

### Теоретический материал

Метод представляет собой именованный блок команд, которые можно выполнять, вызывая метод. В последующих шагах будут обсуждаться статические методы, описанные в том же классе, в котором описан главный метод программы.

**[модификаторы]      тип\_возвращаемого\_значения      название\_метода  
([параметры])**

**{**

**// тело метода**

**}**

Метод в любом случае описывается в классе. Бывают методы **статические** и **нестатические**.

Нестатический метод вызывается из объекта. То есть для вызова нестатического метода нужен объект: указывается имя объекта и через точку имя метода, который вызывается. Если метод статический, то для его вызова объект не нужен.

При вызове статического метода вместо имени объекта указывается имя класса. Но если статический метод вызывается в том же классе, в котором он описан, то имя класса можно не указывать.

**static тип имя(аргументы)**

```
{
```

```
// Код метода
```

```
}
```

Если метод в качестве тела определяет только одну инструкцию, то мы можем сократить определение метода. Например, допустим у нас есть метод:

```
void SayHello()
```

```
{
```

```
    Console.WriteLine("Hello");
```

```
}
```

Мы можем его сократить следующим образом:

```
void SayHello() => Console.WriteLine("Hello");
```

То есть после списка параметров ставится оператор `=>`, после которого идет выполняемая инструкция.

**Параметры** позволяют передать в метод некоторые входные данные. Параметры определяются через запятую в скобках после названия метода в виде:

```
тип_метода    имя_метода    (тип_параметра1    параметр1,  
тип_параметра2 параметр2, ...)
```

```
{
```

```
// действия метода
```

```
}
```

**Пример:**

```
void PrintMessage(string message)
```

```
{
```

```
    Console.WriteLine(message);
```

```
}
```

```
PrintMessage("Hello work");    // Hello work
```

По умолчанию при вызове метода необходимо предоставить значения для всех его параметров. Но С# также позволяет использовать необязательные параметры. Для таких параметров нам необходимо объявить значение по умолчанию. Также следует учитывать, что после необязательных параметров все последующие параметры также должны быть необязательными:

```
void PrintPerson(string name, int age = 1, string company = "Undefined")
```

```
{
```

```
    Console.WriteLine($"Name: {name}    Age: {age}    Company:  
{company}");
```

```
}
```

В предыдущих примерах при вызове методов значения для параметров передавались в порядке объявления этих параметров в методе. То есть аргументы передавались параметрам **по позиции**. Но мы можем нарушить подобный порядок, используя именованные параметры:

```
void PrintPerson(string name, int age = 1, string company = "Undefined")  
  
{  
  
    Console.WriteLine($"Name: {name} Age: {age} Company: {company}");  
  
}  
  
PrintPerson("Tom", company:"Microsoft", age: 37);
```

**Рекурсия** — это метод программирования или математического решения задач, при котором функция вызывает сама себя для решения более простой версии исходной задачи. В рекурсивных алгоритмах решение задачи сводится к решению одного или нескольких подзадач того же типа, что и исходная задача.

Для правильной работы рекурсивной функции необходимо два условия: Базовый случай (условие завершения), при котором функция перестает вызывать себя и возвращает результат. Рекурсивный случай, при котором функция продолжает вызывать себя для решения более простых подзадач.

Рекурсия может быть **прямой** или **косвенной**, в зависимости от того, как функция вызывает саму себя.

## **Прямая рекурсия**

**Прямая рекурсия** происходит, когда функция вызывает саму себя напрямую. Это наиболее распространенный тип рекурсии, при котором внутри тела функции происходит прямой вызов этой же функции

## **Косвенная рекурсия**

**Косвенная рекурсия** возникает, когда функция не вызывает саму себя напрямую, но вызывает другую функцию, которая в свою очередь вызывает исходную функцию. Таким образом, функция вызывает себя через одну или несколько промежуточных функций.

Также рекурсия может быть классифицирована на три типа с точки зрения направления вычислений:

**Нисходящая (Top-Down)** рекурсия;

**Восходящая (Bottom-Up)** рекурсия;

**Восходяще-нисходящая (Bidirectional)** рекурсия.

## Задание 1

### Задача:

#### Разворот числа

С клавиатуры вводится число  $n$ , десятичная запись которого не содержит нулей.

Реализуйте рекурсивный метод, который возвращает число, записанное теми же цифрами, но в противоположном порядке.

При решении данной задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика.

Метод должен возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.

#### Пример:

\*\*\*\*\*

*Ввод: 158789457*

*Вывод: 754987851*

\*\*\*\*\*

### Решение:

```
using System;

class Program
{
    static void Main()
    {
        int n;
        while (true)
        {
            Console.WriteLine("Введите целое число, не содержащее нулей: ");
            string input = Console.ReadLine();
            if (!int.TryParse(input, out n))
            {
                Console.WriteLine("Ошибка: введено не целое число.");
                continue;
            }
            if (input.Contains('0'))
            {
                Console.WriteLine("Ошибка: число не должно содержать нулей.");
                continue;
            }
        }
    }
}
```

```

    }
    if (n < 0)
    {
        Console.WriteLine("Ошибка: число должно быть положительным.");
        continue;
    }
    break;
}
int reversed = ReverseNumber(n);
Console.WriteLine($"Результат: {reversed}");
}
static int ReverseNumber(int n)
{
    if (n < 10)
        return n;
    int lastDigit = n % 10;
    int remaining = n / 10;
    int power = (int)Math.Log10(remaining) + 1;
    return lastDigit * (int)Math.Pow(10, power) + ReverseNumber(remaining);
}
}

```

**Ответ:**

Консоль отладки Microsoft Visual Studio

```

Введите целое число, не содержащее нулей: 1234045
Ошибка: число не должно содержать нулей.
Введите целое число, не содержащее нулей: 0123456
Ошибка: введено не целое число.
Введите целое число, не содержащее нулей: 123456
Результат: 654321

```

## Задание 2

### Задача:

#### Функция Аккермана

В теории вычислимости важную роль играет функция Аккермана  $A(m,n)$ , определенная следующим образом:

$$A(m, n) = \begin{cases} n + 1, & m = 0; \\ A(m - 1, 1), & m > 0, n = 0; \\ A(m - 1, A(m, n - 1)), & m > 0, n > 0. \end{cases}$$

С клавиатуры вводятся два целых неотрицательных числа  $m$  и  $n$ .  
Реализуйте рекурсивный метод  $A(m,n)$ .

#### Пример:

\*\*\*\*\*

*Ввод:  $m=2$   $n=3$*

*Вывод:  $A(m,n)=9$*

*Ввод:  $m=3$   $n=3$*

*Вывод:  $A(m,n)=61$*

\*\*\*\*\*

### Решение:

```
using System;

class Program
{
    private const int MAX_RECURSION_DEPTH = 10_000;

    static void Main(string[] args)
    {
        Console.WriteLine("Введите два целых неотрицательных числа m и n:");

        Console.Write("m = ");
        if (!long.TryParse(Console.ReadLine(), out long m) || m < 0)
        {
            Console.WriteLine("Ошибка: m должно быть неотрицательным целым числом.");
            return;
        }

        Console.Write("n = ");
        if (!long.TryParse(Console.ReadLine(), out long n) || n < 0)
        {
            Console.WriteLine("Ошибка: n должно быть неотрицательным целым числом.");
            return;
        }

        try
```



```

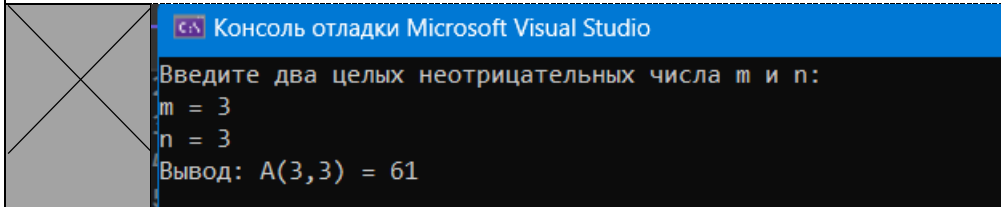
    {
        long result = AckermannSafe(m, n);
        Console.WriteLine($"Вывод: A({m},{n}) = {result}");
    }
    catch (InvalidOperationException ex)
    {
        Console.WriteLine($"Ошибка: {ex.Message}");
    }
    catch (OverflowException)
    {
        Console.WriteLine("Ошибка: переполнение числа (результат слишком велик для long).");
    }
}

public static long AckermannSafe(long m, long n, int depth = 0)
{
    if (depth > MAX_RECURSION_DEPTH)
        throw new InvalidOperationException("Превышен лимит рекурсии. Функция Аккермана растёт слишком быстро для этих аргументов.");

    if (m == 0)
    {
        return checked(n + 1);
    }
    else if (n == 0)
    {
        return AckermannSafe(m - 1, 1, depth + 1);
    }
    else
    {
        long inner = AckermannSafe(m, n - 1, depth + 1);
        return AckermannSafe(m - 1, inner, depth + 1);
    }
}
}

```

**Ответ:**



```

Консоль отладки Microsoft Visual Studio
Введите два целых неотрицательных числа m и n:
m = 3
n = 3
Вывод: A(3,3) = 61

```