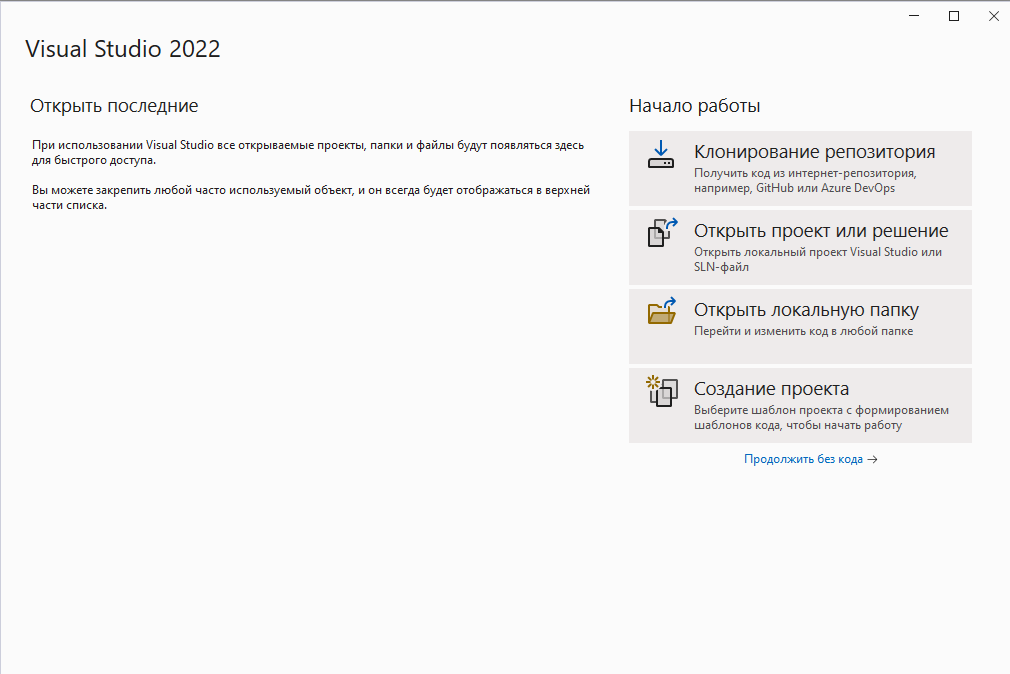
## Лабораторная работа № 1

## Часть 1. Знакомство со средой *Visual Studio* 2022

### Стартовая страница

Для того чтобы запустить интегрированную среду разработки *Visual Studio*, выполните команду: *Пуск* – *Все* *программы – Visual Studio* 2022.

Начальная страница этого инструмента может выглядеть следующим образом (см. рис. 1.1):

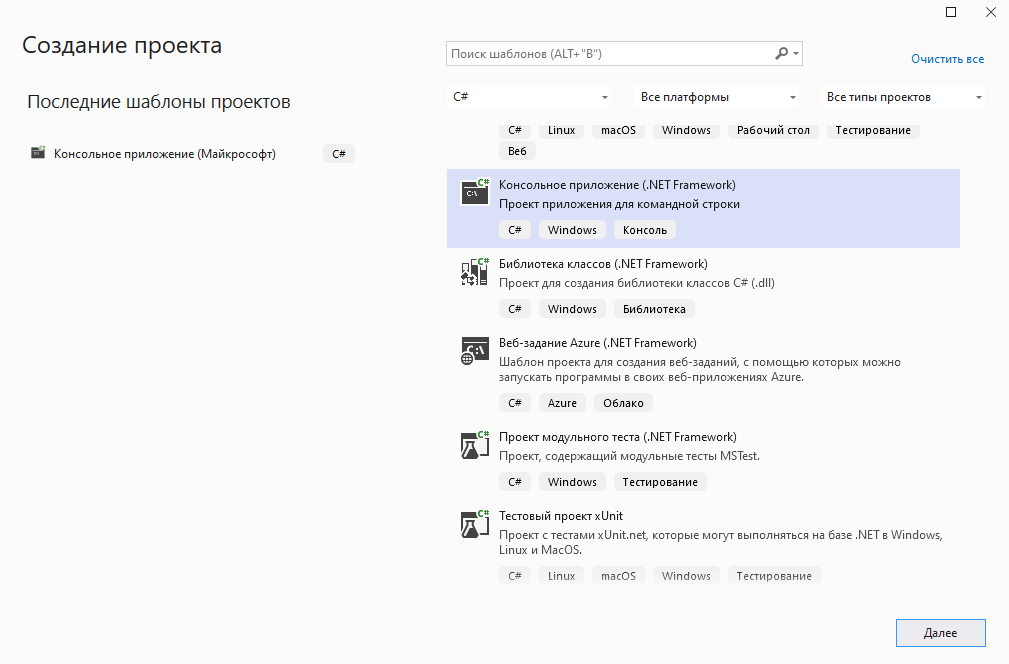


*Рис. 1.1.* Стартовая страница *Microsoft* *Visual Studio* 2022

С Начальной страницы обеспечивается быстрый доступ к открытию уже разработанных проектов и к созданию новых.

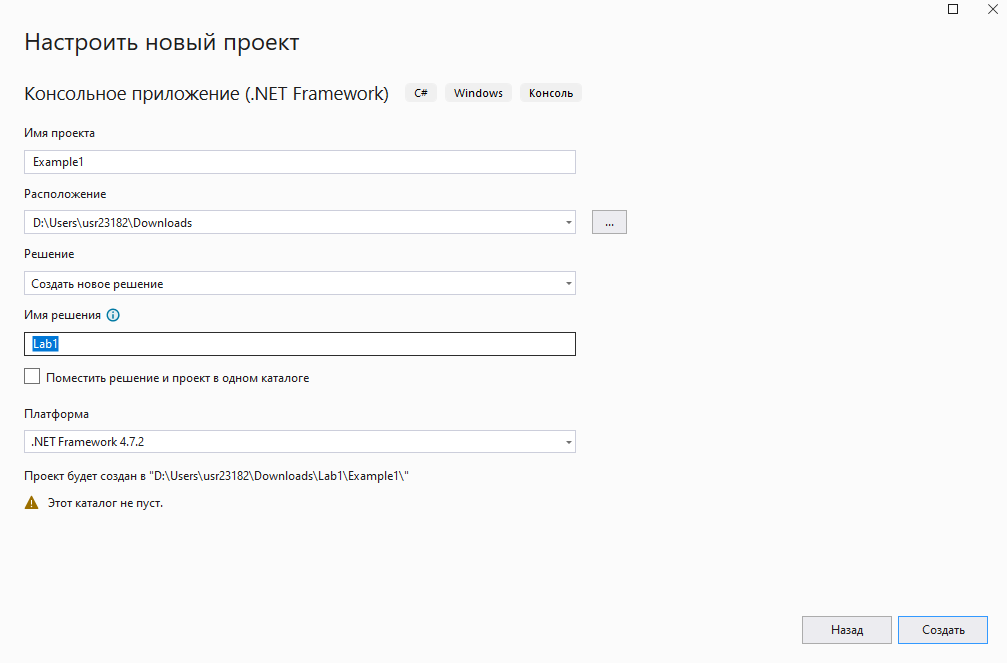
### Создание приложения

Создать новое приложение можно через пункт меню *Файл* – *Создать* – *Проект…* или на Начальной странице выбрать *Создание проекта*... После этого откроется диалоговое окно Создание проекта(рис. 1.2).



*Рис. 1.2.* Создание нового проекта

На практических занятиях мы будем создавать консольные приложения и приложения *Windows Forms* на *C*#. Сейчас создадим консольное приложение. Выберите *Консольное приложение* и нажмите *Далее*.



*Рис. 1.3.* Настройки нового проекта

В появившемся окне настройки нового проекта (рис. 1.3) указываем следующие параметры:

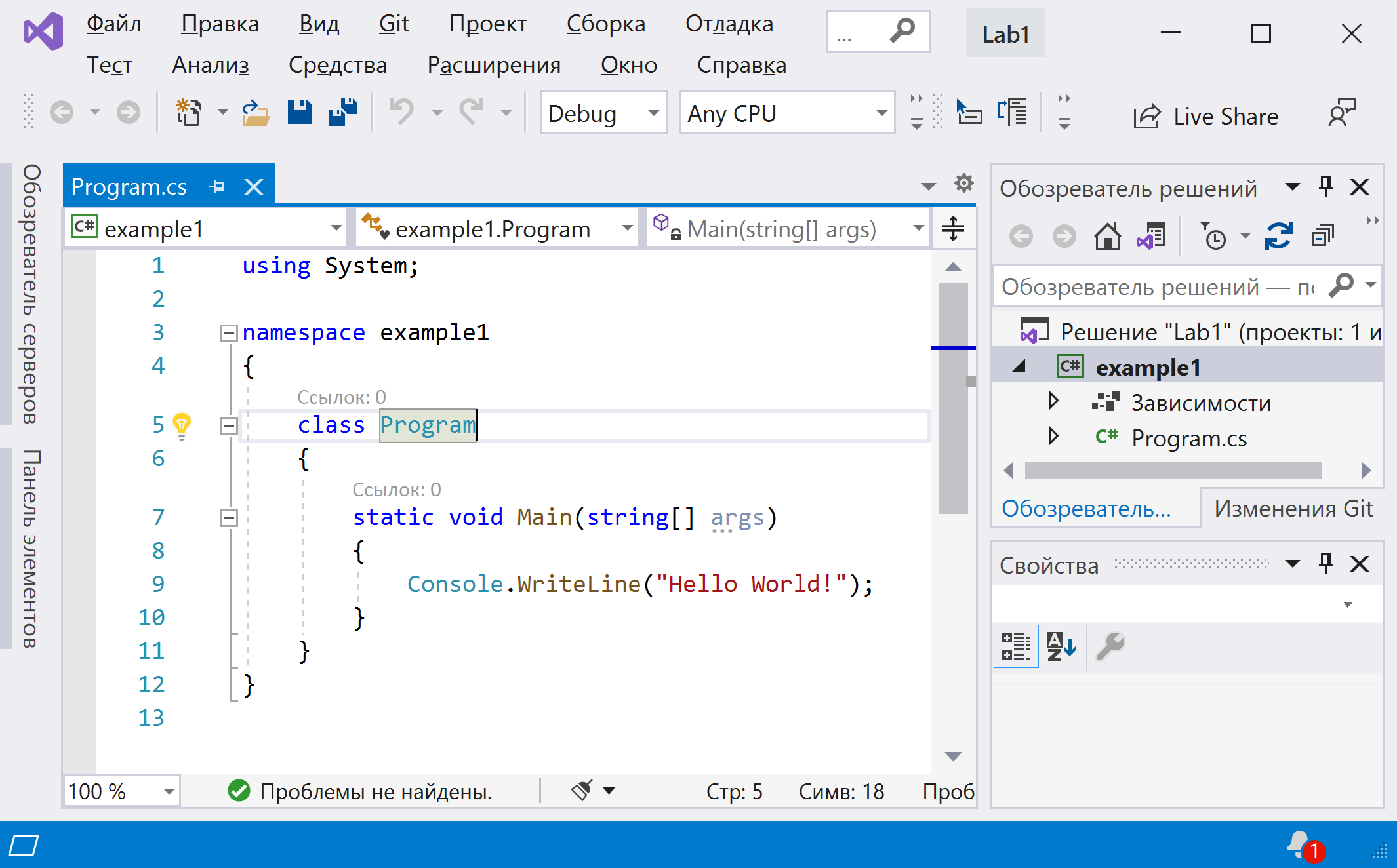
* *Имя* проекта – указываем имя проекта: *Example1*;
* *Расположение* папкиприложения, его удобнее изменять, воспользовавшись кнопкой *…*;
* *Решение: Создать новое решение;*
* *Имя решения* автоматически становится равным имени проекта, измените его на *Lab1*;
* *Поместить решение и проект в одном каталоге* – не ставим галочку, чтобы проект был частью решения.
* нажимаем на кнопку *Далее*.

После этого открывается такое окно (рис. 1.4).

Опишем несколько основных панелей. Для отображения большинства панелей можно воспользоваться пунктом меню *Вид*.

**Главное меню.** Позволяет управлять всеми аспектами работы из одного места.

**Панель инструментов** (или палитра инструментов) содержит кнопки, выполняющие некоторые наиболее часто используемые команды меню. *Visual Studio* 2019 содержит более 30 панелей инструментов. Одна из наиболее часто используемых кнопок на стандартной панели инструментов – запуск приложения ().



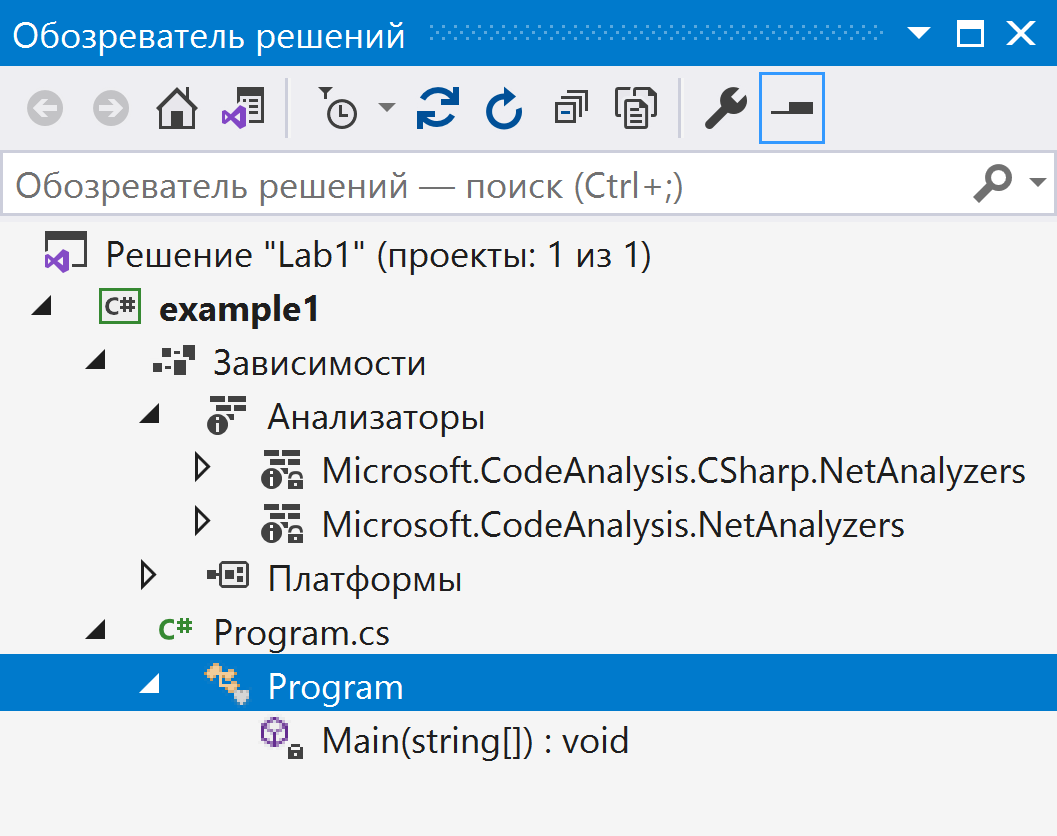
Панель

инструментов

Обозреватель решений

Главное меню

*Рис. 1.4.* Окно создаваемого проекта



*Рис. 1.5.* Окно Обозреватель решений

**Обозреватель решений** позволяет группировать и управлять множеством файлов, составляющих приложение. Решение обычно содержит несколько проектов. В проекте группируются относящиеся к нему файлы. Например, можно создать консольное приложение, *web*‑сайт, приложение *Windows Forms*, библиотеку классов и т. д. Через окно Обозреватель решений обычно добавляют новый элемент (класс, изображение, форму) в приложение. Также он используется и для доступа к этим элементам. Двойной щелчок по элементу в обозревателе решений открывает редактор кода или визуальный конструктор, связанный с файлом данного типа.

Рассмотрим, какие элементы перечислены в окне Обозреватель решений нашего приложения (рис 1.5). Во‑первых, мы видим, что решение *Lab1* содержит один проект с именем *Example1*. Также в этом окне перечислены некоторые файлы, входящие в проект. *Program.cs* – текст программы на языке *C*#.

**Редактор кода *C*#** отображает текст на языке *C*#. Необходимо отметить некоторые элементы редактора кода. Для нашего консольного приложения отображается файл *Program*.*cs* (рис. 1.6).

Новый код

Классы

в файле

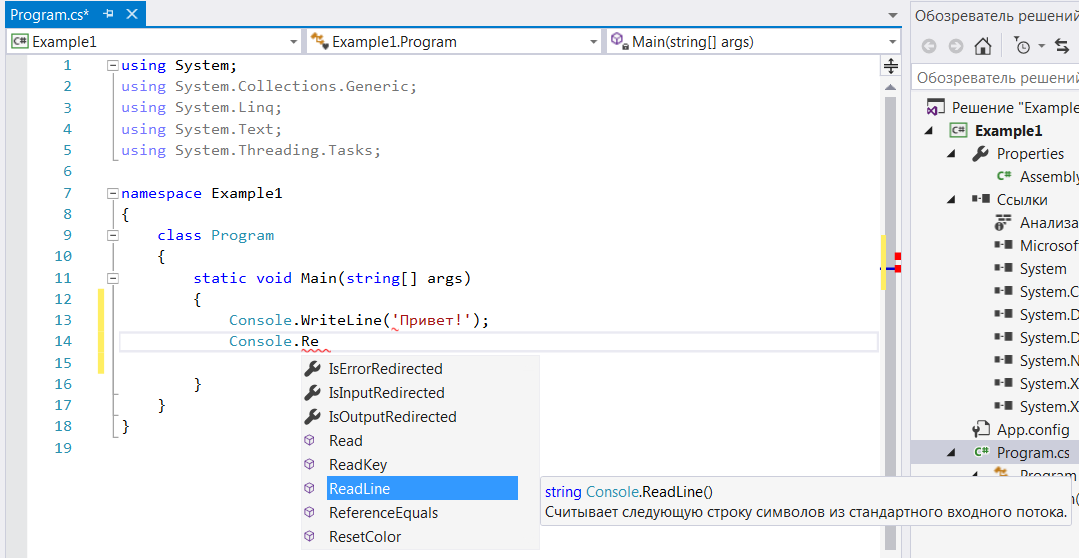
Заголовок   
окна кода

Методы, поля и свойства класса

Демонстрация работы

*IntelliSense*

Развернутая логическая область



*Рис. 1.6.* Окно редактора кода

– Код группируется в логические области (слева). Можно использовать знак минуса для того, чтобы закрыть класс, метод, свойство или другую подобную группу. Эта возможность позволяет скрывать тот код, с которым вы в данный момент не работаете. Можно также создать собственные (именованные) области кода для этой же цели.

– Некоторый код внутри областей помечается слева линиями разных цветов. Желтый цвет указывает новый код, который еще не сохранен. Линия становится зеленой после сохранения и исчезает после того, как вы вновь откроете файл после закрытия. Эта функциональная возможность помогает отслеживать места выполненных во время текущей сессии изменений в коде.

– Сам код выделяется различными цветами. По умолчанию текст имеет черный цвет, ключевые слова – синий, строковые значения – красные, комментарии – зеленые, создаваемые пользователем типы – голубые и т. д.

– Имя открытого файла показано в заголовке окна кода. Наличие звездочки указывает, что с момента последнего сохранения код изменился.

– При наборе кода запускается система интеллектуальной помощи программисту *IntelliSense*. Для быстрого нахождения в списке нужного элемента можно использовать клавиши со стрелками. При наведении указателя мыши на элемент будут показаны подробности данного элемента (текст подсказки будет справа).

– Выпадающие списки в верхней части редактора кода позволяют перемещаться между классами в файле (левый список) и методами, полями и свойствами данного класса (правый список).

Далее рассмотрим текст файла *Program.cs* проекта *Example1*. При создании любого проекта программист выбирает его тип, а *Visual Studio* в соответствии с этим создает «скелет» (заготовку, шаблон) проекта. Для консольного приложения эта заготовка выглядит следующим образом.

using System;

namespace example1

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

В первом разделе директивы *using* разрешают использовать имена стандартных классов из пространства имен *System* непосредственно без указания имени пространства, в котором они были определены.

Ключевое слово *namespace* создает для проекта свое собственное пространство имен, которое по умолчанию называется именем проекта (в нашем случае *example1*), но можно задать и другое имя. Пространство имен ограничивает область применения имен, делая его осмысленным только в рамках данного пространства. Благодаря этому можно давать имена программным объектам, не задумываясь о том, что при взаимодействии приложений может возникнуть конфликт при совпадении имен программных объектов.

Проект представляет собой совокупность взаимодействующих между собой классов. Автоматически создается класс с именем *Program*, содержащий один метод – *Main*(). Этот метод является точкой входа в программу, которой автоматически передается управление в момент запуска проекта, т. е. именно с данного метода начнется выполнение любого приложения.

Тело метода *Main*() содержит один оператор:

static void Main(string[] args)

{

**Console.WriteLine("Hello World!");**

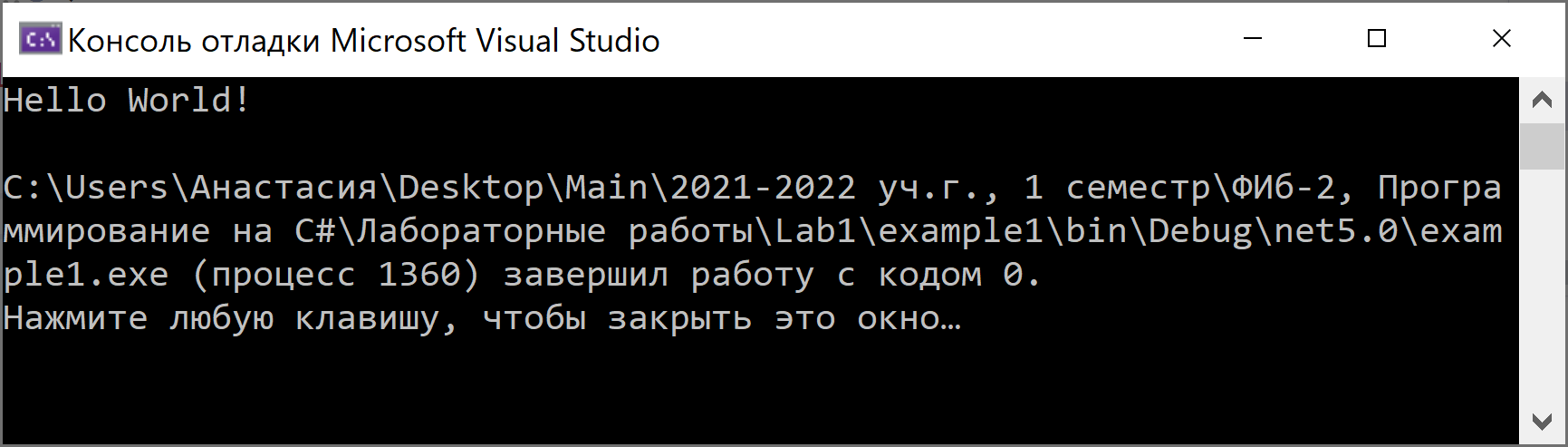
}

Здесь *Console* – имя стандартного класса из пространства имен *System*. Его метод *WriteLine*() выводит на экран текст, указанный в двойных кавычках.

### Запуск приложения

Выберите команду *Начать отладку* из меню *Отладка*, или щелкните по кнопке  панели быстрого доступа, или нажмите *F*5.

Если программа написана без ошибок, то сообщение выведется в консольном окне, которое мелькнет и быстро закроется (зависит от настроек *Сервис – Параметры – Отладка – Автоматически закрыть консоль при остановке отладки*). Чтобы просмотреть сообщение, нужно нажать клавиши <*Ctrl+F*5> или выполнить команду *Отладка – Запуск без отладки.* В нашем случае откроется следующее консольное окно (рис. 1.7):



*Рис. 1.7.* Консольное окно

При нажатии на любую клавишу приложение закрывается.

Чтобы запустить наше приложение с отладкой, но увидеть выводимое сообщение, добавим в метод *Main*() еще один оператор (если в настройках установлено автоматически закрывать окно):

static void Main(string[] args)

{

Console.WriteLine("Hello World!"); // вывод сообщения

Console.ReadKey();

}

Запускаем приложение с отладкой (например, нажав *F*5) и видим консольное окно, в котором выводится сообщение.

Метод *ReadKey*() класса *Console* будет ожидать, пока не будет нажата какая‑либо клавиша, и только тогда консольное окно закроется.

Обратите внимание, что двумя косыми чертами отделяются комментарии. В *Visual Studio* они отображаются зеленым цветом.

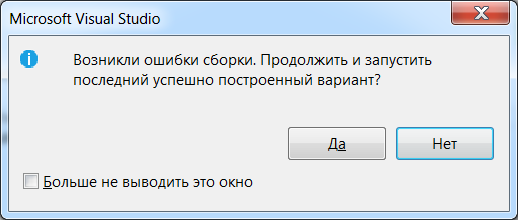
Во время выполнения приложения на палитре инструментов появляется несколько дополнительных иконок, в частности  *Остановить отладку* (<*Shift+F*5>).

Закройте приложение, для этого воспользуйтесь крестиком в правом верхнем углу запущенной формы , или нажмите комбинацию клавиш <*Alt+F*4>, или кнопку , или на клавиатуре нажмите любую клавишу, чтобы выполнился метод *ReadKey*(). При этом исчезнет та дополнительная часть палитры инструментов, которая отображалась, пока приложение было запущено.

Если случайно приложение осталось открытым, многие команды будут недоступными (например, будет нельзя вносить изменения в код).

Изменим наш код – внесем в него ошибку. Например, сотрите точку с запятой после строки, выводящей сообщение.

При нажатии на клавишу *F*5 откроется диалоговое окно, в котором выведется сообщение о том, что обнаружена ошибка, и вопрос, продолжать ли работу дальше (рис. 1.8.).



*Рис. 1.8.* Сообщение об ошибке

Если ответить *Да*, то будет выполнена предыдущая удачно скомпилированная версия программы, иначе процесс будет остановлен и станет активным окно Список ошибок, в котором будут перечислены ошибки.

## Организация ввода‑вывода данных

В языке *C*# отсутствуют операторы ввода и вывода. Вместо них используются специальные объекты, в частности стандартный класс *Console*, определенный в пространстве имен *System*. Консолью называется совокупность стандартных средств ввода и вывода – клавиатуру и монитор.

### Вывод данных

Для вывода данных на экран используются реализованные в классе *Console* методы *Write*()и *WriteLine*(). Первый из них после того, как распечатает сообщение на экране, оставляет курсор в конце выведенной строки, а второй переводит курсор в начало следующей строки.

Рассмотрим пример:

Console.WriteLine(15.8); //константа

int i = 10; //инициализация целочисленной переменной

Console.WriteLine(i); //переменная

Console.WriteLine(i \* 2 + 5); //выражение

После выполнения этих операторов на экране появятся следующие данные:



В качестве параметров методы вывода могут получать конкретные константы, значение переменных (или других идентификаторов), а также результаты выражений. Тип параметров, а это может быть любой встроенный тип, неявно преобразуется в строковый. Кроме того, можно явно указать на изменение типа с помощью метода *ToString*() – следующие два оператора идентичны по выполнению:

Console.Write(i);

Console.Write(i.ToString());

Рассмотрим еще один пример:

Console.WriteLine("i= " + i + " i^2= " + i \* i);

В этом операторе на экран выводится строка, образованная последовательным слиянием строки “*i*= ”, значения переменной *i*, строки “*i*^2= ” и значения *i*\**i*. Если изначально *i* было присвоено значение 10, то на экран выведется:



Заметьте, что для наглядного вывода результата программист должен сам заботиться о необходимых пробелах.

Рассмотрим следующие операторы:

int i = 10;

Console.WriteLine(i \* 4 + 34);

Console.Write("Result is " + i \* 4 + 34);

Что в итоге будет выведено?

После вызова метода *WriteLine*() на экран будет выведен результат вычисления выражения – число 74. Вызов метода *Write*() выведет строку “*Result is* 4034”! Объясните почему?

Существует еще один способ использования операторов вывода – форматный. Он является наиболее универсальным, поскольку позволяет не только выводить данные на экран, но и управлять форматом их вывода. Рассмотрим пример:

int x = 10;

Console.WriteLine("F({0})={1}", x, x \* x);

На экран выводится:



Это строка, формат которой задан первым аргументом метода, при этом вместо параметра {0} выводится значение *x*, а вместо {1} – значение *x*\**x*.

В общем, при форматном способе вывода первый параметр определяет формат выходной строки, а следующие нумеруются с нуля и подставляются на соответствующие места, обозначенные числами в фигурных скобках. Параметры перечисляются через запятую.

Если в фигурных скобках в строке – первом параметре – указывается число, превышающее максимальный номер параметров для подстановки, то при выполнении возникает ошибка. Пример:

Console.Write("{0} {1} {2}", x, y); //ошибка

Кроме удобства управления форматом вывода такой способ использования методов *Write*() и *WriteLine*() предоставляет еще ряд преимуществ. Рассмотрим их.

* **Использование управляющих последовательностей**

Иногда в программе бывает необходимо вывести некоторый символ, не имеющий графического обозначения (например, табуляция или звуковой сигнал), или же символ, уже имеющий для компилятора какое‑то специальное назначение (допустим, кавычки). В таких случаях можно воспользоваться управляющей последовательностью – совокупностью из двух символов, первый из которых – обратная косая черта, которые вместе интерпретируются как одиночный символ. Все управляющие последовательности приведены в табл. 1.1.

*Таблица 1.1*

**Управляющие последовательности**

|  |  |
| --- | --- |
| Символы | Назначение |
| \*a* | Звуковой сигнал |
| \*b* | Возврат на шаг назад (удаление предыдущего символа) |
| \*f* | Перевод страницы |
| \*n* | Перевод строки |
| \*r* | Возврат каретки (удаление всех символов строки, стоящих до текущего) |
| \*t* | Горизонтальная табуляция |
| \*v* | Вертикальная табуляция |
| \\ | Обратная косая черта |
| \’ | Апостроф |
| \” | Кавычки |

Пример:

Console.Write("Привет!\nПойдем в кинотеатр \"Октябрь\".");

В результате на экране появится следующее сообщение (курсор мигает сразу после сообщения, в конце второй строки):



* **Управление размером поля вывода**

Используется такая строка формата: {*n*, *m*}, где

* *n* определяет номер идентификатора из списка аргументов метода вывода *WriteLine*() или *Write*() (напомним, нумерация начинается с нуля);
* *m* – количество позиций, отводимых под значение данного идентификатора; задаёт ширину области, в которую будет помещено форматированное значение. Если *m* положительное, то значение идентификатора будет выровнено по правому краю, а если отрицательное, то по левому. Если выделенных позиций для размещения значения идентификатора окажется недостаточно, то необходимое количество позиций добавится автоматически.

Рассмотрим этот способ форматирования на примере:

int x = 103;

Console.WriteLine("|{0,5}|{0, 5}|{0,1}|", x);

Здесь мы трижды управляем размером поля вывода. Каждый раз выводится одно и то же число *x*, равное 103, но за счет параметра, указываемого в фигурных скобках вторым, способ отображения отличается. В первом случае под *x*, содержащее три разряда, выделяется пять позиций, и число выравнивается по правому краю (два пробела остается впереди). Во втором случае указывается параметр –5, т. е. под тот же *x* выделяется пять позиций, но выравнивание происходит по левому краю (два пробела остается после числа). В третьем варианте указан параметр 1, но под число выделяется не одна позиция, а столько, сколько ему необходимо по минимуму, а именно три. На экран данные выводятся в следующем виде:



* **Управление размещением вещественных данных**

С помощью предыдущего способа можно выводить вещественные числа (например, переменные типа *double*), но нельзя управлять тем, сколько позиций выделяется под целую часть, а сколько – под дробную.

double pi = Math.PI; //число Пи

Console.WriteLine("PI={0,10}", pi);

Console.WriteLine("PI={0,20}", pi);

Число Пи мы определили с помощью свойства класса *Math* (некоторые свойства и методы этого статического класса перечислены в табл. 1.2).

*Таблица 1.2*

**Некоторые свойства и методы класса *Math***

|  |  |  |
| --- | --- | --- |
| Название | Вид | Описание |
| *Abs*() | Статический метод | Возвращает абсолютное значение (модуль) числа |
| *Acos*() | Статический метод | Возвращает угол, косинус которого равен указанному числу |
| *Asin*() | Статический метод | Возвращает угол, синус которого равен указанному числу |
| *Atan*() | Статический метод | Возвращает угол, тангенс которого равен указанному числу |
| *Cos*() | Статический метод | Возвращает косинус указанного угла |
| *E* | Свойство | Представляет основание натурального логарифма, определяемое константой *e* |
| *Exp*() | Статический метод | Возвращает *e*, возведенное в указанную степень |
| *Log*() | Статический метод | Возвращает натуральный логарифм указанного числа с основанием *e* |
| *Log10*() | Статический метод | Возвращает логарифм указанного числа с основанием 10 |
| *Max*() | Статический метод | Возвращает большее из двух чисел |
| *Min*() | Статический метод | Возвращает меньшее из двух чисел |
| *PI* | Свойство | Представляет отношение длины окружности к ее диаметру, определяемое константой π |
| *Pow*() | Статический метод | Возвращает указанное число, возведенное в указанную степень |
| *Round*() | Статический метод | Округляет число до ближайшего целого |
| *Sin*() | Статический метод | Возвращает синус указанного угла |
| *Sqrt*() | Статический метод | Возвращает квадратный корень из указанного числа |
| *Tan*() | Статический метод | Возвращает тангенс указанного угла |
| *Truncate*() | Статический метод | Вычисляет целую часть заданного числа |

Переменная *pi* в нашем примере содержит 14 разрядов после запятой. В первом вызове метода *WriteLine*() мы выделили позиций меньше, чем требуется (10), остальные были добавлены автоматически, а во втором вызове – больше (20), поэтому перед числом остается несколько пробелов.



Вторым параметром в строке формата мы указываем лишь сколько разрядов выделяется под все число целиком. Для более удобного вывода вещественных чисел существует другой способ форматированного вывода: используется строка вида {n:##.###}, где

* *n* определяет номер идентификатора из списка аргументов метода вывода *Write*() или *WriteLine*(),
* ##.### – формат вывода вещественного числа (в данном случае под целую часть числа отводится две позиции, под дробную – три). Если выделенных позиций для размещения целой части значения идентификатора окажется недостаточно, то автоматически добавится необходимое количество позиций.

Пример (трижды выводим ту же переменную *pi*):

Console.WriteLine("PI={0:##.####}", pi);

Console.WriteLine("PI={0:#.###}", pi);

Console.WriteLine("PI={0:.#}", pi);

На экране будет выведено следующее:



Заметьте, что число округляется, а не просто происходит отбрасывание разрядов. Кроме того, независимо от того, сколько позиций выделяется под целую часть, отображается она всегда одинаково (вся целиком), поэтому проще всего использовать третий вариант вывода целой части. Если под дробную часть выделить больше позиций, чем имеется в числе, выведены будут лишь те разряды, которые в числе содержатся, остальные символы ‘#’ будут проигнорированы.

* **Управление форматом числовых данных**

Строка формата имеет вид {*n*: <Формат>*m*}, где

* *n* – номер идентификатора из списка параметров метода вывода *Write*() или *WriteLine*();
* *m* – описатель точности, управляет количеством позиций, выделяемых для числа, или числом разрядов в дробной части форматированного результата (см. табл. 1.3);
* <Формат> определяет тип формата данных, возможные значения перечислены в табл. 1.3:

Формат можно указывать не только заглавной, но и соответствующей прописной буквой.

*Таблица 1.3*

**Возможные значения формата**

|  |  |  |
| --- | --- | --- |
| Формат | Тип формата | Смысл параметра *m* |
| *D* | Целочисленный – используется толь­ко с целыми числами | Задается минимальное количество цифр, при необходимости к результату сначала дописываются нули |
| *E* | Экспоненциальное представление чисел (инженерный формат) | Задается количество разрядов после запятой, по умолчанию 6 |
| *F* | Представление чисел с фиксированной точкой | Задается количество разрядов после запятой, по умолчанию 2 |
| *G* | Общий формат | Задается количество разрядов на все число |
| *N* | Стандартное форматирование с использованием запятых и пробелов в качестве разделителей между разрядами | Задается количество разрядов после запятой, по умолчанию 2 |
| *X* | Шестнадцатеричный формат | Задается минимальное количество разрядов, при необходимости к результату сначала дописываются нули |
| *P* | Процентный (к результату приписывается знак %) | Задается количество разрядов после запятой, по умолчанию 2 |

Пример (в нем используются различные варианты форматного вывода):

Console.WriteLine(" D Format:{0,14:D}\t{0,10:D2}\t{0,14:D5}", 1234);

Console.WriteLine(" E Format:{0,14:E}\t{0,10:E2}\t{0,14:E7}", 1234.567);

Console.WriteLine(" F Format:{0,14:F}\t{0,10:F1}\t{0,14:F4}", 1234.567);

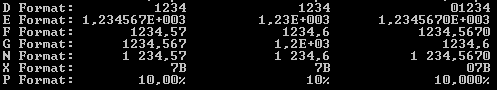
Console.WriteLine(" G Format:{0,14:G}\t{0,10:G2}\t{0,14:G5}", 1234.567);

Console.WriteLine(" N Format:{0,14:N}\t{0,10:N1}\t{0,14:N4}", 1234.567);

Console.WriteLine(" X Format:{0,14:X}\t{0,10:X1}\t{0,14:X3}", 123);

Console.WriteLine(" P Format:{0,14:P}\t{0,10:P0}\t{0,14:P3}", 0.1);

На экране выводится следующая информация:

**

### Ввод данных

Для ввода данных класс *Console* предоставляет три метода: *ReadLine*(), *Read*() и *ReadKey*().

Метод *ReadKey*() считывает один символ или обрабатывает одно нажатие пользователем функциональной клавиши (игнорируются одиночные нажатия на клавиши *Alt*, *Ctrl*, *Shift*, *CapsLock*, *NumLock*, *ScrollLock*, *PrtScr*, *Pause*; комбинации клавиш с *Alt*, *Ctrl*, *Shift* методом *ReadKey*() обрабатываются). С помощью этого метода можно определить некоторую информацию о нажатой клавише. Мы будем использовать метод *ReadKey*() преимущественно для того, чтобы окно консоли не закрывалось, пока не будет прочитана выведенная там информация.

Метод *Read*() считывает один символ (тип *char*) из входного потока. В качестве результата возвращает число (*int*) – код символа.

Пример:

Console.Write("Введите символ: ");

int i = Console.Read();

char c = (char)i;

Console.Write("Число=" + i + " Символ=" + c);

Console.ReadKey();

После запуска приложения на экран выводится сообщение «Введите символ: » и метод *Read*() ожидает ввода некоторого символа. Мы ввели символ “*q*” и нажали на клавишу *Enter*. Символ, считанный с клавиатуры (а точнее, его код), сохранился в переменной *i*. Затем с помощью явного преобразования (неявного преобразования из *int* в *char* не существует) число *i* было преобразовано в символ *c*, и вся информация (число и символ) с помощью метода *Write*() была выведена на экран.



Курсор после этого мигает непосредственно после выведенного сообщения. Как только будет нажата какая-либо клавиша, приложение будет закрыто. Если бы мы ввели не один, а несколько символов, с помощью метода *Read*() считался бы только первый из них.

В отличие от метода *Read*(), метод *ReadLine*() считывает строку символов из входного потока. В качестве результата возвращает строку (тип *string*). Пример:

Console.Write("Введите строку: ");

string s = Console.ReadLine();

Console.Write("Строка = " + s);

Console.ReadKey();

Пример выполнения:



Для того чтобы получить из вводимой строки числовое значение, что часто требуется в программах, необходимо воспользоваться преобразованием данных с помощью метода *Parse*(), реализованного для всех числовых типов данных. Для преобразования строкового представления целого числа в тип *int* используется метод *int.Parse*(), для преобразования строкового представления в вещественное – *float.Parse*() или *double.Parse*(). В случае, если входная строка имеет неверный формат и преобразование реализовать невозможно, выполнение программы прерывается из‑за ошибки.

Пример использования метода *Parse*():

Console.Write("Введите число: ");

string s = Console.ReadLine();

int i = int.Parse(s);

Console.Write("Число = " + i);

Console.ReadKey();

Пример выполнения:



Также можно воспользоваться сокращенным вариантом:

Console.Write("Введите число: ");

int i = int.Parse(Console.ReadLine());

Console.Write("Число = " + i);

Console.ReadKey();

Выполнение этого фрагмента кода будет таким же, как и предыдущего фрагмента.

Заметьте: принято, чтобы ввод с клавиатуры предварялся приглашением, а выводимые значения – пояснениями.

Рассмотрим еще один пример, в котором будут использованы методы ввода *ReadLine*() и *ReadKey*() и методы вывода *Write*() и *WriteLine*().

Console.Write("Введите ваше имя: ");

string s = Console.ReadLine();

Console.WriteLine("Мы рады Вас приветствовать, {0}, в нашей программе!", s);

Console.ReadKey();

Пример выполнения этого фрагмента кода:



В лекциях в некоторых приводимых примерах мы будем опускать вызов метода *ReadKey*(), если он не будет нести какой‑то смысловой нагрузки, кроме задержки на экране. Напомним, что если запускать без отладки (<*Ctrl*+*F*5>), то метод *ReadKey*() или другие для задержки сообщения не потребуются.

**Задания для самостоятельного выполнения**

*Замечание:* для хранения нескольких связанных проектов в одном месте (например, программ из одной лабораторной работы) можно объединить их в общее решение. Кроме того, что на диске все проекты одного решения будут храниться в одной папке, при работе в *Visual Studio* будет удобнее между ними перемещаться. Для этого при создании первого проекта укажите имя проекта, например *Task1*, а также установите галочку “*Создать каталог для решения*” и укажите имя решения *Lab1*. Чтобы добавить к решению еще один проект, выполните такую последовательность действий:

* в окне Обозреватель решений наведите мышь на верхнюю строку *Решение “Lab1” (проектов: 1)*, щелкните правой кнопкой мыши и в выпавшем контекстном меню выберите *Добавить… Создать проект*;
* в окне добавления проекта измените *Имя* на *Task2* и нажмите “*OK*”.

После этого текущее решение будет содержать два проекта, которые хранятся в одной общей папке. Необходимо щелкнуть правой кнопкой мыши на имени проекта в Обозревателе решений и выбрать команду *Назначить в качестве запускаемого проекта*. В противном случае будет запускаться предыдущий проект.

Для каждого нового проекта нужно повторить те же действия по добавлению проекта к решению и заданию проекта стартовым.

1. Напишите программу, запрашивающую у пользователя длину, ширину и высоту помещения (в метрах), а также стоимость одного квадратного метра обоев и определяющую общую стоимость обоев, необходимых для оклейки всех стен и потолка помещения. Все значения могут быть не целыми.
2. Найти значение выражения: +7 (*x*, *y* и *z* вводятся с клавиатуры; результат отобразить с точностью до пяти знаков после запятой).
3. Напишите программу, которая запрашивает с клавиатуры названия театра и спектакля, время представления и выводит сообщение (например, если введено название театра «Этсетера», название спектакля «Тихие ночи», время представления «21:00», цена билета «2350»): «Приглашаем в наш театр “Этсетера”. У нас вы можете насладиться спектаклем “Тихие ночи” в 21:00. Стоимость билета 2350 руб.».
4. Напишите программу, которая запрашивает с клавиатуры пять целых чисел и выводит на экран таблицу, в которой для каждого из этих чисел выводится само число, его модуль, квадратный корень и значение в пятой степени. При выводе нецелых значений выводите два знака после запятой. Столбики таблицы должны быть подписаны.

## Часть 2. Решение задач с использованием операторов ветвления

При решении задач из этой части лабораторной работы кроме использованного ранее материала (типы *int*, *double* и *string*; операторы ввода и вывода) вам необходимо будет использовать операторы ветвления.

Рассмотрим два примера.

**Упражнение 1.1.** Найти сумму цифр трехзначного числа, введенного с клавиатуры.

**Решение**

static void Main()

{

Console.WriteLine("Введите трехзначное число: ");

int x = int.Parse(Console.ReadLine());

if ((x < 100) || (x > 999)) Console.WriteLine("Вы ввели

не трехзначное число!");

else

{

int a = x / 100; //первая цифра

int b = x / 10 % 10; //вторая цифра

int c = x % 10; //третья цифра

int sum = a + b + c;

Console.WriteLine("Сумма цифр числа равна " + sum);

}

Console.ReadKey();

}

Поясним некоторые операторы:

* для определения, является ли число трехзначным, используется два условия, объединенных логической операцией ИЛИ (||). Каждое условие заключено в отдельные скобки, все составное условие также заключено в скобки;
* кроме логической операции ИЛИ можно использовать логические операции И (&&) и отрицание (!);
* в нашем примере используется оператор ветвления *if*. Он может иметь, а может и не иметь ветки *else*, кроме того, он может быть вложенным;
* если по синтаксису требуется один оператор, а нужно записать несколько, то эти операторы заключаются в фигурные скобки и называются блоком;
* при выполнении действий над целыми числами операция «/» выдает в качестве результата целую часть от деления, а операция «%» – остаток от деления. В нашем примере эти операции позволили определить цифры числа.

**Упражнение 1.2.** С клавиатуры вводится номер цвета радуги, нужно определить сам цвет.

**Решение**

Для решения этой задачи тоже можно воспользоваться оператором *if*, но гораздо удобнее – оператором выбора *switch*:

static void Main()

{

Console.WriteLine("Введите номер цвета: ");

int n = int.Parse(Console.ReadLine());

string s;

switch (n)

{

case 1: s = "красный"; break;

case 2: s = "оранжевый"; break;

case 3: s = "желтый"; break;

case 4: s = "зеленый"; break;

case 5: s = "голубой"; break;

case 6: s = "синий"; break;

case 7: s = "фиолетовый"; break;

default: s = "ошибка"; break;

}

if (s != "ошибка") Console.WriteLine("Ваш цвет " + s);

else Console.WriteLine("В радуге всего семь цветов!");

Console.ReadKey();

}

Любая метка *case* должна содержать оператор *break* для перехода на следующий после *switch* оператор. Метка *default* не обязательная, она «сработает» только если не подошла ни одна из предыдущих меток. В некоторых случаях для захода в метку нужно указать более одного значения – тогда они перечисляются через двоеточие вместе с ключевым словом *case*.

Обратите внимание: в условии значение строковой переменной *s* сравнивается операцией «не равно» – она записывается «!=». Операция сравнения «равно» записывается «==».

**Задания для самостоятельного выполнения**

1. Напишите программу, определяющую площадь треугольника по трем сторонам (стороны *a*, *b* и *c* вводятся с клавиатуры; выводить два знака после запятой). Предварительно проверьте, могут ли такие стороны составлять треугольник.
2. Определить, является ли семизначное число, введенное с клавиатуры, палиндромом. Программа должна содержать проверку того, что введенное число действительно семизначное.
3. Вывести на экран номер четверти, которой принадлежит точка с координатами (*x*, *y*), или же написать, что точка является началом координат или лежит на одной из осей *x* или *y*. Минимизируйте в коде количество условий.
4. Выведите на экран все возможные трехзначные числа, полученные путем перестановки цифр трехзначного числа, вводимого с клавиатуры.
5. Вводится дата (день, месяц и год числами). Определить название месяца, время года и количество дней, оставшихся до конца месяца. Программа должна проверять корректность введенной даты.
6. Студент прошел тест, оцениваемый в баллах от нуля до пяти. Вывести на экран оценку, в зависимости от балла: 5 – «отлично», 4 – «хо­ро­шо», 3 – «удовлетворительно», 2, 1 или 0 – «неудовлетворительно» (использовать оператор *switch*).
7. Напишите программу, запрашивающую имя и возраст в годах человека мужского пола. Определить возрастную категорию (до года – «младенец», от года до 11 лет – «ребенок», от 12 до 15 лет – «подросток», от 16 до 25 лет – «юноша», от 26 до 70 лет – «мужчина», от 70 до 99 лет – «старик», более 100 лет – «долгожитель»). Вывести на экран сообщение: (например, введены имя Иван и возраст 20) «Иван – юноша. Ему 20 лет.».

***Баллы:*** каждое задание оценивается в один балл.