.UInt16	16-битное целое без знака	0 : 65 535
uint	System.UInt32	32-битное целое без знака	0 : 2^{32}-1
ulong	System.UInt64	64-битное целое без знака	0 : 2^{64}-1

Язык C# также поддерживает и типы с плавающей точкой (таблица 2.2).

Таблица 2.2 – Типы с плавающей точкой C#.

Имя типа	Тип CTS	Описание	Кол-во знаков	Диапазон (минимум : максимум)
float	System.Single	32-битное с плавающей точкой одинарной точности	7	от $\pm 1.5\times10^{-45}$ до $\pm 3.4\times10^{38}$
double	System.Double	64-битное с плавающей точкой двойной точности	15/16	от $\pm 5.0\times10^{-324}$ до $\pm 1.7\times10^{308}$

В таблице 2.3 представлен десятичный тип C#. Данный тип реализован для финансовых операций.

Таблица 2.3 – Десятичный тип C#.

Имя типа	Тип CTS	Описание	Кол-во знаков	Диапазон (минимум : максимум)
decimal	System.Decimal	128-битное с плавающей точкой в десятичной нотации с высокой точностью	28	от $\pm 1.0\times10^{-28}$ до $\pm 7.9\times10^{28}$

Как и во многих языках программирования существует булевский тип (таблица 2.4).

Таблица 2.4 – Булевский тип.

Имя типа	Тип CTS	Значения
bool	System.Boolean	true или false

Для хранения одиночных символов в языке C# используется тип char (таблица 2.5)

Таблица 2.5 – Булевский тип.

Имя типа	Тип CTS	Значения
char	System.Char	Представляет отдельный 16-битный (Unicode) символ

Литералы типа char записываются как одиночные, заключенные в одинарные кавычки символы: ‘F’, ‘w’, ‘ц’, ‘Я’ и т.д.

В переменных типа char можно хранить и специальные символы в виде управляющих последовательностей (таблица 2.6).

Таблица 2.6 – Представление символов в виде управляющих последовательностей.

Управляющая последовательность	Символ
\’	Одиночная кавычка
\”	Двойная кавычка

\\	Обратный слэш
\0	Пусто
\a	Предупреждение (звуковой сигнал)
\b	Забой
\f	Подача формы
\n	Новая строка
\r	Возврат каретки
\t	Символ табуляции
\v	Вертикальная табуляция

Если отдельные символы объединены в строку, то необходимо использовать тип string, который отображается на тип CTS – System.String.

2.2 Объявление и инициализация переменных в C#

Синтаксис объявления переменных в C# выглядит следующим образом:

ТипДанных ИдентификаторПеременной;

Например:

```
int a;
```

Этот код объявляет переменную типа int с именем a. Компилятор не позволит использовать эту переменную до тех пор, пока она не будет инициализирована (т.е. пока ей не будет присвоено значение).

Для инициализации переменной a необходимо написать следующий код:

```
a = 123;
```

Переменную можно инициализировать во время объявления:

```
int b = 7;
```

или

```
string str = "Hello, World!!!";
```

Синтаксис C# позволяет объявить несколько переменных (и инициализировать их) одного типа в одной синтаксической конструкции.

Например:

```
float b, i, myPerem, U_t = 0.3F, z_11 = 23.56F;
```

В данном примере объявляется 5 переменных типа float, некоторые из них инициализируются в процессе объявления.

2.3 Объявление и инициализация констант в C#

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

```
const char simv = 'A';
const double pi = 3.14;
```

Очевидно, что при таком объявлении, поменять значения simv и pi в дальнейшем будет нельзя.

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место

пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом описанном в работе №1.

2. Изучите материал в разделе «Теоретическое обоснование» данной работы.

3. Модифицируйте исходный файл как показано на рис. 2.1.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace LR_One
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12             int a;
13             int b = 7;
14
15             string str = "Hello, World!!!";
16
17             a = 123;
18
19             Console.ReadKey();
20         }
21     }
22 }
```

Рисунок 2.1 – Объявление переменных в C#.

4. Рассмотрим приведенный пример подробнее:

4.1. В строке 12 объявляется переменная типа int с именем a.

4.2. В строке 13 объявляется переменная типа int с именем b, причем при объявлении для нее устанавливается начальное значение, равное 7.

4.3. В строке 15 объявляется переменная типа string с именем str и инициализируется строковым значением «Hello, World!!!».

4.4. В строке 17 переменной a присваивается целочисленное значение 123.

5. Запустите приложение на выполнение. У вас должно появиться пустое окно консольного приложения. Очевидно, что исходный код, представленный на рис. 2.1 не предполагает вывода какой-либо информации на экран.

6. Для вывода информации на экран воспользуемся функцией Console.WriteLine, изученной в работе 1. Добавим в исходный файл следующие строки (строки 19 – 21 на рис. 2.2).

```
6  namespace LR_One
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12             int a;
13             int b = 7;
14
15             string str = "Hello, World!!!";
16
17             a = 123;
18
19             Console.WriteLine(a);
20             Console.WriteLine(b);
21             Console.WriteLine(str);
22
23             Console.ReadKey();
24         }
25     }
26 }
27
```

Рисунок 2.2 – Добавление строк кода для вывода значений объявленных переменных на экран.

7. В примере на рис. 2.2 используется простой вывод, то есть имя переменной просто передается в качестве параметра функции Console.WriteLine.

8. Для выполнения форматного вывода (изменения формата представления выводимой информации) необходимо реализовать вывод в следующем виде (рис. 2.3 изменены строки 19-21):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            int a;
            int b = 7;

            string str = "Hello, World!!!";

            a = 123;

            Console.WriteLine("Значение переменной a равно {0}", a);
            Console.WriteLine("a значение переменной b равно {0}", b);
            Console.WriteLine("значение a+b = {0}+{1} = {2}", a, b, a+b);
            Console.WriteLine(str);

            Console.ReadKey();
        }
    }
}
```

Рисунок 2.3 – Вывод информации с использованием форматных строк.

9. Внимательно изучите код полученной программы, Затем выполните индивидуальное задание.

Индивидуальное задание.

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 в соответствие с вариантом:

Вариант	Выражение для вычисления
1	$F = a\_1 + b - a \cdot (x + y5)$
2	$Se = w111 + bt - x + y \cdot w$
3	$R\_x = a \cdot b + b / t - x + f \cdot i\_2$
4	$c = gh + b \cdot q3 - x + y / w$
5	$H = \frac{g \cdot h}{d17} + b / h1 - \frac{x + y}{4}$
6	$Z = \frac{35}{f} + y \cdot f - \frac{f + y}{4}$

7	$a = \frac{35}{a} \cdot z + z \cdot a - \frac{a + Et}{4}$
8	$A0 = \frac{35}{G\_1} \cdot Zvcw + G\_1 \cdot a - \frac{a + Zvcw}{a}$
9	$Se = w111 + bt - x + y \cdot w$
10	$R\_x = a \cdot b + b / t - x + f \cdot i\_2$
11	$g = w \cdot h + b \cdot q3 - q3 + y / w$
12	$ee = \frac{g \cdot h}{d17} + d17 / h - \frac{g + d17 + h}{4}$
13	$Zze = \frac{35}{f} + y \cdot f - \frac{f + y}{4}$
14	$t = \frac{35}{a} \cdot z + z \cdot a - \frac{a + Et}{4}$
15	$A0 = \frac{35}{G\_1} \cdot Zvcw + G\_1 \cdot a - \frac{a + Zvcw}{a}$
16	$Zxy = a\_5 + b - a\_5 \cdot (b + y)$
17	$ch = \frac{6}{f1} \cdot z + z \cdot f1 - \frac{f1 + Et}{6}$
18	$ch = rx \cdot z + z \cdot f1 - \frac{f1 + Et}{rx}$
19	$\_H = k1 + k2 - k3 + \frac{k1 \cdot k2}{k3}$
20	$Fy = a\_1 + b - a\_1 \cdot (x + y)$
21	$\_G\_1 = y1 + y2 + y3 - \frac{y1 + y2 + y3}{3}$
22	$R = 2 \cdot x + 2 \cdot y - 4 \cdot x \cdot y + z$
23	$g11 = t \cdot z + z \cdot f1 - \frac{f1 + U}{t}$
24	$Y = Z \cdot (z + x) + z / Z - \frac{z + Z}{x + X}$
25	$U = rx \cdot z + z \cdot Y - \frac{Y + z}{rx}$

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое переменная? Как объявляется переменная?

2. Как объявляется константа? Чем константа отличается от переменной?

3. Какие типы значений применяются C#?

4. Чем тип char отличается от типа string?

5. Как производится инициализация переменных? Как производится инициализация констант?

6. Что такое управляющие последовательности?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [1], [7].

ПРАКТИЧЕСКАЯ РАБОТА 3. ИСПОЛЬЗОВАНИЕ ВОЗМОЖНОСТЕЙ КОНСОЛЬНОГО ВВОДА-ВЫВОДА.

1. Цель и содержание

Цель работы: изучить команды ввода и вывода данных в консольном приложении.

Задачи работы:

– научиться применять функции Console.Write и Console.WriteLine;

– научиться применять функции Console.Read и Console.ReadLine;

– научиться использовать форматы вывода для различных типов данных.

2. Теоретическая часть

В предыдущих лабораторных работах уже использовалась команда Console.WriteLine. Рассмотрим ее синтаксис подробнее:

```
Console.WriteLine("Строка текста");
```

Эта запись означает вызов статического метода класса Console. В качестве параметра функции передается строка, которая выводится в консоль.

Существует также функция:

```
Console.Write("Строка текста");
```

Этот метод также осуществляет вывод, только не переводит каретку на следующую строку.

Рассмотренные методы также позволяют осуществлять форматный вывод, аналогичный функции printf языка Си, например:

```
int a = 100, b = 13, c = 23, d = 0, e = 0;
Console.WriteLine("Переменная a = {0}, переменная b = {1}," +
"переменная c = {2}, " +
"переменная d = {3}, " +
"переменная e = {4}", a, b, c, d, e);
```

В данном примере в качестве первого параметра передается строка, содержащая маркеры в фигурных скобках. При выводе на места маркеров будут подставлены параметры, которые следуют за строкой.

При форматном выводе можно задавать ширину поля вывода, для этого необходимо воспользоваться следующим приемом:

```
int a = 100, b = 13, c = 23, d = 0, e = 0;
Console.WriteLine("Переменная a = {0, 5}, переменная b = {1, 5}," +
"переменная c = {2, 5}, " +
"переменная d = {3, 5}, " +
"переменная e = {4, 5}", a, b, c, d, e);
```

В данном примере видно, чтобы задать ширину вывода необходимо использовать в формате {n, w}, где n – порядковый номер параметра, а w – ширина области вывода (знак w позволяет осуществлять выравнивание выводимого значения по левому или правому краю).

При выводе также можно добавлять строку формата вместе с необязательным значением точности (таблица 2.1).

Таблица 3.1 – Основные строки формата.

Стока	Описание
C	Локальный формат валюты
D	Десятичный формат. Преобразует целое к основанию 10 и снабжает ведущими нулями, если есть спецификатор точности.
E	Научный (экспоненциальный) формат. Спецификатор точности устанавливает количество десятичных

	разрядов (по умолчанию 6).
F	Формат с фиксированной запятой. Спецификатор точности задает количество десятичных разрядов.
G	Общий формат. Форматирование E или F.
N	Числовой формат. Форматирует число с разделителями тысяч – запятыми.
P	Процентный формат
X	Шестнадцатиричный формат. Спецификатор точности используется для указания ведущих нулей.

Пример использования формата:

```
float a, b, c, res;

Console.Write("Введите a > ");
a = Convert.ToSingle(Console.ReadLine());

Console.Write("Введите b > ");
b = Convert.ToSingle(Console.ReadLine());

Console.Write("Введите c > ");
c = Convert.ToSingle(Console.ReadLine());

res = (a + b) * c;

Console.WriteLine("\n\n");

Console.WriteLine("Денежный формат {0:C} \n", res);
// Console.WriteLine("Десятичный формат {0:D10} \n", res);
Console.WriteLine("Экспоненциальный формат {0:E} \n", res);
Console.WriteLine("Формат с фиксированной запятой {0:F3, 7} \n", res);
Console.WriteLine("Общий формат {0:G} \n", res);
Console.WriteLine("Числовой формат {0:N} \n", res);
Console.WriteLine("Процентный формат {0:P} \n", res);
// Console.WriteLine("Шестнадцатиричный формат {0:X} \n", res);
```

Изучите представленный пример. Выполните данный код в среде выполнения.

В классе Console также существуют методы, позволяющие считывать информацию, вводимую пользователем:

```
string str;

str = Console.ReadLine();

Console.WriteLine("вы ввели строку: " + str);
```

В данном примере строка, введенная пользователем, считывается в переменную str, затее осуществляется ее вывод. Наберите данный пример в среде VS и изучите код.

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом, представленным в работе №1.
2. Выполните индивидуальное задание, выполнив вывод результата с использованием всех возможных форматов, представленных в таблице 2.1.

Индивидуальное задание.

Объявите требуемые переменные, значения переменным пользователь должен присваивать в процессе выполнения программы (считываются с консоли), выведите на экран с использованием форматной строки значения переменных и результат вычисления выражения во всех возможных форматах:

Вариант	Выражение для вычисления
1.	$F = a\_1 + \sin(b) - a \cdot (\lvert x \rvert + y5)$
2.	$Se = \frac{tg(w111) + bt}{x} - \frac{x}{bt} + y \cdot w$
3.	$R\_x = \frac{a \cdot b}{\sin(f)} + b/t - x + f \cdot i\_2$
4.	$c = \cos(gh) + b \cdot \sqrt{q3} - \lvert x - y \rvert / w$
5.	$H = \frac{\sqrt{g \cdot h}}{d17} + \cos(b)/h1 - \frac{x + y}{4}$
6.	$Z = \frac{35}{\lvert f \rvert} + \cos(y \cdot f) - \frac{f + y}{4}$
7.	$a\_q = \frac{35}{a} \cdot z + z \cdot a - \frac{a + Et}{4}$
8.	$A0 = \frac{35}{G\_1} \cdot \cos(Zvcw) + G\_1 \cdot a - \frac{a + \sin(Zvcw)}{a}$
9.	$Se = \sqrt{w111} + \sin(bt) - x + y \cdot w$
10.	$R\_x = a \cdot b + \sqrt{\frac{b}{t}} - x + f \cdot \lvert i\_2 \rvert$
11.	$g = \sqrt{w \cdot h} + \lvert b \rvert \cdot q3 - \cos(q3) + y/w$
12.	$ee = \frac{g \cdot h}{d17} + d17/h - \sqrt{\frac{g + d17 + h}{4}}$
13.	$Zze = \sqrt{\frac{35}{f}} + \cos(y \cdot f) - \frac{f + y}{4}$

14.	$t = \frac{35}{a} \cdot z + \sin(z \cdot a) - \frac{a + \sqrt{Et}}{4}$
15.	$A0 = \frac{35}{G\_1} \cdot \sqrt{Zvcw} + G\_1 \cdot a - \left\lvert \frac{a - Zvcw}{a} \right\rvert$
16.	$Zxy = \sqrt{\frac{a\_5}{b}} + b - a\_5 \cdot (\sin(b) + y)$
17.	$ch = \frac{6}{f1} \cdot \sqrt{z} + z \cdot \sin(f1) - \frac{f1 + Et}{6}$
18.	$ch = \cos(rx) \cdot z + \left\lvert z \cdot f1 - \frac{f1 + Et}{rx} \right\rvert$
19.	$\_H = k1 + \sin(k2) - k3 + \sqrt{\frac{k1 \cdot k2}{k3}}$
20.	$Fy = a\_1 + \lvert b - \sin(a\_1) \cdot (x + y) \rvert$
21.	$\_G\_1 = y1 + \sqrt{y2} + \left\lvert y3 - \frac{y1 + y2 + y3}{3} \right\rvert$
22.	$R = \sqrt{2 \cdot \sin(x)} + \lvert 2 \cdot y - 4 \cdot x \cdot y \rvert + z$
23.	$g11 = t \cdot z + \sqrt{z} \cdot \left\lvert f1 - \frac{f1 + U}{t} \right\rvert$
24.	$Y = Z \cdot \sin(z + x) + \left\lvert z / Z - \frac{\sin(z) + Z}{x + X} \right\rvert$
25.	$U = \sqrt{\lvert rx - z \rvert} + z \cdot \left\lvert Y - \frac{Y + z}{rx} \right\rvert$

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Как оформляется комментарий?
2. Какие функции используются для вывода информации в консоль?
3. Какие функции используются для считывания информации с консоли?
4. Какие форматы вывода вы знаете? Опишите их.
5. Какой класс содержит статические методы для конвертирования значений и приведения их к требуемому типу?
6. Какой класс содержит статические методы – математические функции? Что такое управляющие последовательности?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [4], [7], [8].

ПРАКТИЧЕСКАЯ РАБОТА 4. УПРАВЛЕНИЕ ПОТОКОМ ВЫПОЛНЕНИЯ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРОВ IF, SWITCH

1. Цель и содержание

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

Задачи работы:

– научиться применять условныq оператор if;

– научиться применять оператор цикла for;

– научиться применять операторы выбора switch.

2. Теоретическая часть

2.1 Условный оператор.

В предыдущих лабораторных работах были рассмотрены вопросы программирования на языке C# с использованием только последовательного выполнения операторов программы.

В языке C# (как и в большинстве языков) можно вводить условия, циклы, ветвления.

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

Синтаксис условного оператора:

if (*условие*)
    оператор
[else if (*условие*)
    оператор
...
else
    оператор]

В данном синтаксисе *оператор* может быть составным оператором (группа операторов языка, заключенные в фигурные скобки). Условный оператор содержит только одно обязательное зарезервированное слово – if. Все остальные конструкции не являются обязательными (в синтаксическом описании на это указывают скобки [ ]).

Пример использования условного оператора:

```
int a = 100, b = 13, c = 23, d = 0, e = 0;


if (a == 100) // Если а равно 100
{
    // Увеличиваем b на 1000
    b += 1000;
}
else if (a < 100) // иначе если а меньше 100
{
    // изменяем значение d
    d = b + c;
    // и изменяем значение e
    e = a + b;
}
else // иначе (остальные варианты не предусмотренные первыми двумя условиями)
    // edtkbxbdftv a на 1
    a++;

// продолжается последовательное выполнение программы
Console.WriteLine("{0}", a);
```

В представленном примере сначала проверяется является ли значение переменной a равным 100.

1. Если «Да», то выполняется единственный оператор (увеличение b на 1000). На этом выполнение всего условного оператора завершено и программа переходит к выполнению операторов, следующих за ним, то есть к выводу значения a.

2. Если первое условие не выполнилось, проверяется второе – является ли a меньше 100 (конструкция else if). Если это утверждение истинно, то выполняется составной оператор, состоящий из двух – изменение значений переменных d и e. Затем осуществляется переход к выводу переменной a.

3. Если не выполнилось ни одно условие с оператором if, то выполняется оператор, указанный после зарезервированного слова else. В данном случае это несоставной оператор (a увеличивается на 1), поэтому фигурные скобки можно не писать.

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

Еще один пример условного оператора:

```
int a = 100;

if (a >= 100)
{
    Console.WriteLine("Значение переменной a больше или равно 100");
}
```

В данном случае вывод в консоль выполнится только если a больше либо равно 100.Иначе условный оператор ничего не выведет. Фигурные скобки в данном примере можно опустить.

Обратите внимание, что для проверки на равенство используется оператор ==, а не =. Знак «равно» используется в C# для присваивания значений. Следует понимать, что условное выражение, стоящее в конструкции if должно возвращать булево значение.

Например, следующий условный оператор всегда будет выполняться:

```
if (true)
    Console.WriteLine("Всегда выводится");
else
    Console.WriteLine("Никогда не выводится");
```

2.2 Оператор выбора.

Синтаксис оператора:

```
switch  (перем)
{
    case  константа1:
        оператор _1;
        break;
    case  константа2 :
        оператор _ 2;
        break;
...
    default :
        оператор _ n;
        break;
}
```

В данном операторе осуществляется выбор оператора (который может быть составным) в зависимости от значения переменной *перем*. Если *перем* равна значению *константа*1, то выполнится *оператор* _1, если *константа*2 – *оператор* _ 2, и т.д. Если ни одно значение, указанное в каком-либо операторе case, не совпало со значением *перем*, то выполнится оператор, указанный в секции default . Внутри каждой секции case следует указать оператор break, который предотвращает проверку других условий после выполнения данного оператора. Следует обратить внимание, что в секциях case следует использовать только константы (переменные не допускаются).

Пример использование оператора switch … case:

```
// Организация ответа на действие пользователя:
string text = "1 - Вывод группы \n" +
    "2 - Вывод фамилии преподавателя \n" +
    "3 - Вывод названия предмета \n" +
    "4 - Вывод приветствия \n";

Console.Write(text + "Введите команду > ");
int result = Convert.ToInt32(Console.ReadLine());

switch (result)
{
    case 1: { Console.WriteLine("ИСТБ - 101"); break; }
    case 2: Console.WriteLine("Николаев Евгений Иванович"); break;
    case 3: { Console.WriteLine("Технологии программирования"); break; }
    case 4: Console.WriteLine("Привет !"); break;
    default: Console.WriteLine("Недопустимый вариант !!!"); break;
}
```

Изучите представленный пример самостоятельно.

4. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом, представленным в работе №1.

2. Выполните индивидуальное задание. Во всех заданиях переменные X, Y являются вещественными и вводятся пользователем. Количество слагаемых также вводится пользователем. Программа должна вывести сумму заданного числа членов последовательности.

Индивидуальное задание.

Перед выполнением задания требуется самостоятельно определить закономерность изменения членов последовательности, чтобы применить цикл, условный оператор или, если потребуется, оператор выбора.

Вариант	Выражение для вычисления
1	$j = -\frac{\sin^3(Y)}{1\cdot 3} + \frac{\sin^5(X^2)}{3\cdot 5} - \frac{\sin^7(Y^3)}{5\cdot 7} + \frac{\sin^9(X^4)}{7\cdot 9} - \ldots$
2	$Z = \frac{1}{1\cdot 3\cdot 5} - \frac{X}{2\cdot 4\cdot 6} + \frac{Y^2}{3\cdot 5\cdot 7} - \frac{X^3}{4\cdot 6\cdot 8} + \frac{Y^4}{5\cdot 7\cdot 9} - \ldots$
3	$Z = \frac{X^2}{1\cdot 3} - \frac{Y^4}{3\cdot 5} + \frac{X^6}{5\cdot 7} - \frac{Y^8}{7\cdot 9} + \ldots$
4	$Z = \frac{Y^2\cdot X}{2} - \frac{Y^2\cdot X^4}{4} + \frac{X^4\cdot Y^6}{6} - \frac{X^8\cdot Y^6}{8} + \ldots$
5	$A = -\frac{\sin(X)\cdot\lg(Y)}{1!} + \frac{\ln(X^3)\cdot\cos(Y^2)}{3!} - \frac{\sin(X^5)\cdot\lg(Y^3)}{5!} + \frac{\ln(X^7)\cdot\cos(Y^4)}{7!} - \ldots$
6	$Z = 1 - \frac{X}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - \ldots$ (в знаменателе факториал)
7	$s = -\frac{Y\cdot X^2}{1\cdot 3} + \frac{X\cdot Y^3}{2\cdot 4} - \frac{Y\cdot X^4}{3\cdot 5} + \frac{X\cdot Y^5}{4\cdot 6} - \ldots$
8	$Z = 1 - \frac{\sin(X^2)}{2} + \frac{\cos(Y^3)}{3} - \frac{\sin(X^4)}{4} + \frac{\cos(Y^5)}{5} - \ldots$
9	$Z = \frac{1}{1\cdot 3} - \frac{X}{2\cdot 4} + \frac{X^2}{3\cdot 5} - \frac{X^3}{4\cdot 6} + \ldots$
10	$Z = \frac{X^2}{1\cdot 3} - \frac{Y^4}{3\cdot 5} + \frac{X^6}{5\cdot 7} - \frac{Y^8}{7\cdot 9} + \ldots$

11	$p=-\frac{X\cdot\sin(Y)}{1\cdot 2}+\frac{X^3\cdot\cos(Y^2)}{3\cdot 4}-\frac{X^5\cdot\sin(Y^3)}{5\cdot 6}+\frac{X^7\cdot\cos(Y^4)}{7\cdot 8}-\ldots$
12	$p=-\frac{X}{1\cdot 2\cdot 3}+\frac{Y^3}{3\cdot 4\cdot 5}-\frac{X^5}{5\cdot 6\cdot 7}+\frac{Y^7}{7\cdot 8\cdot 9}-\ldots$
13	$Z=1-\frac{\sin(X^2)+Y}{2}+\frac{\cos(Y^3)+X}{3}-\frac{\sin(X^4)+Y}{4}+\frac{\cos(Y^5)+X}{5}-\ldots$
14	$Z=\frac{Y^2+X}{1\cdot 2}-\frac{Y^2-X^4}{2\cdot 4}+\frac{X^4+Y^6}{4\cdot 6}-\frac{X^8-Y^6}{6\cdot 8}+\ldots$
15	$p=1-\frac{\cos(X)\cdot\sin^2(Y)}{1\cdot 2}+\frac{\cos^4(X)\cdot\sin^3(Y)}{3\cdot 4}-\frac{\cos^5(X)\cdot\sin^6(Y)}{5\cdot 6}+\ldots$
16	$s=-\frac{\sin(Y)\cdot X^2}{1\cdot 3}+\frac{X\cdot\cos(Y^3)}{2\cdot 4}-\frac{\sin(Y)\cdot X^4}{3\cdot 5}+\frac{X\cdot\cos(Y^5)}{4\cdot 6}-\ldots$
17	$Z=-\frac{tg(Y)\cdot ctg(X^2)}{1\cdot 3}+\frac{tg(X)\cdot ctg(Y^3)}{2\cdot 4}-\frac{tg(Y)\cdot ctg(X^4)}{3\cdot 5}+\frac{tg(X)\cdot ctg(Y^5)}{4\cdot 6}-\ldots$
18	$Z=\frac{1}{1!}-\frac{X}{2!}+\frac{Y^2}{3!}-\frac{X^3}{4!}+\frac{Y^4}{5!}-\ldots$
19	$Z=\frac{\cos(X)+\lvert Y\rvert}{1!}-\frac{\cos^2(Y)+\lvert X\rvert}{2!}+\frac{\cos^3(X)+\lvert Y\rvert}{3!}-\frac{\cos^4(X)+\lvert Y\rvert}{4!}+\ldots$
20	$Z=\frac{Y^2\cdot X}{2+1}-\frac{Y^2\cdot X^4}{4-2}+\frac{X^4\cdot Y^6}{6+4}-\frac{X^8\cdot Y^6}{8-6}+\ldots$
21	$Z=1+\frac{1-\cos(Y)}{1\cdot 3\cdot 5}-\frac{Y+\cos^2(X)}{2\cdot 4\cdot 6}+\frac{X^2-\cos^3(Y)}{3\cdot 5\cdot 7}-\frac{Y^3+\cos^4(X)}{4\cdot 6\cdot 8}+\ldots$
22	$Z=\frac{X-Y}{1\cdot 3}-\frac{Y^2-X^2}{2\cdot 4}+\frac{X^3-Y^3}{3\cdot 5}-\frac{Y^4-X^4}{4\cdot 6}+\ldots$
23	$Z=1+\frac{1-\cos(Y)}{1\cdot 3\cdot 5}-\frac{Y+\cos^2(X)}{2\cdot 4\cdot 6}+\frac{X^2-\cos^3(Y)}{3\cdot 5\cdot 7}-\frac{Y^3+\cos^4(X)}{4\cdot 6\cdot 8}+\ldots$
24	$Z=\frac{(X-Y)^2}{1\cdot 3}-\frac{(Y-X)^4}{3\cdot 5}+\frac{(X-Y)^6}{5\cdot 7}-\frac{(Y-X)^8}{7\cdot 9}+\ldots$
25	$Z=\frac{Y^2\cdot\sin(X)}{2}-\frac{\cos(Y^2)\cdot X^4}{4}+\frac{\sin(X^4)\cdot Y^6}{6}-\frac{\cos(Y^6)\cdot X^8}{8}+\ldots$

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Как оформляется комментарий?
2. Опишите синтаксис условного оператора. Какие части данного оператора являются обязательными?
3. Опишите синтаксис оператора выбора. Поясните почему следует использовать оператор break внутри каждого выбора.
4. Можно ли с помощью for реализовать бесконечный цикл? Поясните ответ на примерах.

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [2-4].

ПРАКТИЧЕСКАЯ РАБОТА 5. УПРАВЛЕНИЕ ПОТОКОМ ВЫПОЛНЕНИЯ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРА ЦИКЛА FOR

1. Цель и содержание

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

Задачи работы:

– научиться применять оператор цикла for во вложенных циклах;

2. Теоретическая часть

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 (пример 1):

```
/*
* Вычислить сумму чисел от 1 до 1000
*/

// Объявляем переменную, которая хранит значение суммы чисел
System.Int32 Sum = 0;
// Объявляем счетчик цикла
int i;

for (i = 1; i <= 1000; i++)
    Sum += i;

Console.WriteLine("Итог: {0}", Sum);
```

Изучите код самостоятельно (создайте такую программу в VS). Еще один пример для самостоятельного изучения (пример 2):

```
/*
* вычислить сумму всех чисел, делящихся на 3 без остатка
* и находящихся в диапазоне от 1 до 1000000
*/

long Sum = 0;
int i;

for (i = 1; i <= 1000000; i++)
    if (i % 3 == 0)
        Sum += i;
```

Пример, отображающий возможность применения итератора (пример 3):

```
/*
* вычислить сумму всех чисел, делящихся на 3 без остатка
* и находящихся в диапазоне от 1 до 1000000
*/

long Sum = 0;
int i;

for (i = 0; i <= 1000000; i += 3)
        Sum += i;
```

Обратите внимание, что примеры 2 и 3 решают одну и ту же задачу.

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб иывыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

Необходимо модифицировать приложение, разработанное в рамках работы №4 следцющим образом:

1. Программа должна запросить исходные данные для вычисления выражения.

2. Программа вычисляет выражение и выводит результат.

3. Затем программа снова запрашивает ввод исходных данных и цикл повторяется.

С использованием цикла for организуйте указанное поведение программы и самостоятельно определите команду, которая приведет к завершению работы программы (например, нажата клавиша «Escape», пользователь должен ответить на вопрос «Продолжить?» и т.п.).

Индивидуальное задание.

В качестве индивидуального задания необходимо изменить код из работы №4.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Как оформляется комментарий?

2. В разделе «Теоретическое обоснование» приведены примеры 1, 2 и 3. Почему в примерах 2 и 3 для переменной Sum выбран тип long, хотя в примере 1 для Sum выбран тип int? Обоснуйте ответ.

3. Опешите синтаксис оператора цикла for.

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

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [3], [7].

ПРАКТИЧЕСКАЯ РАБОТА 6. УПРАВЛЕНИЕ ПОТОКОМ ВЫПОЛНЕНИЯ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРОВ WHILE

1. Цель и содержание

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

Задачи работы:

– научиться применять оператор цикла с предусловием while;

– научиться применять оператор цикла с постусловием do … while.

2. Теоретическая часть

Цикл while похож на for тем, что является конструкцией с предварительной проверкой условия продолжения цикла. Но синтаксис цикла while более лаконичен:

while (*условие*)
оператор

В данном синтаксисе *оператор* может быть составным оператором (группа операторов языка, заключенные в фигурные скобки).

Следует понимать, что while используется для заранее неизвестного количества повторных выполнений операторов.

Пример выполнения цикла (пример 1):

```
/*
 * Цикл будет выполняться пока пользователь не введет 0
 */

bool Stop = false;
int chislo;

while (!Stop)
{
    Console.Write("Введите число > ");
    chislo = Convert.ToInt32(Console.ReadLine());

    if (chislo == 0)
        Stop = true;
}
```

Оператор break уже встречался в конструкции case оператора switch для выхода из case. Но break часто применяется внутри циклов for, while и do … while. Данный оператор прерывает выполнение цикла и передает управление оператору, следующему за циклом.

Пример, выполняющий то же самое, что и пример 1 (пример 2).

```
/*
 * Цикл будет выполняться пока пользователь не введет 0
 */

bool Stop = true;
int chislo;

while(Stop)
{
    Console.Write("Введите число > ");
    chislo = Convert.ToInt32(Console.ReadLine());

    if (chislo == 0)
        break;
}
```

Следует обратить внимание, что цикл является бесконечным (логическая переменная Stop всегда является true), но выполнение цикла все равно прервется, если пользователь введет 0.

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

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом, представленным в работе №1.
2. Выполните индивидуальное задание. Во всех заданиях переменныеX, Y являются вещественными и вводятся пользователем. Количество

слагаемых пользователем не вводится. Программа должна работать следующим образом:

– пользователю выводится приглашение на ввод X и Y;

– пользователь вводит X и Y;

– программа начинает расчет суммы, при этом выводится результат расчета, полученный на каждой итерации;

– сначала выводится номер итерации (1) и сумма 1-го слагаемого, затем программа останавливается и ждет ввода команды пользователя (1 – продолжить, 0 – прекратить расчет);

– если пользователь продолжает, выводится номер итерации (2) и сумма 2-х слагаемых и снова программа ждет команды пользователя, и т.д.

3. Программа не должна использовать цикл for.

Индивидуальное задание.

Перед выполнением задания требуется самостоятельно определить закономерность изменения членов последовательности, чтобы применить цикл, условный оператор или, если потребуется, оператор выбора.

Вариант	Выражение для вычисления
1	$Z = 1 - \frac{X}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - \ldots$ (в знаменателе факториал)
2	$s = -\frac{Y \cdot X^2}{1 \cdot 3} + \frac{X \cdot Y^3}{2 \cdot 4} - \frac{Y \cdot X^4}{3 \cdot 5} + \frac{X \cdot Y^5}{4 \cdot 6} - \ldots$
3	$Z = 1 - \frac{\sin(X^2)}{2} + \frac{\cos(Y^3)}{3} - \frac{\sin(X^4)}{4} + \frac{\cos(Y^5)}{5} - \ldots$
4	$Z = \frac{1}{1 \cdot 3} - \frac{X}{2 \cdot 4} + \frac{X^2}{3 \cdot 5} - \frac{X^3}{4 \cdot 6} + \ldots$
5	$Z = \frac{X^2}{1 \cdot 3} - \frac{Y^4}{3 \cdot 5} + \frac{X^6}{5 \cdot 7} - \frac{Y^8}{7 \cdot 9} + \ldots$
6	$p = -\frac{X \cdot \sin(Y)}{1 \cdot 2} + \frac{X^3 \cdot \cos(Y^2)}{3 \cdot 4} - \frac{X^5 \cdot \sin(Y^3)}{5 \cdot 6} + \frac{X^7 \cdot \cos(Y^4)}{7 \cdot 8} - \ldots$
7	$p = -\frac{X}{1 \cdot 2 \cdot 3} + \frac{Y^3}{3 \cdot 4 \cdot 5} - \frac{X^5}{5 \cdot 6 \cdot 7} + \frac{Y^7}{7 \cdot 8 \cdot 9} - \ldots$

8	$Z = 1 - \frac{\sin(X^2)+Y}{2} + \frac{\cos(Y^3)+X}{3} - \frac{\sin(X^4)+Y}{4} + \frac{\cos(Y^5)+X}{5} - \ldots$
9	$Z = \frac{Y^2+X}{1\cdot 2} - \frac{Y^2-X^4}{2\cdot 4} + \frac{X^4+Y^6}{4\cdot 6} - \frac{X^8-Y^6}{6\cdot 8} + \ldots$
10	$j = -\frac{\sin^3(Y)}{1\cdot 3} + \frac{\sin^5(X^2)}{3\cdot 5} - \frac{\sin^7(Y^3)}{5\cdot 7} + \frac{\sin^9(X^4)}{7\cdot 9} - \ldots$
11	$Z = \frac{1}{1\cdot 3\cdot 5} - \frac{X}{2\cdot 4\cdot 6} + \frac{Y^2}{3\cdot 5\cdot 7} - \frac{X^3}{4\cdot 6\cdot 8} + \frac{Y^4}{5\cdot 7\cdot 9} - \ldots$
12	$Z = \frac{X^2}{1\cdot 3} - \frac{Y^4}{3\cdot 5} + \frac{X^6}{5\cdot 7} - \frac{Y^8}{7\cdot 9} + \ldots$
13	$Z = \frac{Y^2\cdot X}{2} - \frac{Y^2\cdot X^4}{4} + \frac{X^4\cdot Y^6}{6} - \frac{X^8\cdot Y^6}{8} + \ldots$
14	$A = -\frac{\sin(X)\cdot\lg(Y)}{1!} + \frac{\ln(X^3)\cdot\cos(Y^2)}{3!} - \frac{\sin(X^5)\cdot\lg(Y^3)}{5!} + \frac{\ln(X^7)\cdot\cos(Y^4)}{7!} - \ldots$
15	$Z = \frac{\cos(X)+\lvert Y\rvert}{1!} - \frac{\cos^2(Y)+\lvert X\rvert}{2!} + \frac{\cos^3(X)+\lvert Y\rvert}{3!} - \frac{\cos^4(X)+\lvert Y\rvert}{4!} + \ldots$
16	$Z = 1 + \frac{1-\cos(Y)}{1\cdot 3\cdot 5} - \frac{Y+\cos^2(X)}{2\cdot 4\cdot 6} + \frac{X^2-\cos^3(Y)}{3\cdot 5\cdot 7} - \frac{Y^3+\cos^4(X)}{4\cdot 6\cdot 8} + \ldots$
17	$Z = \frac{X-Y}{1\cdot 3} - \frac{Y^2-X^2}{2\cdot 4} + \frac{X^3-Y^3}{3\cdot 5} - \frac{Y^4-X^4}{4\cdot 6} + \ldots$
18	$Z = 1 + \frac{1-\cos(Y)}{1\cdot 3\cdot 5} - \frac{Y+\cos^2(X)}{2\cdot 4\cdot 6} + \frac{X^2-\cos^3(Y)}{3\cdot 5\cdot 7} - \frac{Y^3+\cos^4(X)}{4\cdot 6\cdot 8} + \ldots$
19	$Z = \frac{(X-Y)^2}{1\cdot 3} - \frac{(Y-X)^4}{3\cdot 5} + \frac{(X-Y)^6}{5\cdot 7} - \frac{(Y-X)^8}{7\cdot 9} + \ldots$
20	$Z = \frac{Y^2\cdot\sin(X)}{2} - \frac{\cos(Y^2)\cdot X^4}{4} + \frac{\sin(X^4)\cdot Y^6}{6} - \frac{\cos(Y^6)\cdot X^8}{8} + \ldots$
21	$p = 1 - \frac{\cos(X)\cdot\sin^2(Y)}{1\cdot 2} + \frac{\cos^4(X)\cdot\sin^3(Y)}{3\cdot 4} - \frac{\cos^5(X)\cdot\sin^6(Y)}{5\cdot 6} + \ldots$
22	$s = -\frac{\sin(Y)\cdot X^2}{1\cdot 3} + \frac{X\cdot\cos(Y^3)}{2\cdot 4} - \frac{\sin(Y)\cdot X^4}{3\cdot 5} + \frac{X\cdot\cos(Y^5)}{4\cdot 6} - \ldots$
23	$Z = -\frac{tg(Y)\cdot ctg(X^2)}{1\cdot 3} + \frac{tg(X)\cdot ctg(Y^3)}{2\cdot 4} - \frac{tg(Y)\cdot ctg(X^4)}{3\cdot 5} + \frac{tg(X)\cdot ctg(Y^5)}{4\cdot 6} - \ldots$
24	$Z = \frac{1}{1!} - \frac{X}{2!} + \frac{Y^2}{3!} - \frac{X^3}{4!} + \frac{Y^4}{5!} - \ldots$
25	$Z = \frac{Y^2\cdot X}{2+1} - \frac{Y^2\cdot X^4}{4-2} + \frac{X^4\cdot Y^6}{6+4} - \frac{X^8\cdot Y^6}{8-6} + \ldots$

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Какой цикл лучше while или for?

2. Опишите синтаксис оператора while.

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

4. Опишите синтаксис оператора выбора. Поясните почему следует использовать оператор break внутри каждого выбора.

5. Поясните назначение операторов break и continue.

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [3], [7].

ПРАКТИЧЕСКАЯ РАБОТА 7. УПРАВЛЕНИЕ ПОТОКОМ ВЫПОЛНЕНИЯ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРА DO …WHILE

1. Цель и содержание

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

Задачи работы:

– научиться применять оператор цикла с постусловием do … while.

– изучить особенности оператора do…while;

– научиться использовать оператор цикла do…while для решения практических задач.

2. Теоретическая часть

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

Синтаксис оператора:

```
do
{
        оператор (операторы)
}
while  (условие);
```

Пример использования цикла (пример 1):

```
/*
 * Цикл будет выполняться пока пользователь не введет 0
 */

bool Stop = false;
int chislo;

do
{
    Console.Write("Введите число > ");
    chislo = Convert.ToInt32(Console.ReadLine());

    if (chislo == 0)
        Stop = true;
} while (!Stop);
```

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального

компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом, представленным в работе №1.

2. Выполните индивидуальное задание. Во всех заданиях переменные X, Y являются вещественными и вводятся пользователем. Количество слагаемых пользователем не вводится. Программа должна работать следующим образом:

– пользователю выводится приглашение на ввод X и Y;

– пользователь вводит X и Y;

– программа начинает расчет суммы, при этом выводится результат расчета, полученный на каждой итерации;

– сначала выводится номер итерации (1) и сумма 1-го слагаемого, затем программа останавливается и ждет ввода команды пользователя (1 – продолжить, 0 – прекратить расчет);

– если пользователь продолжает, выводится номер итерации (2) и сумма 2-х слагаемых и снова программа ждет команды пользователя, и т.д.

3. Программа не должна использовать цикл for или while.

Индивидуальное задание.

Необходимо использовать индивидуальное задание из работы №6.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Какой цикл лучше do…while или while?

2. Опишите синтаксис оператора do…while.

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

4. Опишите синтаксис оператора выбора. Поясните почему следует использовать оператор break внутри каждого выбора.

5. Опешите синтаксис оператора do … while.

6. Поясните назначение операторов break и continue.

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [6-8].

ПРАКТИЧЕСКАЯ РАБОТА 8. КЛАССЫ. СТРУКТУРЫ.

1. Цель и содержание

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

Задачи работы:

– научиться объявлять классы;

– научиться создавать объекты классов;

– научиться работать с полями данных и методами классов.

2. Теоретическая часть

2.1 Классы и структуры.

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.

Пример объявления 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;
}
```

При создании как классов, так и структур, используется ключевое слово new, например:

```
MyFirstClass obj = new MyFirstClass();
```

2.2 Структура класса.

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.

2.2.1 Данные-члены.

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: ИмяПоля ИмяОбъекта.

```
MyFirstClass obj = new MyFirstClass();

obj.a = 5;
int cc = obj.a;

obj.fio = "Novak E.I.";
```

Аналогичным образом с классом ассоциируются константы.

События будут рассмотрены в следующих лабораторных работах.

2.2.2 Функции-члены.

Функции-члены – это члены, которые обеспечивают некоторую функциональность для манипулирования данными классов. Они делятся на следующие 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 классов, 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.

В данной работе рассматриваются только методы класса – это функции, ассоциированные с определенным классом.

В C# объявление метода класса состоит из спецификатора доступности, возвращаемого значения, имени метода, списка формальных параметров и тела метода:

```
[модификатор] тип_возврата имя_метода ([список_параметров])
{
    // тело метода
}
```

Например, добавим методы для объявленного ранее класса MyFirstClass:

```
// 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);
    }
}
```

Синтаксис вызова методов аналогичен синтаксису обращения к данным-членам:

```
MyFirstClass obj = new MyFirstClass();

obj.InitClassMembers(10, 0.8F, "Новиков П.Е.");

int abs_a = obj.GetAbsA();
```

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 из-за 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, которая может даже не обращать внимания на то, что в них хранилось первоначально.

Было бы удобно использовать то же соглашение в C#. Однако 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.

Частичные классы.

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.

Статические классы.

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.

Класс Object.

Rлассы .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.

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое

устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом, представленным в работе №1.

2. Изучите пример выполнения задания, представленный в данном разделе.

3. Выполните индивидуальное задание. Задания ориентированы на работу с классами.

Пример выполнения задания.

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 двух этапов: объявление класса Parallelepiped и демонстрация использования объекта данного класса.

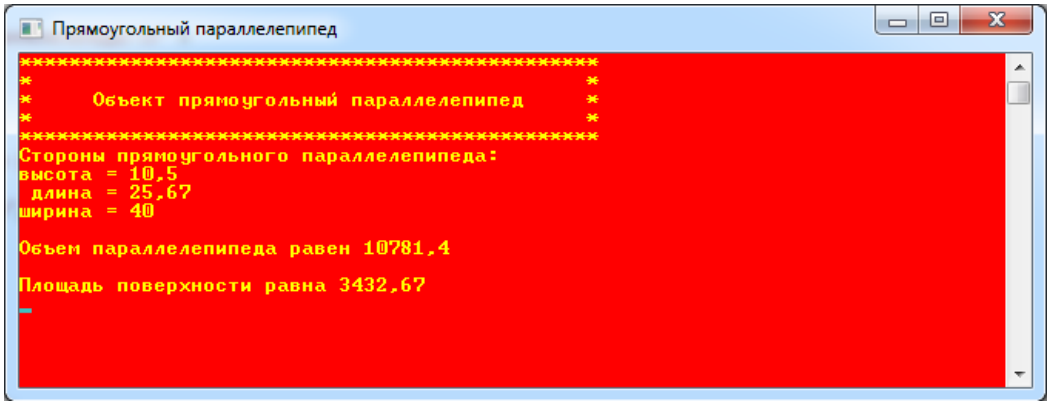
Полный листинг примера:

```
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) { a = pa; }
        public double Get_a() { return a; }

        public void Set_b(double pb) { b = pb; }
        public double Get_b() { return b; }

        public void Set_c(double pc) { c = pc; }
        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 объектк в консоль
        public void PrintFullInformation()
        {
            string str =    "*********************************************\n" +
                            "*                                           *\n" +
                            "*     Объект прямоугольный параллелепипед   *\n" +
                            "*                                           *\n" +
                            "*********************************************";
            Console.WriteLine(str);

            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.Title = "Прямоугольный параллелепипед";
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.BackgroundColor = ConsoleColor.Red;
            Console.Clear();

            Parallelepiped p;

            p = new Parallelepiped();
            p.Set_a(10.5);
            p.Set_b(25.67);
            p.Set_c(40);

            p.PrintFullInformation();

            Console.ReadKey();
        }
    }


}
```

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

В результате выполнения программы отобразится следующее консольное окно с выводом информации:



Индивидуальное задание.

Спроектируйте класс, наполните его требуемой функциональностью, продемонстрируйте работоспособность класса.

Вариант	Выражение для вычисления
1.	Класс «Шар». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
2.	Класс «Куб». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, длины диагонали, а также вывод информации об объекте.
3.	Класс «Сфера». 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 «График y=x». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
6.	Класс «Шар». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
7.	Класс «Матрица MxN». Реализовать инициализацию элементов матрицы случайными числами, вывод матрицы, нахождение максимального и минимального элементов, а также вывод информации об объекте.
8.	Класс «Прямоугольный треугольник». Реализовать ввод и вывод полей данных, вычисление гипотенузы, площади и периметра, а также вывод информации об объекте.

9.	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.
10.	Класс «Цилиндр». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
11.	Класс «Ромб». Реализовать ввод и вывод полей данных (диагонали ромба), вычисление площади, периметра, а также вывод информации об объекте.
12.	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.
13.	Класс «График y=3x+5». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
14.	Класс «Матрица MxN». Реализовать инициализацию элементов матрицы случайными числами, вывод транспонированной матрицы, нахождение среднего арифметического всех элементов, а также вывод информации об объекте.
15.	Класс «Отрезок в пространстве». 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.
16.	Класс «График y=x-10». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
17.	Класс «Матрица MxN». Реализовать инициализацию элементов матрицы случайными числами, вывод транспонированной матрицы, нахождение и вывод среднего арифметического элементов в каждом столбце.
18.	Класс «Куб». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, длины диагонали, а также вывод информации об объекте.
19.	Класс «Сфера». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
20.	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.

21.	Класс «График y=x». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
22.	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.
23.	Класс «График y=-x». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
24.	Класс «Цилиндр». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
25.	Класс «Отрезок в пространстве». 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.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

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 типы членов-данных вы знаете?

5. Какие типы функций-членов класса вы знаете?

6. Как поменять цвет текста в консольном приложении?

7. Как поменять цвет фона в консольном приложении?

8. Какие модификаторы доступности членов класса вы знаете?

9. Какое ключевое слово используется для создания объекта класса?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [8].

ПРАКТИЧЕСКАЯ РАБОТА 9. КОНСТРУКТОР КЛАССА. ПЕРЕГРУЗКА КОНСТРУКТОРОВ КЛАССА.

1. Цель и содержание

Цель работы: понять принципы работы конструктора.Задачи работы:

– научиться объявлять конструктор класса;

– научиться создавать перегруженные конструкторы.

2. Теоретическая часть

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.

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Для выполнения работы необходимо модифицировать приложение, полученное в результате выполнения индивидуального задания работы №13.

2. Модификация сводится к следующему: необходимо объявить и продемострировать использование 3-4 перегруженных конструкторов класса.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое класс?

2. Что такое конструктор класса?

3. Что такое перегрузка методов?

4. Может ли один конструктор класса вызывать другой конструктор? Прежде чем отвечать попробуйте реализовать такой вызов в своем разработанном классе.

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [7-8].

ПРАКТИЧЕСКАЯ РАБОТА 10. МНОГОМОДУЛЬНЫЕ ПРИЛОЖЕНИЯ

1. Цель и содержание

Цель работы: научиться проектировать консольные приложения на основе нескольких сборок.

Задачи работы:

– научиться управлять несколькими разрабатываемыми проектами в одном решении;

– научиться организовывать взаимодействие нескольких модулей приложения.

2. Теоретическая часть

В терминологии технологической платформы .NET не используется понятие «исполняемый файл». Подобные файлы (*.dll или *.exe) характерны для компиляторов небезопасного кода.

В технологии .NET используется термин «сборка». Сборка – это модуль, в котором хранится компилированный управляемый код. Она похожа на классический исполняемый файл (*.exe) или dll-библиотеку, но имеет важное

свойство – она полностью себя описывает. Сборки содержат метаданные, которые включают в себя сведения о сборке и обо всех определенных внутри нее типах, методах и т.д. Сборка может быть частной (доступной только одному приложению) или разделяемой (доступной любому приложению Windows).

В предыдущих лабораторных работах студентам предлагалось спроектировать консольные приложения, которые представляют собой одномодульные приложения – exe-файлы.

В данной работе требуется спроектировать приложение, которое состоит из нескольких сборок (один модуль – файл *.exe, остальные – dll).

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место

пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

Для выполнения работы спроектируем следующее приложение:

Требуется разработать приложение-опрос. Пользователю последовательно задаются вопросы, за каждый вариант ответа начисляются определенные баллы. После завершения опросы выводится сумма набранных баллов и какое-либо резюме. Такая программа может использоваться для оценки остаточных знаний в форме тестирования, для составления опросов, для самообследования и т.п.

Перед началом создания приложения, то есть перед непосредственным программированием, необходимо определиться: какие объекты и явления предметной области и связи между ними должен выявить программист. Очевидно, что это:

– вопрос;

– ответ (несколько для каждого вопроса);

– балл соответствующий конкретному ответу;

– итоговые резюме или оценки, которые выводятся в зависимости от количества набранных баллов.

Второе, что необходимо сделать осуществить физическую декомпозицию данного приложения, то есть сколько отдельных модулей будет содержать приложение. В данном случае:

– 1-ый модуль в виде файла *.exe для запуска приложения;

– 2-ой модуль в виде файла *.dll, содержащий вопросы и ответы;

– 3-ий модуль в виде файла *.dll, содержащий сообщения о результатах тестирования (опроса).

Декомпозиция, логическая и физическая, завершена. Для реализации прложения необходимо выполнить следующие шаги:

1. Создайте консольное приложение в среде MS VS (проект назовите Quiz). После выполнения всех необходимых действий окно Solution Explorer примет следующий вид:

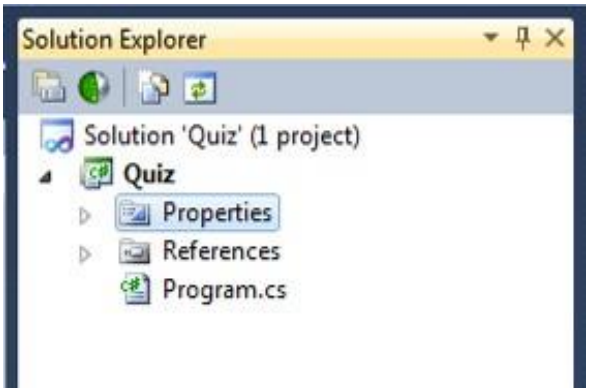


Рисунок 15.1 – Структура одномодульного приложения.

В окне отображено одно решение (Solution) с именем «Quiz» и один проект в рамках решения (с именем «Quiz»). Проект содержит файл кода Program.cs, в котором определена функция Main. Этот проект будет откомпилирован в файл Quiz.exe (то есть файл для запуска).

2. Создадим еще один файл в рамках проекта Quiz, который будет содержать класс Testing, позволяющий реализовывать логику тестирования, то есть вывод вопросов в определенной последовательности, ввод выбора пользователя и т.д. Для этого вызовем контекстное меню проекта (рис. 13.2) и создадим новый файл с именем Testing.cs

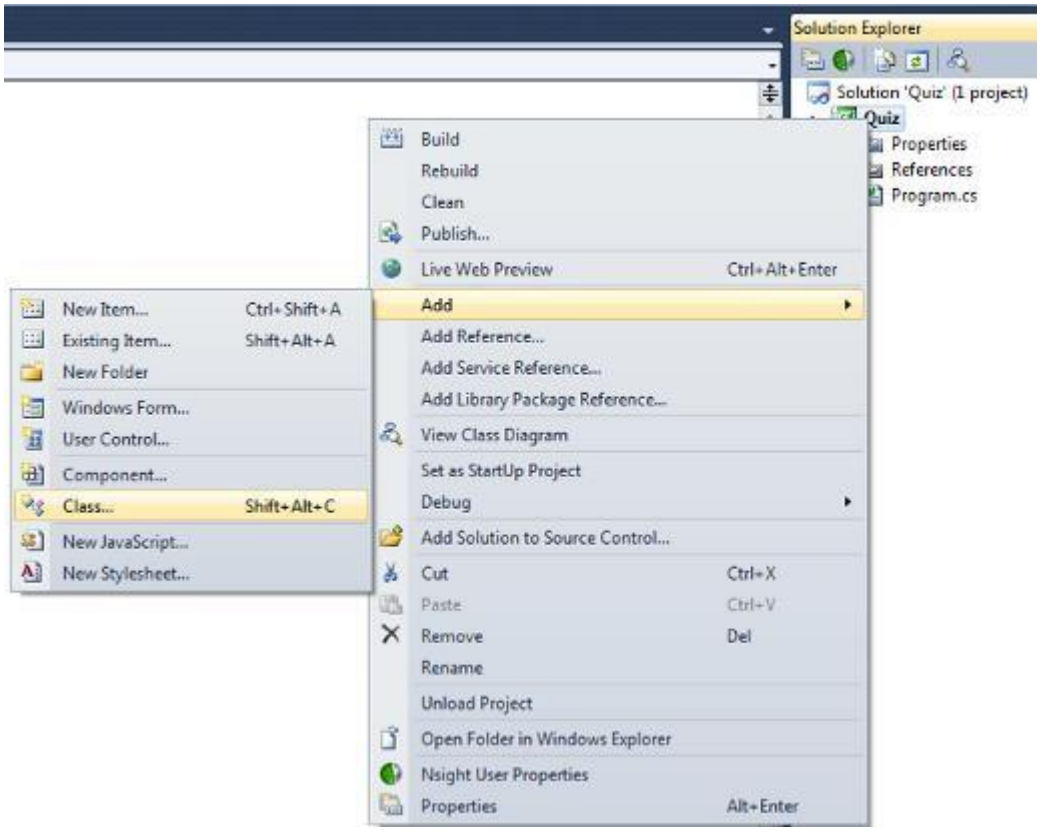


Рисунок 15.2 – Добавления нового класса к проекту.

После выполнения данного действия в окне Solution Explorer состав проекта изменится (рис. 15.3).

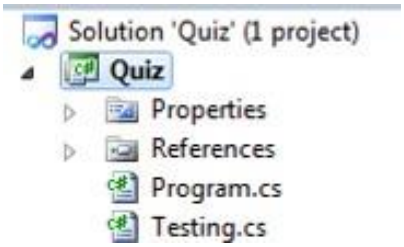


Рисунок 15.3 – Проект с несколькими файлами исходного кода.

На данном этапе класс Testing не содержит никакой логики.

3. Создадим второй модуль в виде dll-библиотеки, которая будет содержать вопросы и ответы. Для этого вызовем команду контекстного меню решения Quiz (рис. 15.4).

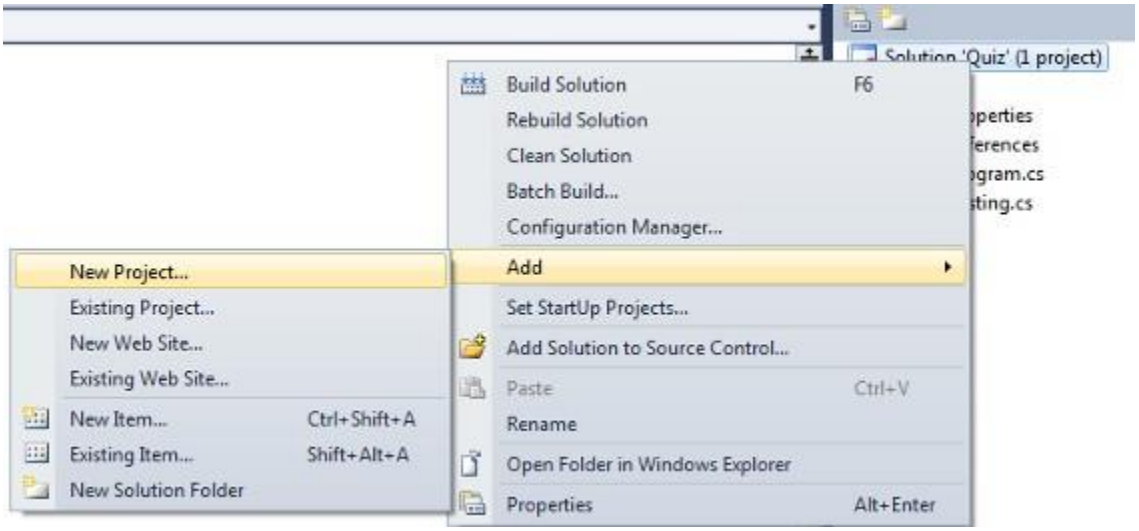


Рисунок 15.4 – Добавление проекта к решению VS.

4. В появившемся диалоговом окне выберем тип проекта «Class Library», в качестве имени проекта укажем «Task» (рис. 15.5)

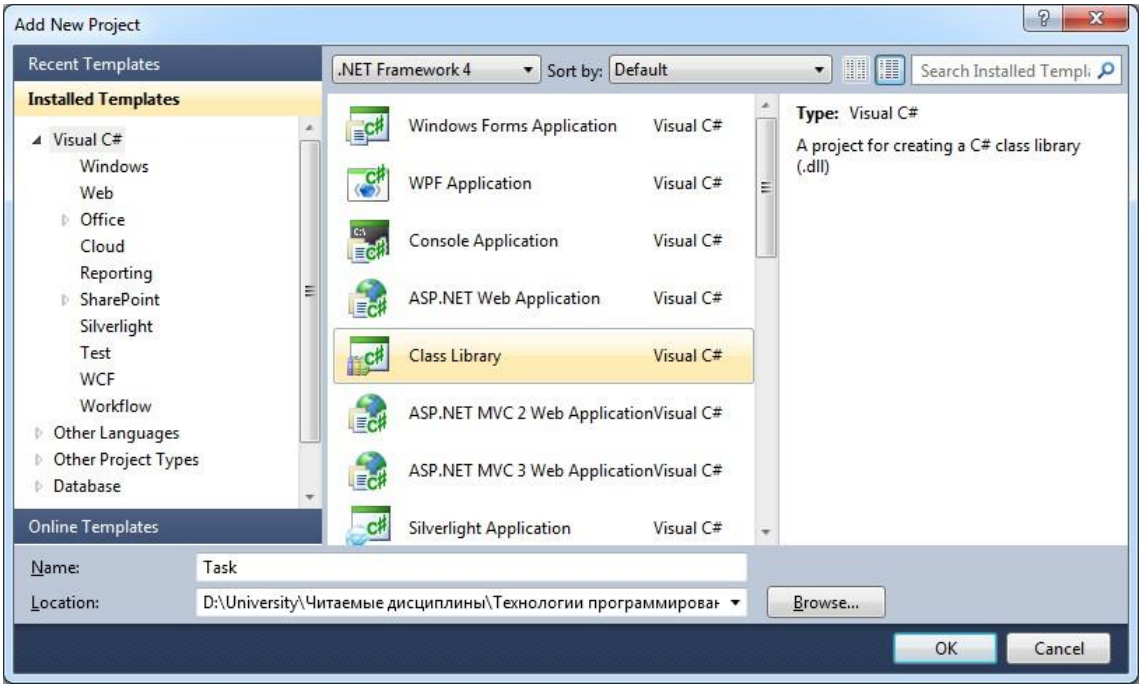


Рисунок 15.5 –Добавление библиотеки классов к решению Quiz.

После добавления нового проекта окно Solution Explorer примет вид, показанный на рис. 15.6

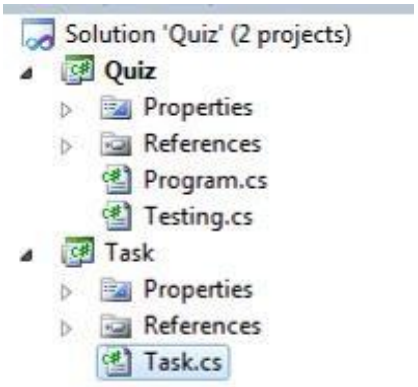


Рисунок 15.6 –Решение с несколькими проектами.

Два проекта в одном решении Quiz, при этом проект Quiz помечен полужирным шрифтом. Это означает, что при вызове команды Run будет запущен именно этот проект. Следует обратить внимание, что файл Task.cs проекта Task содержит только пустой класс без кода. Проект Task в целом не содержит функции Main, то есть не может быть запушен на выполнение.

5. Аналогичным образом добавим к решению еще один проект с именем «Result» и типом «Class Library». Окно Solution Explorer примет вид:

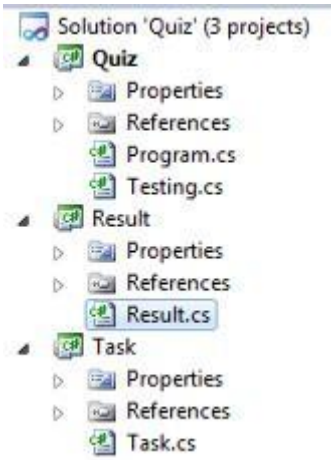


Рисунок 15.7 – Все требуемые проекты, в соответствие с физической декомпозицией, добавлены к решению.

Далее переходим к наполнению классов требуемым функционвалом.

6. Модифицируем файл Task.cs проекта Task следующим образом:

```
namespace Task
{
    // Класс для представления одного тестового задания
    public class SingleTest
    {
        public string Question {get; set;} // вопрос
        public List<string> Answers; // Ответы
        public List<int> Balls; // Баллы за ответы
    }

    // Статический класс для представления списка тестовых заданий
    public static class Task
    {
        // Список отдельных вопросов с ответами
        public static List<SingleTest> AllQuestions;

        // Статический конструктор для инициализации коллекции
        static Task()
        {
            AllQuestions = new List<SingleTest>()
            {
                new SingleTest()
                {
                    Question = "Main - точка входа программы ...",
                    Answers = new List<string>() { "1", "7", "Не знаю" },
                    Balls = new List<int>() { 20, 5, 0 }
                },
                new SingleTest()
                {
                    Question = "Какой оператор не является оператором цикла?",
                    Answers = new List<string>() { "while", "for", "if" },
                    Balls = new List<int>() { -5, 0, 10 }
                },
                new SingleTest()
                {
                    Question = "Как обозначить составной оператор?",
                    Answers = new List<string>() { "{ ... }", "( ... )", "< ... >" },
                    Balls = new List<int>() { 50, 0, 0 }
                },
                new SingleTest()
                {
                    Question = "Что обозначает символ ; ",
                    Answers = new List<string>() { "Пустой оператор", "Конец программы", "Не знаю" },
                    Balls = new List<int>() { 50, 20, 0 }
                }
            };
        }
    }
}
```

Рисунок 15.8 – Классы проекта Task.

Следует обратить внимание на то, что класс Task статический, то есть не требует создания объектов.

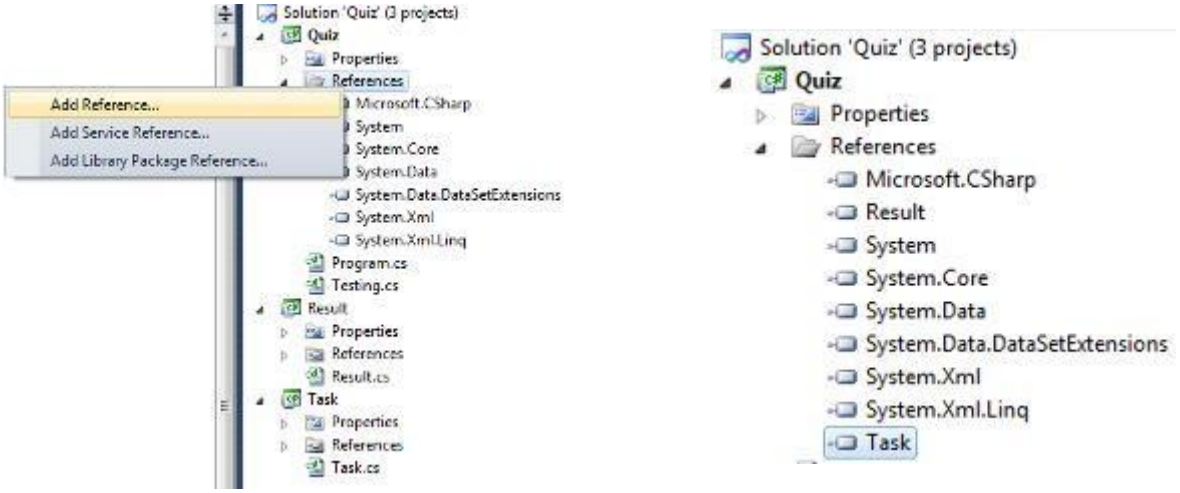
7. Модифицируйте файл Result.cs проекта Result следующим образом:

```
namespace Result
{
    public static class Result
    {
        public static List<string> Messages;

        // Статический конструктор
        static Result()
        {
            Messages = new List<string>()
            {
                "Вы набрали менее 10 баллов! Плохо!",
                "Вы набрали не менее 10 и менее 40 баллов! Нормально!",
                "Вы набрали не менее 40 баллов! Отлично"
            };
        }

        // Метод для возвращения сообщения
        public static string GetMessage(int Ball)
        {
            string mes = "";

            if (Ball < 10)
                mes = Messages[0];
            else if (Ball < 40)
                mes = Messages[1];
            else
                mes = Messages[2];

            return mes;
        }
    }
}
```

Рисунок 15.9 – Класс проекта Result.

Создан один статический файл, для представления результирующих сообщений.

8. Теперь модифицируем класс Tasting проекта Quiz. В своей работе класс будет использовать ранее созданные классы Task и Result, которые находятся в других проектах. Поэтому необходимо установит ссылки на данные проекты. Для этого вызовите команду (рис. 15.10а) контекстного меню папки References проекта Quiz. После этого ссылки на сборки (откомпилированные проекты Task и Result) появятся в списке ссылок Reference (рис. 15.10б).



а *б*

Рисунок 15.10 – Добавление ссылки на проекты: а – команда контекстного меню; б – список ссылок на проекты.

9. Модифицируем файл Testing.cs для реализации логики тестирования (рис. 15.11).

```
class Testing
{
    public string DoTesting()
    {
        int SumBal = 0;
        int ans = 0;

        foreach (Task.SingleTest p in Task.Task.AllQuestions)
        {
            Console.WriteLine("//////////////////////////");
            Console.WriteLine(p.Question);
            Console.WriteLine("//////////////////////////");
            Console.WriteLine("1. "+p.Answers[0]);
            Console.WriteLine("2. " + p.Answers[1]);
            Console.WriteLine("3. " + p.Answers[2]);
            Console.Write("Выберите номер ответа > ");
            ans = Convert.ToInt32(Console.ReadLine());
            SumBal += p.Balls[ans-1];
        }

        return Result.Result.GetMessage(SumBal);
    }
}
```

Рисунок 15.11 – Класс Testing.

10. Модифицируем функцию Main:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Quiz
{
    class Program
    {
        static void Main(string[] args)
        {
            Testing newTest = new Testing();

            Console.WriteLine("\n\n"+newTest.DoTesting());

            Console.ReadKey();
        }
    }
}
```

Рисунок 15.12 – Функция Main.

Запустите полученное приложение. Внимательно проанализируйте код приложения. Попробуйте изменить вопросы и их количество.

Затем, выполните индивидуальное задание.

Индивидуальное задание.

Спроектируйте многомодульное приложение (3 модуля), реализующее поставленную задачу. Перед выполнением приложения необходимо выполнить декомпозицию задачи и выявить основные объекты предметной области.

Вариант	Структура данных
1, 6, 11, 16, 21	Приложение осуществляет сложение обыкновенных дробей, то есть чисел $\frac{1}{3}$, $\frac{7}{12}$ и т.д. Требуется реализовать только сложение. Исходные слагаемые пользователь вводит с клавиатуры.
2, 7, 12, 17, 22	Приложение осуществляет умножение обыкновенных дробей, то есть чисел $\frac{1}{3}$, $\frac{7}{12}$ и т.д. Требуется реализовать только умножение. Исходные множители пользователь вводит с клавиатуры.

Вариант	Структура данных
3, 8, 13, 18, 23	Приложение осуществляет деление обыкновенных дробей, то есть чисел $\frac{1}{3}$, $\frac{7}{12}$ и т.д. Требуется реализовать только деление. Исходные дроби пользователь вводит с клавиатуры.
4, 9, 14, 19, 24	Реализуйте приложение, осуществляющее сокращение обыкновенной дроби. Дробь пользователь вводит с клавиатуры.
5, 10, 15, 20, 25	Реализуйте программу для выполнения операций над комплексными числами: сложение, умножение, вычисление модуля комплексного числа. Исходные комплексные числа пользователь вводит с клавиатуры.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое статический класс?

2. Что такое сборка?

3. Как определить проект по умолчанию в многомодульном решении?

4. Какими способами можно разместить в файлах определение класса?

5. Какой проект начнет выполняться первым если несколько из них в одном решении содержат функцию Main?

6. Что необходимо сделать для того, чтобы появилась возможность использовать классы, определенные в другом проекте?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [1-3].

ПРАКТИЧЕСКАЯ РАБОТА 11. ОПЕРАЦИИ КЛАССОВ. ПЕРЕГРУЗКА ОПЕРАЦИЙ.

1. Цель и содержание

Цель работы: научиться осуществлять перегрузку операторов относительно пользовательских типов.

Задачи работы:

– изучить операции, подлежащие перегрузке;

– получить практические навыки перегрузки операторов.

2. Теоретическая часть

2.1 Операции. Большинство операций пришло в C# из языков C, C++:

Категория	Операции
Арифметические	+, -, *, /, %
Логические	&, |, ^, ~, &&, ||, !
Конкатенация строк	+
Инкремент и декремент	++, --
Сравнение	<, >, >=, >=, ==, !=
Присвоение	+=, -=, /=, *=, %=, =, &=, |=, ^=, <<=, >>=

Доступ к члену класса	.
Индексация	[ ]
Приведение	( )
Условная (тернарная) операция	? :
Создание объектов	new
Информация о типе	sizeof, is, as, typeof,
Контроль исключения, связанного с переполнением	checked, unchecked

2.2 Перегрузка операций относительно класса. Рассмотрим простой класс, например, для представления объекта «Вектор» (для простоты возьмем вектор на плоскости). Пример 1:

```
class Vector
{
    // Поля данных представляют стороны параллелепипеда
    public double X, Y;

    // Конструкторы
    public Vector()
    {

    }

    public Vector(double XCoord, double YCoord)
    {
        X = XCoord;
        Y = YCoord;
    }
}
```

Класс создан, но в математике можно выполнять сложение (вычитание) векторов и умножение вектора на число, а также находить скалярное произведение векторов. Необходимо добавить эту возможность для нашего класса «Вектор». Это значит, что операции, указанные в примере 2 должны иметь смысл.

Пример 2:

```
Vector a = new Vector(4, 10);
Vector b = new Vector(10, 15);

// вычисление суммы векторов
Vector SumVector = a + b;

// Вычисление произведения вектора на число
Vector MultNum = 2 * a;

// вычисление скалярного произведения
double Skalar = a * b;
```

В данной реализации (пример 1) такие операции в коде невозможны. Добавим в класс перегруженные операции.

Пример 3:

```
class Vector
{
    // Поля данных представляют стороны параллелепипеда
    public double X, Y;

    // Конструкторы
    public Vector()
    {

    }

    public Vector(double XCoord, double YCoord)
    {
        X = XCoord;
        Y = YCoord;
    }

    // Перегрузка операции сложения
    public static Vector operator +(Vector left, Vector right)
    {
        return new Vector(left.X + right.X, left.Y + right.Y);
    }

    // Перегрузка операции умножения вектора на число
    public static Vector operator *(double k, Vector vek)
    {
        return new Vector(k*vek.X, k*vek.Y);
    }

    // Перегрузка операции умножения двух векторов (скалярное умножение)
    public static double operator *(Vector left, Vector right)
    {
        return left.X * right.X + left.Y * right.Y;
    }

}
```

Перегрузка операции похожа на объявление метода класса, но существуют отличия: используется ключевое слово static; используется ключевое слово operator и символ операции (+, *, - и т.д.) вместо имени метода.

После того как операции перегружены, возможно выполять следующие действия:

```
class Program
{
    static void Main(string[] args)
    {
        Vector a = new Vector(4, 10);
        Vector b = new Vector(10, 15);

        // вычисление суммы векторов
        Vector SumVector = a + b;

        // Вычисление произведения вектора на число
        Vector MultNum = 2 * a;

        // вычисление скалярного произведения
        double Skalar = a * b;

        // Проведение сложных расчетов
        Vector RES = 2 * a + 3 * b + (a * b) * a;


        Console.ReadKey();
    }
}
```

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку

и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте консольное приложение.

2. Совместно с преподавателем спроектируйте и разработайте класс, относительно которого перегрузите операции +, - /, *, %, ==, > (каждый студент разрабатывает свой класс).

Индивидуальные задания (возможные варианты классов):

«Вектор в пространстве», «Матрица», «Сотрудник», «Компьютер», «Товар», «Объект недвижимости», «Комплексное число», «Куб», «Автомобиль», «Студент», «Книга», «Дисциплина (предмет)».

Возможно предложение своего класса или доработка класса из работы №13.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое конструктор?

2. Что такое перегрузка операторов? Когда применяется данный механизм?

3. Допустим, что для класса создана перегруженная операция сложения. Может ли быть создана еще одна операция сложения для данного класса?

4. Какие операции нельзя перегрузить?

5. Какие операции необходимо перегружать попарно?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [3], [5].

ПРАКТИЧЕСКАЯ РАБОТА 12. ПОСТРОЕНИЕ ИЕРАРХИИ КЛАССОВ

1. Цель и содержание

Цель работы: изучить механизм организации наследования классов.

Задачи работы:

– научиться объявлять производные классы;

– научиться создавать иерархии классов;

– научиться использовать механизм полиморфизма.

2. Теоретическая часть

2.1 Наследование реализации. Наследование реализации (implementation inheritance) означает, что тип происходит от базового типа, получая от него все поля-члены и функции-члены.

Синтаксис наследования реализации:

```
class ПроизводныйКласс : БазовыйКласс
{
    // Данные-члены и функции-члены
}
```

Если при определении класса не указан базовый класс, то C# предполагает, что базовым классом является System.Object.

2.2 Создание иерархии классов. При наследовании реализации производный класс наследует реализацию каждой функции базового типа, если только в его определении не указано, что реализация функции должна быть переопределена.

Определим следующую иерархию классов (рис. 17.1) и продемонстрируем, как **наследуется реализация** и как **переопределяются** свойств и методов.

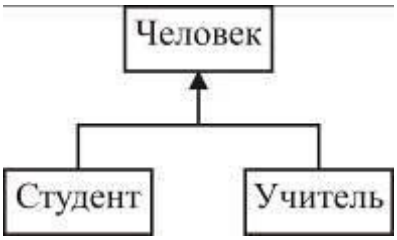


Рисунок 17.1 – Иерархия классов.

Создадим код, описывающий данную иерархию. Определим базовый класс:

```
class Человек
{
    public Человек(string Ф, string И, string О, int Возр)
    {
        Фамилия = Ф; Имя = И; Отчество = О; Возраст = Возр;
    }

    public Человек()
    {
        Фамилия = "нет данных"; Имя = ""; Отчество = "";
        Возраст = 18;
    }

    public string Фамилия, Имя, Отчество;
    private DateTime ДатаРождения;

    public virtual string ФИО
    {
        get { return Фамилия + " " + Имя + " " + Отчество; }
    }

    public int Возраст
    {
        get { return DateTime.Now.Year - ДатаРождения.Year; }
        set
        {
            int ГодРождения = DateTime.Now.Year - value;

            ДатаРождения = Convert.ToDateTime(ГодРождения.ToString()+".01.01");
        }
    }
}
```

Обратите внимание на наличие двух конструкторов, механизм хранения возраста человека и ключевое слово virtual у свойства «ФИО». Ключевое слово virtual указывает, что данное свойство будет переопределено в производном классе.

Определим производный класс «Учитель»:

```
class Учитель: Человек
{
    public Учитель()
        :base()
    {
        УченоеЗвание = УченыеЗвания.Без_Звания;
        УченаяСтепень = УченыеСтепени.Без_Степени;
    }

    public Учитель(string Ф, string И, string О, int Возр, УченыеЗвания УЗ, УченыеСтепени УС)
        :base(Ф, И, О, Возр)
    {
        УченоеЗвание = УЗ;
        УченаяСтепень = УС;
    }

    public УченыеЗвания УченоеЗвание;
    public УченыеСтепени УченаяСтепень;

    public override string ФИО
    {
        get
        {
            return УченаяСтепень.ToString() + ", " + УченоеЗвание.ToString() + ", " + base.ФИО;
        }
    }
}
```

Следует обратить внимание на использование ключевого слова override у свойства «ФИО», которое указывает на то, что данное свойство имеет новую реализацию, отличающуюся от реализации базового класса.

Определим второй производный класс «Студент»:

```
class Студент: Человек
{
    public Студент(string Ф, string И, string О, int Возр, Специальности Спец)
        :base(Ф, И, О, Возр)
    {
        Специальность = Спец;
    }

    public Специальности Специальность;

    public override string ФИО
    {
        get
        {
            return base.ФИО + ", " + Специальность.ToString();
        }
    }
}
```

В обоих производных классах следует обратить внимание на реализацию конструкторов производных классов, использование ключевого слова base в коде свойства «ФИО» и при объявлении конструкторов.

Также интерес представляют типы данных, используемые для членов-данных: «Специальности», «УченыеЗвания», «УченыеСтепени». Вот определения данных типов-перечислений:

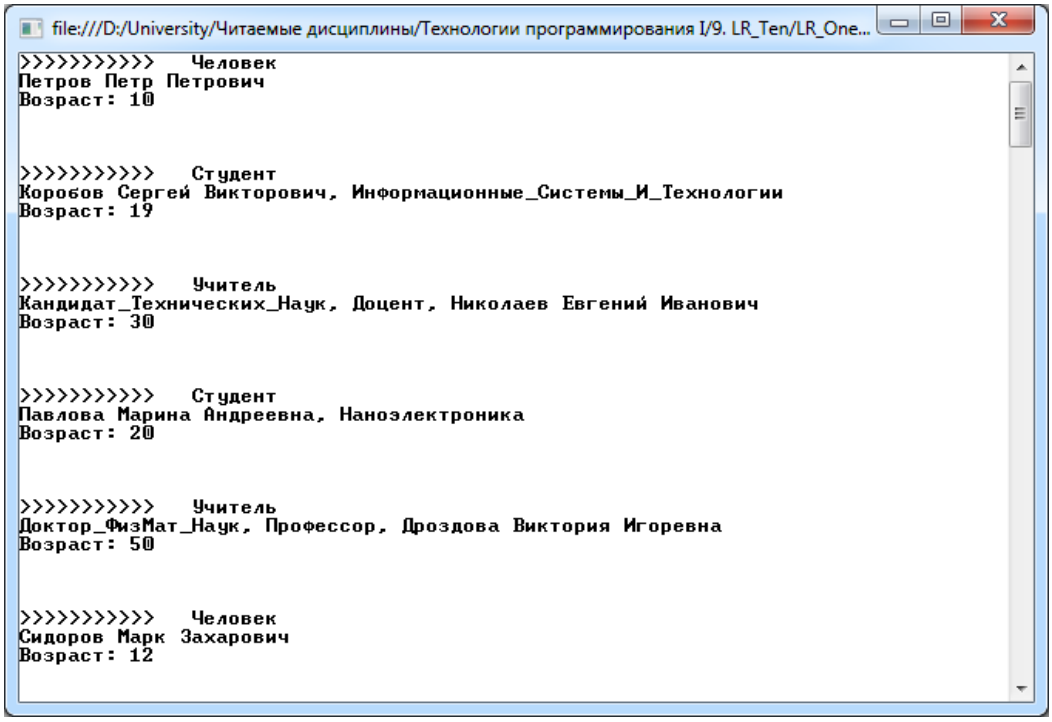
```
public enum УченыеЗвания
{
    Доцент,
    Профессор,
    Академик,
    Без_Звания
}

public enum УченыеСтепени
{
    Кандидат_Технических_Наук,
    Кандидат_ФизМат_Наук,
    Кандидат_Педагогических_Наук,
    Доктор_ФизМат_Наук,
    Без_Степени
}

public enum Специальности
{
    Информационные_Системы_И_Технологии,
    Безопасность_Информационных_Систем,
    Технология_Защиты_Информации,
    Психология,
    Наноэлектроника
}
```

Наконец, продемонстрируем использование объявленной иерархии классов.

В программе объявим массив объектов типа «Человек». Следует понимать, что при объявлении такого массива, его элементам можно присваивать объекты любого производного класса, причем каждый объект будет вести себя по-своему (за счет переопределения свойств и методов).

```
static void Main(string[] args)
{
    // Объявим массив людей:
    Человек[] mas = new Человек[6];

    // Заполним массив значениями разных (!) типов
    mas[0] = new Человек("Петров", "Петр", "Петрович", 10);

    mas[1] = new Студент("Коробов", "Сергей", "Викторович",
        19,
        Специальности.Информационные_Системы_И_Технологии);

    mas[2] = new Учитель("Николаев", "Евгений", "Иванович",
        30,
        УченыеЗвания.Доцент,
        УченыеСтепени.Кандидат_Технических_Наук);

    mas[3] = new Студент("Павлова", "Марина", "Андреевна",
        20,
        Специальности.Наноэлектроника);

    mas[4] = new Учитель("Дроздова", "Виктория", "Игоревна",
        50,
        УченыеЗвания.Профессор,
        УченыеСтепени.Доктор_ФизМат_Наук);

    mas[5] = new Человек("Сидоров", "Марк", "Захарович", 12);

    // Вывод инфорации
    for (int i = 0; i < mas.Length; i++)
    {
        Console.WriteLine(">>>>>>>>>>>    " + mas[i].GetType().Name);
        Console.WriteLine(mas[i].ФИО);
        Console.WriteLine("Возраст: " + mas[i].Возраст.ToString()+"\n");
        Console.WriteLine("\n");
    }

        Console.ReadKey();
}
```

В результате работы программы будет осуществлен следующий вывод:



Полная диаграмма типов в полученном приложении показана на рис. 17.2.

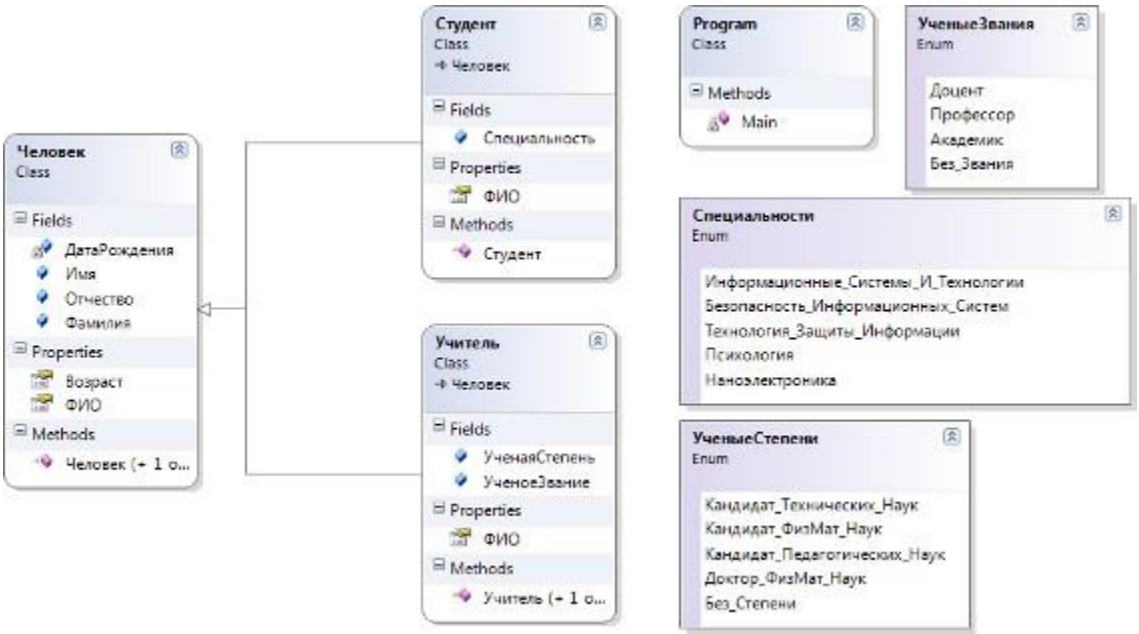


Рисунок 17.2 – Диаграмма типов приложения (создана средствами VS).

Структуры всегда наследуются от System.ValueType. Они могут также наследовать любое количество интерфейсов. Классы всегда наследуются от

одного класса по вашему выбору. Они также могут наследовать любое количество интерфейсов.

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом, представленным в работе №1.

2. Изучите пример создания иерархии классов, представленный в разделе «Теоретическое обоснование» данной работы.

3. Постройте свою иерархию классов в соответствии с индивидуальным заданием. В результате выполнения работы должны быть реализованы следующие механизмы:

– использование типа-перечисления (хотя бы одного);

– использование переопределенного свойства (хотя бы одного);

– использование переопределенного метода (хотя бы одного);

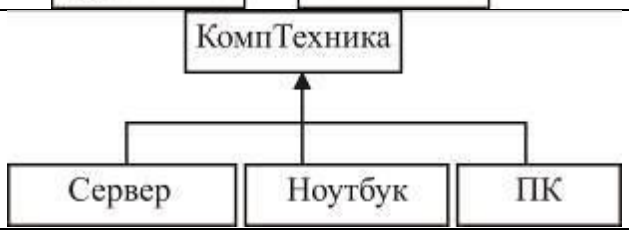
– использование вызова базового конструктора;

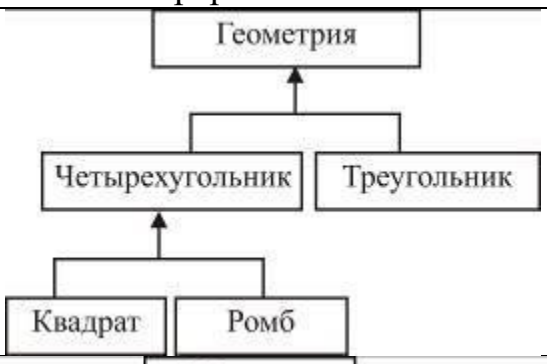
– использование вызова любого базового метода (отличного от конструктора).

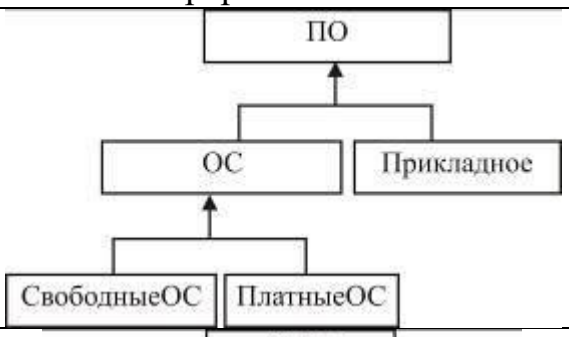
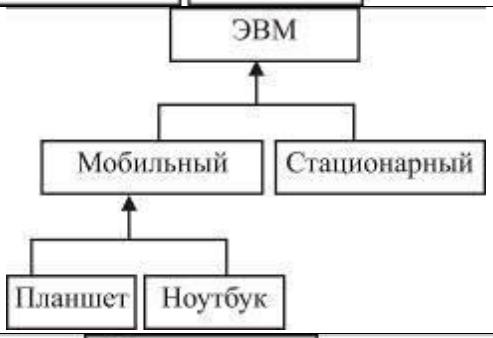
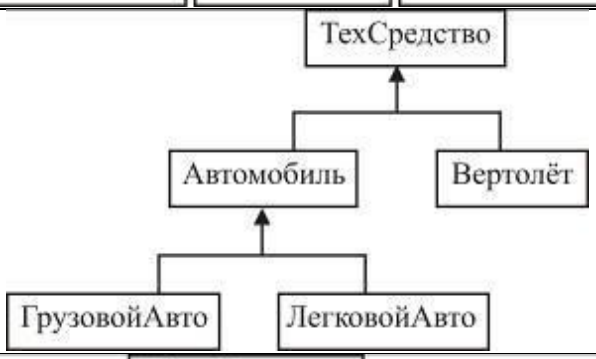
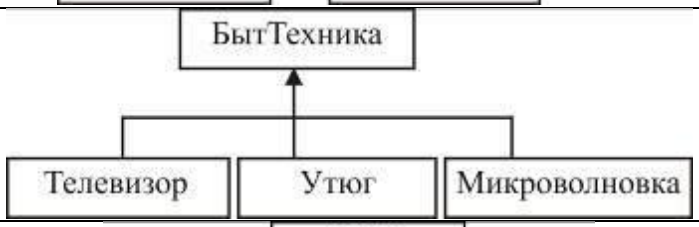
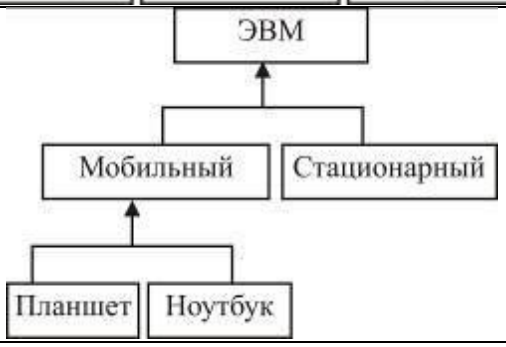
4. Продемонстрируйте использование классов, созданной иерархии (легче всего это сделать с использованием массивов). При защите работы укажите признаки присутствия полиморфного поведения в программе (реализация полиморфизма).

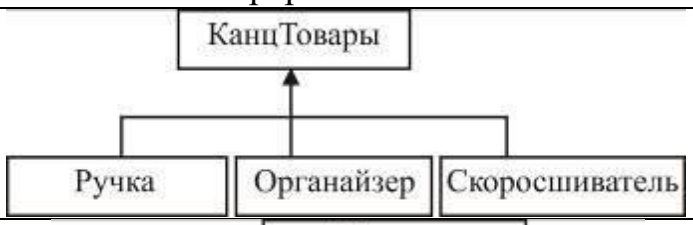
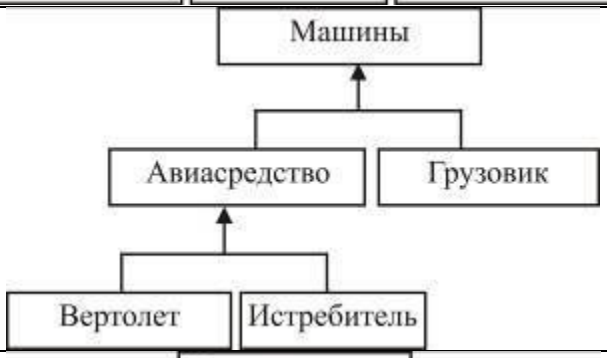
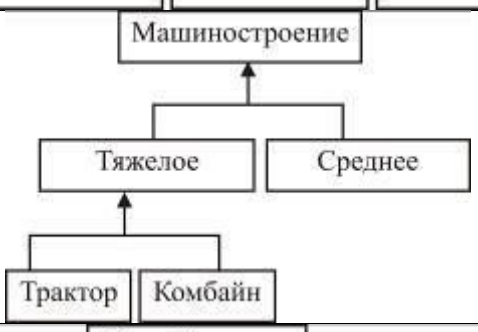
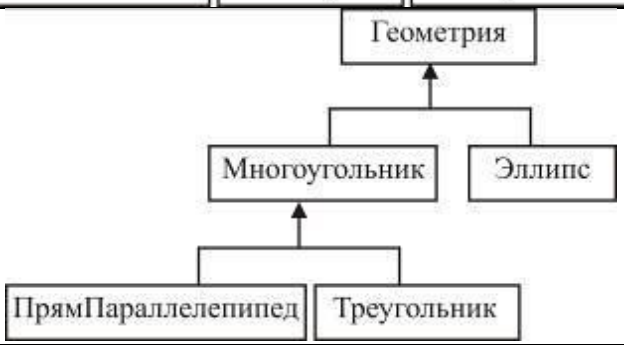
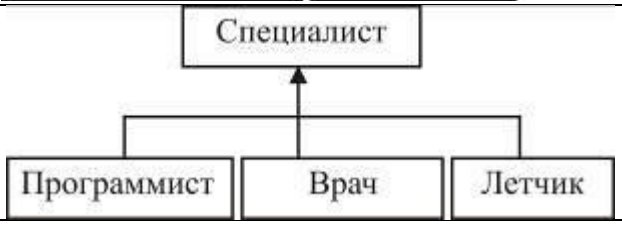
Индивидуальное задание.

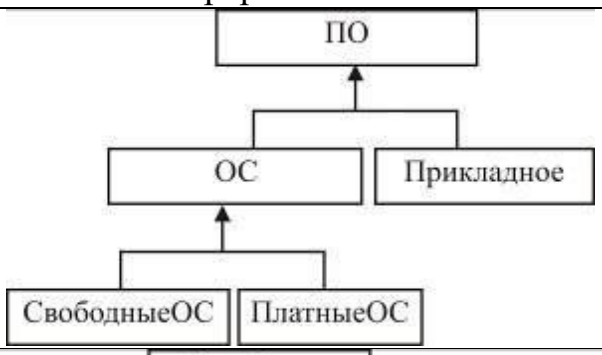
Спроектируйте класс, наполните его требуемой функциональностью, продемонстрируйте работоспособность класса.

Вариант	Иерархия классов
1	
2	

Вариант	Иерархия классов
3	
4	
5	
6	
7	
8	

Вариант	Иерархия классов
9	
10	
11	
12	
13	
14	

Вариант	Иерархия классов
15	
16	
17	
18	
19	
20	
21	

Вариант	Иерархия классов
22	
23	
24	
25	

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое наследование реализации? Как описать синтаксически наследование реализации?

2. Для чего используется ключевое слово base?

3. Можно ли переопределить метод класса? Свойства класса? Данные класса?

4. Как переопределить метод в производном классе?

5. Для чего используется ключевое слово virtual?

6. Для чего используется ключевое слово override?

7. Как поменять цвет фона в консольном приложении?

8. Как построить диаграмму типов данных, используемых в приложении средствами Visual Studio 2008/ 2010?

9. Сколько базовых классов может быть у любого класса в C#?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [5-7].

ПРАКТИЧЕСКАЯ РАБОТА 13. РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ

1. Цель и содержание

Цель работы: изучить принципы работы с типами интерфейсов в C#.

Задачи работы:

– научиться объявлять интерфейсы в C#;

– научиться создавать классы, реализующие интерфейсы;

2. Теоретическая часть

2.1 Наследование интерфейсов. Кроме наследования реализации (implementation inheritance) в языке C# реализован механизм наследования интерфейсов (interface inheritance). Необходимо понимать, что не все объектно-ориентированные языки поддерживают интерфейсы.

Если класс наследует интерфейс, класс как бы берет на себя обязательства реализовать некоторый функционал.

Синтаксис наследования интерфейса не отличается от синтаксиса наследования реализации:

class *КлассРеализующийИнтерфейс* : *Интерфейс*
{
    // *Данные– членыи функции – члены*
}

Отличие от механизма наследования реализации состоит в том, что:

- класс обязательно должен реализовать методы и свойства, продекларированные в интерфейсе (никакой реализации в интерфейсе не существует);

- класс может реализовывать (наследовать) несколько интерфейсов, тогда как базовый класс может быть только один.

2.2 Объявление интерфейсов. Для объявления типа интерфейса используется ключевое слово interface:

```
public interface ICalculate
{
    void Plus(int pPlus);
    void Minus(int pMinus);
}

public interface IVisual
{
    string Name { get; set; }
    void DrawObject();
}
```

В приведенном примере объявлены два интерфейса: ICalculate и IVisual.

Интерфейс ICalculate включает два метода с одним параметром. Из названий видно, что один метод что-то увеличивает, второй – что-то уменьшает на величину параметра.

Второй интерфейс IVisual содержит метод DrawObject без параметров, который призван нарисовать объект, а также свойство Name, которое доступно как для чтения, так и для записи.

Следует понимать, что интерфейсы не предполагают конкретную реализацию методов, а свойства интерфейсов не хранят данных. Интерфейс

должен быть реализован классом, который и определит конкретное наполнение методов и свойств интерфейса.

Для использования объявленных интерфейсов создадим два класса Human (человек) и Car (автомобиль).

```
public class Human
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;
}

public class Car
{
    public Car(string pManufacturer, string pModel, int pVelocity)
    {
        Manufacturer = pManufacturer;
        Model = pModel;
        Velocity = pVelocity;
    }

    private string Manufacturer;
    private string Model;
    private int Velocity;
}
```

В каждом классе определен конструктор, который инициализирует закрытые данные-члены класса.

Требуется для каждого класса реализовать вывод в консоль, а также возможности увеличения скорости автомобиля и возможности изменения возраста у человека.

Этот общий функционал можно реализовать с использованием следующих способов:

1. Добавить все необходимые функции в каждый класс независимо. Этот метод приводит к «разбуханию кода», сложной управляемости кода.

2. Реализовать один класс, например Base (базовый), в котором определить общие методы и свойства (Plus, Minus, DrawObject). Затем использовать класс Base в качестве базового для классов Human и Car, причем в каждом классе переопределить методы базового класса с использованием

ключевого слова override. Этот метод – наследование реализации. Получается иерархия классов, которые по смыслу и наполнению сильно отличаются. Хотя в данном подходе классы и код более упорядочены, все равно возникает избыточность за счет многократного переопределения методов.

3. Определение интерфейсов и реализация возможностей интерфейсов данными классами.

Именно метод 3 будет использован в данной работе.

4. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом, представленным в работе №1.

2. Определите в приложении классы Human и Car, а также интерфейсы ICalculate и IVisual, представленные в разделе «Теоретическое обоснование» данной работы.

3. Реализуем механизм наследования интерфейса ICalculate классом Human. Для этого выполним следующие действия:

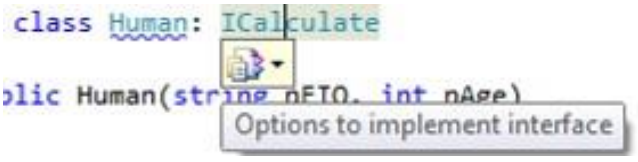
3.1. В определении класса укажем, что класс Human наследует интерфейс ICalculate.

```
public class Human: ICalculate
```

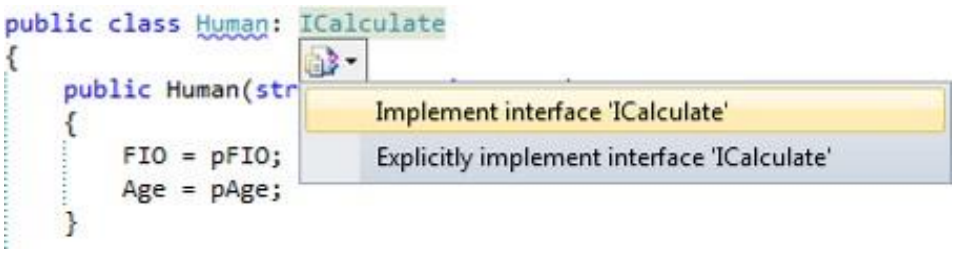
3.2. Обратите внимание, что после этого попытка перекомпилировать проект приведет к ошибкам:

То есть компилятор сообщает, что интерфейс наследуется классом, но методы, заявленные в интерфейсе, классом не реализованы. Реализуем их.

3.3. Наведите курсор мыши на интерактивное подчеркивание и появится выпадающий список:



Раскройте его и выберите команду «Implement interface» (реализовать интерфейс).



3.4. В классе Human появятся два новых метода, как и было объявлено в интерфейсе ICalculate. Определение класса примет вид:

```
public class Human: ICalculate
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;

    public void Plus(int pPlus)
    {
        throw new NotImplementedException();
    }

    public void Minus(int pMinus)
    {
        throw new NotImplementedException();
    }
}
```

Обратите внимание, что в каждый сгенерированный метод среда разработки добавила код вызова исключения, то есть проект скомпилируется без ошибок но в процессе выполнения, при попытке использовать методы Plus или Minus программа завершится с ошибками. Это сделано для того, чтобы программист не забыл реализовать данные методы интерфейсов.

В процессе выполнения пп. 3.1 – 3.3 были использованы возможности Visual Studio по автоматизации реализации интерфейса. Очевидно, что интерфейс можно было реализовать, самостоятельно написав данный код.

– использование вызова базового конструктора;

– использование вызова любого базового метода (отличного от конструктора).

4. Определим окончательную реализацию для методов Plus и Minus:

```
public void Plus(int pPlus)
{
    Age += pPlus;
}

public void Minus(int pMinus)
{
    Age -= pMinus;
}
```

5. Аналогичным образом реализуем наследование интерфейса IVisual для класса Human. Окончательно для класса Human получим:

```
public class Human: ICalculate, IVisual
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;

    public void Plus(int pPlus)
    {
        Age += pPlus;
    }

    public void Minus(int pMinus)
    {
        Age -= pMinus;
    }

    public string Name
    {
        get
        {
            return FIO + " : " +Age.ToString();
        }
        set
        {
            FIO = value;
        }
    }

    public void DrawObject()
    {
        Console.WriteLine
            (
                "    o    \n" +
                "  -----  \n" +
                "    |    \n" +
                "   / \\   \n" +
                "  /   \\  \n"
            );
        Console.WriteLine(Name);
    }
}
```

6. Реализуем интерфейсы ICalculate и IVisual для класса Car:

```
public class Car: IVisual, ICalculate
{
    public Car(string pManufacturer, string pModel, int pVelocity)
    {
        Manufacturer = pManufacturer;
        Model = pModel;
        Velocity = pVelocity;
    }

    private string Manufacturer;
    private string Model;
    private int Velocity;

    public void Plus(int pPlus)
    {
        Velocity += pPlus;
    }

    public void Minus(int pMinus)
    {
        Velocity += pMinus;
    }

    public string Name
    {
        get
        {
            return Manufacturer + " - " + Model +
                " : " + Velocity.ToString() + "km/h";
        }
        set
        {
            Model = value;
        }
    }

    public void DrawObject()
    {
        Console.WriteLine
            (
                "     -----------\n" +
                "____/           \\_____\n" +
                "|                     |\n" +
                "---(@)-----------(@)--- \n"
            );
        Console.WriteLine(Name);
    }
}
```

7. Когда все классы и интерфейсы определены и реализованы можно их использовать. Продемонстрируем использование типов данных созданием соответствующих объектов в функции main.

```
static void Main(string[] args)
{
    Console.BackgroundColor = ConsoleColor.White;
    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Clear();

    Console.Title = "Лабораторная работа №11";

    Human h = new Human("Иванов Иван Иванович", 50);
    h.Plus(5);
    h.Minus(1);
    h.DrawObject();

    Console.WriteLine("\n\n\n");

    Car car = new Car("Hyundai", "ix35", 120);
    car.Plus(25);
    car.Minus(11);
    car.DrawObject();

    Console.ReadKey();
}
```

8. Вид окна разработанного приложения представлен на рис. 18.1:

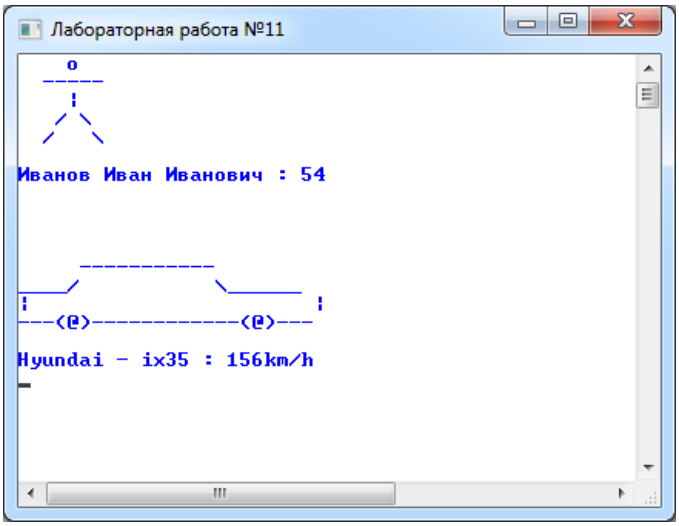


Рисунок 18.1 – Консольное приложение на основе интерфейсных типов.

Диаграмма типов данных, созданная редактором Visual Studio, для разработанного приложения, показана на рис. 18.2 (сравните ее с диаграммой на рис. 17.2 в работе №17):

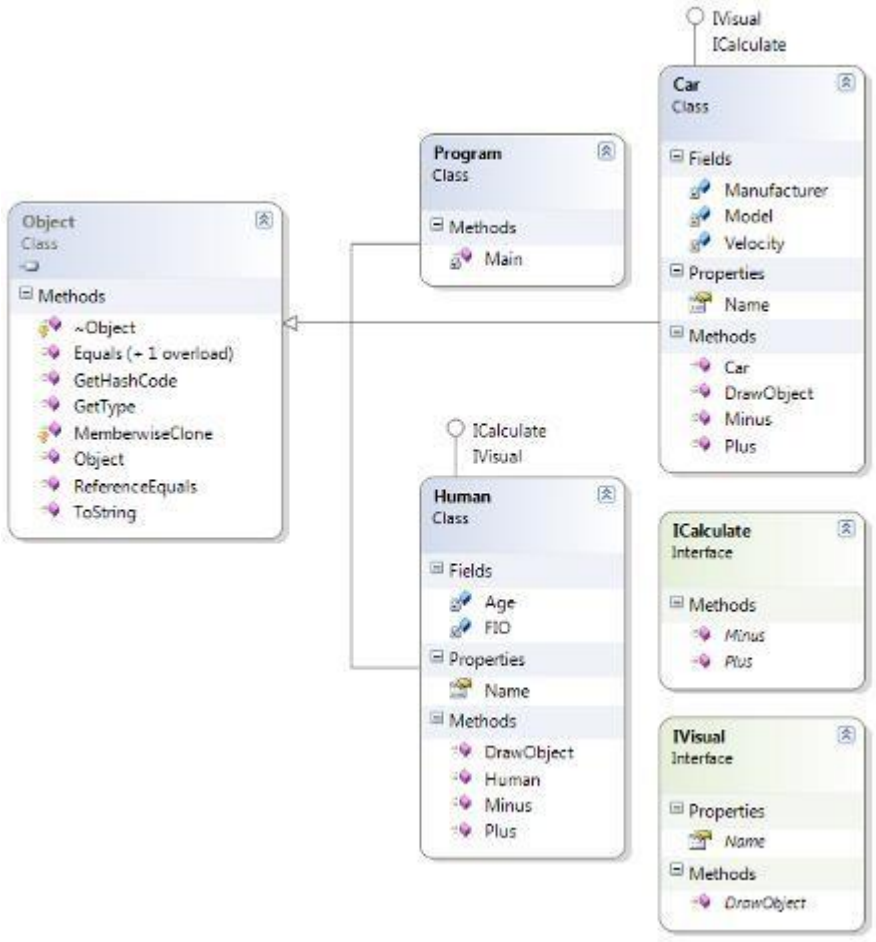


Рисунок 18.2 – Диаграмма классов приложения.

Индивидуальное задание.

Спроектируйте классы, наполните их требуемой функциональностью, определите интерфейс, продемонстрируйте использование объектов класса и методов интерфейса. Постройте диаграмму классов своего приложения средствами Visual Studio.

В качестве классов можно использовать иерархию классов, разработанную в работе №17. Интерфейс спроектируйте и реализуйте самостоятельно.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое наследование реализации? Как описать синтаксически наследование реализации?

2. Для чего используется ключевое слово base?

3. Можно ли переопределить метод класса? Свойства класса? Данные класса?

4. Что такое наследование интерфейса? Укажите основные отличия от наследования реализации.

5. Для чего используется ключевое слово virtual? Для чего используется ключевое слово override?

6. Может ли один класс наследовать несколько классов? Несколько интерфейсов?

7. Внимательно изучите код примеров данной работы и подумайте, каким образом специфицируются методы интерфейса public, private или protected? В чем причина применения именно такого модификатора доступа?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [6].

ПРАКТИЧЕСКАЯ РАБОТА 14. ФАЙЛОВЫЙ ВВОД-ВЫВОД. РАБОТА С КАТАЛОГАМИ. РАБОТА С ФАЙЛАМИ.

1. Цель и содержание

Цель работы: научиться использовать механизмы файлового ввода-вывода.

Задачи работы:

– научиться применять классы для работы с файлами;

– научиться применять классы для работы с каталогами;

– научиться использовать потоки ввода-вывода.

2. Теоретическая часть

2.1 Классы .NET Framework для реализации операций ввода-вывода.

Весь ввод и вывод в .NET Framework подразумевает использование потоков. Поток – это абстрактное представление последовательного устройства. Последовательное устройство – это нечто такое, что хранит данные в линейной структуре и точно таким же образом обеспечивает доступ к ним: считывает или записывает по одному байту за одну единицу времени.

Сохранение устройства абстрактным означает, что лежащие в основе источник/приемник данных могут быть скрыты. Такой уровень абстракции обеспечивает повторное использование кода и позволяет писать более обобщенные процедуры, потому что нет необходимости заботиться о действительной специфике передачи данных.

Для обработки файлов в C# необходима ссылка на пространство имен System.IO. При открытии файла создается объект, с которым ассоциируется поток. Потоки обеспечивают механизмы связи между файлами и программами.

Среда .NET Framework предоставляет все необходимые инструменты для эффективного использования файлов в приложениях.

Основные классы, необходимые программисту:

1. Object – исходный базовый класс для всех классов платформы .NET Framework и корень иерархии типов.

2. File – статический служебный класс, предоставляющий множество статических методов, которые дают возможность перемещать, создавать, копировать, удалять файлы, опрашивать и обновлять атрибуты, а также создавать объекты потоков FileStream.

3. Directory – статический служебный класс, предоставляющий множество статических методов для перемещения, копирования и удаления каталогов.

4. Path – служебный класс, используемый для манипулирования путевыми именами.

5. MarshalByRefObject – разрешает доступ к объектам через границы доменов приложения в приложениях, поддерживающих удаленное взаимодействие, это базовый класс для всех классов .NET, позволяющих удаленное взаимодействие.

6. FileInfo – представляет физический файл на диске, имеет методы для манипулирования этим файлом. Для любого объекта, который читает или пишет в этот файл, должен быть создан объект Stream. Все методы FileInfo доступны из объектной переменной, поэтому, если необходимо выполнить

только одно действие, более эффективным может оказаться использование метода File, а не соответствующего экземпляра метода FileInfo. Для всех методов FileInfo требуется путь к файлу, с которым проводится операция. Все статические методы класса FileInfo выполняют проверку безопасности для всех методов. Если необходимо использовать объект неоднократно, рекомендуется использовать соответствующий метод экземпляра FileInfo, поскольку в этом случае проверка безопасности будет требоваться не всегда.

7. DirectoryInfo – представляет физический каталог на диске и предоставляет методы уровня экземпляра для манипулирования каталогом. Класс DirectoryInfo работает точно так же, как класс FileInfo. Это объект, представляющий отдельный каталог на машине. Подобно классу FileInfo, многие из вызовов методов дублируются между Directory и DirectoryInfo.

8. FileSystemInfo – служит базовым классом для FileInfo и DirectoryInfo, обеспечивая возможность работы с файлами и каталогами одновременно, используя полиморфизм.

9. Stream – предоставляет универсальное представление последовательности байтов. Класс Stream является абстрактным базовым классом всех потоков.

10. FileStream – представляет файл, который может быть записан, прочитан или то и другое.

11. TextReader – представляет средство чтения, позволяющее считывать последовательные наборы знаков. Этот класс является абстрактным базовым классом для StreamReader, который считывает символы из потоков.

12. TextWriter – представляет средство записи, позволяющее записывать последовательные наборы символов. Этот класс является абстрактным базовым классом для StreamWriter, который записывают символы в потоки.

13. StreamReader – читает символьные данные из потока и может быть создан с использованием класса FileStream в качестве базового.

14. StreamWriter – пишет символьные данные в поток и может быть создан с использованием класса FileStream в качестве базового.

15. Component – предоставляет базовую реализацию интерфейса IComponent и делает возможным совместное использование объектов разными приложениями.

16. FileSystemWatcher – используется для мониторинга файлов и каталогов и представляет события, которые приложение может перехватить, когда в этих объектах происходят какие-то изменения.

Таким образом, эта система классов включает в себя классы для работы с файлами (File, FileInfo), каталогами (Directory, DirectoryInfo) и потоками (FileStream, StreamReader, StreamWriter).

В большинстве случаев для разработки бизнес-приложений достаточно лишь четырех классов для манипулирования файловой системой. Эти классы расположены пространстве имен System.IO и предназначены для работы с файловой системой компьютера, то есть для создания, удаления переноса файлов и каталогов.

Первые два типа – Directory и File реализуют свои возможности с помощью статических методов, поэтому данные классы можно использовать без создания соответствующих объектов (экземпляров классов).

Следующие типы – DirectoryInfo и FileInfo обладают схожими функциональными возможностями с Directory и File, но порождены от класса FileSystemInfo, поэтому реализуются путем создания соответствующих экземпляров классов.

Класс FileSystemInfo предоставляет базовый функционал. Значительная часть членов FileSystemInfo предназначена для работы с общими характеристиками файла или каталога (метками времени, атрибутами и т. п.). Рассмотрим некоторые свойства FileSystemInfo (таблица 19.1).

Таблица 19.1 – Свойства класса FileSystemInfo.

Свойство	Описание
Attributes	Позволяет получить или установить атрибуты для данного объекта файловой системы. Для этого свойства используются значения и перечисления FileAttributes

Свойство	Описание
CreationTime	Позволяет получить или установить время создания объекта файловой системы
Exists	Может быть использовано для того, чтобы определить, существует ли данный объект файловой системы
Extension	Позволяет получить расширение для файла
FullName	Возвращает имя файла или каталога с указанием пути к нему в файловой системе
LastAccessTime	Позволяет получить или установить время последнего обращения к объекту файловой системы
LastWriteTime	Позволяет получить или установить время последнего внесения изменений в объект файловой системы
Name	Возвращает имя указанного файла. Это свойство доступно только для чтения. Для каталогов возвращает имя последнего каталога в иерархии, если это возможно. Если нет, возвращает полностью определенное имя

В FileSystemInfo предусмотрен набор методов. Например, метод Delete() – позволяет удалить объект файловой системы с жесткого диска, а Refresh() – обновить информацию об объекте файловой системы.

2.2 Классы для работы с каталогами файловой системы.

Класс DirectoryInfo наследует члены класса FileSystemInfo и содержит дополнительный набор членов, которые предназначены для создания, перемещения, удаления, получения информации о каталогах и подкаталогах в файловой системе. Наиболее важные члены класса содержатся в таблице 2.2.

Таблица 19.2 – Доступные члены класса DirectoryInfo.

Член	Описание
Create() CreateSubDirectory()	Создают каталог (или подкаталог) по указанному пути в файловой системе
Delete()	Удаляет пустой каталог
GetDirectories()	Позволяет получить доступ к подкаталогам текущего каталога (в виде массива объектов DirectoryInfo)
GetFiles()	Позволяет получить доступ к файлам текущего каталога

Член	Описание
	(в виде массива объектов FileInfo)
MoveTo()	Перемещает каталог и все его содержимое на новый адрес в файловой системе
Parent	Возвращает родительский каталог в иерархии файловой системы

Работа с типом DirectoryInfo начинается с того, что создается экземпляр класса (объект), при вызове конструктора в качестве параметра указывается путь к нужному каталогу. Если необходимо обратиться к текущему каталогу (то есть каталогу, в котором в настоящее время производится выполнение приложения), вместо параметра используется обозначение ".". Например:

```
// Создаем объект DirectoryInfo,
// которому будет обращаться к текущему каталогу
DirectoryInfo dir1 = new DirectoryInfo(".");

// Создаем объект DirectoryInfo,
// которому будет обращаться к каталогу d:\prim
DirectoryInfo dir2 = new DirectoryInfo(@"d:\prim");
```

Если создается объект DirectoryInfo, который связывается с несуществующим каталогом, то будет сгенерировано исключение System.IO.DirectoryNotFoundException.

Свойство Attributes класса DirectoryInfo позволяет получить информацию об атрибутах объекта файловой системы. Возможные значения данного свойства приведены в следующей таблице 19.3.

Таблица 19.3 – Значения свойства Attributes.

Значение	Описание
Archive	Этот атрибут используется приложениями при проведении резервного копирования, а в некоторых случаях - удаления старых файлов
Compressed	Определяет, что файл является сжатым
Directory	Определяет, что объект файловой системы является каталогом
Encrypted	Определяет, что файл является зашифрованным
Hidden	Определяет, что файл является скрытым (такой файл не будет выводиться при обычном просмотре каталога)
Normal	Определяет, что файл находится в обычном состоянии и для него установлены любые другие атрибуты. Этот атрибут не может

Значение	Описание
	использоваться с другими атрибутами
Offline	Файл (расположенный на сервере) кэширован в хранилище off-line на клиентском компьютере. Возможно, что данные этого файла уже устарели
Readonly	Файл доступен только для чтения
System	Файл является системным (то есть файл является частью операционной системы или используется исключительно операционной системой)

Через класс DirectoryInfo программист может собрать информацию о дочерних подкаталогах. Например:

```
static void printDirect(DirectoryInfo dir)
{
    Console.WriteLine("***** " + dir.Name + " *****");
    Console.WriteLine("FullName: {0}", dir.FullName);
    Console.WriteLine("Name: {0}", dir.Name);
    Console.WriteLine("Parent: {0}", dir.Parent);
    Console.WriteLine("Creation: {0}", dir.CreationTime);
    Console.WriteLine("Attributes: {0}", dir.Attributes.ToString());
    Console.WriteLine("Root: {0}", dir.Root);
}

static void Main(string[] args)
{
    DirectoryInfo dir = new DirectoryInfo(@"d:\prim");
    printDirect(dir);
    DirectoryInfo[] subDirects = dir.GetDirectories();
    Console.WriteLine("Найдено {0} подкаталогов", subDirects.Length)
    foreach (DirectoryInfo d in subDirects)
    {
        printDirect(d);
    }
}
```

Метод CreateSubdirectory() позволяет создать в выбранном каталоге как единственный подкаталог, так и множество подкаталогов (в том числе, и вложенных друг в друга). Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim");

//создали подкаталог
dir.CreateSubdirectory("doc");

//создали вложенный подкаталог
dir.CreateSubdirectory(@"book\2008");
```

Метод MoveTo() позволяет переместить текущий каталог по заданному в качестве параметра адресу. При этом возможно произвести переименование каталога. Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim\bmp");
dir.MoveTo(@"d:\prim\letter\Николаев");
```

В данном случае каталог «bmp» перемещается по адресу «d:\prim\letter\Николаев». Так как имя перемещаемого каталога не совпадает с крайним правым именем в адресе нового местоположения каталога, то производится переименование.

Работать с каталогами файловой системы компьютера можно и при помощи класса Directory, функциональные возможности которого во многом совпадают с возможностями DirectoryInfo. Следует учитывать, что члены данного класса реализованы статически, поэтому для их использования нет необходимости создавать объект. Например:

```
// Создание подкаталога Новый
Directory.CreateDirectory(@"d:\prim\Новый");

// Перемещение каталога Новый в каталог 2012
Directory.Move(@"d:\prim\Новый",
               @"d:\prim\2012\Новый");

//переименовали каталог Новый в Старый
Directory.Move(@"d:\prim\2012\Новый",
               @"d:\prim\2012\Старый");
```

Следует учитывать, что удаление каталога возможно только когда он пуст. На практике комбинируют использование классов Directory и DirectoryInfo.

2.3 Классы для работы с файлами.

Класс FileInfo предназначен для организации доступа к физическому файлу, который содержится на жестком диске компьютера. Он позволяет получать информацию об этом файле (например, о времени его создания, размере, атрибутах), а также производить различные операции, например, по созданию файла или его удалению. Класс FileInfo наследует члены класса FileSystemInfo и содержит дополнительный набор членов, который приведен в следующей таблице 19.4

Таблица 19.4 – Члены класса FileInfo.

Член	Описание
AppendText()	Создает объект StreamWriter для добавления текста к файлу.
CopyTo()	Копирует уже существующий файл в новый файл.
Create()	Создает новый файл и возвращает объект FileStream для взаимодействия с этим файлом.
CreateText()	Создает объект StreamWriter для записи текстовых данных в новый файл.
Delete()	Удаляет файл, которому соответствует объект FileInfo.
Directory	Возвращает каталог, в котором расположен данный файл.
DirectoryName	Возвращает полный путь к данному файлу в файловой системе.
Length	Возвращает размер файла.
MoveTo()	Перемещает файл в указанное пользователем место (этот метод позволяет одновременно переименовать данный файл).
Name	Позволяет получить имя файла.
Open()	Открывает файл с указанными пользователем правами доступа на чтение, запись или совместное использование с другими пользователями.
OpenRead()	Создает объект FileStream, доступный только для чтения.
OpenText()	Создает объект StreamReader (о нем также будет рассказано ниже), который позволяет считывать информацию из существующего текстового файла.
OpenWrite()	Создает объект FileStream, доступный для чтения и записи.

Большинство методов FileInfo возвращает объекты классов FileStream, StreamWriter, StreamReader и т. п., которые позволяют различным образом взаимодействовать с файлом, например, производить чтение или запись в него. Например:

```
// Создание объекта FileInfo
FileInfo f = new FileInfo("text.txt");

// Создание файла и потока
StreamWriter fOut =
    new StreamWriter(f.Create());

// Запись текста в файл
fOut.WriteLine("ОДИН ДВА ТРИ...");

// Закрытие файла
fOut.Close();

// Получение информации о файле
Console.WriteLine("Name: {0}", f.Name);
Console.WriteLine("File size: {0}",
    f.Length);
Console.WriteLine("Creation: {0}",
    f.CreationTime);
Console.WriteLine("Attributes: {0}",
    f.Attributes.ToString());
```

Доступ к физическим файлам можно получать и через статические методы класса File. Большинство методов объекта FileInfo представляют в этом смысле зеркальное отражение методов объекта File.

2.4 Потоки в системе ввода-вывода.

Программы на языке C# выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов. Поток (stream) – это абстракция, которая генерирует и принимает данные. С помощью потока можно читать данные из различных источников (клавиатура, файл, память) и записывать в различные источники (принтер, экран, файл, память).

Центральную часть потоковой системы C# занимает класс Stream пространства имен System.IO. Класс Stream представляет байтовый поток и является базовым для всех остальных потоковых классов. Производными от класса Stream являются классы потоков:

1. FileStream – байтовый поток, разработанный для файлового ввода-вывода.

2. BufferedStream – заключает в оболочку байтовый поток и добавляет буферизацию, которая во многих случаях увеличивает производительность программы.

3. MemoryStream – байтовый поток, который использует память для хранения данных.

Программист может реализовать собственные потоковые классы. Однако для подавляющего большинства приложений достаточно встроенных потоков.

2.4.1 Байтовый поток.

Чтобы создать байтовый поток, связанный с файлом, создается объект класса FileStream. При этом в классе определено несколько конструкторов. Чаще всего используется конструктор, который открывает поток для чтения и (или) записи:

```
public FileStream(string path, FileMode mode);
```

Параметр path определяет имя файла, с которым будет связан поток ввода-вывода данных. Параметр mode определяет режим открытия файла, который может принимать одно из возможных значений, определенных перечислением FileMode:

– FileMode.Append – предназначен для добавления данных в конец файла;

– FileMode.Create – предназначен для создания нового файла, при этом если существует файл с таким же именем, то он будет предварительно удален;

– FileMode.CreateNew – предназначен для создания нового файла, при этом файл с таким же именем не должен существовать;

– FileMode.Open – предназначен для открытия существующего файла;

– FileMode.OpenOrCreate – если файл существует, то открывает его, в противном случае создает новый;

– FileMode.Truncate – открывает существующий файл, но усекает его длину до нуля.

Если попытка открыть файл оказалась неудачной, то генерируется одно из исключений:

– FileNotFoundException – файл невозможно открыть по причине его отсутствия;

– IOException – файл невозможно открыть из-за ошибки ввода-вывода;

– ArgumentNullException – имя файла представляет собой null-значение;

– ArgumentException – некорректен параметр mode;

– SecurityException – пользователь не обладает правами доступа;

– DirectoryNotFoundException – некорректно задан каталог.

Другая версия конструктора позволяет ограничить доступ только чтением или только записью:

```
public FileStream(string path, FileMode mode, FileAccess access);
```

Параметры path и mode имеют то же назначение, что и в предыдущей версии конструктора. Параметр access, определяет способ доступа к файлу и может принимать одно из значений, определенных перечислением FileAccess:

– FileAccess.Read – только чтение;

– FileAccess.Write – только запись;

– FileAccess.ReadWrite – и чтение, и запись.

После установления связи байтового потока с физическим файлом внутренний указатель потока устанавливается на начальный байт файла.

Для чтения очередного байта из потока, связанного с физическим файлом, используется метод ReadByte( ). После прочтения очередного байта внутренний указатель перемещается на следующий байт файла. Если достигнут конец файла, то метод ReadByte( ) возвращает значение -1.

Для побайтовой записи данных в поток используется метод WriteByte( ).

По завершении работы с файлом его необходимо закрыть. Для этого достаточно вызвать метод Close( ). При закрытии файла освобождаются системные ресурсы, ранее выделенные для этого файла, что дает возможность использовать их для работы с другими файлами.

```
static void Main(string[] args)
{
    try
    {
        // Создание потока для чтения из файла
        FileStream fileIn =
            new FileStream("ФайлСТекстом.txt",
                FileMode.Open,
                FileAccess.Read);
        // Создание потока для записи в файл
        FileStream fileOut =
            new FileStream("ФайлКопия.txt",
                FileMode.Create,
                FileAccess.Write);
        int i;

        // В цикле производится чтение одного файла
        // и одновременная запись содержимого
        // во второй файл
        while ((i = fileIn.ReadByte()) != -1)
        {
            // запись очередного байта в поток
            fileOut.WriteByte((byte)i);
        }

        // Закрыте файлов
        fileIn.Close();
        fileOut.Close();
    }
    catch (Exception EX)
    {
        Console.WriteLine(EX.Message);
    }
}
```

В данном примере создаются два потока fileIn (для чтения байтов из файла в поток) и fileOut (для записи байтов из потока в файл). Каждый из потоков ассоциируется со своим файлом. Затем в цикле производится побайтовое чтение файла «ФайлСТекстом.txt» до момента, когда функция ReadByte( ) вернет значение -1 (то есть достигнут конец файла). При этом на каждой итерации цикла считывается один байт и на этой же итерации этот байт записывается в файл «ФайлКопия.txt». После всех операция оба потока закрываются. Весь код, осуществляющий работу с файлами заключен в конструкцию try … catch. Это позволяет повысить устойчивость программы к ошибкам.

2.4.2 Символьный поток.

Чтобы создать символьный поток нужно поместить объект класса Stream (например FileStream) внутрь объекта класса StreamWriter или объекта класса StreamReader. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс StreamWriter предназначен для организации выходного символьного потока. В нем определено несколько конструкторов. Один из них записывается следующим образом:

```
public StreamWriter(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока. Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для вывода, и исключение типа ArgumentNullException, если он (поток) имеет null-значение.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

```
public StreamWriter(string path);
```

Параметр path определяет имя открываемого файла.

Например, создать экземпляр класса StreamWriter можно следующим образом:

```
StreamWriter fileOut =
    new StreamWriter(
        new FileStream(
            "ФайлНиколаева.txt",
            FileMode.Create,
            FileAccess.Write));
```

И еще один вариант конструктора StreamWriter:

```
public StreamWriter(string path, bool append);
```

Параметр path определяет имя открываемого файла, а параметр append может принимать значение true – если нужно добавлять данные в конец файла, или false – если файл необходимо перезаписать.

Например:

```
// Добавляем символы в конец файла
StreamWriter fileOut_1 =
    new StreamWriter("МойФ.txt", true);
```

Объявив таким образом переменную fileOut_1 для записи данных в поток можно обратиться к методу WriteLine:

```
// Использование WriteLine
fileOut_1.WriteLine("Строка для записи");
```

В данном случае для записи используется метод, аналогичный статическому методу класса Console. Это действительно схожие механизмы ввода-вывода.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
public StreamReader(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока.

Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для ввода. Создать экземпляр класса StreamReader можно следующим образом:

```
StreamReader fileIn =
    new StreamReader(
        new FileStream(
            "text.txt",
            FileMode.Open,
            FileAccess.Read));
```

Как и в случае с классом StreamWriter у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
public StreamReader(string path);
```

Параметр path определяет имя открываемого файла. Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn =
    new StreamReader
        (@"c:\temp\Николаев.txt");
```

В C# символы реализуются кодировкой Unicode. Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русские символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора StreamReader:

```
StreamReader fileIn =
    new StreamReader(
        @"c:\temp\t.txt",
        Encoding.GetEncoding(1251));
```

Параметр Encoding.GetEncoding(1251) говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode. Тип Encoding реализован в пространстве имен System.Text.

Для чтения данных из потока fileIn можно воспользоваться методом ReadLine. При этом если будет достигнут конец файла, то метод ReadLine вернет значение null.

Рассмотрим пример, в котором данные из одного файла копируются в другой, но уже с использованием классов StreamWriter и StreamReader.

```
static void Main(string[] args)
{
    // Создание символьных потоков
    StreamReader fileIn =
        new StreamReader(
            "ФайлСТекстом.txt",
            Encoding.GetEncoding(1251));
    StreamWriter fileOut =
        new StreamWriter(
            "НовыйФайл.txt",
            false);
    string line;

    // Построчное считывание
    while ((line = fileIn.ReadLine()) != null)
    {
        // ... и запись строки
        fileOut.WriteLine(line);
    }

    fileIn.Close();
    fileOut.Close();
}
```

В данном примере осуществляется копирование содержимого одного символьного файла в другой.

Таким образом, данный способ копирования одного файла в другой, дает тот же результат, что и при использовании байтовых потоков. Однако, его работа будет менее эффективной, т.к. будет тратиться дополнительное время на преобразование байтов в символы. У символьных потоков есть и свои преимущества. Например, можно использовать регулярные выражения для поиска заданных фрагментов текста в файле.

2.4.3 Перенаправление стандартных потоков.

Тремя стандартными потоками, доступ к которым осуществляется через свойства Console.Out, Console.In и Console.Error, могут пользоваться все программы, работающие в пространстве имен System. Свойство Console.Out относится к стандартному выходному потоку. По умолчанию это консоль. Например, при вызове метода Console.WriteLine() информация автоматически передается в поток Console.Out. Свойство Console.In относится к стандартному входному потоку, источником которого по умолчанию является клавиатура. Например, при вводе данных с клавиатуры информация автоматически передается потоку Console.In, к которому можно обратиться с помощью метода Console.ReadLine(). Свойство Console.Error относится к ошибкам в стандартном потоке, источником которого также по умолчанию является консоль. Однако эти потоки могут быть перенаправлены на любое совместимое устройство ввода-вывода, например, на работу с физическими файлами.

Перенаправить стандартный поток можно с помощью методов SetIn(), SetOut() и SetError(), которые являются членами класса Console:

```
static void SetIn(TextReader input);
static void SetOut(TextWriter output);
static void SetError(TextWriter output);
```

Пример перенаправления потоков представлен в следующей программе, демонстрирующей, что стандартный поток вывода перенаправляется в один файл, а поток ввода – в другой:

```
static void Main(string[] args)
{
    int[,] MyArray;
    StreamReader file = new StreamReader("input.txt");
    // Перенаправление на file
    Console.SetIn(file);
    string line = Console.ReadLine();
    string[] mas = line.Split(' ');
    int n = int.Parse(mas[0]);
    int m = int.Parse(mas[1]);
    MyArray = new int[n, m];
    for (int i = 0; i < n; i++)
    {
        line = Console.ReadLine();
        mas = line.Split(' ');
        for (int j = 0; j < m; j++)
        {
            MyArray[i, j] = int.Parse(mas[j]);
        }
    }
    PrintArray("Исходный массив:", MyArray, n, m);
    file.Close();
}

static void PrintArray(string a, int[,] mas, int n, int m)
{
    // Перенаправление на file
    StreamWriter file=new StreamWriter("output.txt");
    Console.SetOut(file);
    Console.WriteLine(a);
    for (int i = 0; i < n; i++)
    {
        for (int j=0; j<m; j++) Console.Write("{0} ", mas[i,j]);
        Console.WriteLine();
    }
    file.Close();
}
```

4. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

Для выполнения работы необходимо спроектировать многомодульное приложение, использующее файлы для ввода и вывода информации.

Например, требуется разработать приложение, которое позволяет:

– генерировать матрицу случайных чисел с сохранением ее в файл;

– считывать матрицу из файла;

– выводить матрицу на экран.

Для выполнения данного задания-примера достаточно в отдельном проекте реализовать класс. Этот проект следует скомпилировать в виде dll-файла. Затем, в основной программе просто необходимо использовать данную библиотеку.

Класс библиотеки назовем Matrix. В нем определим следующие функции:

```
class Matrix
{
    private float[,] matrix;
    int m, n;

    // Конструктор
    public Matrix()...

    // Генерация матрицы заданного размера
    public void GenerateMatrix(int M, int N)...

    // Сохранение сгенерированной матрицы в файл
    public void SaveMatrix(string pFileName)...

    // Загрузка сохраненной матрицы из файла
    public Boolean LoadMatrix(string pFileName)...

    // Вывод матрицы в консоль
    public void PrintMatrix()...
}
```

Рисунок 19.1 – Основные методы класса Matrix.

Листинг метода GenegateMatrix:

```
// Генерация матрицы заданного размера
public void GenerateMatrix(int M, int N)
{
    m = M; n = N;

    Random r = new Random(DateTime.Now.Millisecond);

    matrix = new float[M, N];

    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            matrix[i, j] = (float)r.Next(1000) / 973f;
}
```

Рисунок 19.2 – Метод для создания матрицы заданного размера и заполнения ее случайными числами.

Листинг метода SaveMatrix:

```
// Сохранение сгенерированной матрицы в файл
public void SaveMatrix(string pFileName)
{
    if (matrix.Length > 0)
    {
        if (File.Exists(pFileName))
            File.Delete(pFileName);

        FileInfo f = new FileInfo(pFileName);
        TextWriter tw = f.CreateText();

        tw.WriteLine(m.ToString());
        tw.WriteLine(n.ToString());

        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                tw.WriteLine(i.ToString() + " " + j.ToString() + " " + matrix[i, j].ToString("E10"));

        tw.Close();
    }
}
```

Рисунок 19.3 – Метод для сохранения матрицы в файл.

```
// Загрузка сохраненной матрицы из файла
public Boolean LoadMatrix(string pFileName)
{
    if (File.Exists(pFileName))
    {
        try
        {
            TextReader tr = File.OpenText(pFileName);
            m = Convert.ToInt32(tr.ReadLine());
            n = Convert.ToInt32(tr.ReadLine());

            matrix = new float[m, n];
            string line;
            string[] substring;

            for (int i = 0; i < m; i++)
                for (int j = 0; j < n; j++)
                {
                    line = tr.ReadLine();
                    substring = line.Split(new char[] {' '}, 3);
                    matrix[i, j] = Convert.ToSingle(substring[2]);
                }

            tr.Close();

            return true;

        }
        catch
        {
            return false;
        }
    }

    return false;
}
```

Рисунок 19.4 – Метод загрузки матрицы из файла.

На рис. 19.4 представлен листинг метода LoadMatrix. Листинг метода PrintMatrix:

```
// Вывод матрицы в консоль
public void PrintMatrix()
{
    if (matrix.Length > 0)
    {
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
                Console.Write(matrix[i, j].ToString("E3") + "  ");

            Console.WriteLine();
        }

    }
}
```

Рисунок 19.5 – Метод для вывода матрицы.

Основная программа для использования класса-матрицы представлена на рис. 19.6

```
class Program
{
    static void Main(string[] args)
    {
        Console.BackgroundColor = ConsoleColor.White;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.Clear();

        Matrix m = new Matrix();

        m.GenerateMatrix(10, 5);
        m.SaveMatrix("FileForMatrix.txt");

        if (m.LoadMatrix("FileForMatrix.txt"))
            m.PrintMatrix();

        Console.ReadKey();
    }
}
```

Рисунок 19.6 – Работа с библиотечным классом.

В результате работы данного кода будет создан файл FileForMatrix.txt следующего содержания:

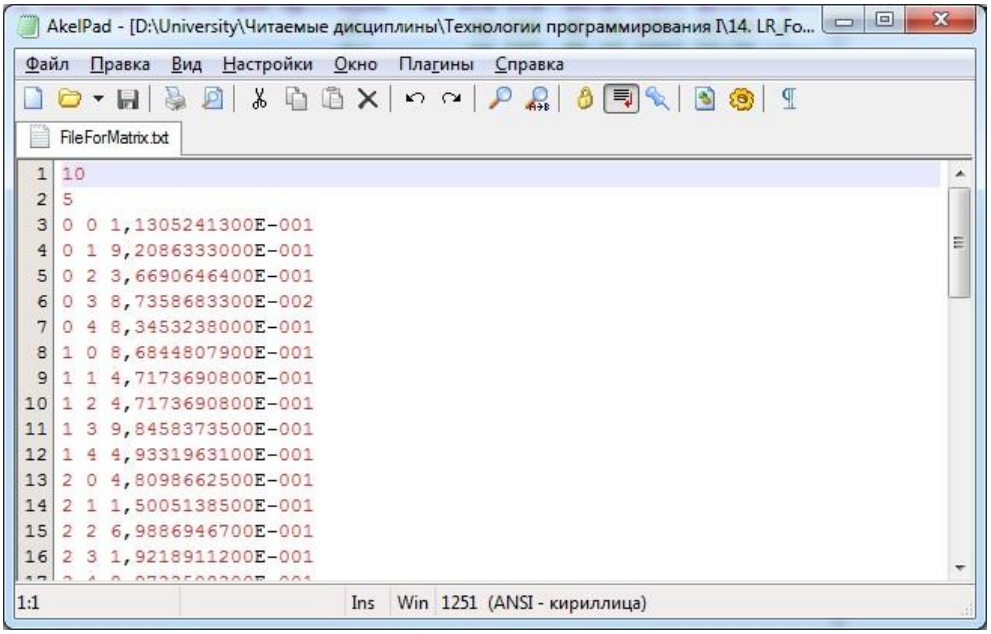


Рисунок 19.7 – Результирующий файл.

На экран будет выведена следующая информация:

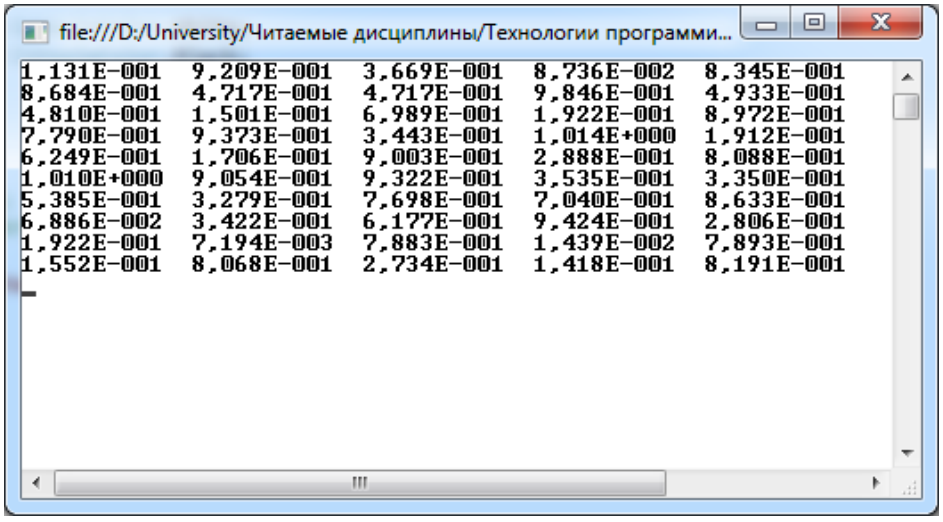


Рисунок 19.8 – Вывод программы в консоль.

Выполните индивидуальное задание, согласно предложенному варианту. В каждом варианте необходимо спроектировать многомодульное приложение (оптимально 2 модуля). Исходные данные в каждом варианте программа получает из входного файла.

Индивидуальное задание.

Вариант	Структура данных
1	Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах.
2	Программа рассчитывает сумму двух матриц, которые хранятся в разных файлах.
3	Программа находит максимальный элемент в двух матрицах, которые хранятся в разных файлах.
4	Программа рассчитывает сумму диагональных элементов двух матриц, которые хранятся в разных файлах.
5	Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах.
6	Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах.
7	Программа рассчитывает сумму элементов главной диагонали первой матрицы и сумму элементов второстепенной диагонали второй матрицы. Матрицы хранятся в разных файлах.
8	Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
9	Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
10	Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах.
11	Программа рассчитывает сумму элементов , находящихся в четном столбце и нечетной строке, в двух матрицах, которые хранятся в разных файлах.
12	Программа рассчитывает сумму элементов четных строк и расположенных на второстепенной диагонали в обеих матрицах, которые хранятся в разных файлах.
13	Программа рассчитывает произведение элементов в двух матрицах, которые хранятся в разных файлах.
14	Программа рассчитывает сумму первой матрицы и матрицы транспонированной относительно второй. Обе матрицы хранятся в отдельных файлах.
15	Программа рассчитывает произведение элементов диагонали первой матрицы и сумму всех элементов второй матрицы. Обе матрицы хранятся в отдельных файлах.
16	Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах.
17	Программа находит максимальный элемент в двух матрицах, которые хранятся в разных файлах.

Вариант	Структура данных
18	Программа рассчитывает сумму элементов четных строк и расположенных на второстепенной диагонали в обеих матрицах, которые хранятся в разных файлах.
19	Программа рассчитывает сумму первой матрицы и матрицы транспонированной относительно второй. Обе матрицы хранятся в отдельных файлах.
20	Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах.
21	Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
22	Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
23	Программа рассчитывает произведение элементов диагонали первой матрицы и сумму всех элементов второй матрицы. Обе матрицы хранятся в отдельных файлах.
24	Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах.
25	Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Какие классы для работы с файловой системой вы знаете?

2. Что такое сборка?

3. Как определить проект по умолчанию в многомодульном решении?

4. Какие классы отвечают за представление файлов в программе?

5. Что такое поток? Какие типы классов потоков используются при работе с файлами?

6. Опишите последовательность действий при необходимости записать одну строку в файл. Приведите примеры использования различных классов.

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [5], [6-8].

ПРАКТИЧЕСКАЯ РАБОТА 15. РЕШЕНИЕ ВЫЧИСЛИТЕЛЬНОЙ ЗАДАЧИ С ПРИМЕНЕНИЕМ ФАЙЛОВОГО ВВОДА-ВЫВОДА.

1. Цель и содержание

Цель работы: научиться использовать возможности файлового ввода-вывода для решения практических задач.

Задачи работы:

– научиться проектировать приложение для реализации хранения данных программы;

– научиться производить выбор оптимальных инструментов для обеспечения сериализации.

2. Теоретическая часть

Перед выполнением работы необходимо повторить теоретический материал работы №19.

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте консольное приложение.

2. Выполните индивидуальное задание. В каждом задании необходимо реализовать вычисление значений функции $f(\mathrm{x},\mathrm{y})$ с заданными шагами Δx и Δy на заданных диапазонах изменения независимых переменных $[x_0;x_1]$, $[y_0;\mathrm{y}_1]$.

3. Необходимо реализовать сохранение рассчитанных значений в файле.

Индивидуальное задание.

Вариант	Выражение для вычисления
1	$f(x,y)=a\cdot\sin(x)+b\cdot\cos(y)$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-5;5]$; $y\in[-5;5]$.
2	$f(x,y)=a\cdot x\cdot\lvert\sin(x+y)\rvert+b\cdot x\cdot\lvert\cos(y+x)\rvert$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-1;10]$; $y\in[-5;5]$.
3	$f(x,y)=a^2\cdot\sin^3(x+y)+\lvert b^3\cdot\cos^2(x+y)\rvert$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
4	$f(x,y)=a^2\cdot\sqrt{\lvert\sin(x)\rvert}+b\cdot\sqrt{\lvert\cos(y)\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
5	$f(x,y)=a\cdot\sqrt{\lvert\sin(x)+a\cdot x\rvert}+b\cdot\sqrt{\lvert\cos(y)+b\cdot y\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
6	$f(x,y)=a\cdot\sqrt{\lvert\sin(x)+a\cdot x\rvert}+b\cdot\sqrt{\lvert\cos(y)+b\cdot y\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
7	$f(x,y)=\sqrt{x^2+y^2-2xy}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
8	$f(x,y)=a\cdot\sin(x)+b\cdot\cos(y)$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-5;5]$; $y\in[-5;5]$.
9	$f(x,y)=a\cdot y\cdot\sin(x+y)+b\cdot x\cdot\cos(y+x)$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-5;5]$; $y\in[-5;5]$.
10	$f(x,y)=a\sqrt{\lvert x+y\rvert}+b\sqrt{\lvert x-y\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
11	$f(x,y)=a\cdot\sqrt{\lvert\sin(x)+a\cdot x\rvert}+b\cdot\sqrt{\lvert\cos(y)+b\cdot y\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
12	$f(x,y)=a\cdot\sqrt{\lvert\sin(x)+a\cdot x\rvert}+b\cdot\sqrt{\lvert\cos(y)+b\cdot y\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
13	$f(x,y)=\sqrt{x^2+y^2-2xy}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
14	$f(x,y)=a\cdot\sin(x)+b\cdot\cos(y)$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-5;5]$; $y\in[-5;5]$.
15	$f(x,y)=a\cdot x\cdot\lvert\sin(x+y)\rvert+b\cdot x\cdot\lvert\cos(y+x)\rvert$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-1;10]$; $y\in[-5;5]$.

16	$f(x,y)=a\cdot(x+y)\cdot\sin^2(x)+b\cdot(x-y)\cdot\cos^2(y)$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-5;5]$; $y\in[-5;5]$.
17	$f(x,y)=a^2\cdot x+b^3\cdot\cos^2(y+x)$; $\Delta x=0{,}05$; $\Delta y=0{,}05$; $x\in[-2;2]$; $y\in[-2;2]$.
18	$f(x,y)=a^2\cdot\sqrt{\lvert b\cdot\sin(x)\rvert}+b^2\cdot\sqrt{\lvert a\cdot\cos(y)\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
19	$f(x,y)=a\cdot x\cdot\lvert\sin(x+y)\rvert+b\cdot x\cdot\lvert\cos(y+x)\rvert$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-1;10]$; $y\in[-5;5]$.
20	$f(x,y)=a\cdot\sin(x)+b\cdot\cos(y)$; $\Delta x=0{,}1$; $\Delta y=0{,}1$; $x\in[-5;5]$; $y\in[-5;5]$.
21	$f(x,y)=a^2\cdot\sin^3(x+y)+b^3\cdot\cos^2(x+y)$; $\Delta x=0{,}05$; $\Delta y=0{,}05$; $x\in[-2;2]$; $y\in[-2;2]$.
22	$f(x,y)=a\cdot\sqrt{\lvert\sin(x)+a\cdot x\rvert}+b\cdot\sqrt{\lvert\cos(y)+b\cdot y\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
23	$f(x,y)=a\sqrt{\lvert x+y^2\rvert}+b\sqrt{\lvert x^2-y\rvert}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.
24	$f(x,y)=a^2\cdot x+b^3\cdot y-\lvert\sin(x)+\cos(y)\rvert$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-1;1]$; $y\in[-1;1]$.
25	$f(x,y)=\sqrt{x^2+y^2-2xy}$; $\Delta x=0{,}01$; $\Delta y=0{,}01$; $x\in[-2;2]$; $y\in[-2;2]$.

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое поток? Какие классы потоков вы знаете?
2. Какие классы пространства System.IO предназначены для работы с каталогами?
3. Какие классы пространства System.IO предназначены для работы с объектами файловой системы – файлами?
4. Как перенаправить стандартный консольный поток?
5. Поясните разницу между классами File и FileInfo?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [5].

ПРАКТИЧЕСКАЯ РАБОТА 16. СОЗДАНИЕ ПРИЛОЖЕНИЯ ПО ТЕХНОЛОГИИ WINDOWS FORMS

1. Цель и содержание

Цель работы: научиться создавать приложение Windows в среде MS Visual Studio и программировать алгоритмы с использованием простых элементов управления.

Задачами работы являются:

– разработка приложения с использованием Windows Forms,

– изучение основных членов класса Form;

– обработка событий от простых элементов управления.

2. Теоретическая часть

Файловая структура проекта MS Visual Studio в случае использования типа поекта Windows Form Application выглядит следующим образом (рис. 21.1):

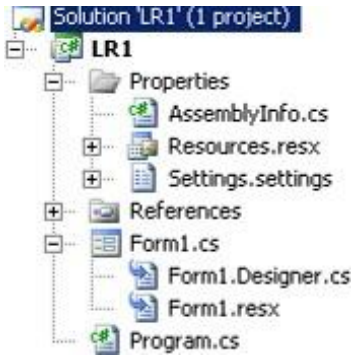


Рисунок 21.1 – Вид окна Solution Explorer для Windows Form Application.

Файл Program.cs содержит класс Program и статический метод Main(), с которого начинается выполнение приложения (рис. 21.2).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace LR1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Рисунок 21.2 – Листинг Program.cs.

В самом начале файла Program.cs выполняется объявление используемых пространств имен с использованием using.

Класс Program содержит метод Main(), который используя статический метод Run класса Application создает и выводит на экран главную форму приложения: Application.Run (new Form1()).

Таким образом, реализуется один из принципов объектно-ориентированного программирования: разграничение обязанностей (т.е. каждый класс выполняет минимально возможное количество операций).

Класс главной формы Form1 (по умолчанию) представлен двумя связанными C#-файлами. Для отображения содержимого Form1.cs необходимо щелкнутьправой кнопкой мыши в окне проектирования главной формы на самой форме или на пиктограмме Form1.cs в окне Solution Explorer (рис. 21.1) и в появившемся контекстном меню выбрать пункт «View Code».

Листинг Form1.cs представлен на рис 21.3.

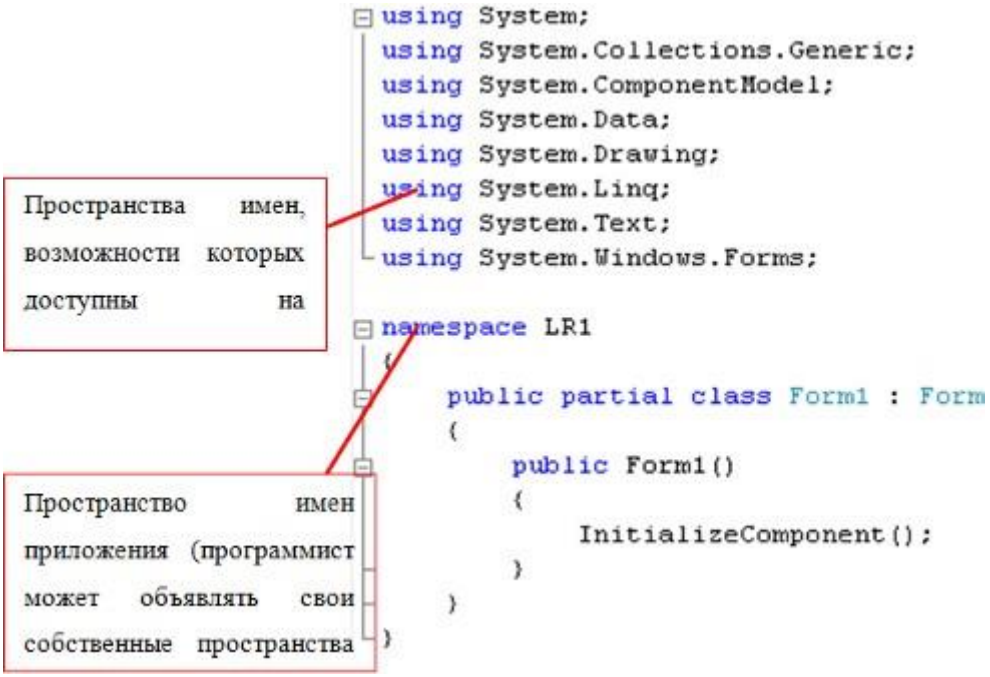


Рисунок 21.3 –Листинг Form1.cs.

Конструктор, созданный по умолчанию, вызывает метод InitializeComponent, который определен в соответствующем файле Form1.Designer.cs (рис. 21.1).

Этот метод создается автоматически, в нем отображаются все изменения, производимые программистом в окне визуального проектирования формы.

Пространства имен, доступные для использования, объявляются в начале файла.

System является базовым пространством имен – в него входят все остальные типы и пространство имен. Конструкция using System; в начале файла программы указывает на то, что весь программный код будет выполняться в данном пространстве имен, поэтому при использовании типов (например Int32), определенных в пространстве System нет необходимости указывать само имя пространства (то есть нет необходимости писать System.Int32).

Программист может самостоятельно вводить пространства имен при написании приложений.

Обработка событий в режиме проектирования. Окно проектирования главной формы приложения (рис. 21.4) можно открыть дважды щелкнув на пиктограмме Form1.cs в окне Solution Explorer (рис. 21.1). При этом откроется вкладка Form1.cs [Design].

Окно свойств примет вид, показанный на рисунке 21.4.

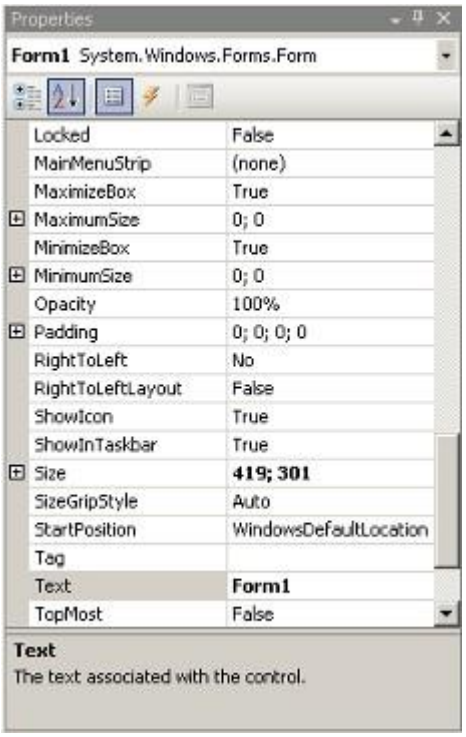


Рисунок 21.4 – Вид окна Свойств (Properties) в режиме проектирования формы.

Кнопка  позволяет отобразить список событий, характерных для элемента управления, активного в данный момент в окне проектирования. Например, если выделить главную форму приложения, то отобразятся события, характерные для класса Form.

Для отработки механизма регистрации событий, рассмотрим обработку наиболее простых событий – событий от мыши.

Для добавления обработчика события мыши необходимо выбрать соответствующее событие (Click, DoubleClick, MouseEnter, MouseLeave, MouseDown, MouseUp, MouseMove,. MouseHover, MeaseWheel) и дважды щелкнуть на поле, расположенном в соседнем от него столбце в окне свойств (таким образом обработчику присвоится имя по умолчанию). В данном поле можно ввести свое название для нового обработчика, или выбрать из выпадающего списка уже существующие обработчики.

После регистрации события в окне «Properties», автоматически добавляются строки кода в файлы Form1.Designer.cs (регистрируется обработчик события) и Form1.cs (добавляетсяметод, ассоциированный с данным событием).

Форма приложения должна быть функциональной и максимально эргономичной. Такие простые вещи, как цветовые комбинации, размеры шрифтов и окон могут сделать приложение намного более привлекательным для пользователя.

В случае если свойство Icon формы установлено, а для свойства ControlBox не указано значение false, то значок появится в левом верхнем углу формы. Обычно принято устанавливать здесь значок приложения.

Свойство FormBorderStyle задает тип рамки, которая появляется вокруг формы. Для этого используется перечисление FormBorderStyle. Допустимые значения этого перечисления: Fixed3D, FixedDialog, FixedSingle, FixedToolWindow, None, Sizable, SizableToolWindow.

Для выполнения работы потребуется основная информация о событиях мыши:

Все обработчики событий мыши принимают параметр типа MouseEventArgs. Поступающий на вход обработчика событий мыши объект типа MouseEventArgs позволяет получить дополнительную информацию о действии мыши, путем введения ряда специальных членов класса (табл. 21.1).

Таблица 21.1 – Свойства типа MouseEventArgs

Свойство	Описание
Button	Содержит информацию о том, какая клавиша была нажата, в соответствии с определением перечня MouseButtons
Clicks	Содержит информацию о том, сколько раз была нажата и отпущена клавиша мыши
Delta	Содержит информацию со знаком, соответствующее числу щелчков, произошедших при вращении колесика мыши
X	Содержит информацию о координате X
Y	Содержит информацию о координате Y

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального

компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

1. Создайте в среде разработки MS VS проект. В качестве типа проекта выбрать «Windows Application» (или «Windows Forms Application» в зависимости от версии .NET Framework)

2. После создания и сохранения проекта измените программу таким образом, чтобы координаты курсора мыши выводились в заголовке главного окна приложения.

Для этого в обработчик события движения мыши над оконной формой (MouseMove) добавьте строку кода:

Text = string.Format("Координаты:{0}, {1}", e.X, e.Y);

3. Добавьте текстовое поле (TextBox) в режиме разработки (для этого необходимо использовать панель элементов управления «ToolBox»). Дополните обработчик движения мыши таким образом, чтобы в текстовом поле отображалась сумма координат указателя мыши.

4. Работающую программу необходимо представить преподавателю.

5. После этого выполните индивидуальное задание в соответствие с вариантом. В каждом задании необходимо вывести значение выражения, предварительно введя значения переменных в соответствующие текстовые поля формы главного окна приложения. Результат выводится в заголовок окна в ответ на нажатие кнопки (кнопку также необходимо поместить на форму).

6. Для выполнения индивидуального задания необходимо использовать математические функции, которые доступны в виде статических методов класса Math.

Индивидуальное задание.

Перед выполнением задания требуется самостоятельно определить закономерность изменения членов последовательности, чтобы применить цикл, условный оператор или, если потребуется, оператор выбора.

Вариант	Выражение для вычисления
1	$Z = \sqrt{\left\|\frac{a \cdot x}{w^a}\right\|} + \sqrt{\|b\|} - \|x + \cos(y)\|$
2	$F = \sqrt{\|d2\|} \cdot x + \sqrt{\|b^3\|} - \left\|x^2 + \cos(y)\right\|$
3	$A = -d \cdot \frac{z}{\sqrt{\|e\|}} + \|\sin(e) + \cos(y)\|$
4	$U = \sqrt{\left\|\frac{f-e}{w}\right\|} + \left\|\sin^2(\frac{e}{w}) + \cos(y)\right\|$
5	$t = \lg(f) - e + \left\|\sin(\frac{w}{t}) + \sqrt{\|e\|}\right\|$
6	$V = \frac{h-e}{q+a} + \left\|\sin(f) + \sqrt{\|\sin(b)\|}\right\|$
7	$Q = \sin(h + \frac{d}{e^{o1}}) - o1 + \left\|\sin(f) + \sqrt{\|\sin(b)\|}\right\|$
8	$E = \sin(h11 + \frac{d12}{\ln(h11)}) - v6 + \left\|\sin(h3) + \sqrt{\|\sin(f)\|}\right\|$
9	$Z = z \cdot ax1 + \sqrt{\left\|e7 + \frac{x}{y}\right\|} - \|x + \cos(as\_9 + y)\|$
10	$RES = \frac{\arg01 - \arg02 + \left\|\sin(\arg03) + \sqrt{\left\|\arg04\right\|}\right\|}{\arg01^{\arg 5} - \lg(\arg 02)}$
11	$U = \frac{v - e\_2 + \left\|\cos(\arg3) + \sqrt{\left\|t\right\|}\right\|}{a\_del * tg(t)}$
12	$t = W + \frac{\cos(g \cdot q)}{W} - e + \left\|\sin(e) + \sqrt{\|o\|}\right\|$
13	$Z = z \cdot \cos(y) + \sqrt{e7} - \|x + \cos(as\_9 + y)\|$
14	$R = q + \left\|\sin^2(e) + \cos(y)\right\| \cdot \cos(s + g)$
15	$F = \sin(\frac{v \cdot x}{y}) \cdot x + \sqrt{\|b^3\|} - \left\|x^2 + \cos(y)\right\|$

16	$res = \frac{F}{\cos(w)} - e + \frac{\left\lvert \sin(\frac{w}{t}) + \sqrt{\lvert e \rvert} \right\rvert}{e}$
17	$t = W + \frac{tg(g \cdot q)}{\ln(20 \cdot x)} - \frac{e}{W} + \left\lvert \sin(e) + \sqrt{\lvert W - e \rvert} \right\rvert$
18	$t = W + \frac{tg(g + q)}{\ln(20 \cdot x)} - \frac{e}{W} + \left\lvert \sin(e) + \sqrt{\lvert W - e \rvert} \right\rvert$
19	$F = \sqrt{\lvert d2 - x \rvert} \cdot x + \sqrt{\lvert b^3 \rvert} - \left\lvert x^2 + \cos(y) \right\rvert$
20	$t = W + \frac{\cos(g \cdot q)}{W} - e + \left\lvert \sin(e) + \sqrt{\left\lvert \frac{o}{e^4} \right\rvert} \right\rvert$
21	$U = \frac{v^3 - e\_2 + \left\lvert \cos(\arg 3) + \sqrt{\lvert t \rvert} \right\rvert}{a\_del * tg(t)}$
22	$F = \sin(\frac{v \cdot x}{y}) \cdot x + \sqrt{\lvert b^3 \rvert} - \left\lvert x^2 + \cos(y) \right\rvert$
23	$v = \sqrt[4]{e\_2} \cdot \sqrt{\frac{\sin(12 * \arg 3)}{e\_2 - 1}} + \left\lvert \cos(\arg 3) + \sqrt{\lvert t \rvert} \right\rvert$
24	$res = \frac{F^{\lvert e \rvert}}{\cos(w + e)} - e + \frac{\left\lvert \sin(\frac{w}{t}) \right\rvert}{25 + \sqrt{\lvert e \rvert}}$
25	$t = \frac{az}{\sqrt{g^3}} + \frac{\cos(g \cdot q)}{az} - g + \left\lvert \sin(q) + \sqrt{\lvert 12 - q \rvert} \right\rvert$

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Какие файлы описывают класс формы?
2. Какие действия необходимо выполнить для создания обработчика события?
3. Где описывается код обработчика события? В каком файле регистрируется обработчик события (метод привязывается к событию)?
4. Как получить доступ к координатам курсора мыши?
5. Какой класс содержит методы, реализующие математические функции?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [6].

ПРАКТИЧЕСКАЯ РАБОТА 17. ПРИМЕНЕНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ В ПРИЛОЖЕНИЯХ WINDOWS

1. Цель и содержание

Цель работы: изучить принципы использования стандартных элементов управления в приложениях Windows.

Задачи работы:

– научиться использовать кнопки, фражки, переключатели;

– научиться использовать списки, выпадающие списки;

– научиться .

2. Теоретическая часть

Понимание иерархии классов элементов управления Windows имеет важное значение при проектировании интерфейсов пользователя, особенно при разработке собственных элементов управления.

В пространстве имен System.Windoiws .Forms есть один особенный класс, служащий базовым для почти каждого создаваемого элемента управления и формы. Это System.Windows.Forms.Control. Класс Control реализует базовую

функциональность создания и отображения всего, что видит пользователь. Класс Control унаследован от класса System.ComponentModel.Component. Класс Component обеспечивает Control всей необходимой инфраструктурой, которая потребуется для того, чтобы его можно было перетаскивать на поверхность проектирования в визуальном конструкторе и помещать в другой объект. Класс Control предоставляет огромный список функциональности классам, унаследованным от него.

Размер и местоположение элемента управления определяется свойствами Height, Width, Top, Bottom, Left и Right, наряду с дополняющими их свойствами Size и Location. Отличие между ними в том, что свойства Height, Width, Top, Bottom, Left и Right принимают целочисленные значения, Size – структуру Size, а Location – структуру Point. Структуры Srze и Point содержат версии координат X,Y. Структура Point обычно связана с местоположением, а Size задает высоту и широту объекта. Size и Point определены в пространстве имен System. Drawing. Обе структуры очень похожи в том отношении, что представляют пару координат X,Y, но также имеют перегруженные операции для облегчения сравнения и преобразования. Так, например, две структуры Size можно складывать вместе. В структуре Point операция Addition переопределена так, что можно прибавлять структуру Size к Point и получать новое значение Point. Это обеспечивает эффект прибавления расстояния к местоположению с получением нового местоположения, что очень удобно, если нужно динамически создавать формы и элементы управления.

Свойство Bounds возвращает объект Rectangle, представляющий область элемента управления. Эта область включает линейки прокрутки и панель заголовка. Rectangle – также часть пространства имен System. Drawing. Свойство ClientSize – это структура Size, представляющая клиентскую область элемента управления, минус линейки прокрутки и панель заголовка.

PointToClient и PointToScreen – методы преобразования, которые принимают Point и возвращают Point. Метод PointToClient принимает Point, представляющую экранные координаты, и транслирует их в координаты,

основанные на текущем клиентском объекте. Это удобно для операций перетаскивания. Метод PointToScreen выполняет прямо противоположную операцию – принимает координаты клиентского объекта и транслирует их в экранные координаты. Методы RectangleToScreen и ScreenToRectangle выполняют те же функции со структурами Rectangle вместо Point.

Свойство Anchor прикрепляет край элемента управления к краю родительского элемента. Это отличается от стыковки тем, что не устанавливает грань по родительскому элементу, а устанавливает некоторую постоянную дистанцию от края. Например, если после прикрепления правого края элемента управления к правому краю родительского элемента размер родительского элемента будет изменен, правый край дочернего элемента останется на том же расстоянии от правого края родительского элемента. Свойство Anchor принимает значение типа перечисления AnchorStyle. Перечисление включает следующие значения: Top, Bottom, Left, Right и None.

С внешним видом элемента управления связаны свойства BackColor и ForeColor, которые принимают значения типа System. Drawing.Color. Свойство BackGroundlmage принимает в качестве значения объект на базе Image. Класс System. Drawing. Image – это абстрактный класс, используемый в качестве базового для классов Bitmap и Metafile.

Свойство BackgroundlmageLayout использует перечисление ImageLayout для установки режима вывода графического изображения в элементе управления. Допустимыми значениями являются Center,Tile, Stretch, Zoom и None.

Свойства Font и Text имеют дело с отображением текста. Чтобы изменить Font, потребуется создать объект Font. При создании объекта Font указывается имя шрифта, размер и стиль.

Для выполнения работы необходимо изучить теоретические основы проектирования с использованием стандартных элементов управления библиотеки .NET Framework.

Класс. Button. Класс Button представляет простую командную кнопку и наследуется от класса ButtonBase. Наиболее часто приходится писать код для обработки события Click кнопки. В следующем фрагменте кода реализован обработчик события Click (обработка данного события не представляет сложностей). С помощью метода PerformClick можно эмулировать событие Click на кнопке без реального щелчка на ней пользователем, что может пригодиться для тестирования построенного пользовательского интерфейса. В окне также имеется понятие кнопки по умолчанию, на которой автоматически совершается щелчок, если пользователь нажимает в этом окне клавишу «Enter». Чтобы идентифицировать кнопку как кнопку по умолчанию, в форме, содержащей эту кнопку, необходимо установить свойство AcceptButton в объект кнопки.

Класс CheckBox (рис. 22.1). Элемент управления CheckBox также унаследован от ButtonBase и используется для приема двух или трех состояний от пользователя.

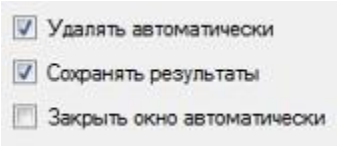


Рисунок 22.1 – Внешний вид CheckBox.

Если установить свойство ThreeState в true, то свойство CheckState элемента CheckBox сможет принимать одно их трех значений перечисления CheckState, описанных в табл. 22.1.

Таблица 22.1 – Состоятия CheckBox (возможные значения перечисления CheckState).

Значение	Описание
Checked	Флажок имеет отметку.
Unchecked	Флажок не имеет отметки.
Indeterminate	В этом состоянии флажок становится серым (неопределенное состояние).

Состояние Indeterminate удобно, когда пользователь должен быть уведомлен, что опция не установлена. Если нужно булевское значение, можно проверить свойство Checked.

События CheckedChanged и CheckStateChanged возникают, когда изменяются значения свойств CheckState и Checked. Для флажка с тремя состояниями понадобится присоединиться к событию CheckStateChanged. Перехват этих событий может быть полезен для установки других значений, основанных на новом состоянии CheckBox.

Класс RadioButton (рис. 22.2). Переключатель (кнопки опций) – элемент управления, унаследованный от ButtonBase.

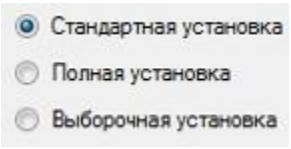


Рисунок 22.2 – Внешний вид RadioButton.

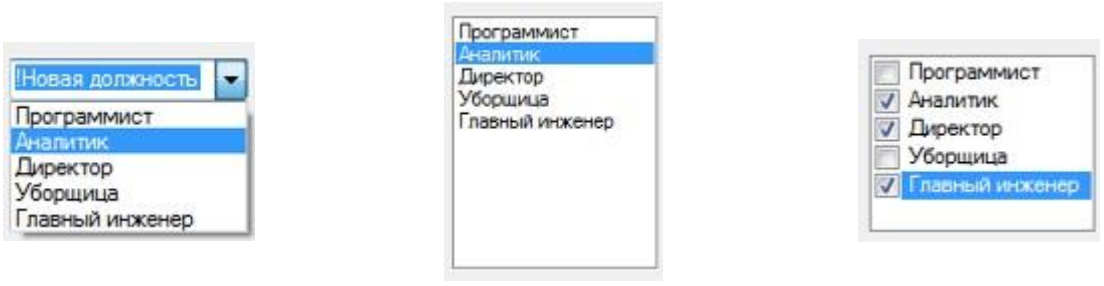
Переключатели обычно используются в группе. Переключатели позволяют пользователю выбирать одну из нескольких опций. При наличии нескольких элементов управления RadioButton в одном контейнере, только один из них может быть выбран в один и тот же момент времени.

Свойство Appearance принимает значение перечисления Appearance, которым может быть Button либо Normal. В случае установки значения Normal переключатель выглядит как маленький кружочек с меткой рядом с ним. Выбор переключателя заполняет этот кружок, а выбор другого переключателя снимает отметку с текущего выбранного переключателя и кружок делается пустым. В случае установки значения Button элемент управления выглядит подобно стандартной кнопке, но работает как переключатель – его выбор означает включение (вдавливание кнопки), а отказ от выбора – выключение (стандартное положение кнопки).

Свойство CheckedAlign определяет местоположение кружка по отношению к тексту метки. Он может быть над меткой, с любой стороны от нее либо под меткой.

Событие CheckedChanged инициируется при любом изменении свойства Checked. Это позволяет предпринимать другие действия на основе нового значения элемента управления.

Классы ComboBox, ListBox и CheckedListBox (рис. 22.3). Элементы управления ComboBox, ListBox и CheckedListBox унаследованы от класса ListControl. Этот класс предоставляет некоторую базовую функциональность управления списками. Наиболее важные аспекты использования списочных элементов управления состоят в добавлении данных и выборе данных из списка.



а) б) в)

Рисунок 22.3 – Внешний вид списков: а – ComboBox; б – ListBox; в – CheckedListBox.

На выбор списка влияет то, как он должен использоваться, и тип данных, которые будут помещены в список. Если есть необходимость во множественном выборе, или если пользователю нужно видеть несколько элементов в списке в одно и то же время, то для этого лучше всего подойдут ListBox и CheckedListBox. Если в любой момент времени в списке может быть выбран только один элемент, то предпочесть следует ComboBox.

Для добавления элементов в списки необходимо обратиться к коллекции ListBox.ObjectCollection, которая доступна через свойство Items списка.

Поскольку коллекция хранит объекты, в список может быть добавлен любой допустимый тип .NET. Чтобы идентифицировать элементы, необходимо установить два важных свойства. Первым свойством является DisplayMember. Его настройка сообщает ListControl, какое свойство объекта должно быть отображено в списке. Другое свойство – ValueMember. Оно представляет собой свойство объекта, которое должно возвращаться в качестве значения. Если в список были добавлены строки, для обоих свойств по умолчанию используются строковые значения.

После загрузки данных в список для их получения могут быть использованы свойства Selectedltem и Selectedlndex. Свойство Selectedltem возвращает объект, выбранный в данный момент. Если список предполагает множественный выбор, то должна применяться коллекция Selectedltems для получения выбранных элементов.

Если для наполнения списка применяется DataBinding, то свойство SelectedValue вернет значение свойства выбранного объекта, которое было установлено в свойстве ValueMember.

Элемент ComboBox – это комбинация поля редактирования и окна списка. Стиль ComboBox задается установкой в свойстве DropDownStyle значения перечисления DropDownStyle (таблица 22.2).

Таблица 22.2 – Стили ComboBox (возможные значения перечисления DropDownStyle).

Значение	Описание
DropDown	Текстовая часть комбинированного списка допускает редактирование, и пользователи могут вводить значение. Также они могут щелкать на кнопке со стрелочкой, чтобы отобразить раскрывающийся список элементов.
DropDownList	Текстовая часть не допускает редактирование. Пользователи должны выбирать значения из списка.
Simple	Подобно DropDown, но список является постоянно видимым.

Класс DateTimePicker (рис. 22.4). Элемент управления DateTimePicker позволяет выбирать значение даты или времени (или то и другое) во множестве разных форматов. Значение DateTime можно отобразить в любом стандартном формате времени и даты. Свойство Format принимает значение перечисления DateTimePickerFormat, которое устанавливает формат Long, Short, Time или Custom. Если свойство Format установлено в DateTimePickerFormat .Custom, с помощью свойства CustomFormat можно задать строку, представляющую формат.

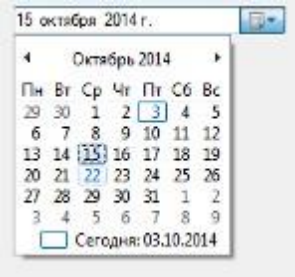


Рисунок 22.4 – Внешний вид DateTimePicker.

Свойство Text возвращает строковое представление значения DateTime, а свойство Value – объект DateTime. С помощью свойств MinDate и MaxDate можно устанавливать максимальное и минимальное значения дат. Когда пользователь щелкает на стрелке вниз, отображается календарь, позволяющий выбрать дату. Доступны свойства, которые позволяют изменить внешний вид календаря, указать фоновые цвета заголовка и месяца, а также цвета переднего плана.

3. Оборудование и материалы

Для выполнения работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб ивыше, свободное дисковое пространство – не менее 1 Гб, графическое

устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

4. Указания по технике безопасности

Техника безопасности при выполнении работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Методика и порядок выполнения работы

Для выполнения работы необходимо выполнить следующую последовательность действий:

1. Создайте проект приложения Windows Forms в среде MS Visual Studio.
2. Реализуйте методы для вычисления выражений в соответствии с вариантом индивидуального задания.
3. При проектировании формы необходимо учитывать следующие особенности задачи:

– выражение может быть вычислено двумя различными способами, то есть пользователь должен иметь возможность выбрать метод расчета (подумайте, какой элемент управления подходит для этого больше всего?);

– в правой части выражения присутствуют неизвестные переменные, которые пользователь может вводить с клавиатуры в текстовые поля;

– в некоторых выражениях присутствуют коэффициенты, выбор значений которых необходимо реализовать с помощью различных списков.

Индивидуальное задание.

Вариант	Выражение для вычисления
1	$Z = \begin{cases} 1 - \frac{X}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{3 + a \cdot j}{b \cdot i}. \end{cases}$
2	$Z = \begin{cases} -\frac{1}{2} + \frac{1}{6} - \frac{X}{24} + \frac{Y}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + j}{i}. \end{cases}$
3	$Z = \begin{cases} -\frac{1}{1} + \frac{Y}{2} - \frac{X}{3} + \frac{Y}{4} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{j^2 + h \cdot i}{i + c \cdot j}. \end{cases}$
4	$Z = \begin{cases} -\frac{2}{2} + \frac{\quad}{6} - \frac{\quad}{24} + \frac{\quad}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i + b \cdot j}{c \cdot i}. \end{cases}$
5	$Z = \begin{cases} -\frac{p}{2} + \frac{p}{6} - \frac{p}{24} + \frac{p}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + b \cdot j}{c \cdot i} \end{cases}$
6	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + a \cdot j^2}{i + j} \end{cases}$

7	$Z = \begin{cases} -\frac{X}{2} + \frac{X^2}{6} - \frac{X^3}{24} + \frac{X^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i^2}{\sqrt[3]{i^3} + b \cdot j}. \end{cases}$
8	$Z = \begin{cases} -\frac{1}{2} + \frac{X}{6} - \frac{X^2}{24} + \frac{X^3}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i}{\sqrt[3]{i^3} + j}. \end{cases}$
9	$Z = \begin{cases} -\frac{f}{2} + \frac{X \cdot f^2}{6} - \frac{X^2 \cdot f^3}{24} + \frac{X^3 \cdot f^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot (i+j)}{i \cdot (i+2)}. \end{cases}$
10	$Z = \begin{cases} 1 - \frac{X}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{3+j}{c \cdot i}. \end{cases}$
11	$Z = \begin{cases} -\frac{1}{2} + \frac{Y}{6} - \frac{X}{24} + \frac{Y}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + j}{\sqrt{j}} \cdot \\ a \end{cases}$
12	$Z = \begin{cases} -\frac{1}{1} + \frac{Y}{2} - \frac{X}{\sqrt{3}} + \frac{Y}{\sqrt{4}} - \\ ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{j^2 + c \cdot i}{\sqrt{j}}. \end{cases}$
13	$Z = \begin{cases} -\frac{}{2} + \frac{}{6} - \frac{}{24} + \frac{}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i + b \cdot j}{\sqrt{j}}. \\ c \cdot i \end{cases}$
14	$Z = \begin{cases} -\frac{p}{2} + \frac{p}{6} - \frac{p}{24} + \frac{p}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + b \cdot j}{\sqrt[3]{c \cdot i}} \cdot \end{cases}$

15	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - ..;. \\ \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{i^2 + j^2}{\sqrt[3]{i^3 + j}}. \end{cases}$
16	$Z = \begin{cases} -\frac{X}{2} + \frac{2 \cdot X^2}{6} - \frac{3 \cdot X^3}{24} + \frac{4 \cdot X^4}{120} - ..;. \\ \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{(i+1)^2}{\sqrt[3]{a \cdot i^3 + b \cdot j}}. \end{cases}$
17	$Z = \begin{cases} -\frac{1}{2} + \frac{X}{6} - \frac{X^2}{24} + \frac{X^3}{120} - ..;. \\ \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{a \cdot i}{\sqrt[3]{i^3 + b \cdot j}}. \end{cases}$
18	$Z = \begin{cases} -\frac{f}{2} + \frac{X \cdot f^2}{6} - \frac{X^2 \cdot f^3}{24} + \frac{X^3 \cdot f^4}{120} - ..;. \\ \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{a \cdot (i + j)}{b \cdot i \cdot (i + 2)}. \end{cases}$
19	$Z = \begin{cases} -\frac{1}{1} + \frac{Y}{3} - \frac{X^2}{\sqrt{5}} + \frac{Y^3}{\sqrt{7}} - ..;. \\ \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{a \cdot j^2 + 2}{\sqrt[j]{b \cdot i^3 + 3}}. \end{cases}$
20	$Z = \begin{cases} -\frac{1}{2} + \frac{X}{6} - \frac{X^2}{24} + \frac{X^3}{120} - ..;. \\ \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{a \cdot i + b \cdot j}{\sqrt[j]{c \cdot i}}. \end{cases}$
21	$Z = \begin{cases} -\frac{p}{1 \cdot 2} + \frac{p}{2 \cdot 6} - \frac{p}{3 \cdot 24} + \frac{p}{4 \cdot 120} - ..;. \\ \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{i^2 + d \cdot j}{\sqrt[i]{c \cdot i^3}}. \end{cases}$
22	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - ..;. \\ \sum_{i=1}^{N} \sum_{j=1}^{R} \frac{i^2 + j^2}{\sqrt[3]{i}}. \end{cases}$

23	$Z=\begin{cases} -\frac{3\cdot Y}{2}+\frac{5\cdot Y^2}{6}-\frac{7\cdot X^3}{24}+\frac{9\cdot Y^4}{120}-..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i+k\cdot j}{c\cdot i^{j}}. \end{cases}$
24	$Z=\begin{cases} -\frac{3\cdot p^2}{2}+\frac{5\cdot p^3}{6}-\frac{7\cdot p^4}{24}+\frac{9\cdot p^5}{120}-..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i^2+b}{c^{\,i}\cdot(i+1)^3}. \end{cases}$
25	$Z=\begin{cases} -\frac{X}{2}+\frac{p\cdot X^2}{6}-\frac{p^2\cdot X^3}{24}+\frac{p^3\cdot X^4}{120}-..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{(a\cdot i+j)^2}{\sqrt[3]{i^{\,3}+b\cdot j}}. \end{cases}$

7. Содержание отчета и его форма

Отчет по работе должен содержать:

1. Номер и название работы.

2. Цели работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Какое событие элемента управления Button обрабатывается в программах чаще всего?

2. Для чего предназначен компонент CheckBox? Назовите основные свойства класса CheckBox.

3. Опишите назначение элемента RadioButton. Какой внешний вид может

принимать данный компонент? Назовите основные свойства класса RadioButton.

4. Назовите классы компонентов для представления списочной информации.

5. Опишите основные свойства класса ListBox. Чем комопнет ListBox отличается от CheckedListBox?

9. Список литературы

Для выполнения работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [6-8].

ЗАКЛЮЧЕНИЕ

В учебном пособии представлены методические указанияк лабораторным работам, которые помогают студентам освоить программирование как род деятельности. Учебное пособие (лабораторный практикум) по дисциплине «Основы алгоритмизации и программирования» предоставляют студентам необходимый теоретический материал, а также позволяют получить практические навыки для дальнейшего самообучения, развития своих навыков.

Материал лабораторных работ представлен на базе современной среды разработки программных средств – интегрированной среды разработки Microsoft Visual Studio. Технологии, доступные разработчику с использованием данной IDE, непрерывно совершенствуются и требуют от программиста непрерывного повышения профессионального уровня. Полный спектр возможностй IDE MS VS выходит далеко за рамки данного учебного пособия, но выполнение заданий лабораторного практикума обеспечивает студентов знаниями, востребованными на современном рынке труда.

В качестве основы построения лабораторного практикума предложена технология MS .NET Framework. Данная технология предлагает широкий спектр алгоритмов и технологий разработки программного обеспечения: широкий набор библиотечных типов; специализированные синтаксические конструкции; современные примитивы программирования; компоненты разработки графических приложений; классы для работы с объектами файловой системы. Все представленные технологии в рамках одного языка позволяют студентам получить практический опыт использования современных инструментов разработки программ.

СПИСОК ЛИТЕРАТУРЫ

1. Рихтер. Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Дж. Рихтер. – СПб.: Питер, 2014. – 896 с.

2. Петцольд Ч. Программирование для Microsoft Windows 8. 6-е изд. / Ч. Петцольд. – СПб.: Питер. 2013 – 1008 с.

3. Хейлсберг А. Язык программирования C#. Классика Computers Science. 4-е изд. / А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. – СПб.: Питер. 2011. – 784 с.

4. Стиллмен Э. Изучаем C#. 3-е изд. / Э. Стиллмен, Дж. Грин. – СПб.: Питер. 2014. – 816 с.

5. Язык программирования C# 5.0 и платформа .NET 4.5. 6-е изд. Пер. с англ. / Э. Троелсен. – М.:Изд. «Издательский дом Вильямс». 2013. – 1312 с.

6. Албахари Дж. C# 5.0. Справочник. Полное описание языка / Дж. Албахари, Б. Албахари. – М.:Изд. «Издательский дом Вильямс». 2013. – 1008 с.

7. Флентов, М. Библия C#. 2-е изд. / М. Флентов. – СПб.: БХВ-Петербург, 2011. – 560 с.

8. Ватсон, Б. C# 4.0 на примерах: пер. с англ. / Б. Ватсон. – СПб.: БХВ-Петербург, 2011. – 608 с.