Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

М. В. МОСКАЛЕВА, П. В. БЫЧКОВ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

Основы языка C#

Практическое пособие

для студентов специальностей
1-40 01 01 «Программное обеспечение
информационных технологий»,
1-40 04 01 «Информатика и технологии программирования»

Гомель
ГГУ им. Ф. Скорины
2019

УДК 004.432 (076)
ББК 32.973.22я73
М82

Рецензенты:
доктор физико-математических наук В. М. Селькин,
доктор физико-математических наук В. Н. Тютянов

Рекомендовано к изданию научно-методическим советом учреждения образования «Гомельский государственный университет имени Франциска Скорины»

Москалева, М. В.

М82 Программирование на языке C# : Основы языка C# : практическое пособие / М. В. Москалева, П. В. Бычков ; Гомельский гос. ун-т им. Ф. Скорины. – Гомель : ГГУ им. Ф. Скорины, 2019. – 47 с.

ISBN 978-985-577-520-2

Практическое пособие предназначено для оказания помощи студентам в овладении современным языком программирования C#. Излагается теоретический материал, даны практические задания и примеры их выполнения по написанию кода на языке C# и использования библиотек классов платформы .NET.

Адресовано студентам специальностей 1-40 01 01 «Программное обеспечение информационных технологий», 1-40 04 01 «Информатика и технологии программирования».

УДК 004.432 (076)
ББК 32.973.22я73

ISBN 978-985-577-520-2

Оглавление

Предисловие

Язык C# – один из современных и актуальных языков программирования. Был разработан группой инженеров под руководством Андерса Хейлсберга в компании Microsoft для платформы Microsoft .NET.

По сравнению с другими языками C# достаточно молодой. Первая версия языка вышла в феврале 2002 года вместе с релизом Microsoft Visual Studio .NET. Текущей версией языка, которая вышла 7 марта 2017 года вместе с Visual Studio 2017, является версия C# 7.0.

C# является языком с C-подобным синтаксисом. Язык имеет строгую статическую типизацию, поддерживает наследование, полиморфизм, перегрузку операторов, указатели на функции-члены классов, события, свойства, исключения, комментарии в формате XML.

Данное практическое пособие предназначено для оказания помощи студентам в овладении языком программирования C#. Излагается теоретический материал, даны практические задания и примеры их выполнения по основам программирования на C#.

Каждая тема пособия содержит краткие теоретические сведения и практические задания с примерами их реализации. Задание по каждой теме состоит из 30 вариантов, что позволяет обеспечить индивидуальное задание для каждого студента учебной группы.

Язык программирования C# изучается студентами 3 курса специальностей 1–40 01 01 «Программное обеспечение информационных технологий» и 1–40 04 01 «Информатика и технологии программирования» в рамках дисциплины «Современные языки программирования».

Тема 1. Основные определения. Разветвляющиеся вычислительные процессы

1.1. Переменные и константы

Для хранения данных в программе применяются переменные. ***Переменная*** представляет именованную область памяти, в которой хранится значение определенного типа. Каждая переменная должна быть определенного типа данных. *Формат объявления переменной:*

```
тип_переменной индефикатор [= значения];
```

Пример:

```
int b, c;
long e = 10;
float f = 5.5f; // чтобы записать число с плавающей точкой типа float, нужно после значения добавлять суффикс f.
```

Язык Си-шарп чувствительный к регистру символов. Переменные max и Max – это не одно и то же.

Константа – это переменная, значение которой не меняется за время ее существования. При объявлении константы она должна обязательно быть проинициализирована значением. *Формат объявления константы:*

```
const  тип_константы  индефикатор = значения;
```

Пример:

```
const int months = 12; // объявление константы
```

1.2. Типы данных

В C# имеются две общие категории встроенных типов, данных: типы значений (таблица 1) и ссылочные типы. Концептуально разница между ними состоит в том, что тип значения (value type) хранит данные непосредственно, в то время как ссылочный тип (reference type) хранит ссылку на значение. Эти типы сохраняются в разных местах памяти: типы значений сохраняются в области, известной как стек, а ссылочные типы – в области, называемой управляемой кучей.

Ссылочные типы данных:

– ***string***: хранит набор символов `Unicode`. Представлен системным типом `System.String`. Этому типу соответствуют символьные литералы.

```
string hello = "Hello";
```

```
string word = "world";
```

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

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

Таблица 1 – Типы значений

Тип	Область значений	Размер	Системный тип
sbyte	-128 до 127	8 бит	System.SByte
byte	0 до 255	8 бит	System.Byte
char	U+0000 до U+ffff	16 бит	System.Char
bool	true или false	1 байт*	System.Boolean
short	-32768 до 32767	16 бит	System.Int16
ushort	0 до 65535	16 бит	System.UInt16
int	-2147483648 до 2147483647	32 бит	System.Int32
uint	0 до 4294967295	32 бит	System.UInt32
long	-9223372036854775808 до 9223372036854775807	64 бит	System.Int64
ulong	0 до 18446744073709551615	64 бит	System.UInt64
float	±1,5*10-45 до ±3,4*1033	4 байта, точность — 7 разрядов	System.Single
double	±5*10-324 до ±1,7*10306	8 байтов, точность — 16 разрядов	System.Double
decimal	(-7,9 * 1028 до 7,9 * 1028) / (100–28)	16 байт, точность — 28 разрядов	System.Decimal

Использование суффиксов. При присвоении значений надо иметь в виду следующую тонкость: все вещественные литералы рассматриваются как значения типа `double`. И чтобы указать, что дробное число представляет тип `float` или тип `decimal`, необходимо к литералу добавлять суффикс: **F/f** – для `float` и **M/m** – для `decimal`.

```
float a = 3.14F;
float b = 30.6f;
decimal c = 1005.8M;
decimal d = 334.8m;
```

Подобным образом все целочисленные литералы рассматриваются как значения типа `int`. Чтобы явным образом указать, что цело-

численный литерал представляет значение типа `uint`, надо использовать суффикс **U/u**, для типа `long` – суффикс **L/l**, а для типа `ulong` – суффикс **UL/ul**:

```
uint a = 10U;
long b = 20L;
ulong c = 30UL;
```

1.3. Преобразование встроенных типов данных

Переменные одного типа можно преобразовывать в переменные другого типа. Преобразование бывает явным и неявным. Неявное преобразование выполняет компилятор. Явное преобразование выполняется программистом с прямым указанием типа, к которому нужно привести переменную. Для такого преобразования требуется наличие оператора преобразования: в круглых скобках перед переменной указывается тип данных, к которому эта переменная должна преобразовываться. Пример:

```
b = (short) a;
```

Существует ещё один способ преобразования данных с помощью класса **Convert**, в котором есть много статических методов (с префиксом **To**):

```
double x = Convert.ToDouble(text);
```

1.4. Комментарии

Комментарии являются немаловажной частью любого языка программирования, так как позволяют удобно пояснять различные участки кода. В C# используются традиционные комментарии в стиле C –однострочные (//...) и многострочные (/* . . . */).

1.5. Операции языка C#

В C# используется большинство операций, которые применяются и в других языках программирования. *Операции* представляют определенные действия над операндами – участниками операции. В качестве операнда может выступать переменная или какое-либо значение (например, число). Операции бывают унарными (выполняются над

одним операндом), бинарными – над двумя операндами и тернарными – выполняются над тремя операндами. Рассмотрим все виды операций.

Арифметические операции представлены в таблице 2.

Таблица 2 – Арифметические операции

Операция	Запись
Сложение	a + b
Вычитание	a - b
Деление	a / b
Умножение	a * b
Нахождение остатка от деления	a % b
Инкрементация (увеличение на 1)	++
Декрементация (уменьшение на 1)	--

Операторы +, -, * и / действуют так, как предполагает их обозначение. Их можно применять к любому встроенному числовому типу данных. Только при делении стоит учитывать, что если оба операнда представляют целые числа, то результат также будет округляться до целого числа:

```
double z = 10/4; //результат равен 2
```

Несмотря на то, что результат операции в итоге помещается в переменную типа `double`, которая позволяет сохранить дробную часть, в самой операции участвуют два литерала, которые по умолчанию рассматриваются как объекты `int`, то есть целые числа, и результат тоже будет целочисленный.

Для выхода из этой ситуации необходимо определять литералы или переменные, участвующие в операции, именно как типы `double` или `float`:

```
double z = 10.0/4.0; //результат равен 2.5
```

Инкрементация и декрементация может быть префиксной и постфиксной. При префиксной форме оператор стоит перед операндом, а при постфиксной – после.

Префиксная форма сначала увеличивает (уменьшает) значение, и после этого выполняются остальные действия, а при постфиксной форме наоборот – сначала выполняются все действия, а после увеличивается (уменьшается) значение. Везде, где можно использовать инкрементацию/декрементацию, стоит это делать, так как она работает быстрее оператора сложения/вычитания.

При выполнении сразу нескольких арифметических операций следует учитывать порядок действий. Приоритет операций от наивысшего к низшему следующий:

– инкремент, декремент;

– умножение, деление, получение остатка;

– сложение, вычитание.

Для изменения порядка следования операций применяются скобки (таблицы 3,4).

Таблица 3 – Операции отношения

Значение	Оператор
равенство	==
не равно	!=
сравнения	<,>,<=,>=

Таблица 4 – Логические операции

Значение	Оператор
логического умножения, или логическое И. Возвращает true, если оба операнда одновременно равны true.	&
логического сложения, или логическое ИЛИ. Возвращает true, если хотя бы один из операндов возвращает true.	|
исключающего ИЛИ. Возвращает true, если либо первый, либо второй операнд (но не одновременно) равны true, иначе возвращает false.	^
логического отрицания. Производится над одним операндом и возвращает true, если операнд равен false. Если операнд равен true, то операция возвращает false.	!
логического сложения. Возвращает true, если хотя бы один из операндов возвращает true.	||
логического умножения. Возвращает true, если оба операнда одновременно равны true.	&&

Операции сдвига. Операции сдвига производятся над разрядами чисел. Сдвиг может происходить вправо и влево:

– *x*<<*y* – сдвигает число *x* влево на *y* разрядов. Например, 4<<1 сдвигает число 4 (которое в двоичном представлении 100) на один разряд влево, то есть в итоге получается 1 000 или число 8 в десятичном представлении;

– *x*>>*y* – сдвигает число *x* вправо на *y* разрядов. Например, 16>>1 сдвигает число 16 (которое в двоичном представлении

10 000) на один разряд вправо, то есть в итоге получается 1 000 или число 8 в десятичном представлении.

Таким образом, если исходное число, которое надо сдвинуть в ту или другую строну, делится на два, то фактически получается умножение или деление на два. Поэтому подобную операцию можно использовать вместо непосредственного умножения или деления на два.

Особый класс операций представляют *поразрядные операции*. Они выполняются над отдельными разрядами числа. В этом плане числа рассматриваются в двоичном представлении, например, число 2 в двоичном представлении – это 10 и имеет два разряда, число 7 – 111 и имеет три разряда (таблица 5).

Таблица 5 – Поразрядные операции

Оператор	Характеристика
&(логическое умножение)	производится поразрядно, и если у обоих операндов значения разрядов равны 1, то операция возвращает 1, иначе возвращается число 0.
| (логическое сложение)	производится по двоичным разрядам, но теперь возвращается единица, если хотя бы у одного числа в данном разряде имеется единица.
^ (логическое исключающее ИЛИ)	производятся поразрядные операции, если у нас значения текущего разряда у обоих чисел разные, то возвращается 1, иначе возвращается 0.
~ (логическое отрицание или инверсия)	поразрядная операция, которая инвертирует все разряды: если значение разряда равно 1, то оно становится равным нулю, и наоборот.

Тернарный оператор (?) относится к числу самых примечательных в C#. Он представляет собой условный оператор. Ниже приведена общая форма этого оператора:

```
логическое выражение ? выражение1 : выражение2
```

Сначала вычисляется логическое выражение. Если оно истинно, то вычисляется выражение1, в противном случае – вычисляется выражение2. Например:

```
max = a > b ? a : b;
```

1.6. Класс Math

В классе **Math** собраны все основные тригонометрические функции, функция возведение числа в степень, нахождение квадратного корня и другие.

Для возведения числа в степень используется функция

```
Pow([число], [степень]);
```

```
a = Math.Pow(b, 2); // возводим переменную b в степень 2.
```

Для нахождения квадратного корня служит функция

```
Sqrt([число]);
```

```
a = Math.Sqrt(b);
```

Для нахождения косинуса и синуса используются `cos([угол в радианах])` и `sin([угол в радианах])` соответственно.

1.7. Простейший ввод – вывод

Чтобы получить данные, вводимые пользователем вручную (т. е. с консоли), применяются команды `Console.ReadLine()` и `Console.Read()`. Формат:

```
<строковая переменная> = Console.ReadLine();
```

Пример:

```
string s = Console.ReadLine();
```

Для того чтобы преобразовать данные к нужному типу, используются функции из структуры `Convert`.

```
y = Convert.ToDouble(Console.ReadLine());
```

Вывод на экран организуется функциями `Console.Write()` и `Console.WriteLine()`. Первая выводит строку, и курсор остается за последним выведенным символом строки. Вторая функция в конце вывода устанавливает курсор экрана в первую позицию следующей строки.

Заметим, что числовые данные и выражения вставляются в функции в качестве аргументов в естественном виде (без кавычек). Если требуется выводить строку-литерал, то такой литерал (по определению – это строка символов в кавычках) записывается в кавычках.

Числовые данные чаще требуют форматирования. Для форматирования вывода функции консольного вывода вызывают специальную функцию `String.Format()`. Сам же формат задается в управляющей строке функции вывода. Поэтому функция вывода будет в этом случае иметь вид:

```
Console.WriteLine("Управляющая строка", аргумент 1, аргумент 2, ..., аргумент n);
```

Структура *управляющей строки* такова:

```
<символы> <спецификация формата> ... <символы> <спецификация формата>
```

Спецификация формата следующая: {N,M:Axx}, где *N* указывает позицию аргумента в списке выводимых переменных (нумерация начи-

нается с 0). M в спецификации формата задает ширину области, в которую будет помещено форматированное значение. Например, выводится число 1 500. Если задать M = 6, то в область из шести позиций выведется четырехпозиционное число 1 500. Если M отсутствует или отрицательно, значение будет выравнено влево, в противном случае – вправо. Итак, например, для вывода аргумента 1 в управляющей строке можно задать информацию (управляющая строка сама пишется в кавычках и отделяется от списка аргументов запятой).

"Значение аргумента1 равно: {0,6}", аргумент1

В таблице 6 приведены поддерживаемые строки стандартных форматов.

Таблица 6 – Описатели формата вывода чисел

Формат	Тип форматируемого значения	Пример
C или c	Валюта	Console.Write("{0:C}", 2.5); Console.Write("{0:C}", -2.5); // $2.50 ($2.50)
D или d	Десятичный формат	Console.Write("{0:D5}", 25); // 00025
E или e	Инженерный формат	Console.Write("{0:E}", 250000); // 2.500000E+005
F или f	Формат с фиксированной точкой	Console.Write("{0:F2}", 25); Console.Write("{0:F0}", 25); // 25.00 25
G или g	Общий формат	Console.Write("{0:G}", 2.5); // 2.5
N или n	Целочисленный формат	Console.Write("{0:N}", 2500000); // 2,500,000.00
X или x	Шестнадцатеричный формат	Console.Write("{0:X}", 250); Console.Write("{0:X}", 0xffff); // FA FFFF

Axx является необязательной строкой форматирующих кодов, которые используются для управления форматированием чисел, даты и времени, денежных знаков и т. д.

1.8. Обработка исключительных ситуаций

Обработка исключений делается с помощью оператора `try`. Формат:

```
try
      {// участок, предназначенный для обработки ошибок}
cath ()
      { // обработчик исключений }
 [finally]
   {// участок завершения обработки исключений;
    // закрытие нужных файлов, разрыв соединений и т.п.}
```

Пример:

```
static void Main(string[] args)
{ string result = "";
  Console.WriteLine("Введите число:");
  try
   { int a = Convert.ToInt32(Console.ReadLine());
     result = "Вы ввели число " + a; }
  catch (FormatException)
   { result = "Ошибка. Вы ввели не число";   }
     Console.WriteLine(result);
     Console.ReadLine();  }
```

Приведем некоторые из часто встречаемых типов исключений:

– *Exception* – базовый тип всех исключений, блок catch, в котором указан тип Exception, будет «ловить» все исключения;

– *FormatException* – некорректный формат операнда или аргумента (при передаче в метод);

– *NullReferenceException* – в экземпляре объекта не задана ссылка на объект, объект не создан;

– *IndexOutOfRangeException* – индекс вне рамок коллекции;

– *FileNotFoundException* – файл не найден;

– *DivideByZeroException* – деление на ноль.

Существует также специальный оператор *throw* (выбросить), с помощью которого можно самому в программе получить исключительную ситуацию, если она возникает (исключительные ситуации генерируются разными средами вне программы (неявно)), а оператор *throw* генерирует такую ситуацию явно.

1.9. Условные операторы

Условные операторы служат для ветвления программы. В зависимости от некоторого условия выполняется тот или другой набор команд. В C# выделяют два условных оператора `if` и `swith`.

1.9.1. Условный оператор if

Условный оператор `if` используется для разветвления процесса вычислений на два направления. Формат оператора:

```
if ([условное выражение]) оператор 1;
[else оператор 2;]
```

Сначала вычисляется логическое выражение. Если оно имеет значение `true`, выполняется первый оператор, иначе – второй. После этого управление передается на оператор, следующий за условным. Ветвь `else` не является обязательной и может отсутствовать. Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок, иначе компилятор не сможет понять, где заканчивается ветвление. Если требуется проверить несколько условий, их объединяют знаками логических условных операций. Пример:

```
if (a<b && a>d) b++;
 else { b*=a; a=0;}
```

1.9.2. Оператор выбора switch

Оператор `swith` (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Формат оператора:

```
switch (выражение)
 { case значение1: {список операторов1; break;}
   case значение2: {список операторов2; break;}
       ...
   case значениеN: {список операторовN; break;}
   default: {список операторовn+1; break;}
 }
```

Выполнение оператора начинается с вычисления выражения. Тип выражения чаще всего целочисленный или строковый. Затем управление передается первому оператору из списка, помеченному константным выражением, значение которого совпало с вычисленным. Пример:

```
switch (op)
 {case '+': res=a+b; break;
      case '-': res=a-b; break;
 }
```

Каждая ветвь переключателя должна заканчиваться явным оператором перехода, а именно оператором `break, goto` или `return`. Подробнее об этих операторах написано в пункте 2.2.

1.10. Практические задания

Разработать консольное приложение, которое по введенному значению аргумента вычисляет значение функции, заданной в виде графика согласно своему варианту. Варианты заданий даны в таблице 7. Осуществить обработку исключительных ситуаций.

Пример выполнения задания. Пусть функция задана в виде графика на рисунке 1.

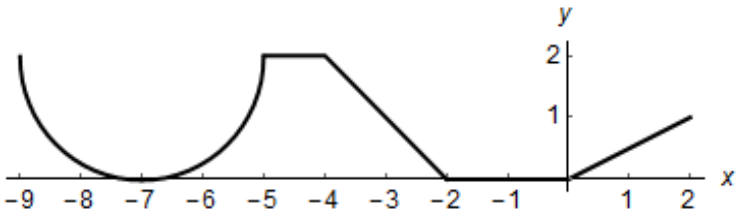


Рисунок 1 – График функции

Код программы:

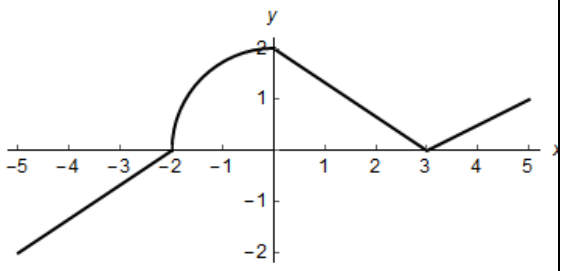
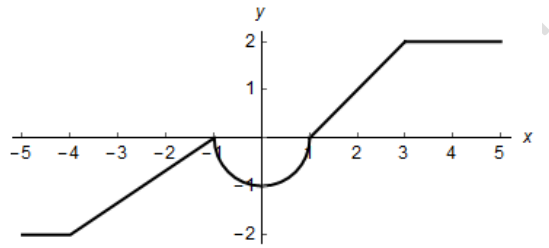
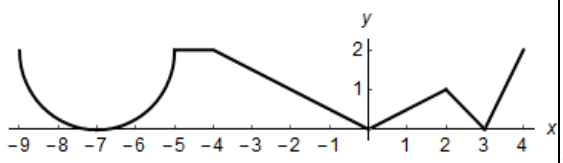
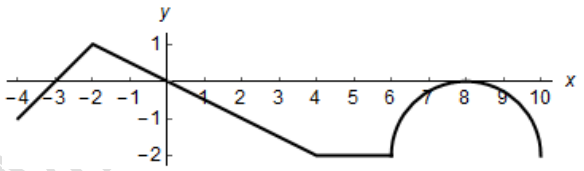
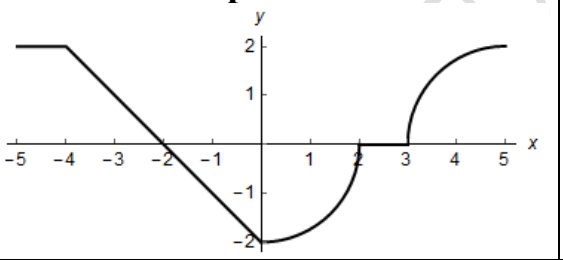
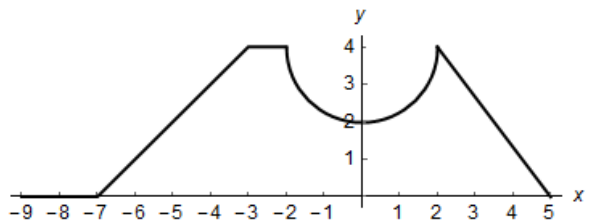
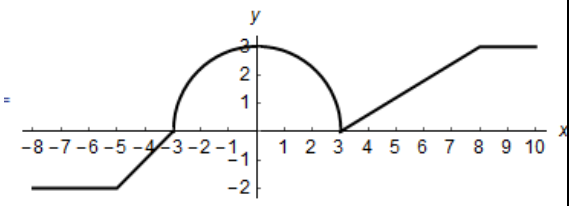
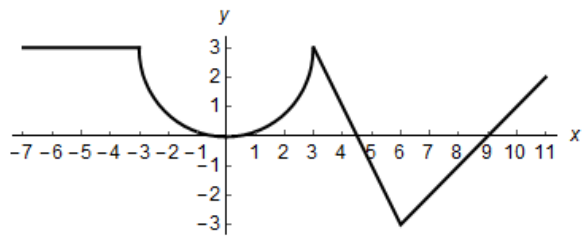
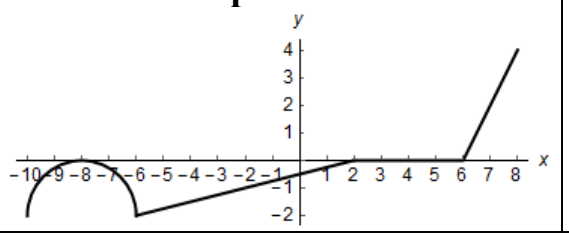
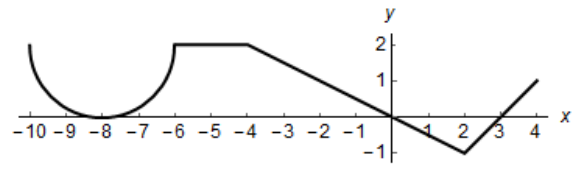
```
using System;
using System.Collections.Generic;
using System.Text;

namespace Lab2_1
{class Program
 {static void Main()
  {double R = 2,f;
   try
   {Console.WriteLine("Введите -9<=x<=2");
    double x = Convert.ToDouble(Console.ReadLine());
     if (x >= -9 && x <= -5)
      f = -Math.Sqrt(Math.Pow(R,2) -
Math.Pow(x+7,2)) + 2;
      else if (x > -5 && x <= -4) f = 2;
       else if (x > -4 && x <= -2) f = - x - 2;
        else if (x > -2 && x <= 0) f = 0;
         else f = x/2;
        Console.WriteLine("Для x=" + x);
        Console.WriteLine("f={0}", f);   }
   catch (FormatException)
    { Console.WriteLine("Неверный ввод"); }
    }}}
```

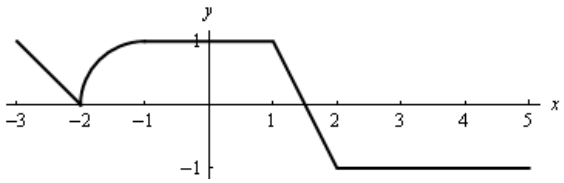
```
Введите -9<=x<=2
5
Для x=5
f=2,5
```

Рисунок 2 – Результат выполнения программы

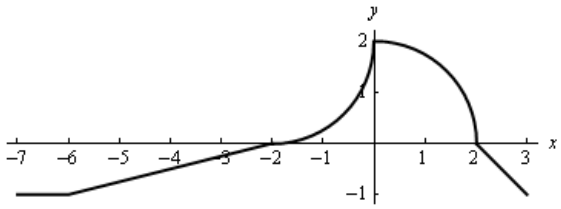
Таблица 7 – Варианты заданий

Вариант 1 	**Вариант 2** 
Вариант 3 	**Вариант 4** 
Вариант 5 	**Вариант 6** 
Вариант 7 	**Вариант 8** 
Вариант 9 	**Вариант 10** 

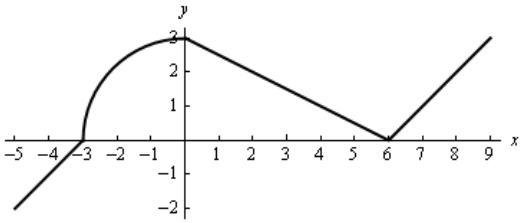
Вариант 11



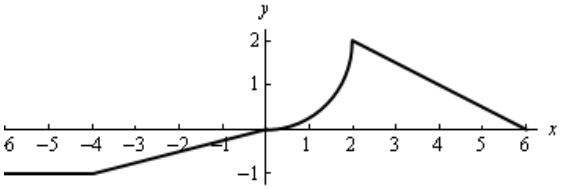
Вариант 12



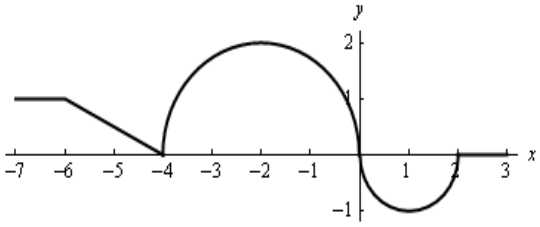
Вариант 13



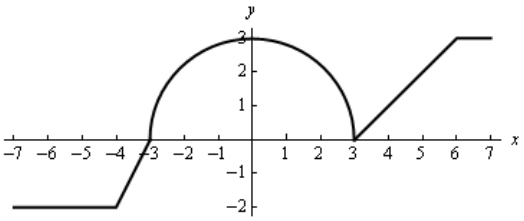
Вариант 14



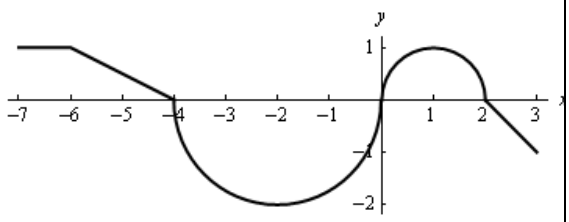
Вариант 15



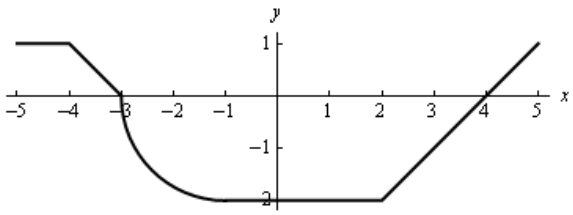
Вариант 16



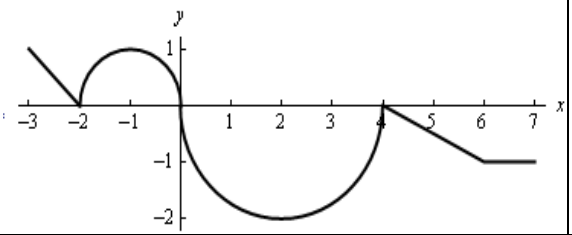
Вариант 17



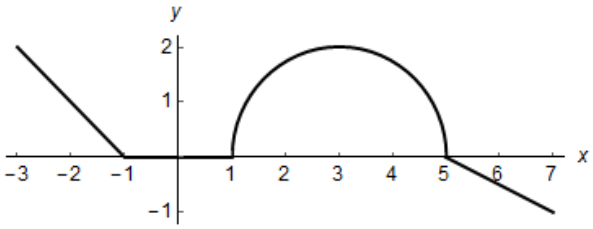
Вариант 18



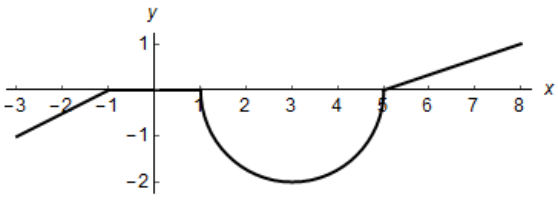
Вариант 19



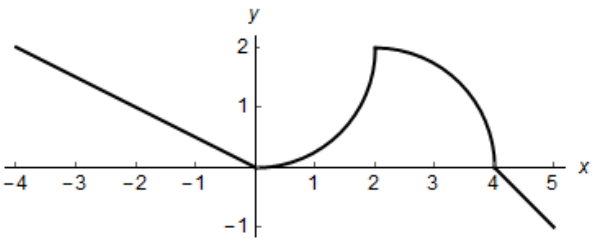
Вариант 20



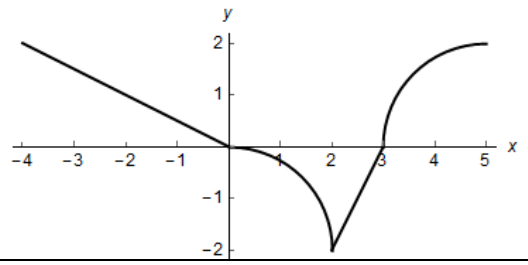
Вариант 21



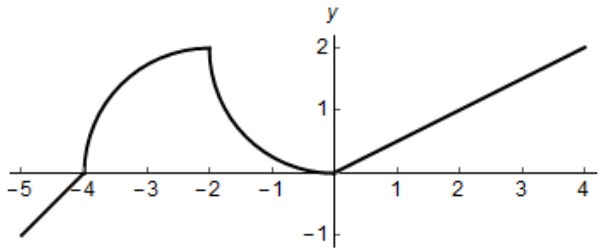
Вариант 22



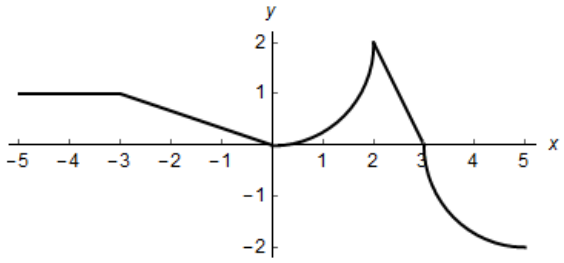
Вариант 23



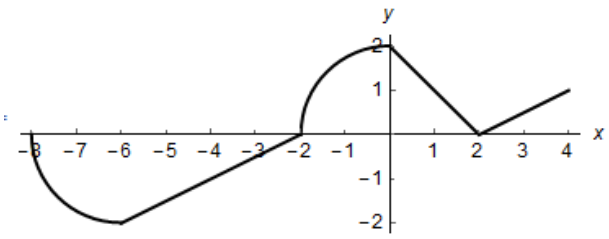
Вариант 24



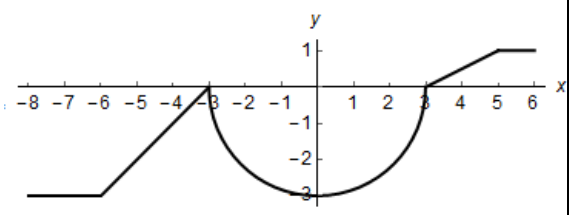
Вариант 25



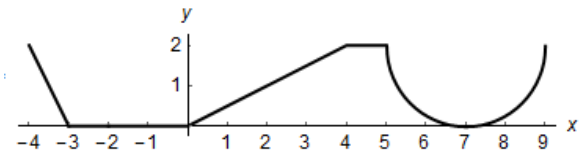
Вариант 26



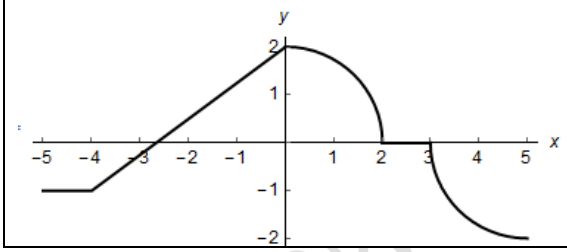
Вариант 27



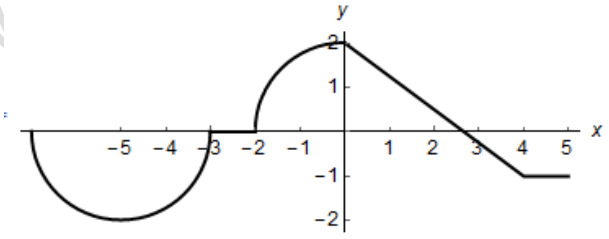
Вариант 28



Вариант 29



Вариант 30



Вопросы для самоконтроля

1. Дайте определения переменной и константы.
2. Перечислите типы данных.
3. Какие операции языка C# вы знаете?
4. Назовите команды простейшего ввода, вывода.
5. Какой оператор используется для обработки исключений?
6. Перечислите условные операторы.

Тема 2. Циклические процессы

2.1. Операторы цикла

Операторы цикла используются для вычислений, повторяющихся многократно. В C# имеется четыре вида циклов:

– цикл с предусловием **while.** *Формат описания оператора*:

```
while (выражение)
 оператор;
```

Пока результат вычисления выражения равен true, то будет выполняться простой или составной оператор (блок). Пример:

```
int a=1;
while (a<10)
 {Console.WriteLine (“a=”+a); a++;}
```

– цикл с постусловием **do.** *Формат описания оператора:*

```
do оператор;
while (выражение);
```

Сначала выполняется простой или составной оператор, образующий тело цикла, а затем вычисляется выражение. Если выражение истинно, тело цикла выполняется еще раз, и проверка повторяется. Пример:

```
char answer;
do {
 Console.WriteLine (“Какая буква загадана?”);
 answer=(char) Console.ReadLine();}
while (answer!= ‘y’);
```

– цикл с параметром **for.** *Формат описания оператора*:

```
for(инициализация; выражение; модификации)
 {операторы;}
```

Инициализация служит для объявления величин, используемых в цикле, и присвоения им начальных значений. *Выражение* типа `bool` определяет условие выполнения цикла: если результат равен true, цикл выполняется. *Модификации* выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла. Пример:

```
int s=0;
 for( int i=1; i<=100; i++)
  s+=i;
```

– цикл перебора **foreach.** Данный оператор применяется для перебора элементов в специальным образом организованной группе данных. *Формат описания оператора*:

```
foreach (тип имя in выражение )
 {тело цикла}
```

Имя задает локальную по отношению к циклу переменную, которая будет по очереди принимать все значения из группы массива *выражение* (в качестве выражения чаще всего применяется имя массива или другой группы данных). Пример вывода элементов массива на экран:

```
int a={1,2,3}
foreach (int x in a )
 Console.Writeln(x);
```

2.2. Операторы передачи управления

В C# есть пять операторов, изменяющих порядок выполнения вычислений:

– *goto* – оператор безусловного перехода;

– *break* – выход из цикла; переходит в точку программы, находящуюся непосредственно за оператором, внутри которого находится оператор break;

– *continue* – переход к следующей итерации в цикле; пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующее итерации;

– *return* – оператор возврата из функции; завершает выполнение функции и передает управление в точку ее вызова;

– *throw* – оператор генерации исключения.

2.3. Практические задания

Разработать консольное приложение, в котором вычислить и вывести на экран в виде таблицы значения функции, заданной с помощью ряда, на интервале $[a,b]$ с шагом h с точностью ε. Таблицу снабдить заголовком и шапкой. Каждая строка таблицы должна содержать значение аргумента, значение функции, подсчитанной через ряд, количество просуммированных членов ряда и значение функции.

Варианты заданий

1. $\ln\dfrac{x+1}{x-1}=2\sum\limits_{n=0}^{\infty}\dfrac{1}{(2n+1)x^{2n+1}}=2\left(\dfrac{1}{x}+\dfrac{1}{3x^3}+\dfrac{1}{5x^5}+...\right),|x|>1.$

2. $e^{-x}=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n x^n}{n!}=1-x+\dfrac{x^2}{2!}-\dfrac{x^3}{3!}+\dfrac{x^4}{4!}-...,|x|<\infty.$

3. $e^{x}=\sum\limits_{n=0}^{\infty}\dfrac{x^n}{n!}=1+x+\dfrac{x^2}{2!}+\dfrac{x^3}{3!}+\dfrac{x^4}{4!}+...,|x|<\infty.$

4. $\ln(x+1)=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n x^{n+1}}{n+1}=x-\dfrac{x^2}{2}+\dfrac{x^3}{3}-\dfrac{x^4}{4}-...,-1<x<1.$

5. $\ln\dfrac{1+x}{1-x}=2\sum\limits_{n=0}^{\infty}\dfrac{x^{2n+1}}{2n+1}=2\left(x+\dfrac{x^3}{3}+\dfrac{x^5}{5}+...\right),|x|<1.$

6. $\ln(1-x)=\sum\limits_{n=1}^{\infty}\dfrac{x^n}{n}=-\left(x+\dfrac{x^2}{2}+\dfrac{x^3}{3}+...\right),-1<=x<1.$

7. $\operatorname{arcctg}x=\dfrac{\pi}{2}+\sum\limits_{n=0}^{\infty}\dfrac{(-1)^{n+1}x^{2n+1}}{2n+1}=\dfrac{\pi}{2}-x+\dfrac{x^3}{3}-\dfrac{x^5}{5}-...,|x|<1.$

8. $\operatorname{arctg}x=\dfrac{\pi}{2}+\sum\limits_{n=0}^{\infty}\dfrac{(-1)^{n+1}}{(2n+1)x^{2n+1}}=\dfrac{\pi}{2}-\dfrac{1}{x}+\dfrac{1}{3x^3}-\dfrac{1}{5x^5}+...,x>1.$

9. $\operatorname{arctg}x=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n x^{2n+1}}{2n+1}=x-\dfrac{x^3}{3}+\dfrac{x^5}{5}-\dfrac{x^7}{7}+...,|x|<1.$

10. $\operatorname{arth}x=\sum\limits_{n=0}^{\infty}\dfrac{x^{2n+1}}{2n+1}=x+\dfrac{x^3}{3}+\dfrac{x^5}{5}+\dfrac{x^7}{7}+...,|x|<1.$

11. $\operatorname{arth}x=\sum\limits_{n=0}^{\infty}\dfrac{1}{(2n+1)x^{2n+1}}=\dfrac{1}{x}+\dfrac{1}{3x^3}+\dfrac{1}{5x^5}+...,x>1.$

12. $\operatorname{arctg}x=-\dfrac{\pi}{2}+\sum\limits_{n=0}^{\infty}\dfrac{(-1)^{n+1}}{(2n+1)x^{2n+1}}=-\dfrac{\pi}{2}-\dfrac{1}{x}+\dfrac{1}{3x^3}-\dfrac{1}{5x^5}+...,x<-1.$

13. $e^{-x^2}=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n x^{2n}}{n!}=1-x^2+\dfrac{x^4}{2!}-\dfrac{x^6}{3!}+\dfrac{x^8}{4!}-...,|x|<\infty.$

14. $\cos x=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n x^{2n}}{(2n)!}=1-\dfrac{x^2}{2!}+\dfrac{x^4}{4!}-\dfrac{x^6}{6!}+...,|x|<\infty.$

15. $\dfrac{\sin x}{x}=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n x^{2n}}{(2n+1)!}=1-\dfrac{x^2}{3!}+\dfrac{x^4}{5!}-\dfrac{x^6}{7!}+\ldots,|x|<\infty.$

16. $\ln x=2\sum\limits_{n=0}^{\infty}\dfrac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}}=2\left(\dfrac{x-1}{x+1}+\dfrac{(x-1)^3}{3(x+1)^3}+\dfrac{(x-1)^5}{5(x+1)^5}+\ldots\right),x>0.$

17. $\ln x=\sum\limits_{n=0}^{\infty}\dfrac{(x-1)^{n+1}}{(n+1)x^{n+1}}=\dfrac{x-1}{x}+\dfrac{(x-1)^2}{2x^2}+\dfrac{(x-1)^3}{3x^3}+\ldots,x>\dfrac{1}{2}.$

18. $\ln x=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n(x-1)^{n+1}}{(n+1)}=(x-1)-\dfrac{(x-1)^2}{2}+\dfrac{(x-1)^3}{3}-\ldots,0<x<2.$

19. $\sqrt{1+x}=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n(2n)!x^n}{(1-2n)n!^2\,4^n}=1+\dfrac{x}{2}-\dfrac{x^2}{8}+\dfrac{x^3}{16}-\ldots,|x|<1.$

20. $\dfrac{1}{1-x}=\sum\limits_{n=0}^{\infty}x^n=1+x+x^2+\ldots,|x|<1.$

21. $\sin x=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n x^{2n+1}}{(2n+1)!}=x-\dfrac{x^3}{3!}+\dfrac{x^5}{5!}-\dfrac{x^7}{7!}+\ldots,|x|<\infty.$

22. $\arcsin x=\sum\limits_{n=0}^{\infty}\dfrac{(2n)!x^{2n+1}}{(2n+1)n!^2\,4^n}=x+\dfrac{x^3}{6}+\dfrac{3x^5}{40}+\ldots,|x|<1.$

23. $\arccos x=\dfrac{\pi}{2}-\sum\limits_{n=0}^{\infty}\dfrac{(2n)!x^{2n+1}}{(2n+1)n!^2\,4^n}=\dfrac{\pi}{2}-\left(x+\dfrac{x^3}{6}+\dfrac{3x^5}{40}+\ldots\right),|x|<1.$

24. $\operatorname{sh} x=\sum\limits_{n=0}^{\infty}\dfrac{x^{2n+1}}{(2n+1)!}=x+\dfrac{x^3}{3!}+\dfrac{x^5}{5!}+\dfrac{x^7}{7!}+\ldots,|x|<\infty.$

25. $\operatorname{ch} x=\sum\limits_{n=0}^{\infty}\dfrac{x^{2n}}{(2n)!}=1+\dfrac{x^2}{2!}+\dfrac{x^4}{4!}+\ldots,|x|<\infty.$

26. $\sin x=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^{n+1}x^{2n-1}}{(2n-1)!}=x-\dfrac{x^3}{3!}+\dfrac{x^5}{5!}-\dfrac{x^7}{7!}+\ldots,|x|<\infty.$

27. $\operatorname{arsh} x=\sum\limits_{n=0}^{\infty}\dfrac{(-1)^n(2n)!x^{2n+1}}{(2n+1)n!^2\,4^n}=x-\dfrac{x^3}{6}+\dfrac{3x^5}{40}-\ldots,|x|<1.$

28. $\dfrac{1}{1+x}=\sum\limits_{n=0}^{\infty}(-1)^n x^n=1-x+x^2-\ldots,|x|<1.$

29. $2^x = \sum_{n=0}^{\infty} \frac{\ln^n 2}{(n)!} x^n = 1 + \frac{\ln 2}{1!} x + \frac{\ln^2 2}{2!} x^2 + ..., |x| < \infty.$

30. $(1+x)^\alpha = \sum_{n=0}^{\infty} \frac{\alpha(\alpha-1)...(\alpha-n+1)x}{(n)!} = 1 + \frac{\alpha}{1!} x + \frac{\alpha(\alpha-1)}{2!} x^2 + ..., |x| < 1.$

Пример выполнения задания.

Пусть функция задана в виде следующего ряда:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ..., |x| < \infty.$$

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Lab3_3
{class Program
 {static void Main()
  {Console.WriteLine("Введите точность");
   double ep = Convert.ToDouble(Console.ReadLine());
   Console.WriteLine("Введите начало отрезка");
   double a = Convert.ToDouble(Console.ReadLine());
   Console.WriteLine("Введите конец отрезка");
   double b = Convert.ToDouble(Console.ReadLine());
   Console.WriteLine("Введите шаг");
   double h = Convert.ToDouble(Console.ReadLine());
   double x = a, s,y;
   const int MaxIter = 500;
   bool done=true;
   int i;
   Console.WriteLine(" _____________________________");
   Console.WriteLine("|x|Сумма ряда|n|Зн. функции|");
   Console.WriteLine(" _____________________________");
    while (x <= b)
     {s = 1; y = s;
      done = true;
      for (i = 1; Math.Abs(s) > ep; i++)
       {s = s * (x / i);
        y = y + s;
        if (i > MaxIter) { done = false; break; }
        }
```

```
   if   (done==true)   Cosole.WriteLine("|{0,5}|{1,18}
|{2,5}|{3,18}|",x,y,i, Math.Exp(x));
    else   Console.WriteLine("|{0,5}|Ряд   расходится
|",x);
        x = x + h;
     }}}
```

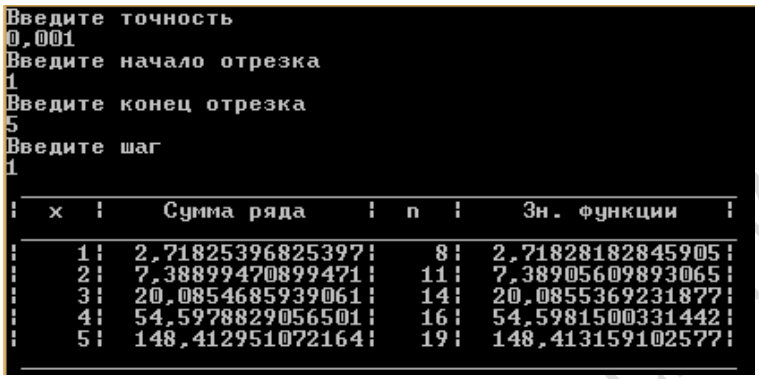


Рисунок 3 – Результат выполнения программы

Вопросы для самоконтроля

1. Назовите операторы циклов в C#.
2. Чем отличается цикл while от do while?
3. Назовите операторы передачи управления.
4. Назовите оператор возврата из функции.

Тема 3. Массивы

Массив – ограниченная совокупность однотипных величин. Массив относится к ссылочным типам данных, то есть располагается в динамической области памяти, поэтому создание массива начинается с выделения памяти под его элементы. Массивы подразделяются на одномерные и многомерные (прямоугольные и ступенчатые).

3.1. Одномерные массивы

Одномерный массив по-другому еще называется вектором, и для доступа к его элементам используется только один индекс. В C# объявление массива имеет такую структуру:

```
тип[] имя_массива = new тип[размер массива];
```

Пример:

```
int[] array = new int[5]; // создаем массив целых чисел
string[] seasons = new string[4] {"зима","весна", "лето","осень"}; //объявление массива строк и его инициализация значениями
```

Если происходит инициализация, оператор new можно упускать:

```
string[] seasons ={"зима","весна","лето","осень"};
```

Доступ к элементам осуществляется по индексу. Следует помнить, что индексация начинается с нуля – первый элемент массива имеет индекс 0, а последний *n*-1, где *n* – размер массива.

3.2. Прямоугольные массивы

Прямоугольный массив имеет более одного измерения. Чаще всего в программах используются двумерные массивы (матрицы). В матрице для доступа к элементам необходимо использовать два индекса.

Количеством индексов, используемых для доступа к элементам массива называется размерность массива. Варианты описания двумерного массива:

```
тип [,]имя;
тип [,]имя = new тип [размерность1, размерность2];
тип [,]имя = {{список инициализаторов1}, {список инициализаторов2}};
```

Примеры:

```
int[,] numbers1 = new int[2,2]; // объявление двумерного массива
int[,,] numbers2 = new int[2,2,3]; // объявление трехмерного массива
int[,] numbers3=new int[3,2] {{6,0},{5,7},{8,9}}; // инициализация двумерного массива
```

3.3. Ступенчатые массивы

Ступенчатый массив – это массив, состоящий из нескольких внутренних массивов, каждый из которых имеет свой размер. Формат описания:

```
тип [][]имя;
```

Под каждый из массивов, составляющих ступенчатый массив, память требуется выделять явным образом:

– `int [][] a = new int [3][]` // выделяем память под 3 строки;

```
a[0] = new int [5]; // выделяем память под 0-ю строку
a[1] = new int [3];
a[2] = new int [4];
```

– `int [][]a = {new int[5],new int[3],new int[4]};`

Доступ к элементам осуществляется по тому же принципу, как и с многомерными массивами, только тут уже участвуют две пары квадратных скобок:

```
a [0][1] = 5;
```

Пример вывода элементов ступенчатого массива:

```
foreach ( int []x in a )
{ foreah (int y in x)
  Console.Write (“ ” + y);
  Console.Writeln(); }
```

3.4. Класс System.Array

В данном классе содержатся свойства и методы для работы с массивами. Ниже приведены некоторые из них.

Свойства:

– `length` – количество элементов массива (по всем размерностям): `n=a.length;`

– `rank` – количество размерностей массива.

Методы:

– `IndexOf` – поиск первого вхождения элемента в одномерном массиве: `Array.IndexOf(a,4);`

– `LastIndexOf` – поиск последнего вхождения элемента в одномерном массиве;

– `Sort` – сортировка элементов одномерного массива.

3.5. Практические задания

Разработать три консольных приложения, реализующих задания по вариантам. Для задания 1 и 2 дан одномерный вещественный массив из *n* элементов, а для задания 3 дана вещественная прямоугольная матрица *n***m*. Обработать все возможные ситуации, возникающие при решении вашей задачи.

Варианты заданий

Вариант 1

1. Найти произведение элементов массива, расположенных между максимальным и минимальным элементами.

2. Отсортировать элементы массива по возрастанию.

3. Найти максимальное из чисел, встречающихся в заданной матрице более одного раза.

Вариант 2

1. Найти сумму элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

2. Отсортировать элементы массива по убыванию.

3. Определить количество столбцов, не содержащих ни одного нулевого элемента.

Вариант 3

1. Найти произведение элементов массива с четными номерами.

2. Найти сумму элементов массива, расположенных между первым нулевым и последним нулевым элементами.

3. Найти номер строки, в которой находится самая длинная серия одинаковых элементов.

Вариант 4

1. Найти сумму положительных элементов.

2. Преобразовать массив таким образом, чтобы элементы располагались в обратном порядке.

3. Найти номер столбца, в которой находится самая длинная серия одинаковых элементов.

Вариант 5

1. Найти сумму элементов массива с нечетными номерами.

2. Найти произведение элементов массива, расположенных между первым нулевым и последним нулевым элементами.

3. Найти количество столбцов, содержащих хотя бы один нулевой элемент.

Вариант 6

1. Найти максимальный элемент массива.

2. Найти сумму элементов массива, расположенных между первым и вторым нулевым элементами.

3. Найти количество строк, в которых положительных элементов больше чем отрицательных.

Вариант 7

1. Найти сумму модулей элементов массива, расположенных после первого отрицательного элемента.

2. Отсортировать элементы массива по убыванию до первого элемента, равного нулю.

3. Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Вариант 8

1. Найти сумму элементов массива, расположенных после первого нечетного элемента.

2. Отсортировать элементы массива по убыванию до первого отрицательного элемента.

3. Найти сумму элементов в тех столбцах, которые содержат хотя бы один положительный элемент.

Вариант 9

1. Найти произведение элементов массива, расположенных между первым и последним четным элементами.

2. Отсортировать элементы массива по убыванию до первого четного элемента.

3. Найти количество строк, содержащих хотя бы один отрицательный элемент.

Вариант 10

1. Найти произведение элементов массива, расположенных после максимального элемента.

2. Отсортировать элементы массива по убыванию до первого элемента, равного s.

3. Найти сумму элементов в тех строках, которые содержат хотя бы один нулевой элемент.

Вариант 11

1. Найти сумму элементов массива, расположенных до первого элемента, равного s.

2. Отсортировать элементы массива по убыванию до последнего отрицательного элемента.

3. Определить номер последней из строк, содержащих хотя бы один четный элемент.

Вариант 12

1. Найти произведение элементов массива, расположенных после максимального по модулю элемента.

2. Отсортировать элементы массива по убыванию до последнего элемента, равного нулю.

3. Определить номер последнего из столбцов, содержащих хотя бы один положительный элемент.

Вариант 13

1. Найти произведение отрицательных элементов массива

2. Отсортировать элементы массива по возрастанию до последнего положительного элемента.

3. Определить номер последнего из столбцов, содержащих хотя бы один нулевой элемент.

Вариант 14

1. Найти сумму модулей элементов массива, расположенных после первого элемента, равного нулю.

2. Отсортировать элементы массива по убыванию до последнего четного элемента.

3. Определить номер последней из строк, содержащих хотя бы один не четный элемент.

Вариант 15

1. Найти сумму элементов массива, расположенных после первого положительного элемента.

2. Отсортировать элементы массива по убыванию до первого элемента, равного s.

3. Найти минимальное из чисел, встречающихся в заданной матрице более одного раза.

Вариант 16

1. Найти сумму модулей элементов массива, расположенных после первого элемента, равного s.

2. Отсортировать элементы массива по возрастанию до последнего нечетного элемента.

3. Определить номер первой из строк, не содержащих ни одного положительного элемента.

Вариант 17

1. Найти произведение элементов массива, расположенных после минимального элемента.

2. Отсортировать элементы массива по возрастанию до первого нечетного элемента.

3. Найти количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент.

Вариант 18

1. Найти произведение положительных элементов массива.

2. Найти произведение элементов массива, расположенных после минимального по модулю элемента.

3. Найти сумму элементов в тех строках, которые не содержат отрицательных элементов.

Вариант 19

1. Найти произведение элементов массива, лежащих в диапазоне от A до B.

2. Найти произведение элементов массива, расположенных между первым и последним нечетным элементами.

3. Найти количество строк, содержащих хотя бы один нулевой элемент.

Вариант 20

1. Найти произведение элементов массива, меньших *C*.

2. Найти сумму элементов массива, расположенных до первого отрицательного элемента.

3. Найти количество строк, содержащих более одного нулевого элемента.

Вариант 21

1. Найти произведение элементов массива, больших *C*.

2. Отсортировать элементы массива по возрастанию до первого положительного элемента.

3. Найти количество строк, среднее арифметическое элементов которых меньше заданной величины.

Вариант 22

1. Найти количество элементов массива, меньших *C*.

2. Найти произведение элементов массива, расположенных до первого элемента, равного нулю.

3. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент.

Вариант 23

1. Найти количество положительных элементов массива.

2. Найти сумму элементов массива, расположенных до последнего элемента равного s.

3. Найти номер первой из строк, содержащих хотя бы один положительный элемент.

Вариант 24

1. Найти количество отрицательных элементов массива.

2. Найти сумму модулей элементов массива, расположенных после первого четного элемента.

3. Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями.

Вариант 25

1. Найти количество элементов массива, больших *C*.

2. Найти сумму элементов массива, расположенных между первым и вторым нечетным элементами.

3. Определить номер первого из столбцов, не содержащих ни одного отрицательного элемента.

Вариант 26

1. Найти количество элементов массива, лежащих в диапазоне от A до B.

2. Найти сумму элементов массива, расположенных между первым и вторым четным элементами.

3. Найти сумму модулей элементов, расположенных выше главной диагонали.

Вариант 27

1. Найти количество элементов массива, равных нулю.

2. Найти сумму элементов массива, расположенных между первым и вторым отрицательным элементами.

3. Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Вариант 28

1. Найти максимальный по модулю элемент массива.

2. Найти сумму элементов массива, расположенных между первым и вторым положительным элементами.

3. Найти такие *k*, при которых *k*-я строка матрицы совпадает с *k*-м столбцом.

Вариант 29

1. Найти номер минимального элемента массива.

2. Найти сумму элементов массива, расположенных до последнего отрицательного элемента.

3. Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Вариант 30

1. Найти минимальный по модулю элемент массива.

2. Найти сумму элементов массива, расположенных до последнего положительного элемента.

3. Найти сумму элементов в тех столбцах, которые не содержат отрицательных элементов.

Примеры выполнения заданий

Задание 1. Найти произведение элементов массива с четными номерами. (Результаты представлены на рисунке 4).

```
using System;
using System.Collections.Generic;
using System.Text;

namespace lab5_1
{class Program
{static void Main(string[] args)
 {Console.WriteLine("Введите n");
  int n = Convert.ToInt32(Console.ReadLine());
  double pr=1;
  int i;
```

```
  Console.WriteLine("Введите элементы массива");
  int[] a=new int[n];//Создаем массив из 10 элементов
   //ввод массива
  for (i=0;i<a.Length;i++)
   a[i]=Convert.ToInt32(Console.ReadLine());
  Console.WriteLine();
   //вывод массива
  PrintArray("Исходный массив", a);
  //обработка
    for (i = 0; i < n; i++)
     if (a[i]%2 == 0) pr = pr * a[i];
  //вывод результата
if (pr != 1)  Console.WriteLine("Произведение четных
элементов массива = " + pr);
 else Console.WriteLine("Нет четных элементов ");}
   //функция для вывода элементов массива
public static void PrintArray(string header, int[] a)
 {Console.WriteLine(header);
   for (int i = 0; i < a.Length; ++i)
    Console.Write("\t" + a[i]);
     Console.WriteLine();  }
   } }
```

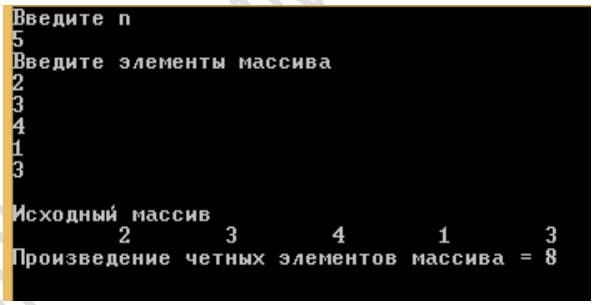


Рисунок 4 – Результат работы программы

Задание 2. Дан прямоугольный массив. Найти номер строки, в которой находится максимальный элемент. (Результаты представлены на рисунке 5).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{class Program
 {static void Main(string[] args)
```

```
  {int i, j, i_max=1,max;
   Console.WriteLine("Введите n");
   int n = Convert.ToInt32(Console.ReadLine());
   Console.WriteLine("Введите m");
   int m = Convert.ToInt32(Console.ReadLine());
   int[,] a = new int[n, m];
   Console.WriteLine("Введите элементы массива");
    for (i = 0; i < n; i++)
     for (j = 0; j < m; j++)
      a[i, j] = Convert.ToInt32(Console.ReadLine());
   Console.WriteLine();
   //вывод на печать вектора
    PrintArray("Исходный массив", a, n, m);
   //обработка
    for (i = 0; i < n; i++)
     { max = a[i, 1];
       for (j = 0; j < m; j++)
        if (a[i, j] >= max)
         {max = a[i, j];
          i_max = i;}
     }
     Console.WriteLine("Номер строки, в которой нахо-
дится максимальный элемент " + i_max);
     Console.Read();
           }

public static void PrintArray(string header, int[,]
a, int n1, int m1)
  { Console.WriteLine(header);
     for (int i = 0; i < n1; ++i)
      { for (int j = 0; j < m1; ++j)
         Console.Write("\t" + a[i, j]);
       Console.WriteLine();
   }}
}}
```

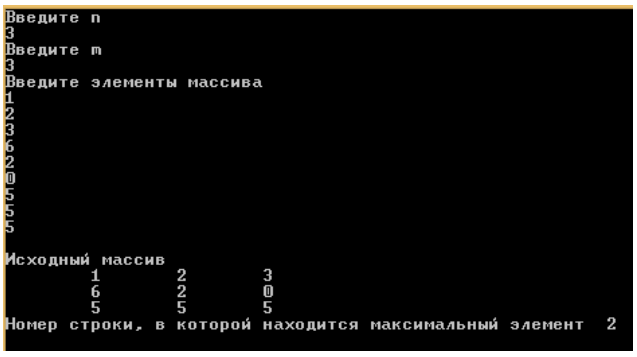


Рисунок 5 – Результат работы программы

Вопросы для самоконтроля

1. Какие бывают массивы?
2. Каков формат описания одномерных массивов?
3. Что такое ступенчатые массивы?
4. Каков формат описания ступенчатых массивов?
5. В каком классе содержаться методы и свойства для работы с одномерными массивами?
6. Какой метод предназначен для сортировки элементов одномерного массива?
7. Какое свойство предназначено для определения количества элементов массива?

Тема 4. Методы, функции, структуры и кортежи

4.1. Методы

Методы – набор операторов, которые выполняют определенные действия. Общее определение методов выглядит следующим образом:

```
[модиф-ры] тип_возвр._зн. назв._мет. ([параметры])
{// тело метода}
```

Рассмотрим на примере метода Main:

```
static void Main(string[] args)
 {Console.WriteLine("привет мир!");}
```

Ключевое слово `static` является модификатором. Далее идет тип возвращаемого значения. В данном случае ключевое слово `void` указывает на то, что метод ничего не возвращает. Такой метод еще называется процедурой. Далее идет название метода – `Main`, и в скобках параметры – `string[] args`. В фигурные скобки заключено тело метода – все действия, которые он выполняет.

Чтобы вызвать метод в программе, надо указать имя метода, а после него в скобках значения для его параметров:

```
string message = Hello(); // вызов первого метода.
```

4.2. Функции

В отличие от процедур функции возвращают определенное значение. Например:

```
int Factorial()
{return 1;}
```

В функции в качестве типа возвращаемого значения вместо `void` используется любой другой тип. В данном случае тип `int`. Функции также отличаются тем, что мы обязательно должны использовать оператор `return`, после которого ставится возвращаемое значение. Также стоит отметить, что возвращаемое значение всегда должно иметь тот же тип, что значится в определении функции.

4.3. Параметры методов

Параметры представляют собой переменные, которые определяются в сигнатуре метода и создаются при его вызове. В C# для обмена данными между вызывающей и вызываемой функциями предусмотрено четыре типа параметров:

– *параметры-значения*;

– *параметры-ссылки* – описываются с помощью ключевого слова `ref`;

– *выходные параметры* – описываются с помощью ключевого слова `out`:

– *параметры-массивы* – описываются с помощью ключевого слова `params`.

Ключевое слово предшествует описанию типа параметра. Если оно опущено, параметр считается параметром-значением. Параметр-массив может быть только один и должен располагаться последним в списке, например:

```
public int Calculate (int a, ref int b, out int c, params int[] d ) ...
```

При *передаче по значению* метод получает копии значений аргументов, и операторы метода работают с этими копиями. Доступа к исходным значениям аргументов у метода нет, а следовательно, нет и возможности их изменить. Параметр-значение описывается в заголовке метода следующим образом:

```
тип имя
```

Пример заголовка метода, имеющего один параметр-значение целого типа:

```
void P(int x)
```

При *передаче по ссылке* (по адресу) метод получает копии адресов аргументов, он осуществляет доступ к ячейкам памяти по этим адресам и может изменять исходные значения аргументов, модифицируя параметры. Признаком параметра-ссылки является слово ref перед описанием параметра:

```
ref тип имя
```

Пример заголовка метода, имеющего один параметр-ссылку целого типа:

```
void P(ref int x)
```

Выходные параметры. Выше мы использовали входные параметры. Но параметры могут быть также выходными. Чтобы сделать параметр выходным, перед ним ставится модификатор `out`:

```
static void Sum(int x, int y, out int a)
 {a = x + y;}
```

Здесь результат возвращается не через оператор `return`, а через выходной параметр. Использование в программе:

```
int x = 10; int z;
   Sum(x, 15, out z);
```

Причем, как и в случае с `ref`, ключевое слово `out` используется как при определении метода, так и при его вызове.

Также обратите внимание, что методы, использующие такие параметры, обязательно должны присваивать им определенное значение.

4.4. Кортежи

Массивы комбинируют объекты одного типа, а кортежи (`tuple`) могут комбинировать объекты различных типов. Кортежи предоставляют удобный способ для работы с набором значений, который был добавлен в версии C# 7.0. *Кортеж* представляет набор значений, заключенных в круглые скобки:

```
var tuple = (5, 10);
```

В данном случае определен кортеж `tuple`, который имеет два значения: 5 и 10. Обращаться к каждому из элементов кортежа можно через поля с названиями `Item[поряд._номер_поля_в_кортеже]`:

```
Console.WriteLine(tuple.Item1); // 5
```

Существуют различные обобщенные классы `Tuple` для поддержки различного количества элементов от одного до восьми. В случае если имеется более восьми элементов, которые нужно включить в кортеж, можно использовать определение класса `Tuple` с восемью параметрами. Последний параметр называется `TRest`, в котором должен передаваться сам кортеж. Таким образом, есть возможность создавать кортежи с любым количеством параметров.

Также можно дать названия полям кортежа:

```
var tuple = (count:5, sum:10);
 Console.WriteLine(tuple.count); // 5
 Console.WriteLine(tuple.sum); // 10
```

Теперь, чтобы обратиться к полям кортежа, используются их имена, а не названия `Item1` и `Item2`.

Кортежи могут передаваться в качестве параметров в метод, могут быть возвращаемым результатом функции, либо использоваться иным образом.

4.5. Структуры

Структура – это более простая версия классов. Все структуры наследуются от базового класса `System.ValueType` и являются типами значений. Структуры могут содержать в себе обычные переменные и методы.

Структуры объявляются при помощи ключевого слова `struct`. Для примера создадим структуру `Book`, в которой будут храниться переменные для названия, автора и года издания книги. Кроме того, структура будет содержать метод для вывода информации о книге на консоль:

```
struct Book
{ public string name;
  public string author;
  public int year;
   //метод вывода информации
 public void Info()
{Console.WriteLine($"Книга'{name}'(автор{author})
была издана в {year} году");} }
```

Чтобы можно было использовать переменные и методы структуры, из любого места программы мы ставим перед переменными и методом модификатор доступа `public`. Структуру можно задать как внутри пространства имен, так и внутри класса, но не внутри метода. Экземпляр структуры можно создавать без ключевого слова `new`:

```
Book b;
 b. Name = "BookName";
```

По сути структура `Book` представляет новый тип данных. Мы также можем использовать массив структур:

```
Book[] books=new Book[3];
 books[0].name = "Война и мир";
 books[0].author = "Л. Н. Толстой";
 books[0].year = 1869;
 books[1].name = "Преступление и наказание";
 books[1].author = "Ф. М. Достоевский";
 books[1].year = 1866;
 // вывод массива структур
 foreach (Book b in books)
  {b.Info();}
```

Структуры подходят для создания несложных типов, таких как точка, цвет, окружность. Если необходимо создать множество экземпляров подобного типа, используя структуры, мы экономим память, которая могла бы выделяться под ссылки в случае с классами.

Конструкторы в структурах. Кроме обычных методов структура может содержать специальный метод – конструктор, который присваивает всем полям некоторые значения по умолчанию. Вызов конструктора имеет следующий синтаксис:

```
new название_структуры ([список_параметров])
```

Пример описания конструктора:

```
struct Book
 {  …
 // конструктор
   public Book(string n, string a, int y)
    { name = n;
      author = a;
      year = y;  }
    … }
```

Конструктор по сути является обычным методом, только не имеет возвращаемого значения, и его название всегда совпадает с именем структуры или класса. Теперь используем его:

```
Book book=new Book("Динка", "В.А. Осеева",1985);
 book.Info();
```

Теперь нам не надо вручную присваивать значения полям структуры – их инициализацию выполнил конструктор.

4.6. Практические задания

Задание 1. Разработать приложение с меню, состоящим из 4 пунктов. Переделать задачи из темы «Массивы» в виде методов с передачей параметров, реализовать передачу данных по значению, по ссылке, использовать выходные параметры и параметры-массивы. Подключить каждую задачу к меню (1, 2, 3 пункт).

Задание 2. На 4-й пункт меню реализовать следующее задание: описать структуру (по вариантам). Создать массив структур из 5 элементов, заполнить их программно. Разработать метод, который будет возвращать полученный кортеж (по вариантам) с 5 элементами, по

исходному кортежу с параметрами (по вариантам). Варианты даны в таблице 8.

Таблица 8 – Варианты заданий

Вариант	Структура	Исходный кортеж (входные параметры)	Полученный кортеж (выходные параметры)
1	2	3	4
1	**Магазин:** название, владелец, адрес, 3 оценки покупателя	название, адрес, дата открытия, оценка покупателя	название, владелец, код магазина, используя комбинацию владелец + дата открытия, средняя оценка покупателей, да или нет (определить, совпадает ли адрес).
2	**Театр:** название представления, время, тип места (партер, балкон), ряд, 3 цены билетов	название представления, № места, цена, время	название представления, тип места, номер билета, используя комбинацию тип места + ряд + № места, среднюю цену, да или нет (определить, совпадает ли время).
3	**Работники:** фамилия, название отдела, код должности, год поступления на работу, заработная плата за три месяца.	фамилия, номер трудового договора, заработная плата за последний месяц, год	фамилия, название отдела, номер пропуска, используя комбинацию код должности + номер трудового договора, среднюю заработную плату, да или нет (определить стаж работы > 5 лет).
4	**Магистрант:** фамилия, факультет, код специальности, оценки за три экзамена.	фамилия, номер группы, год поступления, оценка за последний экзамен	фамилия, факультет, номер зачетки, используя комбинацию код специальности и номер группы, средний балл за четыре экзамена, 1 или 2 (определить на каком курсе учится).
5	**Кинотеатр:** название фильма, время, тип фильма (3d или нет), ряд, цена билетов (3)	название фильма, место, цена, время	название фильма, тип места, номер билета, используя комбинацию тип фильма + ряд + место, среднюю цену, да или нет (определить, совпадает ли время).
6	**Анкета:** фамилия, гражданство, адрес, 3 оценки.	фамилия, год рождения, оценка	фамилия, гражданство, пароль, используя комбинацию фамилия + год рождения, средняя оценка, количество полных лет.

Продолжение таблицы 8

1	2	3	4
7	**Фильмотека:** название фильма, жанр, страна, 3 цены	название фильма, год, страна, цена	название, жанр, код фильма, используя комбинацию жанр + год, средняя цена, да или нет (определить совпадает ли страна).
8	**Книжный:** название, количество страниц, автор, 3 цены	название, количество страниц, год, цена	название, автор, код книги, используя комбинацию автор + год, средняя цена, да или нет (определить, совпадает ли количество страниц).
9	**Регистрация:** фамилия, пол, адрес, 3 оценки.	фамилия, фамилия на английском, год рождения, оценка	фамилия, пол, email, используя комбинацию фамилия на английском + год рождения, средняя оценка, количество полных лет.
10	**Сериалы:** название, количество сезонов, страна, 3 оценки	название, год, количество сезонов, оценка	название, страна, код сериала, используя комбинацию страна + год, средняя оценка, да или нет (определить совпадает ли количество сезонов).
11	**Сувениры:** название, материал, страна, 3 цены	название, страна, дата изготовления, цена	Название, материал, код сувенира, используя комбинацию страна + дата, средняя цена, да или нет (определить, совпадает ли страна).
12	**Цветы:** название, вид, страна, наличие горшка, 3 цены	название, страна, дата, цена	Название, вид, код цветка, используя комбинацию вид + дата, средняя цена, да или нет (определить, совпадает ли страна).
13	**Мультфильмы:** название, продолжительность, страна, 3 цены	название, год, продолжительность, цена	название, страна, код мультфильма, используя комбинацию страна + год, средняя цена, да или нет (определить, совпадает ли продолжительность).
14	**Посуда:** название, материал, тип, 3 цены	название, тип, дата, цена	Название, материал, код посуды, используя комбинацию материал + дата, средняя цена, да или нет (определить, совпадает ли тип).
15	**Статьи:** название, количество страниц, журнал, 3 оценки	название, количество страниц, год, оценка	название, журнал, код статьи, используя комбинацию журнал + год, средняя оценка, да или нет (определить, совпадает ли количество страниц).
16	**Авторизация:** фамилия, пол, год рождения, 3 оценки.	фамилия, фамилия на английском, оценка	фамилия, пол, пароль, используя комбинацию фамилия на английском + год рождения, средняя оценка, количество полных лет.
17	**Телевизоры:** название, разрешение, страна, гарантия, 3 цены	название, страна, дата поставки, цена	название, разрешение, код телевизора, используя комбинацию страна + дата, средняя цена, да или нет (определить, совпадает ли страна).

Продолжение таблицы 8

1	2	3	4
18	**Журнал:** название, количество страниц, издательство, 3 цены	название, количество страниц, номер, цена	название, автор, код журнала, используя комбинацию издательство + номер, средняя цена, да или нет (определить, совпадает ли количество страниц).
19	**Альбомы:** название, жанр, количество дорожек, 3 цены	название, количество дорожек, автор, цена	название, жанр, код альбома, используя комбинацию жанр + автор, средняя цена, да или нет (определить, совпадает ли количество дорожек).
20	**Поезда:** станция прибытия, станция отбытия, машинист, продолжительность поездки, 3 цены	станция прибытия, продолжительность, время отправки, цена	станция прибытия, станция отбытия, номер поезда, используя комбинацию машинист + время отправки, средняя цена, да или нет (определить, совпадает ли продолжительность).
21	**Мебель:** название, материал, тип, 3 цены	название, тип, дата изготовления, цена	название, материал, код мебели, используя комбинацию материал + дата, средняя цена, да или нет (определить, совпадает ли тип).
22	**Ткани:** название, материал, цвет, ширина, 3 цены	название, цвет, количество рулонов, цена	название, материал, код ткани, используя комбинацию цвет + количество рулонов, средняя цена, да или нет (определить, совпадает ли цвет).
23	**Аэрофлот:** пункт назначения, тип самолета, количество мест, 3 цены	пункт назначения, количество мест, время отправки, цена	пункт назначения, тип самолета, номер рейса, используя комбинацию пункт назначения + время отправки, средняя цена, да или нет (определить, совпадает ли количество мест).
24	**Игрушки:** название, материал, тип, 3 цены	название, тип, для какого возраста, цена	название, материал, код игрушки, используя комбинацию тип + для какого возраста, средняя цена, да или нет (определить, совпадает ли тип).
25	**Инструменты:** название, материал, вид, 3 цены	название, вид, количество, цена	название, материал, код инструмента, используя комбинацию вид + количество, средняя цена, да или нет (определить, совпадает ли вид).
26	**Самолеты:** пункт назначения, тип самолета, количество мест, 3 цены	пункт назначения, количество мест, пилот, цена	пункт назначения, тип самолета, номер самолета, используя комбинацию тип самолета + количество мест, средняя цена, да или нет (определить, совпадает ли количество мест).

Окончание таблицы 8

1	2	3	4
27	**Порт:** пункт назначения, тип судна, количество мест, 3 цены	пункт назначения, количество мест, время отправки, цена	пункт назначения, тип судна, номер судна, используя комбинацию пункт назначения + время отправки, средняя цена, да или нет (определить, совпадает ли количество мест).
28	**Лекарства:** название, назначение, тип, 3 цены	название, тип, для какого возраста, цена	название, назначение, код лекарства, используя комбинацию тип + для какого возраста, средняя цена, да или нет (определить, совпадает ли тип).
29	**Семена:** название, страна, тип, 3 цены	название, тип, количество, цена	название, страна, код семян, используя комбинацию тип + количество, средняя цена, да или нет (определить, совпадает ли тип).
30	**Техника:** название, страна, тип, 3 цены	название, тип, срок эксплуатации, цена	название, страна, код техники, используя комбинацию тип + срок эксплуатации, средняя цена, да или нет (определить, совпадает ли тип).

Пример выполнения задания 2. Создать структуру «Студент» с полями: фамилия, факультет, специализация, оценки за три экзамена. Создать массив структур из 2 элементов, заполнить их программно. Разработать метод, который будет возвращать кортеж с 5 элементами, по кортежу с параметрами: фамилия, курс, год рождения, оценка за последний экзамен. Нужно вывести фамилию, группу, используя комбинацию: специальность и курс, средний балл за четыре экзамена (Результаты представлены на рисунке 6).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{class Program
{struct stud
   { public string fio;
     public string fakultet;
     public string spec;
     public int o1, o2, o3;
```

```
   }
static  Tuple<string,  string,  float,  string>  Cor-
teg(string fam1, int kurs1,int god1, int o4 )
 {string p = fam1,grup1,otv;
  float sum=0, sr;
  stud[] st = new stud[3];
  st[0].fio = "Петров";
  st[0].fakultet = "Математический";
  st[0].spec = "ПО";
  st[0].o1 = 4;  st[0].o2 = 8;  st[0].o3 = 9;
  st[1].fio = "Сидоров";
  st[1].fakultet = "Биологический";
  st[1].spec = "Б";
  st[1].o1 = 10;  st[1].o2 = 7;  st[1].o3 = 8;
int nom = -1;
for (int i = 0; i <= 2; i++)
 if (st[i].fio == fam1)
  { nom = i;
    break;  }
if (nom != -1)
 {grup1 = st[nom].spec+"-" + Convert.ToString(kurs1);
  sum = st[nom].o1 + st[nom].o2 + st[nom].o3 + o4;
  sr = sum / 4;  }
 else { grup1 = "не определена"; sr = o4/4f; }
if (2017 - god1 >= 18)   otv = "да";
 else otv = "нет";
return Tuple.Create<string, string, float, string>(p,
grup1, sr, otv); }

static void Main(string[] args)
 {int c;
  string text;
  Console.WriteLine("vvod kurs=");
  int kurs=Convert.ToInt16(Console.ReadLine());
  Console.WriteLine("vvod familiya=");
  string fam = Console.ReadLine();
  Console.WriteLine("vvod god=");
  int god = Convert.ToInt16(Console.ReadLine());
  Console.WriteLine("vvod ocenky=");
  int ocen = Convert.ToInt16(Console.ReadLine());
  var myTuple = Corteg(fam,kurs,god,ocen);
  Console.WriteLine("Фамилия:{0}\n"+          "Группа:
{1}\n"+"Средний  балл:{2}\n"+"Совершеннолетний:{3}\n",
```

```
myTuple.Item1,myTuple.Item2,myTuple.Item3,          my-
Tuple.Item4);
   Console.ReadLine();  }}}
```

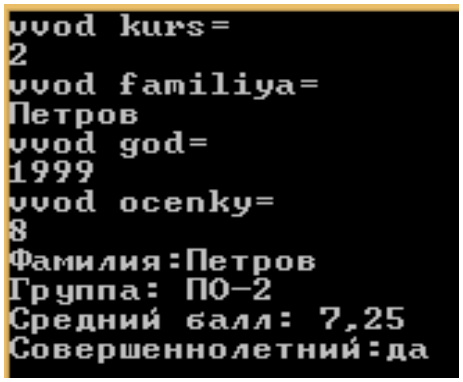


Рисунок 6 – Результат выполнения программы

Вопросы для самоконтроля

1. Что такое метод?
2. Чем функция отличается от метода?
3. Перечислите типы параметров метода?
4. Что такое структура?
5. Как описывается структура?
6. Что такое кортеж?

Литература

1. METANIT.COM [Электронный ресурс]: сайт о программировании. – Режим доступа: https://metanit.com/sharp/tutorial/. – Дата доступа: 24.03.2018.

2. Павловская, Т. А. C#. Программирование на языке высокого уровня: учебник для вузов / Т. А. Павловская. – СПб. : Питер, 2015. – 432 с.

3. ProfessorWeb .Net & Web Programming [Электронный ресурс]. – Режим доступа : https://professorweb.ru/my/csharp/. – Дата доступа : 24.03.2018.

4. Бишоп, Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. – М. : Бином. Лаборатория знаний, 2011. – 472 с.

5. Васильев, Алексей C#. Объектно-ориентированное программирование / Алексей Васильев. – М. : Питер, 2012. – 320 с.

6. Культин, Н. Microsoft Visual C# в задачах и примерах / Н. Культин. – М. : БХВ-Петербург, 2012. – 314 с.

7. Подбельский, В. В. Язык C#. Базовый курс / В. В. Подбельский. – М. : Финансы и статистика, 2013. – 408 с.

8. Visual Studio 2010 для профессионалов / Ник Рендольф [и др.]. – М. : Диалектика, 2011. – 584 с.

Производственно-практическое издание

Москалева Марина Валерьевна
Бычков Павел Владимирович

Программирование на языке C#
Основы языка C#

Практическое пособие

Редактор *В. И. Шкредова*
Корректор *В. В. Калугина*

Подписано в печать 01.04.2019. Формат 60x84 1/16.
Бумага офсетная. Ризография. Усл. печ. л. 2,8
Уч.-изд. л. 3,1. Тираж 25 экз. Заказ 183.

Издатель и полиграфическое исполнение:
учреждение образования
«Гомельский государственный университет имени Франциска Скорины».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1452 от 17.04.2017.
Специальное разрешение (лицензия) № 02330 / 450 от 18.12.2013.
Ул. Советская, 104, 246019, Гомель.