

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** - общезыковые 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.

```
Ссылка 2
class TypeConverter
{
    Ссылка 1
    public void PerformImplicitConversion()
    {
        byte bbyte = 20;
        sbyte ssbyte = bbyte;
        short sshort = bbyte;
        ushort ushort = bbyte;
        int iint = bbyte;
        uint uint = bbyte;
        long llong = bbyte;
        ulong ulong = bbyte;

        float ffloat = bbyte;
        double ddouble = bbyte;
        decimal ddecimal = bbyte;

        bool bbool = bbyte;


        char cchar = bbyte;
        string sstring = bbyte;

        object oobject = bbyte;

        Console.WriteLine($"Byte -> short = {sshort} \n Byte -> ushort = {ushort} \n Byte -> int = {iint} \n Byte -> uint = {uint} \n Byte -> long = {llong}");
        Console.WriteLine($"Byte -> float = {ffloat} \n Byte -> double = {ddouble} \n Byte -> float = {ddecimal}");
        Console.WriteLine($"Byte -> object = {oobject}");
    }
}
```

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

```
bool bbool = bbyte;
...
```

 (локальная переменная) `byte bbyte`

CS0029: Не удастся неявно преобразовать тип "byte" в "bool".

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:

Неявное преобразование:

```
Byte -> short = 20
Byte -> ushort = 20
Byte -> int = 20
Byte -> uint = 20
Byte -> long = 20
Byte -> ulong = 20
```

```
Byte -> float = 20
Byte -> double = 20
Byte -> float = 20
Byte -> object = 20
```

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)

	byte	sbyte	short	ushort	int	uint	long	ulong	float	double	decimal	bool	char	string	object
byte	#####	-	+	+	+	+	+	+	+	+	+	-	-	-	+
sbyte		#####													
short			#####												
ushort				#####											
int					#####										
uint						#####									
long							#####								
ulong								#####							
float									#####						
double										#####					
decimal											#####				
bool												#####			
char													#####		
string														#####	
object															#####

Далее сделаем аналогичный эксперимент с sbyte:

Ссылка: 1

```

public void PerformImplicitConversion()
{
    sbyte ssbyte = 20;
    byte bbyte = ssbyte;
    short sshort = ssbyte;
    ushort ushort = ssbyte;
    int iint = ssbyte;
    uint uint = ssbyte;
    long llong = ssbyte;
    ulong ulong = ssbyte;

    float ffloat = ssbyte;
    double ddouble = ssbyte;
    decimal ddecimal = ssbyte;

    bool bbool = ssbyte;

    char cchar = ssbyte;
    string sstring = ssbyte;

    object oobject = ssbyte;

```

И так далее для всех типов данных. Промежуточные шаги:

```
public void PerformImplicitConversion()
{
    int iint = 20;

    byte bbyte = iint;
    sbyte ssbyte = iint;
    short sshort = iint;
    ushort ushort = iint;

    uint uint = iint;
    long llong = iint;
    ulong ulong = iint;

    float ffloat = iint;
    double ddouble = iint;
    decimal ddecimal = iint;

    bool bbool = iint;

    char cchar = iint;
    string sstring = iint;

    object oobject = iint;
}
```

	byte	sbyte	short	ushort	int	uint	long	ulong	float	double	decimal	bool	char	string	object
byte	#####	-	+	+	+	+	+	+	+	+	+	-	-	-	+
sbyte	-	#####	+	-	+	-	+	-	+	+	+	-	-	-	+
short	-	-	#####	-	+	-	+	-	+	+	+	-	-	-	+
ushort	-	-	-	#####	+	+	+	+	+	+	+	-	-	-	+
int	-	-	-	-	#####	-	+	-	+	+	+	-	-	-	+
uint						#####									
long							#####								
ulong								#####							
float									#####						

```
double ddouble = ffloat;
decimal ddecimal = ffloat;

bool bbool = ffloat;

char cchar = ffloat;
string sstring = ffloat;
```

В итоге получим заполненную таблицу:

	byte	sbyte	short	ushort	int	uint	long	ulong	float	double	decimal	bool	char	string	object
byte	#####	-	+	+	+	+	+	+	+	+	+	-	-	-	+
sbyte	-	#####	+	-	+	-	+	-	+	+	+	-	-	-	+
short	-	-	#####	-	+	-	+	-	+	+	+	-	-	-	+
ushort	-	-	-	#####	+	+	+	+	+	+	+	-	-	-	+
int	-	-	-	-	#####	-	+	-	+	+	+	-	-	-	+
uint	-	-	-	-	-	#####	+	+	+	+	+	-	-	-	+
long	-	-	-	-	-	-	#####	-	+	+	+	-	-	-	+
ulong	-	-	-	-	-	-	-	#####	+	+	+	-	-	-	+
float	-	-	-	-	-	-	-	-	#####	+	-	-	-	-	+
double	-	-	-	-	-	-	-	-	-	#####	-	-	-	-	+
decimal	-	-	-	-	-	-	-	-	-	-	#####	-	-	-	+
bool	-	-	-	-	-	-	-	-	-	-	-	#####	-	-	+
char	-	-	-	+	+	+	+	+	+	+	+	-	#####	-	+
string	-	-	-	-	-	-	-	-	-	-	-	-	-	#####	+
object	-	-	-	-	-	-	-	-	-	-	-	-	-	-	#####

## 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.

```
public void PerformExplicitConversion()
{
    byte bbyte = 20;
    sbyte ssbyte = (sbyte)bbyte;
    short sshort = (short)bbyte;
    ushort ushort = (ushort)bbyte;
    int iint = (int)bbyte;
    uint uint = (uint)bbyte;
    long llong = (long)bbyte;
    ulong ulong = (ulong)bbyte;

    float ffloat = (float)bbyte;
    double ddouble = (double)bbyte;
    decimal ddecimal = (decimal)bbyte;


    bool bbool = (bool)bbyte;

    char cchar = (char)bbyte;
    string sstring = (string)bbyte;

    object oobject = (object)bbyte;
}
```

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:

```
bool bbool = (bool)bbyte;
```

 (локальная переменная) `byte bbyte`

CS0030: Не удастся преобразовать тип "byte" в "bool"

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

	byte	sbyte	short	ushort	int	uint	long	ulong	float	double	decimal	bool	char	string	object
byte	#####	+	+	+	+	+	+	+	+	+	+	-	+	-	+
sbyte	+	#####	+	+	+	+	+	+	+	+	+	-	+	-	+
short	+	+	#####	+	+	+	+	+	+	+	+	-	+	-	+
ushort	+	+	+	#####	+	+	+	+	+	+	+	-	+	-	+
int	+	+	+	+	#####	+	+	+	+	+	+	-	+	-	+
uint	+	+	+	+	+	#####	+	+	+	+	+	-	+	-	+
long	+	+	+	+	+	+	#####	+	+	+	+	-	+	-	+
ulong	+	+	+	+	+	+	+	#####	+	+	+	-	+	-	+
float	+	+	+	+	+	+	+	+	#####	+	+	-	+	-	+
double	+	+	+	+	+	+	+	+	+	#####	+	-	+	-	+
decimal	+	+	+	+	+	+	+	+	+	+	#####	-	+	-	+
bool	-	-	-	-	-	-	-	-	-	-	-	#####	-	-	+
char	+	+	+	+	+	+	+	+	+	+	+	-	#####	-	+
string	-	-	-	-	-	-	-	-	-	-	-	-	-	#####	+
object	+	+	+	+	+	+	+	+	+	+	+	+	+	+	#####

Стандартной ситуацией в явном преобразовании оказалось то, что все типы можно преобразовать в любые другие, НО исключениями являются тип `Bool` и `String`, которых удастся преобразовать только в `Object` и в которых не удастся 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:

```
Ссылка: 5
class Animal { }
Ссылка: 4
class Dog : Animal { }
Ссылка: 3
class Cat : Animal { }
```

Далее воспользуемся приведением is

```
Ссылка: 0
class Program
{
    Ссылка: 0
    static void Main(string[] args)
    {
        Animal animal = new Dog();

        if (animal is Dog)
        {
            Dog dog = (Dog)animal;
            // Выполнить действия с dog
            Console.WriteLine("Это собака");
        }
        else if (animal is Cat)
        {
            Cat cat = (Cat)animal;
            // Выполнить действия с cat
            Console.WriteLine("Это кошка");
        }
        else
        {
            // Объект animal не является ни Dog, ни Cat
            Console.WriteLine("Это не собака и не кошка");
        }
    }
}
```

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

```
Это собака
```



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`

```
class Animal { }
Ссылка: 3
class Dog : Animal { }
Ссылка: 3
class Cat : Animal { }

Ссылка: 0
class Program
{
    Ссылка: 0
    static void Main(string[] args)
    {
        Animal animal = new Animal();

        if (animal is Dog)
        {
        }
```

В этом случае вывод будет выглядеть так:

```
Это не собака и не кошка
```

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

```

using System;

namespace ConsoleApp1
{
    Ссылка: 3
    class Animal { }
    Ссылка: 3
    class Dog : Animal { }
    Ссылка: 0
    class Cat : Animal { }

    Ссылка: 0
    class Program
    {
        Ссылка: 0
        static void Main(string[] args)
        {
            Animal animal = new Dog();

            Dog dog = animal as Dog;
            if (dog != null)
            {
                // Преобразование удалось, можно работать с dog
                Console.WriteLine("Это собака");
            }
            else
            {
                // Преобразование не удалось
                Console.WriteLine("Это не собака");
            }
        }
    }
}

```

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

```
namespace ConsoleApp1
```

```
{
```

```
    Ссылка: 4
```

```
    class Animal { }
```

```
    Ссылка: 0
```

```
    class Dog : Animal { }
```

```
    Ссылка: 2
```

```
    class Cat : Animal { }
```

```
    Ссылка: 0
```

```
    class Program
```

```
    {
```

```
        Ссылка: 0
```

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

```
        {
```

```
            Animal animal = new Animal();
```

```
            Cat cat = animal as Cat;
```

```
            if (cat != null)
```

```
            {
```

```
                // Преобразование удалось, можно работать с c
```

```
                Console.WriteLine("Это кошка");
```

```
            }
```

```
            else
```

```
            {
```

```
                // Преобразование не удалось
```

```
                Console.WriteLine("Это не кошка");
```

```
            }
```

```
        }
```

```
    }
```

Консоль отладки Microsoft

Это не кошка

C:\Users\dmityr\Desktop  
боту с кодом 0 (0x0).  
Чтобы автоматически з  
томатически закрыть к  
Нажмите любую клавишу

#### 4. Пользовательские типы данных

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

```
public class Person
{
    Ссылка: 1
    public string Name { get; set; }
    Ссылка: 1
    public int Age { get; set; }

    Ссылка: 1
    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
}

Ссылка: 3
public class Car
{
    Ссылка: 1
    public string Model { get; set; }
    Ссылка: 1
    public int Year { get; set; }

    Ссылка: 0
    public Car(string model, int year)
    {
        Model = model;
        Year = year;
    }
}
```

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

```
Ссылка: 0
public void PerformUserType() {

    Person person = new Person("John Doe", 35);
    // Преобразовываем объект Person в объект Car
    Car car = (Car)person;

}
```

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:

```

Ссылка: 4
public class Person
{
    Ссылка: 2
    public string Name { get; set; }
    Ссылка: 2
    public int Age { get; set; }

    Ссылка: 1
    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }

    public static explicit operator Car(Person person)
    {
        return new Car($"{person.Name}'s car", DateTime.Now.Year - person.Age);
    }
}

```

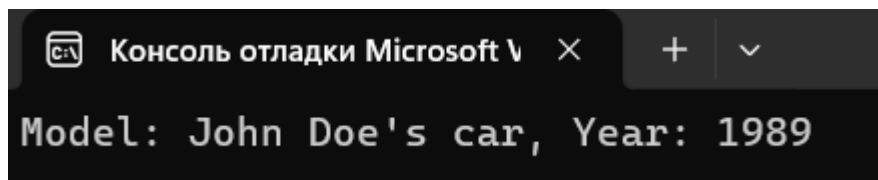
Такой статический оператор явного преобразования (*explicit operator* `Car(Person person)`) определяет, как объект `Person` может быть преобразован в объект `Car`. Теперь среда программирования не будет жаловаться на наше преобразование. В качестве значения, которое мы возвращаем это переменная типа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

```

Ссылка: 0
public void PerformUserType() {
    Person person = new Person("John Doe", 35);
    // Преобразовываем объект Person в объект Car
    Car car = (Car)person;
    Console.WriteLine($"Model: {car.Model}, Year: {car.Year}");
}
Ссылка: 4

```

В качестве вывода мы получим:



Консоль отладки Microsoft V × + ▾

Model: John Doe's car, Year: 1989

## 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.

```
class Program
{
    Ссылка: 0
    static void Main(string[] args)
    {
        string input1 = "42";
        string input2 = "hello";

        // Преобразование с использованием TryParse
        if (int.TryParse(input1, out int result1))
        {
            Console.WriteLine($"Успешное преобразование: {result1}");
        }
        else
        {
            Console.WriteLine("Ошибка преобразования");
        }

        if (int.TryParse(input2, out int result2))
        {
            Console.WriteLine($"Успешное преобразование: {result2}");
        }
        else
        {
            Console.WriteLine("Ошибка преобразования");
        }
    }
}
```

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:

```
Успешное преобразование: 42
Ошибка преобразования
```

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 собственным им типам:

```

namespace ConsoleApp1
{
    Ссылка: 0
    class Program
    {
        Ссылка: 0
        static void Main(string[] args)
        {
            // Преобразование с использованием Convert
            string input1 = "42";
            string input2 = "3.14";
            string input3 = "true";

            int result1 = Convert.ToInt32(input1);
            double result2 = Convert.ToDouble(input2);
            bool result3 = Convert.ToBoolean(input3);

            Console.WriteLine($"Преобразование int: {result1}");
            Console.WriteLine($"Преобразование double: {result2}");
            Console.WriteLine($"Преобразование bool: {result3}");
        }
    }
}

```

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:

```

string input = "hello";

// Преобразование с использованием Convert
int result = Convert.ToInt32(input);

Console.WriteLine($"Преобразование int: {result}");

```

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

```

Исключение не обработано
System.FormatException: "The input string 'hello' was not in a correct format."

Спросить Copilot | Показать стек вызовов | Просмотреть сведения | Копировать подробности | Запуск сеанса Live Share
Параметры исключений

```

Поэтому можем обработать ошибку вручную:

```
string input = "hello";

// Преобразование с использованием Convert
try
{
    // Преобразование с использованием Convert
    int result = Convert.ToInt32(input);
    Console.WriteLine($"Преобразование int: {result}");
}
catch (FormatException)
{
    Console.WriteLine("Ошибка преобразования: неверный формат");
}
```

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

```
Ссылка: 0
static void Main(string[] args)
{
    string input1 = "42";
    string input2 = "hello";

    // Преобразование с использованием Parse
    try
    {
        int result1 = int.Parse(input1);
        Console.WriteLine($"Успешное преобразование: {result1}");
    }
    catch (FormatException)
    {
        Console.WriteLine("Ошибка преобразования");
    }

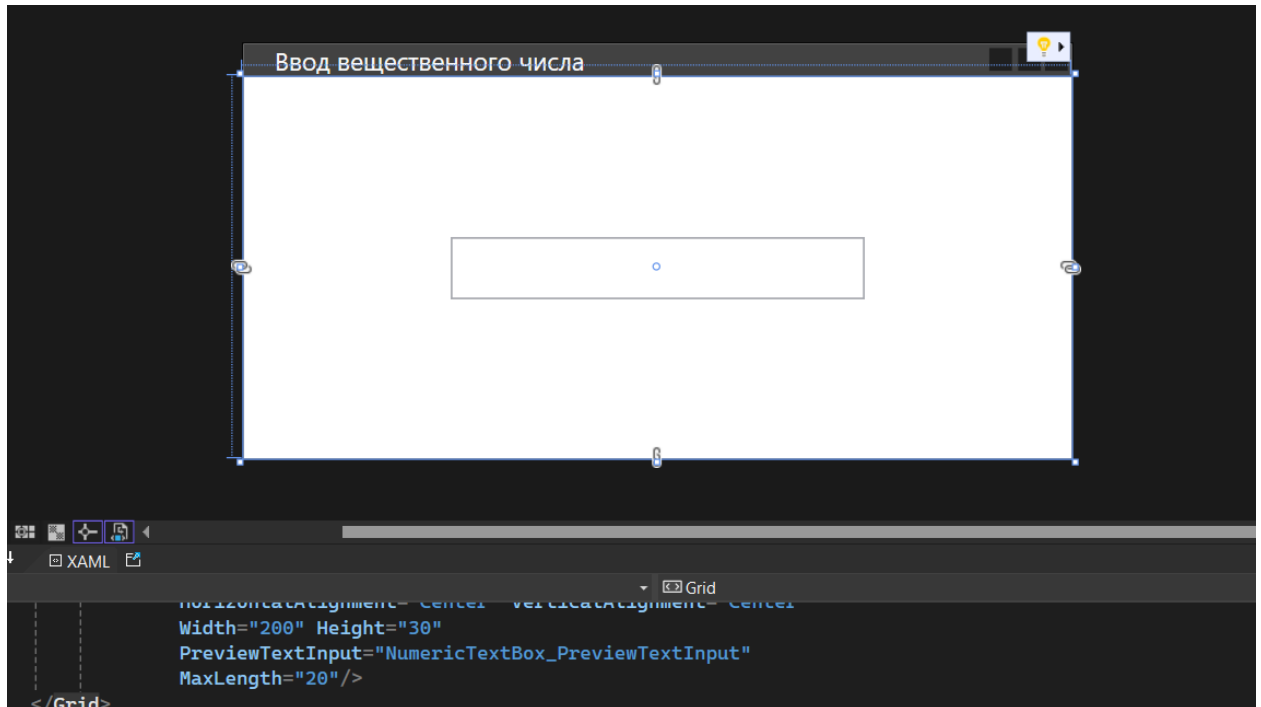
    try
    {
        int result2 = int.Parse(input2);
        Console.WriteLine($"Успешное преобразование: {result2}");
    }
    catch (FormatException)
    {
        Console.WriteLine("Ошибка преобразования");
    }
}
```

Здесь нам также необходимо обрабатывать исключения. В отличие 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)

```
Ссылка: 1
private void NumericTextBox_PreviewTextInput(object sender, TextCompositionEventArgs e)
{
    // Проверяем вводимый символ
    bool isValid = IsValidInput(NumericTextBox.Text, e.Text);

    // Если ввод некорректный, предотвращаем его добавление в TextBox
    e.Handled = !isValid;

    // Выводим сообщение о корректности ввода
    if (isValid)
    {
        ResultLabel.Content = "Корректное вещественное число";
    }
    else
    {
        ResultLabel.Content = "Некорректный ввод!";
    }
}

// Метод для проверки, является ли вводимый символ допустимым
Ссылка: 1
private bool IsValidInput(string currentText, string newInput)
{
    // Объединяем текущий текст с новым вводимым символом
    string fullText = currentText + newInput;

    // Регулярное выражение для вещественного числа со знаком
    string pattern = @"^[-]?(\d+)?([.]\d*)?$";

    return Regex.IsMatch(fullText, pattern);
}
```

class System.String  
Represents text as a sequence of UTF-16 code units.

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:

Корректное вещественное число

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

Некорректный ввод!

## 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.

```
Ссылка 1
private void ConvertButton_Click(object sender, RoutedEventArgs e)
{
    if (TypeComboBox.SelectedItem == null || ConvertToComboBox.SelectedItem == null)
    {
        ResultTextBlock.Text = "Пожалуйста, выберите тип и преобразование.";
        return;
    }

    string input = InputTextBox.Text;
    string type = ((ComboBoxItem)TypeComboBox.SelectedItem).Content.ToString();
    string convertTo = ((ComboBoxItem)ConvertToComboBox.SelectedItem).Content.ToString();

    Converter converter = new Converter();
    dynamic convertedValue = converter.Convert_input_Explicited(input, GetChoice(type));

    if (convertedValue is string) // Проверка на ошибку
    {
        ResultTextBlock.Text = $"Ошибка явного преобразования: {convertedValue}";
        return;
    }

    dynamic convertedValueTo = converter.Convert_input_Explicited(convertedValue, GetChoice(convertTo));
    string explicitResult = convertedValueTo is string ? convertedValueTo : convertedValueTo.ToString();

    dynamic implicitValue = converter.Convert_input_Implicit(convertedValue, GetChoice(convertTo));
    string implicitResult = implicitValue is string ? implicitValue : "Успешно";

    ResultTextBlock.Text = $"Явное преобразование: {explicitResult}\nНеявное преобразование: {implicitResult}";
}
```

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

```
Ссылка 2
private int GetChoice(string type)
{
    switch (type)
    {
        case "bool": return 1;
        case "sbyte": return 2;
        case "int": return 3;
        case "uint": return 4;
        case "long": return 5;
        case "ulong": return 6;
        case "float": return 7;
        case "double": return 8;
        case "decimal": return 9;
        case "char": return 10;
        case "string": return 11;
        case "object": return 12;
        case "byte": return 13;
        default: return -1;
    }
}
```

Перейдем к методу преобразования из пользовательского интерфейса. Именно он используется, чтобы перевести в выбранный 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.

```
Ссылка 1
public dynamic Convert_input_Explicited(dynamic input, int choice)
{
    if (input == null)
        return "Входное значение не может быть null.";

    switch (choice)
    {
        case 1: // bool
            bool boolResult;
            return bool.TryParse(input.ToString(), out boolResult) ? boolResult : "Не удалось преобразовать в bool.";
        case 2: // sbyte
            sbyte sbyteResult;
            return sbyte.TryParse(input.ToString(), out sbyteResult) ? sbyteResult : "Не удалось преобразовать в sbyte.";
        case 3: // int
            int intResult;
            return int.TryParse(input.ToString(), out intResult) ? intResult : "Не удалось преобразовать в int.";
        case 4: // uint
            uint uintResult;
            return uint.TryParse(input.ToString(), out uintResult) ? uintResult : "Не удалось преобразовать в uint.";
        case 5: // long
            long longResult;
            return long.TryParse(input.ToString(), out longResult) ? longResult : "Не удалось преобразовать в long.";
        case 6: // ulong
            ulong ulongResult;
            return ulong.TryParse(input.ToString(), out ulongResult) ? ulongResult : "Не удалось преобразовать в ulong.";
        case 7: // float
            float floatResult;
            return float.TryParse(input.ToString(), out floatResult) ? floatResult : "Не удалось преобразовать в float.";
        case 8: // double
            double doubleResult;
            return double.TryParse(input.ToString(), out doubleResult) ? doubleResult : "Не удалось преобразовать в double.";
        case 9: // decimal
            decimal decimalResult;
            return decimal.TryParse(input.ToString(), out decimalResult) ? decimalResult : "Не удалось преобразовать в decimal.";
        case 10: // char
            return (input.ToString().Length == 1) ? input.ToString()[0] : "Не удалось преобразовать в char.";
        case 11: // string
            return (string)input;
        case 12: // object
            return (object)input; // всегда успешно
        case 13: // byte
            byte byteResult;
            return byte.TryParse(input.ToString(), out byteResult) ? byteResult : "Не удалось преобразовать в byte.";
        default:
            return "Неверный выбор.";
    }
}
```

Подобным образом напишем метод неявного преобразования

```
public dynamic Convert_input_Implicit(dynamic input, int choice)
{
    switch (choice)
    {
        case 1: // bool
            return input is bool ? "Успешно" : "Не удалось преобразовать в bool.";
        case 2: // sbyte
            return input is sbyte ? "Успешно" : "Не удалось преобразовать в sbyte.";
        case 3: // int
            return input is int ? "Успешно" : "Не удалось преобразовать в int.";
        case 4: // uint
            return input is uint ? "Успешно" : "Не удалось преобразовать в uint.";
        case 5: // long
            return input is long ? "Успешно" : "Не удалось преобразовать в long.";
        case 6: // ulong
            return input is ulong ? "Успешно" : "Не удалось преобразовать в ulong.";
        case 7: // float
            return input is float ? "Успешно" : "Не удалось преобразовать в float.";
        case 8: // double
            return input is double ? "Успешно" : "Не удалось преобразовать в double.";
        case 9: // decimal
            return input is decimal ? "Успешно" : "Не удалось преобразовать в decimal.";
        case 10: // char
            return input is char ? "Успешно" : "Не удалось преобразовать в char."; // Неявное преобразование не должно быть успешным
        case 11: // string
            return "Не удалось преобразовать в string."; // Прямое преобразование не должно быть успешным
        case 12: // object
            return "Успешно"; // всегда успешно
        case 13: // byte
            return input is byte ? "Успешно" : "Не удалось преобразовать в byte.";
        default:
            return "Неверный выбор.";
    }
}
```

И явного преобразования:

```
public dynamic Convert_explicit_out(dynamic input, int choice)
{
    try
    {
        switch (choice)
        {
            case 1: // bool
                return Convert.ToBoolean(input) ? "Успешно" : "Не удалось преобразовать в bool.";
            case 2: // byte (заменял sbyte на byte, так как sbyte не является стандартным типом в C#)
                return Convert.ToByte(input) == input ? "Успешно" : "Не удалось преобразовать в byte.";
            case 3: // int
                return Convert.ToInt32(input) == input ? "Успешно" : "Не удалось преобразовать в int.";
            case 4: // uint
                return Convert.ToInt32(input) == input ? "Успешно" : "Не удалось преобразовать в uint.";
            case 5: // long
                return Convert.ToInt64(input) == input ? "Успешно" : "Не удалось преобразовать в long.";
            case 6: // ulong
                return Convert.ToInt64(input) == input ? "Успешно" : "Не удалось преобразовать в ulong.";
            case 7: // float
                return Convert.ToSingle(input) == input ? "Успешно" : "Не удалось преобразовать в float.";
            case 8: // double
                return Convert.ToDouble(input) == input ? "Успешно" : "Не удалось преобразовать в double.";
            case 9: // decimal
                return Convert.ToDecimal(input) == input ? "Успешно" : "Не удалось преобразовать в decimal.";
            case 10: // char
                return Convert.ToChar(input) == input ? "Успешно" : "Не удалось преобразовать в char.";
            case 11: // string
                return input is string ? "Успешно" : "Не удалось преобразовать в string.";
            case 12: // object
                return "Успешно"; // всегда успешно
            case 13: // byte
                return Convert.ToByte(input) == input ? "Успешно" : "Не удалось преобразовать в byte.";
            default:
                return "Неверный выбор.";
        }
    }
    catch
    {
        return "Не удалось преобразовать.";
    }
}
```

Примеры работы:

Converter

Введите значение:

12

Тип: int

Преобразовать в: char

Преобразовать

Явное преобразование: Успешно

Неявное преобразование: Не удалось преобразовать в char.

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

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

```
Ссылка: 1
private void CalculateButton_Click(object sender, RoutedEventArgs e)
{
    decimal number, initialGuess;
    int rootDegree;

    if (!decimal.TryParse(NumberTextBox.Text, out number) ||
        !int.TryParse(RootDegreeTextBox.Text, out rootDegree))
    {
        MessageBox.Show("Вы ввели неверные значения");
        return;
    }

    // Находим первое приближение
    initialGuess = number / rootDegree;
    ResultTextBlock3.Text = $"Первичное приближение: {initialGuess}";

    decimal epsilon = 1E-10m; // Точность
    var (result, iter) = NewtonMethod(number, rootDegree, initialGuess, epsilon);

    // Стандартное вычисление корня (через double)
    double standardResult = Math.Pow((double)number, 1.0 / rootDegree);

    ResultTextBlock1.Text = $"Корень {rootDegree}-й степени из {number} (метод Ньютона) равен {result}";
    ResultTextBlock2.Text = $"Понадобилось {iter} итераций";
    ResultTextBlock4.Text = $"Корень {rootDegree}-й степени из {number} (стандартный метод) равен {standardResult}";
}
```

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.

```
static (decimal result, int iterations) NewtonMethod(decimal number, int rootDegree, decimal initialGuess, decimal epsilon)
{
    decimal guess = initialGuess;
    decimal previousGuess;
    int iter = 0;

    do
    {
        previousGuess = guess;
        // Формула метода Ньютона для нахождения корня степени rootDegree
        guess = ((rootDegree - 1) * guess + number / Pow(guess, rootDegree - 1)) / rootDegree;
        iter++;
    }
    while (Math.Abs(guess - previousGuess) > epsilon);

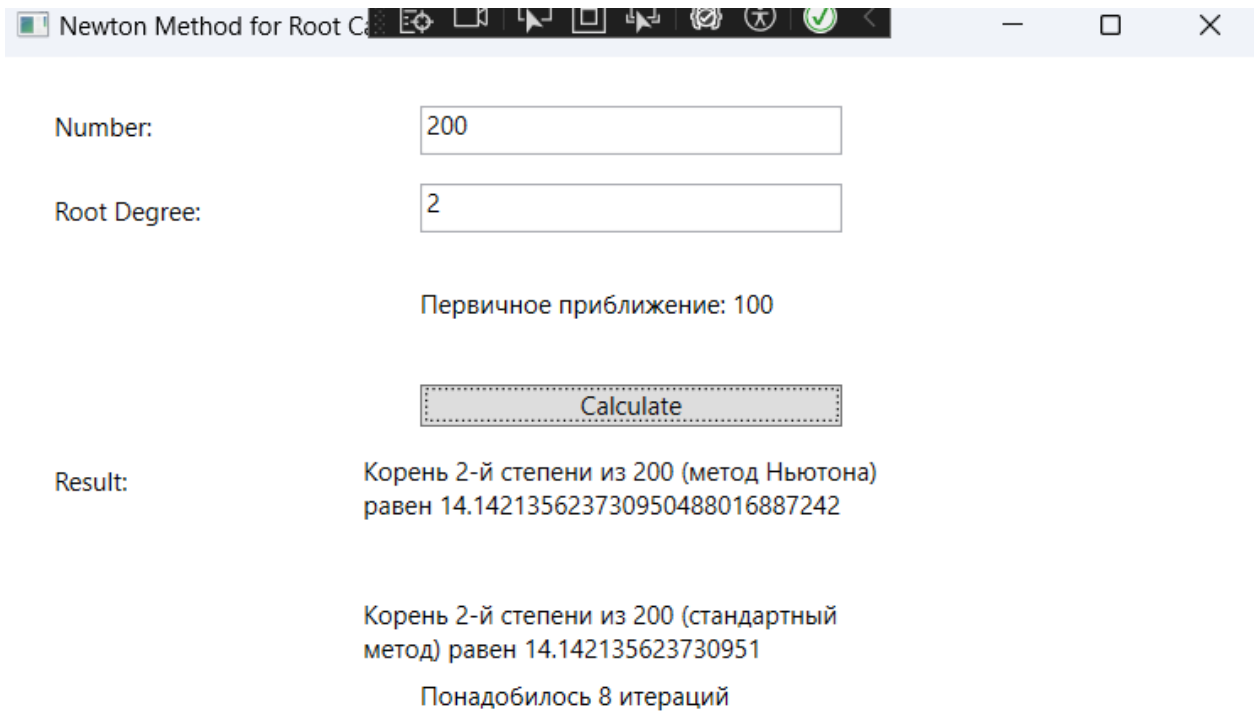
    return (guess, iter);
}
```

В переменную newGuess высчитывается новое приближение по формуле Ньютона

$$x_{k+1} = \frac{1}{n} \left( (n - 1)x_k + \frac{A}{x_k^{n-1}} \right);$$

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:



Newton Method for Root C...

Number:

Root Degree:

Первичное приближение: 100

Result: Корень 2-й степени из 200 (метод Ньютона)  
равен 14.142135623730950488016887242

Корень 2-й степени из 200 (стандартный  
метод) равен 14.142135623730951

Понадобилось 8 итераций

### 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 производится так: если  $y$  нечётно,  $D = 2n + 1$ , тогда используем приближение для корня  $2 * 10^n$ , если  $D$  чётно,  $D = 2n + 2$ , тогда используем  $6 * 10^n$ .



Ссылка 1

```
public static double GetInitialApproximation(double y)
{
    // Заменяем десятичный разделитель в строковом представлении с '.' на ','
    // Это нужно, чтобы соответствовать текущей культуре, где запятая используется как разделитель
    string yString = y.ToString(CultureInfo.InvariantCulture).Replace('.', ',');

    if (!double.TryParse(yString, NumberStyles.Any, CultureInfo.CurrentCulture, out double yAdjusted))
    {
        throw new FormatException("Не удалось распознать число с заданным десятичным разделителем.");
    }

    int D; // Переменная для хранения количества цифр перед десятичной точкой

    // Если число больше или равно 1
    if (yAdjusted >= 1)
    {
        // Определяем количество цифр перед десятичной точкой
        D = (int)Math.Floor(Math.Log10(yAdjusted)) + 1;
        int n = D / 2; // Делим количество цифр на 2 для дальнейших вычислений

        // Возвращаем первичное приближение в зависимости от четности D
        return D % 2 == 0 ? 6 * Math.Pow(10, n - 1) : 2 * Math.Pow(10, n);
    }
    else
    {
        // Если число меньше 1, определяем количество нулей после десятичной точки
        string decimals = yString.Substring(yString.IndexOf(',') + 1);
        int zeroCount = decimals.Count(c => c == '0');
        D = -zeroCount; // Устанавливаем D со знаком минус

        int n = Math.Abs(D / 2); // Берем абсолютное значение для дальнейших вычислений
        return 2 * Math.Pow(10, n); // Возвращаем первичное приближение
    }
}
```

### 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:

```
private void btnConvert_Click(object sender, RoutedEventArgs e)
{
    if (int.TryParse(txtDecimal.Text, out int decimalNumber) && int.TryParse(txtBase.Text, out int baseNumber))
    {
        if (baseNumber >= 2 && baseNumber <= 16)
        {
            if (decimalNumber < 0)
            {
                MessageBox.Show("Decimal number must be positive");
            }
            else
            {
                string result = DecimalToAnyBase(decimalNumber, baseNumber);
                txtResult.Text = result;
            }
        }
        else
        {
            MessageBox.Show("Base must be between 2 and 16");
        }
    }
    else
    {
        MessageBox.Show("Please enter valid decimal number and base");
    }
}
```



Число:

Система счисления:

Convert

Результат:

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

Ссылка: 1

```
private string DecimalToAnyBase(int decimalNumber, int baseNumber)
{
    if (decimalNumber == 0)
        return "0";

    StringBuilder result = new StringBuilder();
    while (decimalNumber > 0)
    {
        int remainder = decimalNumber % baseNumber;
        char digit = GetDigit(remainder);
        result.Insert(0, digit);
        decimalNumber /= baseNumber;
    }

    return result.ToString();
}
```

Ссылка: 1

```
private char GetDigit(int remainder)
{
    if (remainder < 10)
        return (char)('0' + remainder);
    else
        return (char)('A' + remainder - 10);
}
```

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.

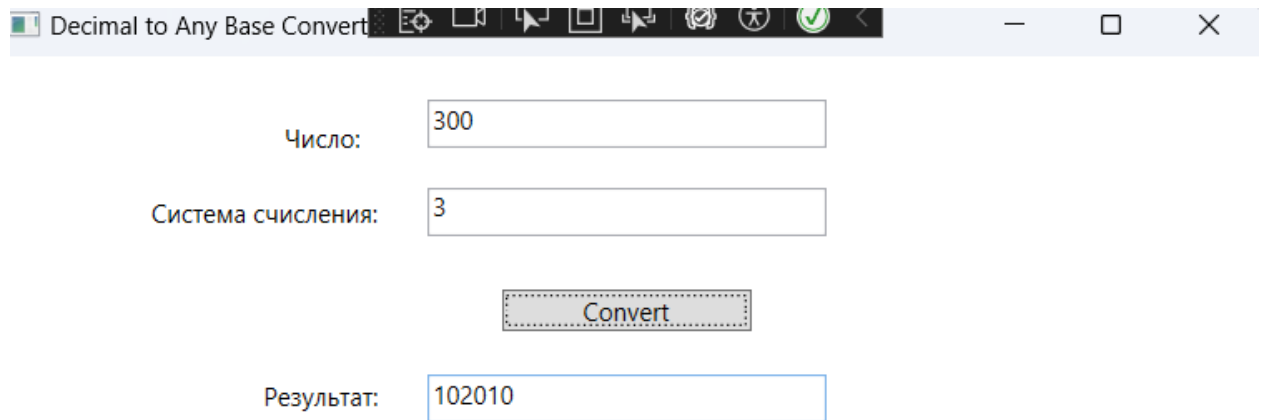
## 2. Диапазон 10-15:

- 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:



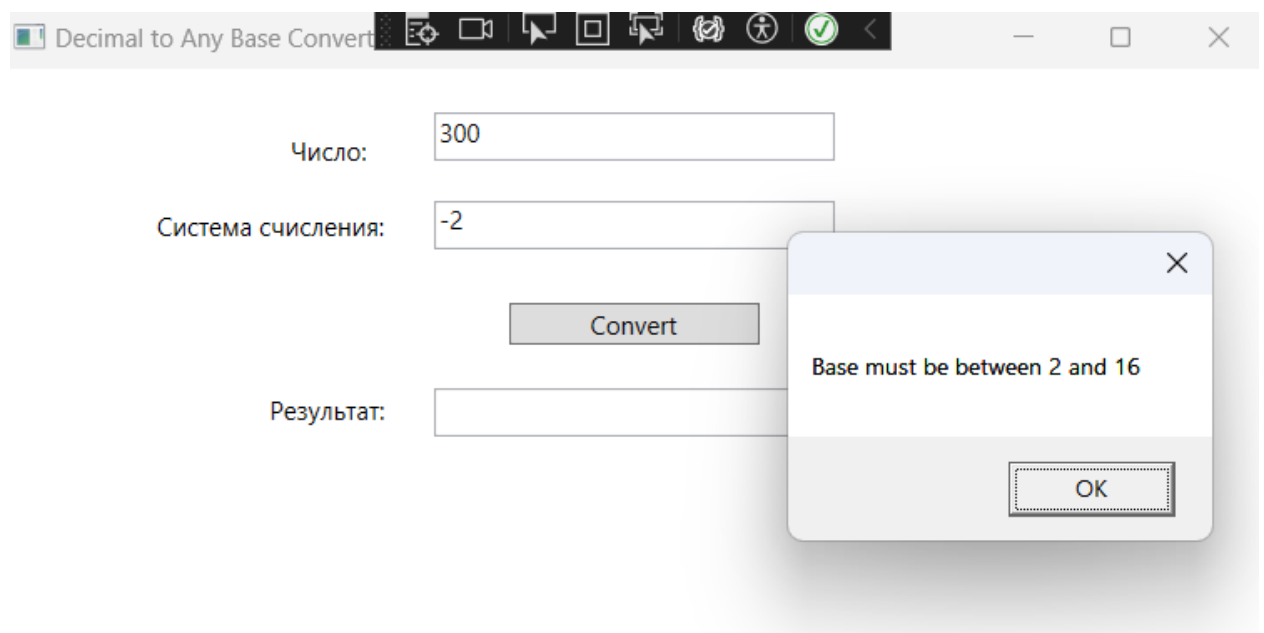
Decimal to Any Base Convert

Число: 300

Система счисления: 3

Convert

Результат: 102010



Decimal to Any Base Convert

Число: 300

Система счисления: -2

Convert

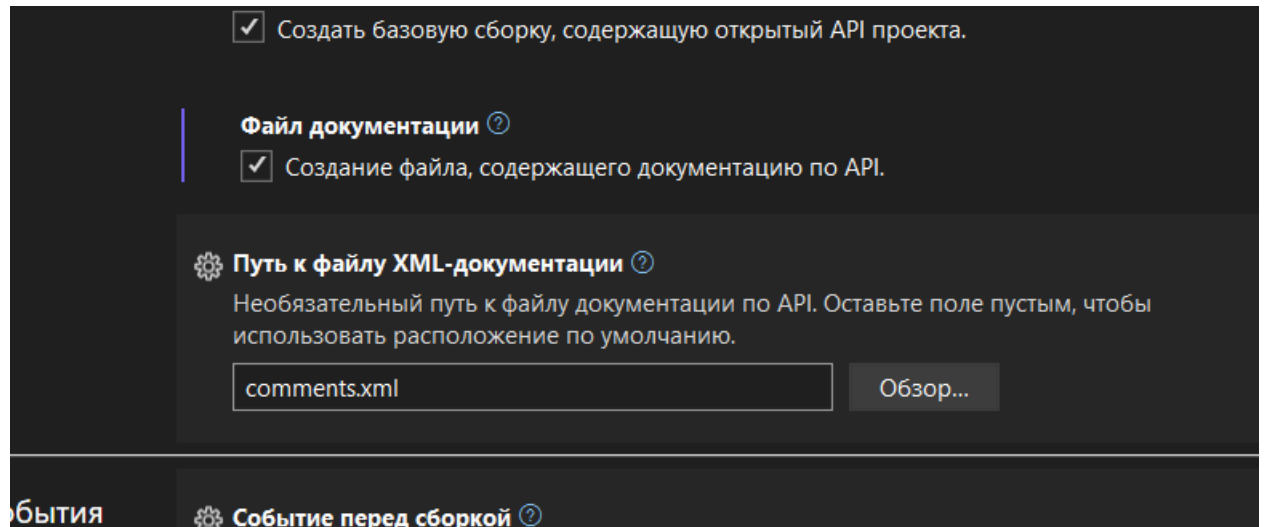
Результат:

Base must be between 2 and 16

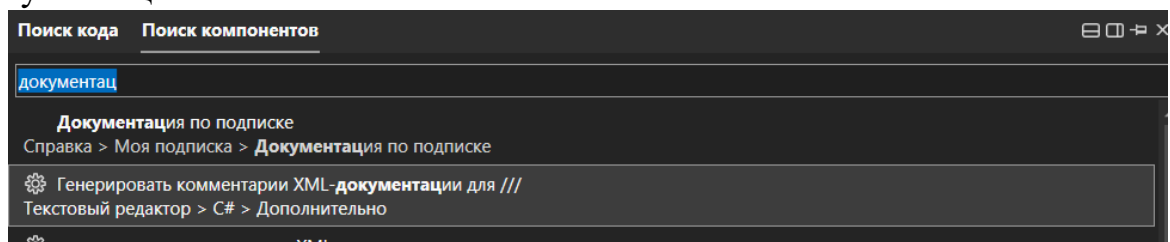
OK

#### 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

```
<?xml version="1.0"?>
<doc>
  <assembly>
    <name>Hard_WPF</name>
  </assembly>
  <members>
    <member name="T:Hard_WPF.App">
      <summary>
        Interaction logic for App.xaml
      </summary>
      <summary>
        App
      </summary>
    </member>
    <member name="M:Hard_WPF.App.InitializeComponent">
      <summary>
        InitializeComponent
      </summary>
    </member>
    <member name="M:Hard_WPF.App.Main">
      <summary>
        Application Entry Point.
      </summary>
    </member>
  </members>
</doc>
```

## Ответы на вопросы:

### 1. Какие типы преобразований 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 эpsilon для типа Decimal? Чему оно равно?

Машинное  $\epsilon$  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 0.00000000000000000000000000001m (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.

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

$$x_{n+1} = \frac{1}{2} \left( x_n + \frac{a}{x_n} \right)$$

где  $x_n$  — текущее приближение, а  $x_{n+1}$  — следующее приближение.

## 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,  $\epsilon$ ), итерации прекращаются. Формула для проверки:

$$|x_{n+1} - x_n| < \epsilon$$

## 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.**