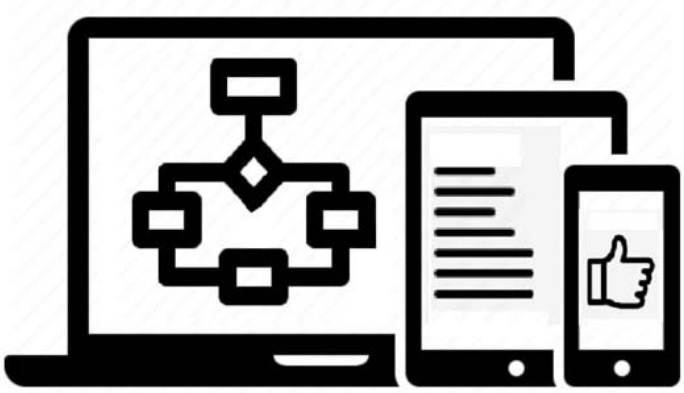
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

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

Часть 1



Могилев 2024

УДК 004.4
ББК 32.973
О75

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

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

Составитель ст. преподаватель А. И. Кашпар

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

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

Учебное издание

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Часть 1

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

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

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

Содержание

Введение

Основная цель дисциплины «Основы алгоритмизации и программирования» – обеспечение студентов базовыми знаниями программирования, привитие студентам навыков постановки, подготовки и решения различных задач на языке программирования высокого уровня и формирование фундаментальной основы для изучения последующих дисциплин.

Программа курса «Основы алгоритмизации и программирования» предусматривает выполнение студентами контрольных и курсовой работ.

Контрольная работа для студентов заочной формы обучения является одной из форм промежуточного контроля знаний и проводится с целью проверки и оценки степени усвоения учебного материала при самостоятельной работе студентов в межсессионный период и умения применять приобретенные знания при решении практических вопросов.

Формой контрольной работы студентов является письменная аудиторная контрольная работа (далее – АКР), выполняемая в период лабораторно-экзаменационной сессии.

Основная часть контрольной работы включает решение задач. Для каждой задачи контрольной работы приводят условие задачи, код программы и схемы алгоритма программы. При этом после условия выделяют исходные и выходные данные, определяют имена идентификаторов для данных и указывают тип.

Схема алгоритма решения задачи должна выполняться в соответствии с ГОСТ 19.701–90.

В методических рекомендациях на примерах рассматриваются различные алгоритмы, методы и приемы написания программ, структуры данных, типичные ошибки, которые совершают начинающие (и не только) программисты. Весь материал разбит на отдельные части в соответствии с заданиями контрольных работ. Логика изложения материала, в основном, соответствует учебной программе.

1 Структура программы. Программирование линейных алгоритмов

В ходе изучения дисциплины «Основы алгоритмизации и программирования» рассматривается язык программирования C#.

C# (произносится си-шарп) – язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Разработан в 1998–2001 гг. группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор с C# входит в стандартную установку самой .NET.

В методических рекомендациях все примеры программ протестированы с использованием компилятора Microsoft Visual Studio 2012. После запуска интегрированной среды необходимо создать проект. Для этого выполняем следующую последовательность команд: File (Файл) > New (Создать)> Project ... (Проект ...).

В открывшемся окне New Project(Создание проекта):

– выберем Templates (Шаблоны) -> Visual C# -> Windows;

– выберем тип Console Application (Консольное приложение);

– введем имя проекта в текстовом поле Name(Имя) (например, Zad1);

– в строке Location (Расположение) определим положение на диске, куда нужно сохранять проект (например Z:\Gruppa\FIO);

– щелкните левой кнопкой мыши на кнопке ОК.

У нас появится шаблон консольного приложения. В шаблоне присутствует только заготовка текста программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Zad1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

В открывшемся редакторе можно создавать программу, но прежде необходимо ознакомиться со структурой простейшей программы на языке C#.

Теперь рассмотрим сам текст программы.

using System – это директива, которая разрешает использовать имена стандартных классов из пространства имен *System* непосредственно без указания имени пространства, в котором они были определены.

Ключевое слово *namespace* создает для проекта свое собственное пространство имен, которое по умолчанию называется именем проекта. В нашем

случае пространство имен называется Zad1. Однако программист вправе указать другое имя. Пространство имен ограничивает область применения имен, делая его осмысленным только в рамках данного пространства.

C# – объектно-ориентированный язык, поэтому написанная на нем программа будет представлять собой совокупность взаимодействующих между собой классов. Автоматически был создан класс с именем Program (в других версиях среды может создаваться класс с именем Class1).

Данный класс содержит только один метод – метод Main(). Метод Main() является точкой входа в программу, т. е. именно с данного метода начнется выполнение приложения. Каждая программа на языке C# должна иметь метод Main ().

После круглых скобок в фигурных скобках { } записывается тело метода, т. е. те операторы, которые требуется выполнить.

Любая заготовка при написании метода имеет вид:

static void Main(string[] args)
{
1) *объявление переменных;*
2) *ввод исходных данных;*
3) *расчет результата;*
4) *вывод результата;*

}

Для хранения данных в программе надо выделить достаточно места в оперативной памяти. Для этого необходимо объявить переменные. Синтаксис объявления переменных:

тип имя_переменной1[, имя_переменной1 …];

Имена переменным дает программист, исходя из их назначения.

Имя может состоять только из латинских или русских букв, цифр и знака подчеркивания _ и не должно начинаться с цифр. Прописные и строчные буквы различаются, например, myname, myName и MyName - три различных имени.

При описании любой переменной нужно указать ее *тип*.

Основные типы:

int (short, long, ushort, uint, ulong) – целочисленные;

double (float, decimal) – вещественные;

char – символьный;

string – строковый;

bool – логический.

Программа при вводе данных и выводе результатов взаимодействует с внешними устройствами. Совокупность стандартных устройств ввода (клавиатура) и вывода (экран) называется консолью. В языке C# для работы с консолью используется стандартный класс Console, определенный в пространстве имен System.

Для **вывода** данных используется метод *WriteLine*() (или *Write*() – вывод без перевода на новую строку), реализованный в классе Console, который поз-

воляет организовывать вывод данных на экран.

Существует несколько способов применения данного метода.

1 *Console.WriteLine(x);*//на экран выводится значение переменной x.

2 *Console.WriteLine("x=" + x +"y=" + y);* /* на экран выводится строка, образованная последовательным слиянием строки "x=", значения x, строки "y=" и значения y */.

3 *Console.WriteLine("x={0} y={1}", x, y);* /* на экран выводится строка, формат которой задан первым аргументом метода, при этом вместо параметра {0} выводится значение x, а вместо {1} – значение y. Первый аргумент определяет формат выходной строки. Следующие аргументы нумеруются с нуля */.

Последний вариант использования метода WriteLine является наиболее универсальным, потому что он позволяет не только выводить данные на экран, но и управлять форматом их вывода.

В выходной строке могут использоваться управляющие последовательности (определенный символ, предваряемый обратной косой чертой):

\a – звуковой сигнал
\b – возврат на шаг назад
\f – перевод страницы
\n – перевод строки
\r – возврат каретки
\t – горизонтальная табуляция
\v – вертикальная табуляция
\\ – обратная косая черта
\’ – апостроф
\” – кавычки

Формат числовых данных имеет вид:

{n,m: <спецификатор>t},

где *n* определяет номер идентификатора из списка аргументов;

m – минимальное количество позиций (размер поля вывода), отводимых под значение данного идентификатора;

<спецификатор> определяет формат данных;

t – количество позиций для дробной части значения вещественного идентификатора или минимальное количество цифр целочисленного формата.

В качестве спецификаторов могут использоваться следующие значения:

D или **d** – целочисленный (используется только с целыми числами);

E или **e** – экспоненциальное представление чисел;

F или **f** – представление чисел с фиксированной точкой;

N или **n** – стандартное форматирование с использованием запятых и пробелов в качестве разделителей между разрядами.

Например:

```
int a=234;
double x=345.677;
Console.WriteLine(“a1={0,5:d} x1={1,7:f2}\nx2={1:e3}”,a,x);
```

Результат выполнения программы (один символ – одна клетка) представлен на рисунке 1.1.

Для **ввода** данных обычно используется метод *ReadLine*, реализованный в классе Console. Особенностью данного метода является то, что в качестве результата он возвращает строку (string).

a	1	=			2	3	4		x	1	=		3	4	5	,	6	8		
x	2	=	3	,	4	5	7	e	+	0	0	2								

Рисунок 1.1

Пример 1

int x = int.Parse(Console.ReadLine()); //преобразование введенной строки в число.

Для преобразования строкового представления целого числа в тип int используем метод int.Parse(), который реализован для всех числовых типов данных. Таким образом, если потребуется преобразовать строковое представление в вещественное, можем воспользоваться методом float.Parse() или double.Parse(). В случае, если соответствующее преобразование выполнить невозможно, то выполнение программы прерывается.

В задачах вычислительного типа над переменными и константами могут выполняться арифметические операции и функции.

Выражения строятся из операндов (констант, переменных, функций) объединенных знаками операций и скобками. При вычислении выражения определяется его значение и тип. Эти характеристики однозначно задаются значениями и типами операндов, входящих в выражение, и правилами вычисления выражения. В таблице 1.1 представлены основные операции в порядке убывания приоритетов.

Таблица 1.1

Приоритет	Категория	Операция	Порядок
0	Первичные	f(x) a[x] x++ x-- new sizeof(t) typeof(t)	Слева направо
1	Унарные	+ - ! ~ ++x --x (T)x	Слева направо
2	Мультипликативные (умножение)	* / %	Слева направо
3	Аддитивные (сложение)	+ -	Слева направо
4	Сдвиг поразрядный	<< >>	Слева направо
5	Отношения, проверка типов	< > <= >= is as	Слева направо
6	Эквивалентность	== !=	Слева направо
7	Логическое И	&	Слева направо
8	Логическое исключающее ИЛИ (XOR)	^	Слева направо
9	Логическое ИЛИ (OR)	|	Слева направо
10	Условное И	&&	Слева направо
11	Условное ИЛИ	||	Слева направо
12	Условное выражение	?:	Справа налево
13	Присваивание	= *= /= %= += -= <<= >>= &= ^= |=	Справа налево

Одна из основных операций языка C# – это операция присваивания. В ка-

честве ее левого операнда может использоваться только модифицируемое именующее выражение, т. е. ссылка на некоторую именованную область памяти, значение которой доступно изменениям.

Формат операции простого присваивания (=): *операнд_2 = операнд_1;*

В результате выполнения этой операции вычисляется значение операнда_1 и результат записывается в операнд_2. Возможно связать воедино сразу несколько операторов присваивания, записывая такие цепочки: $a = b = c = 100$. Выражение такого вида выполняется справа налево: результатом выполнения $c = 100$ является число 100, которое затем присваивается переменной *b*, результатом чего опять является 100, которое присваивается переменной *a*.

Кроме простой операции присваивания, существуют сложные операции присваивания, например, умножение с присваиванием (*=), деление с присваиванием (/=), остаток от деления с присваиванием (%=), сложение с присваиванием (+=), вычитание с присваиванием (-=) и т. д.

Все сложные операции присваивания представляют собой сокращенную запись простого присваивания:

– для случая, который может быть описан как X = XÅY;

– то же, в записи составного присваивания X Å=Y.

В качестве символа Å могут быть использованы знаки операций: +; -; *; /; % и др.

Например, *f* += 5; то же самое, что и *f* = *f* + 5.

Различают пять операций, выполняющих арифметические действия над числами (+, -, *, /, %).

Выполнение первых трёх операций очевидно и не требует пояснения. А вот операция деления (/) зависит от типа данных. Если тип данных, участвующих в операции, вещественный, то результат получится по правилам деления с целой и дробной частью. Если тип целый, то результатом деления также является целое число.

Операция деления по модулю (%) возможно только для целых чисел. Оно обеспечивает получение остатка от деления двух целых чисел. Например, в результате операции 15 % 4 получится число 3. Таким образом, пара операций / и % обеспечивает для целых чисел специфическую возможность: можно получить и целую часть, и остаток от деления.

К арифметическим операциям относятся две специфические унарные операции – инкремент (++) и декремент (--). Эти операции позволяют изменять значения переменных на 1. Инкремент – увеличивает, обозначается знаком ++. Декремент – уменьшает, обозначается знаком --. По сути дела, эти операции являются сокращённой формой записи для выражений типа $i = i + 1$ и $i = i - 1$. Например, вместо $a = a + 1$; можно записать: *a*++; (постфиксный вариант) или ++*a*; (префиксный вариант). Аналогично: --Prim; Prim--;

Между двумя последними формами нет никакой разницы, если эти операции являются единственными в записи выражения. Если данные операции используются в сложном выражении, то при постфиксном варианте записи значение переменной сначала используется, а потом изменяется на 1. При префиксном, наоборот, сначала изменяется, а потом используется.

Например:

```
t = 5; // t = 5
n = 4 * t++; // n = 20, t = 6
k = ++n / 3; // n = 21, k = 7
b = --k / t;     // k = 6, b = 1
```

C# содержит большое количество встроенных математических функций, которые реализованы в классе Math пространства имен System и представлены в таблице 1.2.

Таблица 1.2

Название	Описание	Пример использования	Математическая запись
Abs()	Модуль	Math.Abs(x-y)	$\lvert x-y\rvert$
Cos( )	Косинус	Math.Cos(Math.PI/2)	$\cos\frac{\pi}{2}$
E	Число e	Math.E	$\approx 2,71828$
Exp( )	Экспонента	Math.Exp(x-2)	e^{x-2}
Log( )	Натуральный логарифм	Math.Log(x)/ Math.Log(y,2)	$\frac{\ln x}{\log_2 y}$
Log10( )	Десятичный логарифм	Math.Log10(y/x)	$\lg\frac{y}{x}$
Max(,)	Максимум из двух значений	Math.Max(x , y)	$\max(x,y)$
Min(,)	Минимум из двух значений	Math.Min(x-y,x+y)	$\min(x-y,x+y)$
PI	Число π	Math.PI	π
Pow(,)	Возведение в степень	Math.Pow(x,2)	x^2
Round( )	Простое округление	Math.Round(-2.346) Math.Round(-2.346,2)	-2.346≈-2 -2.346≈-2.35
Sign( )	Знак числа	Math.Sign(-2.346)	-1
Sin( )	Синус (выражение в радианах)	Math.Pow(Math.Sin(2*a,3)	$\sin^3 2a$
Sqrt( )	Квадратный корень	Math.Sqrt(x-2)	$\sqrt{x-2}$
Tan( )	Тангенс	1/Math.Tan(x)	$1/\mathrm{tg}\,x=\mathrm{ctg}\,x$

Для условного обозначения алгоритмических операций применяются схемы алгоритма (таблица 1.3).

Алгоритм – это определенным образом организованная последовательность действий, за конечное число шагов приводящая к решению задачи.

Свойства алгоритмов:

– определенность;

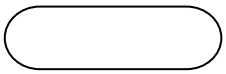
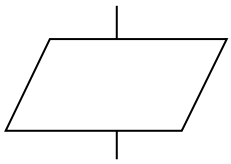
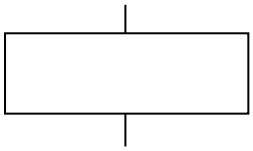
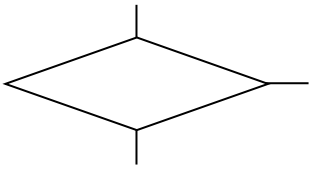
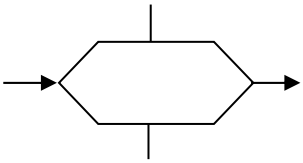
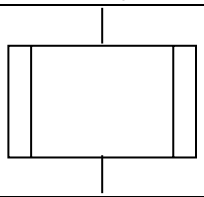
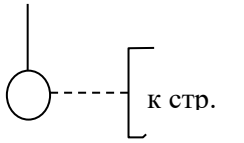
– дискретность;

– целенаправленность;

– конечность;

– массовость.

Таблица 1.3

Символ	Значение	Применение
	Завершение	Начало, конец обработки данных или выполнения программы
	Данные	Обозначает ввод, вывод данных
	Процесс	Обработка данных любого вида (выполнение операции или группы операций)
	Решение	Выбор направления выполнения программы в зависимости от некоторых переменных условий
	Подготовка	Описывается подготовка данных для выполнения повторяющихся действий
	Типовой процесс	Одна или несколько операций, которые определены в другой программе, модуле
	Соединитель (узел)	Используется при разрыве линий схемы алгоритма

Потоки данных и потоки управления отображаются линиями. Направления слева направо и сверху вниз являются стандартными, нестандартные направления обозначаются стрелками. Следует избегать пересечений линий.

Пример 2 – Вычислить значение функции

$$f(x,y)=\frac{|x|+\sin^3(y+5)}{x+\frac{1}{3}}.$$

Перед написанием любой программы надо четко определить, что в нее требуется ввести и что мы должны получить в результате.

В данном случае:

– в качестве исходных данных выступают два вещественных числа x и y;

– в качестве результата – значение функции, вещественное число f.

Текст программы:

```
using System;
namespace Primer1
{
  class Program
  {
    static void Main(string[] args)
    {
      double x, y, f;  //объявление переменных вещественного типа
      Console.WriteLine("Введите x и y:"); //вывод приглашения к вводу
      x = double.Parse(Console.ReadLine()); //ввод исходных данных
      y = double.Parse(Console.ReadLine());
      f = (Math.Abs(x) + Math.Pow(Math.Sin(y + 5), 3)) / (x + 1.0 / 3); //вычисление результата
      Console.WriteLine("Результат: f({0:f2},{1:f2})={2:f4}", x, y, f);  //вывод результата
      Console.ReadKey();//ожидание нажатия любой клавиши для задержки экрана
    }
  }
}
```

Схема алгоритма программы имеет вид, представленный на рисунке 1.2.

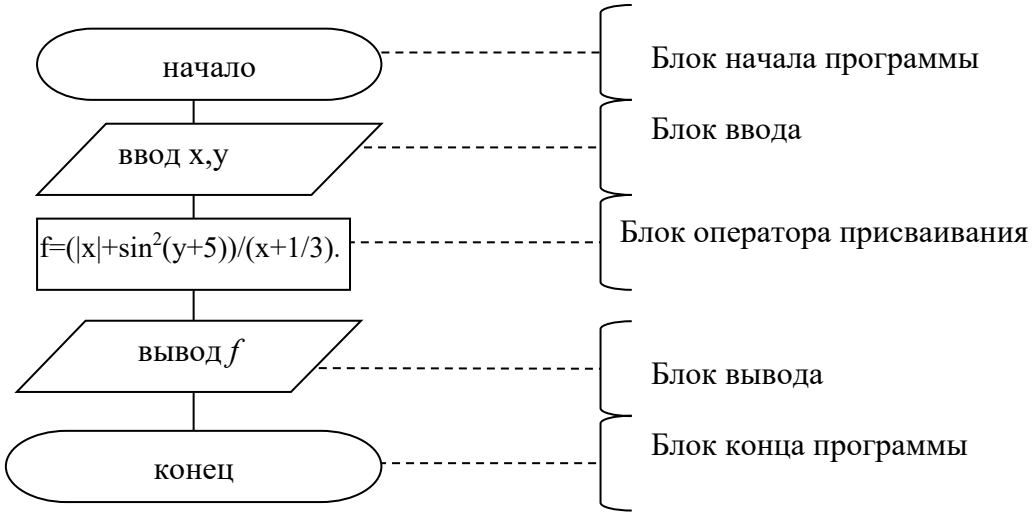


Рисунок 1.2

Задание для самостоятельного решения

Даны x, y, z. Вычислить a, b:

1) $a = \dfrac{1+\sin(z-2)}{y^3/3+\cos^2 x}$; $\quad b = \dfrac{1+\sin^2(x+y)}{2+\left|x-2y/\left(1+x^2y^2\right)\right|}$;

2) $a = \ln\left|\left(y-\sqrt{|z|}\right)\cdot\left(x-\dfrac{y}{x+z^2/4}\right)\right|$; $\quad b = x-\dfrac{y^2}{2}+\dfrac{z^3}{3}$;

3) $a=\dfrac{1+\sin^2(x+y)}{2+\left|x-2x/\left(1+x^2y^2\right)\right|}$; $b=\cos^2\left(\operatorname{arctg}\left(\dfrac{1}{z}\right)\right)$;

4) $a=\dfrac{2\cos(x-\pi/6)}{1/2+\sin^2 y}$; $b=1+\dfrac{z^2}{3+z^2/5}$;

5) $a=y+\dfrac{x}{y^2+\left|\dfrac{x^2}{y+x^3/3}\right|}$; $b=\left(1+\operatorname{tg}^2\dfrac{z}{2}\right)^2$;

6) $a=(1+y)\dfrac{x+y/\left(x^2+4\right)}{e^{-x-2}+1/\left(x^2+4\right)}$; $b=\dfrac{1+\cos(y-2)}{x^4/2+\sin^2 z}$;

7) $a=\dfrac{3+e^{y-1}}{1+x^2\left|y-\operatorname{tg}z\right|}$; $b=1+\left|y-x\right|+\dfrac{(y-x)^2}{2}+\dfrac{\left|y-x\right|^3}{3}$;

8) $a=\dfrac{\sqrt{\left|x-1\right|}-\sqrt[3]{\left|y\right|}}{1+\dfrac{x^2}{2}+\dfrac{y^2}{4}}$; $b=x\left(\operatorname{arctg}z+e^{-(x+3)}\right)$;

9) $a=\dfrac{1+\cos^2(z-2)}{(y-1)^4/3+\operatorname{tg}x^2}$; $b=\operatorname{tg}^3\left(\arcsin\left(\dfrac{1}{x}\right)\right)$;

10) $a=y\left(\arccos z+e^{-(x+3)}\right)$; $b=\dfrac{2\sin(x-\pi/6)}{1/2+\log_2\left|y\right|}$.

2 Программирование разветвляющихся алгоритмов. Оператор if

Большинство операторов управления программой в любых языках программирования, включая C#, основываются на проверке условий, определяющих, какого рода действие необходимо выполнить. В результате проверки условий можно получить истину или ложь. В противоположность другим языкам, где вводится специальный тип для хранения истины и лжи, в C# истине соответствует любое ненулевое значение, включая отрицательные числа. Лжи соответствует ноль.

Стандартная форма записи оператора **if** следующая:

if *(выражение)*
оператор;
else
оператор;

где *оператор* может быть простым или составным (надо помнить, что в C# *составной оператор* – это группа операторов, заключенных в фигурные скобки).

Оператор **else** не обязателен.

В условии могут использоваться следующие операции.

Операции отношений (сравнения):

< – меньше, чем;

> – больше, чем;

<= – меньше или равно, чем;

>= – больше или равно, чем;

== –равно;

!= – не равно.

Операнды в этих операциях должны быть арифметического типа. Результат операции логический: false (ложь) или true (истина).

Логические операции:

&& – конъюнкция (И) арифметических операндов или отношений. Результат истина, если два операнда имеют значение истина;

|| – дизъюнкция (ИЛИ) арифметических операндов или отношений. Результат истина, если хотя бы один из операндов истина.

Примеры отношений и логических операций:

4 < 9 (≡ true); 3 = = 5 (≡ false);
3 != 5 || 3==5 (≡ true); (3+4>5) && (3+5>4) && (4+5>3) (≡ true).

Стандартная форма оператора **if** с составными операторами следующая:

if (*выражение*)
{
последовательность операторов1
}
else
{
последовательность операторов2
}

Если выражение истинно, выполняется блок операторов, следующий за **if**; иначе выполняется блок операторов, следующий за **else**. Всегда выполняется код, ассоциированный или с **if**, или с **else**, но никогда не выполняются два кода одновременно.

Оператор **if** на схеме алгоритма представлен на рисунке 2.1.

Вложенные операторы if. Типичной программистской конструкцией является *вложенные if*. Данная конструкция выглядит следующим образом:

if (*выражение*)
оператор;
else if (*выражение*)
оператор;
else if (*выражение*)

оператор;
…
else
оператор.

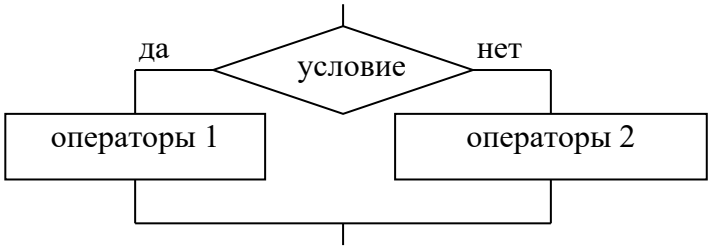


Рисунок 2.1

Условия вычисляются сверху вниз. Когда обнаруживается истинное условие, то выполняется оператор, связанный с этим условием, а остальная часть конструкции игнорируется. Если не найдено ни одного истинного условия, выполняется оператор, соответствующий последнему **else**. Последний оператор **else** часто играет роль *оператора*, *выполняемого по умолчанию*, т. е., если все условия ложны, то выполняется оператор, соответствующий последнему **else**. Если последний оператор **else** отсутствует, то не выполняется никаких действий в случае ложности всех условий.

При написании программы с использованием условного оператора обратите внимание на следующие моменты:

– условие должно быть в круглых скобках;

– после условия точка с запятой не ставится (если это не пустой оператор);

– если к *if* или к *else* относится более одного оператора, то они объединяются в операторные скобки { };

– в условии проверки на равенство должна использоваться операция сравнения (==);

– условие принадлежности диапазону $a < x < b$ записывается в виде

$$\textbf{\textit{if}}(x > a \;\&\&\; x < b) \ldots .$$

Обработка исключений. Язык C# реагирует на ошибки и нештатные ситуации с помощью механизма обработки исключений. Исключение – это объект, генерирующий информацию о «необычном программном происшествии». При этом важно проводить различие между ошибкой в программе, ошибочной ситуацией и исключительной ситуаций.

Ошибка в программе допускается программистом при ее разработке. Например, вместо операции сравнения (==) используется операция присваивания (=).

Ошибочная ситуация вызвана действиями пользователя. Например, пользователь вместо числа ввел строку. Такая ошибка способна вызывать исключение. Для обработки ошибочных и исключительных ситуаций в C# используется специальная подсистема обработки исключений.

Управление обработкой исключений основывается на использовании оператора try.

Синтаксис оператора:

```
try
{
…// контролируемый блок
}
catch
{
…//один или несколько блоков обработки исключений
}
finally
{
…//блок завершения
}
```

В блок try помещаются программные инструкции, которые нужно проконтролировать на предмет исключений. Если исключение возникает в этом блоке, оно будет перехвачено и обработано с помощью блока catch. Любой код, который должен быть обязательно выполнен при выходе из блока try, помещается в блок finally. Рассмотрим пример, демонстрирующий, как отследить и перехватить исключение.

```
static void Main(string[] args)
{
   try
   {
      int x = int.Parse(Console.ReadLine());// возможен ввод нечисловых значений
      int y = 1 / x; //ошибка при делении на 0
      Console.WriteLine("y={0}", y);
      Console.WriteLine("Успешное завершение программы.");
   }
   catch (Exception e)       // *
   {
      Console.WriteLine("Ошибка: " + e.Message);
   }
   Console.ReadKey();
}
```

Рассмотрим, как обрабатываются исключения в данном примере. Когда возникает исключение, выполнение программы останавливается и управление передается блоку catch. Поэтому команды из блока try, расположенные ниже строки, в которой возникло исключение, никогда не будут выполнены.

Обработчик исключений позволяет не только отловить ошибку, но и вывести информацию о ней.

Пример **–** Найти корни квадратного уравнения вида $ax^2 + bx + c = 0$. Если корней нет, вывести соответствующее сообщение.

В данной задаче:

– в качестве исходных данных выступают три вещественных числа *a*, *b* и *c* (коэффициенты квадратного уравнения);

– в качестве результата – корни квадратного уравнения, вещественные числа *x*1 и *x*2.

Текст программы:

```
static void Main(string[] args)
{
   try // контролируемый блок
   {
      double a, b, c, x1, x2, D;  //объявление переменных вещественного типа
      Console.WriteLine("Введите a,b и c:"); //вывод приглашения к вводу
      a = double.Parse(Console.ReadLine()); //ввод исходных данных
      b = double.Parse(Console.ReadLine());
      c = double.Parse(Console.ReadLine());
      D = Math.Pow(b, 2) + 4 * a * c; //вычисление дискременанта
      if (D > 0)
      {
         x1 = (–b – Math.Sqrt(D)) / (2 * a);
         x2 = (–b + Math.Sqrt(D)) / (2 * a);
         //вывод результата
         Console.WriteLine("Уравнение имеет два корня x1={0:f3}, x2= {1:f3}", x1, x2);
      }
      else if (D == 0)
         //вывод результата
         Console.WriteLine("Уравнение имеет корень x1= " + (-b / (2 * a)));
      else
         Console.WriteLine("Уравнение не имеет корней"); //вывод результата
   }
   catch(Exception err)//блок обработки исключений
   {
      Console.WriteLine("Ошибка: {0}", err.Message);
   }
   Console.ReadKey();//ожидание нажатия любой клавиши для задержки экрана
}
```

Схема алгоритма программы имеет вид, представленный на рисунке 2.2.

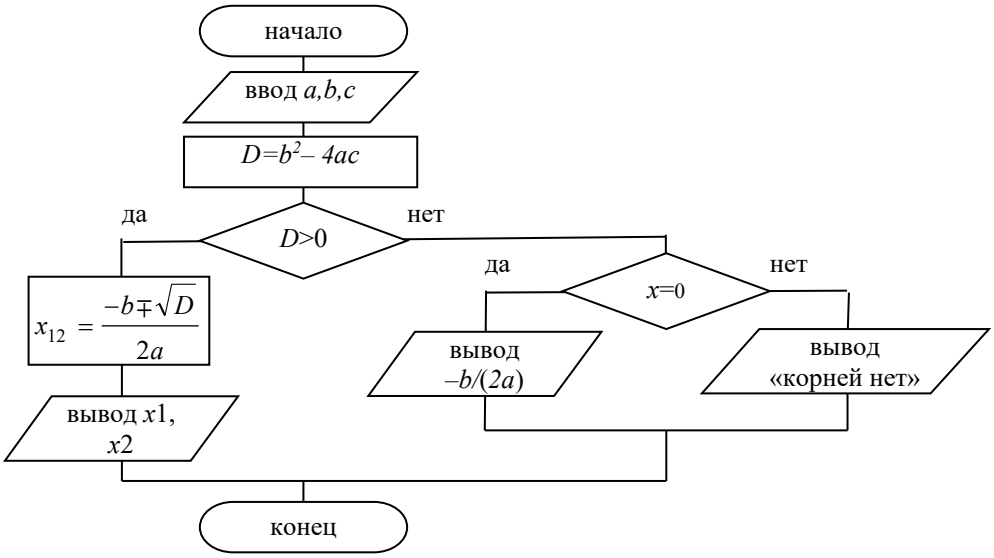


Рисунок 2.2

Задание для самостоятельного решения

Написать программу для вычисления значения функции, в зависимости от выполнения условий:

1) $y=\begin{cases}\sin(x+2), & x<1{,}35;\\ x+3{,}5\operatorname{tg}x, & 2\le x\le 4;\\ \sqrt{|\,2{,}56x-0{,}35\,|}, & \text{в других случаях;}\end{cases}$

2) $y=\begin{cases}5{,}6(1+\operatorname{tg}x), & x<0{,}5\pi;\\ \sin x+6, & \pi\le x\le 3\pi;\\ 2{,}56\sqrt{x^{2x}+1}, & \text{в других случаях;}\end{cases}$

3) $y=\begin{cases}6(\cos^2 x-\sin^2 x), & x<-4;\\ \sqrt{3{,}2x^2+\operatorname{tg}^2 x}, & -1\le x<2;\\ e^{\cos(2{,}58x)}, & \text{в других случаях;}\end{cases}$

4) $y=\begin{cases}\operatorname{tg}(|\,2x+4{,}2\,|)-\lg|\,x\,|, & x<1;\\ \sin x+\sqrt{6x}, & 2\le x\le 5;\\ 6+\operatorname{arctg}\left(\dfrac{2x}{1+\sqrt{x}}\right), & \text{в других случаях;}\end{cases}$

5) $y=\begin{cases}e^{|\sin(x)|}\operatorname{tg}(2{,}3+x), & x<-1;\\ \operatorname{arctg}x, & -1\le x\le 0{,}5;\\ -1+\sqrt{23{,}5x}, & \text{в других случаях;}\end{cases}$

6) $y=\begin{cases}3e^{\sin x}\operatorname{tg}2x, & x\le 2;\\ 2{,}47\lg x+x^{2x}, & 3<x<4;\\ \sqrt{|\cos^2 x|}+4, & \text{в других случаях;}\end{cases}$

7) $y=\begin{cases}\sqrt{1+2{,}4x^2}, & x<1;\\ \lg 5{,}9x, & 2\le x\le 3;\\ \sqrt{x^2+1}+2x, & \text{в других случаях;}\end{cases}$

8) $y=\begin{cases}2+x^{2+x}, & x<0{,}1\pi;\\ \sin^2(x^2+0{,}5), & \pi\le x\le 1{,}5\pi;\\ \sin^2\left(\dfrac{2{,}45}{\sqrt{x-1{,}2}}\right), & \text{в других случаях;}\end{cases}$

9) $y=\begin{cases} -\dfrac{\operatorname{tg}(x^{2x})+x^2}{\cos(x+2)}, & x<-5; \\ 12+5x, & 0\le x<5; \\ \operatorname{tg}2x\sqrt{\left|x^{2-x}+x^2\right|}, & \text{в других случаях;} \end{cases}$

10) $y=\begin{cases} \dfrac{e}{\operatorname{tg}(2+x)}, & x<1; \\ \ln(|\,(x+2)^2-x^2\,|), & 2\le x\le 4; \\ \sin^2\left(\dfrac{x-6}{3}\right), & \text{в других случаях.} \end{cases}$

3 Операторы цикла

Операторы цикла используются для организации многократно повторяющихся вычислений. Любой цикл состоит из тела цикла, т. е. тех операторов, которые выполняются несколько раз, начальных установок, модификации *параметра* цикла и проверки условия продолжения выполнения цикла. Один проход цикла называется итерацией.

Все циклы в C# выполняют тело цикла, пока условие истинно.

Стандартный вид цикла **for** следующий:

for (инициализация; условие; модификация)
тело_цикла.

Оператор **for** имеет три главные части.

Инициализация – это место, где обычно находится оператор присваивания, используемый для установки начального значения переменной цикла.

Условие – это место, где находится выражение, определяющее условие работы цикла.

Модификация – это место, где определяется характер изменения переменной цикла на каждой итерации.

Эти три важные части должны разделяться точкой с запятой. Цикл **for** работает до тех пор, пока условие истинно. Когда условие становится ложным, выполнение программы продолжается с оператора за циклом **for**.

Порядок выполнения. Переменной *счётчика цикла* присваивается начальное значение (инициализация) и проверяется условие; если условие неверно, то *тело цикла* не выполняется и управление передается на оператор, следующий за конструкцией **for**. Если же условие выполняется, то выполняется *тело цикла*, затем изменяется значение *счётчика цикла* и снова проверяется условие. Данный процесс будет выполняться, пока условие не станет ложным.

Рассмотрим принцип работы оператора на примере, где осуществляется вывод чисел от 1 до 4 включительно (рисунок 3.1).

В данной программе переменная **x** изначально установлена в 1. Поскольку **x** меньше 4, выводится значение 1, после чего **x** увеличивается на 1 и проверяется условие: по-прежнему ли **x** меньше либо равно 4. Данный процесс продолжается до тех пор, пока **x** не станет больше 4, и в этот момент цикл прервется. В данном примере **x** является *переменной цикла*, которая изменяется и проверяется на каждой итерации цикла.

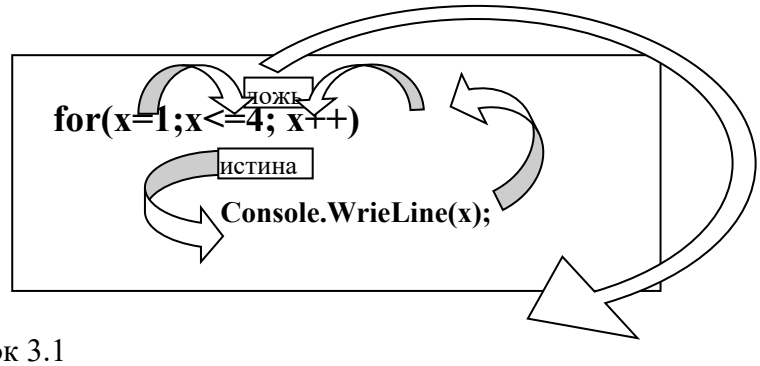


Рисунок 3.1

Изображение в блок-схемах показано на рисунке 3.2.

Рисунок 3.2

При написании программы с использованием оператора цикла **for** обратите внимание на следующие моменты:

– если тело цикла состоит более чем из одного оператора, то они заключаются в операторные скобки;

– после условий цикла точка с запятой не ставится (если тело цикла не пустой оператор);

– одна или все из частей оператора for могут отсутствовать, но точки с запятой надо оставить на своих местах.

Пример 1 – Построить таблицу значений для функции

$$f(x)=\begin{cases}\sqrt{|ax|}, & \text{если } x\le 0 \text{ и } a=b;\\ a\sin bx, & \text{если } 0<x\le b \text{ или } a \text{ не равно } 0;\\ 0, & \text{в остальных случаях.}\end{cases}$$

Значения a, b, $X_{нач}$, $X_{кон}$, dX ввести с клавиатуры.

Исходными данными являются начальное значение аргумента *Xn*, конечное значение аргумента *Xk*, шаг изменения аргумента *dX* и параметры *a*, *b*, *c*. Все величины – вещественные. Программа должна выводить таблицу, состоящую из двух столбцов – значений аргумента *x* и соответствующих им значений функции *f*.

Текст программы.

```
static void Main(string[] args)
{
```

```
try
 {
  //объявление переменных вещественного типа
  double a, b, Xn, Xk, dX, x, f;
  //ввод исходных данных
  Console.WriteLine("Введите коэффициенты a и b:");
  a = double.Parse(Console.ReadLine());
  b = double.Parse(Console.ReadLine());
  Console.WriteLine("Введите Хнач, Хкон и шаг dX:");
  Xn = double.Parse(Console.ReadLine());
  Xk = double.Parse(Console.ReadLine());
  dX = double.Parse(Console.ReadLine());
  for (x = Xn; x <= Xk; x += dX)  //условия цикла
  {
     if (x <= 0 && a == b)
        f = Math.Sqrt(Math.Abs(a * x));
     else if ((x > 0 && x <= b) || a != 0)
        f = a * Math.Sin(b * x);
     else
        f = 0;
     //вывод
     Console.WriteLine("x= {0,5:f2}, f(x)= {1,8:f4}",x,f);
  }
 }
catch(Exception e)
{          Console.WriteLine("Ошибка: {0}", err.Message);        }
 Console.ReadKey();
}
```

Схема алгоритма программы имеет вид, представленный на рисунке 3.3.

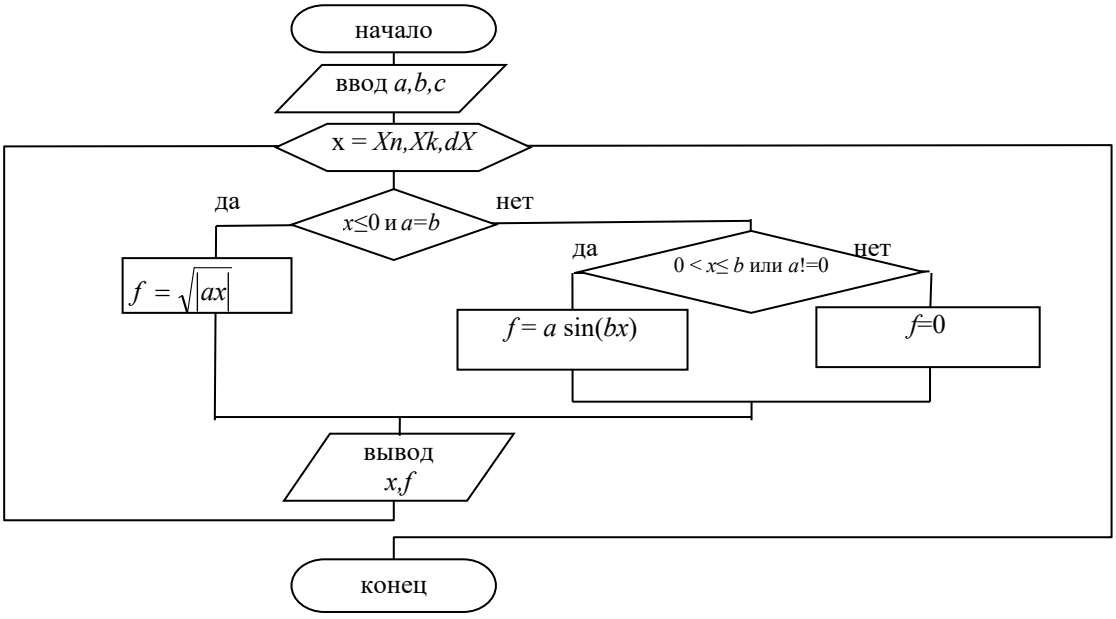


Рисунок 3.3

Задание 1 для самостоятельного решения

Для задач 1–10 вычислить и вывести на экран в виде таблицы значения функции F на интервале от $X_{нач}$ до $X_{кон}$ с шагом dX. Значения a, b, c, $X_{нач}$, $X_{кон}$, dX (действительные числа) ввести с клавиатуры.

1) $F=\begin{cases} ax^2+b & \text{при } x<0 \text{ и } b\neq 0; \\ \dfrac{x-a}{x-c} & \text{при } x>0 \text{ и } b=0; \\ \dfrac{x}{c} & \text{в остальных случаях.} \end{cases}$

2) $F=\begin{cases} ax^2+bx+c & \text{при } a<0 \text{ и } c\neq 0; \\ \dfrac{-a}{x-c} & \text{при } a>0 \text{ и } c=0; \\ a(x+c) & \text{в остальных случаях.} \end{cases}$

3) $F=\begin{cases} a-\dfrac{x}{10+b} & \text{при } x-a<0 \text{ и } c\neq 0; \\ \dfrac{x-a}{x-c} & \text{при } x-a>0 \text{ и } b=0; \\ 3x+\dfrac{2}{c} & \text{в остальных случаях.} \end{cases}$

4) $F=\begin{cases} -ax^2-b & \text{при } x<5 \text{ и } c\neq 0; \\ \dfrac{x-a}{x} & \text{при } x>5 \text{ и } c=0; \\ \dfrac{-x}{c} & \text{в остальных случаях.} \end{cases}$

5) $F=\begin{cases} ax^2+b^2x & \text{при } a<0 \text{ и } x\neq 0; \\ x-\dfrac{a}{x-c} & \text{при } a>0 \text{ и } x=0; \\ 1+\dfrac{x}{c} & \text{в остальных случаях.} \end{cases}$

6) $F=\begin{cases} \dfrac{1}{ax}-b & \text{при } x+5<0 \text{ и } c=0; \\ \dfrac{x-a}{x} & \text{при } x+5>0 \text{ и } x\neq 0; \\ \dfrac{10x}{c-4} & \text{в остальных случаях.} \end{cases}$

7) $F=\begin{cases} -ax-c & \text{при } c<0 \text{ и } x\neq 0; \\ \dfrac{x-a}{a-c} & \text{при } c>0 \text{ и } x=0; \\ bx(c-a) & \text{в остальных случаях.} \end{cases}$

8) $F=\begin{cases} ax^2+b^2x & \text{при } c<0 \text{ и } b\neq 0; \\ \dfrac{x+a}{x+c} & \text{при } c>0 \text{ и } b=0; \\ \dfrac{ax}{cb^2} & \text{в остальных случаях.} \end{cases}$

9) $F=\begin{cases} -ax^2 & \text{при } c<0 \text{ и } a\neq 0; \\ \dfrac{a-x}{cx} & \text{при } c>0 \text{ и } a=0; \\ \dfrac{-x}{c-ax} & \text{в остальных случаях.} \end{cases}$

10) $F=\begin{cases} ax^2-bx+c & \text{при } x<3 \text{ и } b\neq 0; \\ \dfrac{x-a}{x-c} & \text{при } x>3 \text{ и } b=0; \\ \dfrac{x}{c} & \text{в остальных случаях.} \end{cases}$

Цикл с предусловием имеет вид:

while (условие) тело_цикла;

Условие определяет условие повторения тела цикла, представленного простым или составным оператором. Выполнение оператора начинается с вычисления условия. Если оно истинно (не равно false), выполняется тело цикла. Если при первой проверке выражение равно false, цикл не выполнится ни разу. Тип условия должен быть арифметическим или приводимым к нему. Условие вычисляется перед каждой итерацией цикла.

Изображение **while** в блок-схемах показано на рисунке 3.4.

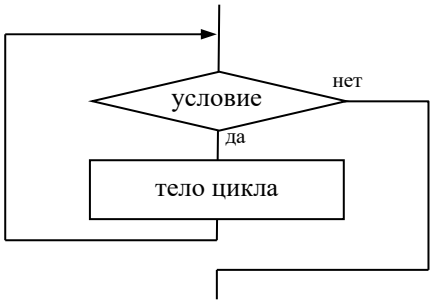


Рисунок 3.4

Пример 2 – Найти сумму цифр введенного целого числа.

Исходными данными является целое число *n*, выходными данными будет сумма цифр – целое число sum.

Для выделения цифр в числе воспользуемся следующим алгоритмом:

1) остаток от деления на 10 дает последнюю цифру числа;
2) при делении на 10 последняя цифра числа отбрасывается;
3) будем выполнять действия 1 и 2 и находить сумму выделяемых цифр, пока в числе не будут отброшены все цифры, т. е. пока число будет больше 0.

На рисунке 3.5 представлены текст и схема алгоритма программы.

```
static void Main(string[] args)
{
 try
 {
  int n, sum;
  Console.WriteLine("Введите число:");
  n = int.Parse(Console.ReadLine());
  sum = 0;
  while (n > 0) //условие цикла
  { //увеличиваем sum на последнюю цифру
   sum += n % 10;
   //отбрасываем последнюю цифру
   n /= 10;
  }
  Console.WriteLine("Сумма цифр: {0:d}", sum);
 }
 catch (Exception e)
 {
     Console.WriteLine("Ошибка " + e);
 }
 Console.ReadKey();
}
```

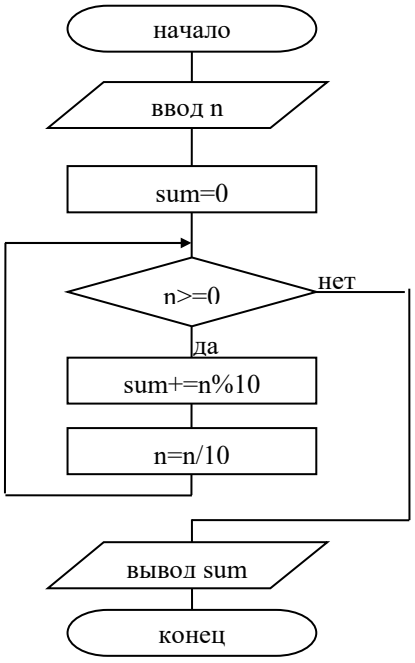


Рисунок 3.5

В противоположность циклам for и while, сначала проверяющим условие, цикл do...while проверяет условие в конце. То есть цикл do..while всегда выполняется, по крайней мере, один раз. Стандартный вид цикла do/while следующий:

```
do
{
  тело_цикла;
}
while (условие);
```

Сначала выполняется простой или составной оператор, составляющий тело цикла, а затем вычисляется выражение. Если оно истинно (не равно false), тело цикла выполняется еще раз. Цикл завершается, когда выражение станет равным false или в теле цикла будет выполнен какой-либо оператор передачи управления.

Изображение **do...while** в блок-схемах показано на рисунке 3.6.

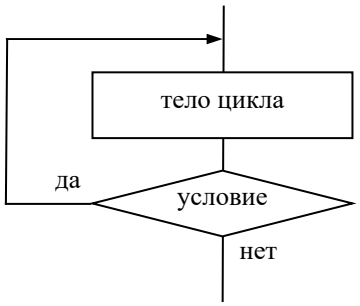


Рисунок 3.6

Пример 3 – Написать программу вычисления значения функции $\cos x$ с помощью бесконечного ряда Тейлора с точностью ε по формуле

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{2n!} + \ldots .$$

Для достижения заданной точности необходимо суммировать члены ряда до тех пор, пока очередной член по модулю не будет меньшим ε.

Воспользуемся рекуррентной формулой для получения последующего члена ряда через предыдущий:

$$C_{n+1} = T \cdot C_n ,$$

где T – некоторый множитель.

Найдем его:

$$C_n = \frac{(-1)^n x^{2n}}{(2n)!};$$

$$C_{n+1} = \frac{(-1)^{n+1} x^{2(n+1)}}{(2(n+1))!};$$

$$T = \frac{C_{n+1}}{C_n} = \frac{(-1)^{n+1} x^{2(n+1)} (2n)!}{(-1)^n x^{2n} (2(n+1))!} = -\frac{x^2}{(2n+1)(2n+2)}.$$

На рисунке 3.7 представлена схема алгоритма программы.

```
static void Main(string[] args)
{
 try
 {
  const int MaxIter = 100;
  double x, eps;
  double Cn, y;// член ряда и сумма.
  int n; // количество итераций.
  Console.WriteLine("Введите x и точность:");
  x = double.Parse(Console.ReadLine());
  eps = double.Parse(Console.ReadLine());
  y = Cn = 1;
  n = 0;
  do
  {  //очередной член ряда
   Cn *= -x * x / ((2 * n + 1) * (2 * n + 2));
   y += Cn;//вычисление суммы
   n++;
   if (n > MaxIter)
   {
    Console.WriteLine("Ряд расходится.");
    break;//оператор выхода из цикла
   }
  }
  while (Math.Abs(Cn) > eps);
  if (Math.Abs(Cn) <= eps)
  {
   Console.WriteLine("cos ({0:f2})={1:f5}", x, y);
   Console.WriteLine("Для проверки: {0:f5}",
                                    Math.Cos(x));
  }
 }
 catch (Exception e)
 { Console.WriteLine("Ошибка: " + e.Message);}
 Console.ReadKey();
}
```

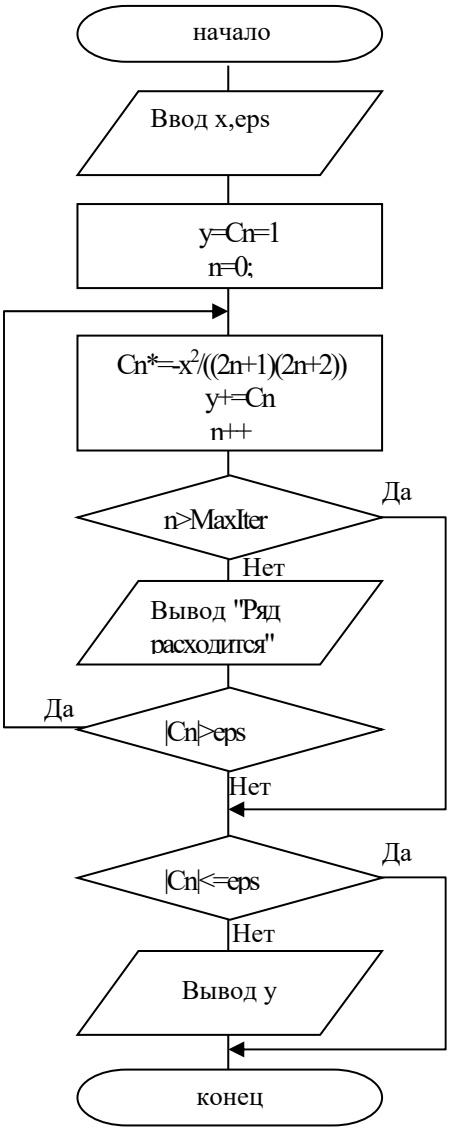


Рисунок 3.7

Задание 2 для самостоятельного решения

Вычислить и вывести на экран значение функции, заданной с помощью ряда Тейлора с точностью ε. Вывести значение функции и количество просуммированных членов ряда. Для вычисления последующего члена ряда использовать рекуррентную формулу:

1) $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots; \quad |x| < \infty;$

2) $e^{-x} = \sum_{n=0}^{\infty} \frac{(-1)^n x^n}{n!} = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \ldots + (-1)^n \frac{x^n}{n!} + \ldots; \quad |x| < \infty;$

3) $\ln\frac{x+1}{x-1}=2\sum_{n=0}^{\infty}\frac{1}{(2n+1)x^{2n+1}}=2(1+\frac{1}{x}+\frac{1}{3x^3}+\frac{1}{5x^5}+\ldots;\quad |x|>1;$

4) $\sin x=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{(2n+1)!}=\frac{x}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\cdots;\quad |x|<\infty;$

5) $\cos x=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n}}{(2n)!}=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}-\cdots;\quad |x|<\infty;$

6) $\ln(x+1)=\sum_{n=0}^{\infty}(-1)^n\frac{x^{n+1}}{n+1}=\frac{x}{1}-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\cdots;\quad -1<x\le 1;$

7) $\ln\frac{1+x}{1-x}=2\sum_{n=0}^{\infty}\frac{x^{2n+1}}{2n+1}=2(x+\frac{x^3}{3}+\frac{x^5}{5}+\frac{x^7}{7}+\ldots;\quad |x|<1;$

8) $\operatorname{arcctg}x=\frac{\pi}{2}+\sum_{n=0}^{\infty}(-1)^{n+1}\frac{x^{2n+1}}{2n+1}=\frac{\pi}{2}-\frac{x}{1}+\frac{x^3}{3}-\frac{x^5}{5}+\cdots;\quad |x|\le 1;$

9) $\operatorname{arctg}x=\frac{\pi}{2}+\sum_{n=0}^{\infty}\frac{(-1)^{n+1}}{(2n+1)x^{2n+1}}=\frac{\pi}{2}-\frac{1}{x}+\frac{1}{3x^3}-\frac{1}{5x^5}+\cdots;\quad |x|\le 1;$

10) $\ln(1-x)=-\sum_{n=1}^{\infty}\frac{x^n}{n}=-\left(\frac{x}{1}+\frac{x^2}{2}+\frac{x^3}{3}+\frac{x^4}{4}+\cdots\right);\quad -1\le x<1.$

4 Массивы

Массив – это совокупность переменных одного типа, к которым обращаются с помощью общего имени. Доступ к отдельному элементу массива может осуществляться с помощью индекса. C#-массивы относятся к ссылочным типам данных и реализованы как объекты. Фактически имя массива является ссылкой на область динамической памяти, в которой последовательно размещается набор элементов определенного типа. Выделение памяти под элементы происходит на этапе инициализации массива. А за освобождением памяти следит система сборки «мусора». Массивы могут иметь одну или несколько размерностей.

Одномерный массив. Одномерный массив – это фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент имеет свой номер. Нумерация элементов массива в C# начинается с нуля, т. е., если массив состоит из 10 элементов, то его элементы будут иметь индексы от 0 до 9.

Одномерный массив в C# реализуется как объект, поэтому его создание представляет собой двухступенчатый процесс (таблица 4.1):

1) объявляется ссылочная переменная на массив;

2) выделяется память под требуемое количество элементов базового типа и ссылочной переменной присваивается адрес нулевого элемента в массиве.

Базовый тип определяет тип данных каждого элемента массива. *Размер* определяет количество элементов в массиве и может быть целым числом или

целочисленной переменной.

Таблица 4.1

Форма записи	Пояснение
тип_элементов [] имя_массива; *Например*: *int [ ] a;* *n=int.Parse(Console.ReadLine());* *a=new int[n];*	Описана ссылка на одномерный массив, которая в дальнейшем может быть использована: для адресации на уже существующий массив передачи массива в метод в качестве параметра отсроченного выделения памяти под элементы массива
тип_элементов [] имя_массива = new тип_элементов [размер]; *Например*: *int []a=new int [10];*	Объявлен одномерный массив заданного типа и выделена память под одномерный массив указанной размерности. Адрес данной области памяти записан в ссылочную переменную. Элементы массива равны нулю
тип_элементов [] имя_массива ={список инициализации}; *Например*: *int []a={4,-1, 3, -2, 6};*	Выделена память под одномерный массив, размерность которого соответствует количеству элементов в списке инициализации. Адрес этой области памяти записан в ссылочную переменную. Значение элементов массива соответствует списку инициализации

Для доступа к элементу массива используется следующий синтаксис:

имя_массива[*индекс*];

где *индекс* – целое число или целочисленная переменная из диапазона [0; *размер* – 1].

Для работы с массивами используются циклы. Следующий фрагмент программы вводит вещественный массив с клавиатуры и выводит его на дисплей:

```
double[] x=new double [10];  // резервирует место для 10 вещественных элементов
int i;
for (i = 0; i < 10; i++) //ввод элементов массива в цикле
{   Console.Write("Введите %d-й элемент массива: ",i+1);
    x[i]=double.Parse(Console.ReadLine());          }
//вывод массива на экран
for (i = 0; i < 10; i++)
    Console.Write("{0:f3}\t", x[i]);
```

Выход за границы массива в C# расценивается как ошибка, в ответ на которую генерируется исключение – *IndexOutOfRangeException.*

Пример 1 – В последовательности действительных чисел найти количество положительных элементов и обменять минимальный элемент с первым.

Для поиска количества положительных элементов используется следующий алгоритм: просматриваем поочередно все элементы и если элемент массива больше нуля, увеличить счетчик элементов на единицу.

Чтобы поменять местами минимальный и первый элемент массива, необходимо найти местоположения минимального элемента, т. е. его индекс. Для этого необходимо провести следующие операции.

1 Задать начальные значения для индексов минимального элемента (например, равные нулю).

2 Просмотреть массив, поочередно сравнивая каждый его элемент с ранее найденным минимумом. Если очередной элемент меньше минимума, принять этот элемент за новый минимум (т. е. запомнить его индекс).

Для обмена необходимо использовать дополнительную переменную. Процесс обмена проиллюстрируем на рисунке 4.1.

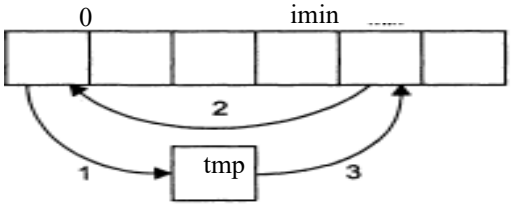


Рисунок 4.1

На рисунке 4.2 представлена схема алгоритма программы.

Класс Random и его методы. Класс Random служит для генерирования случайных чисел. Для того чтобы вызывать методы класса, нужно вначале создавать экземпляр (объект) класса.

Random rnd = new Random();

Конструктор с параметром целого типа – Random(int) – обеспечивает важную возможность генерирования повторяющейся серии случайных чисел.

Метод **Next**() при каждом вызове возвращает положительное целое, равномерно распределённое в некотором диапазоне. Диапазон задаётся параметрами метода:

Next () – целые положительные числа в диапазоне [0, int.MaxValue];

Next (max) – целые положительные числа в диапазоне [0, max];

Next (min, max) – целые положительные числа в диапазоне [min, max].

Метод **NextDouble**() возвращает новое случайное число, равномерно распределённое в интервале [0, 1].

Например, одномерный массив, размер которого случайное число от 10 до 100, заполнить действительными случайными числами из диапазона [*a, b*]:

```
 Random rnd = new Random();
double[] m = new double[rnd.Next(10,100)];     // объявление массива
Console.Write("a= ");
double a = double.Parse(Console.ReadLine());
Console.Write("b= ");
double b = double.Parse(Console.ReadLine());
for (int i = 0; i < m.Length; i++) //.Length - количество элементов массива
{
 m[i] = a+(b-a)*rnd.NextDouble();
  Console.Write("{0:f3}\t", m[i]);
}
Console.WriteLine();
```

Текст программы.

```
static void Main(string[] args)
{ try
  {
    Console.Write("Введите количество элементов: ");
    int n = int.Parse(Console.ReadLine());  //размер массива
    double[] b = new double[n];     // объявление массива
    double tmp;
    int i, imin, pol;
    // ввод массива
    for (i = 0; i < n; i++)
    {  Console.Write("b[{0}]=", i + 1);
      b[i] = double.Parse(Console.ReadLine());
    }
    //подсчет количества положительных
    pol = 0;
    for (i = 0; i < n; i++)
      if (b[i] > 0)
        pol++;
    // принимаем за наименьший первый из элементов
    imin = 0;
    for (i = 0; i < n; i++)
      // если нашли меньший элемент
      if (b[i] < b[imin])
        imin = i; //запоминаем его номер
    // обмен элементов b[0] и b[imin]:
    tmp = b[0];    //1
    b[0] = b[imin]; //2
    b[imin] = tmp; //3
    //вывод полученного массива
    for (i = 0; i < n; i++)
      Console.Write(b[i] + "\t");
    Console.WriteLine();
  }
  catch (Exception e)
  {
     Console.WriteLine("Ошибка: " + e.Message);
   }
  Console.ReadKey();
}
```

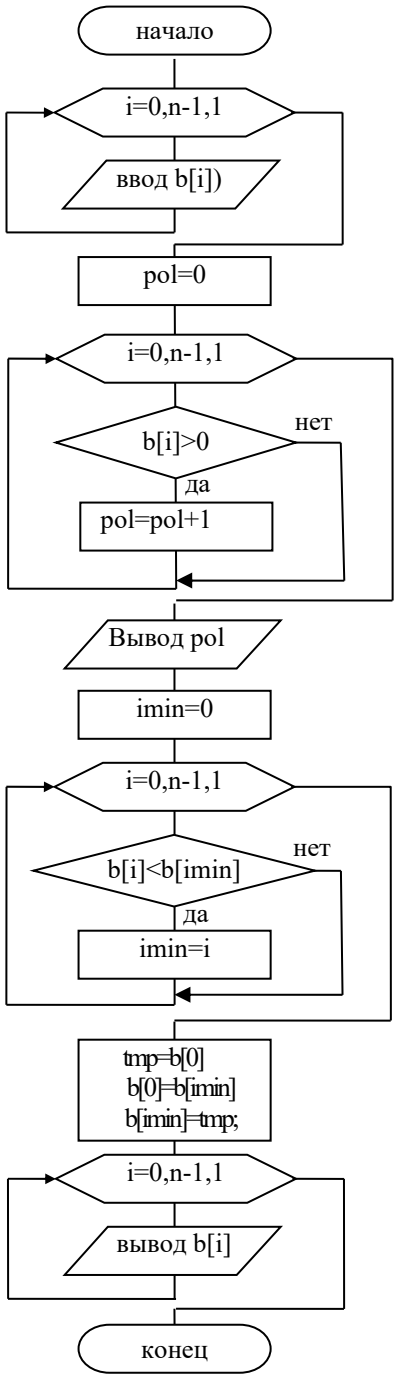


Рисунок 4.2

Задание 1 для самостоятельного решения

1 В одномерном массиве, состоящем из *n* целых элементов, вычислить:

а) номер минимального элемента массива;

б) сумму модулей элементов массива, расположенных после первого отрицательного элемента.

2 В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

а) номер максимального элемента массива;

б) сумму элементов массива, расположенных после первого положительного элемента.

3 В одномерном массиве, состоящем из *n* целых элементов, вычислить:

а) количество элементов массива, лежащих в диапазоне от A до B;

б) сумму элементов массива, расположенных после максимального элемента.

4 В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

а) количество элементов массива, равных 0;

б) сумму элементов массива, расположенных после минимального элемента.

5 В одномерном массиве, состоящем из *n* целых элементов, вычислить:

а) количество элементов массива, больших C;

б) произведение элементов массива, расположенных после максимального по модулю элемента.

6 В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

а) количество отрицательных элементов массива;

б) сумму модулей элементов массива, расположенных после минимального по модулю элемента.

7 В одномерном массиве, состоящем из *n* целых элементов, вычислить:

а) количество положительных элементов массива;

б) сумму элементов массива, расположенных после последнего элемента, равного нулю.

8 В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

а) количество элементов массива, меньших C;

б) сумму целых частей элементов массива, расположенных после последнего отрицательного элемента.

9 В одномерном массиве, состоящем из *n* целых элементов, вычислить:

а) произведение отрицательных элементов массива;

б) сумму положительных элементов массива, расположенных до максимального элемента.

10 В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

а) номер максимального элемента;

б) сумму элементов массива, расположенных до минимального элемента.

Двумерные массивы. Язык C# позволяет создавать многомерные массивы. Многомерные массивы имеют более одного измерения. Чаще всего используются двумерные массивы, которые представляются в виде матриц. Каждый элемент массива имеет два индекса, первый определяет номер строки, второй – номер столбца, на пересечении которых находится элемент. Нумерация строк и столбцов начинается с нуля.

Объявить двумерный массив можно одним из предложенных способов:

тип [,] имя_массива;
тип [,] имя_массива = new тип [размер1, размер2];
тип [,] имя_массива={{элементы 1-ой строки}, ... , {элементы n-ой строки}};

Например:

```
int [,] a;
int [,] a= new int [3, 4];
int [,] a={{0, 1, 2}, {3, 4, 5}};
```

Для доступа к элементу многомерного массива указываются все его индексы

имя_массива[*индекс1, индекс2*];

где индекс – целое число или целочисленная переменная из диапазона [0; размер$_i$-1].

Например, для доступа к элементу в третьей строке пятого столбца массива *d* следует использовать d[2,4] (не забываем, что в массивах индексация начинается с нуля).

Для работы с двумерными массивами используются вложенные циклы: первый перебирает по порядку строки, второй – элементы в соответствующей строке.

В следующем примере вводится двумерный массив и затем выводится в виде матрицы на экран:

```
int i, j;
int[,] matr = new int[3, 4];
for (i = 0; i < 3; i++)
  for (j = 0; j < 4; j++)
  {
    Console.Write("matr[{0},{1}] = ", i + 1, j + 1);
    matr[i, j] = int.Parse(Console.ReadLine());
  }
for (i = 0; i < 3; i++)
{
  for (j = 0; j < 4; j++)
    Console.Write("{0,7:d}", matr[i,j]);
  Console.WriteLine();
}
```

Пример 2 – Написать программу, которая определяет в целочисленной матрице номер строки, содержащей наибольшее количество элементов, равных нулю.

Исходными данными будет являться матрица *b* (точнее ее элементы). Матрицу заполним случайными числами от -2 до 2.

Результат программы – номер строки с наибольшим количеством нулей istr.

Для поиска строки с наибольшим количеством нулей будем использовать следующий алгоритм: для каждой строки будем находить количество нулей Kol

и сравнивать с максимальным количеством нулей MaxKol. Если значение Kol больше MaxKol, то в переменной MaxKol запоминаем количество нулей в данной строке и запоминаем номер строки.

На рисунке 4.3 представлена схема алгоритма программы.

```
static void Main(string[] args)
 {
  try
  {
   Random r = new Random();
   // описание двухмерного массива
   Console.WriteLine("Введите количество
                                        строк:");
   int nstr = int.Parse(Console.ReadLine());
   Console.WriteLine("Введите количество
                                        столбцов:");
   int nstb = int.Parse(Console.ReadLine());
   int[,] b = new int[nstr, nstb];
   int i, j;
   int istr = -1, MaxKol = 0, Kol;
   //заполнение матрицы числами от -2 до 2
   for (i = 0; i < nstr; i++)
   {
    for (j = 0; j < nstb; j++)
    {
     b[i, j] = r.Next(-2, 2);
     Console.Write("{0,5:d}", b[i, j]);
    }
    Console.WriteLine();
   }
   istr = -1; MaxKol = 0;
   for (i = 0; i < nstr; i++)
   { // просмотр массива по строкам
    Kol = 0;
    for (j = 0; j < nstb; j++)
     if (b[i, j] == 0)
      Kol++;
    if (Kol > MaxKol)
    {
     istr = i;
     MaxKol = Kol;
    }
   }
   if (istr == -1)
    Console.WriteLine("Нулевых элементов нет.");
   else
    Console.WriteLine("Номер строки с наибольшим количеством нулей: {0}", istr + 1);
  }
  catch (Exception e)
  {     Console.WriteLine("Ошибка: " + e.Message);   }
  Console.ReadKey();  }
```

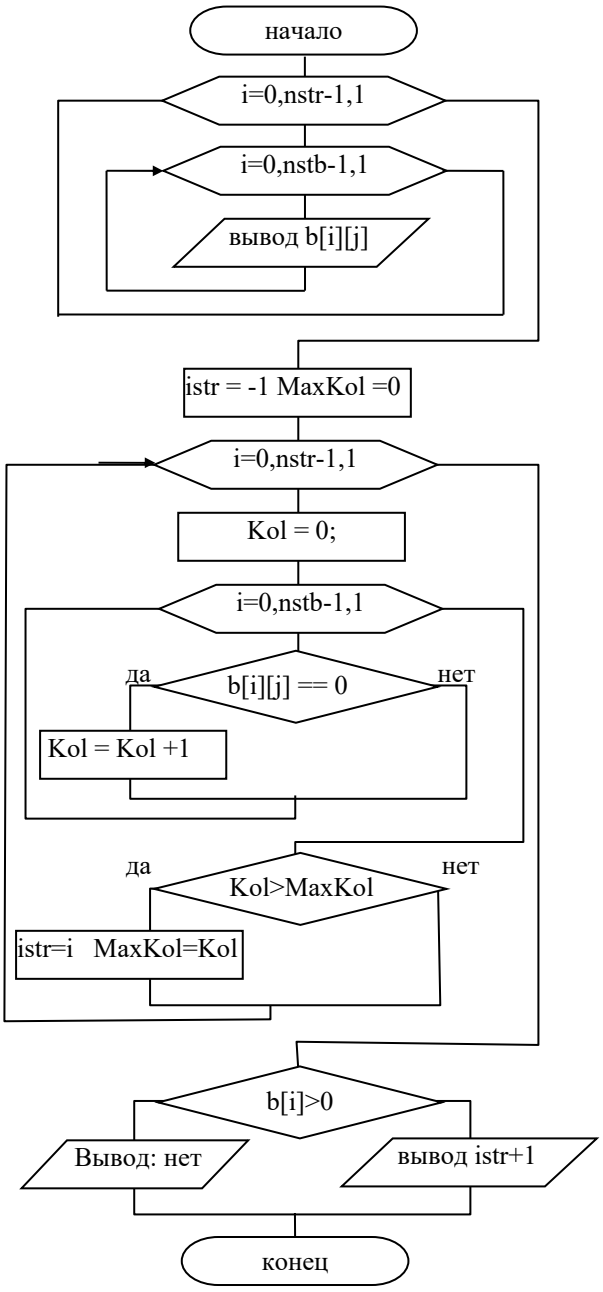


Рисунок 4.3

Задание 2 для самостоятельного решения

Для заданий 1–10 дана матрица целых чисел размером $n \times n$.

1 Подсчитать среднее арифметическое четных элементов, расположенных ниже главной диагонали.

2 Подсчитать среднее арифметическое ненулевых элементов, расположенных над побочной диагональю.

3 Подсчитать среднее арифметическое элементов, расположенных под побочной диагональю.

4 Поменять местами столбцы по правилу: первый с последним, второй с предпоследним и т. д.

5 Поменять местами две средних строки, если количество строк четное, и первую со средней строкой, если количество строк нечетное.

6 Поменять местами два средних столбца, если количество столбцов четное, и первый со средним столбцом, если количество столбцов нечетное.

7 Если количество строк в массиве четное, то поменять строки местами по правилу: первую строку со второй, третью – с четвертой и т. д. Если количество строк в массиве нечетное, то оставить массив без изменений.

8 Если количество столбцов в массиве четное, то поменять столбцы местами по правилу: первый столбец со вторым, третий – с четвертым и т. д. Если количество столбцов в массиве нечетное, то оставить массив без изменений.

9 Определить, есть ли в данном массиве строка, состоящая только из положительных элементов.

10 Определить, есть ли в данном массиве столбец, состоящий только из отрицательных элементов.

5 Методы: основные понятия

Метод – это именованная последовательность описаний и операторов, выполняющая какое-либо законченное действие, к которой можно обратиться по имени. Он описывается один раз, а вызываться может многократно.

Любая программа на C# состоит из методов, один из которых должен иметь имя Main (с него начинается выполнение программы). Метод начинает выполняться в момент вызова. Синтаксис определения метода:

```
[спецификаторы] тип_результата имя_метода ([список_параметров])
{
   объявления;
   тело_метода;
    return <результат>;
}
```

где *спецификаторы* являются необязательными элементами синтаксиса описания метода. Из всех спецификаторов в дальнейшем будет использоваться спецификатор *static*, который позволит обращаться к методу класса без создания

его экземпляра;

тип_результата определяет тип значения, возвращаемого методом. Это может быть любой тип. Если метод не возвращает никакого значения, необходимо указать тип *void* (в этом случае в теле метода отсутствует оператор *return*);

имя_метода – идентификатор, заданный программистом с учетом требований, *накладываемыми* на идентификаторы в C#, отличный от тех, которые уже использованы для других элементов программы в пределах текущей области видимости;

список_параметров представляет собой последовательность пар, состоящих из типа данных и идентификатора, разделенных запятыми. Параметры – это переменные или константы, которые получают значения, передаваемые методу при вызове. Если метод не имеет параметров, то *список_параметров* остается пустым;

результат определяет значение, возвращаемое методом оператором *return* в точку вызова. Тип результата должен соответствовать *типу_результата* или *приводится* к нему. Если метод описан как *void*, выражение не указывается или оператор *return* опускается.

Для **вызова** метода необходимо указать его имя и в круглых скобках через запятую передать набор аргументов в соответствии с параметрами, указанными в заголовке метода.

имя_метода(список аргументов);

Если тип возвращаемого методом значения не *void*, он может входить в состав выражений или, в частном случае, располагаться в правой части оператора присваивания.

В определении и при вызове одного и того же метода типы и порядок следования параметров должны совпадать.

Параметры метода.

Использование параметров является основным способом обмена информацией между вызываемым и вызывающим методами. Параметры, перечисленные в заголовке определения метода, называются *формальными,* а записанные в операторе вызова метода – *фактическими.*

При вызове метода, в первую очередь, вычисляются выражения, стоящие на месте фактических параметров; затем выделяется память под формальные параметры метода в соответствии с их типом, и каждому из них присваивается значение соответствующего фактического параметра.

Существует два способа передачи параметров в метод: по значению и по адресу.

Передача по значению.

Синтаксис:

– *вызов метода*:

имя_метода(имя_фактичекого_параметра);

– *определение метода*:

static тип имя_метода (тип имя_формального_параметра);

При передаче по значению в стек заносятся копии значений фактических параметров, и операторы метода работают с этими копиями. Доступа к исходным значениям параметров у метода нет, и поэтому при изменении формальных параметров фактические параметры не изменяются.

Передача по адресу.

Используются два синтаксиса:

1) с помощью ссылки:

а) *вызов метода*:

имя_функции(*ref* имя_фактического_параметра);

б) *определение метода*:

static тип имя_функции(*ref* тип имя_формального_параметра);

2) с помощью выходных параметров:

а) *вызов метода:*

имя_функции (*out* имя_фактичекого_параметра);

б) *определение метода*:

static тип имя_функции (*out* тип имя_формального_параметра);

При передаче по адресу в стек заносятся копии адресов фактических параметров, а функция осуществляет доступ к ячейкам памяти по этим адресам, т. е. при изменении значений формальных параметров значения фактических параметров также изменяются. При передаче по адресу в качестве фактических параметров нельзя использовать выражения, а только имена переменных. В отличии от out-параметров, ref-параметры перед вызовом метода должны быть проинициализированы.

```
static void f(int a, ref int b, out int c)
{
   a++; b++; c=10;
}
static void Main(string[] args)
{  int i = 1, j = 2, k=3;
   Console.WriteLine("i  j  k");
   Console.WriteLine("{0}  {1}  {2}", i, j, k);  //1 2 3
   f(i, ref j, out k);
   //вывод результатов после вызова функции
   Console.WriteLine("{0}  {1}  {2}", i, j, k); //1 3 10
   Console.ReadKey();   }
```

Пример – Написать программу для вычисления

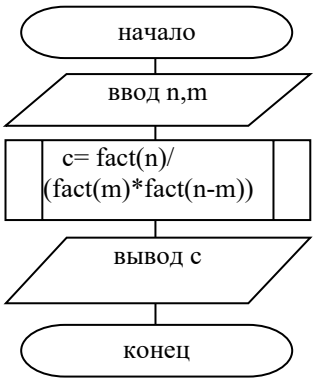
$$C_n^m = \frac{n!}{m!(n-m)!},$$

где n – факториал числа, $n! = 1 \cdot 2 \cdot 3 \ldots n$.

Вычисления факториала числа оформим в виде метода, в качестве параметра которого целое число, факториал которого необходимо вычислить. Результат выполнения функции – целое число, равное факториалу параметра. В основной программе 3 раза вызываем функции для получения факториалов чисел *n, m и n-m.*

На рисунке 5.1 представлена схема алгоритма программы.

```
using System;
namespace PrimerMet
{
  class Program
  {
    // определение метода
    static int fact(int a)
    {
      int i, p = 1;
      if (a > 1)
        for (i = 1; i <= a; i++)
          p = p * i;
      return p; //возврат значения p в точку вызова
    }
//основной метод, с которого начинается программа
    static void Main(string[] args)
    {
      try
      {  int n, m, c;
        Console.WriteLine("Введите n и m");
        n = int.Parse(Console.ReadLine());
        m = int.Parse(Console.ReadLine());
        // вызовы метода fakt
        c = fact(n) / (fact(m) * fact(n - m));
        Console.WriteLine("C = {0}", c);
      }
      catch (Exception e)
      {Console.WriteLine("Ошибка: " + e.Message); }
      Console.ReadKey();
    }
  }}
```



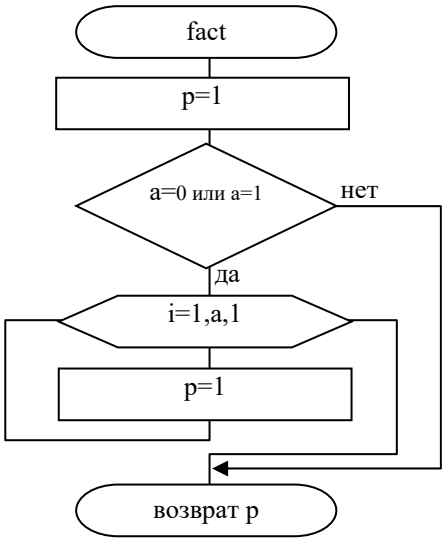


Рисунок 5.1

Задание для самостоятельного решения

1 Определить метод MeanA(X, Y, Z), вычисляющий среднее арифметическое из трех чисел по формуле (X + Y + Z)/3. С помощью этого метода найти среднее арифметическое для троек (A, B, C), (A, B, D), (A, C, D), если даны действительные числа A, B, C, D.

2 Определить метод RectS(x_1, x_2, y_1, y_2), вычисляющий площадь прямоугольника со сторонами, параллельными осям координат, по координатам его противоположных вершин (x_1, y_1) и (x_2, y_2) . С помощью этого метода найти площади двух прямоугольников с данными противоположными вершинами.

3 Определить метод MeanG(*X*, *Y*, *Z*), вычисляющий среднее геометрическое из трех чисел по формуле $\sqrt[3]{XYZ}$. С помощью этого метода найти среднее геометрическое для троек (*A*, *B*, *C*), (*A*, *B*, *D*), (*A*, *C*, *D*), если даны действительные числа *A*, *B*, *C*, *D*.

4 Разработать метод Len(*x*1, *y*1, *x*2, *y*2), который вычисляет длину отрезка по координатам вершин (*x*1, *y*1) и (*x*2, *y*2). С помощью данного метода найти периметр треугольника, заданного координатами своих вершин.

5 Разработать метод SqP(*a*, *b*), который по катетам *a* и *b* вычисляет гипотенузу. С помощью данного метода найти гипотенузы трех заданных прямоугольных треугольников.

6 Разработать метод SqT(*x*, *y*, *z*), который по длинам сторон треугольника *x*, *y*, *z* вычисляет его площадь. С помощью данного метода вычислить площади двух заданных треугольников.

7 Определить метод RectP(x_1, x_2, y_1, y_2), вычисляющий периметр прямоугольника со сторонами, параллельными осям координат, по координатам его противоположных вершин (x_1, y_1) и (x_2, y_2). С помощью этого метода найти периметры двух прямоугольников с данными противоположными вершинами.

8 Определить метод Fact2(*N*), вычисляющий двойной факториал:

$$N!! = 1 \cdot 3 \cdot 5 \cdot \ldots \cdot N, \text{ если } N \text{ нечетное},$$

$$N!! = 2 \cdot 4 \cdot 6 \cdot \ldots \cdot N, \text{ если } N \text{ четное}.$$

С помощью этого метода найти двойные факториалы пяти данных чисел.

9 Определить метод Digit(*N*), возвращающий количество цифр в целом числе *N*. Для четырех введенных чисел найти количество цифр в каждом из них.

10 Определить метод DigitN(*N*, *K*), возвращающий *K*-ю цифру в целом числе *N*. Для трех введенных чисел вывести вторую цифру.

Список литературы

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

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

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

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