Министерство науки и высшего образования РФ
ФГАОУ ВПО
Национальный исследовательский технологический университет «МИСиС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

Контрольная работа №1
по дисциплине «Программирование и Алгоритмизация»

Выполнил:
студент группы БИВТ-24-5

Черных Богдан

Проверил:
Стучилин В. В.

Москва, 2024

Вариант 5

Так как по условию «ОБЯЗАТЕЛЬНО каждая непустая строка примера должна содержать комментарий», значит, к каждой строке в коде будет прилагаться комментарий, он может быть расположен выше самой строки, но тем не менее он будет.

Блок №1. Переменные, типы данных, константы. Арифметические и логические операции.

А) **Var** — это ключевое слово в C#, которое позволяет не указывать явно тип переменной, а позволяет компилятору автоматически вывести тип на основе присвоенного значения. Использование var делает код более читаемым, особенно когда тип переменной очевиден из контекста или если тип сложный.

Пример:

```
var number = 21;  // Тип переменной выводится как int
var message = "Hello, Programming";  // Тип выводится как string
var list = new List<int>();  // Тип выводится как List<int>
```

В этом примере компилятор автоматически определяет типы переменных на основе присвоенных значений.

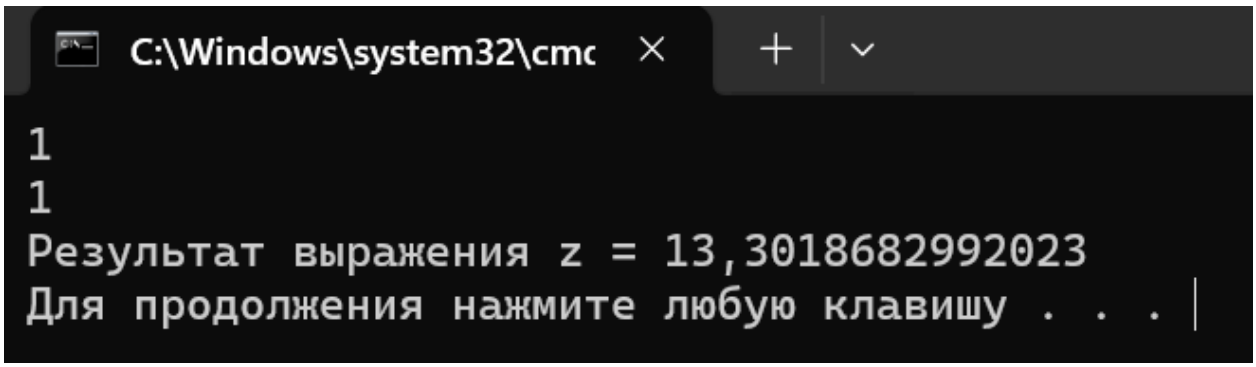
Константы в C# определяются с помощью ключевого слова const.
Константы — это неизменяемые значения, которые задаются при объявлении и не могут быть изменены во время выполнения программы.
Пример:

```
const double Pi = 3.14159; // Присваиваем значение числу Пи
const long MaxValue = 100; // Присваиваем значение переменной MaxValue
```

Константы обязательно инициализируются при объявлении, и их значение не может быть изменено после компиляции.

Б)

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        double x = Convert.ToDouble(Console.ReadLine()); // Читаем введенное
значение x и конвертируем его в формат double
        double y = Convert.ToDouble(Console.ReadLine()); // Читаем введенное
значение y и конвертируем его в формат double
        double z = Math.Pow(5, Math.Pow(x, y)) + Math.Pow(Math.Pow(3, x), 2) - (y
* (Math.Asin(y) - Math.PI / 6)) / (Math.Abs(x) + (1 / (Math.Pow(x, 2) + 1))); //
Вычисление
        Console.WriteLine($"Результат выражения z = {z}");// Выводим результат
вычислений переменной z в консоль
    }
}
```



B) Рассмотрим выражение по шагам:

1) (false && true) — это всегда false, так как оба операнда должны быть истинными, чтобы результат был истинным.
2) (true || false) — это true, так как хотя бы один операнд истинный.
3) Теперь выражение принимает вид: !(false || true).
4) (false || true) — это true.
5) Применяем отрицание !true, что дает false.

Пояснение: в выражении есть логическое "И" (&&), которое возвращает false, и логическое "ИЛИ" (||), которое возвращает true. Однако итоговое выражение включает отрицание (!), которое инвертирует результат с true на false.

Ответ: результат выражения false.

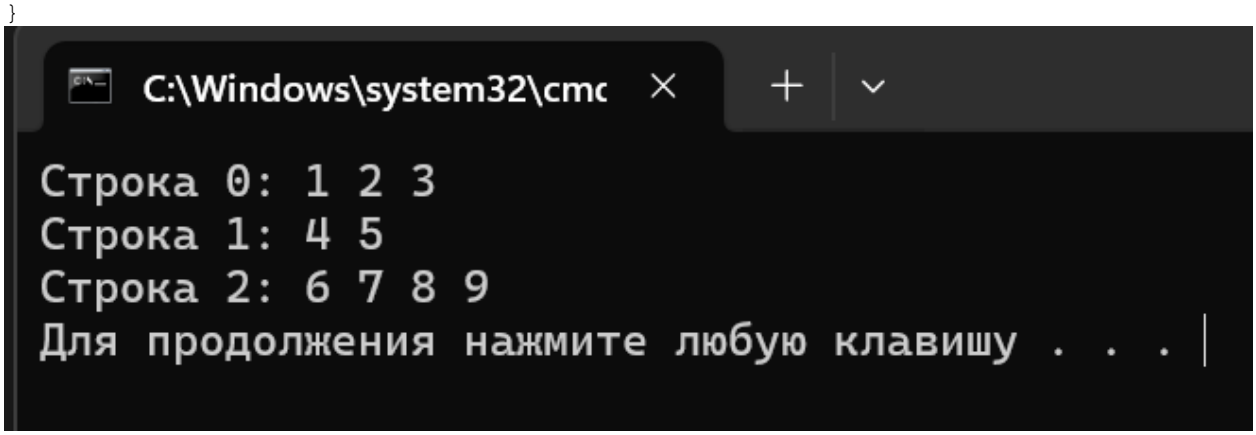
Блок №2. Условные операторы. Массивы.

A) **Ступенчатые массивы** (jagged arrays) в C# — это массивы массивов, где каждый элемент основного массива является отдельным массивом. Особенность ступенчатых массивов заключается в том, что вложенные массивы могут иметь разную длину, что позволяет гибко управлять количеством элементов в каждой "строке" массива.

Пример:

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        // Создаем ступенчатый массив, где каждая строка может иметь разное
количество элементов
        int[][] jaggedArray = new int[3][];

        // Инициализируем первую строку массива с тремя элементами
        jaggedArray[0] = new int[] { 1, 2, 3 };

        // Инициализируем вторую строку массива с двумя элементами
```

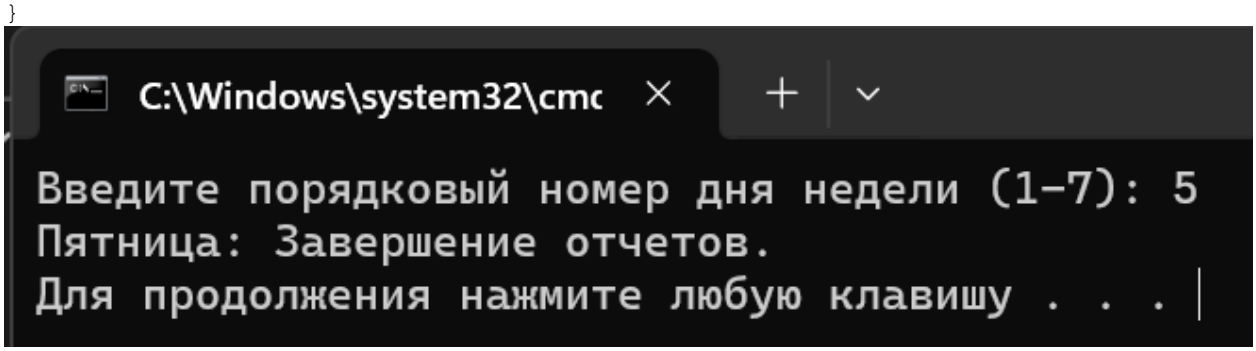
```
            jaggedArray[1] = new int[] { 4, 5 };

            // Инициализируем третью строку массива с четырьмя элементами
            jaggedArray[2] = new int[] { 6, 7, 8, 9 };

            // Проходим по всем строкам массива
            for (int i = 0; i < jaggedArray.Length; i++)
            {
                // Для каждой строки выводим её элементы
                Console.Write($"Строка {i}: ");
                for (int j = 0; j < jaggedArray[i].Length; j++)
                {
                    // Выводим каждый элемент строки
                    Console.Write(jaggedArray[i][j] + " ");
                }
                // Переход на новую строку после завершения вывода элементов строки
                Console.WriteLine();
            }
        }
}
```



Б)
Программа для работы с расписанием на неделю (используя оператор switch):

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        // Запрашиваем у пользователя порядковый номер дня недели
        Console.Write("Введите порядковый номер дня недели (1-7): ");

        // Читаем введенное значение и конвертируем в тип int
        int day = Convert.ToInt32(Console.ReadLine());

        // Используем оператор switch для отображения расписания на конкретный
день
        switch (day)
        {
            case 1:
                // Если день = 1, выводим расписание на понедельник
                Console.WriteLine("Понедельник: Встреча с командой в 10:00.");
                break;
            case 2:
                // Если день = 2, выводим расписание на вторник
                Console.WriteLine("Вторник: Работа над проектом.");
                break;
            case 3:
                // Если день = 3, выводим расписание на среду
                Console.WriteLine("Среда: Совещание в 14:00.");
                break;
```

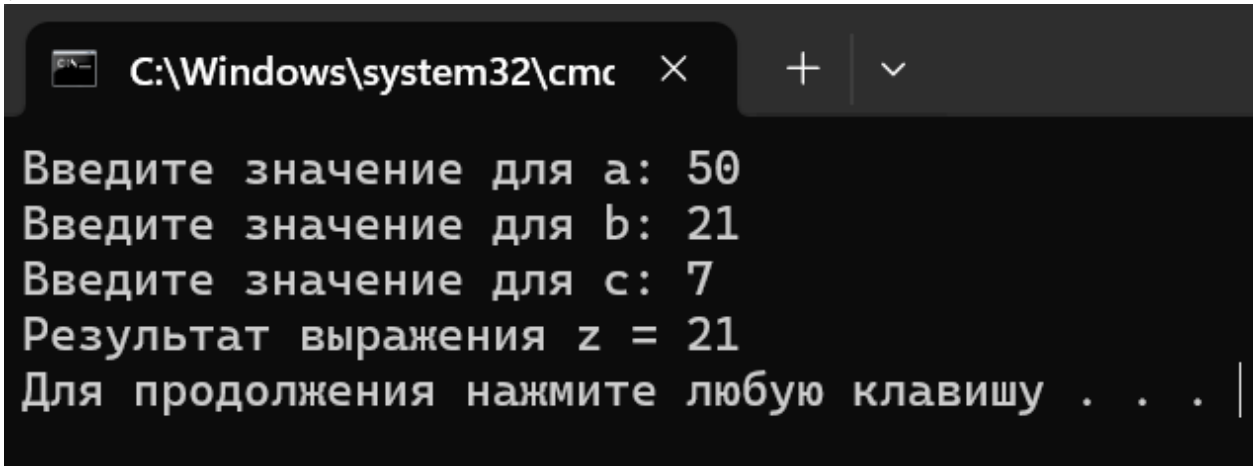
```
            case 4:
                // Если день = 4, выводим расписание на четверг
                Console.WriteLine("Четверг: Обучающий семинар.");
                break;
            case 5:
                // Если день = 5, выводим расписание на пятницу
                Console.WriteLine("Пятница: Завершение отчетов.");
                break;
            case 6:
                // Если день = 6, выводим расписание на субботу
                Console.WriteLine("Суббота: Время для отдыха.");
                break;
            case 7:
                // Если день = 7, выводим расписание на воскресенье
                Console.WriteLine("Воскресенье: Семейный день.");
                break;
            default:
                // Если введено число не в диапазоне 1-7, выводим ошибку
                Console.WriteLine("Некорректный номер дня недели.");
                break;
        }
    }
}
```



B) Программа для вычисления выражения z = max(min(a, b), c):

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        // Запрашиваем у пользователя значение для переменной a
        Console.Write("Введите значение для a: ");

        // Читаем введенное значение и конвертируем его в формат double
        double a = Convert.ToDouble(Console.ReadLine());

        // Запрашиваем у пользователя значение для переменной b
        Console.Write("Введите значение для b: ");

        // Читаем введенное значение и конвертируем его в формат double
        double b = Convert.ToDouble(Console.ReadLine());

        // Запрашиваем у пользователя значение для переменной c
        Console.Write("Введите значение для c: ");

        // Читаем введенное значение и конвертируем его в формат double
        double c = Convert.ToDouble(Console.ReadLine());

        // Находим минимальное значение между a и b
        double min_ab = Math.Min(a, b);

        // Находим максимальное значение между min_ab и c
        double z = Math.Max(min_ab, c);
```

```
        // Выводим результат вычислений переменной z
        Console.WriteLine($"Результат выражения z = {z}");
    }
}
```



Блок №3. Циклы. Операторы перехода.

А) **break** — оператор, который немедленно завершает выполнение текущего цикла (например, for, while или do-while). Когда программа встречает break, она выходит из цикла, даже если условие цикла еще выполняется.

continue — оператор, который пропускает оставшуюся часть текущей итерации цикла и переходит к следующей итерации. В отличие от break, цикл продолжается, но текущая итерация прерывается.

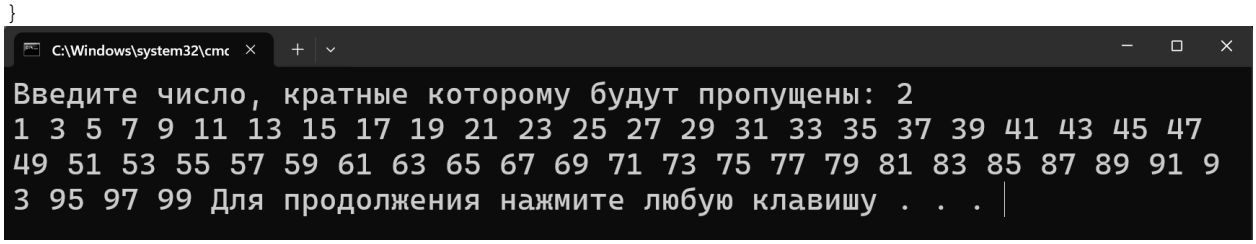
Примеры:

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        // Пример с использованием оператора break
        for (int i = 0; i < 10; i++)
        {
            // Если i равно 5, прерываем цикл
            if (i == 5)
            {
                break; // Цикл завершится при достижении значения i = 5
            }
            // Выводим значение переменной i на консоль
            Console.WriteLine($"i = {i}");
        }

        // Выводим сообщение о завершении цикла с оператором break
        Console.WriteLine("Цикл завершен при использовании break.");

        // Пример с использованием оператора continue
        for (int j = 0; j < 10; j++)
        {
            // Если j равно 5, пропускаем текущую итерацию и переходим к следующей
            if (j == 5)
            {
                continue; // Пропускаем вывод числа 5
            }
            // Выводим значение переменной j на консоль
```

```
            Console.WriteLine($"j = {j}");
        }

        // Выводим сообщение о завершении цикла с оператором continue
        Console.WriteLine("Цикл завершен при использовании continue.");
    }
}
```

Б) Программа для вывода чисел от 0 до 100, пропуская кратные введенному числу:

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        // Запрашиваем у пользователя число, кратные которому будут пропущены
        Console.Write("Введите число, кратные которому будут пропущены: ");

        // Читаем введенное значение и конвертируем его в тип int
        int skipMultiple = Convert.ToInt32(Console.ReadLine());

        // Цикл для вывода чисел от 0 до 100
        for (int i = 0; i <= 100; i++)
        {
            // Если i кратно skipMultiple, пропускаем эту итерацию
            if (i % skipMultiple == 0)
            {
                continue; // Пропускаем кратные числа
            }
            // Выводим текущее число
            Console.Write(i + " ");
        }
    }
}
```



В) Код программы по блок-схеме:

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        // Ввод значения N
        int N = Convert.ToInt32(Console.ReadLine());

        // Инициализация переменных S и I
        int S = 0; // Сумма
        int I = 0; // Счетчик

        // Цикл, который продолжается пока I < N
        while (I < N)
        {
            // Увеличиваем значение S на 1
            S = S + 1;
```

```
            // Увеличиваем значение I на 1
            I = I + 1;
        }

        // Вывод значения S
        Console.WriteLine($"Значение S = {S}");
    }
}
```

Блок №4. Обработка исключений. Работа со строками.

А) **Обработка исключений в C#** — это механизм, который позволяет программе корректно реагировать на возникновение ошибок или исключительных ситуаций (например, ошибки ввода, деление на ноль, неверные типы данных и т.д.). Он помогает предотвратить внезапное завершение программы и позволяет обработать ошибку, предложив пользователю решение или выполнив альтернативные действия.

Основные элементы обработки исключений:

1) **try** — блок кода, в котором выполняются потенциально опасные операции. Если в этом блоке произойдет ошибка (исключение), выполнение программы перейдет в соответствующий блок catch.
2) **catch** — блок кода, который перехватывает исключение и выполняет действия для его обработки. В нем можно также определить тип исключения, которое мы ожидаем (например, FormatException или DivideByZeroException).
3) **finally** — необязательный блок, который выполняется после выполнения try и catch, независимо от того, возникло исключение или нет. Он часто используется для освобождения ресурсов (например, закрытие файлов или соединений).
4) **Исключения** (Exceptions) — это объекты, которые описывают ошибку или неожиданное поведение программы. В C# исключения являются объектами классов, которые наследуются от класса System.Exception.

Примеры:

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        try // код, который может вызвать ошибку.
        {
            // Запрашиваем у пользователя ввод числа
            Console.Write("Введите число: ");
            // Пытаемся конвертировать введенные данные в тип int
            int number = Convert.ToInt32(Console.ReadLine());
            // Выводим введенное число
            Console.WriteLine($"Вы ввели число: {number}");
        }
        catch (FormatException) // код для обработки исключений.

        {
```

```
            // Обрабатываем ошибку, если пользователь ввел некорректные данные
(например, не число)
            Console.WriteLine("Ошибка: введено некорректное значение, введите
число.");
        }
        finally // выполняется всегда, независимо от успешного или неуспешного
выполнения кода в try.
        {
            // Сообщение, которое выводится в любом случае, независимо от того,
было ли исключение
            Console.WriteLine("Программа завершена.");
        }
    }
}
```

Б)
Метод **Contains** класса String используется для проверки, содержит ли строка заданную подстроку. Он возвращает значение true, если подстрока найдена в строке, и false в противном случае. Этот метод чувствителен к регистру, т.е. различает заглавные и строчные буквы.

Пример:

```
// Исходная строка для проверки
string sentence = "Привет, мир!";
// Проверяем, содержит ли строка подстроку "Привет"
bool containsWord = sentence.Contains("Привет");
// Выводим результат проверки на экран
Console.WriteLine($"Содержит ли строка 'Привет': {containsWord}");
```

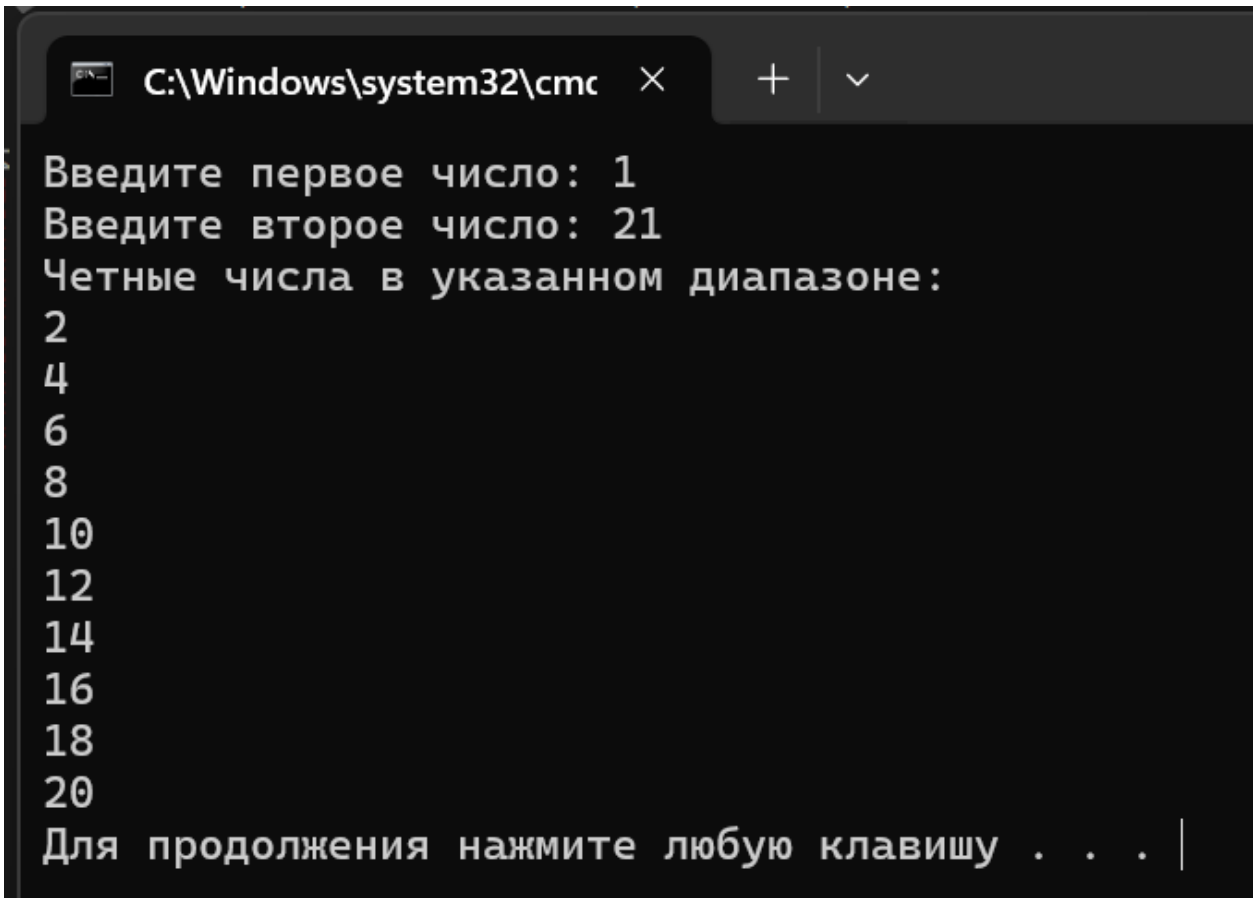
Метод **Compare** класса String используется для сравнения двух строк. Он возвращает целое число: 1) 0, если строки равны. 2) Положительное число, если первая строка больше второй. 3) Отрицательное число, если первая строка меньше второй.
Сравнение выполняется посимвольно, учитывая алфавитный порядок символов.
Пример:

```
string string1 = "яблоко"; // Строка для сравнения
string string2 = "апельсин"; // Строка для сравнения
                             // Сравниваем строки, результат сохранен в переменной
comparisonResult
int comparisonResult = String.Compare(string1, string2);
// Выводим результат сравнения на экран
Console.WriteLine($"Результат сравнения строк: {comparisonResult}");
```

В) Программа для вывода четных чисел в диапазоне с обработкой исключений:

```
using System; //Нужно для корректной работы кода
class Program // Создаем программу
{
    static void Main() // функция Main()
    {
        int start, end; // Объявляем переменные для начала и конца диапазона

        // Цикл продолжается до тех пор, пока ввод не будет корректным
        while (true)
```

```
        {
            try
            {
                // Запрашиваем у пользователя первое число и пытаемся его
конвертировать в int
                Console.Write("Введите первое число: ");
                start = Convert.ToInt32(Console.ReadLine());

                // Запрашиваем у пользователя второе число и пытаемся его
конвертировать в int
                Console.Write("Введите второе число: ");
                end = Convert.ToInt32(Console.ReadLine());

                // Проверяем, что второе число больше первого
                if (end <= start)
                {
                    // Если условие не выполняется, выводим сообщение и просим
ввести данные заново
                    Console.WriteLine("Ошибка: второе число должно быть больше
первого.");
                    continue; // Пропускаем текущую итерацию цикла, чтобы
запросить ввод снова
                }

                // Если все проверки пройдены, выходим из цикла
                break; // Завершаем цикл, так как данные введены корректно
            }
            catch (FormatException)
            {
                // Обрабатываем ошибку, если введены некорректные данные
(например, текст вместо числа)
                Console.WriteLine("Ошибка: введите корректные числовые
значения.");
            }
        }

        // Цикл для вывода четных чисел от start до end
        Console.WriteLine("Четные числа в указанном диапазоне:");
        for (int i = start; i <= end; i++) // Цикл от start до end
        {
            // Проверяем, является ли число четным
            if (i % 2 == 0)
            {
                // Выводим четное число на экран
                Console.WriteLine(i);
            }
        }
    }
}
```



Конец Работы.