МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

О.К. ГОЛОВНИН, А.А. СТОЛБОВА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ С ЭЛЕМЕНТАМИ СИСТЕМНОГО АНАЛИЗА ИНФОРМАЦИИ

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве учебного пособия для обучающихся по основной образовательной программе высшего образования по направлению подготовки 09.03.01 Информатика и вычислительная техника

С А М А Р А
Издательство Самарского университета
2019

УДК 004.42
ББК 32.973
Г61

Рецензенты: канд. техн. наук, доц. Л.С. Зеленко,
д-р техн. наук, проф. А.В. Иващенко

Головнин, Олег Константинович
Г61 Программирование на языке высокого уровня с элементами системного анализа информации: учеб. пособие / *О.К. Головнин, А.А. Столбова*. – Самара: Изд-во Самарского университета, 2019. – 132 с.: ил.

ISBN 978-5-7883-1430-3

Учебное пособие содержит краткий теоретический и практический материал, касающийся разработки компьютерных программ на языке высокого уровня C# с применением методов системного анализа для информационного моделирования. Рассмотрен процесс создания компьютерных программ, описаны основные типы данных, операторы, приведены базовые структурные элементы алгоритмов и способы работы с массивами. Внимание уделено системному анализу предметной области и объектно-ориентированному программированию. Рассмотрены вопросы создания графического интерфейса программ Windows Forms. Приведены задания для самостоятельной практической работы. Ориентировано на студентов высших учебных заведений, изучающих современные информационные технологии, системный анализ и программирование, в том числе обучающихся по направлению 09.03.01 Информатика и вычислительная техника. Подготовлено на кафедре информационных систем и технологий.

УДК 004.42
ББК 32.973

ISBN 978-5-7883-1430-3

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Учебное пособие содержит краткий теоретический и практический материал, касающийся разработки компьютерных программ на языке высокого уровня C# с применением методов системного анализа для информационного моделирования. В пособии рассмотрен процесс создания компьютерных программ, описаны основные типы данных, операторы, приведены базовые структурные элементы алгоритмов и способы работы с массивами. Внимание уделено системному анализу предметной области и объектно-ориентированному программированию. Рассмотрены вопросы создания графического интерфейса программ Windows Forms. Приведены задания для самостоятельной практической работы.

Учебное пособие предназначено для формирования профессиональных знаний и практических навыков применения методов программирования при исследовании систем и автоматизации процессов путем выбора и обоснования рациональных решений через предварительную оценку эффективности.

Учебное пособие ориентировано на студентов высших учебных заведений, изучающих современные информационные технологии, системный анализ и программирование, в том числе обучающихся по направлению 09.03.01 Информатика и вычислительная техника.

1 ПРОЦЕСС СОЗДАНИЯ ПРОГРАММ

1.1 Программирование: основные понятия, термины и определения

Программирование – процесс создания компьютерной программы.

Программа – совокупность данных и команд, предназначенная для реализации алгоритмов на компьютерных устройствах.

Никлаус Вирт (1985 г.) сформулировал следующее высказывание:

«Алгоритмы + Структуры данных = Программа».

Алгоритм – набор команд, описывающих порядок действий для достижения некоторого результата.

Команда – описание операции, которую должно выполнить компьютерное устройство.

Структура данных – совокупность данных, между которыми существуют некоторые отношения.

Данные – представления информации в формализованном виде.

Язык программирования – формальный язык, предназначенный для записи компьютерных программ.

1.2 Структура программы в Visual Studio

Основная среда разработки программ на языке C# – Visual Studio. В Visual Studio для разработки программ используются проекты (Project) и решения (Solution). Проект содержит совокупность классов, ресурсов, прочих файлов и выступает в качестве основы, формирующей программу или библиотеку. Проект характеризуется типом, который определяет порядок шаблон (каркас) проекта, фор-

мируемый Visual Studio. Для структурирования проектов применяются пространства имен (Namespace), которые объединяют в одну логическую группу близкие классы.

Общая структура программы приведена на рисунке 1.

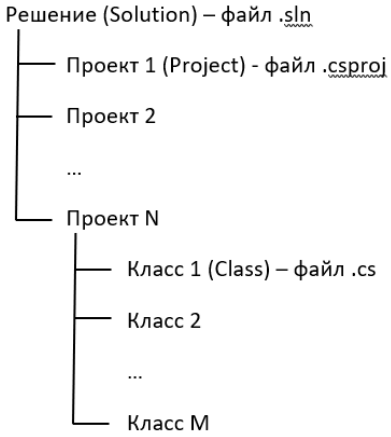


Рис. 1. Структура программы в Visual Studio

Пример простейшей программы:

```
// Так выглядит комментарий к коду

// Сокращаем обращение к классам,
// размещенным в пространстве имен System
using System;

// Объявляем пространство имен
namespace MyFirstProgram
{
   /* Так выглядит
   многострочный комментарий */

   // Объявляем класс
```

```
    class Program
    {
        // Объявляем метод
        static void Main(string[] args)
        {
            // «System» можно не писать, если есть «using System;»
            // Выводим на экран консоли «Hello World!»
            System.Console.WriteLine("Hello World!");
            // Ждем нажатия любой клавиши
            System.Console.ReadKey();
        }
    }
}
```

1.3 Компиляция и исполнение программ

Исходный код программы должен быть скомпилирован – переведен в язык машинных команд, которые могут быть исполнены аппаратным обеспечением компьютера – центральным процессором. Скомпилированная программа представляет собой, как правило, исполняемый файл с расширением «exe»; скомпилированная программная библиотека – файл с расширением «dll».

При классическом подходе файл с расширением «exe» может быть запущен на исполнение процессором, поскольку машинный язык содержит команды процессора, однако .NET Framework действует иначе – исполняемый файл содержит команды на языке Common Intermediate Language (CIL), который выполняет исполняющая среда Common Language Runtime (CLR).

1.4 Исполняющая среда CLR

Исполняющая среда CLR во время выполнения программы транслирует команды на языке CIL в язык команд центрального про-

цессора конкретного компьютера, тем самым обеспечивая следующие преимущества:

- программа, скомпилированная однажды, выполняется на любом центральном процессоре, который поддерживает исполняющая среда.
- исполняющая среда максимально эффективно задействует набор команд текущего центрального процессора.

Недостаток такого подхода – затраты времени на трансляцию программы во время выполнения.

1.5 Средства разработки программ

Код на языке программирования C# может быть написан в любом текстовом редакторе, однако основное средство разработки программ на языке программирования C# с использованием .NET Framework – интегрированная среда разработки Microsoft Visual Studio. В среду разработки интегрированы следующие инструменты, которые могут быть задействованы при разработке программы:

- текстовый редактор с подсветкой синтаксиса;
- компилятор;
- отладчик;
- обозреватель решений;
- редактор свойств;
- профайлер и др.

2 ТИПЫ ДАННЫХ

2.1 Основные типы данных

C# – язык программирования со строгой типизацией, т.е. для всех создаваемых объектов должен объявляться тип. Компилятор осуществляет контроль типов для предотвращения возникновения ошибок. Рассмотрим основные типы данных – числовые, символьные, логический, финансовый.

1) Числовые:
   а) Целые:
      – беззнаковые:
         - byte – 1 байт – целое число от 0 до 255;
         - ushort – 2 байта – целое число от 0 до 65535;
         - uint – 4 байта – целое число от 0 до 2^{32};
         - ulong – 8 байт – целое число от 0 до 2^{64}.
      – знаковые:
         - sbyte – 1 байт – целое число от -128 до 127;
         - short – 2 байта – целое число от -32768 до 32767;
         - int – 4 байта – целое число от -2^{31} до $(2^{31}-1)$;
         - long – 8 байт – целое число от -2^{63} до $(2^{63}-1)$.

   б) Действительные (вещественные):

– float – 4 байт – вещественное число со знаком в диапазоне от $\pm 1.5\times10^{-45}$ до $\pm 3.4\times10^{38}$ (одинарная точность, примерно 6-9 десятичных цифр);

– double – 8 байт – вещественное число со знаком в диапазоне от $\pm 5.0\times10^{-324}$ до $\pm 1.7\times10^{308}$ (двойная точность, примерно 15-17 десятичных цифр).

2) Символьные:

а) char – 2 байта – один символ Unicode;

б) string – размер зависит от количества символов – строка.

3) Логический (булевский): bool – 1 байт – принимает значения true/false (правда/ложь).

4) Финансовый: decimal – 16 байт – вещественное число со знаком в диапазоне от $\pm 1.0 \times 10^{-28}$ до $\pm 7.9 \times 10^{28}$ (примерно 28-29 десятичных цифр).

Целые числовые типы данных представляются в памяти компьютера следующим образом (рисунок 2). Единица хранения информации – бит, который может принимать два значения: 0 (нет, false) или 1 (да, true). 8 бит составляют 1 байт. Старший бит используется для хранения знака (если тип данных – со знаком).

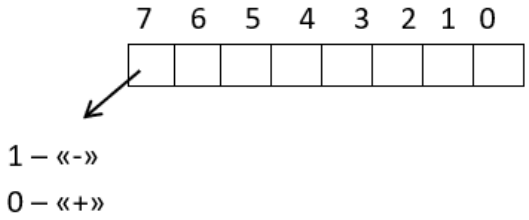


Рис. 2. Целое число со знаком в памяти компьютера

Примеры целых чисел в памяти компьютера:

00000000 = ноль
00000001 = 1
11111111 = 255
00100100 = $2^5 + 2^2 = 36$
10000000 = -128
01111111 = $2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 127$

Символьные типы данных используют кодировку Unicode (UTF). Первые 0..127 (0..7F) кодов кодировки UTF совпадают с кодировкой American Standard Code for Information Interchange (ASCII) (таблица 1).

Таблица 1. **Кодировка ASCII**

	0	1	2	…	9	A	…	F
0								
1								
…								
4		A	B					0
0		a	b					0
7								

Пример символа «A» в памяти компьютера:

$$4^1 1^0{}_{16} = 4 * 16^1 + 1 * 16^0 = 65_{10} = 'A'$$

Действительные числа представляются следующим образом (рисунок 3).

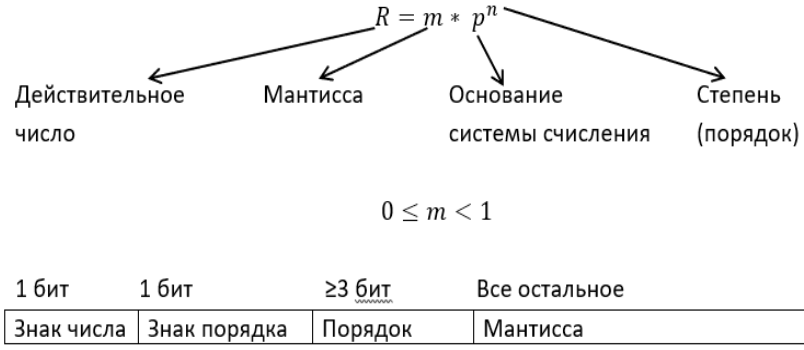


Рис. 3. Представление действительного числа в памяти компьютера

Примеры объявления переменных различных типов данных:

```
string hello = "Hello world!";
string no = " ";

char a = 'a';

int l = -84;
int k=1;

flout x = 3.5f; // Без f будет double

double d = 5.55;
double d1 = 5.55d;

bool b = false;

long lon = 371L;
```

Строки включаются в двойные кавычки ("Это строка"), символы – в одинарные кавычки (*'Б'*). Таким образом, "Б" – это тоже строка, но состоящая из одного символа.

2.2 Типы значения и ссылочные типы

Типы данных по способу хранения в памяти можно разделить на 2 вида: типы значения и ссылочные типы (рисунок 4). Типы значения хранятся в стеке, ссылочные типы – в куче.

Стек – структура данных, организовывается по принципу LIFO (Last In – First Out). Доступ в стеке всегда есть только к последнему добавленному в стек элементу. Пример – стопка тарелок.

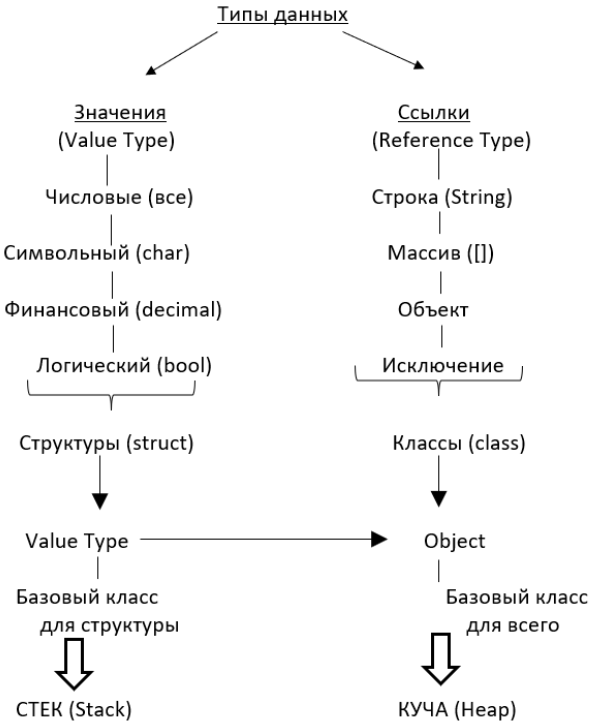


Рис.4. Классификация типов данных

В оперативной памяти каждому потоку выполнения выделяется определенный размер стека, т.е размер стека – фиксированная величина. Превышение лимита приводит к переполнению стека – StackOverflowException.

Справочно: существует структура данных, работающая наоборот – очередь – она организована по принципу FIFO (First In – First Out). Пример – очередь на кассе.

Куча – хранилище данных, расположенное в оперативной памяти, которое допускает динамическое выделение памяти. Местоположение данных в куче определяется указателем-ссылкой IntPtr. Куча обеспечивает произвольный доступ к любому элементу, но каждый

раз требуется вычислять физическое расположение данных в куче по указателю и типу данных.

Таким образом, стек – быстрый (всегда известен адрес последнего элемента), небольшой, в нем организован доступ только к последнему элементу.

Куча – медленная (требуется вычислять адрес элемента), большая (ограничена размером оперативной памяти), в ней организован доступ к любому элементу по указателю. Выделение памяти в куче – операция «new». В .NET Framework при использовании исполняющей среды CLR не надо следить за освобождением памяти в куче, за это отвечает сборщик мусора. В неуправляемом коде следуют вручную освобождать память, иначе происходит так называемая «утечка памяти».

2.3 Значения по умолчанию

Для ссылочных типов значение по умолчанию – «null», для типов-значений – «0», для структур – шаблон нулевых бит. Для получения значения по умолчанию существует следующая команда:

default(<Тип данных>)

Пример использования команды:

```
string s = default(string); // В переменную s помещается null
int i = default(int); // В переменную i помещается 0
bool b = default(bool); // В переменную i помещается false
```

Для того, чтобы иметь возможность присвоить значение «null» переменной, имеющей тип-значение, существует механизм Nullable-типов, который не рассматривается в настоящем учебном пособии.

2.4 Преобразование типов

Существует два способа преобразовать один тип данных в другой:

1) Явное (приведение) – нужен оператор (потеря данных).

В случаях, когда преобразование типов данных может привести к потере данных, используется явное преобразование, требующей специального оператора приведения – круглые скобки с указанием типа, к которому требуется привести исходный.

Пример явного преобразования:

```
float f0 = -5.8f;
int f1 = (int)f0;   // f1 равен -5, дробная часть отбрасывается
uint f2 = (uint)f1;   // f2 равен 5, знак отбрасывается
```

2) Неявное – данные не теряются, специальных действий не требуется.

Неявное преобразование задействуется в тех случаях, когда не происходит потери данных, например: int приводится к long, float, double; float приводится к double. Оператор преобразования не требуется.

Пример неявного преобразования:

```
float f0 = -5.8f;
double f1 = f0; // f1 равен -5.8d
```

2.5 Операции с типами данных

Операция – какое-либо действие над операндами. Основные операции, доступные в языке C#, приведены ниже.

Арифметические операции: «+» – сложение, «-» – вычитание, «*» – умножение, «/» – деление, «%» – остаток от деления. Круглые скобки «(», «)» используются для указания приоритетов операций.

Операция сравнения: «>» – больше, «<» – меньше, «>=» – больше или равно, «<=» – меньше или равно, «==» – равно, «!=» – не равно.

Логические операции: «&&» – логическое «И» (конъюнкция), «||» – логическое «ИЛИ» (дизъюнкция), «!» – логическое «НЕ» (отрицание).

Пример арифметических операций:

```
float f0 = -5.8f;
float f1 = 10.1f;
float f3 = f0 + f1;
float f4 = (f0 + f3) / 2.3f;

int f5 = 0;
f5 = f5 + 5;
f5 += 5; // Эта операция эквивалентна предыдущей
f5 += 1;
f5++; // Эта операция эквивалентна предыдущей
f5 -= 1;
f5--; // Эта операция эквивалентна предыдущей
```

Таким образом, выделяются сокращённые операции: «+=», «-=», «*=», «/=» и т.д. Операции увеличения и уменьшения на 1 соответственно: «++» – инкремент, «--» – декремент.

Операции сдвига («>>» и «<<») и битовые операции AND («&»), OR («|»), XOR(«^») не рассмотрены в настоящем учебном пособии.

3 ОПЕРАТОРЫ

Оператор – наименьшая часть языка программирования (команда или набор команд). Рассмотрим основные операторы языка программирования C#. Тут и далее угловыми скобками «<>» обозначены действия, не относящиеся к синтаксису рассматриваемого оператора, квадратными скобками «[]» обозначены необязательные синтаксические конструкции.

3.1 Логические операторы

Логические операторы позволяют осуществить проверку условия и осуществить выбор действий, наступающих в случае, если условие истинно.

1) Логический оператор:

if (<Логическое выражение>) <Действие>
[else <Альтернативное действие>];

<Логическое выражение> должно возвращать булевский тип. <Действие> и <Альтернативное действие> представляют собой команду или набор команд, заключенные в операторные скобки.

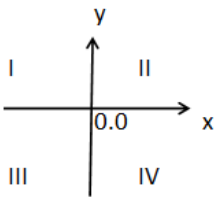
2) Тернарный логический оператор:

<Логическое выражение> ? <Действие> : <Альтернативное действие >;

<Логическое выражение> должно возвращать булевский тип. <Действие> и <Альтернативное действие> представляют собой одну команду.

Пример использования логических операторов:

Задача – ввести с клавиатуры точку – координатную пару (X, Y) и вывести на экран сведения о расположении точки на координатном пространстве (рисунок 5): одна из четвертей I – IV, ось X, ось Y, начало координат.



I

II

III

IV

нк(0,0)

ось x

ось y

Рис. 5. Расположение точки в пространстве

```
static void Main(string[] args)
    {
       Console.Write("Введите x = ");
       string sx = Console.ReadLine();
       Console.Write("Введите y = ");
       string sy = Console.ReadLine();
       int x = int.Parse(sx);
       int y = int.Parse(sy);
       if (x==0 && y==0)
       {
          Console.Write("Начало координат");
       }
      else // Продолжить решение задачи…
    }
```

Различные варианты решения задачи определения координатной четверти приведены в Приложении А. Приведенные решения ставят своей целью минимизацию количества условных операторов.

Пример использования тернарного логического оператора:

Задача – разделить целое число X на целое число Y, а в случае, если Y 0, то получить 1.

```
int res = y != 0 ? x / y :1;
```

3.2 Оператор присваивания

Оператор присваивания позволяет задать значение переменной:

<Переменная> = <Выражение>;

Если переменная представляет собой тип-значение, то в ней размещается само значения выражения. Если переменная представляет собой ссылочный тип данных, то в ней размещается ссылка на объект.

Не следует путать оператор присваивания «=» и операцию сравнения на равенство «==».

Пример задания значений переменным:

```
int a;
a = 260;
a = 260 + 357;
a = a + 3;

double d;
d = 36.6;
d = 36.6 / a;
```

3.3 Оператор выбора

Оператор выбора позволяет в зависимости от значения переменной выбрать один вариант из множества альтернатив:

```
switch (<Переменная выбора>)
        {
                case <Вариант 1>:<Действие 1>;
                                        break;
                case <Вариант 2>:<Действие 2>;
                                        break;
                // ...
                case <Вариант N>:<Действие N>;
                                        break;
                default: < Действие по умолчанию>;
                        break;
        }
```

В C# 6 и более ранних версиях переменная выбора может быть одного из следующих типов: char, string, bool, int, long. В C# 7 и новее – любым выражением, отличным от null.

Пример использования оператора выбора:

```
int input = 3; // Вариант Г
string output;
switch(input)
{
   case 0:
      output = "Вариант А";
      break;
   case 1:
      output = "Вариант Б";
      break;
```

```
    case 2:
        output = "Вариант В";
        break;
    case 3:
        output = "Вариант Г";
        break;
    default:
        output = "Такого варианта нет";
        break;
}
Console.WriteLine(output);
```

3.4 Оператор перехода

Оператор перехода позволяет осуществить безусловный переход к метке, установленной в коде программы:

goto <Метка>;

Метка формируется следующим образом:

<Метка>: <Команда>;

Важно: использование операторов такого вида не рекомендуется, поскольку нарушает принципы структурного программирования (Эдсгер Вибе Дейкстра, 1968 г.).

3.5 Операторные скобки

Структурирование кода осуществляется с использованием операторных скобок:

{<Блок кода>}

Блок кода может не содержать команд, но, как правило, в блок кода включаются одна или несколько команд. Команды в блоке кода выполняются последовательно. Операторные скобки ограничивают область видимости переменных, объявленных внутри них: переменная, объявленная в операторных скобках, доступна только в пределах этих операторных скобок. Блок кода может содержать другие блоки кода, ограничиваемые операторными скобками. Тогда область видимости переменных распространяется на внутренние вложенные блоки.

Пример использования операторных скобок:

```
{
   int a = 1;
   {
      int b = a + 1;
      {
         int c = b + 1;
         Console.WriteLine(a);
         Console.WriteLine(b);
         Console.WriteLine(c);
      }
      Console.WriteLine(a);
      Console.WriteLine(b);
      Console.WriteLine(c); // Ошибка доступа
   }
   Console.WriteLine(a);
   Console.WriteLine(b); // Ошибка доступа
   Console.WriteLine(c); // Ошибка доступа
}
```

3.6 Комментарии

Комментирование кода позволяет включить в текст исходной программы описания, которые не оказывают влияние на результат

компиляции и размер итоговой программы. Комментарии позволяют прояснять некоторые моменты в исходном коде с помощью кратких текстовых сообщений. Кроме этого, комментарии позволяют не удалять код, который в данный момент не требуется, а превратить его в комментарий, с целью дальнейшего удаления или, наоборот, возвращения в рабочее состояние.

Однострочный комментарий включается в исходный код программы с помощью символов «//», стоящих в начале комментария:

```
// Однострочный комментарий
```

Многострочный комментарий включается в исходный код программы с помощью символов «/*», обозначающих начало комментария, и «*/», обозначающих конец многострочного комментария:

```
/* Это
многострочный
комментарий */
```

Использование многострочных комментариев не рекомендуется, поскольку общепринятая практика сводится к тому, чтобы писать код, понятный без комментариев. Если код требует обширного комментария, то такой код должен быть переписан. Не рекомендуется оставлять в исходном коде закомментированный исходный код, поскольку наличие такого кода затрудняет чтение программы.

Не следует путать комментарии с документаций (Summary), которая так же, как и комментарии, не включается в компилируемую программу или библиотеку, но может быть использована для формирования описаний классов для программистов и контекстной подсказки в интегрированной среде разработки. Для написания документации необходимо в интегрированной среде разработки Visual Studio рядом с документируемым классом или членом класса поставить символы «///», которые будут заменены на блок описания, соответствующей текущему шаблону.

Примеры документации:

```
/// <summary>
/// Метод возвращает количество элементов массива.
/// Если массив равен null или не имеет элементов,
/// метод вернет -1.
/// </summary>
/// <param name="array">Массив для подсчета.</param>
/// <returns>Количество элементов в массиве.</returns>
int GetLength(int[] array)
{
   if (array != null && array.Length > 0)
         return array.Length;
   else return -1;
}
```

3.7 Итерационный цикл

Цикл (перебор) – оператор, предназначенный для организации многократного повторения команд, составляющих тело цикла.

Итеративный цикл «for» предназначен для повторения тела цикла заданное количество раз. Структура итеративного цикла «for» приведена на рисунке 6.

Рис. 6. Структура цикла for

Таким образом, для повторения одной команды:

```
for (<Инициализация>; <Условие>; <Итератор>)
   <Команда>;
```

Для повторения одной команды или блока кода (команды выполнятся последовательно друг за другом):

```
for (<Инициализация>; <Условие>; <Итератор>)
{
   <Команда 1>;
   <Команда 2>;
   ...
   <Команда N>;
}
```

<Условие> – это условие продолжения цикла, т.е. пока условие является истиной, цикл продолжается.

Пример цикла for, который выполнится 5 раз:

```
for (int i = 0; i<5; i++)
{
   Console.WriteLine(i);
}
```

Тут *i* – локальная переменная цикла, которая вне пределов цикла не существует (находится за границами видимости). Не рекомендуется изменять переменную цикла внутри тела цикла.

Пример бесконечного цикла for:

```
for(;;)
{
   <Тело цикла>
}
```

Пример цикла for с переменной цикла типа char:

```
for (char i = “A”; i < “D”; i++) {...} // A, B, C
```

Пример цикла for с обратным отсчетом:

```
for (int i = 5; i>0; i--)
{
   Console.WriteLine(i); // 5 4 3 2 1
}
```

Хорошая практика программирования заключается в применении целых числовых типов или символьного char для объявления переменной цикла. Не рекомендуется использовать переменные цикла действительного типа – float, double.

Пример цикла for с переменной цикла типа float:

```
for (float i = 1 000 000 000; i < 1 000 000 010; i++)
   Console.WriteLine(i);
```

Ожидается, что в данном случае цикл выполнится 10 раз, однако в данном случае цикл не закончит своё выполнение – получен бесконечный цикл, поскольку в представлении float числа 1 000 000 000 и 1 000 000 010 равны 1.0E+9. Прибавление единицы не изменяет этого значения.

Вопрос для самоконтроля: сколько раз выполнится цикл и что будет выведено на экран?

```
for (int i = 0; i<10; i++)
{
   i=i+2;
   Console.WriteLine(i);
}
```

3.8 Цикл с постусловием

Цикл с постусловием осуществляет выполнение тела цикла, после чего осуществляет проверку условия, и пока условие выполняется, цикл повторяется:

```
do
   {<Тело цикла>}
while (<Условие>);
```

<Условие> – это условие продолжения его выполнения, т.е. цикл продолжает свою работу только в том случае, если условие является истиной. Этот цикл выполнится 1 или более раз.

Пример цикла с постусловием do:

```
int n=0;
do
{
   Console.WriteLine(n);   // 0 1 2 3 4
   n++;
}
while (n<5);
```

Пример бесконечного цикла do:

```
do
{
   <Тело цикла>
}
while (true);
```

3.9 Циклы с предусловием

Цикл с предусловием осуществляет проверку условия и выполняет тело цикла, пока условие выполняется:

```
while (<Условие>)
   <Тело цикла>
```

<Условие> – это условие входа в цикл и продолжения его выполнения, т.е. цикл продолжает свою работу только в том случае, если условие является истиной. Этот цикл выполнится 0 и более раз.

Пример цикла с предусловием while:

```
int n = 0;
while (n<5)
{
   Console.WriteLine(n);   // 0 1 2 3 4
   n++;
}
```

Пример бесконечного цикла while:

```
while (true)
{
   <Тело цикла>
}
```

3.10 Цикл-перебор

Цикл-перебор позволяет выполнить тело цикла для каждого элемента в наборе элементов:

```
foreach (<Элемент> in <Набор элементов>)
{
   <Тело цикла>
}
```

Структурный подход к программированию подразумевает отсутствие в циклах операторов: break; continue; goto.

4 МАССИВЫ

Массив – регулярная структура однотипных данных. Элементы массива объединены общим именем и идентифицируются индексами.

4.1 Одномерные массивы

В одномерном массиве (векторе) каждый элемент идентифицируется одним индексом (рисунок 7). Индекс первого элемента – 0.

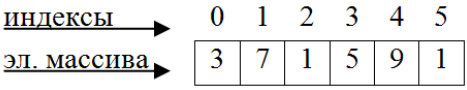


Рис. 7. Структура одномерного массива

Пример объявления одномерных массивов:

```
// Так не рекомендуется:
int[] a=new int[6]; // Массив целых чисел

// Рекомендуется выделить размерность
// в отдельную переменную
int n =6; // Размерность вектора
int [] a=new int [n]; // Требуется, чтобы n>0
int [] b=new int [2*n];

string c = new string[n]; // Массив строк

double d = new double[n]; // Массив действительных чисел
```

Пример обращения к элементам массива:

```
a[0] // Обращение к первому элементу
a[1] // Обращение ко второму элементу
a[n-1] // Обращение к последнему элементу
a[n] // Ошибка – выход за границы массива
```

Пример ввода вектора с клавиатуры:

```
Console.WriteLine(“Размерность вектора =”);
int n= int.Parse(Console.ReadLine());
int[] a=new int [n];
for(int i = 0;i < n; i++)
{
   Console.WriteLine("a[" + i +"] = ");
   a[i] = int.Parse(Console.ReadLine());
}
```

Пример вывода вектора на экран:

```
Console.WriteLine("Это вектор: ");
for(int i = 0; i < n; i++)
{
   Console.WriteLine("a[{0}] = {1}", i, a[i]);
}
```

Пример вывода вектора на экран с помощью foreach:

```
foreach(int element in a)
{
   Console.WriteLine(element);
}
```

4.2 Двумерные массивы

Двумерный массив (матрица) определяет два индекса для идентификации элемента – номер строки и столбца (рисунок 8).

	0	1	2	3
0	a_{11}	a_{12}	a_{13}	a_{14}
1	a_{21}	a_{22}	a_{23}	a_{24}
2	a_{31}	a_{32}	a_{33}	a_{34}
3	a_{11}	a_{12}	a_{13}	a_{14}

Рис. 8. Структура двумерного массива

Элемент массива $a_{i,j}$ описывается с помощью индексов: i – номер строки; j номер столбца.

Пример объявления массива:

```
int n=3; // Число строк
int m=4; // Число столбцов
int[,] a=new int [n, m];
```

Пример вывода двумерного массива на экран:

```
for(int i=0; i<n; i++)
{
    for(int j=0; j<m; j++)
{
            Console.WriteLine("a[{0},{1}]={2}", i, j, a[i, j]);
    }
}
```

4.3 Многомерные массивы

Многомерные массивы для идентификации элемента требуют более двух индексов, т.е. число измерений в них больше 2.

Пример объявления многомерных массивов:

```
int[,,] a=new int [n, m, k]; // Трехмерный
int[,,,] b= new [n, m, k, t]; // Четырехмерный
```

4.4 Ступенчатые массивы

Ступенчатые массивы представляют собой массивы, содержащие в качестве элементов другие массивы.

Пример объявления ступенчатого массива:

```
int [][] a =new int [6][]; // Массив, состоящий из 6 массивов
a[0]=new int[3]; // Первый элемент массива – массив из 3 чисел
a[1]=new int[5]; // Второй элемент массива – массив из 5 чисел
```

4.5 Работа с массивами

При работе с массивами следует обратить внимание на следующие моменты:

- Обращение к индексированной переменной дольше, чем к неиндексированной.
- Не рекомендуется изменять размер массива после объявления.
- Можно получить количество элементов в массиве через его свойство: a.Length (для вектора – размерность n, для матрицы – n*m и т.п.).

– Существуют различные способы объявления и инициализации массивов:

```
int[] a;
int[] a=new int [5];
int[] a=new int []{5,6,7,8};
int[] a={5,6,7,8};
int[,] a={{5,6},{7,8}};
```

– Сравнение массивов осуществляется по ссылке, а не по составу.
– Переменная-массив – это всегда ссылка на массив в куче, поэтому не требуется дополнительных указаний на способ передачи массива по ссылке.
– Операции сложения, умножения, вычитания, деления для массивов не определены.

5 АЛГОРИТМЫ ПРОГРАММИРОВАНИЯ

5.1 Базовые структуры алгоритмов

Существует 3 базовых структуры алгорима: следование или последовательность (рисунок 9), ветвление (рисунок 10), цикл (рисунок 11). Любая программа может быть представлена с помощью этих трёх управляющих структур (Коррадо Бём и Джузеппе Якопини, 1965 г.).

Рис. 9. Следование

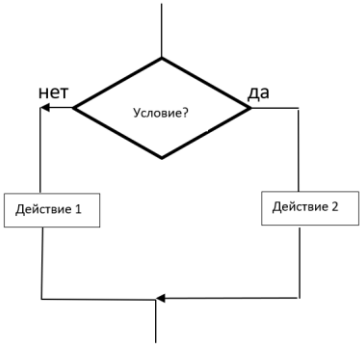


Рис. 10. Ветвление

Рис. 11. Цикл

5.2 Элементы схем алгоритмов

Основные элементы схем алгоритмов приведены на рисунке 12.

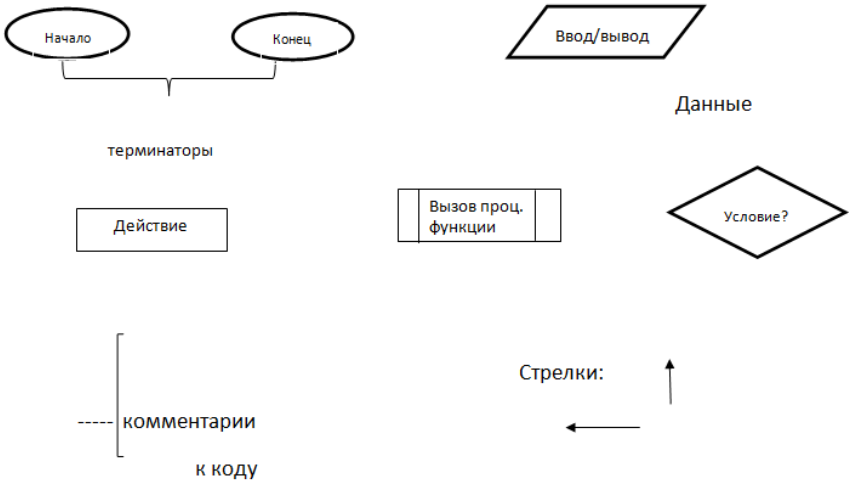


Рис. 12. Основные элементы схем алгоритмов

5.3 Алгоритмы обработки массивов

В этом разделе рассмотрены основные классические алгоритмы работы с массивами.

Инвертирование вектора, т.е. элементы исходного вектора копируются в вектор с изменением порядка на обратный: $a_0 => b_n$, $a_1 => b_{n-1}$ и т.д.

Пример инвертированного вектора приведен на рисунке 13.

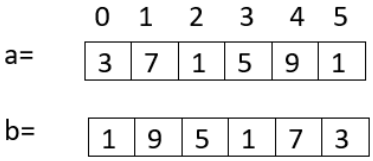


Рис. 13. Инвертирование вектора

Пример алгоритма инвертирования вектора в самом себе:

```
int temp = 0;
for(int i=0; i<n/2; i++)
{
    temp = a[i];
    a[i] = a[n-i-1];
    a[n-i-1] = temp;
}
```

Нахождение **максимального элемента** в векторе.

Пример:

```
int max=a[0];
for(int i=1; i<n; i++)
    if (a[i] > max)
            max=a[i];
```

Нахождение **максимального элемента** в векторе и его индекса.
Пример:

```
int max =a[0];
int maxIndex=0;
for(int i=1; i<n; i++)
    if (a[i]>max)
    {
            max=a[i];
            maxIndex=i;
    }
```

Нахождение **суммы элементов матрицы**.
Пример:

```
int sum=0;
for(int i=0 ;i<n; i++)
    for(int j=0; j<m; j++)
            sum += a[i, j];
```

Далее рассмотрены алгоритмы нахождения суммы элементов матрицы, стоящих ниже или выше главной или побочной диагоналей, а также четвертей, образованных диагоналями матрицы.

Пример нахождения суммы элементов, расположенных ниже главной диагонали (рисунок 14):

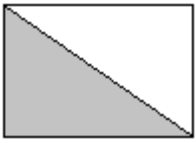


Рис. 14. Элементы ниже главной диагонали

```
int sum=0;
for(int i=0; i<n; i++)
    for(int j=0; j<i+1; j++)
            sum += a[i, j];
```

Пример нахождения суммы элементов, расположенных выше главной диагонали (рисунок 15):

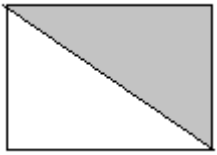


Рис. 15. Элементы выше главной диагонали

```
for(int i=0; i<n; i++)
    for(int j=i; j<n; j++)
            sum += a[i, j];
```

Пример нахождения суммы элементов, расположенных выше побочной диагонали (рисунок 16):

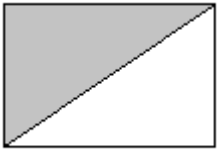


Рис. 16. Элементы выше побочной диагонали

```
for(int i=0; i<n; i++)
    for(int j=0; j<n-i; j++)
            sum+=a[i, j];
```

Пример нахождения суммы элементов, расположенных ниже побочной диагонали (рисунок 17):

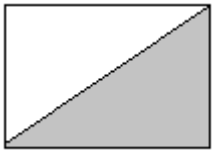


Рис. 17. Элементы ниже побочной диагонали

```
for(int i=0; i<n; i++)
    for(int j=n-i-1; j<n; j++)
            sum+=a[i, j];
```

Пример нахождения суммы элементов, расположенных в верхней четверти матрицы (рисунок 18):

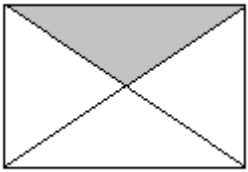


Рис. 18. Элементы в верхней четверти

```
for(int i=0; i<(n+1)/2; i++)
    for(int j=i; j<n-i; j++)
            sum+=a[i, j];
```

Пример нахождения суммы элементов, расположенных в нижней четверти матрицы (рисунок 19):

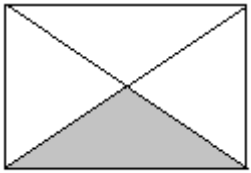


Рис. 19. Элементы в нижней четверти

```
for(int i=n/2; i<n; i++)
    for(int j=n-i-1; j<i+1; j++)
            sum+=a[i, j];
```

Пример нахождения суммы элементов, расположенных в левой четверти матрицы (рисунок 20):

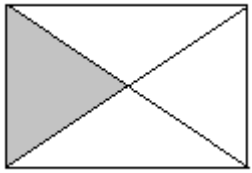


Рис. 20. Элементы в левой четверти

```
for(int i=0; i<(n+1)/2; i++)
    for(int j=i; j<n-i; j++)
            sum+=a[j, i];
```

Пример нахождения суммы элементов, расположенных в правой четверти матрицы (рисунок 21):

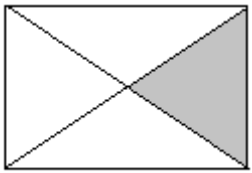


Рис. 21. Элементы в правой четверти

```
for(int i=n/2; i<n; i++)
        for(int j=n-i-1; j<i+1; j++)
                sum+=a[j, i];
```

Пример нахождения суммы элементов, расположенных на главной диагонали (рисунок 22):

Рис. 22. Главная диагональ матрицы

```
for(int i=0 ;i<n; i++)
        sum+=a[i, i];
```

Пример нахождения суммы элементов, расположенных на побочной диагонали (рисунок 23):

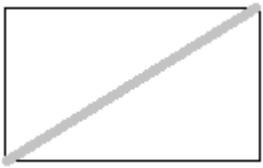


Рис. 23. Побочная диагональ матрицы

```
for(int i=0; i<n; i++)
    sum+=a[i, n-i-1];
```

Пример транспонирования матрицы – замены строк на столбцы:

```
for(int i=0;i<n; i++)
    for(int j=0; j<m; j++)
            b[j, i]=a[i, j];
```

Пример транспонирования матрицы самой в себе (только для квадратных матриц):

```
for(int i=0; i<n-1; i++)
for(int j=i+1; i<n; j++)
{
        temp=a[i, j];
        a[i, j]=a[j, i];
        a[j, i]=temp;
}
```

Пример умножения матриц (рисунок 24): $C_{ij}=\sum_{k=0}^{l} a_{ik}b_{kj}$

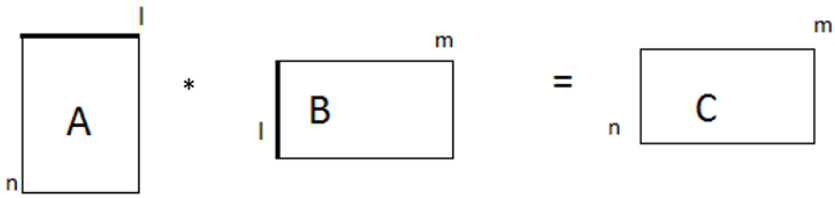


Рис. 24. Умножение матриц

```
int sum=0;
for(int i=0; i<n; i++)
for(int j=0; j<m; j++)
{
    sum=0;
    for(int k=0; k<l; k++)
    {
            sum = sum + a[i, k] * b[k, j];
    }
    c[i, j]=sum;
}
```

Пример сортировки выбором (сортировка вектора путем обмена максимального и текущего первого – рисунок 25):

3 5 7 1 9 4 8
9 5 7 1 3 4 8
9 8 7 1 3 4 5
9 8 7 5 3 4 1
9 8 7 5 4 3 1

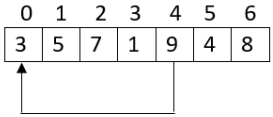


Рис. 25. Сортировка выбором

```
for(int k=0; k<n-1; k++)
{
    max=a[k];
    maxInd=k;
    for(int i=k+1; i<n; i++)
            if(a[i]>max)
            {
                    max=a[i];
                    maxInd=i;
            }
            temp=a[k];
            a[k]=a[maxInd];
            a[maxInd]=temp;
}
```

Пример сортировки «пузырьком» (попарное сравнение и обмен):

3 7 1 5 9
3 1 7 5 9
3 1 5 7 9
1 3 5 7 9

```
for(int k=0; k<n-1; i++)
for(int i=0; i<n-1-k; j++)
{
    if(a[i]>a[i+1])
    {
            temp=a[i];
            a[i]=a[i+1];
            a[i+1]=temp;
    }
}
```

Пример поиска индекса первого вхождения числа в вектор (если элемента нет, то -1):

```
int i=0;
while(i < n && a[i] != x)
i++;
if (i>n) i=-1;
```

Пример поиска индекса последнего вхождения числа в вектор (если элемента нет, то -1):

```
int i=0;
while(i >= 0 && a[i] != x)
i--;
```

5.4 Алгоритмы обработки текстовых строк

При работе с текстовыми строками следует обратить внимание на следующие моменты.

- Тип данных «string» или эквивалентно «String» – ссылочный тип.
- Существуют различные способы объявления и инициализации текстовых строк, не стоит путать пустые строки "" и null:

```
// Строка не инициализирована (null)
string s;

// Объявление и инициализация значением null
string snull = null;

// Объявление и инициализация
string s0="Строка";

// Пустая строка
string s1="";

// Пустая строка
string s2=string.Empty;
```

– Строка – это массив символов, поэтому к элементу строки (символу) можно обращаться по индексу, например:

```
s0[2] // Это символ «р».
```

– Возможно получить число символов в строке:

```
s0.Length // 6 символов;
```

– Операция объединения строк называется конкатенацией (это не операция сложения):

```
s = s0 + s1;
```

– Существует операция, проверяющая строку на равенство null или *"":*

```
string.IsNullOrEmpty(<Строка для проверки>)
```

6 КЛАССЫ И СТРУКТУРЫ

6.1 Описание класса

Класс – это программная сущность, описывающая тип данных и его реализацию. Реализация может быть частичной или полной. При создании экземпляра класса образуется объект.

Описание класса выглядит следующим образом:

[<Спецификаторы>] class <Имя класса> [:<Базовый класс>, <Интерфейсы>]
{
[<Тело класса>]
}

Имя класса должно быть уникальным в рамках текущего пространства имен. Базовый класс может быть только один, т.е. язык C# не поддерживает множественное наследование. Секция кода *<Интерфейсы>* может содержать несколько интерфейсов, тогда говорят о том, что класс реализует указанные интерфейсы.

6.2 Спецификаторы

Спецификаторы (или модификаторы) позволяют настроить уровень доступа класса и задать основные характеристики.

Спецификаторы доступа:

- *public* – общедоступный;
- *protected* – доступен только для наследников;
- *internal* – доступен в пределах сборки;

– *protected internal* – доступен в пределах сборки и в наследниках;
– *private* – доступен только в пределах класса.
– *private protected* – доступен в пределах класса или в наследниках, расположенных в пределах сборки.

Другие основные спецификаторы:

– *static* – статический класс (или член класса);
– *abstract* – абстрактный класс (или нереализованный член класса);
– *sealed* – запрещает наследование от класса (или переопределение члена класса);
– *new* – переопределяет класс (или член класса), унаследованный от предка.

Если класс объявлен как *static*, то он называется статическим. Все члены статического класса так же должны быть статическими. Нельзя создать экземпляр статического класса.

Если класс объявлен как *abstract*, то он называется абстрактным. Абстрактные классы допускают пропуски в реализации их членов. Нельзя создать экземпляр абстрактного класса.

Если класс объявлен как *sealed*, то он называется запечатанным. Запечатанный класс не может быть использован в качестве базового класса, т.е. от запечатанного класса нельзя наследоваться.

Если класс объявлен как *new*, то он переопределит класс, расположенный в базовом классе.

6.3 Основные члены класса

Основные члены класса:

– поля (field) – описывают внутреннее состояние объекта;
– константы (const) – неизменяемое значение;
– методы (method) – действия и поведение;
– конструкторы (ctor) – инициализатор объекта класса;

– свойства (property) – методы доступа к полям класса;
– события (event) – уведомления о возникновении ситуации;
– вложенные классы (class).

Помимо перечисленных членов класса, существуют другие, которые не рассматриваются в настоящем учебном пособии. Обработка событий рассматривается в разделе «Графический интерфейс пользователя».

Поля

Поля описывают внутреннее состояние класса. Описание поля выглядит следующим образом:

[<Спецификаторы>] <Тип> <Имя> [= <Начальное значение>];

Спецификаторы доступа описаны в предыдущем разделе. Помимо них, могут быть использованы следующие спецификаторы:

– *readonly* – поле только для чтения (его начальное значение можно задать в конструкторе);
– *volatile* – поле считывается из памяти каждый раз при обращении к нему (для совместного использования несколькими потоками).

Поскольку поля описывают внутреннее состояния класса, то плохим стилем программирования считается объявление публичных полей.

Пример объявления полей:

```
public class A
{
    private int _c = 6;
    private string _hello = “Hello!”;
    public static readonly float Pi = 3.14f;
}
```

Константы

Константы описывают значения, которые не могут измениться.

Пример объявления константы:

```
public class A
{
    public const double PI = 3.14;
}
```

Отличие констант от полей, объявленных как *static readonly,* состоит в том, что значения констант подставляются компилятором в итоговый код, а значения *static readonly* считываются из памяти.

Методы

Методы представляют собой конструкции кода, предназначенные для выполнения каких-либо действий:

```
[<Спецификаторы>] <Возвращаемый тип> <Имя метода> ([<Список параметров>])
{
    [<Тело метода>]
}
```

Для возврата значения из метода используется ключевое слово «*return [<Возвращаемое значение>]*;». Если метод не возвращает значения, то <Возвращаемый тип> указывается как *void*. Имя метода и список параметров называется сигнатурой метода. При этом важны не имена параметров, а их тип и порядок следования.

Входные параметры метода могут иметь различный тип (рисунок 26): значение, ссылка (ref), выходной (out).

Параметр, передаваемый в метод без модификаторов, копирует своё значение для передачи методу. Параметр, объявленный как ссылка, копирует ссылку на своё значение для передачи методу. Зна-

чение параметра, объявленного как выходной, должно быть обязательно задано внутри метода.

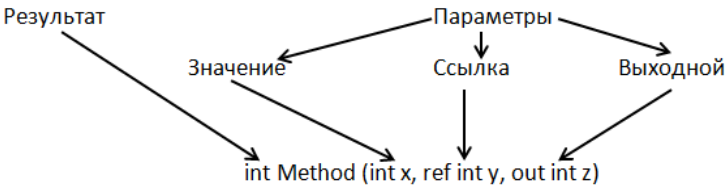


Рис. 26. Входные параметры метода

Спецификаторы доступа описаны в предыдущем разделе. Другие спецификаторы, используемые при объявлении методов:

- *virtual* – метод можно переопределять в потомках;
- *abstract* – метод не имеет реализации, его обязательно необходимо переопределить в потомке;
- *override* – переопределяет абстрактный или виртуальный метод базового класса;
- *sealed* – запрещает переопределять метод.

Пример объявления метода:

```
int Method (int x, ref int y, out int z)
{
    int res = x+y; // Выполняется сложение
    //res = res + z; // Не компилируется
    x = 371;  // Не изменяет значение x в вызывающем коде
    y = 173; // Изменяет значение y в вызывающем коде
    z = 500; // z обязательно необходимо присвоить значение
    return res; // Возвращает результат сложения x и y
}
```

Пример вызова метода:

```
int a = 700;  // a необходимо обязательно инициализировать
int b; // b инициализировать необязательно
```

```
// Вызов метода и получения результата в переменную res:
int res = Method (500, ref a, out b);
int x = 100;
// Вызов метода без сохранения результата:
Method (x, ref a, out b);
```

Конструкторы

Конструктор – инициализатор объекта класса, выполняется во время создания класса. Имя конструктора совпадает с именем класса. Конструктор не возвращает значения (даже void). Класс может содержать описание нескольких конструкторов, но с разными сигнатурами. Если в классе нет описания конструктора, то компилятор создает конструктор по умолчанию, который присваивает полям значения по умолчанию.

Конструктор может иметь входные параметры. К конструкторам можно применить спецификаторы доступа. Конструктор может быть статическим, к статическим конструкторам не применяются спецификаторы доступа.

Описание конструктора похоже на описание метода, за исключением отсутствия возвращаемого значения:

```
[<Спецификаторы>] <Имя конструктора> ([<Список параметров>])
{
    [<Тело конструктора>]
}
```

Пример объявления конструкторов:

```
class PointXY
  {
    // Поля
    private int _x = 0;
    private int _y = 0;
```

```
    // Конструкторы
    public PointXY()
    {
       _x = 0;
       _y = 0;
    }

    public PointXY(int x, int y)
    {
       /* Ключевое слово this – ссылка на текущий объект */
       this._x = x;
       this._y = y;
    }

     // Конструктор вызывает другой конструктор
     // this – вызов конструктора
    public PointXY(int x):this(x,0)
    {

    }
}
```

Пример использования конструкторов:

```
PointXY p1 = new PointXY(); // Конструктор по умолчанию
PointXY p2 = new PointXY(5, 7); // с 2-мя параметрами
PointXY p3 = new PointXY(3); // с 1 параметром
```

Пример вызова конструктора базового класса:

```
class PointXYZ :PointXY
  {
    private int _z = 0;
```

```
    public PointXYZ(int x, int y, int z):base(x, y)
    {
        this._z = z;
    }
}
```

Свойства

Свойства обеспечивают специальный синтаксис для объявления методов доступа к полям класса.

Ниже проиллюстрирован классический подход в объявлении методов для доступа к полям класса.

Пример методов доступа к полям класса:

```
class PointXY
{
    // Поля
    private int _x = 0;
    private int _y = 0;

    // Конструкторы
            ...

    // Свойства
            ...

    // Методы
            ...

    // Метод доступа для получения значения x
    public int GetX()
    {
        return this._x;
    }
```

```
    // Метод доступа для задания значения x
    public void SetX(int x)
    {
        this._x = x;
    }
}
```

В случае использования свойств два метода *GetX и SetX* заменятся следующей конструкцией:

```
    public int X
    {
        get { return this._x; }
        set { this._x = value; }
    }
```

Пример вызова методов и свойств:

```
PointXY p1 = new PointXY();
int x = p1.GetX(); // Получение значения с помощью метода
int x1 = p1.x; // Получение значения с помощью свойства
p1.SetX(10); // Задание значения с помощью метода
p1.X = 10; // Задание значения с помощью свойства
```

В случае, когда не требуется отдельная логика работа с полями, рекомендуется использовать автосвойства:

```
public int X { get; set; }
public int Y { get; set; }
```

Т.е. нет необходимости в объявлении полей. При компиляции свойства транслируются в поле и набор методов доступа к нему.

Модификаторы доступа могут применяться в свойствах:

public int Y { get; private set; }

Свойства могут быть описаны без секций set или get:

public int Mul { get {return _x*_y;} } // Значение нельзя задать
public int X { set { _x = value; } } // Значение нельзя получить

Свойства поддерживают дополнительные действия:

```
public int X
    {
       get { return _x; }
       set
       {
          if (value >= 0) _x = value;
          else _x = 0;
       }
    }
```

6.4 Правила именования

Рекомендуется соблюдать следующие правила именования:

- Имена классов – «ЭтоИмяКласса».
- Имена полей – «_имяПоля».
- Имена свойств – «ИмяСвойства».
- Имена методов – «ИмяМетода».
- Имена параметров – «имяПараметра».
- Имена локальных переменных – «имяПеременной».
- Имена констант – «ИМЯ_КОНСТАНТЫ».
- Переменные, используемые в циклах: i, j, k, t.
- Не рекомендуется использовать имена i, j, k, t для именования переменных и параметров.

– Чем больше область действия, тем понятнее и длиннее имя.
– Все поля рекомендуется делать приватными или защищенными;
– Не использовать числа, строки, символы в явном виде (объявлять константы или переменные с понятным именем) – так называемые «магические числа» не должны встречаться в коде.
– Инициализировать переменные при объявлении, объявлять их как можно ближе к месту использования.
– Соблюдать выравнивание кода относительно друг друга, так называемая «лесенка».
– Располагать члены класса в следующем порядке: поля, конструкторы, свойства, методы.

6.5 Структуры

Структура – это тип значения, который используется для инкапсуляции связанных данных, небольших по размеру. Структуры не наследуются, но могут реализовывать интерфейсы. Структуры размещаются в стеке, а не в куче, что обеспечивает экономию памяти на ссылках и времени доступа к ним.

Например, при объявлении массива из 2000 простых объектов PointXY потребуется выделить память для хранения ссылок на каждый объект, размещенный в куче. Использование структуры позволит сократить затраты памяти и увеличить скорость работы. Следует помнить, что размер стека не позволяет размещать в нем большое количество данных, поэтому если структура не является компактной, следует предпочесть использование класса.

Пример объявления структуры:

```
struct MyStruct
{
```

```
    int _x;
    int _y;

    public MyStruct(int x, int y)
    {
            _x = x;
            _y=y;
    }
}
```

Пример использования структуры:

```
MyStruct ms=new MyStruct();
MyStruct ms1; // Эквивалентно ms.
MyStruct ms2=new MyStruct(1,2);
MyStruct ms3=ms2; // Копирование значений, не ссылок
```

7 СТАНДАРТНЫЕ КЛАССЫ

7.1 Исключения: правила обработки, отлов, выброс исключений

При появлении ошибок в программе во время выполнения происходит выброс исключения (Exception).

Примеры известных исключений:

- DivideByZeroException – возникает при делении на 0;
- IndexOutOfRangeException – индекс находится вне границ массива.

В зависимости от типа исключения, они могут или не могут быть обработаны программой. Пример исключения, которое не может быть обработано – StackOverflowException – переполнение стека.

Обработка исключений осуществляется в соответствии со следующим шаблоном:

```
try
{
<Защищаемый код>
}
catch (<Имя исключения>)
{
<Обработчик исключения>
}
finally
{
<Завершение>
}
```

Секция кода «try» должна быть объявлена обязательно. Секция кода «catch» может повторятся несколько раз, может отсутствовать. Секция кода «finally» может быть объявлена только один раз, либо отсутствовать. В обработчике исключений обязательно должна присутствовать либо секция «catch», либо секция «finally», либо обе.

При возникновении ошибки в защищаемом коде происходит последовательная проверка блоков «catch» сверху вниз: выполняется поиск первого блока, который может обработать возникшее исключение. Именно поэтому блоки «catch» должны располагаться от наиболее специфичных к наиболее общим – при указании общего блока первым он обработает все возникшие исключения. Секция кода «finally» выполнится в любом случае – не важно, возникла ошибка или нет.

Пример обработки исключительной ситуации:

```
float res = 0;
try
{
float x = GetX();
float y = GetY();
res = x/y;
}
catch (DivideByZeroException ex)
{
Console.WriteLine("Ошибка: деление на ноль!");
Console.WriteLine(ex.Message); // Отладочная информация
res = -1;
}
catch (Exception ex)
{
Console.WriteLine("Неизвестная ошибка!");
}
```

```
catch
{
Console.WriteLine("Ошибка вне среды исполнения!");
}
finally
{
Console.WriteLine("Этот блок выполнится всегда.");
}
Console.WriteLine(res);
```

Пример создания пользовательского исключения:

```
public class MyFirstException : Exception  {}
```

Пример выбора исключения:

```
throw new MyFirstException();  // throw – выброс исключения
```

7.2 Потоки данных: чтение и запись

Поток – последовательность байтов, используемая для записи и чтения из вспомогательного запоминающего устройства. Примеры вспомогательных устройств: сеть, оперативная память, каналы, сокеты, порты. Stream – абстрактный базовый класс для потоков.

У потока 3 основных операции:

- чтение (информация переносится из потока в структуру данных);
- запись (информация переносится из структуры данных в поток);
- поиск (поиск информации в потоке).

Основные потоки:

- FileStream – для записи в файлы;

– MemoryStream – для записи в оперативную память;
– NetworkStream – для записи в сетевой сокет;
– CryptoStream – для записи с шифрованием.

Средства чтения и записи в потоки:

– BinaryReader, BinaryWriter – для чтения и записи двоичных данных;
– TextReader, TextWriter – для чтения и записи символов и строк.

Классы для ввода/вывода информации определены в пространстве имен System.IO.

7.3 Работа с файлами и каталогами

Файл – упорядоченная и именованная последовательность байтов, имеющая постоянное хранилище. Примеры постоянного хранилища – жесткий диск, флэш-карта.

В пространстве имен System.IO определены классы для ввода/вывода информации.

Основные классы для работы с файлами: File (статический) и FileInfo (экземплярный). Классы позволяют выполнить создание, копирование, удаление, перемещение, открытие файлов.

Основные классы для работы с каталогами: Directory (статический) и DirectoryInfo (экземплярный). Классы позволяют выполнить создание, копирование, удаление, перемещение, открытие файлов.

Основной класс для работы с путями к файлу или каталогу – Path.

8 ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

8.1 Основные этапы визуального программирования

Визуальное программирование – разработка программы через визуальные конструкторы, содержащие графические объекты.

Процесс визуального программирования включает два основных этапа:

1. Проектирования пользовательского интерфейса – выбор необходимых для реализации пользовательского интерфейса компонентов из библиотеки графических компонентов и элементов управления, настройка их свойств: расположение, размеры, подписи, цвета и др.

2. Разработка кода для обработчиков событий – определение поведения программы при возникновении того или иного события: нажатие кнопки, ввод текста, движение мыши и др.

8.2 Окна, графические компоненты

При разработке графических пользовательских интерфейсов для программ, функционирующих под управлением операционной системы Windows, за базовый компонент принимается окно (или форма).

Создание графического интерфейса программ с помощью технологии Windows Forms, встроенной в .NET Framework, подразумевает выбор из следующих компонентов:

- Form – окно, имеющее заголовок и настраиваемые кнопки: свернуть, развернуть, закрыть;
- Button – кнопка, имеющая заголовок;
- TextBox – поле ввода текста;
- Label – не редактируемая надпись;

– MenuStrip – меню;
– ToolStrip – панель инструментов;
– StatusStrip – панель статуса;
– RadioButton – кнопка в виде круга с точкой для выбора одного возможного варианта из нескольких;
– CheckBox – кнопка в виде квадрата с галочкой для выбора нескольких возможных вариантов;
– ComboBox – выпадающее меню для выбора одного варианта из нескольких с возможностью поиска;
– TabControl – компонент для управления вкладками;
– NumericUpDown – компонент для выбора числовых значений;
– DateTimePicker – компонент для выбора даты и времени;
– DataGridView – таблица для вывода данных;
– Panel – панель для группировки элементов;
– GroupBox – панель для группировки элементов с заголовком (обычно в ней располагают несколько RadioButton).

Основные графические компоненты, используемые при проектировании пользовательского интерфейса, приведены на рисунке 27.

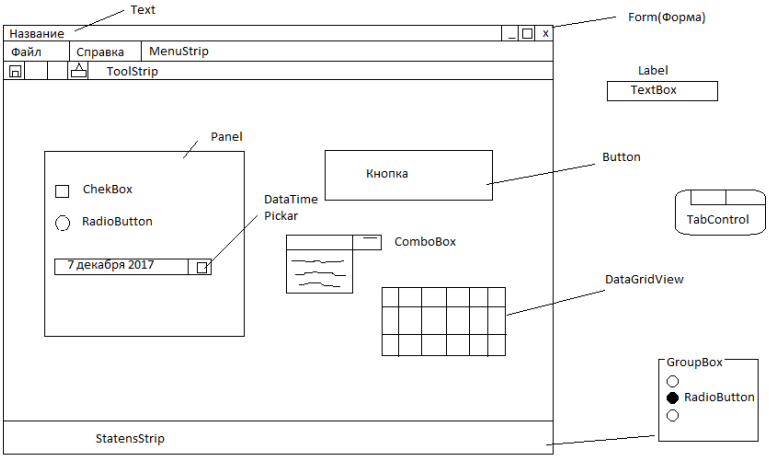


Рис. 27. Основные графические элементы управления

8.3 Взаимодействие с пользователем

Взаимодействие с пользователем в графическом интерфейсе основано на механизме событий. Например, при нажатии пользователем кнопки на форме, клику кнопкой мыши по какому-либо компоненту, вводу символов с клавиатуры – происходит возникновение события, которые могут быть проигнорированы или обработаны. Текст программы, написанный в обработчике события, определяет поведение программы.

Пример обработки нажатия на кнопку, при котором показывается новое информационное окно:

```
// Создание кнопки (сгенерировано автоматически)
this.button1 = new System.Windows.Forms.Button();
//
// button1
//
// Задание свойств кнопки (сгенерировано автоматически)
this.button1.Location = new System.Drawing.Point(401, 84);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(75, 23);
this.button1.Text = "button1";
// Подписка на событие (сгенерировано автоматически)
this.button1.Click += new System.EventHandler(this.button1_Click);

// Обработчик нажатия на кнопку
private void button1_Click(object sender, EventArgs e)
{
        MessageBox.Show("Вы нажали на кнопку.");
}
```

9 ПРОЕКТИРОВАНИЕ И СИСТЕМНЫЙ АНАЛИЗ

В процессе проектирования системы необходимо определить классы предметной области, определить их атрибутный и функциональный состав, интерфейсы и иерархию наследования, установить отношения. Проект системы должен соответствовать решаемой задаче и учесть возможность изменения требований в процессе разработки. Необходимо избежать или свести к минимуму необходимость перепроектирования системы.

9.1 Свойства системы

Основные свойства системы:

- многомерность и иерархичность, обусловленная большим числом взаимосвязанных между собой элементов;
- эмерджентность;
- многофункциональность элементов;
- многокритериальность, обусловливаемая имманентностью (несовпадением) целей отдельных элементов;
- сложное (вероятностное и динамическое) поведение, проявляющееся во взаимосвязи подсистем и требующее обратной связи при управлении;
- сложностью информационных процессов;
- необходимостью высокой автоматизации управления.

9.2 Этапы проектирования системы

Проектирование систем подразделяют на следующие этапы:

- техническое задание;

– научно-исследовательская работа;
– эскизный проект;
– технический проект;
– рабочий проект;
– технология изготовления и испытания спроектированного объекта, внесения коррекции.

На этапе разработки технического задания решаются следующие задачи:

– поиск необходимой научно-технической информации;
– анализ выбранной информации, формулировка требований;
– перечисление функций;
– определение условий работоспособности и ограничений;
– определение требований к выходным параметрам;
– определение характеристик модулей;
– разработка алгоритмов.

На этапе научно-исследовательской работы необходимо решение следующих задач:

– формулирование критериев качества и управления;
– управление экспериментами;
– проведение экспериментов, обработка результатов;
– разработка математических моделей и их идентификация;
– формирование обобщенного критерия качества;
– решение задачи оптимизации (обобщенный критерий – целевая функция);
– поиск принципиальной возможности построения системы;
– разработка новых технических средств, в том числе средств контроля и измерений.

На этапе эскизного проектирования производится решение следующих задач:

– разрабатывается эскиз проектируемой системы с детальной разработкой ее возможностей;

– принимаются предварительные проектные решения, оформляются проектные документы;
– производятся расчеты для разработки проектных документов.

На этапе технического проектирования детализируются и уточняются решения, принятые при эскизном проектировании. Большинство решений и документов, принятых и сформированных на этапах эскизного и технического проектирования, используются только для выполнения рабочего проектирования.

На этапе рабочего проектирования основным видом выполняемых работ является оформление проектных решений в виде чертежей (диаграмм) и спецификаций к ним. Современные средства вычислительной техники позволяют полностью автоматизировать оформление чертежей и спецификаций.

При проектировании технологии производят:
– поиск и выбор исходной информации (об объекте, подлежащем изготовлению);
– анализ и обработку данных в целях определения маршрутов обработки, последовательности технологических операций и режимов их проведения, потребности в инструменте и измерительном оборудовании, в создании специальной оснастки;
– оформление соответствующей технологической документации.

Проектирование системы состоит из двух основных этапов:
– обоснование исходных данных (технического задания) для проектирования;
– проектирование системы для сформулированных исходных данных.

9.3 Принципы разработки системы

При разработке системы соблюдаются следующие принципы.

Принцип *оптимальности* предполагает оптимизацию решений, обеспечивающих максимальный, интенсивно увеличивающийся во

времени прирост промышленного потенциала на основе достижений науки и техники.

Принцип *агрегирования* позволяет осуществить членение сложной многоуровневой системы на компоненты с целью снижения размерности решаемых задач, а также поэтапного ввода объектов.

Принцип *управляемости* – возможность воздействовать на состояние системы.

Принцип *автоматизации* обеспечивает оперативную и качественную переработку информации с воздействием в необходимых случаях на объект управления.

Принцип *стандартизации* гарантирует информационное единство, совместимость, инвариантность, преемственность проектных решений и др.

Принцип *системного единства* состоит в том, что на всех стадиях создания, функционирования и развития системы целостность ее должна обеспечиваться связями между образующими компонентами, а также функционированием специальной подсистемы (системы) управления.

Принцип *развития* требует, чтобы любой объект разрабатывался и функционировал как развивающаяся система, в которой предусмотрена возможность совершенствования компонентов системы и связей между ними на основе принципов агрегирования и стандартизации.

Принцип *надежности* предполагает работоспособность объекта, построенного в общем случае из ненадежных элементов, за счет специальных средств стабилизации заданных характеристик надежности системы и ее элементов.

9.4 Принципы объектно-ориентированного проектирования

Существует 5 основных принципов проектирования в объектно-ориентированном проектировании (аббревиатура – SOLID).

Принцип *единственной обязанности* – каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс, все его сервисы должны быть направлены на обеспечение этой обязанности.

Принцип *открытости-закрытости* – программные сущности (классы, модули, функции и др.) должны быть открыты для расширения, но закрыты для изменения.

Принцип *подстановки* Барбары Лисков – функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.

Принцип *разделения интерфейсов* – много специализированных интерфейсов лучше, чем один универсальный.

Принцип *инверсии зависимостей* состоит из двух положений:

- модули верхних уровней не должны зависеть от модулей нижних уровней, оба типа модулей должны зависеть от абстракций;
- абстракции не должны зависеть от деталей, детали должны зависеть от абстракций.

Пример проектирования программы с использованием изложенных принципов приведен в Приложении Б.

10 ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ ПОДГОТОВКИ

10.1 Описание условия задания

Требуется написать программный код консольного приложения. В состав проекта консольного приложения входят два класса – *Program* и *PreTest*. Класс *Program* содержит статический метод *static void Main()*, который является точкой входа для приложения. Нестатический класс *PreTest* содержит следующие члены, специфицируемые индивидуальным вариантом студента (приведены ниже):

- поле со ссылкой на вектор (одномерный массив), содержащий элементы указанного типа;
- поле указанного типа;
- конструктор без параметров, который инициализирует поля указанными значениями;
- конструктор с двумя параметрами, инициализирующий передаваемыми параметрами поля класса;
- свойство, обеспечивающее указанный уровень доступа к вектору;
- общедоступный метод без параметров, не возвращающий значение и выполняющий указанный код;
- общедоступный метод без параметров, возвращающий значение указанного типа и выполняющий указанный код;
- общедоступный статический метод с указанным входным параметром, возвращающий значение указанного типа и выполняющий указанный код.

Метод *Main()* класса *Program* должен содержать код, который последовательно выполняет следующие действия:

– создает экземпляр класса *PreTest* с помощью конструктора без параметров;
– создает экземпляр класса *PreTest* с помощью конструктора с параметрами, используя в качестве параметров значения, вводимые пользователем;
– далее работа ведется с любым из созданных объектов;
– считывает ссылку на вектор из экземпляра класса *PreTest* в локальную переменную;
– вызывает метод экземпляра класса *PreTest*, не возвращающий значение;
– вызывает метод экземпляра класса *PreTest*, возвращающий значений, и выводит результат выполнения метода в консоль;
– вызывает статический метод класса *PreTest*, передавая в качестве входного параметра вектор, записанный ранее в локальную переменную, и выводит результат выполнения метода в консоль.

Обязательные требования к программному коду:

– вывод данных на экран сопровождается пояснениями;
– предотвращение возможных исключительных ситуаций и ошибок выполнения в штатном режиме работы;
– корректная обработка возникающих исключительных ситуаций;
– наличие комментариев в коду;
– адекватное именование переменных, членов класса и др.;
– соблюдение объектно-ориентированной парадигмы и положений структурного программирования;
– визуальное упорядочивание кода по уровням («лесенка»).

Приветствуется:

– оптимизация операций;
– контроль вводимых в приложение данных;
– использование методов в классе *Program* для структуризации метода *Main()*;

– правильное распределение кода по уровням (например, логика – в классе *PreTest*, работа с пользовательским интерфейсом (консолью) – в классе *Program*).

10.2 Варианты заданий

Вариант № 1	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0, 1, 1, 111}
Значение для инициализации поля в конструкторе без параметров	657
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Прибавить значение поля ко всем элементам вектора.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть сумму таких элементов вектора, которые меньше значения, записанного в поле.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 2	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1, 3, 6, 666}
Значение для инициализации поля в конструкторе без параметров	476
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Записать во все элементы вектора значение поля.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть сумму таких элементов вектора, которые больше или равны значению, записанному в поле.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число отрицательных элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 3	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0, 2, 4, 543}
Значение для инициализации поля в конструкторе без параметров	34
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить все значения в векторе, которые больше нуля.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть произведение таких элементов вектора, которые больше значения, записанного в поле.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число нулевых элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 4	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-36, 3, 35, 34}
Значение для инициализации поля в конструкторе без параметров	562
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Записать во все элементы вектора значение поля, поделенное на 2.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть среднее арифметическое элементов вектора.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число ненулевых элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 5	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0, 1, 5, 555}
Значение для инициализации поля в конструкторе без параметров	23
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить значения элементов в векторе, которые больше значения, записанного в поле.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть сумму таких элементов вектора, которые меньше значения, записанного в поле.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число отрицательных элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 6	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1, 3, 6, 4765}
Значение для инициализации поля в конструкторе без параметров	256
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить значения элементов в векторе, которые меньше либо равны значению, записанному в поле.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс последнего положительного элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число четных элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 7	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0, 1, 25, 7654}
Значение для инициализации поля в конструкторе без параметров	2456
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Удвоить значения элементов в векторе, которые больше 0.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс первого положительного элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число нечетных элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 8	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1, 2456, 6, 3565}
Значение для инициализации поля в конструкторе без параметров	346
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить значения элементов в векторе с четным индексом.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс последнего отрицательного элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число нечетных отрицательных элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 9	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0, 1, 54, 345}
Значение для инициализации поля в конструкторе без параметров	63
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Присвоить значение поля элементам в векторе с нечетным индексом.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс первого отрицательного элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число четных положительных элементов в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 10	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1, 3, 36, 345}
Значение для инициализации поля в конструкторе без параметров	346
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить значения элементов в векторе, кратных значению поля.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс последнего нулевого элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число положительных элементов, стоящих на четных местах, в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 11	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0, 1, 35, 467}
Значение для инициализации поля в конструкторе без параметров	53
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Изменить знак значения элементов в векторе с четным индексом.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс первого нулевого элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число отрицательных элементов, стоящих на четных местах, в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 12	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1, 3, 45, 643}
Значение для инициализации поля в конструкторе без параметров	62
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Удвоить значения элементов в векторе, которые не делятся на 2 без остатка.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть произведение нечетных элементов вектора.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число элементов, кратных 6, в векторе, передаваемом в метод.
Тип входного параметра	int[]
Тип выходного параметра	int

Вариант № 13	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0, 1, 245, 245}
Значение для инициализации поля в конструкторе без параметров	63
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Разделить на 3 значения элементов в векторе, которые делятся на 3 без остатка.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть произведение четных элементов вектора.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть новый вектор, полученный из исходного вектора, передаваемого в метод, выбирая элементы, больше 5.
Тип входного параметра	int[]
Тип выходного параметра	int[]

Вариант № 14	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1, 245, 6, 524}
Значение для инициализации поля в конструкторе без параметров	45
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Разделить на 3 значения элементов в векторе, которые делятся на 2 без остатка.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть сумму элементов вектора, которые меньше значения, записанного в поле, разделенного на 2.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть новый вектор, полученный из исходного вектора, умножая каждый элемент на 4.
Тип входного параметра	int[]
Тип выходного параметра	int[]

Вариант № 15	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0, 324, 562, 24}
Значение для инициализации поля в конструкторе без параметров	45
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Поменять знак элементов вектора на противоположный.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Умножить каждый элемент вектора на значение поля и вернуть сумму полученных произведений.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть новый вектор, полученный из исходного вектора, передаваемого в метод, выбирая элементы, меньше 5.
Тип входного параметра	int[]
Тип выходного параметра	int[]

Вариант № 16	
Поля	
Тип элементов вектора	int
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-452, 42, 856, 634}
Значение для инициализации поля в конструкторе без параметров	25
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Перемножить элементы вектора и записать полученное произведение в поле.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Перемножить элементы вектора, умножить полученное произведение на значение поля и вернуть 0.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть новый вектор, полученный из исходного вектора, меняя порядок следования элементов на обратный.
Тип входного параметра	int[]
Тип выходного параметра	int[]

Вариант № 17	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0f, 1f, 1f, 111f}
Значение для инициализации поля в конструкторе без параметров	657
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Прибавить значение поля ко всем элементам вектора.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть сумму таких элементов вектора, которые меньше значения, записанного в поле.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть число элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 18	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1f, 3f, 6f, 666f}
Значение для инициализации поля в конструкторе без параметров	476
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Записать во все элементы вектора значение поля.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть сумму таких элементов вектора, которые больше или равны значению, записанному в поле.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть число отрицательных элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 19	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0f, 2f, 4f, 543f}
Значение для инициализации поля в конструкторе без параметров	34
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить все значения в векторе, которые больше нуля.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть произведение таких элементов вектора, которые больше значения, записанного в поле.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть число нулевых элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 20	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-36f, 3f, 35f, 34f}
Значение для инициализации поля в конструкторе без параметров	562
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Записать во все элементы вектора значение поля, поделенное на 2f.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть среднее арифметическое элементов вектора.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть число ненулевых элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 21	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0f, 1f, 5f, 555f}
Значение для инициализации поля в конструкторе без параметров	23
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить значения элементов в векторе, которые больше значения, записанного в поле.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть сумму таких элементов вектора, которые меньше значения, записанного в поле.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть число отрицательных элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 22	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1f, 3f, 6f, 4765f}
Значение для инициализации поля в конструкторе без параметров	256
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить значения элементов в векторе, которые меньше либо равны значению, записанному в поле.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс последнего положительного элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число четных элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 23	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0f, 1f, 25f, 7654f}
Значение для инициализации поля в конструкторе без параметров	2456
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Удвоить значения элементов в векторе, которые больше 0f.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс первого положительного элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число нечетных элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 24	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1f, 2456f, 6f, 3565f}
Значение для инициализации поля в конструкторе без параметров	346
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить значения элементов в векторе с четным индексом.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс последнего отрицательного элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число нечетных отрицательных элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 25	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0f, 1f, 54f, 345f}
Значение для инициализации поля в конструкторе без параметров	63
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Присвоить значение поля элементам в векторе с нечетным индексом.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс первого отрицательного элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число четных положительных элементов в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 26	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1f, 3f, 36f, 345f}
Значение для инициализации поля в конструкторе без параметров	346
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Обнулить значения элементов в векторе, кратных значению поля.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс последнего нулевого элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число положительных элементов, стоящих на четных местах, в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 27	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0f, 1f, 35f, 467f}
Значение для инициализации поля в конструкторе без параметров	53
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Изменить знак значения элементов в векторе с четным индексом.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть индекс первого нулевого элемента.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть число отрицательных элементов, стоящих на четных местах, в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 28	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1f, 3f, 45f, 643f}
Значение для инициализации поля в конструкторе без параметров	62
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Удвоить значения элементов в векторе, которые не делятся на 2f без остатка.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть произведение нечетных элементов вектора.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть число элементов, кратных 6, в векторе, передаваемом в метод.
Тип входного параметра	float[]
Тип выходного параметра	int

Вариант № 29	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0f, 1f, 245f, 245f}
Значение для инициализации поля в конструкторе без параметров	63
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Разделить на 3f значения элементов в векторе, которые делятся на 3f без остатка.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть произведение четных элементов вектора.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть новый вектор, полученный из исходного вектора, передаваемого в метод, выбирая элементы, больше 5.
Тип входного параметра	float[]
Тип выходного параметра	float[]

Вариант № 30	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-1f, 245f, 6f, 524f}
Значение для инициализации поля в конструкторе без параметров	45
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Разделить на 3f значения элементов в векторе, которые делятся на 2f без остатка.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Вернуть сумму элементов вектора, которые меньше значения, записанного в поле, разделенного на 2.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть новый вектор, полученный из исходного вектора, умножая каждый элемент на 4.
Тип входного параметра	float[]
Тип выходного параметра	float[]

Вариант № 31	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{0f, 324f, 562f, 24f}
Значение для инициализации поля в конструкторе без параметров	45
Свойства	
Уровень доступа к вектору	Общедоступный, только чтение.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Поменять знак элементов вектора на противоположный.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Умножить каждый элемент вектора на значение поля и вернуть сумму полученных произведений.
Тип возвращаемого значения метода	float
Статический метод	
Код статического метода	Вернуть новый вектор, полученный из исходного вектора, передаваемого в метод, выбирая элементы, меньше 5.
Тип входного параметра	float[]
Тип выходного параметра	float[]

Вариант № 32	
Поля	
Тип элементов вектора	float
Тип поля	int
Значения по умолчанию для полей	
Значение для инициализации вектора в конструкторе без параметров	{-452f, 42f, 856f, 634f}
Значение для инициализации поля в конструкторе без параметров	25
Свойства	
Уровень доступа к вектору	Общедоступный, чтение и запись.
Метод без входных параметров, не возвращающий значение	
Код метода без параметров, не возвращающего значение	Перемножить элементы вектора и записать полученное произведение в поле.
Метод без входных параметров, возвращающий значение	
Код метода без параметров, возвращающего значение	Перемножить элементы вектора, умножить полученное произведение на значение поля и вернуть 0.
Тип возвращаемого значения метода	int
Статический метод	
Код статического метода	Вернуть новый вектор, полученный из исходного вектора, меняя порядок следования элементов на обратный.
Тип входного параметра	float[]
Тип выходного параметра	float[]

10.3 Тест для самоконтроля

Настоящий тест призван обеспечить не только контроль знаний, но и развить способность к саморазвитию и самообразованию, поскольку многие вопросы вызовут интерес и потребуют обращения к справочной литературе.

1. Особенность статических классов:

а) все члены класса должны быть статическими

б) только методы класса должны быть статическими

в) только поля класса должны быть статическими

г) только конструкторы должны быть статическими

2. Что будет выведено на экран в результате выполнения следующего кода?

```
int x=16;
int y=5;
Console.Write(x/y);
```

а) 3.2

б) 3

в) 3,2

г) 16

3. Что будет выведено на экран в результате выполнения следующего кода?

```
int x = 6;
Console.Write(x++);
```

а) 5

б) 6

в) 7

г) 8

4. Что будет выведено на экран в результате выполнения следующего кода?

```
int a=4;
```

int b=6;

b=a++;

Console.Writeline(++b);

а) 10

б) 6

в) 4

г) 5

5. Какой тип имеет переменная temp?

var temp = 14.55;

а) string

б) float

b) integer

г) double

6. Что будет выведено на экран в результате выполнения следующего кода?

int a =8;

int b= ++a;

if (a>5) b-=3;

else b=9;

Console.WriteLine(b);

а) 9

б) 6

в) 7

г) 8

7. Какая операция используется для проверки на равенство?

а) ==

б) =

в) <>

г) !=

8**. Как найти квадратный корень из числа x?**

а) nameof(x)

б) typeof(x)

в) Math.Sqrt(x)

г) Math.sqrt(x)

9. Назначение условных операторов?

а) Настройка условий использования программы

б) Ветвление программы

в) Взаимодействие с пользователем

г) Настройка условия запуска программы

10. Сколько раз выполнится тело следующего цикла?

```
int x= 1;
while (x++<5)
{
    if(x%2==0)
      x+=2;
}
```

а) 3

б) 1

в) 2

г) 5

11. Сколько раз выполнится тело следующего цикла?

```
int x=1;
for(;x<7;)
{
   x+=2;
}
```

а) 3

б) 2

в) 4

г) 7

12. Что будет выведено на экран в результате выполнения следующего кода?

```
int a=2;
do
{
  a+=3;
} while(a<4);
Console.Write(a);
```

а) 3
б) 4
в) 1
г) 5

13. Результат вычисления выражения a&&b является истиной, если:

а) и a, и b являются ложными
б) либо a, либо b истинно
в) и a, и b являются истинными
г) либо a, либо b ложно

14. Что будет выведено на экран в результате выполнения следующего кода?

```
int x=5; int y=12;
if(x>10 || y/x>1)   Console.Write(y-x);
else  Console.Write(y);
```

а) 12
б) 5
в) 7
г) 10

15. Если «a» истинно, а «b» ложно, чему равен результат вычисления выражения !(a&&b)?

а) не определен
б) истина
в) ложь
г) одновременно и ложь, и истина

16. Чему будет равно значение «x» после выполнения следующего кода?

```
int x=5;
int y=3;
x=(x>y)?y:x;
```

а) 2

б) 5

в) 3

г) 8

17. Сколько раз выполнится тело следующего цикла?

```
do {}
while(false);
```

а) 0

б) 1

в) 2

г) Получен бесконечный цикл

18. Какой тип нужно использовать, чтобы метод не возвращал значения?

а) bool

б) var

в) void

г) IntPtr

19. Что будет выведено на экран в результате выполнения следующего кода?

```
static int Vol(int x, int y=3, int z= 1)
{
  return x*y*z;
}
static void Main (string[] args)
{
 Console.WriteLine(Vol(2, 4));
}
```

а) 2
б) 4
в) 8
г) 1

20. Что будет выведено на экран в результате выполнения следующего кода?

```
static void Test(int x)
{
  x=8;
}
static void Main()
{
  int a=5;
  Test(a);
  Console.WriteLine(a);
}
```

а) Ошибка компиляции
б) 8
в) 5
г) 13

21. Что будет выведено на экран в результате выполнения следующего кода?

```
static void Print(int a)
{
 Console.WriteLine(a*a);
}
static void Print(double a)
{
 Console.WriteLine(a+a);
}
static void Main(string[] args)
{
```

```
  Print(3);
}
```

а) 9

б) 6

в) 3

г) Ошибка времени выполнения

22. Сколько циклов можно вложить в другой цикл?

а) Ни одного

б) Любое количество

в) Один

г) Только два

23. Что будет выведено на экран в результате выполнения следующего кода?

```
static int Test(out int x, int y=4)
{
  x=6;
  return x*y;
}
static void Main(string[] args)
{
   int a;
   int z=Test(out a);
   Console.WriteLine(a+z);
}
```

а) 30

б) 24

в) 6

г) 36

24. Какая память используется для размещения объектов?

а) Дек

б) Стек

в) Очередь

г) Куча

25. Инкапсуляция позволяет:

а) Присваивать значения переменным

б) Скрывать детали реализации класса

в) Объявлять метод

г) Разъединять массив на его элементы

26. Что из перечисленного является модификатором доступа?

а) internal

б) Main

в) privet

г) closed

27. Когда вызывается нестатический конструктор нестатического класса?

а) Никогда

б) При объявлении класса

в) При создании конструктора

г) При создании объекта класса

28. Что будет выведено на экран в результате выполнения следующего кода?

```
int[] arr={8, 2, 6};
int y=0;
foreach(int x in arr)
{
    y+=x/2;
}
Console.Write(y);
```

а) 8

б) 2

в) 6

г) 1

29. Сколько измерений имеет следующий массив?

int[, , ,] arr;

а) 3

б) 4

в) 5

г) 2

30. Массив является

а) Типом значения

б) Ссылочным типом

в) Посылочным типом

г) Бесславными типом

10.4 Комплексные задания

Ниже приведены задания, направленные на формирование навыков системного анализа, информационного моделирования и разработки.

Вариант 1

Разработать класс «Матрица». Класс должен содержать:

- закрытые поля;
- свойства (если необходимо);
- конструктор;
- метод, преобразующий матрицу к треугольному виду (методом Гаусса);
- метод вывода матрицы (по строкам и столбцам);
- метод инициализации матрицы случайным образом.

Размерность матрицы n, где n < 11.

Добавить другие методы, если необходимо.

Написать программу, позволяющую протестировать разработанные методы (создать меню для выбора действий).

Предусмотреть обработку исключений там, где это необходимо.

Вариант 2

Разработать класс «Матрица». Класс должен содержать:

- закрытые поля;
- свойства (если необходимо);
- конструктор;
- метод, реализующий сложение двух матриц;
- метод, реализующий вычитание двух матриц;
- метод, реализующий умножение двух матриц;
- метод вывода матрицы (по строкам и столбцам);
- метод инициализации матрицы случайным образом.

Размерность матрицы m x n, где m < 11 и n < 11. Организовать проверку, возможно ли сложить/вычесть/перемножить матрицы.

Добавить другие методы, если необходимо.

Написать программу, позволяющую протестировать разработанные методы (создать меню для выбора действий).

Предусмотреть обработку исключений там, где это необходимо.

Вариант 3

Разработать класс «Комплексное число» в тригонометрической форме представления. Класс должен содержать:

- закрытые поля;
- свойства (если необходимо);
- конструктор;
- метод, реализующий сложение двух чисел;
- метод, реализующий вычитание двух чисел;
- метод, реализующий умножение двух чисел;
- метод, реализующий деление двух чисел;
- метод вывода комплексного числа в тригонометрической форме.

Добавить другие методы, если необходимо.

Написать программу, позволяющую протестировать разработанные методы (создать меню для выбора действий).

Предусмотреть обработку исключений там, где это необходимо.

Вариант 4

Разработать класс «Студент». Класс должен содержать:

– Закрытые поля: фамилия, имя, отчество, дата рождения, факультет, курс, массив оценок;

– Свойства (если необходимо);

– Конструктор;

– Метод вычисления среднего балла;

– Метод вывода информации о студенте.

Написать программу, позволяющую протестировать разработанные методы, а также:

– создать массив из n человек;

– вывести полную информацию из массива на экран;

– вывести имена студентов заданного факультета;

– вывести имена студентов заданного курса.

Создать меню для выбора действий.

Добавить другие методы, если необходимо.

Предусмотреть обработку исключений там, где это необходимо.

Вариант 5

Создать класс «Man». Класс должен содержать следующие члены класса:

– Закрытые поля: имя, возраст (от 10 до 60), уровень здоровья (в процентах от 0 до 100), индикатор состояния объекта (жив или мертв), замок (Винтерфелл, Утес Кастерли, Ривердан).

– Конструктор без параметров, позволяющий создать экземпляр класса со значениями по умолчанию: имя Unknown, возраст – 10, здоровье – 50, замок выбрать случайным образом.

Конструктор с параметрами, позволяющий создать экземпляр класса с заданным именем и замком. Значения возраста и уровня здоровья генерировать случайным образом в указанных диапазонах.

По умолчанию каждый новый экземпляр класса считается живым.

– Свойство, позволяющее получить информацию о состоянии объекта (жив или мертв).

– Метод «Говорить». Возможны 4 типа сообщений: Привет, меня зовут …; Мне … лет; Я абсолютно здоров! (если уровень здоровья больше 50); Кажется, я заболел (если уровень здоровья меньше 50). Метод возвращает сообщение. Организовать случайный выбор произносимой фразы. Использовать оператор выбора switch.

– Метод «Идти». Если уровень здоровья экземпляра больше 50, то метод возвращает сообщение следующего формата: Имя_экземпляра гуляет по городу. Если уровень здоровья экземпляра меньше 30, то метод возвращает сообщение: Имя_экземпляра болен, он не может гулять по городу. Предусмотреть ситуацию, когда экземпляр класса умер и выдать сообщение о том, что он не может передвигаться.

– Метод «Убить». Индикатор состояния объекта – мертв.

Написать программу, позволяющую протестировать разработанные методы (создать меню для выбора действий), а также:

– создать массив из n человек;

– вывести полную информацию из массива на экран;

– организовать поиск людей, которые проживают в указанном замке, вывести их имена;

Добавить другие методы, поля, свойства, если необходимо.

Предусмотреть обработку исключений там, где это необходимо.

СПИСОК ЛИТЕРАТУРЫ

1 Акчурин, Э.А. Программирование на языке C# в Microsoft Visual Studio .NET [Текст] / Э.А. Акчурин. – Самара: ИУНЛ ПГУТИ, 2010. – 128 с.

2 Антонов, А.В. Системный анализ: учебник / А.В. Антонов. – М.: Инфра-М, 2017. – 368 с.

3 Басс, Л. Архитектура программного обеспечения на практике [Текст] / Л. Басс, П. Клементс, Р. Кацман. – СПб.: Питер. – 2006. – 576 с.

4 Бхаргава, А. Грокаем алгоритмы [Текст] / А. Бхаргава – СПб.: Питер, 2018. – 288 с.

5 Вирт, Н. Алгоритмы и структуры данных [Текст] / Н. Вирт. – СПб.: ДМК Пресс, 2016. – 272 с.

6 Глухих, И. Теория систем и системный анализ: учеб. пособие [Текст] / И. Глухих. – Тюмень: Тюменский государственный университет, 2019. – 148 с.

7 ГОСТ 2.105-95 Единая система конструкторской документации. Общие требования к текстовым документам [Текст]. – Введ. 1996-06-30. – М.: Издательство стандартов, 1995. – 31 с.

8 Кнут, Д.Э. Искусство программирования. Том 1. Основные алгоритмы / Д.Э. Кнут. – М.: Вильямс, 2017. – 720 с.

9 Макконелл, Дж. Анализ алгоритмов. Активный обучающий подход [Текст] / Дж. Макконелл. – М.: Техносфера, 2013. – 415 с.

10 Макконнелл, С. Совершенный код [Текст] / С. Макконнелл. – СПб.: БХВ, 2017. –896 с.

11 Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения [Текст] / Р. Мартин. – СПб.: Питер, 2018. – 352 с.

12 Николаев, Е.И. Объектно-ориентированное программирование [Текст] / Е.И. Николаев. – Ставрополь: СКФУ, 2015. – 225 с.
13 Объектно-ориентированный анализ и проектирование с примерами приложений [Текст] / Г. Буч, Р.А. Максимчук, М.У. Энгл, Б.Дж. Янг, Дж. Коналлен, К.А. Хьюстон. – М.: Вильямс, 2017. – 720 с.
14 Окулов, С.М. Программирование в алгоритмах [Текст] / С.М. Окулов. – М.: Бином, 2013. – 384 с.
15 Павловская, Т.А. C#. Программирование на языке высокого уровня [Текст] / Т.А. Павловская. – СПб.: Питер, 2014. – 432 с.
16 Портал разработчиков Microsoft Developer Network [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru.
17 Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2019. – 368 с.
18 Рефакторинг. Улучшение проекта существующего кода [Текст] / М. Фаулер, К. Бек, Дж. Брант, Д. Робертс. – СПб.: Вильямс, 2017. – 448 с.
19 Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# [Текст] / Дж. Рихтер. – СПб.: Питер, 2019. – 896 с.
20 Сообщество разработчиков программного обеспечения [Электронный ресурс]. – Режим доступа: https://stackoverflow.com.
21 СТО 02068410-004-2018 Общие требования к учебным текстовым документам [Текст]. – Самара: Самарский университет, 2018. – 36 с.
22 Структуры данных и алгоритмы [Текст] / А.В. Ахо, Дж.Э. Хопкрофт, Дж.Д. Ульман. – М.: Вильямс, 2018. – 400 с.
23 Тихомирова, О. Управление проектом. Комплексный подход и системный анализ [Текст] / О. Тихомирова. – М.: Инфра-М, 2018. – 300 с.

ПРИЛОЖЕНИЕ А

Решение задачи координатных четвертей

Ниже приведены различные варианты решения задачи определения координатных четвертей.

```
static string If3Math(int x, int y)
{
    var val = Math.Sign(x) + Math.Sign(y) * 3;
    var abs = Math.Abs(val);

    if (abs == 0)
    {
        return "Начало координат.";
    }

    if (abs == 1)
    {
        return "Ось X.";
    }

    if (abs == 3)
    {
        return "Ось Y.";
    }

    return 4 - abs / 2 - abs / val + " четверть.";
}

private static string If0Dictionary(int x, int y)
```

```
    {
        return new Dictionary<object, string>
        {
            { new { X = +1, Y = +1 }, "1 четверть" },
            { new { X = -1, Y = +1 }, "2 четверть" },
            { new { X = -1, Y = -1 }, "3 четверть" },
            { new { X = +1, Y = -1 }, "4 четверть" },
            { new { X = 0, Y = 0 }, "Начало координат" },
            { new { X = -1, Y = 0 }, "Ось X" },
            { new { X = +1, Y = 0 }, "Ось X" },
            { new { X = 0, Y = -1 }, "Ось Y" },
            { new { X = 0, Y = +1 }, "Ось Y" }
        }
            [new { X = Math.Sign(x), Y = Math.Sign(y) }];
    }
}

    static string If3Replace(int x, int y)
        {
            string result = "01234xy";

            if (x == 0 || y == 0)
            {
                result = result.Replace('1', ' ');
                result = result.Replace('2', ' ');
                result = result.Replace('3', ' ');
                result = result.Replace('4', ' ');
            }
            else
            {
```

```
    result = result.Replace('0', ' ');
    result = result.Replace('y', ' ');
    result = result.Replace('x', ' ');
}

if ((y >= 0 || x == 0) && !(x == 0 && y == 0))
{
    result = result.Replace('3', ' ');
    result = result.Replace('4', ' ');
    result = result.Replace('0', ' ');
}
else
{

    result = result.Replace('1', ' ');
    result = result.Replace('2', ' ');
    result = result.Replace('x', ' ');
    result = result.Replace('y', ' ');
}

if ( x*y >= 0 && x != 0  )
{
    result = result.Replace('2', ' ');
    result = result.Replace('4', ' ');
    result = result.Replace('y', ' ');
}
else
{
    result = result.Replace('1', ' ');
    result = result.Replace('3', ' ');
    result = result.Replace('x', ' ');
```

```
        }
        result = result.Trim();
        result = x + "," + y + "     " + result;
        return result;
    }

static string If4(int x, int y)
{
        if (x*y > 0 && x > 0)
        {
           return "1 четверть";
        }
        else
        {
           return "3 четверть";
        }

        if (x*y < 0 && x < 0)
        {
           return "2 четверть";
        }
        else
        {
           return "4 четверть";
        }

        if (x*y = 0 && x != 0)
        {
           return "Ось Y";
        }
        else
```

```
    if (x*y = 0 && y != 0)
    {
       return "Ось X";
    }
    return "Начало координат";
}
```

ПРИЛОЖЕНИЕ Б
Информационное моделирование программы «Крестики–Нолики»

Ниже приведено описание процесса системного анализ информации при программировании консольной игры «Крестики–Нолики».

Анализ предметной области и формирование ограничений:

- Минимальный размер поля: 3×3.
- Все поля квадратные: n×n.
- Максимальный размер поля:10×10.
- Число игроков: 2.
- Возможные варианты исхода: победа, поражение, ничья.
- Крестики ходят первые.

Определение функциональных возможностей программы:

- Игра для двух игроков.
- Игра в консольном режиме.
- Ввод данных осуществляется с клавиатуры.
- Возможность выйти из игры в любой момент времени.
- Визуализация состояния игрового поля после каждого хода.
- Отображение результата при окончании игры.
- Контроль вводимых символов.

Для реализации обозначенных требований разработан прототип программы, учитывающий функциональные возможности и ограничения:

```
// Игровое поле
public class Field
{
        public static readonly int MaxSize=10; // Макс. размер поля
        public static readonly int MinSize=3; // Мин. размер поля
```

```
        private int _size; // Текущий размер поля
        private int [,] _cells; // Поле: 0-ничего,1-крестик, 2-нолик

        public Field(int n)
        {
                Size=n;
                _cells = new int[Size, Size];
        }
        public int Size
        {
                get { return _size; }
                private set
                {
                        if(value >= MinSize && value <= MaxSize)
                        {
                                _size=value;
                        }
                        else
                        {
                                throw new
                                ArgumentOutOfRangeException();
                        }
                }
        }

public bool CanMove(int n, int m)
{
        // Проверка на возможность хода в указанную ячейку
}
public override string ToString()
{
```

```
        // Вывод игрового поля
}

// Закончилась ли игра?
public  bool IsInProgress
{
        get{ ... }
}

public void Move(int n, int m)
{
        // Выполнить ход
}

// Результаты игры
public GameResult  Result
{
        get{ ... }
}

class Program
{
        static void   Main()//Запуск игры
        {
                Console.WriteLine("Добро пожаловать в X-0");
                Console.WriteLine ("Допустим размер поля от {0}
                                до {1}", Field.MinSize, Field.MaxSize);
                Console.WriteLine ("Введите размер поля: ");
                int size=InputInt(Field.MinSize, Field.MaxSize);
                Field field =new Field(size);
                while(field.IsInProgress)
```

```
            {
                // Вывод текущего состояния
                Console.WriteLine(field.ToString());
                // Ввод данных о ходе с клавиатуры: n, m
                …
                // Осуществление хода по введённым данным
                if (!field.CanMove(n, m))
                {
                    // Ход в указанные ячейки недопустим.
                }
                else field.Move(n, m);
            }
            // Вывод результата игры field.Result

}

public static int InputInt(int min, int max)
{
    int res=0;
    string str=Console.ReadLine();
    while(!int.TryParse(str, out res))
    {
        Console.WriteLine (“Указано недопустимое
                                значение”);
        Console.WriteLine (“Допустимо значение в диапазоне
                            от {0} до {1}”, min, max);
        str= Console.ReadLine();
    }
    return res;
}
```

```
// Результаты игры.
public enum GameResult
{
        None=0;
        XWin=1;
        0Win=2;
        Draw=3;
}
```

Учебное издание

Головнин Олег Константинович,
Столбова Анастасия Александровна

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ С ЭЛЕМЕНТАМИ СИСТЕМНОГО АНАЛИЗА ИНФОРМАЦИИ

Учебное пособие

В авторской редакции

Компьютерная верстка Л.Р. Дмитриенко

Подписано в печать 25.09.2019. Формат 60x84 1/16.
Бумага офсетная. Печ. л. 8,25.
Тираж 50 экз. Заказ .

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086 Самара, Московское шоссе, 34.

Изд-во Самарского университета.
443086 Самара, Московское шоссе, 34.