Министерство образования РФ
Вологодский государственный университет
Машиностроительный техникум

Программирование на языке высокого уровня

Учебное пособие

Специальность 09.02.04 Информационные системы (по отраслям)

Вологда 2015

ББК 34.97

Учебное пособие по дисциплине «Программирование на языке высокого уровня» предназначено для студентов специальности 09.02.04 Информационные системы (по отраслям). Настоящие издание является частью учебно-методического комплекса по дисциплине «Программирование на языке высокого уровня». Учебное пособие включает теоретический материал курса, практические примеры и индивидуальные задания для закрепления и проверки знаний. Содержание заданий относится к основным темам дисциплины, которые предусмотрены для изучения федеральным государственным образовательным стандартом.

Утверждено редакционно-издательским отделом ВоГУ

Составитель: Дегтярев М.Е., преподаватель первой категории МТ ВоГУ

Рецензент: Таланцева Д.Ю., преподаватель первой категории МТ ВоГУ
Македонская С.А., директор НОУ «УЦ Мезон»

СОДЕРЖАНИЕ

1 КЛАССИФИКАЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ 4
2 СВЯЗЬ C# СО СРЕДОЙ .NET FRAMEWORK 7
2.1 Достоинства языка C# 7
2.2 Платформа .Net Framework 8
2.3 Работа с Visual Studio 2010 10
3 ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ C# 15
3.1 Типы данных 16
3.2 Переменные и константы 19
3.3 Комментарии 25
3.4 Управляющие последовательности символов 25
3.5 Преобразования типов 27
3.6 Оператор присваивания 28
3.7 Арифметические операторы, класс Math 30
3.8 Условные операторы 33
3.9 Операторы отношения и логические операторы 36
3.10 Циклы 38
3.11 Операторы перехода 42
4 МАССИВЫ И СТРОКИ 46
4.1 Массивы 46
4.2 Многомерные массивы 50
4.3 Символы и строки 54
4.4 Регулярные выражения 61
4.5 Форматирующие строки 66
5 ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ 70

1 КЛАССИФИКАЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Существуют различные классификации языков программирования. Наиболее распространена классификация, в которой все языки программирования делят на языки низкого и высокого уровня.

Если язык программирования близок к естественному языку, то он называется языком высокого уровня, если ближе к машинным командам, – языком низкого уровня.

В группу языков низкого уровня входят машинные языки и языки символического кодирования: Автокод, Ассемблер. Операторы этого языка – это те же машинные команды, но записанные мнемоническими кодами, а в качестве операндов используются не конкретные адреса, а символические имена. Все языки низкого уровня ориентированы на определенный тип компьютера, т. е. являются машинно-ориентированные.

Машинно-ориентированные языки – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.).

К языкам программирования высокого уровня относят C++, C#, Java, JavaScript, Python, PHP, Ruby, Perl, Паскаль, Delphi и др.. Языкам высокого уровня свойственно умение работать с комплексными структурами данных. В большинстве из них интегрирована поддержка строковых типов, объектов, операций файлового ввода-вывода и т. п..

Эти языки машинно-независимы, т.к. они ориентированы не на систему команд той или иной ЭВМ, а на систему операндов, характерных для записи определенного класса алгоритмов. Однако программы, написанные на языках высокого уровня, занимают больше памяти и медленнее выполняются, чем программы на машинных языках.

Программу, написанную на языке программирования высокого уровня, ЭВМ не понимает, поскольку ей доступен только машинный язык. Поэтому для

перевода программы с языка программирования на язык машинных кодов используют специальные программы – трансляторы.

Существует три вида транслятора: интерпретаторы (это транслятор, который производит пооператорную обработку и выполнение исходного кода программы), компиляторы (преобразует всю программу в модуль на машинном языке, после чего программа записывается в память компьютера и лишь потом исполняется) и ассемблеры (переводят программу, записанную на языке ассемблера, в программу на машинном языке).

Языки программирования также можно разделять на поколения:

– языки первого поколения: машинно-ориентированные с ручным управлением памяти на компьютерах первого поколения;

– языки второго поколения: с мнемоническим представлением команд, так называемые автокоды;

– языки третьего поколения: общего назначения, используемые для создания прикладных программ любого типа. Например, Бейсик, Кобол, Си и Паскаль;

– языки четвертого поколения: усовершенствованные, разработанные для создания специальных прикладных программ, для управления базами данных;

– языки программирования пятого поколения: языки декларативные, объектно-ориентированные и визуальные. Например, Пролог, ЛИСП (используется для построения программ с использованием методов искусственного интеллекта), Си#, Visual Basic, Delphi.

Еще языки программирования подразделяются на процедурные и непроцедурные.

В процедурных языках программа явно описывает действия, которые необходимо выполнить, а результат задается только способом получения его при помощи некоторой процедуры, которая представляет собой определенную последовательность действий.

Среди процедурных языков выделяют в свою очередь структурные и операционные языки. В структурных языках одним оператором записываются це-

лые алгоритмические структуры: ветвления, циклы и т.д. В операционных языках для этого используются несколько операций. Широко распространены следующие структурные языки: Паскаль, Си, Ада. Среди операционных известны Фортран, Бейсик, Фокал.

К непроцедурному программированию относятся функциональные и логические языки.

В функциональных языках программа описывает вычисление некоторой функции. Обычно эта функция задается как композиция других, более простых, те в свою очередь делятся на еще более простые задачи и т.д. Один из основных элементов функциональных языков – рекурсия. Оператора присваивания и циклов в классических функциональных языках нет.

В логических языках программа вообще не описывает действий. Она задает данные и соотношения между ними. После этого системе можно задавать вопросы. Машина перебирает известные и заданные в программе данные и находит ответ на вопрос. Порядок перебора не описывается в программе, а неявно задается самим языком. Классическим языком логического программирования считается Пролог. Программа на Прологе содержит, набор предикатов–утверждений, которые образуют проблемно–ориентированную базу данных и правила, имеющие вид условий.

Можно выделить еще один класс языков программирования – объектно-ориентированные языки высокого уровня. На таких языках не описывают подробной последовательности действий для решения задачи, хотя они содержат элементы процедурного программирования. Объектно-ориентированные языки, благодаря богатому пользовательскому интерфейсу, предлагают человеку решить задачу в удобной для него форме.

2 СВЯЗЬ C# СО СРЕДОЙ .NET FRAMEWORK

2.1 Достоинства языка C#

C# (произносится «си шарп») — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft.

Хейлсберг входил в число ведущих разработчиков одной из самых популярных сред разработки - Delphi. В Microsoft он участвовал в создании версии Java - J++, так что опыта в написании языков и сред программирования ему не занимать. Как отмечал сам Андреас Хейлсберг, C# создавался как язык компонентного программирования, и в этом одно из главных достоинств языка, направленное на возможность повторного использования созданных компонентов. Из других объективных факторов отметим следующие:

– C# создавался параллельно с каркасом Framework .Net и в полной мере учитывает все его возможности - как FCL, так и CLR;

– C# является полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами;

– C# является мощным объектным языком с возможностями наследования и универсализации;

– C# является наследником языков C/C++, сохраняя лучшие черты этих популярных языков программирования. Общий с этими языками синтаксис, знакомые операторы языка облегчают переход программистов от C++ к C#.

– сохранив основные черты своего великого родителя, язык стал проще и надежнее. Простота и надежность, главным образом, связаны с тем, что на C# хотя и допускаются, но не поощряются такие опасные свойства C++ как указатели, адресация, разыменование, адресная арифметика;

– благодаря каркасу Framework .Net, ставшему надстройкой над операционной системой, программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java. Эффективность кода

даже повышается, поскольку исполнительная среда CLR представляет собой компилятор промежуточного языка, в то время как виртуальная Java-машина является интерпретатором байт-кода;

– мощная библиотека каркаса поддерживает удобство построения различных типов приложений на C#, позволяя легко строить Web-службы, другие виды компонентов, достаточно просто сохранять и получать информацию из базы данных и других хранилищ данных;

– реализация, сочетающая построение надежного и эффективного кода, является немаловажным фактором, способствующим успеху C#.

C# спроектирован и разработан специально для применения с .NET Framework.

2.2 Платформа .Net Framework

.NET Framework — служить средой для поддержки разработки и выполнения сильно распределенных компонентных приложений. Она обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows.

Базовые функциональные возможности платформы .NET включают в себя:

1. Возможность обеспечения взаимодействия с существующим программным кодом.

Эта возможность, несомненно, является очень хорошей вещью, поскольку позволяет комбинировать существующие двоичные единицы COM (т.е. обеспечивать их взаимодействие) с более новыми двоичными единицами .NET и наоборот. С выходом версии .NET 4.0 эта возможность стала выглядеть даже еще проще, благодаря добавлению ключевого слова dynamic.

2. Поддержка для многочисленных языков программирования.

Приложения .NET можно создавать с помощью любого множества языков программирования (C#, Visual Basic. F#. S# и т.д.). При этом в .NET код,

написанный на любом языке компилируется в код на промежуточном языке (Intermediate Language - IL).

3. Полная интеграция языков.

В .NET поддерживается межъязыковое наследование, межъязыковая обработка исключений и межъязыковая отладка кода. При этом .NET использует общий исполняющий механизм, основным аспектом которого является хорошо определенный набор типов, который способен понимать каждый, поддерживающий .NET язык.

Так же в .NET был полностью переделан способ разделения кода между приложениями за счет введения понятия сборки (assembly) вместо традиционных библиотек DLL. Сборки обладают формальными средствами для управления версиями и допускают одновременное существование рядом нескольких различных версий сборок.

4. Усовершенствованная поддержка для создания динамических веб-страниц.

Хотя в классической технологии ASP предлагалась довольно высокая степень гибкости, ее все равно не хватало из-за необходимости использования интерпретируемых сценарных языков, а отсутствие объектно-ориентированного дизайна часто приводило к получению довольно запутанного кода ASP. В .NET предлагается интегрированная поддержка для создания веб-страниц с помощью ASP.NET. В случае применения ASP.NET код создаваемых страниц поддается компиляции и может быть написан на любом поддерживающем .NET языке высокого уровня, например, C# или Visual Basic 2010. В новой версии .NET эта поддержка улучшилась еще больше, сделав возможным применение новейших технологий вроде Ajax и jQuery.

5. Эффективный доступ к данным.

Набор компонентов .NET, известный под общим названием ADO.NET, позволяет получать эффективный доступ к реляционным базам данных и многим другим источникам данных. Также предлагаются компоненты, позволяющие получать доступ к файловой системе и каталогам. В частности, в

.NET встроена поддержка XML, позволяющая манипулировать данными, импортируемыми и экспортируемыми на платформы, отличные от Windows.

6. Установка с нулевым воздействием.

Сборки бывают двух типов: разделяемые и приватные. Разделяемые сборки представляют собой обычные библиотеки, доступные всему программному обеспечению, а приватные сборки предназначены для использования только с определенными программами. Приватные сборки являются полностью самодостаточными, поэтому процесс их установки выглядит просто. Никакие записи в системный реестр не добавляются; все нужные файлы просто размещаются в соответствующей папке файловой системы.

7. Visual Studio.

Вместе с .NET поставляется среда разработки Visual Studio, которая способна одинаково хорошо справляться как с кодом на языке C++, C# и Visual Basic, так и с кодом ASP.NET или XML.

2.3 Работа с Visual Studio 2010

Профессиональные разработчики программного обеспечения .NET наверняка располагают самым серьезным в этой сфере продуктом производства Microsoft, который называется Visual Studio. Этот продукт представляет собой самую функционально насыщенную и наиболее приспособленную под использование на предприятиях IDE-среду. Такая мощь, несомненно, имеет свою цену, которая варьируется в зависимости от версии Visual Studio. Как не трудно догадаться, каждая версия поставляется со своим уникальным набором функциональных возможностей.

Visual Studio представляет собой полностью интегрированную среду разработки. Она спроектирована таким образом, чтобы делать процесс написания кода, его отладки и компиляции в сборку для поставки конечным потребителям как можно более простым. На практике это означает, что Visual Studio является очень сложным приложением с многодокументным

интерфейсом, в котором можно делать практически все, что касается разработки кода. Ниже перечислены основные возможности Visual Studio:

1. С помощью встроенного текстового редактора можно подготавливать тексты программ на языке C# (а также Visual Basic и C++). Текстовый редактор обладает довольно мощными возможностями. Например, при вводе текста программы он автоматически компонует его на странице, создавая между строками необходимые отступы, выравнивая открывающие и закрывающие фигурные скобки блоков кода и выделяя ключевые слова цветом. Кроме того, по мере ввода кода он выполняет его проверку на предмет синтаксических ошибок и подчеркивает фрагменты, которые будут вызывать ошибки при компиляции, что также называется отладкой на стадии проектирования. В редакторе реализовано средство IntelliSense, которое обеспечивает автоматическое отображение имен классов, полей или методов при начале их ввода, а также списки параметров, которые поддерживают все доступные перегруженные версии методов при начале ввода параметров для методов.

2. Визуальный редактор форм редактор позволяет размещать желаемые элементы управления для пользовательского интерфейса и доступа к данным в проекте, а Visual Studio затем автоматически добавляет в исходные файлы код на языке C#, который необходим для создания экземпляров этих элементов в проекте. (Это возможно потому, что все элементы управления в .NET представляют собой экземпляры определенных базовых классов.)

3. Вспомогательные окна позволяют просматривать и изменять различные аспекты проекта, вроде классов в исходном коде, а также свойства (и их начальные значения), которые доступны для классов Windows Forms и Web Forms. Вдобавок такие окна могут применяться для указания параметров компиляции, например, того, на какие сборки должен ссылаться код.

4. Возможность компиляции прямо в среде разработки. Вместо того чтобы выполнять компиляцию проекта, запуская компилятор C# из командной строки, можно выбрать соответствующий пункт меню в среде разработки. Visual Studio самостоятельно вызывает компилятор и передает ему все

необходимые параметры командной строки, указывающие, на какие сборки должен ссылаться код и какой вид должна иметь сборка на выходе (например, исполняемый файл или библиотека *.dll). При желании Visual Studio может также автоматически запускать скомпилированный исполняемый файл на выполнение, позволяя проверить его работу.

5. Интегрированный отладчик. Из-за природы программирования код редко, когда выполняется правильно с первого раза. Visual Studio обеспечивает гладкое подключение отладчика, позволяя создавать точки останова и отслеживать значения переменных, не покидая среду разработки.

6. Доступ к другим программам. Visual Studio предоставляет доступ к целому ряду других утилит, которые позволяют просматривать и изменять различные аспекты компьютера или сети, не покидая среды разработки. Благодаря этим инструментам, можно просматривать выполняющиеся службы и активные соединения с базами данных, заглядывать в таблицы на сервере SQL Server и даже посещать веб-сайты с использованием окна Internet Explorer.

7. Интегрированная справочная система MSDN. Visual Studio позволяет получать доступ к документации MSDN прямо из среды IDE. В случае, например, возникновения сомнений по поводу предназначения того или иного ключевого слова во время работы с текстовым редактором, можно выделить это ключевое слово и нажать клавишу <F1>, в результате чего Visual Studio автоматически подключится к MSDN и отобразит подходящие разделы справки. Аналогично, если нужно посмотреть, что означает та или иная ошибка компиляции, потребуется выделять сообщение с ошибкой и нажать <F1>.

8. Visual Studio содержит графические редакторы и конструкторы XML, обеспечивает поддержку разработки программ Windows, ориентированных на мобильные устройства, поддержку разработки программ Microsoft Office и Windows Workflow Foundation, содержит встроенную поддержку рефакторинга кода и инструменты визуального конструирования классов.

9. Visual Studio для языков C#, Visual Basic и J# предлагает 12 возможных видов проектов. Среди них есть пустой проект, в котором

изначально не содержится никакой функциональности; есть также проект, ориентированный на создание Web-служб. Чтобы не усложнять задачу проблемами пользовательского интерфейса, будем рассматривать консольные приложения.

Для создания консольного приложения необходимо выбрать в меню File (Файл) пункт New Project (Создать Проект). Выбрать шаблон Visual C# и далее консольное приложение, в соответствии с рисунком 1.

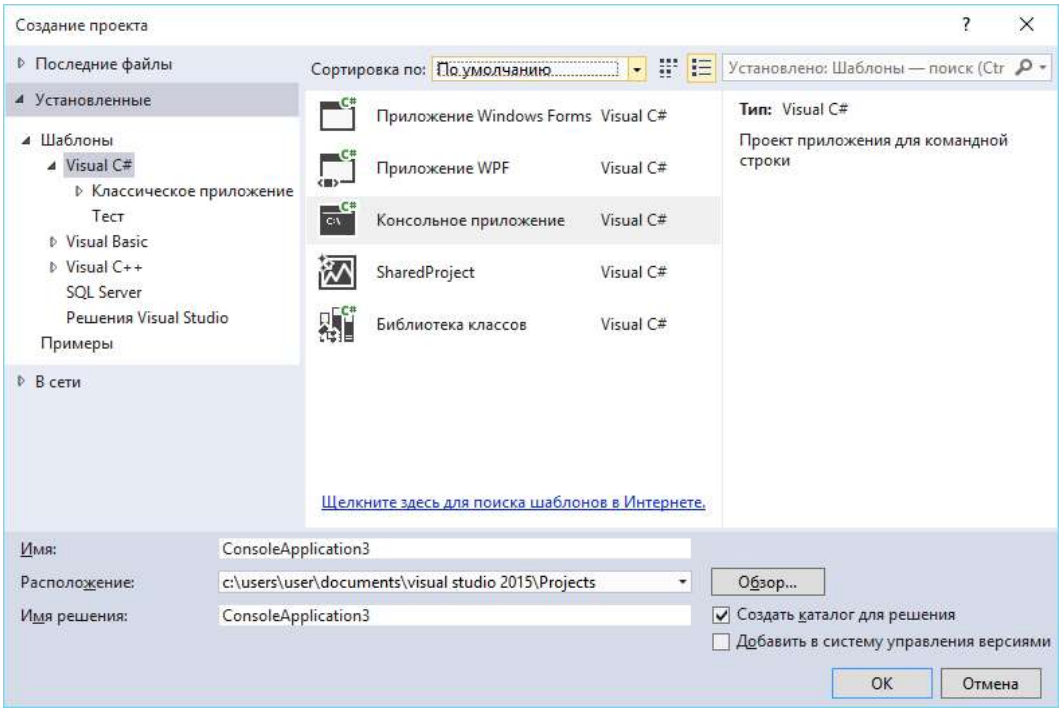

Рисунок 1 – Окно выбора шаблона в Visual Studio

Шаблон консольного приложения выглядит в соответствии с рисунком 2:

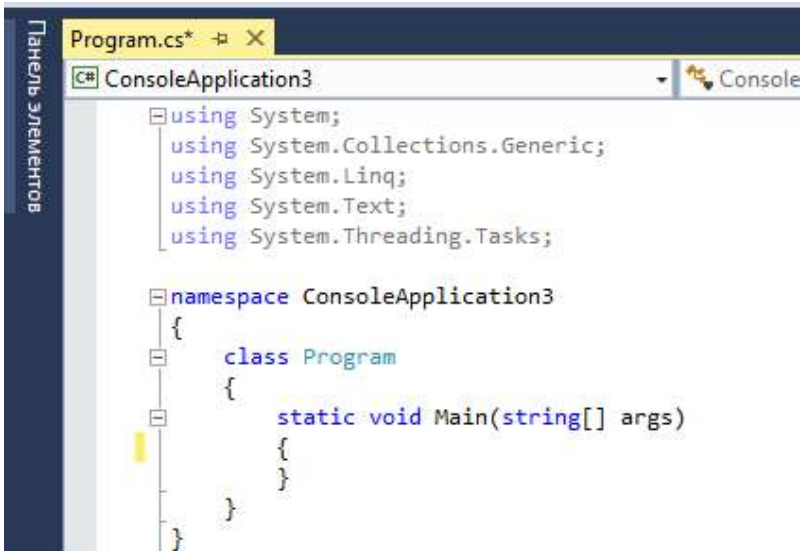

Рисунок 2 – Шаблон консольного приложения

Разберем основные строки данного шаблона.

using System – эта строка означает, что в программе используется пространство имен System. В C# пространство имен определяет область объявлений. Благодаря пространству имен одно множество имен отделяется от других. По существу, имена, объявляемые в одном пространстве имен, не вступают в конфликт с именами, объявляемыми в другом пространстве имен. Ключевое слово using констатирует тот факт, что в программе используются имена в заданном пространстве имен.

С помощью ключевого слова *namespace* объявляется пространство имен, с которым должен быть ассоциирован класс. Весь код в последующих фигурных скобках рассматривается как принадлежащий этому пространству имен. Оператор *using* специфицирует пространство имен, которое должен просматривать компилятор в поисках классов, упомянутых в коде, но не определенных в текущем пространстве имен.

class Program - в этой строке ключевое слово *class* служит для объявления вновь определяемого класса. Класс является основной единицей инкапсуляции в C#, а *Program* — это имя класса. Определение класса начинается с открывающей фигурной скобки "*{*" и оканчивается закрывающей фигурной скобкой "*}*". Элементы, заключенные в эти фигурные скобки, являются членами класса. Не вдаваясь пока что в подробности, достаточно сказать, что в C# большая часть действий, выполняемых в программе, происходит именно в классе.

Метод *Main* является точкой входа в программу. Формально класс, в котором определяется метод *Main()*, называется объектом приложения. Хотя в одном исполняемом приложении допускается иметь более одного такого объекта (это может быть удобно при проведении модульного тестирования), при этом обязательно необходимо информировать компилятор о том, какой из методов *Main()* должен использоваться в качестве входной точки.

3 ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ C#

Напишем простую программу, которая желает доброго утра.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApplication14
{
  class Program
  {
        static void Main(string[] args)
        {
              Console.WriteLine("С добрым утром");
              Console.ReadLine();
        }
  }
}
```

Для просмотра результата необходимо нажать F5 или «начать отладку». Результатом отладки данного приложения будет консолное окно в соответствии с рисунком 3.

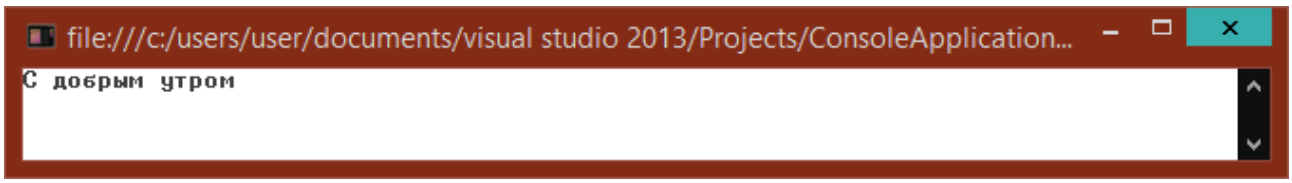

Рисунок 3 – Результат выполнения программы «Доброе утро»

Метод *Console.WriteLine* записывает указанные данные с текущим признаком конца строки в стандартный выходной поток.

Метода *Console.ReadLine* считывает строку символов из стандартного

входного потока.

Далее, будет указываться код, только метода *Main*.

3.1 Типы данных

Типы данных имеют особенное значение в C#, поскольку это строго типизированный язык. Это означает, что все операции подвергаются строгому контролю со стороны компилятора на соответствие типов, причем недопустимые операции не компилируются. Следовательно, строгий контроль типов позволяет исключить ошибки и повысить надежность программ. Для обеспечения контроля типов все переменные, выражения и значения должны принадлежать к определенному типу. Такого понятия, как "бестиповая" переменная, в данном языке программирования вообще не существует. Более того, тип значения определяет те операции, которые разрешается выполнять над ним. Операция, разрешенная для одного типа данных, может оказаться недопустимой для другого.

В C# имеются две общие категории встроенных типов данных: типы значений и ссылочные типы. Они отличаются по содержимому переменной. Концептуально разница между ними состоит в том, что тип значения (value type) хранит данные непосредственно, в то время как ссылочный тип (reference type) хранит ссылку на значение в соответствии с рисунком 4.

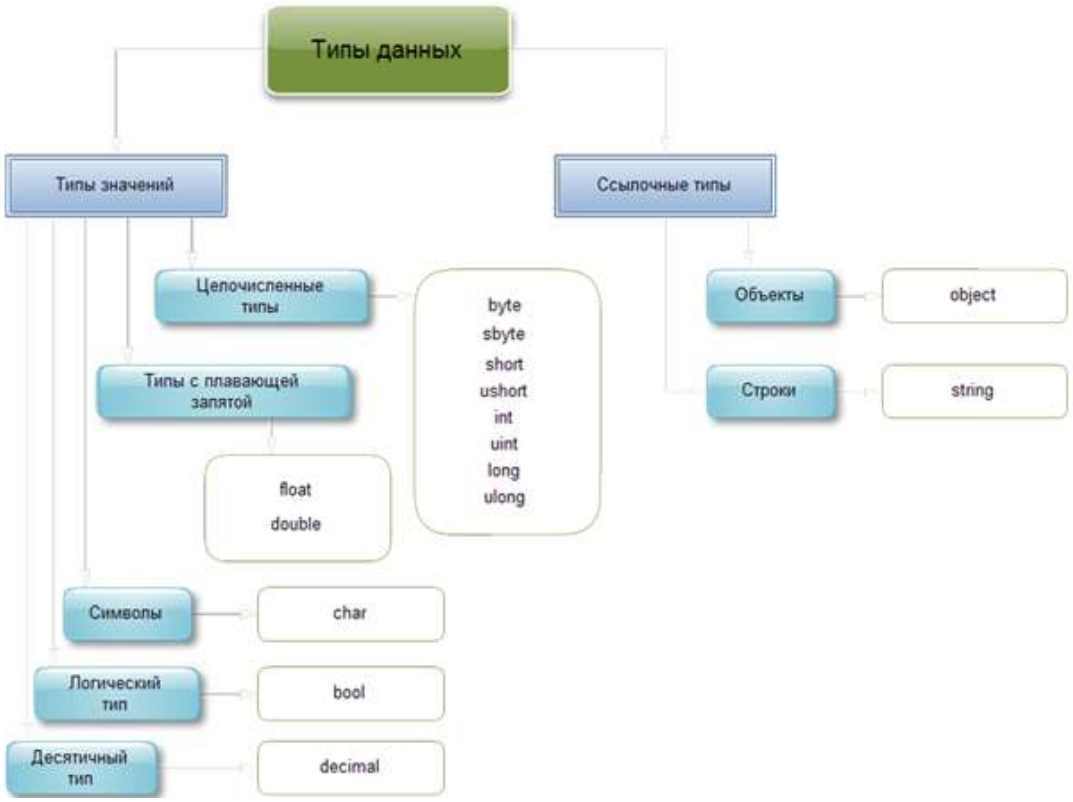

Рисунок 4 – Типы данных C#

Рассмотрим подробнее типы значений:

1. Целочисленные типы.

В C# определены восемь целочисленных типов: *byte, sbyte, short, ushort, int, uint, long* и *ulong*. В таблице 1 представлены их диапазон и разрядность в битах:

Таблица 1 - целочисленные типы данных C#

Тип	Системный тип	Разрядность в битах	Диапазон
byte	System.Byte	8	0:255
sbyte	System.SByte	8	-128:127
short	System.Int16	16	-32768 : 32767
ushort	System.UInt16	16	0 : 65535
int	System.Int32	32	-2147483648 : 2147483647
uint	System.UInt32	32	0 : 4294967295
long	System.Int64	64	-9223372036854775808 : 9223372036854775807
ulong	System.UInt64	64	0 : 18446744073709551615

Всем целочисленным переменным значения могут присваиваться в десятичной или шестнадцатеричной системе обозначений. В последнем случае

требуется префикс 0x. Так число 166 мохет быть записано как 0xA6.

2. Типы с плавающей точкой.

Типы с плавающей точкой позволяют представлять числа с дробной частью. В C# имеются две разновидности типов данных с плавающей точкой: *float* и *double*. Они представляют числовые значения с одинарной и двойной точностью соответственно. Так, разрядность типа *float* составляет 32 бита, что приближенно соответствует диапазону представления чисел от -3.4×10^{38} до $+3.4\times10^{38}$. А разрядность типа *double* составляет 64 бита, что приближенно соответствует диапазону представления чисел от $\pm5.0\times10^{-324}$ до $\pm1.7\times10^{308}$.

Тип данных *float* предназначен для меньших значений с плавающей точкой, для которых требуется меньшая точность. Тип данных *double* больше, чем *float*, и предлагает более высокую степень точности (15 разрядов).

Если нецелочисленное значение жестко кодируется в исходном тексте (например, 12.3), то обычно компилятор предполагает, что подразумевается значение типа *double*. Если значение необходимо специфицировать как *float*, потребуется добавить к нему символ F (или f): float x = 12.3f.

3. Десятичный тип данных.

Для представления чисел с плавающей точкой высокой точности предусмотрен также десятичный тип *decimal*, который предназначен для применения в финансовых расчетах. Этот тип имеет разрядность 128 бит для представления числовых значений в пределах от 1×10^{-28} до $7{,}9\times10^{+28}$. По сравнению с типами данных с плавающей запятой, диапазон значений типа decimal меньше, а точность выше до 28 (а иногда и 29) десятичных разрядов.

4. Символы.

В C# символы представлены не 8-разрядным кодом, как во многих других языках программирования, например C++, а 16-разрядным кодом, который называется юникодом (Unicode). C# определен тип *char*, представляющий 16-разрядные значения без знака в пределах от 0 до 65 535. При этом стандартный набор символов в 8-разрядном коде ASCII является подмножеством юникода в пределах от 0 до 127. Следовательно, символы в коде ASCII по-прежнему

остаются действительными в C#.

Для того чтобы присвоить значение символьной переменной, достаточно заключить это значение (т.е. символ) в одинарные кавычки: *char ch 'Z'*.

Наравне с представлением *char* как символьных литералов, их можно представлять как 4-разрядные шестнадцатеричные значения Unicode (например, '\u0041'), целочисленные значения с приведением (например, (char) 65) или же шестнадцатеричные значения (например, '\x0041').

5. Логический тип данных.

Тип *bool* представляет два логических значения: "истина" и "ложь". Эти логические значения обозначаются в C# зарезервированными словами *true* и *false* соответственно. Следовательно, переменная или выражение типа *bool* будет принимать одно из этих логических значений. Кроме того, в C# не определено взаимное преобразование логических и целых значений. Например, 1 не преобразуется в значение *true*, а 0 — в значение *false*.

3.2 Переменные и константы

Синтаксис объявления переменных в C# выглядит следующим образом:

Тип_данных Идентификатор;

Например: *int i;*

Объявить можно переменную любого действительного типа. Важно подчеркнуть, что возможности переменной определяются ее типом. Например, переменную типа *bool* нельзя использовать для хранения числовых значений с плавающей точкой. Кроме того, тип переменной нельзя изменять в течение срока ее существования. В частности, переменную типа int нельзя преобразовать в переменную типа *char*.

Все переменные в *C#* должны быть объявлены до их применения. Это нужно для того, чтобы уведомить компилятор о типе данных, хранящихся в переменной, прежде чем он попытается правильно скомпилировать любой оператор, в котором используется переменная. Это позволяет также

осуществлять строгий контроль типов в C#.

3.2.1 Инициализация переменной

Задать значение переменной можно, в частности, с помощью оператора присваивания. Кроме того, задать начальное значение переменной можно при ее объявлении. Для этого после имени переменной указывается знак равенства (=) и присваиваемое значение: *int i = 10*. Если две или более переменные одного и того же типа объявляются списком, разделяемым запятыми, то этим переменным можно задать, например, начальное значение: *int x = 5, y = 10, z =15*;.

Инициализация переменных демонстрирует пример обеспечения безопасности C#. Компилятор C# требует, чтобы любая переменная была инициализирована некоторым начальным значением, прежде чем можно было обратиться к ней в какой-то операции. В большинстве современных компиляторов нарушение этого правила определяется и выдается соответствующее предупреждение, но компилятор C# трактует такие нарушения как ошибки. Это предохраняет от нечаянного получения значений «мусора» из памяти, оставшегося там от других программ.

В приведенных выше примерах в качестве инициализаторов переменных использовались только константы, но в C# допускается также динамическая инициализация переменных с помощью любого выражения, действительного на момент объявления переменной:

```
int i1 = 3, i2 = 4;
double result = Math.Sqrt(i1*i1 + i2*i2);
```

В данном примере объявляются три локальные переменные *i1, i2, result,* первые две из которых инициализируются константами, а переменная *result* инициализируется динамически с использованием метода *Math.Sqrt()*, возвращающего квадратный корень выражения. Следует особо подчеркнуть, что в выражении для инициализации можно использовать любой элемент, действительный на момент самой инициализации переменной, в том числе вызовы методов, другие переменные или литералы.

Помимо этого переменную можно задать случайным числом используя класс *Random*:

Random rnd = new Random;

int x = rnd.Next(10,100);

В данном примере переменная *x* будет двузначным числом, т.е. лежать в диапазоне [10,100).

3.2.2 Неявно типизированные переменные

Как пояснялось выше, все переменные в C# должны быть объявлены. Как правило, при объявлении переменной сначала указывается тип, например: *int* или *bool*, а затем имя переменной. Но начиная с версии C# 3.0, компилятору предоставляется возможность самому определить тип локальной переменной, исходя из значения, которым она инициализируется. Такая переменная называется неявно типизированной.

Неявно типизированная переменная объявляется с помощью ключевого слова *var* и должна быть непременно инициализирована. Для определения типа этой переменной компилятору служит тип ее инициализатора, т.е. значения, которым она инициализируется: *var i = 12;* (переменная *i* инициализируется целочисленным литералом). Единственное отличие неявно типизированной переменной от обычной, явно типизированной переменной, — в способе определения ее типа. Как только этот тип будет определен, он закрепляется за переменной до конца ее существования.

3.2.3 Область видимости переменных

Область видимости, или контекст переменной — это часть кода, в пределах которого доступна данная переменная. В общем случае такая область определяется правилами:

– поле, также известное как переменная-член класса, находится в области видимости до тех пор, пока в этой области находится содержащий поле класс;

– локальная переменная находится в области видимости до тех пор, пока закрывающая фигурная скобка не укажет конец блока операторов или метода, в

котором она объявлена;

– локальная переменная, объявленная в операторах цикла *for, while* или подобных им, видима в пределах тела цикла.

3.2.4 Конфликты областей видимости локальных переменных

Использование в больших программах одних и тех же имен переменных в разных частях программы является обычной практикой. Это нормально до тех пор, пока области видимости этих переменных не перекрываются и находятся в совершенно разных частях программы, таким образом исключая любую неоднозначность. Однако следует иметь в виду, что локальные переменные с одним и тем же именем не могут быть объявлены дважды в одном и том же контексте, поэтому вы не сможете поступить так, как показано ниже:

```
int x = 20;
// какой-то код
int x = 30;
```

Рассмотрим следующий пример кода:

```
static void Main(string[] args)
 {
        for (int i=0; i<10; i++)
        {
              Console.Write(" {0}", i);
        }//здесь i покидает область видимости
        Console.WriteLine();
        //можно вновь объявить i
        for (int i = 0; i > -10; i--)
        {
              Console.Write(" {0}", i);
        }//i покидает область видимости
        Console.ReadLine();
       var j = i * i; /*данный код не выполнится, т.к. i не определена в
```

```
текущем контексте*/
 }
```

Важно отметить, что переменная *i* объявляется в этом коде два раза в пределах одного и того же метода. Это можно делать, поскольку переменные *i* объявляются в двух отдельных циклах, поэтому каждая из них локальна в пределах собственного цикла.

3.2.5 Конфликты областей видимости полей и локальных переменных

В некоторых случаях два идентификатора с одинаковыми именами (хотя и не совпадающими полностью уточненными именами) и одинаковой областью видимости можно различить, и тогда компилятор допускает объявление второй переменной. Причина в том, что C# делает принципиальное различие между переменными, объявленными на уровне типа (полями) и переменными, объявленными в методах (локальными). Рассмотрим следующий фрагмент кода:

```
class Program
 {
        static int j = 10;
        static void Main(string[] args)
        {
                int j = 20;
                Console.WriteLine(j);
        }
 }
```

Этот код компилируется, несмотря на то, что здесь в контексте метода *Main()* присутствуют две переменных с именем *j*: переменная *j*, определенная на уровне класса и существующая до тех пор, пока не будет уничтожен класс (когда завершится метод *Main()*, а вместе с ним и программа), и переменная *j*, определенная внутри *Main()*. В данном случае новая переменная с именем *j*, объявленная в методе *Main()*, скрывает переменную уровня класса с тем же именем. Поэтому когда вы запустите этот код, на дисплее будет отображено число 20.

3.2.6 Константы

Как следует из названия, константа — это переменная, значение которой не меняется за время ее существования. Предваряя переменную ключевым словом *const* при ее объявлении и инициализации, вы объявляете ее как константу:

const int a = 100;

Это значение не может быть изменено.

Ниже перечислены основные характеристики констант:

– они должны инициализироваться при объявлении, и однажды присвоенные им значения никогда не могут быть изменены;

– значение константы должно быть вычислено во время компиляции. Таким образом, инициализировать константу значением, взятым из другой переменной, нельзя. Если все-таки нужно это сделать, используйте поля только для чтения;

– константы всегда неявно статические. Однако вы не должны (и фактически не можете) включать модификатор static в объявление константы.

Использование констант в программах обеспечивает, по крайней мере, три преимущества:

– константы облегчают чтение программ, заменяя "магические" числа и строки читаемыми именами, назначение которых легко понять;

– константы облегчают модификацию программ. Например, предположим, что в программе C# имеется константа SalesTax (налог с продаж), которой присвоено значение 6 процентов. Если налог с продаж когда-нибудь изменится, вы можете модифицировать все вычисления налога, просто присвоив новое значение этой константе, и не понадобится просматривать код в поисках значений и изменять каждое из них, надеясь, что оно нигде не будет пропущено;

– константы позволяют избежать ошибок в программах. Если попытаться присвоить новое значение константе где-то в другом месте программы, а не там, где она объявлена, компилятор выдаст сообщение об ошибке.

3.3 Комментарии

Комментарии являются немаловажной частью любого языка программирования, т.к. позволяют удобно пояснять различные участки кода. В C# используются традиционные комментарии в стиле C — однострочные (//...) и многострочные (*/* . . . **/):*

//Это однострочный комментарий

*/*Это уже*

Многострочный комментарий/*

Все, что находится в однострочном комментарии — от // до конца строки — игнорируется компилятором, как и весь многострочный комментарий, расположенный между /* и */. Очевидно, что в многострочном комментарии не может присутствовать комбинация */, поскольку она будет трактоваться как конец комментария.

Многострочный комментарий можно помещать в одну строку кода:

Console.WriteLine (/ Здесь идет комментарий! */ "Это скомпилируется");*

Встроенные комментарии вроде этого нужно применять осторожно, потому что они могут ухудшить читабельность кода. Однако они удобны при отладке, скажем, когда необходимо временно попробовать запустить программу с указанным другим значением:

*DoSomethingMethod (Width, /*Height*/ 100);*

Символы комментария, включенные в строковый литерал, конечно же, трактуются как обычные символы:

string s = "/ Это просто нормальная строка */";*

3.4 Управляющие последовательности символов

Большинство печатаемых символов достаточно заключить в одинарные кавычки, но набор в текстовом редакторе некоторых символов, например возврата каретки, вызывает особые трудности. Кроме того, ряд других символов, в том числе одинарные и двойные кавычки, имеют специальное

назначение в C#, поэтому их нельзя использовать непосредственно. По этим причинам в C# предусмотрены специальные управляющие последовательности символов представленные в таблице 2.

Таблица 2 – управляющие последовательности C#

Управляющая последовательность	Описание
\a	Звуковой сигнал (звонок)
\b	Возврат на одну позицию
\f	Перевод страницы (переход на новую страницу)
\n	Новая строка (перевод строки)
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\0	Пустой символ
\'	Одинарная кавычка
\"	Двойная кавычка
\\	Обратная косая черта

Рассмотрим пример использования управляющих последовательностей:

```
{
  // Используем перенос строки
  Console.WriteLine("Первая строка\nВторая строка\nТретья строка\n");
  // Используем горизонтальную табуляцию
  Console.WriteLine("Один\tДва\tТри");
  Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 5.

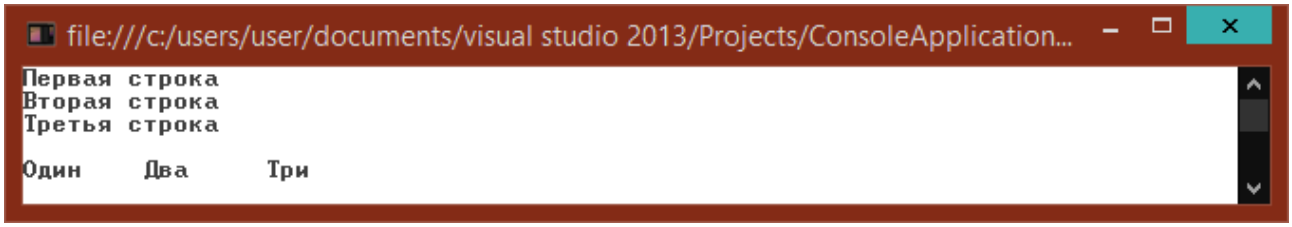

Рисунок 5 – пример действия управляющих последовательностей

3.5 Преобразования типов

В C# преобразования типов данных делятся на неявные и явные. К неявным относятся те преобразования, результат выполнения которых всегда успешен и не приводит к потере точности данных. Неявные преобразования выполняются автоматически. Для арифметических данных это означает, что в неявных преобразованиях диапазон типа назначения содержит в себе диапазон исходного типа. Например, преобразование из типа *byte* в тип *int* относится к неявным, поскольку диапазон типа *byte* является подмножеством диапазона *int*. Это преобразование всегда успешно и не может приводить к потере точности. Заметьте, преобразования из целочисленных типов к типам с плавающей точкой относятся к неявным. Хотя здесь и может происходить некоторое искажение значения, но точность представления значения сохраняется, например, при преобразовании из *long* в *double* порядок значения остается неизменным.

К явным относятся разрешенные преобразования, успех выполнения которых не гарантируется или может приводить к потере точности. Такие потенциально опасные преобразования должны быть явно заданы программистом. Преобразование из типа *int* в тип *byte* относится к явным, поскольку оно небезопасно и может приводить к потере значащих цифр. Заметьте, не для всех типов существуют явные преобразования. В этом случае требуются другие механизмы преобразования типов, которые будут рассмотрены позже.

3.5.1. Преобразования строкового типа

Важным классом преобразований являются преобразования в строковый тип и наоборот. Преобразования в строковый тип всегда определены, поскольку, напомню, все типы являются потомками базового класса *Object*, следовательно, обладают методом *ToString()*. Для встроенных типов определена подходящая реализация этого метода. В частности, для всех подтипов арифметического типа метод *ToString()* возвращает в подходящей форме строку, задающую соответствующее значение арифметического типа. Заметьте, метод *ToString()*

можно вызывать явно, но, если явный вызов не указан, то он будет вызываться неявно, всякий раз, когда по контексту требуется преобразование к строковому типу. Рассмотрим соответствующий пример:

```
{
  string s = "Владимир Петров ";
  string s1 = "Возраст: ";
  int x = 27;
  s = s + s1 + x;
  Console.WriteLine( s);
  Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 6.

Рисунок 6 – пример преобразования в строковый тип

3.5.2 Преобразования из строкового типа

Явных преобразований между арифметикой и строками не существует. Но в C# имеются соответсвенные механизмы. Для этой цели можно использовать методы класса *Convert*, встроенного в пространство имен *System:*

```
{
Console.WriteLine("Введите возраст");
int x = Convert.ToInt32(Console.ReadLine());
}
```

3.6 Оператор присваивания

Оператор присваивания обозначается одиночным знаком равенства (=). В C# оператор присваивания действует таким же образом, как и в других языках программирования. Ниже приведена его общая форма:

Имя_переменной = выражение;

Здесь имя_переменной должно быть совместимо с типом выражения. У оператора присваивания имеется одна интересная особенность, о которой вам будет полезно знать: он позволяет создавать цепочку операций присваивания. Рассмотрим следующий фрагмент кода:

```
int x, y, z;
x = y = z = 10;
```

В приведенном выше фрагменте кода одно и то же значение 10 задается для переменных x, y и z с помощью единственного оператора присваивания. Это значение присваивается сначала переменной z, затем переменной y и, наконец, переменной x. Такой способ присваивания "по цепочке" удобен для задания общего значения целой группе переменных.

В C# предусмотрены специальные составные операторы присваивания, упрощающие программирование некоторых операций присваивания. Например:

```
x = x + 1;
```

Можно переписать следующим образом

```
x += 1;
```

Сокращенные операторы присваивания представлены в таблице 3.

Таблица 3 – сокращенные операторы присваивания

Оператор	Аналог
+=	x = x + 1
-=	x = x - 1
*=	x = x * 1
/=	x = x / 1
%=	x = x % 1
|=	x = x | 1
^=	x = x ^ 1

Составные операторы присваивания записываются более кратко, чем их несоставные эквиваленты. Поэтому их иногда еще называют укороченными операторами присваивания.

У составных операторов присваивания имеются два главных

преимущества. Во-первых, они более компактны, чем их "несокращенные" эквиваленты. И во-вторых, они дают более эффективный исполняемый код, поскольку левый операнд этих операторов вычисляется только один раз. Именно по этим причинам составные операторы присваивания чаще всего применяются в программах, профессионально написанных на C#.

3.7 Арифметические операторы, класс Math

3.7.1 Для выполнения простых арифметических действий в C# существуют операторы, которые представлены в таблице 4.

Таблица 4 – арифметические операторы

Оператор	Действие
+	Сложение
-	Вычитание, унарный минус
*	Умножение
/	Деление
%	Деление по модулю
--	Декремент
++	Инкремент

Операторы +, -, * и / действуют так, как предполагает их обозначение. Их можно применять к любому встроенному числовому типу данных.

Действие арифметических операторов не требует особых пояснений, за исключением следующих особых случаев. Прежде всего, не следует забывать, что когда оператор / применяется к целому числу, то любой остаток от деления отбрасывается; например, результат целочисленного деления 13/3 будет равен 4. Остаток от этого деления можно получить с помощью оператора деления по модулю (%), который иначе называется оператором вычисления остатка. Он дает остаток от целочисленного деления. Например, 13 % 3 равно 1. В C# оператор % можно применять как к целочисленным типам данных, так и к типам с плавающей точкой. Поэтому 13.0 % 3.0 также равно 1. В этом

отношении C# отличается от языков C и C++, где операции деления по модулю разрешаются только для целочисленных типов данных. Давайте рассмотрим следующий пример:

```
{
 int n1, n2;
 float f1, f2;
 n1 = 13 / 3;
 n2 = 13 % 3;
 f1 = 13.0f / 3.0f;
 f2 = 13.0f % 3.0f;
 Console.WriteLine("Результат и остаток от деления 13 на 3: {0} {1}",n1,n2);
 Console.WriteLine("Результат и остаток от деления 13.0 на 3.0: {0:f3} {1}", f1, f2);
 Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 7.

Рисунок 7 – пример выполнения операций деления

Оператор инкремента (*++*) увеличивает свой операнд на 1, а оператор декремента (--) уменьшает операнд на 1. Следовательно, операторы:

x++;x--;

равнозначны операторам:

x = x +1; x = x -1;

3.7.2 Для проведения более сложных арифметических вычислений можно применять класс *Math*, который предоставляет константы и статические методы для тригонометрических, логарифмических и иных общих математических

функций. Наиболее распространенные методы класса *Math*, представлены в таблице 5.

Таблица 5 – методы класса Math

Имя	Описание
Abs	Возвращает абсолютное значение числа
Acos	Возвращает угол, косинус которого равен указанному числу.
Asin	Возвращает угол, синус которого равен указанному числу.
Atan	Возвращает угол, тангенс которого равен указанному числу.
Cos	Возвращает косинус указанного угла.
Exp	Возвращает e, возведенное в указанную степень.
Log(Double)	Возвращает натуральный логарифм (с основанием e) указанного числа.
Log(Double, Double)	Возвращает логарифм указанного числа в системе счисления с указанным основанием.
Log10	Возвращает логарифм с основанием 10 указанного числа.
Max	Возвращает большее из двух чисел одинакого типа
Min	Возвращает меньшее из двух чисел одинакого типа
Pow	Возвращает указанное число, возведенное в указанную степень.
Round	Округляет заданное число до ближайшего целого.
Sin	Возвращает синус указанного угла.
Sqrt	Возвращает квадратный корень из указанного числа.
Tan	Возвращает тангенс указанного угла.

Помимо методов, класс *Math* содержит два поля, которые представлены в таблице 6.

Таблица 6 – поля класса Math

Имя	Описание
E	Представляет основание натурального логарифма, определяемое константой e.
PI	Представляет отношение длины окружности к ее диаметру, определяемое константой π.

Расмотрим пример в котором необходимо найти значение функции $y = \frac{3x^2}{\sqrt{x-2}}$при аргументе вводимом с клавиатуры:

```
{
        Console.WriteLine("Введите x");
        double x = Convert.ToDouble(Console.ReadLine());
```

```
double y = 3*Math.Pow(x,3)/Math.Sqrt(x-2);
Console.WriteLine("При x={0} y={1}",x,y);
Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 8.

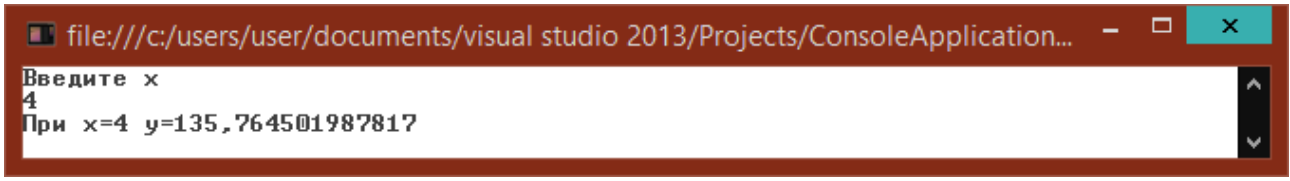

Рисунок 8 – результат использование класса Math

3.8 Условные операторы

Условные операторы позволяют управлять потоком выполнения программы, чтобы не выполнялась каждая строка кода, как она следует в программе. Давайте рассмотрим условные операторы языка C#/

3.8.1 Оператор *if*

Для организации условного ветвления язык C# унаследовал от C и C++ конструкцию *if...else*. Синтаксис оператора выглядит следующим образом:

```
if (условие)
{операторы};
else
{операторы};
```

Если по каждому из условий нужно выполнить более одного оператора, эти операторы должны быть объединены в блок с помощью фигурных скобок *{...}*. (Это также касается других конструкций C#, в которых операторы могут быть объединены в блок — таких как циклы *for* и *while*).

В операторе *if* могут применяться сложные выражения, и он может содержать операторы *else*, обеспечивая выполнение более сложных проверок. При построении сложных выражений в C# используется вполне ожидаемый набор логических операторов. Рассмотрим следующий пример:

```
{
  Console.WriteLine("Введите число ");
        double x = Convert.ToDouble(Console.ReadLine());
  if (x < 0)
        Console.WriteLine("x – отрицательное число");
  else if (x >0)
        Console.WriteLine("x – положительное число");
  else Console.WriteLine("x равен нулю");
  Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 9.

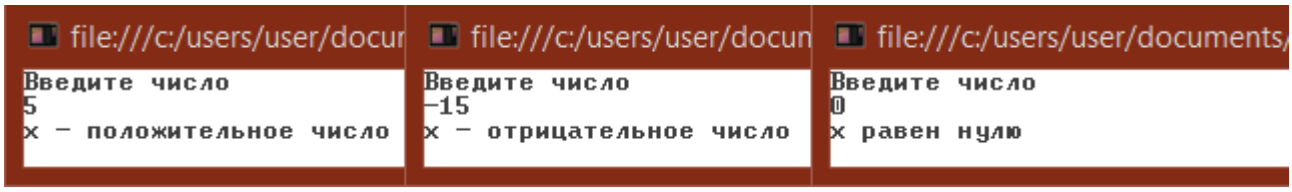

Рисунок 9 – применение условного оператора if

3.8.2 Оператор switch

Вторым оператором выбора в C# является оператор *switch*, который обеспечивает многонаправленное ветвление программы. Следовательно, этот оператор позволяет сделать выбор среди нескольких альтернативных вариантов дальнейшего выполнения программы. Несмотря на то что многонаправленная проверка может быть организована с помощью последовательного ряда вложенных операторов *if*, во многих случаях более эффективным оказывается применение оператора *switch*. Этот оператор действует следующим образом. Значение выражения последовательно сравнивается с константами выбора из заданного списка. Как только будет обнаружено совпадение с одним из условий выбора, выполняется последовательность операторов, связанных с этим условием. Ниже приведена общая форма оператора *switch*:

```
switch(выражение)
{
```

case константа1:

последовательность операторов

break;

case константа2:

последовательность операторов

break;

...

default:

последовательность операторов

break;

}

Оператор *switch...case* в C# немного безопаснее, чем его эквивалент C++. В частности, он запрещает "сквозные" условия почти во всех случаях. Это значит, что если часть *case* вызывается в начале блока, то фрагменты кода за последующими частями *case* не могут быть выполнены, если только не используется явно оператор *goto* для перехода к ним. Компилятор обеспечивает это ограничение за счет того, что требует, чтобы за каждой частью *case* следовал оператор *break*, в противном случае он выдает ошибку.

Важно отметить, что заданное выражение в операторе *switch* должно быть целочисленного типа (*char*, *byte*, *short* или *int*) или же строкового. А выражения других типов, например с плавающей точкой, в операторе *switch* не допускаются. Зачастую выражение, управляющее оператором *switch*, просто сводится к одной переменной. Кроме того, константы выбора должны иметь тип, совместимый с типом выражения. В одном операторе *switch* не допускается наличие двух одинаковых по значению констант выбора. Рассмотрим пример:

```
{
  Console.WriteLine("Введите язык (C#, VB или C++)");
  string myLanguage = Console.ReadLine();
  switch (myLanguage)
  {
```

```
            case "C#":
            Console.WriteLine("Вы выбрали язык C#");
            break;
            case "VB":
            Console.WriteLine("Вы выбрали язык Visual Basic");
            break;
            case "C++":
            Console.WriteLine("Вы выбрали язык C++");
            break;
            default:
            Console.WriteLine("Такой язык я не знаю");
            break;
        }
    }
```

Результат выполнения программы представлен в соответствии с рисунком 10.

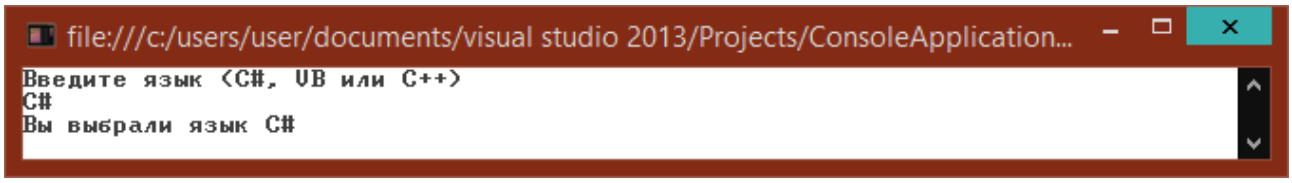

Рисунок 10 – применение оператора выбора switch

3.9 Операторы отношения и логические операторы

В обозначениях оператор отношения и логический оператор термин отношения означает взаимосвязь, которая может существовать между двумя значениями, а термин логический — взаимосвязь между логическими значениями "истина" и "ложь". И поскольку операторы отношения дают истинные или ложные результаты, то они нередко применяются вместе с логическими операторами. Именно по этой причине они и рассматриваются совместно.

Результатом выполнения оператора отношения или логического оператора

является логическое значение типа *bool*.

Операторы отношения и логические операторы представлены в таблице 7.

Таблица 7 – Операторы отношения и логические операторы

Операторы отношения C#		Логические операторы C#	
Оператор	Значение	Оператор	Значение
==	Равно	&	И
!=	Не равно	|	ИЛИ
>	Больше	^	Исключающее ИЛИ
<	Меньше	&&	Укороченное И
>=	Больше или равно	||	Укороченное ИЛИ
<=	Меньше или равно	!	НЕ

В целом, объекты можно сравнивать на равенство или неравенство, используя операторы отношения == и *!=*. А операторы сравнения <, >, <= или >= могут применяться только к тем типам данных, которые поддерживают отношение порядка. Следовательно, операторы отношения можно применять ко всем числовым типам данных. Но значения типа *bool* могут сравниваться только на равенство или неравенство, поскольку истинные (*true*) и ложные (*false*) значения не упорядочиваются. Пример:

```
{
short d = 10, f = 12;
bool var1 = true, var2 = false;
if (d < f)
        Console.WriteLine("d < f");
if (d != f)
        Console.WriteLine("d != f");
if (var1 & var2)
        Console.WriteLine("Данный текст не выведется");
if (!(var1 & var2))
        Console.WriteLine("!(var1 & var2) = true");
        if (var1 | var2)
        Console.WriteLine("var1 | var2 = true");
```

Console.ReadLine();

}

Результат выполнения программы представлен в соответствии с рисунком 11.

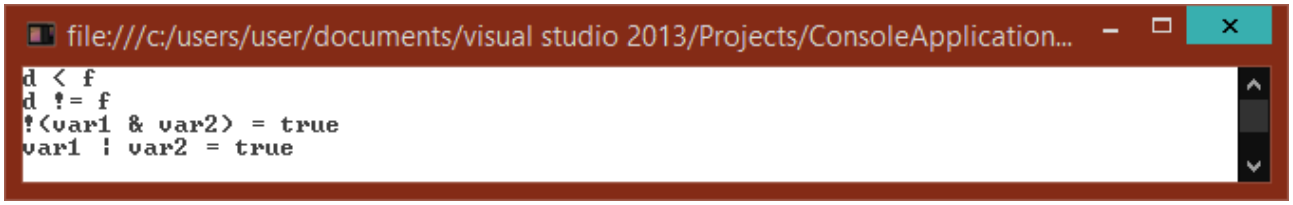

Рисунок 11 – применение операторов отношения и логических операторов

В C# предусмотрены также специальные, укороченные, варианты логических операторов И и ИЛИ, предназначенные для получения более эффективного кода. Поясним это на следующих примерах логических операций. Если первый операнд логической операции И имеет ложное значение (*false*), то ее результат будет иметь ложное значение независимо от значения второго операнда. Если же первый операнд логической операции ИЛИ имеет истинное значение (*true*), то ее результат будет иметь истинное значение независимо от значения второго операнда. Благодаря тому что значение второго операнда в этих операциях вычислять не нужно, экономится время и повышается эффективность кода.

Укороченная логическая операция И выполняется с помощью оператора *&&*, а укороченная логическая операция ИЛИ — с помощью оператора ||. Этим укороченным логическим операторам соответствуют обычные логические операторы *&* и |. Единственное отличие укороченного логического оператора от обычного заключается в том, что второй его операнд вычисляется только по мере необходимости.

3.10 Циклы

3.10.1 Цикл for

Цикл *for* в C# предоставляет механизм итерации, в котором определенное условие проверяется перед выполнением каждой итерации. Синтаксис этого

оператора следующий:

for (инициализатор; условие; итератор)
{операторы};

Инициализатор – это выражение, вычисляемое перед первым выполнением тела цикла (обычно инициализация локальной переменной в качестве счетчика цикла). Инициализация, как правило, представлена оператором присваивания, задающим первоначальное значение переменной, которая выполняет роль счетчика и управляет циклом.

Условие – это выражение, проверяемое перед каждой новой итерацией цикла (должно возвращать true, чтобы была выполнена следующая итерация).

Итератор – выражение, вычисляемое после каждой итерации (обычно приращение значения счетчика цикла).

Обратите внимание на то, что эти три основные части оператора цикла *for* должны быть разделены точкой с запятой. Выполнение цикла *for* будет продолжаться до тех пор, пока проверка условия дает истинный результат. Как только эта проверка даст ложный результат, цикл завершится, а выполнение программы будет продолжено с оператора, следующего после цикла *for*.

Расмотрим пример программы, которая выведет степени двойки от первой до десятой:

```
{
 for (int i = 1; i <= 10; i++)
        Console.Write(Math.Pow(2, i) + " ");
  Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 12.

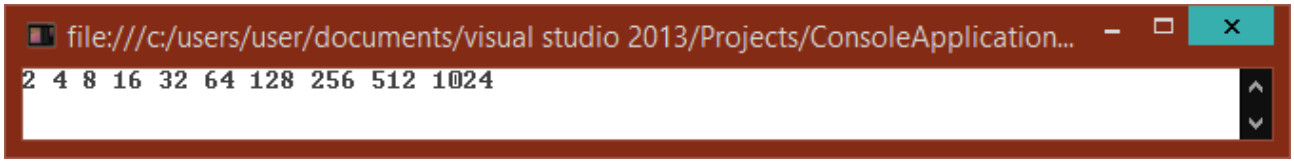

Рисунок 12 – применение цикла for

3.10.2 Цикл while

Подобно *for*, *while* также является циклом с предварительной проверкой. Синтаксис его аналогичен, но циклы *while* включают только одно выражение:

while(условие)

{операторы};

В этом цикле оператор выполняется до тех пор, пока условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла.

Как и в цикле *for*, в цикле *while* проверяется условное выражение, указываемое в самом начале цикла. Это означает, что код в теле цикла может вообще не выполняться, а также избавляет от необходимости выполнять отдельную проверку перед самим циклом.

Также рассмотрим пример со степенями двойки:

```
{
  int i = 0;
        while (i<10)
        {
          i++;
          Console.Write(Math.Pow(2, i)+" ");
        }
  Console.ReadLine();
}
```

3.10.3 Цикл do. . . while.

Цикл *do...while* в C# — это версия *while* с постпроверкой условия. Это значит, что условие цикла проверяется после выполнения тела цикла. Следовательно, циклы *do...while* удобны в тех ситуациях, когда блок операторов должен быть выполнен как минимум однажды. Ниже приведена общая форма оператора цикла *do-while*:

do

{операторы;}

while (условие);

При наличии лишь одного оператора фигурные скобки в данной форме записи необязательны. Тем не менее они зачастую используются для того, чтобы сделать конструкцию *do-while* более удобоночитаемой и не путать ее с конструкцией цикла *while.* Цикл *do-while* выполняется до тех пор, пока условное выражение истинно. В качестве примера использования цикла *do-while* расмотрим вновь расмотрим возведение двойки в степени от 1 до 10:

```
{
  int i=0;
  do
        {
          i++;
          Console.Write(Math.Pow(2, i)+" ");
        }
  while (i<10);
  Console.ReadLine();
}
```

3.10.4 Цикл foreach.

Цикл foreach служит для циклического обращения к элементам коллекции, представляющей собой группу объектов. В C# определено несколько видов коллекций, каждая из которых является массивом. Ниже приведена общая форма оператора цикла foreach:

foreach (тип имя_переменной_цикла in коллекция)

оператор;

Оператор цикла *foreach* действует следующим образом. Когда цикл начинается, первый элемент массива выбирается и присваивается переменной цикла. На каждом последующем шаге итерации выбирается следующий элемент массива, который сохраняется в переменной цикла. Цикл завершается, когда все элементы массива окажутся выбранными.

В качестве примера рассмотрим программу вычисления квадратов числа

пяти случайгых двузначных чисел:

```
{
 //объявляем массив
 int[] myArr = new int[5];
 Random ran = new Random();
 //инициализируем массив
 for (int i = 0; i < 5; i++)
        {
            myArr[i] = ran.Next(10, 100);
        }
 //вычисляем квадрат каждого элемента массива
 foreach (int el in myArr)
        Console.WriteLine("{0} в квадрате равно {1}",el,el*el);
 Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 13.

Рисунок 13 – применение цикла foreach

3.11 Операторы перехода

В C# есть несколько операторов, изменяющих естественный порядок выполнения команд: оператор безусловного перехода *goto*, оператор выхода *break*.

3.11.1 Оператор goto

Оператор *goto* представляет собой оператор безусловного перехода. Когда в

программе встречается оператор *goto*, ее выполнение переходит непосредственно к тому месту, на которое указывает этот оператор. Для выполнения оператора *goto* требуется метка — действительный в C# идентификатор с двоеточием. Метка должна находиться в том же методе, где и оператор *goto*, а также в пределах той же самой области действия.

Синтаксис оператора имеет вид:

goto <метка>;

В теле той же функции должна присутствовать ровно одна конструкция вида:

<метка>: <оператор>;

В качестве примера реализуем код нахождения степеней двойки:

```
{
  int i = 1;
  Link1:
  Console.Write(Math.Pow(2, i) + " ");
  i++;
  if (i<=10) goto Link1;
  Console.ReadLine();
}
```

3.11.2 Оператор break

С помощью оператора *break* можно специально организовать немедленный выход из цикла в обход любого кода, оставшегося в теле цикла, а также минуя проверку условия цикла. Когда в теле цикла встречается оператор *break*, цикл завершается, а выполнение программы возобновляется с оператора, следующего после этого цикла. Оператор *break* можно применять в любом цикле, предусмотренном в C#. Если оператор *break* применяется в целом ряде вложенных циклов, то он прерывает выполнение только самого внутреннего цикла.

```
{
  // В данном цикле выведутся степени двойки от 1 до 5 вместо 10
```

```
 for (int i = 1; i < 10; i++)
 if (i <= 5)
        Console.Write(Math.Pow(2, i) + " ");
 else break;
 Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 14.

Рисунок 14 – применение оператора break

3.11.3 Оператор continue

Оператор перехода к следующей итерации цикла *continue* пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации (повторение тела цикла). В циклах *while* и *do-while* оператор *continue* вызывает передачу управления непосредственно условному выражению, после чего продолжается процесс выполнения цикла. А в цикле for сначала вычисляется итерационное выражение, затем условное выражение, после чего цикл продолжается. В качестве примера выведем двойку в степенях кратных двум:

```
{
 for (int i = 1; i <= 10; i++)
        {
            if (i % 2 != 0) continue;
            Console.Write(Math.Pow(2, i) + " ");
        }
 Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 15.

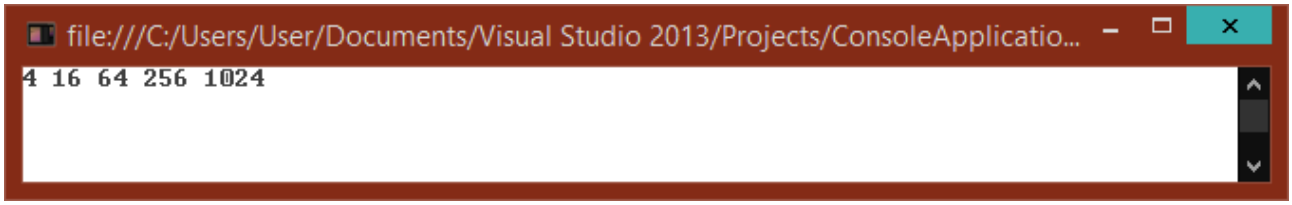

Рисунок 15 – применение оператора continue

4 МАССИВЫ И СТРОКИ

4.1 Массивы

Массивы представляет собой совокупность переменных одного типа с общим для обращения к ним именем. В C# массивы могут быть как одномерными, так и многомерными. Массивы служат самым разным целям, поскольку они предоставляют удобные средства для объединения связанных вместе переменных.

Массивами в C# можно пользоваться практически так же, как и в других языках программирования. Тем не менее у них имеется одна особенность: они реализованы в виде объектов, поэтому ля того чтобы воспользоваться массивом в программе, требуется двухэтапная процедура. Во-первых, необходимо объявить переменную, которая может обращаться к массиву. И во-вторых, нужно создать экземпляр массива, используя оператор new.

```
int[] myArr = new int[5];
```

4.1.1 Инициализация массива

Заполнение массива возможно несколькими способами. Во-первых – это заполнение по элементно:

- вручную:

```
int[] myArr = new int[5];
myArr[0] = 3;
myArr[1] = 2;
myArr[2] = 6;
myArr[3] = 8;
myArr[4] = 5;
```

- *использую цикл for:*

```
int[] myArr = new int[5];
Random ran = new Random();
for (int i = 0; i < 5; i++)
{
```

```
  myArr[i] = ran.Next(1, 15);
}
```

В данном случае наш массив будет состоять из случйных чисел от 1 до 14 включительно.

Во-вторых – использованием специального синтаксиса инициализации массивов. Для этого необходимо перечислить включаемые в массив элементы в фигурных скобках { }. Такой синтаксис удобен при создании массива известного размера, когда нужно быстро задать его начальные значения:

```
int[] myArr = {10,20,30,40,50};
int[] myArr = new int[] {10,20,30,40,50};
int[] myArr = new int[5] {10,20,30,40,50};
```

В случае применения синтаксиса с фигурными скобками размер массива указывать не требуется, поскольку этот размер автоматически вычисляется на основе количества элементов внутри фигурных скобок. Кроме того, применять ключевое слово *new* не обязательно.

4.1.2 Определение массива объектов

В большинстве случаев при определении массива тип элемента, содержащегося в массиве, указывается явно. Но учитывая, что в основе каждого типа в системе типов .NET (в том числе фундаментальных типов данных) в конечном итоге лежит базовый класс *System.Object*. В результате получается, что в случае определения массива объектов находящиеся внутри него элементы могут представлять собой что угодно:

```
object[] arrObj = { 5, 15.14, false, "ИС"};
```

4.1.3 Массив как объект

Мы уже говорили о том, что массивы в C# реализованы как объекты. Если говорить более точно, то они реализованы на основе базового класса *Array*, определенного в пространстве имен *System*. Данный класс содержит различные свойства и методы. Например, свойство *Length* позволяет определять количество элементов в массиве.

```
int[] myArr = { 1, 2, 3, 4, 5 };
for (int i = 0; i < myArr.Length; i++)
  Console.WriteLine(myArr[i]);
Console.ReadLine();
```

Свойства и методы класса *Array* представлены в таблице 8.

Таблица 8 – Свойства и методы класса Array

Элемент	Вид	Описание
Length	свойство	Количество элементов массива (по всем размерностям)
BinarySearch	статический метод	Двоичный поиск в отсортированном массиве
Clear	статический метод	Присваивание элементам массива значений по умолчанию
Copy	статический метод	Копирование заданного диапазона элементов одного массива в другой
CopyTo	экземплярный метод	Копирование всех элементов текущего одномерного массива в другой массив
GetValue	экземплярный метод	Получение значения элемента массива
IndexOf	статический метод	Поиск первого вхождения элемента в одномерный массив
LastIndexOf	статический метод	Поиск последнего вхождения элемента в одномерный массив
Reverse	статический метод	Изменение порядка следования элементов на обратный
SetValue	экземплярный метод	Установка значения элемента массива
Sort	статический метод	Упорядочивание элементов одномерного массива

Вызов статических методов происходит через обращение к имени класса, например, *Array.Sort(myArray)*. В данном случае мы обращаемся к статическому методу *Sort* класса *Array* и передаем данному методу в качестве параметра объект *myArray* – экземпляр класса *Array*.

Обращение к свойству или вызов экземплярного метода производится через обращение к экземпляру класса, например, *myArr.Length* или *myArr.GetValue(i)*.

```
{
  int[] MyArr;
```

```
Console.Write("Введите размерность массива: ");
int n = Convert.ToInt32(Console.ReadLine());
MyArr = new int[n];
for (int i = 0; i < MyArr.Length; i++)
        {
                Console.Write("a[{0}]=",i);
                MyArr[i] = Convert.ToInt32(Console.ReadLine());
        }
Console.WriteLine("исходный массив:");
for (int i = 0; i < MyArr.Length; i++)
        {
                Console.Write(MyArr[i]+" ");
        }
Array.Sort(MyArr);
Console.WriteLine("\nотсортированный массив массив:");
for (int i = 0; i < MyArr.Length; i++)
        {
                Console.Write(MyArr[i] + " ");
        }
Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 16.

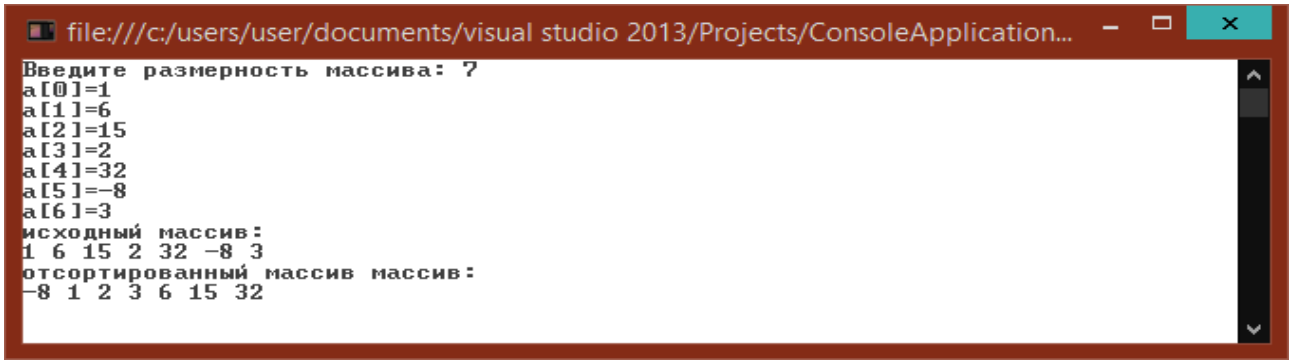

Рисунок 16 – применение оператора continue

4.2 Многомерные массивы

Многомерным называется такой массив, который отличается двумя или более измерениями, причем доступ к каждому элементу такого массива осуществляется с помощью определенной комбинации двух или более индексов. Многомерный массив индексируется двумя и более целыми числами.

4.2.1 Двумерные массивы

Простейшей формой многомерного массива является двумерный массив. Местоположение любого элемента в двумерном массиве обозначается двумя индексами. Такой массив можно представить в виде таблицы, на строки которой указывает один индекс, а на столбцы — другой. Пример объявления и инициализации двумерного массива показан ниже:

```
int[,] myArr = new int[4, 5];
Random ran = new Random();
for (int i = 0; i < 4; i++)
{
 for (int j = 0; j < 5; j++)
        {
                myArr[i, j] = ran.Next(1, 15);
                Console.Write("{0}\t", myArr[i, j]);
        }
  Console.WriteLine();
}
```

В двумерном массиве *myArr* индекс *i* обозначет номер строки, а *j* номер столбца, обратите внимание , что нумерация начинается с 0.

C# допускаются массивы трех и более измерений. Ниже приведена общая форма объявления многомерного массива:

```
mun[,...,] имя_массива = new mun[размер1, размер2, ... размерN];
```

4.2.2 Ступенчатые массивы

Двумерный массив можно представить в виде таблицы, в которой длина

каждой строки остается неизменной по всему массиву. Но в C# можно также создавать специальный тип двумерного массива, называемый ступенчатым массивом. Ступенчатый массив представляет собой массив массивов, в котором длина каждого массива может быть разной. Следовательно, ступенчатый массив может быть использован для составления таблицы из строк разной длины.

Ступенчатые массивы объявляются с помощью ряда квадратных скобок, в которых указывается их размерность. Например, для объявления двумерного ступенчатого массива служит следующая общая форма:

тип [][] имя массива = new тип[размер] [];

Размер обозначает число строк в массиве. Память для самих строк распределяется индивидуально, и поэтому длина строк может быть разной.

Пример использования ступенчатого массива:

```
{
 int i = 0;
 // Объявляем ступенчатый массив
 int[][] myArr = new int[4][];
 myArr[0] = new int[4];
 myArr[1] = new int[6];
 myArr[2] = new int[3];
 myArr[3] = new int[4];
 // Инициализируем ступенчатый массив
 for (; i < 4; i++)
        {
                myArr[0][i] = i;
                Console.Write("{0}\t",myArr[0][i]);
        }
 Console.WriteLine();
 for (i = 0; i < 6; i++)
        {
                myArr[1][i] = i;
```

```
                Console.Write("{0}\t", myArr[1][i]);
            }
        Console.WriteLine();
        for (i = 0; i < 3; i++)
        {
            myArr[2][i] = i;
            Console.Write("{0}\t", myArr[2][i]);
        }
        Console.WriteLine();
        for (i = 0; i < 4; i++)
            {
                myArr[3][i] = i;
                Console.Write("{0}\t", myArr[3][i]);
            }
        Console.ReadLine();
    }
```

Результат выполнения программы представлен в соответствии с рисунком 17.

Рисунок 17 – пример реализации ступенчатого массива

После создания ступенчатого массива доступ к его элементам осуществляется по индексу, указываемому в отдельных квадратных скобках.

4.2.3 Применение свойства *Length* при обращении со ступенчатыми массивами

Особый случай представляет применение Length при обращении со ступенчатыми массивами. В этом случае с помощью данного свойства можно получить длину каждого массива, составляющего ступенчатый массив. Давайте рассмотрим следующий пример:

```
{
  int[][] myArr = new int[3][];
  myArr[0] = new int[4];
  myArr[1] = new int[10];
  myArr[2] = new int[1];
  Console.WriteLine("Общая длина всего массива: " + myArr.Length);
  Console.WriteLine("\nДлина первой цепи: " + myArr[0].Length);
  Console.WriteLine("\nДлина второй цепи: " + myArr[1].Length);
  Console.WriteLine("\nДлина третьей цепи: " + myArr[2].Length);
  Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 18.

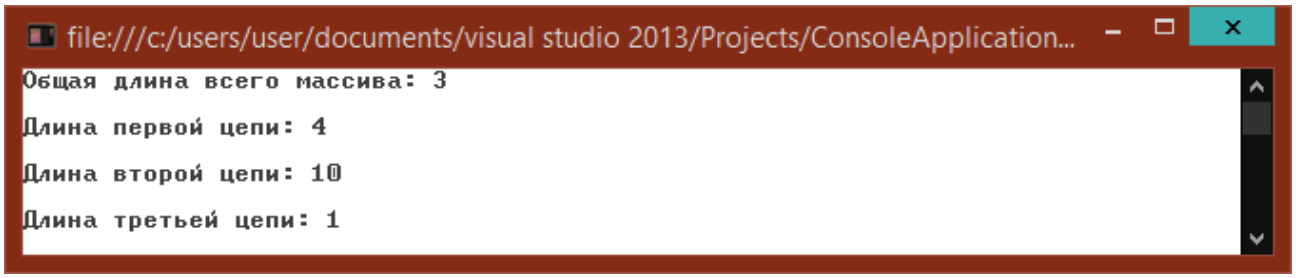

Рисунок 18 – пример применения свойства Length в ступенчатом

Свойство *Length* используется в ступенчатом массиве *myArr*. Напомним, что двумерный ступенчатый массив представляет собой массив массивов. Следовательно, когда используется выражение *myArr.Length* то в нем определяется число массивов, хранящихся в массиве myArr (в данном случае — 3 массива). А для получения длины любого отдельного массива, составляющего ступенчатый массив, служит выражение *myArr[0].Length*.

4.3 Символы и строки

Обработка текстовой информации является одной из самых распространенных задач современного программировании. C# предоставляет для ее решения широкий набор средств: символы *char*, неизменяемые строки

string, изменяемые строки *StringBuider.*

4.3.1 Символы char

Символьный тип *char* предназначен для хранения символа в кодировке Unicode. Символьный тип относится к встроенным типам данных C# и соответствует стандартному классу *Char* библиотеки .Net из пространства имен *System*. В этом классе определены статические методы, позволяющие задавать вид и категорию символа, а также преобразовывать символ в верхний или нижний регистр, в число. Основные методы, представлены в таблице 9.

Таблица 9 – Свойства и методы класса Char

Метод	Описание
GetNumericValue	Возвращает числовое значение символа, если он является цифрой, и -1 в противном случае.
GetUnicodeCategory	Возвращает категорию Unicode-символа. В Unicode символы разделены на категории, например цифры (DecimalDigitNumber), римские цифры (LetterNumber), разделители строк (LineSeparator), буквы в нижнем регистре (LowercaseLetter) и т.д.
IsControl	Возвращает true, если символ является управляющим.
IsDigit	Возвращает true, если символ является десятичной цифрой.
IsLetter	Возвращает true, если символ является буквой.
IsLetterOrDigit	Возвращает true, если символ является буквой или десятичной цифрой.
IsLower	Возвращает true, если символ задан в нижнем регистре.
IsNumber	Возвращает true, если символ является числом (десятичным или шестнадцатеричным).
IsPunctuation	Возвращает true, если символ является знаком препинания.
IsSeparator	Возвращает true, если символ является разделителем.
IsUpper	Возвращает true, если символ задан в верхнем регистре.
IsWhiteSpace	Возвращает true, если символ является пробельным (пробел, перевод строки, возврат каретки).
Parse	Преобразует строку в символ (строка должна состоять из одного символа).
ToLower	Преобразует символ в нижний регистр
ToUpper	Преобразует символ в верхний регистр

В следующем примере рассмотрим применение данных методов:

```
{
  char b = 'B', c = '\x64', d = '\u0127';
```

```
        Console.WriteLine("{0}, {1}, {2}", b, c, d);
        Console.WriteLine("{0}, {1}, {2}", char.ToLower(b), char.ToUpper(c),
char.GetNumericValue(d));
        char a;
        do
        {
            Console.WriteLine("Введите символ: ");
            a = char.Parse(Console.ReadLine());
            Console.WriteLine("Введен символ {0}, его код {1}, его категория
            {2}", a, (int)a, char.GetUnicodeCategory(a));
            if (char.IsLetter(a)) Console.WriteLine("Буква");
            if (char.IsUpper(a)) Console.WriteLine("Верхний регистр");
            if (char.IsLower(a)) Console.WriteLine("Нижний регистр");
            if (char.IsControl(a)) Console.WriteLine("Управляющий символ");
            if (char.IsNumber(a)) Console.WriteLine("Число");
            if (char.IsPunctuation(a)) Console.WriteLine("Разделитель");
        } while (a != 'e');
        Console.ReadLine();
    }
```

Результат выполнения программы представлен в соответствии с рисунком 19.

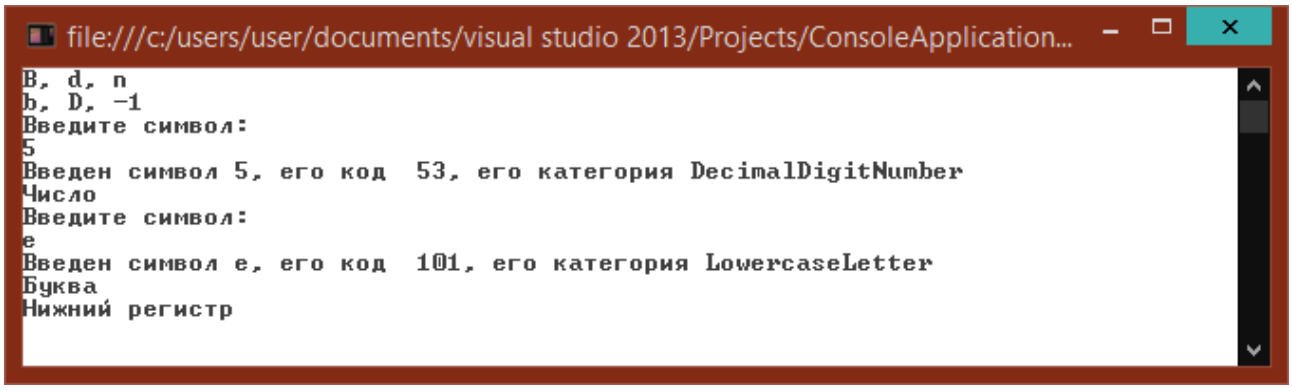

Рисунок 19 – пример методов класса Char

4.3.2 Неизменяемые строки string

Тип *string*, предназначенный для работы со строками символов в кодировке Unicode, является встроенным типом C#. Ему соответствует базовый тип класса *System.String* библиотеки .Net. Каждый объект *string* – это неизменяемая последовательность символов Unicode, т.е. методы, предназначенные для изменения строк, возвращают измененные копии, исходные же строки остаются неизменными.

Создать строку можно несколькими способами:

– *string s; // инициализация отложена*;

– *string s="кол около колокола"; //инициализация строковым литералом*;

– *string s=new string (' ', 20); //конструктор создает строку из 20 пробелов*;

– *int x = 12344556; //инициализировали целочисленную переменную*;

string s = x.ToString(); //преобразовали ее к типу string

– *char [] a={'a', 'b', 'c', 'd', 'e'}; //создали массив символов*

string v=new string (a); // создание строки из массива символов

Класс *string* обладает богатым набором свойств и методов для сравнения строк, поиска в строке и других действий со строками, которые представлены в таблице 10.

Таблица 10 – методы класса String

Название	Вид	Описание
Compare	Статический метод	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки с учетом или без учета регистра.
CompareTo	Метод	Сравнение текущего экземпляра строки с другой строкой.
Concat	Статический метод	Слияние произвольного числа строк.
Copy	Статический метод	Создание копии строки
Empty	Статическое поле	Открытое статическое поле, представляющее пустую строку
Format	Статический метод	Форматирование строки в соответствии с заданным форматом

Продолжение таблица 10

IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny	Экземплярные методы	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора в данную строку.
Insert	Экземплярный метод	Вставка подстроки в заданную позицию
Join	Статический метод	Слияние массива строк в единую строку. Между элементами массива вставляются разделители.
Length	Свойство	Возвращает длину строки
PadLeft, PadRigth	Экземплярные методы	Выравнивают строки по левому или правому краю путем вставки нужного числа пробелов в начале или в конце строки.
Remove	Экземплярный метод	Удаление подстроки из заданной позиции
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки или символа новыми подстрокой или символом.
Split	Экземплярный метод	Разделяет строку на элементы, используя разные разделители. Результаты помещаются в массив строк.
StartWith, EndWith	Экземплярные методы	Возвращают true или false в зависимости от того, начинается или заканчивается строка заданной подстрокой.
Substring	Экземплярный метод	Выделение подстроки, начиная с заданной позиции
ToCharArray	Экземплярный метод	Преобразует строку в массив символов
ToLower, ToUpper	Экземплярные методы	Преобразование строки к нижнему или верхнему регистру
Trim, TrimStart, TrimEnd	Экземплярные методы	Удаление пробелов в начале и конце строки или только с одного ее конца.

Напомим, что вызов статических методов происходит через обращение к имени класса, например, *String.Concat(str1, str2)*, в остальных случаях через обращение к экземплярам класса, например, *str.ToLower()*.Рассмотрим пример программы, которая использует несколько из вышеуказанных методов:

```
{
  // Сравним первые две строки
  string s1 = "это строка";
  string s2 = "это текст, а это строка";
  if (String.CompareOrdinal(s1, s2) != 0)
        Console.WriteLine("Строки s1 и s2 не равны");
```

```
if (String.Compare(s1, 0, s2, 13, 10, true) == 0)
        Console.WriteLine("При этом в них есть одинаковый текст");
// Конкатенация строк
Console.WriteLine(String.Concat("\n" + "Один, два ", "три, четыре"));
// Поиск в строке
// Первое вхождение подстроки
if (s2.IndexOf("это") != -1)
        Console.WriteLine("Слово \"это\" найдено в строке, оно " +
"находится на: {0} позиции", s2.IndexOf("это"));
// Последнее вхождение подстроки
if (s2.LastIndexOf("это") != -1)
        Console.WriteLine("Последнее вхождение слова \"это\" находится "
        + "на {0} позиции", s2.LastIndexOf("это"));
// Поиск из массива символов
char[] myCh = { 'Ы', 'х', 'т' };
if (s2.IndexOfAny(myCh) != -1)
        Console.WriteLine("Один из символов из массива ch " +
        "найден в текущей строке на позиции {0}", s2.IndexOfAny(myCh));
// Определяем начинается ли строка с заданной подстроки
if (s2.StartsWith("это текст") == true)
Console.WriteLine("Подстрока найдена!");
Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 20.

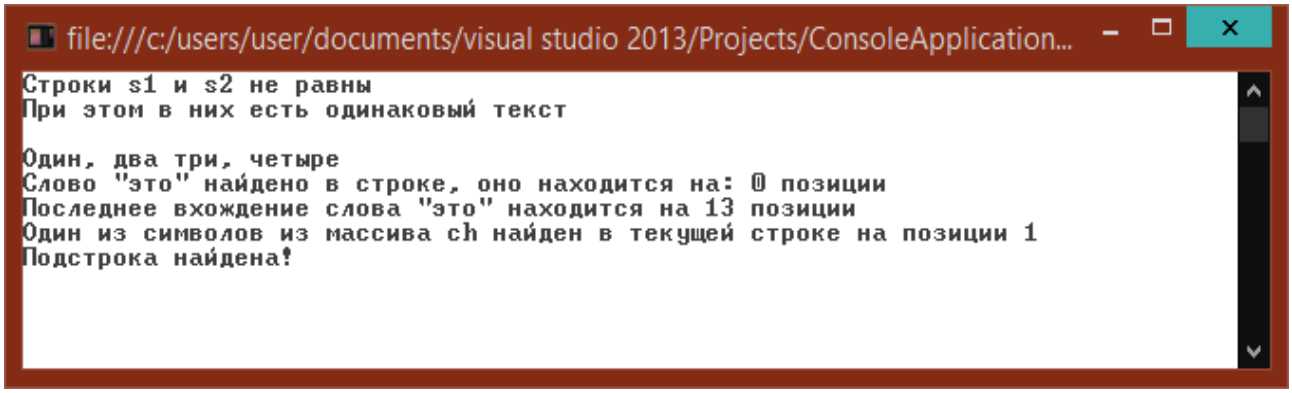

Рисунок 20 – пример использования методов класса String

4.3.3 Изменяемые строк

Чтобы создать строку, которую можно изменять, в C# предусмотрен класс *StringBuilder*, определенный в пространстве имен *System.Text*. Объекты этого класса всегда объявляются с явным вызовом конструктора класса (через операцию *new*) . Примеры создания изменяемых строк:

– *StringBuilder a =new StringBuilder();*

//создание пустой строки, размер по умолчанию 16 символов;

– *StringBuilder b = new StringBuilder("abcd");*

//инициализация строки и выделение необходимой памяти;

StringBuilder c = new StringBuilder(100);

//создание пустой строки и выделение памяти под 100 символов;

– *StringBuilder d = new StringBuilder("abcd", 100);*

//инициализация строки и выделение памяти под 100 символов;

– *StringBuilder d = new StringBuilder("abcd", 1, 3,100);*

//инициализация подстрокой "bcd", и выделение памяти под 100 символов;

Основные элементы класса представлены в таблице 11.

Таблица 11 – методы класса StringBuilder

Название	Вид	Описание
Append	Экземплярный метод	Добавление данных в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки string.
AppendFormat	Экземплярный метод	Добавление форматированной строки в конец строки

Продолжение таблицы 11

Capacity	свойство	Получение и установка емкости буфера. Если устанавливаемое значение меньше текущей длины строки или больше максимального, то генерируется исключение ArgumentOutOfRangeException
Insert	Экземплярный метод	Вставка подстроки в заданную позицию
Length	изменяемое свойство	Возвращает длину строки. Присвоение ему значения 0 сбрасывает содержимое и очищает строку
MaxCapacity	неизменное свойство	Возвращает наибольшее количество символов, которое может быть размещено в строке
Remove	Экземплярный метод	Удаление подстроки из заданной позиции
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
ToString	Экземплярный метод	Преобразование в строку типа string
Chars	изменяемое свойство	Возвращает из массива или устанавливает в массиве символ с заданным индексом. Вместо него можно пользоваться квадратными скобками []
Equals	Экземплярный метод	Возвращает true, только если объекты имеют одну и ту же длину и состоят из одних и тех же символов
CopyTo	Экземплярный метод	Копирует подмножество символов строки в массив char

Как видим, методы класса *StringBuilder* менее развиты, чем методы класса *String*, но они позволяют более эффективно использовать память за счет работы с изменяемыми строками. Рассмотрим пример использования некоторых методов.

```
{
  StringBuilder hello = new StringBuilder("Добрый день. ", 100);
  hello.AppendFormat("Сегодня хорошая погода!");
  // Зашифруем строку, хранящуюся в переменной hello
  Console.WriteLine("Исходная строка: \n {0}\n", hello);
  for (int i = 'я'; i >= 'а'; i--)
  hello = hello.Replace((char)i, (char)(i + 1));
  Console.WriteLine("Зашифрованная строка:\n {0}", hello);
  Console.ReadLine();
```

```
}
```

Результат выполнения программы представлен в соответствии с рисунком 20.

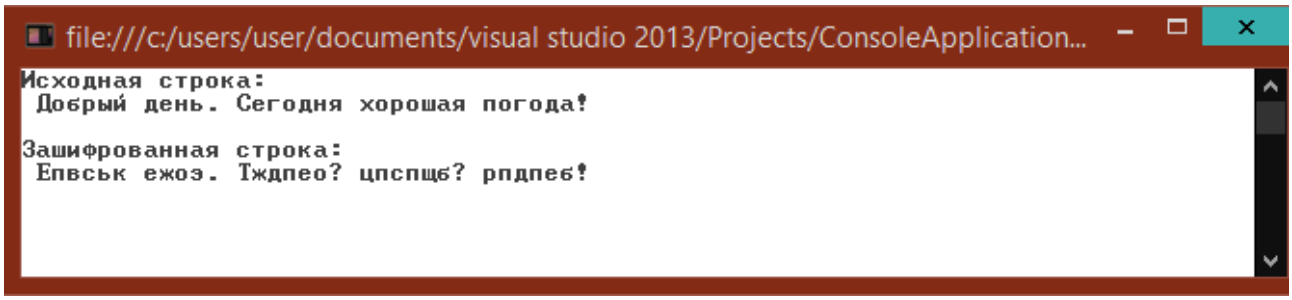

Рисунок 20 – пример использования методов класса StringBuilder

4.4 Регулярные выражения

Стандартный класс *string* позволяет выполнять над строками различные операции, в том числе поиск, замену, вставку и удаление подстрок. Тем не менее, есть классы задач по обработке символьной информации, где стандартных возможностей явно не хватает. Чтобы облегчить решение подобных задач, в .Net Framework встроен более мощный аппарат работы со строками, основанный на регулярных выражениях.

Регулярные выражения предназначены для обработки текстовой информации и обеспечивают:

1. Эффективный поиск в тексте по заданному шаблону.
2. Редактирование текста.
3. Формирование итоговых отчетов по результатам работы с текстом.

Подробно рассмотрим первые два аспекта применения регулярных выражений.

4.4.1 Метасимволы в регулярных выражениях

Регулярное выражение - это шаблон, по которому выполняется поиск соответствующего фрагмента текста. Язык описания регулярных выражений состоит из символов двух видов: обычных символов и метасимволов. Обычный символ представляет в выражении сам себя, а метасимвол – некоторый класс

символов.

Наиболее употребительные метасимволы представлены в таблице 12.

Таблица 12 – метасимволы

Класс символов	Описание	Пример
.	Любой символ, кроме \n.	Выражение c. t соответствует фрагментам: cat, cut,c#t, c{t и т.д.
[]	Любой одиночный символ из последовательности, записанной внутри скобок. Допускается использование *диапазонов символов*.	Выражение c[*aui*]t соответствует фрагментам: cat, cut, *cit*. Выражение c[a-c]t соответствует фрагментам: cat,*cbt*, cct.
[^]	Любой одиночный символ, не входящий в последовательность, записанную внутри скобок. Допускается использование *диапазонов символов*.	Выражение c[^*aui*]t соответствует фрагментам: *cbt*,cct, c2t и т.д. Выражение c[^a-c]t соответствует фрагментам: cdt, *cet*, c%t и т.д.
\w	Любой алфавитно - цифровой символ.	Выражение c\wt соответствует фрагментам: *cbt*, cct,c2t и т.д., но не соответствует фрагментам c%t, c{t и т.д.
\W	Любой не алфавитно - цифровой символ.	Выражение c\Wt соответствует фрагментам: c%t, c{t,c. t и т.д., но не соответствует фрагментам *cbt*, cct,c2t и т.д.
\s	Любой пробельный символ.	Выражение\s\w\w\w\s соответствует любому слову из трех букв, окруженному пробельными символами.
\S	Любой не пробельный символ.	Выражение \s\S\S\S\s соответствует любым трем непробельным символам, окруженным пробельными.
\d	Любая десятичная цифра	Выражение c\dt соответствует фрагментам: c1t, c2t,c3t и т.д.
\D	Любой символ, не являющийся десятичной цифрой	Выражение c\Dt не соответствует фрагментам: c1t, c2t,c3t и т.д.

Кроме метасимволов, обозначающие классы символов, могут применяться уточняющие метасимволы, представленые в таблице 13.

Таблица 13 – уточняющие метасимволы

Уточняющие символы	Описание
^	Фрагмент, совпадающий с регулярными выражениями, следует искать только в начале строки
$	Фрагмент, совпадающий с регулярными выражениями, следует искать только в конце строки
\A	Фрагмент, совпадающий с регулярными выражениями, следует искать только в начале многострочной строки
\Z	Фрагмент, совпадающий с регулярными выражениями, следует искать только в конце многострочной строки
\b	Фрагмент, совпадающий с регулярными выражениями, начинается или заканчивается на границе слова, т.е. между символами, соответствующими метасимволам \w и \W
\B	Фрагмент, совпадающий с регулярными выражениями, не должен встречаться на границе слов

В регулярных выражениях часто используются повторители – метасимволы, которые располагаются непосредственно после обычного символа или группы символов и задают количество его повторений в выражении. Повторители представлены в таблице 14.

Таблица 14 – повторители

Повторители	Описание	Пример
*	Ноль или более повторений предыдущего элемента	Выражение ca*t соответствует фрагментам: ct, cat, caat, caaat и т.д.
+	Одно или более повторений предыдущего элемента	Выражение ca+t соответствует фрагментам: cat, caat, caaat и т.д.
?	Не более одного повторения предыдущего элемента	Выражение ca?t соответствует фрагментам: ct, cat.
{n}	Ровно n повторений предыдущего элемента	Выражение ca{3}t соответствует фрагменту: caaat. Выражение (cat){2}соответствует фрагменту: c a tcat.
{n,}	По крайней мере nповторений предыдущего элемента	Выражение ca{3, }t соответствует фрагментам: c aaa t, caaaat, caaaaaaat и т.д. Выражение (cat){2, } соответствует фрагментам: catcat, catcatcat и т.д.
{n, m}	От n до m повторений предыдущего элемента	Выражение ca{2, 4}t соответствует фрагментам: c aa t, caaat, caaaat.

Регулярное выражение записывается в виде строкового литерала, причем перед строкой необходимо ставить символ @, который говорит о том, что

строку нужно будет рассматривать и в том случае, если она будет занимать несколько строчек на экране. Однако символ @ можно не ставить, если в качестве шаблона используется шаблон без метасимволов.

Примеры регулярных выражений:

– слово rus – *@"rus"* или *"rus";*

– номер телефона в формате xxx-xx-xx – *@"\d\d\d-\d\d-\d\d"* или *@"\d{3}(-\d\d){2}"*;

– номер автомобиля – *@"[A-Z]\d{3}[A-Z]{2}\d{2,3}RUS"*;

4.4.2 класс Regex

Безусловно, задачу поиска и замены подстроки в строке можно решить на C# с использованием различных методов *System.String* и *System.Text.StringBuilder*. Однако в некоторых случаях это потребует написания большого объема кода C#. Если вы используете регулярные выражения, то весь этот код сокращается буквально до нескольких строк. Пространство имен библиотеки базовых классов *System.Text.RegularExpressions* содержит все объекты платформы .NET Framework, имеющие отношение к регулярным выражениям. Важнейшим классом, поддерживающим регулярные выражения, является класс *Regex*, который представляет неизменяемые откомпилированные регулярные выражения. Для описания регулярного выражения в классе определено несколько перегруженных конструкторов:

– *Regex()* – создает пустое выражение;

– *Regex(String)* – создает заданное выражение;

– *Regex(String, RegexOptions)* – создает заданное выражение и задает параметры для его обработки с помощью элементов перечисления *RegexOptions* (например, различать или нет прописные и строчные буквы).

В таблице 15 представлена часть информации о перечислении *RegexOptions*, экземпляр которого можно передать конструктору класса *RegEx*:

Таблица 15 – структура перечисления RegexOptions

Член	Описание
CultureInvariant	Предписывает игнорировать национальные установки строки.
ExplicitCapture	Модифицирует способ поиска соответствия, обеспечивая только буквальное соответствие.
IgnoreCase	Игнорирует регистр символов во входной строке.
IgnorePatternWhitespace	Удаляет из строки не защищенные управляющими символами пробелы и разрешает комментарии, начинающиеся со знака фунта или хеша.
Multiline	Изменяет значение символов ^ и $ так, что они применяются к началу и концу каждой строки, а не только к началу и концу всего входного текста.
RightToLeft	Предписывает читать входную строку справа налево вместо направления по умолчанию — слева направо (что удобно для некоторых азиатских и других языков, которые читаются в таком направлении).
Singleline	Специфицирует однострочный режим, в котором точка (.) символизирует соответствие любому символу.

Поиск фрагментов строки, соответствующих заданному выражению, выполняется с помощью методов *IsMatch*, *Match*, *Matches* класса *Regex*.

Метод *IsMatch* возвращает *true*, если фрагмент, соответствующий выражению, в заданной строке найден, и *false* в противном случае.

Например,

Попытаемся определить, есть ли в заданных строках номера телефона в формате xx-xx-xx или xxx-xx-xx:

```
static void Main()
{
  Regex r = new Regex(@"\d{2,3}(-\d\d){2}");
  string text1 = "tel:123-45-67";
  string text2 = "tel:25dre6";
  if (r.IsMatch(text1))
  Console.WriteLine("{0} - это номер телефона", text1);
  else Console.WriteLine("{0} - вы сюда не дозвонитесь", text1);
  if (r.IsMatch(text2))
  Console.WriteLine("{0} - это номер телефона", text2);
```

```
  else Console.WriteLine("{0} - вы сюда не дозвонитесь", text2);
  Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 21.

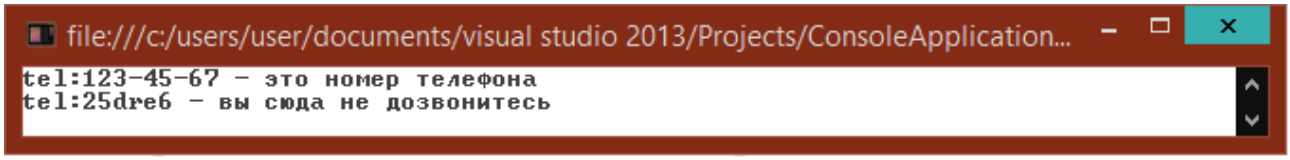

Рисунок 21 – пример использования регулярных выражений

4.5 Форматирующие строки

Если вы хотите, чтобы разрабатываемые классы были дружественными к пользователю, они должны предлагать средства для отображения своих строковых представлений в любом из форматов, которые могут понадобиться пользователю. В исполняющей среде .NET определен стандартный способ достижения этого — интерфейс *IFormattable*.

4.5.1 Форматирование числовых символов

Рассмотрим спецификаторы для числовых символов,которые представлены в таблице 16.

Таблица 16 – спецификаторы формата для чисел

Спецификатор	Применяется к	Значение	Пример
C	Числовым типам	Символ местной валюты	$835.50 (США) 835.50р.(Россия)
D	Целочисленным типам	Обычное целое	835
E	Числовым типам	Экспоненциальная нотация	8.35E+002
F	Числовым типам	С фиксированной десятичной точкой	835.50
G	Числовым типам	Обычные числа	835.5
N	Числовым типам	Формат чисел, принятый в данной местности	4,384.50 (/США 4 384,50(Европа)
P	Числовым типам	Процентная нотация	835,000.00%
X	Целочисленным типам	Шестнадцатеричный формат	1a1f

Стоит отметить, что полный список спецификаторов формата значительно

длинее, поскольку другие типы данных добавляют собственные спецификаторы.

Пример:

```
{
  double d = 12.05667, i = 0.25;
  Console.WriteLine("Значение переменной d = {0:F}, а i = {1:P}", d, i);
  Console.ReadLine();
}
```

Результат выполнения программы представлен в соответствии с рисунком 22.

Рисунок 22 – пример использования форматирующих строк

4.5.2 Форматирование даты и времени

Помимо числовых значений, форматированию нередко подлежит и другой тип данных: *DateTime*. Это структура, представляющая дату и время. Значения даты и времени могут отображаться самыми разными способами.

Форматирование даты и времени осуществляется с помощью спецификаторов формата. Спецификаторы формата даты и времени приведены таблице 17.

Таблица 17 – спецификаторы даты и времени

Спецификатор	Формат
D	Дата в длинной форме
d	Дата в краткой форме
F	Дата и время в длинной форме
f	Дата и время в краткой форме
G	Дата — в краткой форме, время — в длинной
g	Дата и время — в краткой форме
M	Месяц и день
m	То же, что и M
O	Формат даты и времени, включая часовой пояс. Строка, составленная в формате O, может быть преобразована обратно в эквивалентную форму вывода даты и времени. Это так называемый "круговой" формат

Продолжение таблицы 17

R	Дата и время в стандартной форме по Гринвичу
s	Сортируемый формат представления даты и времени
T	Время в длинной форме
t	Время в краткой форме
U	Длинная форма универсального представления даты и времени; время отображается как универсальное синхронизированное время (UTC)
u	Краткая форма универсального представления даты и времени
Y	Месяц и год

Рассмотрим пример демонстрирующий применение спецификаторов даты и времени.

```
{
DateTime myDate = DateTime.Now;
Console.WriteLine("Дата в формате d: {0:d}\nB формате D: {0:D}", myDate);
Console.WriteLine("Дата в формате f: {0:f}\nB формате F: {0:F}", myDate);
Console.WriteLine("Дата в формате g: {0:g}\nB формате G: {0:G}", myDate);
Console.WriteLine("Дата в формате m: {0:m}\nB формате M: {0:M}", myDate);
Console.WriteLine("Дата в формате r: {0:r}\nB формате R: {0:R}", myDate);
Console.WriteLine("Дата в формате o: {0:o}\nB формате O: {0:O}", myDate);
Console.WriteLine("Дата в формате s: {0:s}", myDate);
Console.WriteLine("Дата в формате t: {0:t}\nB формате T: {0:T}", myDate);
Console.WriteLine("Дата в формате u: {0:u}\nB формате U: {0:U}", myDate);
Console.WriteLine("Дата в формате y: {0:y}\nB формате Y: {0:Y}", myDate);
Console.ReadLine();
```

}

Результат выполнения программы представлен в соответствии с рисунком 23.

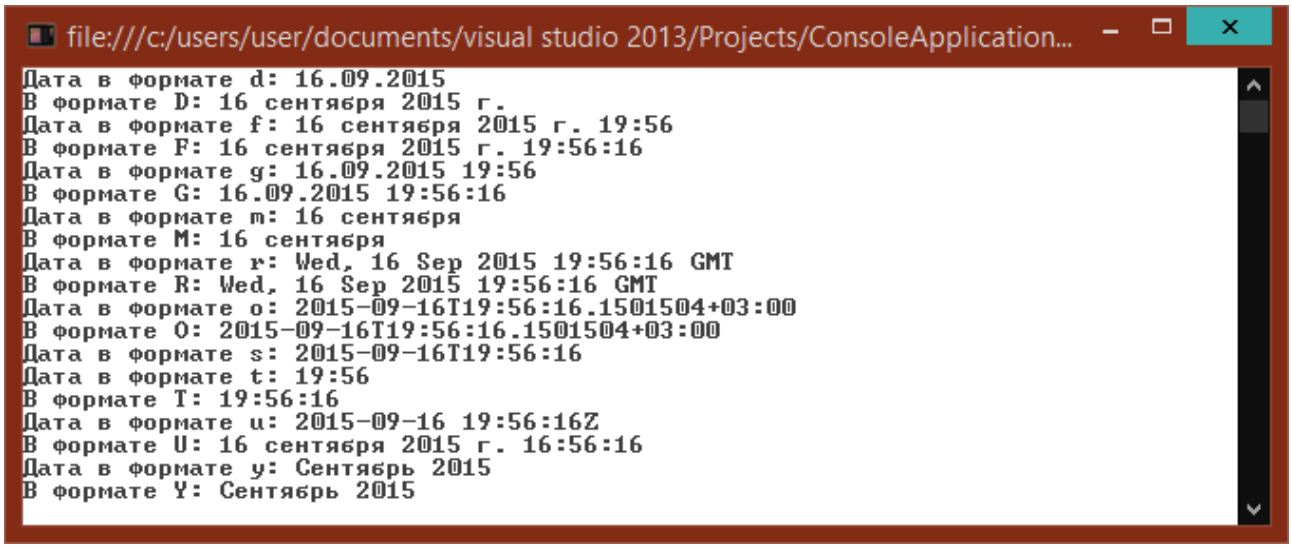

Рисунок 23 – пример использования форматирующих строк

5 ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Задание №1

Вычислить значение функции при A, B, C – константы (задать значение самостоятельно), x вводится с клавиатуры.

№	Задание	№	Задание
1	$y = \frac{3A - B}{A + 2B}(A - C)cos^3 x + lnx$	16	$y = \frac{\sqrt{x} + Ax}{x + e^{B+C}}$
2	$y = lnBcosx^3 \sqrt{\frac{x^A}{AC}}$	17	$y = \frac{x^2 + \lvert Bx - 3C\rvert}{\ln(x^3 + BC + A)}$
3	$y = tg\frac{C}{A - B} + e^{x+1} - \frac{1 + x}{\sin^2 x}$	18	$y = A^x + \frac{C}{x^B} + \sqrt{\lvert A - x\rvert}$
4	$y = \frac{x^{A-1}}{1 + \frac{x^{B-2}}{1 + x^C}}$	19	$y = \frac{A + Bx + x^{A-B}}{C + \sqrt{\lvert A - B\rvert}}$
5	$y = -\frac{Ae^{\cos x} + \ln(C - B)}{\sin(x + B)}$	20	$y = -e^A - tg\left(\frac{B}{C}\right) + \sqrt{\cos Bx}$
6	$y = 2{,}56x^3 \ln(A + B^{x-C})$	21	$y = -e^{\lvert B-x\rvert}(tg^4 x + A)^C$
7	$y = \frac{\sqrt{sin^2(x^3 + B) + C^x}}{A}$	22	$y = \frac{1 + \sin(x + A)}{\left\lvert B - \frac{2A}{1 + x^C}\right\rvert}$
8	$y = -2\sqrt{B^x + \frac{5x^4}{A^2 + B - C}}$	23	$y = \sqrt{1 + \frac{x^2}{A} + \frac{x^3}{B} + \frac{x^4}{C}}$
9	$y = e^{\frac{x}{b}}(sin^2(2Ax) + \cos x)$	24	$y = \frac{\sqrt{A(\sqrt{x} + x^B)}}{cos^2 C}$
10	$y = \sin A\,\lvert\cos x\rvert + \frac{A + B}{lnC}$	25	$y = \frac{\cos x - \frac{A}{(B - x)}}{1 + (C - x)^2}$

11	$y = \frac{1}{\sin x} + \ln\left\lvert tg\ \frac{B}{A}\right\rvert + Cx$	26	$y = \frac{x + 3\lvert x - A\rvert + x^2}{\lvert x - B\rvert C + x^2}$
12	$y = \cos(1 - \frac{\sqrt{Ax + C}}{B^2})$	27	$y = \lvert x - A\rvert \left(1 + \frac{sin^2 B}{\sqrt{x + C}}\right)$
13	$y = \frac{\sqrt{A + \sqrt{B + \sqrt{C + x}}}}{lnx}$	28	$y = \frac{x + \frac{A}{2}}{2\lvert x + B\rvert}(x + 1)^{-1/\sin C}$
14	$y = \frac{lnA - x^{A-B}}{C + \sqrt{\lvert A - B\rvert}}$	29	$y = \lvert x\rvert \left(1 - \frac{cos^3 A}{\sqrt{x + C}}\right)$
15	$y = \operatorname{tg} A\ \lvert\sin x\rvert - \frac{lnA}{A + B}$	30	$y = \sqrt{1 + \frac{x^2}{A} + \lvert\frac{-B}{x}\rvert + \frac{x^4}{C}}$

Задание №2

Решите задачу с использованием условного оператора.

№	Задача
1	С клавиатуры вводятся числа A, B, X, Y. Выяснить, можно ли уместить прямоугольник со сторонами X, Y в прямоугольнике со сторонами A, B (и наоборот).
2	Выяснить существует ли треугольник со сторонами вершин A, B, C, если да, то найти его периметр.
3	С клавиатуры вводятся числа X, Y. Вычислить значение функции z=log(x-y) – x/y. Проверить принадлежность x и y области допустимых значений функции.
4	Определить, сколько среди заданных чисел A, B, C, D отрицательных. И заменить все отрицательные на положительные.
5	Определить, сколько среди заданных чисел A, B, C, D лежат в диапазоне (-10,25). Заменить их на противоположные, и узнать количество чисел, лежащих в диапазоне.
6	С клавиатуры вводятся числа A, B, C, D. Найти минимальное из них и возвести в квадрат.
7	Выяснить существует ли треугольник со сторонами A, B, C, если да, то определить является ли он равнобедренным.
8	С клавиатуры вводятся числа X, Y. Вычислить значение функции z=log(x/y) – 1/x. Проверить принадлежность x и y области допустимых значений функции.
9	Определить, сколько среди заданных чисел A, B, C, D положительных. И возвести все положительные в квадрат.
10	Определить, сколько среди заданных чисел A, B, C, D кратно 5. И увеличить их в 5 раз.
11	Определить, сколько среди заданных чисел A, B, C, D лежат в диапазоне [-30,17]. Заменить их на противоположные, и узнать количество чисел, лежащих в диапазоне.
12	С клавиатуры вводятся числа W, X, Y, Z. Найти максимальное из них и увеличить его вдвое.
13	С клавиатуры вводятся числа X, Y. Вычислить значение функции z=sqrt(x-y) + x/y. Проверить принадлежность x и y области допустимых значений функции.
14	Определить, сколько среди заданных чисел A, B, C, D нечетных. И увеличить их вдвое.
15	Определить, сколько среди заданных чисел A, B, C, D кратно 3. И возвести их в куб.
16	Определить, сколько среди заданных чисел A, B, C, D лежат в диапазоне (-46,27]. Заменить их на противоположные, и узнать количество чисел, лежащих в диапазоне.
17	С клавиатуры вводятся числа A, B, C проверить выполняется ли равенство A<B<C, если да, то переменной B присвоить сумму A и C, иначе C присвоить разность B и A.
18	С клавиатуры вводятся числа A, B, C проверить выполняется ли равенство A>B>C, если да, то переменной A присвоить произведение B и C, иначе B сумму A и C.

19	Выяснить существует ли треугольник со сторонами A, B, C, если да, то определить является ли он равносторонним.
20	Определить, сколько среди заданных чисел A, B, C, D кратно 9. И уменьшить их в 3 раза.
21	С клавиатуры вводятся числа X, Y. Вычислить значение функции z=sqrt(y/x) +1/x. Проверить принадлежность x и y области допустимых значений функции.
22	Определить, сколько среди заданных чисел A, B, C, D не равных нулю. И вычислить их квадратный корень.
23	Определить, сколько среди заданных чисел A, B, C, D четных. И уменьшить их вдвое.
24	Определить, сколько среди заданных чисел A, B, C, D кратно 10. И увеличить их в 10 раз.
25	Определить, сколько среди заданных чисел A, B, C, D лежат в диапазоне [-13,33]. Заменить их на противоположные по знаку, и узнать количество новых чисел, лежащих в диапазоне.
26	С клавиатуры вводятся числа A, B, C, D. Найти четные и уменьшить их вдвое.
27	Выяснить существует ли треугольник со сторонами A, B, C, если да, то определить является ли он прямоугольным.
28	С клавиатуры вводятся числа X, Y. Вычислить значение функции z= 1/(y-x). Проверить принадлежность x и y области допустимых значений функции.
29	Определить, сколько среди заданных чисел A, B, C, D положительных. И найти натуральный логарифм от каждого положительного.
30	Определить, сколько среди заданных чисел A, B, C, D лежат в диапазоне (-13,23]. Заменить отрицательные на противоположные по знаку.

Задание №3

Решите задачу с использованием оператора *switch*. Написать обработчик для не входящих в диапазон значений.

№	Задача
1	Приведите введенную с клавиатуры цифру от 0 до 9 к его словесному представлению.
2	Составить программу, которая по введенной заглавной букве выводит название стран мира (минимум 15 стран).
3	Составить программу, которая по введенной заглавной букве выводит название городов России (минимум 15 городов).
4	Составить программу, которая по введенной начальной букве выводит название цветов радуги.
5	Составить программу, которая по введенному порядковому номеру выводит название дня недели.
6	Составить программу, которая позволяет ввести номер месяца и вывести его название.
7	Составить программу, которая позволяет ввести номер месяца и вывести сезон в котором он находится
8	Составить программу, которая по введенной заглавной букве выводит мужские имена (минимум 15 имен).
9	Составить программу, которая по введенной заглавной букве выводит женские имена (минимум 15 имен).
10	По автомобильному номеру региона вывести его название (1-15)
11	По автомобильному номеру региона вывести его название (16-30)
12	Составить программу, которая по числовой версии ОС Windows NT выведет ее название (минимум 15 версий).
13	Составить программу, которая по шестнадцатеричному коду цвета вывести его название (минимум 15 цветов).
14	По автомобильному номеру региона вывести его название (31-45)
15	По автомобильному номеру региона вывести его название (46-60)
16	Составить программу, которая по введенной заглавной букве выводит название столиц мира (минимум 15 столиц).
17	Составить программу, которая по введенной заглавной букве выводит название рек России (минимум 15 рек).
18	Составить программу, которая по введенному номеру планеты выведет ее название.
19	Составить программу, которая по введенной заглавной букве выводит фамилии (минимум 15 фамилий).
20	Составить программу, которая по первым двум буквам выводит марки планшетов (минимум 15 марок).

21	Составить программу, которая по введенному номеру знака зодиака выведет его название.
22	Составить программу, которая по введенной заглавной букве выводит название браузера (минимум 15 браузеров)
23	Составить программу, которая по введенной заглавной букве выводит марки автомобилей (минимум 15 марок)
24	С клавиатуры вводятся 2числа. Составить программу, которая выводит меню и выполняет указанные действия с заданными целыми числами: 1 - сумма чисел 2 – разность чисел 3 - выход
25	Составить программу, которая по введенной заглавной букве выводит название стран Африки (минимум 15 стран).
26	По автомобильному номеру региона вывести его название (61-75)
27	По автомобильному номеру региона вывести его название (76-90)
28	Составить программу, которая по введенной заглавной букве выводит название столиц Мира (минимум 15 городов).
29	С клавиатуры вводятся 2числа. Составить программу, которая выводит меню и выполняет указанные действия с заданными целыми числами: 1 - произведение чисел 2 - частное чисел 3 - выход
30	Составить программу, которая по введенной заглавной букве выводит производителей бытовой техники (минимум 15 производителей).

Задание №4

Вычислить значение для n вводимого с клавиатуры.

№	Задание	№	Задание
1	$S=\sum_{i=3}^{n}\frac{(x+1)^3}{(4i)}$	16	$S=\sum_{i=-1}^{n}\frac{(x-1)^3}{(2+i)}$
2	$S=\sum_{i=6}^{n}\frac{x^{i+1}}{(2i-1)^2}$	17	$S=\sum_{i=-1}^{n}\frac{x^{i+2}}{(3i+2)^2}$
3	$S=\sum_{i=-4}^{n}\frac{(x+1)^{2i-1}}{2i+9}$	18	$S=\sum_{i=2}^{n}\frac{(x+1)^{2i}}{2i-3}$
4	$S=\sum_{i=0}^{n}\frac{x^3}{(1+3i)}$	19	$S=\sum_{i=-15}^{n}\frac{x^4}{(7-3i)}$
5	$S=\sum_{i=2}^{n}(-1)^{i-1}\frac{1+x}{(2i-1)}$	20	$S=\sum_{i=-1}^{n}(-1)^{i+1}\frac{1-2x}{(2i+1)}$
6	$S=\sum_{i=-8}^{n}\frac{2x}{(2i-1)(2i+15)}$	21	$S=\sum_{i=4}^{n}\frac{2+x}{(2i+1)(2i-5)}$
7	$S=\sum_{i=-5}^{n}\frac{(x-1)^2}{(i+8)}$	22	$S=\sum_{i=0}^{n}\frac{(1-x)^2}{\sqrt{(i+1)}}$
8	$S=\sum_{i=-3}^{n}\frac{(8-x)^2}{(4+i)}$	23	$S=\sum_{i=1}^{n}\frac{(3-2x)^3}{\sqrt{(2i+2^i)}}$
9	$S=\sum_{i=2}^{n}\frac{(x-1)^{2i}}{2i-3}$	24	$S=\sum_{i=3}^{n}\frac{(1-x)^{2i+2}}{2+i/3}$
10	$S=\sum_{i=2}^{n}\frac{(1-x)^{1-i}}{3i-2}$	25	$S=\sum_{i=0}^{n}\frac{(1+x)^{1-i}}{\sqrt{3i+2}}$
11	$S=\sum_{i=9}^{n}\frac{(x+1)^4}{(i-6)}$	26	$S=\sum_{i=2}^{n}\frac{\lvert x-15\rvert}{\sqrt{(i-1)^3}}$

12	$S = \sum_{i=1}^{n} \frac{(3x-2)^3}{i^2}$	27	$S = \sum_{i=-15}^{n} \frac{\sqrt{(3x-2)^{\lvert i \rvert}}}{i^2}$
13	$S = \sum_{i=-20}^{n} (-1)^{i+1} \frac{1+x}{(15-3i)}$	28	$S = \sum_{i=2}^{n} (-1)^{i+1} \frac{1-x}{(15+3i)}$
14	$\sum_{i=2}^{n} \frac{\lvert x+5 \rvert}{\sqrt{(2i-3)^2}}$	29	$S = \sum_{i=1}^{n} 2^i \frac{1+x}{(5+3i)}$
15	$S = \sum_{i=1}^{n} \frac{(3x-15)^2}{i^3}$	30	$S = \sum_{i=1}^{n} \frac{(3-2x)^4}{\sqrt{(2^i-i)}}$

Задание №5

Выполнить задание используя условный оператор и оператор цикла.

№	Задание	Функция
1	Переменная x изменяется в интервале от – 5 до 15 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x-6)! & \text{при } 7 \le x \le 10 \\ (x-9)! & \text{при } 12 \le x \le 15 \\ x+3 & \text{в остальных случаях} \end{cases}$
2	Переменная x изменяется в интервале от – 7 до 13 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x+2)! & \text{при } -1 \le x \le 4 \\ (x-4)! & \text{при } 6 < x \le 10 \\ x^4 & \text{в остальных случаях} \end{cases}$
3	Переменная x изменяется в интервале от – 10 до 12 с шагом 1. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x+6)! & \text{при } -4 \le x \le 2 \\ (x-5)! & \text{при } 7 < x \le 10 \\ x^2 & \text{в остальных случаях} \end{cases}$
4	Переменная x изменяется в интервале от – 12 до 17 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение	$y = \begin{cases} (x+8)! & \text{при } -7 \le x \le 3 \\ (x-6)! & \text{при } 8 \le x < 13 \\ tg(x) & \text{в остальных случаях} \end{cases}$
5	Переменная x изменяется в интервале от – 15 до 12 с шагом 1. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x+9)! & \text{при } -8 \le x \le 3 \\ (x-6)! & \text{при } 8 \le x \le 12 \\ tg(x) & \text{в остальных случаях} \end{cases}$
6	Переменная x изменяется в интервале от – 5 до 20 с шагом 1. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x-7)! & \text{при } 8 \le x \le 10 \\ (x-1)! & \text{при } 12 \le x \le 20 \\ x+3 & \text{в остальных случаях} \end{cases}$
7	Переменная x изменяется в интервале от – 10 до 10 с шагом 1. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x-2)! & \text{при } 3 \le x \le 6 \\ (x-4)! & \text{при } 6 \le x \le 10 \\ x^4 & \text{в остальных случаях} \end{cases}$
8	Переменная x изменяется в интервале от – 14 до 11 с шагом 1. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x-2)! & \text{при } 3 \le x \le 6 \\ (x-4)! & \text{при } 7 < x \le 10 \\ x^4 & \text{в остальных случаях} \end{cases}$
9	Переменная x изменяется в интервале от – 5 до 10 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x+5)! & \text{при } -2 \le x \le 4 \\ (x-3)! & \text{при } 6 < x \le 9 \\ x^2 & \text{в остальных случаях} \end{cases}$
10	Переменная x изменяется в интервале от – 15 до 12 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x+9)! & \text{при } -8 \le x \le 4 \\ (x-6)! & \text{при } 8 < x \le 12 \\ tg(x) & \text{в остальных случаях} \end{cases}$

11	Переменная x изменяется в интервале от – 5 до 20 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x-7)! & \text{при } 8 \le x \le 11 \\ (x-11)! & \text{при } 12 < x \le 19 \\ x+5 & \text{в остальных случаях} \end{cases}$
12	Переменная x изменяется в интервале от – 10 до 12 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x-1)! & \text{при } 2 \le x \le 5 \\ (x-3)! & \text{при } 6 < x \le 10 \\ x^4 & \text{в остальных случаях} \end{cases}$
13	Переменная x изменяется в интервале от – 13 до 12 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x-3)! & \text{при } 4 \le x \le 7 \\ (x-5)! & \text{при } 8 < x \le 10 \\ x^2 & \text{в остальных случаях} \end{cases}$
14	Переменная x изменяется в интервале от – 13 до 12 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x-3)! & \text{при } 4 \le x \le 7 \\ (x-5)! & \text{при } 8 < x \le 10 \\ x^2 & \text{в остальных случаях} \end{cases}$
15	Переменная x изменяется в интервале от – 15 до 14 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x+9)! & \text{при} -8 \le x \le 4 \\ (x-6)! & \text{при } 8 < x \le 12 \\ tg(x) & \text{в остальных случаях} \end{cases}$
16	Переменная x изменяется в интервале от – 5 до 22 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x-6)! & \text{при } 8 \le x \le 11 \\ (x-10)! & \text{при } 12 < x \le 19 \\ x+4 & \text{в остальных случаях} \end{cases}$
17	Переменная x изменяется в интервале от – 11 до 14 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x-2)! & \text{при } 3 \le x \le 6 \\ (x-4)! & \text{при } 7 < x \le 10 \\ x^3 & \text{в остальных случаях} \end{cases}$
18	Переменная x изменяется в интервале от 1 до 11 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x-1)! & \text{при } 2 \le x \le 6 \\ (x-4)! & \text{при } 7 < x \le 10 \\ x^2 & \text{в остальных случаях} \end{cases}$
19	Переменная x изменяется в интервале от – 5 до 15 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x+3)! & \text{при} -2 \le x \le 4 \\ (x-5)! & \text{при } 6 < x \le 9 \\ 5x & \text{в остальных случаях} \end{cases}$
20	Переменная x изменяется в интервале от – 9 до 14 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x+5)! & \text{при} -4 \le x \le 4 \\ (x-6)! & \text{при } 8 < x \le 12 \\ tg(x) & \text{в остальных случаях} \end{cases}$
21	Переменная x изменяется в интервале от – 3 до 23 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x-6)! & \text{при } 8 \le x \le 11 \\ (x-10)! & \text{при } 12 < x \le 19 \\ x+6 & \text{в остальных случаях} \end{cases}$

22	Переменная x изменяется в интервале от –11 до 14 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x-2)! & \text{при } 2 \le x \le 6 \\ (x-4)! & \text{при } 7 < x \le 10 \\ 4x & \text{в остальных случаях} \end{cases}$
23	Переменная x изменяется в интервале от – 7 до 12 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x-3)! & \text{при } 3 \le x \le 6 \\ (x-7)! & \text{при } 6 < x \le 9 \\ x^2 & \text{в остальных случаях} \end{cases}$
24	Переменная x изменяется в интервале от – 4 до 16 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x+5)! & \text{при } -4 \le x \le 4 \\ (x-5)! & \text{при } 6 < x \le 9 \\ 5x^2 & \text{в остальных случаях} \end{cases}$
25	Переменная x изменяется в интервале от – 12 до 16 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x+7)! & \text{при } -6 \le x \le 3 \\ (x-4)! & \text{при } 5 \le x \le 12 \\ tg\left(\frac{x}{2}\right) & \text{в остальных случаях} \end{cases}$
26	Переменная x изменяется в интервале от – 8 до 15 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x-2)! & \text{при } 3 \le x \le 10 \\ (x+6)! & \text{при } -5 \le x \le 1 \\ 2x & \text{в остальных случаях} \end{cases}$
27	Переменная x изменяется в интервале от – 5 до 25 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x+2)! & \text{при } -1 \le x \le 4 \\ (x-4)! & \text{при } 6 \le x \le 10 \\ x^3 & \text{в остальных случаях} \end{cases}$
28	Переменная x изменяется в интервале от – 14 до 16 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x+5)! & \text{при } -4 \le x \le 6 \\ (x-3)! & \text{при } 7 < x \le 10 \\ 2x^2 & \text{в остальных случаях} \end{cases}$
29	Переменная x изменяется в интервале от – 15 до 20 с шагом 3. Вывести на экран значение функции в зависимости от переменной. И найти максимальное значение.	$y = \begin{cases} (x+5)! & \text{при } -4 \le x \le 5 \\ (x-3)! & \text{при } 8 < x \le 9 \\ \frac{x^2}{2} & \text{в остальных случаях} \end{cases}$
30	Переменная x изменяется в интервале от – 10 до 20 с шагом 2. Вывести на экран значение функции в зависимости от переменной. И найти минимальное значение.	$y = \begin{cases} (x+3)! & \text{при } -2 \le x \le 5 \\ (x-3)! & \text{при } 6 < x \le 10 \\ x^2/2 & \text{в остальных случаях} \end{cases}$

Задание №6

Решить задачу с применением одномерных массивов.

№	Задание
1	Дан массив 9 элементов, задается случайным образом в диапазоне от -10 до 15 найти сумму элементов, стоящих на нечетных позициях.
2	Ввести с клавиатуры информацию о температуре воздуха за 2 недели в массив. Определить сколько раз она была ниже нуля.
3	Дан массив 12 элементов, задается случайным образом в диапазоне от -99 до 99 найти сумму отрицательных элементов.
4	Дан массив A 18 элементов, задается случайным образом в диапазоне от -25 до 40. Сформировать массив B из элементов массива A, где вначале стоят элементы с четных позиций, а потом с нечетных
5	Составить массив A из отрицательных значений функции Z = sin x / cos (x-2) для x, изменяющихся на отрезке [-11;12] с шагом 0,56.
6	Дан массив 60 элементов, задается случайным образом трехзначными числами, найти сумму элементов, имеющих индексы кратные 5;
7	Ввести с клавиатуры информацию о температуре воздуха за 2 недели. Определить среднюю температуру за данный период.
8	Дан массив 20 элементов, задается случайным образом двузначными числами найти сумму элементов, стоящих на четных позициях.
9	Составить массив C из положительных значений функции Z = cos (x+2) / sin x для x, изменяющихся на отрезке [-10;15] с шагом 0,35.
10	Дан массив 18 элементов, задается случайным образом в диапазоне от -63 до 48 найти минимальный, среди элементов, стоящих на нечетных позициях.
11	Дан массив 60 элементов, задается случайным образом двузначными числами, найти произведение элементов, имеющих индексы кратные 3;
12	Дан массив 25 элементов, задается случайным образом в диапазоне от -42 до 65 найти максимальный, среди элементов, стоящих на четных позициях.
13	Дан массив 45 элементов, задается случайным образом двузначными числами, найти произведение элементов, имеющих индексы кратные 5;
14	Дан массив 25 элементов, задается случайным образом в диапазоне от -33 до 33 найти максимальный, среди элементов, стоящих на нечетных позициях.
15	Дан массив 15 элементов, задается случайным образом в диапазоне от -50 до 50 найти сумму положительных элементов.
16	Дан массив 50 элементов, задается случайным образом трехзначными числами, найти сумму элементов, имеющих индексы кратные 3;
17	Ввести с клавиатуры информацию о температуре воздуха за 2 недели. Определить сколько раз она была выше нуля.
18	Дан массив 50 элементов, задается случайным образом двузначными числами, найти среднее арифметическое элементов, имеющих индексы кратные 4;

19	Составить массив B из отрицательных значений функции Z = cos x / sin(x-2) для x, изменяющихся на отрезке [-5;10] с шагом 0,67.
20	Дан массив A 20 элементов, задается случайным образом в диапазоне от -50 до 50. Сформировать массив B из элементов массива A, где вначале стоят положительные элементы, а потом отрицательные.
21	Дан массив 16 элементов, задается случайным образом в диапазоне от -79 до 29 найти минимальный, среди элементов, стоящих на четных позициях.
22	Дан массив 40 элементов, задается случайным образом трехзначными числами, найти сумму элементов, имеющих индексы кратные 4;
23	Дан массив 50 элементов, задается случайным образом двузначными числами, найти среднее арифметическое элементов, имеющих индексы кратные 3;
24	Составить массив M из положительных значений функции Z = sin x / cos(x+2) для x, изменяющихся на отрезке [-15;14] с шагом 0,83.
25	Дан массив 60 элементов, задается случайным образом двузначными числами, найти среднее арифметическое элементов, имеющих индексы кратные 5;
26	Дан массив 20 элементов, задается случайным образом в диапазоне от -65 до 34 найти среднее арифметическое, среди элементов, стоящих на четных позициях.
27	Дан массив 10 элементов, задается случайным образом трехзначными числами, найти произведение элементов кратных 6;
28	Дан массив 35 элементов, задается случайным образом двузначными числами, найти среднее арифметическое элементов, имеющих индексы кратные 4;
29	Составить массив M из положительных значений функции $Z = \frac{(x-10)^3}{-x}$ для x, изменяющихся на отрезке [-15;14] с шагом 0,83.
30	Дан массив 60 элементов, задается случайным образом двузначными числами, найти максимальный элемент среди кратных 7;

Задание №7

Решить задачу с применением многомерных массивов.

№	Задание
1	Дана матрица 7*8, задается случайными двузначными числами. Упорядочить матрицу по возрастанию элементов первой строки.
2	Дана матрица 7*6, задается случайными двузначными числами. Найти произведение элементов четных столбцов.
3	Дана матрица 7*7, задается случайными числами в диапазоне от -20 до 20. Найти произведение положительных элементов главной диагонали.
4	Дана матрица 6*8, задается случайными двузначными числами. Найти сумму элементов нечетных столбцов.
5	Дана матрица 5*7, задается случайными двузначными числами. Упорядочить матрицу по убыванию элементов первого столбца строки.
6	Дана матрица 8*8, задается случайными числами в диапазоне от -33 до 29. Найти произведение отрицательных элементов главной диагонали.
7	Дана матрица 8*6, задается случайными двузначными числами. Упорядочить матрицу по возрастанию элементов первого столбца.
8	Дана матрица 6*6, задается случайными числами в диапазоне от -40 до 56. Найти произведение положительных элементов побочной диагонали.
9	Дана матрица 6*7, задается случайными двузначными числами. Упорядочить матрицу по убыванию элементов последнего столбца.
10	Дана матрица 8*8, задается случайными числами в диапазоне от -43 до 22. Найти произведение отрицательных элементов побочной диагонали.
11	Дана матрица 7*8, задается случайными двузначными числами. Найти сумму элементов четных столбцов.
12	Дана матрица 6*10, задается случайными двузначными числами. Упорядочить матрицу по возрастанию элементов четвертой строки.
13	Дана матрица 8*7, задается случайными двузначными числами. Найти сумму элементов нечетных строк.
14	Дана матрица 10*9, задается случайными двузначными числами. Упорядочить матрицу по убыванию элементов пятой строки.
15	Дана матрица 8*8, задается случайными двузначными числами. Упорядочить матрицу по возрастанию элементов шестого столбца.
16	Дана матрица 9*8, задается случайными двузначными числами. Упорядочить матрицу по убыванию элементов седьмого столбца.
17	Дана матрица 8*8, задается случайными числами в диапазоне от -30 до 30. Найти сумму положительных элементов главной диагонали.
18	Дана матрица 5*7, задается случайными двузначными числами. Упорядочить матрицу по возрастанию элементов последней строки.

19	Дана матрица 9*9, задается случайными числами в диапазоне от -24 до 27. Найти сумму отрицательных элементов главной диагонали.
20	Дана матрица 9*6, задается случайными двузначными числами. Упорядочить матрицу по убыванию элементов последней строки.
21	Дана матрица 9*9, задается случайными числами в диапазоне от -38 до 23. Найти сумму положительных элементов побочной диагонали.
22	Дана матрица 6*9, задается случайными двузначными числами. Упорядочить матрицу по убыванию элементов первой строки.
23	Дана матрица 7*7, задается случайными числами в диапазоне от -44 до 67. Найти сумму отрицательных элементов побочной диагонали.
24	Дана матрица 8*8, задается случайными двузначными числами. Найти произведение элементов четных строк.
25	Дана матрица 9*6, задается случайными двузначными числами. Найти сумму элементов четных строк.
26	Дана матрица 8*8, задается случайными двузначными числами. Упорядочить матрицу по возрастанию элементов последнего столбца.
27	Дана матрица 7*5, задается случайными двузначными числами. Найти произведение элементов нечетных строк.
28	Дана матрица 8*7, задается случайными двузначными числами. Найти среднее арифметическое элементов нечетных строк.
29	Дана матрица 9*7, задается случайными двузначными числами. Упорядочить матрицу по убыванию элементов третьего столбца
30	Дана матрица 9*9, задается случайными двузначными числами. Найти произведение элементов нечетных столбцов.

Задание №8

Дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания.

№	Задание
1	Вывести только те слова сообщения, в которых содержится заданная подстрока.
2	Вывести только те слова сообщения, которые не более заданной длины,
3	Вывести только те слова сообщения, которые начинаются с прописной буквы.
4	Вывести только те слова сообщения, которые содержат хотя бы одну цифру.
5	Удалить из сообщения все слова, которые заканчиваются на заданный символ.
6	Удалить из сообщения все слова, содержащие заданный символ (без учета регистра).
7	Удалить из сообщения все однобуквенные слова (вместе с лишними пробелами)
8	Удалить из сообщения все повторяющиеся слова (без учета регистра).
9	Подсчитать сколько раз заданное слово встречается в сообщении.
10	Подсчитать сколько слов, состоящих только из прописных букв, содержится в сообщении.
11	Найти все самые длинные слова сообщения и вывести их.
12	Найти все самые короткое слово сообщения и вывести их.
13	Вывести только те слова, которые встречаются в тексте ровно один раз.
14	Вывести только те слова, которые встречаются более n раз.
15	Вывести слова сообщения в порядке возрастания их длин.
16	Определите, содержится ли в сообщении заданное слово.
17	Вывести все слова заданной длины.
18	Вывести на экран все слова сообщения, записанные с заглавной буквы.
19	Удалить из сообщения все знаки препинания.
20	Удалить из сообщения все русские слова.
21	Удалить из сообщения только те русские слова, которые начинаются на гласную букву.
22	Заменить все английские слова на многоточие.
23	Определить, какой из двух заданных символов встречается чаще в строке.
24	Определить, имеются ли в строке два соседствующих одинаковых символа.
25	Удалить среднюю букву, если длина строки нечетная, и две средних, если длина строки четная.
26	Удвоить каждое вхождение заданного символа.
27	Заменить все вхождения подстроки substr1 на подстроку substr2, substr1 и substr2 задаются с клавиатуры.

28	Подсчитать сумму всех содержащихся в ней цифр.
29	Подсчитать количество содержащихся в ней цифр.
30	Найти порядковые номера первого и последнего вхождения заданного символа.