СОДЕРЖАНИЕ

1. Лабораторная работа №1…………...………………………………………….2

2. Лабораторная работа №2………………………………………………………5

3. Лабораторная работа №3……………………………………………………..11

4. Лабораторная работа №4……………………………………………………..17

5. Лабораторная работа №5……………………………………………………..24

6. Лабораторная работа №6……………………………………………………..28

7. Лабораторная работа №7……………………………………………………..32

**Лабораторная работа №1**

**Тема: «Линейные программы»**

**Цель работы:** Изучение основ программирования на языке C#.

**Задание:** Написать программу для расчета по двум заданным формулам на языке программирования С#.

Вариант 19. .



**Краткая теория**

На языке C# класс Math, содержащий стандартные математические функции, без которых трудно обойтись при построении многих выражений. Этот класс содержит два статических поля, задающих константы E (число е) и PI (число пи), а также 23 статических метода.

Методы задают:

* тригонометрические функции - Sin, Cos, Tan;
* обратные тригонометрические функции -ASin, ACos, ATan, ATan2(sinx, cosx);
* гиперболические функции - Tanh, Sinh, Cosh;
* экспоненту и  логарифмические функции - Exp, Log, Log10;
* модуль, корень, знак - Abs, Sqrt, Sign;
* функции округления - Ceiling, Floor, Round;
* минимум, максимум, степень, остаток - Min, Max, Pow, lEEEReminder.

**Практическая часть**

Код программы:

using System;

using System.Collections.Generic;

using System.Text;

namespace TeorAlg\_lab1

{

class Program

{

static void Main(string[] args)

{

double z1, z2, a;

Console.Write("Введите а=");

a = Convert.ToDouble(Console.ReadLine());

z1 = 5 - 2 \* Math.Pow(a,2);

z1 /= (1 + a + Math.Pow(a, 2)) / (2 \* a + Math.Pow(a, 2)) + 2 - (1 - a + Math.Pow(a, 2)) / (2 \* a - Math.Pow(a, 2));

Console.WriteLine("z1=" + z1);

z2 = (4 - Math.Pow(a, 2)) / 2;

Console.WriteLine("z2=" + z2);

if (z1 == z2)

Console.WriteLine("z1=z2");

Console.WriteLine("Для выхода нажми что-нибудь");

Console.ReadKey(true);

}

}

}



Рисунок 1 – Блок-схема реализующая алгоритм выполнения вычислений

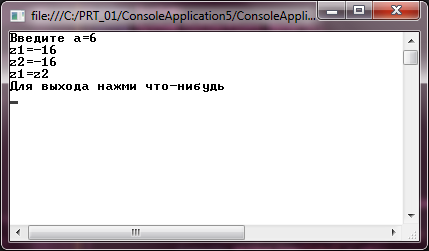


Рисунок 2 – Пример выполнения программы

**Вывод:** Получены навыки написания линейных программ на языке C# в среде программирования Microsoft Visual Studio.

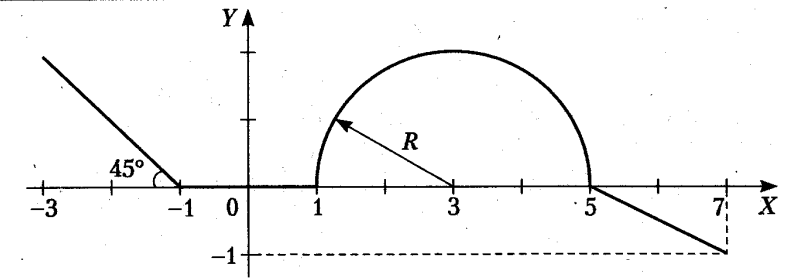
**Лабораторная работа №2**

**Тема: «Разветвляющиеся вычислительные процессы »**

**Цель работы:** Изучение операторов ветвления на языке C#.

**Задание:**  Часть 1. Написать программу, вычисляющую значение кусочной функции, заданной на графике (Рисунок 1).

Рисунок 1 – График кусочной функции



Часть 2. Написать программу, которая определяет, принадлежит ли пара (х,у) закрашенной области (Рисунок 2).

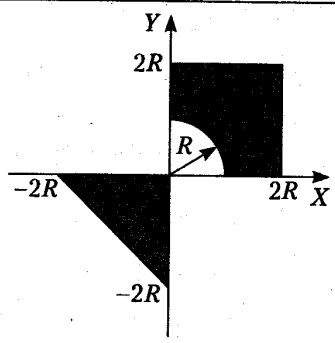


Рисунок 2 – Изображение закрашенной области

**Краткая теория**

Для организации условного ветвления язык C# унаследовал от С и С++ конструкцию if...else. Синтаксис:

if (условие)

оператор (операторы)

else

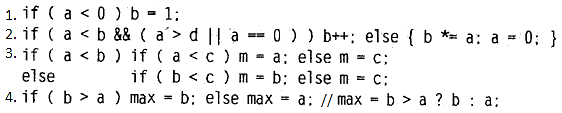
оператор (операторы)

Если по каждому из условий нужно выполнить более одного оператора, эти операторы должны быть объединены в блок с помощью фигурных скобок {...}.

Стоит обратить внимание, что в отличие от языков С и С++, в C# условный оператор if может работать только с булевскими выражениями, но не с произвольными значениями вроде -1 и 0.

В операторе if могут применяться сложные выражения, и он может содержать операторы else, обеспечивая выполнение более сложных проверок. Синтаксис похож на применяемый в аналогичных ситуациях в языках С (С++) и Java. При построении сложных выражений в C# используется вполне ожидаемый набор логических операторов.

Примеры использования условного оператора if:



Вторым оператором выбора в C# является оператор switch, который обеспечивает многонаправленное ветвление программы. Следовательно, этот оператор позволяет сделать выбор среди нескольких альтернативных вариантов дальнейшего выполнения программы. Несмотря на то, что многонаправленная проверка может быть организована с помощью последовательного ряда вложенных операторов if, во многих случаях более эффективным оказывается применение оператора switch. Этот оператор действует следующим образом. Значение выражения последовательно сравнивается с константами выбора из заданного списка. Как только будет обнаружено совпадение с одним из условий выбора, выполняется последовательность операторов, связанных с этим условием. Ниже приведена общая форма оператора switch:

switch(выражение) {

case константа1:

последовательность операторов

break;

case константа2:

последовательность операторов

break;

case константаЗ:

последовательность операторов

break;

default: //Введено что-то другое

последовательность операторов

break;

}

Оператор switch...case в C# он немного безопаснее, чем его эквивалент С++. В частности, он запрещает "сквозные" условия почти во всех случаях. Это значит, что если часть case вызывается в начале блока, то фрагменты кода за последующими частями case не могут быть выполнены, если только не используется явно оператор goto для перехода к ним. Компилятор обеспечивает это ограничение за счет того, что требует, чтобы за каждой частью case следовал оператор break, в противном случае он выдает ошибку.

Важно отметить, что заданное выражение в операторе switch должно быть целочисленного типа (char, byte, short или int), перечислимого или же строкового. А выражения других типов, например с плавающей точкой, в операторе switch не допускаются. Зачастую выражение, управляющее оператором switch, просто сводится к одной переменной. Кроме того, константы выбора должны иметь тип, совместимый с типом выражения. В одном операторе switch не допускается наличие двух одинаковых по значению констант выбора.

**Практическая часть**

Код программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TeorAlg\_lab2\_var19

{

class Program

{

static void Main(string[] args)

{

double x, y=0,R;

Console.WriteLine("Первая часть задания"); //Задание. Часть 1

do

{

Console.Write("Введите x=");

x = Convert.ToDouble(Console.ReadLine());

if (x < -1)

y = -(x + 1);

else if (-1 <= x && x <= 1)

y = 0;

else if (1 < x && x < 5)

y = Math.Sqrt(4 - Math.Pow((x - 3), 2));

else if (5 <= x)

y = -0.5 \* x + 2.5;

Console.WriteLine("y=" +y);

Console.WriteLine("Приступить ко второй части задания?

(Да - любой символ, Нет - нет)");

}

while (Console.ReadLine() == "нет");

Console.WriteLine("Вторая часть задания"); //Задание. Часть 2

do

{

Console.Write("Введите R=");

R = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите x=");

x = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите у=");

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

if ((y<=2\*R&&x<=2\*R&&x>=0&&!(y<=Math.Sqrt((R+x)\*(R-x))))||

(x<=0&&x>=-2\*R&&y<=0&&y<=2\*R-x))

Console.WriteLine("Пара (х;у) принадлежит области");

else

Console.WriteLine("Пара (х;у) не принадлежит области");

Console.WriteLine("Завершить задание? (Да - любой символ, Нет - нет)");

}

while (Console.ReadLine() == "нет");

}

}

}



Рисунок 3 – Блок-схема реализации программы

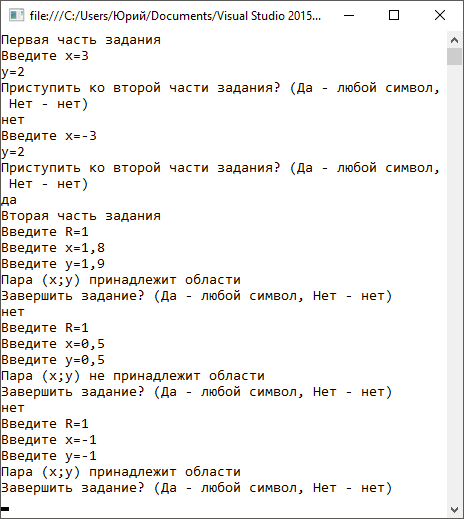


Рисунок 4 – Пример выполнения программы

**Вывод:** Получены навыки написания программ с ветвлениями на языке C# в среде программирования Microsoft Visual Studio.

**Лабораторная работа №3**

**Тема: «Организация циклов»**

**Цель работы:** Ознакомиться с конструкциями циклов на языке C#.

**Задание:**  Вариант 19.

Часть 1. Написать программу, выводящую значения на некотором интервале с заданным шагом, кусочной функции, посчитанной в предыдущей работе.

Часть 2. Организовать цикл для второй части задания предыдущего задания для десяти итераций программы.

Часть 3. Вычислить и вывести на экран в виде таблицы значения функции заданной с помощью ряда Тейлора, на некотором интервале с заданной точностью.

**Краткая теория**

В C# имеются четыре различных вида циклов (for, while, do...while и foreach), позволяющие выполнять блок кода повторно до тех пор, пока удовлетворяется определенное условие.

Цикл for

Цикл for в C# предоставляет механизм итерации, в котором определенное условие проверяется перед выполнением каждой итерации. Синтаксис этого оператора показан ниже:

for (инициализатор; условие; итератор)

  { оператор (операторы); }

Здесь:

Инициализатор – это выражение, вычисляемое перед первым выполнением тела цикла (обычно инициализация локальной переменной в качестве счетчика цикла). Инициализация, как правило, представлена оператором присваивания, задающим первоначальное значение переменной, которая выполняет роль счетчика и управляет циклом;

Условие – это выражение, проверяемое перед каждой новой итерацией цикла (должно возвращать true, чтобы была выполнена следующая итерация);

Итератор – выражение, вычисляемое после каждой итерации (обычно приращение значения счетчика цикла).

Эти три основные части оператора цикла for должны быть разделены точкой с запятой. Выполнение цикла for будет продолжаться до тех пор, пока проверка условия дает истинный результат. Как только эта проверка даст ложный результат, цикл завершится, а выполнение программы будет продолжено с оператора, следующего после цикла for.

Цикл while

Подобно for, while также является циклом с предварительной проверкой. Синтаксис его аналогичен, но циклы while включают только одно выражение:

while (условие)

{ оператор (операторы); }

где оператор — это единственный оператор или же блок операторов, а условие означает конкретное условие управления циклом и может быть любым логическим выражением. В этом цикле оператор выполняется до тех пор, пока условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла.

Как и в цикле for, в цикле while проверяется условное выражение, указываемое в самом начале цикла. Это означает, что код в теле цикла может вообще не выполняться, а также избавляет от необходимости выполнять отдельную проверку перед самим циклом.

Цикл do. . . while

Цикл do...while в C# — это версия while с постпроверкой условия. Это значит, что условие цикла проверяется после выполнения тела цикла. Следовательно, циклы do...while удобны в тех ситуациях, когда блок операторов должен быть выполнен как минимум однажды. Ниже приведена общая форма оператора цикла do-while:

do {

операторы;

} while (условие);

При наличии лишь одного оператора фигурные скобки в данной форме записи необязательны. Тем не менее они зачастую используются для того, чтобы сделать конструкцию do-while более удобочитаемой и не путать ее с конструкцией цикла while. Цикл do-while выполняется до тех пор, пока условное выражение истинно.

**Практическая часть**

Код программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace TA\_lab3\_var19

{

class Program

{

static void Main(string[] args)

{

Console.Write("Выборка заданий: \n\tЧасть 1 - 1 \n\tЧасть 2 - 2 \n\tЧасть 3 - 3\nВыберете задание: ");

switch (Console.ReadLine()) {

case "1": Console.WriteLine("\nЧасть-1");

double x = -4, y = 0, dx;

Console.WriteLine("Таблица значений кусочной функции");

Console.WriteLine(" \_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine(" | x | y |");

Console.WriteLine(" |\_\_\_\_|\_\_\_\_\_|");

for (dx = 1; x < 9; x = x + dx)

{

if (x < -1)

y = -(x + 1);

else if (-1 <= x && x <= 1)

y = 0;

else if (1 < x && x < 5)

y = Math.Sqrt(4 - Math.Pow((x - 3), 2));

else

y = -0.5 \* x + 2.5;

Console.WriteLine(" |" + String.Format("{0,4:0.#}", x) + "|" + String.Format("{0,5:0.##}", y) + "|");

Console.WriteLine(" |\_\_\_\_|\_\_\_\_\_|");

}

break;

case "2": Console.WriteLine("\nЧасть-2");

Double R;

Console.WriteLine("Попадания в мишень");

for (int i = 0; i < 10; i++)

{

while (true)

{

try

{

Console.Write("Введите R=");

R = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите x=");

x = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите у=");

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

}

catch (FormatException) { Console.WriteLine("\nОшибка формата данных данных\n"); continue; }

break;

}

if ((y <= 2 \* R && x <= 2 \* R && x >= 0 && !(y <= Math.Sqrt((R + x) \* (R - x)))) || (x <= 0 && x >= -2 \* R && y <= 0 && y <= 2 \* R - x))

Console.WriteLine("Есть попадание!");

else

Console.WriteLine("Мимо!");

}

break;

case "3": Console.WriteLine("\nЧасть-3");

Double ArcSin\_x, ArcSin\_x\_Defect, x\_begin = 0, x\_end = 0;

do

{

try

{

Console.Write("Начало интервала х=");

x\_begin = Convert.ToDouble(Console.ReadLine());

Console.Write("Конец интервала х=");

x\_end = Convert.ToDouble(Console.ReadLine());

Console.Write("Шаг прохода по интервалу dx=");

dx = Convert.ToDouble(Console.ReadLine());

if (x\_begin < -1 || x\_end > 1 || dx > 2 || dx < 0) { Console.WriteLine("\nОшибка ввода данных: -1<x<1, dx<2\n"); continue; }

}

catch (FormatException) { Console.WriteLine("\nОшибка формата данных данных\n"); continue; }

break; //do {..break..}while (true);

} while (true);

Console.WriteLine("\n\tТаблица расчетов ряда Тейлора\n для функции arcsin(x) при точности 0,0001");

Console.WriteLine(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine(" | x |asin\_def| N |arcsin(x)|");

Console.WriteLine(" |\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|");

double e\_defect = 0.0001;

for (x = x\_begin; x < x\_end; x = x + dx)

{

ArcSin\_x = Math.Asin(x);

ArcSin\_x\_Defect = x; //ArcSin\_x\_Defect = x + last\_Taylor\_add; last\_Taylor\_add = NaN при n=0

int n = 1; /\*Счетчик итераций нижеследующего цикла

(количество итераций на 1 меньше т.к. первый член ряда лежит в ArcSin\_x\_Defect = x; \*/

for (Double last\_Taylor\_add = 1; e\_defect < Math.Abs(last\_Taylor\_add); n++) //last\_Taylor\_add=1 -> e\_defect < 1 (100%)

{

double fact\_div = 1, fact\_n = 1;

for (int i = n + 1; i <= 2 \* n; i++) //fact\_div = (2n)!/n! =(n+1)\*(n+2)\*...\*(2n)

{

fact\_div = fact\_div \* i;

fact\_n = fact\_n \* (i-n);

}

last\_Taylor\_add = Math.Pow(x, 2 \* n + 1) \* fact\_div / (Math.Pow(4, n) \* (2 \* n + 1) \* fact\_n); //Вычисление следущего члена ряда, при n=>1

ArcSin\_x\_Defect += last\_Taylor\_add;

}

Console.WriteLine(" |" + String.Format("{0,5:0.#}", x) + "|" + String.Format("{0,8:0.####}", ArcSin\_x\_Defect) + "|" + String.Format("{0,5:0}", n - 1) + "|" + String.Format("{0,9:0.####}", ArcSin\_x) + "|");

Console.WriteLine(" |\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|");

}

break;

default:

Console.WriteLine("\nМне не известно такое задание!");

break;

}

Console.WriteLine("\nВыполнение окончено.");

Console.ReadKey(true);

}

}

}



Рисунок 3 – Блок-схема реализации программы

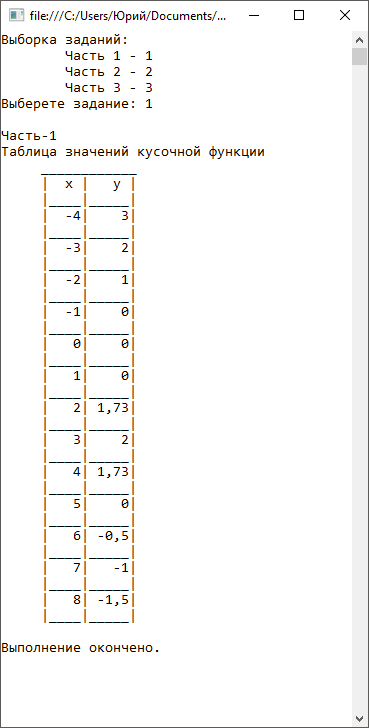
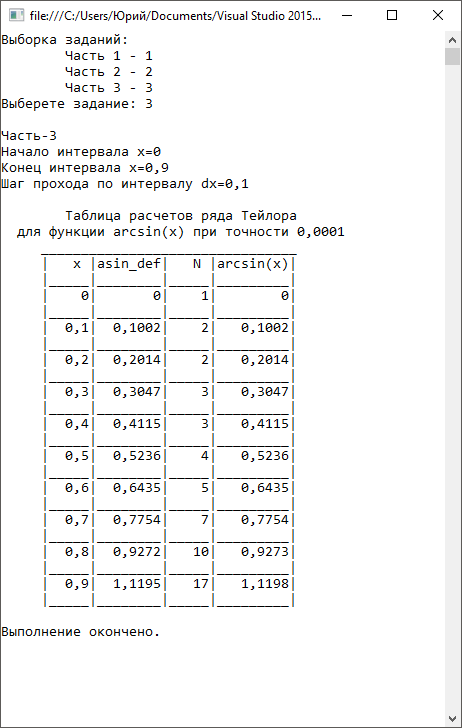


Рисунок 4 – Пример выполнения программы



**Вывод:** Изучена организация циклов на языке C# в среде программирования Microsoft Visual Studio.

**Лабораторная работа №4**

**Тема: «Простейшие классы»**

**Цель работы:** Научиться разрабатывать структуру класса, конструкторы и методы на языке C#.

**Задание:**  Вариант 19. Описать класс "процессор", содержащий сведения о марке, тактовой частоте, объеме кэша, и стоимости. Предусмотреть инициализацию с проверкой допустимости значения полей. В случае недопустимых значений полей выбрасываются исключения. Описать свойства для получения состояния объекта. Описать класс "материнская плата", включающий класс "процессор" и объем установленной оперативной памяти. Предусмотреть инициализацию с проверкой допустимости значений поля объема памяти. в случае недопустимости поля выбрасывать исключение. Описать свойства для получения состояния объекта.

**Краткая теория**

Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В C# используется спецификация класса для построения объектов, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта. При этом очень важно подчеркнуть, что класс является логической абстракцией. Физическое представление класса появится в оперативной памяти лишь после того, как будет создан объект этого класса.

Классы и структуры — это, по сути, шаблоны, по которым можно создавать объекты. Каждый объект содержит данные и методы, манипулирующие этими данными.

Общая форма определения класса

При определении класса объявляются данные, которые он содержит, а также код, оперирующий этими данными. Если самые простые классы могут содержать только код или только данные, то большинство настоящих классов содержит и то и другое.

Вообще говоря, данные содержатся в членах данных, определяемых классом, а код — в функциях-членах. Следует сразу же подчеркнуть, что в C# предусмотрено несколько разновидностей членов данных и функций-членов:

Данные-члены

Данные-члены — это те члены, которые содержат данные класса — поля, константы, события. Данные-члены могут быть статическими (static). Член класса является членом экземпляра, если только он не объявлен явно как static. Давайте рассмотрим виды этих данных:

Поля (field)

Это любые переменные, ассоциированные с классом.

Константы

Константы могут быть ассоциированы с классом тем же способом, что и переменные. Константа объявляется с помощью ключевого слова const. Если она объявлена как public, то в этом случае становится доступной извне класса.

События

Это члены класса, позволяющие объекту уведомлять вызывающий код о том, что случилось нечто достойное упоминания, например, изменение свойства класса либо некоторое взаимодействие с пользователем. Клиент может иметь код, известный как обработчик событий, реагирующий на них.

Функции-члены

Функции-члены — это члены, которые обеспечивают некоторую функциональность для манипулирования данными класса. Они включают методы, свойства, конструкторы, финализаторы, операции и индексаторы:

Методы (method)

Это функции, ассоциированные с определенным классом. Как и данные-члены, по умолчанию они являются членами экземпляра. Они могут быть объявлены статическими с помощью модификатора static.

Свойства (property)

Это наборы функций, которые могут быть доступны клиенту таким же способом, как общедоступные поля класса. В C# предусмотрен специальный синтаксис для реализации чтения и записи свойств для классов, поэтому писать собственные методы с именами, начинающимися на Set и Get, не понадобится. Поскольку не существует какого-то отдельного синтаксиса для свойств, который отличал бы их от нормальных функций, создается иллюзия объектов как реальных сущностей, предоставляемых клиентскому коду.

Конструкторы (constructor)

Это специальные функции, вызываемые автоматически при инициализации объекта. Их имена совпадают с именами классов, которым они принадлежат, и они не имеют типа возврата. Конструкторы полезны для инициализации полей класса.

Финализаторы (finalizer)

Вызываются, когда среда CLR определяет, что объект больше не нужен. Они имеют то же имя, что и класс, но с предшествующим символом тильды. Предсказать точно, когда будет вызван финализатор, невозможно.

Операции (operator)

Это простейшие действия вроде + или -. Когда вы складываете два целых числа, то, строго говоря, применяете операцию + к целым. Однако C# позволяет указать, как существующие операции будут работать с пользовательскими классами (так называемая перегрузка операции).

Индексаторы (indexer)

Позволяют индексировать объекты таким же способом, как массив или коллекцию.

Класс создается с помощью ключевого слова class. Ниже приведена общая форма определения простого класса, содержащая только переменные экземпляра и методы:

class имя\_класса {

// Объявление переменных экземпляра.

доступ тип переменная1;

доступ тип переменная2;

//...

доступ тип переменнаяN;

// Объявление методов.

доступ возращаемый\_тип метод1 (параметры) {

// тело метода

}

доступ возращаемый\_тип метод2 (параметры) {

// тело метода

}

//. . .

доступ возращаемый\_тип методN(параметры) {

// тело метода

}

}

Стоит обратить внимание на то, что перед каждым объявлением переменной и метода указывается доступ. Это спецификатор доступа, например public, определяющий порядок доступа к данному члену класса. Члены класса могут быть как закрытыми (private) в пределах класса, так открытыми (public), т.е. более доступными. Спецификатор доступа определяет тип разрешенного доступа. Указывать спецификатор доступа не обязательно, но если он отсутствует, то объявляемый член считается закрытым в пределах класса. Члены с закрытым доступом могут использоваться только другими членами их класса.

**Практическая часть**

Код программы:

namespace TeorAlg\_lab4\_var19

{

class CPU

{

private string model\_name\_CPU; //Название процессора

private Double cashe\_size; //объем КЭШа

private Double clock\_Rate; //Тактовая частота

private Double price\_CPU; //Цена

public CPU(string Model\_name\_CPU, string Cashe\_size, string Clock\_Rate, string Price\_CPU)

{

try

{

model\_name\_CPU = Model\_name\_CPU;

if (Convert.ToDouble(Cashe\_size) > 0) cashe\_size = Convert.ToDouble(Cashe\_size); else Console.WriteLine("Значение КЭШ памяти должно быть больше нуля");

if (Convert.ToDouble(Clock\_Rate) > 0) clock\_Rate = Convert.ToDouble(Clock\_Rate); else Console.WriteLine("Значение частоты процессора должно быть больше нуля");

if (Convert.ToDouble(Price\_CPU) >= 0) price\_CPU = Convert.ToDouble(Price\_CPU); else Console.WriteLine("Значение стоимости должно быть положительным");

}

catch (FormatException)

{

Console.WriteLine("Неверный формат данных");

}

} //конец конструктора

public CPU(CPU other)

{

model\_name\_CPU = other.model\_name\_CPU;

clock\_Rate = other.clock\_Rate;

cashe\_size = other.cashe\_size;

price\_CPU = other.price\_CPU;

}

public void info\_CPU()

{

Console.WriteLine("Информация о процессоре:\n\tНазвание модели: " + model\_name\_CPU + "\n\tТактовая частота: " + clock\_Rate + " ГГц" + "\n\tОбъем КЭШ памяти: " + cashe\_size + " МБ" + "\n\tСтоимость: " + price\_CPU + " Руб.");

}

}

class MotherBoard

{

private string Model;

private Double RAM;

private Double Price;

public MotherBoard(CPU cpu ,String model, String ram, String price)

{

cpu = new CPU(cpu);

try

{

Model = model;

if (Convert.ToDouble(ram) > 0) RAM = Convert.ToDouble(ram); else Console.WriteLine("Значение оперативной памяти должно быть больше нуля");

if (Convert.ToDouble(price) >= 0) Price = Convert.ToDouble(price); else Console.WriteLine("Значение стоимости должно быть положительным");

}

catch (FormatException)

{

Console.WriteLine("Неверный формат данных");

}

} //конец конструктора

public void info\_MB()

{

Console.WriteLine("Информация о материнской плате:\n\tНазвание модели: " + Model + "\n\tОбъем КЭШ памяти: " + RAM + " МБ" + "\n\tСтоимость: " + Price + " Руб.");

}

class Program

{

static void Main(string[] args)

{

CPU MidNight\_1 = new CPU("MidNight\_1", "8","3","2400");

MotherBoard TotalАgony = new MotherBoard(MidNight\_1, "Total Agony", "8000", "1100");

TotalАgony.info\_MB();

MidNight\_1.info\_CPU();

Console.ReadKey();

}

}

}

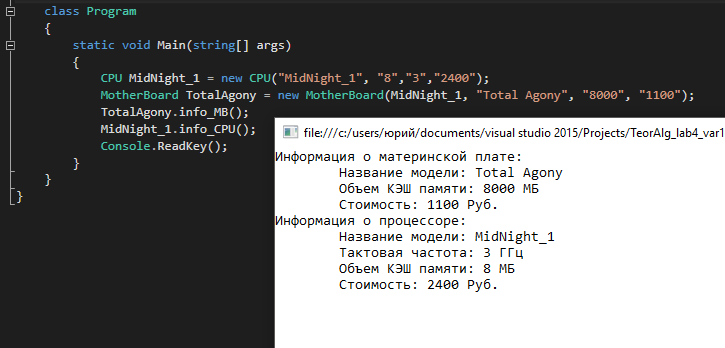


Рисунок 4 – Пример выполнения программы

**Вывод:** Изучена организация циклов на языке C# в среде программирования Microsoft Visual Studio.

**Лабораторная работа №5**

**Тема: «Одномерные массивы»**

**Цель работы:** Научиться работать с одномерными массивами на языке C#.

**Задание:**  Вариант 19. В одномерном массиве, состоящем из вещественных элементов, вычислить: произведение отрицательных элементов, сумму положительных элементов, расположенных до максимального элемента. Изменить порядок следования элементов в массиве на обратный.

**Краткая теория**

В C# массивы фактически являются объектами, а не только адресуемыми областями непрерывной памяти, как в C и C++. Array является абстрактным базовым типом всех типов массивов. Можно использовать свойства и другие члены класса, которые имеет Array. В примере используется свойство Length для получения длины массива. В следующем коде длина массива numbers, равная 5, присваивается переменной lengthOfNumbers:

int[] numbers = { 1, 2, 3, 4, 5 };

int lengthOfNumbers = numbers.Length;

Класс Array позволяет использовать много других полезных методов и свойств для выполнения сортировки, поиска и копирования массивов.

Инициализация массива

Массив можно инициализировать при объявлении. В этом случае спецификация ранга не нужна, поскольку она уже предоставлена по числу элементов в списке инициализации. Примеры.

int[] array1 = new int[] { 1, 3, 5, 7, 9 };

Строковый массив можно инициализировать таким же образом. Ниже приведено объявление строкового массива, в котором каждый элемент инициализируется названием дня:

string[] weekDays = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

При инициализации массива при объявлении можно использовать следующие сочетания клавиш:

int[] array2 = { 1, 3, 5, 7, 9 };

string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

Можно объявить переменную массива без инициализации, но при присвоении массива этой переменной нужно использовать оператор new. Примеры.

int[] array3;

array3 = new int[] { 1, 3, 5, 7, 9 }; // OK

//array3 = {1, 3, 5, 7, 9}; // Error

**Практическая часть**



Рисунок 1 – Блок-схема реализации программы

Код программы:

using System;

using System.Threading;

namespace TeorArg\_lab5\_var19

{

class Program

{

static void Main(string[] args)

{

Double imul\_neg = 1, sum\_pos = 0; //sum\_pos сумма всех положительных

int long\_array = 11, MAX = 100, MIN = -100;

Double[] array = new Double[long\_array]; //объявление

for (int i = 0; i < long\_array; i++) //заполнение

{

array[i] = Rand(MAX, MIN);

Thread.Sleep(150);

}

Console.WriteLine(" Содержание массива");

Console.WriteLine(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine(" | ID |array[ID]|");

Console.WriteLine(" |\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|");

for (int id=0; id< long\_array;id++) //вывод

{

Console.WriteLine(" |" + String.Format("{0,5:0.#}", id) + "|" + String.Format("{0,9:0.##}", array[id]) + "|");

Console.WriteLine(" |\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|");

}

for (int i = 0; i < long\_array; i++) // задание: пункты 1,2

{

if (array[i] < 0) imul\_neg = imul\_neg \* array[i];

else sum\_pos = sum\_pos + array[i];

}

Console.WriteLine("\nПроизведение отрицательных элементов: " + imul\_neg);

Console.WriteLine("\nСумма положительных (без максимального элемента):"+sum\_pos);

Double data;

for (int id = 0; id < long\_array/2; id++)//создание обратного массива

{

data = array[id];

array[id] = array[long\_array - id - 1];

array[long\_array - id - 1] = data;

}

Console.WriteLine("\n\tОбратный массив");

Console.WriteLine(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine(" | ID |array[ID]|");

Console.WriteLine(" |\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|");

for (int id = 0; id < long\_array; id++) //вывод оборотного

{

Console.WriteLine(" |" + String.Format("{0,5:0.#}", id) + "|" + String.Format("{0,9:0.##}", array[id]) + "|");

Console.WriteLine(" |\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|");

}

Console.ReadKey(true);

}

public static Double Rand(int max, int min) //функция случайных значений

{

Random rand = new Random();

return Math.Round(rand.NextDouble(),2)\*rand.Next(min,max);

}

}

}

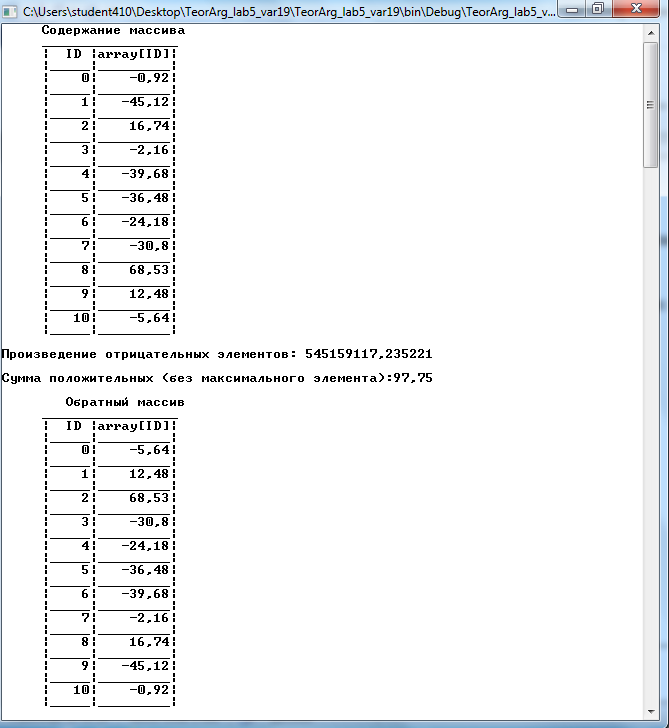


Рисунок 4 – Пример выполнения программы

**Вывод:** Изучена работа с одномерными массивами на языке C# в среде программирования Microsoft Visual Studio.

**Лабораторная работа №6**

**Тема: «Двумерные массивы»**

**Цель работы:** Научиться работать с двумерными массивами на языке C#.

**Задание:**  Вариант 19. Дана целочисленная квадратная матрица. Определить сумму элементов в тех строках, которые не содержат отрицательных элементов; минимум среди элементов диагоналей, параллельных главной диагонали матрицы.

**Краткая теория**

Двумерные массивы

Простейшей формой многомерного массива является двумерный массив. Местоположение любого элемента в двумерном массиве обозначается двумя индексами. Такой массив можно представить в виде таблицы, на строки которой указывает один индекс, а на столбцы — другой.

// Объявляем двумерный массив

int[,] myArr = new int[a,b]; //a,b – высота и ширина массива

**Практическая часть**

Код программы:

using System;

namespace TeorAlg\_lab6\_var19

{

class Program

{

static void Main(string[] args)

{

int edge = 0, sum = 0; //edge - ребро квадратной матрицы

Console.WriteLine("Квадратная матрица");

Console.Write("Укажите кол-во строк(столбцов): ");

edge=Convert.ToInt32(Console.ReadLine());

int[,] matrix = new int[edge, edge]; //объявление

#region Заполнение матрицы (двумерного массива)

Random rand = new Random();

for (int i = 0; i < edge; i++)

{

for (int j = 0; j < edge; j++)

{

matrix[i, j] = rand.Next(-10,100);

}

}

#endregion

#region Форматированный вывод матрицы

for (int i = 0; i < edge; i++)

{

for (int j = 0; j < edge; j++)

{

Console.Write("\_\_\_\_\_\_");

}

Console.WriteLine();

for (int j = 0; j < edge; j++)

{

Console.Write("|" + String.Format("{0,5:0.#}", matrix[i, j]));

}

Console.WriteLine("|");

for (int j = 0; j < edge; j++)

{

Console.Write("|\_\_\_\_\_");

}

Console.WriteLine("|");

}

#endregion

Console.WriteLine("\n");

#region Задание пункт 1

for (int i = 0, sum\_line=0; i < edge; i++)

{

for (int j = 0; j < edge; j++)

{

if (matrix[i, j] >= 0) sum\_line += matrix[i, j];

else { sum\_line = 0; break; }

}

if(sum\_line!=0) Console.WriteLine("Сумма элементов строки "+(i+1)+", не содержаoщей отрицательных элементов: " + sum\_line);

sum += sum\_line;

sum\_line = 0;

}

Console.WriteLine("Сумма элементов строк, которые не содержат отрицательных элементов: " + sum);

#endregion

#region Задание пункт 2

int[] Sum = new int[2 \* (edge - 2)];

for (int pass = 1; pass <= edge-2; pass++)//j(1..4) pass(1..3)

{

int sum\_top = 0, sum\_bot = 0;

for (int i = 0, j = pass; j < edge; i++,j++)

{

sum\_top += matrix[i,j];

sum\_bot += matrix[j,i];

}

Sum[pass-1] = sum\_top;

Sum[2 \* (edge - 2)-pass] = sum\_bot;

}

Array.Sort(Sum);

/\*for (int item=0; item< 2 \* (edge - 2); item++)

{

Console.WriteLine("Диагональ с минимальной суммой элементов: " + Sum[item]);

}\*/

Console.WriteLine("Диагональ с минимальной суммой элементов: " + Sum[0]);

#endregion

Console.ReadKey(true);

}

}

}



Рисунок 1 – Блок-схема реализации программы

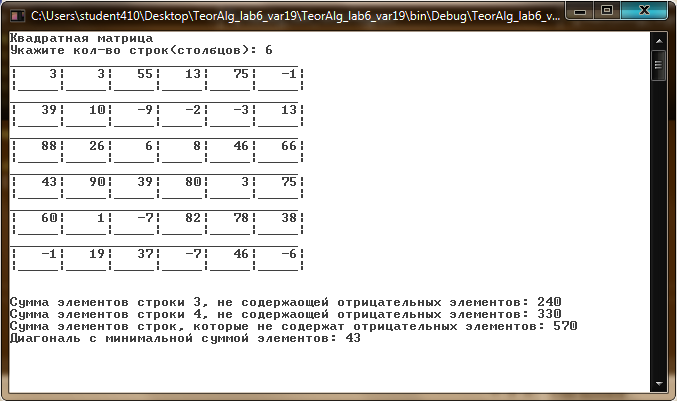


Рисунок 4 – Пример выполнения программы

**Вывод:** Изучена работа с двумерными массивами на языке C# в среде программирования Microsoft Visual Studio.

**Лабораторная работа №7**

**Тема: «Строки»**

**Цель работы:** Научиться работать со строками на языке C#.

**Задание:**  Вариант 19. Написать программу, которая считывает текст из файла и выводит на экран сначала предложения начинающиеся с однобуквенных слов, а затем все остальные.

**Краткая теория**

Строка — это объект типа String, значением которого является текст. Внутренне текст хранится в виде упорядоченной коллекции объектов Char только для чтения. В конце строки C# нет нуль-символов. Поэтому строка C# может содержать любое число внедренных нуль-символов ('\0'). Свойство Length строки представляет число объектов Char, содержащихся в ней, но не число символов Юникода.

String.Split - метод (Char[])

Возвращаемое значение

Тип: System.String[]

Массив, элементы которого содержат подстроки данного экземпляра, разделенные одним или более знаками из separator (массивом Char[]). Каждый элемент в *separator* определяет отдельный знак-разделитель.

Метод String.Replace (String, String)

Возвращает новую строку, в которой все вхождения заданной строки в текущем экземпляре заменены другой заданной строкой.

Класс StreamReader

Реализует объект TextReader, который считывает символы из потока байтов в определенной кодировке. Метод класса ReadToEnd() считывает все символы, начиная с текущей позиции до конца потока. (Переопределяет TextReader.ReadToEnd().)

**Практическая часть**

Код программы:

using System;

using System.IO;

namespace TeorAlg\_lab7\_var19

{

class Program

{

static void Main(string[] args)

{

StreamReader reader = File.OpenText("lab.omg");

string text = reader.ReadToEnd();

Console.WriteLine("\nИсходный текст: \n");

Console.WriteLine(text);

Console.WriteLine("\n\nПреобразованный текст: \n");

text = text.Replace(".", ".//|replace|\\");

text = text.Replace("?", "?//|replace|\\");

text = text.Replace("!", "!//|replace|\\");

string[] sentence = text.Split(new string[] { "//|replace|\\ ", "//|replace|\\" }, StringSplitOptions.RemoveEmptyEntries);

int step = 0;

for (int i = 0; i < sentence.Length; i++)

{

string[] offcut = sentence[i].Split(new string[] { "\r","\n"}, StringSplitOptions.RemoveEmptyEntries);

if (offcut[0].IndexOf(" ") == 1)

{

string temp = sentence[step];

sentence[step] = sentence[i];

sentence[i] = temp; step++;

}

}

for (int item = 0; item < sentence.Length; item++)

{

if (item <= step) { Console.BackgroundColor = ConsoleColor.Red; Console.ForegroundColor = ConsoleColor.Black; }

else { Console.BackgroundColor = ConsoleColor.Black; Console.ForegroundColor = ConsoleColor.Gray; }

Console.WriteLine(sentence[item]);

}

Console.ReadKey();

}

}

} //с оптимизацией под числа string[] offcut = sentence[i].Split(new string[] { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9",".",")","\r","\n" }, StringSplitOptions.RemoveEmptyEntries);



Рисунок 1 – Блок-схема реализации программы

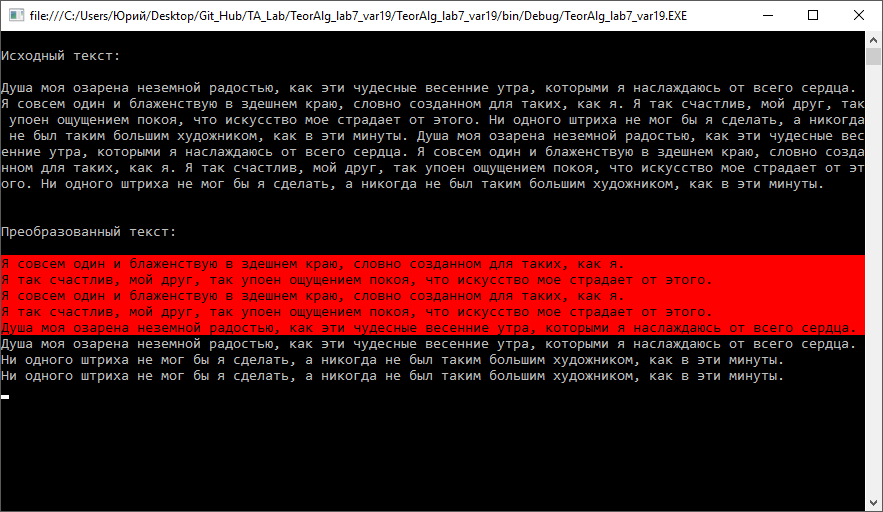


Рисунок 4 – Пример выполнения программы

**Вывод:** Изучена работа со строками на языке C# в среде программирования Microsoft Visual Studio.