МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И

МАССОВЫХ КОММУНИКАЦИЙ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра информатики и вычислительной техники

**Т.А. КОВАЛЕНКО, А.Г. СОЛОДОВ**

Вычислительная техника и языки программирования

Часть 1

УЧЕБНОЕ ПОСОБИЕ

Самара, 2021

УДДК - 004.772

ББК 3.32.97

К-56

Рекомендовано к изданию методическим советом ПГУТИ, протокол

№\_\_\_\_\_ от \_\_\_ \_\_\_\_\_\_\_\_ 2021

Рецензент:

Коваленко Т.А

К-56 Вычислительная техника и языки программирования. Часть 1: учебное пособие/ Т.А. Коваленко, А.Г. Солодов–Самара: ИНУЛ ПГУТИ, 2021. -

ISBN

Учебное пособие предназначено для студентов первого курса дневной и заочной формы обучения. В нем рассмотрены вопросы программирования на языке С# в рамках учебной дисциплины «Информатика».

Пособие представлено в двух частях теоретической и практической. В теоретическое части дается представление о пакете C# его возможностях, которые позволяют решать задачи, используя язык программирования. Вторая часть состоит из 14 лабораторных работ. Все задачи классифицированы, т.е. объединены в некоторые группы.

Пособие позволяет рассмотреть не только теоретические вопросы, но и выполнить самостоятельно лабораторные работы.

Использование данного учебного пособия является хорошим подспорьем для студентов технических специальностей.

Данное пособие поможет студентам использовать навыки программирования в технических приложениях (ОПК-4), повысить знания принципов алгоритмизации и программирования (ОПК-4) и овладеть основными методами работы на компьютере с использованием универсальных прикладных программ (ОПК-4).

Материал, представленный в учебном пособии, является актуальным. Он изложен доступным для студентов языком.

Учебное пособие является необходимым и полезным в учебном процессе.

©, Коваленко Т.А, 2021

Оглавление

[Введение 6](#_Toc67238972)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc67238973)

[Лекция 1 8](#_Toc67238974)

[Процесс создания программного обеспечения 8](#_Toc67238975)

[1.1 Машинный язык. 8](#_Toc67238976)

[1.2 Процесс создания программного обеспечения 9](#_Toc67238977)

[1.3 Классификация программных продуктов 10](#_Toc67238978)

[1.4 Классификация языков программирования 13](#_Toc67238979)

[1.4 Маркетинг ПО 15](#_Toc67238980)

[Лекция 2 17](#_Toc67238981)

[Интегрированная среда разработки (ИСР) Visual C# 17](#_Toc67238982)

[2.1 Основные характеристики. 17](#_Toc67238983)

[2.2 Алфавит языка 23](#_Toc67238984)

[2.3 Словарь языка 24](#_Toc67238985)

[2.4 Структура программы консольного приложения на C# 27](#_Toc67238986)

[2.5 Базовые алгоритмические структуры 28](#_Toc67238987)

[2.6 Методы алгоритмизации задач 31](#_Toc67238988)

[Лекция 3 32](#_Toc67238989)

[Типы данных 32](#_Toc67238990)

[3.1. Типы значений 32](#_Toc67238991)

[3.2. Ссылочные типы 33](#_Toc67238992)

[3.3 Классы 33](#_Toc67238993)

[3.4. Задание типов в объявлениях переменных 34](#_Toc67238994)

[3.5. Преобразования типов 35](#_Toc67238995)

[3.6. Стандартное форматирование чисел 35](#_Toc67238996)

[3.7. Инструкции, команды, выражения 38](#_Toc67238997)

[Лекция 4 41](#_Toc67238998)

[Операторы 41](#_Toc67238999)

[4.1 Основные операторы 41](#_Toc67239000)

[4.2 Унарные операции 42](#_Toc67239001)

[4.3 Мультипликативные операторы 43](#_Toc67239002)

[4.4 Аддитивные операторы 44](#_Toc67239003)

[4.5 Операторы отношений 44](#_Toc67239004)

[4.6 Операторы присвоения 46](#_Toc67239005)

[4.7 Инструкции 48](#_Toc67239006)

[Оператор перехода goto 49](#_Toc67239007)

[Лекция 5 51](#_Toc67239008)

[Инструкции управления 51](#_Toc67239009)

[5.1 Инструкция if 51](#_Toc67239010)

[5.2 Вложенные инструкции if; else … 53](#_Toc67239011)

[5.3 Инструкции выбора switch, case 55](#_Toc67239012)

[Лекция 6 61](#_Toc67239013)

[Циклы 61](#_Toc67239014)

[6.1 Операторы цикла 61](#_Toc67239015)

[6.2 Использование нескольких управляющих переменных цикла 66](#_Toc67239016)

[6.3 Использование переменных цикла c любым шагом их изменения 67](#_Toc67239017)

[6.4 Цикл с предусловием. Инструкция while . . 68](#_Toc67239018)

[6.5 Цикл с постусловием. Инструкция do - while 71](#_Toc67239019)

[6.6 Накопление сумм и произведений 73](#_Toc67239020)

[6.7 Безусловные переходы 76](#_Toc67239021)

[Лекция 7 79](#_Toc67239022)

[Массивы 79](#_Toc67239023)

[7.1 Операции с массивами 82](#_Toc67239024)

[Лекция 8 98](#_Toc67239025)

[Символы и строки 98](#_Toc67239026)

[8.1 Символы 98](#_Toc67239027)

[8.2 Строки 99](#_Toc67239028)

[8.3 Методы обработки строк 102](#_Toc67239029)

[8.4. Строковые функции и процедуры 103](#_Toc67239030)

[Лекция 9 105](#_Toc67239031)

[Классы, объекты и методы. Подпрограммы 105](#_Toc67239032)

[9.1 Введение в классы, объекты и методы 105](#_Toc67239033)

[9.2 Синтаксис класса 106](#_Toc67239034)

[9.3. Основные понятия подпрограммы 107](#_Toc67239035)

[9.4Сравнение вариантов 108](#_Toc67239036)

[9.5 Описание методов (процедур и функций) 110](#_Toc67239037)

[9.6 Аргументы и параметры 111](#_Toc67239038)

[9.7 Тело метода (процедуры и функции) 113](#_Toc67239039)

[Лекция 10 117](#_Toc67239040)

[Исключения 117](#_Toc67239041)

[10.1 Виды исключений: 118](#_Toc67239042)

[10.2 Типы блоков работающих с исключением 119](#_Toc67239043)

[10.3 Форматы ввода/вывода 125](#_Toc67239044)

[Лекция 11 128](#_Toc67239045)

[Приложение под ОС Windows 128](#_Toc67239046)

[11.1. Основные характеристики приложения Windows Form 128](#_Toc67239047)

[11.2 Окна «Конструктора формы», «Редактора кода», «Свойства элементов» 130](#_Toc67239048)

[11.3. Работа с проектом 132](#_Toc67239049)

[11.4 Шаблон кода: 134](#_Toc67239050)

[11.4 Разработка приложения 135](#_Toc67239051)

[Лекция 12 146](#_Toc67239052)

[Графика в C# 146](#_Toc67239053)

[12.1. Рисованные изображения 146](#_Toc67239054)

[12.2 Объект Graphics 146](#_Toc67239055)

[12.3 Методы рисования 149](#_Toc67239056)

[12.4 Методы заливки 150](#_Toc67239057)

[12.5 Рисование с помощью примитивов 151](#_Toc67239058)

[12.6 Рисование графика функции 152](#_Toc67239059)

[12.7 Растровая графика 156](#_Toc67239060)

[Лабораторные работы 161](#_Toc67239061)

[Лабораторная работа № 1 162](#_Toc67239062)

[Численные типы в языке C# 162](#_Toc67239063)

[Лабораторная работа № 2 164](#_Toc67239064)

[Логические операции 164](#_Toc67239065)

[Лабораторная работа № 3 165](#_Toc67239066)

[Линейные структуры 165](#_Toc67239067)

[Лабораторная работа № 3 168](#_Toc67239068)

[Структура ветвление 168](#_Toc67239069)

[Лабораторная работа № 4 174](#_Toc67239070)

[Циклы с неизвестным числом повторений 174](#_Toc67239071)

[Лабораторная работа 5 178](#_Toc67239072)

[Циклы с заданным числом повторений 178](#_Toc67239073)

[Приложение А 181](#_Toc67239074)

[Стандартные блок-схемы 181](#_Toc67239075)

[Приложение Б 183](#_Toc67239076)

[Встроенные базовые типы 183](#_Toc67239077)

[Типы целых чисел 183](#_Toc67239078)

[Типы чисел с плавающей запятой 184](#_Toc67239079)

[Значения типов по умолчанию 184](#_Toc67239080)

[Приложение В 185](#_Toc67239081)

[Класс System.Math 185](#_Toc67239082)

[Поля класса 185](#_Toc67239083)

[Методы класса. Имена с заглавной буквы 185](#_Toc67239084)

# Введение

Данное пособие рассчитано для использования студентами 1 курса на лабораторно–практических занятиях по предмету «Вычислительная техника и языки программирования».

Этот предмет состоит из двух частей. В первой части мы изучаем язык программирования С#/ и это пособие предназначено для ознакомления с основными методами, применяемыми для решения различных задач.

Сегодня можно выделить три основных языка программирования, которые, являются ключевыми: язык Java, С++, C#.

Проанализировав рынок использования языков программирования в настоящее время, был сделан вывод, что язык C# востребован на рынке, поэтому учебное пособие посвящено изучению этого языка. Так же следует отметить что, все три языка сходны по структуре и их написанию, поэтому если изучить хотя бы один из них, при желании можно выучить и другие.

Язык программирования C# был разработан в качестве эффективного, надежного и простого в использовании средства. В настоящее время C# является частью системы Visual Studio.NET, предназначенной для удобной работы с платформой .NET Framework и создания нового способа написания надежного программного обеспечения для высокопроизводительных серверов, компактных мобильных устройств и многого другого.

Первым рассматривается консольное приложение, здесь в лабораторном комплексе особое внимание уделяется блок схемам, умение строить решение задачи по блок схеме. Затем рассматривается блок с применением Windows Form. Пособие предусматривает получение студентами навыков самостоятельного написания программ.

Каждая тема содержит теоретический материал и примеры использования языка программирования для решения конкретных задач. В учебном пособии дается описания основных вычислительных алгоритмов, тексты программ и описание стандартных функций пакета C#, реализующих изученные вычислительные алгоритмы.

Для закрепления теоретического материала предусматривается выполнение лабораторных работ по основным рассматриваемым темам.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В теоретической части рассматриваются вопросы написания программ для решения задач с помощью языка C#. Описываются простейшие конструкции программы, структура языка. Дается определение алгоритма, переменной, понятие блок схемы. Подробно рассматривается вопрос построения программы с помощью C#.

**Цель теоретической части:** Дать представление о пакете C# его структуре и внешнем виде. Ознакомиться с основными функциями в этой системе. Уяснить предназначение функций. Понять что такое идентификатор, переменная, константа, типы данных, код программы. Научиться писать программы на языке C#.

# Лекция 1

# Процесс создания программного обеспечения

**Цель лекции:** Ознакомиться с основными понятиями, встречающимися при изучении программирования. Уяснить что такое машинный язык, из чего состоит процесс создания программного обеспечения, что в себя включают программные продукты и какова их классификация.

Прежде чем программировать компьютер, мы должны понять, как он работает.

***Кодинг*** *–* это постоянная борьба с машиной**.** Нужно заставлять её делать то, что тебе нужно. Поэтому любой программист просто обязан знать его внутренности. Компьютер состоит из следующих основных компонентов: процессор, память, видеокарта, винчестер (жёсткий диск) и различные разъёмы для подключения дополнительных устройств. Все эти компоненты связаны между собой с помощью шлейфов и шин.

Вся информация в компьютере хранится на винчестере. Когда ты запускаешь какую-нибудь программу, то она сначала загружается в память и только потом процессор начинает выполнять содержащиеся в ней инструкции. Чем больше программа, тем дольше

она загружается.

Результат работы программы выводится на экран через видеокарту. На любой видеокарте есть чип памяти, в котором отображается всё содержимое экрана. Когда нужно вывести что-то на экран, просто копируются данные в видеопамять, и видеокарта автоматически выводит его содержимое на монитор.

Это всё, что необходимо знать о работе компьютера.

## 1.1 Машинный язык.

Данные на диске хранятся в двоичном виде. Даже текстовые файлы на диске выглядят в виде нулей и единиц. Точно так же выглядит и любая программа, только её называют машинным кодом. Давай с ним познакомимся немного поближе.

Любая программа представляет собой последовательность команд. Эти команды называются *процессорными инструкциями*.

По этим инструкциям процессор определяет, что и как ему нужно делать. Когда ты запускаешь программу, компьютер загружает её машинный код в память и начинает выполнять. Наша задача, написать эти инструкции, чтобы компьютер понял, что мы от него хотим.

Реальная программа, которую выполняет компьютер, представляет собой последовательность единиц и нулей. Такую последовательность называют машинным языком. Но человек не способен эффективно думать единицами и нулями. Для нас легче воспринимается осмысленный текст, а не сумасшедшие числа в двоичной системе измерения, с которой мы не привыкли работать.

Например, команда складывания двух регистров выглядит так: #03C3. Нам это мало о чём говорит, и запомнить такую команду очень тяжело. На много проще написать «сложить число1+ число2».

Первое время программисты писали в машинных кодах, пока кому-то не пришла в голову идея: «Почему бы не писать текст программы на понятном языке, а потом заставлять компьютер переводить этот текст в машинный код?». Идея действительно заслуживала внимания. Так появился первый компилятор – программа, которая переводила текст программ в машинный код.

Множество управляющих сигналов можно связать с набором 0 и 1, которые можно интерпретировать, как число. Например, 0110001100110101.

Программа, с которой работает процессор, это последовательность чисел, называемая машинным кодом.

## 1.2 Процесс создания программного обеспечения

***Программа*** – упорядоченная последовательность команд компьютера для решения задачи.

***Программное обеспечение (ПО)*** – совокупность программ обработки данных и необходимых для их эксплуатации документов.

Процесс создания программ можно представить как последовательность следующих действий:

* постановка задачи,
* алгоритмизация решения задачи
* программирование.

Постановка задачи связана с конкретизацией основных параметров ее реализации, определением источников и структуры входной (исходные данные) и выходной (вид документов, сигналы управления) информации.

***Алгоритм*** – система точно сформулированных правил, определяющая процесс преобразования входной информации в выходную информацию за конечное число шагов.

В алгоритме отражаются логика и способ формирования результатов решения с указанием необходимых расчетных формул. В него входят логические условия и соотношения для контроля достоверности выходных результатов.

Алгоритм решения задачи имеет ряд обязательных свойств:

* дискретность – разбиение процесса обработки информации на простые этапы (шаги), выполнение которых компьютером или человеком не вызывает затруднений;
* определенность – однозначность выполнения каждого шага преобразования информации;
* выполнимость – конечность действий алгоритма решения задачи, позволяющая получить желаемый результат за конечное число шагов;
* массовость – пригодность алгоритма для решения определенного класса задач.

Способы описания алгоритмов:

* Словесный - описание порядка действий на естественном языке;
* Графический - с использованием блок-схем алгоритма в виде графических символов. Размеры блоков стандартизированы;
* Программный - текст на языке программирования. Лаконичный, наглядный.

Алгоритм, написанный на языке программирования, называется ***программой***.

При работе с алгоритмами используют понятие оператора.

***Оператор*** – это формальная запись инструкций по выполнению некоторой последовательности действий.

***Программирование*** – теоретическая и практическая деятельность, связанная с созданием программ.

Все программы можно разделить на два класса:

* утилитарные программы
* программные продукты.

Утилитарные программы («программы для себя») предназначены для удовлетворения нужд разработчиков. Чаще всего они играют роль сервиса в технологии обработки данных.

Программные продукты (изделия) предназначены для удовлетворения потребностей пользователей и представляют собой комплекс взаимосвязанных программ для решения определенной задачи массового спроса.

Как правило, программные продукты требуют сопровождения, которое осуществляется фирмами-распространителями программ (дистрибьюторами), реже – фирмами-разработчиками.

Сопровождение программного продукта – поддержка его работоспособности, переход на его новые версии, внесение изменений, исправление обнаруженных ошибок и т.п.

## 1.3 Классификация программных продуктов

По области использования можно выделить три класса программных продуктов:

* системное программное обеспечение
* пакеты прикладных программ
* инструментарий технологии программирования.

***Системное программное обеспечение*** – совокупность программ и программных комплексов для обеспечения работы компьютера и сетей ЭВМ. Данный класс программных продуктов тесно связан с типом компьютера и является его неотъемлемой частью (рис.1.1).

Базовое программное

обеспечение

Операционная система

Операционная оболочка

Сетевая операционная

система

Системное программное обеспечение

Сервисное программное

обеспечение

Программы диагностики работоспособности компьютера

Антивирусные программы

Программы обслуживания дисков

Программы архивирования данных

Программы обслуживания сети

Рис. 1.1 – Классификация системного программного обеспечения

Базовое программное обеспечение, как правило, поставляется вместе с компьютером, а сервисное может быть приобретено дополнительно.

***Пакеты прикладных программ*** (ППП) – комплекс взаимосвязанных программ для решения задач определенного класса конкретной предметной области.

Предметная (прикладная) область – совокупность связанных между собой функций (задач управления), с помощью которых достигается выполнение поставленных целей.

***Инструментарий технологии программирования*** – программные продукты поддержки (обеспечения) технологии программирования (см. рис.1.2).

Инструментарий технологии программирования

Средства для создания приложений

Средства для создания информационных систем (CASE-технология)

Локальные средства

Встроенные в систему

реализации

Рис.1.2 – Классификация инструментария технологии программирования

Интегрированные системы

Языки

программирования

Инструментальная среда пользователя

Независимые от системы

реализации

Средства для создания приложений – совокупность языков и систем программирования, а также различные программные комплексы для отладки и поддержки создаваемых программ.

Язык программирования – формальный язык для описания алгоритма решения задачи на компьютере. Языки программирования по синтаксису образования их конструкций можно условно разделить на следующие классы:

* машинные (машинные коды), воспринимаемые аппаратной частью ЭВМ;
* машинно-ориентированные (ассемблеры), отражающие структуру конкретного типа процессора;
* алгоритмические (Паскаль, Си и др.), отражающие структуры алгоритма и не зависящие от типа процессора;
* проблемно-ориентированные (Лисп, Симула и др.), предназначенные для решения задач определенного класса.

Для создания программы на выбранном языке программирования нужно иметь следующие компоненты:

* транслятор – программа, автоматически преобразующая исходный текст в промежуточный объектный код (двоичный файл с расширением .obj).

Трансляция может выполняться с использованием компиляторов или интерпретаторов. Компилятор полностью обрабатывает весь текст программы, отыскивает синтаксические и семантические (смысловые) ошибки и при их отсутствии генерирует объектный код. Интерпретатор анализирует структуру каждого очередного оператора языка и затем сразу его исполняет. Нередко компилятор выполняет оптимизацию, в результате чего законченная программа работает в сотни раз быстрее программы, выполняемой с помощью интерпретатора;

* редактор связей (компоновщик) – программа, связывающая объектные модули (объектные коды отдельных исходных текстовых файлов большой программы) и добавляющая к ним машинные коды подпрограмм, реализующих различные стандартные функции. На выходе компоновщика формируется исполнимый код, который можно запустить на любом компьютере, где установлена операционная система, для которой эта программа создавалась. Как правило, итоговый файл имеет расширение .ехе или .com.

Инструментальная среда пользователя представлена специальными средствами: библиотека функций, макрокоманды и многое другое.

***Интегрированные системы*** программирования предназначены для повышения производительности труда программистов. Они включают в себя:

* текстовый редактор, где ключевые слова и идентификаторы выделяются разными цветами и шрифтами и, кроме того, автоматически проверяется правильность синтаксиса программы непосредственно во время ее ввода;
* компилятор, редактор связей и библиотеки функций;
* отладчик, который позволяет анализировать работу программы во время ее выполнения с целью обнаружения и устранения ошибок. С его помощью можно последовательно выполнять отдельные операторы исходного текста по шагам, наблюдая при этом, как меняются значения различных переменных.

В интегрированных системах все этапы создания программы автоматизированы: после того как исходный текст введен, его компиляция и сборка выполняются одним нажатием клавиши.

***CASE-технология* –** программный комплекс, автоматизирующий весь технологический процесс анализа, проектирования, разработки и сопровождения сложных программных систем. Основное достоинство CASE-технологии – это поддержка коллективной работы над проектом за счет возможности работы в локальной сети разработчиков. Поддержание экспорта/импорта любых фрагментов проекта, организационного управления проектом. Средства CASE-технологий делятся на две группы:

* встроенные в систему реализации – все решения по проектированию и реализации привязаны к выбранной системе управления базами данных;
* независимые от системы реализации – все решения по проектированию ориентированы на унификацию разрабатываемых алгоритмов и программ и средств их документирования, что обеспечивает большую гибкость в выборе средств реализации.

## 1.4 Классификация языков программирования

ЭВМ исполняет программу в машинных кодах. А составляют программу люди на удобном для себя языке.

Различают языки: высокого уровня и низкого уровня (машинно-ориентированные) (табл.1.1 и табл.1.2)

Языки высокого уровня бывают:

* процедурно-ориентированные. Содержат набор универсальных команд;
* проблемно-ориентированные. Имеют команды узкого назначения;
* объектно-ориентированные. Программирование на уровне объектов;
* событийно-ориентированные. Программирование на уровне событий;
* визуальные. Поддерживают визуальное программирование;
* комплексные. Поддерживают многие из перечисленных свойств.

Различают пять поколений языков программирования:

Начало 1950-х годов. Язык Ассемблера. Его принцип "Одна инструкция – одна строка". Инструкция на языке однозначно соответствует машинному коду команды.

Начало 1950-х – конец 1960-х годов. Язык символического Ассемблера. В нем появилось понятие переменной.

1960-е годы. Универсальные языки программирования.

С начала 1970-х годов до настоящего времени. Проблемно-ориентированные языки для создания проектов в узкой предметной области.

С середины 1990-х годов до настоящего времени. Языки с автоматизацией программирования. Примеры – языки визуального программирования.

Таблица 1.1

Языки низкого уровня

|  |  |
| --- | --- |
| Язык | Расшифровка |
| Assembler | Ассемблер |
| Macro Assembler | Макро Ассемблер |

Таблица 1.2

Языки высокого уровня

|  |  |  |
| --- | --- | --- |
| Язык | Расшифровка | Примечание |
| Fortran | Formula Translator | От слов - транслятор формул. |
| BASIC | Beginner’s All-purpose Symbolic  Instruction Code | Многоцелевой мнемокод для начинающих. Создан в 1960-е годы. |
| Visual Basic | Визуальный BASIC | Язык 5-го поколения. Его версия – рабочий язык пакета Microsoft Office. |
| Cobol | Common Business Oriented Language | Язык для задач в экономике бизнесе. |
| Algol | Algorithmic Language | Создан для описания алгоритмов. Не получил широкого распространения. |
| Pascal |  | Универсальный язык. Создан в 1970-х годах. Один из часто применяемых. |
| С |  | Язык для системного программирования. Создан в 1970-х годах компанией Bell. |
| С++ |  | Объектно-ориентированное расширение С++. Создан в 1980 году Страуструпом. |
| С# | С шарп | Многоплатформенная версия С++ |
| Java |  | Модификация С для Internet. В нем удалены низкоуровневые возможности языка С. |

***Показатели качества ПО***

Перечень показателей качества:

* документированность,
* эффективность,
* простота использования
* удобство эксплуатации,
* мобильность,
* совместимость,
* испытуемость,
* стоимость.

Показатели качества производителя ПО гарантируются международным сертификатом стандарта качества ISO 9000.

Сертификат выдается либо международный, либо на определенную территорию (например, на Восточную Европу).

Разработанная в США методология CMM (CMM =Capability Maturity Model – Модель зрелости.) сертифицирует производителя ПО по 5 уровням.

На основе CMM создана методология PSP, позволяющая резко повысить мастерство программистов. Она включает:

* составление календарного плана работы,
* хронометраж работ,
* рекомендации по недопущению ошибок,
* анализ проекта после завершение работы.

Программист должен не только знать язык программирования, но и уметь планировать свой труд.

***Стиль программирования***

Хороший стиль программирования включает в себя:

* стандартизацию средств
* верные комментарии,
* использование пробелов,
* многоярусную запись текста,
* выбор наглядных идентификаторов,
* упорядочивание списков,
* в строке 1 оператор,
* вынос из циклов постоянных операторов,
* бригадное программирование.

Принцип простоты (KISS)

* Keep делай
* It это
* Simple проще,
* Stupid глупец,

## 1.4 Маркетинг ПО

Различают три вида ПО:

1. Коммерческое

* Издатель заказывает ПО у исполнителя и выделяет средства на работу.
* Издатель получает все имущественные права на созданный продукт.
* Исполнитель может получить некоторый процент (роялти) с каждой проданной копии ПО.
* Издатель оплачивает расходы на упаковку, рекламу, подготовку документации и др.
* За исполнителем сохраняются авторские права на ПО.

1. Условно-бесплатное (Shareware)

* Автор бесплатно представляет клиенту ознакомительную версию ПО.
* Клиент может за определенную плату приобрести полную рабочую версию.

1. Бесплатное ПО (Freeware, Public Domain)

* Автор может попросить заплатить ему некоторую сумму, не настаивая на этом.
* Клиент может с этим не согласиться.

**Краткие итоги**

В лекции были рассмотрены основные понятия, встречающиеся в курсе программирования. Даны определения, что такое алгоритм, машинный язык, программирование. Освещены вопросы инструментария технологий программирования. Дано описания языков высокого и низкого уровня.

**Контрольные вопросы**

1. Дайте определение понятиям «кодинг», «машинный язык», «программа», «алгоритм».
2. Какие команды называют процессорными инструкциями?
3. Из чего состоит процесс программного обеспечения?
4. Что такое системное программное обеспечение?
5. Чем характеризуются пакеты прикладных программ?
6. Дайте определение инструментарию технологий программирования.
7. Для чего предназначены интегрированные системы программирования?
8. Каков перечень показателей качества?
9. Что в себя включает стиль программирования?
10. Перечислите и охарактеризуйте виды ПО.
11. Перечислите языки низкого уровня.
12. Перечислите языки высокого уровня.

# Лекция 2

# Интегрированная среда разработки (ИСР) Visual C#

**Цель лекции:** Ознакомиться с интегрированной средой разработки Visual C#, понять процедуру запуска и работы этой системы. Уяснить что такое консольное приложение, как с ним работать. Изучить структуру папок, процесс компиляции, язык программы, основные алгоритмические структуры.

## 2.1 Основные характеристики.

ИСР Visual C# позволяет создавать, компилировать, тестировать и редактировать проект в единой среде программирования. ИСР входит в комплект языка C#.

Версия ИСР Visual C# Express является бесплатной и скачивается с официального сайта компании Microsoft. После установки требуется ввести код активации. Он отправляется бесплатно на адрес электронной почты, который указывается при активации. Данная операция направлена для сбора статистики использования (популярности) данной среды разработки.

ИСР Visual C# – это комбинация нескольких технологий:

* Высокопроизводительный компилятор C# в машинный код;
* Объектно-ориентированная модель компонент (основные объекты, которые группируются в классы);
* Визуальное построение приложений;
* Мощный отладчик для поиска и устранения ошибок;
* Имеются средства для построения БД. Приложение можно использовать как для локального, так и для клиент - серверного вариантов.

Язык программирования C# предназначен для:

* профессионалов - разработчиков информационных систем;
* пользователей - для быстро решения своих задач.

Запуск Visual C# осуществляется с помощью Главного меню или двойным щелчком по пиктограмме Visual C#.

Главное окно содержит (см. рис.2.1):

Строка заголовка с именем ИСР (Microsoft Visual C#).

Строка главного меню ИСР.

Панели инструментов для быстрого выполнения часто используемых команд

* В центре могут размещаться основные окна (на вкладках, если их несколько):

Начальная страница.

Дизайнеры.

Редакторы кода.

Свойства.

* Слева размещается панель элементов с компонентами.
* Справа размещаются:

Обозреватель решений.

Окно классов.

* Внизу размещается окно «Список ошибок».
* Окно «Начальная страница» содержит вложенные поля:

Создать проект

Открыть проект

Последние проекты.

Последние новости (при подключенном Интернете).

Начало работы.

Приветствие.

Обучение.

Обновление.



Рис.2.1 – Основной экран среды

ИСР Visual C# является одно документной средой, то есть позволяет работать только с одним проектом.

ИСР позволяет создавать решения, включающие набор проектов. Мы будем работать с проектами двух типов:

***Консольное приложение*** (Console Application) – под операционную систему MS DOS с символьным интерфейсом.

Для его создания используется команда (Файл \ Создать проект \ Консольное приложение \ ОК).

Консоль – это монитор и клавиатура. Консольное приложение позволяет использовать маломощные компьютеры, отличается малыми размерами и ограниченными функциональными возможностями.

Приложение (Application) – под операционную систему Windows с графическим интерфейсом.

Для его создания используется команда (Файл \ Создать проект \ Приложение Windows Forms \ ОК).

Приложение позволяет использовать все функциональные возможности операционной системы Windows, требует мощный компьютер, отличается большими размерами исполняемого файла.

Следует отметить, что решение задач в обоих видах приложений с точки зрения алгоритмизации абсолютно одинаково.

***Консольное приложение***

Консольное приложение создается при выборе его в диалоговом окне командой Файл \Создать проект \Консольное приложение \ОК (см. рис.2.2).

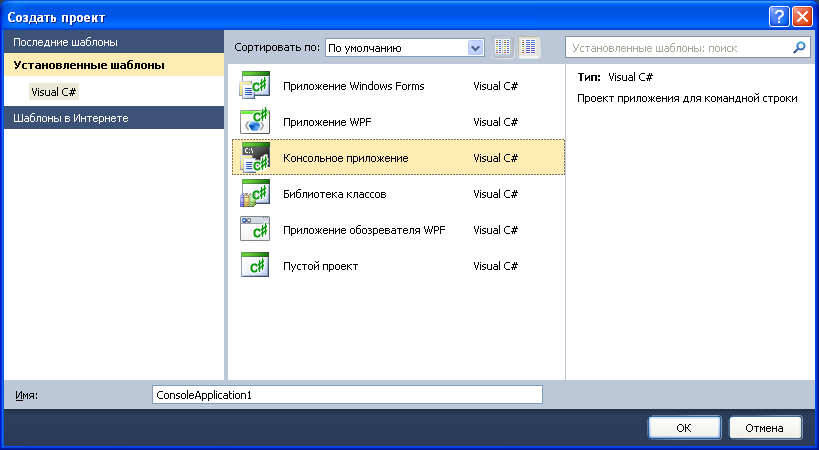


Рис.2.2 – Создать проект «Консольное приложение»

Среда Visual C# формирует в редакторе кода заготовку для проекта (рис.2.3).

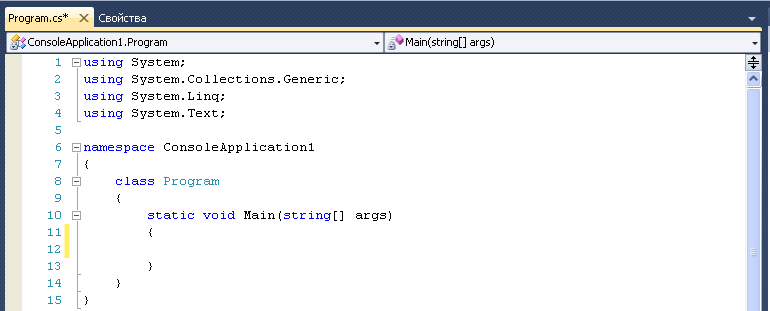


Рис.2.3 – Шаблон кода

***В Окне Редактора кода размещены:***

* Шаблон кода, который формируется ИСР автоматически (эти строки редактированию не подлежат!).
* По умолчанию среда предлагает проекту заголовок ConsolApplication1.Program.
* В строках модули using указаны ссылки на модули, подключаемые к программе.
* В строке 10 начало главного метода(Main). Метод Main — это точка входа приложения C#. Когда приложение запускается, первым вызывается именно метод Main. В программе на C# может существовать только одна точка входа. Ключевое слово void определяет метод. Вы можете создавать дополнительные методы с другими именами например для создания подпрограмм.
* В операторные скобки {..} (строка 12) вводятся разделы объявлений переменных, констант, типов, необходимые для решения задачи, а также код, определяющий выполняемые проектом функции.

***Сохранение проекта***

Выполняется командой Файл \ Сохранить все. В диалоговом окне «Сохранить проект» с помощью кнопки Обзор на дереве папок отыскать нужную папку, открыть ее и вести Имя проекта в одноименное поле Имя, а в поле Имя решения ввести имя папки, где сохраняется проект . Нажать на кнопку «Сохранить». В строке заголовка проекта отобразится имя папки размещения проекта, а в обозревателе решений изменится имя проекта (рис.2.4)

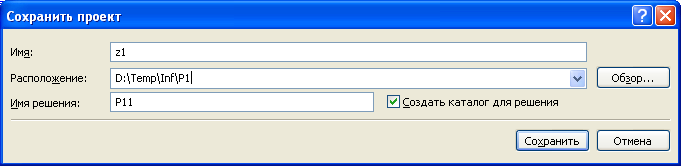


Рис.2.4 – Окно сохранения проекта

Проект сохраняется в структуре папок (таб.2.1, рис.2.5):

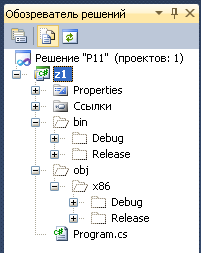


Рис.2.5 – Структура папок

Файлы проекта располагаются в одном каталоге P11.

Для каждого нового проекта целесообразно создавать отдельный каталог (P11).

Исполняемые (Bin) и объектные (Obj) файлы образуются при компиляции (построении) проекта.

Таблица 2.1

Название попок

|  |  |
| --- | --- |
| Решение P11  Z1  Properties  Ссылки  Bin  Debug  Release  Obj  Debug  Release | Папка решения  Папка проекта  Папка свойств  Папка ссылки на модули  Папка двоичных файлов  Папка файлов отладки  Папка файлов выпуска  Папка объектных файлов  Папка файлов отладки  Папка файлов выпуска |

***Компиляция*** – это процесс преобразования исходной программы, написанной на языке высокого уровня (ЯВУ), в исполняемую программу.

Процесс компиляции состоит из двух этапов:

Проверка текста программы на отсутствие синтаксических ошибок (отладка);

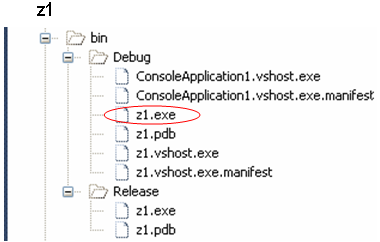
Генерация исполняемой программы (exe-файл).

В результате компиляции ИСР создает исполняемый файл Project.exe. (Z1.exe)

***Возможны два режима отладки***:

Команда Отладка \ Начать отладку создает файлы, размещающиеся в папках Debug (отладка). Построение в режиме отладки, в компонуемые файлы включаются символы отладки, режим оптимизации исключается. Это может увеличить размеры файлов.

Команда Отладка \ Построить решение размещает файлы в папках Release (выпуск). В компонуемые файлы символы отладки не включают, и компилятор использует режим оптимизации кода (например, исключает не использованные переменные). Это может уменьшить размеры файлов.

Компиляция может быть выполнена на любой стадии разработки проекта.

Имя приложения совпадает с именем файла проекта. А само приложение является автономным.

При компиляции проекта создается сборка, сохраняемая на диске как управляемый файл Z1.exe. Из сборки при исполнении генерируется исполняемый бинарный файл, который исполняется на лету.

***Запускать проект можно***:

* из ИСР Visual C# командой начать отладку или кнопкой 
* из среды Windows исполняемого файла \*.exe, как и любое другое приложение, (например, с помощью программы Проводник или папки Мой компьютер) (рис.2.6).

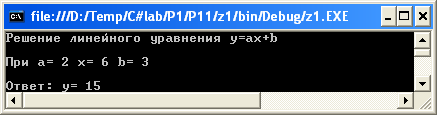


Рис.2.6 –Окно программы после запуска

***Оконное приложение***создается при выборе его в диалоговом окне командой *Файл* \ *Создать проект \*Приложение Windows Forms\*ОК*. (рис. 2.7)

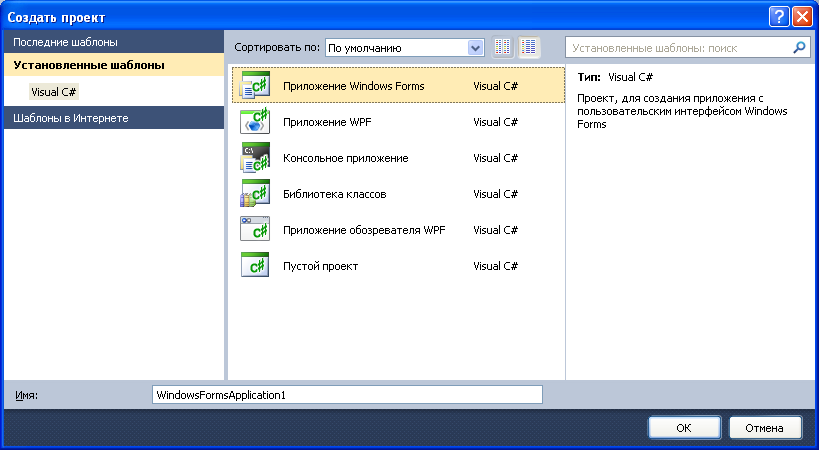


Рис.2.7 – Запуск оконного приложения

ИСР Visual C# создает проект с пустой формой и формирует в редакторе кода заготовку для проекта (рис.2.8).

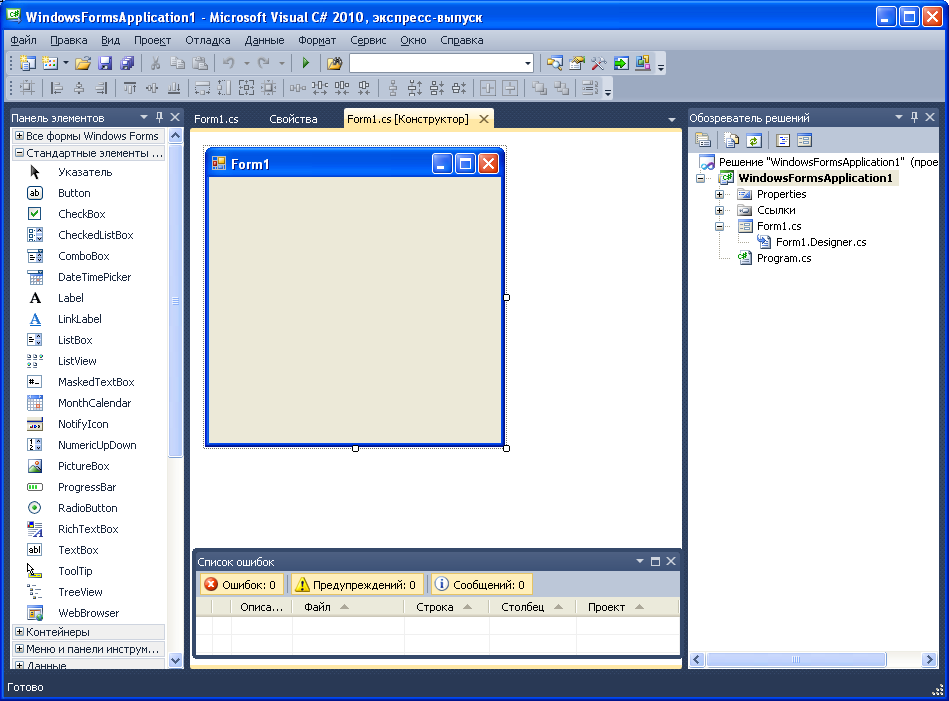


Рис.2.8 – Окно Windows Form

## 2.2 Алфавит языка

Алфавит (или множество литер) языка программирования С# составляют символы таблицы кодов unicod. Алфавит C# включает в себя:

1. Буквы латинского алфавита:

A…Z - 26 заглавных букв,

a...z - 26 строчных букв, (строчные и прописные буквы различаются)

\_ - знак подчеркивания.

2. Строчные и прописные буквы кириллицы.

3. Арабские цифры.

0 . . 9 - 10 цифр.

4. Специальные символы:

- знаки арифметических операций

+ − \* /

- отношения

< >

- знаки пунктуации

. : ; ,

- скобки

( ) [ ] { }

- символы

@ (эт), ' (апостроф), '', $ , #, ^

4. Составные символы.

== - равно,

!= - не равно,

. . - диапазон значений,

<= - меньше или равно,

>= - больше или равно,

Также в C# существует локализация национального алфавита, так в русской версии можно использовать в качестве идентификаторов символы кирилицы.

## 2.3 Словарь языка

Неделимые последовательности знаков образуют слова. Алфавит С# служит для построения слов, которые в C++ называются лексемами.

***Лексемы*** (слова) подразделяются на:

* ключевые (зарезервированные) слова,
* стандартные идентификаторы,
* идентификаторы пользователя,
* знаки (символы) операций;
* литералы;
* разделители.

Почти все типы лексем (кроме ключевых слов и идентификаторов) имеют собственные правила словообразования.

Лексемы обособляются разделителями. Этой же цели служит множество пробельных символов, табуляция, символ новой строки и комментарии.

***Комментарии***

Комментарии – это пояснительный текст, который компилятором игнорируются. В C# определены комментарии:

// - однострочный комментарий и /\*…\*/ - многострочный комментарий, где

/\* - символы начала многострочного комментария;

\*/ - символы конца многострочного комментария.

***Идентификаторы (имена)***

В языке C# используется кодировка unicod. Это означает:

* Чувствительность к регистру, M и m - это разные переменные.
* Допустимо использовать для идентификаторов символы кириллицы.

На имена языка накладываются ограничения:

* Первый символ – обязательно буква.
* Следующие символы – буквы, цифры, символ подчеркивания.

В языке C# для многословных имен можно применять символы подчеркивания или разделять слова, используя в начале слова заглавные буквы.

Возможны два стиля имен:

* Pascal, с заглавной буквы начинается каждое слово идентификатора, включая первое. Например, ЭтоМойИдентификатор.
* camel, первая буква строчная, но имя может содержать заглавные буквы (визуально это горбы, camel = верблюд). Например, этоМойИдентификатор.

***Ключевые слова***

Ключевые слова — это предварительно определенные зарезервированные идентификаторы, имеющие специальные значения для компилятора, смысл которых фиксирован в языке.

Ключевые слова используются для инструкций (команд) C#.

В ИСР предусмотрены встроенные средства контроля правильности идентификаторов. Среда не примет неправильный идентификатор, и вы получите сообщение о допущенной ошибке.

Примеры ключевых слов: and, break, byte, case, class, const, do, double, else, false, float, for, goto, if, int, interface, object, out, return, short, string, switch, true, uint, using, while…

Стандартные идентификаторы – это имена элементов языка (типов, констант, процедур, функций).

Например: Sin, Pi, Double, Cos

Идентификаторы пользователя – применяются для обозначения имен констант, переменных, процедур, функций и типов данных.

Требования к идентификаторам языка C#:

* могут содержать прописные и строчные латинские и русские буквы, арабские цифры;
* не могут начинаться с цифр.

Таблица 2.1

Идентификаторы и их задание

|  |
| --- |
| Правильные идентификаторы  MyProgram МояПрограмма  moyaProgramma x\_1  moyaPervayaProgramma x2 |
| Неправильные идентификаторы  My program 1x  12MyProgram x-1  Мой Проект x 5 |

***Числа***

Десятичные числа:

1 1234 Целые без знака

-1 +1234 Целые со знаком

23.45 Вещественные без знака

-23.45 +23.45 Вещественные со знаком

***Переменные и константы***

Переменная – это синтаксическая единица, значение которой может изменяться, однако имя остается прежним. Представляет числовое или строковое значение или объект класса. Значение, хранящееся в переменной, может измениться.

В C# переменные объявляются с определенным типом данных и именем. Тип указывает точный объем памяти, который следует выделить для хранения значения при выполнении приложения. Переменная может быть объявлена в любом месте кода. При объявлении нужно указать тип переменной, задавать ее значение не обязательно.

Формат объявления переменной:

тип имя\_переменной;

doubley, z; // вещественные переменные y, z

Формат объявления переменной с инициализацией (со значением):

тип имя\_переменной=значение;

Примеры объявления переменных с присвоения ей значения (**инициализацией**).

int x = 1; // целочисленная переменная x получает значение 1

strings = "group"; // строковая переменная s получает значение group

char g = 'x'; // символьная переменная g получает значение x

***Константа*** является другим типом. Она хранит значение, присваиваемое по завершении компиляции программы, и никогда после этого не изменяется. Константы объявляются с помощью ключевого слова **const,** их использование способствует повышению удобочитаемости кода.

const int speedLimit = 55; // целочисленная константа speedLimit получает значение 55

const double pi = 3.14159265358979323846264338327950; //вещественная константа pi (π)

## 2.4 Структура программы консольного приложения на C#

Для упорядочения и оформления кода в языке C# используются классы. В действительности весь выполняемый код C# должен содержаться в классе, что справедливо и для короткой программы типа "Hello Student!". Ниже приведен код программы, отображающей в окне консоли сообщение "Hello Student!".

// A Hello Student! program in C#

Пространства имен определяют для классов уникальные полные имена. Пространство имен позволяет хранить одно множество имен отдельно от другого. Имена пространств не будут конфликтовать друг с другом. Ключевое слово using – это заявление о том, что программа использует имена в заданном пространстве имен.

Например, пространство имен System содержит класс Console, который включает методы для чтения и записи в окне консоли.

Для использования метода WriteLine, определенного в классе Console, который содержится в пространстве имен System, без предварительного определения пространства имен следует использовать строку кода

System.Console.WriteLine("Hello, Student!");

Если заранее объявить директиву using System, то она предполагает пространство имен System и впоследствии можно написать Console.WriteLine("Hello, Student!");

Листинг 2.1 Программа в консольном приложении

using System; //**Пространство имен**

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1 //Имя встроенного класса Console

{

class Program // **Объявление нового класса** Program

{ // Определение этого класса между {…}

static void Main(string[] args) // **Вызываемый метод Main**

{ // **Объявление переменных, констант**…

Console.WriteLine("Hello Student!"); // **Раздел исполнения**

Console.WriteLine("Нажмите любую клавишу");// **Операторы**

Console.ReadKey();

}

}

}

Любая C# программа начинается с вызова метода Main.

Метод Main является точкой входа консольного приложения C# или приложения Windows. При запуске приложения метод Main является первым вызываемым методом.

В программе C# возможна только одна точка входа.

class Program

{

static void Main(string[] args)

{

// Строки команд

}

}

* static – ключевое слово, определяет способ выделения памяти под экземпляр.
* void – ключевое слово, определяет, что метод не возвращает значений.
* (string[ ] args) – аргументы, передаваемые программе. Если программе передаются аргументы, то они передаются в виде массива с указанием типа и имени. Например, string[ ] – тип массива строк, arg – имя этого массива. Альтернативный вариант - простое перечисление пар (тип - значение) с разделением запятыми.

Если программе не передаются аргументы, то метод просто Main().

## 2.5 Базовые алгоритмические структуры

Алгоритм любой сложности может быть представлен комбинацией трёх базовых структур:

* следование;
* ветвление;
* повторение (цикл).

Структура **"следование"** означает, что несколько операторов должны быть выполнены последовательно друг за другом и только один раз за время выполнения данной программы.

Оператор

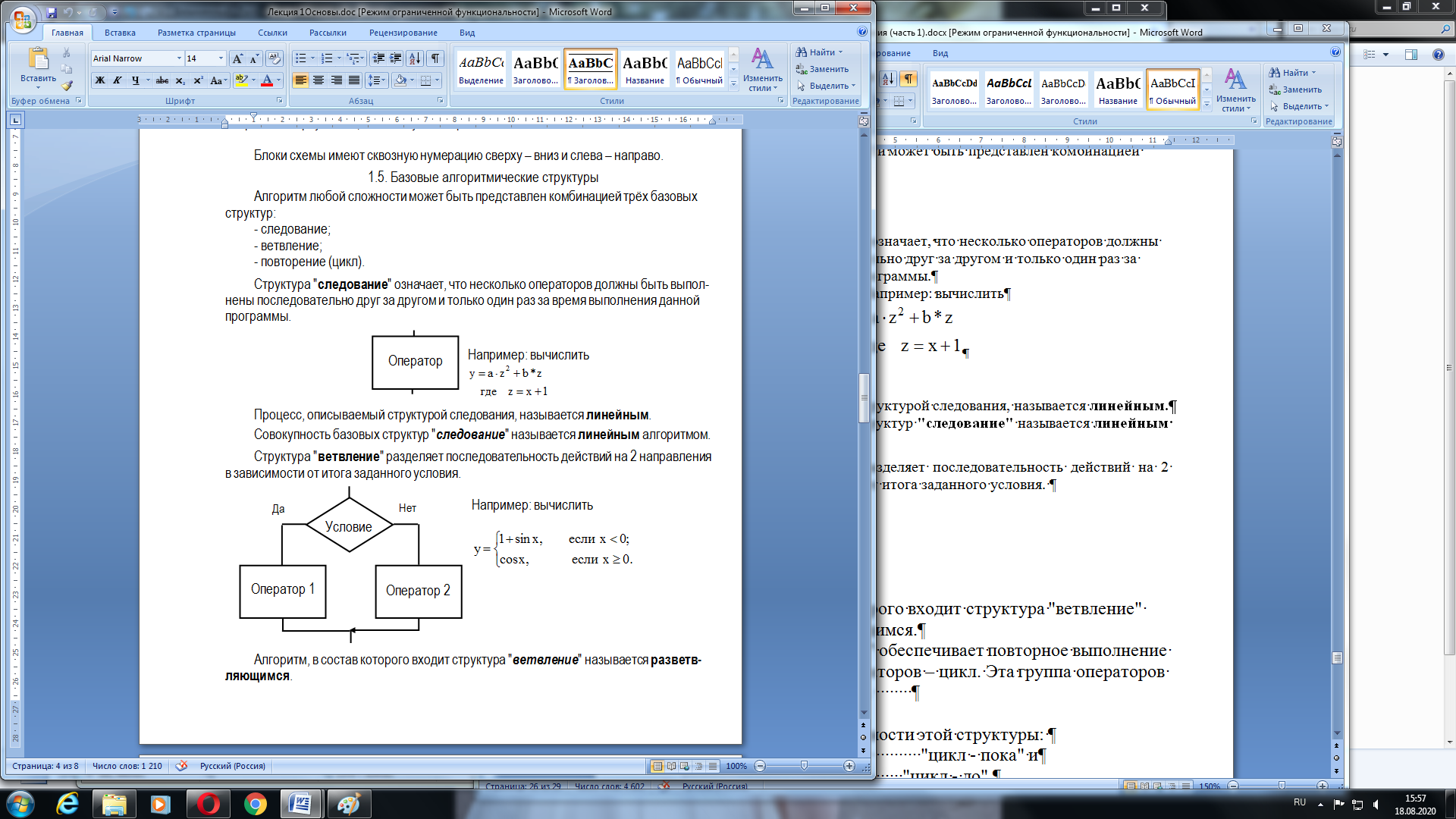
Например: вычислить



Процесс, описываемый структурой следования, называется **линейным.**

Совокупность базовых структур **"следование"** называется **линейным алгоритмом.**

Структура **"ветвление"** разделяет последовательность действий на 2 направления в зависимости от итога заданного условия.



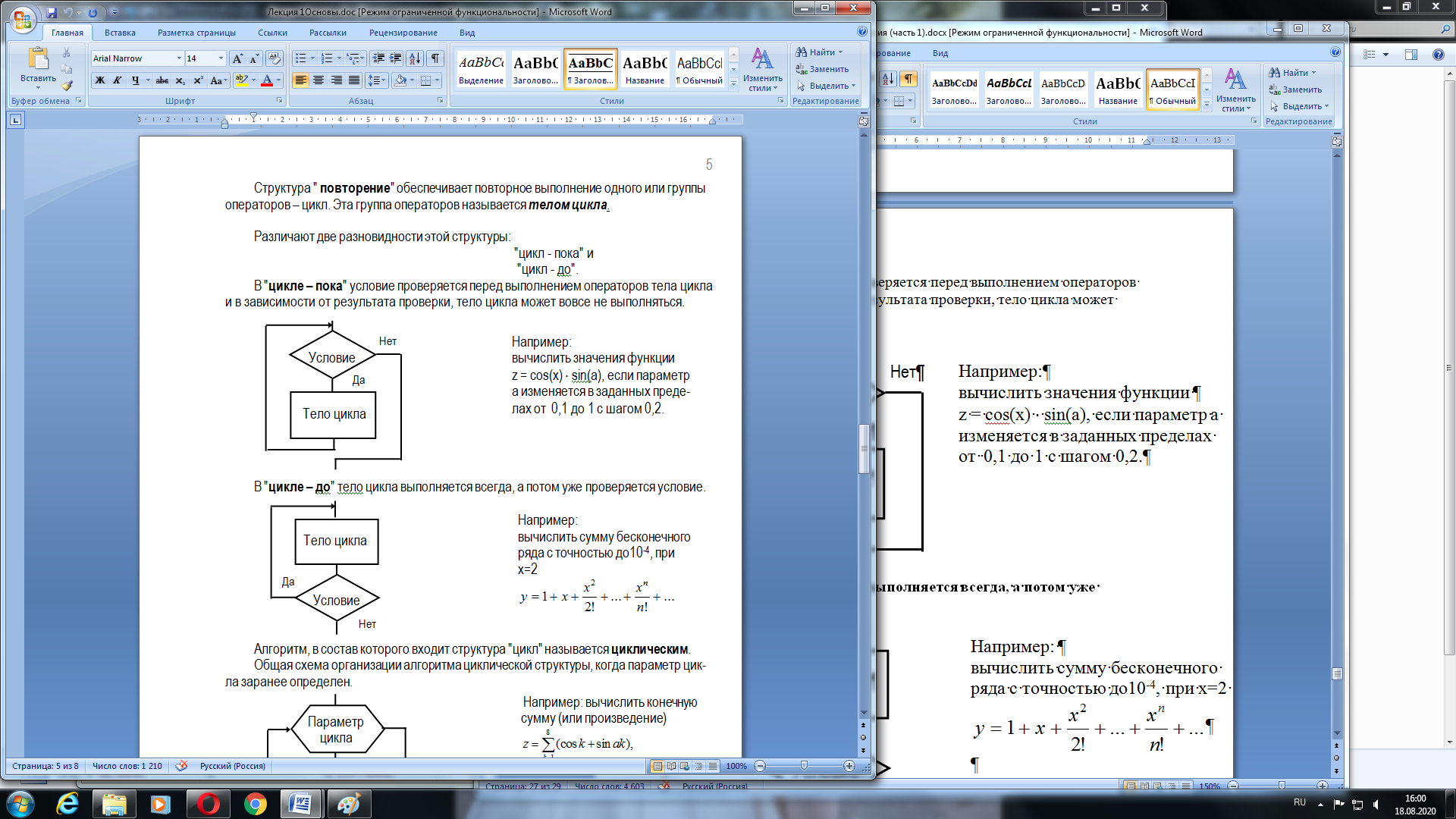
Алгоритм, в состав которого входит структура **"ветвление"** называется **разветвляющимся.**

Структура **" повторение"** обеспечивает повторное выполнение одного или группы операторов – цикл. Эта группа операторов называется телом цикла. Данная структура используется, когда требуется выполнить одну и ту же группу одних и тех же операторов несколько раз.

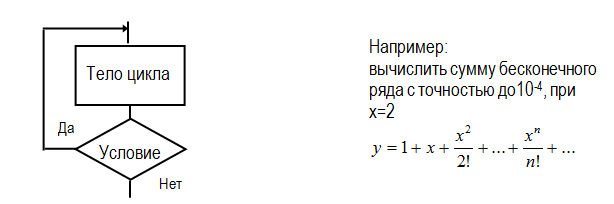
Различают две разновидности этой структуры:

* "цикл - пока"
* "цикл - до".

В **"цикле – пока**" условие необходимости осуществления действий проверяется перед выполнением операторов тела цикла и в зависимости от результата проверки, тело цикла может выполняться или не выполниться ни разу. То есть **пока** результат проверки условия «истина» тело цикла выполняется.

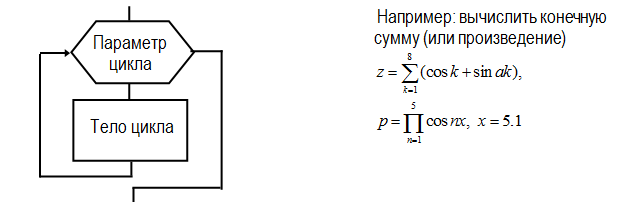


В **"цикле – до"** тело цикла выполняется хотябы один раз, так как условие необходимости повторного выполнения проверяется проверяется после. То есть **до тех пор пока** результат проверки условия «истина» тело цикла выполняется.



Алгоритм, в состав которого входит структура **"цикл"** называется **циклическим.**

Общая схема организации алгоритма циклической структуры, когда параметр цикла заранее определен.



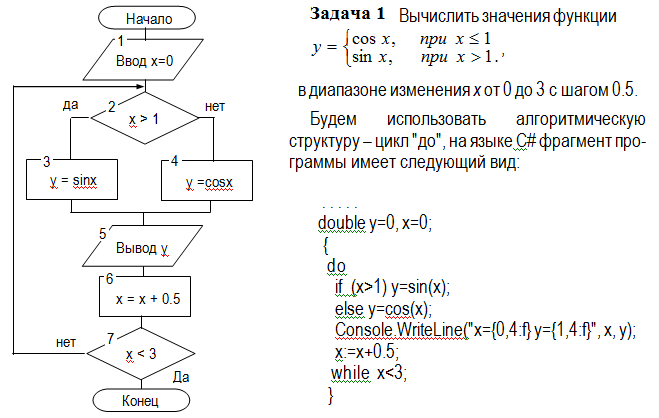
Параметр цикла – это переменная (индекс), которая изменяется (с шагом) при каждом новом выходе на повторение.

Циклы могут содержать внутри себя другие циклы. Такие структуры называются вложенными. Например, вычисление конечных сумм и произведений 

## 2.6 Методы алгоритмизации задач

Реальные алгоритмы представляют собой совокупность всех рассмотренных базовых структур. Наиболее часто употребляются:

* линейные вычисления,
* ветвления,
* выбор из большого количества альтернатив одного,
* циклы, (вложенные циклы),
* подпрограммы.



**Краткие итоги.** В лекции были рассмотрены следующие вопросы: ИСР Visual C# ее характеристики, работа консольного приложения, алфавит языка C#, базовые алгоритмические структуры и методы алгоритмизации задач.

**Контрольные вопросы**

1. Консольное приложение его работа и назначение.
2. Опишите процесс компиляции.
3. Алфавит языка C# его составляющие.
4. Дайте определение «лексемам», «идентификаторам», «ключевым словам», «переменные».
5. Опишите структуру программы консольного приложения.
6. Перечислимте базовые алгоритмические структуры.

# Лекция 3

# Типы данных

**Цель лекции:** Ознакомиться с типами данных применяемых в программах на языке C#. Уяснить, как применяются типы данных и для чего они нужны. Разобрать вопрос задания типов данных.

Язык C# является типизированным языком. Каждая переменная и константа имеет тип, как и выражение, результатом вычисления которого является значение.

***Тип данных*** определяет множество значений, которые могут принимать элементы программы. Введение типа дало возможность автоматического поиска ошибок при компиляции, что приводит к созданию надежных программ.

Тип определяет для элемента программы:

* объем памяти для размещения. По типу компилятор определяет размер памяти для размещения значений элемента,
* место для хранения переменной типа,
* минимальное и максимальное значения, которые могут принимать данные,
* разрешенные операции.
* Типы могут быть:
* стандартные (не требуют объявления),
* пользователя (тип пользователя должен быть объявлен дополнительно).

Типы Т1 и Т2 идентичны, если:

* имеют один и тот же идентификатор типа,
* один объявлен эквивалентным другому (один из них может быть преобразован к другому). Например, целый в вещественный, символьный в строковый.

Система типов языка C# включает следующие категории:

* Типы-значения // в переменных типа значения хранятся данные
* Ссылочные типы // в переменных хранятся ссылки на фактические данные
* Типы указателей // в переменных хранятся адреса ячеек памяти из диапазона значений и специального значения — нулевого адреса.

Существует возможность преобразовать тип значения в ссылочный тип и обратно в тип значения с помощью упаковки-преобразования и распаковки-преобразования.

## 3.1. Типы значений

Переменные, основанные на типах значений, содержат непосредственно значения. При присвоении переменной одного типа значений другому создается копия присваиваемого значения. В этом заключается отличие от переменных ссылочного типа, при присвоении которых копируются ссылки на объекты, но не сами объекты.

Все типы значений являются неявными производными от System.ValueType .

Типы значений состоят из двух основных категорий:

* Структура **struck**. Это тип значения, который используется для объявления небольших групп связанных переменных, например координат точки.
* Перечисление **enum**. Оно состоит из набора именованных констант, который называется списком перечислителя.

По умолчанию первому элементу задан номер 0, а последующие – увеличиваются на 1. Например: enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};

Структуры делятся на следующие категории:

* ***Числовые*** типы. Они могут быть целочисленные, вещественные – с плавающей запятой, десятичные.
* ***Логический*** bool. Ключевое слово bool используется при объявлении переменных для хранения логических значений, true и false.
* Пользовательские структуры.

## 3.2. Ссылочные типы

Переменные ссылочных типов, называемые объектами, они сохраняют ссылки на фактические данные. Для объявления ссылочных типов используют ключевые слова:

* Класс **class.**
* Интерфейс **interface.**
* Делегат **delegate**.
* Динамический **dynamic.**
* Объект **object.**
* Строка **string**.

## 

## 3.3 Классы

Класс – совокупность объектов с одинаковыми свойствами. Определив класс, его можно использовать, загрузив в память. Класс, загруженный в память, называется объектом или экземпляром класса. Экземпляр класса создается с помощью ключевого слова **new**. ***Каждый экземпляр класса занимает отдельную область памяти***.

Подобно тому, как на основе одного чертежа можно построить несколько зданий, можно создать любое количество объектов одного класса. Часто используют массивы или списки, содержащие множество объектов одного класса с независимыми значениями.

Классы объявляются с помощью ключевого слова **class**. Класс может содержать:

* Методы; // подпрограммы для обработки данных объекта какого-то класса
* Свойства; // характеристики объекта, его параметры
* Поля; // предназначены для хранения информации об объекте
* События; // сигналы, формируемые пользователем, ОС, программой
* Делегаты; // способ передачи метода
* Вложенные классы.

***Тип object***

Переменным типа **object** можно назначать значения любых типов. Когда переменная типа значения преобразуется в объект, говорят, что она упаковывается. Когда переменная типа object преобразуется в тип значения, говорят, что она распаковывается.

***Тип string***

Тип string представляет последовательность из нуля или более символов в кодировке Юникод. Строка C# представляет собой группу одного или нескольких символов, заключенных в **двойные кавычки** и объявленных с помощью ключевого слова string,

string Приветствие = "Hello, Friend!";

Строковые объекты являются неизменяемыми, после создания их нельзя изменить.

## 3.4. Задание типов в объявлениях переменных

При объявлении переменной или константы в программе необходимо задать ее тип. Синтаксис объявления:

тип Имя = Значение;

Примеры объявления некоторых переменных:

float Вес = 400.2; // переменная Вес типа float (с плавающей зпт.)

string Имя = ''ЭтоЯ'' ; // переменная Имя типа string

char Буква = 'A'; // переменная Буква символ

int[ ] Массив = { 0, 1, 2, 3, 4, 5 }; // переменная Массив целых чисел

После объявления переменной она не может быть повторно объявлена с новым типом, и ей нельзя присвоить значение, несовместимое с ее объявленным типом.

Например, нельзя объявить переменную типа int и затем присвоить ей логическое значение **true**.

Однако значения могут быть преобразованы в другие типы, например, при их присвоении новым переменным или при передаче в качестве аргументов метода.

Например, число -120 – (тип sbyte) в число -00000000120 – (тип short).

Преобразование типов, которое ***не приводит*** к потере данных и автоматически выполняется компилятором, называется **неявным**.

Для преобразования, которое может привести к ***потере*** данных, необходимо в исходном коде указать тип преобразования. Такое преобразование называется **явным**. Оно может привести к ошибке.

Например, число 0.81245826**13425765** (тип double) в число 0.81245826 (тип float).

C# имеет встроенные типы для представления значений целых чисел, вещественных чисел, логических выражений, текстовых символов, десятичных значений и других данных.

В приложение Б приведены таблицы с перечислением базовых типов и их краткое описание, типы целых чисел, типы чисел с плавающей запятой и значение типов по умолчанию.

## 3.5. Преобразования типов

Все вычисления происходят с использованием типа double. Другие типы чисел могут применяться для уменьшения занимаемой памяти. При их использовании перед вычислением они преобразуются в тип double. Различают преобразования:

* **Неявные** преобразования используются для совместимых типов. Значения источника полностью отображаются приемником, т.к. тип приемника больше, чем тип источника. Например, преобразование int в double (int – подмножество double) выполняется автоматически, его не надо заказывать.
* **Явные** преобразования используются для несовместимых типов. Например, при преобразовании double в int (int – подмножество double) значения источника не полностью отображаются приемником.

Если типы несовместимы, и ошибка допустима, то преобразование возможно, но его нужно при объявлении явно заказать: перед преобразуемым выражением в круглых скобках добавляется префикс идентификации конечного типа.

Например,

double db=12.94;

int i = (int) db;

### 

## 3.6. Стандартное форматирование чисел

Все числа выводятся в консоль в виде строк символов. Потому перед выводом применяется форматирование результата.

Все числовые типы поддерживают определенный набор описателей формата. Для чисел поддерживаются национальные стандарты. Например, для России разделитель целой и дробной части числа − запятая.

Метод вывода в консоль содержит в двойных кавычках строку вывода, в состав которой входят выводимые символы и описатели форматов вывода не строковых данных, вставляемые в строку в нужных местах. После строки через запятые перечисляются имена выводимых данных. Количество описателей формата и не строковых данных одинаково, **нумерация данных начинается с нуля**.

Описатель формата помещается в фигурные скобки и имеет структуру:

"{<Номер позиции вывода>,<Число позиций>:<Буква>d}"

**Пример 1.** Если указывается только номер позиции вывода, то выводимые не строковые данные форматируются по умолчанию. Строка кода:

Console.WriteLine("p={0}",p); даст результат → **p=2,8322092165912**

**Пример 2.** Выводятся значения x, y (номера позиции вывода 0 и 1), формат по умолчанию.

Console.WriteLine("x = {0} y = {1}", x, y); → **x=-2,1548 y=15,125**

**Пример 3**. Выводится значение x (номер позиции вывода 0), под число отводится 8 позиций, используется денежный формат (буква C).

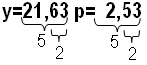
Console.WriteLine("x = {0,8:C}", x); →

**Пример 4**. Выводится значение y (номер позиции вывода 0), используется общий формат (буква G – реально полученное при вычислении количество символов).

Console.WriteLine("y = {0:G}", y); → **y=65,1253**

**Пример 5**. Выводится значение z (номер позиции вывода 0), используется экспоненциальный (научный) формат (буква E – 6 символов после запятой).

Console.WriteLine("z = {0:E}", z); →

**Пример 6.** Выводятся значения x, y (номера позиции вывода 0 и 1), под числа отводится 5 позиций, используется фиксированный формат (буква F – 2 символа после запятой).

Console.WriteLine("y={0,5:f} p={1,5:f}", y, p); →

**Пример 7.** Выводится значение s (номер позиции вывода 0) по шаблону. В отображении числа отводится 1 позиция для целой и 3 позиции для дробной части числа.

Console.WriteLine("s= {0**:**#**.**###}", s); **→ s= 7,825**

Имеется набор стандартных форматов. Синтаксис записи формата:

<Буква описания формата>d.

Описатель формата - это алфавитный символ, определяющий строковое представление объекта, к которому он применяется. Также строка формата может содержать необязательный описатель точности d, определяющий, сколько цифр отображается в результирующей строке. Если спецификаторы пропускаются, то используются их значения по умолчанию. Предусмотрены следующие форматы, представленные в таблице 3.1:

Таблица 3.1

Форматы чисел

|  |  |  |  |
| --- | --- | --- | --- |
| Буква (загл/стр) | Формат | Примеры | d по  умолч. |
| G g | Общий | -123.456 -> -123.456 | Факт |
| F f | Фиксированная запятая | 1234.569 -> 1234.57 | 2 |
| N  n | Число | 1234.569 -> 1,234.57  Запятая разделяет группы | 2 |
| E  e | Экспоненциальный  (научный) | 1052.0329112756 -> 1.052033E+003 | 6 |
| D d | Десятичный | **-**1234 ("D6") -> **-**001234 | Мин. |
| C  c | Валюта | 123.458 -> $123.46  Используются символы валют | 2 |
| P p | Проценты | 1 -> 100.00 % | 2 |

Описание форматов:

* Формат G – общий. Поддерживается всеми числовыми типами данных. Но результат наиболее компактен при использовании экспоненциального формата или формат с фиксированной запятой. Описатель точности - количество значащих цифр.
* Формат F - фиксированная запятая. Результат - цифры целой и дробной частей с необязательным отрицательным знаком. Поддерживается всеми числовыми типами данных. Описатель точности - количество цифр дробной части (по умолчанию 2).
* Формат N – число. Результат - цифры целой и дробной частей, разделители групп и разделитель целой и дробной частей с необязательным отрицательным знаком. Поддерживается всеми числовыми типами данных. Описатель точности - желаемое число знаков дробной части.
* Формат E – экспоненциальный (научный). Результат – экспоненциальное отображение. Поддерживается всеми числовыми типами данных. Описатель точности - количество цифр дробной части. Описатель точности по умолчанию 6.
* Формат D – десятичный. Результат - целочисленные цифры с необязательным отрицательным знаком. Поддерживается только целочисленными типами данных. Описатель точности - минимальное число цифр. Описатель точности по умолчанию - минимальное необходимое число цифр.
* Формат C – валюта. Результат – значение валюты. Если есть символ, то он выводится. Поддерживается всеми числовыми типами данных. Описатель точности - количество цифр дробной части.
* Формат P – проценты. Результат – число, умноженное на 100 и отображаемое с символом процента. Поддерживается всеми числовыми типами данных. Описатель точности - желаемое число знаков дробной части.

***Символы char***

Тип Char – это один символ в коде unicod. Символ помещается в **апострофы** (**одиночные кавычки)**. Синтаксис объявления:

char Имя = ' Символ ';

char S = ' z ';

## 3.7. Инструкции, команды, выражения

Код приложений в C# состоит из инструкций (команд) с ключевыми словами и выражениями с операторами.

В инструкцию включают объявление переменных, присвоение значений, вызов методов, ветвление на один или другой блок кода, в зависимости от заданного условия. Порядок выполнения инструкций в программе называется потоком выполнения.

**Инструкция** - строка кода, которая **заканчивается точкой с запятой**.

**Блочная инструкция -** набор инструкций **в фигурных скобках { }**. Может содержать вложенные блоки **{ {…} }**.

**Выражение** — это строка кода, которая определяет способ вычисления некоторого значения. Выражение состоит из:

* Операндов, // данные, над которыми выполняются действия
* Знаков операций,
* Круглых скобок // используются для изменения приоритетности операций.

Выражение включают в состав инструкции. Пример с простым выражением:

х = 100; // Данная инструкция выполняет присвоение значения 100 переменной х.

Инструкцией такого вида можно инициализировать любое число переменных с одним и тем же значением, например 20:

a = b = c = d = e = 20;

В объявлении константы необходимо задать значение.

* double Площадь = 0; // Переменная с инициализацией
* double Радиус = 2;
* const double pi = 3.14159; // Константа pi с конечным числом символов.

Инструкция с выражением вычисляет значение выражения, которое сохраняется в указанной переменной.

Площадь = pi \* (Радиус \* Радиус);

***Разделители***

В языке С# как разделители рассматриваются:

* пробелы,
* знаки табуляции,
* переход на новую строку.

В инструкциях языка С# лишние разделители компилятором игнорируются.

Можно написать:

myValue=100; или myValue  =  100;

Компилятор обработает эти две инструкции как абсолютно идентичные. Исключение состоит в том, что пробелы в пределах строки не игнорируются. В строке кода:

Console.WriteLine("Я изучаю C# !");

// каждый пробел между словами «Я», «изучаю», «C#» и знаком «!» будет обрабатываться, как отдельный символ строки.

Разделители применяются для того, чтобы сделать программу более удобной для программиста, для компилятора разделители абсолютно безразличны.

Разделители в тексте программы позволяют компилятору находить и анализировать ключевые слова языка.

***Математические операции***

Для выполнения сложных математических операций, например в тригонометрии, используется класс платформ Math. В приложении В приведены таблицы Класс System.Math поля и методы.

Примеры использования методов Sin (вычисление синуса) и Sqrt (вычисление квадратного корня) и константа PI (системная константа с большим количеством знаков), Pow(возведения в степень), Exp(вычисления экспоненты)

using System;



double d = Math.Sin(Math.PI/2);



double q = Math.Sqrt(144);



double y = (Math.Exp(x)+1)/( Math.Pow(p,1/3) - Math.Abs(Math.Pow(s,3)-1));



double t = Math.Pow(2,x)\* Math.Cos(b\*y) -

Math.Pow(3,y)\*Math.Pow(Math.Sin(b\*x),2);



double n = b\*t+a;



double s = Math.Exp(-b\*t)\* Math.Sin(n)-

Math.Sqrt(Math.Abs(n));

***Приоритеты операций***

1. Вычисления в ( ) // Сначала вычисления в самых внутренних ()

2. ! унарные + / - // Отрицание операнда и добавление знака +/-

3. \* / // Операции типа умножения \* /

4. + − // Операции типа сложения + -

5. < > <= >= // Операции типа сравнения, отношения

6. == != // Операции типа сравнения, эквивалентность

7. & // Операция лог. умножения И

8. | // Операции лог. сложения ИЛИ

**Краткие итоги.** В лекции были рассмотрены типы данных. Как они задаются, какие бывают типы данных. Изучена система типов язык С#, преобразование типов данных из одного в другой и вопросы форматирования.

**Контрольные вопросы**

1. Что определяет тип данных?
2. Что включает в себя система типов данных языка C#?
3. Из чего состоят типы значений?
4. Перечислите ссылочные типы и как они задаются.
5. Приведите примеры задание переменных, что необходимо при этом знать.
6. Какие преобразования типов вы знаете?
7. Чем отличается стандартное форматирование от не стандартного?
8. Дайте определение что такое «выражение».
9. Для чего применяются разделители?
10. Что необходимо знать при выполнении математических операций?
11. Приведите примеры вычисления математических операций.
12. Как строятся приоритеты операций?

# Лекция 4

# Операторы

**Цель лекции:** Ознакомиться с понятием «операторы», как они применяются в программах на языке C#. Уяснить, для чего нужны операторы.

**Оператор –** этозапись инструкций по выполнению некоторой последовательности действий. Операторы используются для выполнения вычислений, присвоения значений, проверки на равенство и неравенство и т. д.

В языке C# имеется большой набор операторов. Они представляют собой символы, определяющие операции, которые необходимо выполнить с выражением.

Операторы в выражениях исполняются с приоритетами:

* высший приоритет имеют основные операторы,
* далее мультипликативные (типа умножения),
* затем аддитивные (типа сложения) и
* далее отношения (типа сравнения).

## 4.1 Основные операторы

Таблица 4.1

Основные операторы

|  |  |
| --- | --- |
| Оператор | Действие |
| x.y | Оператор "точка" используется для доступа к членам класса. Формат - Класс.Член. |
| (x) | Круглые скобки (…) используются для указания порядка выполнения операций в выражении. Наивысший приоритет – операции в самых внутренних скобках. |
| A[x] | Квадратные скобки [ ]. Используются для доступа к элементу массива, его индекс необходимо заключить в скобки. Для одномерных массивов A[1]. Для многомерных массивов индексы разделяются запятыми B[1,2]. |
| new | Используется для создания экземпляра класса.  Class1 Имя = new Class1(); |
| Typeof(имя типа) | Используется для получения типа объекта.  System.Type type = typeof (имя типа); |

**Пример 1:**

private void button1\_Click(object sender, EventArgs e)

{

button1.Visible = false;

label1.Text = "Студент";

textBox1.Text = "Группа ПИЭ-01";

}

**Пример 2:**



double x, a, z;

a = 0.6; x = 0.5;

z = (Math.Pow(Math.Sin(x + 5), 2) - Math.Abs(x)) /

(Math.Exp(-a \* x) + Math.Pow(x,3));

Console.WriteLine("z={0:E}", z);

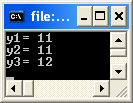
Console.ReadKey();

## 4.2 Унарные операции

Таблица 4.2

Унарные операции

|  |  |
| --- | --- |
| Оператор | Действие |
| + X | Унарный плюс. Это знак числа X. Он использован по умолчанию. |
| ++X | Префиксная форма инкремента. Увеличение X на 1 перед использованием. |
| X ++ | Постфиксная форма инкремента. Увеличение X на 1 после использования. |
| - X | Унарный минус. Это знак числа X. |
| -- X | Префиксная форма декремента. Уменьшение X на 1 перед использованием. |
| X -- | Постфиксная форма декремента. Уменьшение X на 1 после использования. |
| ! X | Логическое отрицание. Унарный оператор, который выполняет над операндом X операцию НЕ. Он задан для типа bool и меняет значение операнда **true** на **false**, или наоборот. |
| ~X | Поразрядное дополнение. Инвертирование каждого бита целого X. |
| (T) x | Явное преобразование x в тип T |

**Пример 3:**

int x=10, y1, y2, y3;

y1 = ++x; //x=11

y2 = x++; //x=11

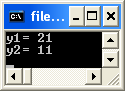
y3 = x; //x=12

Console.WriteLine("y1= " + y1);

Console.WriteLine("y2= " + y2);

Console.WriteLine("y3= " + y3);

В обоих случаях переменная х получит значение 11. Разница состоит лишь в том, в какой момент она станет равной 11 (до присвоения ее значения переменной y или после).

**Пример 4:**

int x=10, y1, y2;

y1 =x + ++x;

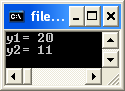
y2 = x;

Console.WriteLine("y1= " + y1);

Console.WriteLine("y2= " + y2);

Console.ReadKey();

Инструкция x + ++x выполняется следующим образом: сначала запоминается исходное значение х=10, затем инкрементируется х (х=10+1=11). Далее суммируются новое значение x с исходным (11+10=21), а результат суммирования присваивается y.

**Пример 5:**

int x = 10, y1, y2;

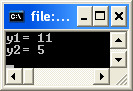
y1 = x + x++;

y2 = x;

Console.WriteLine("y1= " + y1);

Console.WriteLine("y2= " + y2);

Console.ReadKey();

Инструкция x + x++ выполняется следующим образом: сначала суммируются х+х (10+10=20), результат суммирования присваивается y, затем инкрементируется переменная х (10+1=11) для дальнейшего использования в программе.

**Пример 6**

int x=6, y1, y2;

y1 =x + --x;

y2 = x;

Console.WriteLine("y1= " + y1);

Console.WriteLine("y2= " + y2);

## 4.3 Мультипликативные операторы

Таблица 4.3

Мультипликативные операторы

|  |  |
| --- | --- |
| Оператор | Действие |
| X \* Y | Умножение. Вычисляет произведение двух операндов. |
| X / Y | Деление. Делит X на Y. При делении целых чисел результат всегда является целочисленным. Остаток отбрасывается. |
| X % Y | Остаток. Вычисляет остаток после деления X на Y. |
| X & Y | Логическое умножение (И - AND). Вычисляет X и Y независимо от X. |

**Пример7:**

int ires, ioct;

double dres, doct;

ires = 10 / 3;

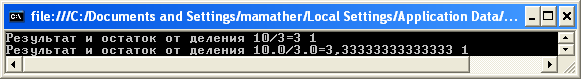
ioct = 10 % 3;

dres = 10.0 / 3.0;

doct = 10.0 % 3.0;

Console.WriteLine("Результат и остаток от деления 10/3=" + ires + " " + ioct);

Console.WriteLine("Результат и остаток от деления 10.0/3.0=" + dres + " " + doct);



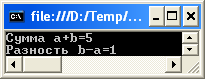
## 4.4 Аддитивные операторы

Таблица 4.4

Аддитивные операторы (типа сложения)

|  |  |
| --- | --- |
| Оператор | Действие |
| X + Y | Сложение. Для числовых типов он вычисляет сумму X + Y.  Для строкового типа он объединяет X и Y. |
| X − Y | Вычитание. Для числовых типов вычисляет разность X - Y. |
| X | Y | Логическое сложение (ИЛИ - OR). Вычисляет X и Y независимо от значения X. |
| X ^ Y | Сложение по модулю 2 (Исключающее ИЛИ - XOR).  Вычисляет X и Y независимо от значения X. |

**Пример 8:**

 int a, b, z, t;

a = 2; b=3;

z=a+b;

Console.WriteLine("Сумма a+b="+ z );

t=b-a;

Console.WriteLine("Разность b-a="+ t );

## 4.5 Операторы отношений

Результат выполнения операторов отношений и логических операторов имеет тип bool.

В C# на равенство (==) и не равенство (!=) можно сравнивать все объекты ( x==’a’ ).

Но операторы сравнения (<, >, <=, >=) можно применять только к типам, которые поддерживают только отношение упорядочения, т.е к числовым типам.

Например, x>2, z==0, Math.Sin(x)<=5.

Таблица 4.5

Операторы отношений

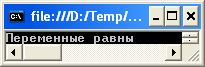
|  |  |
| --- | --- |
| Оператор | Действие |
| X == Y | Равно. Возвращает значение true, если X == Y, в противном случае возвращается значение false. |
| X != Y | Не равно. Возвращает значение true, если X != Y, в противном случае возвращается значение false. |
| X < Y | Меньше. Возвращает значение true, если X < Y, в противном случае возвращается значение false. |
| X > Y | Больше. Возвращает значение true, если X > Y, в противном случае возвращается значение false. |
| X <= Y | Меньше или равно. Возвращает значение true, если X <= Y, в противном случае возвращается значение false. |
| X >= Y | Больше или равно. Возвращает значение true, если X >= Y , в противном случае возвращается значение false. |

**Пример 9:**

int a, b;

a = 3; b=3;

if (a==b) Console.WriteLine("Переменные равны");

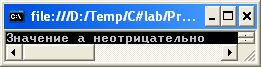
 Console.ReadKey();

**Пример 10:**

int a, b;

a = 3; b=0;

if (a==b) Console.WriteLine("Переменные равны");

 else if (a>=0) Console.WriteLine("Значение а неотрицательно");

Console.ReadKey();

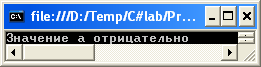
**Пример 11:**

int a, b;

a = -3; b=0;

if (a==b) Console.WriteLine("Переменные равны");

else if (a>=0) Console.WriteLine("Значение а неотрицательно");

 else if (a<0) Console.WriteLine("Значение a отрицательно");

Console.ReadKey();

Операнды логических операций (&, |, !, ^) должны иметь тип **bool**.

Вычисление с использованием логических операций происходит по разрядно.

Логические операторы выполняют логические операции в соответствии с таблицей 4.6:

Таблица 4.6

Логические операции

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Операнды | | лог. И | лог ИЛИ | не Х1 | исключающее ИЛИ |
| X1 | X2 | X1 & X2 | X1 | X2 | ! X1 | X1 ^ X2 |
| 0  0  1  1 | 0  1  0  1 | 0  0  0  1 | 0  1  1  1 | 1  1  0  0 | 0  1  1  **0** |

Сокращенные операторы && (И), || (ИЛИ). Различия между обычной и сокращенной версиями заключается в том, что при обычной версии вычисляются оба операнда, а при сокращенной версии второй операнд вычисляется только при необходимости.

## 4.6 Операторы присвоения

Операторы присвоения задают новое значение переменной. Имя переменной и результат вычисления выражения должны быть одного типа или совместимыми по присвоению.

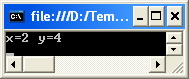
Присвоение бывает простое и сложное. При **простом** присвоении оператор состоит из одного символа (=). Синтаксис оператора:

ИмяПеременной = выражение;

Например, x = x + 1;

y = Math.Sin(x);

Присваиваемое значение должно иметь тип, совпадающий с типом переменной, или допускающий неявное преобразование.

**Пример 12:**

int x=2; double y;

y = x \* x;

Console.WriteLine("x={0} y={1}", x, y);

Console.ReadKey();

В противном случае можно использовать явное преобразование, используя синтаксис:

ИмяПеременной = (тип переменной слева от =) выражение;



**Пример 13:**

double x = 2.5; float y=0;

y = (float) (x \* x);

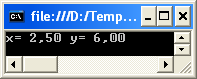
Console.WriteLine("x={0,4:f} y={1,4:f}", x, y);

Console.ReadKey();

**Пример14:**

double x = 2.5; int y=0;

y = (int) (x \* x); // 6,25 округляется до 6

 Console.WriteLine("x={0,5:f} y={1,5:f}", x, y);

Console.ReadKey();

// Приведение double в int ведет к потере дробной части (в примере 0,25)

**Пример 15:**

 double n; int y;

n=5;

y = (int)(Math.Sqrt(n));

Console.WriteLine("y= " +y);

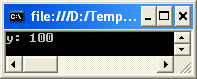
Console.ReadKey();

**Пример 16:**

byte b,y;

b = 10;

y = (byte)(b \* b); //результат представляет выражение int и требует

 преобразование

Console.WriteLine("y: " +y);

Console.ReadKey();

Оператор **сложного** присвоения состоит из нескольких знаков без разделителей. Правый символ – знак простого присвоения, слева дополнительные символы, указывающие на тип дополнительной операции, выполняемой перед присваиванием (табл.4.7).

Таблица 4.7

Составные операторы и их действия

|  |  |
| --- | --- |
| Составные операторы | Действие |
| X += Y | X = X + Y |
| X –= Y | X = X – Y |
| X \*= Y | X = X \* Y |
| X /= Y | X = X / Y |
| X %= Y | X = X % Y |
| X &= Y | X = X & Y |
| X |= Y | X = X | Y |
| X ^= Y | X = X ^ Y |

Преимущества составных операторов:

* Составные операторы компактнее своих «длинных» эквивалентов.
* Их наличие приводит к созданию более эффективного кода (т.к. операнд при этом вычисляется только один раз).

**Примеры** использования составных операторов:

x=x+10 может бать записано x+=10

x=x–100 может бать записано x–=100

## 4.7 Инструкции

Код приложений в C# состоит из инструкций (команд) с ключевыми словами и выражениями с операторами.

В инструкцию включают объявление переменных, присвоение значений, вызов методов, ветвление на один или другой блок кода, в зависимости от заданного условия.

**Инструкция** – строка кода, которая **заканчивается точкой с запятой**.

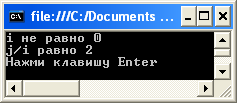
**Блочная инструкция –** набор инструкций **в фигурные скобки { }**. Программный блок представляет собой группирование двух и более инструкций заключенные в фигурные скобки. Последние могут содержать вложенные блоки { {…} }.

Используется для превращения группы операторов в один там, где по правилам языка может применяться только один оператор, а действий несколько.

Программные блоки не снижают динамику выполнения программ.

**Пример** использования однострочных инструкций и блок многострочной инструкции:

**Вариант 1**

static void Main(string[] args)

{

int i, j, d;

i=5;

j=10;

if (i != 0)

{

Console.WriteLine("i не равно 0");

d = j / i;

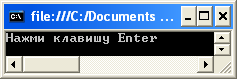
Console.WriteLine("j/i равно "+ d);

}

Console.WriteLine("Нажми клавишу Enter");

Console.ReadKey();

}

Вариант 2:

int i, j, d;

i=0;

j=10;

if (i !=0)

{

Console.WriteLine("i не равно 0");

d = j / i;

Console.WriteLine("j/i равно "+ d);

}

Console.WriteLine("Нажми клавишу Enter");

Console.ReadKey();

***Безусловный переход вызовом функций***

Когда компилятор находит в основном тексте программы имя функции, то происходит приостановка выполнения текущего кода программы и осуществляется переход к найденной функции. Когда функция выполнится и завершит свою работу, то произойдет возврат в основной код программы, на ту инструкцию, которая следует за именем функции.

Имя функции должно содержать пару круглых скобок (), даже если у функции нет аргументов. Это признак функции или метода. Например, Math.Sin(x).

## Оператор перехода goto

Он предназначен для изменения порядка выполнения операторов программы.

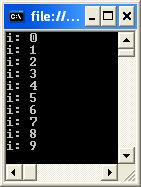
Инструкция **goto** используется следующим образом:

В коде программы создается метка с именем (например, M).

Организуется переход на эту метку инструкцией goto M.

Имя метки M в коде обязательно должно заканчиваться двоеточием (:). Оно указывает на точку в программе, с которой будет выполняться программа после использования инструкции goto.

**Пример**. Вывести последовательность целых чисел до 9. Метка M – начало вывода цифр. Обычно инструкция goto связана с условием.

 public static int Main()

{

int i = 0;

**M:** Console.WriteLine("i: {0 } ", i);

i = i + 1;

if (i < 10) goto M;

// Метка должна быть уникальной.

}

***Условные операторы. Сложные условия***

В программе условие – это выражение логического типа (Bool), которое может принимать одно из двух значений **True**(истина) или **False**(ложь).

Примеры простых условий:

(Summa<1000) // значение переменной Summa < 1000?

(Math.Pow(x,2)-10\*x>=5) // выражение x2 -10·x≥5 ?

(x % 2 = 0) // остаток от деления x на 2 равен 0?

Примеры сложных условий:

(Day==6 **|** Day==7) // день суббота или воскресенье

(Sum>100 **&** Day==7) //скидка на товар стоимостью более 100у.е. в воскресный день

(m==1 **&** d>=1 & d<=10) //дни с 1 по 10 в первом месяце года.

***Логические операторы & | имеют более низкий приоритет, чем операторы сравнения, и поэтому в условиях скобки можно не использовать***.

**Краткие итоги.** В лекции были изучены операторы, которые применяются при работе с языком C#. Рассмотрены примеры работы операторов.

**Контрольные вопросы:**

1. Дайте определение понятию «оператор».
2. Каковы приоритеты выполнения операторов?
3. Приведите примеры работы основного оператора.
4. Приведите примеры работы унарного оператора.
5. Приведите примеры работы мультипликативного и аддитивного операторов.
6. Перечислите операторы отношений и приведите пример их работы.
7. Дайте определение «операторы присвоения» и приведите пример их работы, назовите их преимущества.
8. Дайте определение понятию «инструкция».
9. Назовите оператор перехода и какова его функция.

# Лекция 5

# Инструкции управления

**Цель лекции**. Разобрать инструкцию if, где она применяется и как работает. Изучить вложенные инструкции, инструкции выбора, как они работают.

## 5.1 Инструкция if

Применяется для ветвления по двум ветвям.

**Полный формат** (синтаксис) инструкции:

**if** (условие) инструкция 1; // одна инструкция1 языка

**else** инструкция 2; // одна инструкция2 языка

Условие

Инструкция1

Инструкция2

Да

Нет

Если условие выполняется, то исполняется

инструкция 1, в противном случае исполняется

инструкция 2. Фраза **else** может отсутствовать,

т.к. это **не**обязательная часть инструкции if.

В общем виде, когда надо выполнить несколько действий, инструкция **if** записывается следующим образом:

**if** (условие)

{

Блок инструкций 1; //несколько действий

}

**else**

{

Блок инструкций 2; //несколько действий

}

Условие

Инструкции11

Инструкции21

Да

Нет

Инструкции12

Инструкции22

Фраза **else** может отсутствовать, т.к. это **не**обязательная часть инструкции if.

Если условие выполняется, то исполняется Блок инструкций 1, в противном случае исполняется Блок инструкций 2.

Фигурные скобки записываются в случае, если в блоке более, чем одна инструкция.

Выполняется инструкция **if** следующим образом:

* 1. Вычисляется значение условия (правда или ложь?);
  2. Если условие истинно (**true),** то выполняются инструкции1**.** На этомвыполнение инструкций **if** завершается, инструкции, следующие за словом **else,** не будут выполнены. Если условие ложно (**false**), то выполняются инструкции, следующие за словом **else,** а инструкции за словом **if** игнорируются.

**Пример 1.** Поиск максимального числа из двух чисел x и y.

x>y

max=x

max=y

Да

Нет

// *1-й вариант решения*

int y, x, max;

x=12; y=5;

if (x>y) max=x;

else max=y;

Console.WriteLine("Максимальное "+ max);

**Сокращенный формат if**

Сокращенная форма используется для усеченного алгоритма, когда ветвь с альтернативным решением отсутствует.

нет

max=y

x<y

max=x

да

**if** (условие) инструкция;

// *2-й вариант решения задачи поиска*

максимального числа их двух чисел:

int y, x, max;

x=12; y=5;

max=x;

if(x<y) max=y;

Console.WriteLine("Максимальное "+ max);

Условие

Инструкции11

Да

Нет

Инструкции12

В случае нескольких действий в одной ветви:

**if** (условие)

{

Блок инструкций 1;

}

## 5.2 Вложенные инструкции if; else …

Применяются для множественного ветвления. Синтаксис инструкции:

нет

Инструкции31

условие1

да

нет

условие2

да

Инструкции11

Инструкции21

Инструкции22

Инструкции32

Инструкции12

**if** (условие\_1)

{

Блок инструкций 1;

}

**else**

**if** (условие\_2)

{

Блок инструкций 2;

}

**else**

{

Блок инструкций 3;

}

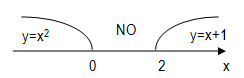
Фраза **else** может отсутствовать, т.к. т.к. это **не**обязательная часть инструкции if.

Если условие1 выполняется, то исполняется Блок инструкций 1, в противном случае проверяется условие2. Если оно выполняется, то исполняется Блок инструкций 2, в противном случае Блок инструкций 3.

В случае, если в какой-либо ветви необходимо выполнить одну инструкцию, то блочная инструкция **в виде фигурных скобок { }** в этой ветви не нужна.

**Пример 2**.

Задача. Вычислить значение функции: 



double y, x;

Console. WriteLine ("Введите х");

//ввод данных с клавиатуры

x = Convert.ToDouble(Console.ReadLine());

/\*преобразует вводимое с клавиатуры число

в виде строки в вещественный тип данных \*/

if (x > 2)

{

y = x + 1;

Console.WriteLine("y= " + y);

}

else if (x < 0)

{

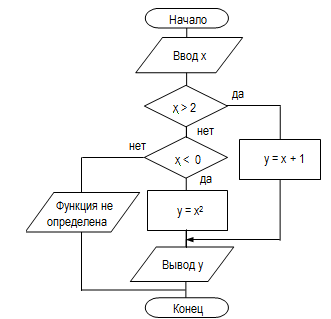
y = x \* x;

Console.WriteLine("y= " + y);

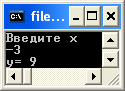
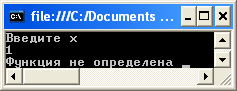
}

else Console.WriteLine("Функция не определена ");

Console.ReadKey();



Ответы решения задачи:



При помощи логических операторов **&** («И») и **|** («ИЛИ») из простых условий можно строить сложные.

Например:



1) 2)

С1

С2

a

С2

b

b

a

Примеры записи условия:

**if** (x<=b & x>=a) <выражение C1>;

**else** <выражение C2>;

или

**if** (x<a | x>b) <выражение C2>;

**else** <выражение C1>;

## 5.3 Инструкции выбора switch, case

Применяется для множественного ветвления на произвольное число направлений.

С его помощью можно выбрать вариант из множества альтернатив.

Если в условии задачи имеется сложный набор условий, то использование вложенных инструкций **if**…**else** приводит к громоздкому коду. Для упрощения кода лучше воспользоваться инструкцией **switch** (переключение)

Инструкция **switch** выбирает нужное действие из списка возможных, размещенных во фразах выбора **case (**выбор**)**.

Общий формат записи:

**switch** (выражение)

{

**case** константа\_1 **:** инструкция действия; инструкция прерывания;

**case** константа\_2 **:** инструкция действия; инструкция прерывания;

……..

**default**:**:** инструкция действия; инструкция прерывания;

}

Выполняются инструкции следующим образом:

1. Сначала вычисляется значение выражения.

2. Полученное значение выражения последовательно сравнивается с константами из списка констант секций case.

3. Если значение выражения (или переменной) совпадает с константой из списка, то выполняется соответствующая этому списку группа инструкций в секции **case**  до тех пор, пока не встретиться инструкция прерывания.

4. Если значение выражения (или переменной) не совпадает ни с одной константой из списка, то, выполняется последовательность инструкций, следующая за **default.**

Синтаксис инструкций **switch**, **case** позволяет не писать **default** и соответствующую последовательность инструкций. В этом случае, если значение выражения не совпадает ни с одной константой из всех списков, то выполняется следующая за **case** инструкция программы.

**Элемент выражения** (помещено в круглые скобки) возвращает константу. **Должен** **быть целочисленного типа** (например, **char**, **byte**, **int**), значение которого определяет дальнейший ход выполнения программы.

**Выражения, имеющие тип с плавающей точкой не разрешены.**

Иногда в качестве управляющего **switch** – выражения используется просто переменная.

**Case** – константы должны быть литералами, тип которых совместим с типом заданного выражения. При этом никакие две **case** – константы в одной **switch** конструкции не могут иметь идентичных значений.

В качестве инструкции прерывания действия используют **break**, которая прерывает выполнение инструкции **switch**. Альтернативой может быть и инструкции **goto**, которую обычно применяют для перехода в другое место программы.

**Пример 3**. Программа запрашивает день недели. В зависимости от введенного номера выводится строка из списка с комментарием.

using System;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

int d;

Console.Write("Введите день недели ");

d= Convert.ToInt16(Console.ReadLine());//преобразует вводимое с клавиатуры число в виде строки в целый тип данных

switch (d)

{

case 1: Console.WriteLine("Понедельник - рабочий день"); break;

case 2: Console.WriteLine("Вторник - рабочий день"); break;

case 3: Console.WriteLine("Среда - рабочий день"); break;

case 4: Console.WriteLine("Четверг - рабочий день"); break;

case 5: Console.WriteLine("Пятница - рабочий день"); break;

case 6: Console.WriteLine("Суббота - самоподготовка"); break;

case 7: Console.WriteLine("Воскресенье - выходной день"); break;

default: Console.WriteLine("Неверный ввод данных. Введите 1<=d<=7"); break;

}

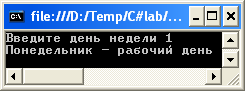
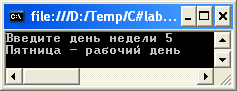
Console.ReadKey();

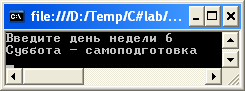
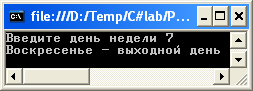
}

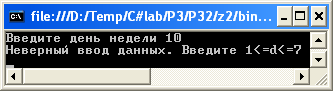
}

}

Результаты работы программы:



 . . .

****

**Пример 4**. Программа принимает с клавиатуры код клавиш A, B, C и в зависимости от ее номера выводит на дисплей текстовый комментарий.

static void Main(string[] args)

{

char q;

Console.WriteLine("Введите букву "); //ввод данных с клавиатуры

q =Convert.ToChar(Console.ReadLine()); /\*преобразует вводимое с клавиатуры число в виде строки в символьный тип данных\*/

switch (q) // переменная символьного типа

{

case 'A': Console.WriteLine("q содержит A"); break;

case 'B': Console.WriteLine("q содержит В"); break;

case 'C': Console.WriteLine("q содержит C"); break;

}

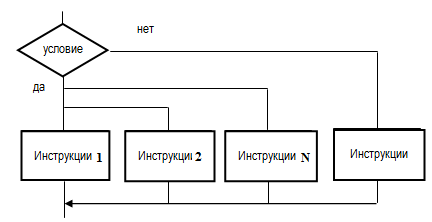
Console.ReadKey();

}

Необязательная инструкция default отсутствует.

В общем виде, когда надо выполнить несколько действий, операторs switch,  case записывается следующим образом:

**switch**  (выражение)

**case** константа\_n1**:**

{

//инструкции1;

}

**case** константа\_n2**:**

{

//инструкции2;

}

**case** константа\_nn**:**

{

//инструкции nn;

}

**default**:

{

//инструкции;

}

В консольном приложении вводить данные можно программно, присваивая переменным или константам значения в соответствии с объявленным ранее типом или непосредственно с клавиатуры при исполнении программы. Например:

**x = Convert.ToDouble(Console.ReadLine());** //вещественный тип

**d = Convert.ToInt16(Console.ReadLine());** //целый тип

**z = Convert.ToInt32(Console.ReadLine());** //целый тип

**ch=Convert.ToChar(Console.ReadLine());** //символьный тип

Методы содержат процедуру чтения с последующим образованием новой строки (ReadLine()) и функцию преобразования вводимых с клавиатуры данных, воспринимаемых в виде строки в тип данных, соответствующих объявленному диапазону.

**Пример 5.** Используя оператор выбора, вычислить значения функции:



Конец

Ввод x

Начало

Да

Нет

Ввод А

y = cos(x)

A = 0, 1, -1

y = (x-10)⋅ 8

y = A ⋅ 7⋅ x

Вывод x, y

[ Введите параметр А

M

А введено

не верно!

static void Main(string[] args)

{

int A;

double x,y;

Console.WriteLine("Введите x ");

x= Convert.ToDouble(Console.ReadLine());

M:Console.WriteLine("Введите параметр А ");

A= Convert.ToInt16(Console.ReadLine());

switch (A)

{

case 0:y=Math.Cos(x); break;

case 1:y=(x-10)\*8; break;

case -1:y=A\*7\*x; break;

default:

{

Console.WriteLine("A введено не верно!");

goto M;

}

}

Console.WriteLine(" А=" + A );

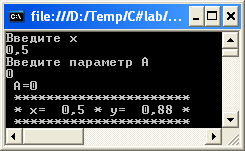
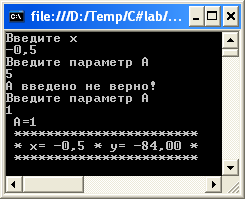
Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

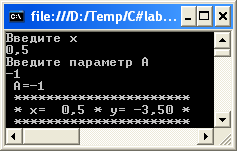
Console.WriteLine(" \* x= {0,4} \* y= {1,6:f} \*",x,y);

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

Console.ReadKey();

}





Программа была выполнена три раза с разными значениями параметра А, что бы показать, правильное решение задачи.

**Краткие итоги.** В лекции рассмотрены инструкции необходимые для решения задач с условием ветвления. Для этого были изучены следующие инструкции: if, else, switch, case. И дан пример работы оператора выбора goto.

**Контрольные вопросы:**

1. Применение инструкции if и как она выполняется.
2. Приведите пример выполнения инструкции if/
3. Опишите работу вложенных инструкций if, else.
4. Инструкции выбора switch, case.
5. Формат записи инструкций switch, case.
6. Как выполняются инструкции switch, case.
7. Приведите пример выполнения инструкции switch, case.
8. Приведите пример выполнения оператора goto.

# Лекция 6

# Циклы

**Цель лекции.** Изучить решение циклической формы с помощью языка C#. Рассмотреть операторы цикла, их назначение и применение. Дать определение цикла.

## 6.1 Операторы цикла

Алгоритмы решения многих задач являются **циклическими**, т.е. для достижения результата определенная последовательность действий должна быть выполнена несколько раз. Циклом называется группа инструкций, повторяющихся многократно с разными данными. Например, программа контроля знаний. Для циклов применяются инструкции: goto, for, while, do while.

**Инструкция goto**

Инструкция goto в первых языках программирования была основой для реализации циклов и многократных переходов, вследствие чего возникала запутанность кода программы. Опытные программисты стараются ее не использовать, но для того, чтобы узнать все возможности языка, рассмотрим и ее.

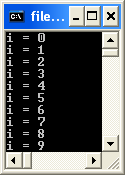
Инструкция goto используется следующим образом:

* В коде программы создается метка с именем (например, M).
* Организуется переход на эту метку инструкцией goto M.

Имя метки M в коде обязательно должно заканчиваться двоеточием (:). Оно указывает на точку в программе, с которой будет выполняться программа после использования инструкции goto.

Обычно инструкция goto привязывается к условию.

**Пример 1**. Используя цикл (метка M - начало), написать программу, в которой в консоль выводится последовательность чисел.



public static int Main()

{

int i = 0;

M : Console.WriteLine("i= {0} ", i);

i = i + 1;

if (i < 10) goto M;

Console.ReadKey();

}

}

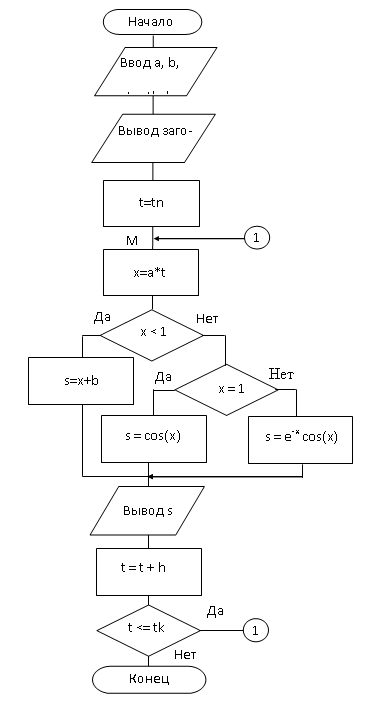
Решение циклических задач можно организовать на основе структуры ветвление или с помощью инструкций циклов.

**Пример 2.** Вычислить на ЭВМ значения функций



при а=1.3, b=1.29, t = [ 0.1… 2.2], t изменяется в диапазоне от 0.1 до 2.2 с шагом h=0.3.

Организовать вывод значений аргумента и вычисленного значения функции в виде таблицы с заголовком.



static void Main(string[] args)

{

double tn = 0.1, tk = 2.2, h = 0.3, t=0.1, x, s;

const double a = 1.3, b = 1.29;

Console.WriteLine("Таблица расчета функции s");

Console.WriteLine();

t = tn;

M : x = a \* t;

if (x < 1) s = x + b;

else if (x == 1) s = Math.Cos(x);

else s=Math.Exp(x)\*Math.Cos(x);

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

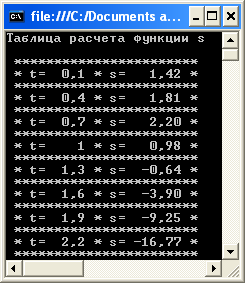
Console.WriteLine(" \* x= {0,4} \* s= {1,6:f} \*",x,s);

t=t+h;

if (t<=tk) goto M;

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

Console.ReadKey();

 }

**Инструкция for**

Служит для организации циклов с заранее известным числом повторений. Циклы, в которых количество повторений заранее определено, называются регулярными.

В цикле изменение индекса цикла заложено в инструкцию. Задаются - начальное значение индекса (инициализация), условие выполнения, правило изменения индекса после итерации. Разделители для параметров инструкции for – точка с запятой (;).

Формат записи цикла for для повторного использования одной инструкции имеет вид:

Тело цикла

Индекс цикла и изменение

Инструкция

* for (инициализация; условие выполнения; итерация) инструкция;

Если цикл предназначен для повторного выполнения программного блока, то его формат:

for (индекс цикла = начало; условие выполнения; изменение индекса)

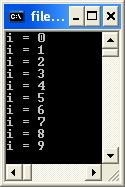
{

Инструкции тела цикла;

}

Существует 2 варианта записи инструкции: инкрементный и декрементный.

**Инкрементный вариант:**

Программа использует цикл, в котором в консоль выводится последовательность чисел от 0 до 9 (шаг = 1).

static void Main(string[] args)

{

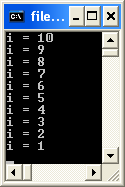
for (int i = 0; i < 10; i++) // шаг = 1

Console.WriteLine("i = {0} ", i);

Console.ReadKey();

}

**Декрементный вариант:**

Программа использует цикл, в котором в консоль выводится последовательность чисел от 10 до 1 (шаг = -1).

static void Main(string[] args)

{

for (int i = 10; i > 0; i−−) // шаг = −1

Console.WriteLine("i = {0} ", i);

Console.ReadKey();

}

Выполняется оператор **For** следующим образом:

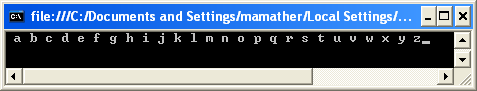
* 1. Сначала вычисляется и запоминается начальное и конечное значения индекса цикла (индекс – порядкового типа).
  2. Далее индексу цикла присваивается начальное значение.
  3. Затем значение индекса цикла сравнивается с конечным значением.
  4. Далее, пока индекс цикла ≤ конечного значения (в 1 варианте оператора), выполняется очередная итерация цикла, в противном случае – выход из цикла.
  5. При завершении цикла индекс цикла объявляется **неопределенным**.

**Пример 3**. Вывести на экран буквы от A до Z (инкрементный вариант)

static void Main(string[] args)

{

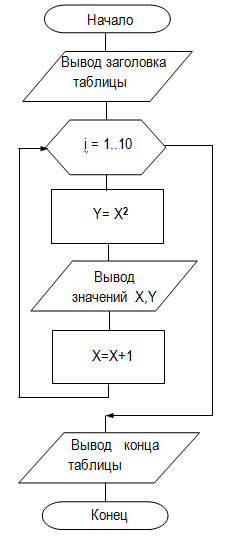
for (char i = 'a'; i <= 'z'; i++)

 Console.Write(" "+ i);

Console.ReadKey();

}

**Пример 4**. Написать программу, которая выводит таблицу квадратов первых десяти целых положительных чисел (инкрементный вариант).



static void Main(string[] args)

{

int x = 1, y = 1;

Console.WriteLine("Таблица квадратов:");

Console.WriteLine();

Console.WriteLine("------------------");

Console.WriteLine(" x | y ");

Console.WriteLine("------------------");

for (byte i = 1; i <= 10; i++)

{

y = x \* x;

Console.WriteLine(" x = {0,2} | y = {1,3}", x, y);

x=x+1;

}

Console.WriteLine("------------------"); ;

Console.ReadKey();

}

**Пример 5**. Обратный отсчет времени перед запуском (декрементный вариант)

static void Main(string[] args)

{

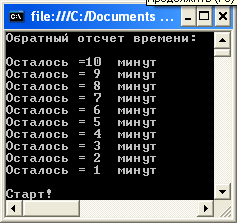
Console.WriteLine("Обратный отсчет времени:");

Console.WriteLine();

for (byte second = 10; second >= 1; second--) //отрицательное приращение

{

Console.Write("Осталось ={0,2} ", second);

Console.WriteLine(" минут");

}

Console.WriteLine();

Console.WriteLine("Старт!");

Console.ReadKey();

}

## 6.2 Использование нескольких управляющих переменных цикла

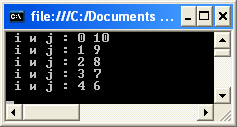
Для управления циклом for можно использовать две и больше переменных. В этом случае инструкции инициализации и итерации для каждой из этих переменных отделяются запятыми.

**Пример 6.** Программа использует цикл, в котором в консоль выводятся последовательности чисел от 0 до 4 (шаг = 1) и от 10 до 6 (шаг = -1).

static void Main(string[] args)

{

int i, j;

for (i = 0, j = 10; i < j; i++, j--)

Console.WriteLine(" i и j : " + i + " " + j);

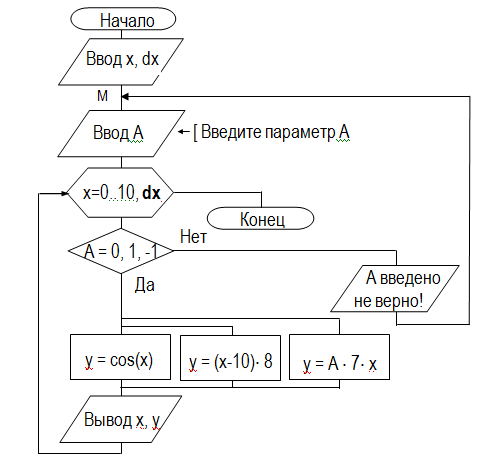
Console.ReadKey();

}

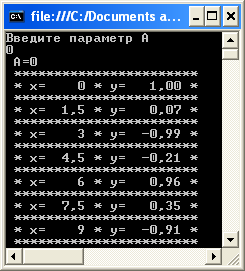
## 6.3 Использование переменных цикла c любым шагом их изменения

**Пример 7**. Используя оператор выбора, вычислить значения функции:

 если x изменяется в интервале 0..10 с шагом dx=1.5



int A;

double x = 0, y, dx = 1.5; ;

M: Console.WriteLine("Введите параметр А ");

A = Convert.ToInt16(Console.ReadLine());

Console.WriteLine(" А=" + A);

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

for (x = 0; x <= 10; x = x + dx)

{

switch (A)

{

case 0: y = Math.Cos(x); break;

case 1: y = (x - 10) \* 8; break;

case -1: y = A \* 7 \* x; break;

default:

{

Console.WriteLine("A введено не верно!");

goto M;

}

}

Console.WriteLine(" \* x= {0,4} \* y= {1,6:f} \*", x, y);

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

}

Console.ReadKey();

## 6.4 Цикл с предусловием. Инструкция while . .

Цикл с инструкцией **while ..** используется в том случае, если последовательность действий надо выполнить несколько раз, причем количество повторений заранее не известно и может быть определено, только во время работы программы.

Процессы, в которых количество повторений заранее не определено, называются итерационными. А сами циклы – называются итеративными.

В этом виде цикла условие стоит перед телом цикла, поэтому этот цикл называют с предусловием. Тело цикла повторяется, **пока выполняется условие**. Тело цикла первый раз выполняется с проверкой условия.

Форма цикла while с одиночной инструкцией:

**while** (Условие) инструкция;

В случае повторного выполнения программного блока

формат цикла выглядит следующим образом:

**while** (Условие)

**Условие**

**Инструкция**

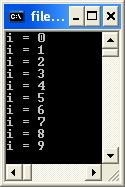
**Да**

**Нет**

{

Инструкции тела цикла;

}

**Пример 8**. Программа использует цикл, в котором в консоль выводится последовательность чисел от 0 до 9.

static void Main(string[] args)

{

int i = 0;

while (i < 10)

{

Console.WriteLine("i = {0 } ", i);

i = i + 1;

}

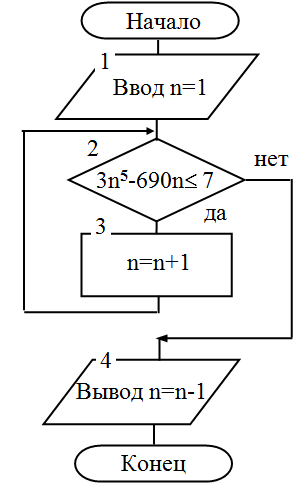
Console.ReadKey();

}

Инструкция **While** выполняется следующим образом:

* 1. Сначала вычисляется значение выражения условия.
  2. Если значение выражения условия равно **True** (истина), то выполняются очередные инструкции тела цикла. После этого снова проверяется выполнение условия. Если условие выполняется, то инструкции цикла выполняются еще раз. И так до тех пор, пока условие не станет ложным (**False**).
  3. Если значение выражения условия равно **False** (ложно), то цикл прекращается, выполнение оператора цикла заканчивается и выполняется следующий оператор, предусмотренный программой.

**Пример 9.** Вычислить наибольшее ***положительное*** целое число n, удовлетворяющее условию: 3n5 - 690n ≤ 7.



static void Main(string[] args)

{

int n = 1;

while (3\*Math.Pow(n,5)-690\*n <= 7)

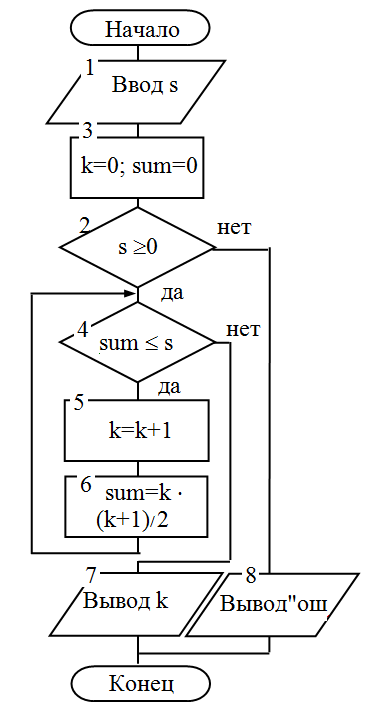
n = n + 1;

Console.WriteLine("n = {0 } ", n-1);

Console.ReadKey();

}

**Пример 10.** Определить число К натуральных чисел, сумма которых не превышает S.



static void Main(string[] args)

{

double sum=0, s; int k = 0;

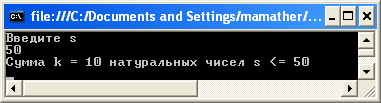
Console.WriteLine("Введите s");

//ввод данных с клавиатуры

s = Convert.ToDouble(Console.ReadLine());

/\*преобразует вводимое с клавиатуры число

в виде строки в вещественный тип данных \*/

if (s >= 0)

{

while (sum < s)

{

k=k+1;

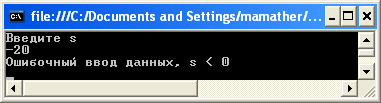
sum=k\*(k+1)/2;

}

Console.Write("Сумма k = {0}", k);

Console.WriteLine(" натуральных чисел s <= {0}", s);

}

else Console.WriteLine("Ошибочный ввод данных, s < 0 " );

Console.ReadKey();

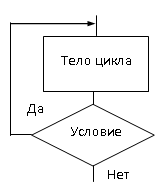
}

## 6.5 Цикл с постусловием. Инструкция do - while

Цикл с инструкцией **do . . while** используется в том случае, если последовательность действий надо выполнить несколько раз, причем количество повторений заранее не известно и может быть определено во время работы программы.

Относится к итерационным процессам.

В этом виде цикла условие стоит после тела цикла, поэтому этот цикл называют с постусловием. Тело цикла повторяется, пока выполняется условие. Тело цикла первый раз выполняется без проверки условия. Выход из цикла при **не выполнении условия**.

Эта циклическая инструкция работает по принципу: «Повторить — пока выполняется условие».

Ее синтаксис выглядит следующим образом:

do

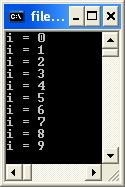
{

Инструкции тела цикла;

}

while (Условие);

**Пример 11**. Программа использует цикл, в котором в консоль выводится последовательность чисел.

static void Main(string[] args)

{

int i = 0;

do

{

Console.WriteLine("i = {0} ", i);

i = i + 1;

}

while (i < 10);

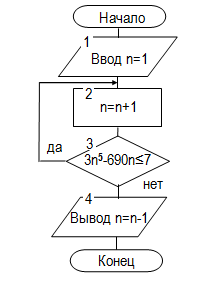
Console.ReadKey();

}

Инструкция **do…while** выполняется следующим образом:

* 1. Сначала выполняются находящиеся между **do…while** инструкции тела цикла.
  2. Затем вычисляется значение выражения условия.
  3. Если значение выражения условия истинно (**True)**, то инструкции тела цикла выполняются еще раз.
  4. Если значение выражения условия ложно (**False**), то выполнение цикла прекращается и выполняется следующий оператор, предусмотренный ходом выполнения программы.

Цикл **do…while** используется для организации приближенных вычислений с заданной точностью, задач поиска и обработки данных в массиве или в файле.

**Пример 12**. Вычислить наибольшее положительное целое число n, удовлетворяющее условию: 3n5 - 690n **≤** 7.

static void Main(string[] args)

{

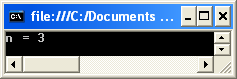
int n = 1;

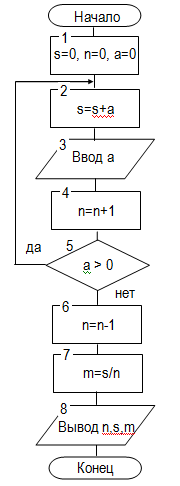
do n = n + 1;

while (3 \* Math.Pow(n, 5) - 690 \* n <= 7);

Console.WriteLine("n = {0 } ", n - 1);

Console.ReadKey();

 }

**Пример 13.** Написать программу, вычисляющую сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры.

static void Main(string[] args)

{

int s = 0, a = 0, n = 0; double m;

Console.WriteLine("Введите положительные числа");

Console.WriteLine("Для выхода из цикла введите 0");

do

{

s = s + a;

Console.Write(">> ");

a = Convert.ToInt16(Console.ReadLine());

n = n + 1;

}

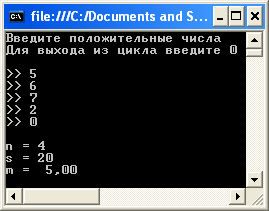
while (a > 0);

n = n - 1;

Console.WriteLine("n = {0} ", n);

Console.WriteLine("s = {0} ", s);

m = s / n;

Console.WriteLine("m = {0,5:f} ", m);

Console.ReadKey();

}

## 6.6 Накопление сумм и произведений

**Пример 14** . Написать программу вычисляющую сумму и произведение чисел от 1 до 10.

Рассмотрим процесс накопления S суммы и P произведения чисел от 1 до 10.

Накопление суммы Накопление произведения

S0=0 // 0

S1=S0 + 1 // 0 + 1=1

S2=S1 + 2 // 1 + 2=2

S3=S2 + 3 // 2 + 3=5

S4=S3 + 4 // 5 + 4=9

. . . . . . . . . . . . . . . . . . . .

S10=S9 + 10 // 45 + 10=55

P0=1 // 1

P1=P0 \* 1 // 1 \* 1=1

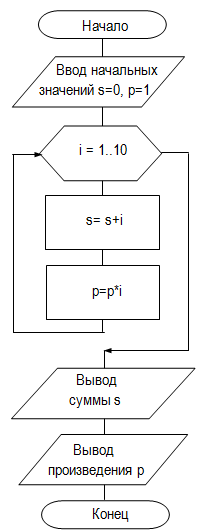
P2=P1 \* 2 // 1 \* 2=2

P3=P2 \* 3 // 2 \* 3=6

P4=P3 \* 4 // 6 \* 4=24

. . . . . . . . . . . . . . . . . . . . . . . . . . . .

P10=P9 \* 10 // 362880 \* 10=3628800



Блок схема и программа накопления S и P:

static void Main(string[] args)

{

int s, p, i;

s = 0;

p = 1;

for (i = 1; i <= 10; i++)

{

s = s + i;

p = p \* i;

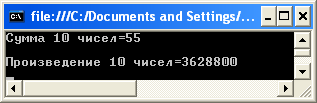
}

Console.WriteLine("Сумма 10 чисел=" + s);

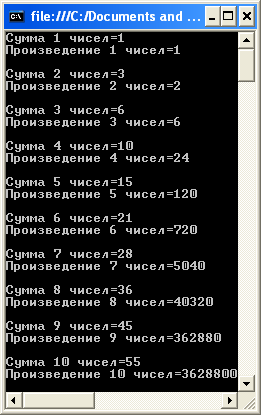
Console.WriteLine();

Console.WriteLine("Произведение 10 чисел=" + p);

Console.ReadKey();

 }

Программные блоки не снижают динамику выполнения программ. Наоборот, их наличие позволяет в одном цикле вычислить как сумму, так и произведение.

Для вывода всех частичных сумм и произведений их вычисления надо поместить в блок тело цикла.

static void Main(string[] args)

{

int s, p, i;

s = 0;

p = 1;

for (i = 1; i <= 10; i++)

{

s = s + i;

Console.WriteLine("Сумма "+ i + " чисел=" + s);

p = p \* i;

Console.WriteLine("Произведение " + i +

" чисел=" + p);

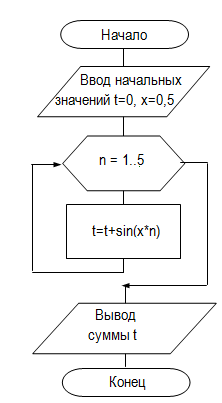
Console.WriteLine();

}

Console.ReadKey();

}

**Пример 15**. Написать программу вычисляющую сумму членов ряда:

t=sin(x\*1)+ sin(x\*2)+ sin(x\*3)+ sin(x\*4)+ sin(x\*5)

t0 = 0

t1 = t0 + sin(x\*1)

t2 = t1 + sin(x\*2)

t3 = t2 + sin(x\*3)

t4 = t3 + sin(x\*4)

t5 = t4 + sin(x\*5)

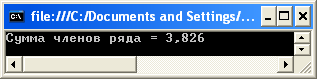
static void Main(string[] args)

{

int n; double x = 0.5, t=0;

for (n = 1; n <= 5; n++)

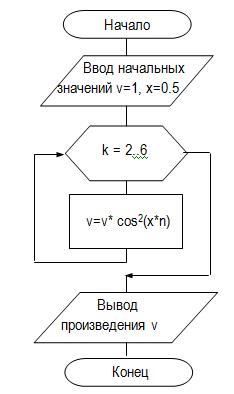
t =t + Math.Sin(x\*n);

 Console.WriteLine("Сумма членов ряда = {0:#.###}", t );

Console.ReadKey();

}

**Пример 16.** Написать программу вычисляющую произведение членов ряда:

v=cos2(x\*2) \* cos2(x\*3) \* cos2(x\*4) \* cos2(x\*5) \* cos2(x\*6)

v0 = 1

v1 = v0 \* cos2(x\*2)

v2 = v1 \* cos2(x\*3)

v3 = v2 \* cos2(x\*4)

v4 = v3 \* cos2(x\*5)

v5 = v4 \* cos2(x\*6)

static void Main(string[] args)

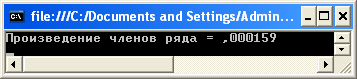
{

int k; double x = 0.5, v=1;

for (k = 2; k <= 6; k++)

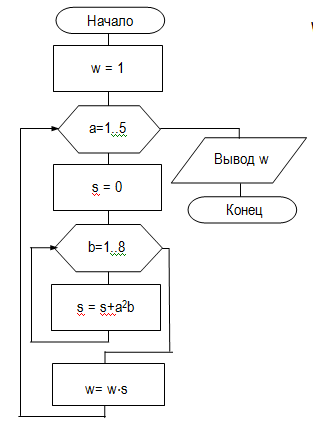
v = v \* Math.Pow(Math.Cos(x\*k),2);

Console.WriteLine("Произведение членов ряда = {0:#.######}", v);

 Console.ReadKey();

}

**Пример 17.** Написать программу вычисляющую функцию

:

W=12\*(1 + 2 + 3 + 4 + 5 + 6 + 7+ 8)\*

22\*(1 + 2 + 3 + 4 + 5 + 6 + 7+ 8)\*

32\*(1 + 2 + 3 + 4 + 5 + 6 + 7+ 8)\*

42\*(1 + 2 + 3 + 4 + 5 + 6 + 7+ 8)\*

52\*(1 + 2 + 3 + 4 + 5 + 6 + 7+ 8)\*

static void Main(string[] args)

{

int a=1,b=1,s=0,w=1;

for (a = 1; a <= 5; a++)

{

s=0;

for (b = 1; b <= 8; b++)

s = s + a\*a \* b;

w = w \* s;

}

 Console.WriteLine("w = {0}", w);

Console.ReadKey();

}

## 6.7 Безусловные переходы

Бывают ситуации, когда необходимо прекратить выполнение цикла досрочно, опустив выполнение кода, оставшегося в теле цикла и проверку условного выражения.

С помощью инструкции **break** можно организовать немедленный выход из цикла.

**Пример 18**. В консоль выводятся только отрицательные числа в заданном диапазоне

static void Main(string[] args)

{

for (int i = -10; i <= 10; i++)

{

if (i > 0) break; //Завершение цикла при i>0

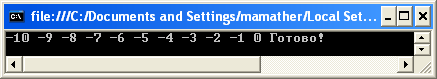
Console.Write(i + " ");

}

Console.WriteLine(“Готово!”);

Console.ReadKey();

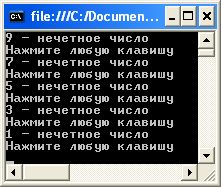
}



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

Инструкция **continue** в отличие от **break** не прерывает хода выполнения цикла. Она лишь приостанавливает текущую итерацию и переходит к следующей итерации.

**Пример**. В консоль выводятся нечетные числа в заданном диапазоне. Поверка осуществляется проверкой остатка от деления на 2, для нечетных чисел он равен 1. В цикле перебираются все числа от 1 до 10. Если очередное число четное, то итерация завершается с пропуском последующих инструкций тела цикла и переходом к следующей итерации.

static void Main(string[] args)

{

for ( int i = 10; i > 0; i--)

{

if ( i%2 !=1 ) continue;

Console.WriteLine("{0} - нечетное число", i);

Console.WriteLine("Нажмите любую клавишу");

Console.ReadKey();

}

}

Здесь выводятся только нечетные числа, поскольку при обнаружении четного числа происходит преждевременный переход к следующей итерации и метод WriteLine не вызывается.

**Инструкция foreach**

Инструкция **foreach** (для каждого) предназначена для обработки массивов. Она повторяет группу вложенных в нее инструкций для каждого элемента массива.

Синтаксис инструкции:

foreach (<ИндексЦикла> in <ИмяМассива>)

{

Инструкции тела цикла;

}

В любой точке блока foreach цикл можно разорвать с помощью ключевого слова **break** или перейти к следующей итерации в цикле с помощью ключевого слова **continue**.

Цикл foreach также может быть разорван при помощи инструкций **goto**, **return** или **throw**.

**Пример**. В консоль выводится содержимого массива целых чисел.

static void Main(string[] args)

{

int[] Massiv = { 0, 1, 2, 3, 5, 8, 13 }; // Определен массив чисел

foreach (int i in Massiv)

{

Console.WriteLine(i);

}

Console.ReadKey();

}

**Краткие итоги.** В лекции были рассмотрены вопросы построения цикла с помощью инструкций goto, for, while, do-while. Построение программ накопления суммы и произведения. Представлены примеры работы всех перечисленных инструкций.

**Контрольные вопросы:**

1. Что называется «циклом»?
2. Как используется инструкция goto для решения цикла?
3. Для чего служит инструкция for?
4. Какие варианты использования инструкции for вы знаете, приведите примеры.
5. Как используются несколько управляющих переменных цикла, приведите примеры.
6. Как используются переменные цикла с любым шагом их измерения, приведите примеры.
7. В каких случаях используется инструкция while, приведите примеры.
8. Как выполняется инструкция while?
9. В каких случаях используется инструкция do. . .while, приведите примеры
10. Как выполняется инструкция …while?
11. Приведите примеры накопления суммы.
12. Приведите примеры накопления произведения.
13. С помощью, каких инструкций организуется безусловный переход, приведете примеры.
14. Как работает инструкция foreach?

# Лекция 7

# Массивы

**Цель лекции**. Дать определение, что такое массив. Изучить структуру массива, какие они бывают. Уяснить, как объявляется формат массива. Что такое инициализация. Рассмотреть примеры решения массива.

Тип массив – структура данных, представляющая собой набор переменных одинакового типа, имеющих общее имя.

Каждый элемент массива однозначно определяется именем и индексом (номером элемента в массиве).

Индексы массива принадлежат целочисленному типу. Массивы позволят легко обрабатывать большое количество связанных переменных.

Массив может быть:

* одномерным A1 [ ]
* двумерным A2 [ , ]



Массив заданный математически имеет вид:

* одномерный - строка - A = (1, 5, 8, 3, 4, 2, 7, 0) или столбец
* двумерный B ( например, 3 – строки и 2 – столбца )

**Формат объявления одномерного массива в C#:**

тип[ ] ИмяМассива = new тип [размер];

**Формат объявления двумерного массива в C#:**

тип [ , ] ИмяМассива = new тип [количество строк, количество столбцов];

тип – **type** – имя типа значений элементов.

[ ] – признак массива. Запятые внутри скобок задают размерность массива. Запятых нет – массив одномерный, запятая одна – массив двумерный.

ИмяМассива – имя массива.

new – оператор, предназначенный для создания объекта любого классового типа, позволяет динамически размещать в памяти элементы массива.

Размер – количество элементов, которые будут храниться в массиве.

Элементы массива могут быть любых типов.

Доступ к элементу массива осуществляется посредством индекса (позицию элемента внутри массива):

ИмяМассива [ НомерЭлемента ].

Индексация массивов начинается с нуля: массив с элементами n индексируется от 0 до n-1. При ***обращении*** к элементу массива, надо указать его имя и номер элемента в квадратных скобках.

**Например**: A[0] – первый элемент массива A.

A[4] – пятый элемент массива А.

B[2,3] – элемент, лежащий на пересечении 3 строки и 4 столбца массива B.

**Примеры объявления массивов:**

static void Main(string[] args)

{

int[ ] А = new int[5]; // одномерный массив А из 5 целых чисел

char[ ] H = new char[10]; // одномерный массив H из 10 символов

double[,] M = new double [2, 3]; // двумерный массив 2х3 вещественных

//чисел, содержащий 2 строки и 3 столбца

}

При объявлении массива можно выполнить его инициализацию, т.е. присвоить начальные значения элементам массива в момент его создания.

**Формат объявления одномерного массива с инициализацией**:

тип [ ] ИмяМассива = {v1,v2,v3,…vn};

Формат объявления двумерного массива в C#:

тип [ , ] ИмяМассива = {{v11,v12,…v1j},…{vi1,vi2,…vij}};

//1-я строка ……… i-я строка

Здесь начальные значения, присваиваемые элементам массива, задаются с помощью последовательности v1, v2, v3,…vn для одномерного массива и vij – для массива размерностью ( i x j ), где i – номер строки, а j – номер столбца для двумерного массива.

**Примеры объявления массивов с инициализацией:**

int[ ] С = new int[ ] { 1, 3, 5, -7, 9 }; // массив С инициализирован пятью элементами

double[ ] D = { 1.5, 2.1, 3.65, 4.7, 5.14, 6.36 }; // Альтернативный синтаксис

char[ ] Q = {'a','b','g'}; //массив Q инициализирован тремя символами

string[ ] Team = {"Zenith", "Dynamo", "Sparta", "Rotor", "CSK"};

int[ , ] M= { { 1, 2, 3 }, { 4, 5, 6 } }; // массив M: 2 строки и 3 столбца инициализирован

int[ ,] L ={{0,2,4,6},{2,9,6,3},{4,7,5,8}, {1,6,5,7}};

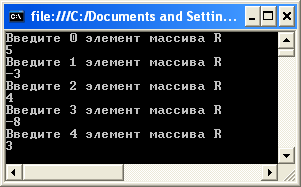
//массив L (4х4) инициализирован. . .

При инициализации допустимо указывать размер массива, но количество элементов списка инициализации должно соответствовать размерности массива. Иначе компилятор выведет сообщение об ошибке.

Например:

Int [ ] nems – new int [ 10 ] { 99, 10, 100, 15, 76, 23, 85, 9 , 87, 49};

В этом объявлении размер массива **nems** явно задан равным 10.

Массив можно инициализировать во время выполнения программы в цикле.

**Например**:

static void Main(string[] args)

{

int[] R = new int[5] ;

for (int k=0; k<=4; k++)

{

Console.Write("Введите {0}", k);

Console.WriteLine(" элемент массива R");

R[k] = Convert.ToInt32(Console.ReadLine());

//ввод элемента массива R

}

Console.ReadKey();

}

В C# также предусмотрена инструкция foreach. Она обеспечивает простой и понятный способ выполнения итерации элементов в массиве.

Синтаксис инструкции foreach:

foreach (int ИндексМассива in ИмяМассива)

{

Инструкции тела цикла

}

int ИндексМассива – тип и имя переменной для номера элемента массива (целое число).

Слово in – в.

ИмяМассива – имя массива.

**Пример**. Следующий код создает массив **Числа** и осуществляет его итерацию с помощью инструкции foreach.

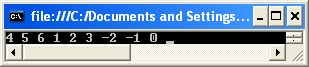
int[] Числа = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };

foreach (int i in Числа)

{

Console.Write("{0} ", i);

}

Вывод в консоль строки: 4 5 6 1 2 3 -2 -1 0 

Операции, производимые над элементами массива, полностью определяются типом этих элементов.

## 7.1 Операции с массивами

Типовые операции при работе с массивами:

* Вывод массива;
* Ввод массива;
* Поиск максимального/минимального элемента массива;
* Поиск заданного элемента массива;
* Сортировка массива.

Часто используемые операции: накопление суммы элементов массива, расчет среднего арифметического значения элементов массива.

**Вывод массива**

Под выводом массива понимается вывод на экран монитора, значений элементов заданного / сформированного массива.

При выводе всех элементов массива удобно использовать оператор цикла ***for*** или ***foreach***, при этом переменная счетчик может быть использована в качестве индекса элементов массива.

Пример вывода одномерного массива Team:

static void Main(string[] args)

{

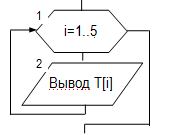
string[] Team = { "Zenit ", "Dynamo ", "Sparta ", "Rotor ", "CSK "

};

foreach (string i in (string[])Team)

{

Console.Write("{0} ", i); //вывод элементов массива Team

 }

Console.ReadKey();

}



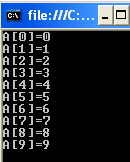
**Ввод массива**

Под вводом массива понимается процесс получения от пользователя во время работы программы, значений элементов массива.

При вводе элементов массива удобно использовать оператор цикла ***for*** или ***foreach***, при этом переменная счетчик может быть использована в качестве индекса элементов массива.

Иногда в качестве элементов массива используют случайные числа, которые можно получить с помощью функции **Random()** - случайное число в диапазоне 0..1.

Программа помещает в массив A числа от 0 до 9.

static void Main(string[] args)

{

int[] A = new int[10]; // одномерный массив А из 10 целых чисел

for (int i=0; i<10; i++)

A[i] = i;

for (int i = 0; i < 10; i++)

Console.WriteLine("A[" + i + "]=" + A[i]);

Console.ReadKey();

}

Таблица 7.1

Схематично представленный массив А

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] | A[9] |

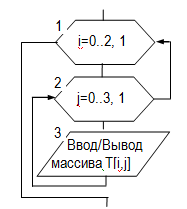
Пример заполнения двумерного массива Т[3x4] числами от 1 до 12:

static void Main(string[] args)

{

int[ , ] T = new int[3,4];

Console.WriteLine("Сформированная матрица");

Console.WriteLine();

for (int i = 0; i < 3; ++i)

{

for (int j = 0; j < 4; ++j)

{

T[i, j] = (i \* 4) + j + 1;

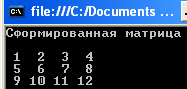
Console.Write("{0,2} ",T[i, j]);

}

Console.WriteLine();

}

Console.ReadKey();

 }

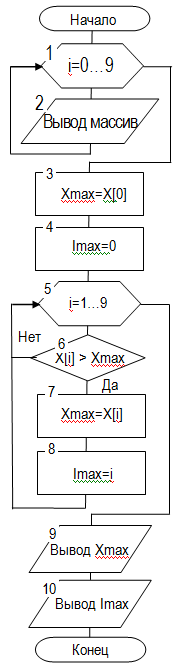
В этом примере элемент массива Т[0,0] примет значение 1, Т[0,1] примет значение 2 … Т[2,3] примет значение 12. Схематично массив Т можно представить:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| индексы | i , j | 0 | 1 | 2 | 3 |
|  | 0 | 1 | 2 | 3 | 4 |
|  | 1 | 5 | 6 | 7 | 8 |
|  | 2 | 9 | 10 | 11 | 12  T[2,3] |

**Поиск максимального ( или минимального ) элемента массива**

**Алгоритм поиска**. Делается предположение, что 1-й элемент массива max (min), затем все элементы массива сравниваются с ним. Если во время проверки обнаруживается, что очередной элемент больше (меньше) принятого за max (min), то этот элемент становится максимальным (минимальным).

**Пример 1.** В массиве X из 10 элементов вычислить наибольший элемент массива и его номер.



static void Main(string[] args)

{

int[] X = new int[10]{-5, 6, -8, 2, 4, 9, -7, 4, 1, 0};

// одномерный массив Х из 10 целых чисел

int Xmax; // максимальный элемент

int Imax; // номер максимального элемента

for (int i = 0; i < 10; i++)

Console.WriteLine("X[" + i + "]=" + X[i]);

Console.WriteLine(); // вывод элементов массива

Xmax = X[0]; // предположение

Imax = 0;

for (int i = 1; i < 10; i++)

if (X[i] > Xmax)

{

Xmax = X[i]; // выявление max

Imax = i;

}

Console.WriteLine("Максимальный элемент: {0} ", Xmax);

Console.WriteLine("Его номер: {0} ", Imax+1);

Console.ReadKey();

}



При выявлении минимального элемента условие > заменяется на < :

if (X[i] < Xmin)

{

Xmin = X[i]; // выявление min

Imin = i; // его номера

}

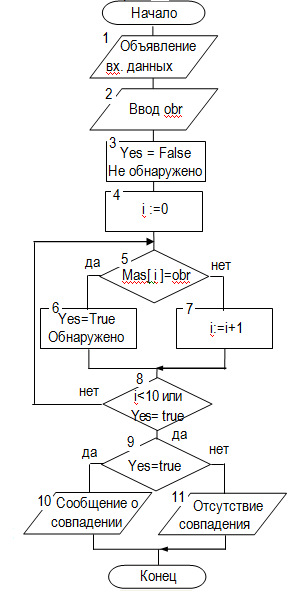
**Поиск заданного элемента массива**

Под ***поиском заданного элемента массива*** понимается необходимость определить, содержит ли массив определенную информацию или нет. Наиболее простой алгоритм поиска – простой перебор неупорядоченных элементов.

Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или пока не будут проверены все элементы.

Для поиска элементов используют операторы цикла **for**, **while** или **do** ... **while**

**Пример 2**. Найти заданный элемент в массиве и вывести его на экран дисплея.



static void Main(string[] args)

{

int[] Mas = new int[10]{-5, 6, -8, 2, 4, 9, -7, 4, 1, 0};

// одномерный массив Mas из 10 целых чисел

int obr; // образец для поиска

bool Yes; // признак обнаружения

int i; //счетчик цикла

for (i = 0; i < 10; i++)

Console.WriteLine("Mas[" + i + "]=" + Mas[i]);

Console.WriteLine();

Console.WriteLine("Введите образец для поиска");

obr = Convert.ToInt16(Console.ReadLine());

Yes = false;

i=0;

do

{

if (Mas[ i ]==obr )

{

Yes=true;

break;

}

else i=i+1;

}

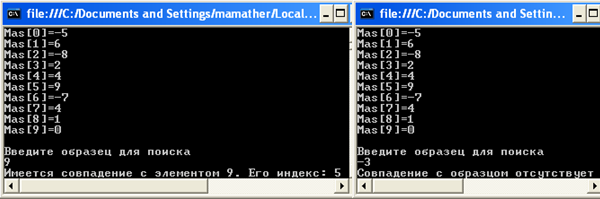
while ( i<10 | Yes == true);

if (Yes == true) Console.WriteLine("Имеется

совпадение с элементом {0}. Его индекс: {1}", Mas[i], i );

else Console.WriteLine("Совпадение с образцом отсутствует ");

Console.ReadKey();

}

**Сортировка массива**

Под ***сортировкой массива*** подразумевается процесс перестановки элементов массива, с целью размещения элементов массива в определенном порядке.

Например, для целых чисел А после сортировки по возрастанию должно выполняться условие:

A[1] ≤ A[2] ≤ A[3] ≤ . . . ≤ A[size], где size – верхний индекс

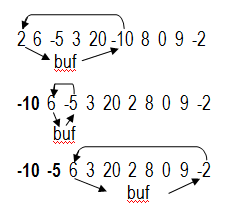
Алгоритм сортировки:

1. Просмотреть массив от 1 элемента, найти min элемент и поместить его на место 1 элемента, а 1-й на место min.

2. Просмотреть массив от 2 элемента, найти min элемент и поместить его на место 2 элемента, а 2-й на место min

3. И так далее до последнего элемента.

Элементы массива А

A[ 1 ]= 2

A[ 2 ]= 6

A[ 3 ]= -5

A[ 4 ]= 3

A[ 5 ]= 20

A[ 6 ]= -10

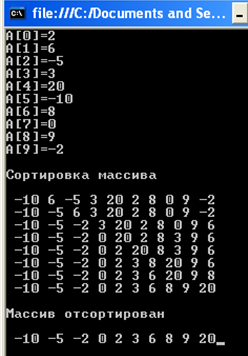
A[ 7 ]= 8

A[ 8 ]= 0

A[ 9 ]= 9

A[ 10 ]= -2

Сортировка массива

-10 6 -5 3 20 2 8 0 9 -2

-10 -5 6 3 20 2 8 0 9 -2

-10 -5 -2 3 20 2 8 0 9 6

-10 -5 -2 0 20 2 8 3 9 6

-10 -5 -2 0 2 20 8 3 9 6

-10 -5 -2 0 2 3 8 20 9 6

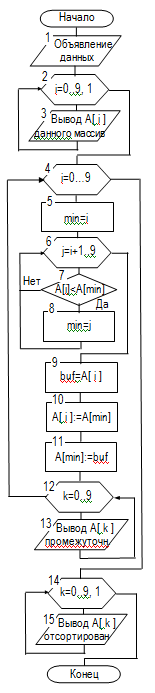
-10 -5 -2 0 2 3 6 20 9 8

-10 -5 -2 0 2 3 6 8 9 20

-10 -5 -2 0 2 3 6 8 9 20

Отсортированный массив

-10 -5 -2 0 2 3 6 8 9 20



**Пример 3.** Отсортировать массив целых чисел по возрастанию.

static void Main(string[] args)

{

int[] A = new int[10]{2, 6, -5, 3, 20, -10, 8, 0, 9, -2};

// одномерный массив А из 10 целых чисел

int min; // минимальный элемент

int buf; // буфер для обмена элементами

int i,j,k; //счетчики цикла

for (i = 0; i < 10; i++)

Console.WriteLine("A[" + i + "]=" + A[i]);

Console.WriteLine();

Console.WriteLine("Сортировка массива");

Console.WriteLine();

for (i = 0; i < 9 ; i++)

{

min=i;

for (j = i+1; j < 10; j++)

if (A[j]<A[min]) min=j;

buf=A[i];

A[i]=A[min];

A[min]=buf;

for (k = 0; k < 10; k++)

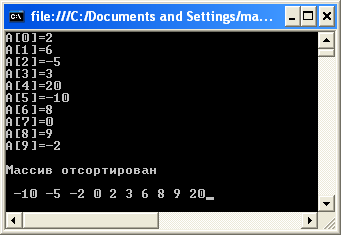
Console.Write(" " + A[k]);

Console.WriteLine();

}

Console.WriteLine();

Console.WriteLine("Массив отсортирован ");

Console.WriteLine();

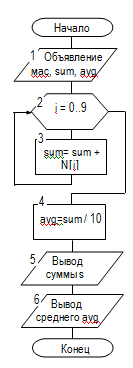
for (k = 0; k < 10; k++)

Console.Write(" " + A[k]);

Console.ReadKey();

}

**Расчет статистических показателей**

****

**Пример 4.** Рассчитать суммарное и среднее значение элементов заданного массива.

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

// одномерный массив А из 10 целых чисел

int sum=0; // суммарное значение

double avg; // среднее значение

for (int i = 0; i < 10; i++)

sum = sum + N[i];

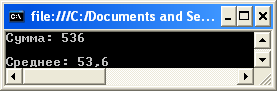
avg=(double)sum/10;

Console.WriteLine("Сумма: "+ sum);

Console.WriteLine();

Console.WriteLine("Среднее: "+ avg);

Console.ReadKey();



int[,] N = {{-99,10,50,18},{78,-23,63,9},{87,49,-55,10},{16,8,95,-16}};

// двумерный массив N из 4x4 целых чисел

int c = 0; // счетчик для подсчета количества элементов

Console.WriteLine("Матрица N[4,4] целых чисел");

Console.WriteLine(); // Вывод матрицы

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

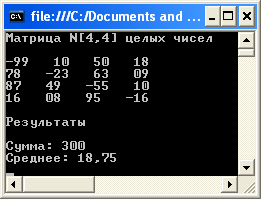
Console.Write(N[i, j].ToString("D2") + " "); // D2 – два десятичных элемента

Console.WriteLine();

}

Console.WriteLine();

int sum = 0; // суммарное значение

double avg; // среднее значение

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

{

sum = sum + N[i,j];

c=c+1;

}

avg = (double)sum / с;

Console.WriteLine("Результаты");

Console.WriteLine();

Console.WriteLine("Сумма: " + sum);

Console.WriteLine("Среднее: " + avg);

Console.ReadKey(); // Пауза

**Использование генератора случайных чисел**

Иногда в качестве элементов массива используют случайные числа, которые можно получить с помощью функции **Random(),** которая генерирует случайное число в диапазоне 0..1.

Перед первым обращением к функции **Random()** необходимо вызовом процедуры **Random** инициализировать программный генератор случайных чисел.

**Пример 5.** Ввод одномерного массива A с использованием программного генератора случайных чисел.

static void Main(string[] args)

{

int n = 0, N=10; // Переменные типа int

Console.WriteLine();

Random r = new Random(); // Для случайных чисел

int[] A = new int[N]; // Массив типа int

Console.WriteLine("Создан массив A случайных целых чисел");

Console.WriteLine();

Console.WriteLine ("Номер n" + " Значение A[n]");

for (n = 0; n < N; n++)

{

A[n] = r.Next(0, 100); // Генерация случайного числа в диапазоне от 0 до 99

Console.WriteLine(" {0} {1,10}", n, A[n] );

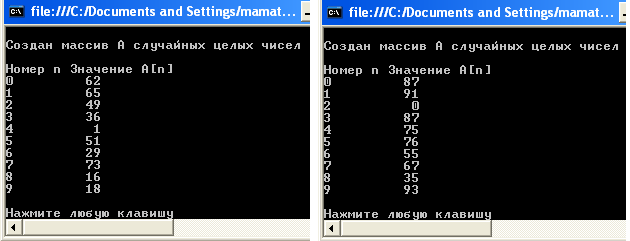
}

Console.WriteLine();

Console.WriteLine("Нажмите любую клавишу");

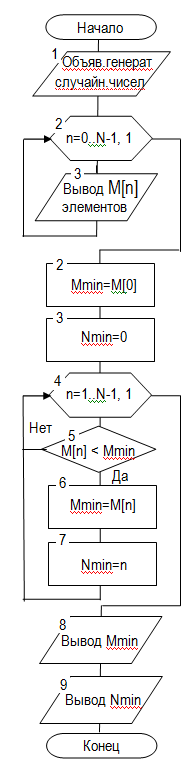
Console.ReadKey(); // Пауза

}



Как видно генератор выдает числа в диапазоне от 0 до 99

**Пример 6**. Вычислить наименьший элемент Mmin массива M и его порядковый номер Nmin. Размерность массива N=10. В программе элементы массива создаются генератором случайных чисел.



int n = 0, N = 10, Nmin = 0, Mmin = 0; // Переменные типа int

Console.WriteLine();

Random r = new Random(); // Для случайных чисел

int[] M = new int[N]; // Массив типа int

Console.WriteLine("Создан массив M случайных целых чисел");

Console.WriteLine();

Console.Write("Номер n" + " Значение M[n]");

Console.WriteLine();

for (n = 0; n < N; n++)

{

M[n] = r.Next(0, 100); // Генерация случайного числа

Console.WriteLine(" {0} {1,10}", n, M[n]);

}

Mmin = M[0]; // Предположение

Nmin = 0;

for (n = 1; n < N; n++)

if (M[n] < Mmin) // Обнаружение минимума

{

Mmin = M[n];

Nmin = n;

}

Console.WriteLine();

Console.WriteLine("Результаты");

Console.WriteLine();

Console.Write("Nmin="); // Вывод Nin

Console.WriteLine(Nmin.ToString());

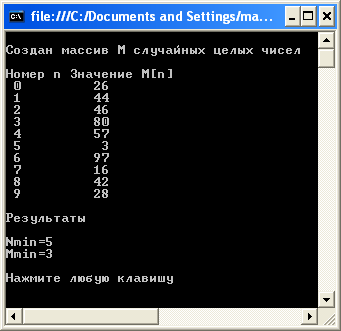
Console.Write("Mmin="); // Вывод Mmin

Console.WriteLine(Mmin.ToString());

Console.WriteLine();

Console.WriteLine("Нажмите любую клавишу");

Console.ReadKey(); // Пауза



**Пример 7.** Вывести квадратную матрицу X размером RxC в виде таблицы. Найти максимальное значение элемента и его координаты: номер строки R и номер столбца C.

{

int r=0, R=0, Rmax=0, c=0, C=0, Cmax=0, Mmax=0; // Переменные типа int

Console.WriteLine("Введите число строк R и столбцов матрицы");

Console.Write("Число строк R="); // Введите R

R = Convert.ToInt32(Console.ReadLine());

Console.Write("Число столбцов C="); // Введите C

C = Convert.ToInt32(Console.ReadLine());

Console.WriteLine();

Random rnd = new Random(); // Для случайных чисел

int[ , ] M = new int[R, C]; // Матрица M[R,C] типа int

for (r = 0; r < R; r++) // Внешний цикл по строкам

{

for (c = 0; c < C; c++) // Внутренний цикл по столбцам

{

int Mm = rnd.Next(0, 100); // Генерация случайного числа от 0 до 99

M[r,c] = Mm;

if (M[r,c] > Mmax) // Обнаружение максимума

{

Mmax = M[r,c]; // Значение максимума

Rmax = r; // Номер строки

Cmax = c; // Номер столбца

}

}

} // Вывод матрицы

Console.WriteLine("Создана матрица M[R,C] случайных целых чисел");

Console.WriteLine();

for (r = 0; r < R; r++)

{

for (c = 0; c < C; c++)

{

Console.Write(M[r, c].ToString("D2")+" ");

}

Console.WriteLine();

}

Console.WriteLine();

Console.WriteLine("Результаты");

Console.WriteLine();

Console.Write("Максимальное значение Mmax="); // Вывод Mmax

Console.WriteLine(Mmax.ToString());

Console.Write("Номер строки Rmax="); // Вывод Rmax

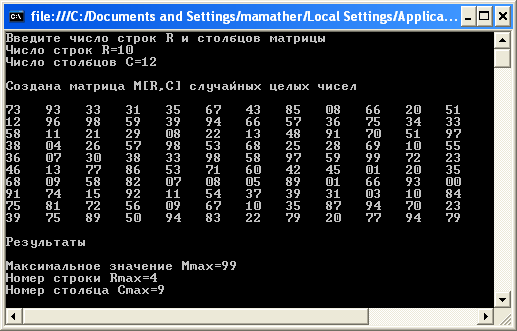
Console.WriteLine(Rmax.ToString());

Console.Write("Номер столбца Cmax="); // Вывод Cmax

Console.WriteLine(Cmax.ToString());

Console.ReadKey(); // Пауза

}



По результату нужно проверить правильность исполнения алгоритма.

**Массивы строк**

Подобно другим типам данных строки могут быть собраны в массивы.

**Пример 8**. В заданном строковом массиве заменить одни слова на другие.

string[ ] str = {"Это ","очень ","простой ","тест." };

Console.WriteLine("Исходный массив: ");

Console.WriteLine();

for (int i=0; i < str.Length; i++)

Console.Write(str[i] + " ");

Console.WriteLine("\n"); //Вывод на печать через клавишу Tab

// Изменяем строку

str[1]="тоже ";

str[3]="тест, не правда ли?";

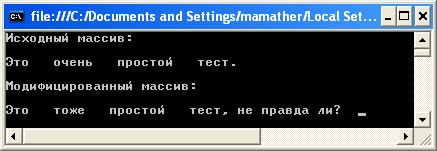
Console.WriteLine("Модифицированный массив: ");

Console.WriteLine();

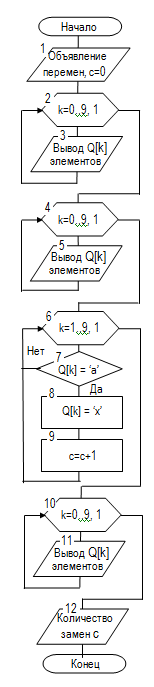
for (int i=0; i < str.Length; i++)

Console.Write(str[i]+" ");

Console.ReadKey(); // Пауза



**Пример 9**. Ввести массив символов и заменить один из символов, например 'a' на другой, например 'x'. Подсчитать количество замен.



char[] Q = new char[10];

char a, x; int c=0; // Количество замен

Console.WriteLine("Ввод символов в массив Q");

for (int k = 0; k < 10; k++)

{

Console.Write("Введите {0}", k);

Console.WriteLine(" элемент массива Q");

Q[k] = Convert.ToChar(Console.ReadLine());

}

Console.WriteLine();

Console.WriteLine("Исходный массив Q: ");

foreach (char k in (char[])Q)

Console.Write("{0} ", k); //вывод элементов массива Q

Console.WriteLine();

for (int k=0; k < 10; k++)

if (Q[k] == 'a') // символ, который надо найти

{

Q[k] = 'x'; // символ, на который надо заменить

c = c + 1; // количество замен

}

Console.WriteLine();

Console.WriteLine("Модифицированный массив Q: ");

foreach (char k in (char[])Q)

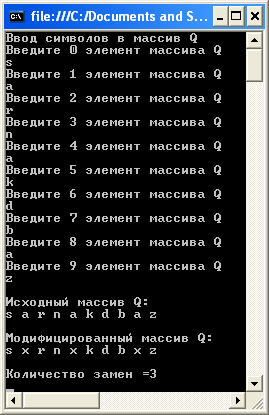
Console.Write("{0} ", k); //вывод элементов массива S

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Количество замен ={0} ", c);

Console.ReadKey(); // Пауза



**Итоги лекции**. В лекции были представлены примеры решения массивов, как одномерных, так и двух мерных. Было дано определение массива, изучены способы задания, вывод. Изучены форматы объявления массива как одномерного, так и двухмерного, инструкция тела. Использование генератора чисел при задании массива.

**Контрольные вопросы**

1. Что такое массив?
2. Описание типа - массив.
3. Какие операторы языка можно использовать для описания массивов?
4. Особенности организации цикла при обработке массивов?
5. Особенности программирования при обработке массивов?
6. Особенность ввода и вывода массивов?
7. Отличие задания одномерного массива от двухмерного.
8. Как создать двухмерный массив случайных целых чисел?
9. Как создать двухмерный массив вещественных чисел?
10. Каков формат объявления двумерного массива?
11. С чего начинается индексация массивов?

# Лекция 8

# Символы и строки

**Цель лекции.** Рассмотреть понятие символа и строки, их различия. Как они задаются в программе. Для каких операций они применяются.

ПК может обрабатывать не только числовую информацию, но и символьную. Символьная информация может быть представлена:

* Отдельными символами
* Строками

## 8.1 Символы

Для хранения и обработки отдельных символов используются переменные типа **Char**.

Тип представляет собой набор символов в кодировке **Unicode**, в котором каждый символ кодируется 16-разрядным двоичным числом (2 байтами).

К символам относятся:

* Буквы русского и латинского алфавитов;
* Цифры;
* Знаки препинания;
* Специальные символы (пробел, <Enter>-“новая строка”);

Объявление символьных данных:

Char Имя переменной; //переменная без инициализации

Char Имя переменной = 'значение'; // переменная с инициализацией

Const char Имя переменной = 'значение'; // константа с инициализацией

**Пример1**:

char sim; // Символьная переменная без инициализации

char q = ' \* '; // Символьная переменная с инициализацией

const char s= ' х '; // Символьная константа

Символьная переменная может получить значение в программе в результате выполнения операции присвоения.

**Пример2:**

char C, S ; // переменные без инициализации

. . . . . . . .

С = ' \* ' ; // переменной С присваивается значение \*

S = ' a '; // переменной S присваивается значение a

. . . . . . . .

**Пример 3.** Выполняет программу диалога: “Вы хотите научиться программировать?”

static void Main(string[] args)

{

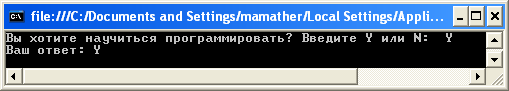
char s;

Console.Write("Вы хотите научиться программировать? Введите Y или N: ");

s = Convert.ToChar(Console.ReadLine());

Console.WriteLine("Ваш ответ: "+ s);

Console.ReadKey();

 }

## 8.2 Строки

Последовательность символов произвольной длины называется строкой. Для обработки и хранения символов можно использовать массив символов или переменные типа **String**.

Объявление строковых данных:

String Имя переменной; //переменная без инициализации

String Имя переменной =''значение''; //переменная с инициализацией

Const String Имя переменной = ''значение''; // константа с инициализацией

**Пример 4**:

string s; // Строковая переменная без инициализации

string s1 = "Привет"; // Строковая переменная с инициализацией

const string s2 = "Я студент"; // Строковая константа

Строковая переменная может получить значение в программе в результате выполнения операции присвоения.

**Пример 5:**

string s1, s2 ; // переменные без инициализации

. . . . . . . .

s1 = "Я студент"; // переменной s1 присваивается значение "Я студент"

s2= "Привет от меня"; // переменной s2 присваивается значение "Привет от меня"

. . . . . . . .

**Пример 6.** Выполняет программу диалога: “Вы хотите научиться программировать?”

static void Main(string[] args)

{

string s;

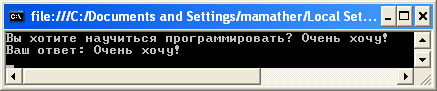
Console.Write("Вы хотите научиться программировать? ");

s = Convert.ToString(Console.ReadLine());

Console.WriteLine("Ваш ответ: " + s);

Console.ReadKey();

}



Результат выполнения строкового выражения является строка символов.

Для строк применимы операции:

* конкатенация (добавление к первой строке второй) //Сцепление строк
* сравнение. // По длине строки

**Пример 7**. Объединение (конкатенация) двух строк.

static void Main(string[] args)

{

string c, h, ch; char q = '!';

Console.Write("Введите 1 строку ");

c = " C# ";

Console.WriteLine(c);

Console.Write("Введите 2 строку ");

h = " Express ";

Console.WriteLine(h);

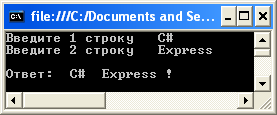
ch = c + h + q;

Console.WriteLine();

Console.WriteLine("Ответ: " + ch);

Console.ReadKey();

}



**Пример 8**. Сравнить содержимое двух строк

static void Main(string[] args)

{

string s1, s2;

Console.Write("Введите 1 строку ");

s1 = Convert.ToString(Console.ReadLine());

Console.WriteLine();

Console.Write("Введите 2 строку ");

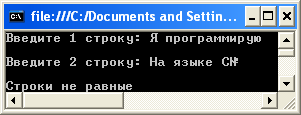
s2 = Convert.ToString(Console.ReadLine());

Console.WriteLine();

if (s1 == s2) Console.WriteLine("Строки равные");

else Console.WriteLine("Строки не равные");

Console.ReadKey();

 }

Переменную типа **String** можно сравнивать с другой переменной типа **String**.

Строки сравниваются посимвольно, начиная с 1 символа:

* Если все символы сравниваемых строк одинаковые, то такие строки считаются равными, в противном случае – не равными. Для равенства двух строк применяют операторы == или !=.
* Операторы отношения >, <, >=, <= c переменными типа String не работают, для их использования необходимо преобразовать строковую переменную в длину строки. Длина строки вычисляется с помощью метода s.Length;
* Получить доступ к отдельному символу строки можно, указав его номер в квадратных скобках после имени переменной (строки).

Например, s[ i ], где i – номер символа строки s.

**Пример 9.** Сравнить содержимое двух строк по их длине:

string s1, s2;

Console.Write("Введите 1 строку: ");

s1 = Convert.ToString(Console.ReadLine());

Console.WriteLine();

Console.Write("Введите 2 строку: ");

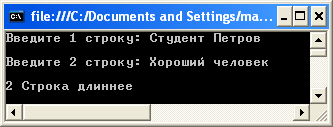
s2 = Convert.ToString(Console.ReadLine());

Console.WriteLine();

if (s1.Length > s2.Length) Console.WriteLine("1 Строка длиннее");

else if (s2.Length > s1.Length) Console.WriteLine("2 Строка длиннее");

else Console.WriteLine("Строки равные");

 Console.ReadKey();

## 8.3 Методы обработки строк

Таблица 8.1

Наиболее частые методы обработки строк

|  |  |
| --- | --- |
| Наименование | Выполняемые функции |
| string ToLower() | Возвращает строчную версию вызывающей строки (нижний регистр) |
| string ToUpper() | Возвращает прописную версию вызывающей строки (верхний регистр) |
| string Copy(string S) | Возвращает копию строки S |
| String length(S) | Возвращает (количество элементов) длину строки S |
| Parse | Преобразует строку в символ. Строка должна состоять из одного символа, иначе возникает ошибка |

**Пример 10**. Вывести прописную и строчную версии строки:

static void Main(string[] args)

{

String s1, s11, s12;

Console.Write("Введите строку: ");

s1 = Convert.ToString(Console.ReadLine());

Console.WriteLine();

s11 = s1.ToLower();

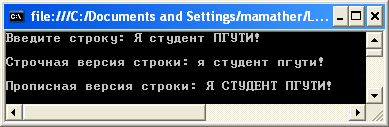
s12 = s1.ToUpper();

Console.WriteLine("Строчная версия строки: " + s11);

Console.WriteLine();

Console.WriteLine("Прописная версия строки: " + s12);

Console.ReadKey();

}

**Пример 11**. Вывести длину строки и ее копию.

static void Main(string[] args)

{

string s1, s2;

s1 = "В программировании используется язык C#";

Console.WriteLine();

Console.WriteLine("s1: " + s1);

s2 = string.Copy(s1);

Console.WriteLine();

Console.WriteLine("s2: " + s2);

Console.WriteLine();

Console.WriteLine("Длина строки s1: " + s1.Length);

Console.ReadKey();

}



## 8.4. Строковые функции и процедуры

В C# имеется символьный класс char, используемый двухбайтовую кодировку Unicode представления символов. Для этого типа в языке определены символьные константы. Константу можно задавать:

* символом, заключенным в одинарные кавычки;
* escape-последовательностью;
* Unicode-последовательностью, задающей Unicode код символа.

Примеры объявления символьных переменных и работа с ними:

static void Main(string[] args)

{

char ch1 = 'A', ch2 = '\x59', ch3 = '\u0058'; //буква A,

//буква Y Escape-последовательность \x59,буква X кодировка Юникода \u0058

char ch = new Char(); //символьный класс, используемый двухбайтовую кодировку

int code; string s;

ch = ch1; // ch объявляется в объектном стиле

code = ch; //преобразование символьного типа в тип int

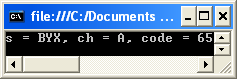
ch1 = (char)(code + 1); //формирование кода буквы B

//преобразование символьного типа в строку

s = ch1.ToString() + ch2.ToString() + ch3.ToString();

Console.WriteLine("s = {0}, ch = {1}, code = {2}", s, ch, code);

Console.ReadKey();

}

Массив символов можно преобразовать в строку методом ToString(), но можно выполнить и обратное преобразование. В C# имеется строковый класс string, задающий строки переменной длины. У класса string есть метод ToCharArray, преобразующий строку в массив символов. Строку можно задавать из:

* - символа, повторенного заданное число раз;
* - массива символов char[];
* - части массива символов.

Примеры объявления строковых переменных и работа с ними:

static void Main(string[] args)

{

string world = "Мир"; //объявление переменной world

string sssss = new string('s', 5); //строковый класс, с буквой s повторяющейся 5 раз

char[ ] yes = "Yes".ToCharArray(); //объявление символьного класса для строки Yes

string stryes = new string(yes); //объявление строкового класса stryes

string strye = new string(yes, 0, 2); //объявление строкового класса для символьного

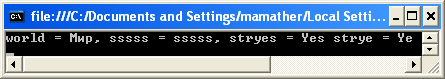
// массива yes с начальной позицией знака = 0 и длиной = 2 символа

Console.WriteLine("world ={0}, sssss ={1}, stryes ={2}"+" strye ={3}",

world, sssss, stryes, strye);

Console.ReadKey();

}



**Краткие итоги.** В лекции были рассмотрены определения символа и строки. Изучены методы обработки строк. Задание символов и строк. Даны примеры объявления строковых переменных и сиволов.

**Контрольные вопросы.**

1. Какие переменные используются для хранения и обработки отдельных символов?
2. Что относится к символам?
3. Как объявляются символы, приведите примеры.
4. Дайте определение строки.
5. Как объявляются строковые данные?
6. Чем отличается переменная с инициализацией, от переменной без инициализации?
7. Назовите часто встречающиеся методы обработки строк.

# Лекция 9

# Классы, объекты и методы. Подпрограммы

**Цель лекции.** Изучить классы, объекты и методы, их характеристики, определение, свойства и как они представляются. Рассмотреть подпрограммы их преимущества, дать определение.

## 9.1 Введение в классы, объекты и методы

Программирование в C# построено на классах. У класса две различные роли:

* модуля и
* типа данных.

***Класс –*** это некий шаблон (модуль), который определяет форму объекта. Или множество объектов, связанных общностью структуры и поведения. Класс определяет, как должен быть построен объект. Однотипные объекты могут *объединяться* в ***классы (группы)***.

Конкретный объект, имеющий структуру этого класса, называется *экземпляром* класса.

Например, объект кнопка button1 – это экземпляр класса кнопок Button. Сам класс определяется общими свойствами, которые имеют все экземпляры этого класса.

Модульность построения – основное свойство программных систем. Система, построенная по модульному принципу, состоит из классов, являющихся основным видов модуля. Можно построить монолитную систему, состоящую из одного модуля, решающая ту же задачу, что и система, состоящая из нескольких простых модулей. Большую систему, создаваемую коллективом разработчиков, без разделения системы на модули построить не удается. Поэтому модульность построения – основное средство борьбы со сложностью системы.

***Класс*** – это особый тип записи, имеющий в своем составе поля, свойства и методы обработки. Каждый модуль имеет содержательную начинку. Класс становится. Модулем и имеет определенное назначение. В основе класса лежит абстрактный тип данных.

*Поля класса* служат для хранения информации об объекте. Одной стороны, свойства можно использовать как поля, например, присваивать им значения с помощью оператора присваивания, с другой стороны, внутри класса доступ к значениям свойств выполняется методами класса.

*Методами* называются процедуры и функции, предназначенные для обработки внутренних данных объекта данного класса (полей). Объект может обладать набором заранее встроенных методов обработки, созданных пользователем, которые выполняются при наступлении заранее определенных событий, например: нажатие кнопки мыши, определенной клавиши, выбор пункта меню и т.п..

*Свойства*– *характеристики* объекта, его параметры. Объект представляется с помощью присущих ему свойств.

Например:

ОБЪЕКТ\_1 (свойство-1, свойство-2, . . . свойство-k).

Свойства объектов различных классов могут пересекаться. Например:

ОБЪЕКТ\_А (. . свойство-n, свойство-m, . . . свойство-k).

ОБЪЕКТ\_B (свойство-1,. . . свойство-n, . . . свойство-k).

## 9.2 Синтаксис класса

**class** <Имя класса> [: список родителей]

{

//объявление переменных экземпляра

доступ тип переменная1;

доступ тип переменная2;

. . . . . . . . . .

доступ тип переменнаяN;

//объявление методов

доступ тип\_возврата метод1(параметры)

{

// тело метода1

}

доступ тип\_возврата метод2(параметры)

{

// тело метода2

}

. . . . . . . .

доступ тип\_возврата методN(параметры)

{

// тело методаN

}

}

Доступ – означает спецификатор доступа, принимающий одно из четырех значений*: public, private, protected, internal*. Он определяет, как к этому члену можно получить доступ. Члены класса могут быть закрытыми в рамках класса или более доступными.

Например, спецификатор public – общедоступные описания означает, что класс открыт и члены класса могут использовать все составные части программного кода .

Спецификаторы private или protected – означают, что члены класса закрыты для пользователя.

Правильно определенный класс должен содержать логически связанные данные.

Все переменные, объявленные на уровне класса, являются полями класса.

Поля задают представления абстракции данных, которую реализует класс.

Поля характеризуют свойства объектов класса.

Описаниями класса являются объявления свойств, методов и событий.

Метод представляет собой подпрограмму, являющуюся элементом класса. Методы выполняют действия над объектами класса. Все объекты одного класса имеют один и тот же набор методов.

Классы, которые мы использовали до сих пор, содержат один метод Main().

static void Main(string[] args)

{

}

## 9.3. Основные понятия подпрограммы

**Подпрограмма** – логически законченная часть программы, которую по имени можно вызывать в разные места программы неограниченное число раз.

Она решает часть общей задачи. У нее есть имя, которое используется при вызове подпрограммы в основную программу.

Подпрограммы бывают 2 видов:

* процедуры и
* функции.

**Функция** отличается от процедуры двумя особенностями:

* всегда вычисляет некоторое (одно) значение, возвращаемое в качестве результата функции,
* функция вызывается в выражениях.

**Процедура** в C# имеет свои особенности:

* возвращает формальный результат void, который указывает на отсутствие результата, возвращаемого при вызове процедуры,
* вызов процедуры является оператором языка,
* имеет входные и выходные аргументы, причем выходных аргументов – ее результатов – может быть достаточно много.

Обычно метод реализуется в виде функции тогда, когда он имеет один выходной аргумент, рассматриваемый как результат вычисления функции (например, sin(0.5)).

В других случаях метод реализуют в виде процедуры.

Программа без структурных элементов называется монолитной. Минимальный элемент такой программы – оператор (инструкция). Она сложна в разработке, отладке и сопровождении.

Структурированная программа называется модульной. Она содержит более крупные компоненты – подпрограммы. Модульная программа проще создается, более понятна.

Программа на языке C# строится из модулей, роль которых играют классы, но каждый из этих модулей имеет содержательную начинку.

Процедуры и функции связываются с классом, они обеспечивают требуемую функциональность класса и называются методами класса. Существуют только как методы некоторого класса и не существуют вне класса.

Синтаксис объявления метода позволяет однозначно определить, чем является метод – процедурой или функцией.

Работа с подпрограммой включает 2 этапа:

* описание метода;
* вызов метода.

Преимущества подпрограммы:

1. в программе нет дублирования кода;

2. повышается надежность программы;

3. улучшается «читаемость» программы;

4. облегчается процесс отладки.

Различают подпрограммы:

* без параметров (),
* с параметрами (params).

## 9.4Сравнение вариантов

Рассмотрим 2 варианта решения одной и той же задачи:

Вывод текстовых блоков со вставкой стандартного разделителя из трех строк из набора символов “ \* ” в конце.

***1 –й вариант (Монолитная программа)***

static void Main(string[] args)

{

Console.WriteLine("Текст и разделители");

Console.WriteLine();

Console.WriteLine("Текст 1"); // Вывод 1 текста

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 20; j++)

Console.Write("\*");

Console.WriteLine(); // Вывод строк разделителя

}

Console.WriteLine();

Console.WriteLine("Текст 2"); // Вывод 2 текста

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 20; j++)

Console.Write("\*");

Console.WriteLine(); // Вывод строк разделителя

}

Console.WriteLine();

Console.WriteLine("Текст 3"); // Вывод 3 текста

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 20; j++)

Console.Write("\*");

Console.WriteLine(); // Вывод строк разделителя

 }

Console.ReadLine();

}

***2 – й вариант (Модульная программа)***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static void DrawStr() // Процедура “Рисовать строки” без параметров

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 20; j++)

Console.Write("\*");

Console.WriteLine();// Вывод строк разделителя

}

}

static void Main(string[] args)

{

Console.WriteLine("Текст и разделители");

Console.WriteLine();

Console.WriteLine("Текст 1"); // Вывод 1 текста

**DrawStr();** // Вызов процедуры “Рисовать строки”

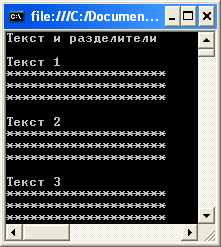
Console.WriteLine();

Console.WriteLine("Текст 2"); // Вывод 2 текста

**DrawStr();** // Вызов процедуры “Рисовать строки”

Console.WriteLine();

Console.WriteLine("Текст 3"); // Вывод 3 текста

** DrawStr();** // Вызов процедуры “Рисовать строки”

Console.ReadLine();

}

}

}

## 9.5 Описание методов (процедур и функций)

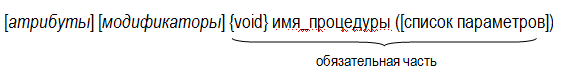
Синтаксически в описании метода различают две части:

* Описание заголовка
* Описание тела метода.

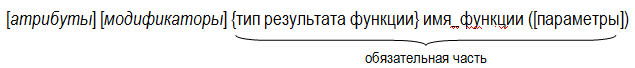
ЗАГОЛОВОК\_МЕТОДА

ТЕЛО\_МЕТОДА

Синтаксис заголовка процедуры:



Синтаксис заголовка функции



имя\_метода ([список параметров]) составляют сигнатуру метода

– обязательная часть в заголовке метода,

[ ] показывают, что атрибуты и модификаторы могут быть опущены при описании метода.

***Модификаторы:***

static – представляет стандартный поток для консольного приложения,

public – показывает, что метод **открыт** и доступен для вызова пользователем и потомками класса,

private – говорит, что метод **закрыт** и предназначен для внутреннего использования в классе и доступен для вызова только в теле методов самого класса.

Если модификатор опущен, то метод является закрытым для пользователя и класса.

Примеры описания методов:

void A() {…};

int B() {…};

public void C() {…};

Методы А и В являются закрытыми, а метод С – открыт.

Методы А и С реализованы процедурами, а метод B – функцией, возвращающей целое значение (int), {…} – тело метода.

## 9.6 Аргументы и параметры

*Параметры* – данные, с которыми работает подпрограмма. Это внутренние данные для подпрограммы и перечисляются в ее заголовке с указанием типов. Список может содержать фиксированное число аргументов, разделяемых запятой – для подпрограммы с параметрами или быть пустым – для подпрограммы без параметров.

Синтаксис объявления двух параметров:

[ *ref* | *out* | *params*] тип\_аргумента1 имя\_аргумента1, тип\_аргумента2 имя\_аргумента2

Например: (out long p1, int p2);

Параметры связаны с аргументами.

*Аргументы* – данные, передаваемые в подпрограмму или возвращаемые из нее. Это внешние для подпрограммы данные, с которыми имеет дело вызывающая часть программы. В подпрограмме им соответствуют параметры. Указываются в списке аргументов при обращении к подпрограмме и являются фактическими параметрами.

*Синонимы:*

Параметры = формальные параметры, условно Р.

Аргументы = фактические аргументы, условно А.

Можно для фиксированного числа формальных параметров передать методу произвольное число фактических аргументов. Для этого в списке формальных параметров необходимо задать ключевое слово params. Оно может появляться в объявлении лишь последнего аргумента списка, объявленного как массив произвольного типа. При вызове метода этому параметру соответствует произвольное число фактических аргументов.

Для аргументов и параметров надо соблюдать:

* одинаковое количество (An=Pn) ,
* одинаковый порядок следования (A1-P1, A2-P2..)
* совместимость типов (типы одинаковы или неявно преобразованные),
* одинаковый статус аргументов (если параметр объявлен с ключевым словом ref или out, то фактический аргумент должен сопровождаться таким же ключевым словом в точке вызова).

Все формальные параметры разделяются на 3 группы:

* входные параметры – задаются без ключевого слова (или с параметром ref); передают информацию методу и их значения в теле метода только читаются;
* выходные параметры – снабжаются ключевым словом out; представляют собой результаты метода, они получают значения в ходе работы метода; в теле метода должен присутствовать оператор присваивания, задающий значение этому параметру;
* обновляемые параметры – снабжаются ключевым словом ref; выполняют обе функции. Их значения используются в ходе вычислений и обновляются в результате работы метода.

Например:

// Процедура Cube

void Cube(out long p2, int p1)

{

p2 = (long)Math.Pow(p1, 3);

Console.WriteLine(“Метод 1”);

}

// Процедура CubeM

void CubeM(out double p2, params double[ ] p)

{

p2 =0; for (int I = 0; I < p.lehgth; i++)

p2 + = Math.Pow(p[ i ], 3);

Console.WriteLine(“Метод 2”);

}

// Функция F

int F(ref int a)

{

retun (a++)

}

Все методы закрыты, поскольку объявлены без модификатора доступа. Cube и CubeM – процедуры, а F – функция. Аргумент р2 – является выходным (сопровождается ключевым словом out ), поэтому в теле метода ему присваивается значение. Аргумент функции F является обновляемым (снабжен ключевым словом ref ), в теле функции используется его значение для получения результата функции, но и само значение параметра изменяется в теле функции. Для процедуры CubeM используется ключевое слово params для самого последнего аргумента р. Т.к. этому параметру будет соответствовать несколько фактических аргументов, число которых может быть произвольным.

**Области действия имен**

Структура подпрограммы идентична структуре программы.

Имена, объявленные в главной программе являются *глобальными.* Они доступны во всех внутренних точках, в том числе и в подпрограммах.

Имена, объявленные в методе, являются *локальными.* Они доступны во всех внутренних точках подпрограммы.

*Локальное* имя во *внешней* программе недоступно. Память под них выделяется автоматически в момент вызова метода.

Если используются *одинаковые* локальное и глобальное имя, то внутри метода локальное имя *блокирует* глобальное.

## 9.7 Тело метода (процедуры и функции)

Синтаксически тело метода является блоком, который представляет собой последовательность инструкций и описание переменных заключенных в {…}.

В теле функции в блоке должен быть хотя бы один оператор, возвращающий значение функции в формате

return <выражение>

Переменные, описанные в блоке, считаются локальными в этом блоке.

**Вызов метода. Синтаксис.**

Функция может быть вызвана в выражении, а процедура – как оператор тела блока. Сам вызов метода, независимо от того, процедура это или функция, имеет один и тот же синтаксис:

Имя\_метода([список фактических аргументов]);

Если это оператор, то вызов завершается точкой с запятой.

Формальный параметр представляет собой идентификатор параметра, а фактический аргумент – «выражение», значительно более сложную синтаксическую конструкцию:

[ ref | out ] выражение

В момент вызова метода происходит:

1. вычисление фактических аргументов, которые являются выражением,

2. в точке вызова создается блок, в котором происходит замена имен параметров фактическими аргументами.

**Функция**

**Пример 1.** Написать функцию, проверяющую, является ли целое число, введенное с клавиатуры четным. Программа должна использовать эту функцию, проверять является ли число четным или нет, и выводить соответствующее сообщение.

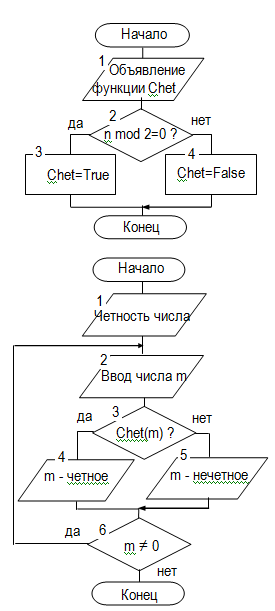
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static bool Chet(int n) // Процедура “Четность”

{

if (n % 2 == 0) return true;

else return false;

}

static void Main(string[] args)

{

int m;

Console.WriteLine("Определение четности аргумента");

Console.WriteLine();

do

{

Console.Write("m >> ");

m = Convert.ToInt32(Console.ReadLine());

if (Chet(m)) Console.WriteLine("Число четное");

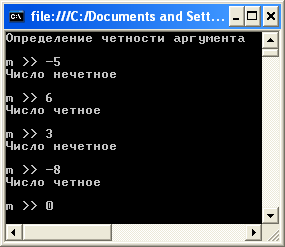
else Console.WriteLine("Число нечетное");

Console.WriteLine();

}

while (m!=0);

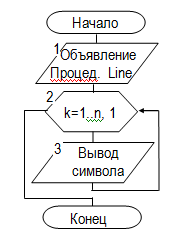
Console.ReadLine();

 }

}

}

**Процедура**

**Пример 2**. Вывести таблицу квадратных корней.

Для оформления таблицы использовать процедуру Line.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

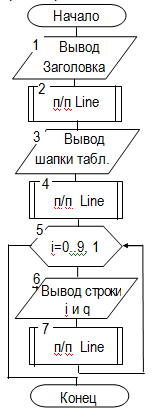
namespace ConsoleApplication1

{

class Program

{

static void Line(int n, char c) // Процедура “Рисовать линию” с параметрами n и с

 {

for (int k = 0; k < n; k++)

Console.Write(c);

Console.WriteLine(); // Вывод строки с символами С

}

static void Main(string[] args)

{

int m = 18;

Console.WriteLine("Таблица квадратных корней");

Console.WriteLine();

Line(m, '='); // Вызов процедуры “Рисовать линию”

Console.WriteLine(" Число | Корень");

Line(m, '='); // Вызов процедуры “Рисовать линию”

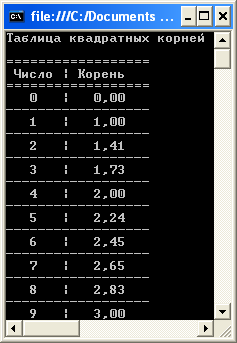
for (int i = 0; i < 10; i++)

{

double q = Math.Sqrt(i);

Console.WriteLine(" {0} | {1,3:f}",i, q);

Line(m, '-'); // Вызов процедуры “Рисовать линию”

 }

Console.ReadLine();

}

}

}

**Краткие итоги.** В лекции были даны определения, что такое классы, объекты, методы. Представлен синтаксис класса, методы представления подпрограммы. Изучены функции и процедуры, относящиеся к подпрограмме, дано их описание.

**Контрольные вопросы.**

1. Дайте определение понятию «класс».
2. Что лежит в основе класса?
3. Каковы свойства и характеристики объекта?
4. Назовите спецификаторы и для чего они применяются.
5. Дайте определение понятиям «функция» и «процедура».
6. Назовите преимущества подпрограммы.
7. Сравните монолитную и модульную программы, назовите их различия.
8. Опишите методы процедур и функций.
9. Дайте определение что такое «параметры» и «Аргументы».
10. Каковы области действия имен?
11. Опишите процедуру вызова метода «Синтаксис»
12. Приведите пример работы функции.
13. Приведите пример работы процедуры.

# Лекция 10

# Исключения

**Цель лекции.** Разобраться что такое исключения, как они проявляются в работе. Изучить обработку исключения, какие обработчики используются.

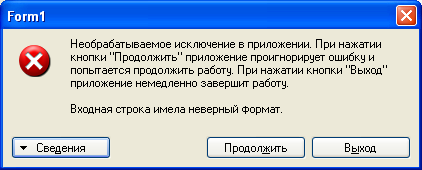
Нарушение в работе программы (не корректное действие) называется **исключением**. Исключение останавливает текущий поток программы и если никакие меры не предпринимаются, программа просто прекращает выполнение. Причиной исключений могут быть ошибки в программе (например, деление числа на ноль) или неожиданный ввод (например, выбор несуществующего файла, или вместо цифры для выполнения арифметических действий – буква). Задачей пользователя является предоставление программе возможности устранить проблемы, не приводя к сбою.

Обработку исключений может выполнять:

* системный обработчик исключений или
* написанный пользователем программный код, который выводит сообщение.

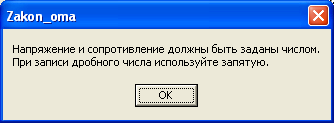
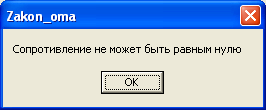
Для обработки исключений используются глобальные и локальные обработчики исключений.

***Глобальные*** обработчики – стандартные предусмотрены операционной системой и вызываются автоматически, если отладчик подключен. Обработчик завершает выполнение программы, выдавая информацию о возникновении исключения. Он не всегда понятен пользователю, не говоря уже о *досрочном завершении программы*.



***Локальные*** обработчики создает сам пользователь в виде сообщений. Программист может перехватить исключительные ситуации и сделать их обработку, после чего продолжится нормальный ход вычислений приложения.

Обработка локальных исключений в C# реализована с помощью ключевых слов, — try, catch и finally — с помощью которых программа обнаруживает исключения, устраняет их и продолжает выполнение. Они способствуют повышению надежности приложений.



## 10.1 Виды исключений:

Исключения имеют типы, являющиеся производными от System.Exception. Некоторые исключения, генерируемые при компиляции:

Таблица 10.1

Типы исключений и их описание

|  |  |
| --- | --- |
| Тип\_Исключения | Описание |
| ArithmeticException | Основной класс исключений, происходящих при выполнении арифметических операций, таких как DivideByZeroException и OverflowException. |
| ArrayTypeMismatchException | Создается, когда тип сохраняемого значения несовместим с фактическим типом массива. |
| DivideByZeroException | Создается при попытке разделить целое число на ноль. |
| IndexOutOfRangeException | Создается при попытке индексирования массива, если индекс меньше нуля или выходит за границы массива. |
| InvalidCastException | Создается, когда неверно выполнено динамическое приведение типов. |
| NullReferenceException | Создается при попытке ссылки на объект, значение которого равно null. |
| OutOfMemoryException | Создается при неудаче попытки выделения памяти с помощью оператора new. Это означает, что память, доступная для среды выполнения, уже исчерпана. |
| OverflowException | Создается при переполнении арифметической операции. |
| StackOverflowException | Создается, когда стековая память переполнена. |

## 10.2 Типы блоков работающих с исключением

Существуют два типа блоков:

**I.** Try **…** Catch // попробовать исключить

**II.** Try **…** Catch … Finally // попробовать исключить в финал

**I. Try, Catch**

Ключевые слова try и catch используются вместе как составной оператор. Для таких блоков характерна вложенность. Если предполагается, что блок кода может вызвать исключение, используют ключевое слово **try** и **catch (параметр)**, чтобы сохранить код, который будет выполнен при возникновении исключения.

Блок, которому предшествует ключевое слово try, называется охраняемым (контролируемым) блоком или **try-блоком**. Блок, которому предшествует конструкция catch (параметр), называется блоком перехватчиком исключения или **catch-блоком.**

При отсутствии блоков try и catch произойдет сбой программы.

Синтаксис инструкции:

try

{

// Блок кода, подлежащий проверки на наличие ошибок

}

catch (<Тип\_Исключения1>)

{

// Код обработчика исключения (<Тип\_Исключения1>)

}

catch (<Тип\_Исключения2>)

{

// Код обработчика исключения (<Тип\_Исключения2>)

}…..

С **try**-блоком может быть связана не одна, а несколько **catch**-инструкций. Какая из них будет выполнена, определит тип исключения. Будет выполнена та **catch**-инструкция, тип исключения которой совпадает с типом сгенерированного исключения (а все остальные будут проигнорированы).

Если исключение не сгенерируется, то **try**-блок завершается нормально, и все его **catch**-инструкции игнорируются. Выполнение программы продолжается с первой инструкции, которая стоит после последней инструкции **catch**. Таким образом, **catch**-инструкция выполняется только в случае, если сгенерировано соответствующее исключение.

**Пример 1**, в котором в результате деления на ноль создается исключение, которое затем перехватывается. В программе генерируется исключение типа **DivideByZeroException** – деление на 0

int x = 0, y = 0; // знаменатель равен 0

try

{

x = 10 / y; // Проверяемая инструкция, возможно деление на 0

Охраняемый блок

Console.WriteLine("Ответ x= "+x);

}

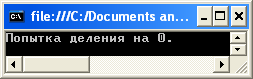
catch (DivideByZeroException) // Обработчик исключения

{

Перехватчик исключения

Console.WriteLine("Попытка деления на 0.");

}



Блоки **try** включают операторы программы, которые могут вызвать исключительную ситуацию, например y=0.

Теперь заменим y=0 на y=2:

{

int x = 0, y = 2; // знаменатель равен, например 2 (отличен от 0)

try

{

x = 10 / y; // Проверяемая инструкция, возможно деление на 0

Console.WriteLine("Ответ x= "+x);

}

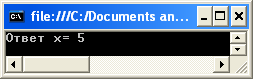
catch (DivideByZeroException) // Обработчик исключения

{

Console.WriteLine("Попытка деления на 0.");

}

}



**Пример 2**, в котором происходит индексация массива и попытка индексировать массив за пределами его границ, вызывает ошибку нарушения диапазона.

static void Main(string[] args)

{

int[] N = new int[4]; // массив для 4-х элементов

try

{

Console.WriteLine("Перед генерированием исключения.");

// связанное с попаданием индекса вне диапазона

for (int i = 0; i < 10; i++) //в цикле индексируется массив от 0 до 9

{

N[i] = i;

Console.WriteLine("N[{0}]:{1}", i, N[i]);

}

Console.WriteLine("Этот текст не отображается");

}

catch (IndexOutOfRangeException)

{

//Перехватываем исключение.

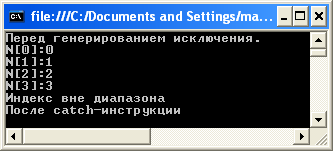
Console.WriteLine("Индекс вне диапазона");

}

Console.WriteLine("После catch-инструкции");

Console.ReadLine();

}



В программе намеренно генерируется исключение типа **IndexOutOfRangeException**, а затем это исключение перехватывается.

В программе объявляется массив для 4-х элементов, а в цикле делается попытка индексировать этот массива от 0 до 9. Как только значение индекса устанавливается равным четырем, генерируется исключение типа IndexOutOfRangeException.

Ключевые аспекты обработки исключений:

1. проверяемый код содержится внутри try-блока,

2. при возникновении исключения выполнение try-блока прекращается, а само исключение перехватывается catch инструкцией,

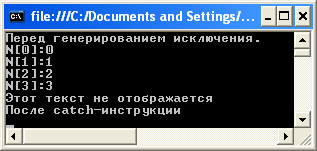
3. catch инструкция не вызывается, а ей передается управление программой.

4. после выполнения catch инструкции программа продолжится со следующей инструкции.

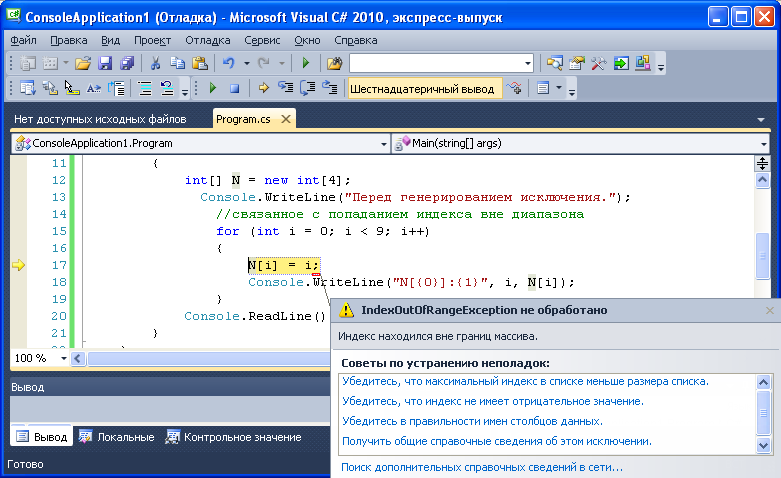
Если try-блоком исключение не сгенерировано, ни одна из catch инструкций не выполняется и управление программой будет передано инструкции, следующей за catch инструкций.

Если заменить в цикле for (int i = 0; i < 10; i++) верхний индекс на

for (int i = 0; i < N.Length; i++), то границы индексирования массива не нарушаются. Поэтому исключение не генерируется и catch-блок не выполняется.



Если не создать локального обработчика исключения, то C# система динамического управления перехватит исключение, сообщит об ошибке и завершит программу.



С try-блоком можно связать несколько catch инструкций. Все catch инструкции должны перехватывать исключения различного типа.

**Пример 3**. Программа перехватывает как ошибку нарушения границ массива, так и ошибку деления на нуль.

static void Main(string[] args)

{

int[] N = {4,8,16,32,64,128,256,512}; // массив из 8 элементов

int[] D = {2, 0, 4, 4, 0, 8}; // массив из 6 элементов

for (int i = 0; i < N.Length; i++) //в цикле по всем элементам

try

{

Console.WriteLine(N[i] + " / "+ D[i] + " = " + N[i]/D[i]);

}

catch (DivideByZeroException)

{

//Перехватываем исключение.

Console.WriteLine("На нуль делить нельзя!");

}

catch (IndexOutOfRangeException)

{

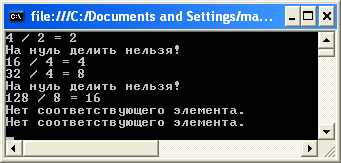
//Перехватываем исключение.

Console.WriteLine("Нет соответствующего элемента.");

}

Console.ReadLine();

}



Каждая **catch** инструкция реагирует только на собственный тип исключения.

**II. Try, Catch, Finally**

Иногда возникает потребность определить программный блок, который должен выполняться по выходу из try/catch блока. Исключение может вызвать ошибку, которая является причиной преждевременного возврата из текущего метода. Удобный путь выхода из этого – блок finally.

Код, содержащийся в блоке finally, выполняется всегда, вне зависимости от возникновения исключения. Чтобы гарантировать возвращение ресурсов, например, убедиться, что файл закрыт, или освободить память от локальных переменных.

Синтаксис инструкции:

class ProgramTryCatchFinally

{

static void Main()

{

try

{

// Блок кода, предназначенный для обработки ошибок

}

catch (<Тип\_Исключения1>)

{

// Обработчик для исключения (<Тип\_Исключения1>).

}

catch (<Тип\_Исключения2>)

{

// Обработчик для исключения (<Тип\_Исключения2>).

}

…..

finally

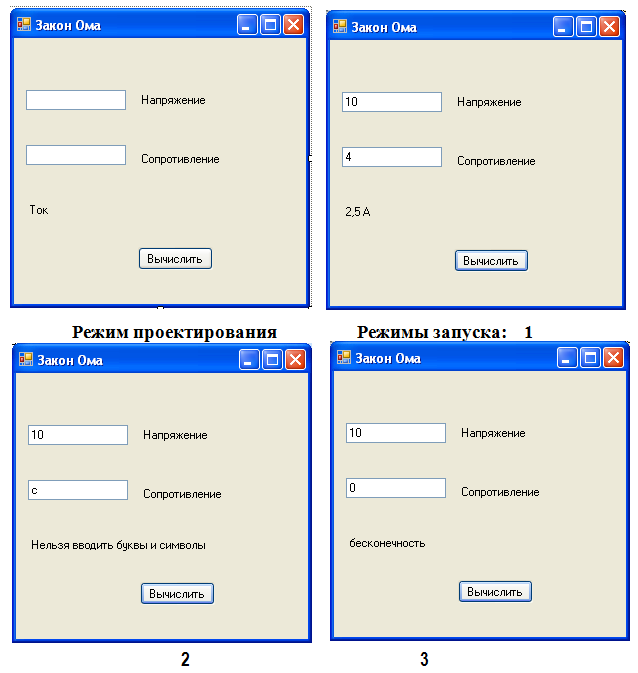
{

// Код завершения обработки исключений.

}

Не зависимо от итога выполнения try/catch блоков, блок finally выполняется обязательно.

**Пример 4.** Используя закон Ома рассчитать величину тока. Проект оконное приложение. Сначала создается интерфейс будущего приложения, а затем обработчик события щелчка по кнопке «Вычислить».



Листинг программы

using System;

using System.Collections.Generic; using System.ComponentModel; using System.Data;

using System.Drawing; using System.Linq; using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

int U = int.Parse(textBox1.Text); // преобразование строкового

int R = int.Parse(textBox2.Text); // отображения в целое

double I = (double)U / R;

label3.Text = I.ToString()+ " A"; // преобразование числа в строку

}

catch

{

label3.Text = "Нельзя вводить буквы и символы";

}

}

}

}

## 10.3 Форматы ввода/вывода

При вводе и выводе информации в языке С# используются функции преобразования представленные в таблице 10.2.

Таблица 10.2

Функции преобразования и их значения

|  |  |
| --- | --- |
| Функция | Значение |
| Convert.ToDouble(s)  Convert.ToInt16(s)  Convert.ToChar(q)  int.Parse( s)  double.Parse(s)  S.ToString()  String.Format(s,x) | Вещественное число, изображением которого является строка s  Целое, изображением которого является строка s  Код символа, изображением которого является символ q  Целое, изображением которого является строка s  Вещественное, изображением которого является строка s  Строка, являющаяся изображением вещественного s  Строка, являющаяся изображением вещественного s в форматном выводе числа. |

Например, под Консольное приложение:

x = Convert.ToDouble(Console.ReadLine()); //вещественный тип

d = Convert.ToInt16(Console.ReadLine()); //целый тип

z = Convert.ToInt32(Console.ReadLine()); //целый тип

ch=Convert.ToChar(Console.ReadLine()); //символьный тип

Console.WriteLine("x={0,3:f} y={1,5:f}", x, y); //форматный вывод переменных х и у,

// где 0 и 1 номера вывода, 3 и 5 – количество выводимых символов,

// f – два символа после запятой

Например, под Оконное приложение

int а = int.Parse(textBox1.Text); //преобразование текста, введенного

// в textBox1 в целое число

double b = double.Parse(textBox1.Text); //преобразование текста, введенного

// в textBox1 в вещественное число

label3.Text = S.ToString()+ " руб."; // преобразование числа S в строку

label1.Text =String.Format(“{0:f 3}”,x); // преобразование числа x в строку c форматным выводом символов в компонент label1 через его свойство Text,

где 0 – индекс переменной, f – мантисса, 3 – количество символов после зпт.

label1.Text =String.Format(“{0,5:f 2}”,x); // преобразование числа x в строку c форматным выводом символов в компонент label1 через его свойство Text,

где 0 – индекс переменной, 5 – количество выводимых символов, f – мантисса, 2 – количество символов после зпт.

**Краткие итоги.** В лекции были рассмотрены исключения, и работа с ними, посредством обработчиков. Приведены примеры работы блоков работающих с исключениями.

**Контрольные вопросы.**

1. Дайте определение понятию «исключение»
2. Как можно выполнить обработку исключений?
3. Перечислите обработчики исключений и их принцип работы.
4. Виды исключений, типы и описания.
5. Блоки, работающие с исключениями, коротко охарактеризуйте их.
6. Ключевые слова try и catch, как они используются в блоке исключения.
7. Приведите пример использования try и catch.
8. Ключевые аспекты обработки исключения при использования try и catch.
9. Программный блок использующий Try, Catch, Finally.
10. Приведите пример использования Try, Catch, Finally.
11. Функции преобразования и их значения.
12. Приведите примеры формата ввода и вывода.

# Лекция 11

# Приложение под ОС Windows

Цель лекции. Изучить приложение ОС Windows, его основные характеристики и принцип работы с ним. Разобрать назначение окон присутствующих в Windows Form.

## 11.1. Основные характеристики приложения Windows Form

При запуске Microsoft Visual C# предоставляет возможность командой Файл \ Создать проект ⇒ создать **приложение Windows Form** – под операционную систему **Windows** с графическим интерфейсом (рис.11.1)

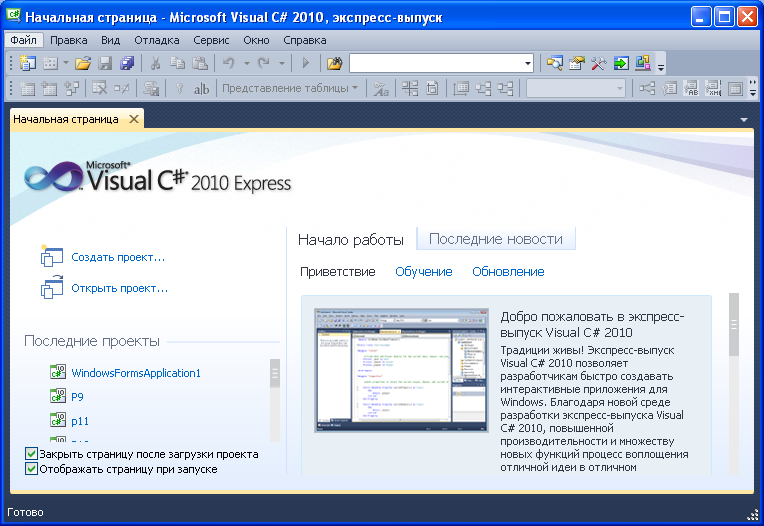


Рис. 11.1 – Начальная страница Visual C#

ИСР позволяет *создавать решения,* включающие набор проектов. Приложение Windows Form позволяет использовать все функциональные возможности операционной системы, отличается большими размерами исполняемого файла. На рисунке 11.2 Представлено окно создание проекта, в нем выбирается команда *Приложение Windows Form* и нажимается кнопка *ОК*.

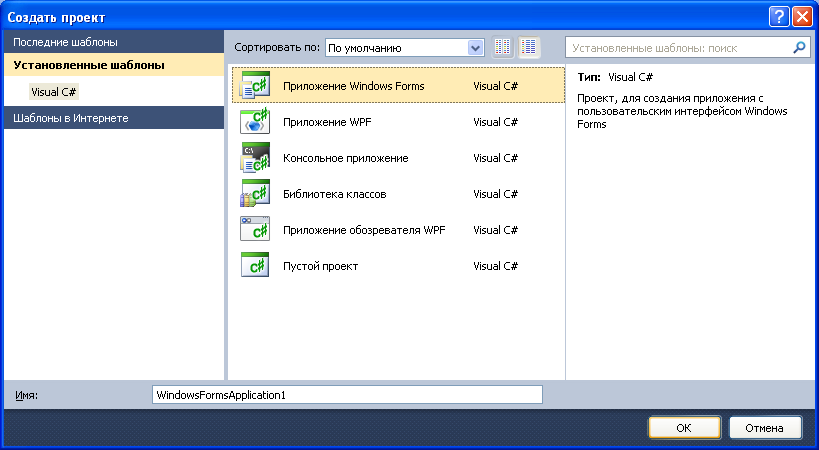
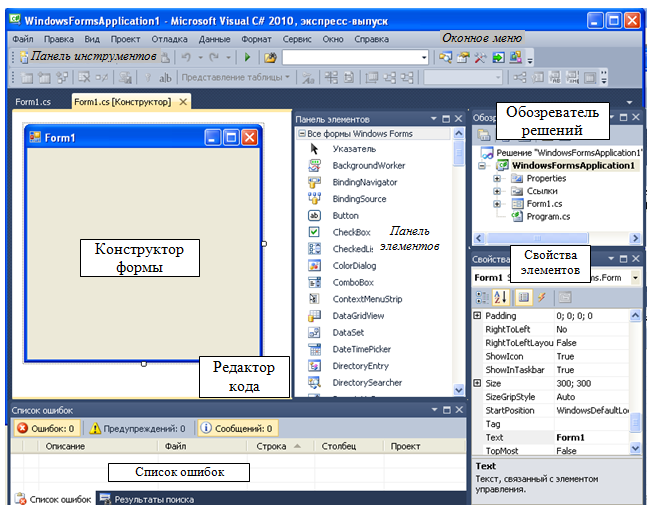


Рис.11.2 – Окно создание проекта

На рис.11.3 Представлен основной экран среды *Windows Form и его содержание.*



Обозреватель решений

Свойства

элементов

Панель

элементов

Рис.11.3 – Основной экран среды

Окно содержит:

* **Строку заголовка**, которая отображает имя открытого проекта;
* **Строку меню** с набором команд для разработки, тестирования приложений;
* **Панель инструментов** с кнопками, соответствующие основным командам меню;
* **Конструктор формы** – средстворазработки приложения,
* **Редактор кода** – для написания адекватных методов объектам приложения,
* **Панель элементов** отображает компоненты (или объекты), с помощью которых создается графическое приложение.
* **Свойства (события) элементов** создаваемого приложения,
* **Список ошибок** – отображающихся при вводе кода и его компиляции,
* **Обозреватель решений,** в котором отображается описание решения.

Microsoft Visual является однодокументной средой, *то есть позволяет работать только с одним проектом*.

Элементы подразделяются на:

1. ***Визуальные*** (кнопки, меню, поля редактирования, переключатели, списки…),
2. ***Логические*** (окна диалога, отчеты, часы…), отображающиеся в процессе запуска приложения.

## 11.2 Окна «Конструктора формы», «Редактора кода», «Свойства элементов»

Окно «Конструктора форм» первоначально имеет заголовок **Form1 –** являетсязаготовкой разрабатываемого приложения**.**

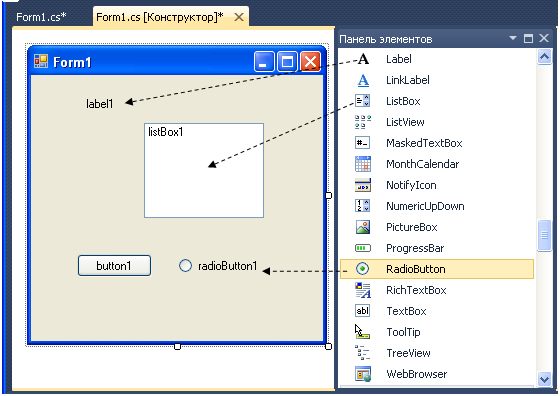
*Форма* – основа разработки, в нее помещают необходимые компоненты, создают интерфейс программы. Объекты выбираются на Панели элементов.

Для размещения объекта на форме необходимо:

1.Щелкнуть мышью на пиктограмме нужного объекта

2. Щелкнуть в том месте Формы, где должен быть он расположен.

Это продемонстрировано на рис. 11.3



11.3 – Окно конструктора формы

Рассмотрим следующее окно «Редактор кода», оно имеет заголовок **Windows Forms.Applicayion.Form1** (рис.11.4)

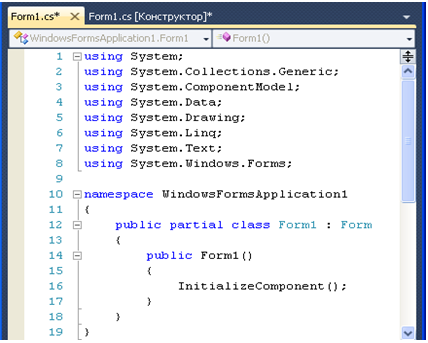


Рис.11.4 – Окно «редактор кода»

*Редактор* содержит шаблон программы - исходный текст модуля разработки. Редактор кода и форма тесно связаны между собой.

В Окне *Редактора кода* отображаются стандартные модули, пространство имен, объявленный по умолчанию класс Form1 и инициализация объектов, установленных на форме.

Окно «Свойства элементов» отражает поведение данного объекта для текущей формы Form1 (рис.11.5)

Окно предназначено для редактирования значений свойств объектов и создания обработчиков событий.

Состоит из страниц / вкладок – свойства и события

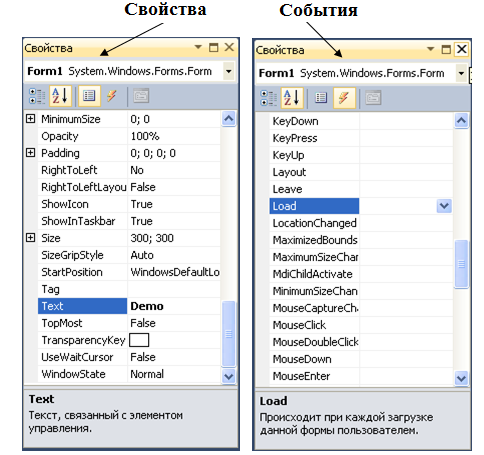


Рис.11.5 – Окно «Свойства элементов»

Страница *Свойство* (список) отображает информацию о текущем компоненте.

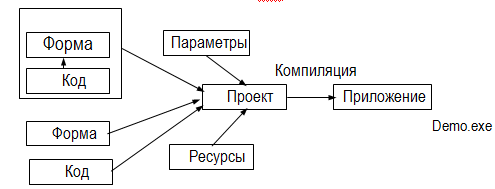
Страница *События* (список событий) определяет процедуру, которую компонент должен выполнить при возникновении указанного события.

## Работа с проектом

***Характеристика проекта***

Приложение (например, Demo) имеет несколько файлов, объединенных в проект:

* Папка, например P, с файлами – Demo
* Файл решение – P.ch
* Файл – опции – P.suo



Файлы проекта располагаются в одном **каталоге** (папке) P.

**Сохранение проекта** – приложения под Windows

Выполняется командой Файл \ Сохранить все как показано на рис.11.6

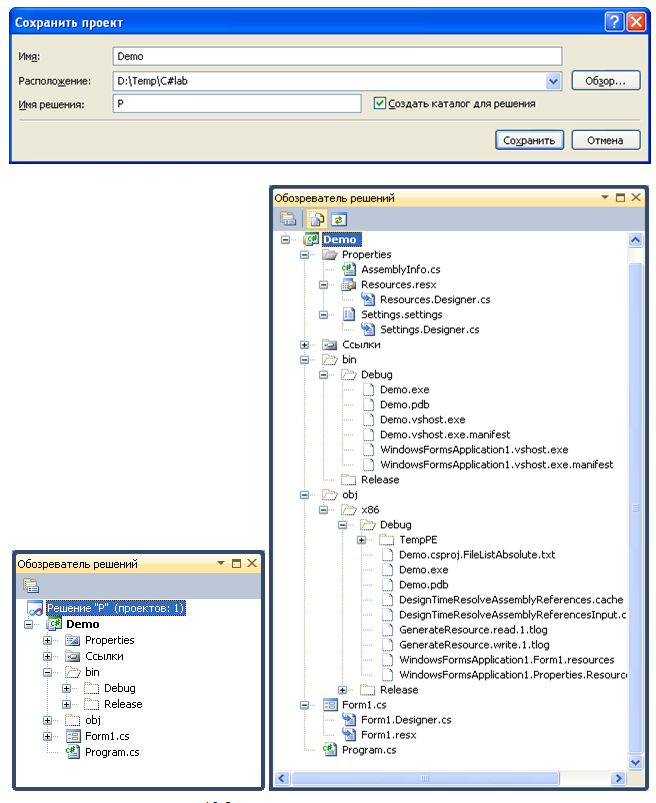


Рис.11.6 – Сохранение проекта

***Запуск и компиляция приложения.***

*Сборка проекта* выполняется командой Отладка/Начать отладку

При этом образуются исполняемые и объектные файлы, перекомпилируются все модули, входящие в проект.

Таблица 11.1

Решение в папке P

|  |  |
| --- | --- |
| Решение P  Demo  Properties  Ссылки  bin  Debug  Demo.exe  Demo.pdb Demo.vshost.exe  Release  obj | Решение  Проект  Свойства  Ссылки  Двоичные файлы  Файлы отладки  Управляемый исполняемый файл  База данных для компилятора  Служебный файл  Файлы выпуска  Объектные файлы |

Исполняемые (bin) и объектные (obj) файлы образуются при компиляции (построении). Возможны два режима решения:

Команда **Построение => Построить решение**. Построение в режиме отладки: в компонуемые файлы включаются символы отладки и режим оптимизации исключается. Это может увеличить размеры файлов. Файлы размещаются в папках Debug.

Команда **Построение => Перестроить решение** (в версии Express отсутствует). Построение отлаженного проекта, когда в компонуемые файлы символы отладки не включаются и компилятор использует режим оптимизации кода (например, исключает не использованные переменные). Это может уменьшить размеры файлов. Файлы размещаются в папках Release.

## 11.4 Шаблон кода:

*Стандартные модули* (по умолчанию)

using System; //системный

using System.Collections.Generic; //с коллекциями

using System.ComponentModel; //с компонентами

using System.Data; //дата/время

using System.Drawing; //с инструментами рисования

using System.Linq; //с языковыми средствами

using System.Text; //для работы с текстом

using System.Windows.Forms; //с формой под Windows

namespace WindowsFormsApplication1 //пространство имен приложения1

{

public partial class Form1 : Form //объявление класса Form1

{

public Form1() // объявление объекта Form1

{

InitializeComponent(); // объявление компонентов

} // на Form1

}

}

Интерфейс Windows.Forms состоит из сотен типов, которые объединены в пространстве имен namespace.

## 11.4 Разработка приложения

Разработка приложения состоит из *двух этапов*:

* Создание интерфейса приложения,
* Определение функциональности приложения с написанием адекватных процедур.

Интерфейс приложения определяет способ взаимодействия пользователя и приложения, т.е. внешний вид формы при выполнении приложения и то, каким способом пользователь управляет приложением.

Для создания *интерфейса* необходимо:

* выбрать нужные объекты на Панели элементов;
* определить внешний вид и функциональные возможности компонентов;
* используя Конструктора Формы, расположить компоненты на форме.

*Функциональность* приложения определяется процедурами, которые выполняются при возникновении определенных событий, происходящих при действиях пользователя с элементами формы.

Для обеспечения *функциональности* приложения необходимо:

* задать в *Окне Свойства* значения свойств и процедур обработки событий;
* написать программный код для заданных процедур обработки событий.

***Простейшее приложение***

Представляет собой заготовку, обеспечивающую все необходимое для приложения. Это форма, для которой уже созданы файлы с описанием формы и шаблоном кода.

Форма отображается при первом запуске ИСР и содержит основные элементы окна Windows:

* заголовок Form1,
* кнопки минимизации, максимизации и закрытия окна, изменения размеров окна и кнопку вызова системного меню.

При создании приложений на форму помещаются компоненты, для которых устанавливаются свойства и создаются обработчики событий.

К простейшему приложению разработчик может добавлять другие формы (Form2…). Например, графический интерфейс с выводом двух функций Sin(x) и Cos(x) (рис.11.7)

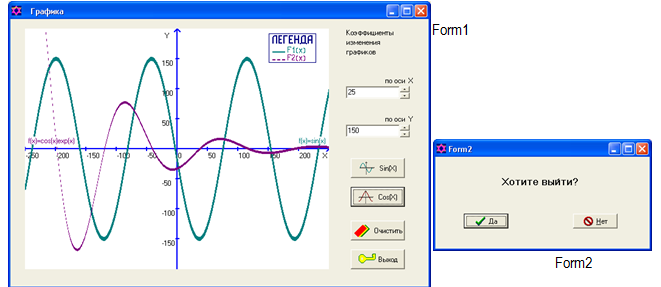


Рис.11.7 – Форма с компонентами

***Создание интерфейса приложения***

Интерфейс приложения составляют компоненты.

**Свойства** компонента это атрибуты, определяющие способ отображения и функционирования компонент (рис.11.8).

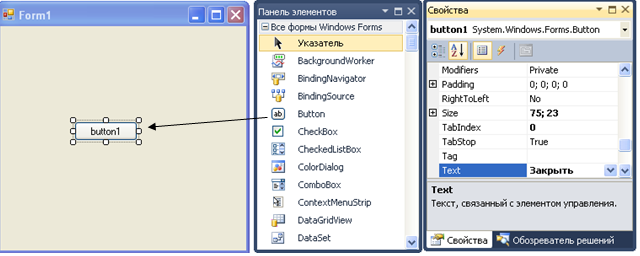


Рис.11.8 – Изменение свойства компонента

Для изменения значения свойств с помощью Окна элементов необходимо:

* щелкнуть мышью по его имени в левой колонке страницы *Свойство*;
* установить значение в соответствующей ячейке правой колонки:
* нажать клавишу <Enter>.

***Типы свойств:***

С

П

И

С

О

К

* *простые* – это те, значения которых являются числами или строками. Например, Text, Name, Left, Top, Width, Height;
* *перечисляемые* – это те, которые могут принимать значения из предложенного набора (списка). Например, тип Bool принимает значение True или False или ForeColor (рис.11.9)

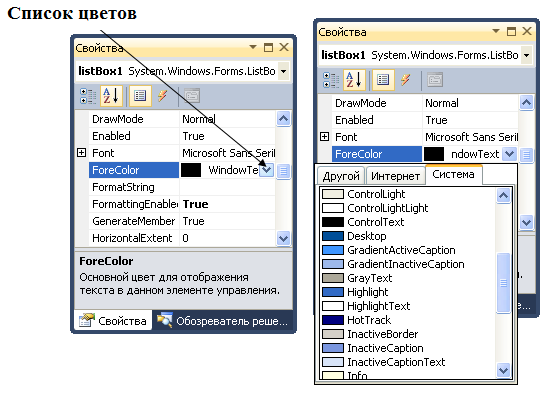


Рис.11.9 – Перечисляемые свойства

* *вложенные –* это те, которые поддерживают вложенные значения (или объекты). Имеются два вида: множества (…) Например, Items и комбинированные значения ( + ) (рис.11.10)..

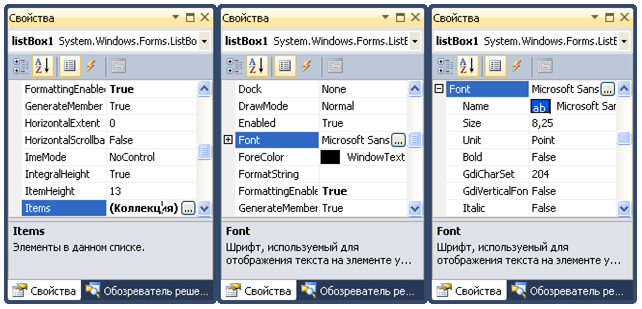


Рис.11.10 – Вложенные свойства

Изменять свойства объектам можно двумя способами:

1. С помощью Окна Свойства (в режиме проектирования приложения – задаются стартовые значения свойств компонентов);
2. Программно – путем создания адекватных процедур с соответствующими строками кода (переопределение свойств в процессе выполнения).

При выполнении приложения свойства компонентов можно изменять с помощью операторов присваивания.

Например,

* изменение программно надписи кнопки *Button1*:

button1**.**Text="Решить"**;** //надпись кнопки

* изменение размеров многострочного редактора richTextBox*1*:

richTextBox*1*.Height = 100; // высота

richTextBox*1*.Width = 150; // ширина

* изменение текста и цвета шрифта в метке label1:

label1.Text = "Студент"; //текст со словом Студент

label1.ForeColor=Color.Red; //цвет шрифта красный



***Определение функциональности приложения***

*Событие* – это то, что происходит во время работы программы. В C# каждому событию присваивается имя.

*Реакция*окна на разного рода действия пользователя определяет *функциональность* приложения.

*Обработчик событий* – определяет действие, которое можно поручить выделенному компоненту. Каждый компонент имеет набор стандартных обработчиков событий, отображающихся в Окне Свойство на вкладке События (рис.11.11).

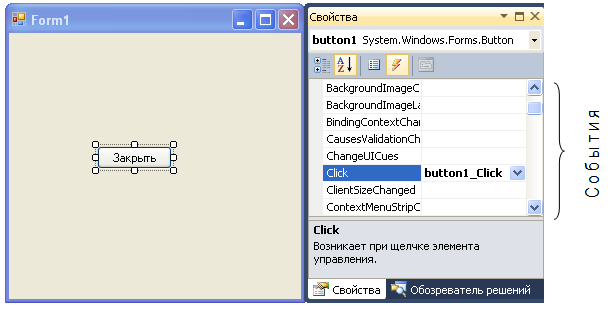


Рис.11.11 – Окно «Свойства»

Чтобы при нажатии на кнопке "Закрыть" кнопка реагировала на событие завершения работы, необходимо создать процедуру обработки события, которая будет вызываться при возникновении данного события (например, OnClick).

С о б ы т и я

1) выделить на форме компонент;

2) перейти на страницу *событий* Окна Свойства;

3) сделать двойной щелчок в области значения события, например OnClick, ( ИСР автоматически создаст в модуле формы заготовку процедуры-обработчика).

4) в редакторе кода, вместе, где установится курсор написать код, который будет выполняться при нажатии кнопки Button1, т.е набрать строку *Close()* (рис.11.12)*;*

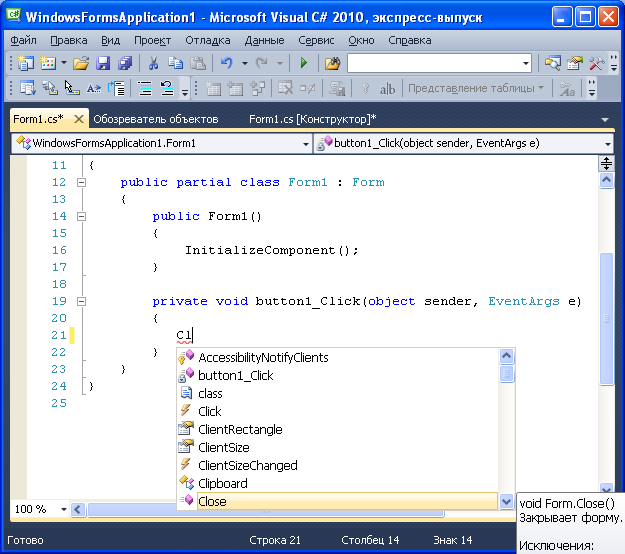
****

Рис.11.12 – Редактор кода

При наборе кода включается интерактивный подсказчик, который лучше использовать для избегания ошибок в наборе слов (рис.11.13).

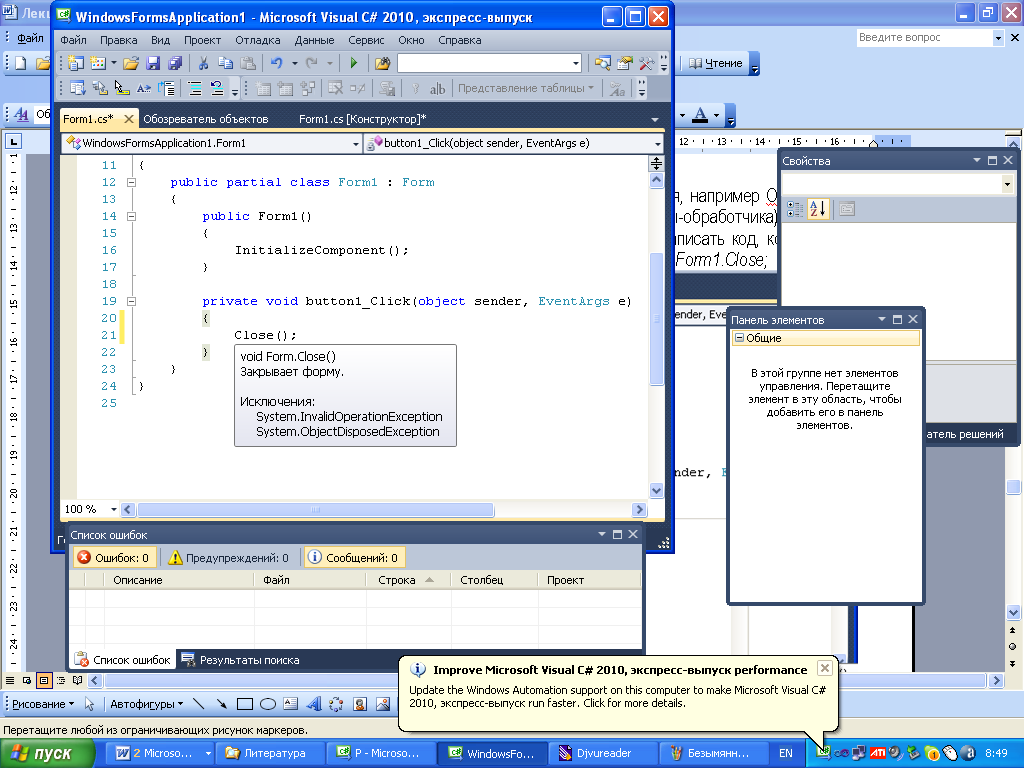
****

Рис.11.13 – Интерактивный подсказчик

Все остальное среда создает автоматически.

Обработчики *щелчка* по объектам, например Button1Click, можно создать и быстрее, путем двойного щелчка по объекту, находящемуся на Форме.

private void button1\_Click(object sender, EventArgs e)

Слова

Private – метод **закрыт** и предназначен для внутреннего использования в классе

void – начало метода – процедура,

Button1\_Click – собственно имя процедуры,

() – список параметров.

Для *удаления* процедуры-обработчика достаточно удалить заготовку и код, который программист вносил в нее самостоятельно. После этого при сохранении или компиляции модуля обработчик будет удален автоматически из всех файлов проекта.

**Пример.** Создать проект для изучения некоторых простых свойств объектов.

Последовательность действий:

1. Создаем Приложение Windows Forms.
2. Устанавливаем на форму нужные компоненты
3. Устанавливаем стартовые значения объектам, например Кнопке надпись Задача.
4. Создаем обработчик события Form1\_Activated – (активизация формы) при исполнении которого свойства объектов меняются программно на иные.
5. Отлаживает проект путем устранения синтаксических ошибок.
6. Проверяет полученные результаты на соответствие поставленной задачи.

3. Изменяет значения Text кнопки button1 на надпись Пример (рис.11.14)

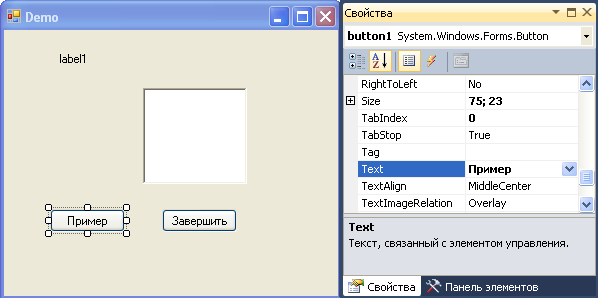


Рис.11.14 – Изменение значения Text

4. Создаем обработчик события Form1\_Activated для выделенной Формы Form1 (рис.11.15)

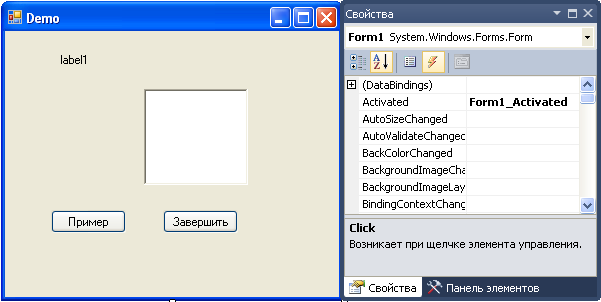


Рис.11.15 – Обработчик событий Form1\_Activated

Записываем код процедуры Form1\_Activated.

4. Создаем обработчик события button1\_Click для выделенной кнопки button1 (рис.11.16)

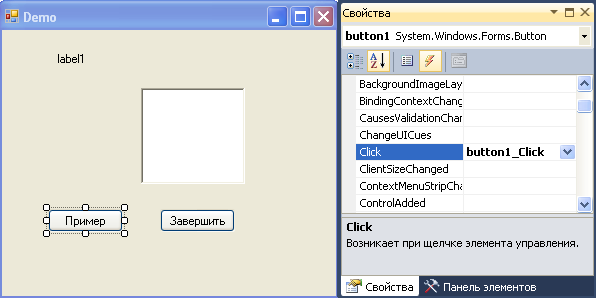


Рис. 11.16 – Обработчик событий button1\_Click

Записываем код процедуры button1\_Click. Аналогично для button2.

Листинг программы:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Activated(object sender, EventArgs e)

{

button1.Text = "Задача"; // надпись на кнопке

richTextBox1.Height = 100; // высота объекта

richTextBox1.Width = 150; // ширина объекта

label1.ForeColor=Color.Red; // цвет шрифта красный

label1.Text = "Студент"; // вывод текста

label1.ForeColor=Color.Red; // цвет шрифта красный

richTextBox1.Text = "Мы изучаем C#";// вывод текста

}

private void button1\_Click(object sender, EventArgs e)

{

label1.Text = "Студент ПГУТИ"; // вывод текста

richTextBox1.ForeColor = Color.Blue; // цвет шрифта синий

richTextBox1.Text = "Мы активно работаем"; //вывод текста

}

private void button2\_Click(object sender, EventArgs e)

{

Close(); // закрытие формы

}

}

}

Результат исполнения: Активизация формы и Щелчок по кнопке «Задача» (рис.11.17)

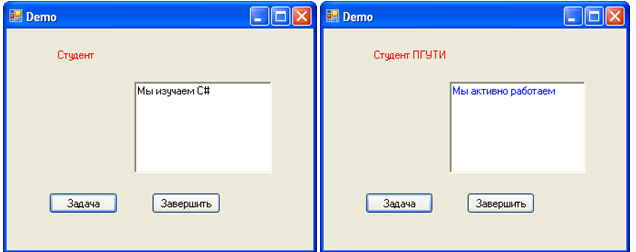


Рис.11.17 – Результат выполнения задачи

***Отладка***

Тестирование – процесс проверки работоспособности программы, в процессе которой выявляются разного рода ошибки.

Процесс поиска и устранения ошибок называется **отладкой**.

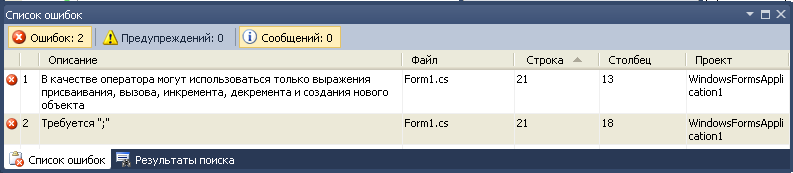
*Классификация ошибок*

Программ без ошибок не бывает. Ошибки допускает любой программист: и начинающий, и имеющий тридцатилетний опыт. Компилятор генерирует исполняемую программу лишь в том случае, если исходный текст не содержит синтаксических ошибок.

В процессе выполнения программы возникают ошибки следующих видов:

* Синтаксические (ошибки времени компиляции),
* Логические (алгоритмические – ошибочные рассуждения),
* Динамические (ошибки времени выполнения) – исключительные ситуации.

*Синтаксические* ошибки возникают при нарушении синтаксиса языка. Они выявляются и устраняются при компиляции программы. Сообщения о найденных ошибках отображаются в Окне Списка ошибок. Например,



При обнаружении в программе неточностей, которые не являются ошибками, компилятор выводит подсказки и предупреждения.

*Логические* ошибки являются следствием реализации неправильного алгоритма и проявляются при выполнении программы. Сообщения компилятором не выдаются, но программа работает не корректно и выдаёт не правильные результаты.

Например, два выражения y=ln(x)/x+5 и y=ln(x)/(x+5) дадут разные результаты.

Поэтому при получении результата его надо анализировать. Составляются тесты и отлаживаются программы на тех значениях входных данных, результат при которых легко проверить путем подсчета его «вручную».

Например, при х=1 ln(x)= 0 и

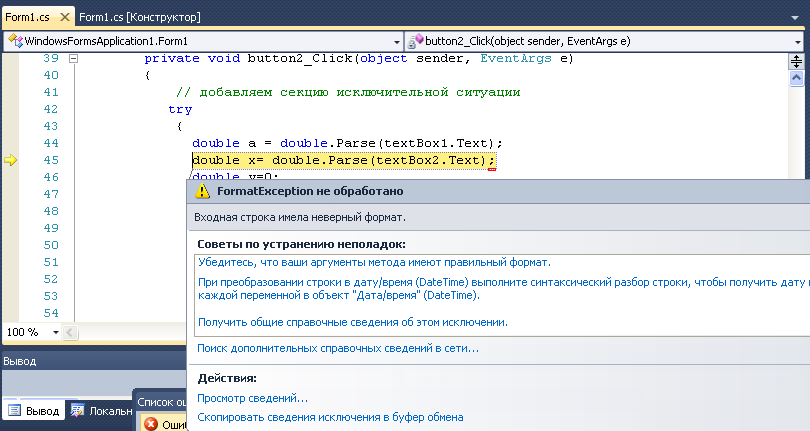
результат первого выражения y=ln(x)/x+5 даст y=1/1+ 6 = 6,

а второго выражения y=ln(x)/(x+5) даст y=1/(1+5)=1/ 6 = 0.1666

*Динамические* ошибки возникают при выполнении программы и являются следствием неправильной работы операторов, подпрограмм, методов программы и операционной системы.

Исключительная ситуация (**exception**) представляет собой нарушение условий выполнения программы, вызывающее прерывание или полное прекращение ее работы.

Например, если пользователь в редактор textBox2 ввел букву, вместо цифры, то ИСР сгенерирует окно ошибки и проект будет закрыт.



Чтобы этого не произошло, вводим свой обработчик исключительных ситуаций и в окне появляется надпись об ошибочном вводе данных.

Пример показан на рис.11.18

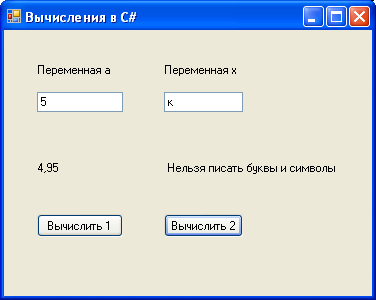


Рис.11.18 – Пример

После внесения изменения в textBox2 и нажатия кнопки «Вычислить» можно продолжить выполнение программы или прервать ее выполнение. Обработка исключительной ситуации состоит в нейтрализации динамической ошибки, вызвавшей ее.

**Краткие итоги.** В лекции были рассмотрены вопросы, относящиеся к приложению ОС Windows. Даны его основные характеристики. Описан процесс создания проекта в Windows Form. Расмотрены характеристики проекта.

**Контрольные вопросы**

1. Перечислите основные характеристики Приложение под ОС Windows.
2. Перечислите элементы Windows Form.
3. Что необходимо для размещения объекта на форме.
4. Что отражает окно свойства элементов?
5. Дайте краткую характеристику проекта.
6. Перечислите этапы разработки приложения.
7. Что необходимо для создания интерфейса?
8. Как обеспечить функциональность приложения?
9. Перечислите типы свойств.
10. Как можно изменить свойства объекта?
11. Приведите примеры изменения свойств объекта.
12. Дайте определение понятию «Событие».
13. Какие виды ошибок возникают в процессе выполнения программы?

# Лекция 12

# Графика в C#

**Цель лекции.** Ознакомиться с правилами работы с графиками. Научиться загружать готовые картинки, рисовать комбинированные фигуры, менять цвет, шрифт.

Среда позволяет:

загрузить в приложение готовые картинки и фотографии,

разработать программы, которые выводят графику на поверхность объекта.

В ИСР C# Express позволяет загружать файлы – рисунки, созданные в других программах, например Paint, Corel Draw и др.

## 12.1. Рисованные изображения

Рисованные изображения отображаются на форме при выполнении программы с помощью различных инструментов.

Изображение при этом представляет собой комбинацию простейших фигур – графических примитивов (точка, линия, круг или прямоугольник).

Каждая точка на форме имеет координаты X и Y. Текущая позиция при рисовании определяется горизонтальной (X) и вертикальной (Y) координатами, заданными в пикселях. Координата Х возрастает при перемещении указателя слева на право, а координата Y – при перемещении его сверху вниз (рис.12.1)

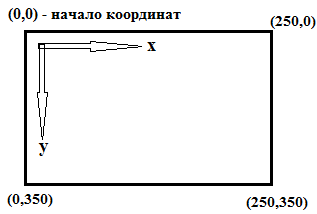


Рис. 12.1– Сетка координат

## 12.2 Объект Graphics

Объект **Graphics** – это указатель на место, где будут рисоваться примитивы. Пусть мы хотим рисовать в форме Windows. Синтаксис задания ссылки на нее:

Graphics g = Graphics.FromHwnd(this.Handle);

Здесь: Graphics – тип объекта, g – имя переменной,

Graphics.FromHwnd(this.Handle) – используемый метод FromHwnd из класса Graphics, который задает ссылку Handle на форму Windows.

В С# инструменты рисования определены в пространстве имен System.Drawing.

Там находятся классы:

Pen (перо). Объекты пера используются в методах рисования линий и контуров геометрических фигур.

Brush (кисть). Объекты кисти используются в методах заливки областей, ограниченных контурами.

**Перо (Pen)**

Объекты пера используются в методах рисования линий и графических фигур.

Объекты Pen выбираются из класса Pens (перья). Класс Pens содержит набор объектов для выбора. У них толщина линии (1 пиксель), стиль линии – сплошная. У каждого объекта свой цвет линии, имя которого идентифицирует объект. Такой объект нельзя редактировать, его можно только применять.

Например, создаем объект myPen, совпадающий с шаблоном:

Pen myPen = Pens.Black;

Объекты Pen с изменяемыми свойствами создаются из класса Pen (перо). В этом случае для объекта пера можно устанавливать много свойств. Основные свойства:

Color – цвет линии;

Brush – ссылка на кисть, используемую в качестве пера ;

Width – толщина линии;

DashStyle – стиль штриховой линии,

DotStyle – стиль пунктирной линии

DashDotStyle – штрих пунктир,

DashDotDotStyle - штрих двойной пунктир,

SolidStyle– стиль непрерывная линия.

Сначала объект myPen можно создать с указанием цвета.

Pen myPen = new Pen(Color.Red);

Затем ему можно изменить свойства:

myPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Solid;

myPen.PenType = System.Drawing.Drawing2D.PenType.SolidColor;

myPen.Width = 2;

Можно задать и красиво пишущее перо, в нем перо это кисть.

Pen myFancyPen = new Pen(myBrush);

**Кисть (Brush)**

Объекты кисти используются в методах заливки графических фигур. Определены кисти разного типа:

Brush – простая кисть, одноцветная заливка.

HatchBrush – кисть со штриховой заливкой.

LinearGradientBrush – кисть с линейной градиентной заливкой, цвет фрагментов фигуры меняется плавно.

PathGradientBrush – кисть с градиентной заливкой, цвет фрагментов фигуры меняется скачкообразно.

Объекты Brush выбираются из класса Brushes, который содержит кисти со сплошной заливкой. Класс Brushes содержит набор объектов для выбора, у которых по умолчанию определен цвет. У каждого объекта выбора имя – это цвет заливки. Например, создаем объект myBrush, совпадающий с шаблоном (рис.12.2):

Brush myBrush = Brushes.Blue; // Заливка синим

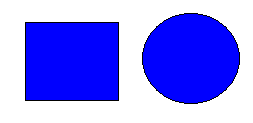


Рис.12.2 Примеры нарисованных графиков

Объекты HatchBrush выбираются из класса HatchBrushes. Класс HatchBrushes содержит набор объектов для выбора, у которых по умолчанию определены стиль заливки HatchStyle, цвет переднего плана ForeColor и цвет фона BackColor (задний план). Определены стили заливки:

* сетка (Cross),
* диагональная сетка (DiagonalCross),
* прямая диагональ (ForwardDiagonal),
* обратная диагональ (BackwardDiagonal) и др.

Например, создаем кисть с заливкой сеткой HatchStyle.Cross:

HatchBrush brush2 = new HatchBrush(HatchStyle.Cross, ForeColor, BackColor) (рис.12.3)

.



Рис. 12.3 – Результат заливки прямоугольника

**Шрифты и текст**

Для вывода на форму текста используется метод DrawString. Рисует заданную текстовую строку в заданном прямоугольнике с помощью определяемых объектов кисти и шрифта, используя атрибуты форматирования заданного формата. Синтаксис метода:

DrawString(S, Font, Brush, RectangleF, StringFormat);

S –текстовая строка для рисования.

Font – шрифт текстовой строки.

Brush – кисть. Определяет цвет и текстуру создаваемого текста.

RectangleF – прямоугольник вывода.

StringFormat – формат, определяющий атрибуты форматирования, такие как междустрочный интервал и выравнивание, которые применяются к создаваемому тексту.

Возможны способы вызова, отличающиеся друг от друга числом аргументов и способом задания прямоугольника вывода. Формат можно не указывать, будет использован формат по умолчанию.

**Font**. Это шрифт текстовой строки. Выбирается с помощью методов класса Font. Предоставляет возможность выбора размера и стиля шрифта. Возможны несколько способов вызова, отличающиеся друг от друга числом аргументов и способом задания нового шрифта. Например:

font МойШрифт = new Font("Arial" , 24 , FontStyle.Bold ) ;

или

font Шрифт = new Font("Arial Narrow" , 14 ) ; // без учета начертания шрифта

**RectangleF**

Это прямоугольник, в котором рисуется строка текста. Задается двумя способами:

4 координаты - левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов.

Объект точка с координатами левого верхнего угла (P) и размеры (H - ширина и W –высота прямоугольника).

## 

## 12.3 Методы рисования

В C# определены методы рисования линий и фигур. Все методы перегружаемые, то есть выполняются по-разному с разными аргументами.

**Рисование с помощью пера Pen**

При рисовании можно использовать перо с разными стилями линий LineStyle. Например, solid (сплошная), Dash (штрих), Dot (пунктир), DashDot (штрих-пунктир), DashDotDot (штрих-пунктир-пунктир).

**DrawLine**.

Прямая линия между двумя точками. Синтаксис метода:

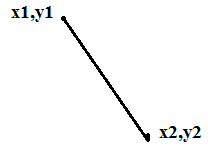
g.DrawLine(pen, x[1], y[1], x[2], y[2],);

Здесь g – где рисуем,

DrawLine – метод рисование линии,

Pen – перо,

x[1], y[1], x[2], y[2] – точки границы линии, отмечены точками.



Можно рисовать окружности, дуги, сектора, прямоугольники, многоугольники (графические примитивы)

**DrawRectangle**

Прямоугольник. Синтаксис метода:

g.DrawRectangle(pen, rect);

Здесь:

g – где рисуем,

DrawRectangle – метод рисования прямоугольника,

pen – перо,

rect – прямоугольник, свойства которого задаются целыми числами

rectangle Прямоугольник = new Rectangle(x, y, w, h);.

Здесь: x, y – координаты левого верхнего угла,

w – ширина прямоугольника,

h – высота прямоугольника.

Возможен вариант задания свойств числами в формате с плавающей точкой

rectangle Прямоугольник = new RectangleL(x, y, w, h) (рис.12.4);

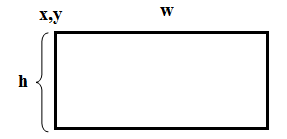


Рис.12.4 – Задание свойств числами в формате с плавающей точкой

**DrawEllipse**

Эллипс. Синтаксис метода:

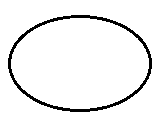
g.DrawEllipse(pen, rect);

Здесь: g – где рисуем,

DrawEllipse – метод рисования эллипса,

pen – перо,

rect – прямоугольная область, в которую вписывается эллипс.



## 12.4 Методы заливки

В C# определены методы заливки фигур. Все методы перегружаемые, то есть выполняются по-разному с разными аргументами.

В примерах заливка разных фигур осуществляется **простой** кистью с Brush.Cyan и кистью HatchBrush с **разными** стилями заливки DashStyle.

**FillRectangle**.

Закрашивает прямоугольник. Синтаксис метода:

g.FillRectangle (brush, p);

Здесь: g – где рисуем,

FillRectangle – метод рисования залитого прямоугольника

brush – кисть,

p – массив точек.

Получаемый результат зависит от стиля заливки. Рисунок слева использована простая кисть с цветом Cyan. Справа использована кисть, у которой стиль заливки BacwardDiagonal (обратная диагональ), цвет переднего плана черный, фона белый (рис.12.5).

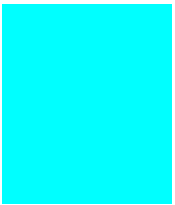
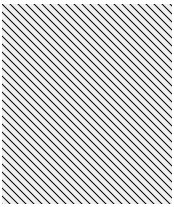
 

Рис.12.5 – Примеры заливки прямоугольника

**FillEllipse**.

Закрашивает эллипс. Синтаксис метода:

g.FillEllipse (brush, rect);

Здесь g – где рисуем,

FillEllipse – метод рисования залитого эллипса,

brush – кисть,

rect – прямоугольная область, в которую вписывается эллипс.

Получаемый результат зависит от стиля заливки. Рисунок слева использована простая кисть с цветом Cyan. Справа использована кисть, у которой стиль заливки DiagonalCross (диагональная сетка), цвет переднего плана черный, фона белый(рис.12.6)

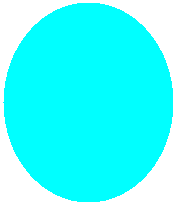
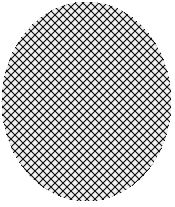
 

Рис.12.6 Примеры заливки эллипса

## 12.5 Рисование с помощью примитивов

using System;

using System.Collections.Generic;

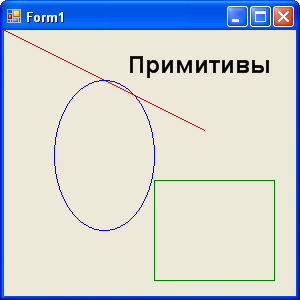
using System.ComponentModel;

using System.Data;

using System.Drawing; // рисование

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

// обработчик события рисования на форме

private void Form1\_Paint(object sender, PaintEventArgs e)

{

using (Graphics g = e.Graphics) //Создание объекта типа График

{

using (Pen pen = new Pen(Color.Red)) //цвет карандаша

g.DrawLine(pen,0,0,200,100); //рисование линии

using (Pen pen = new Pen(Color.Blue))

g.DrawEllipse(pen, new Rectangle(50, 50, 100, 150)); //эллипс

using (Pen pen = new Pen(Color.Green))

g.DrawRectangle(pen, new Rectangle(150, 150, 120, 100));//прямоугольник

string s="Примитивы";

Font font=new Font("Arial",18,FontStyle.Bold);

SolidBrush brush = new SolidBrush(Color.Black);

float x = 120;

float y = 20;

g.DrawString(s, font, brush, x, y); //вывод текста в точку (x,y)

}

}

}

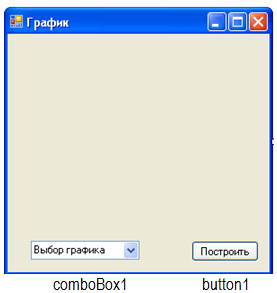
}

## 12.6 Рисование графика функции

Рассмотрим пример программы, которая на форме Windows при соответствующем выборе из списка рисует два графика: sin(x) и cos(x) с текстовыми заголовками. Форма включает контейнер-список «Выбор графика» и кнопку с надписью «Построить», которая запускает обработчик, выполняющий все операции.

В обработчике выполняются следующие действия:

* Вычисляется функция.
* Задается ссылка на объект графики.
* Выводится текст заголовка.
* Рисуются координатные оси.
* Рисуется график функции.



Листинг программы:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

int imax = 100; //число точек в периоде

int t = 2; //число периодов

int amp = 80; //амплитуда

int h = 40; //отступ для текста

int x0 = 20; //начала координат

int y0 = h + amp;

double[] f = new double[imax \* t + 10]; //массив точек

if (comboBox1.Text == "sin(x)") если из списка выбрать строку "sin(x)"

{ // Расчет массива точек для графика "sin(x)"

for (int i = 0; i < imax \* t; i++)

{

f[i] = Math.Round(amp \* Math.Sin(2 \* Math.PI / imax \* i));

}

// Инструменты рисования

Graphics g = Graphics.FromHwnd(this.Handle); // Где рисуем

Pen pen = Pens.Black; // Чем рисуем

// Текст заголовка

g.DrawString("График синусоиды", new Font("Arial", 14), Brushes.Red, 40, 0);

// Рисуем график "sin(x)"

g.DrawLine(pen, x0, y0, x0 + imax \* t, y0); //Рисуем ось X

g.DrawLine(pen, x0, y0 - amp, x0, y0 + amp); //Рисуем ось Y

g.DrawString("Y", new Font("Arial", 10), Brushes.Black, 10, 20);

g.DrawString("X", new Font("Arial", 10), Brushes.Black, x0 + imax \* t, y0);

for (int i = 0; i < imax \* t; i++) //Рисуем график

{

int f1 = y0 - (int)f[i]; //Координата Y[i]

int f2 = y0 - (int)f[i + 1]; //Координата Y[i+1]

g.DrawLine(pen, x0 + i, f1, x0 + i + 1, f2);

}

}

else if (comboBox1.Text == "cos(x)") если из списка выбрать строку "cos(x)"

{ // Расчет массива точек для графика "cos(x)"

for (int i = 0; i < imax \* t; i++)

{

f[i] = Math.Round(amp \* Math.Cos(2 \* Math.PI / imax \* i));

}

// Инструменты рисования

Graphics g = Graphics.FromHwnd(this.Handle); // Где рисуем

Pen pen = Pens.Black; // Чем рисуем

// Текст заголовка

g.DrawString("График косинусоиды", new Font("Arial", 14), Brushes.Blue, 40, 0);

// Рисуем график "cos(x)"

g.DrawLine(pen, x0, y0, x0 + imax \* t, y0); //Рисуем ось X

g.DrawLine(pen, x0, y0 - amp, x0, y0 + amp); //Рисуем ось Y

g.DrawString("Y", new Font("Arial", 10), Brushes.Black, 10, 20);

g.DrawString("X", new Font("Arial", 10), Brushes.Black, x0 + imax \* t, y0);

for (int i = 0; i < imax \* t; i++) //Рисуем график

{

int f1 = y0 - (int)f[i]; //Координата Y[i]

int f2 = y0 - (int)f[i + 1]; //Координата Y[i+1]

g.DrawLine(pen, x0 + i, f1, x0 + i + 1, f2);

}

}

}

}

}

При запуске отображается форма, в которой при выборе типа графика рисуется соответствующий график.

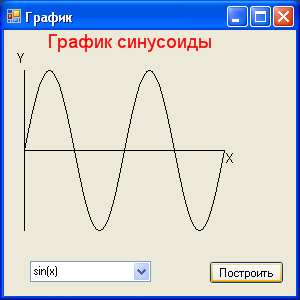
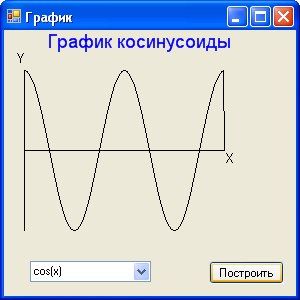
****

Рис. 12.7 – Пример отображения графика

**Пример**. Создать программу отображения диаграммы двух функций sin(x) и cos(x) с использованием компонента Chart.

Проект – WindowsForm приложение. Стиль линий Spline.

Создаем – WindowsForm приложение. В форму заносим компоненты chart1 для отображения диаграммы и button1 для создания стартового обработчика событий.

В окне компонента chart1 в разделе Series определяем коллекцию из двух серий (рис.12.7):

Для 1-й функции с именем sin(x).

Для 2-й функции с именем cos(x).

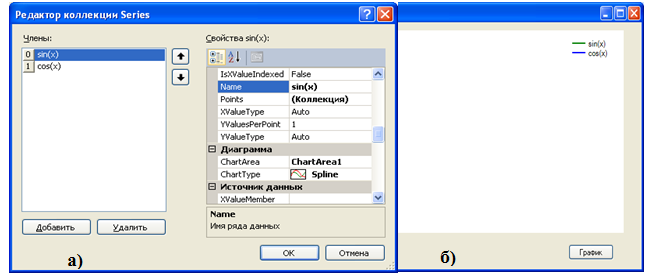


Рис.12.8 – а)Стартовая форма; б) Изменение свойств графиков

using System;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

double y = 0;

for (int x = 0; (x <= 19); x++) //цикл по оси х

{

y = Math.Sin(Math.PI / 5 \* x); //расчет у 1-го графика

chart1.Series["sin(x)"].Points.AddXY(x, y);

y = Math.Cos(Math.PI / 5 \* x); //расчет у 2-го графика

chart1.Series["cos(x)"].Points.AddXY(x, y);

}

}

}

}

Компонент chat с помощью окна Свойства позволяет менять цветовую гамму и тип графика выбранной серии в коллекции (рис.12.9).

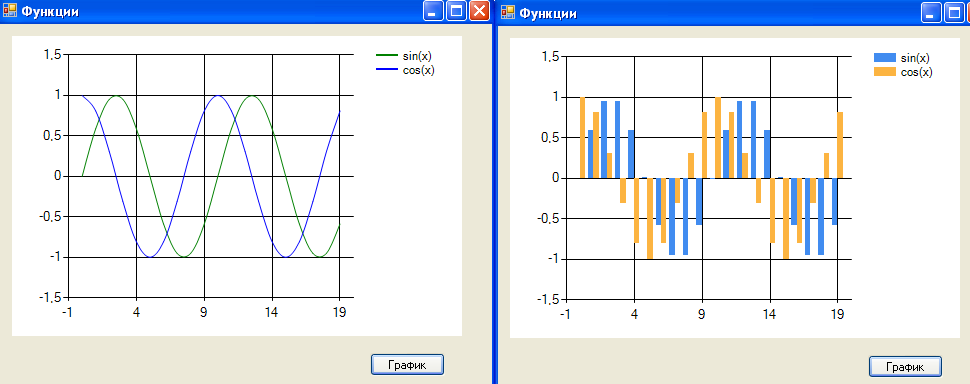


Рис.12.9 – Пример изображения график функций

## 12.7 Растровая графика

Для растровой графики применяются различные форматы:

Bitmap (bmp), попиксельная графика.

TIFF; GIF; PNG; JPEG

**BMP (Bitmap Picture)** — формат хранения [растровых изображений](http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) в виде пикселей.

С форматом BMP работает огромное количество программ, так как его поддержка интегрирована в операционные системы.

Файлы формата BMP могут иметь расширения .bmp, .dib и .rle.

Представляет собой набор пикселей, каждый из которых отображается 3 байтами. В байтах фиксируется интенсивность компонент цветов пикселя R (красный), G (зеленый), B (синий). Это самый точный, но и самый емкий по объему формат.

В формате BMP есть поддержка сжатия по алгоритму [RLE](http://ru.wikipedia.org/wiki/RLE), однако теперь существуют форматы с более сильным сжатием, и из-за большого объёма BMP редко используется в [Интернете](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), где для сжатия без потерь используются [PNG](http://ru.wikipedia.org/wiki/PNG) и более старый [GIF](http://ru.wikipedia.org/wiki/GIF).

**TIFF (Tagged Image File Format)** — формат хранения с использованием тегов. TIFF стал популярным форматом для хранения изображений с большой глубиной цвета. Файлы формата TIFF, как правило, имеют расширение .tiff или .tif.

Имеется возможность сохранять изображение в файле формата TIFF со сжатием и без сжатия. Степени сжатия зависят от особенностей самого сохраняемого изображения, а также от используемого алгоритма. Формат TIFF позволяет использовать следующие алгоритмы сжатия:

[RLE](http://ru.wikipedia.org/wiki/RLE), с обнаружением длинных одноцветных фрагментов.

[Lempel-Ziv-Welch](http://ru.wikipedia.org/wiki/Lempel-Ziv-Welch) (LZW), с использованием словарей повторяющихся фраз, как в архиваторах [ZIP](http://ru.wikipedia.org/wiki/ZIP), архивирование [JPEG](http://ru.wikipedia.org/wiki/JPEG).

**GIF (Graphics Interchange Format**) — формат для обмена изображениями. Формат GIF способен хранить сжатые данные без потери качества в формате [до 256 цветов](http://ru.wikipedia.org/wiki/8-%D0%B1%D0%B8%D1%82%D0%BD%D1%8B%D0%B9_%D1%86%D0%B2%D0%B5%D1%82), использует [LZW](http://ru.wikipedia.org/wiki/LZW)-компрессию, что позволяет неплохо сжимать файлы, в которых много однородных заливок (логотипы, надписи, схемы).

Изображение в формате GIF хранится построчно, поддерживается только формат с индексированной палитрой цветов. Стандарт разрабатывался для поддержки 256-цветовой палитры.

Один из цветов в палитре может быть объявлен «прозрачным». В этом случае в программах, которые поддерживают прозрачность GIF (например, большинство современных [браузеров](http://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80)) сквозь пиксели, окрашенные «прозрачным» цветом будет виден фон.

Формат GIF допускает чересстрочное хранение данных. При этом строки разбиваются на группы, и меняется порядок хранения строк в файле. При загрузке изображение проявляется постепенно, в несколько проходов. Благодаря этому, имея только часть файла, можно увидеть изображение целиком, но с меньшим разрешением.

В чересстрочном GIF сначала записываются строки 1, 5, 9 и т. д. Таким образом, загрузив 1/4 данных, пользователь будет иметь представление о целом изображении. Вторым проходом следуют строки 3, 7, 11, разрешение изображения в браузере ещё вдвое увеличивается. Наконец, третий проход передаёт все недостающие строки (2, 4, 6…). Таким образом, задолго до окончания загрузки файла пользователь может понять, что внутри и решить, стоит ли ждать полной загрузки изображения. Чересстрочная запись незначительно увеличивает размер файла, но это, как правило, оправдывается приобретаемым свойством.

Формат GIF поддерживает анимационные изображения. Фрагменты представляют собой последовательности нескольких статичных кадров, а также информацию о том, сколько времени каждый кадр будет показан на экране. Анимация может быть закольцована, тогда после последнего кадра будет вновь показан первый и так далее.

GIF первоначально был проприетарным форматом, однако срок его [патентной](http://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%82%D0%B5%D0%BD%D1%82) защиты истёк.

**PNG (portable network graphics)** — [растровый](http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) формат хранения графической информации, использующий [сжатие без потерь](http://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B1%D0%B5%D0%B7_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C). PNG был создан как для улучшения, так и для замены формата [GIF](http://ru.wikipedia.org/wiki/GIF) графическим форматом, не требующим лицензии для использования. Обычно файлы формата PNG имеют расширение .PNG (.png).

Неофициально PNG расшифровывают как «PNG is Not GIF» («PNG — это не GIF»)

PNG произносится по-английски, как слово ping.

**JPEG (Joint Photographic Experts Group, по названию** [**организации-разработчика**](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%B4%D0%B8%D0%BD%D1%91%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D1%8D%D0%BA%D1%81%D0%BF%D0%B5%D1%80%D1%82%D0%BE%D0%B2_%D0%BF%D0%BE_%D1%84%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%B8)) — один из популярных [графических форматов](http://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D1%8B), применяемый для хранения [фотоизображений](http://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) и подобных им изображений. Файлы, содержащие данные JPEG, обычно имеют [расширения](http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B8%D0%BC%D0%B5%D0%BD%D0%B8_%D1%84%D0%B0%D0%B9%D0%BB%D0%B0) .jpeg, .jfif, .jpg, .JPG, или .JPE. Однако из них .jpg самое популярное расширение на всех платформах.

Алгоритм JPEG является алгоритмом [сжатия данных с потерями](http://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D1%81_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8F%D0%BC%D0%B8).

При сохранении JPEG-файла можно указать степень качества, а значит и степень сжатия, которую обычно задают в некоторых условных единицах, например, от 1 до 100 или от 1 до 10. Большее число соответствует лучшему качеству, но меньшему сжатию, при этом увеличивается размер файла. Обыкновенно, разница в качестве между 90 и 100 на глаз уже практически не воспринимается.

Широкая поддержка формата JPEG в разнообразном ПО нередко приводит к кодированию в JPEG изображений, для того не предназначенных. Даже безо всякого выигрыша по степени сжатия в сравнении с правильно сделанными [PNG](http://ru.wikipedia.org/wiki/PNG) или [GIF](http://ru.wikipedia.org/wiki/GIF), но с прискорбными последствиями для качества. Например, попытка записать в JPEG изображение, содержащее мелкие контрастные детали (особенно, цветные) приведёт к появлению характерных хорошо заметных [артефактов](http://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%82%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D1%8B_%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D1%8F) даже при высокой «степени качества».

При сжатии изображение преобразуется из цветового пространства RGB в YCbCr ([яркостное](http://ru.wikipedia.org/wiki/YUV) Y и два цветоразностных Cb = Y - B, Cr = Y - R). Для цветоразностных пространств можно за счет прореживания уменьшить размеры, это первый шаг сжатия.

Далее, яростный компонент Y и отвечающие за цвет компоненты Cb и Cr разбиваются на блоки 8х8 пикселов. Каждый такой блок подвергается [дискретному косинусному преобразованию (ДКП)](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BA%D0%BE%D1%81%D0%B8%D0%BD%D1%83%D1%81%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Коэффициенты ДКП затем квантуются по уровням. Высокочастотные коэффициенты подвергаются более сильному квантованию, чем низкочастотные. Это приводит к огрублению мелких деталей на изображении. Чем выше степень сжатия, тем более сильному квантованию подвергаются все коэффициенты.

**JPEG 2000 (или jp2)** — графический формат, который вместо [дискретного косинусного преобразования](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BA%D0%BE%D1%81%D0%B8%D0%BD%D1%83%D1%81%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), характерного для [JPEG](http://ru.wikipedia.org/wiki/JPEG), использует технологию [вейвлет-преобразования](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B9%D0%B2%D0%BB%D0%B5%D1%82-%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), основывающуюся на представлении [сигнала](http://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB) в виде суперпозиции некоторых конечных базовых функций — волновых пакетов.

В результате такой компрессии изображение получается более гладким и чётким, а размер файла по сравнению с [JPEG](http://ru.wikipedia.org/wiki/JPEG) при одинаковом качестве уменьшается ещё на 30 %.

**EXIF** (Exchangeable Image File Format) — [стандарт](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82), позволяющий добавлять к [изображению](http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) информацию, комментирующую его. Например, условия и способы его получения, [авторство](http://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%B0%D0%B2%D0%BE). и др. Получил широкое распространение в связи с появлением [цифровых фотокамер](http://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D1%84%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F). Информация, записанная в этом формате, может использоваться как пользователем, так и различными устройствами, например, [принтером](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%82%D0%B5%D1%80).

Разработчик формата — [Japan Electronics and Information Technology Association](http://ru.wikipedia.org/w/index.php?title=Japan_Electronics_and_Information_Technology_Association&action=edit&redlink=1) (JEITA).

**Пример.** загрузки в контейнер PictureBox1 растровой картинки \*.bmp с преобразованием ее в другие форматы сжатия и отображения в PictureBox2.

Пусть выбран тип JPEG с качество 5. При нажатии на кнопку Старт формируется результат – выводится картинка, а в поле результата прописывается тип картинки и размер файла. Тип результата (TIFF, GIF, PNG, JPEG), выбирается из выпадающего списка combobox1 с заголовком «Выбор формата».

При выборе JPEG дополнительно выбирается желаемое качество результата от 5 до 100 из выпадающего списка-диапазона NumericUpDown. заголовком «Качество».

Картинка получилась со сжатием почти в 100 раз, но с плохим качеством. Изменим качество на 30 и повторим преобразование. Результат с хорошим качеством, но сжатие стало меньше, около 30 раз (рис.12.10).

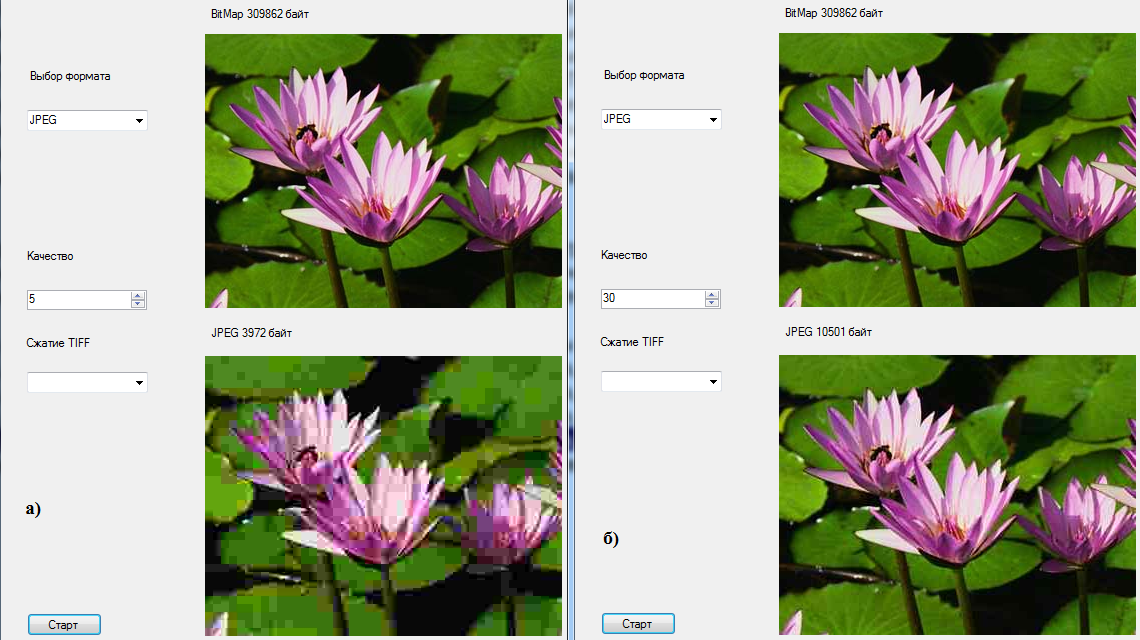


Рис.12.10 – а) Сжатие картинки в 100 раз; б) сжатие в 30 раз

**Итоги лекции.** Изучили представление графики методами C#. Ознакомились с графическими редакторами и способами из задания. С помощью Windos Form.

**Контрольные вопросы**

1. Какие инструменты необходимы для рисования изображения?
2. На что указывает объект Graphics и как он работает?
3. Для чего используются объекты перо, перечислите их.
4. Для чего используются объекты кисть, перечислите их.
5. Метод DrawString, для чего используется, приведите пример.
6. Расскажите о методах рисования.
7. Расскажите о методах заливки.
8. Дайте пример рисования с помощью примитивов.
9. Какие действия выполняются в обработчике при рисовании графиков.
10. Форматы растровой графики перечислите, охарактеризуйте один из них по выбору.
11. Приведите пример работы с растровой графикой.

# Лабораторные работы

Блок лабораторных работ предназначен для закрепления материала и развития навыков программирования.

Состоит из 12 лаб. работ

# Лабораторная работа № 1

# Численные типы в языке C#

**Предмет исследований**

Структура программы на языкеC#. Задание констант, переменных. Типы численных данных и ошибки при преобразованиях типов. Организация простейшего ввода-вывода данных. Программа решения задачи в виде консольного приложения.

**Задание.** Создать консольную программу взаимных преобразований численных типов данных в соответствии с вариантом. В программе должны быть:

Преобразования тип int в другие целочисленные типы.

Взаимные преобразования типа int и типов с плавающей точкой.

Преобразование типа double в типы Single и int.

При преобразованиях нужно проверять получаемые значения и размер в байтах (метод sizeof).

Данные в консоль выводятся инструкцией Console.WriteLine(), в скобках строка. Перед выводом численных данных они должны быть преобразованы в строку методом Convert.ToString(). Для вывода нескольких данных WriteLine() можно использовать двумя способами:

С конкатенацией строк - WriteLine(s1+s2+s3),

С форматным выводом - WriteLine(”{0,L1} {1,L2} {3,L3} “,s1,s2,s3). Имеется одна строка, в которую вставляются форматы вставки подстрок, список имен которых следует за строкой. Формат заключается в фигурные скобки и включает через запятую номер ввода (начиная с нуля) и число пробелов L до вводимой подстроки. В коде программы значения L нужно подобрать, чтобы получит красивую картинку в консоли.

**Варианты к заданию**

|  |  |  |
| --- | --- | --- |
| № | Значение вещественного числа | Значение целого числа |
|  | 123.45 | 150 |
|  | 23.456 | 160 |
|  | 34.567 | 170 |
|  | 456.78 | 180 |
|  | 567.89 | 190 |
|  | 67.890 | 200 |
|  | 78.901 | 210 |
|  | 890.12 | 220 |
|  | 91.012 | 230 |
|  | 109.876 | 240 |

Создать программу взаимных преобразований типов численных данных в соответствии с вариантом.

**Задание 1** Программа преобразование int в целые типы чисел

**Задание 2** Преобразование int в типы double и Single

**Задание 3** Преобразование double в Single и int

В результате должны появиться три отдельные программы рис. 1.1

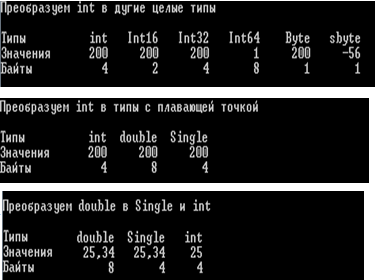


Рис 1. Результат выполнения работы

**Контрольные вопросы**

1. Алфавит языка C#, операции, идентификаторы.
2. Структура программы.
3. Переменные. Их объявление.
4. Форматы представления чисел (с фиксированной и плавающей точкой).
5. Типы целых чисел без знака: Byte.
6. Типы целых чисел cо знаком: int, Int16, Int32, Int64, sbyte.
7. Типы вещественных чисел (с плавающей точкой): double,Single.
8. Совместимость типов при присвоениях.

# Лабораторная работа № 2

# Логические операции

**Предмет исследований:** Логические операции в C#. Разработать алгоритмы решения в соответствии с заданием. Составить программы решения задач.

**Задание:** Выполнить над операндами X и Y операции ИЛИ – OR, И – AND, исключающее ИЛИ – XOR. Выполнить над операндом X логические сдвиги влево и вправо на Z разрядов. Проект – консольное приложение.

**Предварительный расчет:** До выполнения работы произвести соответствующие операции с заданными числами в двоичном эквиваленте с получением десятичного значения.

Результат предварительного расчета сравнить с тем что получили с помощью программы.

**Варианты задания**

Значение чисел X = DE; Y=BC; Z=A+1; Q = «истина» для четных Е; Q = «ложь» для нечетных Е;

**Где ABCDE – последние цифры номера студенческого билета.**

**Примерный результат программы можно просмотреть в открытом окне консоли** (рис.2.1)

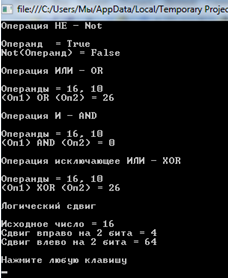


Рис. 2.1

**Контрольные вопросы**

1. Операция НЕ – Not.
2. Операция ИЛИ – OR.
3. Операция И – AND.
4. Операция исключающее ИЛИ – XOR.
5. Логические сдвиги
6. Циклические сдвиги
7. Арифметические сдвиги

# Лабораторная работа № 3

# Линейные структуры

**Предмет исследований:** Запись констант, переменных, стандартных функций. Правила записи арифметических выражений. Арифметические операторы присваивания. Разработка алгоритма решения в соответствии с заданием. Составление программы решения задачи в виде консольного приложения.

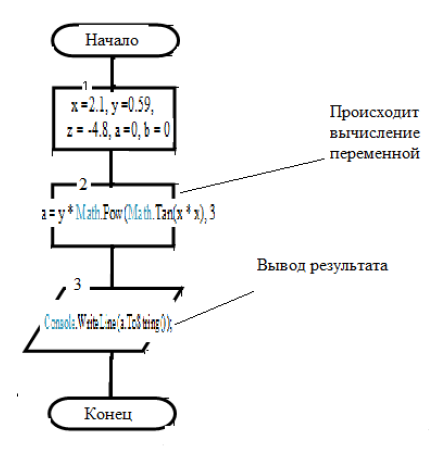
**Задание.** Вычислить значения переменных в соответствии с вариантами задания. Вывести значения вводимых исходных данных и результаты, сопровождая их вывод именами выводимых переменных. Задание выполнить в виде консольного приложения. Номер варианта определяется номером ЭВМ в аудитории (закрепляется на первом занятии на весь семестр).

Ввод данных для вычислений должен производиться с клавиатуры.

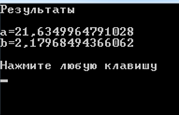
**Работу программы проверить при различных наборах исходных данных не менее 6 раз.** Вводимые данные для вычислений должны содержать положительные и отрицательные вещественные и целые числа.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | № | Задание |
| 1 |  | 8 |  |
| 2 |  | 9 |  |
| 3 |  | 10 |  |
| 4 |  | 11 |  |
| 5 |  | 12 |  |
| 6 |  | 13 |  |
| 7 |  | 14 |  |

**Пример блок схемы**

****

**Внимание**. При вводе данных в консоли разделитель целой и дробной части вещественного числа – запятая. А при вводе данных точка.



**Результат следует сократить до двух знаков после запятой.**

Отчет по лабораторной работе представить преподавателю в следующем виде:

1. Титульный лист
2. Задание.
3. Блок схему
4. Листинг программы
5. Результат выполнения программы.

**Контрольные вопросы**

1. Алфавит языка C#.
2. Операции.
3. Идентификаторы.
4. Типы данных.
5. Структура программы консольного приложения.
6. Где описываются константы, переменные и типы данных?
7. Стандартные функции.
8. Операторы присваивания.
9. Пустая и составная инструкция.
10. Процедуры ввода Read и ReadLine.
11. Процедуры вывода Write и WriteLine.
12. Последовательность действий при выполнении оператора присваивания.
13. Приоритетность выполнения операций в выражениях.
14. Как организовать пропуск одной, двух строк при выводе?

# Лабораторная работа № 4

# Структура ветвление

**Предмет исследований:** Условная и безусловная передача управления; Вычислительные процессы с разветвляющейся структурой. Разработать алгоритмы решения в соответствии с заданием. Составить программы решения задач.

**Ветвление if; else**

**Задание**

Вычислить значения функции по варианту задания. Вывести значения исходных данных и полученные результаты, сопровождая их именами переменных. Значения аргумента взять из указанного диапазона так, чтобы протестировать все ветви программы. Проект – консольное приложение. Значения не определенные исходными данными запрашивать у пользователя.

Работу программы проверить при различных наборах исходных данных не менее 6 раз. Вводимые данные для вычислений должны содержать положительные и отрицательные вещественные и целые числа.

Заполнить отчет

**Примечание. Принципом работы оператора безусловного перехода goto.** Инструкция **goto** используется следующим образом:

В коде программы создается метка с именем (начинается с символа).

Организуется переход на эту метку инструкцией goto «имя метки».

Имя метки в коде обязательно должно заканчиваться двоеточием (:). Оно указывает на точку в программе, с которой будет выполняться программа после использования инструкции goto.

В основном инструкция goto используется с условиями.

**Варианты заданий**

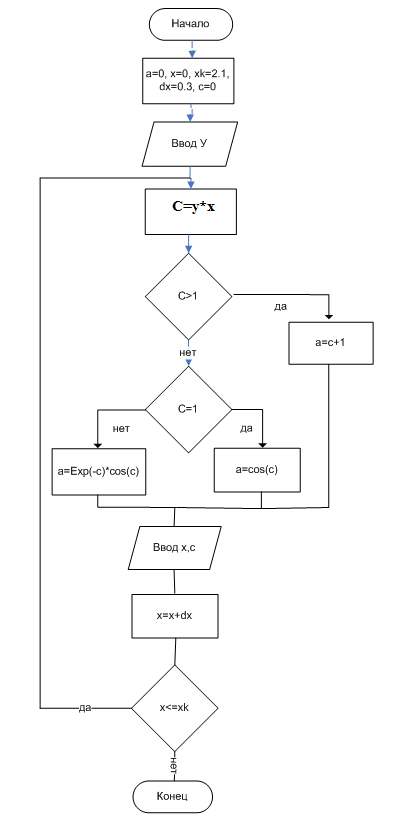
Номер варианта определяется номером ЭВМ в аудитории (закрепляется на первом занятии на весь семестр).

|  |  |  |
| --- | --- | --- |
| Вариант | Задание | Диапазон |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| дополнительно |  |  |

**Пример**. Вычислить при y=1.3, x=[0.. 2.1] с шагом 0.3 значения функции a. Результат вывести в виде таблицы. Проект – консольное приложение.

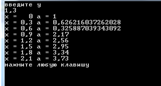


Блок-схема алгоритма представлена на рисунке.



Для организации множественных вычислений введены следующие переменные: **x** - начальное значение, **xk** - конечное значение **dx** - шаг изменения аргумента **x**. Для осуществления повторного вычисления используется проверка условия вхождения переменной **х** в заданный диапазон, и ух=С для упрощения выражения.

Консоль перед закрытием программы:



**Ветвления. Самостоятельное задание.**

Самостоятельно разработать задачу, включающую вложенное ветвление. Условия выбора соответствующих веток выбрать самостоятельно. Функции для веток подобрать самостоятельно, при этом в каждой ветви должно быть не менее двух функций входящих в **Класс System.Math.** Ввод начального, конечного значения вычисляемого диапазона, а также шаг вычисления вводятся пользователем при запуске программы. Результат выполнения выводятся в консоль в виде таблицы.

**Выбор switch; case**

**Задание**

Рассмотреть представленный пример и на основе него самостоятельно разработать программу вычисления математической функции состоящей из ***5 веток. Для выбора ветки используется одна из цифр номера Вашего студенческого билета – ABCDE.***

Производимые вычисления выбрать самостоятельно.

При выполнении программы в зависимости от вводимого целого положительного числа выбирается одна из веток множественного ветвления и производится вычисление с выводом результата. Если введенное значение не найдено выводится сообщение об отсутствии подходящей функции.

**Контрольные вопросы**

1. Какие структуры вычислительных процессов Вы знаете?
2. Как организовать разветвление вычислений?
3. Ветвление if... else.
4. Вложенные ветвления.
5. Инструкция выбора switch.
6. Фраза case.
7. Зачем во фразе case применяется оператор break?

# Лабораторная работа № 5

# Циклы с неизвестным числом повторений

**Предмет исследований.** Организация циклов с неизвестным числом повторений. Инструкции циклов while и do…while. Вычисление суммы членов бесконечного ряда. Разработать алгоритмы решения задачи. Составить программы решения задачи.

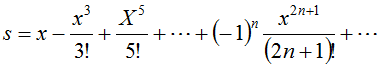
**Цикл while**

**Задание 1**

Вычислить значение суммы членов бесконечного ряда с заданной точностью Е с использованием инструкции цикла **while**. На печать вывести значение суммы и число членов ряда, вошедших в сумму. Проект – консольное приложение.

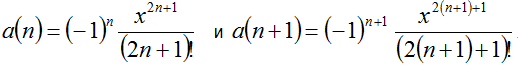


**Пример**. Вычислить значение суммы членов бесконечного ряда

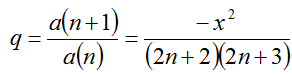


при x = 0.1 с точностью до члена ряда с модулем, меньшим E=0.00001.

Для вычисления очередного члена ряда будем использовать рекуррентное соотношение, связывающее его с предыдущим членом a(n+1) = q\*a(n). Применение рекуррентных формул позволяет избежать вычисления факториала и возведения в произвольную степень. Рекуррентный коэффициент q найдем из выражений для текущего и следующего членов ряда

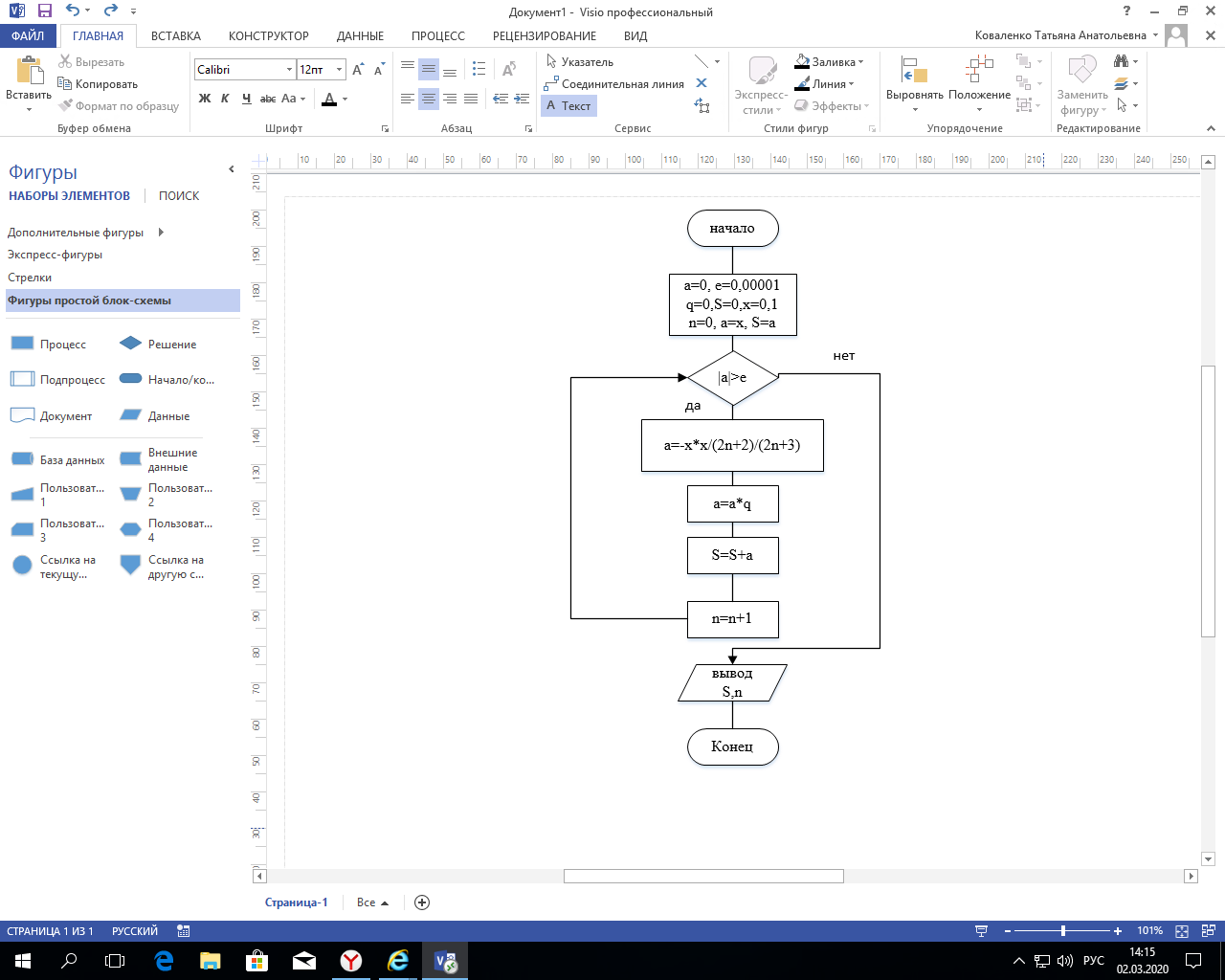


Деля второе выражение на первое, получим



Значение начального члена ряда задаем до цикла путем прямого присваивания (номер начального члена n в разных вариантах равен 0 или 1, правильное значение определяется по формуле текущего члена). В нашем задании n=0, a=x.

**Блок схема**



Консоль перед закрытием программы рис.4.1

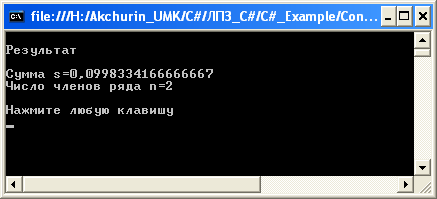


Рис.4.1

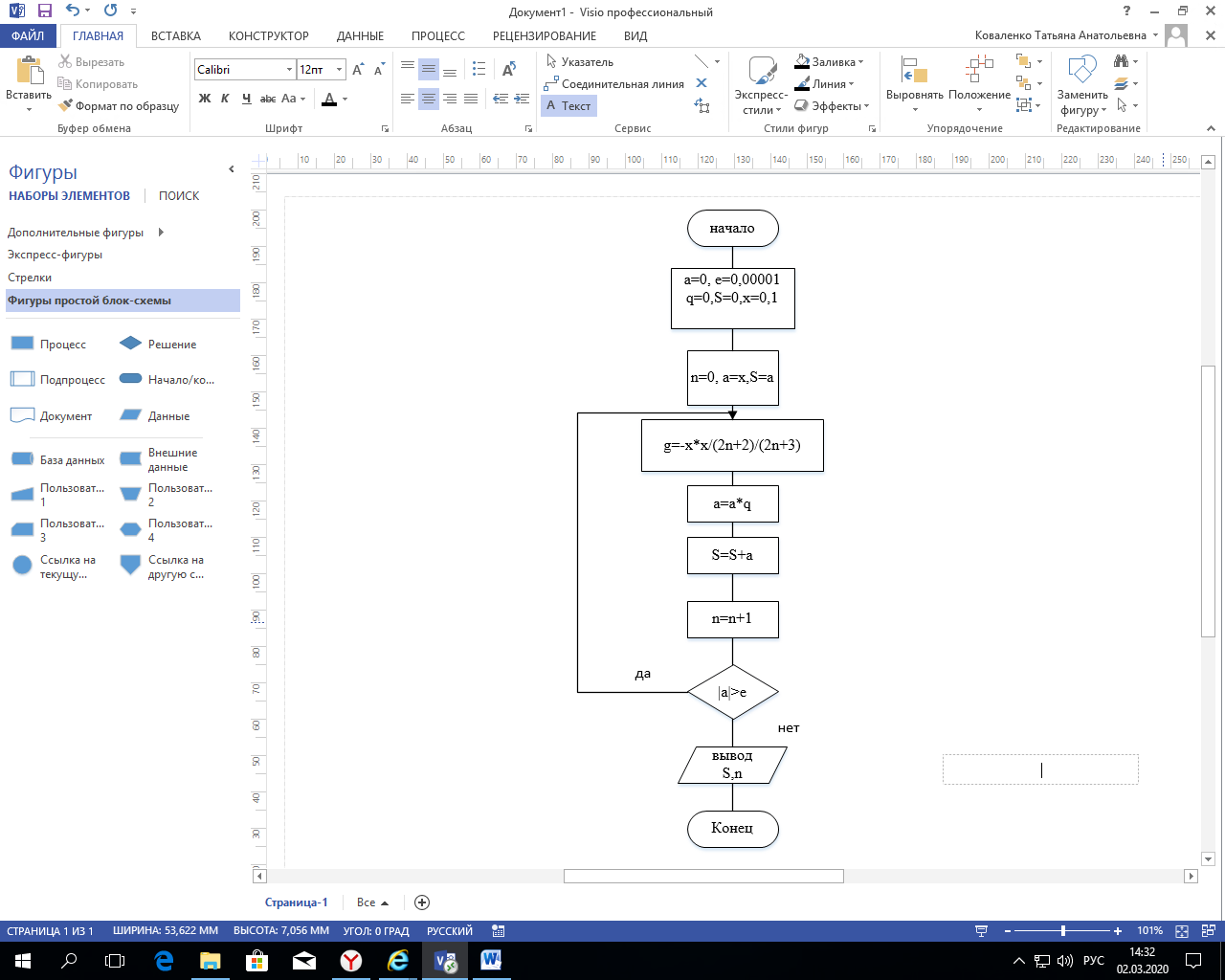
**Задание. Ограничьте значение Суммы до 5 знаков после запятой!**

**Цикл do…while**

**Задание 2.**

Выполнить ту же задачу с применением инструкции цикла do…while. Проект – консольное приложение.

**Блок схема**



**Сравните работы программ while и do…while, объясните чем они отличаются эти процедуры и каков результат.**

**Контрольные вопросы**

1. Циклический процесс с неизвестным числом повторений.
2. Его отличия от цикла с заданным числом повторений.
3. Инструкции языка C# для организации таких циклов. Их сравнение.
4. Синтаксис инструкции while.
5. Как выполнить группу операторов в цикле while?
6. Синтаксис инструкции do…while.
7. Синтаксис инструкции foreach.
8. Прямое вычисление суммы членов бесконечного ряда.
9. Вычисление суммы членов бесконечного ряда по рекуррентной формуле.
10. Условие выхода из цикла при вычислении суммы членов бесконечного ряда.

# Лабораторная работа 6

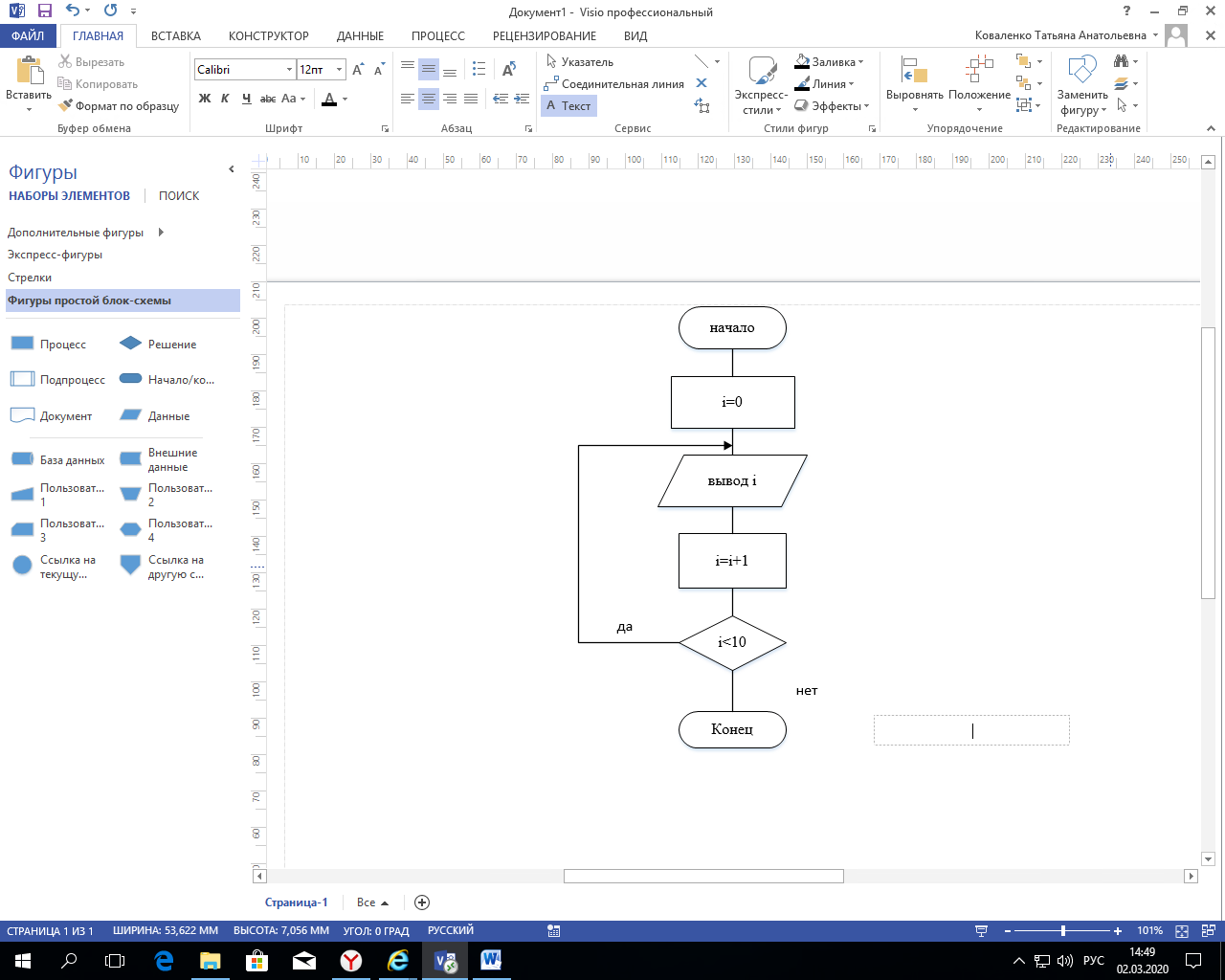
# Циклы с заданным числом повторений

**Предмет исследований.** Организация циклов с известным числом повторений. Инструкция циклов for. Разработать алгоритмы решения задач. Составить программы решения задач.

**Команда Goto и метки**

**Задание 1**. Вывести последовательность чисел (их число две последние цифры в номере зачетной книжки) с использованием инструкции goto и метки. Проект – консольное приложение.

**Блок схема**



Результат прогона рис. 5.1

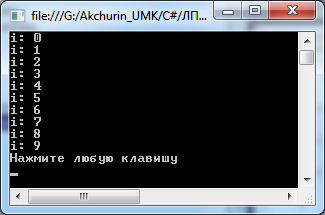


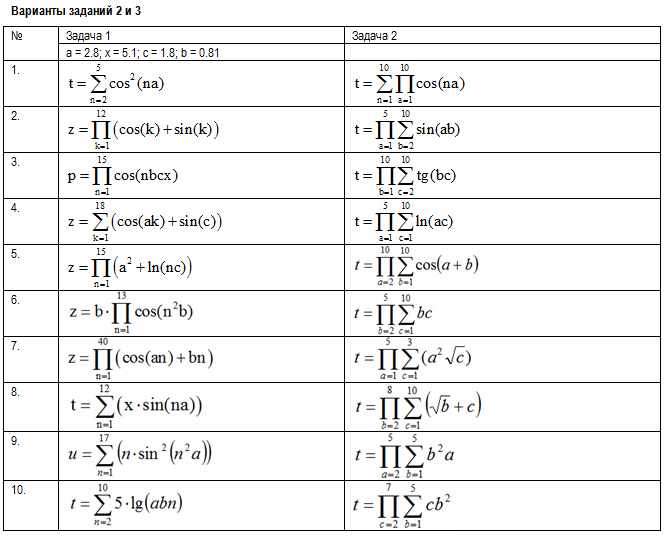
Рис. 5.1– Результат прогона

**Цикл for**

**Задание 2**. Вычислить значения функции с использованием инструкции цикла for. Проект – консольное приложение.

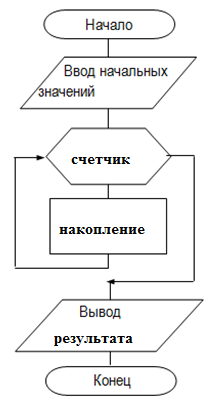
Таблица 5.1

Варианты для 2 и 3 зданий

****

Далее представлена блок схема в общем виде.

**Блок схема**

****

**Вложенные циклы for**

**Задание 3.** Вычислить значения функции с использованием вложенных инструкций цикла for. Вариант задания в таблице Проект – консольное приложение. Пример данного задания рассмотрен в лекции.

**Команда break**

**Задание 4.** Создать программу с использованием команды break. Проект – консольное приложение. В программе ищется первое не простое число из последовательности чисел от N до 1. Варианты заданий – N = две последние цифры номера зачетной книжки.

**Команда continue**

**Задание 5.** Создать программу с использованием команды continue. Проект – консольное приложение. В программе ищется первое нечетное число из последовательности чисел от N до 1. Варианты заданий – N = две последние цифры номера зачетной книжки.

**Контрольные вопросы**

1. Преимущества использования инструкций циклов в программе.
2. Инструкция цикла for.
3. Как организовать цикл с нарастанием индекса?
4. Как организовать цикл с убыванием индекса?
5. Организация вычисления суммы.
6. Организация вычисления произведения.

# Лабораторная работа 7

# Одномерные массивы

**Предмет исследований.** Способы ввода и вывода массивов. Реализация приемов накопления суммы или произведения элементов массивов, запоминания результатов.

**Задание 1**. Обработка одномерного массива.

Таблица 6.1

Варианты задания.

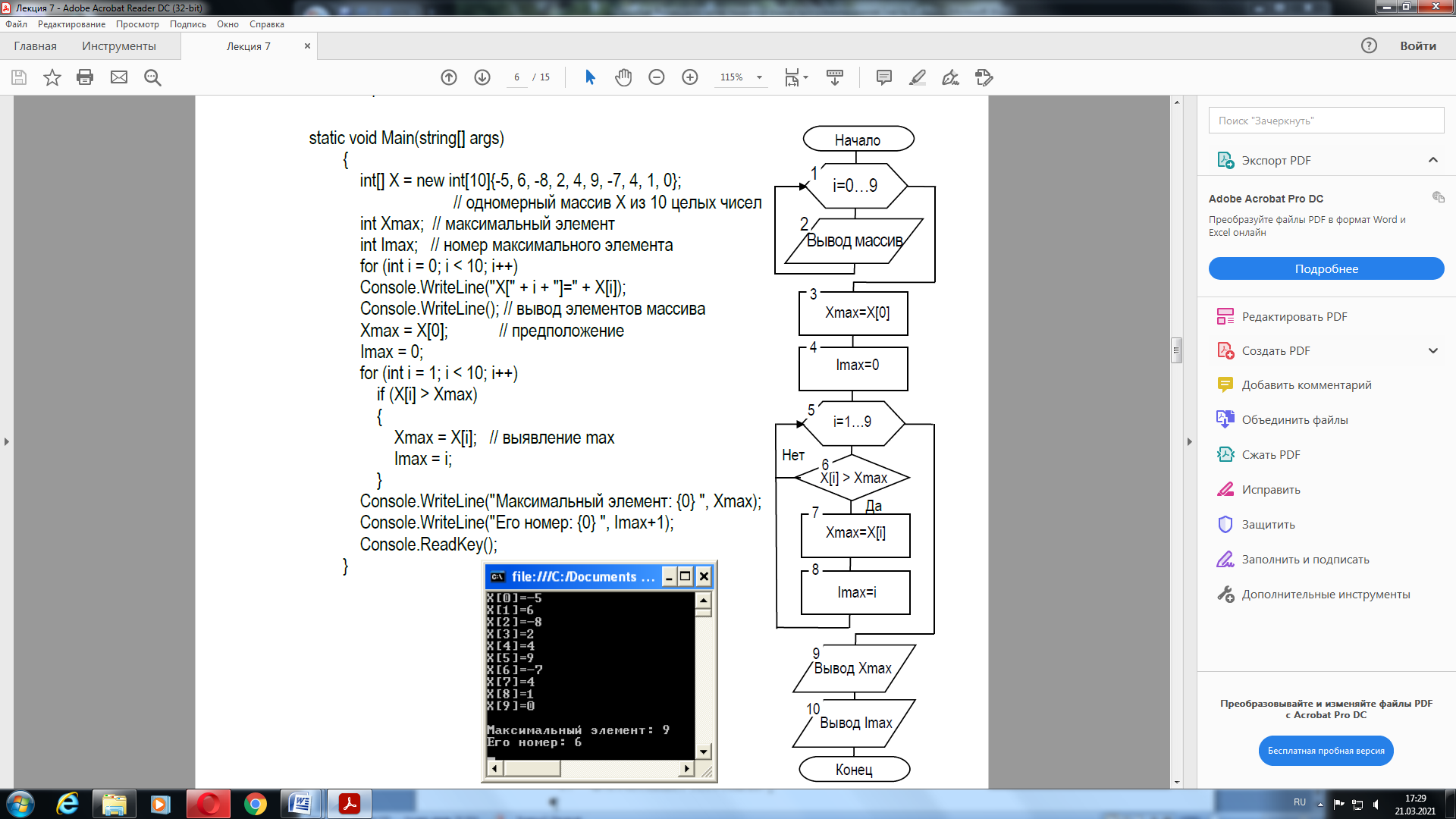
|  |  |
| --- | --- |
|  | Вычислить сумму и количество элементов массива удовлетворяющих следующему условию от -100 до 100. Имя массива задается самостоятельно, размер массива не менее 20. |
|  | Вычислить среднее арифметическое значение элементов массива удовлетворяющих следующему условию: Элементы массива должны быть больше 0. Имя массива задается самостоятельно, элементы массива должны удовлетворять следующему условию от -100 до 100 размер массива не менее 20. |
|  | Задать самостоятельно имя массива, размер массива не менее 20, элементы массива должны удовлетворять следующему условию от -100 до 100. Переписать отрицательные элементы из одного массива в другой. |
|  | Вычислить максимальный из положительных элементов массива и его порядковый номер. Задать самостоятельно имя массива, размер массива не менее 20, элементы массива должны удовлетворять следующему условию от  -100 до 100. |
|  | Вычислить минимальный из отрицательных элементов массива и его порядковый номер. Задать самостоятельно имя массива, размер массива не менее 20, элементы массива должны удовлетворять следующему условию от  -100 до 100. |
|  | Найти минимальный и максимальный элементы массива и поменять их местами. Задать самостоятельно имя массива, размер массива не менее 20, элементы массива должны удовлетворять следующему условию от -100 до 100. |
|  | Найти сумму элементов массива и разделить каждый элемент массива на полученное число. Задать самостоятельно имя массива, размер массива не менее 20, элементы массива должны удовлетворять следующему условию от  -100 до 100. |
|  | Отсортировать массив по возрастанию. Задать самостоятельно имя массива, размер массива не менее 20, элементы массива должны удовлетворять следующему условию от -100 до 100. |
|  | Отсортировать массив по убыванию. Задать самостоятельно имя массива, размер массива не менее 20, элементы массива должны удовлетворять следующему условию от -100 до 100. |

Продолжение табл.6.1

|  |  |
| --- | --- |
|  | Вычислить сумму и среднее арифметическое значения элементов массива удовлетворяющих следующему условию: Элементы массива должны быть больше 0. Имя массив задается самостоятельно, элементы массива должны удовлетворять следующему условию от -100 до 100, размер массива не менее 20. |

**При разработке программы обязательно учитывать вывод исходного массива в окно консоли.**

**Пример**. Вычислить наибольший элемент Xmax массива Х и его порядковый номер Nmax. Размерность массива N=10. Результат – консольное приложение. В программе элементы массива создаются генератором случайных чисел.

****

**Представлена блок схема программы, ваша задача выполнить задание по варианту, при возникновении вопросов обратиться к лекционному материалу.**

**Контрольные вопросы**

1. Что такое массив?
2. Описание типа - массив.
3. Какие операторы языка можно использовать для описания массивов?
4. Особенности организации цикла при обработке массивов?
5. Особенности программирования при обработке массивов?
6. Особенность ввода и вывода массивов?
7. Представление строковых переменных типа String, как одномерных массивов.
8. Вложенные массивы.

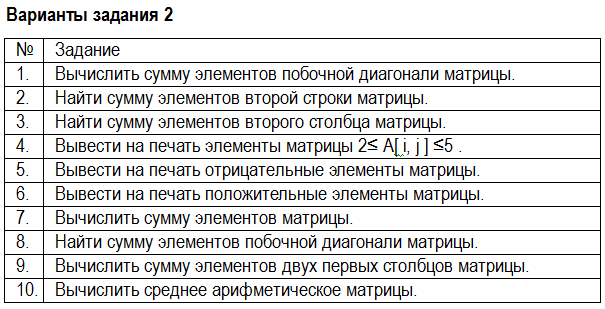
# Лабораторная работа 8

# Обработка двухмерного массива

**Предмет исследований.** Способы описания размеров массивов. Способы ввода и вывода массивов. Реализация приемов накопления суммы или произведения элементов массивов, запоминания результатов.

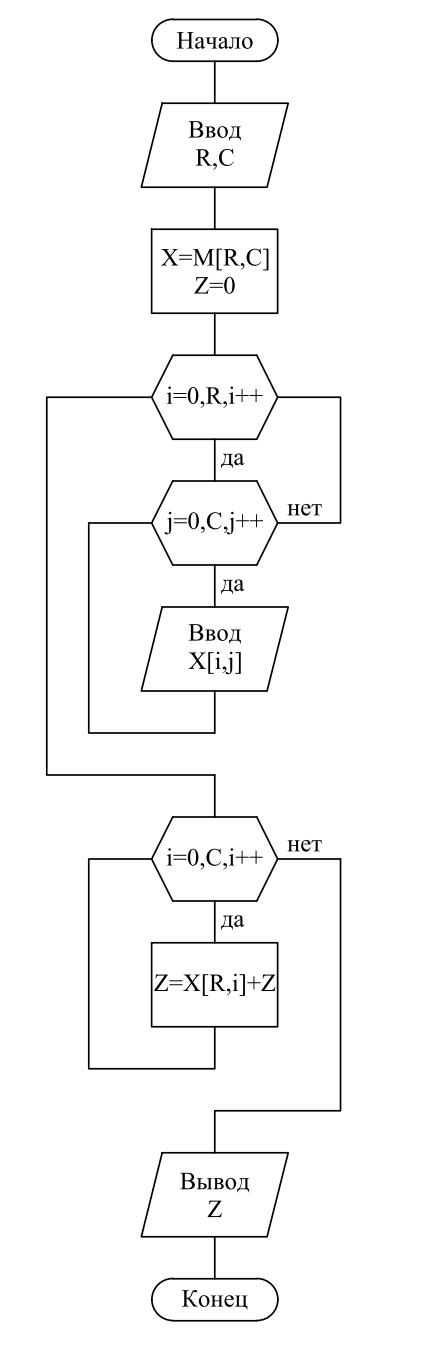
Таблица 7.1

Варианты задания

****

**Пример**. Ввести квадратную матрицу X размером RxC с помощью функции генерирования случайных чисел, допустим ввод с клавиаутуры. При возникновении вопросов обратиться к лекционному материалу.

Представлена блок-схема решения задачи по нахождению суммы последней строки матрицы размером RхC



**Ваша задача выполнить задание по варианту, при возникновении вопросов обратиться к лекционному материалу.**

**Дополнительное задание: Самостоятельно создать задачу для нахождения данных в массиве и решить её.**

**Контрольные вопросы**

1. Отличие задания одномерного массива от двухмерного.
2. Как создать двухмерный массив случайных целых чисел?
3. Как создать двухмерный массив вещественных чисел?
4. Какая инструкция предусмотрена для выполнения итерации элементов в массиве?
5. Каков формат объявления двумерного массива?
6. С чего начинается индексация массивов?

# Лабораторная работа 9

# Подпрограммы

**Предмет исследований**

Правила объявления подпрограмм.

Связь формальных и фактических параметров.

Способы передачи фактических параметров в подпрограмму.

Обращение к функциям.

**Задание**. Создать программу, использующую 2 подпрограммы-функции:

Числа должны вводиться с клавиатуры. Формат числа вещественный, дробная часть вводимого числа не менее 5 знаков.

Первая функция должна возвращатьрезультат произведения чисел.

Вторая функция должна возвращать результат деления двух чисел.

При выводе числа в дробной части отображать только 2 знака.

**Пример. (Обратите внимание, что пример для целых чисел, а в задании для вещественных. При этом в примере 3 разных числа.)**

Создать программу, использующую 2 подпрограммы-функции:

Первая функция должна возвращать наибольший общий делитель NOD двух целых чисел.

Вторая функция должна возвращать наименьшее общее делимое NOK двух целых чисел.

Обе функции в главной программе доложны использоваться с тремя разными целыми числами, вводимыми с клавиатуры. Проект – консольное приложение.

**Листинг программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

class PodProgram

{

// Функция NOD определения наибольшего общего делителя

static int NOD(int x, int y) // Функция NOD

{

if (x != 0)

return NOD((y % x), x); // Рекурсивный вызов

else return y;

}

// Функция NOK определения наименьшего общего делимого

static int NOK(int x, int y) // Функция NOK

{

return (x / NOD(x, y)) \* y;

}

// Главная программа

static void Main()

{

int a, b, c;

Console.WriteLine("Введите через Enter 3 целых числа a b c:");

a = Int32.Parse(Console.ReadLine());

b = Int32.Parse(Console.ReadLine());

c = Int32.Parse(Console.ReadLine());

Console.WriteLine();

Console.WriteLine("Наименьшие общие делимые NOK двух целых чисел");

Console.WriteLine("NOK {0}, {1} = {2}", a, b, NOK(a, b));

Console.WriteLine("NOK {0}, {1} = {2}", a, c, NOK(a, c));

Console.WriteLine("NOK {0}, {1} = {2}", b, c, NOK(b, c));

Console.WriteLine();

Console.WriteLine("Наибольшие общие делители NOD двух целых чисел");

Console.WriteLine("NOD {0}, {1} = {2}", a, b, NOD(a, b));

Console.WriteLine("NOD {0}, {1} = {2}", a, c, NOD(a, c));

Console.WriteLine("NOD {0}, {1} = {2}", b, c, NOD(b, c));

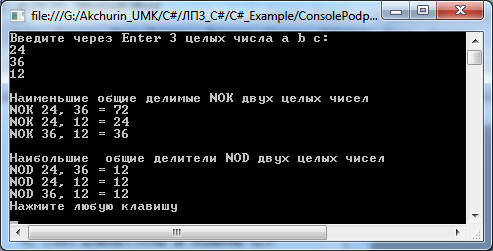
Console.WriteLine("Нажмите любую клавишу");

Console.ReadKey();

}

}

Консоль перед закрытием программы:



**Контрольные вопросы**

1. Что такое подпрограмма? Ее назначение.
2. Правила объявления функций в программе.
3. Правила обращения к функции в программе.
4. Формальные и фактические параметры.

# Лабораторная 10

# Многооконные приложения

**Предмет исследований**

Исследование использование нескольких окон форм (Form) в одном приложении.

Возможность вывода сообщений при работе с приложениями Windows Form.

Открытие и закрытие новых окон в приложениях.

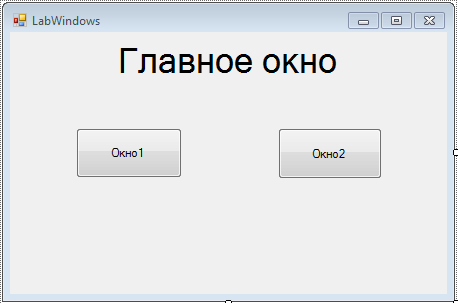
Создание парольного доступа к окнам приложения.

**Задание.** Создайте программу, в которой предусмотрена работа с несколькими окнами, организация парольного доступа к скрытым окном и вывод сообщений об ошибках через MessageBox.

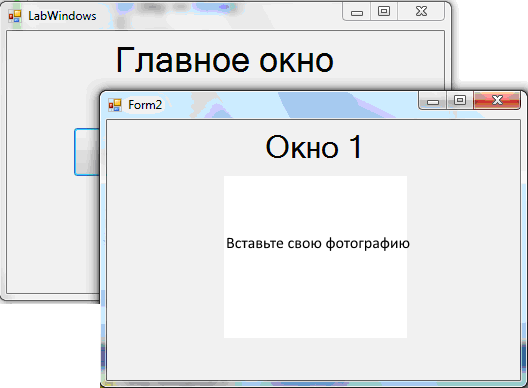
* В примере написано вставьте свою фотографию. Необходимо вставить свою личную фотографию.
* В примере имена форм оставлены такими как создаются автоматически при их создании. Необходимо присвоить заголовкам форм ваше Ф.И.О и номер группы.

**Пример.** В примере показана программа, демонстрирующая работу с многооконными программами и организацию парольного доступа к окнам а так же работу с MessageBox. В программе используются компоненты Button, TextBox и PictureBox. Все эти компоненты можно найти в "Панели элементов" вкладка "Стандартные элементы управления".

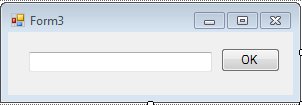
При запуске программы открывается "Главное окно" (форма 1) с двумя кнопками.



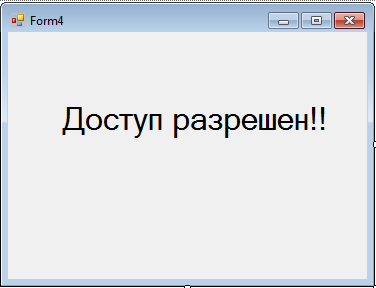
При нажатии на кнопку с именем "Окно 1" открывается “Окно 1” (форма 2) с картинкой. Например



Если пользователь нажимает кнопку "Окно 2", то вызывается форма 4, окно которой не отображается, а из нее вызывается форма 3. Появляется “Окно 3” (форма 3) с запросом пароля.



Если пароль введен правильно, то пользователь получает доступ к “Окну 4” (форма 4).



Окно с запросом пароля автоматически закрываетчя. Если пользователь закрывает окно ввода пароля, то приложение завершается.

Листинг программы (Форма 1)

using System;

using System.Windows.Forms;

namespace LabWindows

{

public partial class Form1: Form

{

public Form1()

{

InitializeComponent();/ // Показать окно 1

}

private void button1\_Click(object sender, EventArgs e)

{

Form2 f2 = new Form2();

f2.Show(); // Показать окно 2

}

private void button2\_Click(object sender, EventArgs e)

{

Form4 f4 = new Form4();

f4.Show(); // Показать окно 4

}

}

}

Листинг программы (Форма 2)

using System.Windows.Forms;

namespace LabWindows

{

public partial class Form2 : Form

{

public Form2()

{

InitializeComponent(); // Показать окно 2

}

}

}

Отображается окно запроса пароля.

Листинг программы (Форма 3)

using System;

using System.Windows.Forms;

namespace LabWindows

{

public partial class Form3 : Form

{

bool key = false;

public Form3()

{

InitializeComponent(); // Показать окно 3

}

private void button1\_Click(object sender, EventArgs e)

{

string pass = "123"; // Это пароль

if (pass == textBox1.Text)

{

key = true;

Form3.ActiveForm.Close(); // Закрыть окно 3

}

else

MessageBox.Show("Ошибка!!! Пароля!!"); // Вывод сообщения

}

private void Form3\_FormClosing(object sender, FormClosingEventArgs e)

{

if (key == false)

{

int a = 0;

Environment.Exit(a); // Закрыть приложение

}

}

}

}

Листинг программы (Форма 4)

using System.Windows.Forms;

namespace LabWindows

{

public partial class Form4 :Form

{

public Form4()

{

Form3 f3 = new Form3();

f3.ShowDialog(); // Показать окно 4

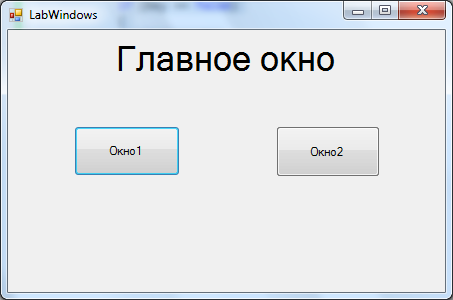
InitializeComponent();

}

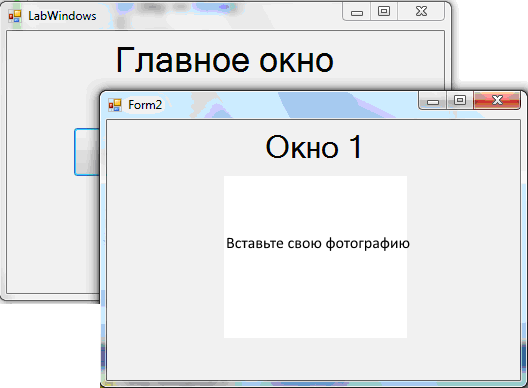
}

}

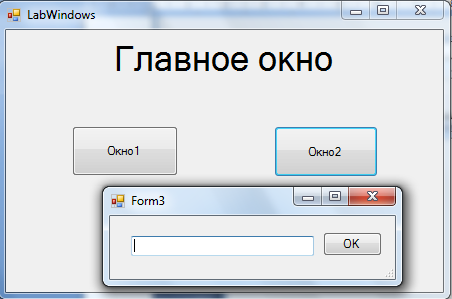
При запуске программы мы видим следующее:



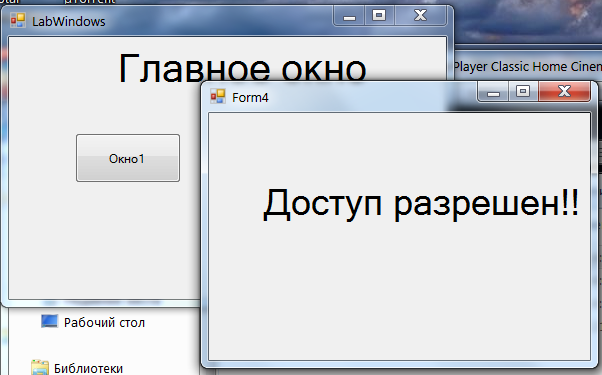
Если нажата кнопка «Окно 1», то



Если нажата кнопка «Окно 2», то



Если пароль правильный, то



**Контрольные вопросы**

1. Открытие новых окон в приложениях.
2. Организация доступа к окнам.
3. Работа с MessageBox.
4. Организация завершение приложения.
5. Интеграция изображений в форму.
6. Закрытие окно в приложении.
7. Работа с событием FormClosing.
8. Объекты Show или Showdialog.

**Лабораторная работа 11**

# Таблицы

**Предмет исследований:**

* Cвойства компонентов label, button, textBox, dataGridView.
* События onClick, onChange, onCreate.
* Как создать код обработчика события.

**Задание**. Создать проект “Таблица конвертации” для конвертации рублей России в разные валюты с учетом комиссии. В таблице предусмотреть возможность редактирования содержимого ячеек, добавления и удаления строк и столбцов.

**Контрольные вопросы**

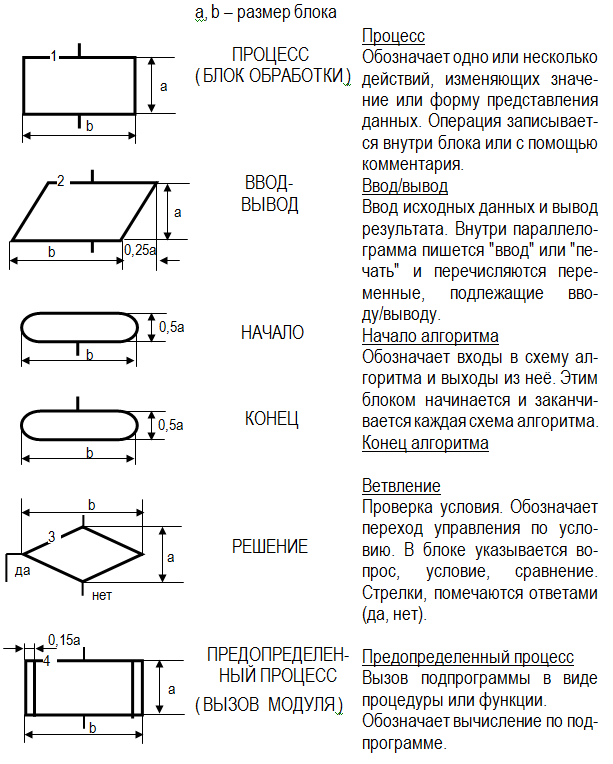
1. События onClick, onCreate.
2. Назначение и свойства компонента кнопка button.
3. Назначение и свойства компонент метка label.
4. Назначение и свойства компонента однострочный редактор textBox.
5. Назначение и свойства компонента dataGridView.
6. Как иници таблицу при старте приложения.
7. Как создать обработчик события onClick.
8. Как использовать строковые данные для вычислений.

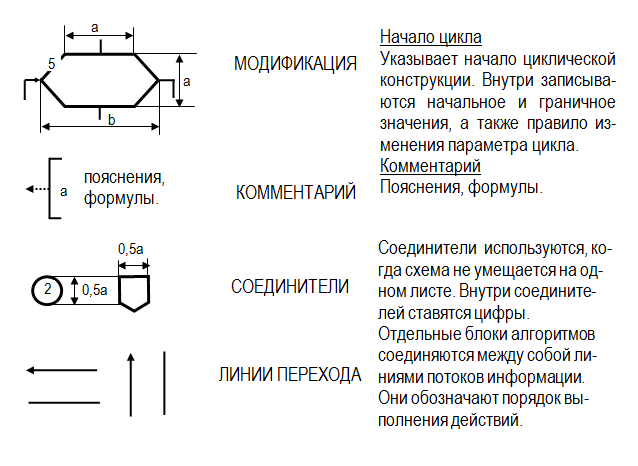
# Приложение А

# Стандартные блок-схемы

**а = 10, 15, 20 . . . мм;**

**b = 1,5 \* а**





Стрелки устанавливаются в конце пути. За основные направления приняты слева – направо и сверху – вниз, в этих случаях стрелки не ставятся.

Блоки схемы имеют сквозную нумерацию сверху – вниз и слева – направо.

# Приложение Б

# Встроенные базовые типы

|  |  |
| --- | --- |
| Описание типа | C# |
| 8-разрядное целое число без знака | byte |
| 8-разрядное целое число со знаком | sbyte |
| 16-разрядное целое число со знаком | short |
| 16-разрядное целое число без знака | ushort |
| 32-разрядное целое число со знаком | int |
| 32-разрядное целое число без знака | uint |
| 64-разрядное целое число со знаком | long |
| 64-разрядное целое число без знака | ulong |
| 32-разрядное с плавающей точкой с обычной точностью | float |
| 64-разрядное с плавающей точкой с двойной точностью | double |
| Логическое значение (true или false) | bool |
| Символ Юникода (16-разрядный) | char |
| 128-разрядное десятичное целое | decimal |
| Корень иерархии объектов | object |
| Строка символов Юникода фиксированной длины | string |

# Типы целых чисел

|  |  |  |
| --- | --- | --- |
| Тип | Диапазон | Размер |
| sbyte | -128 … 127 | 8-разрядное целое число со знаком |
| byte | 0 … 255 | 8-разрядное целое число без знака |
| char | U+0000 … U+ffff | 16-разрядный символ Юникода |
| short | -32 768 … 32 767 | 16-разрядное целое число со знаком |
| ushort | 0 … 65 535 | 16-разрядное целое число без знака |
| int | -2 147 483 648 … 2 147 483 647 | 32-разрядное целое число со знаком |
| uint | 0 … 4 294 967 295 | 32-разрядное целое число без знака |
| long | -9 223 372 036 854 775 808  9 223 372 036 854 775 807 | 64-разрядное целое число со знаком |
| ulong | 0 … 18 446 744 073 709 551 615 | 64-разрядное целое число без знака |
| decimal | -7,9\*1028 … 7,9\*1028  Для финансовых расчетов | 128-разрядное целое число со знаком |

# Типы чисел с плавающей запятой

|  |  |  |
| --- | --- | --- |
| Тип | Диапазон | Точность отображения |
| float | ±1,5e−45 … ±3,4e38 | 7-8 знаков |
| double | ±5,0e−324 … ±1,7e308 | 15 -16 знаков |

# Значения типов по умолчанию

В C# не допускается использование неинициализированных переменных. Что означает, что при объявлении переменных необходимо задать им первоначальные (стартовые) значения, которые в программе могут быть переопределены на другие.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Значение по умолчанию | Тип | Значение по умолчанию |
| bool | false | char | '0' – код с номером 0 |
| byte | 0 | sbyte | 0 |
| int | 0 | uint | 0 |
| short | 0 | ushort | 0 |
| double | 0,0D | float | 0,0F |
| long | 0L | ulong | 0 |

# Приложение В

# Класс System.Math

# Поля класса

|  |  |
| --- | --- |
| Вызов | Действие |
| Math.E | Значение свойства E примерно равно 2,718. |
| Math.LN10 | Значение свойства LN10 примерно равно 2,302. |
| Math.LN2 | Значение свойства LN2 примерно равно 0,693. |
| Math.LOG10E | Свойство LOG10E (константа) приблизительно равно 0,434. |
| Math.LOG2E | Значение свойства LOG2E (константа) приблизительно равно 1,442. |
| Math.SQRT2 | Свойство SQRT2 (константа) приблизительно равно 1,414. |
| Math.PI | Свойство PI является константой, приблизительно равной 3,14159. |

# Методы класса. Имена с заглавной буквы

|  |  |
| --- | --- |
| Вызов | Функция |
| Abs(x) | Абсолютное значение |
| Acos(x) | Обратный косинус |
| Asin(x) | Обратный синус |
| Atan(x) | Обратный тангенс |
| Ceiling(x) | Округление вверх |
| Cos(x) | Косинус |
| Cosh(x) | Косинус гиперболический |
| DivRem(x,y) | Остаток от x/y, числа целые |
| Exp(x) | Экспонента = ex |
| Floor(x) | Округление вниз |
| IEEERemainder(x,y) | Остаток от x/y, числа вещественные |
| Log(x) | Натуральный логарифм |
| Log(x,y) | Логарифм от x по основанию y |
| Log10(x) | Логарифм от x по основанию 10 |
| Max(x,y) | Максимальное из двух |
| Min(x,y) | Минимальное из двух |
| Pow(x,y) | Возводит x в любую степень y |
| Round(x) | Округление до ближайшего целого |
| Sign(x) | Знак числа |
| Sin(x) | Синус |
| Sinh(x) | Синус гиперболический |
| Sqrt(x) | Квадратный корень |
| Tan(x) | Тангенс |
| Tanh(x) | Тангенс гиперболический |