Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра безопасности информационных систем (БИС)

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C#

Составитель: С.С. Харченко

Учебное пособие по лабораторным работам

В-Спектр
Томск, 2026

УДК 004.42
ББК 32.973
Х 22

Х 22 **Харченко С.С., составитель.** Основы программирования на языке C#: учебное пособие. – Томск: В-Спектр (ИП В.М. Бочкарева), 2026. – 102 с. ISBN 978-5-902958-55-0

Учебное пособие содержит теоретические сведения и методические указания, необходимые для выполнения лабораторных работ по дисциплине «Основы программирования», и предназначено для студентов 1-го курса, обучающихся по направлению информационной безопасности. Охарактеризованы требования к лабораторным работам, описана структура и методика выполнения лабораторных работ. В пособии кратко рассматривается порядок разработки программ на языке C# в среде разработки Microsoft Visual Studio 2022 и тестовом редакторе Visual Studio Code. При этом, как и положено, рассматривается алфавит языка, реализация основных типов алгоритмических структур (линейная, разветвленная, циклическая), работа с массивами и функциями на данном языке, работа с классами. В конце данного пособия, приводится глава с заданием для выполнения лабораторных работ по данной дисциплине, пособие поделено на главы, соответствующие темам лабораторных работ.

УДК 004.42
ББК 32.973

Одобрено на заседании кафедры БИС
протокол № 8 от 27 апреля 2026 г.

ISBN 978-5-902958-55-0

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель дисциплины «Основы программирования» – научить студентов разрабатывать алгоритмы и реализовывать их на языке программирования высокого уровня в виде программного обеспечения для персонального компьютера. Для достижения цели дисциплины должна быть решена следующие задачи – познакомить студентов с основными составляющими языка программирования высокого уровня. Ознакомить студентов с основами объектно-ориентированного программирования. Выработать навык написания программного кода у студентов.

Согласно ФГОС ВО и основной образовательной программой, процесс изучения дисциплины «Основы программирования» в зависимости от специальности направлен на формирование компетенции ОПК-7 в одной из следующих интерпретациях:

– ОПК-7. Способен использовать языки программирования и технологии разработки программных средств для решения задач профессиональной деятельности;

– ОПК-7. Способен создавать программы на языке высокого уровня, применять существующие реализации структур данных и алгоритмов;

– ОПК-7. Способен создавать программы на языках общего назначения, применять методы и инструментальные средства программирования для решения профессиональных задач, осуществлять обоснованный выбор инструментария программирования и способов организации программ;

– ОПК-7. Способен создавать программы на языках высокого уровня, применять методы и инструментальные средства программирования для решения профессиональных задач, осуществлять обоснованный выбор инструментария программирования.

В результате изучения дисциплины определены следующие индикаторы достижения компетенции:

– знает основные конструкции и библиотеки языков программирования, принципы построения программ в процедурно-ориентированной и объектно-ориентированной парадигмах;

– умеет реализовывать алгоритмы на языке программирования, работать с интегрированной средой разработки программного обеспечения, проводить оценку вычислительной сложности алгоритма;

– владеет навыками выбора и разработки алгоритмов при решении типовых задач программирования, разработки и тестирования программ по поставленной спецификации.

Указанные задачи частично могут быть решены с помощью данного учебного пособия.

ТЕОРЕТИЧЕСКИЕ ВЫКЛАДКИ

Язык программирования C#

Язык программирования C# (C Sharp) является мощным и универсальным инструментом, который предоставляет возможности для создания разнообразных приложений, начиная от десктопных программ до веб-сервисов. Разработанный компанией Microsoft, C# объединяет в себе простоту использования и высокую производительность, что делает его идеальным выбором для начинающих программистов.

Одной из ключевых особенностей C# является его объектно-ориентированный подход, который позволяет структурировать код в виде объектов, упрощая тем самым процесс разработки. Начинающие могут легко освоить основные концепции, такие как классы и объекты, что обеспечивает более интуитивный подход к программированию.

Язык C# также известен своей мощной системой типов, которая помогает предотвратить ошибки времени выполнения. Это особенно полезно для новичков, поскольку они могут более уверенно создавать программы, получая обратную связь о возможных проблемах еще на этапе компиляции.

Среди других преимуществ C# стоит выделить широкий спектр интегрированных сред разработки (IDE), таких как Visual Studio, которые предоставляют интуитивный интерфейс и многочисленные инструменты для облегчения процесса написания кода. Это создает комфортные условия для обучения и позволяет сосредоточиться на самом процессе программирования, а не на настройке окружения разработки.

Таким образом, для тех, кто только начинает свой путь в программировании, язык C# предоставляет отличную основу, объединяя простоту использования, сильную типизацию и богатые возможности для создания разнообразных программных продуктов

Как правило, чтобы выполнить программу на C#, необходимо пройти через 6 этапов: редактирование, предварительную обработку, компиляцию, компоновку, загрузку и выполнение. В данном пособии мы будем рассматривать программирование в среде Visual Studio 2022. Важно отметить, что каждый из этих этапов имеет свою роль в процессе создания и запуска программы, обеспечивая переход от исходного кода к исполнению на конкретной платформе.

Первый этап представляет создание и редактирование файла с исходным кодом программы. Он может выполняться с помощью простейшего редактора текстов программ. Программист набирает в этом редакторе свою программу. В этом файле программист определяет переменные, функции, классы и другие элементы кода, которые составляют программу. При необходимости он снова обращается к ней и вносит с помощью этого редактора изменения в исходный код программы. Далее программа сохраняется на диске. Файлы с исходным кодом на языке программирования C# имеют расширение «cs».

На втором этапе компилятор начинает препроцессорную обработку текста программы, прежде чем ее компилировать. Обычно это включение других текстовых файлов в файл, который подлежит компиляции. Препроцессорная обработка инициируется компилятором перед тем, как программа будет преобразована в промежуточный язык (IL). Это позволяет забирать нужные программы-функции в текст компилируемой программы до начала процесса компоновки.

Третий этап – это компиляция. Как правило, программы на языке C# содержат ссылки на различные функции, которые определены вне самой программы. Например, в стандартных библиотеках или в личных библиотеках программистов. Программы, предназначенные для платформы .NET Framework, компилируются в промежуточный язык (IL). Перед выполнением какого-либо метода в первый раз JIT-компилятор компилирует IL-код в машинный код для локального компьютера.

Четвертый этап – компоновка. Компоновщик связывает объектный код с кодами отсутствующих функций и создает, таким образом, исполняемый загрузочный модуль. В этот момент включаются библиотеки и другие зависимости, необходимые для выполнения программы. Компоновка генерирует исполняемый файл (обычно с расширением .exe или .dll), который содержит код программы и все необходимые ресурсы.

Пятый этап – загрузка. Перед выполнением программа должна быть размещена в памяти. Это делается с помощью загрузчика, который забирает загрузочный модуль программы с диска и перемещает его в память. В этот момент также выполняется проверка типов, что помогает предотвратить многие ошибки времени выполнения.

Шестой этап – это выполнение. Программа редко заработает с первой попытки. Каждый из названных этапов может заканчиваться ошибкой или неудачей из-за ошибки. Тогда программист должен вернуться к редактированию исходного текста программы. Он должен внести необходимые изменения в текст программы, предварительно его хорошо проанализировав. Затем снова пройти через все этапы работы с исходным текстом программы до получения работающего без ошибок загрузочного модуля.

Настройка рабочего окружения

После запуска Visual Studio 2022 при помощи нажатия горящих клавиш «**Shift+Ctrl+N**» запускаем меню создания нового проекта, в форме поиска шаблона приложения введите «Консольное приложение» (рис. 1), далее выберите проект «Консольное приложение (.Net Framework)» либо «Консольное приложение (Майкрософт)». В более ранних версиях Visual Studio можно встретить «Консольное приложение (.Net Core)» вместо «Консольное приложение (Майкрософт)». Предпочтительнее создавать проект из шаблона на основе .Net Core, так как это позволит создавать кроссплатформенные решения и переносить ваш код на другие платформы (Ubuntu, MacOs).

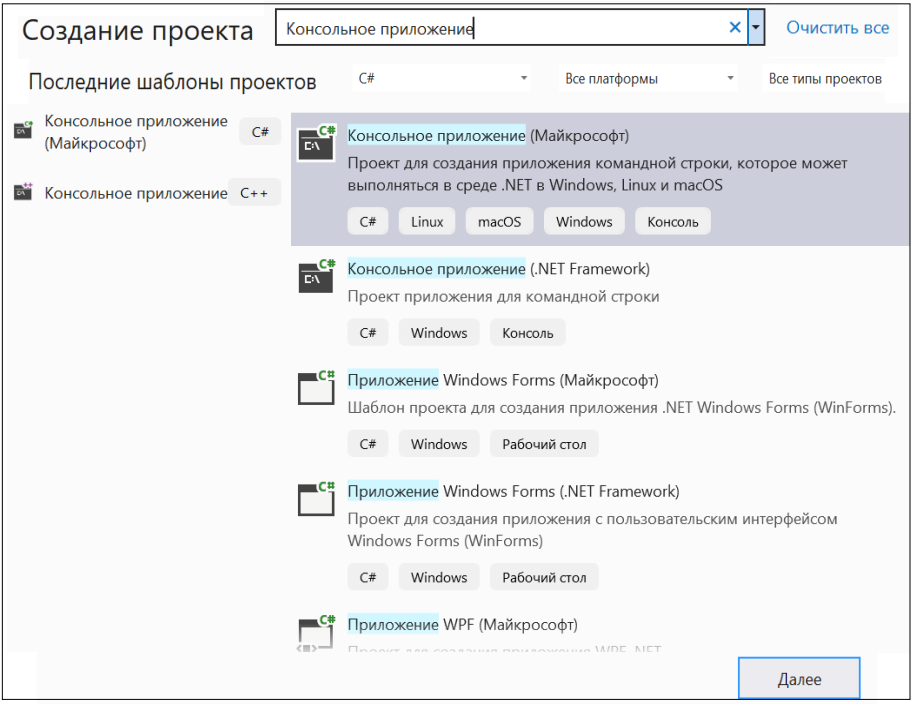
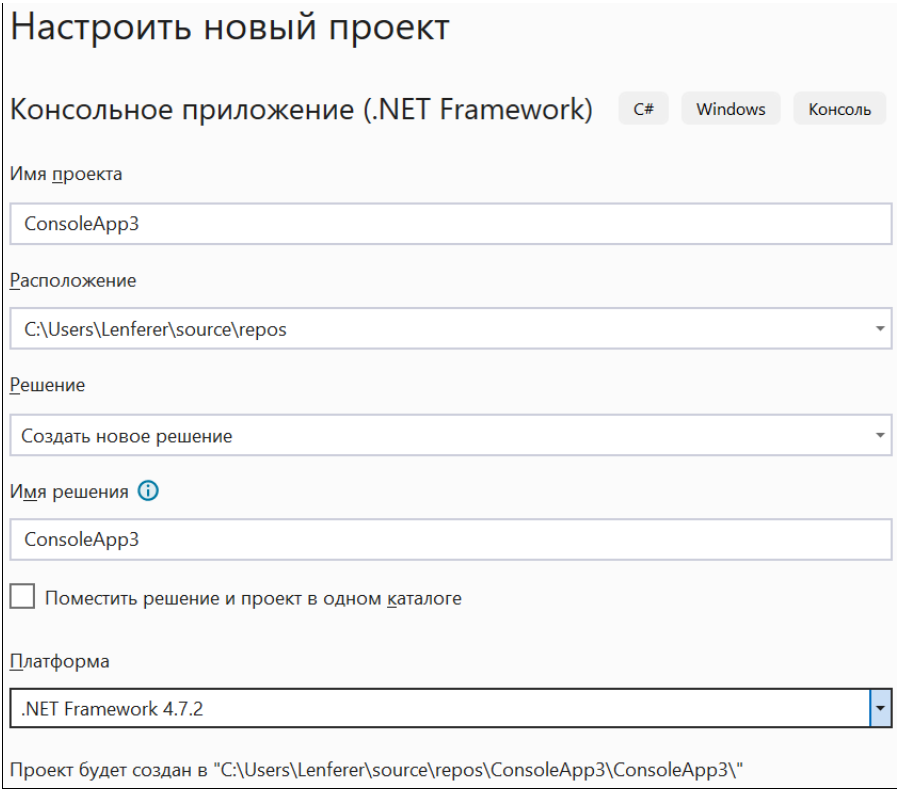


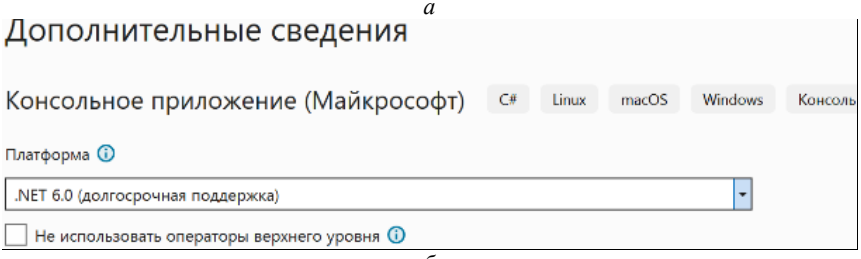
Рис. 1. Меню создания нового проекта

После выбора типа проекта откроется меню настройки проекта, где необходимо выбрать название проекта, его расположение на диске, версию .Net Framework и другие характеристики (рис. 2, *а*, *б*). После создания проекта откроется главное окно Visual Studio для работы с проектом (рис. 3). В правой части окна располагается обозревателей решений, в котором можно увидеть

всю иерархию файлом проекта. Большую часть по центру окна занимает редактор кода, в котором можно открывать сразу несколько вкладок для редактирования разных файлов проекта. В нижней части окна располагаются дополнительные элементы, такие как «Вывод», «Список ошибок» и др. в зависимости от пользовательских настроек.



а



б

Рис. 2. Меню настройки создаваемого проекта

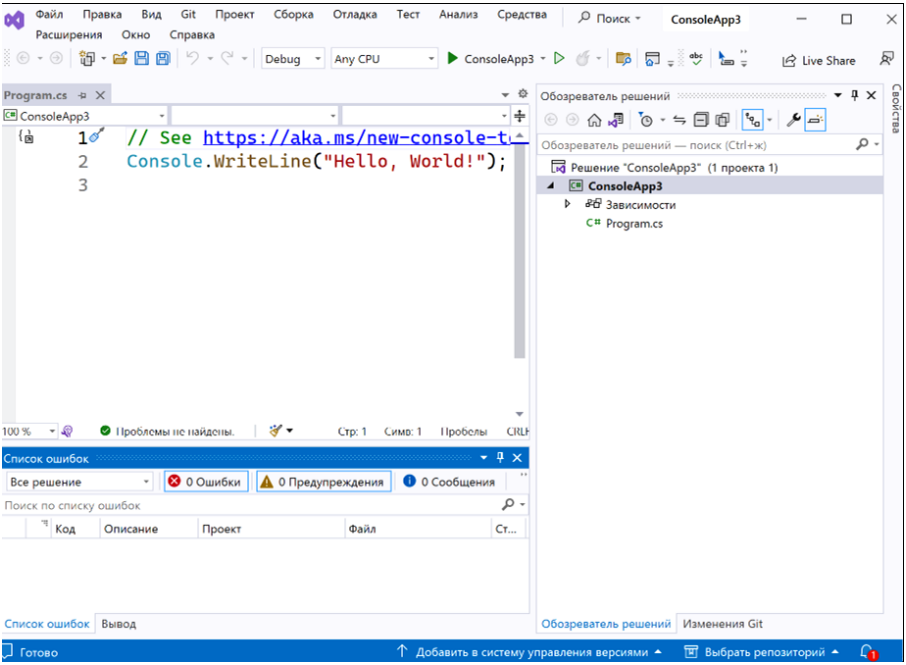


Рис. 3. Основное окно для работы с проектом

Попробуйте ввести простой фрагмент кода, представленной ниже:

Console.WriteLine(«Основы программирования»);

Это строка кода выводит сообщение, которое размещено в кавычках. Название «BasicsOfProgrammingConsoleApplication» может отличаться в вашем коде, оно зависит от того, как вы назвали свое приложение на этапе создания проекта. Для запуска программы ее необходимо сначала скомпилировать и собрать проект, для этого на панели меню выберите **«Сборка – Собрать решение»** или воспользуйтесь горячими клавишами **«Ctrl+Shift+B».** Если все прошло успешно автоматически откроется оснастка «Вывод» (рис. 4).

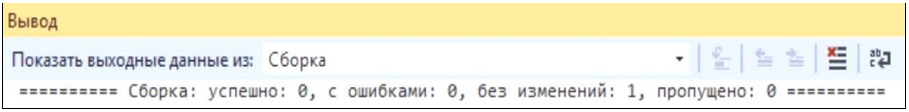


Рис. 4. Сообщение об успешной сборки проекта

Теперь для того, чтобы запустить программу перейдите в меню **«Отладка – Запуск без отладки»**. Если на этом шаге выбрать просто **«Отладка»** программа сразу же закроется после выполнения. После запуска приложения

откроется стандартная консоль и в ней будет выведено сообщения «Основы программирования» (рис. 5).

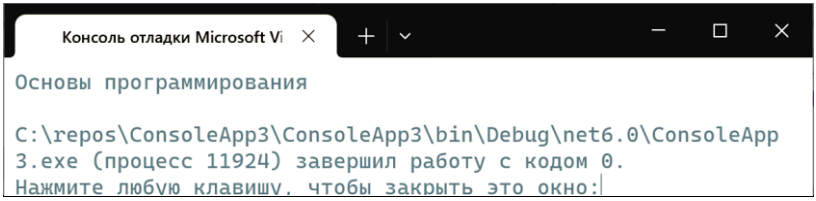


Рис. 5. Результат выполнения программы. Пример 0

Для работы с языком программирования C# в операционной системе Linux (Astra Linux, Ubuntu и т.п.) можно использовать текстовый редактор Visual Studio Code. Рассмотрим пример подготовки рабочего окружения на примере операционной системы Astra Linux. Скачать дистрибутив Visual Studio Code можно на официальном сайте Майкрософт [1]. Далее необходимо открыть скачанный файл с помощью установщика приложений (рис.6,7). Для установки пакета может потребоваться ввести пароль от учетной записи с правами суперпользователя.

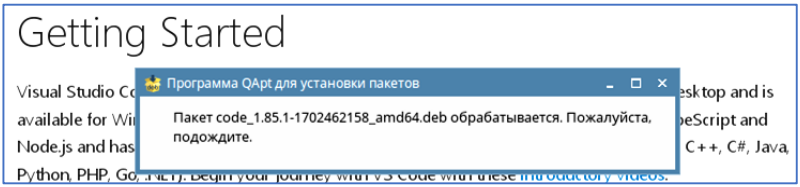


Рис. 6. Запуск установки пакета Visual Studio Code

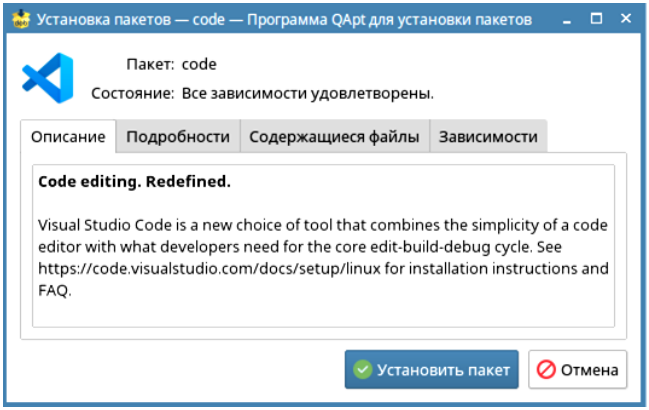


Рис. 7. Установка Visual Studio Code

После установки дистрибутива программа готова к первому запуску. После ее запуска спустя некоторое время она предложит установить русский языковой пакет, а также провести кастомизацию интерфейса. Для возможности использовать язык программирования C# во время разработки и отладки приложения в дальнейшем необходимо скачать специальное дополнение (рис. 8).

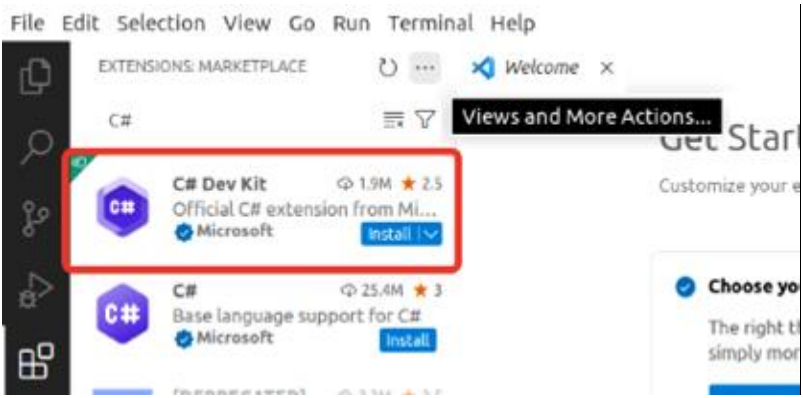


Рис. 8. Установка дополнений в VS Code для работы с C#

После этого редактор кода можно закрыть. Помимо этого, необходимо установить SDK для .NET. Для этого введите в терминале последовательно следующий набор команд:

```
wget https://dot.net/v1/dotnet-install.sh -O dotnet-install.sh
chmod +x ./dotnet-install.sh
./dotnet-install.sh --version latest
echo  'export PATH=$PATH:~/.dotnet' >> ~/.bashrc
echo  'export DOTNET_ROOT=~/.dotnet' >> ~/.bashrc
```

Для проверки корректности установки перезапустите терминал и выполните команду:

```
dotnet -- version
```

В результате выполнения этой команды должна отобразится установленная версия пакета библиотека .Net. После этого базовая подготовка закончена, можно создавать первый проект, для этого в терминале создайте папку, где будут храниться все ваши проекты. Для создания папок и навигации между ними необходимо использовать bash команды [2]. Для создания новой папки используется команда ***mkdir***, для перехода в любую папку на компьютере зная ее путь используется команда ***cd***. На рис. 9 представлен терминал ubuntu и создание папки для хранения всех проектов, для первого приложения и переход в папку с будущем проектом. После этого можно создавать свой первый проект при помощи команды.

```
dotnet new console
```

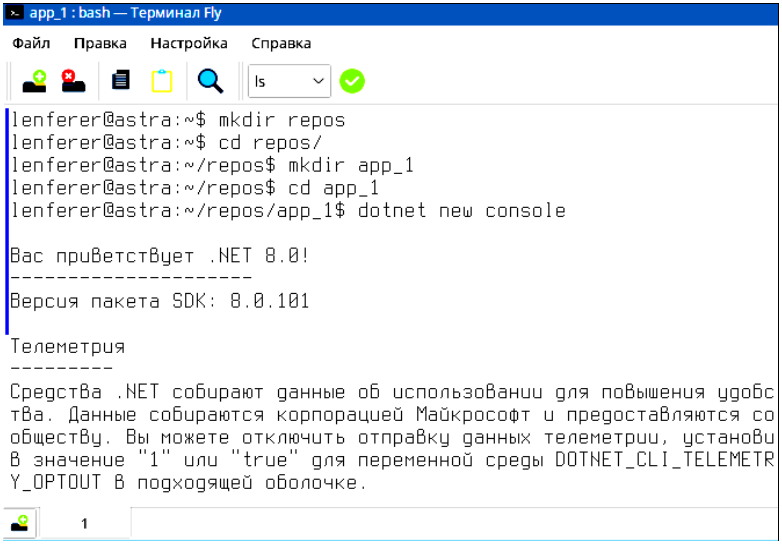


Рис. 9. Создание первого проекта

После этого в папке появятся новые файлы. Это значит, что проект консольного приложения создан и его можно открыть в редакторе кода VS Code в оконном режиме, для этого воспользуйтесь меню **«Файл (File) – Открыть папку (Open Folder)»** и выберите папку в который был создан проект (рис.10) либо воспользуйтесь горячими клавишами **«Ctrl+K, Ctrl+O»**.

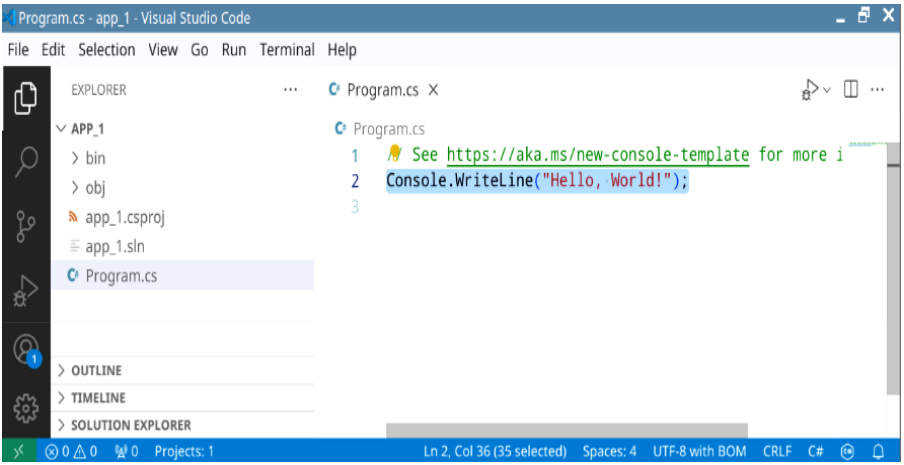


Рис. 10. Сгенерированный проект

Для того, чтобы скомпилировать проект, введите в терминале команду:
dotnet build
А для того, чтобы запустить программу:
dotnet run

Для того чтобы запустить программу в режиме отладки достаточно нажать **«F5»**, чтобы запустить программу без отладки необходимо нажать **«Ctrl+F5»**, так же это можно сделать через меню **«Выполнить (Run) – Запустить отладку (Start Debugging)»**. Отметим, что в режиме отладки программы есть возможность останавливать выполнение программы в любой момент используя точки останова. Для того чтобы добавить точку останова в редакторе кода, когда курсор находится на нужной строчке кода следует нажать F9, либо сделать это через меню **«Выполнить (Run) – Переключить точку останова (Toogle Breakpoint)»**. Процесс установки Visual Studio Code и настройки окружения будет аналогичным для любого дистрибутива Linux, который основан на Debian (Ubuntu, Linux Mint, Kali Linux, Pure OS и др.).

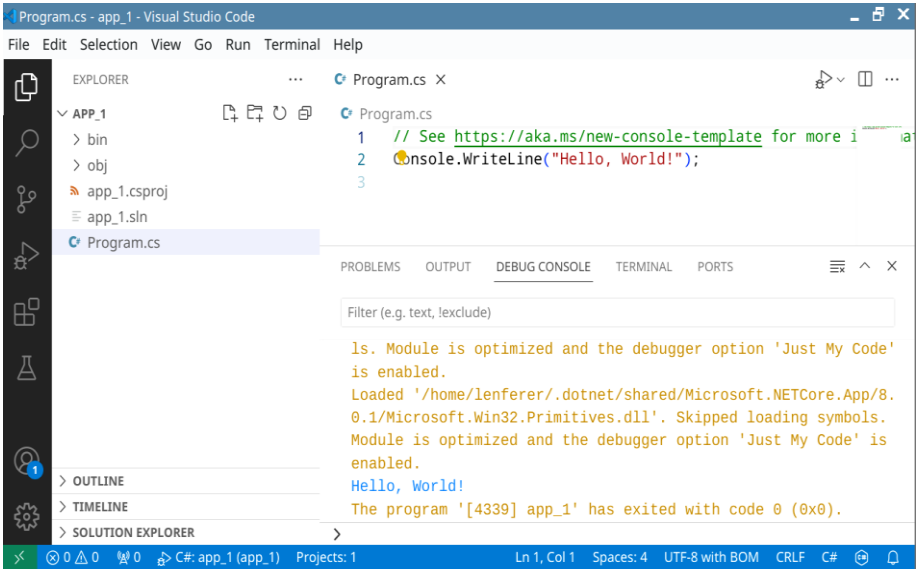


Рис. 11. Результат отладки программы (пример 0)

Типы данных и операнды

Ясно, что всякая программа, так или иначе, обрабатывает данные. Эти данные, как правило, размещаются в переменных. Каждая переменная имеет **имя, тип, размер и значение**. Основные типы с указанием размеров занимаемой памяти и областью значений приведены на рис. 12.

Ключевое слово или тип C#	Диапазон	Размер/точность
sbyte	От -128 до 127	8-разрядное целое число со знаком
byte	От 0 до 255	8-разрядное целое число без знака
short	От -32 768 до 32 767	16-разрядное целое число со знаком
ushort	От 0 до 65 535	16-разрядное целое число без знака
int	От -2 147 483 648 до 2 147 483 647	32-разрядное целое число со знаком
uint	От 0 до 4 294 967 295	32-разрядное целое число без знака
long	От -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	64-разрядное целое число со знаком
ulong	От 0 до 18 446 744 073 709 551 615	64-разрядное целое число без знака
float	От ±1,5 x 10−45 до ±3,4 x 1038	4 байта/6–9 знаков
double	от ±5,0 × 10−324 до ±1,7 × 10308	8 байт/15–17 знаков
decimal	от ±1,0 x 10-28 до ±7,9228 x 1028	16 байт/28-29 знаков
bool	true; false	16 разрядов
char	От U+0000 до U+FFFF	16 разрядов

Рис. 12. Основные типы данных в C#

Все представленные на рисунке типы данных – типы значений, кроме этого, в языке C# есть ссылочные типы – object, string, delegate, dynamic. О ссылочных типах данных поговорим позже, в первых лабораторных может понадобиться только - **«string»**, мы уже использовали его, когда создавали первое приложение, этот тип используется для хранения текстовой информации – строк.

Объявление типов переменных делается соответствующим служебным словом с последующим перечислением имен переменных:

int i, j, k, l, m; – перечисленные переменные будут целого типа и т.п. Причем, объявление типов можно совместить с присваиванием. Присваивание значения переменным делается с помощью команды присваивания «=». Значения для символьных переменных заключаются в одинарные кавычки, для строковых переменных в двойные кавычки: int i = 33; float f = 0.156F; double d = 0.5; char c = '5'; string s = "Привет!";

Обратите внимание, что для вещественных чисел в качестве разделителя используется точка, а не запятая – «0.5», так же для использования числе типа «float» необходимо в конце указывать литер «F», иначе компилятором эточисло будет восприниматься как double. Большинство типов данных – это типы данных, реализующие работу с числами, поэтому должны быть реализованы и опе-

рации работы с ними. Основные арифметические и логические операции, используемые в C++, приведены на рис. 13.

Арифметические операции	Обозначение	Логические операции	Обозначение
Сложение	+	Равно	= =
Вычитание	–	Не равно	!=
Умножение	*	Больше	>
Деление	/	Меньше	<
Вычисление остатка	%	Больше или равно	>=
Присваивание	=	Меньше или равно	<=
Отрицание	!	Логическое умножение	&&
		Логическое сложение	||

Рис. 13. Основные арифметические и логические операции в C#

Присваивание значений переменных, как уже было показано в примерах выше, делается командой присваивания «=»: `int i = 5;` В C# помимо привычных имеются и сокращенные формы записи арифметических операторов:

– «a+=3» эквивалентно «a=a+3»;
– «a–=3» эквивалентно «a=a–3»;
– «a*=3» эквивалентно «a=a*3»;
– «a/=3» эквивалентно «a=a/3»;
– «a%=3» эквивалентно «a=a%3».

Кроме того, существуют операции инкремента и декремента для более удобной записи увеличения или уменьшения переменных на 1, поскольку это часто встречаемая операция:

– «i++» эквивалентно «i=i+1»;
– «++i» эквивалентно «i=i+1»;
– «i--» эквивалентно «i=i-1»;
– «--i» эквивалентно «i=i-1».

Операции инкремента и декремента применяются к целым числам. Различие между постфиксной и префиксной формами иллюстрируется приведенным ниже примером. Например, пусть имеем переменные целого типа t и c. Если c=5, тогда при записи «**t = ++c + 6**» получим, что t равно 12.

Если же применить постфиксную форму (при том же c=5): « **t= c++ + 6**», то получим, что t=11, потому что начальное значение c используется для вычис-

ления выражения до того, как **c** увеличится на единицу операцией инкремента. Этот оператор эквивалентен следующим двум: «**t=c+6; ++c**».

Эти же правила применимы и к операции декремента --. Например, если c=5, то «**t = --c + 6**» даст значение 10 для t, в то время как «**t = 6 + c--**» даст значение 11 для переменной t.

Основные функции необходимые для выполнения лабораторных работ приводятся на рис. 14, более широкий набор всегда можно найти в официальной документации [3].

Вызов функции	Описание
Math.Abs(x)	Возвращает абсолютное значение числа x.
Math.Acos(x)	Возвращает угол, косинус которого равен указанному числу.
Math.Asin(x)	Возвращает угол, синус которого равен указанному числу.
Math.Atan(x)	Возвращает угол, тангенс которого равен указанному числу.
Math.Ceiling(x)	Возвращает наименьшее целое число, которое больше или равно заданному числу.
Math.Cos(x)	Возвращает косинус указанного угла.
Math.Exp(x)	Возвращает e, возведенное в указанную степень.
Math.Floor(x)	Возвращает наибольшее целое число, которое меньше или равно заданному числу.
Math.Log(x)	Возвращает натуральный логарифм (с основанием e) указанного числа.
Math.Log10(x)	Возвращает логарифм с основанием 10 указанного числа.
Math.Max(x, y)	Возвращает большее из двух чисел.
Math.Min(x, y)	Возвращает меньшее из двух чисел.
Math.Pow(x, y)	Возвращает указанное число, возведенное в указанную степень.
Math.Round(x)	Округляет значение до ближайшего целого значения.
Math.Sin(x)	Возвращает синус указанного угла.
Math.Sqrt(x)	Возвращает квадратный корень из указанного числа.
Math.Tan(x)	Возвращает тангенс указанного угла.

Рис. 14. Набор основных математических функций, реализованных в C#

Составной оператор «{...}» – действие состоит в последовательном выполнении содержащихся в нем операторов. Каждая фраза алгоритма заканчивается пустым оператором, который обозначается точкой с запятой «;». Выполнение пустого оператора не меняет состояния программы.

Обычно вначале файла с исходным кодом программы задают так называемые директивы препроцессору. Такие директивы начинаются с ключевого слова `using`, например, директива «`using System;`» подключает пространство имен, которое содержит фундаментальные и базовые классы, определяющие часто используемые типы значений и ссылочных данных, события и обработчики

событий, интерфейсы, атрибуты и исключения обработки. Для вывода на экран служит команда – `Console.Write("Сообщение для вывода");`

Для вывода сообщения и перевода курсора на новую строку можно использовать команду `Console.WriteLine("Сообщение для вывода");`

Для ввода значений переменных в программу в процессе ее исполнения служат команды `Console.Read()` и `Console.ReadLine()`, первая возвращает код ASCII введенного символа, вторая возвращает введенную строку.

В программе можно размещать комментарии, которым предшествует // (двойная дробная черта).

Поскольку языки предшественники языка программирования C# разрабатывались, прежде всего, для решения научно-технических задач, обладая преемственностью здесь имеется целый набор математических функций.

Для примера решение функция $y = \sqrt{a^2 + b^3}$ на языке программирования C# можно представить следующим образом:

```
double y;
int a = 5, b = 6;
y = Math.Sqrt(Math.Pow(a, 2) + Math.Pow(b, 3));
```

Линейные алгоритмы

Алгоритм – это конечный набор правил, который определяет последовательность операций для решения конкретного множества задач и обладает пятью важными чертами: конечность, предопределённость, ввод, вывод, эффективность [2]. Рассмотри подробно свойства, которым должен обладать алгоритм.

Дискретность – алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов. При этом для выполнения каждого шага алгоритма требуется конечный отрезок времени.

Результативность – завершение алгоритма определёнными результатами.

Определённость – в каждый момент времени следующий шаг работы однозначно определяется состоянием системы.

Понятность – алгоритм должен включать только те команды, которыедоступны исполнителю и входят в его систему команд.

Конечность – при правильно заданных начальных данных алгоритмдолжен завершать работу и выдавать результат за определённое число шагов.

Универсальность – алгоритм должен быть применим к разным наборам начальных данных.

Алгоритм называется линейным если выполнение команд выполняется последовательно друг за другом. Реализация алгоритма – исполняемая часть программы заключается в составной оператор, который обозначается фигурными скобками.

Программу, реализующую линейный алгоритм, исполняющуюся прямолинейно от начала и до конца – линейной программой. Рассмотрим простой пример.

Пример 1. Вычислить периметр и площадь прямоугольника, длина которого равна a, а ширина равна b.

Пример: a=3 м, b=4 м. Ответ: периметр = 14 м, площадь = 12 м^2.

Для начала составим алгоритм для решения задачи, опишем его:

– Начало алгоритма.

– Задать численное значение стороны a.

– Задать численное значение стороны b.

– Вычислить площадь S прямоугольника по формуле S=a*b.

– Вывести результат вычислений.

– Конец алгоритма.

Согласно [4] более корректно записывать алгоритма следующим образом:

Алгоритм А:

А.1 начало;

А.2 ввод a, b;

А.3 S ← a*b;

А.4 вывод S;

А.5 остановка;

Не менее популярным и более понятным способом записи алгоритмов является язык блок-схем – графическое описание алгоритмов. При составлении блок-схем алгоритмов необходимо руководствоваться – ГОСТ 19.701–90. Для описание данного алгоритма понадобятся только некоторые из блоков.

Блок «терминатор» – отображает выход во внешнюю среду и вход из внешней среды. При описании алгоритмов следует использовать для обозначения начала и конца алгоритма, внешнее использование или обработки исключений(ошибок). Нередко допускают ошибку и рисуют это блок в виде овала, стоит обратить внимание, что блок терминатор изображается в виде **суперэллипса**.

Блок «данные» – согласно ГОСТ, это блок отображает данные, когда носитель данных не определен, что позволяет использовать для обозначения операций связанные с вводном данных с клавиатуры или другого устройства ввода и вывода информации на экран монитора/печать и другие устройства вывода.

Блок «процесс» – используется для обозначения обработки данных любого вида, выполнения определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации, в блоке следует размещать максимально простые операции, понятные исполнителю.

Блок «комментарий» – используется для добавления пояснений и комментариев к блок-схеме, в целях повышения читаемости алгоритма.

Линии на блок-схемах отображают поток данных или управления. Для повышения удобочитаемости могут быть добавлены стрелки-указатели на линиях.

По умолчанию считается что поток данных или управления направлен сверху-вниз и слева-право, если предполагается что имеем место быть иное направление управления или потока данных, использование стрелок-указателей обязательно.

В блок-схемах следует избегать пересечения линий, для избегания двоякого трактования направления потока данных или управления, этого можно избежать, используя соединитель. Две или более линий могу быть объединены в одну линию, однако место объединения должно быть смещено. Линии к блокам должны подходить либо слева, либо сверху, а выходить справа и снизу. Линии должны выходить и входить в блоки только по центру стороны блока.

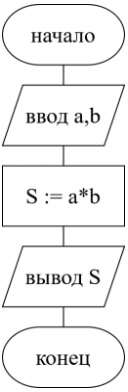


Рис. 15. Блок-схема алгоритма А

На псевдокоде алгоритм решения задачи 1 выглядит следующим образом:

```
begin
var a,b,S
read a,b
S:= a*b
write S
end
```

При описании алгоритмов словесным способом, рекомендуется придерживаться следующих простых правил:

1) обозначать каждый алгоритм заглавной буквой латинского алфавита и нумеровать все шаги, например: алгоритм А и шаги А1, А2, А3 и т.д.;

2) выполнение шагов всегда идет последовательно, если явно не указано другого;

3) последний шаг всегда – шаг «остановка», все ветви алгоритма должны приводить к нему;

4) использовать четкие и емкие синтаксические конструкции естественного языка для обозначения ввода-вывода, разветвления, повторения действий (организация циклов): «ввод …», «вывод …», «если …, то … иначе …», «пока … повторять … после …»;

5) Для обозначения математических и логических операций не использовать обозначения, которые можно трактовать не однозначно. Например, для операции присваивания использовать обозначения «:=» или «←» вместо «=».

После создания алгоритма решения задачи можно приступать к написанию программу, **именно в таком порядке**. Реализация алгоритма А на языке C# будет выглядеть следующим образом:

```
int a, b, S; // Объявление переменных a и b
// Вывод сообщения о необходимости ввода переменной a
Console.Write("Введите значение a = ");
a = int.Parse(Console.ReadLine()); // Ввод переменной a
Console.Write("Введите значение b = ");
b = int.Parse(Console.ReadLine());// Ввод переменной b
```

```
S = a * b; // Вычисление S
Console.WriteLine("S = {0}", S); // Вывод S
```

Разберем некоторые нюансы приведенного кода.

– Все используемые в программе переменные, необходимо обязательно объявить перед использованием;

– `int.Parse(Console.ReadLine())` – преобразует введённые с клавиатуры значение в целочисленный тип;

– `Console.WriteLine("S = {0}", S)` – при выводе вместо `{0}` подставляется переменная S, этот приме называется форматирование строк, вместо него можно использовать интерполяцию строк, тогда эту строчку можно заменить на «`Console.WriteLine($"S = {S}");`».

Проведем «построение» решения (напомним, что для этого достаточно нажать горячие клавиши **Ctrl+Shift+B**) и, если нет ошибок, запускаем на выполнение (**Ctrl+F5**). После запуска на исполнение получим результат как на рис. 16.

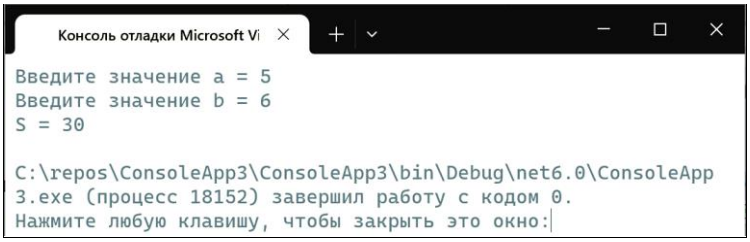


Рис. 16. Результат работы программы (пример 1)

Алгоритм с ветвлением

Разветвляющийся алгоритм – алгоритм, содержащий хотя бы одно условие, в результате проверки которого может осуществляться разделение на несколько альтернативных ветвей алгоритма.

Условный оператор на языке C# имеет вид:

```
if (условие) оператор_1;
else
оператор_2;
```

Обратите внимание, что в таком операторе отсутствует служебное слово `then` и условие обязательно заключается в скобки. `оператор_1` исполняется в случае истинности условия. `оператор_2` – в случае ложности условия. Кроме того, для реализации разветвляющихся алгоритмов в языке C# присутствует оператор выбора:

```
switch (a)
{
    case 1:
        Console.WriteLine("Выбор 1"); break;
```

```
    case 2:
        Console.WriteLine("Выбор 2"); break;
    default:
        Console.WriteLine("Выбор по умолчанию"); break;
}
```

Рассмотрим два примера, где может понадобиться разветвление алгоритма:

Пример 2. Составить программу для вычисления по заданному x значения функции $y = \begin{cases} \sin(x), x \le 0, \\ \cos(x), x > 0. \end{cases}$

Пример 3. Составить программу для вычисления по заданному x значения функции $y = \begin{cases} \sin(x), x = 1, \\ \cos(x), x = -1, \\ 0, \ x \ne 1, x \ne -1. \end{cases}$

Составим алгоритмы для решения задач Алгоритм B и Алгоритм C для примеров 2 и 3 соответственно и составим для них блок-схемы (рис. 17).

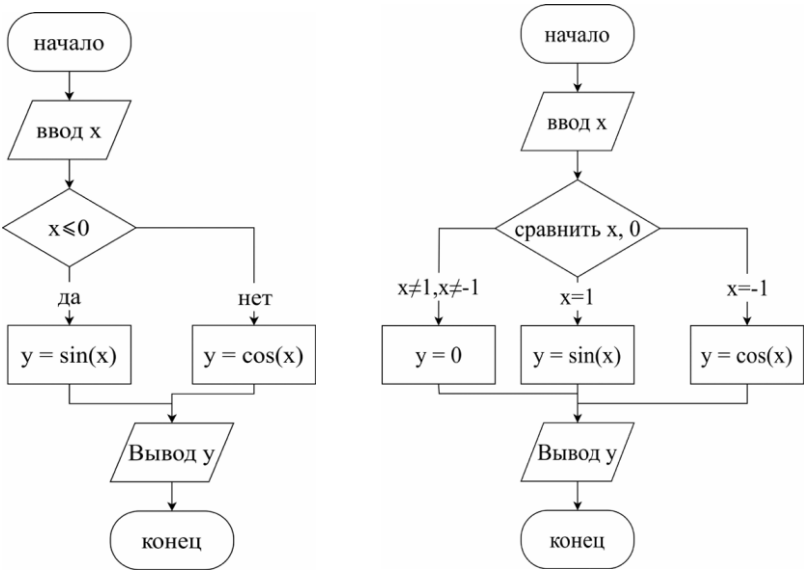


Рис. 17. Блок-схемы алгоритмов B и C

На блок-схемах условные операторы изображаются при помощи блока «решение». Блок «решение» – отображает решение или функцию переключательного типа, имеет всегда один вход и два или более выходов, из которых только один будет выполнен в зависимости от условий, обозначенных в теле этого блока.

Алгоритм B:

B.1 ввод x;

B.2 если x ≤ 0, то B.3, иначе B.4

B.3 y ← sin(x), перейти к B.5;

B.4 y ← cos(x);

B.5 вывод y, остановка;

Алгоритм C:

C.1 ввод x;

C.2 если x = 1, то C.3, иначе если x = -1, то C.4, иначе C.5C.3 y ← sin(x), перейти к C.6;

C.4 y ← cos(x), перейти к C.6;

C.5 y ← 0;

C.6 вывод y, остановка.

Составим программы реализующие разработанные алгоритмы, код программы примера 2 с использованием уловного оператора:

```
double x, y;
Console.Write("Введите значение x = ");
x = double.Parse(Console.ReadLine());
if (x <= 0)
   y = Math.Sin(x);
else
   y = Math.Cos(x);
Console.WriteLine($"y = {y}");
```

Результат работы программы представлен на рис. 18.

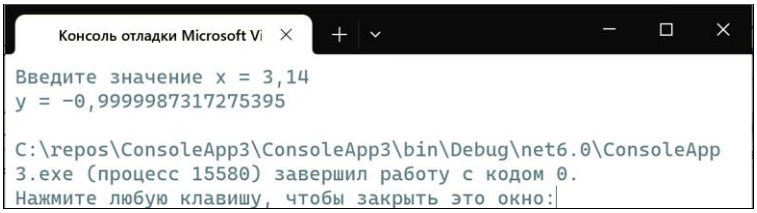


Рис. 18. Результат работы программы (пример 2)

Исходный код программы реализующий алгоритм C:

```
double x, y;
Console.Write("Введите значение x = ");
x = double.Parse(Console.ReadLine());
switch (x)
{
    case 1:
        y = Math.Sin(x);
        break;
```

```
    case -1:
        y = Math.Cos(x);
        break;
    default:
        y = 0;
        break;
}
Console.WriteLine($"y = {y}");
```

Результат работы программы, решающей пример 3 представлен на рис. 19.

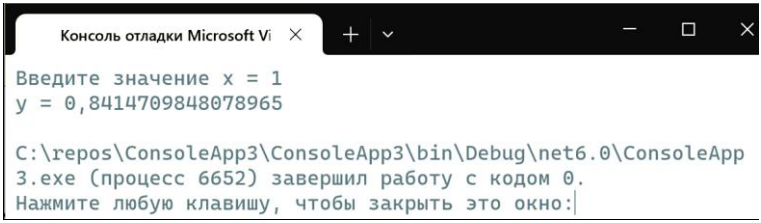


Рис. 19. Результат работы программы (пример 3)

В тех случаях, когда требуется выполнить несколько действий (в случае истинности или в случае ложности условия), то применяют составной оператор:

```
if (условие)
{ оператор_1_1; оператор_1_2; }
else
{ оператор_2_1; оператор_2_2;}
```

Пример 4. Составить программу, которая перераспределит заданные значения x, y так, что в x окажется большее значение, а в y меньшее.

```
double x, y, z;
Console.Write("Введите значение x = ");
x = double.Parse(Console.ReadLine());
Console.Write("Введите значение y = ");
y = double.Parse(Console.ReadLine());
if (x < y)
{
    z = x; x = y; y = z;
}
Console.WriteLine($"x = {x}"); Console.WriteLine($"y = {y}");
```

Обратите внимание, что здесь использована сокращенная форма записи условного оператора. Условие может быть и весьма сложным. Здесь допуска ется конъюнкция условий (одновременное выполнение условий) – условия связываются при помощи «**&&**», дизъюнкция условий (альтернативное выполнение условий) – обозначается «| |» и инверсия условий (обозначается знаком «**!**»).

Результат работы программы, решающей пример 4 представлен на рис. 20.

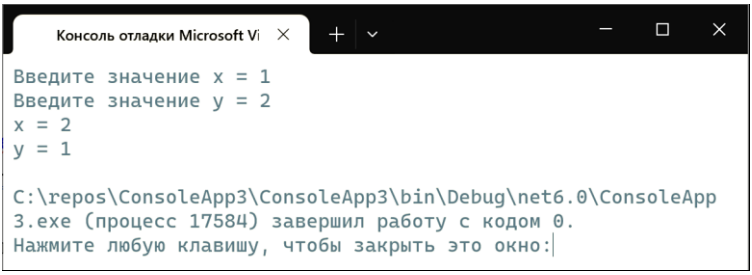


Рис. 20. Результат работы программы (пример 4)

Циклы

Цикл – управляющая конструкция, предназначенная для организации многократного выполнения какого-либо набора инструкций или команд. Циклы могут быть организованны как при помощи специальных операторов языка программирования, так и при помощи операторов сравнения и условного перехода.

Бесконечный цикл – цикл, в котором отсутствует условие выхода из цикла, либо условие выхода из цикла не достижимо в реальности.

Циклический алгоритм – алгоритм, предусматривающий многократное повторение одного и того же действия над новыми исходными данными.

Пример 5. Рассчитать для заданного *N* величину *N!*. Составим алгоритм для решения примера 5.

Алгоритм D:

D.1 ввод N;
D.2 p←1;
D.3 i ← 1;
D.4 p ←p*i; i←i+1;
D.5 если i<N перейти к D.4
D.6 вывод p, остановка;

В результате выполнения алгоритма D в переменной *p* будет находится значения факториала числа N, каждый раз после выполнения шага 4, мы увеличиваем значение переменной i и проверяем достигнул ли переменная i значения N. Такие переменные принято называть счетчиком цикла.

Счетчик цикла – переменная, контролирующая повторы выполнения циклов. Счетчики циклов изменяют своё значение при каждой итерации цикла, принимая уникальное значение для каждой отдельной итерации. Счетчик цикла используется для определения момента, когда цикл должен завершить свою работу. В ГОСТ [3] присутствует символ «подготовка данных» в произвольной форме (в ГОСТ нет ни пояснений, ни примеров), задает входные значения. Используется обычно для задания циклов со счетчиком (рис. 21).

Бывают ситуации, когда условиями окончания цикла нельзя представить в виде значения счетчика цикла. Тогда принято говорить о циклах с условием. Существует два вида таких циклов: с предусловием и постусловием, отличаются они тем в какой момент проверяется условие выполнения цикла. Для обозначения циклов с условием на языке блок-схем используется два вида обозначений для разных ситуаций, символы начала и конца цикла содержат имя и условие. Условие может отсутствовать в одном из символов пары. Расположение условия, определяет тип оператора, соответствующего символам на языке высокого уровня – оператор с предусловием или постусловием (рис. 22).

Рис. 21. Блок-схема цикла с параметром

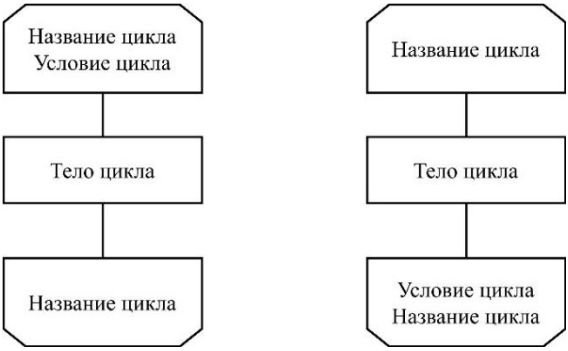


Рис. 22. Блок-схема цикла с условием

В примере 5 для решения задачи более корректно использовать цикл с параметром, поскольку точно известно какое количество итераций необходимо, однако допускаются конструкции с использованием цикла с условием как в алгоритме D. Алгоритм D частный случай цикла с постусловием. Поскольку проверка условия i < N происходит после выполнения тела цикла, если бы проверка происходила до, это был бы цикл с предусловием. Алгоритм D с использованием цикла с параметром будет выглядеть следующим образом:

D.1 Ввод N;
D.2 p←1;
D.3 Для i от 1 до N с шагом 1 повторять D.4, после D.5;
D.4 p ←p*i;
D.5 вывод p, остановка;

В языке программирования C# оператор `for` выполняет оператор или блок операторов, пока определенное логическое выражение равно значению `true`. Оператор `for` определяет разделы инициализатора, условия и итератора:

```
for (инициализатор; условие; итератор)
{
}
```

Например:

```
for (int i = 1; i < N; i++)
{
    p = p * i;
}
```

Таким образом, оператор `for` имплементирует цикл с параметром. В любой момент в блоке операторов `for` вы можете прервать цикл с помощью оператора `break` или перейти к следующей итерации в цикле с помощью оператора `continue`. Также можно выйти из цикла `for` с помощью операторов `goto`, `return` или `throw`.

Листинг программы, реализующей алгоритм D с использованием цикла с параметром, представлен ниже:

```
int N, p = 1;
Console.Write("Введите значение N = ");
N = int.Parse(Console.ReadLine());
for (int i = 1; i < N; i++)
{
    p = p * i;
}
Console.WriteLine($"Факториал N = {p}");
```

Результат работы программы пример 5 представлен на рис. 23.

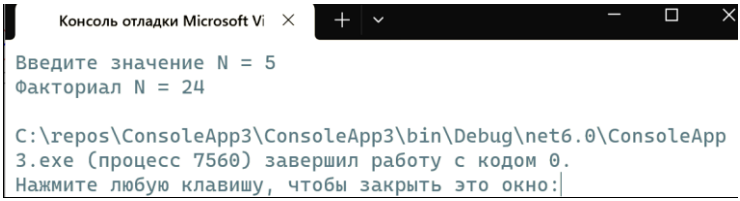


Рис. 23. Результат работы программы (пример 5)

Оператор `while` выполняет оператор или блок операторов, пока определенное логическое выражение равно значению `true`. Так как это выражение оценивается перед каждым выполнением цикла, цикл `while` выполняется ноль или несколько раз. В любой точке блока операторов `while` можно разорвать цикл с помощью оператора `break`. Также можно выйти из цикла `while` с помощью операторов `goto`, `return` или `throw`. Оператор `while` имплементирует цикл с предусловием в языке программирования C#.

Пример использования оператора `while`:

```
while (n > 5)
{
    Console.WriteLine(n); n--;
}
```

Оператор `do-while` выполняет оператор или блок операторов, пока определенное логическое выражение равно значению `true`. Так как это выражение оценивается после каждого выполнения цикла, цикл `do-while` выполняется один или несколько раз. Это отличает его от цикла `while`, который выполняется от нуля до нескольких раз. В любой точке блока операторов `do` можно разорвать цикл с помощью оператора `break` или с помощью операторов `goto`, `return` или `throw`.

Пример использования оператора `do-while`:

```
do
{
    Console.WriteLine(n); n--;
} while (n > 5);
```

Внутри циклов можно организовывать и вложенные циклы, причем число вложений ничем не ограничено. Рассмотрим на примере.

Пример 6. Получить таблицу умножения в виде таблицы Пифагора. Блок-схема алгоритма для решения задачи 5 представлена на рис. 23. Результат работы программы пример 6 представлен на рис. 24. Листинг программы, реализующей алгоритм решения представлен ниже:

```
for (int i = 1; i <= 9; i++)
{
    for (int j = 1; j <= 9; j++)
    {
        int k = i * j;
        if (k >= 10)
            Console.Write($" {k}");
        else
            Console.Write($" {k}");
    }
    Console.WriteLine();
}
```

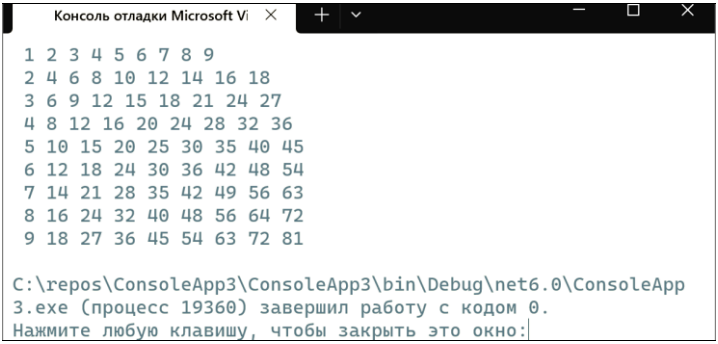


Рис. 24. Результат работы программы (пример 6)

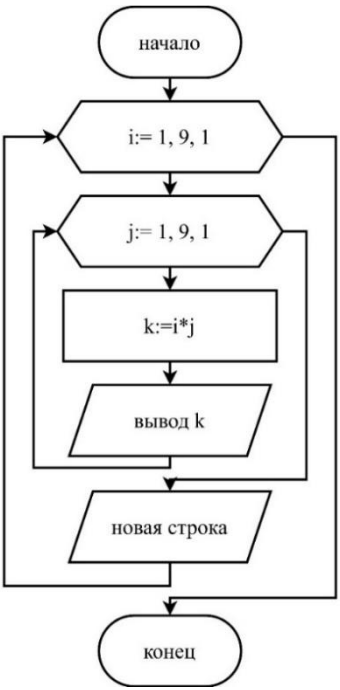


Рис. 25. Блок-схема алгоритма решения примера 6

Обратите внимание при выводе чисел, проводится проверка двузначное число или нет и в зависимости от этого k выводиться либо с одним, либо с двумя предшествующими пробелами, но на блок-схеме это не отражено, поскольку это имеет незначительное значение для самого алгоритма, а относится непо-

средственно к стилизации вывода, нюансы, связанные с форматом выводом, можно опускать при описании алгоритмов.

Рассмотрим пример, когда актуально использовать циклы с условием.

Пример 7. Рассчитать сумму вида: 1–1/2+1/3–1/4+... с заданной точностью *E*. Иначе говоря, проводить суммирование, пока очередное слагаемое является большим *E*.

Блок-схема алгоритма решения примера 7 с использованием цикла с предусловием представлена на рис. 26.

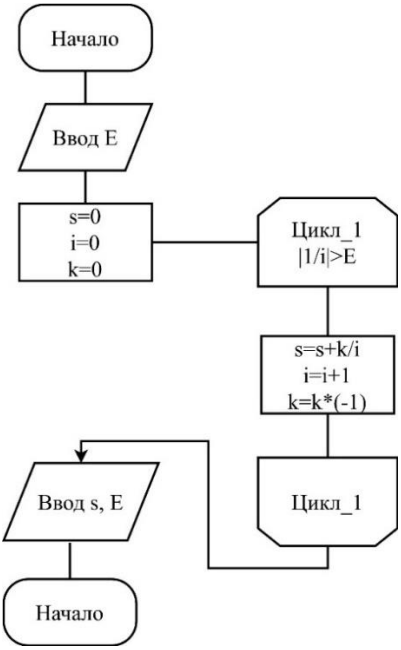


Рис. 26. Блок-схема алгоритма решения примера 7

Листинг программы решения примера 7 представлен ниже:

```
int i = 1, k = 1;
double s = 0, E;
Console.Write("Введите точность E = ");
E = double.Parse(Console.ReadLine());
while (1.0 / i > E)
{
    s = s + (double)k / i; i++;
    k = k * (-1);
}
Console.WriteLine($"Сумма последовательности равна {s} с точностью
{E}");
```

Результат работы программы пример 7 представлен на рис. 27.

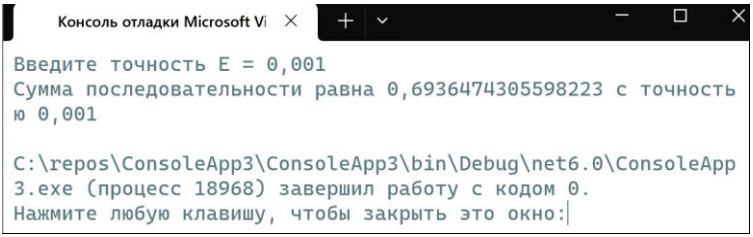


Рис. 27. Результат работы программы (пример 7)

Для изменения порядка работы в цикле в языке C# применяются два оператора: `break` (для досрочного прерывания цикла) и `continue` – для пропуска следующих за ним операций с переходом в конец цикла, но из цикла выхода не производится. Рассмотрим пример, где это может быть необходимо.

Пример 8. Среди K первых членов последовательности вида: 1, 1+1/2, 1+1/2+1/3, ... найти первый, больший заданного числа A. Если же его нет (среди первых K членов), то вывести об этом сообщение.

Листинг программы решения примера 8 представлен ниже:

```
int i = 1, k;
double s = 0, A;
Console.Write("Введите количество k = ");
k = int.Parse(Console.ReadLine());
Console.Write("Введите A = ");
A = double.Parse(Console.ReadLine());
while (i <= k)
{
    s = s + 1.0 / i; if (s > A) break; i++;
}
if (i > k)
    Console.WriteLine("Такого числа нет");
else
    Console.WriteLine($"При i = {i} первое число которое больше A
это {s}");
```

Результаты работы программы примера 8 представлен на рис. 28.

Последним для рассмотрения циклом является цикл `foreach`:

```
foreach (тип переменная in коллекция) { }
```

Данный цикл, как видно из его конструкции, предназначен для последовательного прохода по элементам массива или подобной структуры. Важно отметить, что в данном цикле невозможно пройтись по массиву в обратном порядке, либо использовать не единичный шаг. Переменной в данном случае будет элемент массива. Рассмотрим пример использования этого вида цикла.

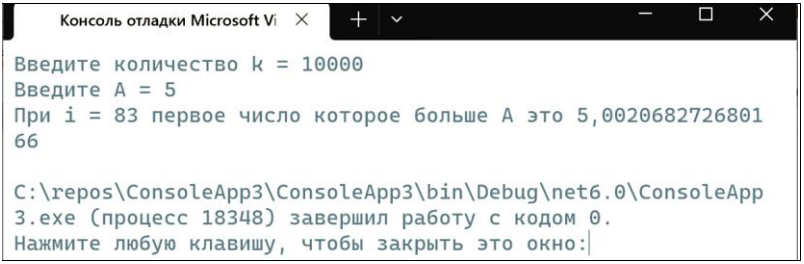


Рис. 28. Результат работы программы (пример 8)

Пример 9. Создать массив чисел от 1 до 5 и массив марок автомобилей, вывести элементы обоих массивов не прибегая к циклам с параметром или условием.

Листинг программы решения примера 9 представлен ниже:

```
int[] numbers = { 1, 2, 3, 4, 5 };
foreach (int element in numbers)
{
    Console.WriteLine(element);
}
Console.WriteLine();
string[] cars = { "Lada", "Honda", "Toyota" };
foreach (string element in cars)
{
    Console.WriteLine(element);
}
```

Результаты работы программы примера 9 представлен на рис. 29.

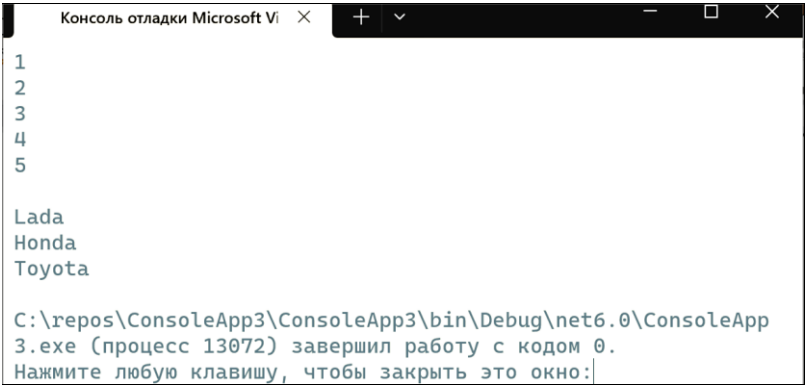


Рис. 29. Работа цикла Foreach (пример 9)

Контрольные вопросы

1. Что такое цикл?
2. Какие разновидности циклов бывают?
3. В каких ситуациях используются те или иные разновидности циклов?
4. Как циклы обозначаются на блок-схемах?
5. Какие операторы используются для пропуска итераций и выхода из цикла?

Массивы

Массив – это упорядоченный набор величин, принадлежащих одному типу данных, обозначаемых одним именем. Данные, являющиеся элементами массива, располагаются в памяти компьютера в определенном порядке, который задается индексами (порядковыми номерами элементов массива).

Размерность массива – это количество индексов, необходимое для однозначной адресации элемента в рамках массива. Форма или структура массива – сведения о количестве размерностей и размере массива для каждой из размерностей. Нуль-мерный массив называется скаляром, одномерный – вектором, двумерный – матрицей.

В C# массив, как и любая переменная, должен быть объявлен. Делается это с помощью служебного слова, указывающего тип с квадратными скобками, затем указывается имя массива, ставится знак равенства и ключевые слова `new`, далее снова указывается тип и в квадратных скобках размерность массива.

```
int[] array1 = new int[5];
```

Кроме того, можно сразу задать и значение элементов массива, сделать это можно одним из следующих способов:

```
int[] array2 = { 1, 2, 3, 4, 5};
int[] array3 = new int[5] { 1, 2, 3, 4, 5 };
```

Заметим, что индексы массива ведут счет с нуля, поэтому запись вида: `int[] array1 = new int[5];` означает, что резервируется память для 5 чисел типа `int` с именем `array1` и порядковыми номерами от 0 до 4. Рассмотрим небольшой пример.

Пример 10. Ввести одномерный массив из 5 целых чисел. Вывести четные по порядковому номеру элементы этого массива.

Листинг программы для решения данного примера приведен ниже:

```
int[] numbers = new int[5]; for (int i = 0; i < 5; i++)
{
    Console.Write($"Введите {i + 1} элемент массива: ");
    numbers[i] = int.Parse(Console.ReadLine());
}
for (int i = 1; i < 5; i += 2)
{
    Console.WriteLine($"{i + 1} элемента массива ра-
вен:{numbers[i]}");
}
```

Обратите внимание при выводе индексов массива мы указывали i+1, т.к. в памяти нумерация элементов массива начинается с 0! Результаты работы программы примера 10 представлен на рис. 30.

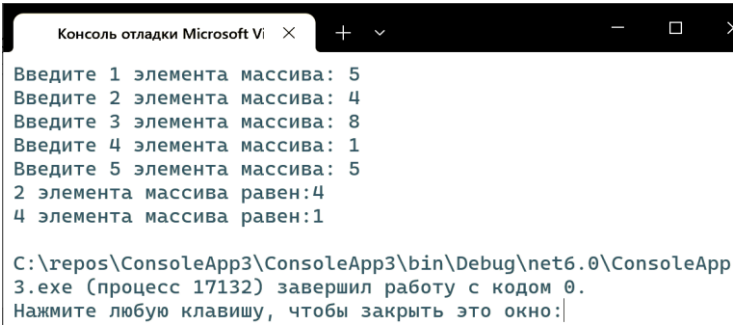


Рис. 30. Результат работы программы (пример 10)

При работе с массивами необходимо внимательно следить за тем, чтобы не выходить за их объявленные границы. Компилятор не предупреждает об этой ошибке! Попытка ввести больше элементов, чем описано, приведет к неверным результатам и необработанному исключению – «Необработанное исключение: System.IndexOutOfRangeException: Индекс находился вне границ массива.», а попытка вывести – выведет случайный результат, находящийся в памяти.

Попробуйте в предыдущем примере описать массив `numbers[5]` (напомним, что при этом максимальный элемент `numbers[4]`, т.к. счет элемент ам идет с нуля), и вы не получите последнего числа (точнее получите какое- то случайное число при выводе).

Еще раз обратим внимание, что работа с целыми переменными требует осторожности. Деление целого на целое дает в результате целое! Если требуется получать «правильный», вещественный, результат следует использовать преобразование. Если же требуется получать остаток от деления целого на целое, то применяют операцию %. Проиллюстрируем использование операции деления и вычисления остатка на простом примере.

Массивы могут быть и многомерными, например, двумерными. В этом случае размерности записываются через запятую:

`int[,] numbers = new int[5, 4];` – целочисленный массив из 5 строк и 4 столбцов.

Напомним, что в математике двумерный массив принято называть матрицей, при этом матрица, у которой число строк равно числу столбцов называется *квадратной*. Если у квадратной матрицы элементы симметрично относительно главной диагонали равны ($a_{21} = a_{12}$ и т.д.), то ее называют *симметричной*. Напомним, что *главная диагональ* матрицы содержит элементы с одинаковыми индексами: a_{11}, a_{22} и т.д. *Побочная диагональ* – вторая диагональ матрицы; *верхний треугольник* – элементы над главной диагональю (включая и элементы на диагонали); *нижний треугольник* – элементы под главной диагональю (включая и элементы на диагонали).

В случае, когда элементы верхнего и нижнего треугольника совпадают, то говорят, что матрица *симметрична*. Рассмотрим простой пример.

Пример 11. Построить квадратную матрицу N на N, заполнить ее, вывести, а экран, после чего вывести числа главной диагонали.

Листинг программы для решения данного примера приведен ниже:

```
int N;
N = int.Parse(Console.ReadLine());
int[,] numbers = new int[N, N];
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        Console.Write($"Введите элемента массива с индексами {i +
1}-{j + 1}: ");
        numbers[i, j] = int.Parse(Console.ReadLine());
    }
}
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        if (numbers[i, j] >= 10)
            Console.Write($" {numbers[i, j]}");
        else
            Console.Write($" {numbers[i, j]}");
    }
    Console.WriteLine();
}
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        if (i == j)
            Console.WriteLine($"Элемент массива с индексами {i}-{j}
равен: {numbers[i, j]}");
    }
}
```

Результат работы программы 11 представлен на рис. 31.

В случае если необходимо сгенерировать случайное число используйте команду:

```
int a = new Random().Next(0, 10);
```

где 0 – минимальное число, 10 максимальное число.

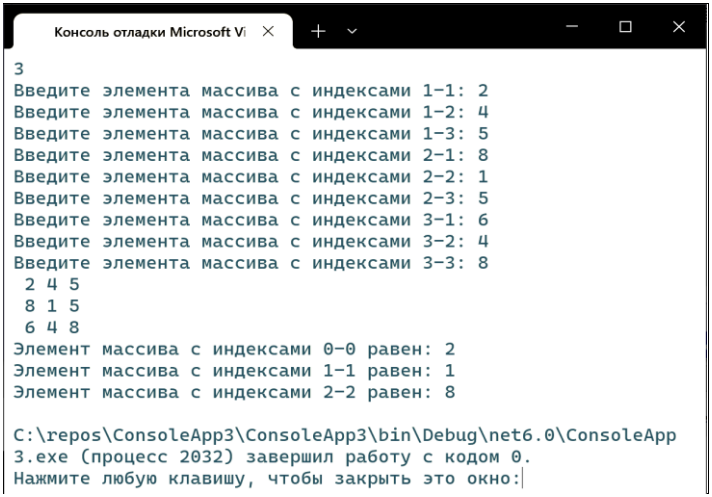


Рис. 31. Результат работы программы (пример 11)

Функции и процедуры

Функция – именованный фрагмент программного кода, к которому можно обратиться из другого места программы. Функция может принимать на вход параметры и должна возвращать некоторое значение, возможно пустое. Входные параметры могут быть обязательными и необязательными. Функции, которые возвращают пустое значение или не возвращают его (в зависимости от языка программирования) принято называть процедурами. Объявление функции, кроме имени, содержит список имён и типов передаваемых параметров, а также, тип возвращаемого функцией значения.

В Си-подобных языках программирования процедуры еще называют void-функциями, поскольку вместо указания типа возвращаемого значение указывается ключевое слово void, которое означает, что функция не возвращает значение. Для того, чтобы использовать ранее определённую функцию, необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию параметры.

В C# можно создавать пользовательские функции. В языке C# и фреймворке .Net Framework, которые мы до сих пор использовали имеется огромное количество встроенных библиотек с набором функций, однако их не всегда достаточно для решения той или иной задачи. Каждая библиотека может содержать в себе одно или несколько пространств имен. Для подключения пространства имен используется запись вида `using System.IO;` где «`using`» это ключевое слово, а «`System.IO`» – название пространства имен, содержащего функции для работы с файловой системой.

Функция определяет локальную область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции. После выполнения тела функции локально переменные функции будут недоступны. Внутри функции можно вызывать другие функции. Существует возможность вызвать функцию внутри самой себя –рекурсивный вызов, а сам процесс последовательных вложенных друг в друга вызовов функций называют рекурсией.

Пример описания функции представлен ниже:

```
int sum(int a, int b)
{
    return a + b;
}
```

Такое описание функции делается вне тела другой функции. В конце описания (после фигурной скобки) точка с запятой не ставится. Функция может иметь только одно возвращаемое значение, которое определяется служебным словом `return`, которое размещается в конце функции (перед закрывающей фигурной скобкой). Внутри функции могут быть описаны свои локальные переменные.

Пример 12. Вычислить площадь пятиугольника, у которого известны длины сторон и двух диагоналей (рис. 32).

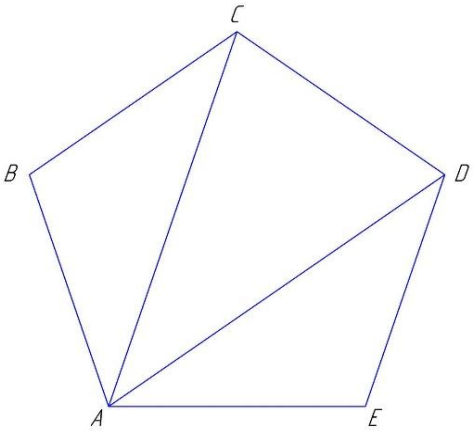


Рис. 32. Пятиугольник с двумя диагоналями

Очевидно, что площадь такого пятиугольника – сумма площадей треугольников, для каждого из которых известны длины сторон. Для вычисления площади каждого треугольника (условно обозначим его стороны за x, y, z) применим формулу Герона: $S = \sqrt{p(p-x)(p-y)(p-z)}$, где $p = (x + y + z)/2$.

Решим эту задачу с использованием функции, для этого составим алгоритмы нахождения площади многоугольника и нахождения треугольника (в виде функции) (рис. 33).

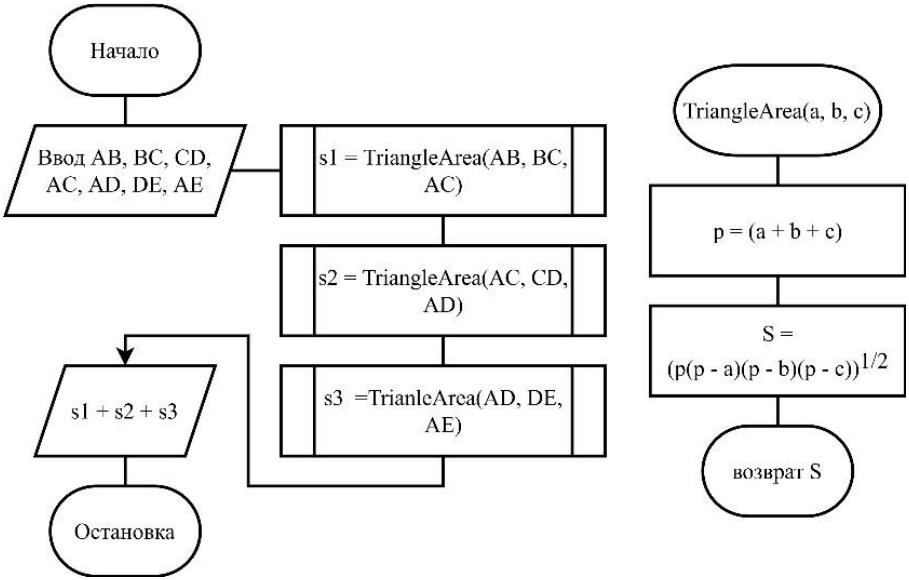


Рис. 33. Блок-схема алгоритма решения примера 12

Листинг программы для решения данного примера приведен ниже:

```
double AB, BC, AC, CD, AD, DE, AE, s1, s2, s3, s;
Console.Write("Введите AB: ");
AB = double.Parse(Console.ReadLine());
Console.Write("Введите BC: ");
BC = double.Parse(Console.ReadLine());
Console.Write("Введите AC: ");
AC = double.Parse(Console.ReadLine());
Console.Write("Введите CD: ");
CD = double.Parse(Console.ReadLine());
Console.Write("Введите AD: ");
AD = double.Parse(Console.ReadLine());
Console.Write("Введите DE: ");
DE = double.Parse(Console.ReadLine());
Console.Write("Введите AE: ");
AE = double.Parse(Console.ReadLine());
s1 = TriangleArea(AB, BC, AC);
s2 = TriangleArea(AC, CD, AD);
s3 = TriangleArea(AD, DE, AE);
```

```
s = s1 + s2 + s3;
Console.WriteLine($"Площадь многоугольника ABCDE равна {s}");

double TriangleArea(double a, double b, double c)
{
    double p, S;
    p = (a + b + c) / 2;
    S = p * (p - a) * (p - b) * (p - c);
    S = Math.Sqrt(S);
    return S;
}
```

Результат работы программы 12 представлен на рис. 34.

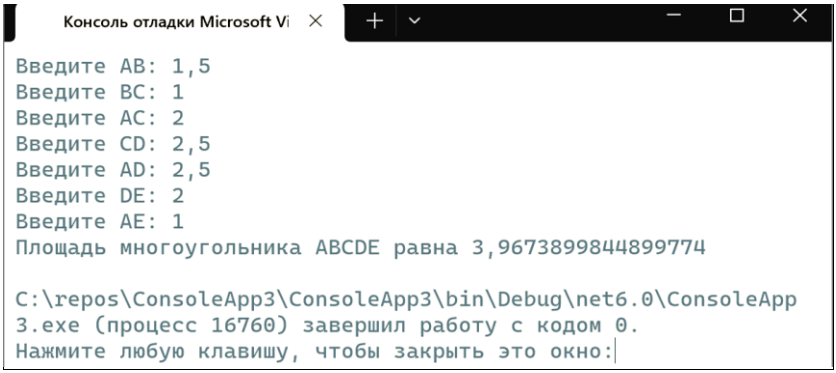


Рис. 34. Результат работы программы (пример 12)

Рекурсия – определение части функции через саму себя, то есть это функция, которая вызывает саму себя, непосредственно в своём теле или косвенно через другую функцию. Количество вложенных вызовов функции называется глубиной рекурсии. Использование рекурсии позволяет осуществлять повторяющиеся вычисления и/или действия без явных повторений или организации циклов в программе, однако также как у цикла с условием должно быть четко оговорено условие остановки рекурсивных вызовов, для предотвращения зацикливания программы. Вторая составляющая любой рекурсивной функции это условие продолжения (шаг рекурсии), где идет рекурсивное обращение. Самый распространённый и понятный пример использования рекурсии – это функция нахождения факториала. Напомним, что факториал любого натурального числа – это перемножение этого числа на факториал числа меньшего на единицу, причем факториал 1 равен 1. Таким образом, здесь «факториал 1 равен 1» – это условие остановки рекурсивных вызовов (базовый случай); «факториал любого натурального числа – это перемножение этого числа на факториал числа меньшего на единицу» – это шаг рекурсии.

Пример 13. Вычислить факториал числа, введенного с клавиатуры с использованием рекурсивного подхода решения задачи.

На языке программирования C# функция нахождения факториала будет выглядеть следующим образом:

```
int Factorial(int n)
{
  // базовый случай
  if (n == 1)
      return 1;
  //шаг рекурсии
  return n * Factorial(n - 1);
}
```

Использовать рекурсивные функции в программном коде нужно использовать так же, как обычные функции:

```
int n, nFactorial;
Console.Write("Введите N: ");
n = int.Parse(Console.ReadLine());
nFactorial = Factorial(n);
Console.WriteLine($"N! = {nFactorial}");
```

Результаты работы программы примера 13 представлен на рис. 35.

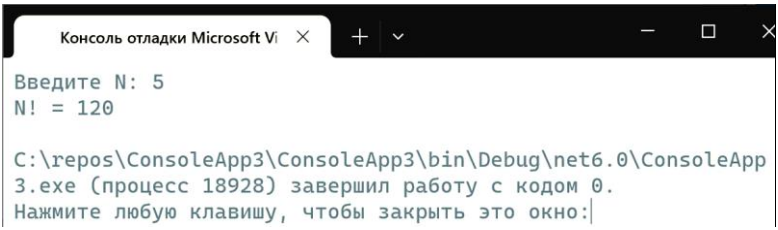


Рис. 35. Результат работы программы (пример 13)

Чтобы составить алгоритм с использование рекурсивного подхода необходимо ответить на два вопроса: Какой базовый случай в задаче? Какой шаг рекурсии или рекурсивное условие?

В нашем примере с нахождением факториала базовым случаем является условие, когда n=1, так как мы знаем, что 1!=1. Шагом рекурсии является вычисление n! = (n–1)! * n.

Блок-схема рекурсивной функции нахождения факториала представлена на рис. 36.

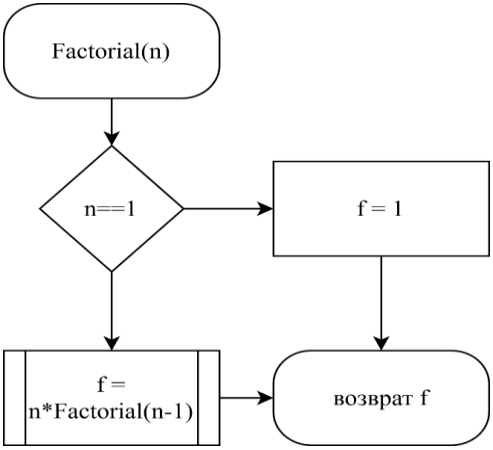


Рис. 36. Блок алгоритма нахождения факториала
с использованием рекурсивного подхода

Все задачи, которые мы до этого решали итеративно при помощи циклов можно решит также при помощи рекурсии, например нахождение суммы ряда: 1 + 2 + 3 + … + n. Здесь базовый случай – это сумма при n = 1 равно 1, шаг рекурсии – сумма n в общем случае равно n + сумма от n–1:

```
int Sum(int n)
{
  // базовый случай
  if (n == 1)
      return 1;
  //шаг рекурсии
  return n + Sum(n - 1);
}
```

Контрольные вопросы

1. Что такое функция?
2. Чем функция отличается от процедуры?
3. Зачем нужны функции?
4. Что такое рекурсия?
5. На какие два вопроса нужно ответить, чтобы решить задачу рекурсивным методом?
6. Каким блоком обозначаются функции на блок-схемах?

Обработка символов

Для описания символьных переменных служит тип char, соответственно массив символов можно объединить в строку. Для представления строк служит тип string.

Но для начала рассмотрим работу просто с отдельными символами. Значение символьной переменной может быть задано как с помощью команды присваивания (обратите внимание, что отдельный символ заключается в апострофы), так и с помощью команды ввода.

```
char a = '!';
do
{
    a = Console.ReadKey().KeyChar;
} while (a != '.');
```

Символы в также имеют коды. Для получения цифрового кода символа достаточно применить операцию перевода в целые: (int) c, где c – символьная переменная. Для получения по коду самого символа достаточно применить операцию по переводу целых в символьные (char) k, где k – целое число (код символа).

Пример 14. Клавиатурный тренажер.

В рассмотренном ниже примере будем также использовать датчик случайных чисел, который мы уже использовали ранее. Наиболее употребительные символы имеют коды в интервале от 33 до 122, поэтому в программе предусмотрено дополнительно ограничение на получение случайных чисел из интервала от 33 до 122. Здесь мы также впервые использует команду goto – метка. Метка задается, как и любой идентификатор (название метки начинается с буквы) и не требует описания. Признаком метки является двоеточие сразу после нее:

```
Random rand = new Random();//инициализация ГПСЧ
int kod;
char neededKey, pressedKey;
Console.WriteLine($"Вводите символы которые видите на экране");
while (true)
{
    kod = rand.Next(33, 122);
    neededKey = (char)kod;
inputKey:
    Console.Write(neededKey);
    pressedKey = Console.ReadKey().KeyChar;
    Console.Write('\b');
    if (neededKey != pressedKey)
    {
        Console.WriteLine("\nОшибка!");
        goto inputKey;
    }
}
```

Результаты работы программы примера 14 представлен на рис. 37.

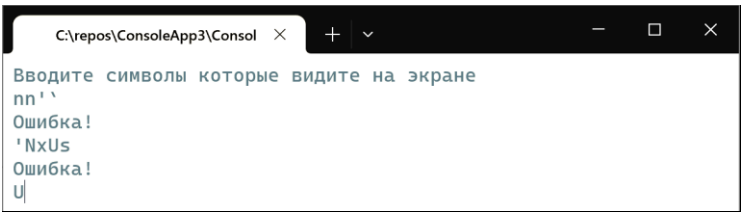


Рис. 37. Клавиатурный тренажер (пример 14)

Также в этом примере используются специальные символы «\b» «\n» - так называемые управляющими последовательностями символов или управляющими символами. Управляющие символы – это символы, встраиваемые в текст, для управления форматом, цветом и другими опциями вывода в текстовом терминале. Здесь использовались символы возврата на один символ назад (этот символ отправляется когда в любом редакторе вы нажимаете клавишу «Backspace» на клавиатуре) и символ перевода на новую строку. Таблица основных управляющих символов представлена ниже.

Таблица 1

Список основных управляющих символов

\'	Одинарная кавычка
\"	Двойная кавычка
\?	Вопросительный знак
\\	Обратная косая черта
\0	Нулевой символ
\a	Звуковой сигнал
\b	Символ возврата на один символ назад («забой»)
\f	Перевод страницы - новая страница
\n	Перевод строки - новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция

Пример 15. Выведите коды всех символов русского языка в виде таблицы.

В рассмотренном ниже примере будем также использовать датчик случайных чисел, который мы уже использовали ранее. Наиболее употребительные символы имеют коды в интервале от 33 до 122:

```
char[] rusalf = "АаБбВвГгДдЕеЁёЖжЗзИиЙйКкЛлМмНнОоПпРрСсТ-тУуФ-
фХхЦцЧчШшЩщЪъЫыЬьЭэЮюЯя".ToCharArray();
int code;
```

```
for (int i = 0; i < rusalf.Length; i++)
{
    code = (int)rusalf[i];
    Console.Write("{0,10}", $"{rusalf[i]} - {code}");
    if ((i + 1) % 4 == 0)
        Console.WriteLine();
}
Console.WriteLine();
```

Результаты работы программы примера 15 представлен на рис. 38.

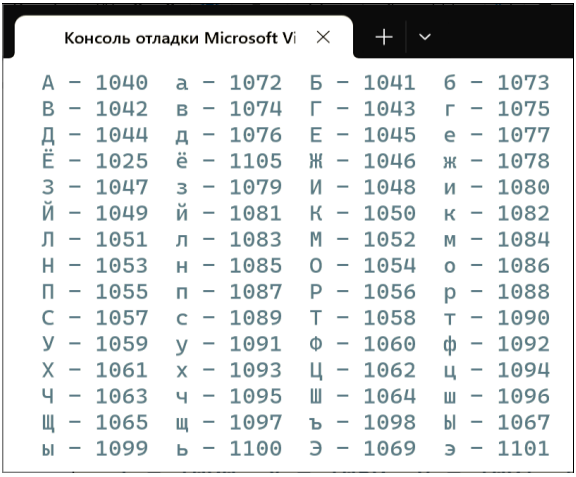


Рис. 38. Коды символов русского алфавита (пример 15)

Здесь в массиве с типом данных char хранятся все буквы русского алфавита, мы поочередно перебираем их получаем код символа и выводим его. Вывод осуществляется столбиками по 4 позиции.

Для ввода и вывода масссива символов с клавиатуры можно использовать следующие команды:

```
char[] rusalf;
rusalf = Console.ReadLine().ToCharArray();
Console.WriteLine(rusalf);
```

Работа с массивами символов ничем не отличается от работы с массивами других типов данных, их также можно сортировать пока какому-либо признаку, выводить при помощи циклов и т.д.

В C# есть специальный тип для хранения строк string, с переменными этого типа можно работать как с обычными переменными сразу задавая и изменяя им значения:

```
string a = "Привет";
a += " как дела";
Console.WriteLine(a);
```

Однако необходимо понимать, что в памяти строки хранятся аналогично массивам символов, и каждый раз, когда добавляется хотя бы один символ в строку в памяти создается новый массив из символов.

Наряду с управляющими символами, в языке программирования C# также имеются символы форматирования. Они в свою очередь предназначены для преобразования чисел.

Таблица основных символов форматирования представлена ниже.

Таблица 2

Символы для форматирования числовых данных

C или c	Используется для форматирования денежных значений
D или d	Используется для форматирования десятичных чисел
E или e	Используется для экспоненциального представления
F или f	Используется для форматирования чисел с точкой
G или g	Представляет общий формат
N или n	Используется для базового числового представления (с запятыми)
X или x	Используется для шестнадцатеричного представления чисел

Данные символы добавляются как суффиксы в строках или консоли при форматированном вводе-выводе. Форматируем одно и тоже число по-разному.

Пример 16. Выведите число 12345 во всех известных вам форматах представления числовых данных.

Листинг кода для примера 16 представлен ниже:

```
Console.WriteLine("Рассмотрим число 12345.");
Console.WriteLine("Формат денежный - {0:C}", 12345);
Console.WriteLine("Формат десятичный - {0:d9}", 12345);
Console.WriteLine("Формат с фиксированным разделителем - {0:f3}",
12345);
Console.WriteLine("Формат с разделением разрядов - {0:n}", 12345);
Console.WriteLine("Формат экспоненциальный - {0:E}", 12345);
Console.WriteLine("Формат шестнадцатиричный - {0:X}", 12345);
Console.WriteLine("Формат общий - {0:g}", 12345);
```

Результат работы программы примера 16 представлен на рис. 39.

Форматирование самой консоли в языке программирования C# происходит не с помощью символов, а с помощью специальных переменных и функций, рассмотрим пример кода ниже (**пример 17**):

```
foreach (char symbol in "Основы программирования")
{
    // Изменим цвет фона за текстом
    if ((int)symbol% 2 == 0)
```

```
            Console.BackgroundColor = ConsoleColor.Red;
        else
            Console.BackgroundColor = ConsoleColor.Blue;
        // Изменим цвет текста
        if ((int)symbol % 3 == 0)
            Console.ForegroundColor = ConsoleColor.Green;
        else Console.ForegroundColor = ConsoleColor.Cyan;
        Console.Write(symbol);
        // Задержка времени чтобы все цвета были разными
        Thread.Sleep(100);
}
// Возвращение настроек по умолчанию
Console.ResetColor();
```

Результат работы программы примера 17 представлен на рис. 40.

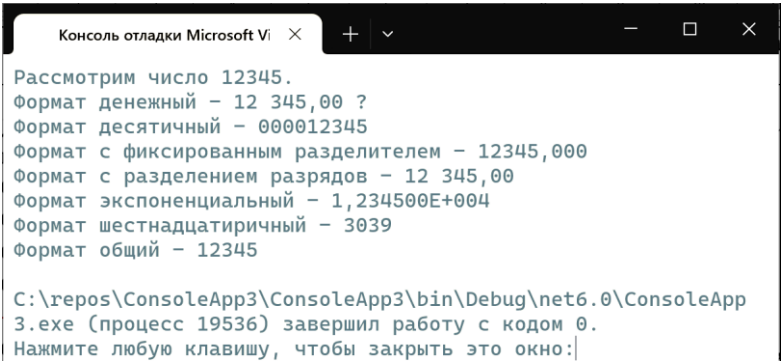


Рис. 39. Число 12345 в разных форматах (пример 16)

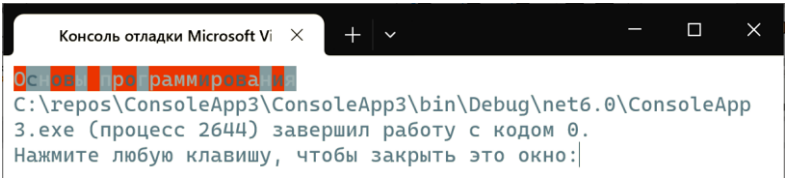


Рис. 40. Измененная консоль (пример 17)

Структуры

Одна из особенностей языка C# – это возможность создавать данные новых типов. Причем, данные нового типа могут быть комбинацией данных различного типа. Такие данные называют структурами. А затем новый структурный тип, как обычно, можно использовать для объявления новых переменных. Доступ к отдельным элементам структуры осуществляется операцией «.»

(точка). Тип структуры представляет собой тип значения, который может инкапсулировать данные и связанные функции. Для определения типа структуры используется ключевое слово struct. Например, мы создадим структуру типа **Coordinates**(координаты) и определим для нее функцию, которая позволяет комфортно представлять ее при выводе на экран:

```
struct Coordinates
{
    public double x;
    public double y;

    public override string ToString()
    {
        return $"(x = {x}, y = {y})";
    }
}
```

Теперь используем созданную структуру в программе (важно чтобы код, использующий структуры находился выше определения самой структуры при расположении всего кода в одном файле):

```
Coordinates coords;
coords.x = 1.0;
coords.y = 2.0;
Console.WriteLine(coords);
```

Рассмотрим **пример 18.** Даны два класса учащихся. Определить, являются ли они параллельными.

Для начала создадим структуру:

```
public struct Klass
{
    public int year;
    public char letter;

    public override string ToString()
    {
        return $"{year}{letter}";
    }
}
```

Теперь можно ее использовать в теле основной программы:

```
Klass first, second;
Console.WriteLine("Введи годы обучения классов");
first.year = int.Parse(Console.ReadLine());
second.year = int.Parse(Console.ReadLine());
Console.WriteLine("Введи литеры классов");
first.letter = Console.ReadKey().KeyChar;
Console.ReadLine();
```

```
second.letter = Console.ReadKey().KeyChar;
Console.ReadLine();
if ((first.year == second.year) &&
    (first.letter != second.letter))
{
    Console.WriteLine("Паралелльные классы!");
}
else
    Console.WriteLine("Не паралелльные классы!");
Console.ReadLine();
```

Результаты работы программы примера 18 представлен на рис. 41.

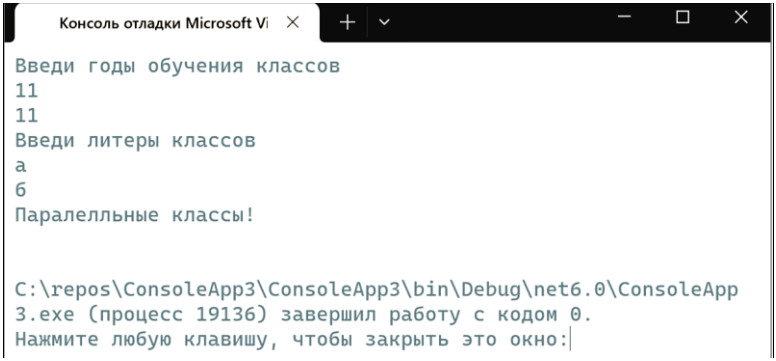


Рис. 41. Результат работы программы (пример 18)

Результат работы программы представлен на рис. 41. Заметим, что в качестве поля структуры может быть в свою очередь структура. Кроме того, из структур можно формировать и массивы. В качестве одного из элементов структуры может быть массив. Кроме того, над структурными переменными можно выполнять операции как над едиными объектами (присваивать одной структурной переменной другую, сравнивать их и т.д.), но только в том случае, если структурные переменные однотипны, т.е. описаны на основе одного структурного типа. При сравнении структуры по умолчанию будут считаться равными если равны значения всех их полей. Если в нашем примере есть необходимость определить не только параллельные классы, но и необходимость сравнить одинаковые ли классы мы создали, можно доопределить структуру операторами сравнения:

```
public static bool operator !=(Klass first, Klass second)
{
    return first.year != second.year ||
        first.letter != second.letter;
}
```

```
    public static bool operator ==(Klass first, Klass second)
    {
        return first.year == second.year &&
            first.letter == second.letter;
    }
```

Теперь мы можем в нашем коде использовать операторы сравнения (==, !=) в нашем коде следующим образом:

```
if (first == second)
   Console.WriteLine($"Два одинаковых класса {first}");
```

Поскольку структуры – это пользовательские типы данных, следовательно, они могут использоваться как тип возвращаемого значения из функции, тип данных элементов одного массива, очереди, стека и др. В реальных задачах может стоять проблема генерации элементов какой-то коллекции с типом созданной вами структуры, в этом случае можно создать функции генерации экземпляра (конкретного значения) вашей структуры. На языке программирования C# для структуры Klass эта функция будет выглядеть следующим образом:

```
Klass KlassGenerator()
{
    Klass klass = new Klass();
    char[] letters = { 'а', 'б', 'в', 'г', 'д' };
    int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
    Random letterGenerator = new Random();
    Random numberGenerator = new Random();
    klass.letter = let-ters[letterGenerator.Next(letters.Length)];
    klass.year = num-bers[numberGenerator.Next(numbers.Length)];
    return klass;
}
```

Использовать эту функцию можно, как и любую другую в любом месте кода, ниже представлен листинг **примера 19** генерации n-го количества классов и проверка не были ли сгенерированы одинаковые. Результат работы примера 19 представлен на рис. 42.

```
Console.WriteLine("Введите количество классов для генерации");
int n = int.TryParse(Console.ReadLine(), out n) ? n : 1;
Klass[] klasses = new Klass[n];
Console.WriteLine("Сгенерированные классы:");
for (int i = 0; i < n; i++)
{
    klasses[i] = KlassGenerator();
    Console.WriteLine(klasses[i]);
}

for (int i = 0; i < n - 1; i++)
{
```

```
    for (int j = i + 1; j < n; j++)
    {
        if (klasses[i] == klasses[j])
        {
            Console.WriteLine($"Два одинаковых класса
{klasses[i]}");
        }
    }
}
```

Результаты работы программы примера 19 представлен на рис. 42.

Продолжая изучение структур в языке программирования C#, рассмотрим их реализацию в самом языке. Пространства имен System.Collections и System.Collection.Generic содержат в себе такие типы данных, как *стек, очередь, словарь* и другие. Данные два пространства имен делят данные структуры на две категории: необобщённые коллекции и обобщённые коллекции.

Необобщённые коллекции спроектированы для оперирования над типами данных в языке программирования C# и, следовательно, являются слабо типизированными.

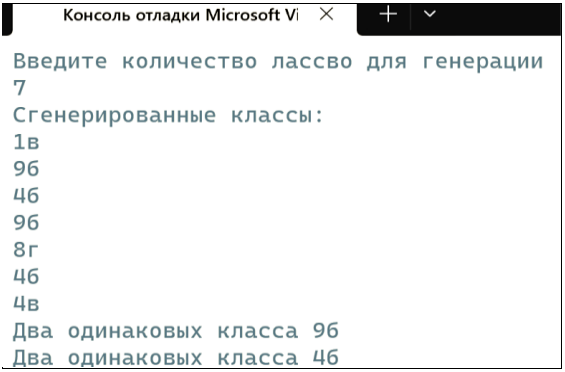


Рис. 42. Результат работы программы (пример 19)

Коллекции

Обобщённые коллекции, в отличие от необобщённых, являются строго типизированными.

Для начала рассмотрим основные необобщённые коллекции:

Таблица 3

Основные члены пространства имен System.Collections

Структура	Описание
ArrayList	Представляет коллекцию с динамически изменяемым размером (список), выдающую содержимое в последовательном порядке
BitArray	Представляет компактный массив битовых значений, которые представляются булевым типом данных
Hashtable	Представляет коллекцию пар «ключ-значение», организованных на основе хэша ключа
Queue	Представляет стандартную очередь предметов, работающую по принципу FIFO («первый вошел – первый вышел»)
SortedList	Представляет коллекцию пар «ключ-значение», отсортированных по ключу и доступных по ключу и по индексу
Stack	Представляет собой стек LIFO («последний вошел – первый вышел»), поддерживающий заталкивание и выталкивание, а также считывание

Рассмотрим работу с необобщённым списком:

```
// Создадим необобщенный список
ArrayList arrayList = new ArrayList();
// Добавим несколько элементов типа "строка"
arrayList.AddRange(new string[] { "Первый", "Второй", "Третий" });
// Выведем количество элементов
Console.WriteLine("В списке содержится {0} элементов.",
    arrayList.Count);
// Добавим один элемент типа "строка"
arrayList.Add("Четвертый!");
// Вновь выведем количество элементов
Console.WriteLine("В списке содержится {0} элементов.",
    arrayList.Count);
// Удалим данный элемент из списка
arrayList.Remove("Третий");
// Удалим элемент с таким индексом
arrayList.RemoveAt(0);
// Выведем все элементы списка
foreach (string element in arrayList)
{
    Console.WriteLine(element);
}
```

А что, если попытаться добавить в наш список еще элементы других типов данных. Представленный ниже пример кода (**пример 20**) способен показать основной недостаток данной категории коллекций. Если мы попробуем добавить в данный отличные от строк типы данных, то программа позволит нам это сделать, однако при попытке вывода всех элементов наша программа выведет ошибку (рис. 43).

```
using System.Collections;

ArrayList arrayList = new ArrayList();
arrayList.AddRange(new string[] { "Первый", "Второй", "Третий" });

Console.WriteLine("В списке содержится {0} элементов.",
    arrayList.Count);

arrayList.Add("Четвертый!");
Console.WriteLine("В списке содержится {0} элементов.",
    arrayList.Count);

arrayList.Remove("Третий");
arrayList.RemoveAt(0);

foreach (string element in arrayList)
{
    Console.WriteLine(element);
}

// Добавим один элемент типа "целое число"
arrayList.Add(123);
// Добавим один элемент логического
arrayList.Add(true);
// Добавим один элемент типа "число с плавающей точкой"
arrayList.Add(3.14);
// Добавим еще кое-что
arrayList.Add(new OperatingSystem(PlatformID.Win32NT,
    new Version(10, 0)));

// Вновь выведем колличество элементов
Console.WriteLine("В списке содержится {0} элементов.",
    arrayList.Count);
Console.WriteLine();

// Выведем все элементы списка
foreach (string element in arrayList)
{
    Console.WriteLine(element);
}
```

Результаты работы программы (пример 20) представлены на рис. 43.

Но если внимательно посмотреть на текст ошибки, то становится понятно, что она вызвана проблемой конвертации из одного типа данных в другой. В данном примере ошибка может быть исправлена исправлением последнего цикла на:

```
foreach (object element in arrayList)
```

Отредактируйте эту строчку кода и запустите программу снова. Данный пример показывает, что необобщённые коллекции способны хранить ***буквально все***, что представляет программисту исключительную гибкость.

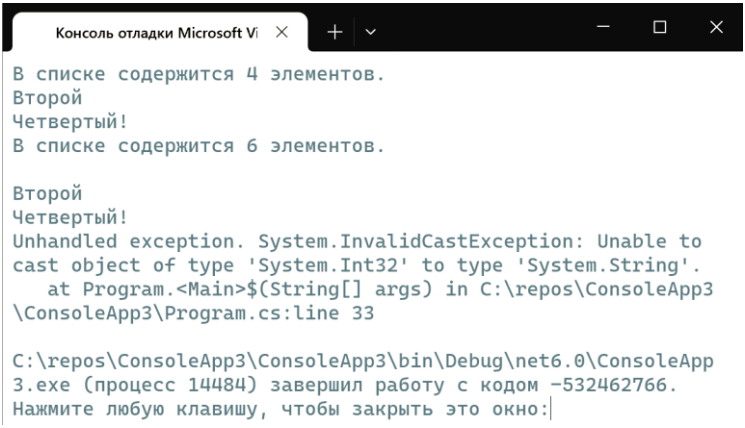


Рис. 43. Результат работы программы (пример 20)

Однако в руках менее продвинутого программиста, необобщённые коллекции зачастую порождают проблемы и ошибки, связанные с несовместимостью типов и операций. Подобные проблемы решаются использованием их строго типизированных аналогов, которыми являются обобщённые коллекции.

Рассмотрим основные обобщённые коллекции:

Таблица 4

Основные члены пространства имен System.Collections.Generic

Структура	Описание
Dictionary<TKey, TValue>	Представляет обобщённую коллекцию ключей и значений
LinkedList<T>	Представляет двухсвязный список
List<T>	Представляет последовательный список элементов с динамически изменяемым размером
Queue<T>	Обобщённая реализация очереди
SortedDictionary <TKey, TValue>	Обобщённая реализация сортированного множества пар «ключ-значение»
SortedSet<T>	Представляет коллекцию предметов, поддерживаемых в сортированном порядке без дубликатов
Stack<T>	Обобщённая реализация стека

Инициализация обобщённых коллекций происходит подобным образом:

```
// Создадим список чисел
List<int> list1 = new List<int> { 1, 2, 3, 4, 5 };
// Создадим список строк с помощью ввода
List<string> list2 = new List<string>();
string insert;
for (int i = 0; i < 3; i++)
{
```

```
    insert = Console.ReadLine();
    list2.Add(insert);
}
```

Создадим также обобщённый список структуры Coordinates из предыдущей главы, а также для структуры, которая ее содержит:

```
// Создадим список структур
List<Coordinatess> cords = new List<Coordinatess>
    {
        new Coordinatess { x = 0, y = 0 },
        new Coordinatess { x = 3, y = 6}
    };
// Создадим список структур, содержащих другую структуру
List<Road> roads = new List<Road>
    {
    new Road { Start = new Coordinatess { x = 0, y = 1 },
        End = new Coordinatess { x = 1, y = 2 }},
    new Road { Start = new Coordinatess { x = 2, y = 3 },
        End = new Coordinatess { x = 3, y = 4 }}
    };
```

```
public struct Road
{
    public Coordinatess Start;
    public Coordinatess End;
}
```

Рассмотрим возможности типа данных List<T> более подробно. Он, как и его необобщенный аналог ArrayList, поддерживает следующие операции:

- **Add():** добавляет новый элемент в конец.
- **AddRange():** добавляет все элементы коллекции или массива в конец.
- **Insert():** вставляет элемент по индексу.
- **InsertRange():** добавляет все элементы коллекции или массива, начиная с индекса.
- **CopyTo():** копирует все элементы в массив.
- **Contains():** возвращает true, если элемент есть в списке.
- **Clear():** очищает список.
- **IndexOf():** возвращает индекс первого вхождения элемента в списке.
- **LastIndexOf():** возвращает индекс последнего вхождения элемента в списке.
- **Remove():** удаляет введенный элемент из списка, если есть дубликаты, удалит первое вхождение.
- **RemoveAt():** удаляет элемент под указанным индексом.
- **RemoveRange():** удаляет введенное количество элементов, начиная с введенного индекса.

– **Reverse():** располагает список в обратном порядке.
– **Sort():** сортирует список (если элементы поддерживают сортировку).
– **Count:** возвращает длину списка.

Перейдем к типу данных Stack<T>. Для него определены следующие операции:

– **Push():** добавляет элемент на верхушку стека.
– **Pop():** извлекает и возвращает первый элемент стека.
– **Peek():** возвращает первый элемент стека без извлечения.
– **Contains():** проверяет наличие в стеке элемента.
– **Clear():** очищает стек.
– **Count:** возвращает длину стека.

Пример работы со стеком (**пример 21**) представлен в листинге ниже:

```
// Создадим стек координат
Stack<Coordinatess> cords = new Stack<Coordinatess>();
// Добавим в него элементов
cords.Push(new Coordinatess { x = 0, y = 0 });
cords.Push(new Coordinatess { x = 10, y = 10 });
cords.Push(new Coordinatess { x = 20, y = 20 });
// Посмотрим первый элемент в стеке
Console.WriteLine("Первая координата в стеке {0}",
cords.Peek().ToString());
Console.WriteLine("Вытащим координату {0}", cords.Pop().ToString());
Console.WriteLine("Теперь первая координата в стеке {0}",
cords.Peek().ToString());
Console.WriteLine("Вытащим координату {0}", cords.Pop().ToString());
Console.WriteLine("Теперь первая координата в стеке {0}",
cords.Peek().ToString());
Console.WriteLine("Вытащим координату {0}", cords.Pop().ToString());
Console.WriteLine();
// Посмотрим длину стека
Console.WriteLine("Стек теперь пуст, его длина = {0}", cords.Count);
```

Результат работы программы (пример 21) представлен на рис. 44.

Перейдем к типу коллекций Queue<T>. Для него в свою очередь определены следующие операции:

– **Dequeue():** извлекает и возвращает элемент из начала очереди.
– **Enqueue():** добавляет элемент в конец очереди.
– **Peek():** возвращает элемент из начала очереди, не удаляя его.
– **Contains():** проверяет наличие в очереди элемента.
– **Clear():** очищает очередь.
– **Count:** возвращает длину очереди.

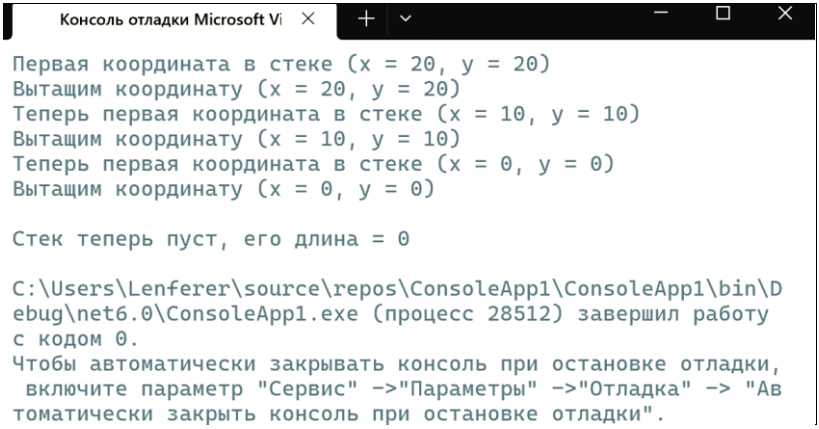


Рис. 44. Результат исполнения программы (пример 21)

Пример работы с очередью (**пример 22**) представлен ниже.

```
// Создадим очередь координат
Queue<Coordinatess> cords = new Queue<Coordinatess>();
// Добавим в него элементов
cords.Enqueue(new Coordinatess { x = 0, y = 0 });
cords.Enqueue(new Coordinatess { x = 10, y = 10 });
cords.Enqueue(new Coordinatess { x = 20, y = 20 });
// Посмотрим первый элемент в очереди
Console.WriteLine("Первая координата в очереди {0}",
cords.Peek().ToString());
Console.WriteLine("Вытащим координату {0}",
cords.Dequeue().ToString());
Console.WriteLine("Теперь первая координата в очереди {0}",
cords.Peek().ToString());
Console.WriteLine("Вытащим координату {0}",
cords.Dequeue().ToString());
Console.WriteLine("Теперь первая координата в очереди {0}",
cords.Peek().ToString());
Console.WriteLine("Вытащим координату {0}",
cords.Dequeue().ToString());
Console.WriteLine();
// Посмотрим длину очереди
Console.WriteLine("Очередь теперь пуста, её длина = {0}",
cords.Count);
```

Результат работы программы (пример 22) представлен на рис. 45.

Тип данных SortedSet<T> поддерживает все операции List<T>, его отличие заключается в том, при вставке или удалении элементов он автоматически

устраивает сортировку всего набора. Однако, он не поддерживает тип данных, которые не поддерживают сортирование. Кроме того, в состав такого списка могут входить только уникальные элементы, одинаковыми (не уникальными) считаются элементы, которые могут располагаться на одном и том же месте в результате сортировки списка. Поддержка сортировки в данном типе данных обеспечивается при использовании интерфейса IComparable<T>.

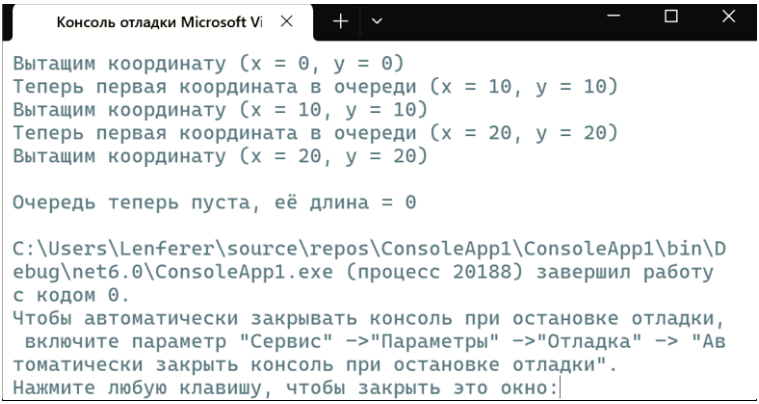


Рис. 45. Результат исполнения программы (пример 22)

Добавим его в нашу структуру:

```
public struct Coordinatess : IComparable<Coordinatess>
{
    public double x;
    public double y;
    // Координата будет считаться большей, если ее сумма x и y
больше
    public int CompareTo(Coordinatess coord)
    {
        if (x + y > coord.x + coord.y)
        {
            return 1;
        }
        else if (x + y > coord.x + coord.y)
        {
            return -1;
        }
        else return 0;
    }

    public override string ToString()
    {
```

```
        return $"(x = {x}, y = {y})";
    }
}
```

Теперь наша структура способна находится в **SortedSet<Coordinatess>**. Рассмотрим поведение использование такого сортированного списка на простом примере используя выше разработанную структуру (**пример 23**).

```
// Создадим сортированный список координат
SortedSet<Coordinatess> coords = new SortedSet<Coordinatess>();

// Добавим в него элементов
coords.Add(new Coordinatess { x = 0, y = 10 });
coords.Add(new Coordinatess { x = 6, y = 7 });
coords.Add(new Coordinatess { x = 15, y = 0 });
coords.Add(new Coordinatess { x = 5, y = 5 });
// Посмотрим длину списка
Console.WriteLine("Размер списка = {0}", coords.Count);

//Выведем все элементы списка
foreach (Coordinatess coord in coords)
{
    Console.WriteLine(coord);
}
```

Результат работы программы (пример 23) представлен на рис. 46.

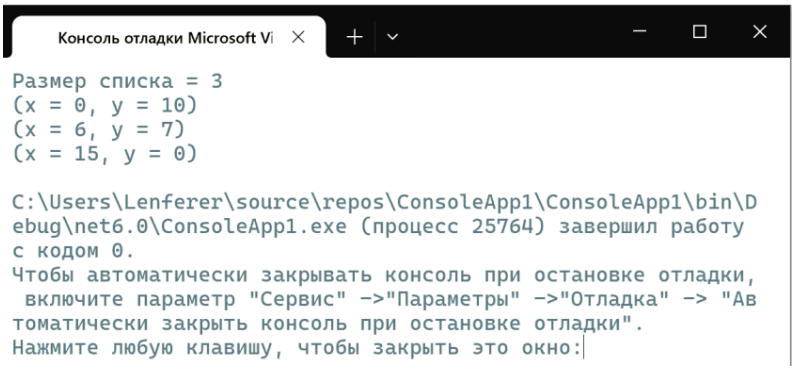


Рис. 46. Результат исполнения программы (пример 23)

Как видно из результата исполнения программы, координата $x = 5$, $y = 5$, не вошла в список, так как ее сумма равна сумме координаты $x = 0$, $y = 10$, что было принято при сортировке за дубликат и не включено в сортированный список.

Еще одной удобной обобщённой коллекцией является словарь. В языке программирования C# он представлен типом данных Dictionary<TKey, TValue>.

Он позволяет хранить любое количество предметов, на которые можно ссылаться через уникальный ключ. Таким образом, вместо использования числовых индексов, как например в List<T>, можно использовать любые другие типы данных. Для него определены следующие операции:

- **Add():** добавляет пару «ключ-значение».
- **Clear():** очищает словарь.
- **ContainsKey():** возвращает true, если словарь содержит введенный ключ, иначе возвращает false.
- **ContainsValue():** возвращает true, если словарь содержит введенное значение, иначе возвращает false.
- **Remove():** удаляет значение с данным ключом.
- **Count:** возвращает длину словаря.
- **Keys: в**озвращает коллекцию ключей.
- **Values:** возвращает коллекцию значений.
- **Item[ключ]:** возвращает значение под данным ключом.

Для словаря также определен тип данных **KeyValuePair<TKey, TValue>**, который является элементом словаря, хранящим пару «ключ-значение».

Рассмотрим использование словаря (**пример 24**).

```
// Создадим словарь с помощью специального синтаксиса
Dictionary<string, Coordinatess> map = new Dictionary<string, Coor-
dinatess>()
{
    ["База"] = new Coordinatess { x = 30, y = 14 },
    ["Опасность!"] = new Coordinatess { x = 9, y = 14 },
};

// Добавим в него новые элементы
map.Add("Сокровище!", new Coordinatess { x = 10, y = 12 });

map.Add("Центр", new Coordinatess { x = 0, y = 0 });

// Получим элемент с ключом "Опасность!"
Console.WriteLine("Опасность находится по координатам {0}",
map["Опасность!"].ToString());

// Выведем все элементы словаря
foreach (KeyValuePair<string, Coordinatess> pair in map)
{
    Console.WriteLine("{0} - {1}", pair.Key, pair.Value.ToString());
}
```

Результат работы программы (пример 24) представлен на рис. 47.

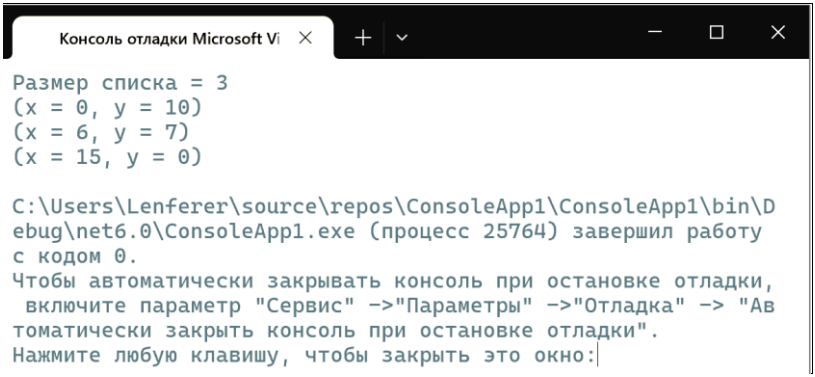


Рис. 47. Результат исполнения программы (пример 24)

Контрольные вопросы

1. Что такое структура?
2. Какие структуры Вам известны?
3. Что такое коллекция в языке программирования C#?
4. В чем заключаются отличия обобщённых и необобщённых коллекций в языке программирования C#?
5. Что такое стек, какие операции для него характерны?
6. Что такое очередь, какие операции для нее характерны?
7. Что такое словарь, какие операции для него характерны?

Обработка исключений

Написание программ является сложным делом, и из-за такой сложности программисты и пользователи часто сталкивается с разнообразными *проблемами*. В одних случаях они возникают из-за *«плохо написанного кода»,* например, по причине вызова за границы массива, в других случаях, из-за ввода неучтенных некорректных данных. Вне зависимости от природы проблемы в конечном итоге программа не работает ожидаемым образом. Для дальнейшего обсуждения подобных аномалий, рассмотрим три распространенных термина, применяемых для их описания:

– *дефекты:* это ошибки, допущенные программистом, к примеру, не закрытый после работы файл.

– *пользовательские ошибки:* это ошибки, которые произошли при неправильной эксплуатации программы, например, неправильный ввод.

– *исключения:* это аномалии, которые трудно учесть при программировании, например, попытка открыть испорченный файл, или подключится к базе данных, которой больше не существует. В данных случаях программист или пользователь обладают низким контролем над подобными обстоятельствами.

С учетом данных определений должно быть понятно, что обработка *исключений* в языке программирования C#, это прием работы с *исключительными ситуациями* во время выполнения. Тем не менее, для дефектов и пользовательских ошибок программная среда будет генерировать соответствующее *исключение*, идентифицирующее возникшую проблему. В качестве примера приведем самые часто встречающиеся:

– **FormatException:** исключение, которое возникает в случае, если формат аргумента недопустим (например, несовместимый тип данных при вводе).

– **IndexOutOfRangeException:** исключение, возникающее при попытке обращения к элементу массива или коллекции с индексом, который находится вне границ (иными словами, выход за пределы массива).

– **FileNotFoundException:** исключение, которое выдается при попытке получить доступ к файлу или каталогу, которых нет на диске.

– **ArgumentException:** это исключение выбрасывается, если один из передаваемых методу аргументов является недопустимым.

– **NullReferenceException:** исключение возникает при попытке доступа к члену типа данных, значение которого равно **null.**

– **ArgumentOutOfRangeException:** исключение, которое выдается, если значение аргумента не соответствует допустимому диапазону значений (например 32 день месяца).

– **DivideByZeroException:** исключение возникает при попытке деления на ноль.

– **InvalidCastException:** исключение, которое выдается в случае недопустимого приведения или явного преобразования (например, строки в число).

– **OverflowException:** исключение, которое выдается, если выполнение арифметической операции, операции приведения к типу или преобразования в проверяемом контексте приводит к переполнению (например, преобразование числа 123456 в short).

– **StackOverflowException:** исключение, которое возникает, когда стек выполнения превышает размер стека программной среды (например, при выполнении слишком большого числа рекурсий).

Вместе с уже запрограммированными случаями выброса исключений, мы можем также собственноручно определять (**пример 25**), в каких случаях и какие исключения должны выбрасываться. Для того, чтобы программа выбросила ошибку, применяется ключевое слово `throw`:

```
int a = Convert.ToInt32(Console.ReadLine());
if (a == 0)
    // Выбросим ошибку, указывающую на то 0 является недопу-стимым
аргументом
    throw new ArgumentException();
else
    Console.WriteLine(a);
```

Результат работы программы пример 25 представлен на рис. 48.

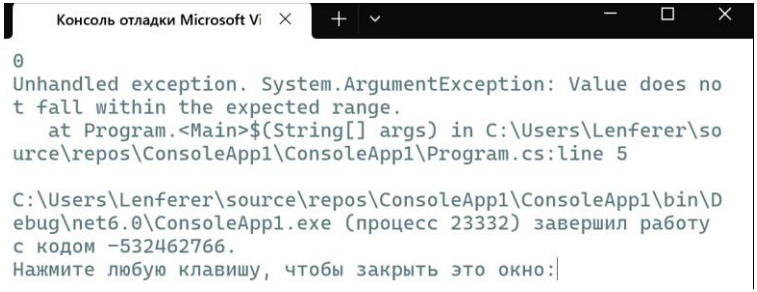


Рис. 48 – Попытка ввода недопустимого для нас нуля в программу (пример 25)

Однако, если просто расписывать подобные «выбросы», то программа будет просто «глохнуть». Если нам известно, что потенциальное исключение может произойти, то было бы неплохо расписать дополнительно для программы инструкции в случае возникновения исключительной ситуации. В языке программирования C# это возможно с помощью конструкции `try { } catch { }`. Включим же данную конструкцию в наш пример (**пример 26**):

```
try
{
    int a = new int();
    try
```

```
    {
        a = Convert.ToInt32(Console.ReadLine());
        if (a == 0 || a == 1)
            throw new ArgumentException();
        else
            Console.WriteLine(a);
    }
    catch (ArgumentException e) when (a != 0)
    {
        Console.WriteLine("Единица тоже недопустимый аргумент");
    }
    catch (FormatException e)
    {
        Console.WriteLine("Недопустимый ввод!");
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("Ноль является недопустимым аргументом!");
    }
}
catch
{
    Console.WriteLine("Ошибку не удалось исправить :( ");
}
```

Результат работы программы (пример 26) представлен на рис. 49.

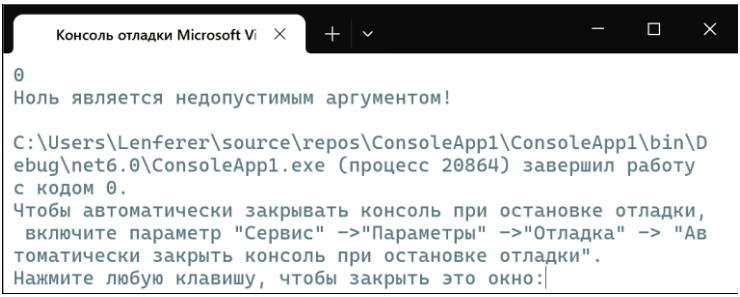


Рис.49. Новая попытка ввода недопустимого нуля в программу (пример 26)

Блок `try { }` должен содержать блок кода, который способен воспроизвести какую-либо исключительную ситуацию. Блок `catch { }` в свою очередь, содержит блок кода, который будет выполняться *в исключительном случае*. Стоит отметить, что первая строка кода

`(int a = Convert.ToInt32(Console.ReadLine()) )`

находится в блоке `try { }`, а значит любая генерируемая ей ошибка заставит программу «заглохуть».

В коде может содержаться несколько блоков `try { }` и `catch { }`, также, один блок `try { }` может содержать несколько блоков `catch { }`. В таком случае для каждого блока `catch { }` должны быть выделены уникальные типы ошибки (**пример 27**):

```
try
{
    int a = Convert.ToInt32(Console.ReadLine());
    if (a == 0)
        throw new ArgumentException();
    else
        Console.WriteLine(a);
}
catch (ArgumentException e)
{
    Console.WriteLine("Ноль является недопустимым аргументом!");
}
catch (FormatException e)
{
    Console.WriteLine("Недопустимый ввод!");
}
```

Результат работы программы (пример 27) представлен на рис. 50, 51.

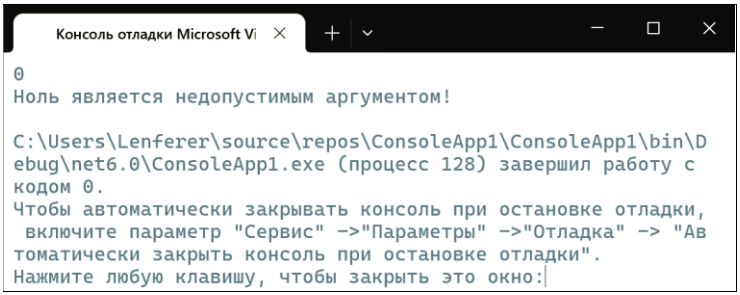


Рис. 50. Обработка исключений ArgumentException (пример 27)

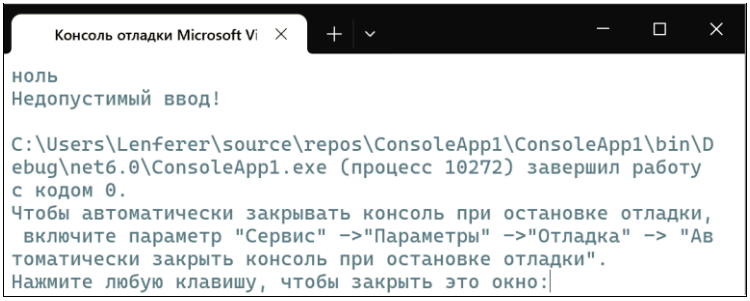


Рис. 51. Обработка исключений FormatException (пример 27)

Стоит также отметить, что без указания конкретной исключительной ситуации для блока `catch { }`, он будет реагировать на все, кроме тех что обрабатываются блоками с явно указанными исключениями.

Блоки `catch { }`, также способны генерировать ошибки, связанные как и с проблемами, возникшими при реагировании на исключительные ситуации, так и на запрограммированные условия (**пример 28**) при помощи ключевого слова `throw`. В таком случае для реагирования на подобную исключительную ситуацию потребуется поместить блок `try { } catch { }` в другой блок `try { } catch { }`:

```
try
{
    int a = new int();
    try
    {
        a = Convert.ToInt32(Console.ReadLine());
        if (a == 0)
            throw new ArgumentException();
        else
            Console.WriteLine(a);
    }
    catch (FormatException e)
    {
        Console.WriteLine("Недопустимый ввод!");
        throw new ArgumentException();
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("Ноль является недопустимым аргументом!");
    }
}
catch
{
    Console.WriteLine("Ошибку не удалось исправить :( ");
}
```

Результат работы программы (пример 28) представлен на рис. 52.

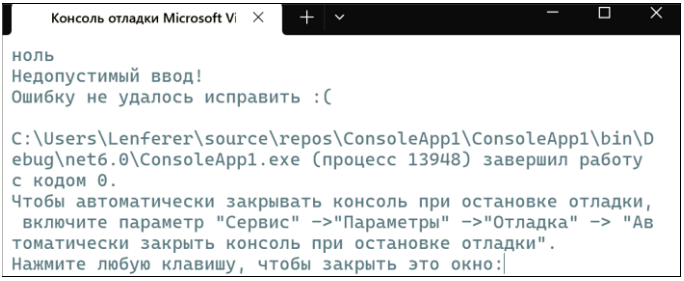


Рис. 52. Выброс ошибки блоком catch (пример 28)

Для дополнительной гибкости, для блоков `catch { }` также возможно задавать условия выполнения (**пример 29**) при помощи ключевого слова `when`, который работает подобно оператору `if`:

```
try
{
    int a = new int();
    try
    {
        a = Convert.ToInt32(Console.ReadLine());
        if (a == 0 || a == 1)
            throw new ArgumentException();
        else
            Console.WriteLine(a);
    }
    catch (ArgumentException e) when (a != 0)
    {
        Console.WriteLine("Единица тоже недопустимый аргумент");
    }
    catch (FormatException e)
    {
        Console.WriteLine("Недопустимый ввод!");
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("Ноль является недопустимым аргументом!");
    }
}
catch
{
    Console.WriteLine("Ошибку не удалось исправить :( ");
}
```

Результат работы программы (пример 29) представлен на рис. 53.

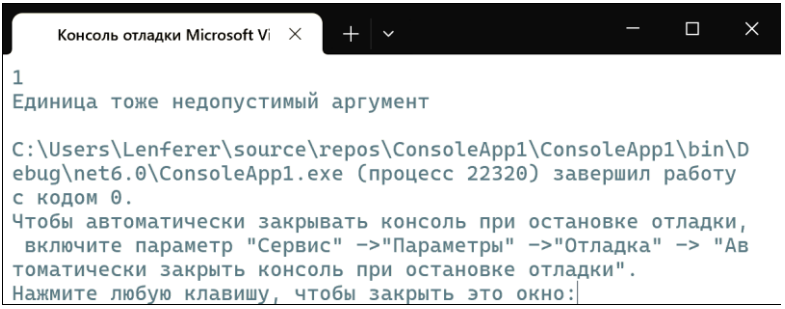


Рис. 53. Обработка ошибки блоком catch-when (пример 29)

Последним инструментом в обработке исключительных ситуаций является блок `finally { }`. Данный блок исполняется как при наличии исключительных ситуаций, так и при их отсутствии (**пример 30**):

```
try
{
    try
    {
        int a = Convert.ToInt32(Console.ReadLine());
        if (a == 0 || a == 1)
            throw new ArgumentException();
        else
            Console.WriteLine(a);
    }
    catch (FormatException e)
    {
        Console.WriteLine("Недопустимый ввод!");
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("0 и 1 является недопустимым аргумен-
том!");
    }
}
catch
{
    Console.WriteLine("Ошибку не удалось исправить :( ");
}
finally
{
    Console.WriteLine("Внешний блок кода исполнен.");
}
```

Результат работы программы (пример 30) представлен на рис. 54, 55.

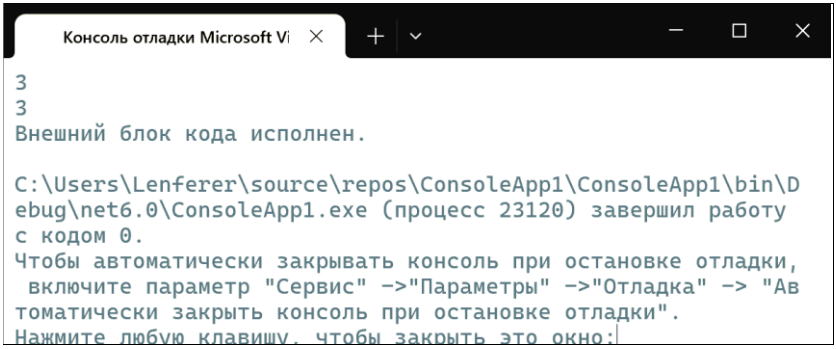


Рис. 54. Работа блока finally при отсутствии исключений (пример 30)

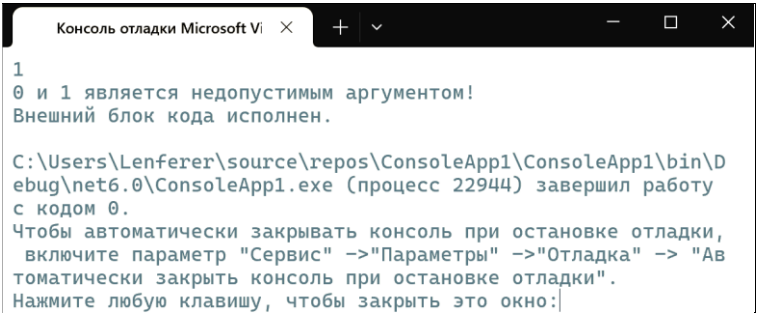


Рис. 55. Работа блока finally при наличии исключений (пример 30)

Контрольные вопросы

1. Что такое исключительная ситуация?
2. Какие исключительные ситуации могут возникнуть при написании и запуске программ?
3. Как устроена обработка исключений в языке C#?
4. Как можно не допустить возникновение исключительной ситуации?

Классы

Класс является основой для создания объектов. Если структура объединяет разнородные по типам переменные, то в классе определяются данные и код, который работает с этими данными. Объекты являются экземплярами класса. Методы и переменные, составляющие класс, называются членами класса. При определении класса объявляются данные, которые он содержит, и код, работающий с этими данными. Данные содержатся в переменных экземпляра, которые определены классом, а код содержится в методах. В C# определены несколько специфических разновидностей членов класса. Это – переменные экземпляра, статические переменные, константы, методы, конструкторы, деструкторы, индексаторы, события, операторы и свойства.

Непосредственно инициализация переменных в объекте (переменных экземпляра) происходит в конструкторе. В классе могут быть определены несколько конструкторов.

Синтаксис класса:

```
// Класс для описания собаки
class Dog
{
    string name; // переменная экземляра "Кличка"
    string breed; // переменная экземляра "Порода"

    // Конструктор класса Dog
    public Dog(string name, string breed)
    {
        this.name = name;
        this.breed = breed;
        Console.WriteLine("Woof Woof");
    }
    //Деструктор класса
    ~Dog()
    {
        Console.WriteLine("Puppy is gone");
    }

    //Свойство класса Кличка
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    //Свойство класса Порода
    public string Breed
    {
        get { return breed; }
```

```
        set { breed = value; }
    }
}
```

Все члены классов имеют уровень доступности или модификатор доступа. Он определяет возможность их использования из другого кода в вашей программе. Следующие модификаторы доступа позволяют указать доступность элемента класса при объявлении:

`public`: доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.

`private`: доступ к типу или члену возможен только из кода в том же объекте `class` или `struct`.

`protected`: доступ к типу или члену возможен только из кода в том же объекте `class` либо в `class`, производном от этого `class`.

`internal`: доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.

`protected internal`: доступ к типу или члену возможен из любого кода в той сборке, где он был объявлен, или из производного `class` в другой сборке.

`private protected`: доступ к типу или члену возможен только из его объявляющей сборки из кода в том же `class` либо в типе, производном от этого `class`.

Члены класса с типом доступа `public` доступны везде за пределами данного класса, с типом доступа `protected` – внутри членов данного класса и производных, с типом доступа `private` – только для других членов данного класса. Тип доступа `internal` применяется для типов, доступных в пределах одной сборки. Модификатор доступа можно не указывать в таком случае – по умолчанию считается что используется модификатор доступа private для методов и атрибутов классов, internal для классов.

При создании объекта класса происходит вызов соответствующего конструктора класса. Конструктор класса – метод для инициализации объекта при его создании. Он имеет то же имя, что и его класс. В конструкторах тип возвращаемого значения не указывается явно. Конструкторы используются для присваивания начальных значений переменным экземпляра, определенным классом, и для выполнения любых других процедур инициализации, необходимых для создания объекта.

Все классы имеют конструкторы независимо от того, определен он или нет. По умолчанию в C# предусмотрено наличие конструктора, который присваивает нулевые значения всем переменным экземпляра (для переменных обычных типов) и значения null (для переменных ссылочного типа). Но если конструктор явно определен в классе, то конструктор по умолчанию использоваться не будет: имя_класса(список_параметров) {тело_конструктора}. Модификатор доступа при конструкторе не может быть private.

Деструктор – метод, вызывающийся автоматически при уничтожении объекта класса (непосредственно перед —сборкой мусора). Деструктор не имеет параметров и возвращаемого значения: ~имя_класса() {тело_деструктора}. Модификатор доступа при деструкторе всегда должен быть private.

Свойство класса – это член, предоставляющий гибкий механизм для чтения, записи или вычисления значения частного поля. Свойства можно использовать, как если бы они были членами общих данных, но фактически они представляют собой специальные методы, называемые *методами доступа*. Это позволяет легко получать доступ к данным и помогает повысить безопасность и гибкость методов. Рассмотрим простой пример.

Пример 31. Комплексное число представляется парой действительных чисел (a,b), где a – действительная часть, b – мнимая часть: a+b*i, здесь i – мнимая единица, i=sqrt(–1). Реализовать класс Complex() для работы с комплексными числами. Обязательно должны присутствовать операции:

Сложение add, (a, bi) + (c, di) = (a + c) + (b + d)i.

Вычитание sub, (a, bi) – (c, di) = (a – c) + (b – d)i.

Умножение mul, (a, bi) * (c, di) = (ac – bd) + (ad + bc)i.

```
class Complex
{
    private int _a;
    private int _b;
    public int A
    {
        get { return _a; }
        set { _a = value; }
    }
    public int B
    {
        get { return _b; }
        set { _b = value; }
    }
    public Complex(int a, int b)
    {
        _a = a;
        _b = b;
    }
    public Complex Add(Complex secondnumber)
    {
        int result_a = _a + secondnumber.A;
        int result_b = _b + secondnumber.B;
        return new Complex(result_a, result_b);
    }
    public Complex Sub(Complex secondnumber)
    {
        int result_a = _a - secondnumber.A;
```

```
        int result_b = _b - secondnumber.B;
        return new Complex(result_a, result_b);
    }
    public Complex Mul(Complex secondnumber)
    {
        int result_a = _a * secondnumber.A - _b * secondnumber.B;
        int result_b = _a * secondnumber.B + _b * secondnumber.A;
        return new Complex(result_a, result_b);
    }
    public override string ToString()
    {
        return $"{_a} + {_b}*i";
    }
}
```

Как видно из примера выше экземпляры класса можно передавать в функции и в методы другого класса, при этом в отличии от структур и передачи обычных переменных значений, экземпляры классов передаются по ссылке, это означает что если внутри функции или метода изменить какое-либо из полей (атрибутов) экземпляра класса (инстанса), то их значения изменятся и в той области видимости, в которой вызывает функция или метод.

Пример использования разработанного класса представлен ниже:

```
Complex a = new Complex(2, 3);
Complex b = new Complex(5, -7);
Console.WriteLine($"a = {a.ToString()}");
Console.WriteLine($"b = {b.ToString()}");
Console.WriteLine($"a + b = {a.Add(b).ToString()}");
Console.WriteLine($"a - b = {a.Sub(b).ToString()}");
Console.WriteLine($"a * b = {a.Mul(b).ToString()}");
```

Результат работы такой программы представлен на рис. 56.

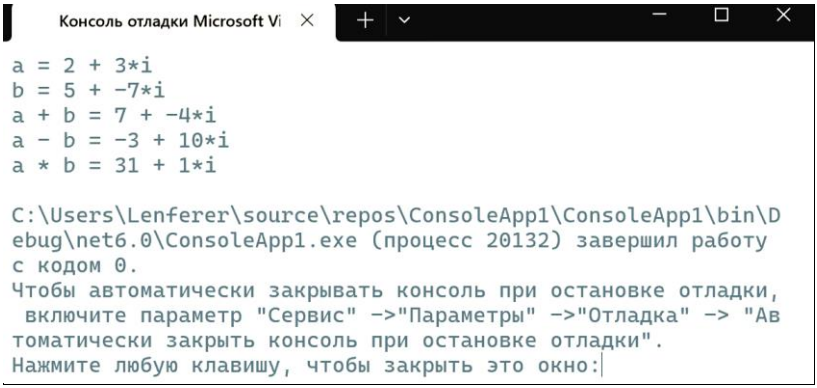


Рис. 56. Результат работы программы (**пример 31**)

Наследование – это свойство, с помощью которого один объект может приобретать свойства другого. При этом поддерживается концепция иерархической классификации, имеющей направление сверху вниз. Используя наследование, объект должен определить только те качества, которые делают его уникальным в пределах своего класса. Он может наследовать общие атрибуты от своих родительских классов.

```
class Quadrilateral
{
    public int d1;
    public int d2;
    public double fi;
    public double Area()
    {
        return 0.5 * d1 * d2 * Math.Sin(fi);
    }
}
class Rhombus : Quadrilateral
{
    public new double Area()
    {
        double fi = 90 * Math.PI / 180.0;
        return 0.5 * d1 * d2 * Math.Sin(fi);
    }
}
```

С помощью наследования создается иерархия классов. Кроме того, можно построить еще одну структуру – иерархию объектов (тогда, когда один объект является частью другого.

При разработке программы возможно также создание собственных классов исключительных ситуаций. Подобно структурам, в языке программирования C# исключение также является типом данных. Чтобы программная среда понимала, что наш новый созданный тип данных является описанием исключительной ситуации, необходимо, чтобы он повторял функционал уже созданных подобных типов:

```
class MyArgumentException : ArgumentException
{
    // Моё исключение, связанное с недопустимым аргументом
}
class MyFormatException : FormatException
{
    // Моё исключение, связанное с недопустимым форматом
}
class MyBaseException : Exception
{
    // Просто моё исключение, без конкретики происхождения
}
```

Каждый тип данных, описывающий исключительную ситуацию, имеет следующие поля.

Таблица 5

Основные данные для типов исключений

Свойство	Описание
Data	Представляет собой словарь с дополнительной информацией. Изначально пуст
Message	Текстовое описание заданной ошибки
StackTrace	Возвращает последовательность вызовов, которая привела к возникновению исключения
TargetSite	Возвращает функцию, вызвавшую исключение

Помимо этого, собственный тип данных исключения можно описывать как структуру, придавая ей иные функции и поля (**пример 32**):

```
try
{
    string input = Console.ReadLine();
    int number = Convert.ToInt32(input);
    if (!input.Contains('0'))
        throw new MyFormatException();
}
catch (MyFormatException e)
{
    // Вызываем функцию и переменную, созданные нами
    Console.WriteLine("Описание ошибки: {0}", e.Message);
    Console.WriteLine("Номер ошибки: {0}", e.GetErrorNumber());
}
catch (FormatException e)
{
    // Вызываем общие для всех типов исключений функции и переменные
    Console.WriteLine("Описание ошибки: {0}", e.Message);
    Console.WriteLine("Функция, вызвавшая исключение: {0}",
e.TargetSite);
    Console.WriteLine("Последовательность вызовов, предшествующая
исключению: {0}", e.StackTrace);
}
// Определим что-то уникальное нашему типу данных
class MyFormatException : FormatException
{
    public new string Message = "Нужны только числа с нулем.";

    public int GetErrorNumber()
    {
        return 100500;
    }
}
```

Результат работы программы (пример 32) представлен на рис. 57, 58.

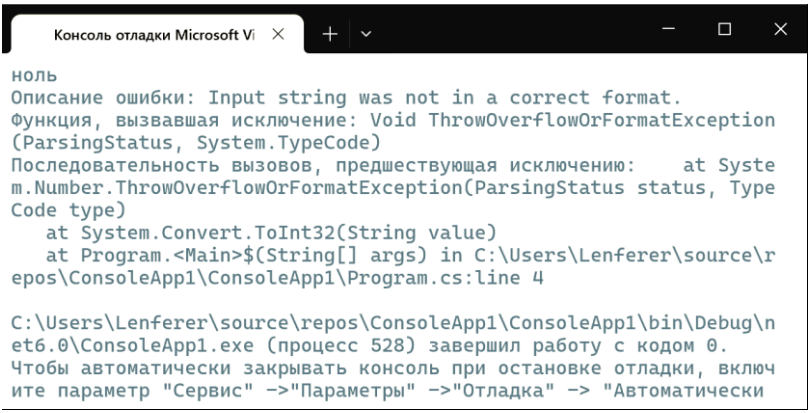


Рис. 57. Результат обработки исключения FormatException (пример 32)

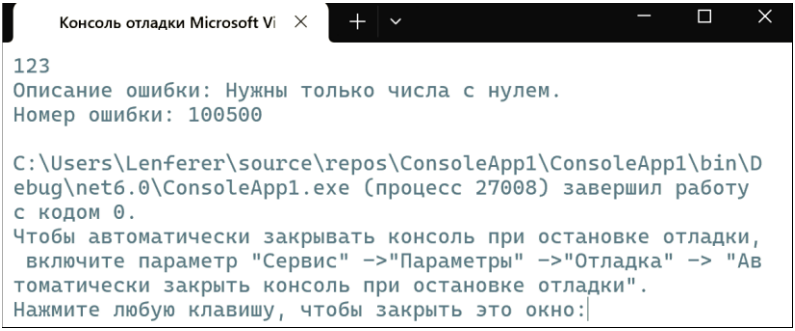


Рис. 58. Результат обработки исключения MyFormatException (пример 32)

ОБЩИЕ ТРЕБОВАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

Выполнение лабораторных работ по дисциплине «Основы программирования» требует решения двух задач по каждой из тем приведенных в пособии и, как результат, создание программ на языке программирования C# в среде разработки Microsoft Visual Studio 2022 или любой другой IDE и написания отчета к работам. Порядок и объем работ, а также требования к содержанию представлены ниже.

На первом занятии необходимо согласовать вариант работ с преподавателем, пример оформления титульного листа представлен в приложении А.

Отчет по лабораторным работам должен содержать следующие разделы:

– титульный лист;
– содержание;
– описание цели и заданий на работу;
– теоретические сведения по теме;
– ход работы;
– заключение;
– список использованных источников;

В ходе выполнения лабораторных работ рекомендуется ознакомиться со списком литературы, который прилагается к данному пособию. Отчет по лабораторной работе должен быть оформлена в соответствии ОС ТУСУР 01–2021.

Ход работы должен содержать следующие подпункты:

– описание алгоритма решения задачи;
– представление алгоритма в виде блок-схемы;
– листинг исходного кода программы;
– скриншоты работы программы;
– выводы о корректности работы программы;

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема работы: ***Циклические программы.***

Цель работы: овладеть навыками разработки алгоритмов с циклами с параметром и с условием, разработки циклических программ на языке высокого уровня.

Задание: составить консольное приложение для решения нижеприведенных задач согласно варианту, согласованному с преподавателем, вводя данные в ходе выполнения программы. Для выполнения предварительно ознакомьтесь с соответствующими разделами данного пособия.

Для задания 1 использовать цикл с параметром, для задания 2 цикл с предусловием либо цикл с постусловием.

Задание 1

1. Для заданного натурального числа n и вещественного x рассчитать сумму: $\cos x + \cos^2 x + \cos^3 x + \ldots + \cos^n x$.
2. Для заданного натурального числа n и вещественного x рассчитать сумму: $\sin x + \sin^2 x + \sin^3 x + \ldots + \sin^n x$.
3. Для заданного натурального числа n и вещественного x рассчитать сумму: $\cos x + \cos x^2 + \cos x^3 + \ldots + \cos x^n$.
4. Для заданного натурального числа n и вещественного x рассчитать сумму: $\sin x + \sin x^2 + \sin x^3 + \ldots + \sin x^n$.
5. Для заданного натурального числа n и вещественного x рассчитать сумму: $\mathrm{tg}x + \mathrm{tg}^2 x + \mathrm{tg}^3 x + \ldots + \mathrm{tg}^n x$. .
6. Для заданного натурального числа n и вещественного x рассчитать сумму: $\mathrm{ctg}x + \mathrm{ctg}^2 x + \mathrm{ctg}^3 x + \ldots + \mathrm{ctg}^n x$.
7. Для заданного натурального числа n и вещественного x рассчитать сумму: $\mathrm{tg}x + \mathrm{tg}x^2 + \mathrm{tg}x^3 + \ldots + \mathrm{tg}x^n$.
8. Для заданного натурального числа n и вещественного x рассчитать сумму: $\mathrm{ctg}x + \mathrm{ctg}x^2 + \mathrm{ctg}x^3 + \ldots + \mathrm{ctg}x^n$.
9. Для заданного натурального числа n рассчитать сумму $1 + 2 + 3 + \ldots + n$ и сравнить со значением $n(n+1)/2$.
10. Для заданного натурального числа n рассчитать сумму $1 + 2 + 3 + \ldots + n$ и сравнить со значением n^2.
11. Для заданного натурального числа n рассчитать сумму $1 + 2 + 3 + \ldots + n$ и сравнить со значением $n(n+1)$.
12. Для заданного натурального числа n рассчитать сумму $1 + 3 + 5 + \ldots + 2n - 1$ и сравнить со значением $n(n+1)/2$.
13. Для заданного натурального числа n рассчитать сумму $1 + 3 + 5 + \ldots + 2n - 1$ и сравнить со значением n^2.
14. Для заданного натурального числа n рассчитать сумму $1 + 3 + 5 + \ldots + 2n - 1$ и сравнить со значением $n(n+1)$.

15. Для заданного натурального числа n рассчитать сумму $2 + 4 + 6 \ldots + 2n$ и сравнить со значением $n(n + 1)/2$.

16. Для заданного натурального числа n рассчитать сумму $2 + 4 + 6 \ldots + 2n$ и сравнить со значением n^2.

17. Для заданного натурального числа n рассчитать сумму $2 + 4 + 6 \ldots + 2n$ и сравнить со значением $n(n + 1)$.

18. Для заданного натурального числа n рассчитать величину $n!$.

19. Для заданного натурального числа n рассчитать величину $n!!$.

20. Для заданных 10 вещественных чисел определить, сколько из них принимает значение, большее заданного числа b.

21. Для заданных 10 вещественных чисел определить, сколько из них принимает значение, меньше заданного числа b.

22. Написать программу, которая выводит квадраты первых десяти целых положительных чисел.

23. Написать программу, которая выводит кубы первых десяти целых положительных чисел.

24. Написать программу, которая вычисляет сумму первых N целых положительных чисел. N задается во время работы программы.

25. Написать программу, которая вычисляет произведение первых N целых положительных чисел. N задается во время работы программы.

Задание 2

1. Дано целое положительное число N. Найти наименьшее целое положительное число K, квадрат которого превосходит N: $K^2 > N$.

2. Дано натуральное число N: верно ли утверждение, что в данном числе N нет такой цифры A, цифра A вводится с клавиатуры.

3. Найти наибольший общий делитель (НОД) двух натуральных чисел A и B.

4. Разработать программу, вычисляющую сумму чисел $S = 1 + 9 + 17 + 25 + \ldots$, до тех пор, пока сумма не превышает 1127.

5. Разработать программу. вычисляющую сумму чисел $S = 1 + 5 + 9 + 13 + \ldots$, до тех пор, пока сумма не превышает 13527.

6. Разработать программу, вычисляющую сумму чисел $S = 2 + 4 + 8 + 16 + \ldots$, до тех пор, пока сумма не превышает 1024.

7. Разработать программу, вычисляющую произведение чисел $S = 1 * 3 * 5 * 7 * \ldots$, до тех пор, пока сумма не превышает 35725914.

8. Разработать программу, вычисляющую произведение чисел $S = 1 * 5 * 9 * 13 * \ldots$, до тех пор, пока сумма не превышает 13527569.

9. Разработать программу, вычисляющую произведение чисел $S = 2 * 4 * 8 * 16 * \ldots$, до тех пор, пока произведение не превышает 984621024.

10. Написать программу, вычисляющую сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры (длина последовательности не ограничена). Для завершения ввода нужно нажать 0.

11. Написать программу, которая определяет максимальное число из введенной последовательности положительных чисел (длина последовательности не ограничена). Для завершения ввода нужно нажать 0.

12. Написать программу, которая определяет минимальное число из введенной последовательности положительных чисел (длина последовательности не ограничена). Для завершения ввода нужно нажать 0.

13. Написать программу для вычисления сопротивления цепи из нескольких проводников, соединенных последовательно (значения сопротивлений вводятся одно за другим и для завершения ввода нажимается 0).

14. Написать программу для вычисления сопротивления цепи из нескольких проводников, соединенных параллельно (значения сопротивлений вводятся одно за другим и для завершения ввода нажимается 0).

15. Спортсмен в первый день пробежал 10 км. Каждый следующий день он увеличивал дневную норму на 10 % от нормы предыдущего дня. Определить через сколько дней спортсмен будет пробегать более 20 км.

16. Спортсмен в первый день пробежал 10 км. Каждый следующий день он увеличивал дневную норму на 10 % от нормы предыдущего дня. Определить через сколько дней спортсмен пробежит суммарный путь более 100 км.

17. Спортсмен в первый день пробежал 3 км. Каждый следующий день он увеличивал дневную норму на 15 % от нормы предыдущего дня. Определить через сколько дней спортсмен будет пробегать более 15 км.

18. Спортсмен в первый день пробежал 3 км. Каждый следующий день он увеличивал дневную норму на 20 % от нормы предыдущего дня. Определить через сколько дней спортсмен пробежит суммарный путь более 57 км.

19. Дано целое положительное число N. Найти наименьшее целое положительное число K, куб которого превосходит N: $K^3 > N$.

20. Дано целое положительное число N. Найти наименьшее целое положительное число K, которое удовлетворяет условию N: $K^3 > N^2$.

21. Дано целое положительное число N. Найти наименьшее целое положительное число K, которое удовлетворяет условию N: $K^2 + 1 > N^2 + N^3$.

22. Дано целое положительное число N. Найти наименьшее целое положительное число K, которое удовлетворяет условию N: $K^2 > N^2 + 100$.

23. Дано целое положительное число N. Найти наименьшее целое положительное число K, которое удовлетворяет условию N: $K^4 > N^4 + N^2$.

24. Дано целое положительное число N. Найти наименьшее целое положительное число K, которое удовлетворяет условию N: $K > N^2 + 2N$.

25. Дано целое положительное число N. Найти наименьшее целое положительное число K, которое удовлетворяет условию N: $K > N + N^{1/2}$.

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема работы: ***Массивы*.**

Цель работы: овладеть навыками работы с простыми структурами данных на примере массивов.

Задание: составить консольное приложение для решения нижеприведенных задач согласно варианту, согласованному с преподавателем, вводя данные в ходе выполнения программы. Для выполнения предварительно ознакомьтесь с соответствующими разделами данного пособия.

Задание 1

1. Дан массив из *N* элементов (вещественные числа). Вычислить: 1) сумму отрицательных элементов массива; 2) произведение элементов массива, расположенных между максимальным и минимальным элементами. Упорядочить элементы по возрастанию.

2. Дан массив из *N* элементов (вещественные числа). Вычислить: 1) сумму положительных элементов массива; 2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами. Упорядочить элементы по убыванию.

3. Дан массив из *N* элементов (целые числа). Вычислить: 1) произведение элементов массива с четными номерами; 2) сумму элементов массива, расположенных между первым и последним нулевыми элементами. Преобразовать массив так, чтобы сначала располагались все положительные элементы, а потом – все отрицательные (элементы, равные 0, считать положительными).

4. Дан массив из *N* элементов (вещественные числа). Вычислить: 1) произведение элементов массива с нечетными номерами; 2) сумму элементов массива, расположенных между первым и последним отрицательными элементами. Сжать массив, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце массива элементы заполнить нулями.

5. Дан массив из *N* элементов (вещественные числа). Вычислить: 1) максимальный элемент массива; 2) сумму элементов массива, расположенных до последнего положительного элемента. Сжать массив, удалив из него все элементы, модуль которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.

6. Дан массив из *N* элементов (вещественные числа). Вычислить:1) минимальный элемент массива; 2) сумму элементов массива, расположенных между первым и последним положительными элементами. Преобразовать массив так, чтобы сначала располагались все элементы, равные нулю, а потом – остальные.

7. Дан массив из *N* элементов (вещественные числа). Вычислить: 1) номер максимального элемента массива; 2) произведение элементов массива,

расположенных между первым и вторым нулевыми элементами. Преобразовать массив так, чтобы сначала располагались все элементы, стоящие в нечетных позициях, а потом – элементы, стоящие в четных позициях.

8. Дан массив из N элементов (вещественные числа). Вычислить: 1) номер минимального элемента массива; 2) произведение элементов массива, расположенных между первым и вторым отрицательными элементами. Преобразовать массив так, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом – элементы, все остальные.

9. Дан массив из N элементов (вещественные числа). Вычислить: 1) максимальный по модулю элемент массива; 2) сумму элементов массива, расположенных между первым и вторым положительными элементами. Преобразовать массив так, чтобы элементы, равные нулю, располагались после всех остальных.

10. Дан массив из N элементов (целые числа). Вычислить: 1) минимальный по модулю элемент массива; 2) сумму модулей элементов массива, расположенных после первого равного нулю элемента. Преобразовать массив так, чтобы сначала располагались элементы, стоящие в четных позициях, а потом – стоящие в нечетных позициях.

11. Дан массив из N элементов (вещественные числа). Вычислить: 1) номер минимального по модулю элемента массива; 2) сумму модулей элементов массива, расположенных после первого отрицательного элемента. Сжать массив, удалив из него все элементы, величина которых находится в интервале [a,b], а потом – все остальные.

12. Дан массив из N элементов (вещественные числа). Вычислить: 1) номер максимального по модулю элемента массива; 2) сумму модулей элементов массива, расположенных после первого положительного элемента. Преобразовать массив так, чтобы сначала располагались все элементы, целая часть которых находится в интервале [a,b], а потом – все остальные.

13. Дан массив из N элементов (вещественные числа). Вычислить: 1) количество элементов массива, лежащих в диапазоне от A до B; 2) сумму модулей элементов массива, расположенных после максимального элемента. Упорядочить элементы массива по убыванию модулей элементов.

14. Дан массив из N элементов (вещественные числа). Вычислить: 1) количество элементов массива, равных 0; 2) сумму элементов массива, расположенных после минимального элемента. Упорядочить элементы массива по возрастанию модулей элементов.

15. Дан массив из N элементов (вещественные числа). Вычислить: 1) количество элементов массива, больших C; 2) произведение элементов массива, расположенных после максимального элемента. Преобразовать массив таким образом, чтобы сначала располагались все отрицательные элементы, а потом все положительные (элементы, равные 0, считать положительными).

16. Дан массив из N элементов (вещественные числа). Вычислить: 1) количество отрицательных элементов массива; 2) сумму модулей элементов массива, расположенных после минимального по модулю элемента. Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.

17. Дан массив из N элементов (целые числа). Вычислить: 1) количество положительных элементов массива; 2) сумму элементов массива, расположенных после последнего элемента, равного нулю. Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает 1, а потом – все остальные.

18. Дан массив из N элементов (вещественные числа). Вычислить: 1) количество элементов массива, меньших C; 2) сумму целых частей элементов массива, расположенных после последнего отрицательного элемента. Преобразовать массив таким образом, чтобы сначала располагались все элементы, отличающиеся от максимального не более, чем на 20%, а потом – все остальные.

19. Дан массив из N элементов (вещественные числа). Вычислить: 1) произведение отрицательных элементов массива; 2) сумму положительных элементов массива, расположенных до максимального элемента. Изменить порядок следования элементов в массиве на обратный.

20. Дан массив из N элементов (вещественные числа). Вычислить: 1) произведение положительных элементов массива; 2) сумму элементов массива, расположенных до минимального элемента. Упорядочить по возрастанию отдельно элементы, стоящие на четных местах, и элементы, стоящие на нечетных местах.

21. Дан массив из N элементов (вещественные числа). Вычислить: 1) произведение отрицательных элементов массива; 2) сумму элементов массива, расположенных между максимальным и минимальным элементами. Упорядочить элементы по возрастанию.

22. Дан массив из N элементов (вещественные числа). Вычислить: 1) произведение положительных элементов массива; 2) сумму элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами. Упорядочить элементы по убыванию.

23. Дан массив из N элементов (целые числа). Вычислить: 1) сумму элементов массива с четными номерами; 2) произведение элементов массива, расположенных между первым и последним нулевыми элементами. Преобразовать массив так, чтобы сначала располагались все положительные элементы, а потом – все отрицательные (элементы, равные 0, считать положительными).

24. Дан массив из N элементов (вещественные числа). Вычислить: 1) сумму элементов массива с нечетными номерами; 2) произведение элементов массива, расположенных между первым и последним отрицательными элементами. Сжать массив, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце массива элементы заполнить нулями.

25. Дан массив из N элементов (вещественные числа). Вычислить: 1) минимальный элемент массива; 2) сумму элементов массива, расположенных до первого положительного элемента. Сжать массив, удалив из него все элементы, модуль которых находится в интервале $[a, b]$. Освободившиеся в конце массива элементы заполнить нулями.

Задание 2

1. Дана матрица 5×5. Для данного натурального M найти сумму тех элементов матрицы, сумма индексов которых равна M.
2. Дана матрица 5×5. Построить одномерный массив $B(5)$, состоящий из сумм элементов строк.
3. Дана матрица 5×5. Построить одномерный массив $B(5)$, состоящий из произведений элементов столбцов.
4. Дана матрица 5×5. Построить одномерный массив $B(5)$, состоящий из наименьших значений элементов строк.
5. Дана матрица 5×5. Построить одномерный массив $B(5)$, состоящий из средних арифметических элементов строк.
6. Дана матрица 5×5. Вывести ее в транспонированном виде (поменять местами строки со столбцами).
7. Дана матрица 5×5. Вывести ее в верхнем треугольном виде (т.е. напечатать только элементы верхнего треугольника и именно в виде треугольника).
8. Дана матрица 5×5. Вывести ее в нижнем треугольном виде.
9. Результаты соревнований по прыжкам в длину представлены в виде матрицы 5×3 (5 спортсменов по 3 попытки у каждого). Указать, какой спортсмен и в какой попытке показал наилучший результат.
10. Результаты соревнований по пятиборью представлены в виде матрицы 5×5 (5 спортсменов и 5 видов соревнований), в которых указаны места, занятые каждым спортсменом в данном виде. Найти лучшего спортсмена (наименьшая сумма мест).

В последующих трех задачах использовать следующее:

Таблица футбольного чемпионата задана квадратной матрицей порядка n, в которой все элементы, принадлежащие главной диагонали, равны нулю, а каждый элемент, не принадлежащий главной диагонали, равен 3,1 или 0 (число очков, набранных в игре: 3 – выигрыш, 1 – ничья, 0 – проигрыш).

11. Найти число команд, имеющих больше побед, чем поражений.
12. Определить номера команд, прошедших чемпионат без поражений.
13. Выяснить, имеется хотя бы одна команда, выигравшая более половины игр.
14. Дана действительная матрица размера $n \times m$, в которой не все элементы равны нулю. Получить новую матрицу путём деления всех элементов данной матрицы на её наибольший по модулю элемент.

15. Дана действительная квадратная матрица порядка 12. Заменить нулями все её элементы, расположенные на главной диагонали и выше неё.

16. Дана действительная матрица размера $n \times m$. Найти сумму наибольших значений элементов её строк.

17. В данной действительной квадратной матрице порядка n найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.

18. В данной действительной матрице размера 6×9 поменять местами строку, содержащую элемент с наибольшим значением, со строкой, содержащей элемент с наименьшим значением. Предполагается, что эти элементы единственны.

19. Дана действительная матрица размера $n \times m$, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением, затем среди этих чисел выбирается наибольшее. Указать индексы элемента с найденным значением.

20. Дана целочисленная квадратная матрица порядка 8. Найти наименьшее из значений элементов столбца, который обладает наибольшей суммой модулей элементов. Если таких столбцов несколько, то взять первый из них.

21. Составить программу формирования двумерного массива $n \times n$ по следующему правилу: элементы главной диагонали приравнять 1, ниже главной диагонали – 0, а выше – сумме индексов.

22. Составить программу формирования двумерного массива из предложенного одномерного так, чтобы первая строка содержала четные по номеру элементы исходного массива, а вторая – нечетные. Предусмотреть случай нечетного количества элементов массива.

23. Составить программу формирования двумерного массива из предложенного одномерного, разделив его на два столбца.

24. Составить программу формирования двумерного массива из предложенного одномерного, разделив его на две строки.

25. Дан двумерный числовой массив. Составить программу обмена местами заданных двух его строк.

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема работы: ***Функции***.

Цель работы: Овладеть навыками использования функций в программировании и рекурсивным подходом решения задач, понять отличия рекурсивного и итеративного подхода решения задачи.

Задание: составить консольное приложение для решения нижеприведенных задач согласно варианту, согласованному с преподавателем, вводя данные в ходе выполнения программы. Для выполнения предварительно ознакомьтесь с соответствующими разделами данного пособия.

Для задания 1 обязательно использовать функции, для задания 2 обязательно использовать рекурсивный подход решения задачи.

Задание 1

1. Даны числа *a*, *b*, *c*, *d*. Получить *x*=max(*a*, *b*), *y*=max(*c*,*d*), *z*=max(*x*,*y*). Вычисление max(*k*, *m*) (большего из двух чисел *k*, *m*) оформить функцией.
2. Даны числа *a*, *b*, *c*, *d*. Получить *x*= min (*a*, *b*), *y*= min (*c*,*d*), *z*= min (*x*,*y*). Вычисление min(*k*, *m*) (меньшего из двух чисел *k*, m) оформить функцией.
3. Оформить функцию stepen(*x*, *n*) от вещественного x и целого n, вычисляющую (через последовательное умножение) x^n и проверить ее.
4. Даны координаты вершин двух треугольников. Определить, какой из них имеет большую площадь.
5. Даны координаты вершин двух треугольников. Определить, какой из них имеет больший периметр.
6. Даны четыре натуральных числа. Найти наименьшее общее кратное (НОК) для этих четырех чисел.
7. Даны четыре натуральных числа. Найти наибольший общий делитель (НОД) для этих четырех чисел.
8. Даны отрезки *a*, *b*, *c*, *d*. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника.
9. Даны отрезки *a*, *b*, *c*, *d*. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать периметр данного треугольника.
10. Дано число *n*. Определить является ли это число простым.
11. Составить программу, которая число, заданное в десятичной системе счисления, переведет в двоичную систему счисления.
12. Составить программу, которая число, заданное в десятичной системе счисления, переведет в восьмеричную.
13. Даны координаты вершин треугольника на плоскости и координаты точки внутри него. Найти минимальное расстояние от этой точки до стороны треугольника.
14. Найти высоты треугольника, заданного координатами своих вершин. Указать наименьшую из них.

15. Написать функцию, которая вычисляет сопротивление параллельной цепи, состоящей из трех проводников. Параметрами ее являются значения сопротивлений. Проверить ее в работе, написав программу с ее использованием.

16. Написать функцию, которая вычисляет сопротивление последовательной цепи, состоящей из трех проводников. Параметрами ее являются значения сопротивлений. Проверить ее в работе, написав программу с ее использованием.

17. Написать программу для вычисления косинуса угла между векторами, заданными своими координатами (скалярное произведение векторов делится на произведение модулей этих векторов).

18. Написать функцию, которая вычисляет доход по вкладу. Исходными данными для функции являются: величина вклада, процентная ставка (годовых) и срок вклада (количество дней). Проверить ее в работе, написав программу с ее использованием.

19. Написать функцию решения квадратного уравнения. Исходными данными для программы должны быть коэффициенты уравнения, а выдавать программы должна сообщение о том, есть корни или нет корней, их количество и их значение.

20. Написать программу сортировки массива целых чисел методом пузырька. Заполнение массива организовать случайными числами. Из вне вводится только размер массива.

Задание 2

1. Найти максимум одномерного массива.
2. Найти минимум одномерного массива.
3. Найти сумму цифр НАТУРАЛЬНОГО числа.
4. Найти сумму цифр ВЕЩЕСТВЕННОГО числа.
5. Найти наибольший общий делитель двух чисел.
6. Найти наименьшее общее кратное (через наибольший общий делитель): НОК=$a\times b$/ НОД.
7. Организовать бинарный поиск заданного числа в массиве (массив предварительно нужно отсортировать).
8. Вычислить значения полиномов Эрмита.
9. Вычислить значения функции Аккермана для заданных m и n.
10. Дано натуральное число n. Выведите все числа от 1 до n.
11. Дано натуральное число N. Выведите информацию о том, является ли точной степенью двойки число N.
12. Дано натуральное число N. Вычислите сумму его цифр.
13. Дано натуральное число N. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами.

14. Дано натуральное число N. Выведите все его цифры по одной, в обычном порядке, разделяя их пробелами.

15. Дано число n. Определить является ли это число простым.

16. Дано слово, состоящее только из строчных латинских букв. Проверьте, является ли это слово палиндромом. При решении этой задачи нельзя пользоваться циклами!

17. Рассчитать значение квадрата целого положительного числа n, если известна зависимость $n^2 = (n–1)^2 + 2\cdot(n–1) + 1$.

18. Имеются 2 целых числа A и B. Необходимо вывести все числа от A до B включительно. Если $A < B$, числа выводятся по возрастанию, в обратном случае – по убыванию.

19. Для заданного натурально числа N вывести последовательность Фибоначчи.

20. Рассчитать последовательность полиномов Лагерра.

ЛАБОРАТОРНАЯ РАБОТА № 4

Тема работы: ***Обработка текста.***

Цель работы: овладеть навыками обработки текстов при помощи строкового, символьного типов, регулярных выражений и закрепить навыки работы циклами и массивами.

Задание: составить консольное приложение для решения нижеприведенных задач согласно варианту, согласованному с преподавателем, вводя данные в ходе выполнения программы. Для выполнения предварительно ознакомьтесь с соответствующими разделами данного пособия.

Примечание: для задания 1 под строками понимаются массивы символьных переменных, тип данных string использовать нельзя, для задания 2 если это необходимо можно использовать тип string.

Задание 1

1 В символьную переменную вводится цифра. Вывести следующую и предыдущую цифры, считая, что за 9 следует 0, а, соответственно, нулю предшествует девятка.

2 Вывести в одну строку нечетные (по порядковому номеру) буквы латинского алфавита: a c e g...

3 Дан текст, заканчивающийся точкой. Является ли этот текст правильной записью целого числа (возможно, со знаком).

4 Дан текст, заканчивающийся точкой. Напечатать этот текст, удалив из него все цифры и знаки «+» или «–».

5 Составить программу, которая определит, является ли заданное слово перевертышем (например, «кок», «шалаш» являются).

6 Составить программу, которая проверяет правописание «жи – ши» (т.е. если в заданном тексте после «ш» или «ж» встретится «ы», то программа должна выдавать сообщение об ошибке).

7 Составить программу, которая проверяет правописание «ча – ща» (т.е. если в заданном тексте после «ч» или «щ» встретится «я», то программа должна выдавать сообщение об ошибке).

8 Составить программу, которая введенное слово напечатает в обратном порядке (например, слово «упал» превратится в «лапу»).

9 Составить программу, которая во введенном слове удалит каждую вторую букву (т.е., например, для слова «символ» напечатает «смо»).

10 Составить программу, которая во введенном слове удалит заданную букву (например, если задано слово «трактор» и буква «т», то получится «ракор»).

11 Написать программу, которая во введенной с клавиатуры строке преобразует строчные (малые) буквы русского алфавита в прописные (заглавные).

12 Написать программу, которая проверяет, является ли введенная с клавиатуры строка двоичным числом.

13 Написать программу, которая проверяет, является ли введенная с клавиатуры строка 16-тиричным числом.

14 Написать программу, которая проверяет, является ли введенная с клавиатуры строка дробным числом.

15 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Подсчитать количество слов в данной последовательности.

16 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти количество слов, начинающихся с буквы б.

17 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти количество слов, оканчивающихся на букву а.

18 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти количество слов, у которых первый и последний символы совпадают между собой.

19 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Преобразовать данную последовательность, заменяя всякое вхождение слова это на слово то.

20 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти длину самого короткого слова.

21 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Написать программу, которая в словах с дефисом меняет местами части до и после дефиса.

22 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Написать программу вывода на экран списка символов, из которых образован заданный текст.

23 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Написать программу, определяющую в заданном тексте для каждой буквы алфавита количество ее употребления.

24 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Написать программу, определяющую в заданном тексте для

каждой буквы алфавита частоту ее употребления (отношение количества употреблений буквы к количеству всех букв в тексте).

25 Дана строка из N символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Дан список из слов различной длины. Составить программу упорядочения слов по их длине.

Задание 2

Задание без вариантов. Допускается парное выполнение этого задания.

Разработать программу – клавиатурный тренажер. В ходе работы программы должны генерироваться символы, которые необходимо вводить пользователю, количество правильно введенных символов должно подсчитываться, при вводе неверного символа, счетчик правильно введенных символов должен обнуляться, по завершению работы тренажера результат верно введенных символов должен выводиться на экран. Выход из режима тренажера должен осуществляться по нажатию клавиши «Esc». При вводе неверного символа необходимо выдавать звуковое сопровождение (системный звук). При запуске приложения должен предоставляться выбор набор символов какого языка необходимо тренировать: русский или английский.

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема работы: ***Структуры.***

Цель работы: овладеть навыками использования структур в программировании.

Задание: составить консольное приложение для решения нижеприведенных задач согласно варианту, согласованному с преподавателем. Для выполнения предварительно ознакомьтесь с соответствующими разделами данного пособия. Предусмотреть возможность генерации экземпляра структуры, предоставить пользователю выбор генерации очередной записи или ручной ввод. Предусмотреть обработку ошибок ввода, при это при возникновении исключения программа не должна прерывать свою работу.

Задание

1. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, буква класса. Найти однофамильцев, обучающихся в одном классе.
2. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, буква класса. Найти тезок (одинаковые имена), обучающихся в одном классе.
3. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, буква класса. Найти двух учащихся, у которых совпадают имя и фамилия.
4. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, буква класса. Вывести фамилию и первую букву имени для всех учеников указанного извне класса.
5. Багаж пассажира характеризуется количеством вещей (целый тип) и общим весом вещей (вещественный тип). Дан список из сведений о багаже 10 пассажиров. Найти багаж, средний вес одной вещи, в котором отличается не более, чем на 0,3 кг от общего среднего веса одной вещи по всему списку.
6. Багаж пассажира характеризуется количеством вещей (целый тип) и общим весом вещей (вещественный тип). Дан список из сведений о багаже 10 пассажиров. Найти число пассажиров, имеющих более двух вещей.
7. Багаж пассажира характеризуется количеством вещей (целый тип) и общим весом вещей (вещественный тип). Дан список из сведений о багаже 10 пассажиров. Число пассажиров, количество вещей которых превосходит среднее число вещей по всему списку.
8. Багаж пассажира характеризуется количеством вещей (целый тип) и общим весом вещей (вещественный тип). Дан список из сведений о багаже 10 пассажиров. Выяснить, имеется ли пассажир, багаж которого состоит из одной вещи весом менее 30 кг.

9. Список книг состоит из 10 записей. Запись содержит поля: фамилия автора (тип string), название книги (тип строка), год издания (тип целый). Найти названия книг данного автора, изданных с 1960 г.

10. Список книг состоит из 10 записей. Запись содержит поля: фамилия автора (тип string), название книги (тип строка), год издания (тип целый). Определить, имеются ли книги с названием «Информатика» и если да, то сообщить фамилии авторов, год издания этих книг.

11. Дана структура, задающая дату вида: Struct date {int day; int month; int year;}; Пользуясь таким структурным типом, составить программу, определяющую: дату следующего (относительно сегодняшнего) дня;

12. Дана структура, задающая дату вида: Struct date {int day; int month; i nt year;}; Пользуясь таким структурным типом, составить программу, определяющую: дату предыдущего дня;

13. Дана структура, задающая дату вида: Struct date {int day; int month; int year;}; Пользуясь таким структурным типом, составить программу, определяющую: дату, которая наступит через m дней;

14. Дана структура, задающая дату вида: Struct date {int day; int month; i nt year;}; Пользуясь таким структурным типом, составить программу, определяющую: дату,которая была за m дней до сегодняшнего дня;

15. Дана структура, задающая дату вида: Struct date {int day; int month; int year;}; Пользуясь таким структурным типом, составить программу, определяющую: число суток, прошедших от заданной даты d1 до даты d2;

16. Дана структура, задающая дату вида: Struct date {int day; int month; int year;}; Пользуясь таким структурным типом, составить программу, определяющую: день недели, выпадающий на дату d1, если известно, что в первый день нашей эры был понедельник.

17. Дана структура, задающая время, вида: Struct time {int hour; int min; int sec;}; Пользуясь таким структурным типом, составить программу: определяющую предшествует ли время t1 времени t2 (в пределах суток);

18. Дана структура, задающая время, вида: Struct time {int hour; int min; int sec;}; Пользуясь таким структурным типом, составить программу: присваивающую параметру t1 время, на 1 сек большее времени t2 (учесть смену суток).

19. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса (только 8-е и 10-е классы): вывести отдельно учеников 10-х классов;

20. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса (только 8-е и 10-е классы): выяснить, на сколько человек в 8-х классах больше, чем в 9-х.

21. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, оценки по трем предметам: вывести отдельно отличников для указанного извне класса;

22. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, оценки по трем предметам: вывести отдельно двоечников (хотя бы одна двойка) для указанного извне класса;

23. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, оценки по трем предметам: вывести отдельно «хорошистов» (оценки 4 и 5) для указанного извне класса;

24. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, оценки по трем предметам: вывести отдельно «троечников» (имеется хотя бы одна тройка, но нет двоек) для указанного извне класса;

25. Дан список учащихся из 10 записей. Каждая запись имеет поля: фамилия, имя, номер класса, оценки по трем предметам: вывести список всех учеников с указанием среднего балла для каждого.

ЛАБОРАТОРНАЯ РАБОТА № 6

Тема работы: ***Коллекции*.**

Цель работы**:** изучить различные типы коллекций в языке программирования C# и их практическое применение для решения конкретных задач.

Задание: Реализуйте интерфейс для взаимодействия с вашим приложением, через консольное меню и простые команды. Протестируйте ваше приложение, используя различные сценарии. Продемонстрируйте практическое применение каждого типа коллекции в вашем приложении, объяснив, как они помогают в решении поставленной задачи. Для описания каждой из сущностей в зависимости от варианта создайте соответствующие структуры. Допускается выполнение задания в парах-тройках. Предварительно согласуйте вариант задания с преподавателем.

1. Создайте консольное приложение, которое будет моделировать систему обработки заказов в интернет-магазине. Реализуйте следующие функции, используя различные типы коллекций:

List: хранение и управление списком товаров в корзине покупателя. Разрешите добавление, удаление и изменение товаров в корзине.

Stack: механизм отката последнего действия. Позвольте пользователю отменять последние действия, такие как добавление или удаление товара из корзины.

Queue: организация очереди заказов на обработку. Когда заказ обрабатывается, он удаляется из очереди.

Сценарии: добавление товаров в корзину, отмена действий, обработка заказов и т.д.

2. Создайте приложение для управления задачами в проекте разработки программного обеспечения. Реализуйте следующие функции, используя различные типы коллекций:

List: хранение и управление списком задач в проекте. Разрешите добавление новых задач, удаление существующих и изменение приоритетов.

Stack: используйте стек для отката изменений. Когда производится изменение в проекте (например, отмена действия), данное изменение должно быть добавлено в стек. При выполнении отката изменений, изменение должно быть извлечено из стека и отменено.

Queue: организация очереди задач на выполнение. Когда разработчик заканчивает работу над одной задачей, он берет следующую задачу из очереди на выполнение.

Сценарии: добавление новых задач, изменение их приоритетов, выполнение и отмена действий.

1. Создайте консольное приложение на языке программирования C#, которое моделирует систему учета складских операций в небольшом складе. Реализуйте следующие функции, используя различные типы коллекций:

List: хранение и управление списком товаров на складе. Разрешите добавление, удаление и изменение товаров.

Stack: реализация механизма приемки товаров на склад. Когда новый товар поступает на склад, он помещается в стек. При выдаче товара со склада, товар извлекается из вершины стека.

Queue: организация очереди на отгрузку товаров. Когда заказ на отгрузку поступает, он добавляется в очередь. Отгрузка товаров происходит в порядке, соответствующем очереди.

Сценарии: приемка новых товаров, отгрузка товаров, изменение состава склада и т. д.

2. Создайте консольное приложение на языке программирования C#, которое моделирует систему обработки запросов в технической поддержке. Реализуйте следующие функции, используя различные типы коллекций:

List: хранение и управление списком запросов от клиентов. Разрешите добавление новых запросов, удаление запросов при их обработке и просмотр списка всех запросов.

Stack: организация механизма «стека вызовов» для регистрации обработанных запросов. Каждый раз, когда обработчик запроса заканчивает работу с запросом, он помещает его в стек. Это позволит осуществлять откат обработанных запросов, если необходимо.

Queue: организация очереди запросов на обработку. Запросы должны обрабатываться в порядке поступления, сначала вошел - первым вышел (FIFO).

Сценарии: когда новый запрос поступает, он добавляется в очередь. Обработчик запросов извлекает запросы из очереди и обрабатывает их. После обработки запроса, его можно отправить в стек для возможного отката, добавление новых запросов, обработка запросов в очереди и откат обработанных запросов.

1. Разработайте консольное приложение на языке программирования C#, которое будет моделировать систему управления транспортными средствами на автостоянке. Реализуйте следующие функции, используя различные типы коллекций:

List: используйте список для хранения информации о транспортных средствах, припаркованных на стоянке. Реализуйте возможность добавления, удаления и поиска транспортных средств в списке.

Stack: создайте стек для моделирования системы стоянки с ограниченным доступом. При добавлении нового транспортного средства на стоянку, его должно быть возможно поместить только вверху стека. При удалении транспортного средства оно должно быть извлечено с вершины стека.

Queue: реализуйте очередь для моделирования ожидания доступного места на стоянке. Когда на стоянке освобождается место, транспортное средство из очереди должно быть автоматически припарковано.

Сценарии: добавление новых транспортных средств, их удаление, поиск, а также управление доступом к стоянке через стек и очередь.

ЛАБОРАТОРНАЯ РАБОТА № 7

Тема работы: ***Основы объектно-ориентированного программирования.***

Цель работы: овладеть навыками работы с классами, использования механизмов инкапсуляции, наследования и полиморфизма.

Задание: составить консольное приложение для решения нижеприведенных задач согласно варианту, согласованному с преподавателем, вводя данные в ходе выполнения программы. Для выполнения предварительно ознакомьтесь с соответствующими разделами данного пособия.

Примечание: для выполнения заданий необходимо разбиться на группы по 2–3 человека. Задание для каждого варианта звучит следующим образом: «В программе продемонстрировать функциональность разработанных классов». Все данные для заполнения полей необходимо вводить с клавиатуры. Обязательно использование абстрактных классов, интерфейсов, конструкторов и деструкторов.

Задание 1

1. Создать класс EngMer для работы с английскими мерами длины: футами и дюймами, при этом учтем, что 1 фут = 12 дюймов. Длина объекта будет задаваться парой чисел (фунты и дюймы), нужно реализовать: сложение и вычитание длин, умножение и деление длин, сравнение длин.
2. Создать класс vector3D, задаваемый тройкой координат. Обязательно должны быть реализованы: сложение и вычитание векторов, скалярное произведение векторов, умножение на скаляр, сравнение векторов, вычисление длины вектора, сравнение длины векторов.
3. Создать класс EngMoney для работы с устаревшей денежной системой Великобритании. В ней использовались фунты, шиллинги и пенсы. При этом: 1 фунт = 20 шиллингов, 1 шиллинг =12 пенсов. Денежные суммы будут задаваться в фунтах, шиллингах и пенсах и результат выдаваться также в этих величинах. Должны быть реализованы: сложение и вычитание, умножение и деление, сравнение сумм.
4. Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long int для рублей и типа int – для копеек. Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой. Реализовать сложение, вычитание, деление сумм, деление суммы на дробное число, умножение на дробное число и операцию сравнения.
5. Создать класс Triangle для представления треугольника. Поля данных должны включать углы и стороны. Требуется реализовать операции: получения и изменения полей данных, вычисления площади, вычисления периметра, вычисления высот, а также определения вида прямоугольника (равносторонний, равнобедренный или прямоугольный).

6. Создать класс Angle для работы с углами на плоскости, задаваемыми величинами в градусах и минутах. Обязательно должны быть реализованы: перевод в радианы, приведение к диапазону 0–360, увеличение и уменьшение угла на заданную величину, получение синуса, сравнение углов.

7. Создать класс Point для работы с точками на плоскости. координаты точки – декартовы. Обязательно должны быть реализованы: перемещение точки по оси X, перемещение по оси Y, определение расстояния до начала координат, расстояние между двумя точками.

8. Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где a – числитель, b – знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы операции: сложения, вычитания, умножения, деления и сравнения дробей, причем результат должен приводиться в виде несократимой дроби (т.е. результат нужно сокращать).

9. Создать класс Date для работы с датами в формате «год.месяц.день». Дата представляется структурой с тремя полями типа unsigned int: для года, месяца и дня. Класс должен включать операции: вычисление даты через заданное количество дней от указанной, вычитание заданного количества дней из даты, определение високосного года, присвоение и получение отдельных частей (год, месяц, день), сравнение дат (равно, до, после), вычисление количества дней между датами.

10. Создать класс Time для работы со временем в формате «час:минута:секунда». Обязательными операциями являются: вычисление разницы между двумя моментами времени в секундах, сложение времени и заданного количества секунд, вычитание из времени заданного количества секунд, сравнение моментов времени, перевод в секунды, перевод в минуты (с округлением до целой минуты).

11. В морской навигации координаты точки измеряются в градусах и минутах широты и долготы. Один градус равен 60 минутам (ранее минуту делили на 60 секунд, но сейчас минуту делят на обычные десятичные доли). Долгота измеряется от 0 до 180° восточнее (E) или западнее (W) Гринвича. Широта принимает значения от 0 до 90° севернее (N) или южнее (S) экватора. Создать класс Koord, включающий следующие три поля: типа int для числа градусов, типа float для числа минут и типа char для указания направления (N, S, W или E). Объект этого класса может содержать значение как широты, так и долготы. Предусмотреть ввод координат точки, указания полушария для указанной точки, вычисления расстояния между двумя точками. Учитывать, что по широте 1° равен 111 км, а по долготе 1° равен 111 км × cos (широты).

12. Реализовать класс Account, представляющий собой банковский счет. В классе должны быть реализованы 4 поля: фамилия владельца, номер счета, процент начисления и сумма в рублях. Необходимо выполнять следующие опе-

рации: сменить владельца счета, снять некоторую сумму со счета, положить деньги на счет, начислить проценты, перевести сумму в доллары, перевести сумму в евро, получить сумму прописью (преобразовать в числительное).

Задание 2

1. Номиналы российских рублей могут принимать значения 1, 2, 5, 10, 50, 100, 500, 1000, 5000. Копейки представить, как 0,01 (1 копейка), 0,05 (5 копеек), 0,1 (10 копеек), 0,5 (50 копеек). Создать класс Money для работы с денежными суммами. Сумма должна быть представлена полями номиналами, значениями которых должны быть количества купюр данного достоинства. Реализовать сложение сумм, вычитание сумм, деление сумм, деление суммы на дробное число, умножение на дробное число и операцию сравнения. Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой.

2. Реализовать класс Bankomat, моделирующий работу банкомата. В классе должны содержаться поля для хранения идентификационного номера банкомата, информации о текущей сумме денег, оставшейся в банкомате, минимальной и максимальной суммах, которые позволяется снять клиенту в один день. Сумма денег представляется полями номиналами 10–1000 (см. задание 12.13). Реализовать метод загрузки купюр в банкомат и метод снятия определенной суммы денег. Метод снятия денег должен проверять на корректность снимаемую сумму (не должна быть меньше минимально допустимой и больше максимально допустимой). Должен быть метод для отображения оставшейся в банкомате суммы.

3. Создать класс Fraction для работы с дробными числами. Число должно быть представлено двумя полями: целая часть – длинное целое со знаком, дробная часть – беззнаковое короткое целое. Реализовать арифметические операции сложения, вычитания, умножения и операции сравнения.

4. Создать класс Goods (товар). В классе должны быть представлены поля: наименование товара, дата оформления, цена товара, количество единиц товара, номер накладной, по которой товар поступил на склад. Реализовать методы изменения цены товара, изменения количества товара (увеличения и уменьшения), вычисления стоимости товара. Должен быть метод для отображения стоимости товара в виде строки.

5. Создать класс BitString для работы с 64-битовыми строками. Битовая строка должна быть представлена двумя полями типа unsigned long. Должны быть реализованы все традиционные операции для работы с битами: and, or, xor, not. Реализовать сдвиг влево shiftLeft и сдвиг вправо shiftRight на заданное количество битов.

6. Создать класс LongLong для работы с целыми числами из 64 бит. Число должно быть представлено двумя полями: long – старшая часть, unsigned

long – младшая часть. Должны быть реализованы арифметические операции (сложение, вычитание, умножение и деление) и сравнение.

7. Создать класс Payment (зарплата). В классе должны быть представлены поля: фамилия-имя-отчество, оклад, год поступления на работу, процент надбавки, подоходный налог, количество отработанных дней в месяце, количество рабочих дней в месяце, начисленная и удержанная сумма. Реализовать методы: вычисления начисленной суммы, вычисления удержанной суммы, вычисления суммы, выдававаемой на руки, вычисления стажа. Стаж вычисляется как полное количество лет, прошедших от года поступления на работу, до текущего года. Начисления представляют собой сумму, начисленную за отработанные дни, и надбавки. Удержания представляют собой отчисления в пенсионный фонд (1% от начисленной суммы) и подоходный налог (13%).

8. Создать класс Ship, который будет содержать данные об учетном номере корабля и его координатах. Для хранения координат используйте поля типа Koord (см. задание 12.11). Разработайте методы для ввода данных о корабле, о выводе его координат (с указанием полушария), метод для вычисления расстояния между кораблями (см. задание 12.11).

9. Создать класс Binary1, который будет содержать число в двоичной системе (в отдельном поле – целая часть, в другом поле – дробная часть). Разработайте методы для ввода двоичных чисел (с дробной частью!), вывода двоичных чисел, методы для вычисления суммы и произведения двоичных чисел.

10. Создать класс Binary2, который будет содержать число в двоичной системе (в отдельном поле – целая часть, в другом поле – дробная часть). Разработайте методы для ввода двоичных чисел (с дробной частью!), вывода двоичных чисел, методы для вычисления разности и деления двоичных чисел.

11. Создать класс Hexadec1, который будет содержать число в 16-ричной системе (в отдельном поле – целая часть, в другом поле – дробная часть). Разработайте методы для ввода 16-ричных чисел (с дробной частью!), вывода 16-ричных чисел, методы для вычисления суммы и произведения 16-ричных чисел.

12. Создать класс Hexadec2, который будет содержать число в 16-ричной системе (в отдельном поле – целая часть, в другом поле – дробная часть). Разработайте методы для ввода 16-ричных чисел (с дробной частью!), вывода 16-ричных чисел, методы для вычисления разности и деления 16ричных чисел.

ЛИТЕРАТУРА

1. Download Visual Studio Code – Mac, Linux, Windows [Электронный ресурс]. – Режим доступа: https://code.visualstudio.com/download (дата обращения: 09.01.2024).

2. Оболочка Bash – шпаргалка для начинающих [Электронный ресурс]. – Режим доступа: https://tproger.ru/translations/bash-cheatsheet (дата обращения: 09.01.2024).

3. Документация по C# [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp (дата обращения: 15.02.2020).

4. Кнут Д.Э. Искусство программирования. – Т. 1: Основные алгоритмы. – 3-е изд.; пер. с англ. – М.: Вильямс, 2016. – 720 с.

5. ГОСТ 19.701–90 (ИСО 5807–85). Единая система программной документации (ЕСПД). Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.

6. Орлов С.А. Технологии разработки программного обеспечения. Разработка сложных программных систем: учеб. пособие для вузов. – СПб.: Питер, 2002. – 464 с.

7. Грекул В.И. Проектирование информационных систем. Курс лекций: учеб. пособие для вузов / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. – М.: Интернет-университет информационных технологий, 2005. – 298 с.

ПРИЛОЖЕНИЕ А

Форма титульного листа отчета по лабораторной работе

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра безопасности информационных систем (БИС)

ЛИНЕЙНАЯ ПРОГРАММА

Отчет по лабораторной работе №1
по дисциплине «Основы программирования»

Студент гр. (номер)
_____ И.О. Фамилия
___(дата)_

Руководитель
Доцент каф. БИС, к.т.н.
_____ _____ С.С. Харченко
оценка
___(дата)_

Томск 2026

Учебно-методическое издание

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C#

С.С. Харченко, составитель

Учебное пособие по лабораторным работам

Верстка – В.М. Бочкаревой
Печатается без корректуры, в авторской редакции

В-Спектр (ИП Бочкарева В.М.)
Подписано к печати 03.06.2026.
Формат 60×84 1/16. Печать трафаретная.
Печ. л. 6,4. Тираж 100 экз. Заказ 10.

Тираж отпечатан ИП В.М. Бочкаревой
ИНН 701701817754
634055, г. Томск, пр. Академический, 13-24, тел. 8-905-089-92-40
Эл. почта: bvm-1@list.ru