

А.М. Минитаева

**СОЗДАНИЕ ВЕБ-СЕРВЕРОВ  
И БЭКЕНД РАЗРАБОТКА НА ЯЗЫКЕ C#**

*Учебно-методическое пособие*

УДК 681.3.06

ББК 22.18

М26

Издание доступно в электронном виде по адресу  
<https://bmstu.press/catalog/item/>

Факультет «Информатика и системы управления»  
Кафедра «Компьютерные системы и сети»

Рецензент:

*Т.Н. Ничушкина*

*Рекомендовано Научно-методическим советом  
МГТУ им. Н.Э. Баумана в качестве учебно-методического пособия*

**Минитаева, А. М.**

М26 Создание веб-серверов и бэкенд разработка на языке C#. Методические указания к выполнению домашних заданий № 1, 2 и 3 по дисциплине «Разработка приложений на языке C#» / А.М. Минитаева. – 1-е изд. – Москва : Издательство МГТУ им. Н.Э. Баумана, 2023. – 30, [8] с. : ил.

ISBN ...

Приведены основные теоретические сведения о языке C#, разработке консольных приложений, бэкенд компонентах информационных систем, веб-серверах и TCP-соединениях, необходимые для выполнения домашних заданий № 1, 2 и 3 по созданию HTTP серверов и бэкенд приложений на языке C#. Рассмотрены примеры.

Для студентов МГТУ им. Н.Э. Баумана, обучающихся по направлениям подготовки «Информатика и вычислительная техника» и «Прикладная информатика» и изучающих дисциплину «Разработка приложений на языке C#».

УДК 681.3.06

ББК 22.18

## ПРЕДИСЛОВИЕ

В языке программирования C# (созданном в компании Microsoft для поддержки собственной среды .NET Framework) проверенные временем средства языков C и C++ усовершенствованы с помощью самых современных технологий для решения полного круга задач, стоящих на сегодняшний день перед разработчиками. Язык C# и среда разработки Visual Studio представляют собой очень эффективный способ создания программного обеспечения для современной инфраструктуры вычислительной обработки данных, которая включает в себя требования кроссплатформенности, операционные системы Windows, Linux, Unix, Android и др., глобальную сеть Internet и локальные вычислительные сети, различные архитектуры процессоров и микроконтроллеров. Освоение языка программирования C# и среды разработки Visual Studio способствует расширению профессиональных возможностей студентов, получающих подготовку по ряду ИТ специальностей.

Учебно-методическое пособие, посвященное принципам backend разработки на языке C# в среде Visual Studio, поможет освоить технологию кроссплатформенной разработки программного обеспечения и выработать практические навыки создания веб-серверов и backend (серверных) компонентов программных продуктов широкого круга предназначения.

Студенты должны научиться самостоятельно создавать проект на языке C# в среде разработки Visual Studio. Для этого в теоретической части приведены необходимые инструкции, пояснения и примеры. В практической части подробно рассмотрены методика создания вычислительного программного обеспечения для практического изучения разработки консольных приложений, процесс разработки HTTP сервера, а также изложена методика создания веб-серверов и backend приложений, обеспечивающих взаимодействие с клиентом посредством JSON запросов.

Выполнение домашних заданий способствует развитию у студентов навыков самостоятельной работы.

**Цель домашних заданий** – изучение принципов разработки приложений на языке C# в среде разработки Visual Studio, предназначенных для выполнения самостоятельных вычислений или обработки данных, а также для выполнения backend (серверной) роли с предоставлением прикладного программного интерфейса (API) клиенту.

### **Задачи домашних заданий:**

- 1) ознакомление со средой разработки Visual Studio и синтаксисом языка программирования C#;
- 2) практическое освоение методики создания приложений на языке C#;
- 3) приобретение навыков технологичной разработки приложений, предназначенных для вычислительных и backend компонентов современных информационных систем.

## МОДУЛЬ 1. РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

Консольное приложение (или консольный интерфейс) – это приложение, которое запускается в командной строке (консоли) операционной системы и взаимодействует с пользователем через текстовый интерфейс. В консольном приложении пользователь вводит команды с помощью клавиатуры и видит вывод информации на экране монитора. Чтобы создать консольное приложение, необходимо сообщить среде разработки Visual Studio, что приложение будет использовать подсистему консоли.

Консольные приложения отлично подходят для выполнения рутинных задач и автоматизации различных процессов. Они могут быть написаны на языке программирования C#, и выполнять широкий спектр задач – от обработки данных до управления процессами операционной системы.

Примерами консольных приложений могут быть утилиты командной строки в ОС Windows или Unix (например, ping, ipconfig, ls), программы для работы с базами данных или сетевыми протоколами, а также различные инструменты для администрирования серверов.

## **Домашнее задание №1 «Консольный калькулятор арифметических выражений»**

### **Создание проекта консольного приложения в среде Visual Studio**

Создание консольного приложения на языке программирования C# в среде разработки Visual Studio состоит из нескольких простых шагов:

1. Откройте Visual Studio и выберите "Создать проект".
2. В открывшемся окне выберите "Консольное приложение".
3. Введите имя проекта и нажмите "Создать".
4. После создания проекта откроется файл Program.cs, в котором уже содержится шаблон кода для консольного приложения (например, метод Main).
5. Внесите нужные изменения в код, чтобы приложение выполняло нужные функции.

Пример кода для вывода приветствия в консоль:

Листинг 1.1 – Консольное приложение для вывода в консоль строки «Привет, мир!»

```
using System;

namespace MyConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
```

```

    {
        Console.WriteLine("Привет, мир!");
        Console.ReadLine(); // ожидание ввода
    }
}

```

6. Сохраните изменения и запустите приложение, нажав F5 или выбрав Debug > Start Debugging из меню Visual Studio.

### *Организация ввода/вывода в консольном режиме*

Для записи (вывода) информации в консоль существуют следующие методы класса `Console`: `Write(выводимая строка в кавычках или переменная строкового типа)`, `WriteLine(выводимая строка в кавычках или переменная строкового типа)`, для чтения (ввода) из консоли – метод `ReadLine()`, который возвращает введенную строку. Метод `ReadLine` всегда возвращает тип `string`, поэтому в случае необходимости преобразования в численный тип следует использовать преобразование данных. Параметром методов `Write`, `WriteLine` тоже должна быть переменная типа `string`. Однако, при выводе информации в консоль можно обойтись и без преобразования. Разница между `Write` и `WriteLine` состоит в том, что после вывода строки `WriteLine` переводит курсор на следующую строку консоли, `Write` оставляет курсор в текущей строке после выведенных данных.

Для преобразования типов данных предназначены методы класса `Convert`. Метод `ToInt32` переводит в целочисленный тип `int`; метод `ToString` переводит в вещественный тип `double`; метод `ToString` предназначен для перевода в строковый тип `string`. Все доступные методы класса `Convert` можно узнать благодаря подсказкам среды Visual Studio: для этого нужно набрать имя класса, поставить точку, и среда разработки покажет полный список его методов.

В следующем примере (листинг 1.2) мы вводим два числа и выполняем с ними вычисления, затем выводим результаты:

Листинг 1.2 – Консольное приложение для ввода двух переменных `n`, `p`, вычисления по их значениям переменных `q`, `r` и вывода их значений в консоль

```

namespace MyConsoleApp2
{
    class Program
    {
        static void Main(string[] args)
        {
            int n;
            double p, q, r;
            string s;

```

```

        Console.WriteLine("n="); //подсказка при вводе
        s = Console.ReadLine(); //ввод строки
        n = Convert.ToInt32(s); //преобразование
                                //строки в целое
        Console.WriteLine("p=");
        p = Convert.ToDouble(Console.ReadLine());
        //ввод, совмещенный с преобразованием
        q = n + p;
        r = 2 / n + p;
        Console.WriteLine("q=" + q);
        //вывод с автоматическим преобразованием
        Console.WriteLine(Convert.ToString(r));
        //вывод с явным преобразованием
        Console.ReadLine();
    }
}
}

```

Если параметр метода `WriteLine` содержит сложение символьной строки и числа/переменной ("ответ: "+5), то выполняется автоматическое преобразование. Можно использовать даже пустую строку, написав `WriteLine(" "+y)`. На внешний вид данных также можно повлиять форматным выводом. В следующем примере (листинг 1.3) мы используем формат для обработки переменных типа `decimal`.

Листинг 1.3 – Консольное приложение с форматным выводом переменной `dec3` типа `decimal`

```

namespace MyConsoleApp3
{
    class Program
    {
        static void Main(string[] args)
        {
            decimal dec1, dec2, dec3;
            string s;
            s=Console.ReadLine();
            dec1=Convert.ToDecimal(s);
            dec2=4.5m; //m или M признак константы decimal
            dec3=dec1+dec2;
            Console.WriteLine("Ответ :{0:###.##}", dec3);
            Console.ReadLine();
        }
    }
}

```

Формат `{0:###.##}` : запись формата состоит из номера аргумента и собственно формата.

## Обработка строки с арифметическим выражением

### Стековый метод

Пусть задана некоторая произвольная строка:

.... $\alpha\beta$ ....

Если два символа строки  $\alpha, \beta \in V$  расположены рядом в сентенциальной форме, то между ними возможны следующие отношения, названные отношениями предшествования:

- 1)  $\alpha$  принадлежит основе, а  $\beta$  – нет, т. е.  $\alpha$  – конец основы:  $\alpha \cdot > \beta$ ;
- 2)  $\beta$  принадлежит основе, а  $\alpha$  – нет, т. е.  $\beta$  – начало основы:  $\alpha < \cdot \beta$  ;
- 3)  $\alpha$  и  $\beta$  принадлежит одной основе, т. е.  $\alpha = \cdot \beta$ ;
- 4)  $\alpha$  и  $\beta$  не могут находиться рядом в сентенциальной форме (ошибка).

Обозначения отношений предшествования:

$< \cdot$  - начало основы;

$\cdot >$  - конец основы;

$=$  - одна основа;

$?$  – ошибка

В арифметических выражениях основой является операция или число, на которое направлено выражение. Например, в выражении " $2 + 3$ " основами являются знак сложения "+", так как это операция, на которую направлено выражение, а также числа 2 и 3, так как они тоже являются базовыми элементами выражения.

Блоки разбора арифметических выражений относятся к частям выражения, которые могут быть выделены для удобства его анализа и вычисления:

- Выражение (expression) - это часть арифметического выражения, которая может содержать один или более термов, операции и скобки. Например, выражение " $(3 + 4) * 5$ " содержит два терма " $3+4$ " и " $5$ ", операцию умножения и скобки.

- Терм (term) - это часть выражения, которая может содержать один или более множителей и операции умножения или деления. Например, терм " $3 + 4$ " содержит два множителя " $3$ " и " $4$ " и операцию сложения.

- Множитель (factor) - это часть терма, которая может быть числом, переменной, функцией или выражением. Например, множитель " $3$ " является числом, а множитель " $x$ " может быть переменной.

- Идентификатор (identifier) - это символьное имя, присвоенное переменной или функции, которое используется для обращения к ним. Например, в выражении " $x + \sin(5)$ " идентификатор " $x$ " обозначает переменную, а идентификатор " $\sin$ " обозначает функцию синуса.

Эти блоки комбинируются между собой, чтобы составить арифметическое выражение. Например, выражение " $2 * x + 5$ " можно разбить на термы " $2*x$ " и " $5$ ", а терм " $2*x$ " на множители " $2$ " и " $x$ ". Идентификатор " $x$ " может быть использован как множитель в разных термах.

Порядок комбинирования блоков в выражении определяется приоритетом операций и порядком выполнения скобок. Например, в выражении "2 + 3 \* 4" сначала будет выполнено умножение "3 \* 4", а затем сложение "2 + 12". Если же изменить порядок скобок в выражении: "(2 + 3) \* 4", то сначала будет выполнено сложение в скобках "2 + 3", а затем умножение на число "4".

Формально блоки Выражение, Терм, Множитель и Идентификатор определяются следующим рекуррентным образом, где вертикальная черта "|" обозначает "или":

$\langle \text{Выражение} \rangle ::= \langle \text{Терм} \rangle | \langle \text{Выражение} \rangle + \langle \text{Терм} \rangle | \langle \text{Выражение} \rangle - \langle \text{Терм} \rangle$

$\langle \text{Терм} \rangle ::= \langle \text{Множитель} \rangle | \langle \text{Терм} \rangle * \langle \text{Множитель} \rangle | \langle \text{Терм} \rangle / \langle \text{Множитель} \rangle$

$\langle \text{Множитель} \rangle ::= (\langle \text{Выражение} \rangle) | \langle \text{Идентификатор} \rangle$

При разборе арифметического выражения стековым методом строка с выражением, начиная слева, последовательно формирует анализируемые символы и их отношения предшествования (рис. 1). Для отношения  $\langle \cdot$  формируется операция переноса (анализируемые символы помещаются в стек), для отношений  $\cdot >$  и  $=$  формируется операция свертки и соответствующая тройка (рис. 1). Результатом свертки может являться выражение, множитель или терм. После свертки анализируемыми символами становится результат выполнения операции ее тройки, затем эти анализируемые символы извлекаются (удаляются) из стека.

	+	*	(	)	◀
▶	<.	<.	<.	?	Выход
+	.>	<.	<.	.>	.>
*	.>	.>	<.	.>	.>
(	<.	<.	<.	=	?
)	.>	.>	?	.>	.>

  

▶A+	▶A*	▶(A	▶A)	▶A◀
+A+	+A*	+(A	+A)	+A◀
*A+	*A*	*(A	*A)	*A◀
(A+	(A*	((A	(A)	(A◀
A)+	A)*	A)(	A))	A)◀

**Рис. 1.** Возможные отношения предшествования и возможные тройки при разборе арифметического выражения стековым методом

**Пример.** Рассмотрим разбор выражения  $\blacktriangleright d+c*(a+b) \blacktriangleleft$  стековым методом. Вышеописанная процедура разбора арифметического выражения наглядно представлена на рисунке 2.



Содержимое стека	Анализируемые символы	Отношение	Операция	Тройка	Результат свертки
►	d+	$<\square$	Перенос		
► d+	c*	$<\square$	Перенос		
► d+ c*	(	$<\square$	Перенос		
► d+ c*(	a+	$<\square$	Перенос		
► d+ c*( a+	b)	$\square>$	Свертка	$R_1 := a + b$	<Выражение>
► d+ c*(	$R_1)$	$=\square$	Свертка	$R_1 := (R_1)$	<Множитель>
► d+ c*	$R_1 \blacktriangleleft$	$\square>$	Свертка	$R_2 := c^* R_1$	<Терм>
► d+	$R_2 \blacktriangleleft$	$\square>$	Свертка	$R_3 := d + R_2$	<Выражение>
►	$R_3 \blacktriangleleft$	Конец			

**Рис. 2.** Пример разбора выражения  $d+c^*(a+b)$  стековым методом

### Обратная польская запись. Алгоритм Бауэра-Замельзона

В польской записи операции расположены перед операндами, в обратной польской записи операнды располагаются перед операциями.

Обратная польская запись представляет собой последовательность команд двух типов:

1.  $K_\eta$ , где  $\eta$  – идентификатор операнда – выбрать число по имени  $\eta$  и заслать его в стек операндов;
2.  $K_\xi$ , где  $\xi$  – операция – выбрать два верхних числа из стека операндов, произвести над ними операцию  $\xi$  и занести результат в стек операндов.

**Пример.**  $A+B*C \Rightarrow K_A K_B K_C K_* K_+$  или  $ABC*+$

### Построение обратной польской записи:

- а) если символ – операнд, то вырабатывается команда  $K_I$ ,
- б) если символ – операция, то выполняются действия для соответствующего сочетания  $\eta$  и  $\xi$  согласно рисунку 3.

		$\xi$					
		+	*	(	)	$\blacktriangleleft$	
$\eta$	►	<.	<.	<.	?	Выход	
	+	.>	<.	<.	.>	.>	
	*	.>	.>	<.	.>	.>	
	(	<.	<.	<.	=	?	
	)	.>	.>	?	.>	.>	

$\eta \backslash \xi$	+	*	(	)	$\leftarrow$
→	I	I	I	?	Вых
+	II	I	I	IV	IV
*	IV	II	I	IV	IV
(	I	I	I	III	?

**Рис. 3.** Определение операции для сочетаний  $\eta$  и  $\xi$

Операции:

I – записать  $\xi$  в стек операций и прочесть следующий символ;

II – сгенерировать  $K_{\eta}$  , записать  $\xi$  в стек операций и прочитать следующий символ;

III – удалить верхний символ из стека операций и прочитать следующий символ;

IV – сгенерировать  $K_{\eta}$  и повторить с тем же входным символом.

**Пример.** Построение троек для  $(a+b*c)/d$ .

Ожидаемый результат:  $K_a K_b K_c K_* K_+ K_d K_ /$  или  $abc*+d/$

Последовательность выполнения операций построения обратной польской записи показана на рисунке 4.

Стек операций	Символ	Действие	Команда
►	(	I	
► (	a		$K_a$
► (	+	I	
► (+	b		$K_b$
► (+	*	I	
► (+ *	c		$K_c$
► (+ *	)	IV	$K_*$
► (+	)	IV	$K_+$
► (	)	III	
►	/	I	
► /	d		$K_d$
► /	←	IV	$K_ /$
►	←	Конец	

**Рис. 4.** Пример построения обратной польской записи выражения  $(a+b*c)/d$

Полученный результат  $K_a K_b K_c K_* K_+ K_d K_ /$  или  $abc*+d/$  обрабатывается слева направо, формируется стек операндов и тройки, результаты троек также помещаются в стек до тех пор, пока в стеке не окажется окончательный результат вычисления выражения (рис. 5).

Стек операндов	Команда	Тройка
$\emptyset$	$K_a$	
a	$K_b$	
a b	$K_c$	
a b c	$K_*$	$T_1 = b*c$
a $T_1$	$K_+$	$T_2 = a+T_1$
$T_2$	$K_d$	
$T_2 d$	$K_ /$	$T_3 = T_2/d$
$T_3$		

**Рис. 5.** Последовательность вычисления результата арифметического выражения  $abc*+d/$  в обратной польской записи

### **Алгоритм вычисления выражения в обратной польской записи:**

1. Создать пустой стек операндов.
2. Разбить арифметическое выражение на токены и пройти по ним слева направо.
3. Если токен является операндом, то поместить его в стек операндов.
4. Если токен является оператором, то извлечь два последних операнда из стека операндов.
5. Выполнить операцию, указанную оператором, над извлеченными операндами.
6. Поместить результат обратно в стек операндов.
7. Повторять шаги 3-6 до тех пор, пока не будет достигнут конец выражения.
8. В результате стек операндов останется с единственным элементом, который будет являться результатом вычисления выражения.

### **Реализация автоматизированного тестирования**

Если функция `algorithm()`, получающая в качестве параметра входную строку с арифметическим выражением, отдает результат его вычисления, то автоматизированное тестирование этой функции может быть реализовано с помощью кода, представленного в листинге 1.4.

#### **Листинг 1.4 – Код модуля автотеста**

```
using System;
using System.IO;

namespace DZ_1
{
    class Program {

        static void Main(string[] args)
        {
            string[] inputs = {"1+2", "(234-11)*34", "6*6/6"}; // Пример
            string[] outputs = {"3", "7582", "6"}; // Пример

            for(int i = 0; i < inputs.Length; i++) {
                Console.WriteLine(inputs[i] + "=" + outputs[i]);
                if (algorithm(inputs[i]) == outputs[i]) {
                    Console.WriteLine("OK");
                } else {
                    Console.WriteLine("Failed");
                }
            }
        }
    }
}
```

Указанные в листинге 1.4 три арифметических выражения необходимо расширить 7 своими дополнительными выражениями и результатами их верного вычисления.

### **Практическая часть**

**Задание:** разработать на языке C# в среде разработки Visual Studio консольное приложение «калькулятор», позволяющее вычислять арифметическое выражение, подаваемое на STDIN. Результат округлять до целого значения.

Требуется реализовать:

- сложение;
- вычитание;
- умножение;
- деление;
- поддержка скобок.

Нужно написать тесты, которые покрывают все операции.

**Пример:**

"(2+3)-4" => 1

"4-(2\*3)" => 2

**Варианты:**

- нечетные по списку – стековым методом;
- четные по списку – методом Бауэра-Земельзона.

### **Структура отчета по домашнему заданию**

Отчет является документом, свидетельствующим о выполнении студентом домашнего задания. Отчет должен включать:

- 1) Титульный лист с номером и названием домашнего задания, идентификатором группы, номером варианта, фамилией студента, датой сдачи отчета по домашнему заданию, местом для подписи преподавателя;
- 2) Введение, цель и задачи домашнего задания;
- 3) Описание работы программы;
- 4) Схему алгоритма;
- 5) Листинг программы (код должен быть самодокументирован, т.е. содержать комментарии), включая модуль автотестирования;
- 6) Демонстрацию работы приложения;
- 7) Демонстрацию пройденных тестов;
- 8) Выводы по домашнему заданию.

## Контрольные вопросы по домашнему заданию №1

- 1) Дайте определение консольного приложения.
- 2) Какие существуют способы вывода информации в консоль? Чем они отличаются?
- 3) Как преобразовать строку в число?
- 4) Как преобразовать число в строку?
- 5) Дайте рекуррентные определения блоков Выражение, Терм, Множитель и Идентификатор.
- 6) Какие существуют отношения предшествования? Как они обозначаются?
- 7) Опишите алгоритм разбора и вычисления арифметического выражения стековым методом.
- 8) Опишите алгоритм разбора арифметического выражения для формирования польской записи.
- 9) Опишите алгоритм вычисления арифметического выражения, записанного в польской записи.
- 10) Как можно автоматизировать процесс тестирования программного модуля?

## МОДУЛЬ 2. РАЗРАБОТКА ВЕБ-СЕРВЕРА

Веб-сервер (web server) – это программа, которая обрабатывает запросы на доступ к веб-страницам и возвращает клиенту (браузеру) запрошенную информацию. Веб-сервер может обслуживать несколько клиентов одновременно, обрабатывая запросы параллельно.

HTTP сервер является разновидностью веб-сервера, который работает с протоколом HTTP (HyperText Transfer Protocol). HTTP – это протокол прикладного уровня для передачи гипертекста (веб-страниц, представляющих собой текст с разметкой HTML) в сети Интернет.

Протокол HTTP использует клиент-серверную модель, где клиент отправляет запрос на сервер, а сервер возвращает ответ. Формат запроса и ответа определен протоколом.

Каждый запрос содержит метод, URL, версию протокола, различные заголовки и данные (необязательно). Методы запроса в HTTP включают GET, POST, PUT и DELETE. Например, GET-запрос используется для получения информации от сервера, а POST-запрос для отправки данных на сервер для дальнейшей обработки.

Каждый ответ содержит версию протокола, код статуса, заголовки и данные (необязательно). Код статуса определяет результат выполнения запроса (например, 200 означает "OK", а 404 означает "Not Found").

Клиент и сервер могут обмениваться более чем одним запросом и ответом в рамках одной сессии, для уменьшения нагрузки на сервер и сеть.

Таким образом, работа HTTP-протокола предполагает передачу данных между клиентом и сервером в формате запрос-ответ, используя определенные методы, заголовки и статус-коды, устанавливаемые протоколом.

## **Домашнее задание №2 «HTTP сервер, отдающий статичную html страницу»**

### **Краткая характеристика TCP-подключения**

Действующее TCP-подключение – это взаимодействие между двумя устройствами по протоколу TCP. Оно устанавливается через сеть между двумя приложениями, одно из которых действует в качестве сервера, а другое в качестве клиента. Подключение устанавливается путем установления соединения между клиентским и серверным приложением, после чего они могут обмениваться данными по протоколу TCP.

Порт – это логический адрес, который используется для идентификации конечной точки сетевого соединения на устройстве. В протоколе TCP/IP порт представляет собой 16-битное число, которое идентифицирует конкретный сетевой процесс на устройстве. Номер порта указывается десятичным числом от 1 до 65535, для передачи информации по HTTP протоколу обычно (но необязательно) используется 80 порт.

В протоколе TCP/IP номера портов делятся на три диапазона: зарезервированные, зарегистрированные и частные.

Зарезервированные порты (1-1023) предназначены для системных служб и протоколов, таких как HTTP (порт 80), HTTPS (порт 443), FTP (порт 21) и т.д. Использование зарезервированных портов в пользовательском приложении не рекомендуется.

Зарегистрированные порты (1024-49151) зарезервированы для приложений и служб, зарегистрированных в Internet Assigned Numbers Authority (IANA). Эти порты могут использоваться приложениями, которые работают в сети Интернет.

Частные порты (49152-65535) предназначены для использования в локальных сетях, обычно в приложениях, которые не работают в Интернете.

Свободными считаются порты, которые не заняты никакими системными службами и приложениями на конкретном устройстве. Поэтому при выборе порта для своего приложения следует проверить, что этот порт не занят другими приложениями на устройстве и не используется системными службами.

Порт и TCP-подключение тесно связаны между собой, так как для установления TCP-подключения необходимо указать номер порта, на котором серверное приложение ожидает подключения. Когда клиентское приложение устанавливает соединение с сервером, оно указывает номер порта, на котором оно желает получить данные от сервера. После этого сервер принимает соединение на этом порту и начинает обмениваться данными с клиентом.

## Классы, используемые для ТСП-подключения

### *Класс TcpListener*

Класс TcpListener в .NET Framework описывает собой объект, который слушает определенный порт на локальном компьютере и принимает входящие ТСП-подключения от клиентов. После принятия подключения объект TcpListener создает новый объект TcpClient, который устанавливает соединение с клиентом.

Вот некоторые из наиболее часто используемых методов и полей класса TcpListener:

#### **Поля:**

- LocalEndpoint: локальный конечный узел, к которому TcpListener привязан;
- Server: сокет, связанный с TcpListener.

#### **Методы:**

- Start(): Запускает прослушивание входящих соединений на указанном порту;
- Stop(): Останавливает прослушивание входящих соединений;
- AcceptTcpClient(): Ожидает и принимает входящее соединение и возвращает объект TcpClient, который устанавливает соединение с клиентом;
- AcceptSocket(): Ожидает и принимает входящее соединение и возвращает сокет, который устанавливает соединение с клиентом;

Класс TcpListener используется для создания ТСП-сервера, который может обрабатывать входящие подключения от клиентов и взаимодействовать с ними по протоколу ТСП. Обычно он используется в сетевых приложениях для обмена данными между клиентами и сервером.

На рисунке 6 представлен пример создания нового объекта класса TcpListener.

```

class Server {
    private int _port { get; }
    TcpListener _listener;
    public Server() {
        _port = 8080;
    }

    public void Start() {
        _listener = new TcpListener(IPAddress.Any, _port);

        try
        {
            _listener.Start();
        }
        catch (Exception e)
        {
            throw;
        }
        while (true)
        {
            TcpClient Client = Listener.AcceptTcpClient();
            ...
        }
    }

    ...

    ~Server()
    {
        if (_listener != null)
        {
            _listener.Stop();
        }
    }
}

```

**Рис. 6.** Создание нового объекта класса TcpListener

Класс TcpListener предоставляет простые методы, которые прослушивают и принимают входящие запросы на подключение в блокирующем синхронном режиме. Метод Start используется, чтобы начать прослушивание входящих запросов на подключение. Он будет помещать входящие подключения в очередь, пока не будет вызван метод Stop. Метод AcceptTcpClient извлекает подключения из очереди входящих запросов на подключение.

#### *Класс TcpClient*

Класс TcpClient в языке программирования C# представляет TCP-клиентское соединение. Он используется для установления соединения с удаленным сервером, отправки и получения данных.

#### **Поля класса TcpClient:**

- AddressFamily - возвращает или задает семейство адресов, которые будут использоваться для соединения. По умолчанию используется AddressFamily.InterNetwork, которое соответствует IPv4;



- Client - возвращает экземпляр класса Socket, который используется для обмена данными с сервером;
- Connected - возвращает true, если клиентское соединение открыто и установлено;
- Available - возвращает количество байт, доступных для чтения из потока сетевых данных.

#### **Методы класса TcpClient:**

- Connect(string host, int port) - устанавливает соединение с удаленным хостом по указанному адресу и номеру порта;
- GetStream() - возвращает объект NetworkStream, связанный с текущим клиентским соединением, который используется для чтения и записи данных;
- Close() - закрывает текущее клиентское соединение;
- Dispose() - освобождает все ресурсы, связанные с текущим экземпляром TcpClient.

Класс TcpClient используется для создания клиентских приложений, которые устанавливают соединение с удаленным сервером и обмениваются данными по протоколу TCP. Приложения могут отправлять и принимать данные в виде байтового потока через сетевой поток, возвращаемый методом GetStream(). Для закрытия соединения следует вызвать метод Close() или Dispose(). Класс TcpClient предоставляет простые методы для подключения, отправки и получения потоковых данных по сети в синхронном режиме блокировки.

Чтобы объект класса TcpClient смог подключиться и обмениваться данными, объекту класса TcpListener необходимо прослушивать входящие запросы на подключение.

### **Создание HTTP сервера**

Для создания HTTP сервера на языке C# с использованием класса TcpListener в Visual Studio следует создать консольное приложение. Для этого следует выбрать пункт "Console Application" при создании нового проекта.

Для создания HTTP сервера на языке C# с использованием класса TcpListener следует выполнить следующие шаги:

1. Создать новый проект в Visual Studio.
2. Добавить в проект файл с расширением ".cs".
3. Добавить необходимые пространства имен:

#### **Листинг 2.1 – Импорт пространств имен для создания HTTP сервера**

```
using System;  
using System.IO;  
using System.Net;  
using System.Net.Sockets;  
using System.Text;
```

```
using System.Threading;
using System.Threading.Tasks;
```

#### 4. Создать класс с именем "HttpServer".

#### Листинг 2.2 – Код класса HttpServer

```
class HttpServer
{
    private readonly TcpListener _listener;
    private readonly bool _useThreadPool;

    public HttpServer(int port, bool useThreadPool)
    {
        _listener = new TcpListener(IPAddress.Any, port);
        _useThreadPool = useThreadPool;
    }

    public void Start()
    {
        _listener.Start();

        Console.WriteLine("Server started.");

        while (true)
        {
            TcpClient client = _listener.AcceptTcpClient();

            if (_useThreadPool)
            {
                ThreadPool.QueueUserWorkItem(ProcessClient, client);
            }
            else
            {
                ProcessClient(client);
            }
        }
    }

    private void ProcessClient(object obj)
    {
        TcpClient client = obj as TcpClient;

        if (client == null)
            return;

        using (NetworkStream stream = client.GetStream())
        {
            byte[] request = new byte[1024];
            int bytesRead = stream.Read(request, 0, request.Length);
        }
    }
}
```

```

string message = Encoding.ASCII.GetString(request, 0, bytesRead);
Console.WriteLine("Received message: " + message);

string response = "HTTP/1.1 200 OK\r\n" +
    "Content-Type: text/html; charset=UTF-8\r\n" +
    "Connection: close\r\n\r\n" +
    "<html><body><h1>Hello, world!</h1></body></html>";

byte[] buffer = Encoding.ASCII.GetBytes(response);

stream.Write(buffer, 0, buffer.Length);
stream.Flush();
}

client.Close();
}
}

```

В этом примере (листинг 2.2) сервер принимает входящие запросы и отправляет ответы с помощью класса `TcpListener`. При получении нового запроса сервер запускает процесс, который обрабатывает этот запрос.

Существуют две реализации работы с запросами (`Thread per Request` и `Pool Thread`), в конструкторе класса `HttpServer(int port, bool useThreadPool)` мы предусмотрели булевский параметр, отвечающий за выбор способа реализации. Реализации `Thread per Request` и `Pool Thread` отличаются тем, как обрабатываются запросы на сервере.

### *Реализация Thread per Request*

В реализации `Thread per Request` каждый новый запрос обрабатывается в отдельном потоке, который создается для этого запроса. Это простой и понятный способ обработки запросов, но может приводить к проблемам с производительностью при большой нагрузке на сервер, так как создание и уничтожение потоков для каждого запроса может занимать много времени и ресурсов.

Листинг 2.3 – Способ запуска серверного объекта класса `HttpServer` для реализации `Thread per Request`

```

HttpServer server = new HttpServer(8080, false);
server.Start();

```

В данной реализации (рисунок 7), где `Worker` – это собственный класс для выполнения полезной нагрузки, создается по потоку на каждого клиента. Она имеет недостаток в том, что сервер может не выдержать высокой нагрузки, но зато можно работать с практически неограниченным количеством клиентов одновременно.

```

public void Start() {
    _listener = new TcpListener(IPAddress.Any, _port);

    try
    {
        _listener.Start();
    }
    catch (Exception e)
    {
        throw;
    }
    while (true) {
        TcpClient Client = Listener.AcceptTcpClient();
        Thread Thread = new Thread(new
ParameterizedThreadStart(WorkerThread));
        Thread.Start(Client);
    }
}

private static void WorkerThread(Object listener)
{
    new Worker((TcpListener)listener);
}

```

**Рис. 7.** Реализация Thread per Request

#### **Преимущества реализации Thread per Request:**

- Простота реализации
- Возможность обрабатывать запросы параллельно

#### **Недостатки реализации Thread per Request:**

- Создание и уничтожение потоков занимает много времени и ресурсов;
- При большой нагрузке на сервер может привести к проблемам с производительностью.

#### *Реализация Pool Thread*

В реализации Pool Thread используется пул потоков для обработки запросов. Вместо создания новых потоков для каждого запроса используются существующие потоки из пула. Это позволяет более эффективно использовать ресурсы сервера и обрабатывать большое количество запросов при меньшем количестве потоков. Однако это может привести к ситуации, когда все потоки в пуле будут заняты обработкой запросов, и новые запросы будут ждать, пока один из потоков не станет доступен.

Листинг 2.4 – Способ запуска серверного объекта класса HttpServer для реализации Pool Thread

```

HttpServer server = new HttpServer(8080, true);
server.Start();

```

Для работы нужно указать количество потоков (рисунок 8). Класс Worker необходим для выполнения полезной нагрузки.

```
int MaxThreadsCount = Environment.ProcessorCount * 4;
ThreadPool.SetMaxThreads(MaxThreadsCount, MaxThreadsCount);
ThreadPool.SetMinThreads(2, 2);

public void Start() {
    _listener = new TcpListener(IPAddress.Any, _port);

    try
    {
        _listener.Start();
    }
    catch (Exception e)
    {
        throw;
    }

    while (true) {
        ThreadPool.QueueUserWorkItem(new WaitCallback(WorkerThread),
        Listener.AcceptTcpClient());
    }
}

private static void WorkerThread(Object listener)
{
    new Worker((TcpListener)listener);
}
```

**Рис. 8.** Реализация Pool Thread

#### **Преимущества реализации Pool Thread:**

- Эффективное использование ресурсов сервера;
- Возможность обрабатывать большое количество запросов при меньшем количестве потоков.

#### **Недостатки реализации Pool Thread:**

- Возможна ситуация, когда все потоки в пуле будут заняты обработкой запросов, и новые запросы будут ждать, пока один из потоков не станет доступен;
- Реализация может быть сложнее, чем в случае Thread per Request.

### **Практическая часть**

**Задание:** реализовать HTTP сервер на языке C# с использованием классов TcpListener и TcpClient, который будет отдавать по определенному порту статичную html страницу, расположенную на жестком диске по определенному пути. В качестве страницы, которую должен будет вывести браузер при обращении по адресу localhost:<номер порта по варианту>, следует создать html файл. Сам файл может быть любым. Желательно, чтобы в этом html файле была форма ввода и кнопка, это понадобится для выполнения 3 домашнего задания.

#### **Варианты:**

- Нечетные по списку – реализовать сервер на Thread per Request;
- Четные по списку – реализовать сервер на Pool Thread.

### **Номер порта по варианту:**

Номер порта выбрать из диапазона 49101-49600, где третий разряд будет определять номер группы студента, а первый и второй разряды – номер его варианта по списку. Например, для студента из группы 3 с номером варианта 21 номер порта должен быть 49321.

Для выполнения домашнего задания следует придерживаться следующего порядка выполнения:

- 1) Создать TcpListener с указанием IP и Порта;
- 2) Запустить TcpListener;
- 3) Принимать клиенту и обрабатывать в зависимости от варианта;
- 4) Распарсить поступивший запрос;
- 5) Обработать ошибки;
- 6) Создать тело HTTP ответа с необходимыми заголовками;
- 7) В тело ответа занести html файл из пути согласно запросу;
- 8) Передать клиенту;
- 9) Закрыть соединение с клиентом;
- 10) Предусмотреть в деструкторе остановку сервера.

### **Структура отчета по домашнему заданию**

Отчет является документом, свидетельствующим о выполнении студентом домашнего задания. Отчет должен включать:

- 1) Титульный лист с номером и названием домашнего задания, идентификатором группы, номером варианта, фамилией студента, датой сдачи отчета по домашнему заданию, местом для подписи преподавателя;
- 2) Введение, цель и задачи домашнего задания;
- 3) Описание работы программы;
- 4) Схему алгоритма;
- 5) Листинг программы (код должен быть самодокументирован, т.е. содержать комментарии);
- 6) Листинг отдаваемой веб-сервером html страницы;
- 7) Демонстрацию работы приложения;
- 8) Выводы по домашнему заданию.

### **Контрольные вопросы по домашнему заданию №2**

- 1) Дайте определение веб-сервера.
- 2) Дайте определение порта TCP-подключения.
- 3) Для чего используется класс TcpListener?
- 4) С помощью каких методов класса TcpListener можно управлять работой объекта этого класса?
- 5) Для чего используется класс TcpClient?

- 6) Как прочитать и записать данные в сетевом потоке?
- 7) На каком принципе основана реализация Thread per Request?
- 8) В чем преимущества и недостатки реализации Thread per Request?
- 9) На каком принципе основана реализация Pool Thread?
- 10) В чем преимущества и недостатки реализации Pool Thread?

## **Домашнее задание №3 «Веб-сервер (backend) с JSON форматом ответа на запрос»**

### **Краткая характеристика JSON формата данных**

Формат JSON (JavaScript Object Notation) – это легковесный формат обмена данными, который основан на языке JavaScript. Он представляет собой текстовый формат, который используется для передачи данных между приложениями.

Структура данных JSON представляет собой коллекцию пар "ключ-значение", где ключом может быть строка, а значением – любой из следующих типов данных: строка, число, логический тип, массив, объект или null. Данные в формате JSON могут быть легко интерпретированы и созданы как на стороне сервера, так и на стороне клиента.

Разметка JSON используется во многих сферах, включая веб-разработку, мобильную разработку, IoT и другие. Он может использоваться для передачи данных между клиентом и сервером, хранения данных в базах данных, обмена данными между различными сервисами и т.д.

Объекты JSON можно легко создавать и разбирать на стороне клиента и сервера с помощью различных библиотек и инструментов, доступных для многих языков программирования. Например, в C# есть библиотеки Newtonsoft.Json и System.Text.Json для работы с данными в формате JSON.

Пример JSON-объекта:

#### **Листинг 2.5 – Пример структуры данных JSON**

```
{
  "name": "John Smith",
  "age": 30,
  "isMarried": true,
  "hobbies": ["reading", "swimming", "traveling"],
  "address": {
    "street": "Main St",
    "city": "New York",
    "state": "NY",
    "zip": "10001"
  }
}
```

В этом примере (листинг 2.5) объект содержит имя, возраст, статус женат/замужем, список хобби и адрес. Содержимое адреса также является JSON объектом, который содержит улицу, город, регион и почтовый индекс.

### Отправка AJAX запроса на веб-сервер

Концепция AJAX (Asynchronous JavaScript and XML) – это технология, которая позволяет отправлять асинхронные запросы на сервер и обновлять часть страницы без необходимости перезагрузки всей страницы. AJAX запросы обычно отправляются с помощью JavaScript-кода.

Запросы AJAX могут использоваться для получения данных с сервера, отправки данных на сервер, обновления содержимого страницы и других задач.

Чтобы отправить AJAX запрос с html страницы, можно использовать JavaScript объект XMLHttpRequest. Пример отправки GET-запроса:

#### Листинг 2.6 – Отправка AJAX запроса с методом GET

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'url_to_api_endpoint');
xhr.onreadystatechange = function() {
    if(xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
        console.log(xhr.responseText); // выводим ответ сервера в консоль
    }
};
xhr.send();
```

В этом примере (листинг 2.6) мы создаем объект XMLHttpRequest, задаем метод запроса и URL-адрес сервера, на который отправляется запрос. Затем мы задаем обработчик события onreadystatechange, который вызывается каждый раз, когда состояние объекта XMLHttpRequest меняется. В случае изменения состояния на DONE с HTTP кодом статуса 200 мы выводим ответ сервера в консоль браузера.

В листинге 2.7 представлен пример кода html страницы, с которой можно отправить строку из поля ввода на сервер, получить ответ от сервера в формате JSON и вывести его содержимое на странице.

#### Листинг 2.7 – Код html страницы с AJAX запросом

```
<!DOCTYPE html>
<html>
  <head>
    <title>AJAX запрос на сервер</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <p>Результат:</p>
```



```

<p id="result"></p>
<form>
  <input type="text" id="inputString" name="inputString">
  <button type="button" onclick="sendRequest()">Посчитать</button>
</form>
<script>
function sendRequest() {
  const inputString = document.getElementById('inputString').value;
  const xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      const response = JSON.parse(this.responseText);
      document.getElementById("result").innerHTML = response.result;
    }
  };
  xhr.open("GET",   "/page.html?inputString="   +   encodeURIComponent(inputString));
  xhr.send();
}
</script>
</body>
</html>

```

В этом примере (листинг 2.7) мы назначили тегу текстового абзаца `<p>` идентификатор "result", по данному идентификатору происходит обращение к элементу страницы с помощью метода `getElementById` класса `document`, для изменения содержимого `html` элемента страницы используется поле `innerHTML`.

## Сериализация JSON

Сериализация данных – это процесс преобразования объекта или структуры данных в последовательность байтов или в другой формат, который может быть сохранен в файле, передан по сети или сохранен в другом месте. Такой формат может быть использован для восстановления данных в исходном виде в будущем.

Сериализация JSON может использоваться для многих целей. Она может использоваться для сохранения состояния приложения, чтобы восстановить его позже, для передачи данных между приложениями или между компьютерами через сеть, а также для хранения данных в базах данных. Она может также использоваться для передачи данных между различными языками программирования, или для обмена данными между сервером и клиентом веб-приложения.

Сериализация – это важный инструмент для работы с данными в современных приложениях, который позволяет эффективно сохранять, передавать и восстанавливать информацию в различных форматах.

Работу с форматом JSON в языке C# можно производить с помощью .NET библиотеки `Newtonsoft.Json` или с помощью `System.Text.Json`.

### *Библиотека Newtonsoft.Json*

Для работы с JSON на C# в среде разработки Visual Studio можно использовать стандартную библиотеку .NET - Newtonsoft.Json.

#### Листинг 2.8 – Создание класса с JSON структурой

```
using Newtonsoft.Json;

public class CalculationResult {
    [JsonProperty("result")]
    public string Result { get; set; }
}
```

#### Листинг 2.9 – Сериализация экземпляра класса в JSON формат

```
CalculationResult calculationResult = new CalculationResult();
calculationResult.Result = "Результат вычислений";
string jsonString = JsonConvert.SerializeObject(calculationResult);
```

#### Листинг 2.10 – Отправка JSON ответа на страницу в AJAX запросе

```
CalculationResult calculationResult = new CalculationResult();
calculationResult.Result = "Результат вычислений";
string jsonString = JsonConvert.SerializeObject(calculationResult);
Response.ContentType = "application/json";
Response.Write(jsonString);
```

В приведенном выше примере (листинг 2.10) мы создали экземпляр класса CalculationResult, заполнили его поля и преобразовали в JSON строку с помощью метода JsonConvert.SerializeObject. Затем мы указали, что тип ответа сервера должен быть "application/json" и записали JSON строку в ответ.

Ответ будет отправлен на страницу, код которой был предоставлен ранее в листинге 2.7. Для этого в AJAX запросе был указан путь к файлу на сервере "/page.html", который будет обрабатывать запрос, и при получении ответа обновит содержимое абзаца с помощью JavaScript.

### *Библиотека System.Text.Json*

Для создания и сериализации JSON на языке C# в среде разработки Visual Studio можно использовать стандартную библиотеку System.Text.Json.

#### Листинг 2.11 – Создание класса с JSON структурой

```
using System.Text.Json;
using System.Text.Json.Serialization;
```

```
class JsonData {  
    public string result { get; set; }  
}  
  
...  
  
var json = new JsonData{ result = "1234" };
```

#### Листинг 2.12 – Сериализация данных и перевод в массив байтов

```
string jsonString = JsonSerializer.Serialize(json);  
byte[] array = Encoding.ASCII.GetBytes(jsonString);
```

### Практическая часть

**Задание:** на основе домашнего задания №2 сделать обработку AJAX запроса веб-сервером. В поле ввода html страницы вводится арифметическое выражение и нажимается кнопка «Посчитать». Код JavaScript отправляет AJAX запрос на сервер. Программа на языке C# из домашнего задания №1 выполняет разбор строки с арифметическим выражением и вычисляет его результат. Далее веб-сервер отдает клиенту (в браузер на html страницу) результат в виде JSON структуры, результат вычислений из которой потом выводится пользователю внутри веб-страницы.

#### Варианты:

- Нечетные по списку – реализовать сериализацию JSON с помощью библиотеки Newtonsoft.Json;
- Четные по списку – реализовать сериализацию JSON с помощью библиотеки System.Text.Json.

Для выполнения домашнего задания следует придерживаться следующего порядка выполнения:

- 1) Создать фрагмент кода на JavaScript, который будет отправлять и обрабатывать запрос;
- 2) Реализовать обработку AJAX запроса, отделять её от запроса на получение страницы;
- 3) Полученные данные отправить в подпрограмму для расчета арифметического выражения;
- 4) Отправить JSON структуру с результатом вычислений пользователю на веб-страницу.

### Структура отчета по домашнему заданию

Отчет является документом, свидетельствующим о выполнении студентом домашнего задания. Отчет должен включать:

- 1) Титульный лист с номером и названием домашнего задания, идентификатором группы, номером варианта, фамилией студента, датой

сдачи отчета по домашнему заданию, местом для подписи преподавателя;

- 2) Введение, цель и задачи домашнего задания;
- 3) Описание работы программы;
- 4) Схему алгоритма;
- 5) Листинг программы (код должен быть самодокументирован, т.е. содержать комментарии);
- 6) Листинг html страницы с AJAX запросом;
- 7) Демонстрацию работы приложения;
- 8) Выводы по домашнему заданию.

### **Контрольные вопросы по домашнему заданию №3**

- 1) Дайте определение формата JSON, приведите пример объекта.
- 2) Какой тип данных является ключом в формате JSON?
- 3) Перечислите все типы данных, которые могут быть значениями в формате JSON.
- 4) Дайте определение AJAX запроса.
- 5) Как составить AJAX запрос и отправить его на веб-сервер с html страницы?
- 6) Можно ли отправить AJAX запрос с HTTP методом POST?
- 7) Как принять AJAX запрос и обработать его в программе на языке C#?
- 8) Дайте определение сериализации структуры данных JSON.
- 9) Какие библиотеки языка C# могут быть использованы для сериализации JSON данных.
- 10) Напишите код создания класса с JSON структурой на языке C#.

# ОГЛАВЛЕНИЕ

Предисловие .....	3
Модуль 1. Разработка консольного приложения.....	3
Домашнее задание №1 «Консольный калькулятор арифметических выражений» .....	4
Создание проекта консольного приложения в среде Visual Studio.....	4
Обработка строки с арифметическим выражением.....	7
Реализация автоматизированного тестирования .....	11
Практическая часть .....	12
Структура отчета по домашнему заданию .....	12
Контрольные вопросы по домашнему заданию №1 .....	13
Модуль 2. Разработка веб-сервера .....	13
Домашнее задание №2 «HTTP сервер, отдающий статичную html страницу».....	14
Краткая характеристика TCP-подключения .....	14
Классы, используемые для TCP-подключения .....	15
Создание HTTP сервера .....	17
Практическая часть .....	21
Структура отчета по домашнему заданию .....	22
Контрольные вопросы по домашнему заданию №2 .....	22
Домашнее задание №3 «Веб-сервер (backend) с JSON форматом ответа на запрос».....	23
Краткая характеристика JSON формата данных .....	23
Отправка AJAX запроса на веб-сервер.....	24
Сериализация JSON.....	25
Практическая часть .....	27
Структура отчета по домашнему заданию .....	27
Контрольные вопросы по домашнему заданию №3 .....	28

*Учебное издание*

**Минитаева** Алина Мажитовна

## **Создание веб-серверов и бэкенд разработка на языке C#**

Редактор **О.М. Королева**

Художник **Я.М. Асинкритова**

Корректор **Л.В. Забродина**

Компьютерная графика **О.В. Левашиной**

Компьютерная верстка **Г.Ю. Молотковой**

Оригинал-макет подготовлен  
в Издательстве МГТУ им. Н.Э. Баумана.

В оформлении использованы шрифты  
Студии Артемия Лебедева.

Подписано в печать **15.05.2023**. Формат **60×90/16**.  
Усл. печ. л. **3,49**. Тираж **150** экз. Изд. № **777-2023**. Заказ

Издательство МГТУ им. Н.Э. Баумана.  
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.  
[press@bmstu.ru](mailto:press@bmstu.ru)  
[www.baumanpress.ru](http://www.baumanpress.ru)

Отпечатано в типографии МГТУ им. Н.Э. Баумана.  
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.  
[baumanprint@gmail.com](mailto:baumanprint@gmail.com)