ФГАОУ ВО «КФУ им. В.И. Вернадского»

Физико-технический институт (структурное подразделение)

|  |
| --- |
|  |

Кафедра компьютерной инженерии и моделирования

Скибинский Дмитрий Константинович

отчет по практической работе №1  
по дисциплине **«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Направление подготовки:

09.03.04 "Программная инженерия"

Оценка - 103



Симферополь, 2024

**Практическая работа №1.   
Тема: Использование программных конструкций C#**

**Цель работы:** научиться создавать простейшие консольные и WPF приложения на языке C# в среде Visual Studio, изучить возможность создания самодокументируемых приложений.

Научиться преобразовывать различные типы данных в C#, познакомиться с типом данных Decimal, научиться грамотно использовать циклы для итерационных вычислений с контролем погрешности, обрабатывать события нажатия клавиш, научиться использовать классы String, StringBuilder, научиться создавать самодокументируемые XML справочные файлы.

**Описание ключевых понятий:**   
**IDE** -интегрированная среда разработки IDE (Integrated Development Envirionment) Visual Studio,   
**FCL** - .NET Framework class library - библиотека классов,   
**CLS** - общеязыковые спецификации CLS, решение (solution), проект (project),   
**namespace** - пространство имен , способ организации системы типов в единую группу. Концепция пространства имён обеспечивает эффективную организацию и навигацию по этой библиотеке. Вне зависимости от языка программирования доступ к определённым классам обеспечивается за счёт их группировки в рамках общих пространств имён, сборка (assembly,  
**Windows Forms** — интерфейс программирования приложений, отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде.  
**Windows Presentation Foundation(WPF)** — аналог WinForms, система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая подсистема в составе .NET Framework, использующая язык XAML...

**Перед выполнением лабораторной работы изучена следующая литература:**

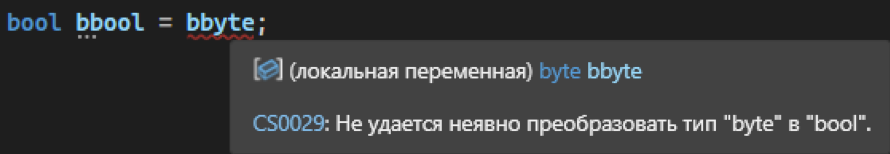
1. Презентация лектора курса: «Основы Net Framework» и «...» (все материалы доступны в облаке на Mail.ru).
2. Прослушана видеолекция
3. Прочитаны 2 лекция по дисциплине «Объектно-ориентированное программирование»
4. Просмотрены практические примеры из видеокурсов по работе с Windows Forms и WPF
5. Получены начальные сведения о спецификации языка C#.
6. Изучены базовые концепции и принципы работы с языком C#

**Выполнены 4 задания, описанных в методических указания к выполнению лабораторных работ.**

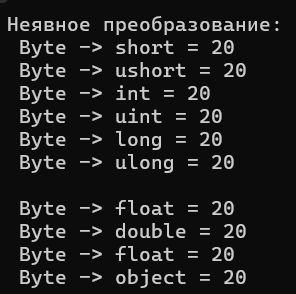
**Задание 1. Преобразование типов**

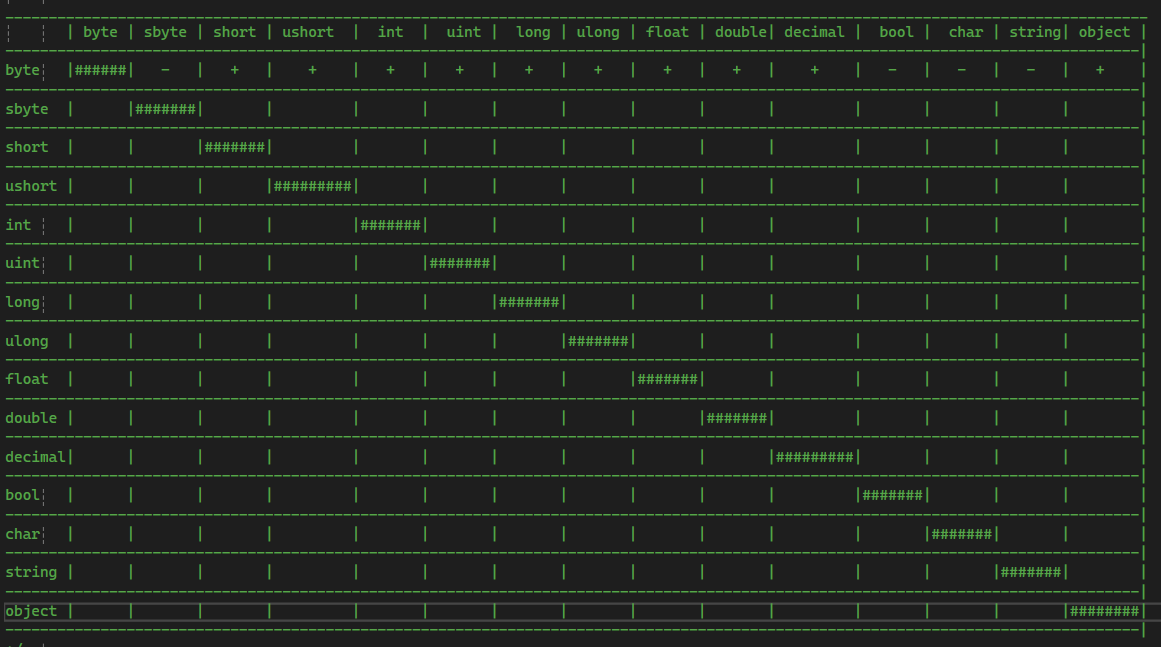
**1. Неявное:**

Для этого нам необходима выбрать переменную какого-то типа, чтобы в дальнейшем пытаться преобразовывать ее во все возможные типы. Для начала возьмем тип byte и попытаемся его преобразовать неявно. Дадим имя переменной bbyte и присвоим ему значение 20.

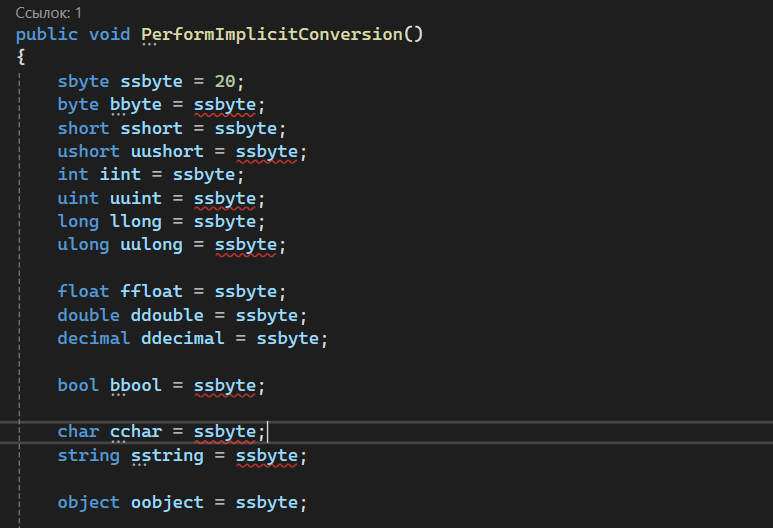
Выписав все преобразования Visual Studio уже, показал, те действия, которые выполнить не удастся:

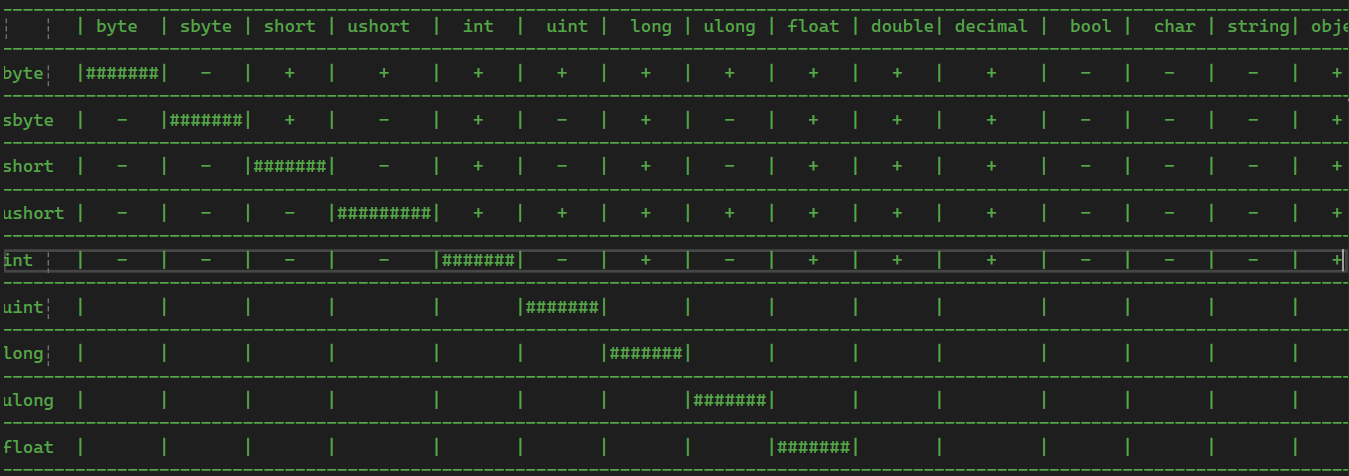
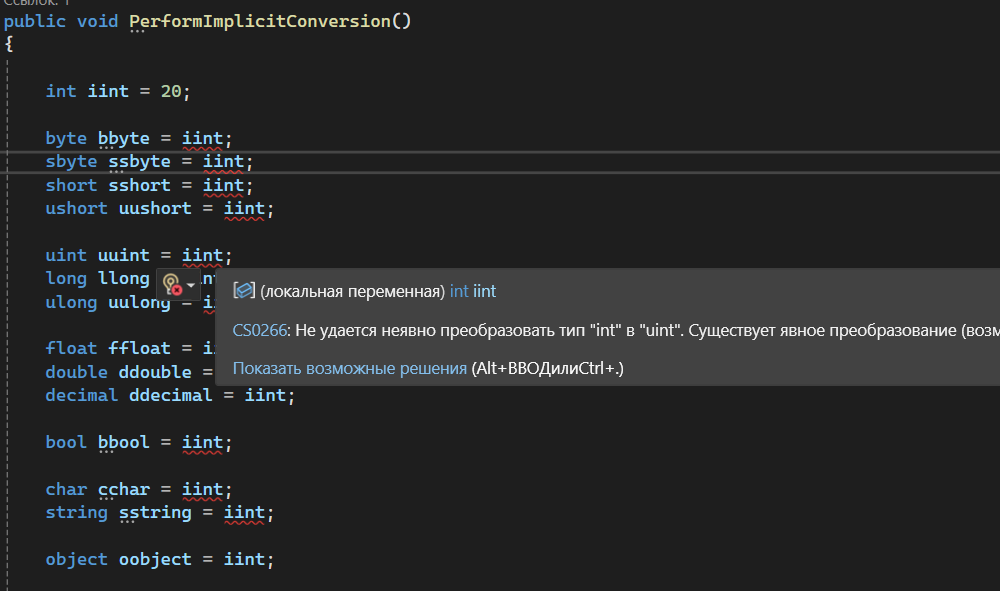
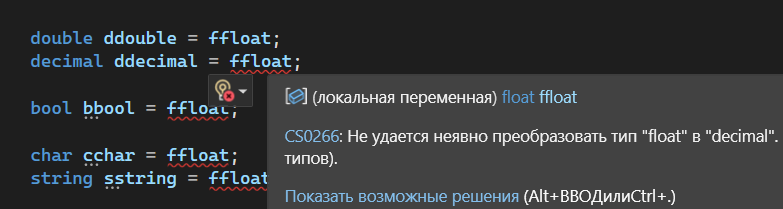
В качестве вывода будем использовать только те переменные, которые удалось преобразовать, чтобы программа корректно отработала. В качестве вывода получим только те преобразования, которые удалось совершить:



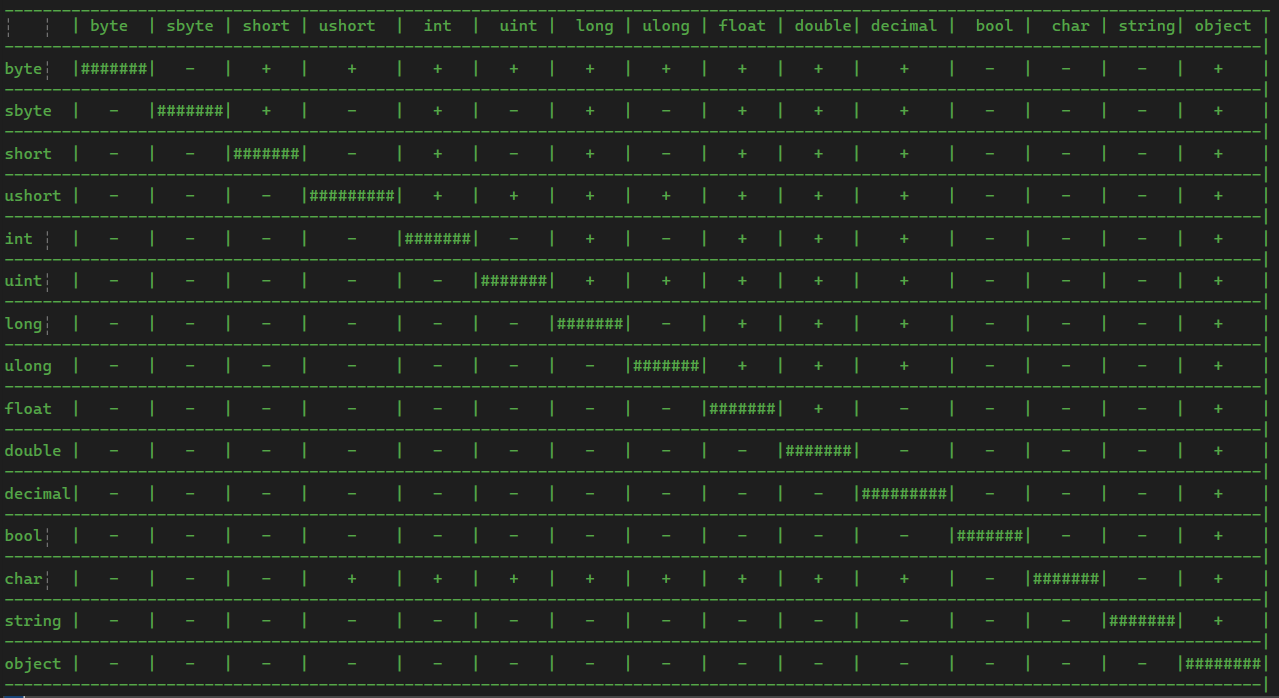
 В комментариях расчертим таблицу со всеми типами данных и заполним на основе полученных данных (первый элемент строки – тот тип, которое мы пытаемся преобразовать; столбцы – типы, в который мы пытаемся преобразовать)

Далее проделаем аналогичный эксперимент с sbyte:

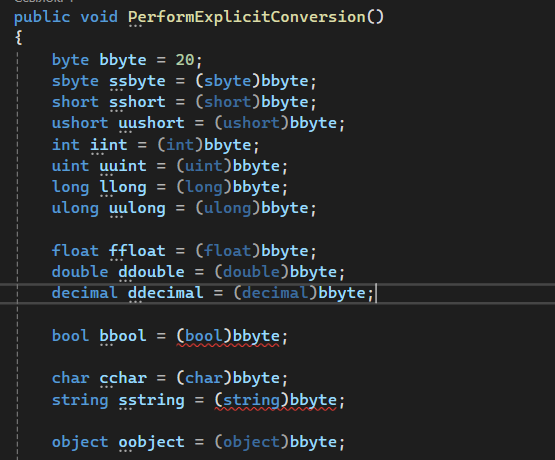


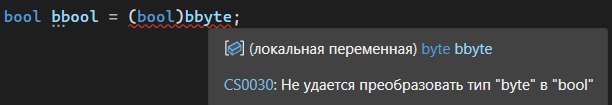
И так далее для всех типов данных. Промежуточные шаги:

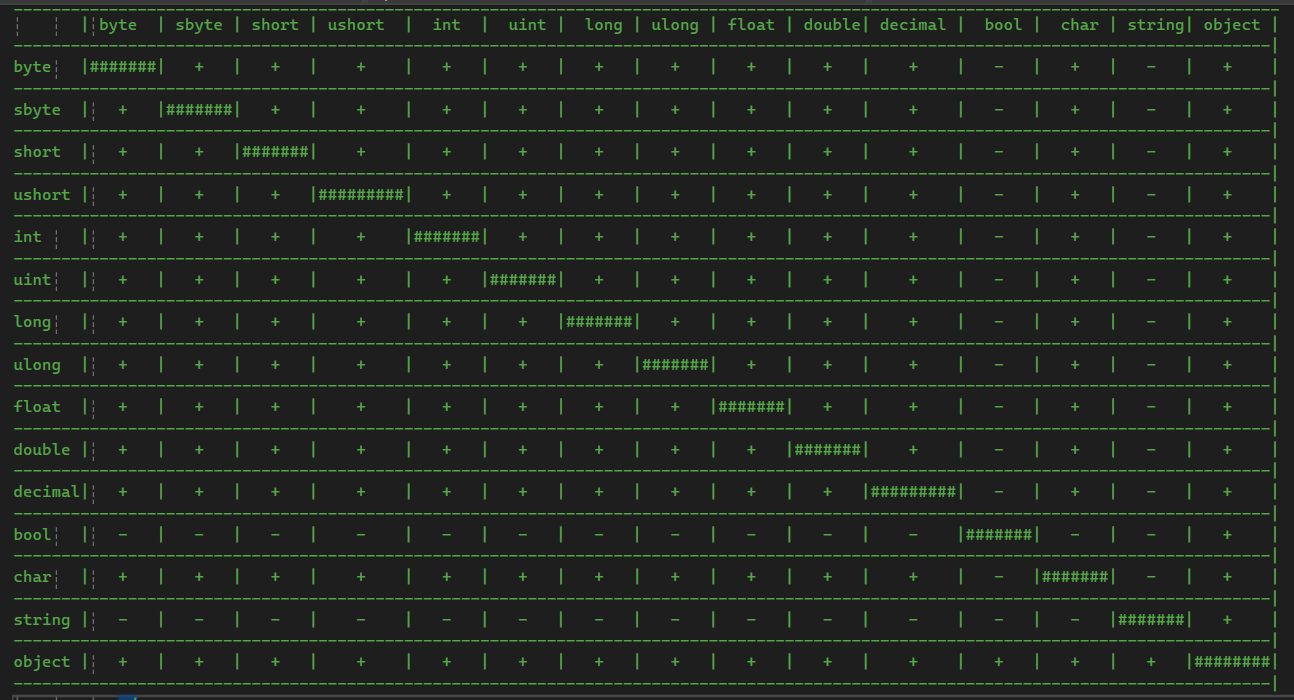
В итоге получим заполненную таблицу:

**2. Явное:**

Явное преобразование (explicit casting) отличается от неявного преобразования (implicit casting) тем, что при явном преобразовании вы должны явно указать тип, в который вы хотите преобразовать значение, например: (int)myDouble. Это может привести к потере данных, если значение не помещается в целевой тип. В отличие от этого, при неявном преобразовании компилятор сам определяет, как безопасно преобразовать значение из одного типа в другой, без необходимости явного указания типа, и, как правило, не приводит к потере данных, так как компилятор выбирает безопасные преобразования.



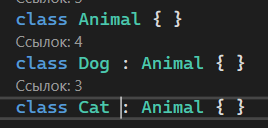
Также начинаем с типа byte и пытаемся его явно преобразовать во все типы. Среда программирования также подсказывает нам, какие преобразования получится сделать, а какие нет:

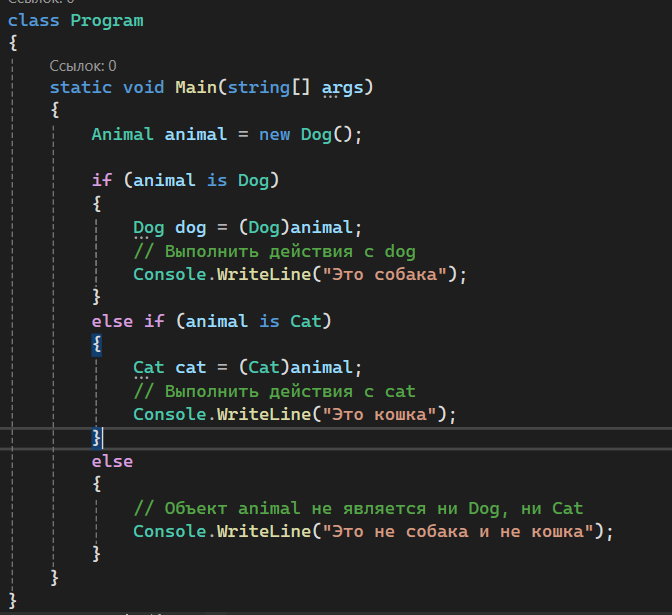
По сравнению с неявным преобразованием, у нас появилось больше типов данных, в которые мы можем преобразовать. Получим результат:

Стандартной ситуацией в явном преобразование оказалось то, что все типы можно преобразовать в любые другие, НО исключениями являются тип Bool и String, которых удается преобразовать только в Object и в которых не удается преобразовать ни один тип, также кроме Object`а.

**3. Безопасное приведение ссылочных типов с помощью операторов as и is**

Для выполнения этого задания создадим несколько простейших классов для приведения:

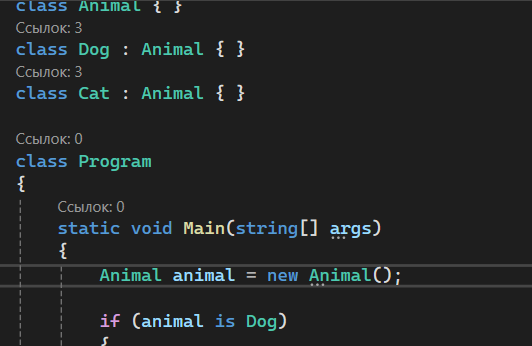


Далее воспользуемся приведением is

В этом примере мы просто попробуем привести схожие классы друг к другу и программа отработала успешно, в качестве вывода получим:

Оператор is используется для проверки, относится ли объект к указанному типу. Он возвращает true, если объект может быть приведён к указанному типу, и false в противном случае.

Чтобы is вернул false, достаточно переменной animal задать тип Animal, который является родителем класса Cat и Dog

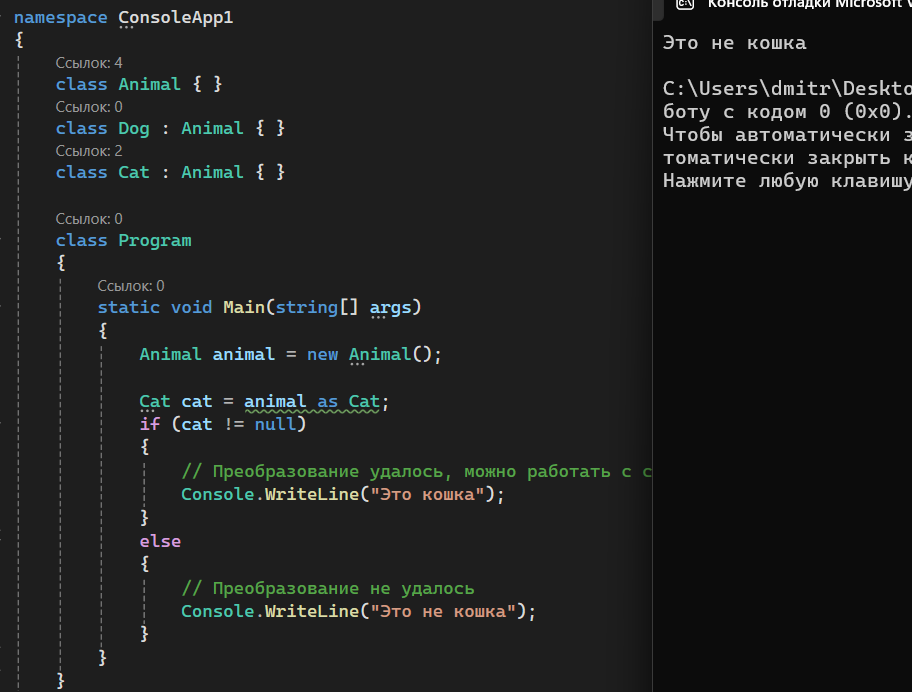


В этом случае вывод будет выглядеть так:



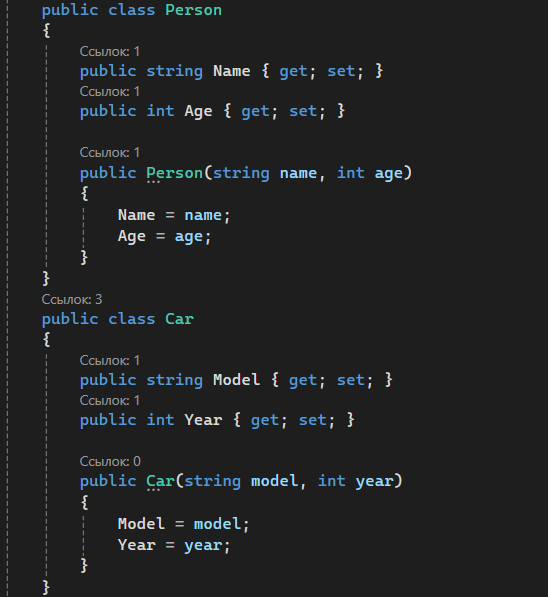
Оператор as выполняет безопасное явное приведение, возвращая null, если приведение невозможно. Возьмем тот же пример и попытаемся привести типы. Также если мы приводим правильно – наследуемого класса к классу родителя, все работает без ошибок

Вывод будет таким же, как и в случае с is. Преобразовав неверно, следует учитывать обработку ошибок сравнивая переменную с null, т.к. as возвращает его, когда преобразование не удалось

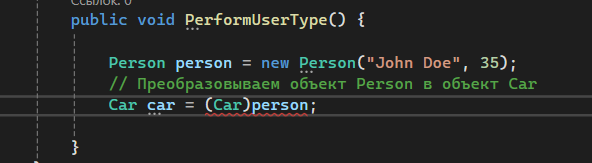


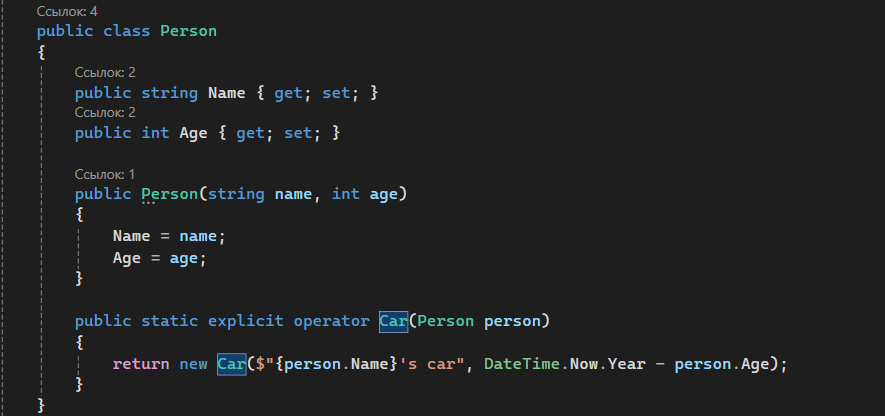
**4. Пользовательские типы данных**

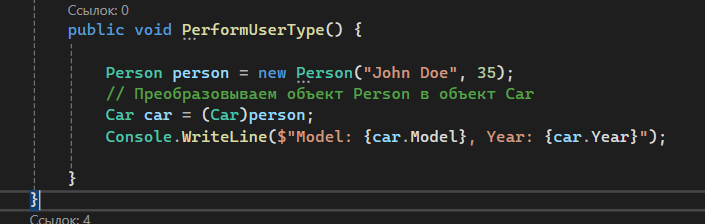
Для начала создадим наши, пользовательские типы данных. Это будут классы Person и Car



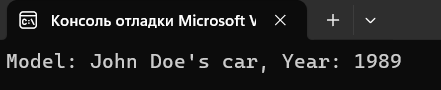
Создадим две переменных. Первую типа Person и передадим ей какие-то значения, вторую типа Car и попытаемся присвоить ему первую переменную. При попытки явного преобразования Visual Studio показывает нам, что сделать этого не удастся:

 Это происходит по причине, что у этих двух классов нет никакого метода, который бы связал эти типы. Однако мы можем это устранить написав такой метод:



Такой статический оператор явного преобразования (*explicit operator Car(Person person)*) определяет, как объект Person может быть преобразован в объект Car. Теперь среда программирования не будет жаловаться на наше преобразование. В качестве значения, которое мы возвращаем это переменная типа Car и пускаем в конструктор два: в значение модели мы добавляем строку чьей машина приходится, в качестве возраста текущий год минус возраст человека.

В качестве вывода мы получим:



**5. TryParse и Convert:**

Для того, чтобы продемонстрировать работу TryParse напишем небольшой код с несколькими преобразованиями. TryParse является методом явного преобразования типов и возвращает значение bool, с помощью чего мы можем с легкостью обработать ошибку.



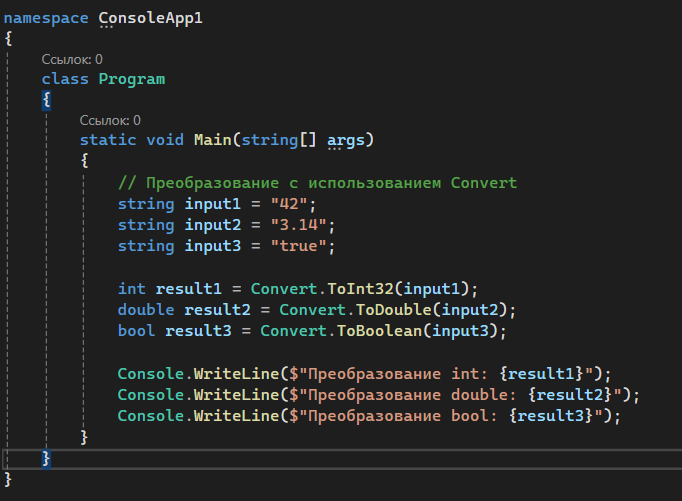
Для того чтобы указать в какой тип нам необходимо преобразовать значение мы указываем тип перед методом. Например: int.TryParse(input, out int result). Input является переменной которую мы хотим преобразовать, а для того, чтобы вынуть преобразованное значение, у метода есть out параметр, который также указывается в скобках.

В качестве вывода получим:

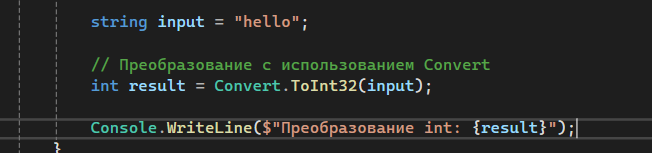


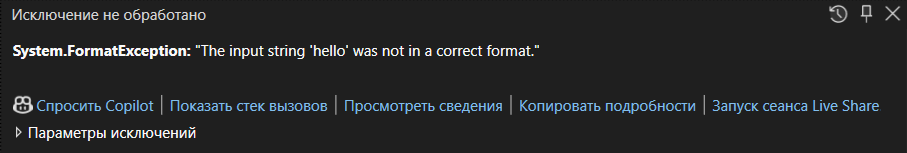
В первом случае нам удалось преобразовать “42” в int, во втором нам не удалось преобразовать “hello” в int

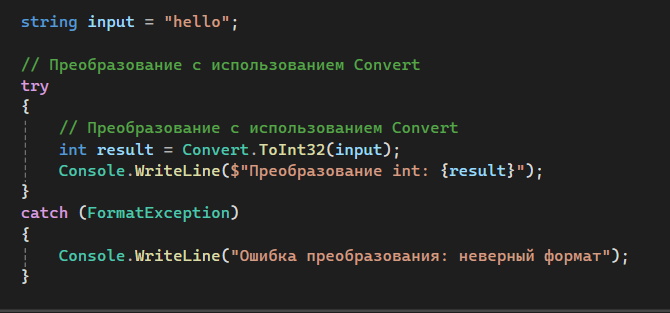
Далее перейдем к методу преобразования при помощи класса Convert. Продемонстрируем большой код. Все три переменные input являются переменными string, и мы попытаемся преобразовать к свойственным им типам:

 Класс Convert предоставляет множество статических методов для преобразования данных между различными типами. Он очень похож на методы TryParse(), но отличается тем, что при неудачном преобразовании он генерирует исключение, а не возвращает false. Это может быть полезно, если мы уверены, что ваши входные данные всегда будут корректными, и не хотите тратить время на проверку ошибок.

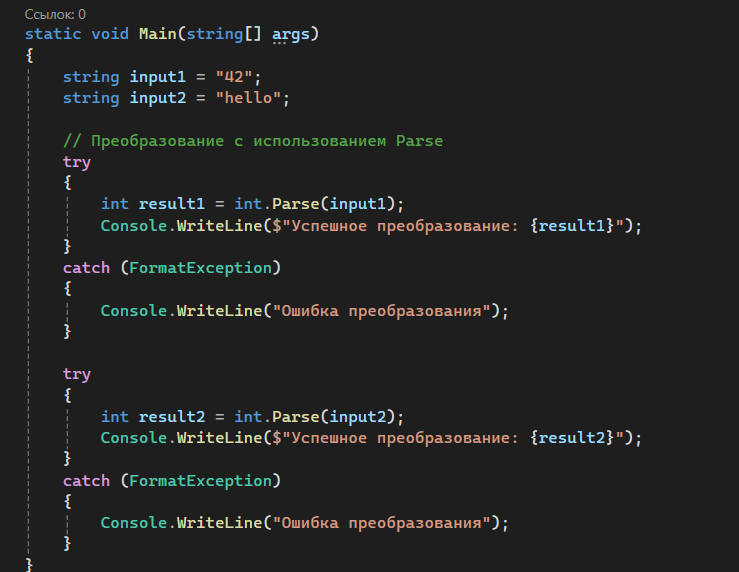
Генерация исключений:



Код запуститься без ошибок, однако при попытке преобразования вылетит ошибка:

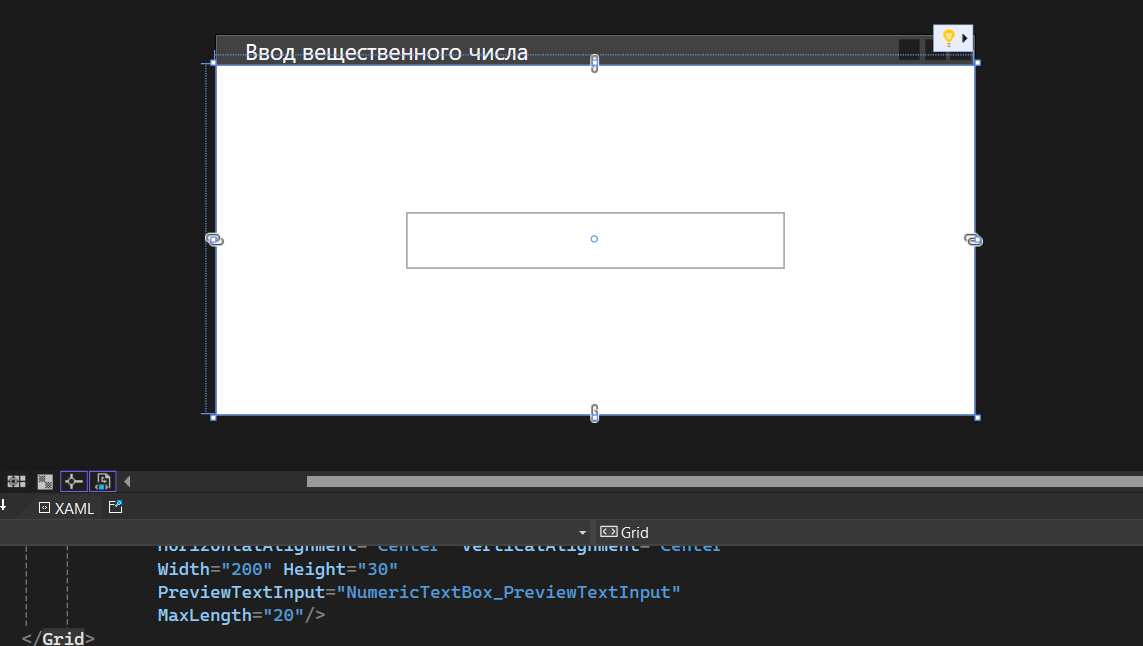
Поэтому можем обработать ошибку вручную:

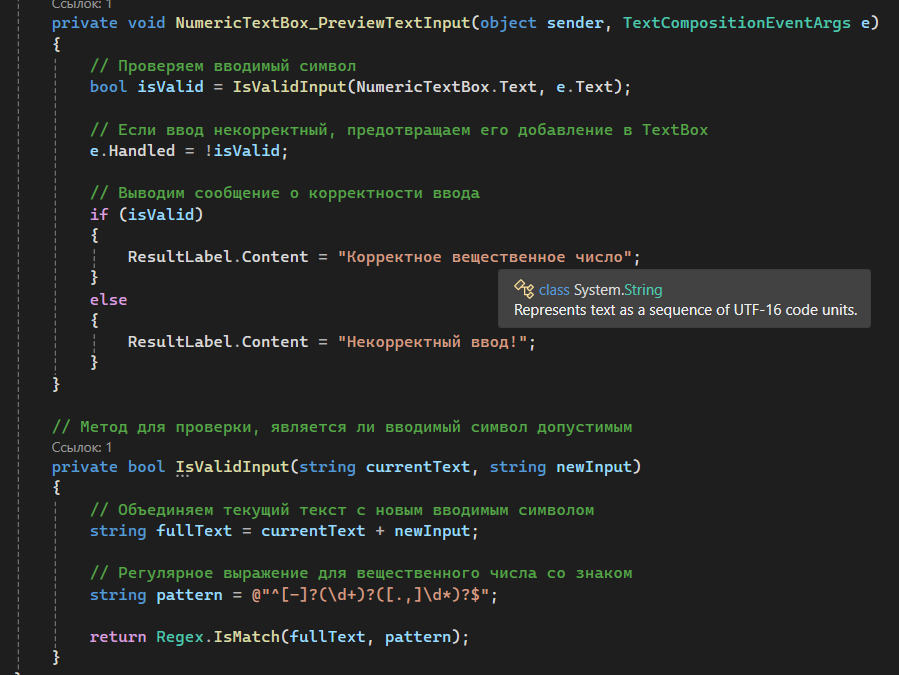
Этот метод похож на обычный метод Parse. Напишем небольшой код, чтобы проверить, как он работает.

 Здесь нам также необходимо обрабатывать исключения. В отличии от TryParse метод возвращает значения преобразованного типа.

**6. Написать программу, позволяющую ввод в текстовое поле TextBox только символов, задающих правильный формат вещественного числа со знаком.**

Для начала создадим окно и TextBox, куда пользователь будет вносить число



После того, как мы считали число, нам необходимо проверить его. Для этого воспользуемся RegEx (регулярными выражениями)

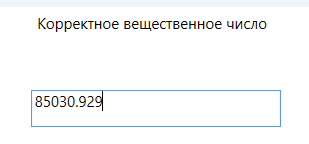
В переменной patter мы указываем что

1. [-]? — допускается один знак минуса.

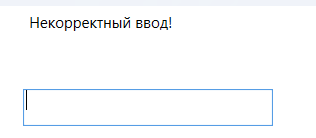
2. (\d+)? — любая последовательность цифр.

3. ([.,]\d\*)? — допускается одна десятичная точка или запятая с последующими цифрами (если есть).

Программа будет постоянно смотреть на ввод, пришло недопустимое изменение, выведет, что число не является вещественным:

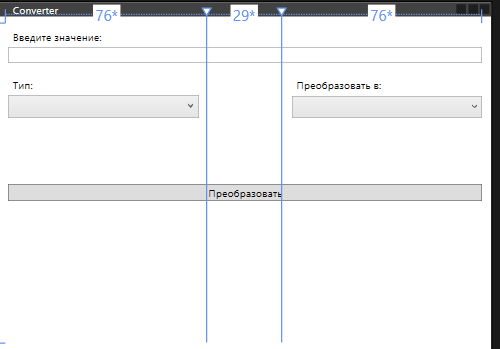


Однако если мы попытаемся ввести букву или запрещенным символ, программа не даст этого сделать и выведет соответствующее сообщение

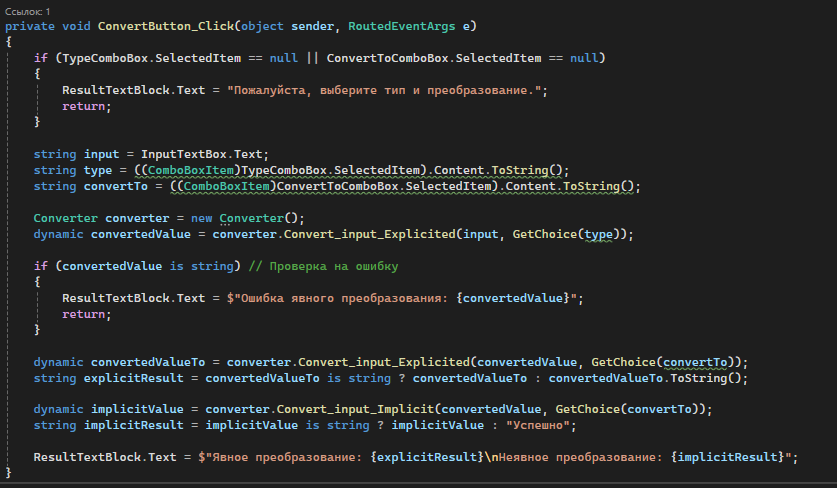


**Углубленное задание. Преобразование любого типа в любой.**

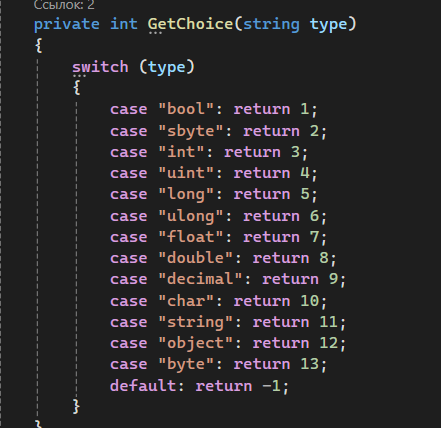
Для начало сделаем интуитивно понятный интерфейс нашего приложения:

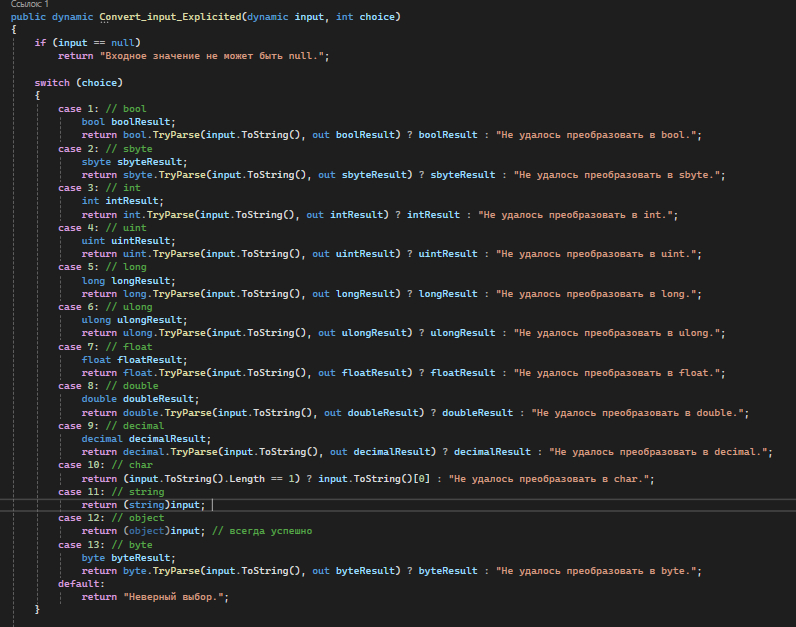


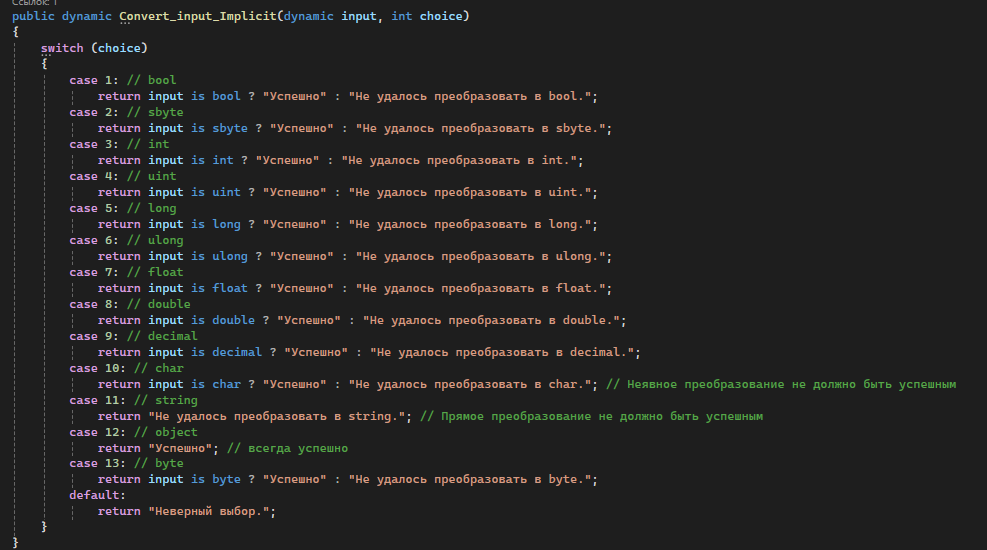
Пользователь будет вводить значение и выбирать какой это тип. Далее ему предстоит указать в какой тип необходимо преобразовать и сама кнопка преобразования. Для выбора используем ComboBox.

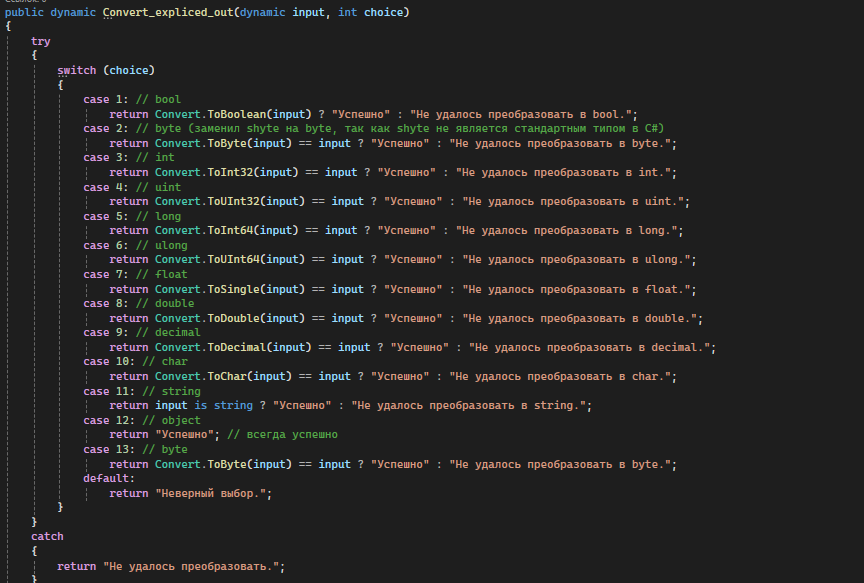
Инициализируем все объекты. Принимать будем значение в string.

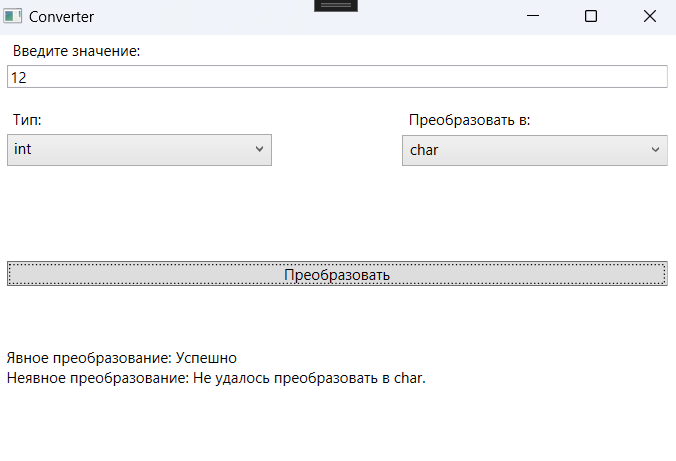
Сделаем класс converter, где будут два метода преобразования типов. В него будем передавать значение пользователя и его выбор. Для выбора напишем простой switch



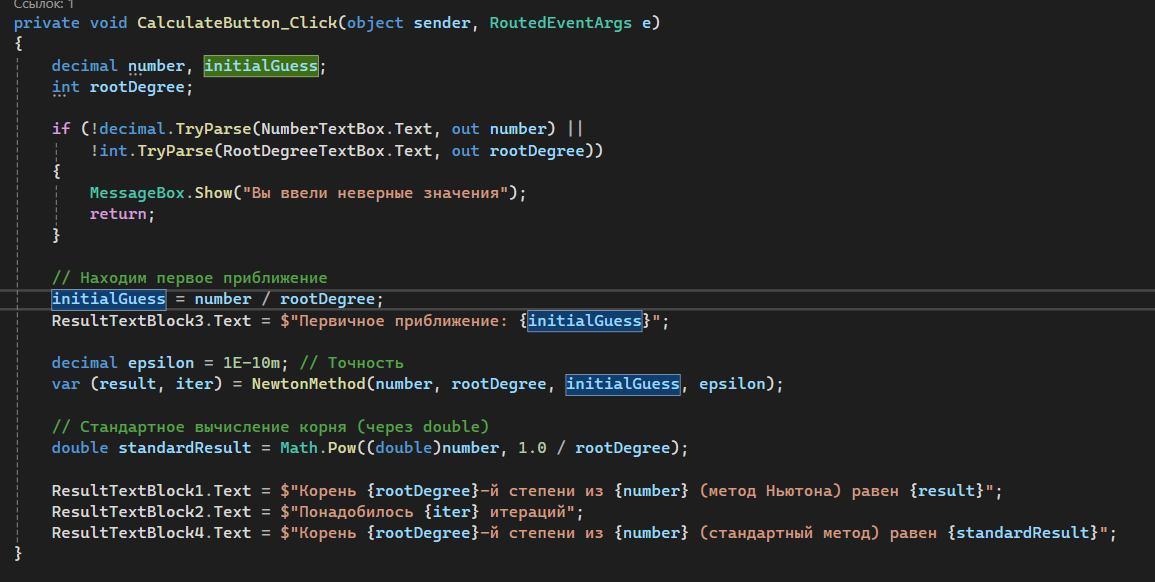
Перейдем к методу преобразования из пользовательского интерфейса. Именно он используется, чтобы перевести в выбранных тип данных введенное пользователем значение. В этом методе используем метод TryParse в связке с тернарным оператором. Если TryParse вернет true, то мы вернем значение типа, которого выбрал пользователь (для этого в начале мы указали, что метод возвращает dynamic – любой тип). Иначе просто возвращаем просто строку, где сообщается, что операция завершилась неуспешно.

Подобным образом напишем метод неявного преобразования

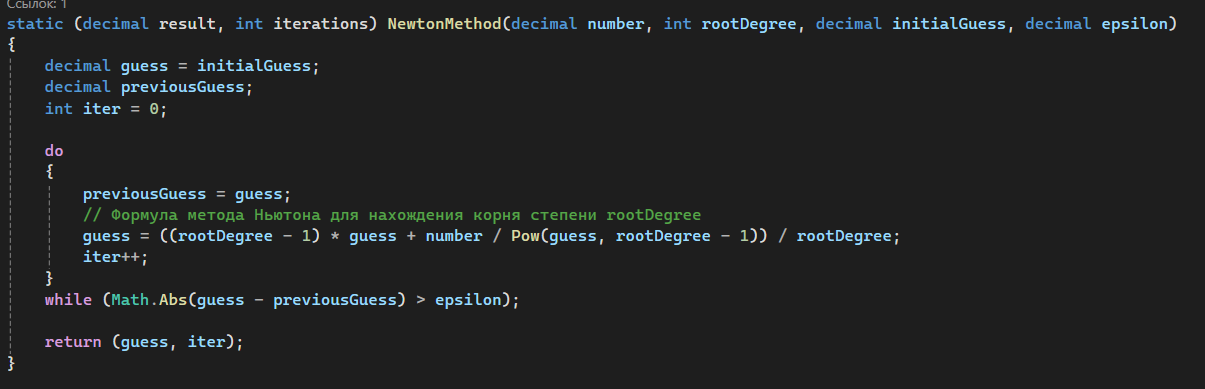
И явного преобразования:

Примеры работы:

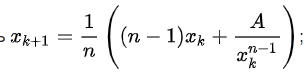
**Задание 2. Нахождение корня при помощи итерационного метода Ньютона**

 Для начала объявим переменные number – число, которое введет пользователь, initialGuess – первое приближение, rootDegree – степень корня.

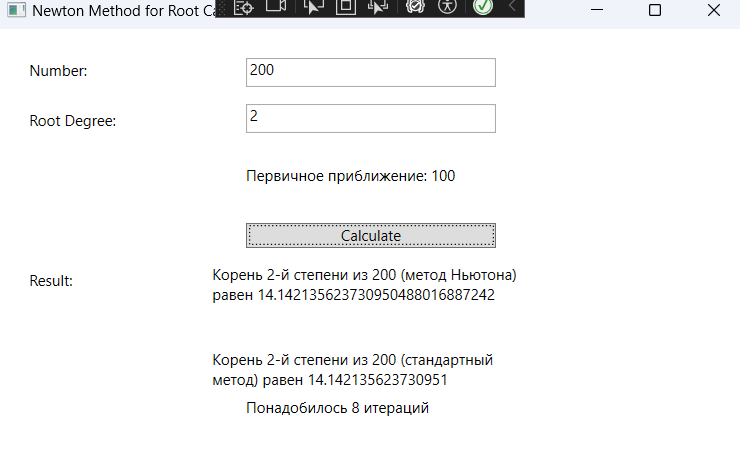
После того, как пользователь ввел число и степень корня, необходимо найти первичное приближение. Точность будет фиксирована и равна 1e-10. Далее введение число, степень корня, начальное приближение и точность передаем методу класса NewtonMethod.

 Обозначим переменную guess и присвоим ей значение первичного приближения, зададим самое большое допустимое значение double`а для переменной difference, которая будет выступать как разность между текущим и следующим приближением, и сделаем счётчик количества итераций.

В переменную newGuess высчитывается новое приближение по формуле Ньютона



Далее сравнивается предыдущее значение и текущее и сохраняется в переменную difference. Следующая итерация наступает если значение difference больше, чем фиксированное значение точности(epsilon). Метод возвращает получившийся результат и количество итераций. Далее эти два значения будут выведены на экран. Теперь сделаем небольшое дополнение и сравним вывод с обычной функцией sqrt.

 Пример работы программы:

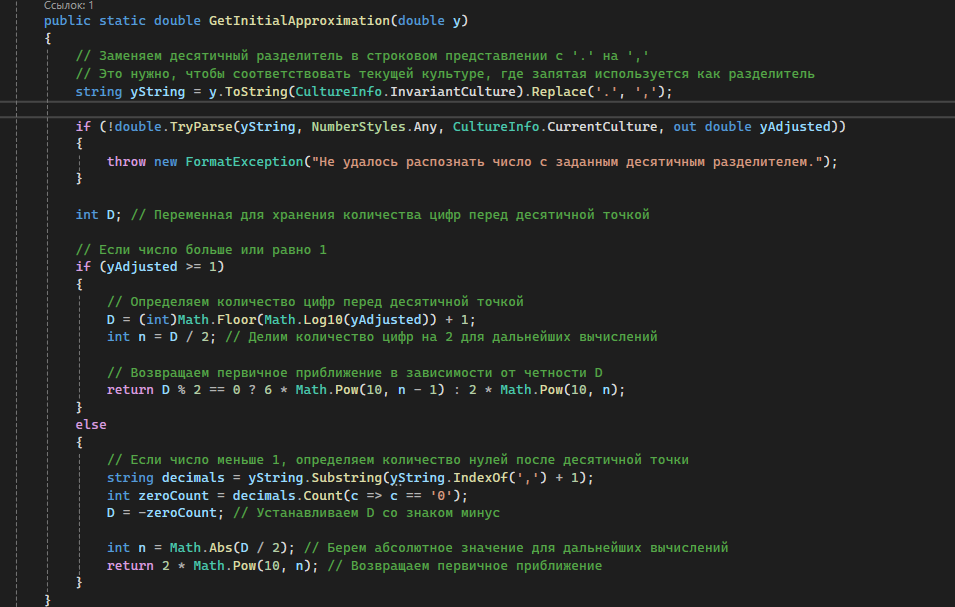
**Углубленное задание. Оптимальный способ нахождения первого приближения.**

Алгоритм: Если число y, из которого вычисляем корень, больше единицы, вычисляем D - число цифр числа y слева от десятичной запятой. Для этого мы воспользуемся десятичным логарифмом и округлением до ближайшего целого.

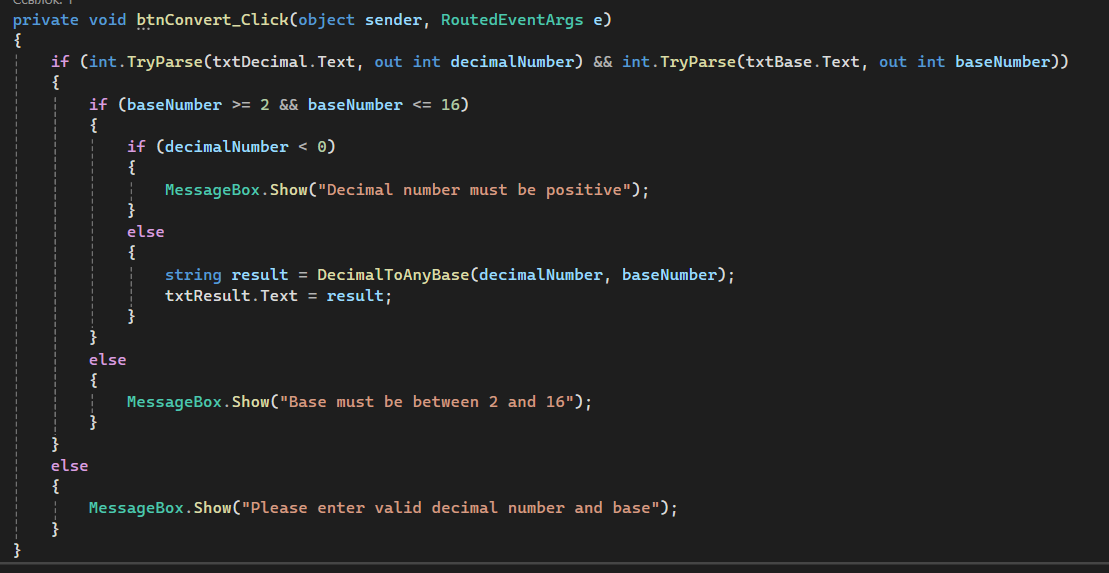
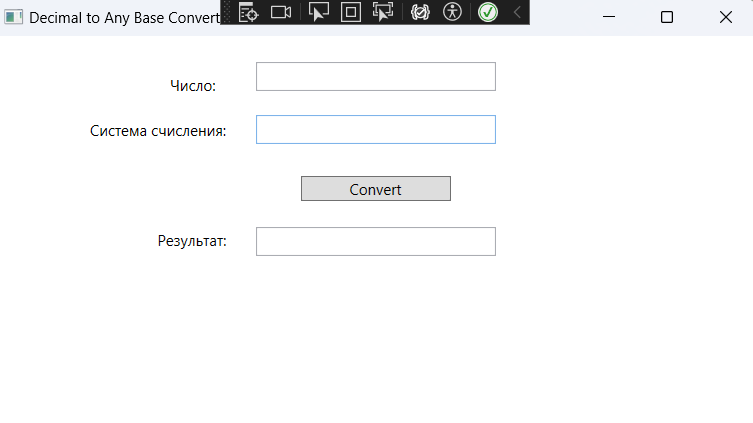
Если y < 1, вычисляем D - число нулей, идущих подряд, справа от десятичной запятой, взятое со знаком минус.

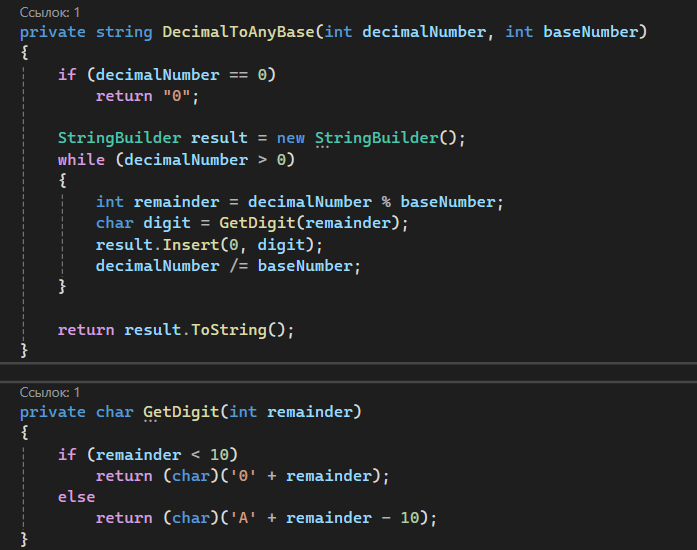
Оценка производиться так: если y нечётно, D = 2n + 1, тогда используем

приближение для корня 2 \* 10n , если D чётно, D = 2n + 2, тогда используем 6 \* 10n .



**Задание 3. Конвертация десятичных целочисленных данных в любую систему исчисления вплоть до шестнадцатеричной**

 Для начала сделаем пользовательский интерфейс и проверку ввода. Неправильным ввод будет если число отрицательное и система счисления меньше 2 или больше 16:

Метод, который преобразует число в любую систему выглядит следующим образом.

Создается объект StringBuilder, который будет использоваться для построения результирующей строки. StringBuilder более эффективен для динамического построения строк.

Цикл while выполняется, пока десятичное число больше 0. Вычисляется остаток от деления десятичного числа на базовое число. Этот остаток будет последней цифрой в результирующей строке. Вызывается метод GetDigit() для получения символа, соответствующего этому остатку (цифра или буква от A до F). Этот символ добавляется в начало строки результата с помощью метода Insert().

Метод GetDigit() работает следующим образом

1. Проверка на диапазон 0-9:

- Сначала мы проверяем, попадает ли remainder в диапазон от 0 до 9.

- Если это так, то мы можем просто преобразовать remainder в соответствующий ASCII-символ цифры.

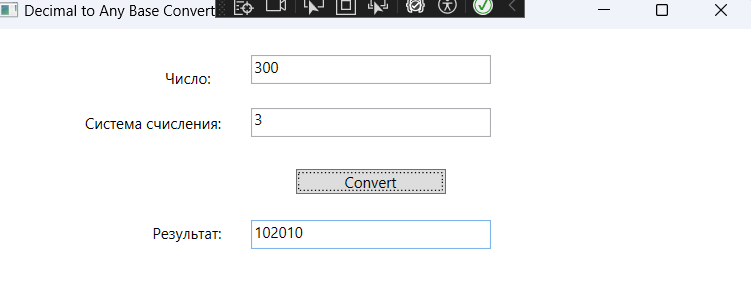
- Это делается с помощью арифметической операции '0' + remainder. Символ '0' имеет ASCII-код 48, и, добавляя к нему remainder, мы получаем символ, соответствующий нужной цифре.

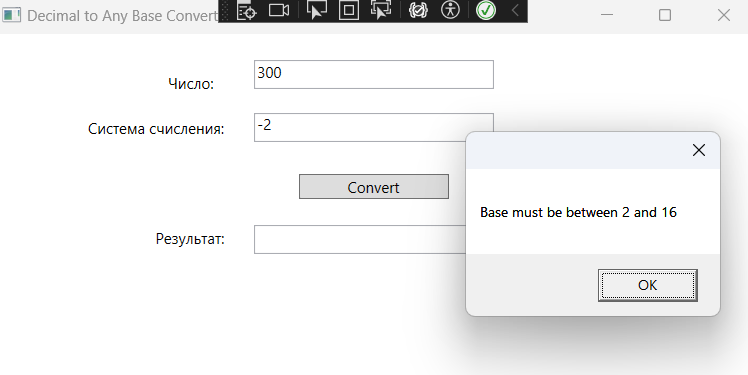
2. Диапазон 10-15:

- Если remainder больше или равен 10, то это означает, что мы имеем дело с буквой от A до F.

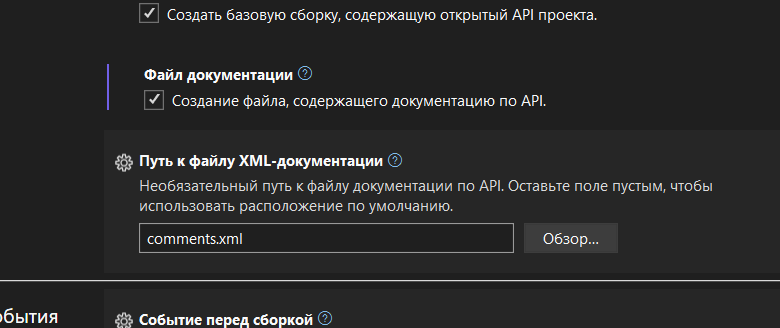
- В этом случае, мы вычисляем символ, соответствующий букве, используя операцию 'A' + remainder - 10.

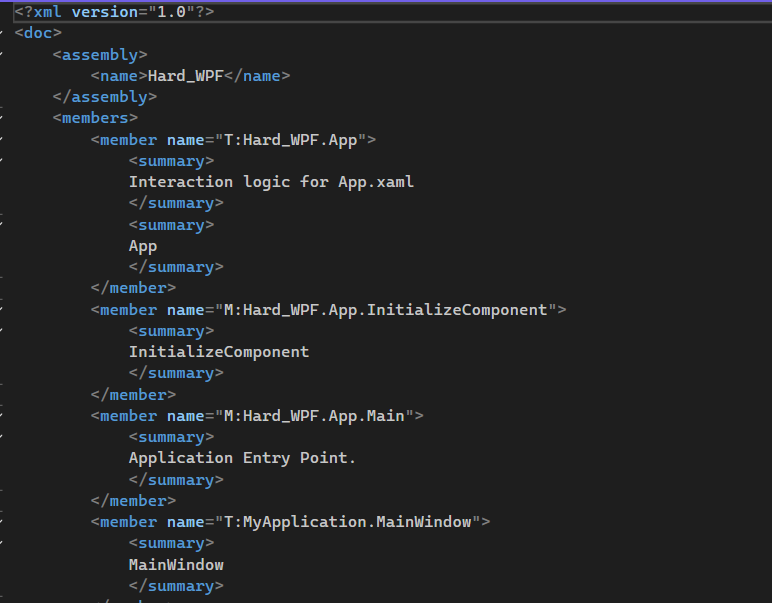
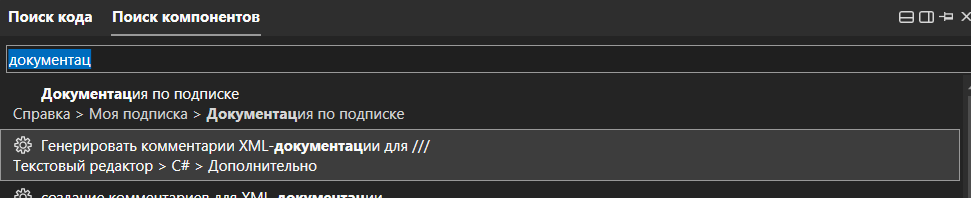
- Символ 'A' имеет ASCII-код 65, и, добавляя к нему remainder - 10, мы получаем символ, соответствующий нужной букве.

Пример работы:



**Задание 4. Создание документации для приложения.**

 В поиске инструментов Visual Studio найдем Путь к файлу XML-документации. Укажем имя файла comments.xml

Также в поиске инструментов найдем инструмент генерации xml документации

В папке текущего проекта создалась документация

**Ответы на вопросы:**

**1. Какие типы преобразаваний переменных вы знаете?**

* + Неявное преобразование (Implicit Conversion): Автоматическое преобразование типов, которое выполняется компилятором без потери данных. Например, преобразование int в long.
  + Явное преобразование (Explicit Conversion): Преобразование, которое требует явного указания типа и может привести к потере данных. Например, преобразование double в int.
  + Преобразование с использованием методов: Использование методов, таких как Convert.ToInt32, Parse, TryParse, и т.д.
  + Пользовательское преобразование: Определение пользовательских операторов преобразования в классах.

**2. Можно ли неявно преобразовать тип Integer в Char?**

Нет, неявное преобразование int в char невозможно, так как это может привести к потере данных. Для этого требуется явное преобразование, например, с использованием приведения типов: char c = (char)intValue;.

**3. Может быть потеряны значащие цифры при неявном преобразовании? Приведите пример.**

Нет, неявное преобразование обычно не приводит к потере данных, так как оно выполняется только тогда, когда преобразование безопасно. Однако, если речь идет о преобразовании с потерей данных, это будет явное преобразование. Например, преобразование double в int может привести к потере дробной части.

**4. Может быть потеряно значение переменной при явном преобразовании? Приведите пример.**

Да, при явном преобразовании может быть потеряно значение. Например, при преобразовании double в int теряется дробная часть:

double d = 123.456;

int i = (int)d; // i будет равно 123

**5. Чем метод TryParse отличается от Parse? Что возвращает метод TryParse?**

* Parse: Преобразует строку в указанный тип данных. Если преобразование невозможно, выбрасывает исключение.
* TryParse: Пытается преобразовать строку в указанный тип данных. Возвращает bool, указывающий на успешность преобразования, и результат преобразования через выходной параметр. Если преобразование невозможно, не выбрасывает исключение.

**6. Что умеет делать класс Convert?**

Класс Convert предоставляет методы для преобразования одного базового типа данных в другой. Он поддерживает преобразование между всеми основными типами данных, такими как int, double, bool, string, и т.д

**7. Как найти машинное эпсилон для типа Decimal? Чему оно равно?**

Машинное эпсилон для типа Decimal можно найти, определив наименьшее значение, которое можно прибавить к 1, чтобы получить значение, отличное от 1. В C# это можно сделать следующим образом:

decimal epsilon = 1.0m;

while (1.0m + epsilon / 2.0m != 1.0m)

{

epsilon /= 2.0m;

}

Console.WriteLine(epsilon);

Машинное эпсилон для типа Decimal равно 0.0000000000000000000000000001m (1e-28).

**8. Кто первым обработает событие нажатия клавиши? Форма, или элемент формы, находящийся в фокусе?**

Элемент формы, находящийся в фокусе, первым обработает событие нажатия клавиши. Если элемент не обрабатывает событие, оно может быть передано форме или другим элементам управления.

Изображение выглядит как текст, Шрифт, белый, чек

Автоматически созданное описание**9. Запишите в виде формулы алгоритм Ньютона вычисления квадратного корня?**

**10. Как контролируется погрешность в методе Ньютона?**

Погрешность в методе Ньютона контролируется путем проверки разницы между текущим и предыдущим приближениями. Если разница меньше заданного порога (например, ϵ*ϵ*), итерации прекращаются. Формула для проверки:

Изображение выглядит как Шрифт, текст, белый, типография

Автоматически созданное описание

**11. В чем отличие классов String и StringBuilder?**

* String: Неизменяемый тип данных. Любая операция, изменяющая строку, создает новый объект строки, что может быть неэффективно при множестве изменений.
* StringBuilder: Изменяемый тип данных. Позволяет эффективно изменять строку без создания новых объектов, что полезно при множестве операций конкатенации или изменений.

**13. Как реализуется запись значений матрицы в Grid?**

**Представлены 4 проекта, реализованных в Visual Studio Community 2022.  
Проекты представлены преподавателю в электронной форме, продемонстрирована их работоспособность, разъяснены детали программного кода.**