Тема 8. Функции.

Учебные вопросы:

1. Определение и использование функций.
2. Передача параметров в функции.
3. Рекурсия.

1. Определение и использование функций.

Что такое функция?

Функция (или метод) в программировании — это самостоятельный блок кода, который выполняет определенную задачу.

Она может принимать входные данные (параметры), выполнять операции над этими данными, а затем возвращать результат.

В C# функции обычно объявляются внутри классов и **могут быть вызваны по имени**.

Основные элементы функции:

Тип возвращаемого значения: Указывает, что функция возвращает (например, **int, string, void** — если функция ничего не возвращает).

Имя функции: Идентификатор, с помощью которого можно вызвать функцию.

Параметры: Данные, которые передаются в функцию для обработки (необязательно).

Тело функции: Набор инструкций, которые выполняются при вызове функции.

Возвращаемое значение: Результат работы функции, который передается обратно вызывающему коду (если функция не является **void**).

Синтаксис объявления функции в C#

В C# функция (или метод) объявляется внутри класса и состоит из нескольких ключевых частей: типа возвращаемого значения, имени функции, параметров и тела функции.

Вот основной синтаксис:

```
<модификатор доступа> <тип возвращаемого значения> <имя функции>(<параметры>)
{
    // Тело функции
    <инструкции>;
    return <значение>; // (если тип возвращаемого значения не void)
}
```

Модификатор доступа (необязательный):

Указывает на видимость функции из других частей программы.

Пока разберем некоторые из них:

- **public** - делает метод доступным из любого места в программе. Такие методы могут быть вызваны из любого другого класса или файла.
- **private** - Методы с модификатором доступа private доступны только внутри того же класса, в котором они объявлены. Эти методы недоступны извне. Если модификатор доступа не указан явно, то для метода в классе по умолчанию устанавливается модификатор **private**.

- **static** указывает, что метод принадлежит самому классу, а не его экземпляру. Это означает, что такие методы можно вызывать без создания объекта класса. Статические методы могут обращаться только к другим статическим членам класса и не могут использовать нестатические поля или методы.

Из метода **Main** напрямую можно вызывать только статические методы.

На данном этапе мы будем создавать функции и вызывать их внутри одного класса, поэтому достаточно указать только один модификатор - **static**

Тип возвращаемого значения:

Определяет тип данных, который функция возвращает после выполнения.

Пример: **int, string, void** (если функция не возвращает значение).

```
public int Add(int a, int b)
```

Имя функции:

Уникальный идентификатор функции в рамках класса.

Имя должно быть говорящим, отражающим, что делает функция.

```
public int Add(int a, int b)
```

Параметры (необязательные):

Перечень входных данных, которые функция принимает для выполнения операции.

Указываются в круглых скобках через запятую.

Каждый параметр имеет тип и имя.

```
public int Add(int a, int b)
```

Тело функции:

Блок кода, заключенный в фигурные скобки {}, который выполняет основные операции функции.

Включает инструкции, выполняемые при вызове функции.

```
{
    int sum = a + b;
    return sum;
}
```

return (необязательно):

Используется для возврата значения из функции, если она не является **void**.

Завершает выполнение функции и возвращает указанное значение.

```
return sum;
```

Если указан тип возвращаемого значения не void, то метод **обязательно** должен возвращать значение, указанного типа.

```
using System;
namespace MyApp
{
    Ссылок: 0
    internal class Program
    {
        // функция не возвращает значение, тип void
        Ссылок: 1
        static void Print(string message)
        {
            Console.WriteLine($"Hello, {message}");
        }

        Ссылок: 0
        static void Main(string[] args)
        {
            Print("Bill");
        }
    }
}
```

```
using System;
namespace MyApp
{
    Ссылок: 0
    internal class Program
    {
        // функция возвращает значение, тип int
        Ссылок: 1
        static int Add(int x, int y)
        {
            return x + y;
        }

        Ссылок: 0
        static void Main(string[] args)
        {
            int a = 5, b = 10;
            int result = Add(a, b);
            Console.WriteLine($"{a} + {b} = {result}");
        }
    }
}
```

Вызов функции в C#

После того как функция была объявлена, ее можно вызвать для выполнения определенной задачи. Вызов функции включает указание имени функции и передачу необходимых параметров, если они требуются.

Синтаксис вызова функции:

```
<имя функции>(<аргументы>);
```

Аргументы: Значения, которые передаются в функцию в качестве параметров. Если функция не принимает параметры, скобки остаются пустыми.

Примеры вызова функций:

Вызов функции, возвращающей значение:

Если функция возвращает значение, это значение можно использовать как часть выражения, присвоить его переменной или передать в другую функцию.

```
int result = Add(5, 3);  // Вызов функции Add с аргументами 5 и 3
Console.WriteLine(result);  // Выводит 8
```

В этом примере: Функция **Add** вызывается с аргументами 5 и 3. Результат вызова (8) сохраняется в переменной **result**. Результат выводится на консоль с помощью Console.WriteLine.

Вызов функции, не возвращающей значение (void):

Если функция имеет тип возвращаемого значения void, она просто выполняет свои действия и ничего не возвращает.

```
PrintMessage("Hello, C#!");  // Вызов функции PrintMessage с аргументом "Hello, C#!"
```

В этом примере:

Функция PrintMessage вызывается с аргументом "Hello, C#!".

Функция выводит это сообщение на консоль, но не возвращает никакого значения.

Вызов функции без параметров:

Если функция не принимает параметров, ее можно вызвать просто указав имя функции с пустыми скобками.

```
ShowGreeting();  // Вызов функции ShowGreeting без аргументов
```

В этом примере:

Функция ShowGreeting вызывается без аргументов.

Функция выводит сообщение "Hello, world!" на консоль.

```
using System;
namespace MyApp
{
    Ссылок: 0
    internal class Program
    {
        // функция без параметров
        Ссылок: 1
        static void Hello()
        {
            Console.WriteLine("Hello, world!!!");
        }

        Ссылок: 0
        static void Main(string[] args)
        {
            Hello();
        }
    }
}
```

Вызов функции внутри другой функции:

Вы можете вызывать одну функцию из другой, что полезно для создания сложных программ.

```
public void Process()
{
    int sum = Add(10, 20);  // Вызов функции Add внутри функции Process
    PrintMessage("Sum is " + sum);  // Вызов функции PrintMessage с результатом
}
```

В этом примере: Внутри функции Process вызывается функция Add, результат которой сохраняется в переменной sum. Затем функция PrintMessage используется для вывода результата на консоль.

Передача результата функции в другую функцию:

Вы можете напрямую передавать результат одной функции в другую.

```
// Вызов функции Add и передача результата в Console.WriteLine
Console.WriteLine(Add(7, 3));
```

В этом примере:

Результат вызова Add(7, 3) сразу передается в Console.WriteLine, которая выводит 10 на консоль.

Основные моменты:

- Функцию (метод) можно объявить только внутри классов, структур или интерфейсов.
- Статические методы, такие как **Main**, не могут напрямую вызывать нестатические методы, т.е. новый метод должен иметь модификатор поведения **static**.
- Вызов функции осуществляется через указание имени функции и передаче необходимых аргументов.
- Если функция возвращает значение, его можно использовать для дальнейших операций.
- Если функция не возвращает значение (**void**), она просто выполняет свои действия без возврата данных.

3. Передача параметров в функции.

Передача параметров - это механизм, позволяющий передавать данные в функцию для выполнения каких-либо действий.

Эти данные называются параметрами функции.

Когда функция вызывается, ей передаются **аргументы**, которые соответствуют **параметрам** функции.

В C# существует несколько способов передачи параметров в функцию:

По значению:

При передаче по значению в функцию передается копия значения аргумента.

Изменения, внесенные в параметр внутри функции, не влияют на исходное значение аргумента.

Это наиболее распространенный способ передачи параметров.

По ссылке (модификатор ref):

При передаче по ссылке в функцию передается ссылка на переменную.

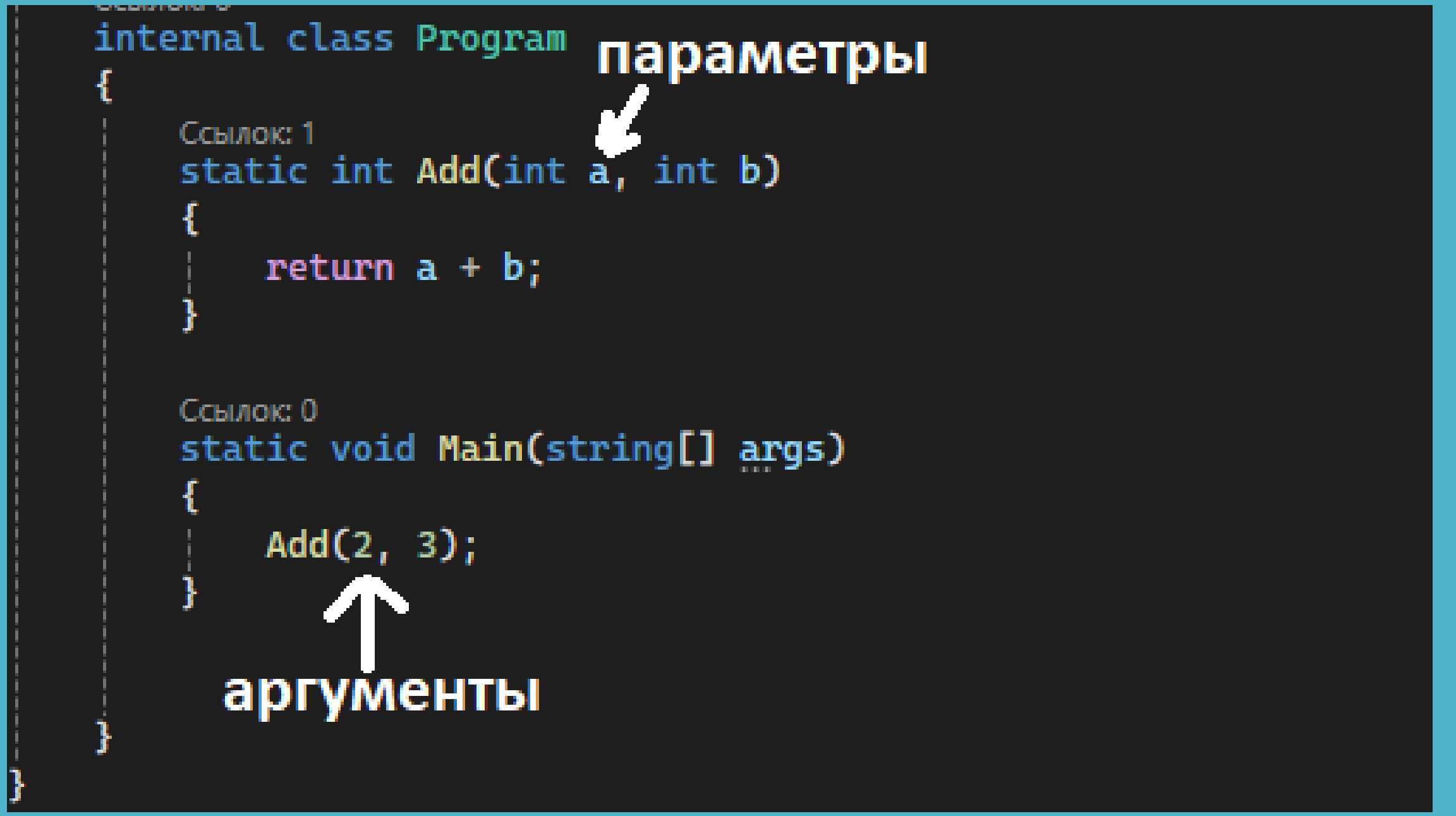
Изменения, внесенные в параметр внутри функции, изменяют исходное значение аргумента.

Перед передачей аргумента по ссылке он должен быть обязательно инициализирован.

Выходной параметр (модификатор out):

Похож на передачу по ссылке, но аргумент не обязательно должен быть инициализирован перед вызовом функции.

Функция гарантированно присвоит значение выходному параметру.


internal class Program
{
Ссылок: 1
static int Add(int a, int b)
{
return a + b;
}
Ссылок: 0
static void Main(string[] args)
{
Add(2, 3);
}
}
}
параметры
аргументы

Передача параметров по значению (по умолчанию)

По умолчанию параметры в C# передаются по значению. Это означает, что в функцию передается копия значения аргумента, и любые изменения, сделанные с параметром внутри функции, не повлияют на исходное значение вне функции.

```
Ссылок: 0
internal class Program
{

    Ссылок: 0
    static void Main(string[] args)
    {
        int number = 10;
        Console.WriteLine($"Значение до вызова ф-ии: {number}");
        // Передаем переменную в метод
        Increment(number);
        // После вызова метода значение переменной не изменится
        Console.WriteLine($"Значение после вызова ф-ии: {number}");
    }
    Ссылок: 1
    static void Increment(int num)
    {
        num += 5;
        Console.WriteLine($"Значение в ф-ии: {num}");
    }
}
}
```

Выбрать Консоль отладки Microsoft Visual Studio

```
Значение до вызова ф-ии: 10
Значение в ф-ии: 15
Значение после вызова ф-ии: 10
```

Проблемы не

Передача параметров по ссылке (ref)

Когда параметр передается по ссылке с использованием ключевого слова **ref**, функция получает доступ к оригинальной переменной.

Изменения внутри функции повлияют на значение переменной вне функции.

При этом переменная должна быть инициализирована до передачи в функцию.

```
internal class Program
{

    Ссылок: 0
    static void Main(string[] args)
    {
        int number = 10;
        Console.WriteLine($"Значение до вызова ф-ии: {number}");
        // Передаем переменную в метод
        Increment(ref number);
        // После вызова метода значение переменной не изменится
        Console.WriteLine($"Значение после вызова ф-ии: {number}");
    }
    Ссылок: 1
    static void Increment(ref int num)
    {
        num += 5;
        Console.WriteLine($"Значение в ф-ии: {num}");
    }
}
```

Консоль отладки Microsoft Visual Studio

```
Значение до вызова ф-ии: 10
Значение в ф-ии: 15
Значение после вызова ф-ии: 15
```

роблемы н

Передача параметров по ссылке с использованием out

Параметры, передаваемые с ключевым словом **out**, также передаются по ссылке, но в отличие от **ref**, переменная не обязана быть инициализирована перед передачей в функцию.

Функция, принимающая **out**-параметр, должна присвоить ему значение до завершения выполнения.

```
internal class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        // perimeter, area объявлены, но не инициализированы
        int sideA = 5, sideB = 10, perimeter, area;

        // Вызов функции для вычисления периметра и площади
        CalculateRectangle(sideA, sideB, out perimeter, out area);

        Console.WriteLine($"Периметр: {perimeter}"); // Вывод: Периметр: 30
        Console.WriteLine($"Площадь: {area}");       // Вывод: Площадь: 50
    }
    // Функция, принимающая параметр по ссылке с помощью 'out'
    Ссылок: 1
    public static void CalculateRectangle(int a, int b, out int perimeter, out int area)
    {
        perimeter = 2 * (a + b);
        area = a * b;
    }
}
```

Передача переменного количества параметров (params)

С помощью ключевого слова **params** можно передавать переменное количество аргументов в функцию. Аргументы собираются в массив, и функция может их обработать как массив.

```
Ссылок: 2
static void PrintNumbers(params int[] numbers)
{
    foreach (int number in numbers)
    {
        Console.WriteLine(number);
    }
}

Ссылок: 0
static void Main(string[] args)
{
    PrintNumbers(1, 2, 3, 4); // Выведет 1, 2, 3, 4
    PrintNumbers(); // Не вызовет ошибки и не выведет ничего
}
```

Передача значений по умолчанию

C# также поддерживает параметры по умолчанию, которые можно не указывать при вызове функции.

Если аргумент не указан, используется значение по умолчанию.

```
internal class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        // вывод: Hello, Alice
        Console.WriteLine(Greet("Alice"));
        //вывод: Hello, Guest
        Console.WriteLine(Greet());
    }

    Ссылок: 2
    static string Greet(string name = "Guest")
    {
        return $"Hello, {name}";
    }
}
```

Когда использовать какой способ?

- По значению: Для передачи данных, которые не нужно изменять внутри функции.
- По ссылке: Когда нужно изменить значение аргумента внутри функции и сохранить эти изменения.
- Выходной параметр: Когда функция возвращает несколько значений.

Выбор способа передачи зависит от конкретной задачи и логики программы.

4. Рекурсия.

В C# рекурсия — это метод, который вызывает сам себя. Рекурсия используется для решения задач, которые можно разбить на меньшие, идентичные подзадачи.

Основные понятия связанные с рекурсией:

- Базовый случай (условие завершения): Это условие, при выполнении которого рекурсия останавливается. Без базового случая рекурсия будет бесконечной и приведет к переполнению стека.
- Рекурсивный случай: Это часть метода, которая вызывает саму себя, решая подзадачи до тех пор, пока не будет достигнут базовый случай.

```
Ссылок: 2
public static int Factorial(int n)
{
    if (n <= 1) // Базовый случай
        return 1;
    else
        return n * Factorial(n - 1); // Рекурсивный случай
}

Ссылок: 0
static void Main(string[] args)
{
    int number = 5;
    int result = Factorial(number);
    Console.WriteLine($"Факториал {number} равен {result}");

}
```

```
Ссылок: 3
public static int Fibonacci(int n)
{
    if (n <= 1) // Базовые случаи
        return n;
    else
        return Fibonacci(n - 1) + Fibonacci(n - 2); // Рекурсивные вызовы
}

Ссылок: 0
static void Main(string[] args)
{
    int n = 10;
    int result = Fibonacci(n);
    Console.WriteLine($"Число Фибоначчи для n = {n} равно {result}");

}
```

Преимущества рекурсии:

- Читаемость: Рекурсивные решения часто бывают более интуитивными и легче понимаются для некоторых задач.
- Элегантность: Рекурсия позволяет выразить некоторые алгоритмы более лаконично.

Недостатки рекурсии:

- Производительность: Рекурсивные вызовы могут быть менее эффективны из-за накладных расходов на создание новых стековых фреймов.
- Сложность отладки: Бесконечная рекурсия может привести к переполнению стека.
- Потенциальная сложность понимания: Не все алгоритмы легко выразить рекурсивно, и понимание рекурсивных функций может потребовать некоторого времени.

Когда использовать рекурсию

- Задачи, которые могут быть разбиты на подзадачи того же типа: Например, вычисление факториала, поиск в дереве, вычисление чисел Фибоначчи.
- Задачи, которые имеют естественную рекурсивную структуру: Например, обход дерева.

Важно помнить

- Базовый случай: Всегда должен быть базовый случай, чтобы прервать рекурсию.
- Стек вызовов: Каждый рекурсивный вызов занимает место в стеке вызовов. Если рекурсия слишком глубокая, может произойти переполнение стека.
- Альтернативы: Часто рекурсивные алгоритмы можно переписать с использованием циклов.

Домашнее задание:

1. Повторить материал лекции.
2. Решить задачи: Учебное пособие, с. 65.

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com

Список литературы:

1. Жаксыбаева Н.Н. Основы объектно-ориентированного программирования: язык C#. Часть 1./ Учебное пособие предназначено для учащихся технического и профессионального образования, Алматы, 2010,

2. https://learn.microsoft.com/ru-ru/dotnet/csharp/