

Министерство образования и науки Российской Федерации
ФГБОУ ВПО «Кубанский государственный технологический университет»

Кафедра компьютерных технологий и информационной безопасности

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Методические указания по выполнению лабораторных работ
для студентов очной формы обучения направления подготовки
10.03.01 Информационная безопасность

Краснодар
2015

Составитель: канд. техн. наук, доц. Частикова В.А.

Языки программирования: методические указания по выполнению лабораторных работ для студентов очной формы обучения направления подготовки (специальности) 10.03.01 - Информационная безопасность./ Сост.: В.А. Частикова. Кубан. гос. технол. ун-т. Каф. КТИБ. – Краснодар, 2015. – 153 с.

Методические указания составлены в соответствии с рабочей программой дисциплины «Языки программирования» для студентов очной формы обучения направления подготовки 10.03.01 – Информационная безопасность.

Содержат описания лабораторных работ, указания к их выполнению, задания и требования к оформлению отчета. Рассмотрены основы программирования на языке высокого уровня C#, основы объектно-ориентированного программирования.

Библиогр.: 4 назв.

Рецензенты

проф. кафедры КТИБ КубГТУ, канд. техн. наук, проф. В.А. Кучер,
проф. кафедры КТИБ КубГТУ, канд. техн. наук, проф. В.Н. Хализев

Содержание

1. Лабораторная работа № 1. Среда разработки Visual Studio.NET.
Консольные приложения.
2. Лабораторная работа № 2. Проектирование простейших вычислительных программ линейной структуры.
3. Лабораторная работа № 3. Программирование ветвящихся процессов.
4. Лабораторная работа № 4. Программирование циклических процессов.
5. Лабораторная работа № 5. Программирование итерационных циклов и вычислительных процессов с усложненной структурой.
6. Лабораторная работа № 6. Программирование задач обработки одномерных массивов.
7. Лабораторная работа № 7. Программирование задач обработки двумерных массивов.
8. Лабораторная работа № 8. Обработка исключительных ситуаций.
9. Лабораторная работа № 9. Обработка текстовой информации.
10. Лабораторная работа № 10. Простейшие классы. Поля, константы, конструкторы, свойства.

Список литературы

Лабораторная работа № 1

СРЕДА РАЗРАБОТКИ VISUAL STUDIO.NET. КОНСОЛЬНЫЕ ПРИЛОЖЕНИЯ

1 Цель и порядок работы

Цель работы - изучить установку Visual Studio, научиться работать с консольными приложениями, получить практические навыки в составлении программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- оформить отчет.

2 Общие сведения

2.1 Установка Visual Studio

Материал, помещенный в данной лабораторной работе, поможет получить навыки написания надежных, стабильных и сопровождаемых программ. Процесс получения таких знаний целесообразно начинать с приобретения инструментов для разработки приложений на языке C#.

Первым шагом на этом пути является установка среды разработки Visual Studio. Visual Studio можно установить:

- используя установочный носитель (например, DVD-диск);
- путем загрузки продукта с веб-сайта.

Размер установочного пакета Visual Studio довольно большой, поэтому, если в распоряжении нет высокоскоростного Интернета, целесообразно устанавливать среду с CD-ROM. Установка Visual Studio должна выполняться от имени учетной записи администратора.

На одном компьютере может работать более одной версии среды Visual Studio, что очень важно, когда на компьютере установлена более ранняя версия, и ее надо сохранить.

Корпорация Microsoft предоставляет полные версии среды разработки Visual Studio, такие как: Standart, Professional и Team. Каждая из этих версий имеет свои возможности и цену.

В лабораторных работах для рассмотрения необходимых примеров, применяется версия Visual Studio Express, так как эта среда разработки является бесплатной и предоставляет все необходимые функциональности, позволяющие начать работать с C# 4.0.

Установка с использованием установочного носителя. Чтобы начать установку Visual Studio, необходимо запустить файл Setup.exe, расположенный в корневой папке установочного носителя Visual Studio.

Установка путем загрузки с веб-сайта продукта. Чтобы начать установку Visual Studio, надо перейти на страницу загрузки Visual Studio 2010 на веб-сайте MSDN.

Параметры командной строки, доступные при установке, перечислены в таблице 1.

Таблица 1- Параметры командной строки, доступные при установке

Параметр	Описание
/?	Отображение параметров командной строки.
/q	Подавление интерфейса пользователя.
/remove	Удаление Visual Studio.
/f	Восстановление компонентов Visual Studio.
/full	Установка компонентов (только в автоматическом режиме).
/norestart	Отмена перезагрузки. Установщик не выполняет перезагрузку компьютера в процессе или после установки. Если перезапуск необходим, программа установки возвращает сообщение 3010.
/CreateUnattend filename.ini	Создайте файл параметров с именем "имя_файла.ini" для выполнения установки в автоматическом режиме.
/UnattendFile filename.ini	Выполните установку в автоматическом режиме, используя созданный INI-файл параметров.
/NoEventLog	Отключите ведение журнала событий.
/NoErrorLog	Отключите ведение журнала ошибок.
/CreateTransform="filename.mst"	Настройте установку и сохраните конфигурацию в указанном файле изменений.

После установки среды разработки Visual Studio рекомендуется зарегистрировать установленный экземпляр программного продукта и проверить наличие обновлений. Кроме того, можно установить дополнительные компоненты, например библиотеку изображений Visual Studio.

Для восстановления Visual Studio необходимо выполнить следующие действия:

— *в ОС Windows XP или более ранней версии:*

1. В диалоговом окне **Установка и удаление программ панели управления** выбрать программный продукт, который необходимо восстановить, и нажать кнопку **Изменить/Удалить**.
2. В окне мастера установки нажать кнопку **Далее**.
3. Нажать кнопку **Восстановить или переустановить**.

— *в ОС Windows Vista:*

1. На странице **Программы и компоненты панели управления** выбрать программный продукт, который необходимо восстановить, и нажать кнопку **Удалить/изменить**.
2. В мастере установки выбрать **Восстановить или переустановить**, а затем нажать кнопку **Далее**.
3. Следовать инструкциям.

2.2 Разработка приложений

Завершение установки среды Visual Studio открывает возможность создания различного рода приложений. Среда Visual Studio.NET позволяет создавать программы различных типов, но в дисциплине «Языки программирования» имеют значение приложения следующих трех типов:

- *Windows-приложения*, которые используют элементы интерфейса Windows, включая формы, кнопки, флажки и пр.;
- *консольные приложения*, предназначенные для выполнения в командной строке и не имеющие пользовательского интерфейса;
- *библиотека классов*, объединяющая классы, которые предназначены для использования в других приложениях, т.е. в данном случае могут быть использованы и в *Windows-приложениях* и в *консольных приложениях*.

Приложение в процессе разработки называется *проектом*. Проект объединяет все необходимое для создания приложения: файлы, папки, ссылки и прочие ресурсы.

Несколько проектов можно объединить в решение (solution). Это облегчает совместную разработку проектов.

2.2.1 Консольные приложения

Среда Visual Studio.NET работает на платформе Windows и ориентирована на создание Windows- и веб-приложений, однако разработчики предусмотрели работу и с консольными приложениями. При запуске консольного приложения операционная система создает так называемое *консольное окно*, через которое идет весь ввод-вывод программы. Внешне это напоминает работу в операционной системе в режиме *командной строки*, когда ввод-вывод представляет собой поток символов.

Консольные приложения наилучшим образом подходят для начального этапа изучения дисциплины «Языки программирования», так как в них не используется множество стандартных объектов, необходимых для создания графического интерфейса, и можно сконцентрировать свое внимание на выработке навыков построения разнообразных алгоритмов и изучении непосредственно языка C#. Поэтому в первой части курса мы будем создавать только консольные приложения.

В следующем разделе рассмотрены самые простые действия в среде: создание и запуск на выполнение консольного приложения на C#.

Создание проекта. Основные окна среды

Начальная страница

После запуска Visual Studio.NET на экране появляется ее окно, вид которого показан на рис.1.

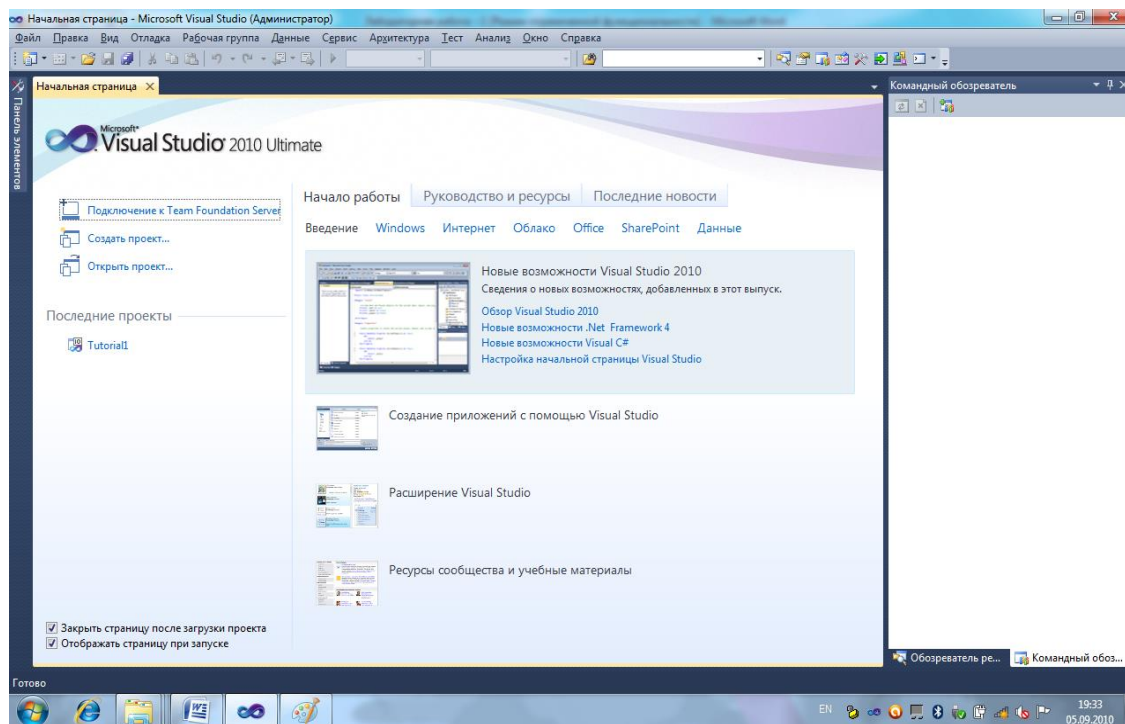


Рисунок 1 – Вид экрана после запуска Visual Studio.NET

В верхней части экрана располагается *главное меню* (с разделами Файл, Правка, Вид и т. д.) и *панели инструментов*.

Почти всю остальную часть экрана занимает диалоговое окно **Начальная страница**. В левой ее части расположены кнопки, которые можно использовать для вызова уже созданного проекта (**Открыть проект**) и создания нового проекта (**Создать проект**).

Создать проект

Выберем режим **Создать проект**, тогда откроется новое окно (рис. 2), информация которого разбита на три части:

В левой части представлены установленные шаблоны:

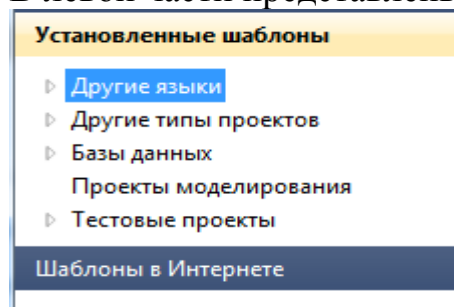


Рисунок 2

Щелчком мышкой по треугольнику **Другие языки** и раскроем его содержание (рис.3).

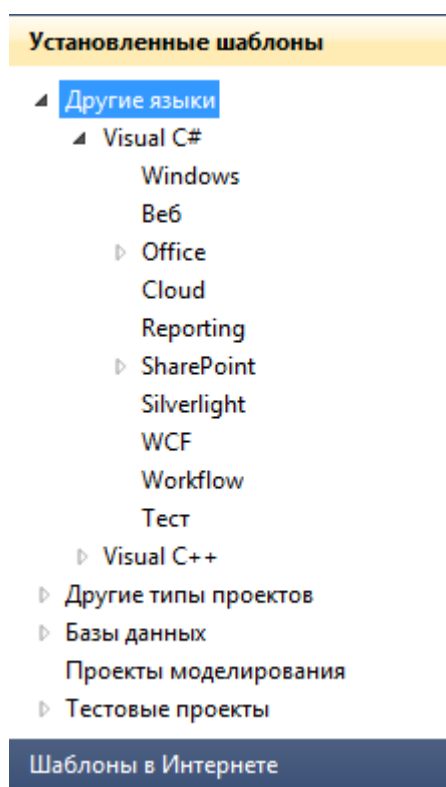


Рисунок 3

Выберем шаблон **Visual C#**, после чего средняя часть экрана высветит все доступные для выбранного шаблона типы приложений, часть из которых показана на рис. 4.

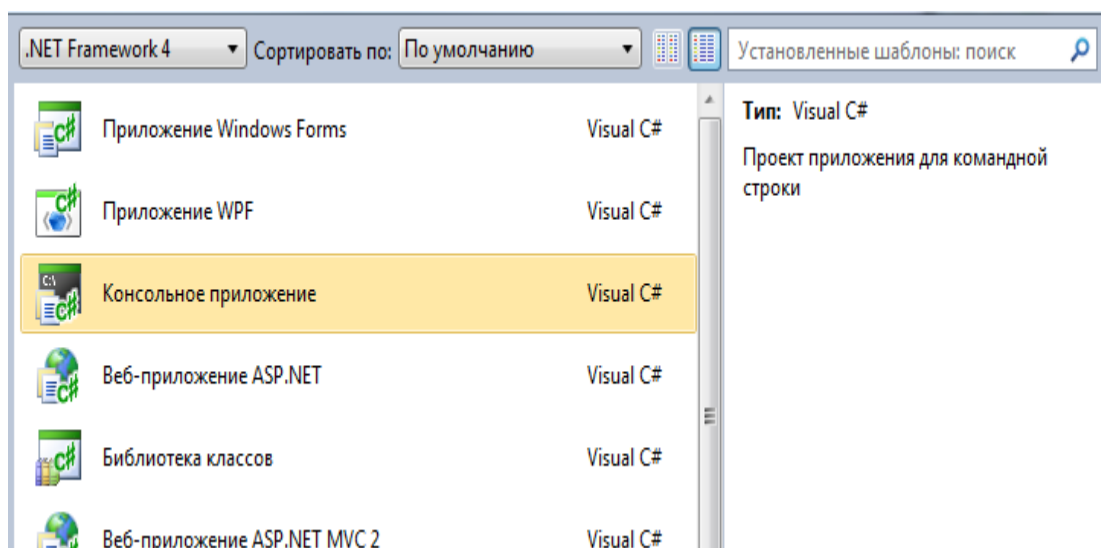


Рисунок 4

Альтернативный способ создания проекта предусмотрен в главном меню, где нужно выбрать команду **Файл>Создать>Проект...**

Выбор типа приложения

Как уже стало понятно, средняя часть экрана **Начальная страница** предназначена для выбора требуемого типа приложения. Щелчком мышкой **Консольное приложение**, в правой части окна появится комментарий к данному выбору.

После всех указанных установок необходимо нажать кнопку **ОК**.

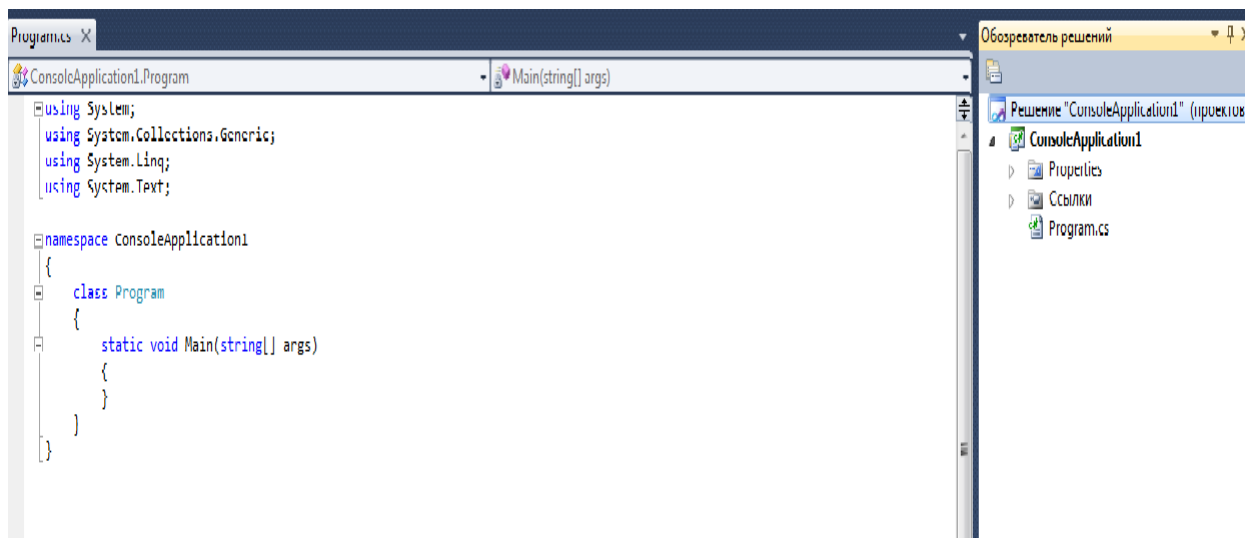


Рисунок 5 - Примерный вид экрана после создания проекта консольного приложения

После щелчка на кнопке **ОК** среда создаст решение и проект с именем **ConsoleApplication1**. Примерный вид экрана приведен на рис. 5.

Главное меню, которое хорошо видно на рис. 1, позволяет разработчику воспользоваться различными возможностями управления проектом, решением, получить полную информацию о ресурсах, сборке, классах, входящих в приложение, их элементов, предков, ссылок на библиотеку и многое другое.

С помощью проводника Windows можно увидеть, что на заданном диске появилась папка с указанным именем, содержащая несколько других файлов и вложенных папок. Среди них — файл проекта (с расширением **csproj**), файл решения (с расширением **sln**) и файл с программным кодом (**Program.cs**).

Нажатием клавиши **F4** или выбором из главного меню **Вид > Окно свойств (Properties)** можно вызвать (рис. 6) в нижней правой части экрана соответствующее окно.

В окне свойств отображаются важнейшие характеристики выделенного элемента. Например, чтобы изменить имя файла, надо выделить этот файл в окне управления проектом и задать в окне свойств новое значение свойства **FileName** (ввод заканчивается нажатием клавиши **Enter**).

Вид	Проект	Построение	Отладка	Рабочая группа
	Код			F7
	Открыть			
	Открыть с помощью...			
	Обозреватель решений			Ctrl+Alt+L
	Командный обозреватель			Ctrl+\, Ctrl+M
	Обозреватель серверов			Ctrl+Alt+S
	Обозреватель архитектуры			Ctrl+\, Ctrl+R
	Окно закладок			Ctrl+K, Ctrl+W
	Иерархия вызовов			
	Классы			Ctrl+Shift+C
	Окно определения кода			Ctrl+\, D
	Обозреватель объектов			F2
	Список ошибок			Ctrl+W, Ctrl+E
	Вывод			Ctrl+Alt+O
	Начальная страница			
	Список задач			Ctrl+Alt+K
	Панель элементов			Ctrl+Alt+X
	Результаты поиска			▶
	Другие окна			▶
	Панели инструментов			▶
	Во весь экран			Shift+Alt+ВВОД
	Назад			Ctrl+Shift+F2
	Вперед			Ctrl+Shift+-
	Следующая задача			Ctrl+Shift+F12
	Предыдущая задача			
	Окно свойств			F4
	Окна свойств			Shift+F4

Рисунок 6

Все операции для работы с проектом сконцентрированы в главном меню **Проект** (рис. 7). Для некоторых из них зарезервированы «горячие» клавиши.

При создании решения автоматически присваивается одно и то же имя и проекту и решению, что не всегда удобно. Для того чтобы присвоить решению новое имя, необходимо выполнить следующие действия:

- в обозревателе решений (Solution Explorer) щелкнуть правой кнопкой мыши по имени решения и в открывшемся контекстном меню выбрать пункт **Переименовать** (Rename);
- изменить имя на новое;
- нажать клавишу **Enter**, чтобы принять изменение.

Таким же образом можно переименовать проекты или любой другой элемент решения.

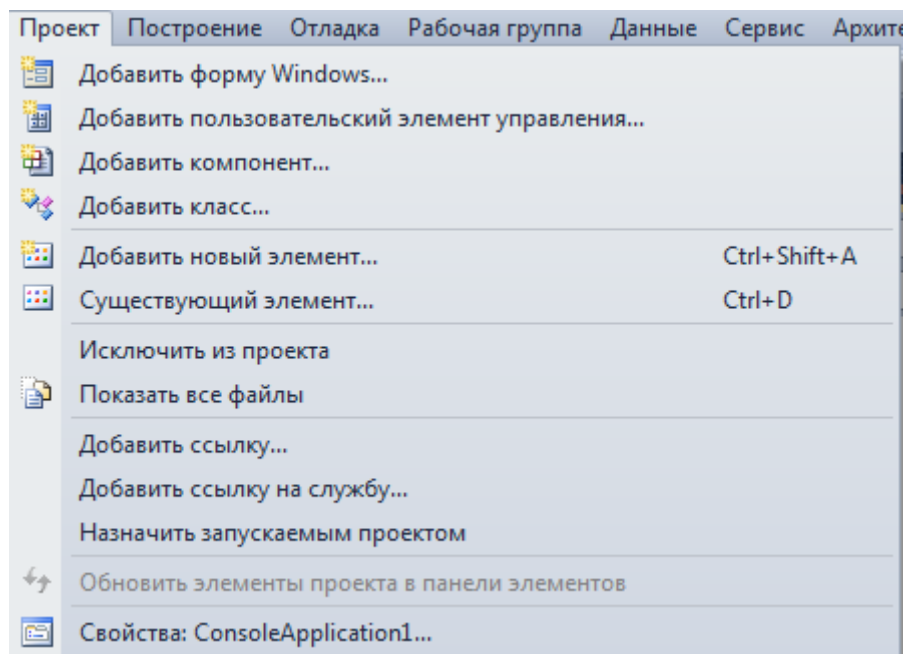


Рисунок 7

Основное пространство экрана (рис. 5) занимает *окно редактора*, в котором располагается текст программы, созданный средой автоматически. Текст представляет собой каркас или шаблон, в который программист добавляет код по мере необходимости.

Ключевые (зарезервированные) слова. Ключевые слова — это слова, имеющие специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Они отображаются синим цветом, комментарии различных типов — серым и тёмно-зелёным, остальной текст — черным. Слева от текста находятся *символы структуры*: щелкнув на любом квадратике с минусом, можно скрыть соответствующий блок кода. При этом минус превращается в плюс, щелкнув на котором, можно опять вывести блок на экран. Это средство хорошо визуализирует код и позволяет сфокусировать внимание на нужных фрагментах.

Макет консольной программы

Рассмотрим каждую строку макета программы, ее основные элементы (листинг 1).

Листинг 1 Макет консольной программы

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Директива `using System` позволяет использовать имена стандартных классов из пространства имен `System`, непосредственно (без указания имени пространства). Ключевое слово `namespace` создает для проекта собственное пространство имен, названное по умолчанию `ConsoleApplication1`. Это сделано для того, чтобы можно было давать программным объектам имена, не заботясь о том, что они могут совпасть с именами в других пространствах имен.

Фигурные скобки являются важным элементом синтаксиса. Каждой открывающей скобке соответствует своя закрывающая, которая обычно располагается ниже по тексту с тем же отступом. Эти скобки ограничивают блок, внутри которого могут располагаться другие блоки, вложенные в него, как матрешки. Блок может применяться в любом месте, где допускается отдельный оператор.

Строки, начинающиеся с двух или трех косых черт, являются комментариями и предназначены для документирования текста программы. С# — объектно-ориентированный язык, поэтому написанная на нем программа представляет собой совокупность взаимодействующих между собой классов. В исходной заготовке программы всего один класс, которому по умолчанию задано имя **Program**. *Описание класса* начинается с ключевого слова `class`, за которым следуют его имя и далее в фигурных скобках — список элементов класса (его данных и функций, называемых также методами).

В данном случае внутри класса только один элемент — метод `Main`. Каждое приложение должно содержать метод `Main` — с него начинается выполнение программы. Все методы описываются по единым правилам.

Упрощенный синтаксис метода:

```
[ спецификаторы ] тип имя_метода ( [ параметры ] )
{
    тело метода: действия, выполняемые методом
}
```

Для наглядности и простоты изложения используется широко распространенный неформальный способ описания, когда необязательные части синтаксических конструкций заключаются в квадратные скобки, текст, который необходимо заменить конкретным значением, пишется по-русски, а возможность выбора одного из нескольких элементов обозначается вертикальной чертой. Например:

```
[ void | int ] имя_метода( ) ;
```

Эта запись означает, что вместо конструкции имя_метода необходимо указать конкретное имя в соответствии с правилами языка, а перед ним может находиться либо слово `void`, либо слово `int`, либо ничего. Символ подчеркивания используется для связи слов вместо пробела, показывая, что на этом месте должен стоять один синтаксический элемент, а не два. В тех случаях, когда квадратные скобки являются элементом синтаксиса, это оговаривается особо.

Таким образом, любой метод должен иметь тип, имя и тело, остальные части описания являются необязательными.

Наряду с понятием «метод» часто используется другое — функция-член класса. Метод является частным случаем функции — законченного фрагмента кода, который можно вызвать по имени. Далее в книге используются оба эти понятия.

```
Console.WriteLine( "Первая программа" );
```

Здесь `Console` — это имя стандартного класса из пространства имен `System`. Его метод `WriteLine` выводит на экран заданный в кавычках текст. Как видите, для обращения к методу класса используется конструкция

имя_класса.имя_метода

Если вы не сделали ошибку в первом же слове, то сразу после ввода с клавиатуры следом за словом `Console` символа точки среда выведет подсказку, содержащую список всех доступных элементов класса `Console`. Выбор нужного имени выполняется либо мышью, либо клавишами управления курсором, либо вводом одного или нескольких начальных символов имени. При нажатии клавиши `Enter` выбранное имя появляется в тексте программы.

Не пренебрегайте возможностью автоматического ввода — это убережет вас от опечаток и сэкономит время. Если подсказка не появляется, это свидетельствует об ошибке в имени или в месте расположения в программе вводимого текста.

Программа должна приобрести вид, приведенный в листинге 1.2 (для того чтобы вы могли сосредоточиться на структуре программы, из нее убраны все комментарии и другие пока лишние для нас детали).

Обратите внимание на то, что после внесения изменений около имени файла на ярлычке в верхней части окна редактора появился символ `*` — это означает, что текст, сохраненный на диске, и текст, представленный в окне редактора, не совпадают. Для сохранения файла воспользуйтесь командой `File > Save` главного меню или кнопкой `Save` на панели инструментов

(текстовый курсор должен при этом находиться в окне редактора). Впрочем, при запуске программы среда сохранит исходный текст самостоятельно.

Листинг 2 Первая программа на C#

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine( "Первая программа" );
        }
    }
}
```

На рис. 8 приведен экран после создания консольного приложения в Visual C# Express Edition.

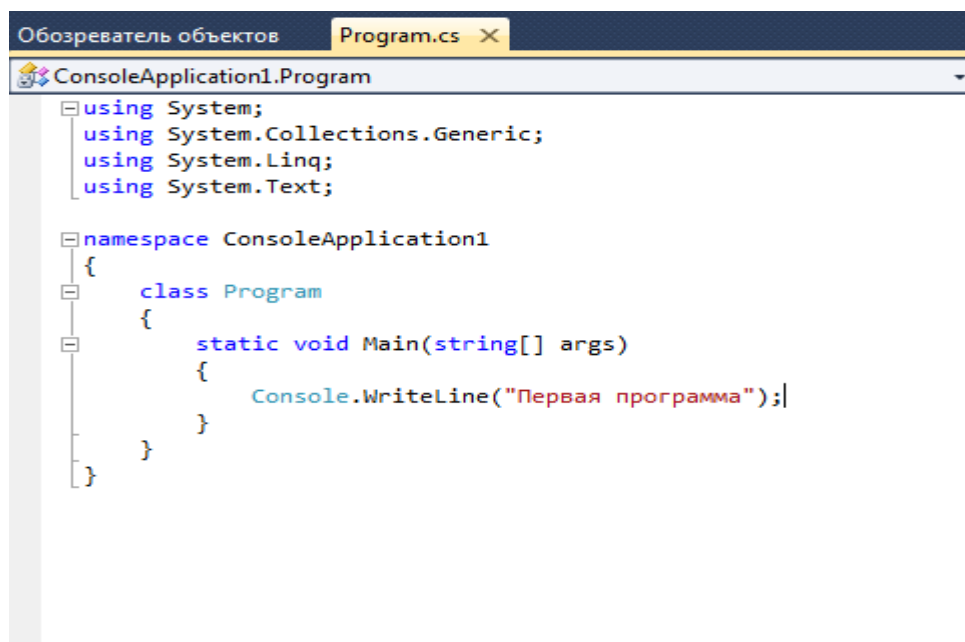


Рисунок 8 - Вид экрана после создания проекта консольного приложения

2.2.2 Запуск программы

Самый простой способ запустить программу — нажать клавишу F5 (или выбрать в меню команду **Отладка>Начать отладку** (Debug >Start)). Если программа написана без ошибок, то фраза "Первая программа" промелькнет перед вашими глазами в консольном окне, которое незамедлительно закроется. Это хороший результат, но для того чтобы пронаблюдать его спокойно, следует воспользоваться клавишами Ctrl+F5 (или выбрать в меню команду **Отладка >Запуск без отладки** (Debug > Start Without Debugging) (рис. 9).

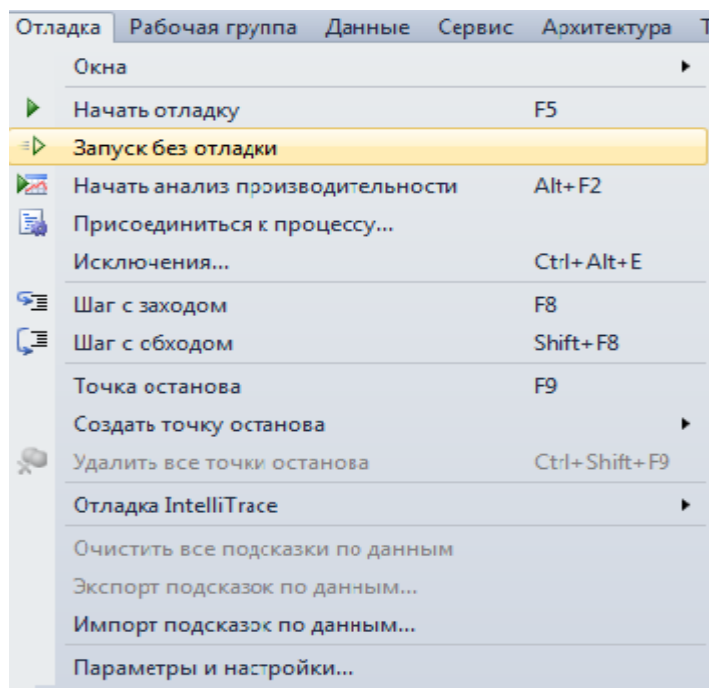


Рисунок 9

После внесения изменений компилятор может обнаружить в тексте программы *синтаксические ошибки*. Он сообщает об этом в окне, расположенном в нижней части экрана.

Давайте внесем в программу синтаксическую ошибку, чтобы впервые увидеть то, что впоследствии вам придется наблюдать многие тысячи раз. Уберите точку с запятой после оператора `Console.WriteLine(...)` и запустите программу заново. Вы увидите диалоговое окно (рис. 10) с сообщением о том, что при построении приложения обнаружены ошибки, и вопросом, продолжать ли дальше (There were build errors. Continue?).

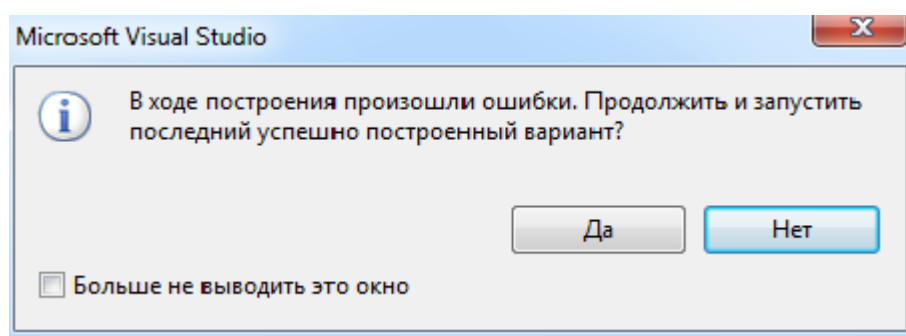


Рисунок 10

Ответив No, обратимся к окну вывода ошибок за разъяснениями (рис. 11).

Описание	Файл	Строка	Столбец	Проект
1 Требуется ";"	Program.cs	12	50	ConsoleApplication1

Рисунок 11

Если сообщения об ошибке не видно, просмотрите содержимое окна с помощью полосы прокрутки, пытаясь найти в нем слово «error».

Двойной щелчок на строке с сообщением об ошибке **Требуется;** (Expected; - означающее, что здесь ожидалась точка с запятой) подсвечивает неверную строку в программе. Для получения дополнительных пояснений можно нажать клавишу F1, при этом в окне редактора появится новая вкладка с соответствующей страницей из справочной системы. Для возвращения к коду класса следует щелкнуть на ярлычке с именем файла в верхней строке редактора.

3. Содержание отчета

3.1. Наименование и цель работы.

3.2. Краткий конспект теоретического материала.

Лабораторная работа № 2

ПРОЕКТИРОВАНИЕ ПРОСТЕЙШИХ ВЫЧИСЛИТЕЛЬНЫХ ПРОГРАММ ЛИНЕЙНОЙ СТРУКТУРЫ

1 Цель и порядок работы

Цель работы - изучить встроенные типы данных и операции языка, используемые при вычислении различного типа выражений, организацию линейных вычислительных процессов, получить практические навыки в составлении линейных программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- написать программу, ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

2.1 Встроенные типы данных

C# содержит две категории встроенных типов данных: *типы значений* и *ссылочные типы*. Термин "тип значения" применяется к переменным, которые непосредственно содержат значения. (Для сравнения: переменные ссылочных типов содержат ссылки на реальные значения). Типы значений также называют *простыми типами*. В C# строго определяется диапазон и поведение каждого типа значения. Например, тип `int` должен быть одинаковым во всех средах выполнения.

Ядро языка C# составляют 13 встроенных типов, которые определяются ключевыми словами C# и доступны для использования в любой C# программе (табл. 1).

Таблица 1 – Встроенные типы данных

ключевое слово	тип
<code>bool</code>	Логический, представляет значения истина/ложь.
<code>byte</code>	8-разрядный целочисленный без знака.
<code>char</code>	Символьный.
<code>decimal</code>	Числовой тип для финансовых вычислений.
<code>double</code>	С плавающей точкой двойной точности.
<code>float</code>	С плавающей точкой.
<code>int</code>	Целочисленный.
<code>long</code>	Тип для представления длинного целого числа.

sbyte	8-разрядный целочисленный со знаком.
short	Тип для представления короткого целого числа.
uint	Целочисленный без знака.
ulong	Тип для представления длинного целого числа без знака.
ushort	Тип для представления короткого целого числа без знака.

В С# определено девять целочисленных типов: char, byte, sbyte, short, ushort, int, uint, long и ulong. Однако тип char в основном используется для представления символов. Типы с плавающей точкой могут представлять числа с дробными компонентами. Таких типов только два: float и double.

Тип bool представляет значения истина/ложь, которые в С# определяются зарезервированными словами true и false. Таким образом, переменная или выражение типа bool будет иметь одно из этих двух значений. В С# не определено ни одно преобразование значения типа bool в целочисленное значение. Например, число 1 не преобразуется в значение true, а число 0 - в значение false.

Для объявления переменной необходимо использовать инструкцию следующего формата:

тип имя_переменной;

Здесь с помощью элемента **тип** задается тип объявляемой переменной, а с помощью элемента **имя_переменной** - ее имя. Можно объявить переменную любого допустимого типа. Все переменные в С# должны быть объявлены до их использования. Это - требование компилятора, поскольку, прежде чем скомпилировать надлежащим образом инструкцию, в которой используется переменная, он должен "знать" тип содержащейся в ней информации.

Переменная до использования должна получить значение. Это можно сделать с помощью инструкции присваивания. Можно также присвоить переменной начальное значение одновременно с ее объявлением. Для этого достаточно после имени переменной поставить знак равенства и указать присваиваемое значение. Общий формат инициализации переменной имеет такой вид:

тип имя_переменной = значение;

2.2 Операции и выражения

В С# предусмотрен широкий набор операций, которые дают в руки программисту мощные рычаги управления при создании разнообразнейших выражений и их вычислении.

Выражение — это правило вычисления значения. В выражении участвуют *операнды*, объединенные *знаками операций*. Операндами простейшего выражения могут быть константы, переменные и вызовы функций.

Например, $y + 9$ — это выражение, в котором $+$ является знаком операции, а y и 9 — операндами. Пробелы внутри знака операции, состоящей из нескольких символов, не допускаются. По количеству участвующих в одной операции операндов операции делятся на *унарные*, *бинарные* и *тернарную*. Операции C# приведены в табл. 2.

Таблица 2 - Операции C#

Категория	Знак операции	Название
Первичные	. x() x[] X++ X-- new typeof checked unchecked	Доступ к элементу Вызов метода или делегата Доступ к элементу Постфиксный инкремент Постфиксный декремент Выделение памяти Получение типа Проверяемый код Непроверяемый код
Унарные	+ - ! ~ ++X --X (тип) x	Унарный плюс Унарный минус (арифметическое отрицание) Логическое отрицание Поразрядное отрицание Префиксный инкремент Префиксный декремент Преобразование типа
Мультипликативные (типа умножения)	* / %	Умножение Деление Остаток от деления
Аддитивные (типа сложения)	+ -	Сложение Вычитание
Сдвига	<< >>	Сдвиг влево Сдвиг вправо
Отношения и проверки типа	< > <= >= is as	Меньше Больше Меньше или равно Больше или равно Проверка принадлежности типу Приведение типа
Проверки на равенство	= !=	Равно Не равно
Поразрядные логические	& ^ 	Поразрядная конъюнкция (И) Поразрядное исключающее ИЛИ Поразрядная дизъюнкция (ИЛИ)
Условные логические	&& 	Логическое И Логическое ИЛИ
Условная	? :	Условная операция

Присваивания	=	Присваивание
	*=	Умножение с присваиванием
	/=	Деление с присваиванием
	%=	Остаток от деления с присваиванием
	+=	Сложение с присваиванием
	-=	Вычитание с присваиванием
	<<=	Сдвиг влево с присваиванием
	>>=	Сдвиг вправо с присваиванием
	&=	Поразрядное И с присваиванием
	^=	Поразрядное исключающее ИЛИ с присваиванием
	=	Поразрядное ИЛИ с присваиванием

Операции в выражении выполняются в определенном порядке в соответствии с *приоритетами*. В табл. 2 операции расположены по убыванию приоритетов, уровни приоритетов разделены в таблице горизонтальными линиями.

Результат вычисления выражения характеризуется *значением* и *типом*. Например, пусть x и y — переменные целого типа и описаны так:

`int x = 7, y = 3;`

Тогда выражение $x + y$ имеет значение 10 и тип `int`, а выражение $x = y$ имеет значение, равное помещенному в переменную a (в данном случае — 3), и тип, совпадающий с типом этой переменной.

Если в одном выражении соседствуют несколько операций одинакового приоритета, операции присваивания и условная операция выполняются *справа налево*, остальные — *слева направо*. Для изменения порядка выполнения операций используются *круглые скобки*, уровень их вложенности практически не ограничен.

Например, $a + b + c$ означает $(a + b) + c$, а $x = b = c$ означает $x = (b = c)$. То есть сначала вычисляется выражение $b = c$, а затем его результат становится правым операндом для операции присваивания переменной x .

Часто перед выполнением операции требуется вычислить значения операндов. Например, в выражении $F(i) + G(i++) * H(i)$ сначала вызываются функции F , G и H , а затем выполняются умножение и сложение. Операнды всегда вычисляются слева направо независимо от приоритетов операций, в которых они участвуют. Кстати, в приведенном примере метод H вызывается с новым значением i (увеличенным на 1).

2.3 Математические функции — класс *Math*

В выражениях часто используются математические функции, например синус или возведение в степень. Они реализованы в классе *Math*, определенном в пространстве имен *System*. С помощью методов этого класса можно вычислить:

- тригонометрические функции: `Sin`, `Cos`, `Tan`;
- обратные тригонометрические функции: `ASin`, `ACos`, `ATan`, `ATan2`;

- гиперболические функции: Tanh, Sinh, Cosh;
- экспоненту и логарифмические функции: Exp, Log, Log10;
- модуль (абсолютную величину), квадратный корень, знак: Abs, Sqrt, Sign;
- округление: Ceiling, Floor, Round;
- минимум, максимум: Min, Max;
- степень, остаток: Pow, IEEEReminder;
- полное произведение двух целых величин: BigMul;
- деление и остаток от деления: DivRem.

Кроме того, у класса есть два полезных поля: число π и число e . Описание методов и полей приведено в табл. 3.

Таблица 3 - Основные поля и статические методы класса *Math*

Имя	Описание	Результат	Примечания
Abs	Модуль	Перегружен (существует несколько версий метода)	x записывается как Abs(x)
Acos	Арккосинус	double	Acos(double x)
Asin	Арксинус	double	Asin(double x)
Atan	Арктангенс	double	Atan(double x)
Atan2	Арктангенс	double	Atan2(double x, double y) - угол, тангенс которого есть результат деления y на x
BigMul	Произведение	long	BigMul (int x, int y)
Ceiling	Округление до большего целого	double	Ceiling(double x)
Cos	Косинус	double	Cos (double x)
Cosh	Гиперболический косинус	double	Cosh (double x)
DivRem	Деление и остаток	Перегружен	DivRem(x, y, rem)
E	База натурального логарифма (число e)	double	2,71828182845905
Exp	Экспонента	double	ex записывается как Exp(x)
Floor	Округление до меньшего целого	double	Floor(double x)
IEEERemainder	Остаток от деления	double	IEEERemainder(double x, double y)
Log	Натуральный логарифм	double	$\log_e x$ записывается как Log(x)
Log10	Десятичный логарифм	double	$\log_{10} x$ записывается как Log10(x)
Max	Максимум из двух чисел	Перегружен	Max(x,y)
Min	Минимум из двух	Перегружен	Min(x,y)

	чисел		
PI	Значение числа пи	double	3,14159265358979
Pow	Возведение в степень	double	x^y записывается как Pow (x,y)
Round	Округление	Перегружен	Round (3.1) даст в результате 3, Round (3.8) даст в результате 4
Sign	Знак числа	int	Аргументы перегружены
Sin	Синус	double	Sin(double x)
Sinh	Гиперболический синус	double	Sinh(double x)
Sqrt	Квадратный корень	double	\sqrt{x} записывается как Sqrt(x)
Tan	Тангенс	double	Tan(double x)
Tanh	Гиперболический тангенс	double	Tanh(double x)

2.4 Структура программы

Рассмотрим каждую строку заготовки программы (листинг 1).

Листинг 1 – Заготовка консольной программы

```

Using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main(string[] args)
        {
            // TODO: Add code to start application here
        }
    }
}

```

Директива `using System` разрешает использовать имена стандартных классов из пространства имен `System`, непосредственно (без указания имени пространства). Ключевое слово `namespace` создает для проекта собственное пространство имен, названное по умолчанию `ConsoleApplication1`. Это сделано для того, чтобы можно было давать программным объектам имена, не заботясь о том, что они могут совпасть с именами в других пространствах имен.

Строки, начинающиеся с двух или трех косых черт, являются комментариями и предназначены для документирования текста программы. `C#` — объектно-ориентированный язык, поэтому написанная на нем программа представляет собой совокупность взаимодействующих между собой классов. В заготовке программы всего один класс, которому задано имя `Class1`. Описание класса начинается с ключевого слова `class`, за которым следуют его имя и далее в фигурных скобках — список элементов класса (его данных и функций, называемых также методами).

Фигурные скобки являются важным элементом синтаксиса. Каждой открывающей скобке соответствует своя закрывающая, которая обычно располагается ниже по тексту с тем же отступом. Эти скобки ограничивают

блок, внутри которого могут располагаться другие блоки, вложенные в него. Блок может применяться в любом месте, где допускается отдельный оператор.

В данном случае внутри класса только один элемент — метод Main. Каждое приложение должно содержать метод Main — с него начинается выполнение программы. Все методы описываются по единым правилам.

Упрощенный синтаксис метода:

```
[ спецификаторы ] тип имя_метода ( [ параметры ] )  
{  
    тело метода: действия, выполняемые методом  
}
```

Для описания языка программирования необязательные части синтаксических конструкций заключаются в квадратные скобки, а текст, который необходимо заменить конкретным значением, пишется по-русски, а возможность выбора одного из нескольких элементов обозначается вертикальной чертой. Например:

```
[ void | int ] имя_метода( ) ;
```

Эта запись означает, что вместо конструкции имя_метода необходимо указать конкретное имя в соответствии с правилами языка, а перед ним может находиться либо слово void, либо слово int, либо ничего. Символ подчеркивания используется для связи слов вместо пробела, показывая, что на этом месте должен стоять один синтаксический элемент, а не два. В тех случаях, когда квадратные скобки являются элементом синтаксиса, это оговаривается особо.

Таким образом, любой метод должен иметь тип, имя и тело, остальные части описания являются необязательными, поэтому они пока игнорируются.

Внутри метода Main помещен комментарий:

```
// TODO: Add code to start application here
```

Это означает: «Добавьте сюда код, выполняемый при запуске приложения». Последуем совету и добавим после строк комментария (но не в той же строке!) строку

```
Console.WriteLine("Первая программа");
```

Здесь Console — это имя стандартного класса из пространства имен System. Его метод WriteLine выводит на экран заданный в кавычках текст. Как видите, для обращения к методу класса используется конструкция

имя_класса.имя_метода

Если вы не сделали ошибку в первом же слове, то сразу после ввода с клавиатуры следом за словом Console символа точки среда выведет подсказку, содержащую список всех доступных элементов класса Console. Выбор нужного имени выполняется либо мышью, либо клавишами управления курсором, либо вводом одного или нескольких начальных символов имени. При нажатии клавиши Enter выбранное имя появляется в тексте программы.

Возможностью автоматического ввода не следует пренебрегать, так как он позволяет избежать опечаток и экономит время. Если подсказка не появляется, это свидетельствует об ошибке в имени или в месте расположения в программе вводимого текста.

Для сохранения файла воспользуйтесь командой File > Save главного меню или кнопкой Save на панели инструментов (текстовый курсор должен при этом находиться в окне редактора). Впрочем, при запуске программы среда сохранит исходный текст самостоятельно.

2.5 Линейные алгоритмы и программы

Линейным называется алгоритм, все блоки которого выполняются последовательно в том порядке, в котором они записаны. Простейшим примером линейного алгоритма является расчет по заданной формуле. Он состоит из трех этапов: ввод исходных данных, вычисление по формуле и вывод результатов (рис. 1).

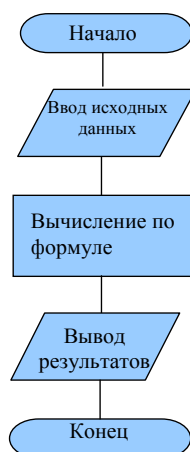


Рисунок 1- Схема линейного алгоритма

Для того чтобы написать программу по данному алгоритму, необходимы знания о том, как организовать ввод и вывод на языке C#. Подробно этот вопрос рассматривается в лабораторной работе № 11, а здесь приводятся только минимально необходимые сведения.

2.4.1 Простейший ввод-вывод

Любая программа при вводе исходных данных и выводе результатов взаимодействует с внешними устройствами. Совокупность стандартных устройств ввода и вывода, используемых на лабораторных работах, - это клавиатура и экран, называемые *консолью*. Обмен данными с консолью является частным случаем обмена с внешними устройствами.

Строго говоря, в языке C#, как и во многих других, нет операторов ввода и вывода. Вместо них для обмена с внешними устройствами применяются стандартные объекты.

Простейшие способы вывода

Для работы с консолью в C# применяется класс `Console`, определенный в пространстве имен `System`. Методы этого класса `Write` и `WriteLine` и будут использоваться в дальнейших программах. Рассмотрим листинг 2.

Листинг 2 – Методы вывода

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 3;
            double y = 4.12;
            decimal d = 600m;
            string s = "Пробный расчет";
            Console.WriteLine ( "y={0} \nd={1}",y,d );
            Console.WriteLine ( "i = " + i);
            Console.WriteLine ( "s = " + s);
        }
    }
}
```

Результат работы программы:

```
y = 4.12
d = 600
i = 3
s = Пробный расчет
```

Метод `WriteLine` может быть использован для вывода значений переменных и литералов различных встроенных типов. Это возможно благодаря тому, что в классе `Console` существует несколько вариантов методов с именами `Write` и `WriteLine`, предназначенных для вывода значений различных типов.

Методы с одинаковыми именами, но разными параметрами называются перегруженными. Компилятор определяет, какой из методов вызван, по типу передаваемых в него величин. Методы вывода в классе `Console` перегружены для всех встроенных типов данных, кроме того, предусмотрены варианты форматного вывода.

Листинг 2 содержит два наиболее употребительных варианта вызова методов вывода. Сначала обратите внимание на способ вывода пояснений к значениям переменных. Пояснения представляют собой строковые литералы. Если метод `WriteLine` вызван с *одним* параметром, он может быть любого встроенного типа, например, числом, символом или строкой. Нам же требуется вывести в каждой строке не одну, а две величины: текстовое пояснение и значение переменной, — поэтому прежде чем передавать их для вывода, их требуется «склеить» в одну строку с помощью операции `+`.

Перед объединением строки с числом надо преобразовать число из его внутренней формы представления в последовательность символов, то есть в

строку. *Преобразование в строку* определено во всех стандартных классах C# — для этого служит метод ToString(). В данном случае он выполняется неявно, но можно вызвать его и явным образом:

```
Console.WriteLine ( "i = " + i.ToString( ) );
```

Оператор

```
Console.WriteLine ( "y={0} \nd={1}",y,d );
```

использует другой вариант метода WriteLine - форматный вывод. При этом оператор вывода содержит более одного параметра. Первым параметром методу передается строковый литерал, содержащий помимо обычных символов, предназначенных для вывода на консоль, *параметры* в фигурных скобках, а также *управляющие последовательности*.

Параметры нумеруются с нуля, перед выводом они заменяются значениями соответствующих переменных в списке вывода: нулевой параметр заменяется значением первой переменной (в данном примере — y), первый параметр — второй, переменной (в данном примере — d) и т. д.

Из управляющих последовательностей чаще всего используются символы перевода строки (\n), как в данном случае, и горизонтальной табуляции (\t).

Простейшие способы ввода с клавиатуры

Рассмотрим простейшие способы *ввода с клавиатуры*. В классе Console определены методы ввода строки и отдельного символа, но нет методов, которые позволяют непосредственно считывать с клавиатуры числа. Ввод числовых данных выполняется в два этапа:

1. Символы, представляющие собой число, вводятся с клавиатуры в строковую переменную.

2. Выполняется преобразование из строки в переменную соответствующего типа.

Преобразование можно выполнить либо с помощью специального класса Convert, определенного в пространстве имен System, либо с помощью метода Parse, имеющегося в каждом стандартном арифметическом классе. В листинге 1 используются оба способа.

Ниже приведены примеры организации ввода для различных типов данных:

Ввод строки:

```
Console.WriteLine ( "Введите строку" );  
string s = Console.ReadLine ( );
```

Ввод символа:

```
Console.WriteLine ( "Введите символ" );  
char c = (char) Console.Read ( );  
Console.ReadLine( );
```

Ввод символа выполняется с помощью метода **Read**, который считывает только один символ из входного потока (оператор 2). Метод возвращает значение типа **int**, представляющее собой код символа, или -1, если символов во входном потоке нет (например, пользователь нажал клавишу Enter). Поскольку нам требуется не **int**, а **char**, а неявного преобразования от **int** к **char** не существует, приходится применить операцию явного преобразования типа.

Метод **Read**, в отличие от **ReadLine**, не очищает буфер, и, если оператор 3 будет отсутствовать, то следующий после него ввод будет выполняться с того места, на котором закончился предыдущий. Поэтому за оператором 2 записан оператор 3, выполняющий вспомогательные функции. Он считывает остаток строки, тем самым очищая буфер.

Ввод целого числа:

```
string buf;  
Console.WriteLine ( "Введите целое число" );  
buf = Console.ReadLine();  
int i =Convert.ToInt32( buf );
```

В приведенном варианте ввода целого числа введенная информация помещается в строковую переменную **buf**, а затем ее значение с помощью метода **Convert.ToInt32** преобразуется в целый тип.

Ввод вещественного числа (при вводе вещественных чисел дробная часть отделяется от целой с помощью запятой, а не точки):

```
Console.WriteLine ( "Введите вещественное число" );  
buf = Console.ReadLine( );  
double x = Convert.ToDouble( buf );
```

или

```
Console.WriteLine ( "Введите вещественное число" );  
buf = Console.ReadLine( );  
double y = double.Parse( buf );
```

Допускается задавать числа с порядком, например, 1,95e-8.

При вводе целых и вещественных чисел используются или методы класса **Convert**, или метод **Parse** класса **Double** библиотеки .NET, который используется здесь через имя типа C# **double**.

2.4.4 Примеры

Пример 1. Написать программу, производящую простые арифметические операции над числами (сложение, вычитание, умножение и деление). Текст программы представлен в листинге 3.

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            float num1, num2;
            Console.Write("Введите первое число:");
            num1 = float.Parse(Console.ReadLine());
            Console.Write("Введите второе число:");
            num2 = float.Parse(Console.ReadLine());
            float summarize = num1 + num2; float multiply = num1 * num2;
            float sub = num1 - num2; float divide = num1 / num2;
            Console.WriteLine(
                "\n" + num1 + " + " + num2 + " = " + summarize +
                "\n" + num1 + " * " + num2 + " = " + multiply +
                "\n" + num1 + " - " + num2 + " = " + sub +
                "\n" + num1 + " / " + num2 + " = " + divide);
            Console.Write("\nДля выхода из программы нажмите [Enter]:");
            string anykey = Console.ReadLine();
        }
    }
}

```

Пример 2

Даны X,Y. Составить программу для получения значения C.

$$C = \frac{|X| - |Y|}{1 - |X * Y|}$$

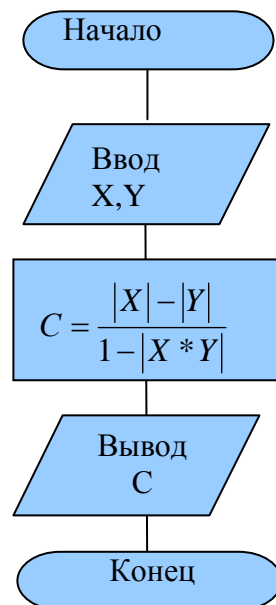


Рисунок 2- Схема алгоритма к примеру 2

Листинг 3 - К примеру 2

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)

```

```

{
    double c, y, x;
    Console.Write("Enter X:");
    x = Convert.ToDouble(Console.ReadLine());
    Console.Write(x);
    Console.Write("Enter Y:");
    y = Convert.ToDouble(Console.ReadLine());
    c = (Math.Abs(x) - Math.Abs(y)) / (1 - Math.Abs(x * y));
    Console.WriteLine("Result = {0}", c);
}
}
}

```

Использованные функции:

- ***System.ConsoleWrite()***, ***System.ConsoleWriteLine()*** - выводит сообщения в консоль;
- ***System.ConsoleReadLine()*** - считывает строки с консоли.
- ***System.Convert.ToDouble()*** - преобразовывает данные строкового типа в вещественное число.
- ***System.Math.Abs()*** - возвращает абсолютное значение числа(модуль)

Пример 3. Вычислить расстояние между двумя точками с координатами (x1,y1) и (x2, y2).

Решение

Расстояние между точками А и В — это длина вектора \overrightarrow{AB} .

Вектор : $\overrightarrow{AB} = \{x_B - x_A, y_B - y_A\}$

Длина вектора : $\overrightarrow{AB} = \sqrt{x^2 + y^2}$

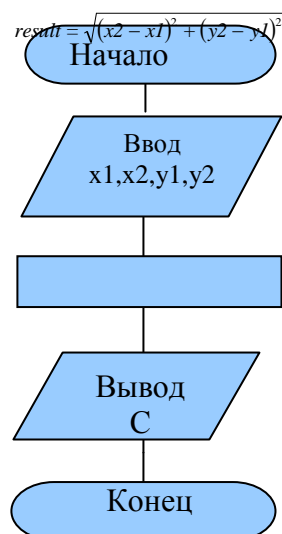


Рисунок 4 - Схема алгоритма к примеру 3

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double x, x1, y, y1, s;
            Console.Write("Enter X for first point:");
            x = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter Y for first point:");
            y = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter X for second point:");
            x1 = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter Y for second point:");
            y1 = Convert.ToDouble(Console.ReadLine());
            s = Math.Sqrt(Math.Pow(x1 - x, 2) + Math.Pow(y1 - y, 2));
            Console.WriteLine("S={0}", s);
        }
    }
}

```

Использованные функции:

- ***System.Math.Pow(x,a)*** — возводит число x в степень a.

3 Варианты заданий для самостоятельной работы

1. Определить объем конуса с заданным радиусом основания R и высотой H.
2. Найти среднее арифметическое кубов двух чисел, вводимых с клавиатуры и среднее арифметическое этих чисел.
3. По двум данным катетам найти гипотенузу и площадь прямоугольного треугольника.
4. Числа X, Y, Z вводятся с клавиатуры. Вычислить их сумму, разность и произведение.
5. Даны A, B, C. Найти площадь треугольника, стороны которого равны A и B, угол между ними C.
6. Смешано V1 литров воды температуры t1 с V2 литрами воды температуры t2. Вычислить объем и температуру полученной смеси. V1, V2, t 1, t2 вводить с клавиатуры.
7. Найти площадь грани, площадь поверхности и объем куба, длина ребра которого вводится с клавиатуры.
8. Координаты вершин треугольника (X1,Y1); (X2,Y2); (X3,Y3) вводится с клавиатуры. Найти периметр треугольника.

9. Координаты вершин треугольника (X1,X2); (X2,Y2); (X3,Y3) вводятся с клавиатуры. Найти площадь полученного треугольника.

10. Вычислить
$$U = \sin((y - |x|) * (x - \frac{y}{z^2 + (x^2/4)}))$$
 и

$$V = \cos(z^2 + \frac{x^2}{4}),$$

если у, х, z - вводятся с клавиатуры.

11. Вычислить

$$U = \frac{1 + \sin^2(x+y)}{\frac{2*x}{2 + |x - \sin(x+y)|}}$$
 и

$$V = x - \frac{x^2}{1 + \sin^2(x+y)},$$

если х, у - вводятся с клавиатуры.

12. Даны с и d. Вычислить:

$$A = \sin^3 |c*x1^3 + d*x2^2 - c*d|$$

где: x1 - больший, x2-меньший корень уравнения $x^2 - 3*x - |c*d| = 0$

13. Вычислить значения $A = 1 - 2*x + 3*x^3 - 4*x^2$ и $B = 1 + 2*x + 3*x^3 + 4*x^2$ при х, вводимом с клавиатуры.

14. Для х, вводимого с клавиатуры, вычислить $C = 2*x^4 - 3*x^3 - 5*x + 6$.

15. Найти корни квадратного уравнения $A*x^2 - B*x + C = 0$ для А, В и С, вводимых с клавиатуры.

16. Найти площадь сектора, радиус которого равен 13.7, а дуга содержит 1.5 радиан.

17. а, b, с - стороны треугольника. Определить углы треугольника.

18. Треугольник задан координатами вершин (x1,y1), (x2,y2) и (x3,y3). Найти периметр треугольника.

19. Дана сторона равностороннего треугольника. Найти площадь этого треугольника.

20. Вычислить период колебания маятника длины L.

21. Определить силу притяжения F между телами массы m1 и m2 на расстоянии r друг от друга.

22. Даны гипотенуза и катет. Найти второй катет и радиус описанной окружности.
23. Дается длина окружности. Найти площадь круга, ограниченного этой окружностью.
24. Найти площадь кольца, внутренний радиус которого равен 20, а внешний радиус - r ($r > 20$).
25. Вычислите дробную часть среднего геометрического трех заданных вещественных чисел.
26. Определить время, через которое встретятся два тела, равноускоренно движущиеся друг к другу. Известны: v_1 и v_2 - начальные скорости; a_1 и a_2 - ускорения; s - расстояние между ними.
27. По условию задачи 3 найти площадь треугольника.
28. Треугольник задан длинами сторон a , b и c . Найти длины высот.
29. Определить периметр правильного n - угольника, описанного около окружности с радиусом R .
30. Три сопротивления R_1 , R_2 и R_3 соединены параллельно. Найти сопротивление соединения.
31. Даны значения a и b . Найти их среднее арифметическое, среднегеометрическое, остатки при делении a на b и b на a .
32. Определить время падения камня на поверхность Земли с высоты h .
33. По условию задачи 3 определить длины медиан.
34. Дано действительное число a . С помощью умножения получить a^4 за две операции; a^8 - за три операции; a^9 - за четыре операции; a^{15} - за пять операций.
35. По условию задачи 3 определить длины биссектрис.
36. Найти радиус окружности, проходящей через три заданные точки на плоскости.
37. Определить объем и площадь боковой поверхности цилиндра с заданным радиусом основания R и высотой H .
38. Ввести сторону квадрата a . Вычислить радиус вписанной окружности.
39. Определить углы треугольника со сторонами a , b и c .
40. Определить напряжения на параллельных сопротивлениях R_1 и R_2 при токе I .
41. Найти объем правильной треугольной пирамиды со стороной при основании a и высотой h .
|
42. Вычислить период колебаний физического маятника с грузом массой m и пружиной жесткости k .

43. Определить коэффициент наклона прямой, заданной на плоскости двумя точками с координатами $(X1, Y1)$ и $(X2, Y2)$.
44. По двум катетам определить радиус вписанной окружности для прямоугольного треугольника.
45. Для эллипса заданы длины полуосей a и b , вводимые с клавиатуры. Вычислить фокусное расстояние и площадь эллипса.
46. Определить время, через которое встретятся два тела, равноускоренно движущиеся друг к другу. Известны: $v1$ и $v2$ - начальные скорости; $a1$ и $a2$ - ускорения; s - расстояние между ними.
47. Определить периметр правильного n - угольника, описанного около окружности с радиусом R .
48. Определить силу притяжения F между телами массы $m1$ и $m2$ на расстоянии r друг от друга.
49. Найти корни квадратного уравнения $A \cdot x^2 - B \cdot x + C = 0$ для A , B и C , вводимых с клавиатуры.
50. a , b , c - стороны треугольника. Определить углы треугольника.
51. Найти площадь грани, площадь поверхности и объем куба, длина ребра которого вводится с клавиатуры.
52. Определить объем конуса с заданным радиусом основания R и высотой H .

4. Содержание отчета

- 4.1. Наименование и цель работы.
- 4.2. Индивидуальное задание на лабораторную работу.
- 4.3. Схема алгоритма решения задачи.
- 4.4. Текст программы на алгоритмическом языке.
- 4.5. Результаты вычислений.

Лабораторная работа № 3

ПРОГРАММИРОВАНИЕ ВЕТВЯЩИХСЯ ПРОЦЕССОВ

1 Цель и порядок работы

Цель работы - изучить процесс построения алгоритмов ветвящихся вычислительных процессов и операторы, используемые при организации программ такого рода, получить практические навыки в составлении программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- разработать схему алгоритма решения задачи;
- написать программу, соответствующую схеме алгоритма;
- ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

Довольно часто необходимо провести анализ некоторых данных и выбор варианта действия в зависимости от результата анализа. Указанной цели служат условные операторы.

В C# существует два вида операторов ветвления: условный оператор `if` и оператор выбора `switch`.


2.1 Алгоритмы разветвленной структуры

Формат записи `if`-инструкции принимает такой вид:

```
if (условие) {  
    последовательность инструкций  
}  
else {  
    последовательность инструкций  
}
```

Если элемент *условие*, который представляет собой условное выражение, при вычислении даст значение *истина*, будет выполнена *if*-инструкция; в противном случае - *else*-инструкция (если таковая существует). Обе инструкции одновременно никогда не выполняются. Условное выражение, управляющее выполнением *if*-оператора, должно иметь тип *bool*. Схема алгоритма условного оператора представлена на рис. 1.

Начало



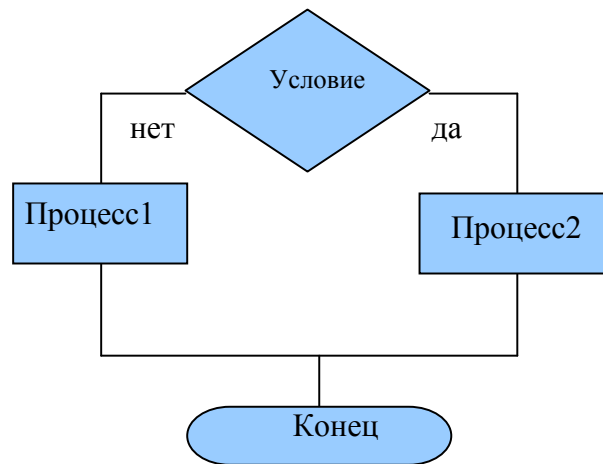


Рисунок 1- Схема алгоритма условного оператора

Вложенные *if*-операторы образуются в том случае, если в качестве элемента инструкция используется другой *if*-оператор. Вложенные *if*-операторы очень популярны в программировании. Главное здесь - помнить, что *else*-инструкция всегда относится к ближайшей *if*-инструкции, которая находится внутри того же программного блока, но еще не связана ни с какой другой *else*-инструкцией.

Оператор *switch* обеспечивает многонаправленное ветвление. Он позволяет делать выбор одной из множества альтернатив. Хотя многонаправленное ветвление можно реализовать и с помощью последовательности вложенных *if*-операторов, для многих ситуаций оператор *switch* оказывается более эффективным решением.

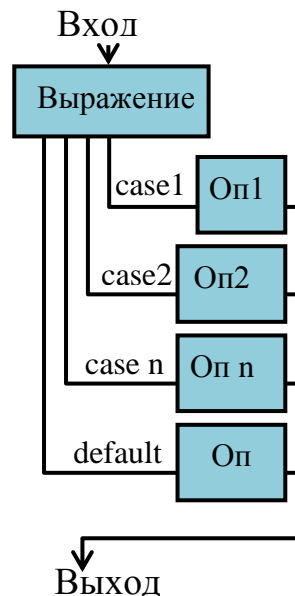


Рисунок 2- Схема алгоритма ветвящегося процесса с оператором *switch*

Общий формат записи оператора *switch* имеет вид:
Switch (выражение)

```

{
    case константа1:
        последовательность инструкций
        break;
    case константа2:
        последовательность инструкций
        break;
    case константа3:
        последовательность инструкций
        break;
    default:
        последовательность инструкций
        break;
}

```

Элемент *выражение* оператора **switch** должен иметь целочисленный тип (например, **char**, **byte**, **short** или **int**) или тип **string**. Выражения, имеющие тип с плавающей точкой, не разрешены. Очень часто в качестве управляющего **switch**-выражения используется просто переменная; **case**-константы должны быть литералами, тип которых совместим с типом заданного выражения. При этом никакие две **case**-константы в одной **switch**-операторе не могут иметь идентичных значений. Последовательность инструкций **default**-ветви выполняется в том случае, если ни одна из заданных **case**-констант не совпадет с результатом вычисления **switch**-выражения.

Ветвь **default** необязательна. Если она отсутствует, то при несовпадении результата выражения ни с одной из **case**-констант никакое действие выполнено не будет. Если такое совпадение все-таки обнаружится, будут выполнены инструкции, соответствующие данной **case**-ветви до тех пор, пока не встретится инструкция **break** или любой другой оператор перехода (**goto**, **return**). Оператор **switch** может быть использован как часть **case**-последовательности внешнего оператора **switch**. В этом случае он называется вложенным оператором **switch**. Необходимо отметить, что **case**-константы внутренних и внешних операторов **switch** могут иметь одинаковые значения, при этом никаких конфликтов не возникнет.

Пример 1. Переменные X, Y, Z имеют положительные значения. Присвоить переменной P значение 1, если можно построить треугольник с длинами сторон X, Y, Z , и 0 — в противном случае.

Решение

В любом треугольнике сумма любых двух сторон всегда больше третьей стороны.

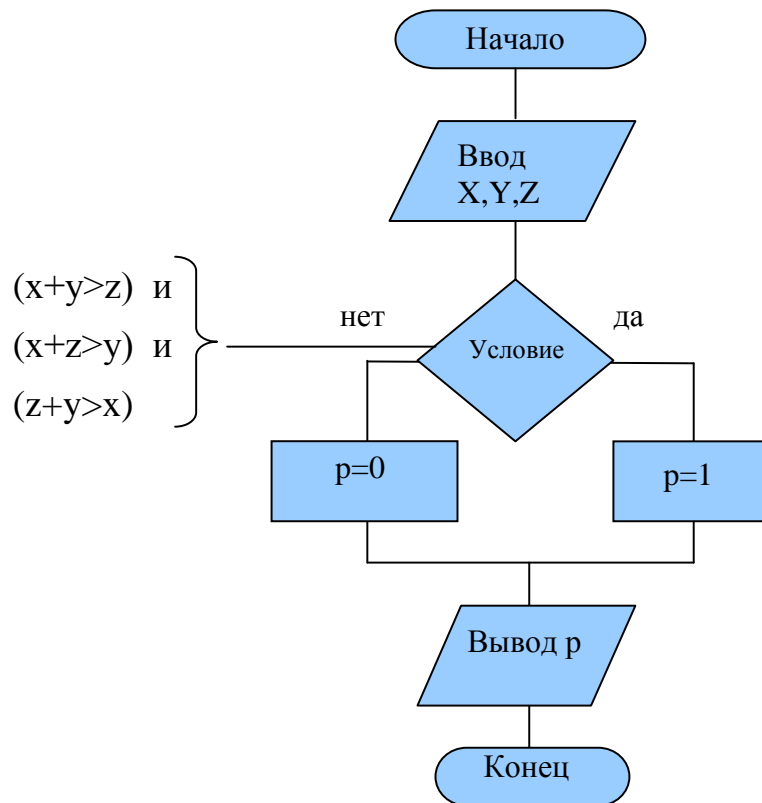


Рисунок 3- Схема алгоритма к примеру 1

Листинг 1-к примеру 1

```
using System;
class lab3z1
{
    public static void Main()
    {
        double x,y,z;
        int p;
        Console.Write("Enter X ");
        x=Convert.ToDouble(Console.ReadLine());
        Console.Write("Enter Y ");
        y=Convert.ToDouble(Console.ReadLine());
        Console.Write("Enter Z ");
        z=Convert.ToDouble(Console.ReadLine());
        if (((x+y)>z) && ((x+z)>y) && ((z+y)>x)) p=1;
        else p=0;
    }
}
```

```

        Console.WriteLine("P={0}", p);
    }
}

```

Пример 2. Даны вещественные числа a, b, c . Найти наибольшее из них.

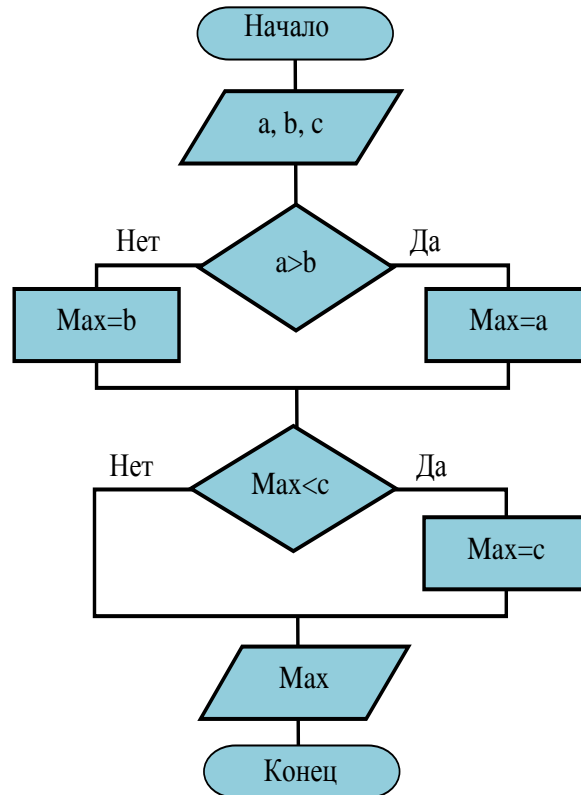


Рисунок 4 - Схема алгоритма к примеру 2

На этой схеме показано, что после ввода трех исходных чисел выполняется сравнение первых двух: a и b . Наибольшее из них присваивается переменной Max . После чего переменная Max сравнивается с переменной c . В том случае, если $Max < c$, значение Max корректируется заменой его на значение c . В заключении найденное наибольшее значение из трех исходных чисел выводится на экран монитора.

Листинг 2 – к примеру 2

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void Main()
        {
            double a, b, c, max;
            Console.Write("Enter A: ");

```

```

a = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter B: ");
b = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter C: ");
c = Convert.ToDouble(Console.ReadLine());
if (a < b) max = b;
else max = a;
if (c > max) max=c;
Console.WriteLine("{0},{1},{2},{3}", a, b, c, max);
}
}
}

```

Пример 3. Переменная целого типа **I** определяет порядковый номер месяца невысокосного года. Оператор **Switch** по порядковому номеру месяца выводит количество дней в данном месяце.

Листинг 3 –к примеру 3

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string buf;
            Console.Write("Введите номер месяца: ");
            buf = Console.ReadLine();
            int a = Convert.ToInt32(buf);
            switch (a)
            {
                case 1: Console.WriteLine ("В месяце 31 день"); break;
                case 3: Console.WriteLine ("В месяце 31 день"); break;
                case 5: Console.WriteLine ("В месяце 31 день"); break;
                case 7: Console.WriteLine ("В месяце 31 день"); break;
                case 8: Console.WriteLine ("В месяце 31 день"); break;
                case 10: Console.WriteLine ("В месяце 31 день"); break;
                case 12: Console.WriteLine ("В месяце 31 день"); break;
                case 2: Console.WriteLine ("В месяце 28 дней"); break;
                default: Console.WriteLine ("В месяце 30 дней");break;
            }
        }
    }
}

```

В программе, решающей сформулированную задачу, введенное значение **I** используется для отыскания требуемого оператора вывода. Выбирается такой оператор **Console.WriteLine**, перед которым за словом **case** есть введенное значение **I**. Поскольку в данной программе нет защиты от некорректной работы пользователя, то любое введенное значение **I>12** или

$I < 1$ также приведет к выводу следующего сообщения: *'В месяце 30 дней'*, так как именно оно помещено в ветвь **default**.

В данном примере I — это переменная целого типа, но вместо нее может быть использовано и выражение, выдающее порядковое значение.

Подчеркнем, что значения констант селектора, помещенные в любую ветвь выбора, не следует смешивать с метками операторов для работы с операторами перехода **Goto**.

3. Варианты заданий для самостоятельной работы

1. Даны целые числа X, Y, Z . Если числа не равны, то заменить каждое из них одним и тем же числом, равным меньшему из исходных, а если равны, то заменить большим из исходных.

2. Даны произвольные числа A, B и C . Если нельзя построить треугольник с такими длинами сторон, то напечатать 0, иначе напечатать 3, 2 или 1 в зависимости от того, равносторонний это треугольник, равнобедренный или какой-либо иной.

3. Для заданного a найти корень уравнения $f(x)=0$, где

$$f(x) = \begin{cases} 2^a x + \text{abs}(a-1) & \text{при } a > 0 \\ e^x & \\ (1+a^2)-1 & \text{иначе.} \end{cases}$$

4. Даны числа A, B, C ($A \neq 0$). Найти вещественные корни уравнения $a x^2 + b x + c = 0$. Если корней нет, то сообщить об этом.

5. По номеру y ($y > 0$) некоторого года определить s — номер его столетия (учесть, что, к примеру, началом 20-го столетия был 1901, а не 1900 год).

6. Вычислить

$$u = \frac{(\max(x, y, z))^2 - 2^x \cdot \min(x, y, z)}{\max(x, y, z) / \min(x, y, z)}$$

x, y, z - заданы.

7. Считая, что стандартные функции SIN и COS применимы только к аргументам на отрезке $[0, \pi/2]$, вычислить $y = \sin(x)$ для произвольного числа x .

Формулы приведения:

$$\sin(x) = \sin(\pi - x)$$

$$\sin(x) = \cos(\pi/2 - x)$$

$$\sin(x) = \sin(2\pi n + x)$$

n -произвольное целое число.

8. Значения переменных A, B, C поменять местами так, чтобы оказалось $A \geq B \geq C$. Переменные A, B, C заданы.

9. Даны числа a, b, c, d, e, f . Напечатать координаты точки пересечения прямых, описываемых уравнениями $a \cdot x + b \cdot y = c$ и $d \cdot x + e \cdot y = f$, либо сообщить, что эти прямые совпадают, не пересекаются или вовсе не существуют.
10. Даны координаты точки (x, y) . Определить, в каком квадранте находится точка, и напечатать номер квадранта.
11. Определить, являются ли значения целочисленных переменных N и M кратными 3. Если оба значения кратны 3, то вычислить их сумму, в противном случае – разность.
12. Вычислить

$$Q = \begin{cases} a_1, & \text{если } x < v_1 \\ a_2, & \text{если } v_1 \leq x < v_2 \\ a_3, & \text{если } v_2 \leq x < v_3 \\ a_4, & \text{если } v_3 \leq x \end{cases}$$

$$\begin{aligned} a_1 &= -\sqrt{x+v_1} \\ a_2 &= -1/(x+b_2); \\ a_3 &= -b_3 \cdot b_2; \\ a_4 &= b_1 + b_3; \end{aligned} \quad , \text{ если } x, b_1, b_2, b_3 \text{ заданы.}$$
13. Определить, являются ли значения целочисленных переменных N и M кратными 3. Если оба значения кратны 3, то вычислить их сумму, в противном случае – разность.
14. Даны значения a, b, c ($a^2 > 0$). Определить, имеет ли уравнение $a \cdot x^2 + b \cdot x + c$ действительные корни. Если нет, то выдать соответствующее сообщение.
15. Даны вещественные числа a, b, c и m . Если $a < b < c > m$, то каждое число заменить наибольшим из них.
16. Даны вещественные x и y . Если x и y отрицательны, то каждое значение заменить его модулем. Если отрицательно одно из них, то оба значения увеличить на 0.5.
17. Даны вещественные x и y . Если оба значения неотрицательны и одно из них не принадлежит отрезку $[0.5; 2.5]$, то оба значения уменьшить в 10 раз. В остальных случаях x и y оставить без изменения.
18. Даны значения x, y и z . Выбрать те из них, которые принадлежат интервалу $(1; 3)$.
19. Даны значения a и b . Меньшее из двух заменить их полусуммой, а большее – их удвоенным произведением.
20. Даны значения a, b и c . Возвести в квадрат те из них, значения которых неотрицательны.

21. Даны значения a , b и c . Проверить, выполняются ли неравенства $a > b > c$ или $a < b < c$. Выдать соответствующие сообщения.
22. Даны значения a , b и c . Удвоить эти числа, если $a > b > c$, и заменить их абсолютными значениями, если это не так.
23. Даны действительные числа $a_1, b_1, c_1, a_2, b_2, c_2$. Проверить условие $a_1 * b_2 - a_2 * b_1 > 0.0001$, при котором система совместна и имеет единственное решение. Найти это решение.
24. Даны действительные числа $x_1, x_2, x_3, y_1, y_2, y_3$. Принадлежит ли начало координат треугольнику с вершинами $(x_1, y_1), (x_2, y_2), (x_3, y_3)$?
25. Даны действительные числа a, b, c и x, y . Выяснить, пройдет ли кирпич с ребрами a, b, c в прямоугольное отверстие со сторонами x и y .
26. Определить, является ли данное целое число k четным?
27. Даны числа a, b, c, d , составляющие четырехзначное число. Определить, все ли четыре цифры различны.
28. Даны целые числа K, L . Если числа не равны, то заменить каждое из них одним и тем же числом, равным большему из исходных, а если равны, то заменить нулями.
29. Даны значения переменных x, y . Выяснить, принадлежит ли точка (x, y) кругу единичного радиуса с центром в начале координат.
30. Выяснить, принадлежит ли точка с координатами (x, y) квадрату со стороной единица и с центром в начале координат.
31. Выяснить, принадлежит ли точка с координатами (x, y) пересечению окружности $x^2 + y^2 = 1$ и параболы $y = x^2$.
32. Выяснить, принадлежит ли точка с координатами (x, y) пересечению окружности $x^2 + y^2 = 1$ и прямой $y = x/2$ в своей нижней части.
33. Если значение переменной W не равно 0 и котангенс от W меньше 0.5, то поменять знак W . В противном случае W присвоить 1.
34. Даны значения x, y и z . Выбрать те из них, которые принадлежат интервалу $(1; 3)$.
35. Даны значения a и b . Меньшее из двух заменить их полусуммой, а большее – их удвоенным произведением.
36. Даны значения a, b и c . Проверить, выполняются ли неравенства $a > b > c$ или $a < b < c$. Выдать соответствующие сообщения.
37. Даны значения a, b и c . Удвоить эти числа, если $a > b > c$, и заменить их абсолютными значениями, если это не так.
38. Определить, будет ли остаток при делении целых чисел i и j , равен одному из заданных значений a и b .
39. Дано четырехзначное число. Определить, все ли четыре цифры различны.

40. Даны целые числа K, L . Если числа не равны, то заменить каждое из них одним и тем же числом, равным большему из исходных, а если равны, то заменить нулями.
41. Даны действительные числа x, y, z . Вычислить $\max(x, y, z)^2 - \min(x, y, z)$.
42. Даны действительные числа a, b, c .
Вычислить: $z = \begin{cases} \max(a, b, c) & \text{при } a > b; \\ \min(a, b, c) & \text{при } b < c; \\ \min(a, b) * \max(a, c) & \text{иначе.} \end{cases}$
43. Даны значения переменных x, y . Выяснить, принадлежит ли точка с координатами (x, y) кругу единичного радиуса с центром в точке с координатами (v, z) .
44. Выяснить, принадлежит ли точка с координатами (x, y) квадрату со стороной единица и с центром в точке с координатами (v, z) .
45. Даны действительные числа a, b, c, d . Если $a > b > c > d$, то числа оставить без изменения. В противном случае значение a должно стать максимальным из всех значений.
46. Определить, является ли данное целое число X нечетным?
47. Дано пятизначное число. По признаку деления определить, делится ли число на 3.
48. Даны целые числа X, Y, Z . Если числа не равны, то заменить каждое из них одним и тем же числом, равным меньшему из исходных, а если равны, то заменить большим из исходных.
49. Даны действительные числа x, y, z .
Вычислить $\min(x, y, z) / (\max(x, y, z) * 3)$.
50. Дана точка с координатами (X, Y, Z) . Определить, находится ли она внутри сферы с центром $(2, 3, 4)$ и радиусом 3.
51. Даны точки A и B с координатами (X_1, Y_1) и (X_2, Y_2) . Найти тангенс угла, образованного прямой AB и осью Y .
52. Дана точка с координатами (X, Y, Z) . Определить длину окружности, проходящей через эту точку и начало координат.

4. Содержание отчета

- 4.1. Наименование и цель работы.
- 4.2. Индивидуальное задание на лабораторную работу.
- 4.3. Схема алгоритма решения задачи.
- 4.4. Текст программы на алгоритмическом языке.
- 4.5. Результаты вычислений.

Лабораторная работа № 4

ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ПРОЦЕССОВ

1 Цель и порядок работы

Цель работы - изучить операторы, используемые при организации циклических вычислительных процессов, получить практические навыки в составлении программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- разработать схему алгоритма решения задачи;
- написать программу, соответствующую схеме алгоритма;
- ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

2.1 Понятие цикла

С помощью уже рассмотренных к данному моменту операторов, можно реализовать в программе только структуры следования и ветвления, которые обеспечивают выполнение операторов, входящих в программу, не более одного раза. Поэтому ясно, что, используя только эти структуры, можно реализовать лишь небольшой процент алгоритмов. Возможностей этих операторов явно недостаточно для решения подавляющего большинства задач, связанных с организацией повторяющихся вычислений, обработкой массивов данных и т. д.

Для реализации подобных вычислений предназначены операторы циклов, которые задают повторяющееся выполнение одного или нескольких операторов (блока) и позволяют записать эти действия в компактной форме.

Цикл — одна из важнейших алгоритмических структур. Переменные, изменяющиеся в цикле, называются переменными цикла. Параметром цикла называется переменная, которая используется при проверке условия продолжения работы цикла и принудительно изменяется в теле цикла, очень часто на одну и ту же величину. Рассмотрим важнейшие концепции организации циклов.

Алгоритм циклической структуры в наиболее общем виде должен содержать:

- подготовку цикла: задание начальных значений переменным цикла перед первым его выполнением;

- тело цикла; действия, повторяемые в цикле для различных значений переменных цикла;
- изменение значений переменных цикла при каждом новом его выполнении;
- управление циклом: проверку условия продолжения работы цикла.

Один проход тела цикла называется итерацией. Если параметр цикла целочисленный, он называется *счетчиком цикла*.

Различают арифметические циклы, количество повторений которых можно определить заранее, и циклы с неизвестным числом повторений (итерационные циклы). В итерационном цикле условие продолжения работы цикла содержит переменные, значения которых изменяются в цикле по рекуррентным формулам. Рекуррентной называется формула, в которой новое значение переменной вычисляется с использованием ее предыдущего значения.

Передавать управление извне внутрь цикла запрещается (при этом возникает ошибка компиляции).

2.2 Операторы управления

Любой цикл завершается, если условие его продолжения не выполняется. Возможно принудительное завершение как текущей итерации, так и цикла в целом. Для этого используются специальные операторы, которые относятся к группе операторов, изменяющих естественный порядок выполнения вычислений.

В C# есть пять таких операторов:

- оператор безусловного перехода *goto*;
- оператор выхода из цикла *break*;
- оператор перехода к следующей итерации цикла *continue*;
- оператор возврата из функции *return*;
- оператор генерации исключения *throw*.

Эти операторы могут передать управление в пределах блока, в котором они использованы, и за его пределы. Передавать управление внутрь другого блока запрещается.

Первые четыре оператора рассматриваются в этой лабораторной работе, а оператору *throw* посвящена специальная лабораторная работа.

Оператор *goto*

Оператор безусловного перехода используется в одной из трех форм:

goto метка;

goto case константное_выражение;

goto default;

В теле функции должна присутствовать ровно одна конструкция вида **метка: оператор;**

Оператор *goto* метка передает управление на помеченный оператор.

Метка — это обычный идентификатор, областью видимости которого

является функция, в теле которой он задан. Метка должна находиться в той же области видимости, что и оператор перехода. После выполнения оператора **Goto** опять восстанавливается естественный порядок выполнения операторов, но начиная уже с помеченного оператора. Если помеченный оператор не является оператором управления, то будет выполняться следующий за ним оператор и т. д.

При использовании **Goto** следует учитывать следующие требования:

- **Goto** не должен передавать сообщения внутрь цикла;
- вход в блок должен осуществляться через заголовок.

В общем случае необходимо избегать применения оператора **Goto** и по возможности стараться писать программы без этого оператора, так как его использование нарушает принципы структурного и модульного программирования, по которым все блоки, образующие программу, должны иметь только один вход и один выход. Использование этой формы оператора безусловного перехода оправдано в двух случаях:

- принудительный выход вниз по тексту программы из нескольких вложенных циклов или переключателей;
- переход из нескольких точек функции вниз по тексту в одну точку (например, если перед выходом из функции необходимо всегда выполнять какие-либо действия).

В остальных случаях для записи любого алгоритма существуют более подходящие средства, а использование оператора **goto** приводит только к усложнению структуры программы и затруднению отладки.

Вторая и третья формы оператора **goto** используются в теле оператора выбора switch.

Оператор:

goto case константное_выражение

передает управление на соответствующую константному выражению ветвь, а оператор

goto default — на ветвь default.

Оператор break

Оператор break используется внутри операторов цикла или выбора для перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится оператор **break**.

Оператор continue

Оператор перехода к следующей итерации текущего цикла continue пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации.

Оператор return

Оператор возврата из функции ***return*** завершает выполнение функции и передает управление в точку ее вызова. Синтаксис оператора:

return [выражение];

Тип выражения должен иметь неявное преобразование к типу функции. Если тип возвращаемого функцией значения описан как `void`, выражение должно отсутствовать.

2.3 Операторы циклов

В С# существует четыре вида циклических операторов: цикл с параметром ***for***, цикл с предусловием ***while***, цикл с постусловием ***do*** и цикл перебора ***foreach***. Все эти структуры повторения имеют как общие моменты в своей организации и функционировании, так и отличия. Далее рассмотрим все структуры повторения более подробно. Но, для того чтобы лучше уяснить общие моменты работы циклических операторов и их отличия друг от друга, реализуем одну и ту же пару простейших задач с помощью различных циклических структур.

Оператор цикла for

Схема алгоритма цикла с параметром представлена на рис. 1.

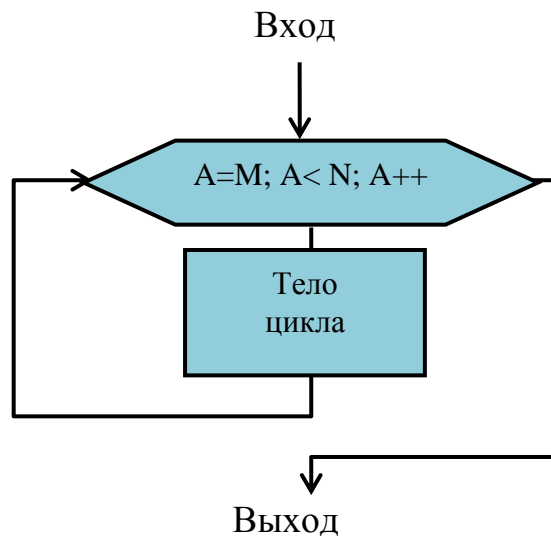


Рисунок 1 - Схема алгоритма цикла с параметром

Для организации цикла такой структуры в С# используется специальный оператор цикла ***for***. Оператор цикла ***for*** это такой оператор, где все управление циклом сосредоточено в его заголовке. Он по сравнению с другими циклическими операторами компактнее и понятнее и любим программистами.

Цикл ***for*** – это одна из наиболее гибких конструкций С#, так как допускает множество вариантов. Весьма важным в работе рассматриваемой структуры повторения является понятие: параметр цикла или управляющая

переменная (переменные) цикла. Назначение их вполне очевидно — управлять работой цикла.

Общий формат записи оператора **for** имеет следующий вид:

```
for (инициализация; условие; итерация)  
{последовательность операторов}
```

Элемент *инициализация* обычно представляет собой инструкцию присваивания, которая, с одной стороны, служит для объявления параметров, используемых в цикле, а с другой стороны, устанавливает им начальные значения. Каждый параметр цикла действует в качестве счетчика. Если в части «*инициализация*» требуется записать несколько операторов присваивания, то их необходимо разделять запятой, например:

```
for ( int i = 0, j = 20; ...  
    int k, m;  
    for (k = 1, m = 0; ...
```

Областью действия переменных, объявленных в части инициализации цикла, является цикл. Инициализация выполняется один раз в начале исполнения цикла. Цикл с параметром реализован как цикл с предусловием.

Элемент *условие* представляет собой выражение типа **bool**, в котором проверяется значение управляющей переменной (переменных) цикла. Результат этого тестирования определяет, выполнится ли цикл **for** еще раз или нет. Цикл **for** будет выполняться до тех пор, пока вычисление элемента *условие* дает истинный результат. Как только условие станет ложным, выполнение программы продолжится с инструкции, следующей за циклом **for**. В общем случае в качестве логического выражения, управляющего работой цикла **for**, может быть использовано любое допустимое выражение, результат которого имеет тип **bool**.

Элемент *итерация* - это выражения, которые определяют, как изменяются значения параметров цикла после каждой итерации.

Простой или составной *оператор* представляет собой тело цикла. Все части заголовка оператора цикла **for** должны отделяться точкой с запятой. Любая из частей оператора **for** может быть опущена (но точки с запятой надо оставить на своих местах!).

Оставив пустым условное выражение цикла **for**, можно создать бесконечный цикл (цикл, который никогда не заканчивается). Например:

```
for (;;) {  
// специально созданный бесконечный цикл  
}
```

Большинство "бесконечных циклов" - это просто циклы со специальными требованиями к завершению.

В отличие от Pascal, в котором к параметру цикла предъявляются довольно жесткие требования, в C# все спокойнее и проще: переменная цикла может быть дробным числом, либо отсутствовать вообще, так же шаг цикла может быть дробным.

В общем случае надо стремиться к минимизации области действия

переменных. Это облегчает поиск ошибок в программе.

Пример 1. Известны интервал ($x_n \div x_k$), шаг изменения аргумента h_x . Написать программу для вычисления значений функции для каждого значения аргумента (задача табуляции функции). Тип аргумента и тип значения функции – вещественные.

С точки зрения алгоритмизации задача табуляции функции является арифметическим циклом, так как число повторений цикла можно легко рассчитать.

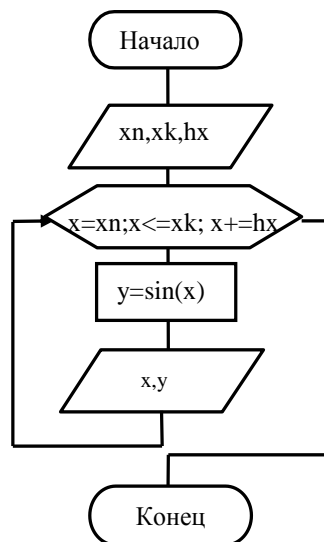


Рисунок 2 - Схема алгоритма с оператором цикла **for** к примеру 1

В данном примере параметром цикла является вещественная переменная x , которая изменяется от x_n до x_k с каждой итерацией на величину h_x .

Листинг 1 – К примеру 1

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double xn, xk, hx, y;
            Console.Write("Enter xn ");
            xn = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter xk ");
            xk = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter hx ");
            hx = Convert.ToDouble(Console.ReadLine());
            for (double x = xn; x <= xk; x += hx)
            {
                y = Math.Sin(x);
                Console.WriteLine("x=" + x + "    y=" + y);
            }
        }
    }
}
```

```

    }
}

```

Пример 2. Вычислить произведение натурального ряда чисел от 1 до 25, т.е. значение факториала 25, математическая запись которого - $25!$.

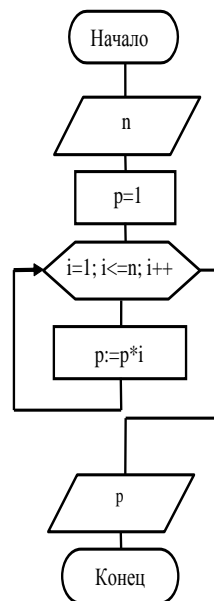


Рисунок 3 - Схема алгоритма с оператором цикла *for* к примеру 2

В данной программе управляющая переменная циклом **i** изменяется от **1** с шагом **+1**, принимая последовательно все значения натурального ряда чисел до **n** включительно, которые и накапливаются как сомножители в произведении **p**. После выхода из цикла значение этого произведения и определит значение искомого факториала, которое будет выведено на экран монитора.

Листинг 2 – К примеру 2

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int p=1,n;
            Console.Write("Enter n ");
            n = Convert.ToInt32(Console.ReadLine());
            for (int i = 1; i <= n; i++)
                p = p * i;
            Console.WriteLine("p="+ p);
        }
    }
}

```

Оператор цикла с предусловием while

Не лишним будет вспомнить схему алгоритма организации цикла с предусловием, представленную на рис.4.

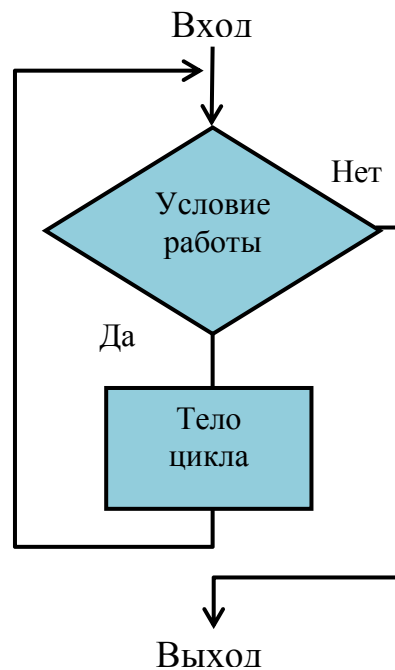


Рисунок 4 - Схема алгоритма цикла с предусловием к примеру 1

Для организации цикла такой структуры в С# используется специальный оператор цикла ***while***. Общая синтаксическая форма данного оператора цикла имеет вид:

```
while (условие) {  
    последовательность операторов  
}
```

Здесь под элементом *последовательность операторов* понимается либо одиночный оператор, либо блок операторов. Работой цикла управляет элемент *условие*, который представляет собой любое допустимое выражение типа ***bool***. Элемент *инструкция* выполняется до тех пор, пока условное выражение возвращает значение ***true***. Как только это условие становится ложным, управление передается оператору, который следует за этим циклом.

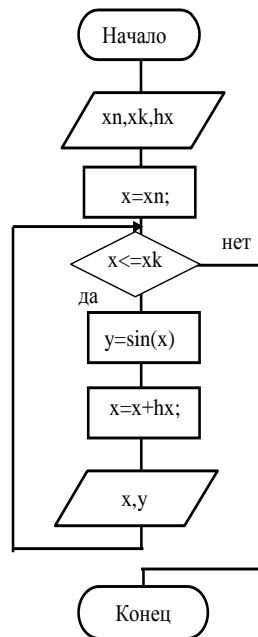


Рисунок 5 - Схема алгоритма с циклом с предусловием к примеру 1

Листинг 3 – с циклом с предусловием к примеру 1

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double xn, xk, hx, y;
            Console.Write("Enter xn ");
            xn = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter xk ");
            xk = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter hx ");
            hx = Convert.ToDouble(Console.ReadLine());
            double x = xn;
            while ( x <= xk)
            {
                y = Math.Sin(x);
                Console.WriteLine("x=" + x + "    y=" + y);
                x += hx;
            }
        }
    }
}

```

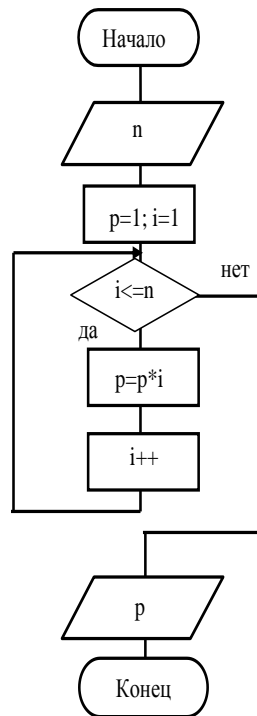


Рисунок 6 - Схема алгоритма с циклом с предусловием к примеру 2

Листинг 4 – с циклом с предусловием к примеру 2

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int p=1,n,i=1;
            Console.Write("Enter n ");
            n = Convert.ToInt32(Console.ReadLine());
            while (i <= n)
            {
                p = p * i;
                i++;
            }
            Console.WriteLine("p="+ p);
        }
    }
}

```

Оператор цикла с постусловием do

Напомним схему алгоритма организации цикла с постусловием, представленную на рис.7.

Эта разновидность циклических структур характеризуется тем, что тело цикла обязательно выполняется хотя бы один раз.

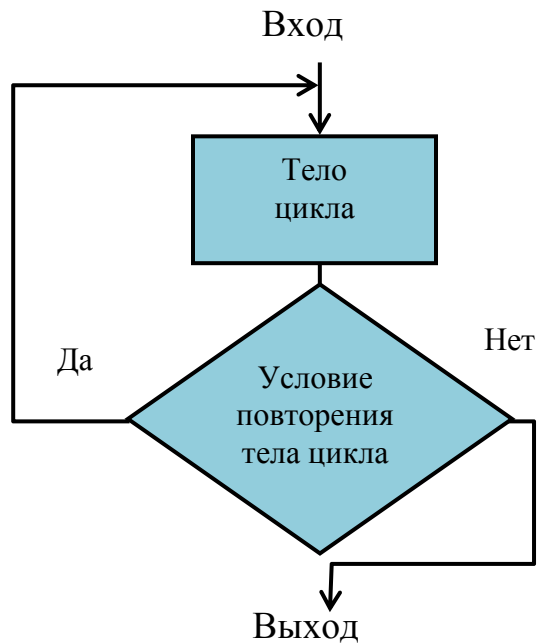


Рисунок 7 - Схема алгоритма цикла с постусловием

Для организации цикла такой структуры в С# используется специальный оператор цикла ***do***. Цикл с постусловием имеет следующий формат:

```
do  
последовательность операторов  
while выражение;
```

Сначала выполняется простой или составной оператор, образующий тело цикла, а затем вычисляется выражение (оно должно иметь тип ***bool***). Если выражение истинно, тело цикла выполняется еще раз, и проверка повторяется. Цикл завершается, когда выражение примет значение ***false***, или в теле цикла будет выполнен какой-либо оператор передачи управления.

Этот вид цикла применяется в тех случаях, когда тело цикла необходимо обязательно выполнить хотя бы один раз, например, если в цикле вводятся данные и выполняется их проверка. Если же такой необходимости нет, предпочтительнее пользоваться циклом с предусловием.

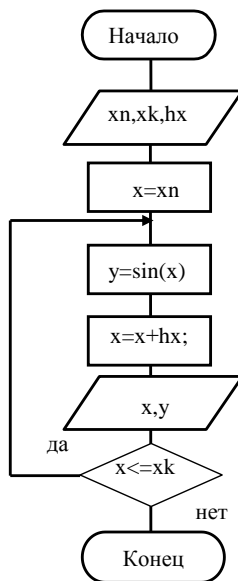


Рисунок 8 - Схема алгоритма с циклом с постусловием к примеру 1

Листинг 5 – с циклом с постусловием к примеру 1

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double xn, xk, hx, y;
            Console.Write("Enter xn ");
            xn = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter xk ");
            xk = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter hx ");
            hx = Convert.ToDouble(Console.ReadLine());
            double x = xn;
            do
            {
                y = Math.Sin(x);
                Console.WriteLine("x=" + x + "   y=" + y);
                x += hx;
            } while (x <= xk);
        }
    }
}

```

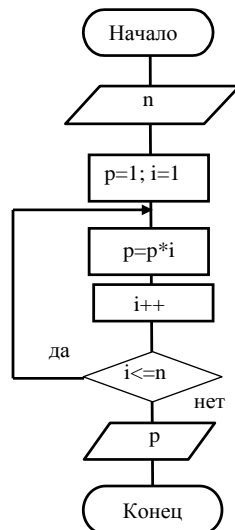


Рисунок 9 - Схема алгоритма с циклом с постусловием к примеру 2

Листинг 6 – с циклом с постусловием к примеру 2

```

using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int p = 1, n, i = 1;
            Console.Write("Enter n ");
            n = Convert.ToInt32(Console.ReadLine());
            do
            {
                p = p * i;
                i++;
            } while (i <= n);
            Console.WriteLine("p=" + p);
        }
    }
}

```

Оператор цикла с перебором *foreach*

Цикл перебора *foreach* используется для просмотра всех объектов из некоторой группы данных – коллекции. В языке С# определен ряд типов коллекций, например, таковыми являются: массивы, списки или другие контейнеры. С этим оператором познакомимся несколько позже.

3. Варианты заданий для самостоятельной работы

1. Дано натуральное n . Написать программу вычисления a^n .
2. Дано натуральное n . Написать программу вычисления значений

$$\frac{1}{1*2} + \frac{1}{2*3} + \dots + \frac{1}{(n-1)*n}, n \geq 2$$

3. Дано натуральное n . Написать программу вычисления значений

$$1 - \frac{1}{2} + \dots + \frac{(-1)^{(n-1)}}{n}$$

4. Дано натуральное n . Написать программу вычисления значений

$$\left(1 + \frac{1}{1^2}\right) * \left(1 + \frac{1}{2^2}\right) * \dots * \left(1 + \frac{1}{n^2}\right)$$

5. Дано число A . Написать программу получения в порядке убывания всех делителей данного числа.

6. Обозначим

$$f = \frac{1}{i^2+1} + \frac{1}{i^2+2} + \dots + \frac{1}{i^2+i+1} \quad i=0,1,\dots$$

Написать программу вычисления произведения

$$f * f * \dots * f \quad . n - \text{дано.}$$

7. Объем V - цилиндрической подковы вычисляется по формуле

$$V = \frac{h}{3b} [a(3r^2 - a^2) + 3r^2 * (b-r)] * \frac{u(n)}{180}$$

Составить алгоритм для построения графика зависимости V от угла u , если a , b и r известны, а u изменяется в диапазоне от u_1 до u_2 с шагом du .

8. Составить алгоритм для расчета функции

$$y = \frac{0,95 * (\sin(x))^3}{1 + 0,95 * x^2}$$

при изменении x от 0 до 12 с шагом $dx=0.2$.

9. Самолет летит из пункта A к пункту B со средней скоростью v . Составить алгоритм для нахождения времени полета t_1 , если есть встречный ветер, скорость которого v_1 , и времени t_2 , если ветра нет. Расстояние между пунктами A и B считать известным и равным S . Скорость ветра v_1 может изменяться от 0 до 15 м/с. Считать шаг изменения $dv_1 = 0.5$ м/с.
10. Цех вводится в строй постепенно, выдавая в первый день $x_1\%$ продукции от нормы, во второй - $x_2\%$, в третий - $x_3\%$,..., в n -й день - $x_n\%$. Составить алгоритм для расчета продукции S за n дней, если в первый день цех выдал A продукции.
11. Составить алгоритм для расчета функции y

при значениях $x=0; 0.1; 0.2; \dots; 10$.

$$y = \begin{cases} \frac{x-1}{2x^2+3}, & \text{если } x \leq 1 \\ 1.05(x-1)^2, & \text{если } x > 1. \end{cases}$$

12. Дана функция $y=0.5+\sin(5x)$, причем x изменяется от 0 до 2π с шагом $dx=\pi/6$. Составить алгоритм для расчета функции y .

13. Постоянная времени T электрической цепи равна $T=RC$, где R и C - соответственно сопротивление и емкость цепи. Составить алгоритм решения задачи при условии, что R изменяется от 10^2 Ом до 10^3 Ом с шагом $dR=10^2$ Ом.

14. Составить алгоритм для вычисления функции

$$z = \begin{cases} \frac{1}{v+y}, & \text{если } 0 < y < 1.36; dy=0.136; \\ 0, & \text{если } y=0; \\ \frac{1}{v-2y}, & \text{если } -1.5 < y < 0; dy=0.15. \end{cases}$$

15. Кинетическая энергия движущегося тела $W=m*v^2/2$, где m - масса тела, v - его скорость. Составить алгоритм для получения зависимости W от m при значениях v , изменяющихся от v_1 до v_2 с шагом dv . Масса изменяется от m_1 до m_2 с шагом dm .

16. Дана функция $x=a*\sin(k*t+2)*\cos(k*t)$. Составить алгоритм для расчета этой функции, если a изменяется от 5 до 7 с шагом 0.12, t изменяется от 4.2 до 6.2 с шагом 0.17 а $k=1,2,3,\dots,12$.

17. Составить алгоритм для нахождения итерационным методом с точностью до ϵ корня уравнения $2*x-3*\sin(x)=0$.

18. Дана функция $y=-2*x^2+3*x+1.5$. Составить алгоритм для поиска максимального значения y , если x изменяется в диапазоне $0.1 \leq x \leq 1$ с шагом $dx=0.01$.

19. Слесарь-сантехник начинает работу в 8 ч. и заканчивает в 17 ч., делая перерыв на обед с 12 до 13 ч. Через каждые 30 мин работы он устраняет течь в кранах водопроводной системы одной из квартир жилого дома. Составить алгоритм для вычисления количества воды Q , которое вытечет из неисправных кранов за рабочий день слесаря - сантехника, если утечка воды в одной квартире составляет Q [л/мин].

20. Даны функции $y_1=x^3$ и $y_2=\sin(x)$. Составить алгоритм для вычисления точки пересечения этих функций при $x>0$. Вычисления производить с точностью до ϵ .
21. Даны целые числа n и $K[n]$. Получить сумму S последних цифр чисел $K[n]$.
22. Дано целое число K . Найти сумму цифр числа K . (Пусть запись K в десятичной системе есть $a[n]...a[2] a[1] a[0]$; найти $a[0]+a[1]+a[2]+...+a[n]$).
23. Даны действительные числа a, h , целое n . Вычислить:
 $f(a) + f(a+h)+f(a+2h)+...+ f(a+nh)$, где $f(x)=(x^2+1)/x$.
24. Даны целое n , действительное x . Вычислить: $\sin x + \sin^2 x + ... + \sin^n x$.
25. Дано целое n . Чему равно произведение его цифр?
26. Дано вещественное число a . Найти среди чисел
 $1; 1+1/2; 1+1/2+1/3 \dots$ первое, большее a .
27. Дано целое K . Сколько цифр в числе 5^K ?
28. Дано вещественное x . Вычислить:
 $s=(x-2)+(x-4)+(x-8)+...+(x-64)$;
 $p=(x-1)*(x-3)*(x-7)...(x-63)$.
29. Даны действительные числа x, a , целое число i . Вычислить:
 $((...((x+a)^2 + a)^2 + ... + a)^2 + a, (0 < a < 9)$.
30. Дано вещественное число x . Вычислить:
 $x - x^3/3! + x^5/5! - x^7/7! + x^9/9! - x^{11}/11! + x^{13}/13!$
31. Вычислить $(1 - \sin 0.1) + (1 + \sin 0.2) + ... + (1 + \sin 1.0)$.
32. Даны целое n , вещественное x . Вычислить:
 $\cos x + \cos x^2 + ... + \cos x^n$
33. Дано целое число n . Найти первую цифру числа n .
34. Дано целое число n . Вычислить: $3+6+...+3*(n-1)+3n$.
35. Дано целое n . Вычислить:
 $1/\sin 1 + 1/(\sin 1 + \sin 2) + ... + 1/(\sin 1 + ... + \sin n)$.
36. Дано целое n . Вычислить: $(1+1/1^2) (1+1/2^2) ... (1+1/n^2)$.
37. Дано вещественное число a , целое n . Вычислить:
 $a * (a-1) * (a-2) * ... * (a-n)$.
38. Дано вещественное число a , целое n . Вычислить:
 $1/a + 1/a^2 + 1/a^4 + ... + 1/a^{2^n}$
39. Дано вещественное число a , целое n . Вычислить:
 $1/a + 1/(a(a+1)) + ... + 1/(a(a+1)...(a+n))$.
40. Дано целое число K . Выяснить, входит ли цифра 3 в запись числа K ?
41. Дано целое число K . Поменять порядок цифр в числе на обратный.
43. Дано целое число n . Получить $a[n]$, имея зависимость $a[0]=1$;
 $a[k]=k*a[k-1]+1/k; k=1,2,...$
44. Дано целое n ($n>4$). Получить $a[n]$, если
 $a[1]=a[2]=0; a[3]=1.5$;
 $a[i]=(i+1)/(i^2+1) a[i-1]-a[i-2]*a[i-3], i=4, 5, ...$
45. Даны вещественные q, r, b, c, d , целое n ($n>2$). Получить

$$x[n], \quad \text{если} \quad x[0]=c; \quad x[1]=d; \\ x[k]=qx[k-1]+rx[k-2]+b, \quad k=2, 3, \dots, n$$

46. Найти произведение $a[0] a[1] \dots a[14]$, если $a[0]=a[1]=1$; $a[k]=a[k-2]+a[k-1]/2^{(k-1)}$, $k=2, 3, \dots$
47. Дано 50 вещественных чисел. Найти величину наибольшего из них.
48. Дано целое $N>0$, за которым следует N вещественных чисел. Определить, сколько среди них отрицательных.
49. Дана непустая последовательность положительных целых чисел, за которой следует 0 (это признак конца последовательности). Вычислить среднее геометрическое этих чисел.
50. Вычислить значение суммы функционального ряда (бесконечности)

$$S = \sum_{n=1}^{\infty} \frac{2 \cdot x^{(2 \cdot n + 1)}}{2 \cdot n + 1}$$

с точностью $E=0.001$ при заданном x .

51. Дано целое $N>0$, за которым следует N целых чисел. Определить, сколько среди них простых чисел.
52. Дано целое $N>0$, за которым следует N целых чисел. Определить, сколько среди них четырехзначных чисел.

4. Содержание отчета

- 4.1. Наименование и цель работы.
- 4.2. Индивидуальное задание на лабораторную работу.
- 4.3. Схема алгоритма решения задачи.
- 4.4. Текст программы на алгоритмическом языке.
- 4.5. Результаты вычислений.

Лабораторная работа № 5

ПРОГРАММИРОВАНИЕ ИТЕРАЦИОННЫХ ЦИКЛОВ И ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С УСЛОЖНЕННОЙ СТРУКТУРОЙ

1 Цель и порядок работы

Цель работы – познакомиться со структурным подходом разработки алгоритмов сложных вычислительных процессов, получить практические навыки программирования подобного рода задач.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- разработать схему алгоритма решения задачи;
- написать программу, соответствующую схеме алгоритма;
- ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

В данной лабораторной работе приводится ряд подходов с подробными рассуждениями к решению некоторых типов различных задач.

2.1 Итерационные циклы

Итерационным циклом называется такой, число повторений которого заранее неизвестно или его сложно рассчитать, а в процессе повторения тела цикла образуется последовательность значений $y_1, y_2, \dots, y_n, \dots$, сходящаяся к некоторому пределу a , т. е.

$$\lim y_n = a.$$

$$n \rightarrow \infty$$

Каждое новое значение y_n в такой последовательности определяется с учетом предыдущего y_{n-1} и является по сравнению с ним более точным приближением к искомому результату. Итерационный цикл заканчивает свою работу в том случае, если для некоторого значения n выполняется условие:

$$|y_n - y_{n-1}| \leq \varepsilon,$$

где ε - допустимая погрешность вычисления результата.

В такой ситуации за результат принимают последнее значение y , т.е. считают, что y_n с заданной точностью представляет значение a .

Задача вычисления суммы бесконечного ряда может служить прекрасной иллюстрацией к пониманию итерационного циклического

процесса. Естественно, вычислять сумму бесконечного ряда имеет смысл в том случае, когда бесконечный ряд является сходящимся. Известно, что ряд сходится, если его общий член z_n при беспредельном возрастании n стремится к нулю, т. е.

$$\lim_{n \rightarrow \infty} z_n = 0; \quad \lim_{n \rightarrow \infty} (s_n - s_{n-1}) = 0.$$

А сумма

$$s_n = z_0 + z_1 + \dots + z_n$$

его первых $(n + 1)$ слагаемых при этом стремится к некоторому пределу S , который и называется суммой ряда, т. е.

$$\lim_{n \rightarrow \infty} s_n = S; \quad \lim_{n \rightarrow \infty} \sum z_n = S.$$

Алгоритм, реализующий подсчет суммы ряда, должен вырабатывать последовательность $s_1, s_2, \dots, s_n, \dots$ при следующем условии окончания суммирования:

$$|z_n| \leq \varepsilon$$

Пример 1. Вычислить значение функции $\cos x$, используя разложение косинуса в ряд с заданной погрешностью ε :

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} + \dots$$

При построении алгоритма данной задачи необходимо:

- определить значение очередного слагаемого z ;
- осуществлять накопление суммы по итерационной формуле:

$$s = s + z.$$

Для определения значения очередного слагаемого z в данном примере целесообразно использовать не прямое его вычисление по общей формуле, а рекуррентное соотношение, которое позволяет существенно сократить количество операций при вычислении его значения:

$$z = z * f_n.$$

Определим сомножитель f_n :

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

Для работы алгоритма (рис.1) весьма важно определить исходную информацию для работы цикла. Подставив в данную формулу $n = 0$, получим начальное значение $z = 1$.

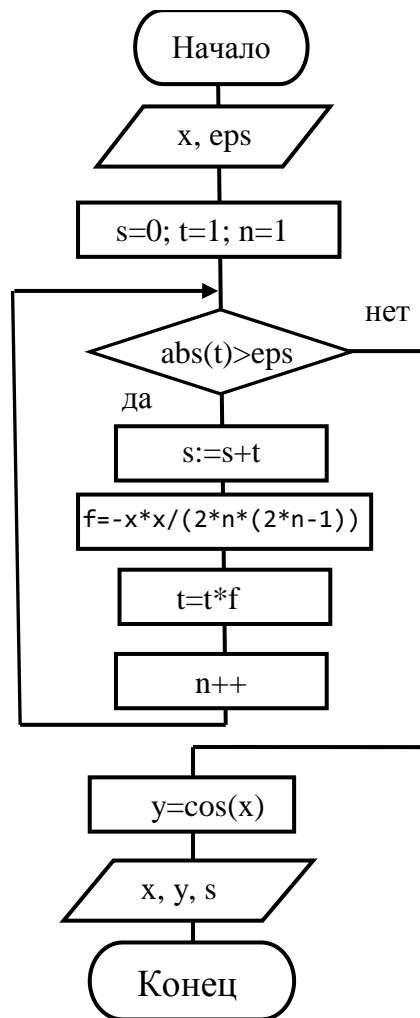


Рисунок 1 – Схема алгоритма к примеру 1

Все вышесказанное реализовано в программе (лист. 1), соответствующей схеме данного алгоритма.

Листинг 1

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double x, eps, s, t, y, f;
            Console.Write("Enter x ");
            x = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter eps ");
            eps = Convert.ToDouble(Console.ReadLine());
            s=0;
            t=1;
            int n=1;
            while (Math.Abs(t) > eps)
            {
                s = s + t;
                f = -x*x / (2*n*(2*n-1));
                t = t*f;
                n++;
            }
            y = Math.Cos(x);
            Console.WriteLine("x = {0}, y = {0}, s = {0}", x, y, s);
        }
    }
}

```

//Начало цикла

```

    }
    y= Math.Cos (x);
    Console.WriteLine("x=" + x + "    y=" + y+ "    s="+s);
}
}
}

```

Для организации цикла по накоплению суммы используется оператор цикла с предусловием, в котором условие $|z| > \varepsilon$ является условием продолжения цикла.

Вторым характерным примером использования итерационных циклов является задача решения алгебраических и нелинейных (трансцендентных) уравнений.

Нахождение корней уравнений вида

$$f(x) = 0$$

осуществляется в два этапа.

Первый этап – *этап локализации* корня – определяется отрезок $[\alpha, \beta]$, в пределах которого находится один и только один корень уравнения. Часто локализация корня осуществляется построением «грубого» графика функции $f(x)$.

Второй этап – *этап уточнения корня* – ведется поиск корня с заданной степенью точности с помощью некоторого итерационного алгоритма.

Наиболее простым методом уточнения корня является метод итераций, заключающийся в следующем. Исходное уравнение $f(x) = 0$, где $f(x)$ - непрерывная функция на отрезке $[\alpha, \beta]$, заменяют эквивалентным уравнением вида:

$$x = \varphi(x)$$

и, зная начальное приближение корня $x_0 \in [\alpha, \beta]$, каждое следующее приближение находят по формуле:

$$x_n = \varphi(x_{n-1}).$$

Вычисления повторяют до тех пор, пока не выполнится условие $|f(x_n)| \leq \varepsilon$, где ε - заданная погрешность вычислений. Отметим, что иногда используют другой способ контроля сходимости, состоящей в сравнении x_n и x_{n-1} . Процесс сходится, если на всем отрезке $[\alpha, \beta]$ $|\varphi'(x)| < 1$, где $\varphi'(x)$ есть производная функции $\varphi(x)$.

Пример 2. Определить корень уравнения $x - \operatorname{tg}(x) = 0$ с погрешностью $\varepsilon = 10^{-3}$ при начальном значении корня $x_0 = 4,5$.

Преобразуем исходное уравнение следующим образом:

$$x = \operatorname{tg} x,$$

$$\text{тогда} \quad f(x) = x - \operatorname{tg} x; \quad \varphi(x) = \operatorname{tg} x.$$

Для определения x_0 удобно использовать следующие графические построения. Построить графики функций $y_1 = x$; $y_2 = \varphi(x) = \operatorname{tg} x$. Точки

пересечения этих графиков и будут корнями исходного уравнения $f(x)=0$. На графике выделить приближенные отрезки $[\alpha, \beta]$ локализации каждого корня. В качестве x_0 можно взять любую точку отрезка.

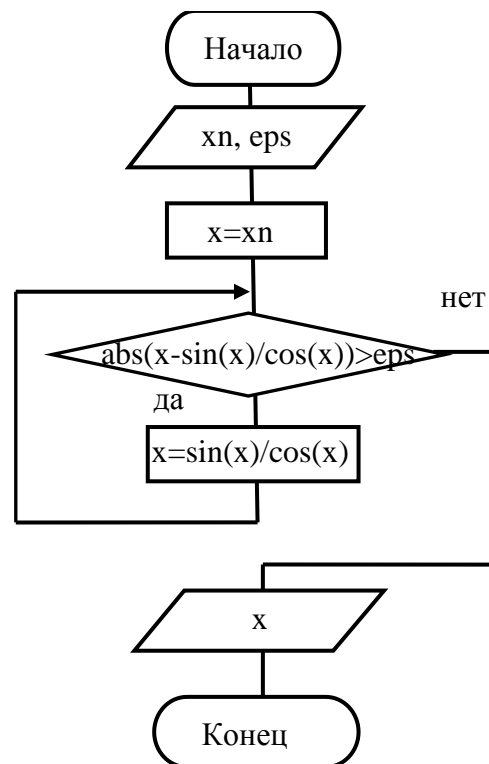


Рисунок 2 – Схема алгоритма к примеру 2

Программа уточнения корня по методу итераций представлена в листинге 2.

Листинг 2

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double x, xn, eps;
            Console.Write("Enter xn ");
            xn = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter eps ");
            eps = Convert.ToDouble(Console.ReadLine());
            x=xn;
            while (Math.Abs (x-Math.Sin(x)/Math.Cos(x))>eps)
                x=Math.Sin(x)/Math.Cos(x);
            Console.WriteLine("x=" + x);
        }
    }
}
  
```

В данной программе на печать в качестве результата выводится последнее значение x , которое удовлетворяет заданному значению Eps .

Для практического контроля сходимости итерационного процесса $x_n = \varphi(x_{n-1})$ можно осуществлять печать промежуточных значений x , последнее из которых и будет корнем уравнения при заданной погрешности.

2.2 Вложенные циклы

Если телом цикла является циклическая структура, то такие циклы называют *вложенными* или *сложными*. Цикл, содержащий в себе другой цикл, называют *внешним*. Цикл, содержащийся в теле другого цикла, называют *внутренним*.

Внутренний и внешний циклы могут быть любыми из трех рассмотренных видов: циклами с параметром, циклами с предусловием, циклами с постусловием. При построении вложенных циклов необходимо соблюдать следующее дополнительное условие: все операторы внутреннего цикла должны полностью лежать в теле внешнего цикла, циклы ни в коем случае не могут пересекаться.

Сложные циклы условно разбивают на *уровни вложенности*. Ниже представлена структура вложенных циклов с параметром, для которой: внешний цикл *1* имеет уровень 0, внутренний цикл *2* – уровень 1, внутренний цикл *3* – уровень 2.

Возможная глубина вложенности циклов (количество уровней) ограничивается объемом имеющейся памяти ЭВМ. Заметим, что цикл *2* является внешним по отношению к циклу *3* и внутренним по отношению к циклу *1*. Параметры циклов разных уровней изменяются не одновременно. Вначале все свои значения изменит параметр самого внутреннего цикла при фиксированных значениях параметров циклов с меньшим уровнем - это цикл *3*. Затем меняется на один шаг значение параметра следующего уровня (цикла *2*) и снова полностью выполняется внутренний цикл и т. д. до тех пор, пока параметры циклов всех уровней не примут все требуемые значения. При этом, если в сложном цикле с глубиной вложенности k число повторений циклов на каждом уровне равно N_0, N_1, \dots, N_k соответственно, то общее количество повторений тела самого внутреннего цикла равно:

$$N = N_0 \cdot N_1 \cdot \dots \cdot N_k.$$

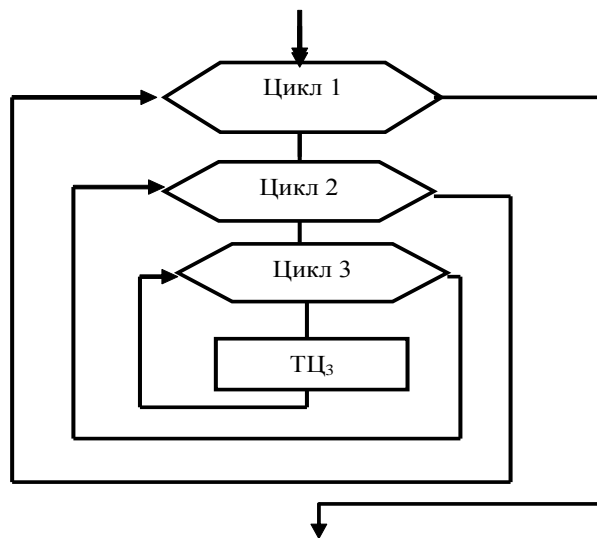


Рисунок 3 – Схема алгоритма с вложенными циклами

На рис. 3 изображен цикл с параметром. Но все изложенное относится и к тем случаям, когда для организации циклов используются другие виды циклических структур: цикл с предусловием или цикл с постусловием.

Рассмотрим конкретную задачу, требующую для своего решения организации вложенных циклов. Такой задачей является задача табулирования функции нескольких переменных.

Пример 3. Построить алгоритм и написать программу вычисления значений функции $z = \cos x + y$, где $x = x_n (h_x) x_k$ и $y = y_n (h_y) y_k$. Аргументы функции x, y – действительные числа.

Для определения значений функции z для всех различных пар (x, y) необходимо процесс вычислений организовать следующим образом. Вначале при фиксированном значении одного из аргументов, например при $x = x_0$, вычислить значения z для всех заданных y : $y_n, y_n + h_y, \dots, y_n$. Затем, изменив значение x на $x + h_x$, вновь перейти к полному циклу изменения переменной y . Данные действия повторить для всех заданных x : $x_n, x_n + h_x, \dots, x_n$. При реализации данного алгоритма требуется структура вложенных циклов: внешнего цикла – для изменения значений переменной x и внутреннего цикла – для изменения значений переменной y . Причем в данной задаче внешний и внутренний циклы можно поменять местами, при этом изменится только очередность изменения аргументов при вычислении функции. В качестве внешнего и внутреннего циклов можно использовать циклы с параметром, циклы с предусловием или с постусловием.

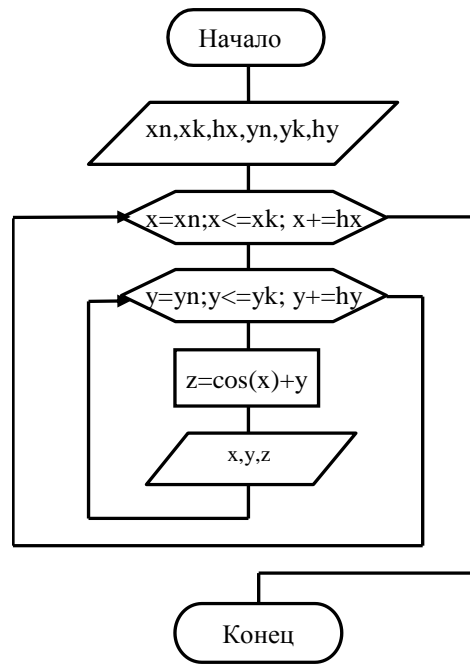


Рисунок 4 – Схема алгоритма к примеру 3

Алгоритм решения поставленной задачи, выполненный с применением цикла с параметром, представлен на рис. 4. Программа, соответствующая этому алгоритму, представлена в листинге 3.

Листинг 3

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double xn, xk, hx, yn, yk, hy, z;
            Console.Write("Enter xn ");
            xn = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter xk ");
            xk = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter hx ");
            hx = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter yn ");
            yn = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter yk ");
            yk = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter hy ");
            hy = Convert.ToDouble(Console.ReadLine());
            for (double x = xn; x <= xk; x += hx)           // Внешний цикл
            {
                for (double y = yn; y <= yk; y += hy)       // Внутренний цикл
                {
                    z = Math.Cos(x) + y;
                    Console.WriteLine("x=" + x + "    y=" + y + "    z=" + z);
                }
            }
        }
    }
}

```

Пример 4. Вычислить с погрешностью ε значения функции $y = \cos(x)$, используя разложение $\cos x$ в ряд, для значений $x = x_n$ (h_x) x_k .

Вычислить значение функции $\cos x$ можно путем разложения его в следующий ряд:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} + \dots$$

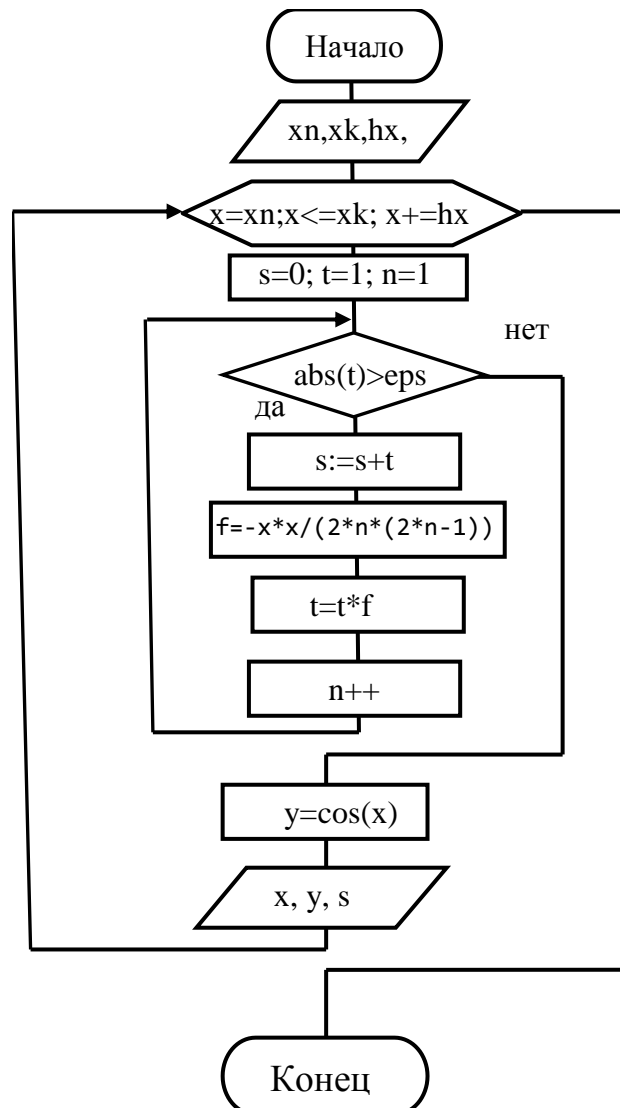


Рисунок 5 – Схема алгоритма к примеру 4

Задача вычисления $y = \cos x$ для фиксированного значения x была рассмотрена в примере 1. В данном случае сумму ряда S необходимо вычислять для каждого значения x из диапазона $[x_n, x_k]$. Следовательно, необходимо использовать структуру вложенных циклов. Как показано на схеме алгоритма (рис. 5), внешний цикл – цикл для изменения значений переменной x (цикл с предусловием). Внутренний цикл – цикл для

вычисления суммы ряда при фиксированном значении x с погрешностью вычислений ε (также цикл с предусловием).

Все вышесказанное реализовано в программе, соответствующей схеме данного алгоритма.

В данном случае сумму ряда S необходимо вычислять для каждого значения x из диапазона $[x_n, x_k]$. Следовательно, необходимо использовать структуру вложенных циклов. Как показано на схеме алгоритма, внешний цикл – цикл для изменения значений переменной x (цикл с предусловием). Внутренний цикл – цикл для вычисления суммы ряда при фиксированном значении x с погрешностью вычислений ε (также цикл с предусловием).

Правильность работы программы оценивается путем сравнения значения s с его вычисляемым по формуле $y=\cos(x)$ значением.

Соответствующая программа представлена в листинге 4.

Листинг 4

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double xn, xk, hx, eps, s, t, y, f;
            Console.Write("Enter xn ");
            xn = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter xk ");
            xk = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter hx ");
            hx = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter eps ");
            eps = Convert.ToDouble(Console.ReadLine());
            double x = xn;
            while (x <= xk)
            {
                //Внешний цикл
                s = 0;
                t = 1;
                int n = 1;
                while (Math.Abs(t) > eps)
                {
                    //Внутренний цикл
                    s = s + t;
                    f = -x * x / (2 * n * (2 * n - 1));
                    t = t * f;
                    n++;
                }
                //Конец внутреннего цикла
                y = Math.Cos(x);
                Console.WriteLine("x=" + x + "    y=" + y + "    s="+s);
                x = x + hx;
            }
            //Конец внешнего цикла
        }
    }
}
```

Выше были рассмотрены основные управляющие конструкции языка Паскаль, позволяющие осуществлять программную реализацию различных ветвлений и циклов. Любой алгоритм (программа) представляет собой

некоторую комбинацию рассмотренных стандартных структур: линейной, разветвляющейся, циклической.

Пример 5. Составить программу нахождения наибольшего общего делителя (НОД) двух целых чисел M и N .

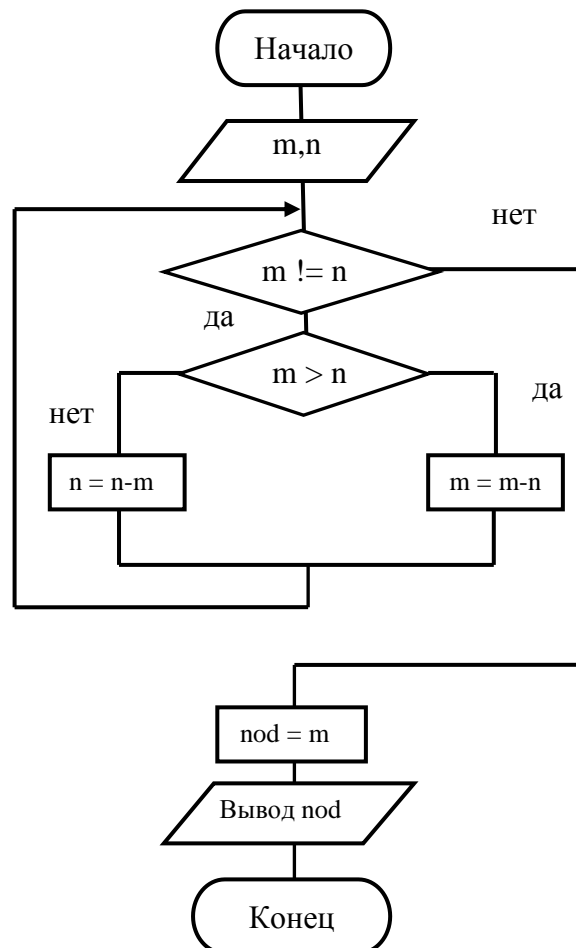


Рисунок 6 – Схема алгоритма к примеру 5

Программа представлена в листинге 5.

Листинг 5

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int m, n, nod;
            Console.Write("Enter m ");
            m = Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter n ");
            n = Convert.ToInt32(Console.ReadLine());
            while (m != n)
            {
                if (m > n) m = m - n;
            }
            nod = m;
        }
    }
}
```

```

        else n= n-m;
    }
    nod= m;
    Console.WriteLine("nod=" + nod);
}
}
}

```

Программа представляет собой сочетание линейной, циклической и разветвляющейся структур. Телом цикла с предусловием является разветвление.

Пример 6.

Составить программу табулирования сложной функции

$$y = f(x) = \begin{cases} (a \sin x + b \cos x)[x]!, & \text{если } x \leq k; \\ \frac{\ln x}{a \sin x + b \cos x}, & \text{если } x > k, \end{cases}$$

действительной переменной $x = x_n (h_x) x_k$. Для вычисления $\ln x$ необходимо воспользоваться формулой:

$$\ln x = \sum_{n=1}^{\infty} \frac{(x-1)^n}{nx^n}, x > \frac{1}{2}.$$

Вычисления произвести с погрешностью ε .

Алгоритм решения поставленной задачи представлен ниже (рис. 7).

Так как вычисление $\ln x$ должно производиться при $x > 1/2$, то при вводе данных необходимо предупредить пользователя о том, чтобы он обратил внимание на значение левой границы интервала изменения x , что и осуществляется в программе.

На первом этапе построим укрупненную схему с выделением операций ввода исходных данных и внешнего цикла по x . В программе этот цикл реализуем с помощью оператора цикла с параметром.

На втором этапе более детально раскроем тело цикла по x без расшифровки вычислений функций факториала и логарифма. Тело цикла представляет собой разветвляющуюся структуру. На последнем этапе раскроем внутренние циклы для вычисления функции $\ln x$ и $[x]!$.

Для вычисления $[x]!$ используем структуру цикла с параметром, так как $[x]! = 1 \cdot 2 \cdot \dots \cdot [x]$.

Вычисление $\ln x$ осуществим накоплением суммы членов заданного ряда с требуемой точностью. Везде, где возможно, постараемся использовать рекуррентные формулы.

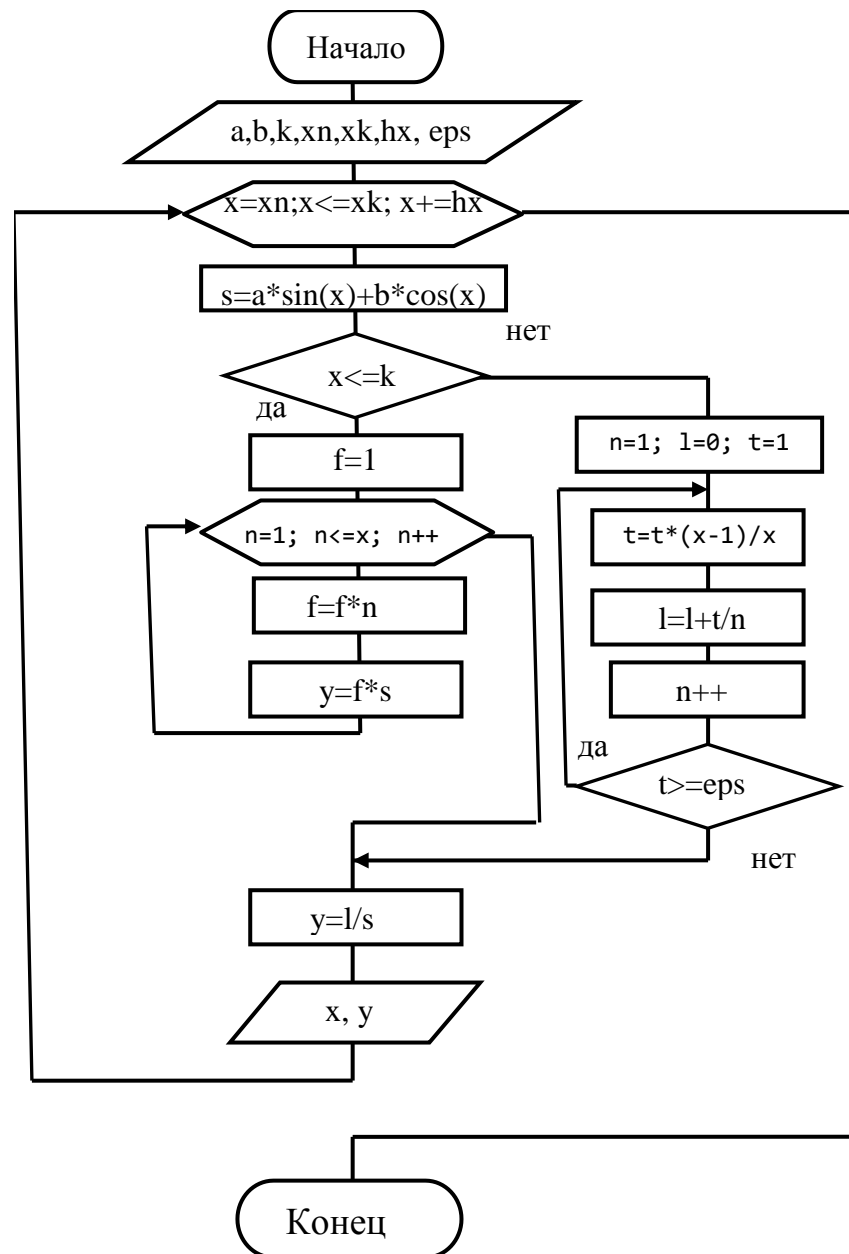


Рисунок 7 – Схема алгоритма к примеру 6

Программа, соответствующая этому алгоритму, представлена в листинге 6.

Листинг 6

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double xn, xk, hx, eps, a, b, k, l, t, s, y = 0;
            int f;
        }
    }
}

```

```

Console.Write("Enter a ");
a = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter b ");
b = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter k ");
k = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter eps ");
eps = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter xn >1/2 ");
xn = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter xk ");
xk = Convert.ToDouble(Console.ReadLine());
Console.Write("Enter hx ");
hx = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("Исходные данные");
Console.WriteLine("xn=" + xn + " xk=" + xk + " hx=" + hx);
Console.WriteLine("a=" + a + " b=" + b + " k=" + k + " eps=" + eps);

for (double x = xn; x <= xk; x += hx)
{
    // Начало внешнего цикла
    s = a * Math.Sin(x) + b * Math.Cos(x);
    if (x <= k)
    {
        // Вычисление факториала
        f = 1;
        for (int n = 1; n <= x; n++)
        {
            f = f * n;
            y = f * s;
        }
    }
    else
    {
        //Вычисление логарифма
        int n = 1; l = 0; t = 1;
        do
        {
            t = t * (x - 1) / x;
            l = l + t / n;
            n++;
        }
        while (t >= eps);
        y = l / s;
    }
    Console.WriteLine("x=" + x + " y=" + y);
}
//Конец внешнего цикла
}
}
}

```

Пример 7. Написать программу, запрашивающую ряд целых чисел, среди которых определяется первое отрицательное число, а затем проверяется: является ли абсолютная величина его простым числом.

Программа решения данной задачи представлена в листинге 7.

Листинг 7

```
using System;
```

```

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int n,x;
            bool b,t;
            Console.WriteLine("Введите количество чисел n ");
            n = Convert.ToInt32(Console.ReadLine());
            b=false;
            t=false;
            for (int i = 1; i <= n; i++)
            {
                Console.WriteLine("Введите число x ");
                x = Convert.ToInt32(Console.ReadLine());
                if (x >= 0) continue;
                t = true;
                for (int j = 2; j <= Math.Abs(x) - 1; j++)
                {
                    if (Math.Abs(x) % j == 0)
                    {
                        b = true;
                        Console.WriteLine("Число не простое");
                        break;
                    }
                }
                break;
            }
            if ((b == false) && (t == true))
                Console.WriteLine("Число простое");
            if (t == false)
                Console.WriteLine("Отрицат. чисел нет");
        }
    }
}

```

Для решения поставленной задачи использованы вложенные циклы. Внешний цикл обеспечивает ввод заданного количества чисел, внутренний цикл используется для проверки абсолютной величины первого отрицательного числа: является ли она простым числом?

В программе в качестве вспомогательных переменных используются две переменные логического типа: *t*- принимает значение *true*, если введено отрицательное число, в противном случае оно сохранит значение *false*, присвоенное ей в начале работы программы.

Переменная *b* примет значение *true* только тогда, когда найдется такое значение *j*, которое поделит $|x|$ без остатка. Поскольку такое значение *j* - делитель $|x|$, то делается вывод о том что, $|x|$ -непростое число. Если *b* сохранило значение *false*, то, следовательно, $|x|$ не имеет других делителей, кроме 1 и $|x|$, а значит оно - простое число.

Для сокращения числа выполняемых операций используются стандартные операторы *continue* и *break*. Оператор *continue* прерывает выполнение очередной итерации внешнего цикла, если введено неотрицательное число. Оператор *break* используется здесь дважды. Один

раз он прерывает поиск других делителей числа, если один уже найден, осуществляя выход из внутреннего цикла. Сразу же после выхода из внутреннего цикла повторное использование оператора **break** реализует выход из внешнего цикла, так как по условию задачи нас интересует только первое отрицательное число.

***Пример 8.** Дано целое число N . Написать программу для получения в порядке убывания всех делителей данного числа. В программе обеспечить пользовательский интерфейс, позволяющий вводить исходные данные и просматривать их, а также неоднократно производить вычисления, не выходя из программы. Для выхода из программы предусмотреть специальную команду.*

Ниже приведены схема алгоритма (рис. 8) и программа для решения данной задачи (лист. 8).

В данной программе для реализации поставленной задачи используется цикл с постусловием. С его помощью организуется возможность решения задачи с изменением исходных данных без выхода из программы и повторного ее запуска. Через пользовательское меню возможно осуществить повторный ввод и просмотр данных, получить результаты решения задачи.

Работа пользовательского интерфейса программы обеспечивается с помощью оператора множественного выбора. В данной программе значение селектора **b** вводится с клавиатуры пользователем. Далее это значение переменной **b** используется в операторе множественного выбора. Таким образом, пользователь, задавая это значение, определяет, какие операции будет выполнять компьютер. Так, если **b=1**, то программа выводит на экран предложение ввести исходные данные и считывает их. Если **b=2**, то на экран выводятся последние значения исходных данных, введенные пользователем. Если **b=3**, то осуществляется собственно решение задачи, и на экран выводится результат. И если **b=4**, то осуществляется выход из программы.

Листинг 8

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int n=0,b;
            do
            {
                Console.WriteLine("- Нажмите 1 и Enter для ввода числа n ");
                Console.WriteLine("- Нажмите 2 и Enter для просмотра исходных данных");
                Console.WriteLine("- Нажмите 3 и Enter для выполнения программы");
                Console.WriteLine("- Нажмите 4 и Enter для выхода из программы");
                b = Convert.ToInt32(Console.ReadLine());
                switch (b)
                {
                    case 1: Console.WriteLine("Введите число n ");
                        n = Convert.ToInt32(Console.ReadLine()); break;
                }
            }
        }
    }
}
```

```

        case 2: Console.WriteLine("n= " +n); break;

        case 3: for (int i = n; i>= 1; i--)
        {
            if (n % i ==0) Console.WriteLine(i);
        }
        break;
    }
    while (b<4);
}
}
}

```

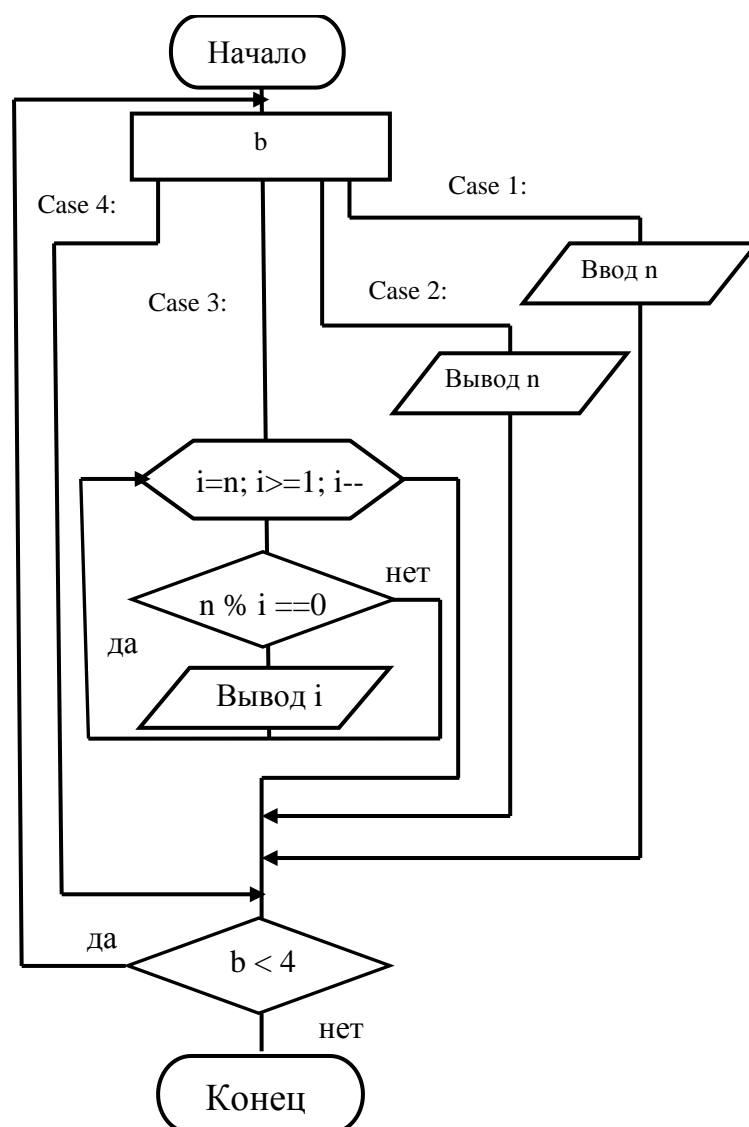


Рисунок 8 – Схема алгоритма к примеру 8

3. Задание для самостоятельного решения

Используя индивидуальные задания к лабораторной работе № 4 и подготовленные в процессе ее выполнения три программы решения задачи с различными операторами циклов, построить алгоритм, написать и отладить соответствующую ему программу с двухуровневым меню, организованном с помощью операторов выбора (см. пример 8).

Внешнее меню обеспечивает режимы:

- ввод исходных данных;
- просмотр результатов;
- вход во внутреннее меню;
- выход из программы.

Внутреннее меню обеспечивает режимы:

- расчет и вывод результатов работы программы с применением оператора цикла с параметром;
- расчет и вывод результатов работы программы с применением оператора цикла с предусловием;
- расчет и вывод результатов работы программы с применением оператора цикла с постусловием;
- выход во внешнее меню.

4 Содержание отчета

- 4.1. Титульный лист.
- 4.2. Краткое теоретическое описание.
- 4.3. Задание на лабораторную работу, включающее математическую формулировку задачи.
- 4.4. Результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений.

Лабораторная работа № 6

ПРОГРАММИРОВАНИЕ ЗАДАЧ ОБРАБОТКИ ОДНОМЕРНЫХ МАССИВОВ

1 Цель и порядок работы

Цель работы - изучить операторы и алгоритмы, используемые при организации вычислительных процессов обработки одномерных массивов данных, получить практические навыки в составлении подобных программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- разработать алгоритм решения задачи;
- написать программу, ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

До настоящего момента в программах были использованы простые переменные. Для хранения значения каждой такой переменной, имеющей свое имя, выделяется своя область памяти. По сути, имя имеет эта выделенная область памяти, которая хранит текущее значение переменной. Если переменных много, программа, предназначенная для их обработки, получается длинной, и в ней не удастся в полной мере использовать компьютерные возможности. Одним из случаев, когда ее удастся сделать лаконичной и эффективной, является ситуация, требующая обработки однотипных величин, которые к тому же должны обрабатываться по одним и тем же правилам.

Поскольку при решении очень многих задач требуется именно такая обработка, для нее в любом процедурном языке существует понятие *массива* — ограниченной совокупности однотипных величин. Введение понятия массива позволяет широко использовать циклические процессы, получая компактные коды программ.

Массив - это последовательность элементов одинакового типа, обращение к которым происходит с использованием общего для всех имени. Каждый элемент массива имеет свой порядковый номер (*индекс*) в последовательности: известно какая величина является в ней первой, какая второй и т.д., и в этом плане массив является упорядоченной последовательностью.

Создание массива начинается с выделения памяти под его элементы.

Элементами массива могут быть величины как значимых, так и ссылочных типов (в том числе массивы).

Массив значимых типов хранит значения, массив ссылочных типов — ссылки на элементы. Всем элементам при создании массива присваиваются значения по умолчанию: нули для значимых типов и *null* — для ссылочных.

Вот, например, как выглядят операторы создания массива из 5 целых чисел и массива из 5 строк:

```
int[ ] a = new int[5];  
string[ ] z = new string[5];
```

В первом операторе описан массив *a*, состоящий из пяти элементов значимого типа *int[]*. Операция *new* выделяет память под 5 целых элементов, и они заполняются нулями (рис. 1).

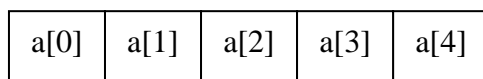


Рисунок 1- Массив из элементов значимого типа

Во втором операторе описан массив *z* ссылочного типа *string[]*. Операция *new* выделяет память под 5 ссылок на строки, и эти ссылки заполняются значением *null*. Память под сами строки, составляющие массив, не выделяется — это будет необходимо сделать перед заполнением массива (рис. 2).

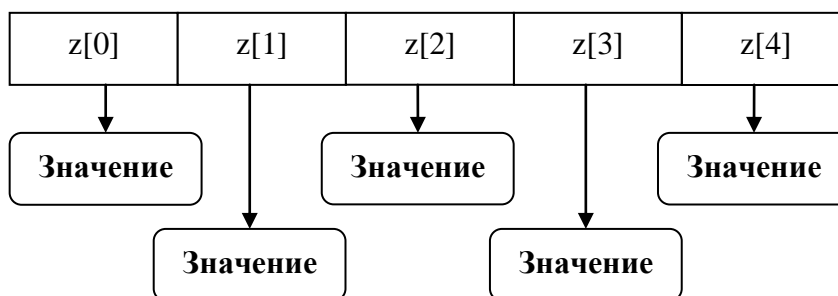


Рисунок 2 - Массив из элементов ссылочного типа

Количество элементов в массиве (*размерность*) задается при выделении памяти и не может быть изменено впоследствии. Размерность может задаваться не только константой, но и выражением. Результат вычисления этого выражения должен быть неотрицательным, а его тип должен иметь неявное преобразование к *int*, *uint*, *long* или *ulong*.

Пример размерности массива, заданной выражением:

```
short m = ... ;  
string[ ] s = new string[m + 1];
```


Элементы массива нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности (например, в описанном выше массиве *a* элементы имеют индексы от 0 до 4). Для обращения к элементу массива конструкция переменная с индексом, где после имени массива указывается порядковый номер элемента в квадратных скобках, например:

`a[4]` `z[i]`

Элементы массива могут участвовать во всех операциях, которые допустимы для простых переменных того же типа. При работе с массивом программисту необходимо внимательно следить за тем, чтобы его индекс не принял значение, лежащее за границами его описания, т.е. не разрешать программе работать с чужой областью памяти.

В С# для предотвращения подобной ситуации автоматически выполняется контроль выхода индекса за его объявленные границы и, если индекс принимает недопустимое значение, генерируется исключение `IndexOutOfRangeException`.

Массивы одного типа можно присваивать друг другу. При этом происходит присваивание ссылок, а не элементов, как и для любого другого объекта ссылочного типа, например:

```
int [ ] a = new int[10];  
int[ ] b = a;                      // b и a указывают на один и тот же массив
```

В С# массивы могут быть одномерными или многомерными. Многомерные массивы, в свою очередь, могут быть прямоугольными и ступенчатыми (невыровненными).

Одномерные массивы

Одномерный массив - это такой массив, для идентификации которого требуется указать значение только одного индекса.

Для объявления одномерного массива используется следующая форма записи:

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

Формат инициализации одномерного массива имеет следующий вид:

```
тип[] имя_массива = [val1, val2, ..., valN];
```

Здесь начальные значения, присваиваемые элементам массива, задаются с помощью последовательности *val1-valN*. Для получения доступа к элементу одномерного массива требуется указать индекс нужного элемента:

```
имя_массива[индекс] = val;
```

Существует несколько вариантов описания массива:

тип[] имя;

тип[] имя = new тип [размерность];

тип[] имя = { список_инициализаторов };

тип[] имя = new тип [] { список_инициализаторов };

тип[] имя = new тип [размерность] { список_инициализаторов };

Примеры описаний (один пример для каждого варианта описания):

```
int[ ] a; // 1 элементов нет
int[ ] b = new int[4]; // 2 элементы равны 0
int[ ] c = { 61, 2, 5, -9 }; // 3 new подразумевается
int[ ] d = new int[ ] { 61, 2, 5, -9 }; // 4 размерность
вычисляется
int[ ] e = new int[4] { 61, 2, 5, -9 }; // 5 избыточное описание
```

Здесь описано пять массивов. Отличие первого оператора от остальных состоит в том, что в нем, фактически, описана только ссылка на массив, а память под элементы массива не выделена. Если список инициализации не задан, размерность может быть не только константой, но и выражением типа, приводимого к целому,

В каждом из остальных массивов по четыре элемента целого типа. Как видно из операторов 3-5, массив при описании можно инициализировать. Если при этом не задана размерность (оператор 4), количество элементов вычисляется по количеству инициализирующих значений. Для полей объектов и локальных переменных можно опускать операцию new, она будет выполнена по умолчанию (оператор 3). Если присутствует и размерность, и список инициализаторов, размерность должна быть константой (оператор 5).

Если количество инициализирующих значений не совпадает с размерностью, возникает ошибка компиляции.

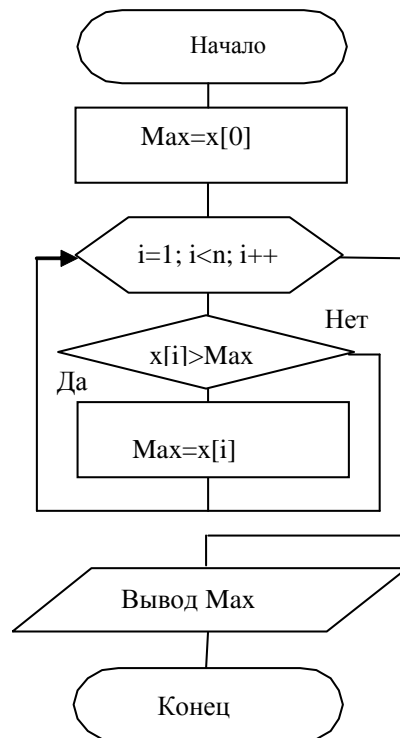
Рассмотрим несколько часто встречающихся на практике задач обработки одномерных массивов.

Некоторые алгоритмы обработки массивов

Пример 1. Поиск наибольшего элемента в одномерном массиве.

Исходными данными в поставленной задаче являются массив x и его размерность n . Решение этой задачи осуществляется с помощью цикла по параметру i , изменяющегося с шагом $+1$ от 1 до $n-1$. Перед входом в цикл переменной ***max*** присваивается значение первого элемента массива $x[0]$.

С каждой итерацией цикла происходит переход к новому элементу массива и сравнение его значения со значением, хранящемся в переменной ***max***. Если значение текущего элемента массива оказывается больше значения ***max***, то переменной ***max*** присваивается значение элемента массива с текущим порядковым номером. После выхода из цикла выводится найденное значение наибольшего элемента массива.



Листинг 1 – Поиск максимального элемента в одномерном массиве

```

using System;
namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 6;
            int[] a = new int[n] { 3, 12, 5, -9, 8, -4 };

            Console.WriteLine("Исходный массив:");
            for (int i = 0; i < n; ++i)
                Console.WriteLine("\t" + a[i]);
            Console.WriteLine();
            int max = a[0];
            for (int i = 0; i < n; ++i)
                if (a[i] > max)
                    max = a[i];
            Console.WriteLine("Максимальный элемент = " + max);
            Console.Read();
        }
    }
}

```

Результаты расчета:

```

Исходный массив :
    3
   12
    5
   -9
    8
   -4

Максимальный элемент = 12

```

Обратите внимание на то, что для вывода элементов исходного массива требуется организовать цикл.

Пример 2. Найти сумму и количество отрицательных элементов, а также максимальный элемент массива.

В данной задаче предыдущий алгоритм входит как составная часть общего решения (листинг 2). Дополнительный независимый цикл с параметром:

```
long sum=0;           // сумма отрицательных элементов
int num = 0;          // количество отрицательных элементов
for ( int i = 0; i < n; ++i )
    if ( a[i] < 0 )
    {
        sum += a[i];
        ++num;
    }
```

введен в программу для выявления отрицательных элементов массива и подсчета их суммы и количества. Для переменных, в которых накапливаются сумма и количество отрицательных элементов, перед входом в цикл определены типы и начальные значения.

Листинг 2 - Работа с одномерным массивом к примеру 2

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            const int n = 6;
            int[ ] a= new int[n] { 3,12,5,-9,8,-4};

            Console.WriteLine( "Исходный массив:" );
            for ( int i =0; i < n; ++i)
                Console.WriteLine( "\t" + a[i] );
            Console.WriteLine( );

            long sum=0;           // сумма отрицательных элементов
            int num = 0;          // количество отрицательных элементов
            for ( int i = 0; i < n; ++i )
                if ( a[i] < 0 )
                {
                    sum += a[i];
                    ++num;
                }

            Console.WriteLine( "Сумма отрицательных = " + sum );
            Console.WriteLine( "Кол-во отрицательных = " + num );

            int max = a[0];       // максимальный элемент
            for ( int i = 1; i < n; ++i )
                if ( a[i] > max ) max = a[i];
            Console.WriteLine( "Максимальный элемент = " + max );
            Console.Read();
        }
    }
}
```

Результаты расчета:

```
Исходный массив :
  3
 12
 5
-9
 8
-4

Сумма отрицательных = -13
Кол-во отрицательных = 2
Максимальный элемент = 12
```

***Пример 3.** Сортировка элементов одномерного массива по убыванию методом выбора.*

Введем дополнительные определения:

X - исходный массив, который нужно упорядочить;

Max- максимальный элемент массива;

Im- индекс максимального элемента;

ch- переменная для обмена элементов;

exch- признак нахождения Max в неотсортированной части массива.

Суть представленного ниже алгоритма заключается в следующем. Упорядочение начинается с поиска элемента, который должен находиться на первом месте в отсортированном массиве. Таким элементом является максимальный элемент всего массива.

После нахождения такого элемента происходит его установка на первое место в массиве, а первый элемент устанавливается на место найденного наибольшего элемента. Далее считая, что первое место в массиве занято необходимым элементом, оно исключается из рассмотрения, и описанные выше действия применяются к его оставшейся части: поиск наибольшего элемента в неупорядоченной части и его обмен с первым элементом этой части.


```

        {
            ch = x[j];
            x[j] = max;
            x[im] = ch;
        }
    }
    Console.WriteLine();
    Console.WriteLine("Результирующий массив:");
    for (int i = 0; i <= n - 1; ++i)
        Console.Write("    " + x[i]);
    Console.Read();
}
}
}

```

Результаты расчета:

```

Исходный массив:
3    12    5    -9    8    -4
Результирующий массив:
12    8    5    3    -4    -9_

```

Пример 4. Сортировка элементов одномерного массива по убыванию методом вставки.

Обозначения:

X - исходный массив, который нужно упорядочить;

n - количество элементов массива;

i,j,k - переменные цикла;

ch - переменная обмена.

Схема алгоритма представлена ниже. Суть метода состоит в следующем. Предполагается, что первый элемент массива уже стоит на своём месте. Чтобы упорядочить второй элемент массива, нужно найти для него такое место в упорядоченной части массива, чтобы упорядоченность сохранялась. Затем требуется вставить этот элемент на найденное место. Чтобы упорядочить все элементы массива, эти действия нужно выполнить $n-1$ раз, т.е. для всех элементов, кроме первого.

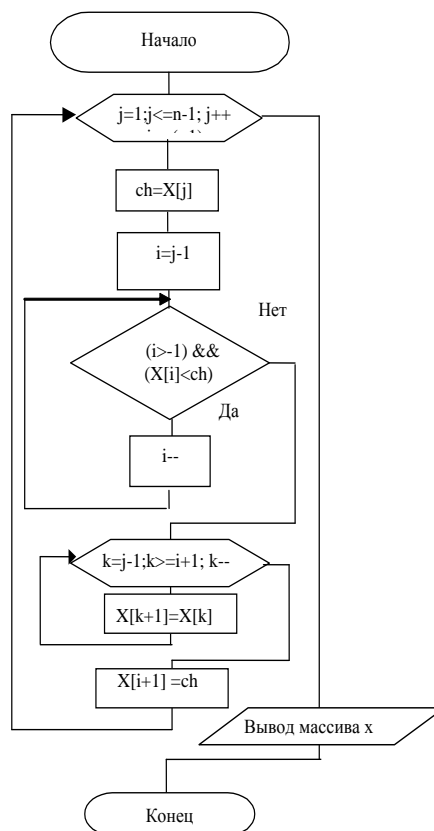
Схема алгоритма решения данной задачи по структуре представляет собой внешний цикл для выполнения указанных выше действий, который содержит еще два вложенных в него следующих друг за другом цикла.

Первый из них организован как цикл с предусловием для поиска в упорядоченной части массива такого элемента массива, чье значение будет больше величины устанавливаемого элемента. Для достижения этой цели в указанном цикле осуществляется продвижение от конца упорядоченной части к ее началу путем уменьшения номера элемента i с каждой итерацией на единицу. Выход из цикла осуществляется тогда, когда такой элемент будет найден, либо будет достигнуто начало упорядоченной части.

Следующий за ним цикл производит перемещение элементов упорядоченной части, начиная с ее конца и заканчивая $i+1$ элементом, на один элемент вправо для освобождения места упорядочиваемому j элементу.

После выхода из цикла (цикл с параметром) производится запись j элемента, хранимого в переменной ch , на освобожденное место.

Схема алгоритма:



Листинг 4 – Сортировка одномерного массива методом «вставки»

Листинг

```

using System;

namespace ConsoleApplication7
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 6;
            int ch;
            int[] x = new int[n] { 3, 12, 5, -9, 8, -4 };
            Console.WriteLine("Исходный массив:");
            for (int i = 0; i <= n - 1; ++i)
                Console.Write("  " + x[i]);
            for (int j = 1; j <= n - 1; j++)
            {
                ch = x[j];
                int i = j - 1;
                while ((i > -1) && (x[i] < ch))
                    i--;
                for (int k = j - 1; k >= i+1; --k)
                    x[k + 1] = x[k];
            }
        }
    }
}

```



```

        x[i+1] = ch;
    }
    Console.WriteLine();
    Console.WriteLine("Результирующий массив:");
    for (int i = 0; i <= n - 1; ++i)
        Console.Write("  " + x[i]);
    Console.Read();
}
}
}

```

Результаты расчета:

```

Исходный массив:
 3  12  5 -9  8 -4
Результирующий массив:
12  8  5  3 -4 -9

```

Пример 5. Сортировка элементов одномерного массива по возрастанию методом «пузырька».

Обозначения:

X - исходный массив, который нужно упорядочить;

n - количество элементов массива;

i,j - переменные цикла;

ch - переменная обмена;

exch - признак существования инверсии.

Суть метода состоит в последовательном просмотре соседних элементов массива. Если они нарушают требуемый порядок следования, то они меняются местами - осуществляется инверсия. Просмотр нужно выполнить n-1 раз.

Очень существенна проверка на наличие инверсии (нарушения порядка). Если в процессе просмотра массива была осуществлена хотя бы одна инверсия (exch=TRUE), то просмотр массива повторяется. В противном случае (exch=FALSE) в массиве уже все элементы упорядочены и целесообразно прервать работу цикла оператором break.

Листинг5– Сортировка одномерного массива методом «пузырька»

```

using System;

namespace ConsoleApplication7
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 6;
            int ch;
            int[] x = new int[n] { 3, 12, 5, -9, 8, -4 };
            Console.WriteLine("Исходный массив:");
            for (int i = 0; i <= n - 1; ++i)
                Console.Write("  " + x[i]);

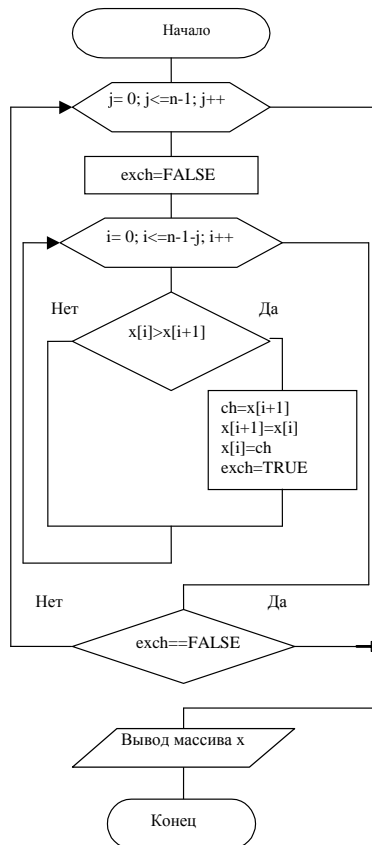
```

```

for (int j = 0; j <= n - 1; j++)
{
    bool exch = false;
    for (int i = 0; i <= n - 2 - j; ++i)
        if (x[i] > x[i + 1])
        {
            ch = x[i + 1];
            x[i + 1] = x[i];
            x[i] = ch;
            exch = true;
        }
    if (exch == false) break;
}
Console.WriteLine();
Console.WriteLine("Результирующий массив:");
for (int i = 0; i <= n - 1; ++i)
    Console.Write("    " + x[i]);
Console.Read();
}
}
}

```

Схема алгоритма:



Результаты расчета:

```

Исходный массив :
3  12  5  -9  8  -4
Результирующий массив :
-9  -4  3  5  8  12

```

Класс System.Array

Для облегчения программирования задач обработки массивов данных в C# все массивы имеют общий базовый класс Array, определенный в пространстве имен System. Приведем несколько полезных методов этого класса (табл. 1).

Таблица 1 - Основные элементы класса *Array*

Элемент	Вид	Описание
Length	Свойство	Количество элементов массива (по всем размерностям)
Rank	Свойство	Количество размерностей массива
BinarySearch	Статический метод	Двоичный поиск в отсортированном массиве
Clear	Статический метод	Присваивание элементам массива значений по умолчанию
Copy	Статический метод	Копирование заданного диапазона элементов одного массива в другой массив
CopyTo	Метод	Копирование всех элементов текущего одномерного массива в другой одномерный массив
GetValue	Метод	Получение значения элемента массива
IndexOf	Статический метод	Поиск первого вхождения элемента в одномерный массив
LastIndexOf	Статический метод	Поиск последнего вхождения элемента в одномерный массив
Reverse	Статический метод	Изменение порядка следования элементов на обратный
SetValue	Метод	Установка значения элемента массива
Sort	Статический метод	Упорядочивание элементов одномерного массива

Свойство Length позволяет реализовывать алгоритмы, которые будут работать с массивами различной длины. Использование этого свойства вместо явного задания размерности исключает возможность выхода индекса за границы массива. В листинге продемонстрировано применение двух элементов класса Array при работе с одномерным массивом.

Листинг 6 – Использование методов класса Array

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            int[] x = { 21, 2, 14, -6, 38, -7, 30};
            Console.WriteLine();
            Console.WriteLine("Исходный массив:");
            for (int i = 0; i < x.Length; ++i)
                Console.Write("  " + x[i]);
            Array.Sort(x);
            Console.WriteLine();
        }
    }
}
```

```

        Console.WriteLine("Упорядоченный массив:");
        for (int i = 0; i < x.Length; ++i)
            Console.Write("    " + x[i]);
        Array.Reverse(x);
        Console.WriteLine();
        Console.WriteLine("Обратный порядок в массиве:");
        for (int i = 0; i < x.Length; ++i)
            Console.Write("    " + x[i]);
        Console.Read();
    }
}
}

```

Методы **Sort**, **Reverse** являются статическими, поэтому к ним обращаются через имя класса **Array**, а не экземпляра, и передают в них имя массива.

Как видно по результатам работы программы метод **Sort** осуществляет сортировку массива по возрастанию. Применив к массиву, отсортированному таким образом, метод **Reverse**, получим массив, отсортированный по убыванию.

Результаты работы:

```

Исходный массив:
 21  2  14  -6  38  -7  30
Упорядоченный массив:
 -7  -6  2  14  21  30  38
Обратный порядок в массиве:
 38  30  21  14  2  -6  -7_

```

Класс Random

Класс **Random**, определенный в пространстве имен **System**, содержит методы, позволяющие при отладке программ генерировать исходные данные, заданные случайным образом. Особенно это удобно при отладке программ обработки массивов данных, поскольку появляется возможность освободить программиста от утомительной работы ввода последовательности данных. Разумеется, подобный вариант может быть использован лишь тогда, когда ход решения задачи позволяет применять данные, полученные путем такой генерации.

В **C#** для получения псевдослучайной последовательности чисел существует два варианта создания экземпляра класса **Random**: конструктор без параметров и конструктор с параметром типа **int**. Рассмотрим их.

Конструктор без параметров:

```
Random a = new Random( );
```

создает уникальную последовательность, так как использует начальное значение генератора, вычисленное на основе текущего времени.

Конструктор с параметром типа **int**:

```
Random b = new Random(5);
```

задает начальное значение генератора, что обеспечивает возможность получения одинаковых последовательностей чисел.

Для получения очередного значения серии пользуются методами, перечисленными в табл. 2.

Таблица 2 - Основные методы класса *System.Random*

Название	Описание
Next()	Возвращает целое положительное число во всем положительном диапазоне типа int
Next(макс)	Возвращает целое положительное число в диапазоне [0, макс]
Next(мин, макс)	Возвращает целое положительное число в диапазоне [мин, макс]
NextDouble()	Возвращает вещественное положительное число в диапазоне [0. 1)
NextBytes(массив)	Возвращает массив чисел в диапазоне [0, 255]

Первые четыре метода позволяют получить одно число. Для получения последовательности случайных чисел необходима организация цикла. Последний метод табл. 2 генерирует массив чисел.

Посмотрим результаты использования методов данного класса, которые дадут представление об их работе (листинг 7).

Листинг 7– Работа с генератором псевдослучайных чисел

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        Random l = new Random( );
        Random d = new Random( 1 );
        const int n = 5;
        Console.WriteLine( "Числа в диапазоне [0, 1]" );
        for (int i = 0; i < n; ++i)
            Console.Write( "{0,6:0.##}", l.NextDouble() );
        Console.WriteLine();
        Console.WriteLine("Числа в диапазоне [0, 100]");
        for ( int i = 0; i < n; ++i )
            Console.Write( " " + d.Next( 100 ) );
        Console.WriteLine();
        Console.WriteLine("Числа в диапазоне [-5, 5]");
        for ( int i = 0; i < n; ++i )
            Console.Write( " " + l.Next( -5, 5 ) );
        Console.WriteLine();
        Console.WriteLine("Массив чисел в диапазоне[0, 255]");
        byte[ ] mas = new byte[n];
        l.NextBytes(mas);
        for (int i = 0; i < n; ++i)
            Console.Write(" " + mas[i] );
    }
}
```

Результат работы программы:

```
Числа в диапазоне [0, 1]
0,65 0,47 0,56 0,14 0,98
Числа в диапазоне [0, 100]
24 11 46 77 65
Числа в диапазоне [-5, 5]
-5 0 -3 -2 4
Массив чисел в диапазоне[0, 255]
159 171 89 207 155
```

3. Варианты заданий для самостоятельной работы

1. Программа. Дано 100 целых чисел. Отсортировать их по возрастанию и распечатать их в обратном порядке по 6 чисел в строке.
2. Дано 100 чисел. Напечатать сначала все отрицательные из них, а затем все остальные.
3. Дано число А. Определить первый отрицательный член последовательности $x(0), x(1), x(2), \dots, x(n)$. Массив заполнить случайным образом.
4. Даны координаты N точек на плоскости $x(0), y(0), \dots, x(n), y(n)$ ($N=20$). Найти номера двух точек, расстояние между которыми наибольшее (считать, что такая пара точек единственная).
5. Вычислить величину: $\frac{x_1 * y_1 + x_3 * y_3 + \dots + x_{29} * y_{29}}{x_0 * y_0 + x_2 * y_2 + \dots + x_{30} * y_{30}}$. Массивы x и y заполнить случайным образом.
6. По заданным вещественным числам $a(0), a(1), \dots, a(20)$ и T вычислить по формуле Горнера значения многочлена $a_{20}x^{20} + a_{19}x^{19} + \dots + a_0$ в точке $x=T$.
7. Преобразовать массив X, содержащий n вещественных чисел, по следующему правилу: элементы массива расположить в обратном порядке и вывести все элементы с четными индексами.
8. Переменной T присвоить значение TRUE, если элементы массива X, содержащего n вещественных чисел, упорядочены строго по возрастанию, а значение FALSE иначе.
9. Преобразовать массив X, содержащий n вещественных чисел, по следующему правилу:
$$x(0)=x(0), x(n)=x(n), x(k)=(x(k-1)+x(k)+x(k+1))/3 \text{ при } k=2,3,\dots,n-1.$$
10. Преобразовать массив X, содержащий n вещественных чисел, по следующему правилу: элементы массива циклически сдвинуть на одну позицию влево.
11. Преобразовать массив X, содержащий n вещественных чисел, по следующему правилу: элементы массива циклически сдвинуть на две позиции влево.
12. Преобразовать массив X, содержащий n вещественных чисел, по следующему правилу (воспользоваться массивом Y как вспомогательным): все отрицательные элементы массива X перенести в его начало, а все остальные - в конец, сохраняя исходное взаимное расположение как среди отрицательных, так и среди остальных элементов.

13. Преобразовать массив X , содержащий n вещественных чисел, по следующему правилу: элементы массива X циклически сдвинуть на k позиций влево.
14. Переменной T присвоить значение $TRUE$, если в массиве X , содержащем n вещественных чисел, нет нулевых элементов и при этом положительные элементы чередуются с отрицательными, и значение $FALSE$ иначе.
15. Переменной K присвоить либо номер первого вхождения Y в массив X , либо число $N+1$, если Y не входит в X .
16. Вычислить $y = x(0) + x(0) * x(1) + x(0) * x(1) * x(2) + \dots + x(0) * x(1) * \dots * x(m)$, где m – либо номер первого отрицательного элемента массива X , либо число N , если в массиве X нет отрицательных элементов.
17. Переменной T присвоить значение $TRUE$ или $FALSE$ в зависимости от того, есть или нет среди элементов массива X , содержащего n целых чисел, хотя бы одно число Фибоначчи.
18. Переменной T присвоить значение $TRUE$ или $FALSE$ в зависимости от того, есть или нет среди элементов массива X , содержащего n целых чисел, не менее двух степеней двойки.
19. Даны действительные числа $a[0], a[2], \dots, a[10]$. Получить (вывести) последовательность переносом первого элемента в конец массива: $a[1], a[2], \dots, a[10], a[0]$, затем $a[2], \dots, a[10], a[0], a[1]$, и т.д. Всего десять последовательностей.
20. Дана последовательность из 100 различных целых чисел. Найти сумму чисел этой последовательности, расположенных между максимальным и минимальным числами (в сумму включить и оба этих числа).
21. Получить последовательность n чисел Фибоначчи (целое число $n > 1$).
22. Дано целое n и вещественные числа $a[0], \dots, a[n]$. Вычислить $a[0] - a[1] + a[2] - \dots + (-1)^{n-1} a[n]$.
23. Дано целое n и вещественные числа a, b ($a < b$). Получить $r[0], r[1], \dots, r[n]$, где $r[i] = a + ih$, где $h = (b - a) / n$.
24. Даны вещественные числа $a[0], a[2], \dots, a[10]$. Получить $b[0], \dots, b[10]$, где $b[i] = (a[i]^2 - a[1]) * b[i-1]$ при $i = 1, 2, 3, \dots, 10$.
25. Получить последовательность $b[0], \dots, b[10]$, где при $i = 0, 1, 2, \dots, 10$ значение $b[i]$ равно $1 + 1/2 + \dots + 1/i$.
26. Даны семь значений емкостей конденсаторов $c[0], \dots, c[6]$. Определить емкости систем конденсаторов, которые получаются при последовательном и параллельном соединении исходных конденсаторов.

27. При $n=15$ и заданных значениях вещественных чисел $a, h, b, d[0], \dots, d[n]$, вычислить $d[0]+d[1](b-a)+d[2](b-a)(b-a-h)+\dots+d[n](b-a)(b-a-h)\dots(b-a-(n-1)h)$.
28. Дано целое число n и целые числа $k[0], k[1], \dots, k[12]$. В последовательности $k[0], \dots, k[12]$ заменить каждый из членов остатком от деления его квадрата на n .
29. Дано целое число n , вещественные числа $a[0], \dots, a[n]$ ($n>3$). Получить $b[i]$, $i=0, 1, \dots, n-2$, используя зависимость $b[i]=a[i+1]+a[i+2]$.
30. Дана последовательность вещественных чисел $a[0], \dots, a[10]$. Подсчитать количество элементов последовательности до второго отрицательного элемента.
31. Даны вещественные числа $x, a[0], \dots, a[10]$. Найти значение целого числа k , при котором $a[k-1]<x<a[k]$.
32. Получить последовательность $b[0], \dots, b[n]$ (при $n=7$), где значение $b[i]$ равно $i!$ ($n!=1*2*\dots*n$; $0!=1$).
33. Даны вещественные числа $a[0], a[1], \dots, a[5]$.
Вычислить: $a[0]/1! + a[1]/2! + \dots + a[5]/5!$.
34. Даны вещественные числа $a[0], \dots, a[n]$ ($n=15$). Получить $\max(a[0], \dots, a[n])$ и $\max(-a[0], a[2], -a[3], \dots, -a[n])$.
35. Даны вещественные числа $b[0], \dots, b[n]$ ($n=10$). Получить $\min(b[0], \dots, b[n])$ и $\min(|b[0]|, \dots, |b[n]|)$.
36. Даны вещественные числа $a[0], a[1], \dots, a[n]$ ($n=12$). Получить $(\min(a[0], \dots, a[n]))^2 - \min(a^2[0], \dots, a^2[n])$.
37. Даны вещественные числа $a[0], \dots, a[n]$ ($n=10$). Исключить из последовательности все числа, равные нулю.
38. Даны целые числа $k[0], \dots, k[n]$ ($n=12$). Найти наименьшую сумму из трех рядом стоящих чисел.
39. Даны целые числа $k[0], \dots, k[n]$ ($n=15$). Найти наибольшее из них, которое кратно трем.
40. Даны вещественные числа $a[0], \dots, a[n]$ ($n=16$). Получить значения i ($1 < i < n-1$), для которых $a[i-1]<a[i]>a[i+1]$.
41. Даны целые числа $k[0], \dots, k[10]$. Вывести те значения, которые по значению больше всего приближаются к целым числам m и n .
42. Пусть $a[0]=\cos(x)$; $a[1]=-\sin(x)$; $a[k]=2a[k-1]-a[k-2]$, $k=2, 3, \dots$. Найти сумму квадратов тех чисел $a[0], \dots, a[20]$, которые не превосходят двух. X вводится с клавиатуры.

43. Даны вещественные числа $a[0], \dots, a[20]$. В этой последовательности определить число соседств двух положительных чисел и двух чисел разного знака.
44. Даны целые числа $k[0], k[1], \dots, k[m]$ ($m=20$). Имеются ли в последовательности два идущих подряд нулевых числа или три подряд одинаковых числа.
45. Даны вещественные числа $a[1], \dots, a[3n]$ ($n=5$). Вычислить сумму чисел из $a[n+1], a[n+2], \dots, a[3n]$, которые превосходят по величине все числа $a[0], \dots, a[n]$.
46. Пусть $a[0]=0.01$; $a[k]=\sin(k+a[k-1])$, $k=1, \dots, 30$. Определить сколько значений последовательности $a[i]$, $i=0, \dots, 30$ с номерами 0, 2, 4, 8, 16, ... имеют значение меньше, чем 0.25.
47. Даны вещественные числа $y[0], \dots, y[n]$ ($n=10$). Найти $\max(|z[0]|, \dots, |z[n]|)$, где $z[i]=y[i]$ при $|y[i]| < 2$ и 0.5 в противном случае.
48. Даны вещественные числа $a[0], \dots, a[n]$ ($n=15$). Найти $\max(a[0], a[0]*a[1], \dots, a[0]*a[1] \dots a[n])$.
49. Даны вещественные числа $x[0], \dots, x[n]$ ($n=12$). Найти количество квадратов нечетных чисел.
50. Даны вещественные числа $a[0], \dots, a[n]$ ($n=10$). Найти количество полных квадратов среди этих чисел.
51. Даны вещественные числа $x[0], \dots, x[n]$ ($n=15$). Выяснить, является ли данная последовательность упорядоченной по убыванию.
52. Даны вещественные числа $a[0], \dots, a[n]$ ($n=10$). Найти длину наименьшего отрезка числовой оси, содержащего все значения $a[0], \dots, a[n]$.
53. Даны вещественные числа $x, a[0], \dots, a[n]$ ($n=12$). Выяснить, во-первых, верно ли, что $a[0] < x < a[12]$, и, во-вторых, верно ли, что $t[1] < x < t[2]$, где $t[1]$ – наименьшее, а $t[2]$ – наибольшее среди $a[0], \dots, a[n]$.
54. Дана последовательность вещественных чисел $a[0], \dots, a[n]$ ($n=15$) и вещественное число x . Вставить x в то место последовательности, где удовлетворяется условие $a[i-1] < x < a[i]$.
55. Дана последовательность целых чисел $k[0], \dots, k[n]$ ($n=12$). Найти номер второго четного члена этой последовательности, если его нет, то выдать соответствующее сообщение.
56. Дана последовательность целых чисел $k[0], \dots, k[n]$ ($n=15$). Найти номер последнего нечетного члена этой последовательности.
57. Даны вещественные числа $x[0], \dots, x[n]$ ($n=10$). Получить $(1+r)/(1+s)$, где r – сумма всех чисел последовательности, которые не превосходят 1, а s – сумма чисел, больших 1.

58. Даны целые числа $k[0], \dots, k[n]$ ($n=15$). Найти максимальное из чисел этой последовательности, предшествующее первому отрицательному.
59. Даны целые числа $a[0], \dots, a[n]$ ($n=12$). Определить все числа, которые равны сумме всех предыдущих.
60. Дан массив целых чисел из 30 элементов. Найти наибольший элемент массива, кратный 5.
61. Даны целые числа $a[0], \dots, a[n]$ ($n=15$). Определить $\min(a[0], a[1], \dots, a[i])$ до первого отрицательного числа последовательности $a[0], \dots, a[n]$.
62. Дан массив целых чисел из 20 элементов. Найти наибольший нечетный элемент массива.
63. Дана последовательность целых чисел из 20 элементов. Найти наименьшее произведение из рядом двух стоящих элементов.
64. Даны целые числа $k[0], \dots, k[n]$ ($n=15$). Найти минимальное из чисел этой последовательности, предшествующее первому нулевому элементу.

4. Содержание отчета

- 4.1. Титульный лист.
- 4.2. Краткое теоретическое описание.
- 4.3. Задание на лабораторную работу, включающее математическую формулировку задачи.
- 4.4. Результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений.

Лабораторная работа № 7

ПРОГРАММИРОВАНИЕ ЗАДАЧ ОБРАБОТКИ ДВУМЕРНЫХ МАССИВОВ

1 Цель и порядок работы

Цель работы - изучить операторы, используемые при организации вычислительных процессов обработки двумерных массивов данных, получить практические навыки в составлении подобных программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- разработать алгоритм решения задачи;
- написать программу, ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

2.1 Прямоугольные массивы

Прямоугольный массив имеет более одного измерения. Многомерным называется такой массив, который характеризуется двумя или более измерениями, а доступ к отдельному элементу осуществляется посредством двух или более индексов.

Чаще всего в программах используются простейшие многомерные массивы - двумерные массивы. В двумерном массиве позиция любого элемента определяется двумя индексами. Если представить двумерный массив в виде таблицы данных, то один индекс означает строку, а второй - столбец. Чтобы объявить двумерный массив целочисленных значений размером $A \times B$, достаточно записать следующее:

```
int[,] имя_массива = new int[A,B];
```

Чтобы получить доступ к элементу двумерного массива, необходимо указать оба индекса, разделив их запятой. Например, чтобы присвоить число Z элементу двумерного массива, позиция которого определяется координатами x и y , можно использовать следующую инструкцию:

```
имя_массива[x,y] = Z;
```

Многомерный массив можно инициализировать, заключив список инициализаторов каждой размерности в собственный набор фигурных скобок. Например, вот каков формат инициализации многомерного массива:

```
int[,] имя_массива = {  
    {val, val, val, ..., val}
```

```

    {val, val val, ..., val}
    ...
    {val, val, val, ..., val}
};

```

Здесь элемент **val** – значение инициализации. Каждый внутренний блок означает строку. В каждой строке первое значение будет сохранено в первой позиции массива, второе значение - во второй и т.д.

Варианты описания двумерного массива:

тип[,] имя;

тип[.] имя = new тип [разм_1, разм_2];

тип[.] имя = { список_инициализаторов };

тип[,] имя = new тип [,] { список_инициализаторов };

тип[,] имя = new тип [разм_1, разм_2] { список_инициализаторов };

Примеры описаний (один пример для каждого варианта описания):

```

int[ , ] a;                                // 1  элементов нет
int[ , ] b = new int [2, 3];               // 2  элементы равны 0
int[ , ] c = {{1, 2, 3}, {4, 5, 6}};       // 3  new подразумевается
int[ , ] c = new int[ , ] {{1, 2, 3}, {4, 5, 6}}; // 4  размерность
вычисляется
int[ , ] d = new int[2,3 ] {{1, 2, 3}, {4, 5, 6}}; // 5  избыточное описание

```

Если список инициализации не задан, размерности могут быть не только константами, но и выражениями типа, приводимого к целому. К элементу двумерного массива обращаются, указывая номера строки и столбца, на пересечении которых он расположен, например:

a[1,4] b[i, j] b[j, i]

Необходимо помнить, что компилятор воспринимает как номер строки первый индекс, как бы он ни был обозначен в программе.

Пример 1. Для целочисленной матрицы размером m*n вычислить сумму элементов в каждой строке (рис. 1).

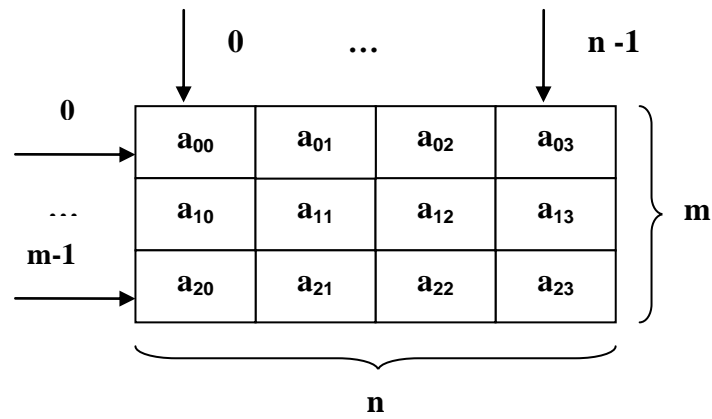


Рисунок 1 - Матрица из m строк и n столбцов

Нахождение суммы элементов каждой строки требует просмотра матрицы по строкам. Схема алгоритма приведена на рис. 2, программа — в листинге 1.

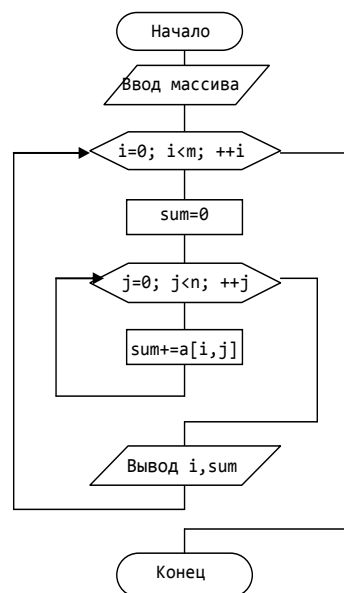


Рисунок 2- Схема алгоритма к примеру 1

Листинг 1- Программа к примеру 1

```

using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            const int m = 3, n = 4;
            int[,] a = new int[m, n] {
                { 2, 2, 8, 9 },
                { 4, 5, 6, 2 },
                { 7, 0, 1, 1 }
            };
        }
    }
}

```

```

        Console.WriteLine("Исходный массив:");
        for ( int i = 0; i < m; ++i)
        {
            for ( int j = 0; j < n; ++j )
                Console.Write( " " + a[i, j]);
            Console.WriteLine();
        }
        for ( int i = 0; i < m; ++i )
        {
            int sum = 0;
            for ( int j = 0; j < n; ++j )
                sum += a[i, j];
            Console.WriteLine("В строке {0} сумма элементов {1}", i, sum );
        }
        Console.Read();
    }
}

```

Результаты работы программы:

```

Исходный массив:
2 2 8 9
4 5 6 2
7 0 1 1
В строке 0  сумма элементов  21
В строке 1  сумма элементов  17
В строке 2  сумма элементов   9

```

Следует обратить внимание на то, что переменная `sum` обнуляется перед циклом просмотра очередной строки матрицы, поскольку для каждой строки его вычисление начинается заново.

2.2 Ступенчатые массивы

В *ступенчатых массивах* количество элементов в разных строках может различаться. В памяти ступенчатый массив хранится иначе, чем прямоугольный: в виде нескольких внутренних массивов, каждый из которых имеет свой размер. Кроме того, выделяется отдельная область памяти для хранения ссылок на каждый из внутренних массивов. Организацию ступенчатого массива иллюстрирует рис. 3.

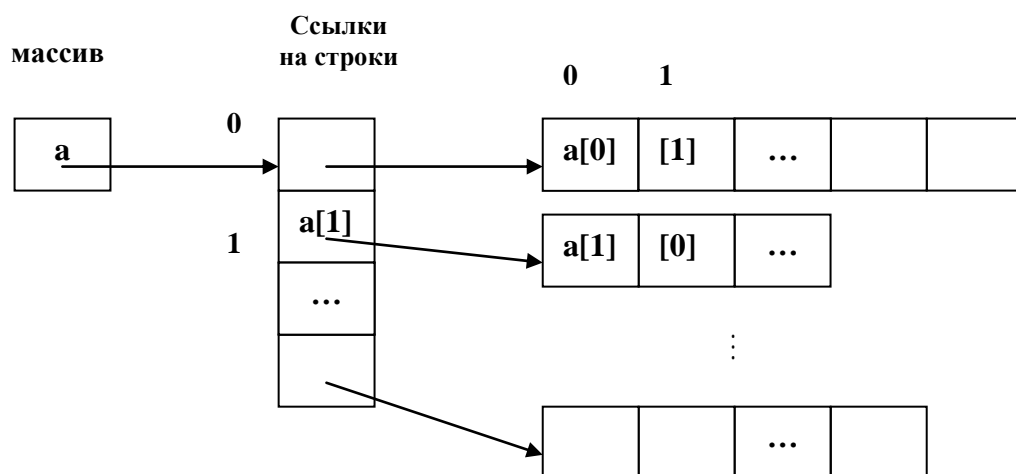


Рисунок 3 - Ступенчатый массив

Описание ступенчатого массива:

тип[][] имя;

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

```
int[ ][ ] a = new int [3][ ]; // выделение памяти под ссылки на три строки
a[0]= new int [5];           // выделение памяти под 0-ю строку (5 элементов)
a[1]=new int [3];            // выделение памяти под 1-ю строку (3 элемента)
a[2] = new int [4];          // выделение памяти под 2-ю строку (4 элемента)
```

Здесь `a[0]`, `a[1]` и `a[2]` — это отдельные массивы, к которым можно обращаться по имени ниже (листинг 3) приведен пример работы со ступенчатым массивом с использованием выделения памяти под каждую строку.

Другой способ выделения памяти:

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

К элементу ступенчатого массива обращаются, указывая каждую размерность в своих квадратных скобках, например:

```
a[1][2]      a[i][j]      a[j][i]
```

В остальном использование ступенчатых массивов не отличается от использования прямоугольных.

Пример 2. Поиск максимального элемента в двумерном массиве.

Решение поставленной задачи осуществляется с помощью цикла по параметру *i*, изменяющегося шагом *+1* от *0* до *m-1* с вложенным циклом по параметру *j*, изменяющемуся шагом *+1* от *0* до *n-1*.

Перед входом в цикл переменной *max* присваивается значение *y[0,0]* элемента массива. С каждым изменением параметра *j* происходит переход к новому элементу той же строки массива и сравнение его значения со значением переменной *max*.

Если значение элемента массива больше, то *max* присваивается значение элемента массива с текущим порядковым номером. Когда при данном значении параметра *i* завершается цикл по параметру *j*, происходит изменение параметра *i* и переход к новой строке двумерного массива.

Листинг 2- Программа к примеру 2

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            const int m = 3, n = 4;
```

```

int[,] y = new int[m, n] {
    { 2, 2, 8, 9 },
    { 4, 5, 6, 2 },
    { 7, 0, 1, 1 }
};

Console.WriteLine("Исходный массив:");
for (int i = 0; i < m; ++i)
{
    for (int j = 0; j < n; ++j)
        Console.Write(" " + y[i, j]);
    Console.WriteLine();
}
int max=y[0,0];
for (int i = 0; i < m; ++i)
{
    for (int j = 0; j < n; ++j)
        if (y[i, j] > max) max = y[i, j];
}
    Console.WriteLine("Наибольший элемент в матрице " + max);
    Console.Read();
}
}
}

```

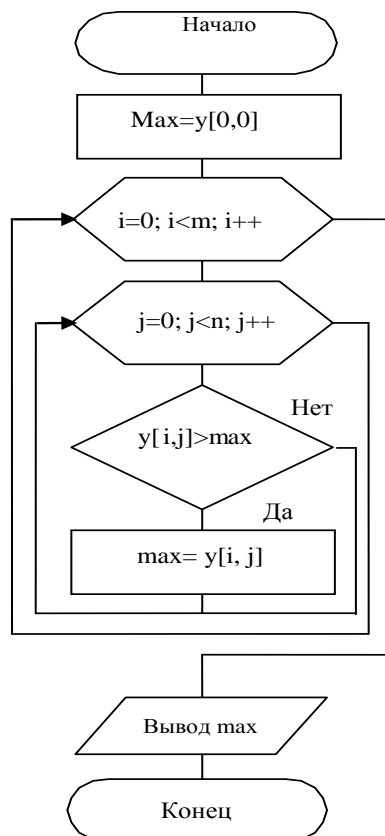


Рисунок 3 - Схема алгоритма к примеру 2

Результаты работы программы:

```
Исходный массив :
2 2 8 9
4 5 6 2
7 0 1 1
Наибольший элемент в матрице 9
```

В листинге 3 продемонстрировано применение элементов класса *Array* при работе со ступенчатым массивом.

Листинг 3 - Использование методов класса *Array* для обработки ступенчатого массива

```
using System;
namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
            int[][] a = new int[3][];
            a[0] = new int[5] { 4, 5, 8, 3, 6 };
            a[1] = new int[3] { 7, 9, 1 };
            a[2] = new int[4] { 6, 5, 3, 1 };

            Console.WriteLine("Исходный массив:");
            for (int i = 0; i < a.Length; ++i)
            {
                for (int j = 1; j < a[i].Length; ++j)
                    Console.Write(" " + a[i][j]);
                Console.WriteLine();
            }
            Console.WriteLine(Array.IndexOf(a[0], 8));
        }
    }
}
```

Необходимо обратить внимание на то, как внутри цикла по строкам определяется длина каждого одномерного массива (длина каждой строки массива).

Результаты работы программы:

```
Исходный массив :
4 5 8 3 6
7 9 1
6 5 3 1
2
```

2.3 Оператор цикла с перебором *foreach*

Цикл перебора *foreach* используется для просмотра всех объектов из некоторой группы данных – коллекции. В языке C# определен ряд типов коллекций, например, коллекцией являются массивы.

Синтаксис записи цикла *foreach*:

foreach(тип имя_переменной *in* имя коллекции) выражение;

где *тип* и *имя_переменной* задают *тип* и *имя итерационной переменной*, которая в процессе итераций цикла ***foreach*** получает значения элементов коллекции, заданной своим именем.

Если в программе в качестве коллекции используется массив данных, то необходимо помнить, что тип итерационной переменной должен совпадать или быть совместимым с базовым типом массива. Необходимо также учитывать, что итерационная переменная как элемент массива доступна только для чтения, и ей нельзя присвоить новое значение, а значит с помощью этой конструкции нельзя изменить и значение элемента массива.

Рассмотрим программу поиска максимального элемента в одномерном массиве.

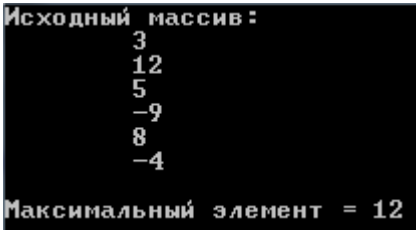
Листинг 4 - Использование оператора цикла *foreach* для поиска наибольшего элемента одномерного массива

```
using System;
namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 6;
            int[] a = new int[n] { 3, 12, 5, -9, 8, -4 };

            Console.WriteLine("Исходный массив:");
            for (int i = 0; i < n; ++i)
                Console.WriteLine("\t" + a[i]);
            Console.WriteLine();
            int max = a[0];
            foreach (int x in a)
            {
                if (x > max)
                    max = x;
            }
            Console.WriteLine("Максимальный элемент = " + max);
            Console.Read();
        }
    }
}
```

Как видно из текста программы цикл ***foreach*** последовательно просматривает все элементы массива ***a*** и присваивает итерационной переменной ***x*** значение очередного элемента массива. Значение ***x*** в дальнейшем используется в операторе ***if*** для коррекции переменной ***max***.

Результаты работы программы:



```
Исходный массив :
    3
   12
    5
   -9
    8
   -4
Максимальный элемент = 12
```

Следует отметить тот факт, что не всегда требуется просматривать всю коллекцию до конца, например, в задаче, где требуется найти первый отрицательный элемент. После того как отрицательный элемент найден, можно прервать работу цикла с помощью оператора ***break***.

При работе с многомерными массивами цикл ***foreach*** возвращает элементы в порядке следования строк.

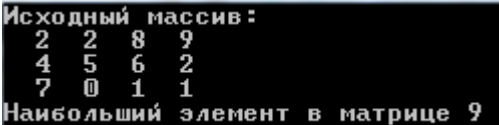
Листинг 5 - Использование оператора цикла *foreach* для поиска наибольшего элемента двумерного массива

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            const int m = 3, n = 4;
            int[,] y = new int[m, n] {
                { 2, 2, 8, 9 },
                { 4, 5, 6, 2 },
                { 7, 0, 1, 1 }
            };

            Console.WriteLine("Исходный массив:");
            for (int i = 0; i < m; ++i)
            {
                for (int j = 0; j < n; ++j)
                    Console.Write(" " + y[i, j]);
                Console.WriteLine();
            }
            int max = y[0, 0];
            foreach (int x in y){
                if (x > max) max = x;
            }
            Console.WriteLine("Наибольший элемент в матрице " + max);
            Console.Read();
        }
    }
}
```

Как видно из текста программы использование одного оператора ***foreach*** позволило заменить два вложенных оператора ***for*** в поиске наибольшего элемента двумерного массива.

Результаты работы программы:

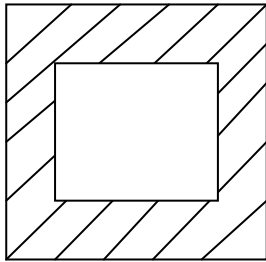


```
Исходный массив :
2 2 8 9
4 5 6 2
7 0 1 1
Наибольший элемент в матрице 9
```

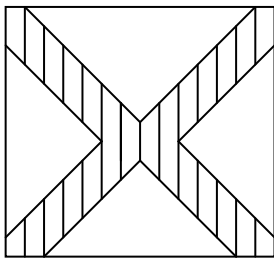
3. Варианты заданий для самостоятельной работы

1. Дана (построчно) вещественная матрица размером 7x4. Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент (один из них) оказался в верхнем левом углу.

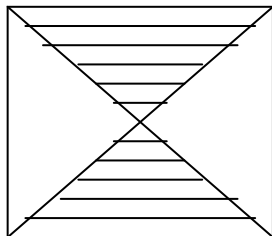
2. Найти S – сумму элементов квадратной матрицы A из заштрихованной области.



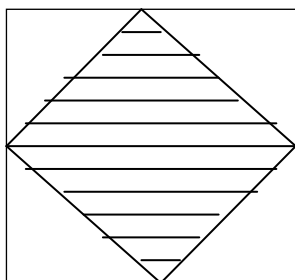
3. Найти S – сумму элементов квадратной матрицы A из заштрихованной области массива A .



4. Дана вещественная матрица размером 7×7 , все элементы которой различны. Найти скалярное произведение строки, в которой находится наибольший элемент матрицы, и столбца с наименьшим элементом.
5. Найти S – сумму элементов квадратной матрицы A из заштрихованной области.



6. По двумерному массиву массиву A получить одномерный массив B , присвоив его k -му элементу значение TRUE, если выполнено указанное ниже условие, и значение FALSE – иначе: все элементы k -го столбца массива A – нулевые.
7. Найти S – сумму элементов квадратной матрицы A из заштрихованной области.



8. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 9 \end{pmatrix}$$

9. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 1 & 2 & 3 & \dots & 10 \\ 11 & 12 & 13 & \dots & 20 \\ 21 & 22 & 23 & \dots & 30 \\ \dots & \dots & \dots & \dots & \dots \\ 91 & 92 & 93 & \dots & 100 \end{pmatrix}$$

10. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 1 & 2 & 3 & \dots & 10 \\ 0 & 1 & 2 & \dots & 9 \\ 0 & 0 & 1 & \dots & 8 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

11. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 1 & 2 & 3 & \dots & 10 \\ 2 & 3 & 4 & \dots & 0 \\ 3 & 4 & 5 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 10 & 0 & 0 & \dots & 0 \end{pmatrix}$$

12. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом::

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

13. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

14. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 1 & 2 & 3 & \dots & 10 \\ 2 & 3 & 4 & \dots & 1 \\ 3 & 4 & 5 & \dots & 2 \\ \dots & \dots & \dots & \dots & \dots \\ 10 & 9 & 8 & \dots & 9 \end{pmatrix}$$

15. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

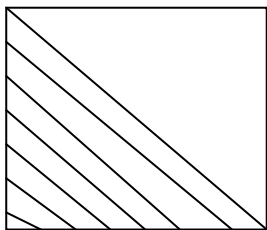
16. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

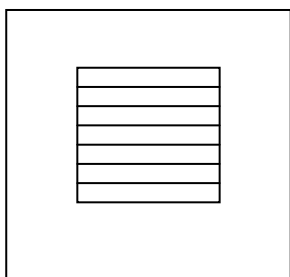
17. Заполнить целочисленный массив A, не вводя значения его элементов с клавиатуры, следующим образом:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{pmatrix}$$

18. Найти S – сумму элементов из заштрихованной области двумерного массива A .



19. Найти S – сумму элементов из заштрихованной области двумерного массива A .



20. По двумерному массиву A получить одномерный массив B , присвоив его k -му элементу значение TRUE, если выполнено указанное ниже условие, и значение FALSE – иначе: элементы k -й строки массива A упорядочены по убыванию. VAR
21. Заполнить целочисленный массив A , не вводя значения его элементов с клавиатуры, следующим образом:

1	12	13	24	25	36
2	11	14	23	26	35
3	10	15	22	27	34
4	9	16	21	28	33
5	8	17	20	29	32
6	7	18	19	30	31

22. Заполнить целочисленный массив A , не вводя значения его элементов с клавиатуры, следующим образом:

1	3	4	10	11	21
2	5	9	12	20	22
6	8	13	19	23	30
7	14	18	24	29	31
15	17	25	28	32	35
16	26	27	33	34	36

23. По двумерному массиву A получить одномерный массив B , присвоив его k -му элементу значение TRUE, если выполнено указанное ниже условие, и значение FALSE – иначе: k -я строка массива A симметрична.

24. Дана вещественная матрица размером 5×6 . Упорядочить ее строки по неубыванию их первых элементов.
25. Дана вещественная матрица размером 5×6 . Упорядочить ее строки по неубыванию суммы их элементов.
26. Дана вещественная матрица размером 5×6 . Упорядочить ее строки по неубыванию их наибольших элементов.
27. Элемент матрицы назовем седловой точкой, если он является наименьшим в своей строке и одновременно наибольшим в своем столбце или, наоборот, является наибольшим в своей строке и наименьшим в своем столбце. Для заданной целой матрицы размером 10×15 напечатать индексы всех ее седловых точек.
28. Определить, является ли заданная целая квадратная матрица 10-го порядка симметричной (относительно главной диагонали).
29. Определить, является ли заданная целая квадратная матрица 10-го порядка ортонормированной, т.е. такой, в которой скалярное произведение каждой пары различных строк равно 0, а скалярное произведение каждой строки на себя равно 1.
30. Определить, является ли заданная целая квадратная матрица 9-го порядка магическим квадратом, т.е. такой, в которой суммы элементов во всех строках и столбцах одинаковы.
31. Дана целочисленная матрица размера $n \times n$. В каждой строке выбрать минимальный элемент. И среди этих минимальных значений выбрать максимальное.
32. Дана действительная матрица размера $m \times n$. Определить число отрицательных элементов в каждой строке.
33. Получить целочисленную матрицу размером $m \times n$, в которой элементы в четных строках и столбцах положительные, а остальные – отрицательные.
34. В целочисленной матрице порядка $n \times n$ указать индексы всех элементов, имеющих наибольшее значение в столбцах.
35. Получить вещественную матрицу a размером $n \times n$, если первая и вторая строка задается формулой $a[i, j] = 2j + 3 / (2 + j)$. А каждая следующая строка есть сумма двух предыдущих.
36. В действительной матрице размером 6×9 поменять местами строку, содержащую элемент с наибольшим значением, со строкой, содержащей элемент с наименьшим значением.
37. Дана целочисленная матрица размером $m \times n$. Получить сумму наибольшего значения элементов матрицы и наименьшего в этой же строке.

38. В действительной матрице размером $m \times n$ найти сумму элементов столбца, содержащего наименьший элемент матрицы.
39. Дана вещественная матрица размером $n \times m$. Найти сумму наибольших значений элементов ее строк.
40. Дана вещественная матрица размером $m \times n$. Найти среднее арифметическое наибольшего и наименьшего значений ее элементов.
41. Все максимальные элементы в столбцах вещественной матрицы размером $m \times n$ заменить нулями.
42. Дана целочисленная матрица порядка $m \times n$. Заменить нулями элементы, расположенные выше главной диагонали и квадратами – значения элементов ниже главной диагонали.
43. Дана целочисленная матрица размером $m \times n$ и целые числа k и l . Определить минимум в столбцах матрицы, имеющих номера k и l .
44. Дана целочисленная матрица размером $n \times n$. Определить, на каком месте располагаются одинаковые элементы, симметричные главной диагонали.
45. Дана целочисленная матрица размером $m \times n$. Найти среднее арифметическое каждого из столбцов с четными номерами.
46. Дана вещественная матрица размером $m \times n$. Найти новую матрицу, полученную путем деления всех элементов исходной матрицы на ее максимальный элемент.
47. Дана вещественная матрица размером $m \times n$. Указать все строки, которые не содержат отрицательных элементов.
48. Дана вещественная матрица размером $m \times n$. Все отрицательные числа заменить на -1 , положительные на $+1$, а нулевые оставить без изменения.
49. Дана вещественная матрица размером $m \times n$. Напечатать номер строки, которая содержит первый из отрицательных элементов.
50. Дана целочисленная матрица размером $m \times n$. Найти сумму всех элементов, расположенных левее столбца, содержащего первый отрицательный элемент.
51. Даны целые числа k , l и целочисленная матрица размером $m \times n$. Поменять местами столбцы с номерами k и l .
52. Дана целочисленная матрица размером $m \times n$. Удалить из этой матрицы строку с номером k ($k < m$).
53. Дана вещественная матрица размером $m \times n$. В строках, содержащих отрицательный элемент, определить максимальный элемент.
54. Дана целочисленная матрица размером $m \times n$. В каждой строке сменить знак максимального по модулю элемента на противоположный.

55. Дана целочисленная матрица размером $m \times n$. Последний отрицательный элемент каждого столбца заменить нулем.
56. Дана целочисленная матрица размером $m \times n$. Положительные элементы умножить на первый элемент соответствующей строки, а отрицательные – на последний, то есть положительные элементы первой строки умножаем на первый элемент первой строки, а отрицательные – на последний элемент также первой строки, то же самое и с остальными строками.
57. Дана целочисленная матрица размером $m \times n$. Заменить все элементы строки с номером k и столбца с номером 1 на противоположные по знаку (элемент, стоящий на пересечении, не изменять).
58. Дана целочисленная матрица размером $m \times n$. К элементам столбца k_1 прибавить элементы столбца k_2 .
59. Даны два двумерных массива одинаковой размерности. Создать третий массив той же размерности, каждый элемент которого равен сумме соответствующих элементов первых двух.
60. Даны два двумерных массива A и B одинаковой размерности. Создать массив C , где каждый элемент равен 1 (True), если соответствующие элементы A и B имеют одинаковый знак, иначе элемент равен 0 (False).
61. Дан двумерный массив размерностью 4×6 , заполненный целыми числами с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен количеству элементов соответствующей строки, больших данного числа.
62. Дан двумерный массив размером $n \times m$, заполненный случайными числами. Определить, есть ли в данном массиве столбец, в котором имеются одинаковые элементы.
63. Дан двумерный массив размерностью 5×6 , заполненный целыми числами с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен наибольшему по модулю элементу соответствующего столбца.
64. Дан двумерный массив размером $n \times m$, заполненный случайными числами. Определить, есть ли в данном массиве строка, в которой имеется два максимальных элемента всего массива.
65. Дан двумерный массив размером $n \times m$, заполненный случайными числами. Определить, есть ли в данном массиве столбец, в котором равное количество положительных и отрицательных элементов.
66. Дан двумерный массив размерностью 6×5 , заполненный целыми числами с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен первому четному элементу соответствующего столбца, если такого нет, то равен нулю.

67. Дан двумерный массив размером $n \times m$, заполненный случайными числами. Определить, есть ли в данном массиве строка, содержащая больше положительных элементов, чем отрицательных.
68. Дан двумерный массив размером $n \times m$. Вставить после столбцов с максимальными элементами столбец из нулей.
69. Дан двумерный массив размером $n \times m$. Вставить между средними строками первую строку.
70. Дан двумерный массив размером $n \times m$. Удалить все столбцы, в которых первый элемент больше последнего.

4. Содержание отчета

- 4.1. Титульный лист.
- 4.2. Краткое теоретическое описание.
- 4.3. Задание на лабораторную работу, включающее математическую формулировку задачи.
- 4.4. Результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений.

Лабораторная работа № 8

ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

1 Цель и порядок работы

Цель работы - изучить операторы, используемые при обработке исключительных ситуаций, возникающих во время выполнения вычислительных процессов, получить практические навыки в составлении программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- написать программу, ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

Исключительная ситуация (или исключение) - это ошибка, которая возникает во время выполнения программы. Используя C# - подсистему обработки исключительных ситуаций, с такими ошибками можно справляться. В C# эта подсистема включает в себя усовершенствованные методы, используемые в языках C++ и Java. Преимущество подсистемы обработки исключений состоит в автоматизации создания большей части кода, который ранее необходимо было вводить в программы "вручную". Обработка исключений упрощает "работу над ошибками", позволяя в программах определять блок кода, именуемый обработчиком исключения, который будет автоматически выполняться при возникновении определенной ошибки. В этом случае не обязательно проверять результат выполнения каждой конкретной операции или метода вручную. Если ошибка возникнет, ее должным образом обработает обработчик исключений.

Еще одним преимуществом обработки исключительных ситуаций в C# является определение стандартных исключений для таких распространенных программных ошибок, как деление на нуль или попадание вне диапазона определения индекса. Чтобы отреагировать на возникновение таких ошибок, программа должна отслеживать и обрабатывать эти исключения. Без знания возможностей C#-подсистемы обработки исключений успешное программирование на C# попросту невозможно.

В C# исключения представляются классами. Все классы исключений должны быть выведены из встроенного класса исключений Exception, который является частью пространства имен System. Таким образом, все исключения - подклассы класса Exception.

Программные инструкции, которые нужно проконтролировать на предмет исключений, помещаются в *try*-блок. Если исключение возникает в этом блоке, оно дает знать о себе выбросом определенного рода информации. Это выброшенное исключение может быть перехвачено программным путем с помощью *catch*-блока и обработано соответствующим образом. Системные исключения автоматически генерируются С#-системой динамического управления. Чтобы сгенерировать исключение вручную, используется ключевое слово *throw*. Любой код, который должен быть обязательно выполнен при выходе из *try*-блока, помещается в блок *finally*.

Ядром обработки исключений являются блоки *try* и *catch*. Эти ключевые слова работают "в одной связке"; формат записи *try/catch*-блоков обработки исключений имеет следующий вид:

```
try {  
    // Блок кода, подлежащий проверке на наличие ошибок.  
}  
catch (Exception exOb) {  
    // Обработчик для исключения типа Exception  
}  
catch (Exception2 exOb) {  
    // Обработчик для исключения типа Exception2  
}
```

Здесь **Exception** - это тип сгенерированного исключения. После "выброса" исключение перехватывается соответствующей инструкцией *catch*, которая его обрабатывает. Как видно из формата записи *try/catch*-блоков, с *try*-блоком может быть связана не одна, а несколько *catch*-инструкций. Какая именно из них будет выполнена, определит тип исключения. Другими словами, будет выполнена та *catch*-инструкция, тип исключения которой совпадает с типом сгенерированного исключения (а все остальные будут проигнорированы). После перехвата исключения параметр **exOb** примет его значение.

Задавать параметр **exOb** необязательно. Если обработчику исключения не нужен доступ к объекту исключения (как это часто бывает), в задании параметра **exOb** нет необходимости. Поэтому во многих примерах этой главы параметр **exOb** не задан.

Важно понимать следующее: если исключение не генерируется, то *try*-блок завершается нормально, и все его *catch*-инструкции игнорируются. Выполнение программы продолжается с первой инструкции, которая стоит после последней инструкции *catch*. Таким образом, *catch*-инструкция (из предложенных после *try*-блока) выполняется только в случае, если сгенерировано соответствующее исключение.

В табл. 1 представлены наиболее часто используемые исключения, определенные в пространстве имен System.

Таблица 1

Исключение	Значение
ArrayTypeMismatchException	Тип сохраняемого значения несовместим с типом массива
DivideByZeroException	Попытка деления на нуль
IndexOutOfRangeException	Индекс массива оказался вне диапазона
InvalidCastException	Неверно выполнено динамическое приведение типов
OutOfMemoryException	Недостаточный объем свободной памяти
OverflowException	Имеет место арифметическое переполнение
NullReferenceException	Попытка использовать нулевую ссылку
StackoverflowException	Переполнение стека

Иногда требуется перехватывать все исключения, независимо от их типа. Для этого используется *catch*-инструкция без параметров. В этом случае создается обработчик "глобального перехвата", который используется, чтобы программа гарантированно обработала все исключения.

Пример. Оптимизировать программу, производящую простые арифметические операции над числами (сложение, вычитание, умножение и деление), используя обработку исключительных ситуаций (обработка ввода чисел и операции деления).

Листинг 1

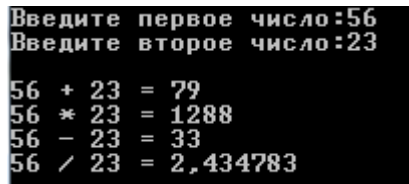
```
using System;
namespace ConsoleApplication
{
    class OurClass
    {
        static void Main(string[] args)
        {
            int num1 = 1, num2 = 2, summarize, multiply, sub;
            double divide = 0;
            Console.Write("Введите первое число:");
            try { num1 = int.Parse(Console.ReadLine()); }
            catch
            {
                Console.WriteLine("Неправильный формат числа!\n" +
                    "В качестве значения первого числа будет 1");
            }
            Console.Write("Введите второе число:");
            try { num2 = int.Parse(Console.ReadLine()); }
            catch
            {
                Console.WriteLine("Неправильный формат числа!\n" +
                    "В качестве значения второго числа будет 2");
            }
            summarize = num1 + num2; multiply = num1 * num2; sub = num1 - num2;
            try { divide = num1 / num2; }
            catch (DivideByZeroException)
            {
                Console.WriteLine("Нельзя делить на нуль!");
            }
            Console.WriteLine(
```

```

        "\n" + num1 + " + " + num2 + " = " + summarize +
        "\n" + num1 + " * " + num2 + " = " + multiply +
        "\n" + num1 + " - " + num2 + " = " + sub +
        "\n" + num1 + " / " + num2 + " = " + divide);
        Console.WriteLine("\nДля выхода из программы нажмите [Enter]:");
        string anykey = Console.ReadLine();
    }
}

```

Результаты работы программы:



```

Введите первое число:56
Введите второе число:23

56 + 23 = 79
56 * 23 = 1288
56 - 23 = 33
56 / 23 = 2,434783

```

3. Варианты заданий для самостоятельной работы

Для своего варианта задания лабораторной работы № 2 оптимизировать программу, включив в нее обработку исключительных ситуаций: ввода данных, операции деления и др.

4. Содержание отчета

- 4.1. Титульный лист.
- 4.2. Краткое теоретическое описание.
- 4.3. Задание на лабораторную работу, включающее математическую формулировку задачи.
- 4.4. Результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений.

Лабораторная работа № 9

ОБРАБОТКА ТЕКСТОВОЙ ИНФОРМАЦИИ

1 Цель и порядок работы

Цель работы - изучить операторы, используемые при организации программ обработки строковых переменных, получить практические навыки в составлении программ.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- разработать алгоритм решения задачи;
- написать программу, ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

С точки зрения практического программирования одним из самых важных его аспектов является обработка текстовой информации, и любой современный язык программирования имеет для решения задач данного типа свои средства. Поскольку любой текст можно рассматривать и как последовательность отдельных символов и как строку, то функциональные возможности наиболее современного языка C# должны быть достаточно полны для выполнения разнообразных преобразований всех видов текстовой информации. Типы данных в C# позволяют работать и с отдельными символами, и с массивами символов, и с различного типа строками. Рассмотрим эти возможности.

2.1 Символьный тип *char*

Символьный тип *char* базируется на стандартном классе *Char* библиотеки *.NET* из пространства имен *System*. Он является встроенным типом языка и предназначен для хранения символов в *Unicode*.

Класс *System.Char* объединяет в себе целый ряд методов (табл. 1), которые позволяют намного облегчить написание программ обработки символьной информации. Набор методов этого класса позволяют распознавать символы различного назначения и выполнять над ними распространенные операции.

Таблица 1 - Основные методы класса *System.Char*

	Метод	Пояснение
1	GetNumericValue	Возвращает числовое значение символа, если он является цифрой, и -1 в противном случае

2	GetUnicodeCategory	Возвращает категорию Unicode-символа
3	IsControl	Возвращает true, если символ является управляющим
4	IsDigit	Возвращает true, если символ является десятичной цифрой
5	IsLetter	Возвращает true, если символ является буквой
6	IsLetterOrDigit	Возвращает true, если символ является буквой или цифрой
7	IsLower	Возвращает true, если символ задан в нижнем регистре
8	IsNumber	Возвращает true, если символ является числом (десятичным или шестнадцатеричным)
9	IsPunctuation	Возвращает true, если символ является знаком препинания
10	IsSeparator	Возвращает true, если символ является разделителем
11	IsUpper	Возвращает true, если символ записан в верхнем регистре
12	IsWhiteSpace	Возвращает true, если символ является пробельным (пробел, перевод строки и возврат каретки)
13	Parse	Преобразует строку в символ (строка должна состоять из одного символа)
14	ToLower	Преобразует символ в нижний регистр
15	ToUpper	Преобразует символ в верхний регистр
16	MaxValue, MinValue	Возвращают символы с максимальным и минимальным кодами (эти символы не имеют видимого представления)

Пример 1. Целью данного примера является демонстрация возможностей некоторых приведенных выше методов класса *System.Char*.

Листинг 1

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            char q;
            do
            {
                Console.WriteLine( "Введите символ: ");
                q = char.Parse (Console.ReadLine());
                if (char.IsLetter(q))           Console.WriteLine("Буква");
                if (char.IsUpper(q))           Console.WriteLine("Верхний рег.");
                if (char.IsLower(q))           Console.WriteLine("Нижний рег.");
                if (char.IsControl(q))          Console.WriteLine("Управляющий");
                if (char.IsNumber(q))           Console.WriteLine("Число");
                if (char.IsPunctuation(q))      Console.WriteLine("Разделитель");
            }
            while (q!='/');
        }
    }
}
```

В программе в цикле с постусловием запрашивается символ и далее с помощью ряда условных операторов определяется: к какой категории введенный символ относится. Выход из цикла организован по символу “/”.

Результаты работы программы:

```
Введите символ:
d
Буква
Нижний рег.
Введите символ:
Z
Буква
Верхний рег.
Введите символ:
3
Число
Введите символ:
/
Разделитель
Для продолжения нажмите любую клавишу . . .
```

2.2 Массив символов

Некоторые алгоритмы обработки символьной информации могут оперировать массивом символов, а, значит, им доступны возможности класса *Array*, приведенные в работе 7. Рассмотрим это на конкретном примере.

Пример 2. Дана строка символов *s* (вводится с клавиатуры). Напечатать в алфавитном порядке: какие строчные русские буквы встречаются в этой строке.

Одним из вариантов решения данной задачи может быть такой. Опишем дополнительную строку *p*, где будет представлен весь русский алфавит строчными буквами, как массив символов. Данный массив, также как и массив символов *s*, получен в программе (лист.2) путем преобразования строки в массив методом *ToCharArray* класса *System.String* (табл. 2). Естественно длина такого массива будет определяться числом 33. Анализ введенного массива *s* в этом случае может быть организован с помощью двух вложенных циклов.

В предложенной программе внешний цикл перебирает посимвольно буквы русского алфавита из массива *p*, и для его построения используется цикл с параметром. Конечное значение параметра этого цикла вычисляется в заголовке цикла с помощью функции *Length(p)*, хотя можно было просто записать его значение явно, равное 33.

Внутренний цикл — это цикл с постусловием. Параметром его является переменная *j*, определяющая индекс анализируемого элемента массива *s* и увеличивающая свое значение в цикле с шагом +1, тем самым, обеспечивая просмотр всех его элементов (от первого до последнего). Если найден элемент массива *s*, совпадающий с рассматриваемой буквой алфавита, определенной во внешнем цикле, то выполняются печать этого

элемента и принудительное завершение внутреннего цикла процедурой **break**. Вот почему дублирование символов в массиве *s* не обнаруживается.

После прерывания выполнения внутреннего цикла внешний цикл начинает работать со следующим элементом массива-алфавита, и для него все повторяется сначала.

Листинг 2

```
using System;
namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
            char[] p = "абвгдежзийклмнопрстуфхцщъыьэюя".ToCharArray();
            Console.WriteLine();
            char[] s = "мама мыла раму".ToCharArray();
            for (int i = 0; i < p.Length; ++i)
            {
                int j = 1;
                do
                {
                    if (p[i] == s[j])
                    {
                        Console.Write(" " + s[j]);
                        break;
                    }
                    j++;
                }
                while (j < s.Length);
            }
            Console.Read();
        }
    }
}
```



2.3 Tun String

Тип **string** предназначен для определения и поддержки символьных строк. Самый простой способ создать объект типа **string** - использовать строковый литерал:

```
string str = "C#-строки - это мощная сила.";
```

Создать строку можно несколькими способами:

```
string s; //инициализация отложена
string t = "программа"; //инициализация строковым литералом
string u = new string('*',10); //конструктор создает строку из 10
```

СИМВОЛОВ

```
char[] a = {'0','0','0'}; //массив для инициализации строки
string v = new string(a); //создание из массива символов
```

Последовательность символов, составляющих строку, изменить нельзя. Это связано с тем, что строки типа *string* относятся к так называемым неизменяемым типам данных. Поэтому обращаться к отдельному элементу строки по индексу (например, *s[i]*) можно только для получения значения, но не для его изменения.

Хотя это кажется серьезным недостатком, на самом деле это не так. Это ограничение позволяет C# эффективно использовать строки. Если понадобится строка, которая должна содержать изменения уже существующей строки, то потребуются создать новую строку. Поскольку неиспользуемые строковые объекты автоматически утилизируются системой сбора мусора.

При описании строковой переменной память под нее отводится по максимуму. При работе же с переменной может использоваться лишь часть этой памяти, реально занятая символами строки в данный момент.

Для строк определены следующие операции:

- присваивание (=);
- проверка на равенство (==); (Строки равны, если имеют одинаковое количество символов и совпадают посимвольно)
- проверка на неравенство (!=);
- обращение по индексу ([]);
- сцепление (конкатенация) строк (+).

***Пример 3.** Дан массив слов. Составить программу, которая вводит исходный массив, затем вводит поочередно различные слова и проверяет, имеются ли такие слова в исходном массиве.*

Сформулированная задача в данной постановке подобна многим другим, осуществляющим обработку элементов одномерного массива. Такие программы были рассмотрены в лабораторной работе 8. Основное отличие данной задачи заключается в том, что в этом случае поиск требуемого элемента должен производиться не в числовом массиве, а в массиве слов.

Программа представлена на листинге 3. Ее исполнительная часть содержит вложенные циклы.

Наличие внешнего цикла объясняется тем, что пользователю предоставляется возможность поиска не одного, а нескольких слов в массиве, содержащем *ks* слов. Чтобы предоставить пользователю неограниченную свободу в поиске, внешний цикл организован с помощью цикла с постусловием, работа которого завершается при введении для поиска слова «*конец*».

Внутренний цикл представляет собой также цикл с постусловием, в котором просматриваются поочередно слова массива, начиная с первого элемента. В том случае, если будет найден элемент массива, совпадающий с введенным во внешнем цикле словом, формируется признак *pr=true* и завершается работа внутреннего цикла. Пользователь переходит к поиску

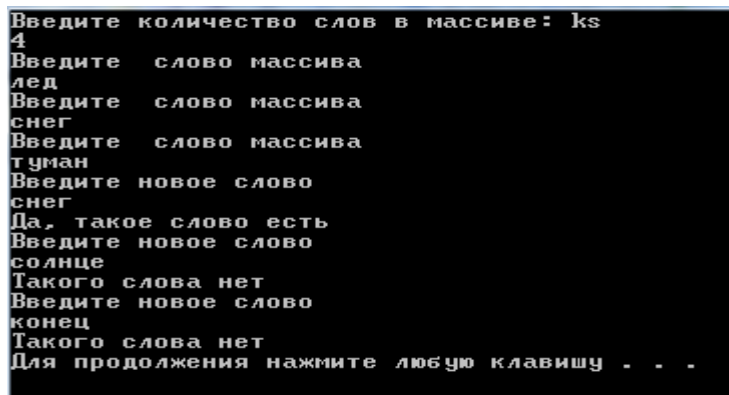
следующего слова. По поиску каждого слова выдается сообщение о том найдено ли оно или нет.

Листинг 3

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int ks,n=15;
            bool pr;
            string s;
            string[] sl=new string[n];
            Console.WriteLine("Введите количество слов в массиве: ks");
            ks = Convert.ToInt32(Console.ReadLine());
            for (int k=1; k<ks; k++)
            {
                Console.WriteLine("Введите слово массива");
                sl[k]=Console.ReadLine();
            }
            do
            {
                Console.WriteLine("Введите новое слово");
                s = Console.ReadLine();
                pr = false;
                int j = 0;
                do
                {
                    if (sl[j] == s) pr = true;
                    j++;
                }
                while ((pr == false) && (j < ks));
                if (pr) Console.WriteLine("Да, такое слово есть");
                else Console.WriteLine("Такого слова нет");
            }
            while (s != "конец");
        }
    }
}
```

Результаты работы программы:



```
Введите количество слов в массиве: ks
4
Введите слово массива
лед
Введите слово массива
снег
Введите слово массива
туман
Введите новое слово
снег
Да, такое слово есть
Введите новое слово
солнце
Такого слова нет
Введите новое слово
конец
Такого слова нет
Для продолжения нажмите любую клавишу . . .
```

Для облегчения работы со строками в языке C# существует специальный класс *System.String*.

Основные методы и свойства класса *System.String*, позволяющие выполнять со строками практически любые действия, представлены в (табл.2).

Таблица 2 - Основные элементы класса *System.String*

	Название	Вид	Описание
1	Compare	Статический метод	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки и подстроки с учетом и без учета регистра и особенностей национального представления дат и т. д.
2	CompareOrdinal	Статический метод	Сравнение двух строк по кодам символов. Разные реализации метода позволяют сравнивать строки и подстроки
3	CompareTo	Метод	Сравнение текущего экземпляра строки с другой строкой
4	Concat	Статический метод	Конкатенация строк. Метод допускает сцепление произвольного числа строк
5	Copy	Статический метод	Создание копии строки
6	Empty	Статическое поле	Пустая строка (только для чтения)
7	Format	Статический метод	Форматирование в соответствии с заданными спецификаторами формата
8	IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny	Методы	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора
9	Insert	Метод	Вставка подстроки в заданную позицию
10	Intern, IsInterned	Статические методы	Возвращает ссылку на строку, если такая уже существует. Если строки нет, Intern добавляет строку во внутренний пул, IsIntern возвращает null
11	Join	Статический метод	Слияние массива строк в единую строку. Между элементами массива вставляются разделители (см. далее)
12	Length	Свойство	Длина строки (количество символов)
13	PadLeft, PadRight	Методы	Выравнивание строки по левому или правому краю путем вставки нужного числа пробелов в начале или в конце строки
14	Remove	Метод	Удаление подстроки из заданной позиции
15	Replace	Метод	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
16	Split	Метод	Разделение строки на элементы, используя заданные разделители. Результаты

			помещаются в массив строк
17	StartsWith, EndsWith	Методы	Возвращает true или false в зависимости от того, начинается или заканчивается строка заданной подстрокой
18	Substring	Метод	Выделение подстроки, начиная с заданной позиции
19	ToCharArray	Метод	Преобразование строки в массив символов
20	ToLower, ToUpper	Методы	Преобразование символов строки к нижнему или верхнему регистру
21	Trim, TrimStart, TrimEnd	Методы	Удаление пробелов в начале и конце строки или только с одного ее конца (обратные по отношению к методам PadLeft и PadRight действия)

Рассмотрим еще один специальный пример, где используем некоторые методы этого класса.

Пример 4.

Листинг 4 – Использование методов класса *System.String*

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int ks=5;
            string s,s1;
            string[] s1=new string[ks];
            for (int k=1; k<ks; k++)
            {
                Console.WriteLine("Введите слово массива");
                s1[k]=Console.ReadLine();
            }
            s = string.Join(":", s1);
            Console.WriteLine(s);
            s1 = s.Replace(":",",");
            Console.WriteLine(s1);
            s = s1.Substring(1);
            Console.WriteLine(s);
            s1 = s.ToUpper();
            Console.WriteLine(s1);
        }
    }
}
```

Результаты работы программы:

```

Введите слово массива
лед
Введите слово массива
дождь
Введите слово массива
туман
Введите слово массива
снег
:лед:дождь:туман:снег
,лед,дождь,туман,снег
лед.дождь.туман.снег
ЛЕД,ДОЖДЬ,ТУМАН,СНЕГ
Для продолжения нажмите любую клавишу . . .

```

Данная программа демонстрирует использование ряда методов класса `System.String`. В ней сначала вводится произвольная строка, а затем осуществляется четыре последовательных вызова различных методов класса ***System.String***.

В операторе `s = string.Join(":", s1);` вызывается статический метод ***Join*** (он вызывается через имя класса) слияния массива строк ***s1*** в единую строку ***s***, помещая между словами в строке разделитель «:».

В операторе `s1=s.Replace(":",",");` вызывается метод ***Replace***, осуществляющий замену всех вхождений символа ":" новым символом, в нашем случае ",", и присваивание полученного результата строке ***s1***.

В операторе `s = s1.Substring(1);` вызывается метод ***Substring***, который возвращает подстроку из строки ***s1***, начиная с первой позиции, отбрасывая первый символ «:» и присваивая результат строке ***s***.

В операторе `s1 = s.ToUpper();` вызывает метод ***ToUpper***, выполняющий преобразование символов строки к верхнему регистру. Результат присваивается строке ***s1***.

***Пример 5.** Дана последовательность, состоящая из 10 слов. Напечатать все различные слова, указав для каждого из них число его вхождений в последовательность.*

Листинг 5 – К примеру 5

```

using System;

namespace primer
{
    class Program
    {
        static void Main(string[] args)
        { /*CompareOrdinal---Сравнение двух строк по кодам символов.
         * Разные реализации метода позволяют сравнивать строки и подстроки
         * CompareTo----Сравнение текущего экземпляра строки с другой строкой
         * 12. Дана последовательность,
         * состоящая из 10 слов.
         * Напечатать все различные слова,
         * указав для каждого из них число его вхождений в последовательность.
         */
            int col = 0;
            int ks = 10;
            string[] s1 = new string[ks];
            Console.WriteLine("Введите 10 слов ");
            for (int k = 0; k < ks; k++)
            {

```



```

        sl[k] = Console.ReadLine();
    }
    Console.WriteLine("Количество вхождений");
    for (int i = 0; i < ks; i++)
    {
        col = 0;
        for (int j = 0; j < ks; j++)
        {
            if (String.Compare(sl[i], sl[j]) == 0) col++;
        }
        Console.WriteLine(sl[i] + "    Количество вхождений слова в
последовательность  " + col);
    }
}
}
}

```

Результат работы программы:

```

Введите 10 слов
вода
лес
права
лось
дес
лес
вода
огонь
окно
окно
Количество вхождений
вода    Количество вхождений слова в последовательность  2
лес     Количество вхождений слова в последовательность  2
права   Количество вхождений слова в последовательность  1
лось    Количество вхождений слова в последовательность  1
дес     Количество вхождений слова в последовательность  1
лес     Количество вхождений слова в последовательность  2
вода    Количество вхождений слова в последовательность  2
огонь   Количество вхождений слова в последовательность  1
окно    Количество вхождений слова в последовательность  2
окно    Количество вхождений слова в последовательность  2

```

2.4 Класс *StringBuilder*

Достаточно широкие возможности класса *System.String*, к сожалению, не могут покрыть все потребности процесса обработки строковой информации. Главным ограничивающим препятствием здесь является требование неизменности его объектов.

В том случае, когда строку все же необходимо изменить, для работы со строками удобно применять другой класс - класс *StringBuilder*. Данный класс, позволяющий изменять значение своих экземпляров, определен в пространстве имен *System.Text*.

При создании экземпляра обязательно использовать операцию *new* и конструктор, например:

```
StringBuilder s = new StringBuilder();
```

Подобная инициализация является наиболее простой – в ней отсутствуют инициализирующие параметры, которые в данном случае

принимают значения по умолчанию. При этом создается пустая строка размером 16 байт.

В общем случае в конструкторе класса могут присутствовать два вида параметров. Первый вид параметров используется для инициализации строки, второй - для определения объема памяти (размера буфера), отводимой под экземпляр, например:

```
StringBuilder stud = new StringBuilder( "student", 50 );
```

При необходимости в целях экономии памяти размер буфера можно уменьшить с помощью свойства **Capacity**. Если же в процессе работы программы вследствие выполнения некоторых методов получена строка, размер которой превышает текущий размер буфера, то размер буфера автоматически будет увеличен.

Емкость буфера, не соответствующая количеству символов в строке, может увеличиваться и в результате прямых указаний программиста.

Впрочем, один или оба параметра (как уже было сказано, и как видно из первого примера) могут отсутствовать.

В следующем примере приведен еще один вид конструктора, где задан объем памяти, выделяемой строке, и ее начальное значение. Объект инициализируется подстрокой длиной 4 символа, начиная с первого (подстрока **"stud"**).

```
StringBuilder stud = new StringBuilder( "student", 1, 4, 50 );
```

Основные элементы класса **StringBuilder** приведены в таблице 3.

Таблица 3 - Основные элементы класса **System.Text.StringBuilder**

Название	Вид	Описание
Append	Метод	Добавление в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки типа <code>string</code>
AppendFormat	Метод	Добавление форматированной строки в конец строки
Capacity	Свойство	Получение или установка емкости буфера. Если устанавливаемое значение меньше текущей длины строки или больше максимального, генерируется исключение <code>ArgumentOutOfRangeException</code>
Insert	Метод	Вставка подстроки в заданную позицию
Length	Свойство	Длина строки (количество символов)
MaxCapacity	Свойство	Максимальный размер буфера
Remove	Метод	Удаление подстроки из заданной позиции
Replace	Метод	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
ToString	Метод	Преобразование в строку типа <code>string</code>

Пример 6. Демонстрация использования методов класса *System.Text.StringBuilder*.

Листинг 5

```
using System;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            string s;
            StringBuilder w = new StringBuilder();
            w.Append( "снег,лед,гололед,дождь " );
            Console.WriteLine(w);
            w.Insert(5,"солнце,");
            Console.WriteLine(w);
            Console.WriteLine("Введите построку");
            s=Console.ReadLine();
            w.Insert(0,s);
            Console.WriteLine(w);
            w.Replace( ",", ";" );
            Console.WriteLine(w);
        }
    }
}
```

Включив в программу строку *using System.Text* мы тем самым сделали доступным класс *StringBuilder*, определенный в его пространстве имен.

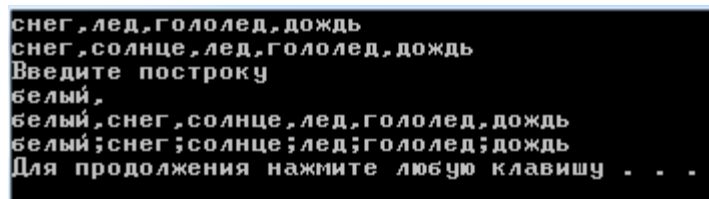
В программе объявлены две строки: *s*-статическая и *w*-динамическая. Поскольку строка *w* не инициализирована, то вызов метода *w.Append("снег,лед,гололед,дождь")* преобразует строку, поместив в нее указанную в методе строку.

Вызов метода *w.Insert(5,"солнце,")* обеспечивает вставку подстроки "солнце," в строку *w*, начиная с позиции 5.

Вызов метода *w.Insert(0,s)* обеспечивает вставку подстроки *s* (белый,) в строку *w*, начиная с позиции 0.

Вызов метода *w.Replace("",";")* производит замену всех вхождений символа "," новым символом ";".

Результаты работы программы:



```
снег,лед,гололед,дождь
снег,солнце,лед,гололед,дождь
Введите построку
белый,
белый,снег,солнце,лед,гололед,дождь
белый;снег;солнце;лед;гололед;дождь
Для продолжения нажмите любую клавишу . . .
```

2.5 Форматирование строк

В предыдущих темах для вывода информации о результатах работы программы использовались наиболее простые преобразования данных. Например, такое:

```
Console.Write(" " + x[i]),
```

где элемент числового массива в результате неявного преобразования в строковый тип и объединения со строкой из двух пробелов выводится на экран.

А оператор:

```
Console.WriteLine ( "x={0} y={1} s={2}", x, y, s);
```

использует форматный вывод. При этом оператор вывода содержит более одного параметра. Первым параметром методу передается строковый литерал, содержащий помимо обычных символов, предназначенных для вывода на консоль, *параметры* в фигурных скобках.

Параметры нумеруются с нуля, перед выводом они заменяются значениями соответствующих переменных в списке вывода: нулевой параметр заменяется значением первой переменной (в данном примере — x), первый параметр — второй, переменной (в данном примере — y), второй параметр — значением переменной s.

После номера параметра можно задать минимальную ширину поля вывода, а также указать спецификатор формата, который определяет форму представления выводимого значения.

В общем виде формат задается следующим образом:

{n [,m[:спецификатор_формата[число]]}

Здесь n — номер параметра. Параметр m определяет минимальную ширину поля, которое отводится под выводимое значение. Если выводимому числу достаточно меньшего количества позиций, неиспользуемые позиции заполняются пробелами. Если числу требуется больше позиций, параметр игнорируется.

Спецификатор формата (табл.4) используются для форматирования арифметических типов при их преобразовании в строковое представление.

Таблица 4 - Спецификаторы формата для строк

Спецификатор	Описание
С или c	Вывод значений в денежном (currency) формате. По умолчанию перед выводимым значением подставляется символ доллара (\$). Непосредственно после спецификатора можно задать целое число, определяющее длину дробной части.
D или d	Вывод целых значений. Непосредственно после спецификатора можно задать целое число, определяющее ширину поля вывода. Недостающие места заполняются нулями.
E или e	Вывод значений в экспоненциальном формате. Непосредственно после спецификатора можно задать целое число, определяющее длину дробной части. Минимальная длина порядка — 3 символа.
F или f	Вывод значений с фиксированной точностью. Непосредственно после спецификатора можно задать целое число, определяющее

	длину дробной части.
G или g	Формат общего вида. Применяется для вывода значений с фиксированной точностью или в экспоненциальном формате в зависимости от того, какой формат требует меньшего количества позиций. Для различных типов величин по умолчанию используется разная ширина вывода, например, для single - 7 позиций, для byte и sbyte - 3, для decimal – 29.
N или n	Вывод значений в формате d, ddd, ddd. ddd..., то есть группы разрядов разделяются разделителями, соответствующими региональным настройкам. Непосредственно после спецификации можно задать целое число, определяющее длину дробной части.
P или p	Вывод числа в процентном формате (число, умноженное на 100, после которого выводится знак %).
R или r	Отмена округления числа при преобразовании в строку.
X или x	Вывод значений в шестнадцатеричном формате.

Листинг 7 - Примеры применения спецификаторов

```
using System;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            int k = 342;
            double w=0.3456;
            Console.WriteLine( k. ToString( "C" ) );
            Console.WriteLine( k. ToString( "D" ) );
            Console.WriteLine(k.ToString("e"));
            Console.WriteLine( k. ToString( "G" ) );
            Console.WriteLine( "{0,6:r}", w );
            Console.WriteLine( "{0,6:0.##}", w );
        }
    }
}
```

Результат работы программы:

```
342.00p .
342
3.420000e+002
342
0.3456
0.35
```

В последнем выводе используется пользовательский шаблон форматирования: после двоеточия задан вид выводимого значения посимвольно. Если указан знак #, на этом месте будет выведена цифра числа, если она не равна нулю. Если указан 0, будет выведена любая цифра, в том числе и 0.

Пользовательский шаблон может также содержать текст, который в

общем случае заключается в апострофы.

3. Варианты заданий для самостоятельной работы

1. Дан непустой текст из заглавных русских букв, за которым следует точка. Определить, упорядочены ли буквы по алфавиту.
2. Напечатать в алфавитном порядке все различные строчные русские буквы, входящие в заданный текст из 200 символов.
3. Дана строка, состоящая из 10 слов. Напечатать эту же последовательность слов, но в обратном порядке.
4. Дана строка, состоящая из 10 слов. Напечатать эту же последовательность слов, но в алфавитном порядке.
5. Дана строка, состоящая из 10 слов. Напечатать те слова последовательности, у которых буквы упорядочены по алфавиту.
6. Дана строка, состоящая из 10 слов. Напечатать те слова последовательности, которые симметричны.
7. Дана последовательность, состоящая из 10 слов. Напечатать те слова последовательности, в которых первая буква слова входит в него еще раз.
8. Дана последовательность, состоящая из 10 слов. Напечатать слова последовательности, предварительно преобразовав каждое из них по следующему правилу: перенести первую букву в конец слова.
9. Дана последовательность, состоящая из 10 слов. Напечатать все слова последовательности, предварительно удалив из слов все последующие вхождения первой буквы данного слова.
10. Дана последовательность, состоящая из 10 слов. Напечатать все слова последовательности, удалив из каждого слова все нечетные буквы.
11. Дан текст из 60 символов. Напечатать этот текст, подчеркивая (ставя минусы в соответствующих позициях следующей строки) все входящие в него гласные.
12. Дана последовательность, состоящая из 10 слов. Напечатать все различные слова, указав для каждого из них число его вхождений в последовательность.
13. В строке S находится не более 80 латинских букв. Напечатать эту строку, добавляя после каждой буквы q букву u .
14. Дан текст, состоящий не менее из 10 слов. Напечатать все слова из текста, отличные от слова `hello`.
15. Дано предложение, состоящее не менее из 10 слов. Напечатать текст, состоящий из последних букв всех слов предложения.

16. Дано предложение, состоящее не менее из 10 слов. Напечатать все слова из предложения, содержащие ровно две буквы d.
17. Дана строка, содержащая английский текст. Найти количество слов, начинающихся с буквы b.
18. Дана строка. Подсчитать в ней количество вхождений букв r, k, t.
19. Дана строка. Определить, сколько в ней символов * , ; :
20. Дана строка, содержащая текст. Найти длину самого короткого слова и самого длинного слова.
21. Дана строка символов, среди которых есть двоеточие (:). Определить, сколько символов ему предшествует.
22. Дана строка, содержащая текст. Ввести на экран слова, содержащие три буквы.
23. Дана строка. Преобразовать ее, удалив каждый символ (*) и повторив каждый символ, отличный от (*).
24. Дана строка. Определить, сколько раз входит в нее группа букв abc.
25. Дана строка. Подсчитать количество букв k в последнем ее слове.
26. Дана строка. Подсчитать, сколько различных символов встречаются в ней. Вывести их на экран.
27. Дана строка. Подсчитать самую длинную последовательность подряд идущих букв a.
28. Упорядочить строку английских слов по алфавиту.
29. Дана строка символов, среди которых есть одна открывающаяся и одна закрывающаяся скобка. Вывести на экран все символы, расположенные внутри этих скобок.
30. Имеется строка, содержащая буквы латинского алфавита и цифры. Вывести на экран длину наибольшей последовательности цифр, идущих подряд.
31. Дан набор слов, разделенных точкой с запятой (;). Определить, сколько в нем слов, заканчивающихся буквой a.
32. Дана строка. Вывести на экран все слова, которые содержат хотя бы одну букву k.
33. Дана строка. Найти в ней те слова, которые начинаются и оканчиваются одной и той же буквой.
34. В строке заменить все двоеточия (:) точкой с запятой (;). Подсчитать количество замен.
35. В строке удалить символ двоеточие (:) и подсчитать количество удаленных символов.

36. В строке между словами вставить вместо пробела запятую.
37. Удалить часть символьной строки, заключенной в скобки (вместе со скобками)
38. Определить, сколько раз в строке встречается заданное слово.
39. В строке имеется одна точка с запятой (;). Посчитать количество символов до точки с запятой и после нее.
40. Дана строчка из n символов. Преобразовать ее, заменив все двоеточия, встречающиеся среди первых $n/2$ символов на вопросительный знак, и заменив точками все восклицательные знаки, встречающиеся среди символов, стоящих после $n/2$ символов.
41. Строка содержит одно слово. Проверить, будет ли оно читаться одинаково справа налево и слева направо (т.е. является ли оно палиндромом).
42. В записке слова зашифрованы – каждое из них записано наоборот. Расшифровать сообщение.
43. Проверить, одинаковое ли число открывающихся и закрывающихся скобок в данной строке.
44. Строка, содержащая произвольный русский текст, состоит не более чем из 200 символов. Написать, какие буквы и сколько раз встречаются в этом тексте. Ответ должен приводиться в грамматически правильной форме : $a - 25$ раз, $k - 3$ раза и т.д.
45. В строковом массиве хранятся фамилии и инициалы учеников класса. Требуется напечатать список класса с указанием для каждого ученика количества его однофамильцев.
46. Строка содержит произвольный русский текст. Проверить, каких букв в нем больше: гласных или согласных.
47. Двумерный массив содержит некоторые буквы русского алфавита, расположенные в произвольном порядке. Написать программу, проверяющую, можно ли из этих букв составить данное слово S . Каждая буква массива используется не более одного раза.
48. Результаты вступительных экзаменов представлены в виде списка их N строк, в каждой строке которого записаны фамилия студента и отметки по каждому из M экзаменов. Определить количество абитуриентов, сдавших вступительные экзамены только на «отлично».
49. Из данной символьной строки выбрать те символы, которые встречаются в ней только один раз.
50. Дано число в двоичной системе счисления. Проверить правильность ввода этого числа (в его записи должны быть только символы 0 и 1).

Если число введено неверно, повторить ввод. При правильном вводе перевести число в десятичную систему счисления.

51. Дана строка, содержащая текст, записанный строчными русскими буквами. Получить в другой строке тот же текст, записанный заглавными буквами.
52. Дана строка, содержащая текст на русском языке. Выяснить, входит ли данное слово в указанный текст, и если да, то сколько раз.
53. Дана строка, содержащая текст на русском языке. В предложениях некоторые из слов записаны подряд несколько раз (предложение заканчивается точкой или знаком восклицания). Получить в новой строке отредактированный текст, в котором удалены подряд идущие вхождения слов в предложениях.
54. Дана строка, содержащая текст, набранный заглавными русскими буквами. Произвести частотный анализ текста, т.е. указать (в процентах), сколько раз встречается та или иная буква.
55. Дана строка, содержащая текст на русском языке. Определить, сколько раз встречается в ней самое длинное слово.
56. Дана строка, содержащая произвольный текст. Проверить, правильно ли в нем расставлены круглые скобки (т.е. находится ли правее каждой открывающей скобки закрывающая, и левее закрывающей – открывающая).
57. Дана строка, содержащая текст на русском языке. Составить в алфавитном порядке список всех слов, встречающихся в этом тексте.
58. Дана строка, содержащая текст на русском языке. Определить, сколько раз встречается в нем самое короткое слово.
59. Дана строка, содержащая текст на русском языке. Выбрать из него только те символы, которые встречаются в нем только один раз. Выборку выполнить в алфавитном порядке.
60. Дана строка, содержащая текст на русском языке и некоторая буква. Найти слово, содержащее наибольшее количество указанной буквы.
61. Дана строка, содержащая текст на русском языке и некоторая буква. Подсчитать, сколько слов начинается с указанной буквы.
62. Дана строка, содержащая текст на русском языке. Найти слово, встречающееся в каждом предложении, или сообщить, что такого слова нет.
63. Дана строка, содержащая текст, включающий русские и английские слова. Подсчитать, каких букв в тексте больше – русских или латинских.
64. Дана строка, содержащая текст. Сколько слов в тексте? Сколько цифр в тексте? Сколько букв в тексте? Сколько символов в тексте?

65. Дана строка, содержащая текст, включающий русские и английские слова. Получить новую строку, заменив в исходной все заглавные буквы строчными и наоборот.
66. Дана строка, содержащая зашифрованный русский текст. Каждая буква заменяется на следующую за ней (буква Я заменяется на А). Получить в новом файле расшифровку данного текста.
67. Дана строка. Удалить из нее все лишние пробелы, оставив между словами не более одного. Результат поместить в новую строку.
68. Дана строка и некоторое слово. Напечатать те предложения строки, которые содержат данное слово более двух раз.
69. Дана строка. Напечатать в алфавитном порядке все слова из данной строки, имеющие заданную длину n.
70. Дана строка, содержащая текст на русском языке. Подсчитать количество слов, начинающихся и заканчивающихся на одну и ту же букву.

4. Содержание отчета

- 4.1. Титульный лист.
- 4.2. Краткое теоретическое описание.
- 4.3. Задание на лабораторную работу, включающее математическую формулировку задачи.
- 4.4. Результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений.

Лабораторная работа № 10

ПРОСТЕЙШИЕ КЛАССЫ. ПОЛЯ, КОНСТАНТЫ, КОНСТРУКТОРЫ, СВОЙСТВА

1 Цель и порядок работы

Цель работы – познакомиться с правилами конструирования простейших классов, получить практические навыки их построения.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя по вариантам;
- написать программу, использующую в соответствии с заданием простейший пользовательский класс;
- ввести программу, отладить и выполнить ее на компьютере;
- оформить отчет.

2 Общие сведения

Класс является основой языка C#, так как все действия в любой программе на этом языке происходят внутри класса. Все программы, использованные при изучении материала предыдущих тем или разработанные студентами самостоятельно, апеллируют к классам. Конечно, это были простые случаи использования классов. Вспомним, что все ранее разработанные программы содержали только один класс, с единственным методом: *Main()*. Не считая, разумеется, использования методов стандартных классов, таких как *Math*, *Array* и т.д.

Язык C# – объектно-ориентированный, а класс определяет основу объекта, его форму, его шаблон, по сути являющимся набором планов, по которым создаются экземпляры класса.

Класс является типом данных, определяемым пользователем. Он должен представлять собой единую логическую сущность, например, являться моделью реального объекта или процесса. Например, класс, где хранятся телефонные номера и список их владельцев, не должен содержать сведения о погоде. Членами класса являются данные и функции, предназначенные для их обработки. Именно класс связывает данные с кодом. Хотя очень простые классы могут содержать только код, или только данные. Так, например, до сих пор в предыдущих темах использован был такой класс, который содержал только один метод.

Общее определение класса:

```
[ атрибуты ] [ спецификаторы ] class имя_класса [ : предки ]  
{тело_класса}
```

Обязательными являются только ключевое слово **class**, а также имя (заданное по общим правилам языка) и тело класса (список его членов),

заключенное в фигурные скобки. Если класс не содержит ни одного члена, то список членов класса будет пустым. Например:

```
class Prim { }
```

При определении класса при необходимости могут быть указаны его предки (базовые классы), а также атрибуты и спецификаторы, определяющие различные характеристики класса. Как видно из определения класса, предки, атрибуты и спецификаторы класса являются необязательными элементами его определения и могут отсутствовать. Класс может быть определен внутри другого класса, в этом случае он является вложенным в другой класс.

Новое описание класса взамен унаследованного от предка в случае использования вложенных классов можно задать с помощью операции ***new***.

Для других элементов программы возможна организация различных типов доступа к членам класса. Тип доступа определяется спецификаторами класса, приведенными в таблице 1.

Таблица 1 - Спецификаторы доступа

№	Спецификатор	Описание
1	public	Доступ не ограничен.
2	protected	Используется для вложенных классов. Доступ только из элементов данного и производных классов.
3	internal	Доступ только из данной программы (сборки).
4	protected internal	Доступ только из данного и производных классов или из данной программы (сборки).
5	private	Используется для вложенных классов. Доступ только из элементов класса, внутри которого описан данный класс.

Спецификаторы доступа могут комбинироваться с другими спецификаторами.

Для классов, которые описываются непосредственно в пространстве имен, по умолчанию подразумевается спецификатор ***internal***, при нем возможен доступ только из данной программы.

Для расширения доступа к членам класса существует вариант - спецификатор ***public***, но его требуется указать явно. Спецификатор ***public*** открывает доступ к члену класса для любого кода программы.

Спецификатор ***private***, наоборот, ограничивает доступ к члену класса только членами этого же класса.

Класс - это абстрактное понятие, определяющее характеристики и поведение некоторого множества экземпляров или объектов этого класса. В программе экземпляр класса может быть создан с помощью операции ***new***:

```
Prim z = new Prim ();
```

Для каждого объекта при его создании в памяти выделяется отдельная область, в которой хранятся его данные.

В классе могут присутствовать статические элементы, которые существуют в единственном экземпляре для всех объектов класса. Часто статические данные называют данными класса, а остальные — данными экземпляра.

Для работы с данными класса используются методы класса (статические методы), для работы с данными экземпляра — методы экземпляра, или просто методы. Методы класса всегда хранятся в единственном экземпляре.

Ограничение доступа к членам класса является неотъемлемой частью объектно-ориентированного программирования, так как оно позволяет предотвратить неправильное использование объектов. Существует несколько общих принципов правильного применения открытого и закрытого доступа.

- Члены класса, которые используются только внутри класса, должны быть закрытыми.
- Данные экземпляра, которые должны находиться в пределах определенного диапазона, должны быть закрытыми, а доступ к ним должен быть реализован через открытые методы, которые могут проверять нахождение значений в пределах этого диапазона.
- Если изменение члена может вызвать эффект, способный распространиться за его пределы, то он должен быть закрытым и доступ к нему должен быть контролируемым.
- Члены, некорректное использование которых может повредить объект, должны быть закрытыми. Доступ к ним должен быть реализован через открытые методы, исключаящие некорректное использование этих членов.
- Методы, получающие значения закрытых данных, должны быть открытыми.
- Переменные экземпляра могут быть открытыми, если нет необходимости делать их закрытыми.

Перечислены общие правила, которые, к сожалению, подчас приходится нарушать. Но в принципе, если им следовать, можно рассчитывать на получение устойчивых объектов.

Потенциал, заложенный в классах, очень велик. И прежде всего, его мощь определяется разнообразием членов класса:

- *Константы* класса хранят неизменяемые значения, связанные с классом.
- *Поля* содержат данные класса.
- *Методы* реализуют вычисления или другие действия, выполняемые классом или экземпляром.

- *Свойства* определяют характеристики класса вместе с методами их записи и чтения.
- *Конструкторы* реализуют действия по инициализации экземпляров или класса в целом.
- *Деструкторы* определяют действия, которые необходимо выполнить до того как объект будет уничтожен.
- *Индексаторы* обеспечивают возможность доступа к элементам класса по их порядковому номеру.
- *Операции* задают действия с объектами с помощью знаков операций.
- *События* определяют уведомления, которые может генерировать класс.
- *Типы* — это типы данных, внутренние по отношению к классу.

В этой теме рассматриваются только некоторые из перечисленных членов класса.

Данные класса

Данные, содержащиеся в классе, могут быть *переменными* или *константами* и задаются в соответствии с уже рассмотренными правилами. Переменные, описанные в классе, называются *полями* класса.

При описании элементов класса можно также указывать атрибуты и спецификаторы, задающие различные характеристики элементов. Синтаксис описания элемента данных приведен ниже:

[атрибуты] [спецификаторы] [const] тип имя [= начальное_значение]

В данном описании обязательными элементами являются *тип* и *имя*. Ситуации использования элементов описания *const*, *начальное_значение* в общем случае не требуют дополнительных пояснений. Опустив для будущих тем описатель *атрибуты*, который в данной теме использоваться не будет, уделим максимальное внимание *спецификаторам* полей и констант (табл.2).

Таблица 2 - Спецификаторы полей и констант класса

№	Спецификатор	Описание
1	new	Новое описание поля, скрывающее унаследованный элемент класса.
2	public	Доступ к элементу не ограничен.
3	protected	Доступ только из элементов данного и производных классов.
4	internal	Доступ только из данной программы (сборки).

5	protected internal	Доступ только из данного и производных классов или из данной программы (сборки).
6	private	Доступ только из данного класса.
7	volatile	Поле может изменяться другим процессом или системой.
8	readonly	Поле доступно только для чтения.
9	static	Одно поле для всех экземпляров класса.

Для констант можно использовать только спецификаторы 1-6. Указание типа доступа необязательно. По умолчанию элементы класса считаются закрытыми (*private*). Для полей класса этот вид доступа является предпочтительным, поскольку поля определяют внутреннее строение класса, которое должно быть скрыто от пользователя. Все методы класса имеют непосредственный доступ к его закрытым полям.

Поля, описанные со спецификатором *static*, а также константы существуют в единственном экземпляре для всех объектов класса, поэтому к ним обращаются не через имя экземпляра, а через имя класса. Если класс содержит только статические элементы, экземпляр класса создавать не требуется. Именно этот факт был использован во всех предыдущих листингах.

Обращение к полю класса выполняется с помощью *операции доступа* (точка). Справа от точки задается имя поля, слева — имя экземпляра для обычных полей или имя класса для статических. В листинге 1 приведены пример простого класса *Prim* и два способа обращения к его полям.

Листинг 1- Класс Prim, содержащий поля и константу

```
using System;
namespace ConsoleApplication2
{
    class Prim
    {
        public int p = 1;           //поле данных типа int, доступ неограничен
        public const double z = -5.31; //константа типа double, доступ неограничен
        public static string s = " Prim "; //статическое поле класса, доступ неограничен
        double y;                  //закрытое поле данных типа double
    }
    class Class1
    {
        static void Main()
        {
            Prim a = new Prim();    // создание экземпляра класса prim
            Console.WriteLine(a.p); // вывод значения поля p экземпляра класса a
            Console.WriteLine(Prim.z); // вывод значения константы z класса Prim
            Console.WriteLine(Prim.s); // вывод значения статическое поля s класса Prim
        }
    }
}
```

Результат работы программы:

```
1
-5.31
Prim
```

Значение закрытого поля *y* из класса *Class1* недоступно.

Все поля сначала автоматически инициализируются нулем соответствующего типа (*0*, *null*). Если в программе присутствует явная инициализация, то после этого нулевое значение поля меняется на заданное в ней значение. Задание начальных значений для статических полей выполняется при инициализации класса, а обычных — при создании экземпляра.

Поля со спецификатором *readonly* предназначены только для чтения. Установить значение такого поля можно либо при его описании, либо в конструкторе (конструкторы рассматриваются далее в этой работе).

Метод

Программисты всегда очень ценили свое время. Изобретать «велосипед» в 101 раз, если есть уже готовый, им по понятным причинам не очень нравилось. Уже достаточно давно в языках высокого уровня существует механизм, позволяющий избегать таких ситуаций.

Этот механизм полезен тогда, когда желательно использовать некий уже опробованный алгоритм в своих целях. Он полезен и тогда, когда встречается ситуация, где одну и ту же группу операторов, реализующих определенный алгоритм, требуется повторить без изменений в нескольких местах программы. В целях экономии памяти такую группу операторов можно определить как самостоятельную единицу, которую в программировании принято называть *подпрограммой*. Её требуется описать один раз, а вызывать для исполнения по имени можно любое количество раз из различных мест программы. При вызове подпрограммы активизируется последовательность образующих ее операторов, преобразованных специальным образом посредством передаваемых ей параметров.

С другой стороны, поскольку большие монолитные программы сложны для разработки, отладки и сопровождения, правильный подход к составлению программы состоит в том, что задача разбивается на подзадачи, которые могут быть реализованы в виде отдельных подпрограмм. Во многих языках для организации подпрограмм используются процедуры и функции. Программы, состоящие из процедур и функций, называются модульными. Таким образом, посредством создания подпрограмм реализуется метод нисходящего проектирования программ.

Модульная программа, как правило, имеет *иерархическую структуру*, при которой процедуры и функции высшего уровня могут вызывать процедуры и функции более низкого уровня.

В языке C# понятия процедуры и функции заменены одним новым. Название ему — метод, который как элемент объектно-ориентированного программирования является членом класса.

Метод — это функциональный член класса, который реализует вычисления или другие действия, выполняемые классом или экземпляром. Методы определяют поведение класса. Он представляет собой законченный фрагмент кода, к которому можно обратиться по имени. Он описывается

один раз, а вызываться может столько раз, сколько необходимо. Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов. В данной теме организация методов не рассматривается. Она подробно будет рассмотрена в следующей теме.

Конструктор

Конструктор предназначен для инициализации объекта. Он вызывается автоматически при создании объекта класса с помощью операции *new*. Имя конструктора совпадает с именем класса. Ниже перечислены свойства конструкторов.

- Конструктор не возвращает значение.
- Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации.
- Если программист не указал ни одного конструктора или какие-то поля не были инициализированы, полям значимых типов присваивается ноль, полям ссылочных типов — значение *null*.
- Конструктор, вызываемый без параметров, называется конструктором по умолчанию.

В листинге 1 начальные значения полей класса заданы при описании класса. Если же при создании объектов требуется присваивать полю разные значения, это следует делать в конструкторе.

В листинге 2 в класс *Prim* добавлен конструктор, а поля сделаны закрытыми. В программе создаются два объекта с различными значениями полей.

Листинг 2- Класс с конструктором

```
using System;
namespace ConsoleApplication1
(
    class Prim
    {
        public Prim(int a, double y)    // конструктор с параметрами
        {
            this.a = a;
            this.y = y;
        }
        public double Gety()            // метод получения поля y
        {
            return y;
        }
        int a;
        double y;
    }

    class Class1
    {
        static void Main( )
        {
            Prim a = new Prim( 10, 0.01 );    // вызов конструктора
            Console.WriteLine( a.Gety( ) );    // результат: 0,01
            Prim b = new Prim( 1. 2.2 );       // вызов конструктора
            Console.WriteLine( b.Gety( ) );     // результат: 2,2
        }
    }
}
```

}

В классе, чтобы обеспечить возможность инициализации объектов разными способами, можно иметь несколько конструкторов, но при этом они должны иметь разные сигнатуры.

Листинг 3- Класс с 2-мя конструкторами

```
class Prim
{
    public Prim( int a )                // конструктор 1
    {
        this.a = a;
        this.y = 0.002;
    }

    public Prim( double y )            // конструктор 2
    {
        this.a = 1;
        this.y = y;
    }
    ...
}

...
Prim x = new Prim( 300 ) ;             // вызов конструктора 1
Prim y = new Prim( 5.71 );            // вызов конструктора 2
```

Если один из конструкторов выполняет какие-либо действия, а другой должен делать то же самое плюс еще что-нибудь, удобно *вызвать первый конструктор из второго*. Для этого используется уже известное ключевое слово **this**, например:

```
class Prim
{
    public Prim( int a )                // конструктор 1
    {
        this.a = a;
    }
    public Prim(int a, double y) : this( a )    // вызов конструктора 1
    {
        this.y = y;
    }
}
```

Конструкция, находящаяся после двоеточия, называется *инициализатором*, то есть тем кодом, который выполняется до начала выполнения тела конструктора.

Вспомним, что все классы в С# имеют общую предка — класс `object`. Конструктор любого класса, если не указан инициализатор, автоматически вызывает конструктор своего предка. Это можно сделать и явным образом с помощью ключевого слова `base`, обозначающего конструктор базового класса. Таким образом, первый конструктор из предыдущего примера можно записать и так:

```
public Prim ( int a ) : base( )        // конструктор 1
{
    this.a = a;
}
```

Конструктор базового класса вызывается явным образом в тех случаях, когда ему требуется передать параметры.

До сих пор речь шла о *конструкторах экземпляра*. Существует второй тип конструкторов — *статические конструкторы*, или *конструкторы*

класса. Конструктор экземпляра инициализирует данные экземпляра, конструктор класса — данные класса.

Статический конструктор не имеет параметров, его нельзя вызвать явным образом. Система сама определяет момент, в который требуется его выполнить. Гарантируется только, что это происходит до создания первого экземпляра объекта и до вызова любого статического метода.

Некоторые классы содержат только статические данные, и, следовательно, создавать экземпляры таких объектов не имеет смысла.

В С# введена возможность описывать статический класс, то есть класс с модификатором *static*. Экземпляры такого класса создавать запрещено, и кроме того, от него запрещено наследовать. Все элементы такого класса должны явным образом объявляться с модификатором *static* (константы и вложенные типы классифицируются как статические элементы автоматически). Конструктор экземпляра для статического класса задавать, естественно, запрещается.

Листинг 3 - Статический класс

```
using System;
namespace ConsoleApplication1
{
    static class D
    {
        static int a = 200;
        static double b = 0.002;

        public static void Print ( )
        {
            Console.WriteLine( "a = " + a );
            Console.WriteLine( "b = " + b );
        }
    }
    class Class1
    {
        static void Main( )
        {
            D.Print( );
        }
    }
}
```

Свойства

Свойства служат для организации доступа к полям класса. Как правило, свойство связано с закрытым полем класса и определяет методы его получения и установки. Синтаксис свойства:

```
[ атрибуты ] [ спецификаторы ] тип имя_свойства
{
    [ get код_доступа ]
    [ set код_доступа ]
}
```

Значения спецификаторов для свойств и методов аналогичны. Чаще всего свойства объявляются как открытые (со спецификатором *public*), поскольку они входят в интерфейс объекта.

Код доступа представляет собой блоки операторов, которые выполняются при получении (*get*) или установке (*set*) свойства. Может отсутствовать либо часть *get*, либо *set*, но не обе одновременно.

Если отсутствует часть *set*, свойство доступно только для чтения (read-only), если отсутствует часть *get*, свойство доступно только для записи (write-only).

В C# введена удобная возможность задавать разный уровень доступа для частей *get* и *set*. Например, во многих классах возникает потребность обеспечить неограниченный доступ для чтения и ограниченный — для записи.

Спецификаторы доступа для отдельной части должны задавать либо такой же, либо более ограниченный доступ, чем спецификатор доступа для свойства в целом. Например, если свойство описано как *public*, его части могут иметь любой спецификатор доступа, а если свойство имеет доступ *protected internal*, его части могут объявляться как *internal*, *protected* или *private*. Синтаксис свойства имеет вид:

```
[ атрибуты ] [ спецификаторы ] тип имя_свойства
{
    [ [ атрибуты ] [ спецификаторы ] get код_доступа ]
    [ [ атрибуты ] [ спецификаторы ] set код_доступа ] }
```

Пример описания свойств:

```
public class Button: Control
{
    private string caption;           // закрытое поле, с которым связано свойство
    public string Caption {           // свойство
        get {                         // способ получения свойства
            return caption;
        }
        set {                         // способ установки свойства
            if (caption != value) {
                caption = value;
            }
        }
    }
    ...
}
```

Двоеточие между именами *Button* и *Control* в заголовке класса *Button* означает, что класс *Button* является производным от класса *Control*.

Метод записи обычно содержит действия по проверке допустимости устанавливаемого значения, метод чтения может содержать, например, поддержку счетчика обращений к полю.

В программе свойство выглядит как поле класса, например:

```
Button ok = new Button( );
ok.Caption = "ОК";           // вызывается метод установки свойства
string s = ok.Caption;       // вызывается метод получения свойства
```

При обращении к свойству автоматически вызываются указанные в нем методы чтения и установки.

Синтаксически чтение и запись свойства выглядят почти как методы. Метод **get** должен содержать оператор **return**, возвращающий выражение, для типа которого должно существовать неявное преобразование к типу свойства. В методе **set** используется параметр со стандартным именем **value**, который содержит устанавливаемое значение.

Вообще говоря, свойство может и не связываться с полем. Фактически, оно описывает один или два метода, которые осуществляют некоторые действия над данными того же типа, что и свойство. В отличие от открытых полей, *свойства обеспечивают разделение между внутренним состоянием объекта и его интерфейсом* и, таким образом, упрощают внесение изменений в класс.

С помощью свойств можно отложить инициализацию поля до того момента, когда оно фактически потребуется, например:

```
class A
{
    private static ComplexObject x;           // закрытое поле
    public static ComplexObject X             // свойство
    {
        get
        {
            if(x == null) {
                x = new ComplexObject ( );    // создание объекта при 1-м обращении
            }
            return x;
        }
    }
    ...
}
```

3. Варианты заданий для самостоятельной работы

Далее для закрепления материала по данной теме приведены задания для самостоятельного выполнения. Для всех заданий необходимо написать программу с описанием простейшего класса со следующими элементами: полями, конструкторами, свойствами. Обработка информации должна, как и ранее, осуществляться в классе, создаваемом системой по умолчанию в пространстве имен **namespace ConsoleApplication1** в методе **Main()**. При возникновении ошибок должны выбрасываться исключения.

1. Описать класс, реализующий десятичный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. При выходе за границы диапазона выбрасываются исключения.
2. Описать класс, реализующий восьмеричный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. При выходе за границы

диапазона выбрасываются исключения.

3. Описать класс, представляющий треугольник. Предусмотреть методы для создания объектов, перемещения на плоскости, изменения размеров и вращения на заданный угол. Описать свойства для получения состояния объекта. При невозможности построения треугольника выбрасывается исключение.
4. Построить описание класса, содержащего информацию о почтовом адресе организации. Предусмотреть возможность отдельного изменения составных частей адреса и проверки допустимости вводимых значений. В случае недопустимых значений полей выбрасываются исключения.
5. Составить описание класса для представления комплексных чисел. Обеспечить выполнение операций сложения, вычитания и умножения комплексных чисел.
6. Составить описание класса для вектора, заданного координатами его концов в трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора (суммы или разности), вычисления скалярного произведения двух векторов, длины вектора, косинуса угла между векторами.
7. Составить описание класса прямоугольников со сторонами, параллельными осям координат. Предусмотреть возможность перемещения прямоугольников на плоскости, изменение размеров, построение наименьшего прямоугольника, содержащего два заданных прямоугольника, и прямоугольника, являющегося общей частью (пересечением) двух прямоугольников.
8. Составить описание класса для представления даты. Предусмотреть возможности установки даты и изменения ее отдельных полей (год, месяц, день) с проверкой допустимости вводимых значений. В случае недопустимых значений полей выбрасываются исключения.
9. Составить описание класса для представления времени. Предусмотреть возможности установки времени и изменения его отдельных полей (час, минута, секунда) с проверкой допустимости вводимых значений. В случае недопустимых значений полей выбрасываются исключения.
10. Составить описание класса многочлена вида $ax^2 + bx + c$. Предусмотреть:
 - вычисление значения многочлена для заданного аргумента;
 - вывод на экран описания многочлена.
11. Описать класс, представляющий треугольник. Предусмотреть создание объектов, вычисление площади, периметра и точки пересечения медиан. Описать свойства для получения состояния объекта. При невозможности построения треугольника выбрасывается исключение.
12. Описать класс, представляющий круг. Предусмотреть создание объектов,

вычисление площади круга, длины окружности и проверки попадания заданной точки внутрь круга. Описать свойства для получения состояния объекта.

13. Описать класс для работы со строкой, позволяющей хранить только двоичное число и выполнять с ним арифметические операции. Предусмотреть инициализацию с проверкой допустимости значений. В случае недопустимых значений выбрасываются исключения.
14. Описать класс дробей — рациональных чисел, являющихся отношением двух целых чисел. Предусмотреть сложение, вычитание, умножение и деление дробей.
15. Описать класс, реализующий двоичный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. При выходе за границы диапазона выбрасываются исключения.
16. Описать класс «комната», содержащий сведения о метраже, высоте потолков и количестве окон. Предусмотреть инициализацию с проверкой допустимости значений полей. В случае недопустимых значений полей выбрасываются исключения. Предусмотреть вычисление площади и объема комнаты и свойства для получения состояния объекта.
17. Описать класс, представляющий нелинейное уравнение вида $ax - \cos(x) = 0$. Предусмотреть решение этого уравнения на заданном интервале методом деления пополам и выбрасывание исключения в случае отсутствия корня. Описать свойства для получения состояния объекта.
18. Описать класс, представляющий квадратное уравнение вида $ax^2 + bx + c = 0$. Предусмотреть решение этого уравнения и выбрасывание исключения в случае отсутствия корней. Описать свойства для получения состояния объекта.
19. Описать класс «процессор», содержащий сведения о марке, тактовой частоте, объеме кэша и стоимости. Предусмотреть инициализацию с проверкой допустимости значений полей. В случае недопустимых значений полей выбрасываются исключения. Описать свойства для получения состояния объекта.
20. Описать класс «материнская плата», включающий класс «процессор» и объем установленной оперативной памяти. Предусмотреть инициализацию с проверкой допустимости значений поля объема памяти. В случае недопустимых значений поля выбрасывается исключение. Описать свойства для получения состояния объекта.
21. Составить описание класса многочлена вида $ax^2 + bx + c$. Предусмотреть:
 - операцию сложения, вычитания и умножения многочленов с получением нового объекта-многочлена;

- вывод на экран описания многочлена.

4. Содержание отчета

- 4.1. Титульный лист.
- 4.2. Краткое теоретическое описание.
- 4.3. Задание на лабораторную работу, включающее математическую формулировку задачи.
- 4.4. Результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений.

Список литературы

1. Малыхина М.П., Частикова В.А. Программирование на языке высокого уровня С#. Учебное пособие. – Краснодар: Изд-во КубГТУ, 2011. – 251 с.
2. Гуриков С.Р. Введение в программирование на языке Visual C#: Учебное пособие. - М.: Форум: НИЦ ИНФРА-М, 2013. - 448 с. – Режим доступа: <http://znanium.com/catalog.php?bookinfo=404441>
3. Дейл Н. Программирование на C++ [Электронный ресурс] / Н. Дейл, Ч. Уимз, М. Хедингтон; Пер. с англ. - М.: ДМК Пресс, 2007. - 672 с.: ил. – Режим доступа: <http://znanium.com/catalog.php?bookinfo=407353>.
4. Частикова В.А. Языки программирования [Электронный ресурс]: электрон. обучающий комплекс / КубГТУ, Каф. компьютер. технологий и информ. безопасности. – Краснодар, 2010.