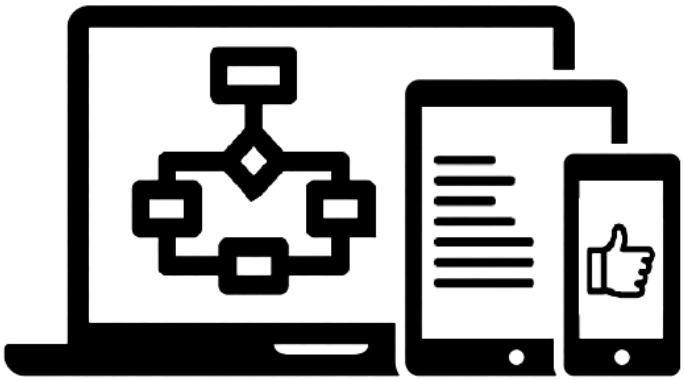
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Методические рекомендации к самостоятельной работе для студентов специальности 6-05-0612-03 «Системы управления информацией» заочной формы обучения

Часть 2



Могилев 2024

УДК 004.4
ББК 32.973
О75

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «6» декабря 2023 г., протокол № 5

Составитель ст. преподаватель А. И. Кашпар

Рецензент канд. техн. наук, доц. В. М. Ковальчук

Методические рекомендации к самостоятельной работе предназначены для студентов специальности 6-05-0612-03 «Системы управления информацией» заочной формы обучения.

Учебное издание

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Часть 2

Ответственный за выпуск	В. В. Кутузов
Корректор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 16 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Часть 2

6 Передача массивов в методы

Массивы могут передаваться в методы в качестве параметров, а также возвращаться из методов. Для возврата массива достаточно объявить массив как тип возврата.

Так как имя массива фактически является ссылкой, то он передается в метод по ссылке и, следовательно, все изменения элементов массива, являющегося формальным параметром, отразятся на элементах соответствующего массива, являющимся фактическим параметром. Размерность массивов в метод можно не передавать, а использовать свойства и методы класса Array:

– *Length* – возвращает количество элементов в массиве (используется для одномерных массивов);

– *GetLength*() – метод, возвращает количество элементов в заданном измерении массива (используется для многомерных массивов). Чтобы получить количество строк, нужно обратиться к методу *GetLength* с параметром 0. Чтобы получить количество столбцов – к методу *GetLength* с параметром 1.

Пример – Написать программу для замены элементов одномерного и двумерного массивов, большие заданного числа, нулем.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp6
{
  class Program
  {
    //метод для ввода одномерного массива
    //тип результата метода - массив (int[])
    static int[] Input1()
    {
      Console.WriteLine("Введите размерность одномерного массива: ");
      int n = int.Parse(Console.ReadLine());
      int[] a = new int[n];
      for(int i=0;i<n;i++)
      {
        Console.Write("Введите {0}-ый элемент массива: ", i+1);
        a[i] = int.Parse(Console.ReadLine());
      }
      return a;//возврат из метода ссылки на массив
     }
    //метод для ввода двумерного массива
    //тип результата метода - двумерный массив (int[,])
    static int[,] Input2()
```

```
    {
      Console.WriteLine("Введите размерности двумерного массива: ");
      int n = int.Parse(Console.ReadLine());
      int m = int.Parse(Console.ReadLine());
      int[,] a = new int[n,m];
      for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
        {
          Console.Write("Введите элемент {0}-ой строки {1}-го столбца:
",i+1,j+1);
          a[i,j] = int.Parse(Console.ReadLine());
        }
       return a;//возврат из метода ссылки на двумерный массив
    }
    //метод для вывода одномерного массива
    //параметр метода - ссылка на массив (int[])
    static void Output(int[] a)
    {
      //размерность одномерного массива возвращаются свойством Length
      for (int i = 0; i < a.Length; i++)
        Console.Write("{0}\t", a[i]);
      Console.WriteLine();
    }
    //перегруженный метод для вывода двумерного массива
    //Перегрузка метода - использование нескольких методов с одним и тем
    // же именем, но различными типами и количеством параметров.
    //Компилятор определяет, какой именно метод требуется вызвать,
    //по типу и количеству фактических параметров.
    //параметр метода - ссылка на массив (int[,])
    static void Output(int[,] a)
    {
      //размерности двумерного массива возвращаются методом GetLength()
      for (int i = 0; i < a.GetLength(0); i++)
      {
        for (int j = 0; j < a.GetLength(1); j++)
          Console.Write("{0}\t", a[i,j]);
        Console.WriteLine();
      }
    }
    //метод для изменения одномерного массива
    //параметры метода - ссылка на массив (int[]) и число k для сравнения
    //так как массив передается по ссылке, то при изменении массива a в
    //методе, изменится и исходный массив
    static void Change(int[] a, int k)
    {
      for (int i = 0; i < a.Length; i++)
        if (a[i] > k)
          a[i] = 0;
    }
    //перегруженный метод для изменения двумерного массива
    static void Change(int[,] a, int k)
```

```
    {
      for (int i = 0; i < a.GetLength(0); i++)
        for (int j = 0; j < a.GetLength(1); j++)
          if (a[i,j] > k)
            a[i,j] = 0;
    }
    static void Main(string[] args)
    {
      //объявление и вызов метода для ввода одномерного массива
      int[] Array1 = Input1();
      //объявление и вызов метода для ввода двумерного массива
      int[,] Array2 = Input2();
      Console.WriteLine("Исходные массивы");
      Output(Array1); //вызов метода для вывода одномерного массива
      Output(Array2); //вызов метода для вывода двумерного массива
      Console.WriteLine("Введите число для сравнения:");
      int M = int.Parse(Console.ReadLine());
      Change(Array1,M); //вызов метода для изменения одномерного массива
      Change(Array2,M); //вызов метода для изменения двумерного массива
      Console.WriteLine("Измененные массивы");
      Output(Array1); //вызов метода для вывода одномерного массива
      Output(Array2); //вызов метода для вывода двумерного массива
      Console.ReadKey();
    }
  }}
```

Задание для самостоятельного решения

1 Для одномерного и двумерного массивов заменить все элементы, меньшие заданного числа, этим числом.

2 Для одномерного и двумерного массивов заменить все элементы, попадающие в интервал [a, b], нулем.

3 Для одномерного и двумерного массивов все элементы, меньшие заданного числа, увеличить в 2 раза.

4 Для одномерного и двумерного массивов подсчитать среднее арифметическое элементов.

5 Для одномерного и двумерного массивов подсчитать среднее арифметическое отрицательных элементов.

6 Для одномерного и двумерного массивов подсчитать количество нечетных элементов.

7 Для одномерного и двумерного массивов подсчитать сумму элементов, попадающих в заданный интервал.

8 Для одномерного и двумерного массивов подсчитать сумму элементов, кратных 9.

9 Для одномерного и двумерного массивов подсчитать количество элементов, не попадающих в заданный интервал.

10 Для одномерного и двумерного массивов подсчитать сумму квадратов четных элементов.

7 Структуры

Структура – это составной объект, в который входят элементы любых типов. В отличие от массива, все элементы которого однотипны, структура может содержать элементы разных типов.

Структуры объявляются с помощью ключевого слова *struct*. Объявление структуры (структурного типа) имеет следующий формат:

```
[спецификаторы] struct имя_структуры
{
[спецификатор] тип_1 элемент_1;
[спецификатор] тип_2 элемент_2;
…
[спецификаторы] тип_n элемент_n;
//методы
} [ список_описателей ];
тело структуры
```

где *struct* – служебное слово;

имя_структуры – конкретное имя структуры;

спецификаторы – спецификаторы доступа, определяют статус доступа структурного типа.

Из модификаторов доступа допускаются только:

public – полная доступность;

internal – доступность только внутри содержащей сборки, это стандартная доступность для невложенных типов;

private – доступность только внутри содержащего типа. Это стандартная доступность для членов класса или структуры (только для вложенных структур).

Для использования доступа к полям снаружи структуры, члены структуры необходимо описывать с модификатором доступа *public*.

Элементы структуры называются *полями структуры* и могут иметь любой тип. Описание структуры определяет новый тип, имя которого можно использовать в дальнейшем наряду со стандартными типами, например:

```
// описание нового типа с именем Student
  struct Student
   {
       public string fio;
       public int num_zac;
       public double sr_bal;
   };
       // определение переменной типа Student и массива типа Student
       Student stud1;
       Student[] gr = new Student[30];
```

Для переменных одного и того же структурного типа определена операция присваивания, при этом происходит поэлементное копирование. Структуру можно передавать в функцию и возвращать в качестве значения функции.

Доступ к полям структуры выполняется с помощью операций выбора «.» (точка), например:

```
stud1.fio="Страусенко";
gr[8].sr_bal = 5;
```

Пример – Описать структуру с именем USER, содержащую следующие поля:

– NAME – фамилия, имя;

– AGE – возраст;

– HEIGHT – рост.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив SPISOK, состоящий из *n* элементов типа USER; записи должны быть упорядочены по росту;

– вывод на экран информации о людях заданного возраста; если такого нет, выдать на дисплей соответствующее сообщение.

Текст программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp6
{
  class Program
  { //описание нового типа структуры
    struct User
    {
      public string Name;
      public int Age;
      public double Height;
      //конструктор, вызываемый при использовании new
      public User(string N, int A, double H)
      {
        Name = N;
        Age = A;
        Height = H;
      }
      //метод для ввода информации User
      public void Input()
      {
        Console.WriteLine("Введите имя:");
        Name = Console.ReadLine();
        Console.WriteLine("Введите возраст:");
        Age = int.Parse(Console.ReadLine());
        Console.WriteLine("Введите рост:");
        Height = double.Parse(Console.ReadLine());
      }
```

```
      //метод для вывода информации User
      public void Output()
      {
        Console.WriteLine("Имя: {0}, возраст: {1}, рост: {2}", Name, Age,
Height);
      }
    }; //конец описания
    //метод сортировки
    static void sort(User[] a)
    {
      int i, k;
      User temp;
      for (k = 1; k < a.Length; k++)
        for (i = 0; i < a.Length - k; i++)
          if (a[i].Height> a[i + 1].Height)
          {
            temp = a[i];
            a[i] = a[i + 1];
            a[i + 1] = temp;
          }
    }
    //метод поиска
    static void poisk(User[] a)
    {
      bool f = false;
      Console.WriteLine("Введите возраст для поиска: ");
      int age=int.Parse(Console.ReadLine());
      for (int i = 0; i < a.Length; i++)
        if (a[i].Age == age)
        { a[i].Output();
          f = true;
        }
      if (f == false)
        Console.WriteLine("Нет в списке.");
    }
    static void Main()
      {
        Console.WriteLine("Введите количество записей");
        int n = int.Parse(Console.ReadLine());
        User[]SPISOK = new User[n];
      //вызывается конструктор для первого элемента
      SPISOK[0] = new User("Иванов И.", 37, 184.0);
      //ввод с клавиатуры остальных элементов
      for (int i = 1; i < n; i++)
      {
        Console.WriteLine("Введите {0}-ую запись:",i+1);
        SPISOK[i].Input();
      }
      //вывод всех элементов на экран
      Console.WriteLine("Исходный список:");
      for (int i =0; i < n; i++)
```

```
      {
        Console.Write((i + 1)+": ");
        SPISOK[i].Output();
      }
      sort(SPISOK);//вызов метода сортировки
      //вывод отсортированного массива на экран
      Console.WriteLine("Отсортированный список:");
      for (int i = 0; i < n; i++)
      {
        Console.Write((i + 1) + ": ");
        SPISOK[i].Output();
      }
      poisk(SPISOK);//вызов метода поиска
      Console.ReadKey();
      }
  }}}
```

Задание для самостоятельного решения

1 Описать структуру с именем STUDENT, содержащую следующие поля:

– NAME – фамилия и инициалы;

– GROUP – номер группы;

– SES – успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив STUD1, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по возрастанию содержимого поля GROUP;

– вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4,0;

– если таковых студентов нет, вывести соответствующее сообщение.

2 Описать структуру с именем STUDENT, содержащую следующие поля:

– NAME – фамилия и инициалы;

– GROUP – номер группы;

– SES – успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив STUD1, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по алфавиту;

– вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2;

– если таковых студентов нет, вывести соответствующее сообщение.

3 Описать структуру с именем AEROFLOT, содержащую следующие поля:

– NAZN – название пункта назначения рейса;

– NUMR – номер рейса;

– TIP – тип самолета.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив AIRPORT, состоящий из семи элементов типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;

– вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры;

– если таких рейсов нет, выдать на дисплей соответствующее сообщение.

4 Описать структуру с именем WORKER, содержащую следующие поля:

– NAME – фамилия и инициалы работника;

– POS – название занимаемой должности;

– YEAR – год поступления на работу.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив TABL, состоящий из десяти структур типа WORKER; записи должны быть размещены по алфавиту;

– вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры;

– если таких работников нет, вывести на дисплей соответствующие сообщение.

5 Описать структуру с именем TRAIN, содержащую следующие поля:

– NAZN – название пункта назначения;

– NUMR – номер поезда;

– TIME – время отправления.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив RASP, состоящий из восьми элементов типа TRAIN; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;

– вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени;

– если таких поездов нет, выдать на дисплей соответствующее сообщение.

6 Описать структуру с именем MARSH, содержащую следующие поля:

– BEGST – название начального пункта маршрута;

– TERM – название конечного пункта маршрута;

NUMER – номер маршрута.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив TRAFIC, состоящий из восьми элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;

– вывод на экран информации о маршруте, номер которого введен с клавиатуры;

– если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

– выдать на дисплей соответствующее сообщение.

7 Описать структуру с именем NOTE, содержащую следующие поля:

– NAME – фамилия, имя;

– TEL – номер телефона;

– BDAY – день рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив BLOCKNOTE, состоящий их восьми элементов типа NOTE; записи должны быть размещены по алфавиту;

– вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры;

– если таких нет, выдать на дисплей соответствующее сообщение.

8 Описать структуру с именем NOTE, содержащую следующие поля:

– NAME – фамилия, имя;

– TEL – номер телефона;

– BDAY – день рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив BLOCKNOTE, состоящий их восьми элементов типа NOTE; записи должны быть упорядочены по датам дней рождения;

– вывод на экран информации о человеке, номер телефона которого введен с клавиатуры;

– если такого нет, выдать на дисплей соответствующее сообщение.

9 Описать структуру с именем ZNAK, содержащую следующие поля:

– NAME – фамилия, имя;

– ZODIAC – знак зодиака;

– BDAY – день рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив BOOK, состоящий из восьми элементов типа ZNAK; записи должны быть упорядочены по датам дней рождения;

– вывод на экран информации о человеке, чья фамилия введена с клавиатуры;

– если такого нет, выдать на дисплей соответствующее сообщение.

10 Описать структуру с именем PRICE, содержащую следующие поля:

– TOVAR – название товара;

– MAG – название магазина, в котором продается товар;

– STOIM – стоимость товара в рублях.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив SPISOK, состоящий из восьми элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям товаров;

– вывод на экран информации о товаре, название которого введено с клавиатуры;

– если такого товара нет, выдать на дисплей соответствующее сообщение.

11 Описать структуру с именем ORDER, содержащую следующие поля:

– PLAT – расчетный счет плательщика;

– POL – расчетный счет получателя;

– SUMMA – перечисляемая сумма в рублях.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив SPISOK, состоящий из восьми элементов типа ORDER; записи должны быть размещены в алфавитном порядке по расчетным счетам плательщиков;

– вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры;

– если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.

8 Символы и строки

Обработка текстовой информации является одной из самых распространенных задач современного программировании. C# предоставляет для ее решения широкий набор средств: символы char, неизменяемые строки string и изменяемые строки StringBuider.

Символы char. Символьный тип char предназначен для хранения символа в кодировке Unicode. Символьная константа – это символ, заключенный в апострофы.

Например:

'C' – обычный символ,

'\n' – управляющая последовательность.

Символьный тип относится к встроенным типам данных C# и соответствует стандартному классу Char библиотеки .Net из пространства имен System. В этом классе определены статические методы, позволяющие задавать вид и категорию символа, а также преобразовывать символ в верхний или нижний регистр, в число. Рассмотрим основные методы (таблица 8.1).

Таблица 8.1

Метод	Описание
GetNumericValue	Возвращает числовое значение символа, если он является цифрой, и –1 в противном случае
GetUnicodeCategory	Возвращает категорию Unicode-символа. В Unicode символы разделены на категории, например цифры (DecimalDigitNumber), римские цифры (LetterNumber), разделители строк (LineSeparator), буквы в нижнем регистре (LowercaseLetter) и т. д.
IsControl	Возвращает true, если символ является управляющим
IsDigit	Возвращает true, если символ является десятичной цифрой
IsLetter	Возвращает true, если символ является буквой
IsLetterOrDigit	Возвращает true, если символ является буквой или десятичной цифрой
IsLower	Возвращает true, если символ задан в нижнем регистре
IsNumber	Возвращает true, если символ является числом (десятичным или шестнадцатеричным)
IsPunctuation	Возвращает true, если символ является знаком препинания
IsSeparator	Возвращает true, если символ является разделителем
IsUpper	Возвращает true, если символ задан в нижнем регистре

Окончание таблицы 8.1

Метод	Описание
IsWhiteSpace	Возвращает true, если символ является пробельным (пробел, перевод строки, возврат каретки)
Parse	Преобразует строку в символ (строка должна состоять из одного символа)
ToLower	Преобразует символ в нижний регистр
ToUpper	Преобразует символ в верхний регистр

В следующем примере рассмотрим применение данных методов:

```
static void Main()
   {
    try
    {
     char b = 'B', c = '\x64', d = '\uffff';
     Console.WriteLine("{0}, {1}, {2}", b, c, d);
     Console.WriteLine("{0}, {1}, {2}", char.ToLower(b), char.ToUpper(c),
char.GetNumericValue(d));
     char a;
     do     //цикл выполняется до тех пор, пока не ввели символ e
     {
      Console.WriteLine("Введите символ: ");
      a = char.Parse(Console.ReadLine());
      Console.WriteLine("Введен символ {0}, его код {1}, его категория {2}", a,
(int)a, char.GetUnicodeCategory(a));
      if (char.IsLetter(a)) Console.WriteLine("Буква");
      if (char.IsUpper(a)) Console.WriteLine("Верхний регистр");
      if (char.IsLower(a)) Console.WriteLine("Нижний регистр");
      if (char.IsControl(a)) Console.WriteLine("Управляющий символ");
      if (char.IsNumber(a)) Console.WriteLine("Число");
      if (char.IsPunctuation(a)) Console.WriteLine("Разделитель");
     } while (a != 'e');
    }
   catch
    {
       Console.WriteLine("Возникло исключение");
  }  }
```

Неизменяемые строки string. Тип string, предназначенный для работы со стоками символов в кодировке Unicode, является встроенным типом C#. Ему соответствует базовый тип класса System.String библиотеки .Net. Каждый объект string - это неизменяемая последовательность символов Unicode, т. е. методы, предназначенные для изменения строк, возвращают измененные копии, исходные же строки остаются неизменными.

Создать строку можно несколькими способами:

1) string s; // инициализация отложена

2) string s="кол около колокола"; //инициализация строковым литералом

3) string s=@"Привет! //символ @ сообщает конструктору string, что сегодня хорошая погода!!! " // строку нужно воспринимать буквально,
// даже если она занимает несколько строк

```
4) string s=new string ('ы', 20); //конструктор создает строку из 20 пробелов
5) int x = 12344556;              //инициализировали целочисленную
                                  //переменную
string s = x.ToString();          //преобразовали ее к типу string
6) char [] a={'a', 'b', 'c', 'd', 'e'};  //создали массив символов
string v=new string (a);          // создание строки из массива символов
7) char [] a={'a', 'b', 'c', 'd', 'e'};
string v=new string (a, 0, 2);    //создание строки из части массива
                                  //символов, при этом:  0 показывает с
                                  // какого символа, 2 – сколько символов
                                  //использовать для инициализации
```

Класс String обладает богатым набором статических и экземплярных методов, представленных в таблице 8.2. Вызов статических методов происходит через обращение к имени класса, например, String.Concat(str1, str2), в остальных случаях через обращение к экземплярам класса, например, str.ToLower().

Таблица 8.2

Название	Вид	Описание
Compare	Статический метод	Сравнение двух строк в лексикографическом (алфавитном) порядке
CompareTo	Метод	Сравнение текущего экземпляра строки с другой строкой
Concat	Статический метод	Слияние произвольного числа строк
Copy	Статический метод	Создание копии строки
Empty	Статическое поле	Открытое статическое поле, представляющее пустую строку
Format	Статический метод	Форматирование строки в соответствии с заданным форматом
IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny	Экземплярные методы	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора в данную строку
Insert	Экземплярный метод	Вставка подстроки в заданную позицию
Join	Статический метод	Слияние массива строк в единую строку. Между элементами массива вставляются разделители
Length	Свойство	Возвращает длину строки
PadLeft, PadRigth	Экземплярные методы	Выравнивают строки по левому или правому краю путем вставки нужного числа пробелов в начале или в конце строки
Remove	Экземплярный метод	Удаление подстроки из заданной позиции
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки или символа новыми подстрокой или символом
Split	Экземплярный метод	Разделяет строку на элементы, используя разные разделители. Результаты помещаются в массив строк

Окончание таблицы 8.2

Название	Вид	Описание
StartWith, EndWith	Экземплярные методы	Возвращают true или false в зависимости от того, начинается или заканчивается строка заданной подстрокой
Substring	Экземплярный метод	Выделение подстроки, начиная с заданной позиции
ToCharArray	Экземплярный метод	Преобразует строку в массив символов
ToLower, ToUpper	Экземплярные методы	Преобразование строки к нижнему или верхнему регистру
Trim, TrimStart, TrimEnd	Экземплярные методы	Удаление пробелов в начале и конце строки или только с одного ее конца

На примере рассмотрим использование данных свойств и методов.

```
static void Main()
   {
string str1 ="Первая строка";
string str2 = string.Copy(str1);
string str3 = "Вторая строка";
string str4 = "ВТОРАЯ строка";
string strUp, strLow;
int result, idx;
Console.WriteLine("str1: " + str1);
Console.WriteLine("Длина строки str1: " +str1.Length);

// Создаем прописную и строчную версии строки str1.
strLow = str1.ToLower();
strUp = str1.ToUpper();
Console.WriteLine("Строчная версия строки str1: " +strLow);
Console.WriteLine("Прописная версия строки str1: " +strUp);
Console.WriteLine();

// Сравниваем строки,
result = str1.CompareTo(str3);
if (result == 0) Console.WriteLine("str1 и str3 равны.");
  else if (result < 0) Console.WriteLine("str1 меньше, чем str3");
          else Console.WriteLine("str1 больше, чем str3");
Console.WriteLine();

//сравниваем строки без учета регистра
result = String.Compare(str3,str4,true);
if (result == 0) Console.WriteLine("str3 и str4 равны без учета
регистра.");
   else Console.WriteLine("str3 и str4 не равны без учета регистра.");
Console.WriteLine();

//сравниваем части строк
result = String.Compare(str1, 4, str2, 4, 2);
if (result == 0)  Console.WriteLine("часть str1 и str2 равны");
  else  Console.WriteLine("часть str1 и str2 не равны");
```

```
Console.WriteLine();

// Поиск строк.
idx = str2.IndexOf("строка");
Console.WriteLine("Индекс первого вхождения подстроки сторка: " + idx);
idx = str2.LastIndexOf("о");
Console.WriteLine("Индекс последнего вхождения символа о: " + idx);

//конкатенация
string str=String.Concat(str1, str2, str3, str4);
Console.WriteLine(str);

//удаление подстроки
str=str.Remove(0,str1.Length);
Console.WriteLine(str);

//замена подстроки "строка" на пустую подстроку
str=str.Replace("строка","");
Console.WriteLine(str);
}
```

Очень важными методами обработки строк, являются методы разделения строки на элементы Split и слияние массива строк в единую строку Join.

```
static void Main()
{
string poems = "тучки небесные вечные странники";
char[] div = { ' '}; //создаем массив разделителей
// Разбиваем строку на части,
string[] parts = poems.Split(div);
Console.WriteLine("Результат разбиения строки на части: ");
for (int i = 0; i < parts.Length; i++)
 Console.WriteLine(parts[i]);
// Теперь собираем эти части в одну строку, в качестве разделителя
используем символ |
string whole = String.Join(" | ", parts);
Console.WriteLine("Результат сборки: ");
Console.WriteLine(whole);
}
```

При работе с объектами класса string нужно учитывать их свойство неизменяемости, т. е. тот факт, что методы изменяют не сами строки, а их копии.

Изменяемые строки. Чтобы создать строку, которую можно изменять, в C# предусмотрен класс StringBuilder, определенный в пространстве имен System.Text. Объекты этого класса всегда объявляются с явным вызовом конструктора класса (через операцию new) . Примеры создания изменяемых строк:

```
//создание пустой строки, размер по умолчанию 16 символов
StringBuilder a =new StringBuilder();
//инициализация строки и выделение необходимой памяти
StringBuilder b = new StringBuilder("abcd");
```

```
//создание пустой строки и выделение памяти под 100 символов
StringBuilder c = new StringBuilder(100);
//инициализация строки и выделение памяти под 100 символов
StringBuilder d = new StringBuilder("abcd", 100);
//инициализация подстрокой "bcd", и выделение памяти под 100 символов
StringBuilder d = new StringBuilder("abcd", 1, 3,100);
```

Основные элементы класса приведены в таблице 8.3.

Таблица 8.3

Название	Вид	Описание
Append	Экземплярный метод	Добавление данных в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки string
AppendFormat	Экземплярный метод	Добавление форматированной строки в конец строки
Capacity	Свойство	Получение и установка емкости буфера. Если устанавливаемое значение меньше текущей длины строки или больше максимального, то генерируется исключение ArgumentOutOfRangeException
Insert	Экземплярный метод	Вставка подстроки в заданную позицию
Length	Изменяемое свойство	Возвращает длину строки. Присвоение ему значения 0 сбрасывает содержимое и очищает строку
MaxCapacity	Неизменное свойство	Возвращает наибольшее количество символов, которое может быть размещено в строке
Remove	Экземплярный метод	Удаление подстроки из заданной позиции
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
ToString	Экземплярный метод	Преобразование в строку типа string
Chars	Изменяемое свойство	Возвращает из массива или устанавливает в массиве символ с заданным индексом. Вместо него можно пользоваться квадратными скобками []
Equals	Экземплярный метод	Возвращает true, только если объекты имеют одну и ту же длину и состоят из одних и тех же символов
CopyTo	Экземплярный метод	Копирует подмножество символов строки в массив char

Методы класса StringBuilder менее развиты, чем методы класса String, но они позволяют более эффективно использовать память за счет работы с изменяемыми строками. Рассмотрим примеры использования данных методов.

```
static void Main()
{
      try
      {
      StringBuilder str=new StringBuilder("Площадь");
```

```
      PrintString(str);
      str.Append(" треугольника равна");
      PrintString(str);
      str.AppendFormat(" {0:f2} см ", 123.456);
      PrintString(str);
      str.Insert(8, "данного ");
      PrintString(str);
      str.Remove(7, 21);
      PrintString(str);
      str.Replace("a", "o");
      PrintString(str);
      StringBuilder str1=new StringBuilder(Console.ReadLine());
      StringBuilder str2=new StringBuilder(Console.ReadLine());
      Console.WriteLine(str1.Equals(str2));
      }
      catch
      {
            Console.WriteLine("Вознико исключение");
      }
}
      static void PrintString(StringBuilder a)
      {
        Console.WriteLine("Строка: "+a);
        Console.WriteLine("Текущая длина строки " +a.Length);
        Console.WriteLine("Объем буфера "+a.Capacity);
        Console.WriteLine("Максимальный объем буфера "+a.MaxCapacity);
        Console.WriteLine();
      }
```

С изменяемой строкой можно работать не только как с объектом, но как с массивом символов:

```
static void Main()
      {
      StringBuilder a = new StringBuilder("2*3=3*2");
      Console.WriteLine(a);
      int k=0;
      for (int i = 0; i < a.Length; ++i )
      if (char.IsDigit(a[i])) k+=int.Parse(a[i].ToString());
      Console.WriteLine(k);
      }
```

На практике часто комбинируют работу с изменяемыми и неизменяемыми строками. Однако если необходимо изменять строку, то в этом случае используют StringBuilder.

Пример – Дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания. Вывести все слова сообщения, которые начинаются и заканчиваются на одну и ту же букву.

```
static void Main()
      {
      Console.WriteLine("Введите строку: ");
      StringBuilder a = new StringBuilder(Console.ReadLine());
      Console.WriteLine("Исходная строка: "+a);
      for (int i=0; i<a.Length;)
            if (char.IsPunctuation(a[i])) a.Remove(i,1);
             else ++i;
      string str=a.ToString();
      string []s=str.Split(' ');
       Console.WriteLine("Искомые слова: ");
      for (int i=0; i<s.Length; ++i)
      if (s[i][0]==s[i][s.Length-1]) Console.WriteLine(s[i]);
      }
```

Задание для самостоятельного решения

Дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания.

1 Вывести только те слова сообщения, которые содержат не более чем *n* букв.

2 Вывести только те слова сообщения, которые начинаются с прописной буквы.

3 Вывести только те слова сообщения, которые содержат хотя бы одну цифру.

4 Удалить из сообщения все слова, которые заканчиваются на заданный символ.

5 Удалить из сообщения все слова, содержащие данный символ (без учета регистра).

6 Удалить из сообщения все однобуквенные слова (вместе с лишними пробелами).

7 Подсчитать сколько раз заданное слово встречается в сообщении.

8 Найти все самые длинные слова сообщения.

9 По правилу расстановки знаков препинания перед каждым знаком препинания пробел отсутствует, а после него обязательно стоит пробел. Учитывая данное правило, проверьте текст на правильность расстановки знаков препинания и, если необходимо, внесите в текст изменения.

10 Вывести только те слова, которые встречаются в тексте ровно один раз.

9 Потоки и файлы

Если необходимо создать входной или выходной поток, связанный с локальным файлом, содержащим двоичные данные, следует воспользоваться классами BinaryWriter и BinaryReader из пространства имен System.IO. Что же касается чтения из файла или записи в файл текстовых данных, то для решения этой задачи обычно используются классы StreamReader и StreamWriter. Для работы с файлами необходимо подключить библиотеку:

```
using System.IO; //для работы с потоками
```

В любом случае, перед тем как читать данные из файла или записывать их в файл, необходимо выполнить операцию *открытия* потока, связанного с файлом. Когда работа с файлом окончена, соответствующий поток необходимо *закрыть* явным образом.

Для работы с двоичными файлами программа должна вначале создать поток класса FileStream, воспользовавшись соответствующим конструктором, например:

```
FileStream fs = new FileStream("mayfile.dat", FileMode.CreateNew);
```

В качестве первого параметра конструктору необходимо передать полный путь к файлу или имя файла, а в качестве второго – режим открытия потока (таблица 9.1).

Таблица 9.1

Режим	Описание
Append	Если файл существует, он открывается. Текущая позиция устанавливается на конец файла. Если указанного файла нет, то он создается. Этот режим можно использовать только совместно с режимом доступа FileAccess. Write. При попытке чтения из файла, открытого подобным образом, возникает исключение ArgumentException
Create	ОС должна создать новый файл. Если указанный файл уже существует, он будет перезаписан
CreateNew	ОС должна создать новый файл. Если указанный файл уже существует, возникнет исключение IOException
Open	Требуется открыть существующий файл. Необходим доступ FilelOPerraissionAccess.Read. Если требуемый файл не найден, возникает исключение System.FileNotFoundException
OpenOrCreate	Если указанный файл существует, он должен быть открыт. В противном случае – ОС должна создать и открыть указанный файл
Truncate	Требуется открыть существующий файл. После открытия файл обрезается до нулевой длины, при этом все ранее хранившиеся в нем данные пропадают. При попытке чтения из файла, открытого подобным образом, возникает исключение

В зависимости от того, для какой цели создается файл, можно выбрать тот или иной режим открытия потока. Например, если поток, связанный с файлом, открывается только для чтения, выбирайте режим FileMode.Open. В этом случае будет открыт существующий файл. Если же файл открывается для записи, то можно либо открыть существующий файл, перезаписав его содержимое или дописав в него новые данные, а также создать новый файл.

В классе FileStream существует несколько конструкторов, позволяющих определить не только путь к файлу, но и режим его открытия, разрешенный доступ к файлу и режим совместного использования файла:

```
FileStream fs = new FileStream(“file.dat”, FileMode.Open, FileAccess.Read);
```

Остановимся на режиме доступа к файлу, передаваемом приведенным выше конструкторам через третий параметр (таблица 9.2). Режим доступа задается как статическая константа класса FileAccess.

Таблица 9.2

Режим	Тип доступа
Read	Доступ только на чтение
ReadWrite	Доступ на чтение и запись
Write	Доступ на запись

При необходимости программа может задать режим совместного использования файла, когда для одного и того же файла одновременно создается несколько потоков. Этот режим задается при помощи статических констант класса FileShare.

Двоичные потоки BinaryWriter и BinaryReader. Двоичные файлы хранят данные в том же виде, в котором они представлены в оперативной памяти, т. е. во внутреннем представлении. Двоичные файлы не применяются для просмотра человеком, они используются только для программной обработки.

Выходной поток BinaryWriter поддерживает произвольный доступ, т. е. имеется возможность выполнять запись в произвольную позицию двоичного файла.

Поток создается на базе файла, воспользовавшись для этого классом FileStream. Далее на базе полученного таким образом потока необходимо создать потоки классов BinaryWriter и BinaryReader, предназначенные соответственно для записи в файл и чтения из файла двоичных данных:

```
BinaryWriter bw = new BinaryWriter(fs);
BinaryReader br = new BinaryReader(fs);
```

После того как программа завершила работу с потоками, она должна их закрыть. Для закрытия потоков используйте метод Close. Закрывайте потоки в порядке, обратном открытию. Если вначале открыли поток FileStream, а затем создали на его базе потоки класса BinaryWriter и BinaryReader, то закрывать потоки нужно в обратном порядке: вначале закройте потоки BinaryWriter и BinaryReader, а затем поток FileStream:

```
FileStream fs   = new FileStream("myfile.datн, FileMode.CreateNew);
BinaryWriter bw = new BinaryWriter(fs);
BinaryReader br = new BinaryReader(fs);
// Работа с потоками
bw.Close();
br.Close();
fs.Close();
```

Для записи двоичных данных в поток BinaryWriter предусмотрено несколько перегруженных методов Write.

Способ применения большинства перечисленных методов интуитивно ясен: для записи данных того или иного типа в файл нужно передать ссылку на экземпляр данных в качестве параметра методу Write. В зависимости от типа данных компилятор выберет нужную перегруженную версию этого метода.

При записи в двоичный поток текстовой строки string она будет предваряться префиксом, содержащим значение длины строки. Строка будет записана в кодировке ASCII.

Последние два перегруженных метода Write позволяют записать в поток массив соответственно двоичных байтов и символов UNICODE. Второй параметр метода задает начальный индекс блока данных в массиве, а третий – размер блока (соответственно в байтах и символах).

Чтобы прочитать данные из потока BinaryReader, нужно использовать один из методов, специально предусмотренных для этого в классе BinaryReader. Приведем краткий список этих методов в таблице 9.3.

Таблица 9.3

Метод	Тип данных, считываемых из потока
ReadBoolean	Один объект типа Boolean
ReadByte	1 байт данных
ReadBytes	Чтение заданного количества байтов данных
ReadChar	Один текстовый символ
ReadChars	Чтение заданного количества текстовых символов
ReadDecimal	Десятичное значение размером 16 байт
ReadSingle	Значение с плавающей десятичной точкой размером 4 байта
ReadDouble	Значение с плавающей десятичной точкой размером 8 байт
ReadSByte	Число со знаком размером 1 байт
Readlntl6	Число со знаком размером 2 байта
Readlnt32	Число со знаком размером 4 байта
Readlnt64	Число со знаком размером 8 байт
ReadUIntl6	Число без знака размером 2 байта
ReadUInt32	Число без знака размером 4 байта
ReadUInt64	Число без знака размером 8 байт
ReadString	Текстовая строка с префиксом длины

Двоичные файлы являются файлами с произвольным доступом, при этом нумерация элементов в двоичном файле ведется с нуля. Произвольный доступ обеспечивает метод Seek. Рассмотрим его синтаксис:

```
Seek(long newPos, SeekOrigin pos)
```

где параметр newPos определяет новую позицию внутреннего указателя файла в байтах относительно исходной позиции указателя, которая определяется параметром pos. В свою очередь, параметр pos должен быть задан одним из значений перечисления SeekOrigin (таблица 9.4).

После вызова метода Seek следующие операции чтения или записи будут выполняться с новой позиции внутреннего указателя файла.

Поток BinaryReader не имеет метода Seek, однако используя возможности

потока FileStream можно организовать произвольный доступ при чтении двоичных файлов.

Таблица 9.4

Значение	Описание
SeekOrigin.Begin	Поиск от начала файла
SeekOrigin.Current	Поиск от текущей позиции указателя
SeekOrigin.End	Поиск от конца файла

Пример 1 – Создать файл и записать в него вещественные числа из диапазона от *a* до *b* с шагом *h*. Вывести на экран все компоненты файла с нечетными порядковыми номерами.

```
using System;
using System.Text;
using System.IO;
namespace MyProgram
{
    class Program
    {
        static void Main()
        {
            Console.Write("a= ");
            double a = double.Parse(Console.ReadLine());
            Console.Write("b= ");
            double b = double.Parse(Console.ReadLine());
            Console.Write("h= ");
            double h = double.Parse(Console.ReadLine());
            //Записываем в файл t.dat вещественные числа из заданного диапазона
            FileStream f = new FileStream("t.dat", FileMode.Open);
            BinaryWriter fOut = new BinaryWriter(f);
            for (double i = a; i <= b; i += h)
            {
                fOut.Write(i);
            }
            fOut.Close();
            //Объекты f и fIn связаны с одним и тем же файлом
            f = new FileStream("t.dat", FileMode.Open);
            BinaryReader fIn = new BinaryReader(f);
            long m = f.Length; //определяем количество байт в потоке
            //Читаем данные из файла t.dat начиная с элемента с номером 1, т.е с 8 байта,
            //перемещая внутренний указатель на 16 байт, т.е. на два вещественных числа
            for (long i = 8; i < m; i += 16)
            {//перемещение маркера на i-ый байт от начала файла
                f.Seek(i, SeekOrigin.Begin);
                a = fIn.ReadDouble();
                Console.Write("{0:f2} ", a);
            }
            fIn.Close();
            f.Close();
        }
    }  }
```

Задание 1 для самостоятельного решения

Работа с двоичными файлами.

1 Создать файл и записать в него степени числа 3. Вывести на экран все компоненты файла с четным порядковым номером.

2 Создать файл и записать в него обратные натуральные числа $1, \frac{1}{2}, \ldots, \frac{1}{n}$. Вывести на экран все компоненты файла с порядковым номером, кратным 3.

3 Создать файл и записать в него *n* первых членов последовательности Фибоначчи. Вывести на экран все компоненты файла с порядковым номером, не кратным 3.

4 Дана последовательность из *n* целых чисел. Создать файл и записать в него все четные числа последовательности. Вывести содержимое файла на экран.

5 Дана последовательность из *n* целых чисел. Создать файл и записать в него все отрицательные числа последовательности. Вывести содержимое файла на экран.

6 Дана последовательность из *n* целых чисел. Создать файл и записать в него числа последовательности, попадающие в заданный интервал. Вывести содержимое файла на экран.

7 Дана последовательность из *n* целых чисел. Создать файл и записать в него числа последовательности, не кратные заданному числу. Вывести содержимое файла на экран.

8 Дана последовательность из *n* вещественных чисел. Записать все эти числа в файл. Вывести на экран все компоненты, не попадающие в данный диапазон.

9 Дана последовательность из *n* вещественных чисел. Записать все эти числа в файл. Вывести на экран все компоненты файла с нечетными номерами, большие заданного числа.

10 Дана последовательность из *n* вещественных чисел. Записать все эти числа в файл. Вывести на экран все компоненты файла с четными номерами, меньшие заданного числа.

Работа с текстовыми файлами. Класс StreamWriter предназначен для организации выходного символьного потока. В нем определено несколько конструкторов. Один из них записывается следующим образом:

```
StreamWriter(Stream stream);
```

где параметр stream определяет имя уже открытого байтового потока.

Например, создать экземпляр класса StreamWriter, можно следующим образом:

```
StreamWriter fileOut=new StreamWriter(new FileStream("text.txt", FileMode.Create, FileAccess.Write));
```

Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для вывода, и исключение типа ArgumentNullException, если он (поток) имеет null-значение.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

```
StreamWriter(string name, bool appendFlag);
```

где параметр name определяет имя открываемого файла; параметр appendFlag может принимать значение true – если нужно добавлять данные в конец файла, или false – если файл необходимо перезаписать.

Например:

```
StreamWriter fileOut=new StreamWriter("c:\\temp\\t.txt");
StreamWriter fileOut=new StreamWriter("t.txt", true);
```

Теперь для записи данных в поток fileOut можно обратиться к методам Write, WriteLine. Это можно сделать следующим образом:

```
fileOut.WriteLine("test");
```

В данном случае в конец файла t.txt будет дописано слово test.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
StreamReader(Stream stream);
```

где параметр stream определяет имя уже открытого байтового потока.

Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для ввода.

Например, создать экземпляр класса StreamWriter можно следующим образом:

```
StreamReader fileIn = new StreamReader(new FileStream("text.txt", File-
Mode.Open, FileAccess.Read));
```

Как и в случае с классом StreamWriter у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
StreamReader (string name);
```

где параметр name определяет имя открываемого файла.

Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn=new StreamReader ("c:\\temp\\t.txt");
StreamReader fileIn=new StreamReader ("c:\\temp\\t.txt", Encod-
ing.GetEncoding(1251));
```

Параметр Encoding.GetEncoding(1251) говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode. Encoding.GetEncoding(1251) реализован в пространстве имен System.Text.

Для чтения данных из потока fileIn можно воспользоваться методом ReadLine. При этом если будет достигнут конец файла, то метод ReadLine вернет значение null. Например, фрагмент программы считывает текстовые строки из файла, вызывая в цикле метод ReadLine:

```
string str;
while((str = sr.ReadLine())!=null);
Console.WriteLine(str);
```

Пример 2 – Дан текстовый файл. Найти количество строк, которые начинаются с данной буквы.

```
using System;
using System.Text;
using System.IO;
using System.Text.RegularExpressions;
namespace MyProgram
{
class Program
{
static void Main()
{
      Console.Write("Введите заданную букву: ");
      char a=char.Parse(Console.ReadLine());
      StreamReader fileIn = new StreamReader("text.txt");
      string text=fileIn.ReadToEnd(); //считываем из файла весь текст
      fileIn.Close();
      int k=0;
      //разбиваем текст на слова используя регулярные выражения
      string []newText=Regex.Split(text,"[ ,.:;]+");
      //подсчитываем количество слов, начинающихся на заданную букву
      foreach( string b in newText)
            if (b[0]==a)++k;
                  Console.WriteLine("k= "+k);
}}}
```

Задание 2 для самостоятельного решения

Работа с текстовыми файлами.

1 Дан текстовый файл. Найти количество строк, которые начинаются и заканчиваются одной буквой.

2 Дан текстовый файл. Найти самую длинную строку и ее длину.

3 Дан текстовый файл. Найти номер самой короткой строки.

4 Дан текстовый файл. Выяснить, имеется ли в нем строка, которая начинается с данной буквы. Если да, то напечатать ее.

5 Дан текстовый файл. Напечатать первый символ каждой строки.

6 Дан текстовый файл. Напечатать все строки, в которых имеется хотя бы один пробел.

7 Дан текстовый файл. Напечатать все строки, длина которых меньше заданного числа.

8 Дан текстовый файл. Переписать в новый файл все его строки, вставив в конец каждой строки ее номер.

9 Дан текстовый файл. Переписать в новый файл все его строки, вставив в конец каждой строки количество символов в ней.

10 Дан текстовый файл. Переписать в новый файл все его строки, длина которых больше заданного числа.

10 Создание Windows-приложений

Процесс создания Windows-приложения состоит из двух основных этапов:

1) визуальное проектирование, т. е. задание внешнего облика приложения;

2) определение поведения приложения путем написания процедур обработки событий.

Визуальное проектирование заключается в помещении на форму компонентов (элементов управления) и задании их свойств и свойств самой формы.

Форма – это экранный объект, обеспечивающий функциональность программы. Формы могут иметь стандартный вид, такой, например, как у программы Microsoft Word, или причудливый – как у Winamp.

Для создания проекта Windows-приложения: загружаем Visual Studio, далее выбираем Файл (File) →Создать (New) →Проект (Project), далее выбираем пункт Visual C# и отмечаем Приложение Windows Forms (Windows Forms Application), даем имя проекта в поле Имя: (Name:), местоположение в поле Расположение: (Location:) и жмем Ok.

После этого Visual Studio откроет наш проект с созданными по умолчанию файлами (рисунок 10.1).

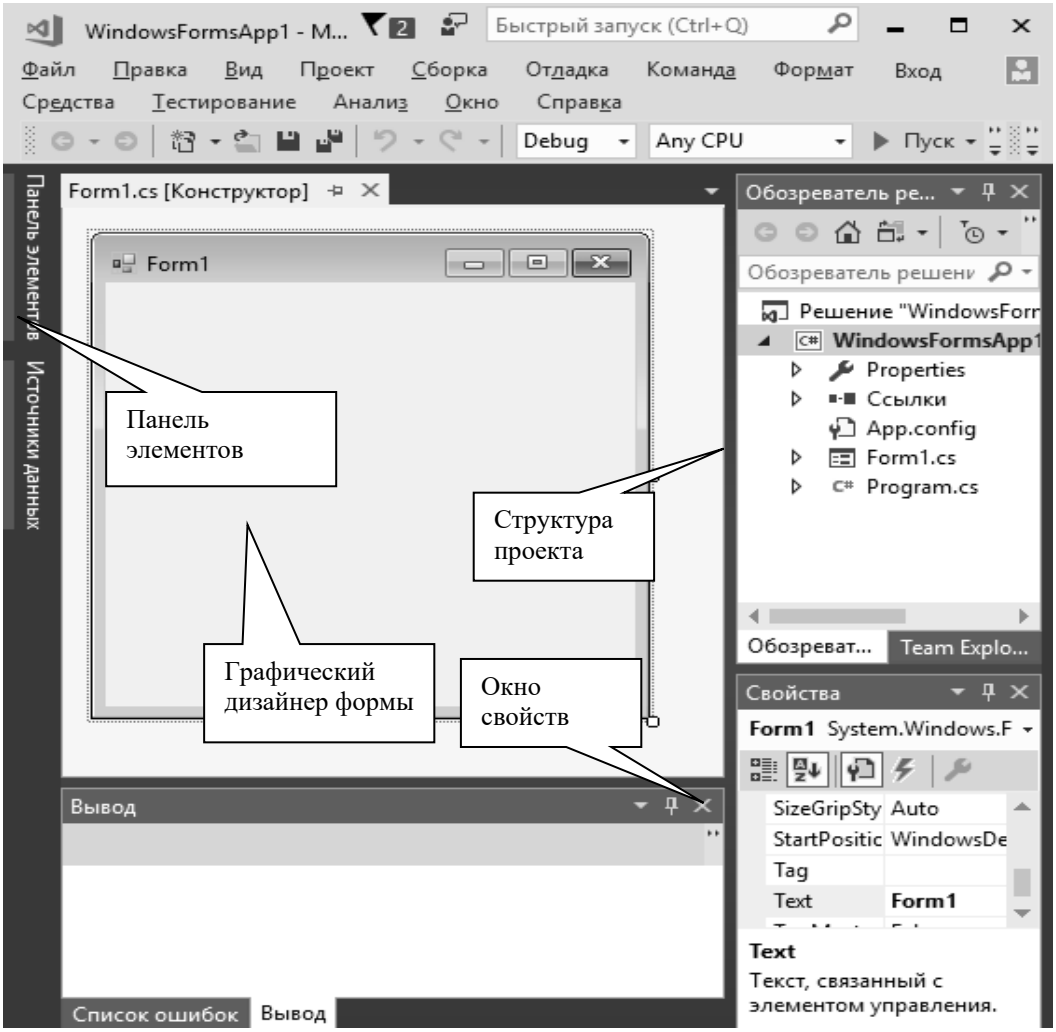


Рисунок 10.1

Большую часть пространства Visual Studio занимает графический дизайнер, который содержит форму будущего приложения. Пока она пуста и имеет только заголовок Form1. Справа находится окно файлов решения/проекта – Solution Explorer (Обозреватель решений). Там и находятся все связанные с нашим приложением файлы, в том числе файлы формы Form1.cs.

Внизу справа находится окно свойств – Properties (Свойства). Так как в данный момент выбрана форма как элемент управления, то в этом поле отображаются свойства, связанные с формой.

Создание оконных приложений сводится к созданию всех необходимых диалоговых окон, а также к размещению на них необходимых элементов. Окно Toolbox (панель инструментов, *View* → *Toolbo*x, или сочетание клавиш *Ctrl* + *Alt* + *X*) содержит компоненты Windows-форм, называемые также элементами управления, которые размещаются на форме. Для размещения нужного элемента управления достаточно просто щелкнуть на нем в окне *Toolbox* или, ухватив, перетащить его на форму. Основные элементы управления представлены в таблице 10.1.

Таблица 10.1

Функция	Элемент управления	Описание
1	2	3
Отображение информации (только для чтения)	Label	Отображает текст, недоступный для непосредственного изменения пользователем
	ProgressBar	Отображает текущее состояние операции для пользователя
Редактирование текста	TextBox	Отображает текст, введенный на этапе разработки, который может редактироваться пользователями во время выполнения или изменить программным способом
	RichTextBox	Включает текст, который отображается в формате в обычный текст или текст в формате (RTF)
	MaskedTextBox	Ограничивает формат вводимых пользователем данных
Команды	Button	Запускает, останавливает или прерываем процесс
	LinkLabel	Отображает текст в виде веб-ссылки и вызывает событие, когда пользователь щелкает этот текст. Обычно текст является ссылкой на другое окно или веб-сайт
Значение параметра	CheckBox	Отображает флажок и надпись для текста. Обычно используется для задания параметров
	RadioButton	Отображает кнопку, которая может включать и выключать
	TrackBar	Позволяет пользователям задавать значения на шкале, перемещая «ползунок» на шкале
Выбор из списка	ListBox	Отображает список текстовых и графических элементов (значков)
	ComboBox	Отображает список элементов раскрывающегося списка

Окончание таблицы 10.1

1	2	3
	CheckedListBox	Отображает прокручиваемый список элементов с флажками
	NumericUpDown	Отображает список чисел, который можно прокручивать с помощью кнопок со стрелками
Отображение данных	DataGridView	DataGridView – предоставляет настраиваемую таблицу для отображения данных
Отображение графики	PictureBox	Отображает графические файлы, такие как растровые изображения и значки, в кадре
Настройка даты	DateTimePicker	Отображает графический календарь, позволяющий пользователю выбрать дату или время
	MonthCalendar	Отображает графический календарь, позволяющий пользователю выбрать диапазон дат
Диалоговые окна	ColorDialog	Отображение диалогового окна выбора цвета, которое позволяет пользователям задать цвет элемента
	FontDialog	Отображает диалоговое окно, которое позволяет пользователям задавать шрифт и его атрибуты
	OpenFileDialog	Отображает диалоговое окно, позволяющее пользователям перейдите и выбрать файл
	SaveFileDialog	Отображает диалоговое окно для сохранения файла
Элементы управления меню	MenuStrip	Создание настраиваемых меню
	ContextMenuStrip	Создание настраиваемых контекстных меню
Группировка других элементов управления	Panel	Группирует набор элементов управления в прокручиваемый фрейм без подписи
	GroupBox	Группирует набор элементов управления (например, переключателей)
	TabControl	Предоставляет страницы с вкладками для организации и доступа к эффективно сгруппированным объектам
	SplitContainer	Предоставляет две панели, разделенные подвижной полосой

Элементы управления представляют собой визуальные классы, которые получают введенные пользователем данные и могут инициировать различные события. Все элементы управления наследуются от класса *Control* и поэтому имеют ряд общих свойств (таблица 10.2).

Таблица 10.2

Свойство	Описание
Anchor	Определяет, как элемент будет растягиваться
BackColor	Определяет фоновый цвет элемента
BackgroundImage	Определяет фоновое изображение элемента
ContextMenu	Контекстное меню, которое открывается при нажатии на элемент правой кнопкой мыши. Задается с помощью элемента ContextMenu
Cursor	Представляет, как будет отображаться курсор мыши при наведении на элемент
Dock	Задает расположение элемента на форме

Окончание таблицы 10.2

Свойство	Описание
Enabled	Определяет, будет ли доступен элемент для использования. Если это свойство имеет значение False, то элемент блокируется
Font	Устанавливает шрифт текста для элемента
ForeColor	Определяет цвет шрифта
Location	Определяет координаты верхнего левого угла элемента управления
Name	Имя элемента управления
Size	Определяет размер элемента
Width	Ширина элемента
Height	Высота элемента
TabIndex	Определяет порядок обхода элемента по нажатию на клавишу Tab
Tag	Позволяет сохранять значение, ассоциированное с этим элементом управления

Обработчик события (event handler) – это фрагмент кода, который выполняется при возникновении того или иного события (например, нажатие на кнопке, изменение текста и изменение положения бегунка). Каждый элемент управления имеет свой набор событий, на которые он способен реагировать.

Назначать событиям обработчики можно в дизайнере или же непосредственно в редакторе кода.

Сами обработчики событий прописываются в редакторе кода.

Чтобы добавить обработчик некоторого события:

– щелкните в дизайнере левой кнопкой мыши интересующий вас элемент управления или невизуальный компонент;

– перейдите к окну Properties и смените режим отображения свойств на режим отображения событий  (кнопка с изображением молнии);

– выберите интересующее вас событие и дважды щелкните по нему левой кнопкой мыши.

В результате этих действий будет сгенерирован пустой обработчик выбранного события, и на экране появится редактор кода.

Пример – Разработать Windows-приложения для выполнения простейшей операции (сложение, вычитание, умножение или деление) с двумя дробями вида $\frac{a}{b}$.

Необходимо:

1) создать проект File → New → Project → Windows Forms Application;

2) на форме расположить (рисунок 10.2):

а) шесть меток (Label);

б) шесть текстовых полей (TextBox);

в) четыре переключателя (RadioButton);

г) командную кнопку (Button);

3) с помощью окна свойств установить значения свойств объектов в соответствии с таблица 10.3 (выделяем объект и задаем значение в правом столбце

напротив имени свойства).

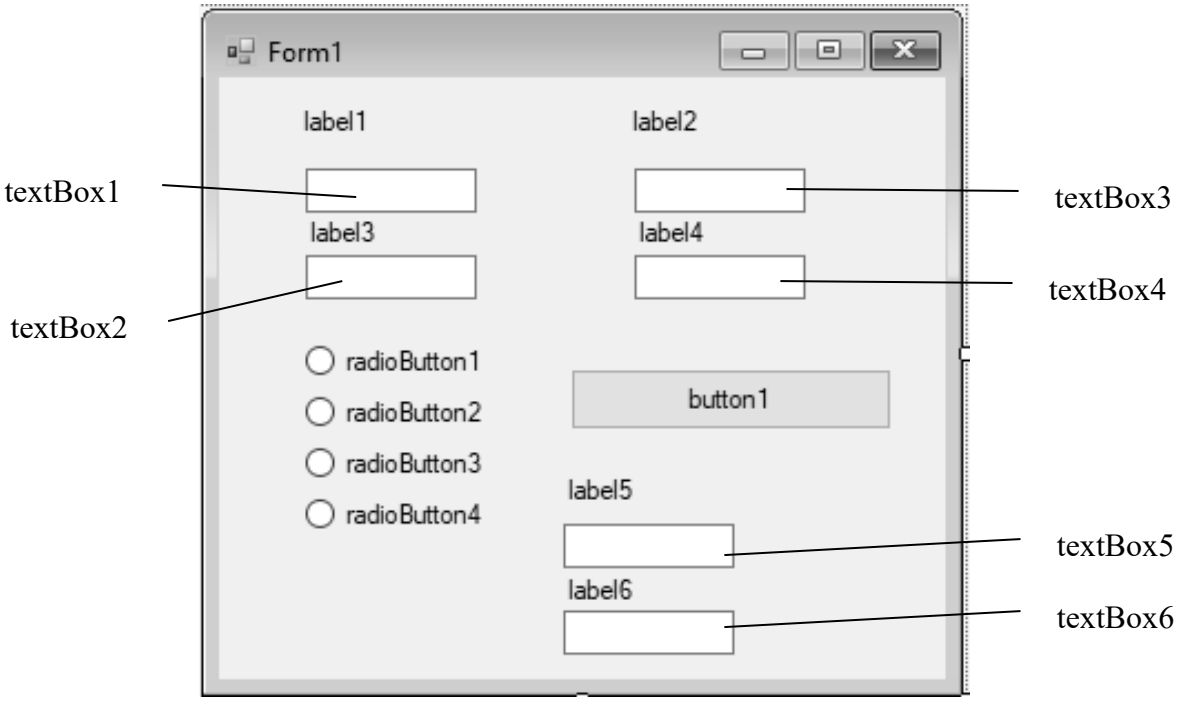


Рисунок 10.2

Таблица 10.3

Имя объекта по умолчанию (значение свойства Name)	Свойство	Значение свойства
Form1	Text	Простейший калькулятор с дробями
label1	Text	Первая дробь:
label2	Text	Вторая дробь:
label5	Text	Результат:
label3, label4, label6	Text	----------
radioButton1	Text	Сложение
radioButton1	Checked	True
radioButton2	Text	Вычитание
radioButton3	Text	Умножение
radioButton4	Text	Деление
button1	Text	Вычислить
textBox5, textBox6	Enabled	False

В результате форма примет вид, представленный на рисунке 10.3.

В нашем примере есть следующее событие: щелчок мышью по командной кнопке <Вычислить>. Создадим методы для обработки этих событий (обработчики событий). Дважды щелкнуть на кнопке <Вычислить>. Это приведёт непосредственно к обработчику события элемента управления, используемому по умолчанию – для кнопки таким событием является событие Click.

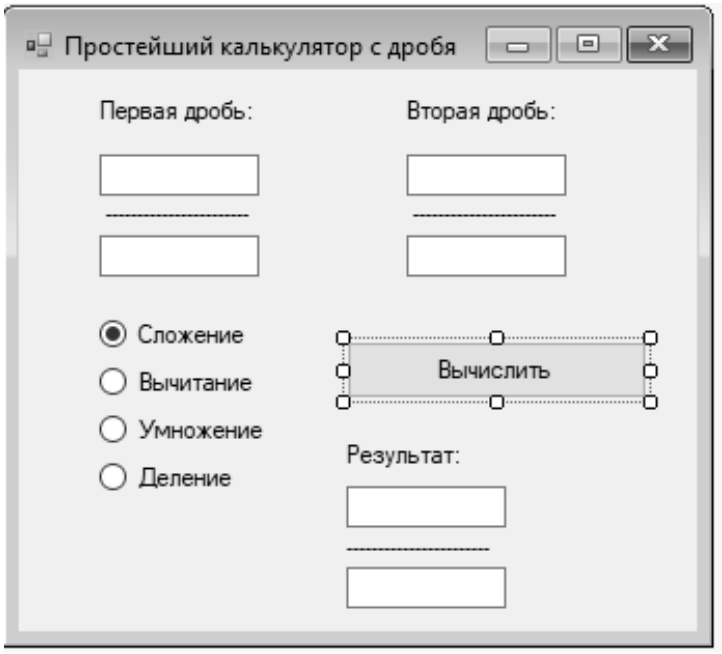


Рисунок 10.3

В окне редактора появится созданный автоматически шаблон обработчика события:

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Заполним этот обработчик:

```
private void button1_Click(object sender, EventArgs e)
{
    int a3=1,b3=1;
    //считываем данные с текстовых полей
    int a1 = int.Parse(textBox1.Text);//числитель1
    int b1 = int.Parse(textBox2.Text);//знаменатель1
    int a2 = int.Parse(textBox3.Text);//числитель2
    int b2 = int.Parse(textBox4.Text);//знаменатель1
    if (radioButton1.Checked)//если выбран 1-й переключатель
    {
        a3 = a1 * b2 + a2 * b1;
        b3 = b1 * b2;
    }
    if (radioButton2.Checked)//если выбран 2-й переключатель
    {
        a3 = a1 * b2 - a2 * b1;
        b3 = b1 * b2;
    }
    if (radioButton3.Checked)//если выбран 3-й переключатель
    {
        a3 = a1 * a2;
        b3 = b1 * b2;
    }
    if (radioButton4.Checked)//если выбран 4-й переключатель
```

```
        {
            a3 = a1 * b2;
            b3 = b1 * a2;
        }
        int nod = НОД(a3, b3);//находим НОД
        //вывод результата
        textBox5.Text = (a3 / nod).ToString();
        textBox6.Text = (b3 / nod).ToString();
    }
    //метод поиска НОД
    static int НОД(int a, int b)
    {
        int c;
        while (b!=0)
        {
            c = a % b;
            a = b;   b = c;
        }
        return a;
    }
```

Запустим программу на выполнение.

Усовершенствуем приложение. Добавим на форму элементы управления MenuStrip, OpenFileDialog и SaveFileDialog, как показано на рисунке 10.4.

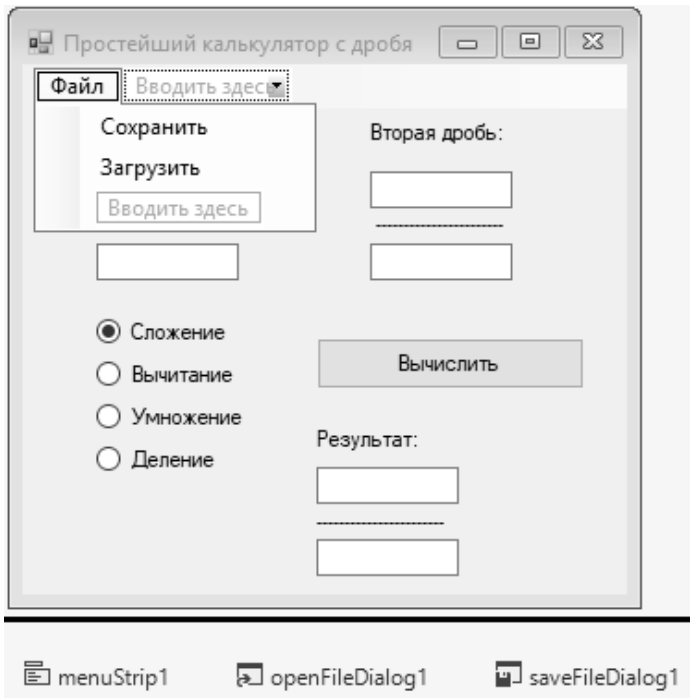


Рисунок 10.4

Создадим обработчики событий для сохранения и загрузки исходных данных (двойной щелчок по соответствующему пункту меню):

```
private void сохранитьToolStripMenuItem_Click(object sender, EventArgs e)
 {
  //открываем диалоговое окно "Сохранить как..."
  if(saveFileDialog1.ShowDialog()==DialogResult.OK)
  { //записываем исходные данные в заданный файл
   StreamWriter f = new StreamWriter(saveFileDialog1.FileName);
   f.WriteLine(textBox1.Text);
   f.WriteLine(textBox2.Text);
   f.WriteLine(textBox3.Text);
   f.WriteLine(textBox4.Text);
   f.Close();
 }  }
 private void загрузитьToolStripMenuItem_Click(object sender, EventArgs e)
 { //открываем диалоговое окно "Открыть"
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  { //считываем исходные данные из выбранного файла
   StreamReader f = new StreamReader(openFileDialog1.FileName);
   textBox1.Text=f.ReadLine();
   textBox2.Text = f.ReadLine();
   textBox3.Text = f.ReadLine();
   textBox4.Text = f.ReadLine();
   f.Close();
  }  }
```

Список литературы

1 **Гуриков, С. Р.** Введение в программирование на языке Visual C#: учебное пособие / С. Р. Гуриков. – Москва: ФОРУМ; ИНФРА-М, 2020. – 447 с.

2 **Дадян, Э. Г.** Современные технологии программирования. Язык C#: учебник: в 2 т. / Э. Г. Дадян. – Москва : ИНФРА-М, 2021. – Т. 1. – 312 с.

3 **Гагарина, Л. Г.** Технология разработки программного обеспечения: учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул; под ред. Л. Г. Гагариной. – Москва: ФОРУМ; ИНФРА-М, 2019. – 400 с.

4 **Немцова, Т. И.** Программирование на языке высокого уровня. Программирование на языке C++: учебное пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев; под ред. Л. Г. Гагариной. – Москва: ФОРУМ; ИНФРА-М, 2021. – 512 с.