

**Московский государственный технический  
университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»  
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»  
Отчет по домашнему заданию  
«Изучение основных конструкций языка C#»

Выполнил:  
студент группы ИУ5-33Б  
Смирнов Артём

Проверил:  
преподаватель каф. ИУ5  
Гапанюк Ю. Е.

Москва, 2023 г.

## **Введение**

C# — это объектно-ориентированный язык программирования, созданный компанией Microsoft и работающий .NET фреймворке. Он был создан в 2000 году по инициативе Anders Hejlsberg. C# имеет корни из семейства C и близок к другим популярным языкам, таким как C++ и Java. Первая версия была выпущена в 2002 году. Последняя версия C# 12 была выпущена в ноябре 2023 года. Его популярность связана с широкими возможностями, безопасностью и отличной производительностью, поэтому C# по сей день используется для создания мобильных и настольных приложений, веб-сервисов, веб-сайтов, игр и много чего другого.

# Основы языка

## Типы данных

В языке C# есть следующие базовые типы данных:

- `bool`: хранит значение `true` или `false` (логические литералы). Представлен системным типом `System.Boolean`

```
bool alive = true;
bool isDead = false;
```

- `byte`: хранит целое число от 0 до 255 и занимает 1 байт. Представлен системным типом `System.Byte`

```
byte bit1 = 1;
byte bit2 = 102;
```

- `sbyte`: хранит целое число от -128 до 127 и занимает 1 байт. Представлен системным типом `System.SByte`

```
sbyte bit1 = -101;
sbyte bit2 = 102;
```

- `short`: хранит целое число от -32768 до 32767 и занимает 2 байта. Представлен системным типом `System.Int16`

```
short n1 = -768;
short n2 = 767;
```

- `ushort`: хранит целое число от 0 до 65535 и занимает 2 байта. Представлен системным типом `System.UInt16`

```
ushort n1 = 0;
ushort n2 = 102;
```

- `int`: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта. Представлен системным типом `System.Int32`. Все целочисленные литералы по умолчанию представляют значения типа `int`:

```
int a = 10;
int b = 0b101; // бинарная форма b = 5
int c = 0xFF;  // шестнадцатеричная форма c = 255
```

- `uint`: хранит целое число от 0 до 4294967295 и занимает 4 байта. Представлен системным типом `System.UInt32`

```
uint a = 10;
uint b = 0b101;
uint c = 0xFF;
```

- `long`: хранит целое число от  $-9\ 223\ 372\ 036\ 854\ 775\ 808$  до  $9\ 223\ 372\ 036\ 854\ 775\ 807$  и занимает 8 байт. Представлен системным типом `System.Int64`

```
long a = -9223372036854775;
long b = 0b101;
long c = 0xFF;
```

- `ulong`: хранит целое число от 0 до  $18\ 446\ 744\ 073\ 709\ 551\ 615$  и занимает 8 байт. Представлен системным типом `System.UInt64`

```
ulong a = 10;
ulong b = 0b101;
ulong c = 0xFF;
```

- `float`: хранит число с плавающей точкой от  $-3.4 \cdot 10^{38}$  до  $3.4 \cdot 10^{38}$  и занимает 4 байта. Представлен системным типом `System.Single`

```
float f = 3_000.5F;
```

- `double`: хранит число с плавающей точкой от  $\pm 5.0 \cdot 10^{-324}$  до  $\pm 1.7 \cdot 10^{308}$  и занимает 8 байта. Представлен системным типом `System.Double`

```
double d = 3D;
```

- `decimal`: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от  $\pm 1.0 \cdot 10^{-28}$  до  $\pm 7.9228 \cdot 10^{28}$ , может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом `System.Decimal`

```
decimal myMoney = 3_000.5m;
```

- `char`: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом `System.Char`. Этому типу соответствуют символьные литералы:

```
char a = 'A';
char b = '\x5A';
char c = '\u0420';
```

- `string`: хранит набор символов Unicode. Представлен системным типом `System.String`. Этому типу соответствуют строковые литералы.

```
string hello = "Hello";
string word = "world";
```

- `object`: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом `System.Object`, который является базовым для всех других типов и классов .NET.

```
object a = 22;
object b = 3.14;
object c = "hello code";
```

## Ввод и вывод

В C# ввод и вывод осуществляется через стандартные потоки ввода и вывода (System.Console). Для этого используются следующие методы:

- Console.ReadLine() - считывает строку текста.
- Console.WriteLine() - выводит строку текста.

Пример использования этих методов для ввода и вывода данных:

```
using System;

Ссылка: 0
class Program
{
    Ссылка: 0
    public static void Main(string[] args)
    {
        Console.WriteLine("Введите имя:");
        string userName = Console.ReadLine();
        Console.WriteLine("Ваше имя: " + userName);
    }
}
```

Результат:

```
C:\> Консоль отладки Microsoft Visual Studio
Введите имя:
Арте́м
Ваше имя: Арте́м
```

## Операторы if, else, switch

Оператор if else используется для выполнения блоков кода в зависимости от условия.  
Пример использования if else:

```
using System;

class Program
{
    static void Main()
    {
        int x = 6;

        if (x % 2 == 0)
        {
            Console.WriteLine("Число чётное");
        }
        else
        {
            Console.WriteLine("Число нечётное");
        }
    }
}
```

Результат:

```
Консоль отладки Microsoft Visual Studio
Число чётное
```

Оператор switch используется для выбора одного из нескольких блоков кода или значений на основе значения переменной или выражения.

```
using System;

class Program
{
    static void Main()
    {
        int day = 4;

        switch (day)
        {
            case 1:
                Console.WriteLine("Понедельник");
                break;
            case 2:
                Console.WriteLine("Вторник");
                break;
            case 3:
                Console.WriteLine("Среда");
                break;
            case 4:
                Console.WriteLine("Четверг");
                break;
            case 5:
                Console.WriteLine("Пятница");
                break;
            default:
                Console.WriteLine("Выходной");
                break;
        }
    }
}
```

Результат:

```
Консоль отладки Microsoft Visual Studio
Четверг
```

## Циклы

В C# имеется несколько видов циклов для обработки данных:

- **Цикл for:** Используется для выполнения блока кода определенное количество раз.
- **Цикл do-while:** Выполняет блок кода один раз, далее повторяет выполненное при соблюдении условия.
- **Цикл while:** Выполняет блок кода пока условие истинно.
- **Цикл foreach:** Используется для перебора элементов в коллекции.

```
using System;

Ссылка: 0
class Program
{
    Ссылка: 0
    static void Main()
    {
        // цикл for
        for (int j = 1; j <= 5; j++)
        {
            Console.WriteLine("Итерация номер " + j);
        }
        // цикл do while
        int i = 0;
        do
        {
            Console.WriteLine("Текущее значение i: " + i);
            i++;
        } while (i < 5);
        // цикл while
        i = 0;
        while (i < 5)
        {
            Console.WriteLine("Текущее значение i: " + i);
            i++;
        }
        // цикл for each
        int[] numbers = { 2, 4, 6, 8, 10 };
        foreach (int number in numbers)
        {
            Console.WriteLine("Текущее число: " + number);
        }
    }
}
```



Результат:

```
Консоль отладки Microsoft Visual Studio
Итерация номер 1
Итерация номер 2
Итерация номер 3
Итерация номер 4
Итерация номер 5
Текущее значение i: 0
Текущее значение i: 1
Текущее значение i: 2
Текущее значение i: 3
Текущее значение i: 4
Текущее значение i: 0
Текущее значение i: 1
Текущее значение i: 2
Текущее значение i: 3
Текущее значение i: 4
Текущее число: 2
Текущее число: 4
Текущее число: 6
Текущее число: 8
Текущее число: 10
```

## Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) является фундаментальным подходом в разработке на C#. В рамках этого стиля программирования программа моделируется в виде объектов, каждый из которых обладает своими уникальными свойствами и методами. Важно, что эти объекты могут взаимодействовать друг с другом, вызывая методы и получая значения свойств.

В основе ООП в C# лежат несколько ключевых принципов:

**Инкапсуляция** представляет собой концепцию, позволяющую скрыть внутренние детали реализации объекта и предоставить только ограниченный интерфейс для взаимодействия с ним.

**Наследование** открывает возможность создания новых классов на основе существующих с добавлением или изменением функциональности.

**Полиморфизм** позволяет использовать один интерфейс для выполнения различных действий в зависимости от типа объекта.

Для создания объектов в C# используются классы, которые определяют свойства и методы объекта. Каждый объект является экземпляром класса и может иметь свои уникальные характеристики и состояние.

Инкапсуляция является ключевым аспектом ООП, так как она способствует скрытию внутренних деталей объекта и предоставлению удобного интерфейса для его взаимодействия. Наследование помогает создавать иерархии классов, уменьшая дублирование кода и обеспечивая расширяемость программы. Полиморфизм, в свою очередь, обеспечивает гибкость, позволяя одному и тому же интерфейсу быть использованным для различных действий в зависимости от типа объекта.

Введенный подход ООП в C# способствует созданию более гибкого и понятного кода, облегчая его поддержку и развитие.

Пример работы с классами:

```
using System;

// Базовый класс
class Animal
{
    // Закрытое поле (инкапсуляция)
    private string species;

    // Конструктор с параметром
    public Animal(string species)
    {
        this.species = species;
    }

    // Метод, который будет переопределен в классах-наследниках (полиморфизм)
    public virtual void MakeSound()
    {
        Console.WriteLine("Звук животного");
    }
}

// Класс-наследник, расширяющий функциональность базового класса
class Dog : Animal
{
    // Конструктор класса-наследника
    public Dog(string species) : base(species) { }
}
```

```

        // Переопределение метода базового класса (полиморфизм)
        public override void MakeSound()
        {
            Console.WriteLine("Гав-гав!");
        }
    }

    // Класс-наследник, расширяющий функциональность базового класса
    class Cat : Animal
    {
        // Конструктор класса-наследника
        public Cat(string species) : base(species) { }

        // Переопределение метода базового класса (полиморфизм)
        public override void MakeSound()
        {
            Console.WriteLine("Мяу!");
        }
    }

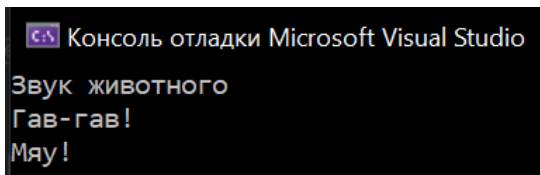
    class Program
    {
        static void Main()
        {
            // Создание объектов классов и вызов методов
            Animal someAnimal = new Animal("Some species");
            someAnimal.MakeSound(); // Вывод: "Звук животного"

            Dog dog = new Dog("Dog species");
            dog.MakeSound(); // Вывод: "Гав-гав!"

            Cat cat = new Cat("Cat species");
            cat.MakeSound(); // Вывод: "Мяу!"
        }
    }
}

```

Результат выполнения:



Консоль отладки Microsoft Visual Studio

```

Звук животного
Гав-гав!
Мяу!

```

## Асинхронное программирование

В этом примере используются ключевые стороны асинхронного программирования. В методе Main устанавливается ключевое слово `async`, что позволяет использовать оператор `await` для ожидания выполнения асинхронных операций. Метод `GetAsync` выполняется асинхронно, и затем с помощью `await` ожидается получение ответа от сервера. Далее, если запрос был успешным, ответ асинхронно считывается и выводится.

Асинхронное программирование в C# позволяет эффективно управлять ресурсами и повысить отзывчивость программы, особенно при выполнении долгих вводовывода операций, таких как сетевые запросы, базы данных и файловая система.

```
using System;
using System.Net.Http;
using System.Text.Json;
using System.Threading.Tasks;

// Ссылка 0
class Program
{
    // Ссылка 0
    static async Task Main()
    {
        // Создаем новый объект HttpClient для выполнения асинхронных HTTP-запросов
        var httpClient = new HttpClient();

        // Асинхронно отправляем GET-запрос на API для получения цитаты дня
        HttpResponseMessage response = await httpClient.GetAsync("https://favqs.com/api/qotd");

        // Проверяем успешность операции
        if (response.IsSuccessStatusCode)
        {
            // Асинхронно считываем и десериализуем содержимое ответа
            string jsonString = await response.Content.ReadAsStringAsync();
            var quoteData = JsonSerializer.Deserialize<QuoteData>(jsonString);
            Console.WriteLine(quoteData.quote.body);
            Console.WriteLine($"Author: {quoteData.quote.author}");
        }
        else
        {
            Console.WriteLine("Ошибка при получении данных");
        }
    }
}

// Ссылка 1
public class QuoteData
{
    // Ссылка 2
    public Quote quote { get; set; }
}

// Ссылка 1
public class Quote
{
    // Ссылка 1
    public string body { get; set; }
    // Ссылка 1
    public string author { get; set; }
}
```

Результат:

```
C:\> Консоль отладки Microsoft Visual Studio

Life isn't about finding yourself. Life is about creating yourself.
Author: George Bernard Shaw
```

## Пример программы

```
namespace To_Do_List
{
    class Program
    {
        static void Main(string[] args)
        {
            List<string> task_list = new List<string>();
            string option = "";
            while(option != "e")
            {
                Console.WriteLine("Что вы хотите сделать ?");
                Console.WriteLine("");
                Console.WriteLine("Нажмите 1, чтобы добавить задание");
                Console.WriteLine("Нажмите 2, чтобы удалить задание");
                Console.WriteLine("Нажмите 3, чтобы просмотреть список заданий");
                Console.WriteLine("Нажмите e, чтобы закрыть программу");
                Console.WriteLine("");

                option = Console.ReadLine();

                if (option == "1")
                {
                    Console.WriteLine("");
                    Console.WriteLine("Введите задание");
                    Console.WriteLine("");
                    string task = Console.ReadLine();
                    task_list.Add(task);
                    Console.WriteLine("");
                    Console.WriteLine("Задание добавлено в список");
                    Console.WriteLine("");
                }
                else if (option == "2")
                {
                    if (task_list.Count > 0) {
                        Console.WriteLine("");
                        for (int i = 0; i < task_list.Count; i++)
                        {
                            Console.WriteLine(i + 1 + " : " + task_list[i]);
                        }

                        Console.WriteLine("");
                        Console.WriteLine("Выберите, какое задание вы хотите удалить из списка ?");

                        Console.WriteLine("");
                        int TaskNumber = Convert.ToInt32(Console.ReadLine()) - 1;
                        task_list.RemoveAt(TaskNumber);
                    }
                    else
                    {
                        Console.WriteLine("");
                        Console.WriteLine("Невозможно - нет заданий в списке");
                        Console.WriteLine("");
                    }
                }
            }

            else if (option == "3")
            {
                if (task_list.Count > 0)
                {
                    Console.WriteLine("");
                    Console.WriteLine("Задания на текущий момент:");
                }
            }
        }
    }
}
```

```

        Console.WriteLine("");

        for (int i = 0; i < task_list.Count; i++)
        {
            Console.WriteLine(i + 1 + " : " + task_list[i]);
        }
        Console.WriteLine("");
    }
    else
    {
        Console.WriteLine("");
        Console.WriteLine("Нет заданий на текущий момент");
        Console.WriteLine("");
    }

}
else if (option == "e")
{
    Console.WriteLine("Выход из программы");
}
else
{
    Console.WriteLine("Неправильный ввод. Попробуйте еще раз");
}

}

}

}

```

## Результат:

Что вы хотите сделать ?

Нажмите 1, чтобы добавить задание

Нажмите 2, чтобы удалить задание

Нажмите 3, чтобы просмотреть список заданий

Нажмите е, чтобы закрыть программу

1

Введите задание

Убраться дома

Задание добавлено в список

Что вы хотите сделать ?

Нажмите 1, чтобы добавить задание

Нажмите 2, чтобы удалить задание

Нажмите 3, чтобы просмотреть список заданий

Нажмите е, чтобы закрыть программу

1

Введите задание

Сходить погулять

Задание добавлено в список

Что вы хотите сделать ?

Нажмите 1, чтобы добавить задание

Нажмите 2, чтобы удалить задание

Нажмите 3, чтобы просмотреть список заданий

Нажмите е, чтобы закрыть программу

2

1 : Убраться дома

2 : Сходить погулять

Выберите, какое задание вы хотите удалить из списка ?

2

Что вы хотите сделать ?

Нажмите 1, чтобы добавить задание

Нажмите 2, чтобы удалить задание

Нажмите 3, чтобы просмотреть список заданий

Нажмите е, чтобы закрыть программу

3

Задания на текущий момент:

1 : Убраться дома

Что вы хотите сделать ?

Нажмите 1, чтобы добавить задание

Нажмите 2, чтобы удалить задание

Нажмите 3, чтобы просмотреть список заданий

Нажмите е, чтобы закрыть программу

е

Выход из программы