МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Методические указания к лабораторным работам для студентов 1-го курса дневного и заочного отделения факультета «Радиотехники и электроники», направления 11.03.03 – «Конструирование и технология электронных средств»

Новосибирск 2015

Составитель: старший преподаватель кафедры КТРС, А.А. Бизяев, к.т.н., доцент кафедры КТРС, К.А. Куратов
Рецензент: к.т.н., доц. каф. ТОР, Ю.В. Морозов

Работа подготовлена на кафедре конструирования и технологии радиоэлектронных средств

Аннотация. Методические указания к выполнению лабораторных позволяют студенту получить практические навыки работы с современными технологиями в области создания программных продуктов и укрепить навыки в области линейного, структурного и объектно-ориентированного программирования.

Оглавление

ЛАБОРАТОРНАЯ РАБОТА №1. ТИПЫ ДАННЫХ, ОПЕРАТОРЫ C#

Цель работы

Получить основы навыков программирования в среде Microsoft Visual Studio.

Задачи

Познакомиться с интегрированной средой визуальной разработки программного обеспечения Microsoft Visual Studio. Изучить основы построения платформы .Net, получить практические навыки программирования на языке C#.

Общие сведения

Система визуального программирования Microsoft Visual Studio

Интегрированная среда разработки IDE (Integrated Development Environment) Microsoft Visual Studio обеспечивает возможность использования всех преимуществ современной технологии Microsoft .Net. В числе основных достоинств Microsoft Visual Studio можно отметить следующие моменты:

- **Высокая производительности труда разработчиков.** Среда разработки Microsoft Visual Studio ориентирована на предоставление эффективных инструментальных средств для разработчиков сложного программного обеспечения. Обеспечивая среду разработки различными языками программирования, дополненную набором окон с интуитивно понятными инструментальными средствами, контекстной справкой и автоматизированными механизмами выполнения разнообразных задач разработки, Microsoft Visual Studio позволяет в сжатые сроки проводить профессиональную разработку программ различного назначения;
- **Поддержка нескольких языков программирования.** Большинство профессиональных разработчиков пользуются различными языками программирования в зависимости от поставленной задачи. Благодаря применению общего конструктора для компонентов, для форматов XML и

HTML, а также наличию единого отладчика, Microsoft Visual Studio предоставляет разработчикам эффективные средства, независимые от языка программирования. Разработчикам ПО при использовании Microsoft Visual Studio уже не придется ограничиваться одним языком программирования, адаптируя свою рабочую среду к особенностям этого языка. Более того, Microsoft Visual Studio позволяет программистам многократно использовать уже имеющиеся у них наработки, а также навыки разработчиков, создающих свои программы на разных языках программирования;

- **Единая модель программирования для всех приложений.** Раньше разработчикам приходилось пользоваться различными сложными приемами программирования, которые зависели от типа приложения, общедоступных веб-приложений, программного обеспечения для мобильных устройств и бизнес-логики промежуточного уровня значительно различались между собой. Среда разработки Microsoft Visual Studio решает данную проблему, предоставляя в распоряжение разработчиков единую модель создания приложений всех категорий.
- **Всесторонняя поддержка жизненного цикла разработки.** Среда Microsoft Visual Studio обеспечивает поддержку всего жизненного цикла разработки: начиная с планирования и проектирования через разработку и тестирование вплоть до развертывания и последующего управления..

Платформа – среда, обеспечивающая выполнение программного кода. Платформа определяется характеристиками процессоров, особенностями операционных систем.

Framework – это инфраструктура среды выполнения программ, определяющее особенности разработки и выполнения программного кода на данной платформе. Framework представляет средства организации взаимодействия с операционной системой и прикладными программами, методы доступа к базам данных, средства поддержки распределенных (сетевых)

приложений, языки программирования, множества базовых классов, унифицированные интерфейсы пользователя, парадигмы программирования.

CLS (Common Language Specification) – общая спецификация языков программирования. Это набор конструкций и ограничений, которые являются руководством для создателей библиотек и компиляторов в среде .NET Framework. Библиотеки, построенные в соответствии с CLS, могут быть использованы из любого языка программирования, поддерживающего CLS. Языки, соответствующие CLS (к их числу относятся языки Visual C#, Visual Basic, Visual C++), могут интегрироваться друг с другом. CLS – это основа межязыкового взаимодействия в рамках платформы Microsoft .NET.

CLR (Common Language Runtime) – среда времени выполнения, которая обеспечивает выполнение сборки. Если кратко, то CLR – это набор служб, необходимых для выполнения управляемого кода.

MSIL (Microsoft Intermediate Language) – промежуточный язык платформы Microsoft .NET. Исходные тексты программ для .NET-приложений пишутся на языках программирования, соответствующих спецификации CLS. Для таких языков может быть построен преобразователь в MSIL. Таким образом, программы на этих языках могут транслироваться в промежуточный код на MSIL. Благодаря соответствию CLS, в результате трансляции программного кода, написанного на разных языках, получается совместимый IL-код.

Языки, для которых реализован перевод на MSIL: Visual Basic; Visual C++; Visual C#, F#;

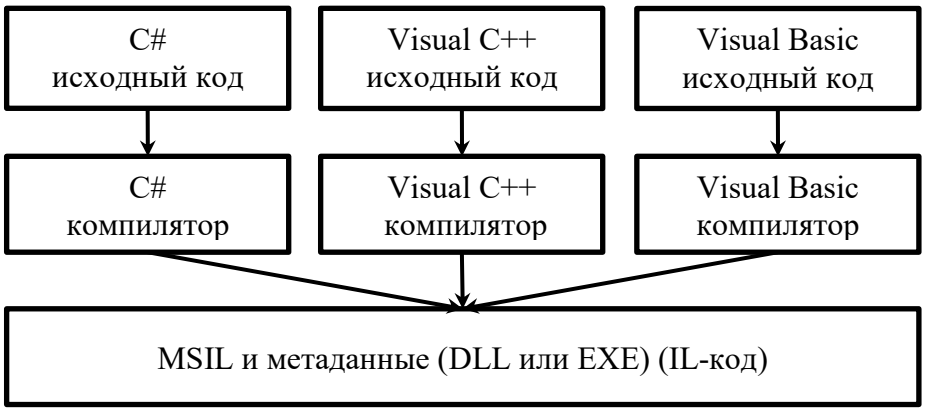


Рис. 1 – Схема трансляции кодов в MSIL

Процессор не может выполнять IL-код. Перевод IL-кода осуществляется JIT-компилятором (Just In Time – в нужный момент), который активизируется CLR по мере необходимости и выполняется процессором. При этом результаты деятельности JIT-компилятора сохраняются в оперативной памяти. Между фрагментом оттранслированного IL-кода и соответствующим блоком памяти устанавливается соответствие, которое в дальнейшем позволяет CLR передавать управление командам процессора, записанным в этом блоке памяти, минуя повторное обращение к JIT-компилятору.

Для CLR все сборки одинаковы, независимо от того, на каких языках программирования они были написаны. Главное – это чтобы они соответствовали CLS. Фактически CLR разрушает границы языков программирования (cross-language interoperability). Таким образом, благодаря CLS и CTS, .NET-приложения оказываются приложениями на MSIL (IL).

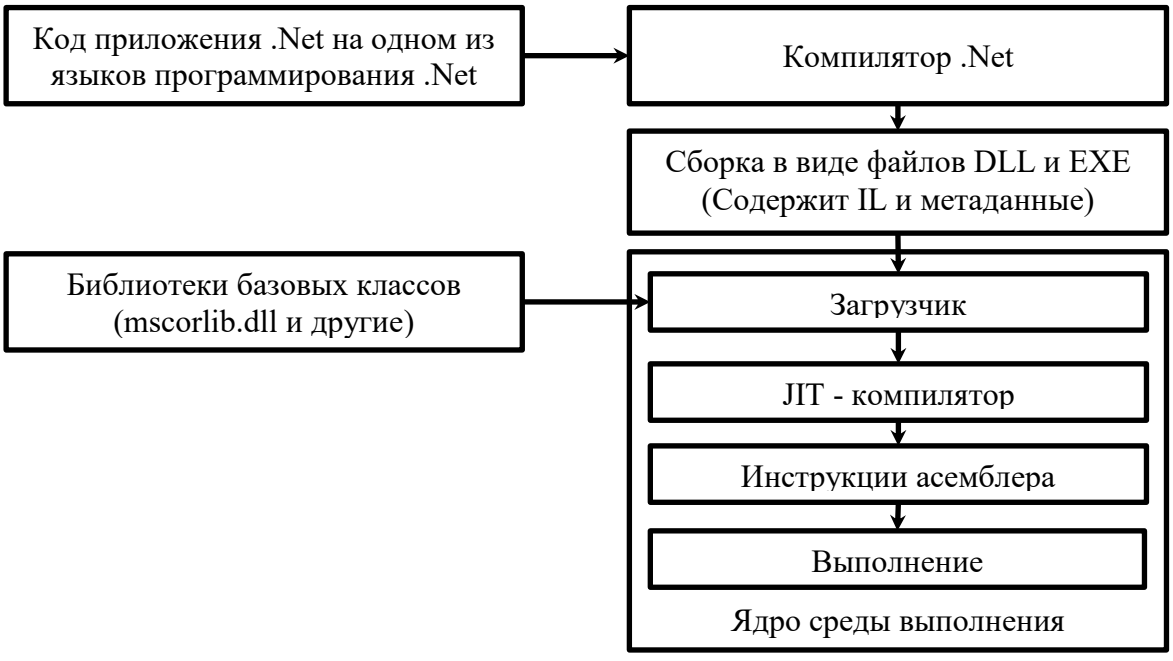


Рис. 2 – Схема выполнения .Net приложения в среде CLR

CTS – Common Type System (Стандартная Система Типов), поддерживаемая всеми языками платформы. Технология .NET основана на парадигме ООП (объектно-ориентированного программирования), поэтому основные типы данных, это классы, структуры, интерфейсы, делегаты и

перечисления. Common Type System является важной частью среды выполнения, определяет структуру синтаксических конструкций, способы объявления, использования и применения общих типов среды выполнения. В CTS сосредоточена основная информация о системе общих предопределенных типов, об их использовании и управлении (правилах преобразования значений).

Пространство имен – это способ организации системы типов в единую группу. Концепция пространства имен обеспечивает эффективную организацию и навигацию по этой библиотеке. Вне зависимости от языка программирования, доступ к определенным классам обеспечивается за счет их группировки в рамках общих пространств имен.

```
System.Data          // Классы для обращения к базам данных
System.Collections   // Классы для работы с контейнерными объектами
System.Diagnostics   // Классы для трассировки и отладки кода
System.Drawing       // Классы графической поддержки
System.IO            // Поддержка ввода/вывода
System.Net           // Поддержка передачи данных по сетям
System.Reflection    // Работа с пользовательскими типами во время выполнения приложения
System.Threading     // Работа с потоками
System.Web           // Работа с web-приложениями
```

Пространство имен

.NET Framework располагает большим набором полезных функций. Каждая из них является членом какого-либо класса. Классы группируются по пространствам имен. Это означает, что в общем случае имя класса может иметь сложную структуру — состоять из последовательности имен, разделенных между собой точками. Последнее имя в этой последовательности собственно и является именем класса. Классы, имена которых различаются лишь последними членами (собственно именами классов) последовательностей, считаются принадлежащими одному пространству имен. Средством "навигации" по пространствам имен, а точнее, средством, которое позволяет сокращать имена классов, является оператор using, формат его запись следующий:

```
using <ИмяПространстваИмен>;
```

В приложении может объявляться собственное пространство имен, а также могут использоваться ранее объявленные пространства. В процессе построения

сборки транслятор должен знать расположение сборок с заявленными для использования пространствами имен. Расположение части сборок известно изначально. Расположение всех остальных требуемых сборок указывается явно (непосредственно в Visual Studio при работе над проектом открыть окно Solution Explorer, выбрать пункт References, далее Add Reference... – там надо задать или выбрать соответствующий .DLL- или .EXE-файл).

В частности, сборка, которая содержит классы, сгруппированные в пространстве имен System, располагается в файле mscorlib.dll. Для обращения к функциям расположенным в файле mscorlib.dll достаточно в заголовке файла прописать:

```
using System;
```

тогда обращение к функции WriteLine(...), являющейся членом класса Console, которое в свою очередь принадлежит пространству имен System, выглядело бы следующим образом:

```
Console.WriteLine("Hello World!");              // в случае прописания строки using System;
System.Console.WriteLine("Hello World!");   // в случае, если строка using System; отсутствует
```

При компиляции модуля, транслятор находит имя функции по полному имени функции (если используется оператор using – то по восстановленному на его основе) находит ее код, который и используется при выполнении сборки.

Примеры ввода вывода данных см.

Приложение 2.

Типы данных

Система типов включает несколько категорий типов:

- типы значений (типы-значения);
- ссылочные типы (типы-ссылки);
- параметризованные типы (типы-шаблоны).

Простые типы – это типы, имя и основные свойства которых известны компилятору. Относительно элементарных типов компилятору не требуется никакой дополнительной информации. Свойства и функциональность этих типов известны. Среди простых типов различаются: целочисленные, с плавающей точкой, decimal, булевский.

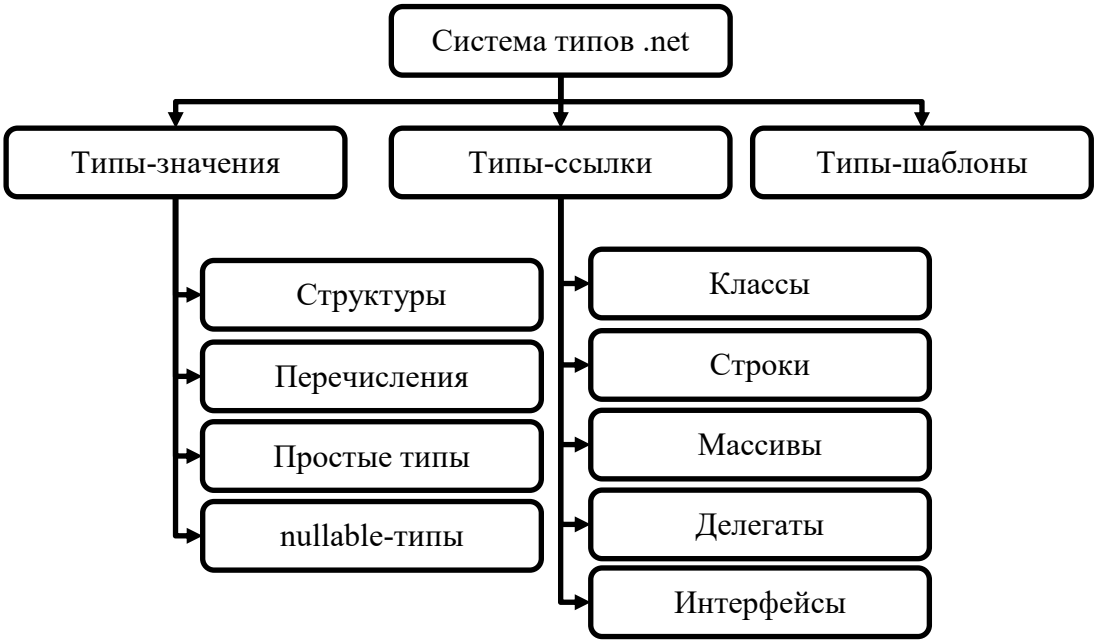


Рис. 3 – Схема типов

Некоторые характеристики простых типов отражены в таблица 1. Используемые в .NET языки программирования основываются на общей системе типов.

Таблица 1 Описание типов данных

Тип	Пространство имен	Описание	Дипазон значений
Sbyte	System.SByte	Целый. 8-разрядное со знаком	–128 ... 127
Byte	System.Byte	Целый. 8-разрядное без знака	0 ... 255
Short	System.Int16	Целый. 16-разрядное со знаком	–32768 ... 32767

ushort	System.UInt16	Целый. 16-разрядное без знака	0 ... 65535
Int	System.Int32	Целый. 32-разрядное со знаком	–2147483648 ... 2147483647
uint	System.UInt32	Целый. 32-разрядное без знака	0 ... 4294967295
long	System.Int64	Целый. 64-разрядное со знаком. Диапазон значений	–9223372036854775808 ... 9223372036854775807
ulong	System.UInt64	Целый. 64-разрядное без знака	0 ... 18446744073709551615
char	System.Char	16 разрядный символ UNICODE	
float	System.Single	Плавающий. 32 разряда (Стандарт IEEE)	
double	System.Double	Плавающий. 64 разряда (Стандарт IEEE)	
decimal	System.Decimal	128-разрядное значение повышенной точности с плавающей точкой	
bool	System.Boolean	Значение true или false	true,false

Преобразование и приведение типов

В программировании переменной одного типа часто присваивается значение переменной другого типа. Например, как показано в следующем фрагменте программы, мы могли бы присвоить переменной типа float значение типа int.

```
int i;          // Объявление переменной i, являюейся целым типом
float f;        //  Объявлени переменной f, являюейся вещественным типом
i = 10;         // Присвоение переменной i значения 10
f = i;          // float-переменной присваивается int-значение.
```

Если в инструкции присваивания смешиваются совместимые типы, значение с правой стороны (от оператора присваивания) автоматически преобразуется в значение "левостороннего" типа. Таким образом, в фрагменте программы значение, хранимое в int-переменной i, преобразуется в значение типа float, а затем присваивается переменной f. Но, поскольку в C# не все типы совместимы и действует строгий контроль типов, не все преобразования типов разрешены в неявном виде. Например, типы bool и int несовместимы. Тем не менее с помощью операции приведения типов все-таки возможно выполнить преобразование между несовместимыми типами. Приведение типов - это выполнение преобразования типов в явном виде.

При присвоении значения одного типа данных переменной другого типа будет выполнено автоматическое преобразование типов, если

- эти два типа совместимы;
- тип приемника больше (т.е. имеет больший диапазон представления чисел), чем тип источника.

При соблюдении этих двух условий выполняется преобразование с расширением, или расширяющее преобразование. Например, тип int — достаточно "большой" тип, чтобы сохранить любое допустимое значение типа byte, а поскольку как int , так и byte - целочисленные типы, здесь может быть применено автоматические преобразование.

Для расширяющих преобразований числовые типы, включая целочисленные и с плавающей точкой, совместимы один с другим. Например, следующая программа совершенно легальна, поскольку преобразование типов из long в double является расширяющим, которое выполняется автоматически.

```
long L;
double D;
L = 100123285L;
D = L;
Console.WriteLine("L и D: " + L + " " + D);
```

Несмотря на возможность автоматического преобразования типов из long в double, обратное преобразование типов (из double в long) автоматически не выполняется, поскольку это преобразование не является расширяющим. Таким образом, следующая версия предыдущей программы недопустима:

```
long L;
double D;
D = 100123285.0;
L = D;                                          // Неверно!!!
Console.WriteLine("L и D: " + L + " " + D);
```

Также не существует автоматического преобразования между типом decimal и float (или double), а также из числовых типов в тип char (или bool). Кроме того, несовместимы и типы char и bool.

Несмотря на большую пользу автоматического преобразования типов оно не в состоянии удовлетворить все нужды программирования, поскольку реализуется только при расширяющем преобразовании между совместимыми типами. Во всех остальных случаях приходится применять приведение к типу.

Приведение к типу — это явно заданная инструкция компилятору преобразовать один тип в другой. Инструкция приведения записывается в следующей общей форме:

```
(тип_приемника) выражение;
```

Здесь элемент тип_приемника определяет тип для преобразования заданного выражения. Например, если вам нужно, чтобы выражение x/y имело тип int , напишите следующие программные инструкции:

```
double x, y;
(int)(x / y);
```

В этом фрагменте кода, несмотря на то, что переменные x и y имеют тип double, результат вычисления заданного выражения приводится к типу int. Круглые скобки, в которые заключено выражение x / y, обязательны. В противном случае (без круглых скобок) операция приведения к типу int была бы применена только к значению переменной x, а не к результату деления. Для получения результата желаемого типа здесь не обойтись без операции приведения, поскольку автоматического преобразования из типа double в int не существует.

Если приведение приводит к сужающему преобразованию, возможна потеря информации. Например, в случае приведения типа long к типу int информация будет утеряна, если значение типа long больше максимально возможного числа, которое способен представить тип int , поскольку будут "усечены" старшие разряды long - значения. При выполнении операции приведения типа с плавающей точкой к цело численному будет утеряна дробная часть простым ее отбрасыванием. Например, при присвоении переменной целочисленного типа числа 1,23 в действительности будет присвоено число 1. Дробная часть (0,23) будет утеряна.

Операторы языка программирования C#

<u>Арифметические операторы</u>

В C# определены следующие арифметические операторы.

\+ Сложение;

- Вычитание, унарный минус;
* Умножение;
/ Деление;
% Деление по модулю;
-- Декремент (уменьшение на 1);
++ Инкремент (увеличение на 1).

Операторы отношений

Операторы отношений оценивают по "двубальной системе" (ИСТИНА/ЛОЖЬ) отношения между двумя значениями, а логические определяют различные способы сочетания истинных и ложных значений. Поскольку операторы отношений генерируют ИСТИНА/ЛОЖЬ-результаты, то они часто выполняются с логическими операторами. Операторы отношений:

== равно;
!= не равно;
> больше;
< меньше;
>= больше или равно;
<= меньше или равно.

Логические операторы

Логические операторы используются в условных конструкциях. Бывают следующие:

&& логическое И;
|| логическое ИЛИ;
! логическое НЕ.

Операторы присвоения

В C# предусмотрены специальные составные операторы присваивания, которые упрощают программирование определенных инструкций присваивания. Лучше всего начать с примера. Рассмотрим следующую инструкцию:

```
x = x + 10;
```

Используя составной оператор присваивания, ее можно переписать в таком виде:

```
x += 10;
```

Пара операторов += служит указанием компилятору присвоить переменной x сумму текущего значения переменной x и числа 10. А вот еще один пример. Инструкция:

```
x = x - 100;
```

аналогична такой:

```
x -= 100;
```

Обе эти инструкции присваивают переменной x ее прежнее значение, уменьшенное на 100.

Составные версии операторов присваивания существуют для всех бинарных операторов. Общая форма их записи такова:

```
переменная op = выражение;
```

Здесь элемент op означает конкретный арифметический или логический оператор, объединяемый с оператором присваивания. Возможны следующие варианты объединения операторов.

+= -= *= /= %= &= |=

Поскольку составные операторы присваивания выглядят короче своих несоставных эквивалентов, то составные версии

часто называют укороченными операторами присваивания.

Составные операторы присваивания обладают двумя заметными достоинствами. Во-первых, они компактнее своих "длинных" эквивалентов. Во-вторых, их наличие приводит к созданию более эффективного кода (поскольку операнд в этом случае вычисляется только один раз). Поэтому в профессионально написанных программах вы часто встретите именно составные операторы присваивания.

Поразрядные операторы

Поразрядные операторы действуют непосредственно на разряды своих операндов. Они определены только для целочисленных операндов и не могут быть использованы для операндов типа bool, float или double.

Поразрядные операторы предназначены для тестирования, установки или сдвига битов (разрядов), из которых состоит целочисленное значение.

Поразрядные операторы очень часто используются для решения широкого круга задач программирования системного уровня, например, при опросе информации о состоянии устройства или ее формировании.

& Поразрядное И;

| Поразрядное ИЛИ;

^ Поразрядное исключающее ИЛИ;

>> сдвиг вправо;

<< сдвиг влево;

~ Дополнение до 1 (унарный оператор НЕ);

Примеры выполнения поразрядных операций см. в Приложение 3.

Приоритеты операций

Все операции выполняются в соответствие с приоритетом от низкого к высокому. Например, приоритет у операции «*» выше чем у оператора «+», поэтому сначала выполняется умножение, а затем сложение.

Приоритетность операций представлена ниже:

1. () [] . (постфикс)++ (постфикс)— new sizeof typeof unchecked
2. ! ~ (имя типа) +(унарный) –(унарный) ++(префикс) —(префикс)
3. / %
4. + –
5. << >>
6. < > <= => is
7. == !=
8. &
9. ^
10. |
11. &&
12. ||
13. ?:
14. = += –= *= /= %= &= |= ^= <<= >>=

Пример 1

Результатом выполнения программы:

```
int i0 = 0; // инициализация переменной целого типа i0 значением 0;
int i1 = 0; // инициализация переменной целого типа i1 значением 0;

int j0 = 0; // инициализация переменной целого типа j0 значением 0;
int j1 = 0; // инициализация переменной целого типа j1 значением 0;
```

```
int k0 = 1; // инициализация переменной целого типа k0 значением 1;
int k1 = 1; // инициализация переменной целого типа k1 значением 1;

j0 = ++i0;  // увеличение i0 на 1 и присовение результата в переменну j0;
j1 = i1++;  // присовение переменной j1 величиной i1 и увеличение i0 на 1;

// вывод на экран
System.Console.WriteLine("i0={0}; i1={1}; j0={2}; j1={3}", i0, i1, j0, j1);
```

В консольное окно будет выведен следующий текст:

```
i0 =1; i1=1; j0=1; j1=0;
```

<u>Управляющие операторы</u>

Инструкция if

С помощью инструкции if можно организовать избирательное выполнение части программы. Действие инструкции if в C# во многом подобно действию одноименной инструкции в любом другом языке программирования. Что касается языков C, C++ и Java, то здесь налицо полная идентичность. Вот как выглядит простейшая форма записи этой инструкции:

```
if (условие) инструкция;
```

Здесь элемент условие представляет собой булево выражение (которое приводится к значению ИСТИНА или ЛОЖЬ). Заданная инструкция будет выполнена, если условие окажется истинным. В противном случае (если условие окажется ложным) заданная инструкция игнорируется. Рассмотрим следующий пример:

```
if (10 < 11)
    Console.WriteLine("10 меньше 11");
```

В данном случае число 10 действительно меньше 11, т.е. условное выражение истинно, поэтому метод WriteLine () будет вызван. Рассмотрим другой пример:

```
if (10 < 9)
    Console.WriteLine("Этот текст выведен не будет.");
```

Здесь же число 10 никак не меньше 9, поэтому вызов метода WriteLine() не произойдет.

Полный формат записи if такой:

```
if (условие)
```

```
{
        последовательность инструкций
}
else
{
        последовательность инструкций
}
```

Если элемент условие, который представляет собой условное выражение, при вычислении даст значение ИСТИНА, будет выполнена if-инструкция; в противном случае — else-инструкция (если таковая существует). Обе инструкции никогда не выполняются. Условное выражение, управляющее выполнением if-инструкции, должно иметь тип bool.

Инструкция switch

Второй инструкцией выбора является switch. Инструкция switch обеспечивает многонаправленное ветвление. Она позволяет делать выбор одной из множества альтернатив. Хотя многонаправленное тестирование можно реализовать с помощью последовательности вложенных if-инструкций, для многих ситуаций инструкция switch оказывается более эффективным решением. Она работает следующим образом. Значение выражения последовательно сравнивается с константами из заданного списка. При обнаружении совпадения для одного из условий сравнения выполняется последовательность инструкций, связанная с этим условием. Общий формат записи инструкции switch такой:

```
switch(выражение)
{
        case константа1:    последовательность инструкций   break;
        case константа2:    последовательность инструкций   break;
        case константа3:    последовательность инструкций   break;
        ...
        default:        последовательность инструкций   break;
}
```

Элемент выражения инструкции switch должен иметь целочисленный или строковый тип. Выражения, имеющие тип с плавающей точкой, не разрешены. Очень часто в качестве управляющего switch-выражения используется просто переменная; case-константы должны быть литералами, тип которых совместим

с типом заданного выражения. При этом никакие две case-константы в одной switch-инструкции не могут иметь идентичных значений.

Последовательность инструкций default-ветви выполняется в том случае, если ни одна из заданных case-констант не совпадет с результатом вычисления switch выражения. Ветвь default необязательна. Если она отсутствует, то при несовпадении результата выражения ни с одной из case-констант никакое действие выполнено не будет. Если такое совпадение все-таки обнаружится, будут выполнены инструкции, соответствующие данной case-ветви до тех пор, пока не встретится инструкция break. Пример использования switch:

Пример 2

```
char ch;
for(ch='A' ; ch <= 'E' ; ch++)
{
        switch (ch)
        {
                case 'A': Console.WriteLine("ch содержит A"); break;
                case 'B': Console.WriteLine("ch содержит B"); break;
                case 'C': Console.WriteLine("ch содержит C"); break;
                case 'D': Console.WriteLine("ch содержит D"); break;
                case 'E': Console.WriteLine("ch содержит E"); break;
        }
}
```

Операторы цикла

Цикл for

Чтобы многократно выполнить последовательность программных инструкций, необходимо организовать цикл. В языке C# циклические конструкции представлены в богатом ассортименте. В этом разделе мы рассмотрим цикл for. Если вы знакомы с C, C++ или Java, то вам будет приятно узнать, что цикл for в C# работает точно так же, как в этих языках. Простейшая форма записи цикла for имеет следующий вид:

```
for(инициализация; условие; итерация) инструкция;
```

В самой общей форме элемент инициализация устанавливает управляющую переменную цикла равной некоторому начальному значению. Элемент условие представляет собой булево выражение, в котором тестируется

значение управляющей переменной цикла. Если результат этого тестирования истинен, цикл for выполняется еще раз, в противном случае его выполнение прекращается. Элемент итерация — это выражение, которое определяет, как изменяется значение управляющей переменной цикла после каждой итерации. Рассмотрим небольшую программу, в которой иллюстрируется цикл for.

Пример 3

```
int count;
for(count = 0; count < 5; count = count+1)
Console.WriteLine("Это счет: " + count);
Console.WriteLine("Готово!");
```

```
Это счет: 0
Это счет: 1
Это счет: 2
Это счет: 3
Это счет: 4
Готово!
```

В этой программе управляющей переменной цикла является count. В выражении инициализации цикла for она устанавливается равной нулю. В начале каждой итерации (включая первую) выполняется проверка условия count < 5. Если результат этой проверки окажется истинным, выполнится инструкция вывода строки WriteLine (), а после нее — итерационное выражение цикла. Этот процесс будет продолжаться до тех пор, пока проверка условия не даст в результате значение ЛОЖЬ, после чего выполнение программы возобновится с инструкции, расположенной за циклом. В C#-программах, написанных профессиональными программистами, редко можно встретить итерационное выражение цикла в том виде, в каком оно представлено в рассматриваемой программе. Другими словами, вряд ли вы увидите инструкции, подобные следующей:

```
count = count + 1;
```

Дело в том, что C# включает специальный оператор инкремента, который позволяет более эффективно выполнить операцию увеличения значения на единицу. Оператор инкремента обозначается двумя последовательными знаками "плюс" (++);

С его помощью предыдущую инструкцию можно переписать следующим образом:

```
count++;
```

Следовательно, начало цикла for в предыдущей программе опытный программист оформил бы так:

```
for(count = 0; count < 5; count++);
```

Оставив пустым условное выражение цикла for, можно создать бесконечный цикл (цикл, который никогда не заканчивается). Например, в следующем фрагменте программы показан способ, который используют многие C#-программисты для создания бесконечного цикла.

```
for ( ; ; ) // Специально созданный бесконечный цикл.
```

Цикл while

Общая форма цикла while имеет такой вид:

```
while (условие) инструкция;
```

Здесь под элементом инструкция понимается либо одиночная инструкция, либо блок инструкций. Работой цикла управляет элемент условие, который представляет собой любое допустимое выражение типа bool. Элемент инструкция выполняется до тех пор, пока условное выражение возвращает значение ИСТИНА. Как только это условие становится ложным, управление передается инструкции, которая следует за этим циклом.

Пример 4

Пример использования цикла типа while:

```
int num;
int mag;
num = 435679;
mag = 0;
Console.WriteLine("Число: " + num);
while (num > 0)
{
   mag++;
   num = num / 10;
}
Console.WriteLine("Порядок: " + mag);
```

```
Число: 435679
Порядок: 6
```

Цикл while работает следующим образом. Проверяется значение переменной num. Если оно больше нуля, счетчик mag инкрементируется, а значение num делится на 10. Цикл повторяется до тех пор, пока num больше нуля. Когда num станет равным нулю, цикл завершится, а переменная mag будет содержать порядок исходного числа. Подобно циклу for, условное выражение проверяется при входе в цикл while, а это значит, что тело цикла может не выполниться ни разу. Это свойство цикла (иллюстрируемое следующей программой) устраняет необходимость отдельного тестирования до начала цикла.

Цикл do-while

Третьим циклом в C# является цикл do-while. В отличие от циклов for и while, в которых условие проверяется при входе, цикл do-while проверяет условие при выходе из цикла. Это значит, что цикл do-while всегда выполняется хотя бы один раз. Его общий формат имеет такой вид:

```
do
 {
        инструкции;
} while (условие);
```

Несмотря на то что фигурные скобки необязательны, если элемент инструкции состоит только из одной инструкции, они часто используются для улучшения читабельности конструкции do-while, не допуская тем самым путаницы с циклом while. Цикл do-while выполняется до тех пор, пока остается истинным элемент условие, который представляет собой условное выражение.

В следующей программе цикл do-while используется для отображения в обратном порядке цифр, составляющих заданное целое число.

Пример 5

```
int num;
int nextdigit;
num = 198;
Console.WriteLine("Число: " + num);
Console.Write("Число с обратным порядком цифр: " ) ;
do
```

```
{
   nextdigit = num % 10;
   Console.Write(nextdigit);
   num = num / 10;
} while (num > 0);
Console.WriteLine();
```

```
Число: 198
Число с обратным порядком цифр: 891
```

Вот как работает этот цикл. На каждой итерации крайняя справа цифра определяется как остаток от целочисленного деления заданного числа на 10. Полученная цифра тут же отображается на экране. Затем результат этого деления запоминается в той же переменной num. Поскольку деление целочисленное, его результат равносилен отбрасыванию крайней правой цифры. Этот процесс повторяется до тех пор, пока число num не станет равным нулю.

Операторы выхода из цикла

Оператор break

С помощью инструкции break можно организовать немедленный выход из цикла, опустив выполнение кода, оставшегося в его теле, и проверку условного выражения. При обнаружении внутри цикла инструкции break цикл завершается, а управление передается инструкции, следующей после цикла. Рассмотрим простой пример.

Пример 6

```
// Используем break для выхода из цикла.
for (int i = -10; i <= 10; i++)
{
if (i > 0) break; // Завершение цикла при i > 0.
                  Console.Write(i + " ");
}
Console.WriteLine("Готово!");
```

Эта программа генерирует следующие результаты:

```
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 Готово!
```

Как видите, несмотря на то, что этот цикл for спроектирован для перебора значений i в диапазоне от -10 до 10, инструкция break "досрочно" прекращает его выполнение, когда значение переменной i становится положительным.

Оператор continue

Помимо средства "досрочного" выхода из цикла, существует средство "досрочного" выхода из текущей его итерации. Этим средством является инструкция continue. Она принудительно выполняет переход к следующей итерации, опуская выполнение оставшегося кода в текущей. Инструкцию continue можно расценивать как дополнение к более "радикальной" инструкции break. Например, в следующей программе используется инструкция

Пример 7

```
// Выводим четные числа между 0 и 100.
for (int i = 0; i <= 100; i++)
{
        if ((i % 2) != 0)
                continue;       // Переход на следующую итерацию.
        Console.WriteLine(i);
}
```

Здесь выводятся только четные числа, поскольку при обнаружении нечетного числа происходит преждевременный переход к следующей итерации, а метод WriteLine () не вызывается.

В циклах while и do-while инструкция continue передает управление непосредственно инструкции, проверяющей условное выражение, после чего циклический процесс продолжает "идти своим чередом". А в цикле for после выполнения инструкции continue сначала вычисляется итерационное выражение, а затем — условное. И только после этого циклический процесс будет продолжен.

Инструкция continue используется программистами не слишком часто, хотя в некоторых случаях он оказывается весьма кстати.

Оператор continue

Инструкция return обеспечивает возврат из метода. Ее можно использовать для возвращения методом значения.

Области видимости переменных

Не менее важным, чем инструкции управления, элементом языка C# является программный блок. Программный блок представляет собой группирование двух или более инструкций. Такое группирование инструкций реализуется посредством их заключения между открывающей и закрывающей фигурными скобками. После создания блок кода становится логическим элементом программы, который можно использовать в любом ее месте, где может находиться одна инструкция. Например, блок может быть частью if- или for-инструкций. Рассмотрим следующую if-инструкцию:

```
if (w < h)
{
    v = w * h;
    w = 0;
}
```

Здесь сравниваются значения переменных w и h, и если оказывается, что w<h, то будут выполнены обе инструкции внутри блока. Следовательно, две инструкции в блоке образуют логический элемент, в результате чего одна инструкция не может быть выполнена без выполнения другой. Важно то, что, если нужно логически связать две или более инструкций, это легко реализуется созданием программного блока. Именно благодаря блокам можно упростить код реализации многих алгоритмов и повысить эффективность их выполнения.

До сих пор все переменные, с которыми мы имели дело, объявлялись в начале метода Main(). Однако в C# разрешается объявлять переменные внутри любого блока. Блок начинается открывающей, а завершается закрывающей фигурными скобками. Любой блок определяет область объявления, или область видимости (scope) объектов. Таким образом, при создании блока создается и новая область видимости, которая определяет, какие объекты видимы для других частей программы. Область видимости также определяет время существования этих объектов. Самыми важными в C# являются области видимости, которые определены классом и методом. Область видимости, определяемая методом, начинается с открывающей фигурной скобки. Но если метод имеет параметры, они также относятся к области видимости метода. Как

правило, переменные, объявленные в некоторой области видимости, невидимы (т.е. недоступны) для кода, который определяется вне этой области видимости. Таким образом, при объявлении переменной внутри области видимости вы локализируете ее и защищаете от неправомочного доступа и/или модификации. Эти правила области видимости обеспечивают основу для инкапсуляции. Области видимости могут быть вложенными. Например, при каждом создании программного блока создается новая вложенная область видимости. В этом случае внешняя область включает внутреннюю. Это означает, что объекты, объявленные внутри внешней области, будут видимы для кода внутренней области. Но обратное утверждение неверно: объекты, объявленные во внутренней области, невидимы вне ее.

Пример программной реализации

В главном меню выберите раздел: Файл->Создать проект…

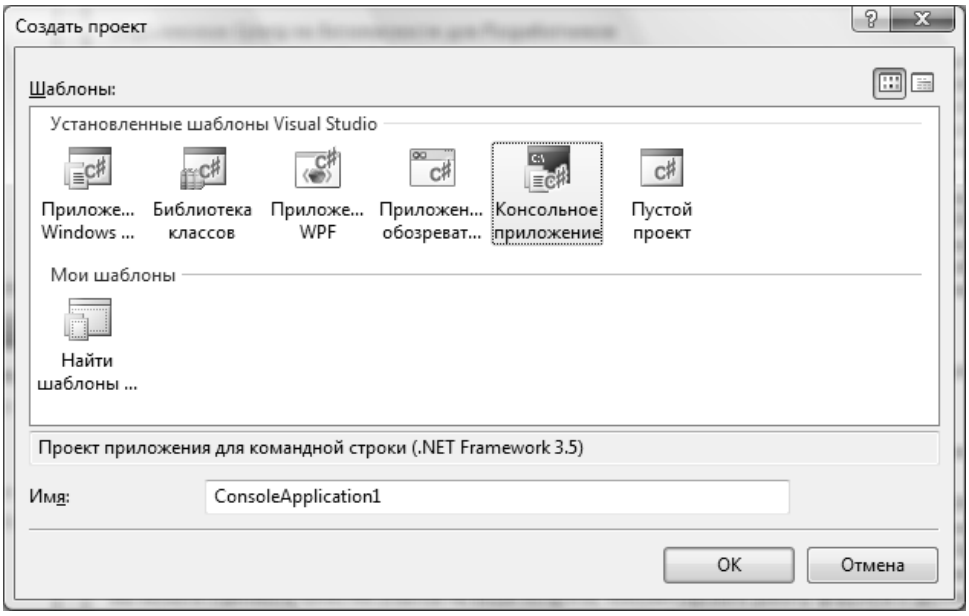


Рис. 4 – Окно выбора шаблона проекта

Microsoft Visual Studio создаст шаблон проекта, как показано на рис. 2

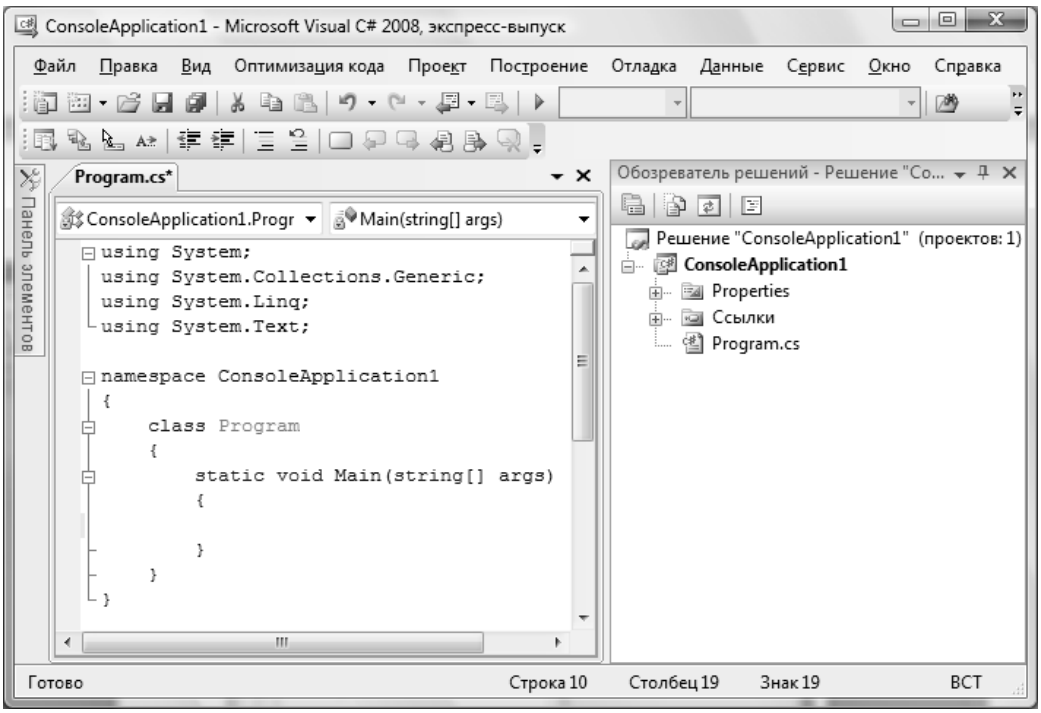


Рис. 5 – Шаблон консольного проекта созданный Microsoft Visual Studio 2008

Строчки:

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

Подключают к проекту пространство имен System, System.Collections.Generic, System.Linq и System.Text. В этих пространствах имен собраны основные, которые наиболее часто используются функции. В C# пространство имен (namespace) определяет декларативную область. Это пространство имен позволяет хранить одно множество имен отдельно от другого. Другими словами, имена, объявленные в одном пространстве имен, не будут конфликтовать с такими же именами, объявленными в другом. В нашей программе используется пространство имен System, которое зарезервировано для элементов, связанных с библиотекой классов среды .NET Framework, используемой языком C#. Ключевое слово using — это своего рода заявление о том, что программа использует имена в заданном пространстве имен.

```
namespace ConsoleApplication1
{
  ...Код программ
}
```

```
Создает новое пространство имен, которое может использовать программист для разработки собственных классов.

class Program
{
    ... Описание класса
}
```

При создании приложения разработчик создает приложение, основанное на разработке класса. C# класс – это базовая единица инкапсуляции. Основной класс каркаса приложения по умолчанию называется Program, в котором, если приложение консольное обязательно должна быть входная функция, которая носит имя Main. Определение класса заключено между открывающей { и закрывающей } фигурными скобками. Таким образом, элементы, расположенные между этими двумя фигурными скобками, являются членами класса. В C# работа программы протекает именно внутри класса. Это одна из причин, по которой все C# программы являются объектно-ориентированными.

```
static void Main(string[] args)
{

}
```

Здесь Main – это имя входной функции (метода), void – это тип возвращаемый этой функцией для операционной системы. String[] args – это строковые параметры, которые можно передать в функцию из командной строки.

Разберем простой пример программы, написанной на языке C#, которая выводит в консольное окно строчку: «Моя первая C#-программа.»

Пример 8

```
static void Main(string[] args)
{
    Console.Title = "My First Project";
    Console.WriteLine("Моя первая C#-программа.");
    Console.ReadLine();
}
```

Сам вывод осуществляется встроенным методом WriteLine (). В данном случае на экране будет отображена строка, переданная методу. Передаваемая методу информация называется аргументом. Помимо текстовых строк, метод WriteLine() может отображать и данные других типов. Console — это имя

встроенного класса, который поддерживает консольные операции ввода-вывода данных. Связав класс Console с методом WriteLine(), вы тем самым сообщаете компилятору, что WriteLine() – член класса Console. Тот факт, что в C# для определения консольного вывода данных используется некоторый объект, является еще одним свидетельством объектно-ориентированной природы этого языка программирования. Обратите внимание на то, что инструкция, содержащая вызов метода WriteLine(), завершается точкой с запятой, как и рассмотренная выше инструкция программы using System. В C# точкой с запятой завершаются все инструкции. Если же вы встречаете строки программы, которые не оканчиваются точкой с запятой, значит, они попросту не являются инструкциями.

С целью, что бы пользователь могу увидеть, что выдало в окно строчка Console.ReadLine(); ждет, когда пользователь нажмет на клавишу Enter.

```
static void Main(string[] args)
{
        int x;          // Здесь объявляется переменная.
        int y;          // Здесь объявляется еще одна переменная.
        x = 100;        // Здесь переменной x присваивается 100.
        Console.WriteLine("x содержит " + x);
        y = x / 2;
        Console.Write("y содержит x / 2: " ) ;
        Console.WriteLine(y);
        Console.ReadLine();
}
```

Строчка:

```
int x;    // Здесь объявляется переменная.
```

объявляет переменную с именем `x` целочисленного типа. В C# все переменные должны быть объявлены до их использования. В объявлении переменной помимо ее имени необходимо указать, значения какого типа она может хранить. Тем самым объявляется тип переменной. В данном случае переменная x может хранить целочисленные значения, т.е. целые числа. В C# для объявления переменной целочисленного типа достаточно поставить перед ее именем ключевое слово int. Таким образом, инструкция int x; объявляет переменную x типа int.

В общем случае, чтобы объявить переменную, необходимо использовать инструкцию следующего формата:

```
тип имя_переменной;
```

Здесь с помощью элемента тип задается тип объявляемой переменной, а с помощью элемента имя_переменной — ее имя. Помимо типа int, C# поддерживает и другие типы данных.

Следующая строка кода выводит значение переменной `x`, предваряя его текстовой строкой " `x` содержит ".

```
Console.WriteLine("x содержит " + x);
```

В этой инструкции знак "плюс" означает не операцию сложения, а последовательное отображение заданной текстовой строки и значения переменной x. В общем случае, используя оператор "+", можно в одной инструкции вызова метода WriteLine() сформировать сцепление элементов в нужном количестве.

Следующая строка кода присваивает переменной y значение переменной x, разделенное на 2.

```
y = x / 2;
```

При выполнении этой строки программы значение переменной x делится на 2, а затем полученный результат сохраняется в переменной y. Таким образом, переменная y будет содержать значение 50. Значение переменной x при этом не изменится.

```
Console.Write("y содержит x / 2: " ) ;
Console.WriteLine(y);
```

Здесь сразу два новых момента. Во-первых, для отображения строки "y содержит x / 2: " используется уже не знакомый нам метод WriteLine (), а встроенный метод Write(). В этом случае выводимая текстовая строка не сопровождается символом новой строки. Это означает, что выполнение очередной операции вывода данных будет начинаться на той же строке. Таким образом, метод Write() аналогичен методу WriteLine (), но с той лишь разницей, что после каждого вызова он не выводит символ новой строки. Во-вторых, обратите внимание на то, что в обращении к методу WriteLine() переменная y

используется самостоятельно, т.е. без текстового сопровождения. Эта инструкция служит демонстрацией того, что как WriteLine(), так и Write() можно использовать для вывода значений любых встроенных C#-типов.

Варианты заданий

Задания выбирается в соответствие с таблицей

Вариант	Задание	Примечание
1	Написать и оформить программу предлагающую пользователю решить арифметические выражения в виде умножения. Программа проверяет правильность ответа и выставляет отметку. Если отметка ниже 4 – программы выводит таблицу умножения, предлагая подучить. Критерий выставления отметок: 0 ошибок – 5; 1 ошибка – 4; 3 ошибки – 3; более 3 ошибок – 2. Количество вопросов 10 шт.	Диалог: Вопрос: 5*5 = ? Ответ:
2	Написать и оформить программу предлагающую пользователю отгадать целое число. Программа должна спрашивать пользователя загаданное число, сравнивать с заданным, выводить результат сравнения в виде диалога «Больше» или «Меньше» до тех пор, пока пользователь не отгадает его. Когда пользователь отгадает, программа должна вывести текст с «Поздравлением».	Случайное число 10.
3	Написать и оформить программу, выводящую числа Фибоначчи. Программа должна спрашивать пользователя начальное и конечное значение выводимых чисел, подсчитать числа до числа введенного пользователем и вывести их на экран. Числа Фибоначчи: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, … каждое последующее число равно сумме двух предыдущих чисел.	
4	Написать и оформить программу выполняющую перевод десятичного числа, введенного пользователем, в двоичное.	
5	Написать и оформить программу, выводящую на экран все простые числа до числа, введенного пользователем.	
6	Написать программу формирования последовательно чисел, двоичный код которых будет: 0000 0000; 0000 0001; 0000 0010; 0000 0101; 0000 1010; 0001 0101; 0010 1010; 0101 0101; 1010 1010; 0101 0101	
7	Написать программу формирования последовательно чисел, двоичный код которых будет: 0000 0000; 0000 0001; 0000 0010; 0000 0100; 0000 1000; 0001 0000; 0010 0000; 0100 0000; 1000 0000; 0000 0000	
8	Написать программу формирования последовательно чисел, двоичный код которых будет: 0000 0000; 1000 0001; 0100 0010; 0010 0100; 0001 1000; 0001 1000; 0010 0100; 0100 0010; 1000 0001; 0000 0000	
9	Написать программу вычисляющую факториал введенного пользователем числа.	
10	Написать и оформить программу выполняющую перевод десятичного числа, введенного пользователем, в систему счисления	

	с основанием 8.	

Контрольные вопросы

1. Запишите все операторы присвоения и поясните.
2. Запишите все логические операторы и поясните их действия.
3. Запишите все поразрядные операторы и поясните их действия.
4. Запишите все операторы цикла и поясните их действия.
5. Запишите все управляющие операторы и поясните их действия.
6. Запишите все математические операторы и поясните их действия.
7. Запишите все типы данных и опишите их.
8. Что такое программный блок и для чего он используется?
9. Что такое преобразование типов данных?
10. Существует ли автоматическое преобразование типов?
11. Что такое приведение типов данных?
12. Чему будет равно значение b в результате выполнения конструкции:

```
byte b = 255;
b+=5;
```

13. Чему будет равно значение i в результате выполнения конструкции:

```
int i = 5;
i = i++;
```

14. Чему будет равно значение i в результате выполнения конструкции:

```
int i = 5;
i = ++i;
```

15. Чему будет равно значение i в результате выполнения конструкции:

```
int i = 5;
i = i++;
```

16. Сколько раз повторится тело цикла**Ошибка! Источник ссылки не айден.**:

```
for (int i = 0, j = 5; ((i < 10)&&(j > 0)); i++)
        System.Console.WriteLine("i = {0} j = {1}", i, j--);
```

17. Чему будет равно значение i в результате выполнения конструкции:

```
int i = 0;
while (i<10)
        System.Console.WriteLine("i = {0}", ++i);
```

18.Чему будет равно значение i в результате выполнения конструкции:

```
int i = 0;
do
{
        System.Console.WriteLine("i = {0}", i++);
}while(++i<10);
```

19.Чему будет равно значение i в результате выполнения конструкции:

```
int i = 1;
i = i << 1;
```

1. Чему будет равно значение i в результате выполнения конструкции:

```
ushort i = 10;
i = (ushort)(i & 2);
```

2. Чему будет равно значение b в результате выполнения конструкции

```
ushort i = 266;
byte b = 0;
b = (byte)(i);
```

3. Приведите примеры ввода вывода переменных строкового типа;
4. Приведите примеры ввода вывода переменных целого типа;
5. Приведите примеры ввода вывода переменных вещественного типа;
6. Приведите примеры ввода вывода переменных типа char;
7. Приведите примеры ввода вывода с помощью операторов цикла вывода на экран чисел от 0 до 10;
8. Приведите примеры поразрядных операций;
9. Объясните назначение оператора continue;
10. Объясните назначение оператора break;
11. Объясните назначение оператора return;
12. Чему будет равно значение i в результате выполнения конструкции:

```
int i = 0;
while (i<10)
{
        if(++i<0)    continue ;
        i++;
}
```

13.Чему будет равно значение i в результате выполнения конструкции:

```
int i = 0;
while (i<10)
```

```
{       i++;    break; }
```

ЛАБОРАТОРНАЯ РАБОТА №2. РАБОТА С МАССИВАМИ

Цель работы

Научиться работать с массивами данных.

Задачи

Укрепить навыки использования операторов языка C#. Приобрести практические навыки работы с массивами данных.

Порядок выполнения работы: изучить теорию к лабораторной работе. Получить допуск к выполнению лабораторной работы. Выполнить задание в соответствие с вариантом, в соответствии личным номером студента. Оформить отчет по лабораторной работе в текстовом редакторе Microsoft Word, в котором отобразить результаты выполнения работы. Защитить лабораторную работу.

Ключевые навыки, которые должен приобрести обучающийся: создавать проекты на основе шаблонов; владение терминологией; создавать циклические структуры; знать операторы ветвления; знать типы данных; знать преобразование типов данных; знать операторы операций.

Общие сведения

Массив (array) — это коллекция переменных одинакового типа, обращение к которым происходит с использованием общего для всех имени. В C# массивы могут быть одномерными или многомерными, хотя в основном используются одномерные массивы. Массивы представляют собой удобное средство группирования связанных переменных. Например, массив можно использовать для хранения значений максимальных дневных температур за месяц, списка цен на акции или названий книг по программированию из домашней библиотеки. Массив организует данные таким способом, который позволяет легко ими манипулировать. Например, если у вас есть массив, содержащий дивиденды, выплачиваемые по выбранной группе акций, то, построив цикл опроса всего массива, нетрудно вычислить средний доход от этих акций. Кроме

того, организация данных в форме массива позволяет легко их сортировать в нужном направлении. Хотя массивы в C# можно использовать по аналогии с тем, как они используются в других языках программирования, C#-массивы имеют один специальный атрибут, а именно: они реализованы как объекты.

Одномерные массивы

Одномерный массив — это список связанных переменных. Такие списки широко распространены в программировании. Например, один одномерный массив можно использовать для хранения номеров счетов активных пользователей сети. В другом можно хранить количество мячей, забитых в турнире бейсбольной командой. Для объявления одномерного массива используется следующая форма записи.

```
тип[] имя_массива = new тип [размер];
```

Здесь с помощью элемента записи тип объявляется базовый тип массива. Базовый тип определяет тип данных каждого элемента, составляющего массив. Поскольку массивы реализуются как объекты, их создание представляет собой двухступенчатый процесс. Сначала объявляется ссылочная переменная на массив, а затем для него выделяется память, и переменной массива присваивается ссылка на эту область памяти. Таким образом, в C# массивы динамически размещаются в памяти с помощью оператора new.

Пример 9

```
int[] sample = new int [10];
```

Это объявление работает подобно любому объявлению объекта. Переменная sample содержит ссылку на область памяти, выделенную оператором new. Доступ к отдельному элементу массива осуществляется посредством индекса. Индекс описывает позицию элемента внутри массива. В C# первый элемент массива имеет нулевой индекс. Поскольку массив sample содержит 10 элементов, его индексы изменяются от 0 до 9. Чтобы получить доступ к элементу массива по индексу, достаточно указать нужный номер

элемента в квадратных скобках. Так, первым элементом массива sample является sample[0], а последним sample[9].

Демонстрация использования одномерного массива.

Пример 10

```
using System;
class ArrayDemo
{
   public static void Main()
   {
      int[] sample = new int[10];
      int i ;
      for(i = 0 ; i < 10; i = i+1)
         sample[i] = i;
      for(i = 0 ; i < 10; i = i+1)
         Console.WriteLine("sample[" + i + " ] : " + sample[i]);
   }
}
```

Инициализация массива

В предыдущей программе значения массиву nums были присвоены вручную, т.е. с помощью десяти отдельных инструкций присваивания. Существует более простой путь достижения той же цели: массивы можно инициализировать в момент их создания. Формат инициализации одномерного массива имеет следующий вид:

```
тип[] имя_массива = [val1, val2, . . . , valN];
```

Здесь начальные значения, присваиваемые элементам массива, задаются с помощью последовательности val1—valN. Значения присваиваются слева направо, в порядке возрастания индекса элементов массива. C# автоматически выделяет для массива область памяти достаточно большого размера, чтобы хранить заданные значения инициализации (инициализаторы). В этом случае нет необходимости использовать в явном виде оператор new. Теперь рассмотрим более удачный вариант программы Average.

Пример 11

```
// Вычисление среднего арифметического от множества значений.
using System;
class Average
{
   public static void Main()
```

```
  {
    int[] nums = { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };
    int avg = 0;
    for(int i=0; i < 10; i )
      avg = avg + nums[i];
    avg = avg / 10;
    Console.WriteLine("Среднее: " + avg);
  }
}
```

Хотя, как уже было отмечено выше, в этом нет необходимости, при инициализации массива все же можно использовать оператор new. Например, массив nums из предыдущей программы можно инициализировать и таким способом, хотя он и несет в себе некоторую избыточность.

```
int[] nums = { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };
```

Несмотря на избыточность new-форма инициализации массива оказывается полезной в том случае, когда уже существующей ссылочной переменной массива присваивается новый массив. Например:

```
int[] nums;
nums = new int[] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };
```

Здесь массив nums объявляется в первой инструкции и инициализируется во второй. И еще. При инициализации массива допустимо также явно указывать его размер, но размер в этом случае должен соответствовать количеству инициализаторов. Вот, например, еще один способ инициализации массива nums.

```
int[] nums = new int[10] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };
```

В этом объявлении размер массива nums явно задан равным 10.

Двумерные массивы

Простейший многомерный массив — двумерный. В двумерном массиве позиция любого элемента определяется двумя индексами. Если представить двумерный массив в виде таблицы данных, то один индекс означает строку, а второй — столбец. Чтобы объявить двумерный массив целочисленных значений размером 10x20 с именем table, достаточно записать следующее:

```
int[,] table = new int[10, 20];
```

Обратите особое внимание на то, что значения размерностей отделяются запятой. Синтаксис первой части этого объявления

```
[,]
```

означает, что создается ссылочная переменная двумерного массива. Для реального выделения памяти для этого массива с помощью оператора new используется более конкретный синтаксис:

```
int[10, 20];
```

Тем самым обеспечивается создание массива размером 10x20, причем значения размерностей также отделяются запятой. Чтобы получить доступ к элементу двумерного массива, необходимо указать оба индекса, разделив их запятой. Например, чтобы присвоить число 10 элементу массива table, позиция которого определяется координатами 3 и 5, можно использовать следующую инструкцию:

```
table[ 3 , 5] = 10;
```

Рассмотрим пример программы, которая заполняет двумерный массив числами от 1 до 12, а затем отображает содержимое этого массива.

Пример 12

```
// Демонстрация использования двумерного массива.
using System;
class TwoD
{
  public static void Main
  {
    int t, i;
    int[,] table = new int[3, 4];
    for(t=0; t < 3; ++t)
    {
      for(i=0; i < 4; ++i)
      {
        table[t,i] = (t*4)+i+l;
        Console.Write(table[t,i] + " " ) ;
        Console.WriteLine() ;
      }
    }
  }
}
```

В этом примере элемент массива table[0,0] получит число 1, элемент table[0,1] — число 2, элемент table[0,2] — число 3 и т.д. Значение элемента table[2,3] будет равно 12.

Массивы трех и более измерений

В C# можно определять массивы трех и более измерений. Вот как объявляется многомерный массив:

```
ТИП[, .. , ] имя = new тип[размер1, ..., размеры] ;
```

Например, с помощью следующего объявления создается трехмерный целочисленный массив размером 4x10x3:

```
int [, ,] multidim = new m t [ 4 , 10, 3];
```

Чтобы присвоить число 100 элементу массива multidim, занимающему позицию с координатами 2,4,1, используйте такую инструкцию:

```
multidim[2, 4, 1] = 100;
```

Рассмотрим программу, в которой используется трехмерный массив, содержащий 3x3x3-матрицу значений.

Пример 13

```
// Программа суммирует значения, расположенные на диагонали 3x3x3-матрицы.
using System;
class ThreeDMatrix
{
   public static void Main()
   {
      int[,,] m = new int[3, 3, 3];
      int sum = 0;
      int n = 1;
      for(int x=0; x < 3; x++)
         for(int y=0; y < 3; y++)
            for(int z=0; z < 3; z++)
               m[x, y, z] = n++;
      sum = m[0,0,0] + m [ l , l , l ] + m[2, 2, 2 ] ;
      Console.WriteLine("Сумма первой диагонали: " + sum);
   }
}
```

Результаты выполнения этой программы:

```
Сумма первой диагонали: 42
```

Инициализация многомерных массивов

Многомерный массив можно инициализировать, заключив список инициализаторов каждой размерности в собственный набор фигурных скобок. Например, вот каков формат инициализации двумерного массива:

```
тип[,] имя_массива = {
```

```
   {val, val, val, ..., val}
   {val, val, val, ..., val}
   {val, val, val, ..., val}
}
```

Здесь элемент val — значение инициализации. Каждый внутренний блок означает строку. В каждой строке первое значение будет сохранено в первой позиции массива, второе значение — во второй и т.д. Обратите внимание на то, что блоки инициализаторов отделяются запятыми, а точка с запятой становится только после закрывающей фигурной скобки.

Например, следующая программа инициализирует массив sqrs числами от 1 до 10 и квадратами этих чисел.

Пример 14

```
// Инициализация двумерного массива.
using System;
class Squares
{
   public static void Main()
   {
      int[,] sqrs =
          {
                { 1, 1 }, { 2, 4 },  { 3, 9 }, { 4, 16 },  { 5, 25 }, { 6, 36 }, { 7, 49 }, { 8, 64 }, { 9, 81 }, { 10, 100}
          };
      int i, j;
      for(i=0; i < 10; i++)
      {
         for(j=0; j < 2; j++)
            Console.Write(sqrs[i,j] + " " );
         Console.WriteLine();
      }
   }
}
```

Рваные массивы

Если двумерный массив можно представить в виде таблицы, то прямоугольный массив можно определить как массив, строки которого имеют одинаковую длину. Однако C# позволяет создавать двумерный массив специального типа, именуемый рваным, или с рваными краями. У такого массива строки могут иметь различную длину. Следовательно, рваный массив можно использовать для создания таблицы со строками разной длины. Рваные

массивы объявляются с помощью наборов квадратных скобок, обозначающих размерности массива. Например, чтобы объявить двумерный рваный массив, используется следующий формат записи:

```
ТИП[][] имя = new тип[размер][];
```

Здесь элемент размер означает количество строк в массиве. Для самих строк память выделяется индивидуально, что позволяет строкам иметь разную длину. Например, следующий фрагмент программы при объявлении массива jagged выделяет память для его первой размерности, а память для его второй размерности выделяется "вручную".

```
int[][] jagged = new int[3][];
jagged[0] = new int[4];
jagged[1] = new int[3];
jagged[2] = new int[5];
```

Следующая программа демонстрирует создание рваного двумерного массива.

Пример 15

```
// Демонстрация рваных массивов,
using System;
class Jagged
{
  public static void Main()
  {
    int[][] jagged = new int[3][];
    jagged[0] = new int[4];
    jagged[1] = new int[3];
    jagged[2] = new int[5];
    int i;

    // Сохраняем значения в первом массиве.
    for(i=0; i < 4; i++)
      jagged[0][i] = i;

    // Сохраняем значения во втором массиве.
    for(i=0; i < 3; i++)
      jagged[l][i] = i;

    // Сохраняем значения в третьем массиве.
    for(i=0; i < 5; i++)
      jagged[2][i] = i;

    // Отображаем значения первого массива.
    for(i=0; i < 4; i++)
      Console.Write(jagged[0][i] + " " ) ;
    Console.WriteLine();
```

```
    // Отображаем значения второго массива.
    for(i=0; i < 3; i++)
      Console.Write(jagged[1][i] + " " ) ;
    Console.WriteLine();

    // Отображаем значения третьего массива.
    for(i=0; i < 5; i++)
      Console. Write (jagged[2][i] + " ");
    Console .WriteLine();
  }
}
```

Рваные массивы используются нечасто, но в некоторых ситуациях они могут оказаться весьма эффективными. Например, если вам нужен очень большой двумерный массив с неполным заполнением (т.е. массив, в котором используются не все его элементы), то идеальным решением может оказаться массив именно такой, неправильной формы.

Варианты заданий

Вариант	***Задание***
1	Сформировать одномерный массив. Найти в массиве и вывести значение наиболее часто встречающегося элемента.
2	Сформировать одномерный массив. Найти в массиве элемент, наиболее близкий к среднему арифметическому суммы его элементов.
3	Сформировать одномерный массив. Найти в массиве элемент, наиболее далекий от среднего арифметического суммы его элементов.
4	Сформировать массив простых чисел, не больше заданного.
5	Сформировать одномерный массив. Найти наименьшее общее кратное всех элементов массива (то есть число, которое делится на все элементы).
6	Сформировать одномерный массив. Найти наибольший общий делитель всех элементов массива (число, на которое делятся все элементы массива без остатка).
7	Сформировать матрицу целочисленных элементов размером 5x5. Транспонировать матрицу.
8	Сформировать матрицу целочисленных элементов. Переставить все столбцы матрицы в обратном порядке относительно сформированной.
9	Сформировать матрицу целочисленных элементов. Переставить все строки матрицы в обратном порядке относительно сформированной.
10	Сформировать матрицу целочисленных элементов. Найти сумму всех элементов матрицы.
11	Сформировать матрицу целочисленных элементов. Найти произведение всех элементов матрицы.
12	Сформировать матрицу целочисленных элементов. Сформировать массив элементами которого являются сумма всех элементов в строке сформированной матрицы.
13	Сформировать матрицу целочисленных элементов. Сформировать массив элементами которого являются сумма всех элементов в столбце сформированной

	матрицы.

Контрольные вопросы

1. Что такое массив данных?
2. Привести примеры инициализации массивов.
3. Привести примеры формирования одномерного массива.
4. Привести примеры формирования многомерного массива.
5. Привести примеры формирования рванного массива.
6. Привести пример вывода на экран таблицы Пифагора.
7. Пояснить, что будет результатом кода:

```
for(i=10; i >0; --i)
   Console.Write(i + i, " ");
```

8. Пояснить, что будет результатом кода:

```
for(i=10; (i >0)||(i<5); --i)
   Console.Write(i, " ");
```

9. Пояснить, что будет результатом кода:

```
for(i=0; i >0; ++i)
   Console.Write(--i, " ");
```

ЛАБОРАТОРНАЯ РАБОТА №3. СОЗДАНИЕ КЛАССОВ, ФУНКЦИЙ, РЕКУРСИВНЫХ ФУНКЦИЙ

Цель работы

Приобрести навыки в реализации функций, рекурсивных функций, перегрузки операторов. Приобрести навыки работы с математическими функциями, реализованными в пространстве имен System.Math.

Общие сведения

Общая форма определения класса

Класс — это шаблон, который определяет форму объекта. Он задает как данные, так и код, который оперирует этими данными. C# использует спецификацию класса для создания объекта. Объекты — это экземпляры класса. Таким образом, класс — это множество намерений (планов), определяющих, как должен быть построен объект.

Определяя класс, вы определяете данные, которые он содержит, и код, манипулирующий этими данными. Несмотря на то, что очень простые классы могут включать только код или только данные, большинство реальных классов содержат и то, и другое. Данные содержатся в переменных экземпляров, определяемых классом, а код — в методах. Однако важно с самого начала отметить, что класс определяет также ряд специальных членов данных и методов-членов, например статические переменные, константы, конструкторы, деструкторы, индексаторы, события, операторы и свойства. Класс создается с помощью ключевого слова class. Общая форма определения класса, который содержит только переменные экземпляров и методы, имеет следующий вид:

```
class имя_класса
{
  // Объявление переменных экземпляров.
  доступ тип переменная!;
  доступ тип переменная2;

  доступ тип переменная;
  // Объявление методов.

  доступ тип_возврата метод1(параметры)
```

```
  {
    // тело метода
  }
  доступ тип_возврата метод2(параметры)
  {
    // тело метода
  }

  доступ тип_возврата методN(параметры)
  {
    // тело метода
  }
}
```

Объявление каждой переменной и каждого метода предваряется элементом доступ. Здесь элемент доступ означает спецификатор доступа (например, **public**), который определяет, как к этому члену можно получить доступ. Класса могут быть закрытыми в рамках класса или более доступными. Спецификатор доступа определяет, какой именно разрешен тип доступа. Спецификатор доступа необязателен, и, если он не указан, подразумевается, что этот член закрыт (**private**). Члены с закрытым доступом (закрытые члены) могут использоваться только другими членами своего класса.

Определение класса

Для иллюстрации мы создадим класс, который инкапсулирует информацию о зданиях (домах, складских помещениях, офисах и пр.). В этом классе (назовем его Building) будут храниться три элемента информации о зданиях (количество этажей, общая площадь и количество жильцов). Ниже представлена первая версия класса Building. В нем определены три переменные экземпляра: floors, area и occupants. Обратите внимание на то, что класс Building не содержит ни одного метода. Поэтому пока его можно считать классом данных.

```
class Building
{
  public int floors;          // количество этажей
  public int area;            // общая площадь основания здания
  public int occupants;       // количество жильцов
}
```

Переменные экземпляра, определенные в классе Building, иллюстрируют общий способ их объявления. Формат объявления переменной экземпляра такой:

```
доступ тип имя_переменной;
```

Здесь элемент доступ представляет спецификатор доступа, элемент тип — тип переменной экземпляра, а элемент имя_переменной — имя этой переменной. Таким образом, если не считать спецификатор доступа, то переменная экземпляра объявляется так же, как локальная переменная. В классе Building все переменные экземпляра объявлены с использованием модификатора доступа public, который, как упоминалось выше, позволяет получать к ним доступ со стороны кода, расположенного даже вне класса Building.

Определение class создает новый тип данных. В данном случае этот новый тип данных называется Building. Это имя можно использовать для объявления объектов типа Building. Помните, что объявление class — это лишь описание типа; оно не создает реальных объектов. Таким образом, предыдущий код не означает существования объектов типа Building. Чтобы реально создать объект класса Building, используйте, например, такую инструкцию:

```
Building house = new Building();                    // Создаем объект типа Building.
```

После выполнения этой инструкции house станет экземпляром класса Building, т.е. обретет «физическую» реальность. Подробно эту инструкцию мы рассмотрим ниже. При каждом создании экземпляра класса создается объект, который содержит собственную копию каждой переменной экземпляра, определенной этим классом. Таким образом, каждый объект класса Building будет содержать собственные копии переменных экземпляра floors, area и occupants. Для доступа к этим переменным используется оператор "точка" (.). Оператор "точка" связывает имя объекта с именем его члена. Общий формат этого оператора имеет такой вид:

```
объект.член
```

Как видите, объект указывается слева от оператора "точка", а его член — справа. Например, чтобы присвоить переменной floors значение 2, используйте следующую инструкцию.

```
house.floors = 2;
```

В общем случае оператор "точка" можно использовать для доступа как к переменным экземпляров, так и методам. Рассмотрим полную программу, в которой используется класс Building.

Пример 16

```
// Программа, в которой используется класс Building.
using System;
class Building
{
  public int floors;          // количество этажей
  public int area;            // общая площадь основания здания
  public int occupants;       // количество жильцов
}

// Этот класс объявляет объект типа Building,
class BuildingDemo
{
  public static void Main()
  {
    Building house = new Building(); // Создание объекта типа Building,
    int areaPP; // Площадь, приходящаяся на одного жильца.

    // Присваиваем значения полям в объекте house,
    house.occupants = 4;
    house.area = 2500;
    house.floors = 2;

    // Вычисляем площадь, приходящуюся на одного жильца дома.
    areaPP = house.area / house.occupants;
    Console.WriteLine("Дом имеет:\n " + house.floors + " этажа\n " +
      house.occupants + " жильца\n " +
      house.area + " квадратных футов общей площади, из них\n " +
      areaPP + " приходится на одного человека");
  }
}
```

Эта программа состоит из двух классов: Building и BuildingDemo. Внутри класса BuildingDemo метод Main() сначала создает экземпляр класса Building с именем house, а затем получает доступ к переменным этого экземпляра house, присваивая им конкретные значения и используя эти значения в вычислениях. Важно понимать, что Building и BuildingDemo — это два отдельных класса. Единственная связь между ними состоит в том, что один класс создает экземпляр другого. Хотя это отдельные классы, код класса BuildingDemo может получать доступ к членам класса Building, поскольку они объявлены открытыми, т.е. public-

членами. Если бы в их объявлении не было спецификатора доступа public, доступ к ним ограничивался бы рамками класса Building, а класс BuildingDemo не имел бы возможности использовать их. Если предыдущую программу назвать UseBuilding.cs, то в результате ее компиляции будет создан файл UseBuilding.exe. Классы Building и BuildingDemo автоматически становятся составными частями этого исполняемого файла. При его выполнении получим такие результаты:

```
Дом имеет:
2 этажа
4 жильца
2500 квадратных футов общей площади, из них
625 приходится на одного человека
```

В действительности совсем не обязательно классам Building и BuildingDemo находиться в одном исходном файле. Можно поместить каждый класс в отдельный файл и назвать эти файлы Building, cs и BuildingDemo. cs, соответственно. После этого необходимо дать компилятору команду скомпилировать оба файла и скомпоновать их. Для этого нужно поместить оба файла в проект и выполнить команду построения этого проекта.

Прежде чем идти дальше, имеет смысл вспомнить основной принцип программирования классов: каждый объект класса имеет собственные копии переменных экземпляра, определенных в этом классе. Таким образом, содержимое переменных в одном объекте может отличаться от содержимого аналогичных переменных в другом. Между двумя объектами нет связи, за исключением того, что они являются объектами одного и того же типа. Например, если у вас есть два объекта типа Building и каждый объект имеет свою копию переменных floors, area и occupants, то содержимое соответствующих (одноименных) переменных этих двух экземпляров может быть разным. Следующая программа демонстрирует это.

Рекурсия

Рекурсия – метод определения класса объектов или методов предварительным заданием одного или нескольких (обычно простых) его базовых

случаев или методов, а затем заданием на их основе правила построения определяемого класса, ссылающегося прямо или косвенно на эти базовые случаи.

Другими словами, рекурсия — способ общего определения объекта или действия через себя, с использованием ранее заданных частных определений. Рекурсия используется, когда можно выделить самоподобие задачи.

В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная рекурсия), например, функция A вызывает функцию B, а функция B — функцию A. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

Мощь рекурсивного определения объекта в том, что такое конечное определение способно описывать бесконечно большое число объектов. С помощью рекурсивной программы же возможно описать бесконечное вычисление, причём без явных повторений частей программы.

Пример написания рекурсивной функции

Создайте проект. Добавьте в него класс, для этого в меню «Проект» выберите строчку «Добавить класс…».

Выберите шаблон «Класс» и введите имя класса factorial

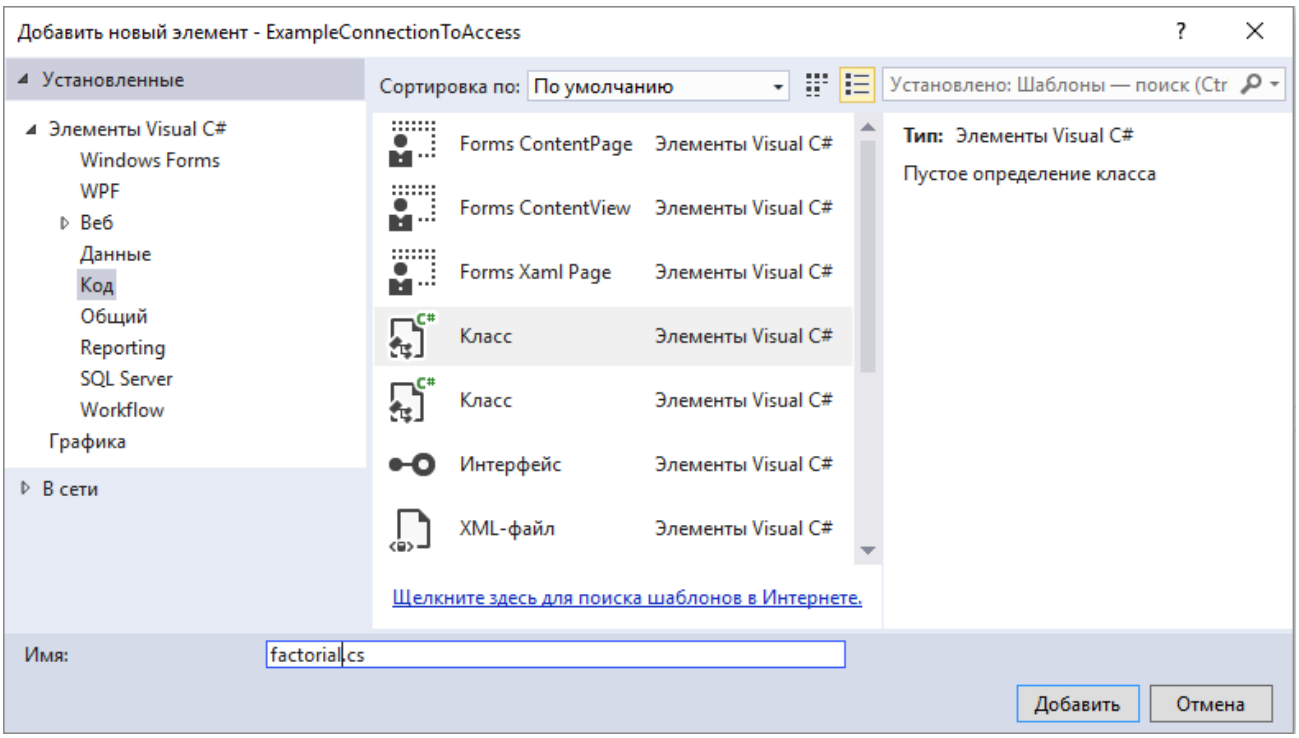


Рис. 6 – Диалоговое окно «Выбор шаблона элемента»

Опишите класс следующим образом:

Пример 17

```
using System;
namespace Lab3
{
  public class factorial
  {
    public factorial()
    {
    }

    public int function1(int value)
    {
      if (value == 1)
        return 1;
      if (value == 0)
        return 1;
      return value * function1(value - 1);
    }

  }
}
```

Здесь функция public int function1(int value) является рекурсивной функцией вычисляющей факториал передаваемого в нее значения (value).

Для использования созданного класса в вашем проекте необходимо создать объект factorial и используя данный объект вызвать функцию function1 следующим образом:

```
static void Main(string[] args)
{
factorial f = new factorial();
    int i = f.function1(5);
Console.Write("5!=" + i.ToString());
}
5!=120
```

Модификаторы доступа

В концепции технологии .net существуют следующие модификаторы доступа:

public – неограниченный доступ.

protected – доступ ограничен для содержащего класса или типов, которые являются производными от содержащего класса.

internal – доступ ограничен текущей сборкой.

protected internal – доступ ограничен текущей сборкой или типами, которые являются производными от содержащего класса.

private – доступ ограничен содержащим типом.

Задание на лабораторную работу

Задание: реализовать класс, реализующий три функции выполняющие вычисление значения функции выбранной в соответствии с таблица 2 с точностью до 0,001:

1. Путем использования функций реализованных в пространстве имен System.Math;
2. Путем разложения функции в ряд и циклического суммирования членов ряда;
3. Путем написания рекурсивной функции, суммирующей члены ряда;

Варианты заданий выбираются в соответствии с таблица 2.

Таблица 2. Варианты заданий к лабораторной работе

Вариант	Вид функции
1	$z = \frac{\sin(x)}{1+x}$
2	$z = 1 + 2 * \cos(x - 1)$
3	$z = \frac{e^x - e^{-x}}{2}$
4	$z = \frac{x}{2}\text{arctg}(x)$
5	$z = \frac{a^x - a^{-x}}{a}$
6	$z = \frac{\ln(1+x)}{1+x}$
7	$z = 2 + \cos(x)$
8	$z = \sin(3 * x)$
9	$z = \frac{x}{2}\text{arctg}(x)$
10	$z = 7 * \sin(x - 1)$

Контрольные вопросы

1. Что такое класс?
2. Что такое экземпляр класса?
3. Привести пример инициализации экземпляра класса.
4. Для чего служит спецификатор public?
5. Для чего служит спецификатор private?
6. Привести пример получения доступа к открытому члену класса?
7. Привести пример получения доступа к закрытому члену класса?
8. Что такое рекурсия?
9. Написать и объяснить принцип работы рекурсивной функции вычисляющей факториал.
10. Написать и объяснить принцип работы циклической функции вычисляющей факториал.
11. Привести пример создания класса.
12. Привести пример создания экземпляра класса.

ЛАБОРАТОРНАЯ РАБОТА №4. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Цель работы

Приобрести навыки программирования применяя концепцию объектно-ориентированного программирования.

Общие сведения

Инкапсуляция

Инкапсуляция – это механизм программирования, который связывает код (действия) и данные, которыми он манипулирует, и при этом предохраняет их от вмешательства извне и неправильного использования. Код, данные или обе эти составляющие объекта могут быть закрытыми внутри него или открытыми. Закрытый код или закрытые данные известны лишь остальной части этого объекта и доступны только ей. Это означает, что к закрытому коду или данным не может получить доступ никакая другая часть программы, существующая вне этого объекта. Если код или данные являются открытыми, к ним получить доступ другие части программы. Как правило, открытые части объекта используются для обеспечения управляемого интерфейса с закрытыми элементами. Основной единицей инкапсуляции в C# является класс. Класс определяет форму объекта. Он задает как данные, так и код, который будет оперировать этими данными. В C# класс используется для создания объектов. Объекты — это экземпляры класса. Код и данные, которые составляют класс, называются членами класса. Данные, определенные в классе, называются переменными экземпляра (instance variable), а код, который оперирует этими данными, методами-членами (member method), или просто методами. "Метод" — это термин, применяемый в C# для обозначения подпрограммы.

Пример 18

```
public class ClassLab4
{
    private int value1 = 1;    // закрытый член класса
    public int value2 = 2;     // открытый член класса
```

```
    private int GetValue1()    // закрытый метод класса
    {
      return value1;
    }

    public int GetValue2()     // открытый метод класса
    {
      return value2;
    }

    public int AddValue()      // открытый член класса
    {
      return value1 + value2;
    }
}
```

В примере показано описание класса ClassLab4 с двумя членами класса и тремя методами класса.

Метод GetValue1() – закрытый метод класса, возвращающий значение закрытой переменной value1;

Метод GetValue2() – открытый метод класса, возвращающий значение открытой переменной value2;

Метод AddValue() – открытый метод класса, возвращающий результат суммы закрытой и открытой переменной класса.

Наследование

Наследование — это процесс, благодаря которому один объект может приобретать свойства другого. Благодаря наследованию поддерживается концепция иерархической классификации. В виде управляемой иерархической (нисходящей) классификации организуется большинство областей знаний. Например, яблоки Голден являются частью классификации яблоки, которая в свою очередь является частью класса фрукты, а тот – частью еще большего класса пища. Таким образом, класс пища обладает определенными качествами (съедобность, питательность и пр.), которые применимы и к подклассу фрукты. Помимо этих качеств, класс фрукты имеет специфические характеристики (сочность, сладость и пр.), которые отличают их от других пищевых продуктов. В классе яблоки определяются качества, специфичные для яблок (растут на деревьях, не тропические и пр.). Класс Голден наследует качества всех

предыдущих классов и при этом определяет качества, которые являются уникальными для этого сорта яблок. Если не использовать иерархическое представление признаков, для каждого объекта пришлось бы в явной форме определить все присущие ему характеристики. Но благодаря наследованию объекту нужно доопределить только те качества, которые делают его уникальным внутри его класса, поскольку он (объект) наследует общие атрибуты своего родителя. Следовательно, именно механизм наследования позволяет одному объекту представлять конкретный экземпляр более общего класса.

Пример 19

Рассмотрим класс TwoDShape, в котором определяются атрибуты "обобщенной" двумерной геометрической фигуры (например, квадрата, прямоугольника, треугольника и т.д.).

```
// Класс двумерных объектов
public class TwoDShape
{
    public double width;      // Ширина
    public double height;     // Высота

    public void ShowDim()
    {
      Console.WriteLine("Ширина = {0}, высота = {1}", width, height);
    }
}
```

```
// Треугольник унаследован от базового класса двумерных объектов
public class Triangle : TwoDShape
{
    public String style; // Тип треугольника

    // Вычисляем площадь треугольника
    public double area()
    {
      return height * width / 2;
    }

    // Отображаем размеры треугольника
    public void ShowDim()
    {
      Console.WriteLine("Ширина = {0}, высота = {1}", width, height);
    }
```

```
    // Отображаем стиль треугольника
    public void ShowStyle()
    {
      Console.WriteLine("Треугольник " + style);
    }
}
```

В классе Triangle создается специфический тип объекта класса TwoDShape, в данном случае треугольник. Класс Triangle содержит все элементы класса TwoDShape и, кроме того, поле style , метод area() и метод ShowStyle(). В переменной style хранится описание типа треугольника, метод area () вычисляет и возвращает его площадь, а метод ShowStyle() отображает данные о типе треугольника.

```
// Прямоугольник унаследован от базового класса двумерных объектов
public class Rectangle : TwoDShape
{
    // Возвращает "истина" если прямоугольник является квадратом
    public bool isSquare()
    {
      if (width == height)
        return true;
      return false;
    }

    // Метод вычисляет площадь прямоугольника
    public double area()
    {
      return width * height;
    }
}
```

Класс Rectangle включает класс TwoDShape и добавляет метод isSquare(), который определяет, является ли прямоугольник квадратом, и метод area(), вычисляющий площадь прямоугольника.

Полиморфизм

Полиморфизм – это качество, которое позволяет одному интерфейсу получать доступ к целому классу действий. Простым примером полиморфизма может послужить руль автомобиля. Руль (интерфейс) остается рулем независимо от того, какой тип рулевого механизма используется в автомобиле. Другими словами, руль работает одинаково в любом случае: оснащен ли ваш автомобиль рулевым управлением прямого действия, рулевым управлением с

усилителем или реечным управлением. Таким образом, поворот руля влево заставит автомобиль поехать влево независимо от типа используемого в нем рулевого управления. Достоинство такого единообразного интерфейса состоит, безусловно, в том, что, если вы знаете, как обращаться с рулем, вы сможете водить автомобиль любого типа.

Концепцию полиморфизма часто выражают такой фразой: "один интерфейс – много методов". Это означает, что для выполнения группы подобных действий можно разработать общий интерфейс. Полиморфизм позволяет понизить степень сложности программы, предоставляя программисту возможность использовать один и тот же интерфейс для задания общего класса действий. Конкретное (нужное в том или ином случае) действие (метод) выбирается компилятором. Программисту нет необходимости делать это вручную. Его задача — правильно использовать общий интерфейс.

Интерфейсы синтаксически подобны абстрактным классам. Однако в интерфейсе ни один метод не может включать тело, т.е. интерфейс в принципе не предусматривает какой бы то ни было реализации. Он определяет, что должно быть сделано, но не уточняет, как. Коль скоро интерфейс определен, его может реализовать любое количество классов. При этом один класс может реализовать любое число интерфейсов. Для реализации интерфейса класс должен обеспечить тело (способы реализации) методов, описанных в интерфейсе. Каждый класс может определить собственную реализацию. Таким образом, два класса могут реализовать один и тот же интерфейс различными способами, но все классы поддерживают одинаковый набор методов. Следовательно, код, "осведомленный" о наличии интерфейса, может использовать объекты любого класса, поскольку интерфейс для всех объектов одинаков. Предоставляя программистам возможность применения такого средства программирования, как интерфейс, C# позволяет в полной мере использовать аспект полиморфизма, выражаемый как "один интерфейс — много методов".

Интерфейсы объявляются с помощью ключевого слова interface . Вот как выглядит упрощенная форма объявления интерфейса:

```
interface имя
{
тип_возврата имя_метода1 {список_параметров) ;
тип_возврата имя_метода2 {список_параметров) ;
// . . .
тип_возврата имя_методаЫ(список_параметров) ;
}
```

Имя интерфейса задается элементом имя. Методы объявляются с использованием лишь типа возвращаемого ими значения и сигнатуры. Все эти методы, по сути, — абстрактные. Как упоминалось выше, для методов в интерфейсе не предусмотрены способы реализации. Следовательно, каждый класс, который включает интерфейс, должен реализовать все его методы. В интерфейсе методы неявно являются открытыми (public-методами), при этом не разрешается явным образом указывать спецификатор доступа.

Итак, если интерфейс определен, один или несколько классов могут его реализовать. Чтобы реализовать интерфейс, нужно указать его имя после имени класса подобно тому, как при создании производного указывается базовый класс. Формат записи класса, который реализует интерфейс, таков:

```
class имя_класса : имя_интерфейса {
// тело класса
}
```

Пример 20

Рассмотрим следующий пример: создадим класс описывающий работу руля, который вращается водителем в диапазоне от -70° до 70°.

7

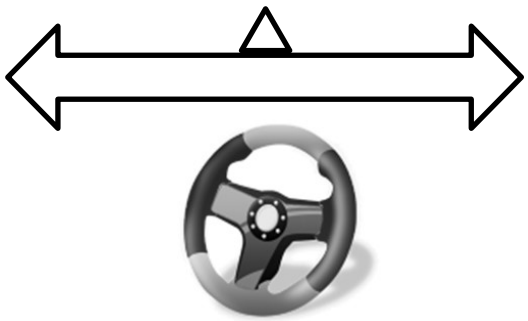


Рис. 7 – Пояснение к примеру

```
// Интерфейс описывающий принцип работы руля
public interface ISteer
  {
    int MoveToLeft(int value);
    int MoveToRight(int value);
    int GetPosition();
    int SetZerro();
    int SetPosition(int value);
  }

public class Steer : ISteer
{
    private int Position = 0;   // положение руля

    Функция описывающая поворот руля влево
    public int MoveToLeft(int value)
    {
      Position = Position - value;
      if (Position < -70)
        Position = -70;
      return Position;
    }

    Функция описывающая поворот руля вправо на значение value
    public int MoveToRight(int value)
    {
      Position = Position + value;
      if (Position > 70)
        Position = 70;
      return Position;
    }

    Возвращает текущее положение руля на значение value
    public int GetPosition()
    {
      return Position;
    }

    Возвращает положение руля
    public int SetZerro()
    {
      Position = 0;
      return Position;
    }

    Устанавливает положение руля в значение value
    public int SetPosition(int value)
    {
      Position = value;
      if (Position < -70)
        Position = -70;
      if (Position > 70)
        Position = 70;
      return Position;
    }
```

```
}

Steer SteerAutomat = new Steer();              // Руль с усилителем
Steer SteerHand = new Steer();                 // Руль без усилителя

int pos = SteerAutomat.SetPosition(80);        // Устанавливая руль положение 80  функция
                                               // установит его в положение 70 (pos=70)

SteerAutomat.Position = 80;              // Закрытый член класса, компилятор выдаст ОШИБКУ!

SteerHand.MoveToLeft(60);                // Повернуть руль влево на 60 градусов.
```

Сокрытие имен с помощью наследования интерфейсов

В производном интерфейсе можно объявить член, который скрывает член, определенный в базовом интерфейсе. Это происходит при совпадении их сигнатур. Такое совпадение вызовет предупреждающее сообщение, если член производного интерфейса не модифицировать с помощью ключевого слова new.

Явная реализация членов интерфейса

При реализации члена интерфейса можно квалифицировать его имя с использованием имени интерфейса. В этом случае говорят, что член интерфейса реализуется явным образом, или имеет место его явная реализация. Например, при определении интерфейса

```
interface IMyiF
{
    int myMeth(int x ) ;
}
```

вполне допустимо реализовать интерфейс IMyiF следующим образом

```
class MyClass : IMyIF
{
     int IMyIF.myMeth(int x)
     {
          return x / 3;
     }
}
```

Как видите, при реализации метода myMeth() члена интерфейса IMyiF указывается его полное имя, включающее имя интерфейса. Явная реализация членов интерфейса может понадобиться по двум причинам. Во-первых, реализуя метод с использованием полностью квалифицированного имени, вы

тем самым обозначаете части закрытой реализации, которые не "видны" коду, определенному вне класса. Во-вторых, класс может реализовать два интерфейса, которые объявляют методы с одинаковыми именами и типами. Полная квалификация имен позволяет избежать неопределенности ситуации.

Закрытая реализация

Следующая программа содержит интерфейс с именем IEven, который определяет два метода isEven() и isOdd(), устанавливающие факт четности и нечетности числа, соответственно. Класс MyClass реализует интерфейс IEven, причем его член isOdd() реализуется явным образом.

```
// Явная реализация члена интерфейса
using System;
interface IEven
{
	bool isOdd(int x ) ;
	bool isEven(int x) ;
}
class MyClass : IEven
{
	// Явная реализация,
	bool IEven.isOdd(int x)
	{
		if((x%2) != 0) return true;
		else return false;
	}

	// Обычная реализация,
	public bool isEven(int x)
	{
		IEven o = this;		// Ссылка на вызывающий объект.
		return !o.isOdd(x);
	}
}

class Demo
{
	public static void Main()
	{
		MyClass ob = new MyClass();
		bool result;
		result = ob.isEven(4);
		if ( result == true )
			Console.WriteLine("4 - четное число.");
		else Console.WriteLine("3 - нечетное число.");
	// result = ob.isOdd(); // Ошибка, член не виден.
}
}
```

Поскольку метод isOdd() реализован в явном виде, он недоступен вне класса MyClass. Такой способ реализации делает его надежно закрытым. Внутри класса MyClass к методу isOdd() можно получить доступ только через ссылку на интерфейс. Вот почему он прекрасно вызывается для объекта ob в реализации метода isEven().

Задание на лабораторную работу

Вариант	***Задание***
1	Построить программу для работы со структурой Дата. Программа должна обеспечивать простейшие функции для работы с данными структурами: увеличение/уменьшение на 1 день, ввод значений, вывод значений.
2	Построить программу для работы со структурой Время. Программа должна обеспечивать простейшие функции для работы с данными структурами: увеличение/уменьшение на 1 час, минуту, секунду, ввод значений, вывод значений.
3	Построить программу для работы со структурами - строками. Структура должна включать следующие поля: массив для хранения строки, его длину, время создания строки. Программа должна обеспечивать простейшие функции для работы с данными структурами: изменение строки, вывод строки, нахождение подстроки в строке.
4	Построить программу для работы со структурами - окнами. Структура должна включать соответствующие поля: размер окна, его положение на экране, цвет. Программа должна обеспечивать простейшие функции для работы с данными структурами: отображение окна, удаление окна, изменение цветов.
5	Построить программу для работы со структурами - многочленами. Структура должна включать соответствующие поля: порядок, набор коэффициентов. Программа должна обеспечивать простейшие функции для работы с данными структурами: вычисление значения многочлена для данного параметра, вывод многочлена в удобной форме.
6	Построить программу для работы со структурами - квадратными матрицами. Структура должна включать соответствующие поля: порядок, набор коэффициентов. Программа должна обеспечивать простейшие функции для работы с данными структурами: ввод матрицы, транспонирование матрицы, вывод матрицы в удобной форме.
7	Построить программу для работы со структурами - правильными дробями. Структура должна включать соответствующие поля: числитель, знаменатель. Программа должна обеспечивать простейшие функции для работы с данными структурами: сложение, вычитание, умножение, деление, вывод дроби в удобной форме.
8	Построить программу для работы со структурами - комплексными числами. Структура должна включать соответствующие поля: вещественную и мнимую часть числа. Программа должна обеспечивать простейшие функции для работы с данными структурами: сложение, вычитание, умножение, деление, вывод числа в удобной форме.
9	Построить программу для работы со структурой Стек FIFO. Программа должна обеспечивать простейшие функции для работы с данными структурами: Добавление в стек данных, /Взятие из стека данных, вывод значений на экран.

10	Построить программу для работы со структурой Стек FILO. Программа должна обеспечивать простейшие функции для работы с данными структурами: Добавление в стек данных, /Взятие из стека данных, вывод значений на экран.

Контрольные вопросы

1. Что в объектно-ориентрованном программировании понимается под инкапсуляцией?
2. Что в объектно-ориентрованном программировании понимается под полиморфизмом?
3. Что в объектно-ориентрованном программировании понимается под наследованием?
4. Привести пример использования инкапсуляции.
5. Привести пример использования полиморфизма.
6. Привести пример использования наследования.
7. Написать структуру кода реализующий процедуру транспонирования матрицы из базового класса представляющего матрицу.
8. Написать структуру кода реализующий процедуру обратной матрицы из базового класса представляющего матрицу.
9. Написать структуру кода реализующий процедуру перемножения матриц.
10. Написать структуру кода реализующий процедуру перемножения матриц.
11. Назовите все модификаторы доступа к членам класса и объясните их назначение.

ЛАБОРАТОРНАЯ РАБОТА №5. СОЗДАНИЕ WINDOWS ПРИЛОЖЕНИЙ

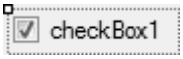
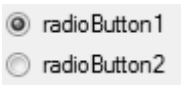
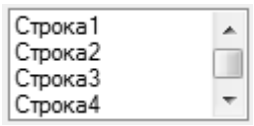
Цель работы

Приобрести навыки создания Windows-приложений основываясь на компонентах библиотеки Framework.

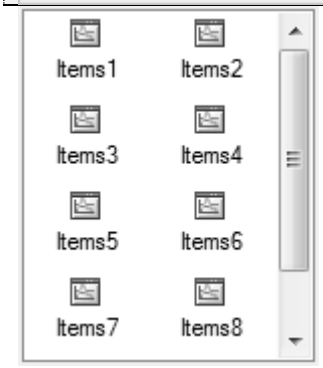
Общие сведения

В библиотеке Framework имеется множество визуальных и не визуальных классов компонент позволяющие упростить работу программиста по написанию программ под Windows приложения.

Таблица 3 – Основные визуальные компоненты Framework

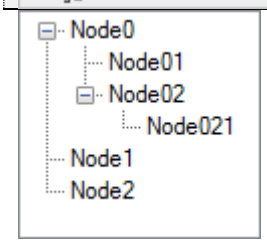
Компонент	**Основные свойства объектов**
A Label	
	Компонент позволяющий выводить текстовую информацию на экран. Основное свойство Label.Text – поле, которое содержит текстовую информацию.
Button	
	Компонент «Кнопка». Основное свойство Button.Text – надпись на кнопке. Основной метод Click –реакция на нажатие кнопки пользователем.
CheckBox	
	Индикатор состояния. Основное свойство: CheckBox.Checked принимающее два состоянию fase|true CheckBox.Text – текстовое поле. Основное событие – CheckedChanged, возникающее в тот момент, когда пользователь меняет состояние поля Checked.
RadioButton	
	Набор компонентов образующих группу, возможную принимать одно из множества значений входящих в группу. Основное свойство отображающее
ListBox	
	Список объектов. Основное свойство: Items предоставляющее коллекцию объектов с методами добавления, удаления. Индекс выбранного элемента определяется свойством listBox1.SelectedIndex Сам выбранный элемент определяется свойством listBox1.SelectedItem

ListView



Список объектов. Основное свойство:
Items предоставляющее коллекцию объектов с методами добавления, удаления.

TreeView



Список объектов отображаемых в виде дерева. Основное свойство: node предоставляющее коллекцию объектов с методами добавления, удаления родительских и дочерних ветвей элементов списка.

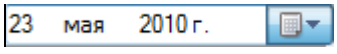
ComboBox

Список объектов. Представляющий выпадающий список. Основное свойство: Items предоставляющее коллекцию объектов с методами добавления, удаления.

NumericUpDown

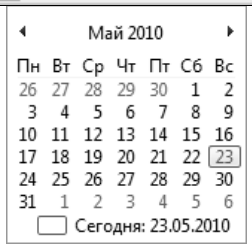
Инкрементальный компонент, позволяющий увеличивать, уменьшать значение Value, отображаемого в элементе на значение указанное в свойстве Increment.

DateTimePicker



Календарь в виде выпадающего списка. Основное свойство Value, отображаемое в выбранной даты.

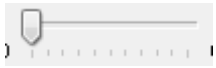
MonthCalendar

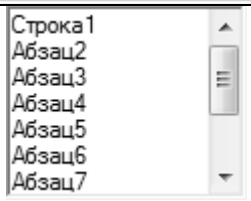


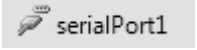
Календарь в виде календаря. Основное свойство Value, отображаемое в выбранной даты.

ProgressBar

Компонент отображающий изменение величины Value, которое может принимать значения из от значений указанных в полях Minimum и Maximum

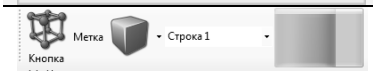
TrackBar	
	Компонент отображающий изменение величины Value, которое может принимать значения из от значений указанных в полях Minimum и Maximum с частотой указанной в поле TickFrequency.

RichTextBox	
	Компонент, представляющий текстовое поле, в котором весь текст размещается в коллекции Lines – список из строковых переменных, представляющая методы добавления, удаления элементов списка.

SerialPort	
	Компонент работы с последовательным портом компьютера RS-232.

TextBox	
	Текстовое поле. Основное свойство textBox1.Text – отображающее

Timer	
	таймер – компонент, предоставляющий обрабатывать события описанных в обработчеке события Tick, через интервал времени указанного в свойстве Interval. Поле Enabled – включает|выключает таймер.

ToolStrip	
	Компонент панель инструментов позволяющая создавать панель на которой можно размещать кнопки, списки, метки, которые позволяют управлять программой пользователю.

StatusStrip	
	Статусная строка позволяющая создавать панель на которой можно размещать кнопки, списки, метки, которые позволяют управлять программой пользователю.

Задание на лабораторную работу

Вариант	***Задание***
1	Написать программу работы со списками, позволяющая добавлять, удалять элемент в список, редактировать элемент списка. Для отображения элемента списка использовать компонент ListBox.
2	Написать программу работы со списками, позволяющая добавлять, удалять элемент в список, редактировать элемент списка. Для отображения элемента списка использовать компонент ListView.
3	Написать программу работы со списками, позволяющая добавлять, удалять элемент в список, редактировать элемент списка. Для отображения элемента списка использовать компонент TreeView.
4	Написать программу работы со списками, позволяющая добавлять, удалять

	элемент в список, редактировать элемент списка. Для отображения элемента списка использовать компонент ComboBox.
5	Написать программу будильник, выводящий строку во время заданное пользователем.
6	Разработать интерфейс и программу реализующую перевод мер длины.
7	Разработать интерфейс и программу реализующую перевод мер веса.
8	Разработать интерфейс и программу реализующую перевод мер расстояний.
9	Разработать интерфейс и программу реализующую конвертацию денежных единиц.
10	Разработать интерфейс и программу реализующую перевод мер температуры.
Дополнительные задания	
11	Разработать интерфейс и программу секундомер выполняющую отсчет с момента нажатия на кнопку до повторного нажатия на эту же кнопку.
12	Разработать интерфейс и программу игру «Угадай число».
13	Разработать интерфейс и программу позволяющую добавлять в список дату с температурой и выполняющую подсчет средней температуры.

Контрольные вопросы

1. Что собой представляет Framework?
2. Какие основные концепции заложены в технологию Framework?
3.

ЛАБОРАТОРНАЯ РАБОТА №6. ТЕХНОЛОГИЯ РАБОТЫ С БАЗАМИ ДАННЫХ ADO .NET

Цель работы

Приобрести навыки работы с базами данных с применением технологии ADO.NET.

Задачи лабораторной работы

Научиться пользоваться компонентами поддерживающие обращение к базе данных. Научиться выполнять SQL запросы к базам данных с применением технологии ADO.

Общие сведения

База данных — это совокупность структур, предназначенных для хранения информации и программных модулей, осуществляющих управление данными, их выборку, сортировку. Информация базы данных хранится в одной или нескольких таблицах. Любая таблица с данными состоит из набора однотипных записей, расположенных друг за другом. Они представляют собой строки таблицы, которые можно добавлять, удалять или изменять. Каждая запись является набором именованных полей, или ячеек, которые могут хранить самую разнообразную информацию, начиная от даты рождения и заканчивая подробным описанием кулинарного рецепта. Однотипные поля разных записей образуют столбец таблицы.

Записи одной таблицы могут содержать ссылки на данные другой таблицы, например, в таблице со списком товаров могут храниться ссылки на справочник производителей товаров с их адресами и другими реквизитами. При этом записи, касающиеся разных товаров, могут указывать на одного и того же производителя. Такое взаимодействие таблиц называется связью.

Другие модули базы данных предназначены для обработки информации, хранящейся в таблицах. С помощью запросов производится выборка данных, отвечающих определенным условиям. Формы предназначены для

форматированного ввода и восприятия информации. Отчеты обеспечивают вывод (как правило, на принтер) красочно оформленного списка записей с заголовками, пунктами и подпунктами.

Типы межтабличных связей

Отношение «один-ко-многим»

Рассмотрим базу данных, в которой учитываются заказы, включающую таблицы «Клиенты» и «Заказы». Клиент может разместить любое количество заказов. Следовательно, у любого клиента, представленного в таблице «Клиенты», может быть много заказов, представленных в таблице «Заказы». Поэтому связь между таблицами «Клиенты» и «Заказы» — это отношение «один-ко-многим».

Чтобы создать отношение «один-ко многим» в структуре базы данных, добавьте первичный ключ на стороне «один» в таблицу на стороне «многие» в виде дополнительного поля. В данном примере необходимо добавить новое поле — поле «Код» из таблицы «Клиенты» — в таблицу «Заказы» и назвать его «Код клиента». После этого Access сможет использовать номер «Код клиента» из таблицы «Заказы» для поиска клиента каждого заказа.

Отношение «многие-ко-многим»

Рассмотрим связь между таблицей «Продукты» и таблицей «Заказы». Один заказ может включать несколько продуктов. С другой стороны, отдельный продукт может содержаться в нескольких заказах. Следовательно, для каждой записи таблицы «Заказы» может существовать несколько записей в таблице «Продукты» и наоборот. Такой тип связи называется отношением «многие-ко-многим», поскольку для каждого продукта может быть много заказов и наоборот. Обратите внимание, что для обнаружения существующего отношения между таблицами важно рассмотреть обе его стороны.

Чтобы представить отношение «многие-ко-многим», нужно создать третью (связующую) таблицу, в которой отношение «многие-ко-многим» разбивается

на два отношения «один-ко-многим». Первичные ключи двух таблиц вставляются в третью таблицу. В результате в третьей таблице сохраняются все экземпляры отношения. Например, таблицы «Заказы» и «Продукты» имеют отношение «многие-ко-многим», определяемое созданием двух отношений «один-ко-многим» в таблице «Заказано». В одном заказе может быть много продуктов, и каждый продукт может появляться во многих заказах.

Отношение «один-к-одному»

При отношении «один-к-одному» каждая запись в первой таблице может иметь не более одной связанной записи во второй таблице и наоборот. Отношения этого типа используются нечасто, поскольку обычно сведения, связанные таким образом, хранятся в одной таблице. Отношение «один-к-одному» используется для разделения таблицы, содержащей много полей, с целью отделения части таблицы по соображениям безопасности, а также с целью сохранения сведений, относящихся к подмножеству записей в главной таблице. После определения такого отношения у обеих таблиц должно быть общее поле.

Технология ADO.NET

Microsoft ADO.NET – технология доступа к данным, которая представляет собой набор классов, входящих в состав .Net Framework, и предназначена для обеспечения единообразного доступа к данным. ADO.NET разделяет доступ к данным и манипуляции с ними. Соединенные классы, предоставляемые поставщиками данных ADO.NET, обеспечивают соединение с источником данных, выполнение команд и считывание результатов. Отсоединенные классы позволяют обращаться к данным и производить манипуляции с ними в автономном режиме, а затем синхронизировать изменения с соответствующим источником данных.

Технология ADO.NET предназначена для обеспечения доступа к данным таких баз данных как Microsoft Access, Microsoft SQL Server, Oracle и источникам данных Microsoft Excel, Microsoft Outlook, текстовым файлам.

Для подключения к источнику данных, выполнения команд и получения результатов используется поставщик данных .NET. В составе .NET Framework входят следующие поставщики данных: Microsoft SQL Server, Oracle, OLE DB и ODBC.

Каждый поставщик данных обязан предоставлять следующие соединенные классы:

Connection – уникальный сеанс работы с источником данных;

Command – выполняет SQL-операторы и хранимые процедуры над источником данных;

DataReader – предоставляет последовательный доступ только для чтения к потоку результатов запроса;

DataAdapter – выполняет роль моста между соединенными и отсоединенными классами; служит также для наполнения отсоединенного набора данных (DataSet) и внесения изменений в источник данных, сделанных в отсоединенном наборе данных.

DataSet – функционирующая в оперативной памяти реляционная база данных;

DataTable – отдельная таблица данных, располагающая в оперативной памяти;

DataColumn – схема столбца локальной таблицы;

DataRow – строка в локальной таблице;

DataView – представление локальной таблицы с возможностью привязки данных, используемое для пользовательской сортировки, фильтрации, поиска, редактирования и навигации;

DataRelation – связь «родитель/потомок» между двумя таблицами в наборе данных;

Constraint – ограничение на один или несколько столбцов в локалной таблице, служащее для поддержания целостности данных.

Набор данных не содержит информации об источнике данных, из которых он наполнялся, в нем хранятся как текущая, так и исходная версии данных, что позволяет внести изменения в источник данных позднее.

BindingNavigator – представляет пользовательский интерфейс для перехода и обработки для элементов управления на форме, которые привязываются к данным.

BindingSource – инкапсулирует источник данных для форм.

DataGridView – отображает данные в настраиваемой таблице.

Язык запросов SQL

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволила создать компактный язык с небольшим набором предложений. Язык SQL может использоваться как для выполнения запросов к данным, так и для построения прикладных программ.

Основные категории команд языка SQL предназначены для выполнения различных функций, включая построение объектов базы данных и манипулирование ими, начальную загрузку данных в таблицы, обновление и удаление существующей информации, выполнение запросов к базе данных, управление доступом к ней и ее общее администрирование.

Основные категории команд языка SQL:

- **DDL** (Data Definition Language) – язык определения данных позволяет создавать и изменять структуру объектов базы данных, например, создавать и удалять таблицы. Основными командами языка DDL являются следующие: CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX;
- **DML** (Data Manipulation Language) – язык манипулирования данными, используется для манипулирования информацией внутри объектов реляционной базы данных посредством трех основных команд: INSERT, UPDATE, DELETE;

- **DQL** – язык запросов включает всего одну команду SELECT - команда используется для формирования запросов к реляционной базе данных;
- **DCL** (Data Control Language) – язык управления данными. Команды управления данными позволяют управлять доступом к информации, находящейся внутри базы данных. Как правило, они используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями. Команды управления данными: GRANT, REVOKE.
- **Команды администрирования данных** - команды с помощью которых пользователь осуществляет контроль за выполняемыми действиями и анализирует операции базы данных; они также могут оказаться полезными при анализе производительности системы;
- **Команды управления транзакциями** - позволяют управлять транзакциями базы данных: COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

Команда SELECT

Команда SELECT - инструктирует базу данных, чтобы извлечь информацию из таблицы.

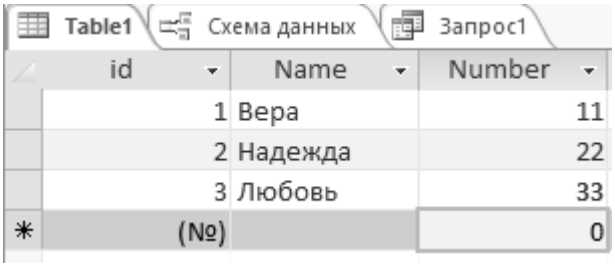
```
SELECT *|{[DISTINCT | ALL] <value expression>.,..} FROM {<table name> [<alias> ] }.,..
[WHERE <predicate>]
[GROUP BY { <column name> | <integer> }.,..]
[HAVING <predicate>] [ORDERBY {<column name> | <integer> }.,..]
[{UNION [ALL]
SELECT*|{[DISTINCT | ALL] < value expression >.,..} FROM {<table name> [<alias>]} .,..
[WHERE <predicate> [GROUP BY {<columnname> | <integer>}.,..]
[HAVING <predicate>] [ORDER BY {<columnname> | <integer>}.,..] } ] ...;
```

где:

- FROM – определяются имена используемых таблиц;
- WHERE – выполняется фильтрация строк объекта в соответствии с заданными условиями;
- GROUP BY – образуются группы строк, имеющих одно и то же значение в указанном столбце;

- HAVING – фильтруются группы строк объекта в соответствии с указанным условием;
- SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;
- ORDER BY – определяется упорядоченность результатов выполнения операторов;
- <value expression> – выражение, которое производит значение. Оно может включать в себя или содержать <column name>. <table name> Имя или синоним таблицы или представления;
- <alias> – временный синоним для <table name>, определённый в этой таблице и используемый только в этой команде;
- <predicate> – условие, которое может быть верным или неверным для каждой строки или комбинации строк таблицы в предложении FROM.
- <column name> – имя столбца в таблице;
- <integer> – число с десятичной точкой. В этом случае, оно показывает <value expression> в предложении SELECT с помощью идентификации его местоположения в этом предложении.

Пример: есть таблица Table1 с полями id-идентификатор, Name – некоторое имя, Number – какое-то число.



id	Name	Number
1	Вера	11
2	Надежда	22
3	Любовь	33
(№)		0

Задание: выполнить запрос на выборку всей таблицы.

```
SELECT id, Name, Number FROM Table1
или
SELECT * FROM Table1
```

Команда INSERT, UPDATE, DELETE

Команда INSERT – вставляет новую запись в таблицу

```
INSERT INTO < table name> [(<column name> .,. ] { VALUES ( <value expression> .,.. ) } | <query>
```

```
UPDATE <tablename> SET { | }.„. .< column name> = <value expresslon> [ WHERE <predlcate> | WHERE CURRENT OF <cursor name> (*только для вложения*) ]
```

```
DELETE FROM <table name> [ WHERE <predicate> | WHERE CURRENT OF <cursor name> (*только для вложения*) ]
```

ЭЛЕМЕНТ ОПРЕДЕЛЕНИЕ <cursor name> Имя курсора используемого в этой программе. <query> Допустимая команда SELECT

<u>Пошаговая инструкция программы управления набором данных</u>

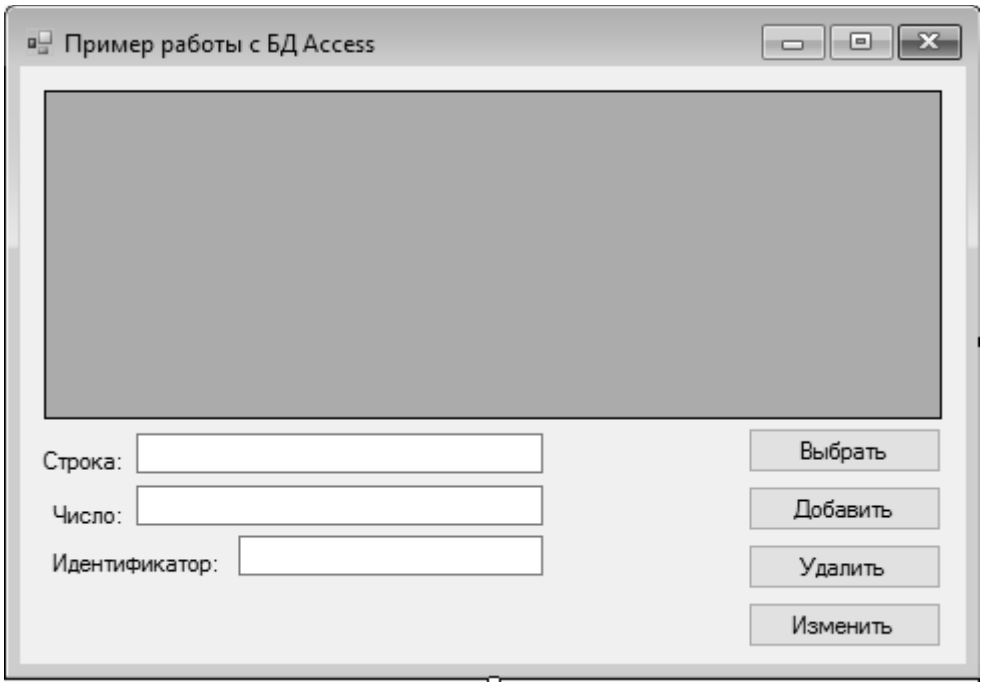
Пример написания программного кода, демонстрирующий управление наборами данных хранящихся в БД Access представлен ниже.

1. Создайте в БД Access таблицу Table1, с полями id – идентификатор (счетчик); Name – имя (строка); Number – номер (число).

Table1

id	Name	Number
1	Вера	11
2	Надежда	22
3	Любовь	33
(№)		0

2. Создайте Windows Application приложение вида:



В событие на кнопку «Выбрать» впишите следующий код:

```
String FileName = @"C:\dbAccess.accdb";
        String ConnectionString = @"Provider = Microsoft.ACE.OLEDB.12.0; Data Source = " + FileName;
        String select = "SELECT * FROM Table1";
        OleDbDataAdapter adapter = new OleDbDataAdapter();
        DataTable dt = new DataTable();
        adapter.SelectCommand = new OleDbCommand(select, new
OleDbConnection(ConnectionString));
        adapter.Fill(dt);
        dataGridView1.DataSource = dt;
```

В событие на кнопку «Добавить» впишите следующий код:

```
String insert = "INSERT INTO Table1 ([Name], [Number]) VALUES (?, ?)";
String select = "SELECT * FROM Table1";
String FileName = @" C:\dbAccess.accdb";
String ConnectionString = @"Provider = Microsoft.ACE.OLEDB.12.0; Data Source = " + FileName;

OleDbDataAdapter adapter = new OleDbDataAdapter(select, new
OleDbConnection(ConnectionString));
OleDbConnection connection = new OleDbConnection(ConnectionString);
OleDbCommand comm = new OleDbCommand(insert, connection);

comm.Parameters.Add("[Name]", OleDbType.Char, textBox1.Text.Length, "Name").Value =
(Object)(textBox1.Text);
comm.Parameters.Add("[Number]", OleDbType.Integer, 4, "Number").Value =
(Object)(Int32.Parse(textBox2.Text));
adapter.InsertCommand = comm;
connection.Open();
adapter.InsertCommand.ExecuteNonQuery();
connection.Close();
```

В событие на кнопку «Удалить» впишите следующий код:

```
String delete = "DELETE FROM Table1 WHERE ([id] = ?)";
String select = "SELECT * FROM Table1";
String FileName = @"C:\dbAccess.accdb";
String ConnectionString = @"Provider = Microsoft.ACE.OLEDB.12.0; Data Source = " + FileName;

OleDbDataAdapter adapter = new OleDbDataAdapter(select, new
OleDbConnection(ConnectionString));

OleDbConnection connection = new OleDbConnection(ConnectionString);
OleDbCommand comm = new OleDbCommand(delete, connection);
comm.Parameters.Add("[id]", OleDbType.Integer, 4, "id").Value =
(Object)(Int32.Parse(textBox3.Text));

connection.Open();
adapter.DeleteCommand = comm;
adapter.DeleteCommand.ExecuteNonQuery();
connection.Close();
```

В событие на кнопку «Изменить» впишите следующий код:

```
String FileName = @"C:\dbAccess.accdb";
String ConnectionString = @"Provider = Microsoft.ACE.OLEDB.12.0; Data Source = " + FileName;
String select = "SELECT * FROM Table1";
String update = "UPDATE Table1 SET [Name] = ?, [Number] = ? WHERE([id] = ?)";
```

```
OleDbDataAdapter adapter = new OleDbDataAdapter(select, new
OleDbConnection(ConnectionString));
OleDbConnection connection = new OleDbConnection(ConnectionString);
OleDbCommand comm = new OleDbCommand(update, connection);

comm.Parameters.Add("[Name]", OleDbType.Char, textBox1.Text.Length, "Name").Value =
(Object)(textBox1.Text);
comm.Parameters.Add("[Number]", OleDbType.Integer, 4, "Number").Value =
(Object)(Int32.Parse(textBox2.Text));
comm.Parameters.Add("[id]", OleDbType.Integer, 4, "id").Value =
(Object)(Int32.Parse(textBox3.Text));

connection.Open();
adapter.UpdateCommand = comm;

adapter.UpdateCommand.ExecuteNonQuery();
connection.Close();
```

Задание на лабораторную работу

Вариант	***Задание***
1	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о группе.
2	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о фильмах.
3	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о автомобилях.
4	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о .
5	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о.
6	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о.
7	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о.
8	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о.
9	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о колебаниях температуры в течение месяца, квартала, года.
10	Разработать интерфейс и программу управления базой данных (БД), хранящейся в формате Microsoft Access. В БД должна хранится информация о транспортных средствах.

Контрольные вопросы

1. Опишите этапы управления записями в базе данных.
2. Поясните назначение элемента DataTable.
3. Поясните назначение элемента DataSet.
4. Поясните назначение элемента Adapter.

5. Поясните назначение элемента Connection.
6. Приведите пример SQL запроса на выборку данных в таблице.
7. Приведите пример SQL запроса на вставку данных в таблицу.
8. Приведите пример SQL запроса на изменение данных в таблице.
9. Приведите пример SQL запроса на удаление данных в таблице.

ЛАБОРАТОРНАЯ РАБОТА №7. WEB-ТЕХНОЛОГИЯ ASP .NET

Создание Windows Form приложений

Цель работы

Приобрести навыки создания динамических web-сайтов с применением технологии ASP.NET.

Общие сведения

ASP.Net – это технология, позволяющая работать динамическому сайту на стороне Web-сервера и предоставляет использование функционала .Net Framework.

Пример создания динамического web-сайта рассмотрим по шаговой инструкции:

Шаг №1. Создание проекта web-сайта.

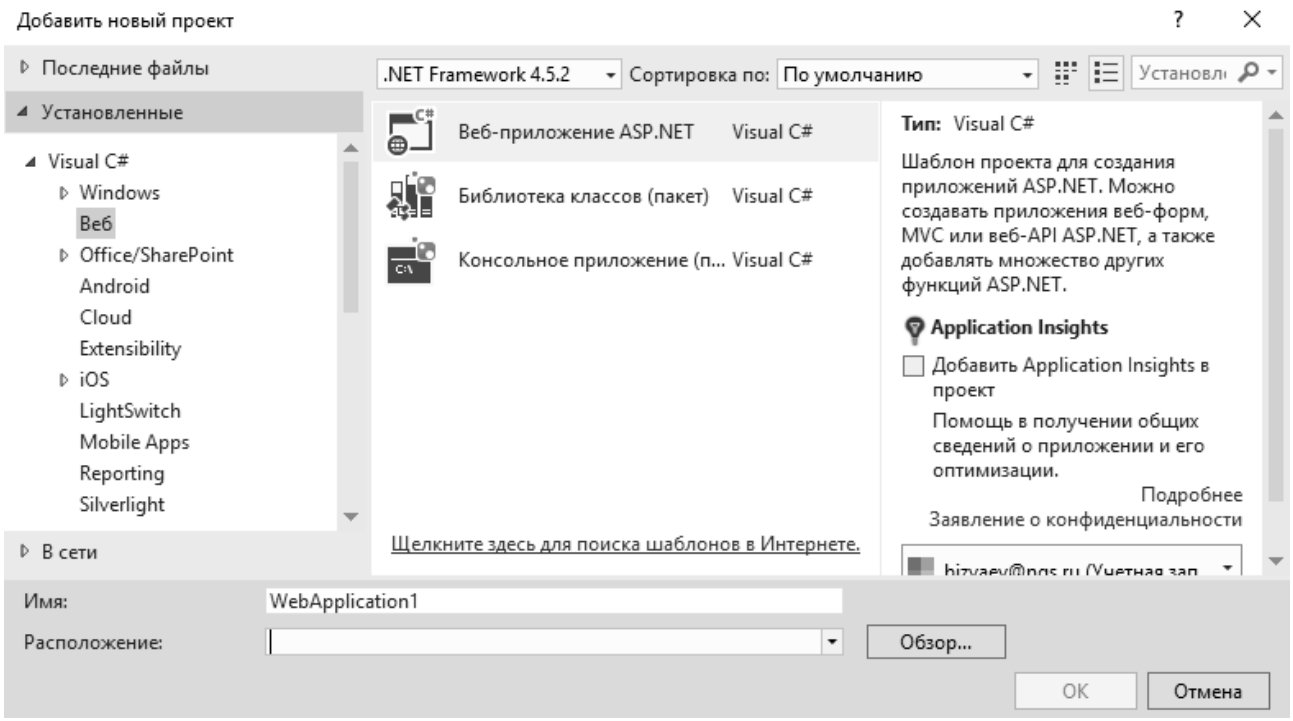


Рис. 8 – Окно создания проекта

Шаг №2. Выбор типа нового проекта на основе шаблона.

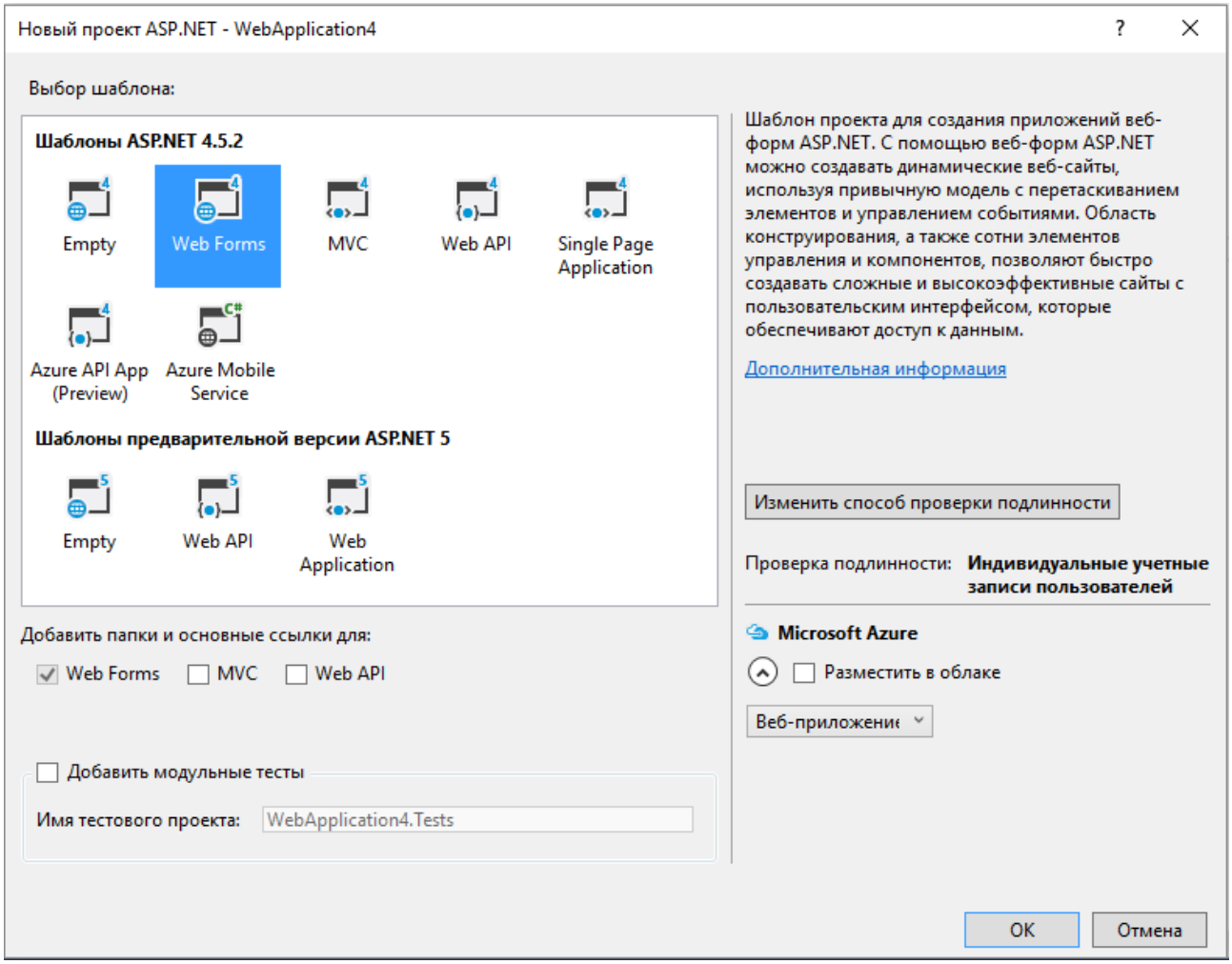


Рис. 9 – Окно выбора шаблона проекта

Шаг №3. Формирование формы сайта с объектами ASP.NET

Наберите следующий текст в файл Default.aspx:

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="WebApplication3._Default" %>
<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
  <header runat="server">
    Пример отправки данных на сервер!
  </header>
<asp:Label ID="Label1" runat="server" Text="Отправить сообщение в глобальную сеть!"
style="font-weight: 700"></asp:Label>
 <hr/>
<asp:Label ID="Label2" runat="server" Text="Сообщение: "></asp:Label>
<asp:TextBox ID="TextBox1" runat="server" Text ="Сообщение!"> </asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Отправить" OnClick="Button1_Click" />
  <hr /> <br />
<asp:Label ID="Label3" runat="server" Text="Принятое сообщение: "></asp:Label>
<asp:Label ID="Label4" runat="server" Text="Label"></asp:Label>
</asp:Content>
```

Файл *.aspx – формирует визуальную форму сайта, в которую можно вставлять различные объекты классов Framework заключенные в теги с описанием свойств.

Строка:

```
<asp:Label ID="Label1" runat="server" Text="Отправить сообщение в глобальную сеть!"
style="font-weight: 700"></asp:Label>
```

описывает объект класса Label с свойством Text которому присвоено значение «Отправить сообщение в глобальную сеть!» и который будет формироваться на стороне сервера (runat=server).

Строка:

```
<asp:TextBox ID="TextBox1" runat="server" Text ="Сообщение!"> </asp:TextBox>
```

формирует текстовое поле для ввода данных через объект TextBox.

Строка:

```
<asp:Button ID="Button1" runat="server" Text="Отправить" OnClick="Button1_Click" />
```

реализует объект Button, событие на нажатие которого будет выполняться на стороне сервера.

Шаг №4. Реализация механизмов событий с объектами

Следующий текст программы в событие на нажатие кнопки на стороне сервера.

```
protected void Button1_Click(object sender, EventArgs e)
{
        Label4.Text = TextBox1.Text;
}
```

Шаг №5. Запуск программы.

Запустите программу и нажмите на кнопку «Отправить».

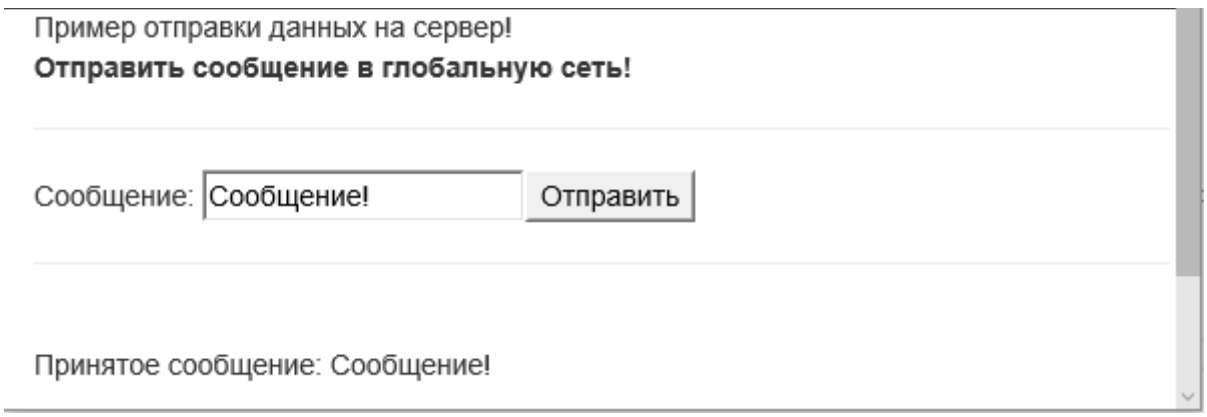


Рис. 10 – Результат работы программы

Задание на лабораторную работу

Вариант	***Задание***
1	Разработать веб-сайт группы с применением технологии ASP.NET состоящей из 1-2 страниц
2	Разработать компонент, выполняющий вычисление на стороне сервера операций +, -, *, / двух чисел введенных пользователем на веб-странице.
3	Разработать компонент, выполняющий конвертацию массы из одной системы в другую.
4	Разработать компонент, выполняющий конвертацию длины из одной системы в другую на стороне сервера.
5	Разработать компонент, выполняющий конвертацию массы из одной системы в другую на стороне сервера.
6	Разработать веб-сайт группы с применением технологии ASP.NET
7	Разработать компонент, выполняющий вычисление на стороне сервера операций +, -, *, / двух чисел введенных пользователем на веб-странице.
8	Разработать компонент, выполняющий конвертацию массы из одной системы в другую.
9	Разработать компонент, выполняющий конвертацию длины из одной системы в другую на стороне сервера.
10	Разработать компонент, выполняющий конвертацию массы из одной системы в другую на стороне сервера.

Контрольные вопросы

1. Что такое динамический сайт?
2. Что такое статический сайт?
3. Что собой представляет объекты ASP.NET?
4. Где могут выполняться механизмы реализации объектов ASP.NET?

Приложение 1 Пример оформления лабораторной работы

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Конструирование и технология радиоэлектронных средств»

Отчет по лабораторной работе №1

по дисциплине: «Информатика, часть 2»

Выполнил:

Факультет: РЭФ

Группа: РК6-62

Студент: Бизяев А.А.

Преподаватель: Бизяев А.А.

Балл:

Оформление	Код	Тест	Итого

Новосибирск, 2016

Задание: написать программу выводящую таблицу Пифагора.

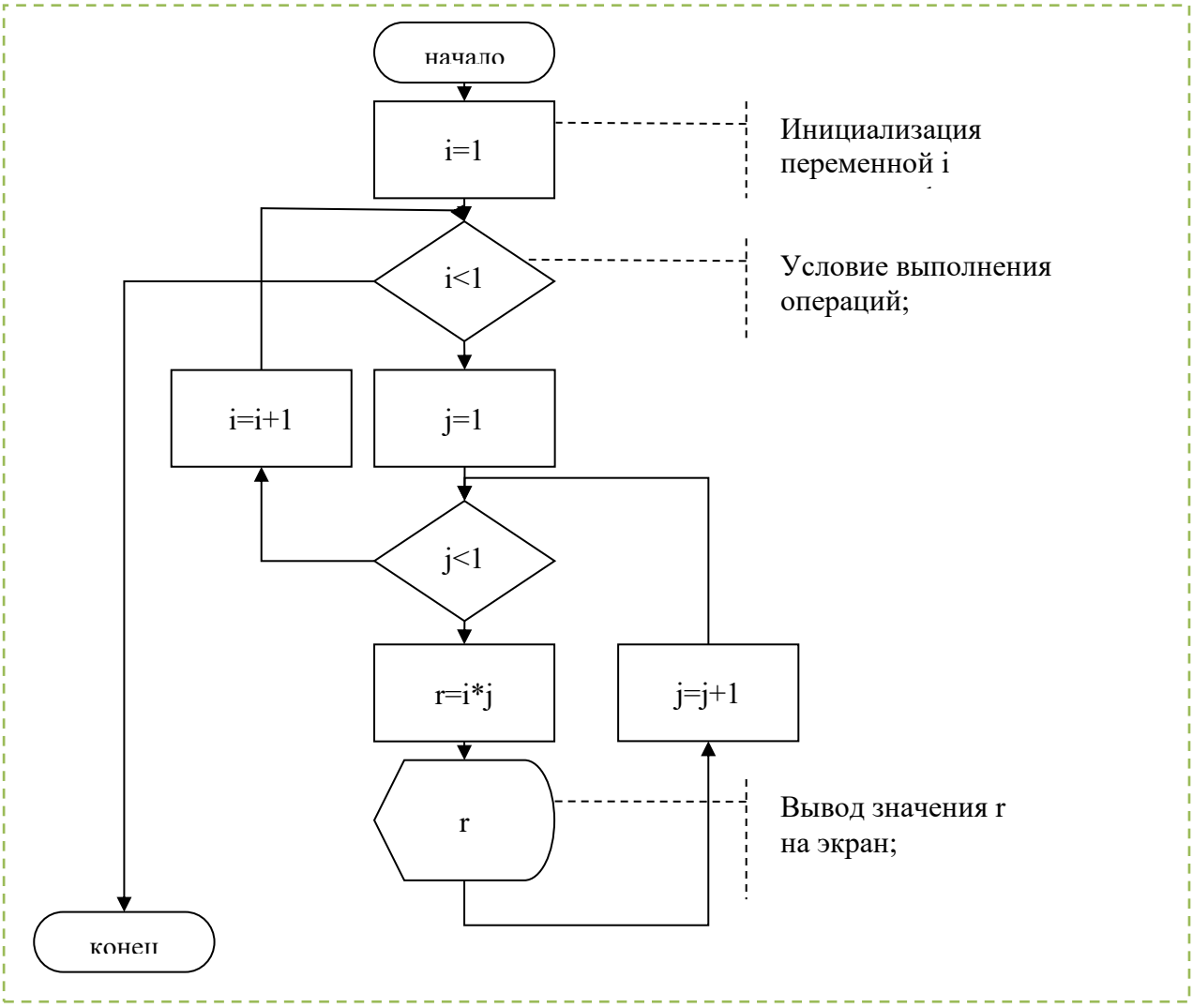


Рис. 1 Блок схема алгоритма программы «Таблица Пифагора», выводящего таблицу умножения на экран

Листинг программы «Таблица Пифагора»:

```
System.Console.Title = "Таблица пифагора";                    // задаем заголовок окна
System.Console.ForegroundColor = ConsoleColor.Green;
for (int i = 1, j = 1, r = 0; i < 10; i++)                    // задаем цикл от 1 до 9
{                                                             // начало тела цикла 1
for (j=1; j < 10; j++)                                        // задаем цикл от 1 до 9
        {                                                     // начало тела цикла 2
          r = i * j;                                          // вычисляем значение r
          System.Console.Write("{0}*{1}={2} \t", i, j, r);    // выводим на экран
        }                                                     // конец тела цикла
        System.Console.WriteLine();          // вставляем новую строку
}                                            // конец тела цикла1
System.Console.Read();                       // ждем, пока пользователь не нажмет на клавишу
```

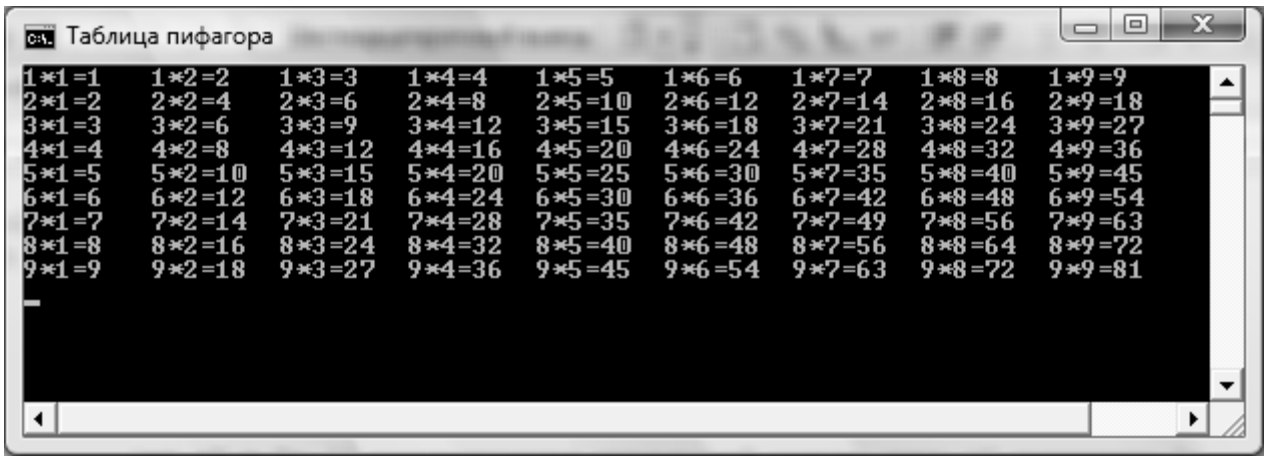


Рис. 1 Результат выполнения программы

Приложение 2 Пример ввода и вывода данных

```
// Пример ввода/вывода строки
String FirstName = Console.ReadLine();    // Считываем строку
String LastName = Console.ReadLine();     // Считываем строку
Console.WriteLine("FirstName: {0} \nLastName: {1}", FirstName, LastName); // Выводим строку

// Пример ввода/вывода числа целого типа
string iStr1 = Console.ReadLine();       // Ситываем строку
string iStr2 = Console.ReadLine();       // Ситываем строку
int i1 = Int32.Parse(iStr1);             // Преобразовываем строку в число
int i2 = Int32.Parse(iStr2);             // Преобразовываем строку в число
int Sum = i1 + i2;                       // Вычислыем сумму двух чисел
Console.WriteLine("i1: {0}; i2: {1}; Sum: {2}", i1, i2, Sum);      // Выводим строку


// Пример ввода/вывода числа вещественного типа
string fStr1 = Console.ReadLine();        // Ситываем строку
string fStr2 = Console.ReadLine();        // Ситываем строку
Double f1 = Double.Parse(fStr1);          // Преобразовываем строку в число
Double f2 = Double.Parse(fStr2);          // Преобразовываем строку в число
Double div = f1/f2;
Console.WriteLine("f1: {0}; f2: {1}; Div: {2}", f1, f2, div);      // Выводим строку

// Пример ввода/вывода символа
char ch = (Char)Console.Read();          // Ситываем символ
Console.WriteLine("Char: {0}", ch);     // Выводим символ

Console.WriteLine("Нажмите любую клавишу");      // Выводим строку
Console.ReadKey(false);                  // Ждем нажатия на клавишу
```

Результат работы программы:

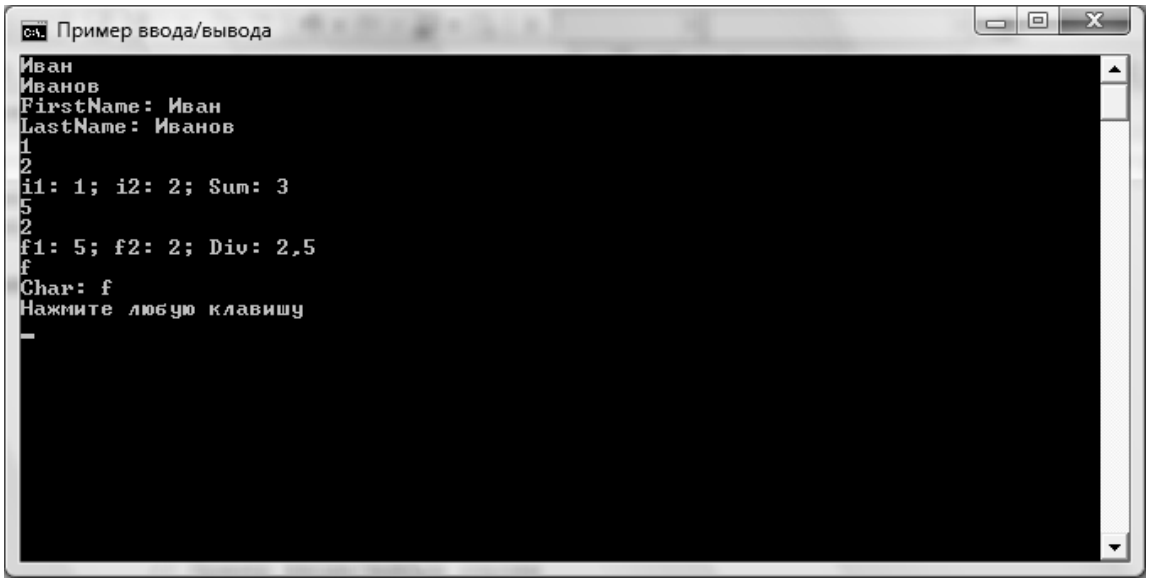


Рис. – Результат работы программы

Приложение 3 Пример поразрядных операций

```
Console.Title = "Пример поразрядных операций";
int val1 = 1;
val1 = val1 << 1;
Console.WriteLine("[1<<1]  \tVal1= {0}", val1);

int val2 = 10;
val2 = val2 >> 1;
Console.WriteLine("[10>>1] \tVal2= {0}", val2);

int val3 = 10;
val3 = val3 | 5;
Console.WriteLine("[10|5]  \tVal3= {0}", val3);

int val4 = 15;
val4 = val4 & 2;
Console.WriteLine("[15&2]  \tVal4: {0}", val4);

int val5 = 10;
val5 = val5 ^ 2;
Console.WriteLine("[10^2]  \tVal5: {0}", val5);

Byte val6 = 1;
val6 = (Byte)~val6;
Console.WriteLine("[~10]   \tVal6: {0}", val6);
```

Результат работы программы:

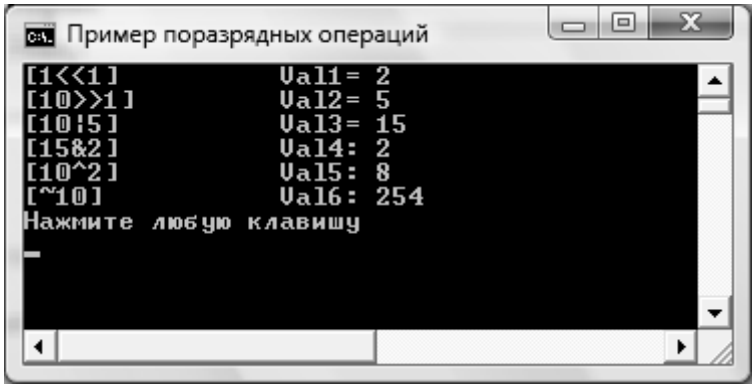


Рис. - Результат работы программы.

Приложение 4

Таблица 4

	Функции
	$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + (-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!}$
	$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots + (-1)^n \cdot \frac{x^{2n}}{(2n)!}$
	$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}$
	$\mathrm{arctg}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \cdots + (-1)^n \cdot \frac{x^{2n+1}}{2n+1}$
	$a^x = 1 + \frac{x \cdot \ln(a)}{1!} + \frac{x^2 \cdot \ln^2(a)}{2!} + \cdots + \frac{x^n \cdot \ln^n(a)}{n!}$
	$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + (-1)^n \cdot \frac{x^{n+1}}{n+1}$
	$\mathrm{ch}(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \cdots + \frac{x^{2n}}{(2n)!}$
	$\mathrm{sh}(x) = \frac{x}{1!} + \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots + \frac{x^{2n+1}}{(2n+1)!}$

Порядок выполнения работы: изучить теорию к лабораторной работе. Получить допуск к выполнению лабораторной работы. Выполнить задание в соответствие с вариантом, в соответствии личным номером студента. Оформить отчет по лабораторной работе в текстовом редакторе Microsoft Word, в котором отобразить результаты выполнения работы. Защитить лабораторную работу.

Литература

1. Шилдт Г. C#. Учебный курс. – СПб.: Питер; Издательская группа BHV, 2003. – 512 с.
2. Шилдт Г. Полный справочник по C#.: Пер. с англ. – М.: Издательский дом "Вильямс", 2004. – 752 с.:
3. Культин Н.Б. C# в задачах и примерах

ИНФОРМАТИКА

Методические указания

Редактор *А.А.Бизяев*
Технический редактор *А.А.Бизяев*
Компьютерная верстка *А.А. Бизяев*

Подписано в печать 00.00.2010. Формат 60 × 84 1/16. Бумага офсетная. Тираж 000 экз. Уч.-изд. л. 1,25. Печ. л. 1,5. Изд. № 00. Заказ № 00000. Цена договорная.

Отпечатано в типографии
Новосибирского государственного технического университета
630092, г. Новосибирск, пр. К. Маркса,20