Министерство науки и высшего образования Российской Федерации
Казанский национальный исследовательский
технологический университет

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

Учебно-методическое пособие

Казань
Издательство КНИТУ
2024

УДК 004.6(075)
ББК 32.97я7
И73

Печатается по решению редакционно-издательского совета Казанского национального исследовательского технологического университета

Рецензенты:
канд. техн. наук, доц. С. Г. Николаева
канд. техн. наук, доц. Р. Ш. Минязев

Авторы: С. А. Понкратова, А. С. Понкратов, Л. Ю. Кошкина, Д. В. Тунцев

И73 Интегрированная среда разработки для решения прикладных задач : учебно-методическое пособие / С. А. Понкратова, А. С. Понкратов, Л. Ю. Кошкина, Д. В. Тунцев; Минобрнауки России, Казан. нац. исслед. технол. ун-т. – Казань: Изд-во КНИТУ, 2024. – 84 с.

ISBN 978-5-7882-3546-2

Рассмотрена концепция интегрированной среды разработки для создания приложений решения прикладных задач. Описаны базовые алгоритмические конструкции и принципы создания графического пользовательского интерфейса. Приведены примеры решения типовых алгоритмических задач, а также контрольные вопросы и задания.

Предназначено для бакалавров направлений подготовки 18.03.01 «Химическая технология», 18.03.02 «Энерго- и ресурсосберегающие процессы в химической технологии, нефтехимии и биотехнологии», 21.03.01 «Нефтегазовое дело», 27.03.01 «Стандартизация и метрология», 27.03.02 «Управление качеством», изучающих дисциплину «Информационные технологии».

Подготовлено на кафедре химической кибернетики.

УДК 004.6(075)
ББК 32.97я7

ISBN 978-5-7882-3546-2

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Умение решать прикладные задачи с применением информационно-коммуникационных технологий (ИКТ) – одна из важнейших компетенций инженера-технолога. В рамках освоения этой компетенции обучающийся должен приобрести практический опыт решения типовых алгоритмических задач, начиная от их формализации и заканчивая тестированием готового решения.

Интегрированная среда разработки (IDE) является неотъемлемым инструментом для программистов. Она обеспечивает эффективную разработку программного кода, улучшает производительность и предлагает широкий спектр функциональных возможностей. Выбор IDE зависит от ваших предпочтений, языка программирования и конкретных требований к решению прикладной задачи.

В данном пособии будет рассматриваться язык программирования C# (Си Шарп). Он является основным языком разработки программ на платформе .NET (дот-нет) корпорации Microsoft, удачно сочетая в себе испытанные средства программирования с самыми последними новшествами и предоставляя возможность для эффективного и практичного написания приложений Web и Windows. Следует отметить, что язык Microsoft Visual C# в настоящее время является одним из наиболее продвинутых объектно-ориентированных языков.

Для успешной разработки приложений на языке программирования C# необходимо освоить следующие ключевые аспекты:

– особенности интегрированной среды разработки Visual Studio, включающей навигацию и доступ к многочисленным инструментам разработки программного обеспечения;

– создание программных кодов на языке C# с применением базовых алгоритмов обработки информации с оценкой их сложности;

– создание пользовательского графического интерфейса (освоение методов создания привлекательного и доступного интерфейса, реализующего требуемые функции; обучение работе с элементами управления с помощью дизайнера и программным способом).

В пособии приведены примеры проектов для решения прикладных задач с использованием базовых алгоритмов обработки информации, а также контрольные вопросы, позволяющие самостоятельно оценить уровень подготовки.

ОСНОВНЫЕ ПОНЯТИЯ

Прикладная задача

Прикладная задача представляет собой специфический тип профессионально ориентированной проблемы, включающий описание ситуации, имеющей непосредственную связь с практической деятельностью, решение которой осуществляется с использованием современных информационно-коммуникационных технологий.

Прикладная задача имеет практическую значимость, а именно: вопрос должен быть поставлен в таком виде, в каком он обычно ставится на практике, искомые и данные величины – взяты из реальной практической деятельности. Для решения такой задачи необходимо выполнить ее алгоритмизацию, которая является ключевым этапом при программировании и разработке программного обеспечения.

Алгоритмизация – процесс формирования алгоритма решения задачи, результатом которого является выделение этапов процесса переработки данных, формализованная запись содержания этих этапов, а также создание четких инструкций, которые компьютер может понять и выполнить.

Алгоритм решения задачи – формально описанная последовательность действий, которые необходимо выполнить для получения требуемого результата. Разработка алгоритма состоит в пошаговом описании предлагаемого решения задачи. Результатом этапа проектирования будет алгоритм, представленный в виде словесного описания или блок-схемы.

Алгоритм, описанный с помощью языка программирования, является компьютерной программой. Язык программирования задает набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под ее управлением.

Компьютерная программа представляет собой последовательность инструкций (операторов). Проверка разработанной компьютерной программы осуществляется с помощью отладки и тестирования.

Отладка – это процесс поиска и исправления ошибок в исходном коде программы. Процесс выполнения программы при конкретных вариантах значений исходных данных называется тестированием (тестом).

Этапы решения прикладной задачи с использованием информационно-коммуникационных технологий представлены в табл. 1.

Таблица 1

Этапы решения прикладной задачи

Наименование этапа	Описание
Постановка задачи	Проанализировать условия задачи (понять задачу, выделить все объекты). Составить таблицы: исходных данных, результатов, идентификаторов
Математическая формализация	Составить модель задачи, записать математические соотношения, связывающие результаты с исходными данными
Построение алгоритма	Привести расчетные соотношения, выявленные при построении модели к виду, реализуемому на ЭВМ. Построить схему алгоритма, подготовить текстуальное описание
Составление программы на языке программирования	Выбрать язык программирования, уточнить способы организации данных, записать алгоритм на выбранном языке программирования
Отладка и тестирование программы	Выполнить отладку программы, т. е. поиск и исправление ошибок: синтаксических (в записи конструкций языка программирования) и семантических (при использовании недопустимых значений). Провести тестирование с использованием контрольных примеров с учетом всех возможных ветвей алгоритма
Проведение расчетов	Провести расчеты по реальным исходным данным для получения адекватных результатов
Анализ полученных результатов	Провести анализ результатов решения задачи. При необходимости произвести уточнение математической модели с последующей корректировкой алгоритма и программы

В результате согласования между заказчиком и исполнителем всех перечисленных этапов составляется техническое задание (ТЗ) в соответствии с ГОСТ 19.201-78 «Единая система программной документации. Техническое задание. Требования к содержанию и оформлению».

IDE Visual Studio

Интегрированная среда разработки (Integrated Development Enviroment – IDE) – специализированное программное обеспечение для автоматизации процессов проектирования, редактирования, отладки, тестирования и выполнения программных кодов, а также комплексной поддержки разработчика на всех этапах жизненного цикла крупных проектов.

IDE включает в себя следующие инструменты:

– редактор кода для набора и редактирования исходного кода программ;

– сервисы для проверки и запуска программного кода;

– расширения для решения дополнительных задач разработки.

Важной составляющей IDE являются:

– представление проекта, включающее все файлы проекта и их расположение;

– схематическое представление, показывающее структуру кода в древовидном формате;

– панель навигации, помогающая быстро перемещаться по кодовой базе.

Выбор IDE основан на следующих первостепенных критериях:

– язык программирования;

– поддержка нужной операционной системы;

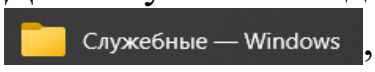
– совместная разработка;

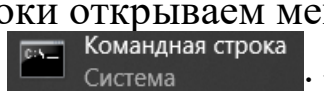
– необходимые для работы функции;

– ресурсы компьютера.

Visual Studio представляет собой среду разработки программ, созданную корпорацией Microsoft для выполнения всего цикла программного обеспечения в одном продукте[1]. Это комплексная IDE, которая предназначена для написания, редактирования, отладки и сборки кода, а затем развертывания приложения. Помимо редактирования и отладки кода, Visual Studio включает компиляторы, средства завершения кода, систему управления версиями, расширения и многие другие функции для улучшения процесса разработки программного обеспечения.

Visual Studio предназначена для создания приложений для Windows, Android и iOS, а также веб-приложений и облачных служб. В интегрированной среде разработки Visual Studio программисты имеют возможность проектировать как консольные приложения, так и приложения с графическим интерфейсом по технологии Windows Forms.

В операционной системе Windows консольное приложение работает в окне командной строки, которое также называют окном консоли. Для запуска командной строки открываем меню *Пуск*, выбираем папку 

, а далее 

. Это и будет окно консоли, в котором пользователи взаимодействуют с операционной системой или текстовым консольным приложением.

[1] https://learn.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2022

Консольное приложение – программа, которая не имеет графического пользовательского интерфейса.

Основные задачи любого консольного приложения:

1) прием входных данных;

2) отображение выходных данных на консоли командной строки.

При взаимодействии с пользователем консольное приложение работает в окне консоли, в котором отображается только текстовая информация, а ввод данных осуществляется с клавиатуры (рис. 1).

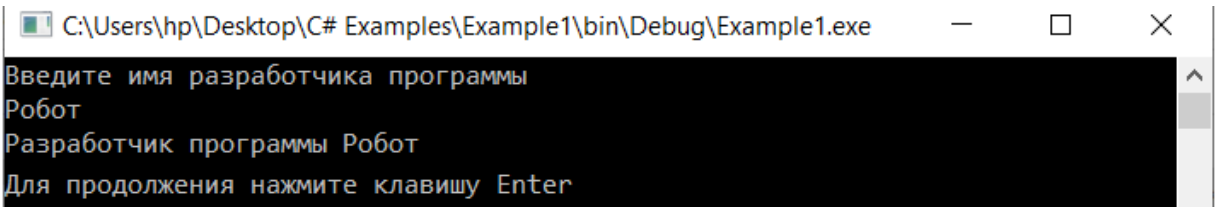


Рис. 1. Окно консольного приложения

Для разработки приложений с графическим интерфейсом в .NET Framework предназначена библиотека классов Windows Forms, которая содержит многофункциональные элементы пользовательского интерфейса: метки, кнопки, текстовые поля, поля со списком, флажки, переключатели и др. (рис. 2).

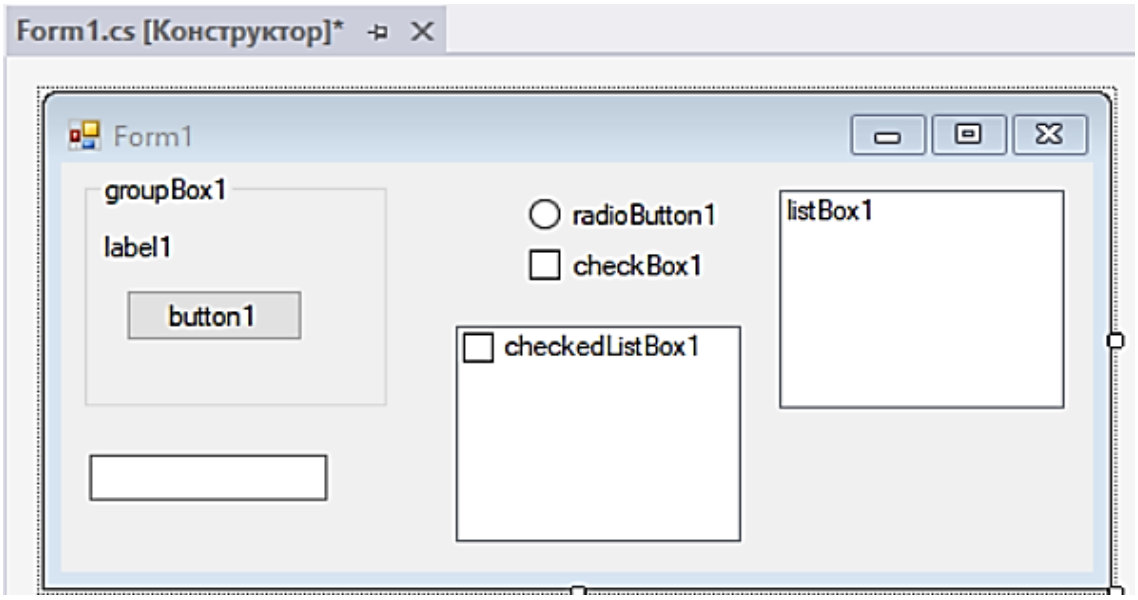


Рис. 2. Окно графического приложения

Следует отметить, что Visual Studio – многофункциональная среда разработки, которая позволяет разрабатывать приложения с помощью множества языков и платформ.

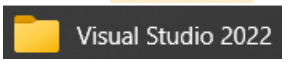
.NET – платформа надежного и безопасного многоязыкового программирования. Она основана на единой для всех языков инфраструктуре (CLI), общей системе типов (CTS), общей системе поддержки выполнения (CLR).

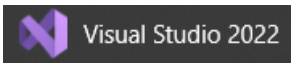
C# – простой, надежный и масштабируемый язык программирования, который статически типизирован, т. е. требует явного объявления переменных, что облегчает разработчикам поиск ошибок в коде. Данный язык также отлично подходит для компиляции, обеспечивая отличную производительность и скорость.

Для запуска Visual Studio 2022 используем один из вариантов:

а) ярлык на рабочем столе 

;

б) меню *Пуск*, папка 

и файл 

.

Для установки Visual Studio 2022 на свой компьютер переходим по ссылке полного описания установки и выбираем специальную версию для учебных целей Microsoft Visual Studio Community.

Решение в IDE Visual Studio

Решение – это разрабатываемое приложение, которое используется для управления несколькими взаимосвязанными проектами.

В интегрированной среде разработки Visual Studio принята концепция решения (solution), состоящего из одного или нескольких проектов. Независимо от того, какого вида программа разрабатывается, Visual Studio позволяет создать решение, в котором будут содержаться проекты. После этого в решение при необходимости можно добавлять другие новые или существующие проекты.

Решение можно интерпретировать как контейнер для одного или нескольких проектов, связанных между собой. Проекты внутри решения должны быть написаны на одном и том же языке программирования или иметь одинаковый тип. Например, решение может содержать веб-приложение ASP.NET, написанное на языке программирования Visual Basic, библиотеку на языке F# и WPF-приложение на языке C#. Решение позволяет пользователю открыть все эти проекты в IDE Visual Studio, а также управлять общими настройками для их создания и развертывания.

Основной тип файла решения – файл с расширением 

Обозреватель решений – главное окно, в котором пользователь работает с решениями и проектами.

На рис. 3 показан *Обозреватель решений* C#, содержащий три проекта.

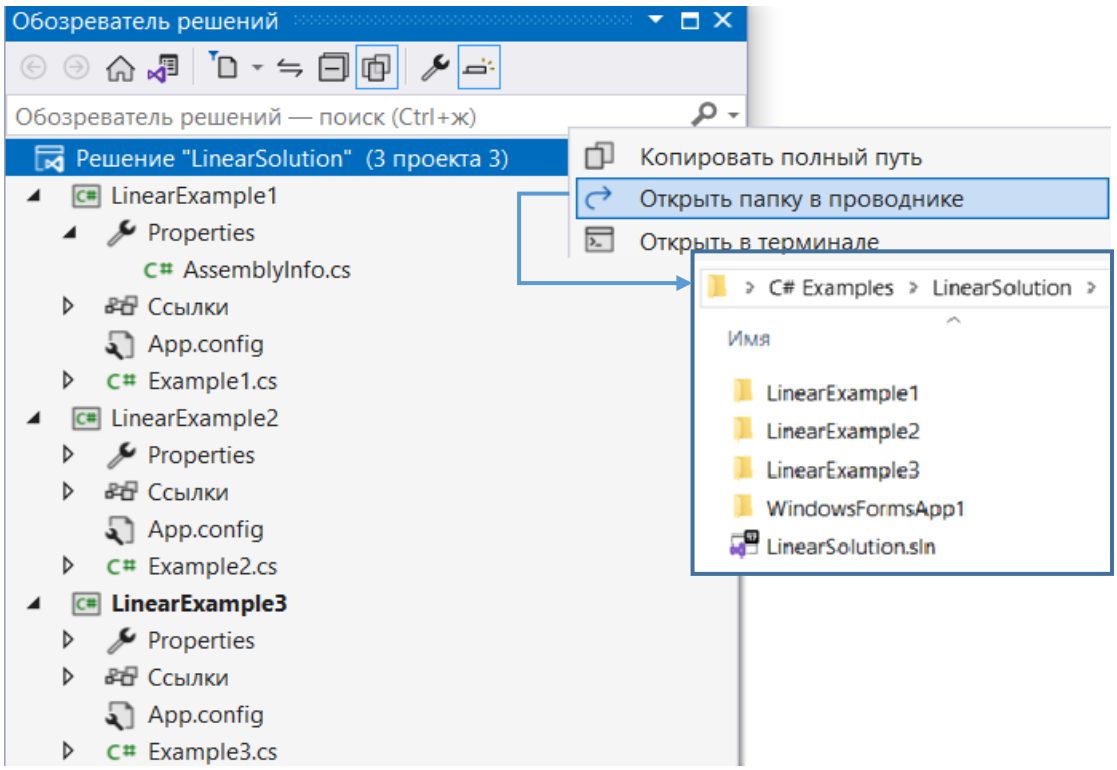


Рис. 3. Окно обозревателя решений

При открытии решения в среде Visual Studio все содержащиеся в нем проекты загружаются автоматически. Решения позволяют структурировать множество проектов.

Для просмотра папки решения в контекстном меню выбирается команда Открыть папку в проводнике. Папка *LinearSolution* содержит файл решения *LinearSolution.sln*, а также три папки файлов проектов и папку файлов интерфейса *WindowsFormsApp1*.

Проекты одного решения могут быть зависимыми или независимыми. Один и тот же проект может входить в несколько решений. После создания проекта появляется возможность просматривать его, решение и связанные с ними элементы, а также управлять ими с помощью *Обозревателя решений*.

При создании нового проекта он может быть помещен в уже существующее решение. Или может быть создано новое решение, содержащее данный проект.

Проект в IDE Visual Studio

Проект – разрабатываемое приложение (программа), состоящее из набора файлов, содержащих информацию обо всех компонентах, используемых в данном приложении. При создании проекта можно выбрать его тип, а затем Visual Studio создаст его каркас в соответствии с выбранным типом.

Проект в Visual Studio содержит:

– файл проекта – файл с расширением .csproj;

– файлы (один или несколько) исходного текста программы – с расширением .cs;

– файлы с описанием окон формы – с расширением .designer.cs;

– файлы ресурсов – с расширением .resx;

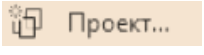
– исполняемые .exe-файлы в папке bin;

– некоторые служебные файлы, например, *App.config*.

Файл проекта – это XML-документ, содержащий все необходимые для сборки проекта сведения и инструкции – содержимое, требования к платформе, сведения о версиях, параметры веб-сервера или сервера базы данных, а также выполняемые задачи. Файл проекта автоматически создается и изменяется средой Visual Studio. Он не предназначен для редактирования разработчиком и содержит информацию, относящуюся к одной программной задаче (одному приложению).

Таким образом, структура проекта – это все файлы и библиотеки, которые можно увидеть в *Обозревателе решений*.

В IDE Visual Studio проекты подразделяются на разные категории, самая большая из которых – Windows. Она, в частности, содержит выполняемые проекты Windows Forms (окна, меню и т. д.), проекты приложений Console (пользовательский интерфейс в виде командной строки) и приложения Windows Presentation Foundation (WPF). Эта категория Windows включает также несколько типов библиотечных сборок, на которые легко ссылаться из других проектов. К ним относятся как библиотеки классов, так и библиотеки элементов управления для приложений Windows Forms и WPF. Файлы библиотек классов имеют общеизвестные расширения DLL. К этой же категории относится и тип проектов Windows Service.

Файл *Program.cs* с кодом C# (рис. 4) создается в проекте по умолчанию после выбора команды *Файл* → *Создать* → 

.

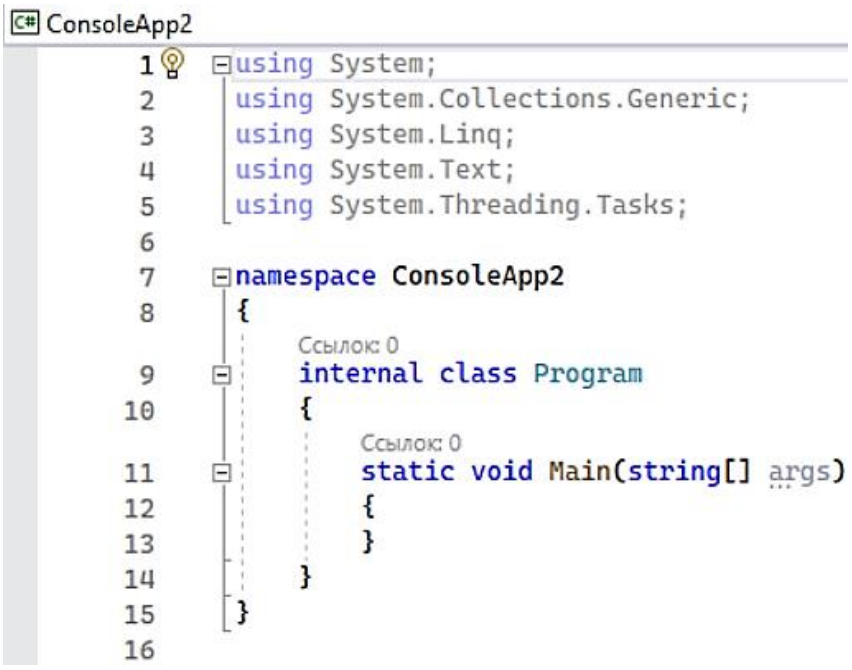


Рис. 4. Содержимое файла Program.cs по умолчанию

Поскольку C# является объектно-ориентированным языком программирования, данная программа представляет собой совокупность объектов, каждый из которых – экземпляр определенного класса. Методология программирования в этом случае подразумевает, что объект (экземпляр) выступает в роли отдельного представителя класса, имеющего конкретное состояние (атрибуты) и способы (методы), присущие данному классу и полностью им определяемые.

Итак, для создания программы в C# необходимо описать класс. Шаблон и пример описания класса имеют следующий вид:

```
class имя_класса
  {
    // Тело класса
  }
```

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

Описание класса начинается с ключевого слова *class*. Далее следует имя класса, которое можно отождествлять с названием программы, например *Program*. Тело класса (оно заключается в фигурные скобки) содержит описание главного метода *Main* (название является неизменным). Главный метод выполняется автоматически при запуске проекта, т. е. последовательно выполняются инструкции при вызове метода *Main*. Причем в программе может быть только один главный метод.

Описание главного метода начинается с ключевого слова *static*, после которого располагается ключевое слово *void*, означающее, что данный метод не возвращает результат. Главный метод – это и есть программа, заключенная в фигурные скобки. Он может принимать параметр типа *string array*, который используется для передачи аргументов командной строки.

Пространство имен *namespace имя_пространства_имен* – это своего рода контейнер, позволяющий организовать код программы в логические блоки с некоторой общей идеей. Пространство имен ограничивается фигурными скобками (см. рис. 4) – открывающей (строка *8*) и закрывающей (строка *15*). Таким образом, все, что находится между этими двумя строками, принадлежит пространству имен *ConsoleApp2* (имя пространства имен обычно совпадает с именем проекта).

Ключевое слово *using* означает, что подключается пространство имен, а *System* – название нужного пространства имен. Пространство имен *System* содержит фундаментальные и базовые классы, определяющие часто используемые типы значений и ссылочных данных, события и обработчики событий, интерфейсы, атрибуты и исключения обработки. Например, класс *Math* предоставляет константы и статические методы для тригонометрических, логарифмических и иных общих математических функций, класс *Console* – стандартные потоки для консольных приложений (входной, выходной и поток сообщений об ошибках), а класс *Convert* преобразует значение одного базового типа данных к другому базовому типу данных.

Интерфейс в IDE Visual Studio

Концепция интерфейса – одна из главных концепций в разработке программного обеспечения.

Интерфейс – это набор инструментов, который позволяет пользователю взаимодействовать с программой.

Графический пользовательский интерфейс (Graphical User Interface – GUI) представляет собой визуальную оболочку ПО, обеспечивающую взаимодействие между пользователем и вычислительной системой посредством отображаемых элементов управления и информационных компонентов. Данный тип интерфейса является ключевым компонентом системы и обеспечивает диалог с пользователем на уровне визуализированной информации.

Графические объекты пользовательского интерфейса называются *элементами управления*, к которым относятся кнопки, списки, надписи, флажки, переключатели и др.

Требования и рекомендации по элементам графического пользовательского интерфейса устанавливает ГОСТ Р ИСО 9241-161-2016 «Эргономика взаимодействия человек-система. Часть 161. Элементы графического пользовательского интерфейса».

В работе[2] предложена методика работы с требованиями к графическому интерфейсу пользователей (GUI), которая позволяет систематизировать подход к проектированию функций IT-продукта. Первое, что должен сделать аналитик, – выделить отдельные элементы интерфейса, а потом задать себе ряд вопросов по логике работы каждого из них. Ответы на эти вопросы будут определять поведение системы и позволят создать наиболее оптимальный для пользователя интерфейс.

Интегрированная среда разработки Visual Studio предлагает большое количество элементов управления для построения графического пользовательского интерфейса (GUI) при разработке приложения. Для размещения элементов на форме используется панель элементов *Toolbox*. Форма – это окно или диалоговое окно, содержащее структурированное отображение элементов управления, которое составляет пользовательский интерфейс приложения.

Создавать объекты для основных графических элементов управления позволяют классы из пространства имен *System.Windows.Forms*.

Наиболее часто используемые элементы управления представлены на рис. 5.

Каждый элемент управления характеризуется множеством атрибутов, задающих его положение на форме, размеры, цвет и поведение. Наиболее значимые из этих свойств представлены на рис. 6.

В основе технологии создания приложений с графическим интерфейсом лежат две задачи. Первая связана с созданием элементов управления графического интерфейса. Вторая подразумевает определение реакции программы на действия с элементами управления графического интерфейса, потому что с каждым элементом управления связано определенное количество событий.

---

[2] Памятка для составления требований при проектировании GUI. URL: https://about.rtlabs.ru/laboratory/gui

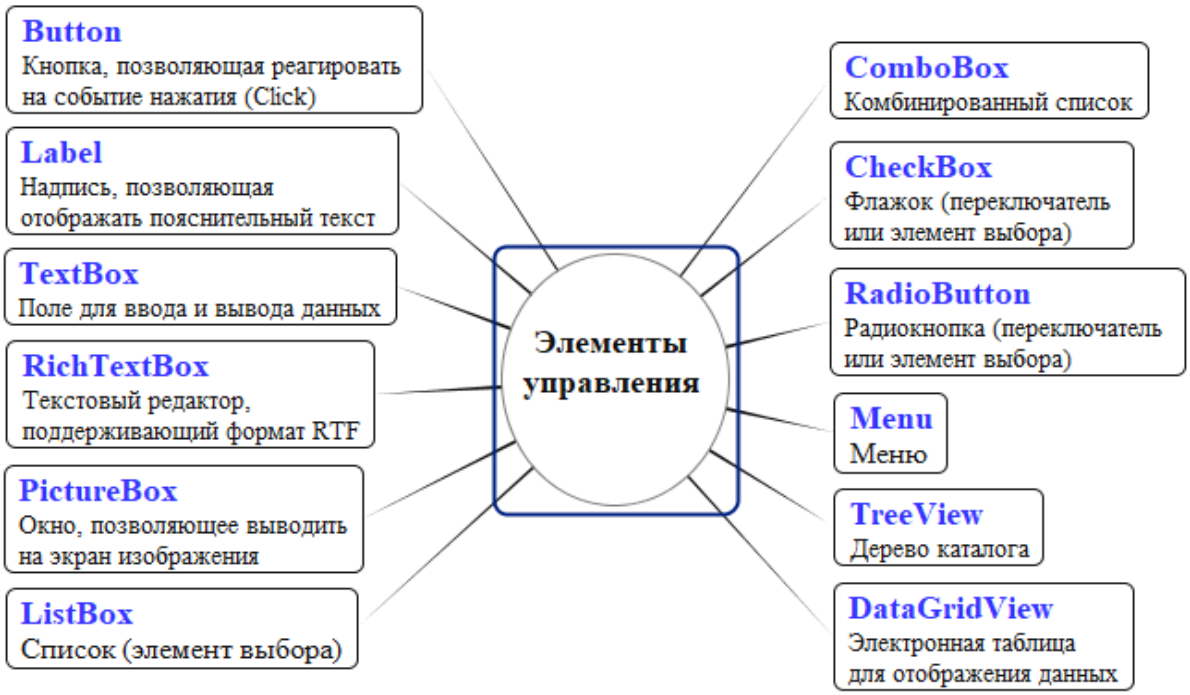


Рис. 5. Графические элементы управления

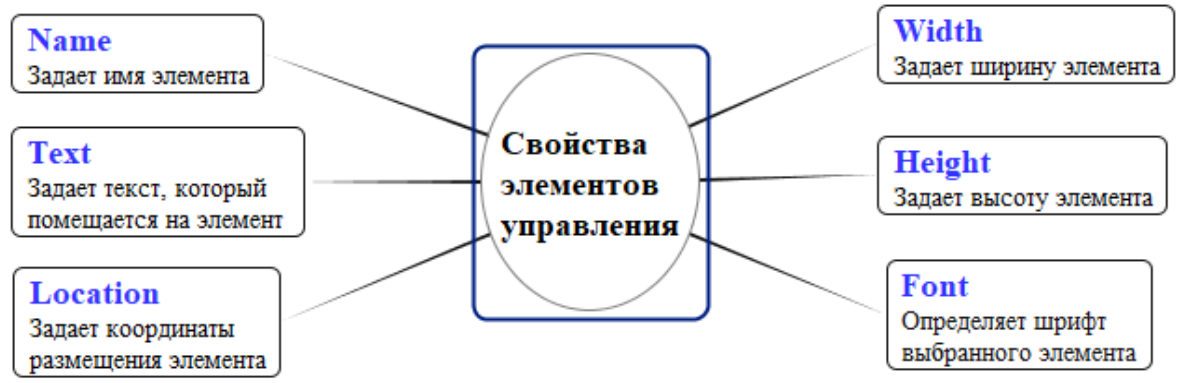


Рис. 6. Свойства элементов управления

Контрольные вопросы

1. Что представляет собой прикладная задача?
2. Что необходимо выполнить для решения прикладной задачи?
3. Перечислите этапы решения прикладной задачи.
4. Что такое интегрированная среда разработки программ?
5. Назовите основные инструменты интегрированной среды.
6. Перечислите критерии выбора IDE.
7. Какие приложения можно разрабатывать в Visual Studio?
8. Что представляет собой Visual Studio Community?
9. Перечислите варианты запуска Visual Studio.

10. Что такое решение в Visual Studio?

11. Каким файлом описывается решение в Visual Studio?

12. Что представляет собой папка решения и в каком окне Visual Studio она доступна?

13. Как посмотреть с помощью окна *Обозреватель решений* папку с решением на диске?

14. Что представляет собой проект в IDE Visual Studio?

15. Может ли проект быть частью одного или нескольких решений?

16. Могут ли решения включать другие решения?

17. Что представляет собой файл проекта и какое расширение имени файла имеет проект на языке программирования C#?

18. Перечислите особенности объектно-ориентированного программирования.

19. Расскажите об особенностях использования ключевого слова *class*.

20. Назовите ключевое слово для описания пространства имен.

21. Какое расширение имени файла имеет текст программы на языке программирования C#?

22. Что представляет собой графический интерфейс пользователя?

23. Как называются графические объекты пользовательского интерфейса?

24. Что представляет собой форма?

25. Перечислите наиболее часто используемые элементы управления и свойства элементов управления.

26. Назовите две задачи, которые лежат в основе технологии создания приложений с графическим интерфейсом.

ПРОЕКТ 1. АЛГОРИТМИЧЕСКАЯ СТРУКТУРА «СЛЕДОВАНИЕ»

Цель: разработать алгоритм решения прикладной задачи с применением базовой структуры следования (линейный алгоритм) и решить ее в соответствии с табл. 1.

1. Постановка задачи[3]

Определить потери теплоты шарообразным выпарным аппаратом, если его внутренний диаметр d_1 = 1,5 м, внешний (вместе с изоляцией) диаметр d_2 = 2,0 м, а средний коэффициент теплопроводности стенки $\lambda_{ср}$ = 0,12 Вт/(м·град). Температура рабочего тела внутри шара t_1 = 127 °С, температура наружного воздуха t_2 = 27 °С. Коэффициенты теплоотдачи: α_1 = 200 Вт/(м^2 град), α_2 = 8 Вт/(м^2 град).

Составить таблицы исходных данных (табл. 2) и результатов (табл. 3) с описанием имен переменных и типов данных. Округлить результат в программе: для коэффициента $к_ш$ – до четырех знаков после запятой; для потери теплоты Q – до двух знаков после запятой.

Таблица 2

Описание входных переменных

Наименование величины	Значение	Единицы измерения	Переменная		Тип данных
			в формуле	в программе	
Внутренний диаметр	1,5	м	d_1	d1	double
Внешний диаметр	2,0	м	d_2	d2	double
Средний коэффициент теплопроводности	0,12	Вт/(м·град)	$\lambda_{ср}$	kcp	double
Температура внутри шара	127	°С	t_1	t1	int
Температура наружного воздуха	27	°С	t_2	t2	int
Коэффициент теплоотдачи	200	Вт/(м^2 град)	α_1	k1	double
Коэффициент теплоотдачи	8	Вт/(м^2 град)	α_2	k2	double

[3] Нащокин В. В. Техническая термодинамика и теплопередача: учебное пособие для вузов. М., 1980. 469 с.

Таблица 3

Описание выходных переменных

Наименование величины	Единицы измерения	Переменная		Тип данных
		в формуле	в программе	
Коэффициент теплопередачи для шаровой стенки	Вт/град	$к_ш$	ksh	double
Потеря теплоты шарообразным выпарным аппаратом	Вт	Q	Q	double

2. Математическая формализация задачи

В задаче рассматривается передача теплоты через шаровую стенку. Перенос теплоты от одной подвижной среды (горячей) к другой (холодной) через одно- или многослойную твердую стенку любой формы называется теплопередачей.

При граничных условиях третьего рода для полого шара известны: внутренний (d_1) и внешний (d_2) диаметры, температура горячего теплоносителя внутри шара (t_1) и температура холодного теплоносителя (t_2), коэффициент теплоотдачи от горячей жидкости к внутренней поверхности шара (α_1) и коэффициент теплоотдачи от наружной поверхности шара к окружающей среде (α_2).

Потерю теплоты аппаратом вычисляем по уравнению

$$Q = к_ш \pi (t_1 - t_2). \quad (1)$$

Коэффициент теплопередачи для шаровой стенки находим по формуле

$$к_ш = \frac{1}{\frac{1}{\alpha_1 d_1^2} + \frac{1}{2\lambda}\left(\frac{1}{d_1} - \frac{1}{d_2}\right) + \frac{1}{\alpha_2 d_2^2}}. \quad (2)$$

Обратная величина, определяемая по уравнению

$$\frac{1}{к_ш} = \frac{1}{\alpha_1 d_1^2} + \frac{1}{2\lambda}\left(\frac{1}{d_1} - \frac{1}{d_2}\right) + \frac{1}{\alpha_2 d_2^2}, \quad (3)$$

называется общим термическим сопротивлением шаровой стенки.

3. Построение алгоритма

1. *Представление расчетных соотношений (1), (2) в виде инструкций линейного вида на языке программирования C#.* Для записи различных математических операций предназначен класс *Math*, который подключается через пространство имен *using System*, что позволяет использовать классы и другие типы из пространства имен *System* без необходимости указывать полное имя.

Для использования методов класса *Math* необходимо указывать *Math.метод()*. Для записи формулы (2) применяется метод *Math.Pow* (выражение, степень) – возведение в степень, для записи уравнения (1) – метод *Math.PI* (возвращение числа π):

```
ksh = 1 / (1 / (k1 * Math.Pow(2, d1)) + 1 / (2 * kcp) * (1 / d1 - 1 / d2))
        + 1 /( k2 * Math.Pow(2, d2));

Q = ksh * Math.PI*(t1 - t2);
```

В записи выражений следует правильно расставить скобки в соответствии с последовательностью арифметических действий. Чтобы не повторять каждый раз слово *Math* перед использованием данных методов, в заголовке программы можно указать директиву подключения пространства имен класса *Math* на весь класс *using static System.Math*.

2. *Построение схемы линейного алгоритма*. Схемы программ отображают последовательность операций в программе. Логическую структуру алгоритма любой сложности можно представить в виде комбинации следующих трех базовых алгоритмических структур: следование; ветвление; повторение.

Следование – самая простая из трех структур. Все инструкции этой структуры выполняются один раз, причем в том порядке, в котором они записаны. Алгоритм, основанный на данной конструкции, называется линейным. Действия алгоритма изображаются стандартными геометрическими фигурами в соответствии с ГОСТ 19.701-90 «Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».

Для составления блок-схем используются программные инструменты, позволяющие визуализировать представление об алгоритмах и решениях модели прикладной задачи.

Блок-схемы линейного алгоритма в общем виде и для *Проекта 1* представлены на рис. 7.

Для разработки схем программ используется программный инструмент StarUML[4].

3. *Текстуальное описание в программе*. Текстуальное описание в программировании C# осуществляется с помощью комментариев, которые повышают читаемость кода и помогают понять цель его разработки. Комментарий – это текст, который игнорируется компилятором.

[4] StarUML для Windows. URL: https://staruml.softonic.ru/

Комментарий в C# оформляется одним из трех способов (рис. 11) при помощи:

а) однострочного комментария //, например:

– в отдельной строке: `// объявление переменных` ;

– в строке инструкции: `d1 = 1.5; // инициализация переменной`;

б) многострочного комментария /*…*/:

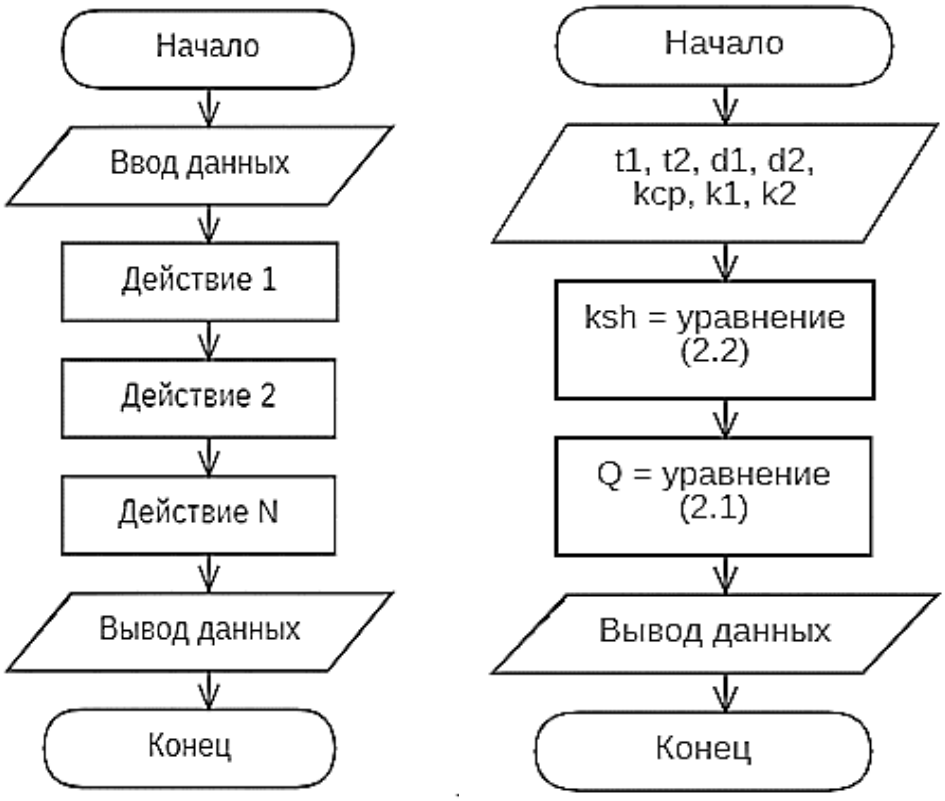
```
/* начало комментария
Многострочный
комментарий
конец комментария */
```

;

в) комментария XML (eXtensible Markup Language) – это расширяемый язык разметки, например:

```
/// <summary>
/// Главный метод. Точка входа в программу
/// </summary>
```

(чтобы вставить данный комментарий, нужно перед названием класса, поля, свойства или метода поставить тройной слеш: ///….).



а *б*

Рис. 7. Блок-схема линейного алгоритма:
а – в общем виде; б – для Проекта 1

Visual Studio поддерживает вывод подсказок, которые подгружаются из XML-комментариев. При наведении указателя мыши на метод *Main()* можно увидеть, что было написано в комментарии (рис. 8).

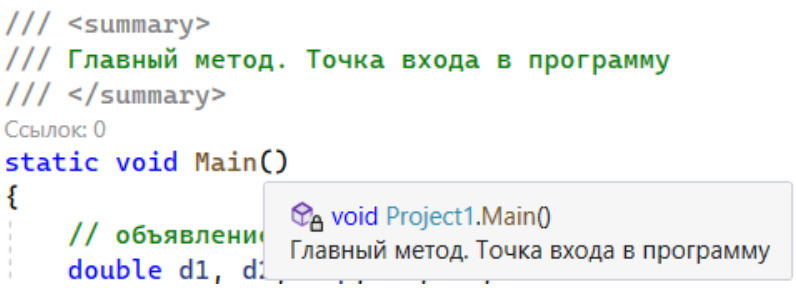


Рис. 8. XML-комментарий в C#

Основные используемые теги для объяснения кода:

а) <summary> – основной тег для общего резюме или объяснения того, что делает тот или иной фрагмент кода;

б) <returns> – тег для методов, имеющих возврат значения, где указывается ожидаемое поведение функции относительно результата ее выполнения;

в) <param> – тег для детального пояснения аргументов метода, передаваемых в функцию во время вызова, раскрывая тип, смысл и возможные ограничения соответствующего параметра.

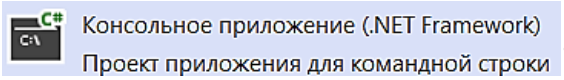
4. Составление программы на языке программирования

Программа на языке C# состоит из последовательности инструкций (законченных выражений), после каждой из которых обязательно ставится специальный символ «;»: `int t1, t2;`.

Переменная в C# – именованная область памяти, где хранятся данные определенного типа, имеющая свое имя и значение. Имя служит для обращения к области памяти, в которой хранится значение. Перед использованием любая переменная должна быть объявлена: `int t1, t2;`, а затем инициализирована: `t1 = 127; t2 = 27;`, она также может быть и объявлена, и инициализирована: `int t1 = 127;`.

Составим программу для решения прикладной задачи с применением базовой структуры следования (линейная программа).

Создадим новый проект *Project1*. Запускаем Visual Studio2022. Открываем среду разработки при помощи клавиши *Esc* или команды *Продолжить без кода*. Выбираем *Файл* → *Создать* → *Проект*, затем



. Активировав кнопку *Далее* в окне *Настроить новый проект*, задаем имя проекта *Project1* (рис. 9). Устанавливаем его место расположения, создав папку с именем *Application C#*.

Устанавливаем флажок *Поместить решение и проект в одном каталоге*. Выбираем платформу с помощью последней версии .NET Framework и нажимаем кнопку *Создать*.

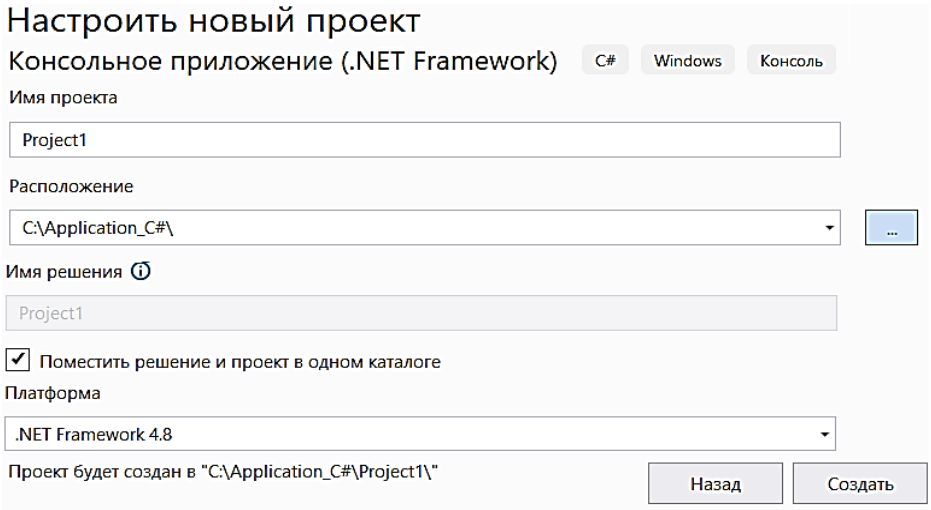


Рис. 9. Окно настройки проекта

На рис. 10 представлено окно IDE Visual Studio с программой проекта. Вводим текст программы в соответствии с рисунком (для записи арифметических выражений используем класс *Math*, который подключается через пространство имен *using System*).

По условию задачи результат в программе необходимо округлить: для коэффициента $к_{ш}$ – до четырех знаков после запятой, для потери теплоты Q – до двух знаков после запятой. Для задания формата вывода дробных значений с определенной точностью используем спецификатор *f* метода *String.Format*: `(String.Format("\tQ = {0:f2}", Q))`

Для отображения/отключения нумерации строк используем команду *Средства* → *Параметры* → *Текстовый редактор* → *C#* и включаем/отключаем на правой панели флажок *Номера строк*. В окне *Обозреватель решений*, которое находится справа, можно увидеть структуру разрабатываемого проекта. Данное инструментальное окно обеспечивает удобное визуальное представление решения, проектов и элементов (см. рис. 10). Если это окно отсутствует на экране, то его можно подключить с помощью команды меню *Вид* → 

На панели инструментов в верхней части окна *Обозреватель решений* имеются кнопки для переключения из представления решения в представление папок, фильтрации ожидающих изменений, отображения всех файлов, сворачивания всех узлов, просмотра страниц свойств, предварительного просмотра кода в редакторе кода и др. Параллельно с открытием окна проекта в папке *Application_C#* (см. рис. 10) будет создан проект с минимальным набором стандартных файлов и папок.

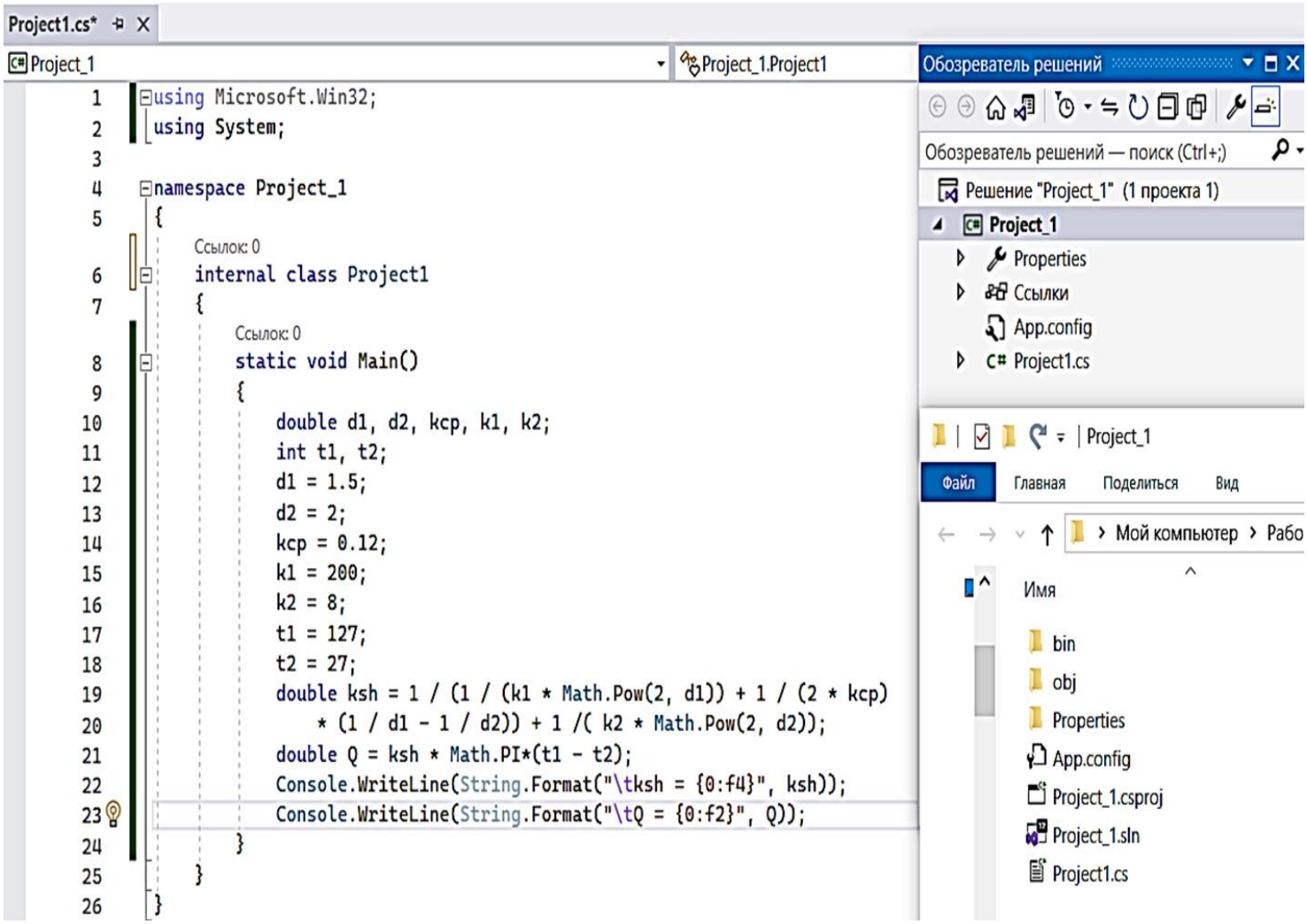


Рис. 10. Окно Проекта 1

На рис. 11 представлена программа, где, убрав слово *Math* в арифметических выражениях, указываем директиву подключения пространства имен класса *Math* на весь класс *using static System.Math*, а также размещаем все виды комментариев для документирования кода программы. Для комментирования большого фрагмента инструкций кода используем значки на панели инструментов , означающие возможность:

– закомментировать (значок) выделенные строки;

– раскомментировать (значок) выделенные строки.

Project_1 — Project_1.Project1

```
1   using System;
2   using static System.Math;
3
4   namespace Project_1
5   {
6       /* начало комментария
7        Многострочный
8        комментарий
9        конец комментария */
            Ссылок: 0
10      internal class Project1
11      {
12          /// <summary>
13          /// Главный метод. Точка входа в программу
14          /// </summary>
                Ссылок: 0
15          static void Main()
16          {
17              // объявление переменных
18              double d1, d2, kcp, k1, k2;
19              int t1, t2;
20              d1 = 1.5; // инициализация переменной
21              d2 = 2;
22              kcp = 0.12;
23              k1 = 200;
24              k2 = 8;
25              t1 = 127;
26              t2 = 27;
27              double ksh = 1 / (1 / (k1 * Pow(2, d1)) + 1 / (2 * kcp)
28                  * (1 / d1 - 1 / d2)) + 1 /( k2 * Pow(2, d2));
29              double Q = ksh * PI*(t1 - t2);
30              Console.WriteLine(String.Format("\tksh = {0:f4}", ksh));
31              Console.WriteLine(String.Format("\tQ = {0:f2}", Q));
32          }
33      }
34  }
```

Рис. 11. Структура программы Проекта 1

5. Отладка и тестирование программы

Отладка – это процесс поиска и исправления ошибок в исходном коде программы. Отладчик Visual Studio представляет собой программный инструмент, который позволяет контролировать выполнение кода, т. е. вести наблюдение за поведением программы во время выполнения и выявлять имеющиеся проблемы. Прежде чем запустить данный отладчик, проверяем код на наличие волнистых «красных» и «зеленых» линий.

«Красные» линии – это синтаксические ошибки, которые необходимо исправить, например, `kcp = 0,12;`. В данном примере ошибка указывает на неверный разделитель между целой и дробной частью, поэтому следует заменить запятую на точку: `kcp = 0.12;`.

«Зеленые» линии – это предупреждения о логических ошибках, например ошибка преобразования типа и др. Если эти линии не получается исправить, то по крайней мере нужно их проанализировать.

Для запуска программы в режиме без отладки выбираем в меню *Отладка* → *Запуск без отладки* или нажимаем кнопку на панели инструментов ▷ *Запуск без отладки*.

Проект скомпилировался и запустился. Скомпилированное приложение появится в папке проекта *C:\Application_C#\Project1\bin\Debug* с именем проекта Project_1 и расширением *exe* (приложение).

Результат выполнения программы с форматированным выводом данных представлен на рис. 12.

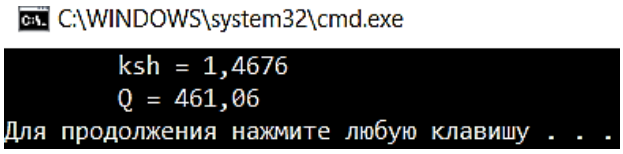


Рис. 12. Окно консольного приложения Проекта 1

Открываем программу в режиме отладки (т. е. приложение запускается с присоединенным отладчиком). Выбираем в меню *Отладка* → *Начать отладку* или активируем кнопку с зеленой стрелкой ▶ *Начать отладку*. После этого запускаем пошаговое выполнение кода с помощью функциональной клавиши *F11* (рис. 13).

IDE Visual Studio в процессе отладки предоставляет пользовательскую визуализацию для переменных и объектов. Визуализаторы отладчика доступны из подсказки данных, которая отображается при наведении указателя мыши на переменную (подсветка переменных) или из окна *Отладка* → *Окна* → *Вывод*. При запуске программы в режиме отладки значения переменных видны прямо в исходном коде, что позволяет оценить работу программы, выявить возможные ошибки и неточности.

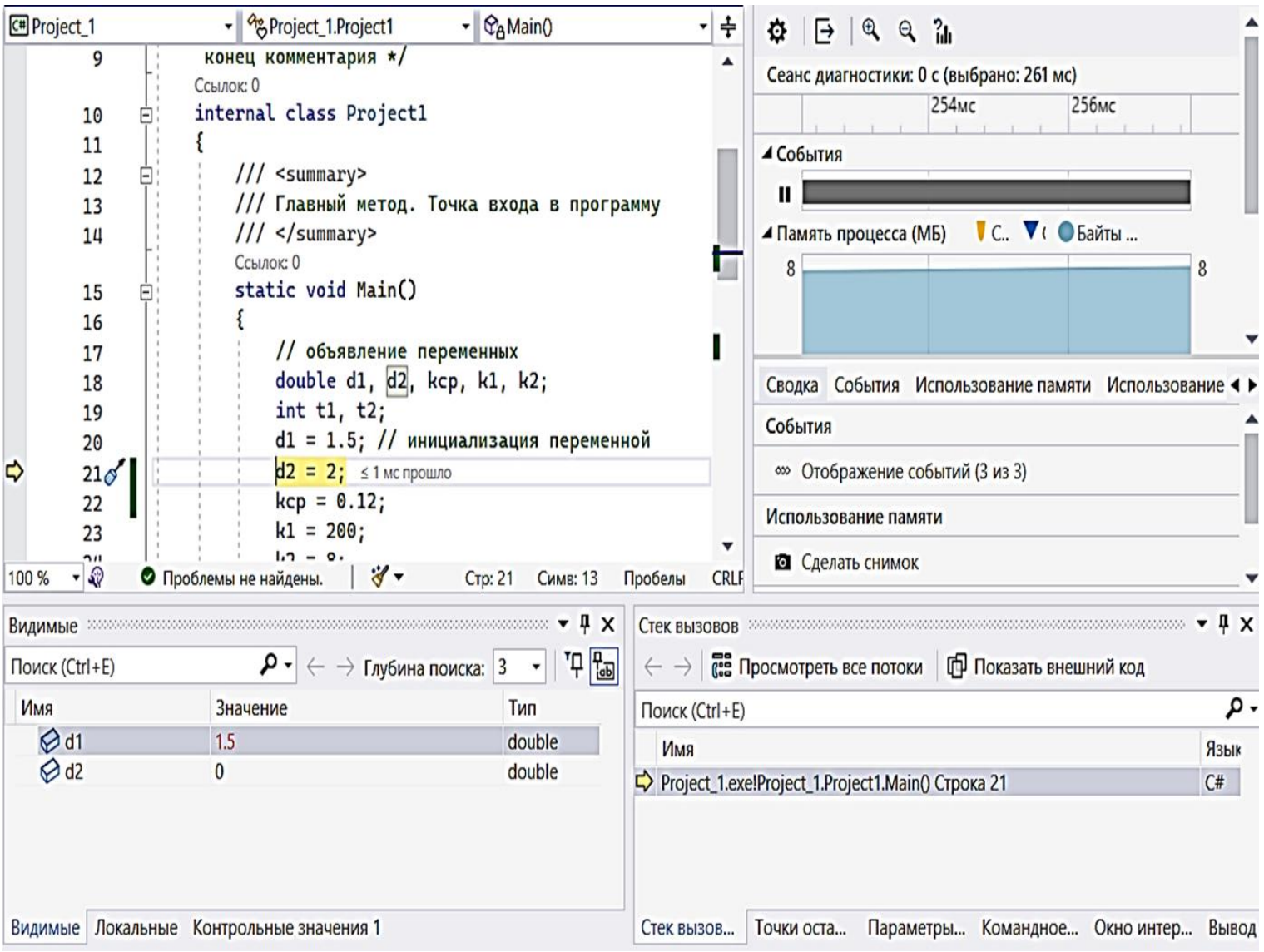


Рис. 13. Окно проекта в режиме пошаговой отладки

На этапе тестирования необходимо проверить работоспособность программы для различных входных данных, которые позволяют проверить все вычисления и условные переходы. Тестирование программы можно определить как процесс испытания программного продукта на конечном наборе тестов.

В табл. 4 представлены исходные данные для двух тестов.

Таблица 4

Входные данные для тестирования

Наименование величины	Тест 1	Тест 2
Внутренний диаметр, d1	0,5	1,5
Внешний диметр, d2	2,0	4,0
Средний коэффициент теплопроводности, kcp	0,10	0,15
Температура внутри шара, t1	110	150
Температура наружного воздуха, t2	10	50
Коэффициент теплоотдачи, k1	100	300
Коэффициент теплоотдачи, k2	4	12

Для проверки работоспособности программы по *Тесту 1* и *Тесту 2* добавим в *Проект 1* ввод данных с консоли – метод *Console.ReadLine()*, который возвращает строку символов в формате *string*.

Для получения с консоли числовых данных методы из класса *Convert* пространства имен *System* переводят данные типа *string* к соответствующему числовому типу данных. Для этого в пространстве определены статические методы. Например, *ToInt32()* преобразует строковый тип данных в числовой целочисленный тип данных, а *ToDouble()* – в вещественный тип данных. Также можно использовать *Parse()*, преобразующий строку в объект текущего типа.

Внесем изменения в текст *Project1.cs* в соответствии с кодом, приведенным на рис. 14.

Запускаем программу в режиме ▷ *Запуск без отладки*.

6. Проведение расчетов и анализ полученных результатов

Последний этап решения прикладной задачи заключается в проведении расчетов и анализе полученных результатов, а при необходимости – корректировке математической модели.

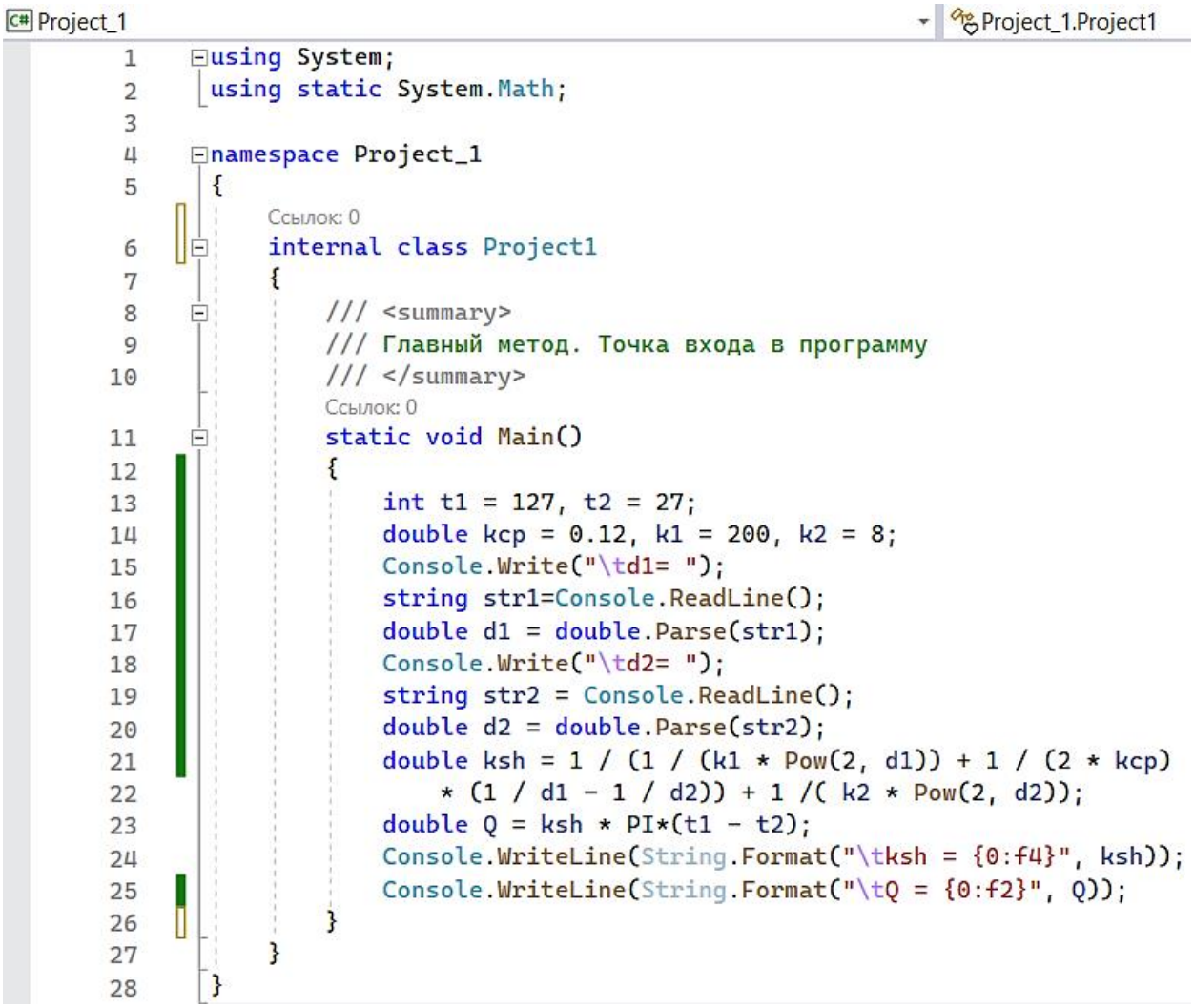


Рис. 14. Окно проекта в режиме тестирования

Контрольные вопросы

1. Какой класс предназначен для записи математических операций? Через какое пространство имен он подключается?
2. Какой метод используется для операции возведение в степень?
3. Что представляет собой алгоритм?
4. Какой алгоритм называется линейным?
5. Нарисуйте блок-схему линейного алгоритма в общем виде.
6. Что означает текстуальное описание в программе?
7. Что представляет собой комментарий в Visual Studio?
8. Назовите три способа оформления комментария.
9. Что представляет собой XML-комментарий?
10. Какой тег является основным для объяснения кода?
11. Что представляют собой переменная, ее объявление и инициализация?

12. Какие типы данных используются для объявления числовых переменных?

13. Какой метод применяется для задания формата вывода дробных значений с определенной точностью?

14. Для чего предназначено окно *Обозреватель решений*?

15. Что представляет собой процесс отладки программы?

16. Что представляет собой отладчик Visual Studio?

17. Назовите два режима для отладки программ.

18. Как запустить пошаговое выполнение кода?

19. Дайте характеристику этапа тестирования.

20. Какой метод позволяет вводить данные с консоли?

21. Какой метод позволяет конвертировать данные в числовые значения?

22. Какой метод преобразует строку в объект текущего типа?

ПРОЕКТ 2. АЛГОРИТМИЧЕСКАЯ СТРУКТУРА «ВЕТВЛЕНИЕ»

Цель: разработать алгоритм решения прикладной задачи с применением базовой структуры ветвления (разветвляющийся алгоритм) и решить ее в соответствии с табл. 1.

1. Постановка задачи[5]

Проверить технологическую эффективность работы горизонтальных песколовок по показателям эффективности (нормативным значениям) их работы. Указать характер и причины нарушений, разработать план мероприятий по их устранению.

Песколовки предназначены для выделения из сточных вод тяжелых минеральных примесей – главным образом песка. Для повышения эффективности процесса задерживания частиц песка размером свыше 0,25 мм аппаратное оснащение песколовок рассчитано на поддержание оптимальных технологических параметров со следующими контролируемыми показателями:

– скорость движения сточных вод в круговом потоке: при минимальном притоке – не менее 0,15 м/с, при максимальном притоке – не более 0,3 м/с;

– содержание песка в сыром осадке первичных отстойников: не более 8 %.

– плотность осадка: не менее 1,6 г/см^3 и не более 1,8 г/см^3;

– зольность осадка: не менее 70 и не более 95 %.

При эксплуатации песколовок ежемесячно осуществляется технологический контроль их работы. Нарушение нормальной работы песколовок связано с повышенным выносом песка или с задержанием органических соединений, способных к загниванию. Одним из определяющих моментов при эксплуатации песколовок является регулирование скоростей в зависимости от расхода поступающих сточных вод, которые для горизонтальных песколовок должны быть выдержаны в пределах 0,15–0,3 м/с.

Необходимо составить таблицу исходных данных (табл. 5) с описанием имен переменных и типов данных.

[5] Оценка эффективности работы водоочистного оборудования: учебное пособие / Ф. Ю. Ахмадуллина, А. С. Волков, Р. К. Закиров, Е. С. Балымова. Казань: Изд-во КНИТУ, 2022. 124 с.

Таблица 5

Описание входных переменных

Контролируемый показатель	Значение	Единицы измерения	Переменная в программе	Тип данных
Нормативная скорость воды	0,2	м/с	v	double
Содержание песка в сыром осадке отстойников	4,0	%	osv_p	double
Плотность осадка	1,7	г/см^3	p	double
Зольность осадка	75	%	A	double

Характер и причины нарушений работы горизонтальных песколовок, а также план мероприятий по их устранению представить в виде таблицы (табл. 6).

Таблица 6

Описание результатов эксперимента

Условие принятия решения	Сверхнормативная скорость подачи воды (более 0,3 м/с)	Низкая скорость протока (менее 0,15 м/с)
Характер нарушений	Вынос больших количеств песка в последующие звенья сооружений, что может быть связано с гидравлической перегрузкой Несоблюдение оптимальной периодичности отгрузки осадка	Содержание большого количества органических примесей в осадке из песколовок, способных к загниванию
Мероприятия по устранению нарушений	Подключение резервной песколовки при постоянных гидравлических перегрузках Ремонт распределительных устройств Увеличение частоты отгрузки осадка после сильных дождей	Отключение одной/нескольких песколовок или отделений песколовок Реконструкция или замена на аэрируемые песколовки

2. Математическая формализация задачи

Эффективность работы песколовок состоит в увеличении коэффициента удерживания минеральных частиц, расширении водоакцепторной пропускной способности (т. е. производительности), а также обеспечении стабильной надежности работы указанных сооружений.

Применение песколовок позволяет удалять из сточных вод взвешенные вещества минерального происхождения (преимущественно песка) крупностью свыше 0,25 мм. Принцип действия этих устройств основан на осаждении тяжелых примесей под действием силы тяжести.

Скорость движения сточных вод в песколовках ($0{,}5 < v < 0{,}3$ м/с) способствует тому, чтобы в процессе работы сооружения оседали преимущественно вещества минерального происхождения. Снижение скорости приводит к оседанию веществ преимущественно органического происхождения, что, в свою очередь, вызывает трудности при их обработке и из-за снижения зольности осадка, а увеличение – к выносу песка с песколовки.

Нормативные показатели работы горизонтальных песколовок:

– содержание песка в сыром осадке отстойников – менее 8 %;

– плотность осадка – $1{,}6 < p < 1{,}8$ г/см^3;

– зольность осадка – $70 < A < 95$ %.

3. Построение алгоритма

1. *Представление логических выражений сравнения нормативных и контролируемых показателей в виде описания разветвляющегося вида (табл. 7).*

Таблица 7

Показатели для горизонтальных песколовок

Показатель	Нормативное значение	Контролируемое значение
Нормативная скорость воды, м/с	$0{,}5 < v < 0{,}3$	0,2
Содержание песка в сыром осадке отстойников, %	Менее 8	4
Плотность осадка, г/см^3	$1{,}6 < p < 1{,}8$	1,7
Зольность осадка, %	$70 < A < 95$	75

Рассмотрим построение логических выражений для основного показателя, которым является нормативная скорость воды (см. табл. 6).

Если контролируемое значение входит в интервал нормативных значений, следовательно, горизонтальные песколовки функционируют в штатном режиме. В этом случае программа выдает соответствующее сообщение о режиме работы сооружений: <В НОРМЕ>.

Если контролируемое значение меньше нормативного значения, то горизонтальные песколовки работают с нарушением технологического режима, и программа выдает следующие сообщения:

– нарушение работы: <НИЖЕ НОРМЫ>;

– условие принятия решения: <Низкая скорость протока>;

– характер нарушений: <Содержание большого количества примесей>;

– мероприятия по устранению отмеченных недостатков: <Отключение одной или нескольких песколовок>.

Если контролируемое значение больше нормативного значения, то горизонтальные песколовки работают с нарушением технологического режима. В этом случае программа выдает следующие сообщения:

– нарушение работы: <ВЫШЕ НОРМЫ>;

– условие принятия решения: <Сверхнормативная скорость подачи воды>;

– характер нарушений: <Вынос больших количеств песка>;

– мероприятия по устранению: <Подключение резервной песколовки>.

2. *Построение схемы разветвляющегося алгоритма*. Логическую структуру алгоритма любой сложности можно представить в виде комбинации трех базовых алгоритмических структур: следования, ветвления, повторения.

Ветвление – это алгоритмическая конструкция, в которой в зависимости от результата проверки условия (*да/true* или *нет/false*) предусмотрен выбор одной из двух последовательностей действий. Алгоритм, основанный на конструкции ветвления, называется разветвляющимся. Действия алгоритма изображаются стандартными геометрическими фигурами в соответствии с ГОСТ 19.701-90 «Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».

В разветвляющемся алгоритме обязательным блоком является блок условия – решение (проверка условия, реализующая условный переход).

Блок-схемы разветвляющегося алгоритма для полной и неполной формы ветвления представлены на рис. 15.

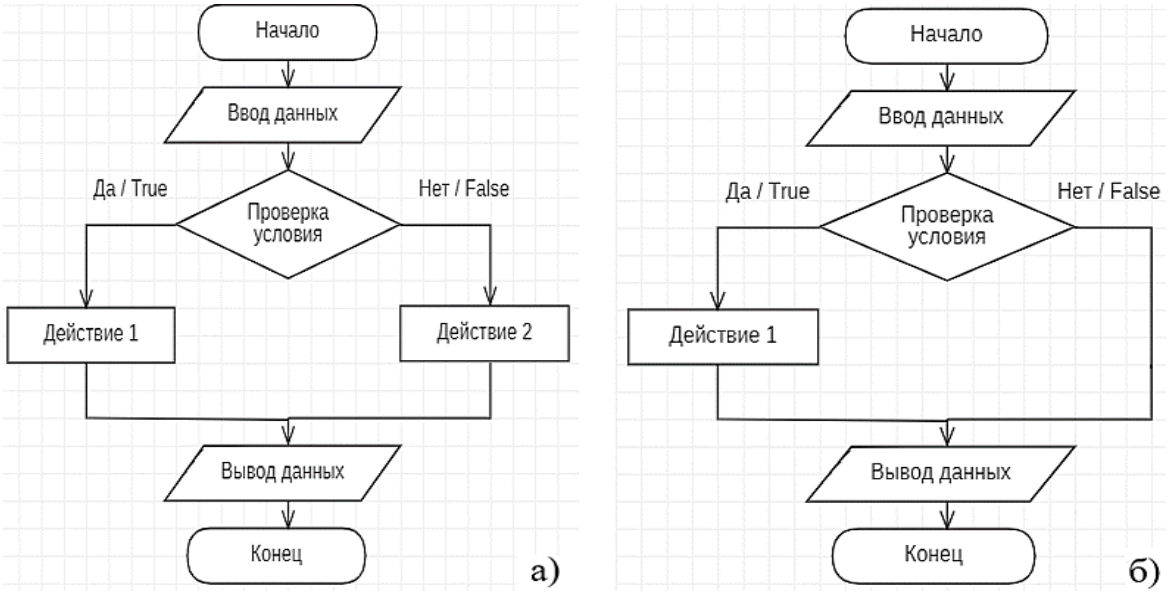


Рис. 15. Блок-схема разветвляющегося алгоритма: а – в полной форме; б – в неполной форме

Конструкция ветвления описывается условным оператором, который начинается с ключевого слова *if* (табл. 8).

Таблица 8

Описание условного оператора

<table>
<tr><td colspan="2">Шаблон оператора if</td></tr>
<tr><td>для полной формы</td><td>для вложенных условий</td></tr>
<tr><td>

```
if(условие)
{
 // Команды, если условие Да/True
}
else
{
 // Команды, если условие Нет/False
}
```

</td><td rowspan="3">

```
if
{
 // Команды, если условие Да/True
}
else
{
 // Команды, если условие Нет/False
}
if (условие)
{
 // Команды, если условие Да/True
}
else
{
 // Команды, если условие Нет/False
    if (условие)
    {
     // Команды, если условие Да/True
    }
    else
     // Команды, если условие Нет/False
}
```

</td></tr>
<tr><td>для неполной формы</td></tr>
<tr><td>

```
if(условие)
{
 // Команды, если условие Да/True
}
```

</td></tr>
</table>

Для записи нескольких условий сразу применяются союзы: *ИЛИ* (объединение), а также *И* (пересечение). Для них определены следующие логические операторы:

&& – означает *И*; выражение считается истинным, если истинно каждое из составных выражений («и то, и другое»);

|| – означает *ИЛИ*; выражение считается истинным, если истинно хотя бы одно из составных выражений («или то, или другое»).

Например, двойное неравенство

$$-3 \le x \le 4$$

записывается в виде двух логических выражений: $x \ge -3$ и $x \le 4$, соединенных специальным оператором *И*:

$$(x \ge -3 \;\&\&\; x \le 4).$$

3. *Текстуальное описание в программе.* Текстуальное описание в разветвляющейся программе также содержит все три способа описания комментария: однострочный, многострочный и XML-комментарий (рис. 16).

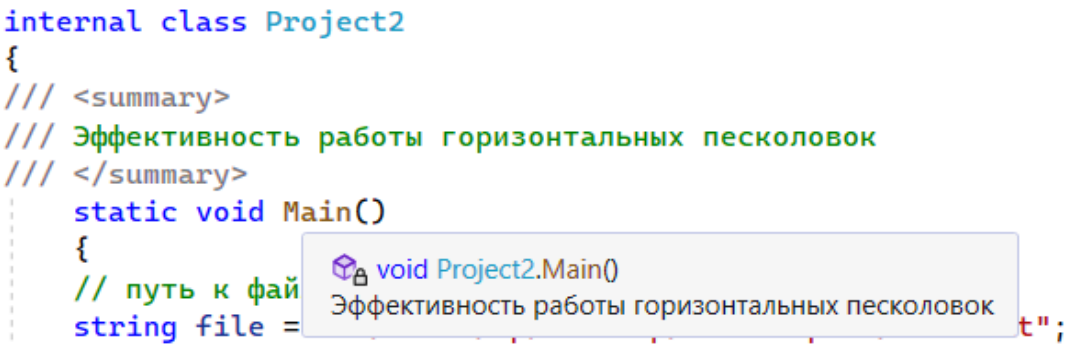
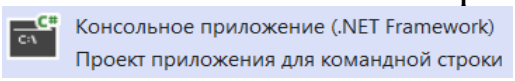


Рис. 16. Комментарии в Проекте 2

4. Составление программы на языке программирования

Составляем программу для решения прикладной задачи с применением базовой структуры ветвления (разветвляющаяся программа).

Создаем новый проект *Project2*. Запускаем Visual Studio2022. Открываем среду разработки при помощи клавиши *Esc* или команды *Продолжить без кода*. Выбираем *Файл* → *Создать* → *Проект*. Открываем



. Активируем кнопку *Далее* и в окне *Настроить новый проект* задаем имя проекта *Project2*.

Определяем место расположения проекта в папке *Application_C#*. Устанавливаем флажок *Поместить решение и проект в одном каталоге*, инициируем кнопку *Создать*. Изменяем имя текстового файла *Program.cs* на *Project2.cs*. Открыв контекстное меню и выбрав *Переименовать*, вводим новое имя файла программы *Project2*. Подтверждаем переименование в появившемся диалоговом окне.

Далее создаем программу для проверки эффективности работы горизонтальных песколовок по нормативному показателю скорости подачи воды:

```
using System;
using System.Diagnostics.Eventing.Reader;

namespace Project2
{
   internal class Project2
   {
   /// <summary>
   /// Эффективность работы горизонтальных песколовок
   /// </summary>
      static void Main()
      {
         Console.WriteLine("\t\t\tПесколовки: Горизонтальные\n");
```

```
            Console.WriteLine("\t\tНормативные значения работы песколовок:");
            Console.WriteLine("\tНормативная скорость воды, м/с: 0,15 < v < 0,3");
            Console.WriteLine("\tСодержание песка в сыром осадке отстойников, %: < 8");
            Console.WriteLine("\tПлотность осадка, г/см3: 1,6 - 1,8");
            Console.WriteLine("\tЗольность осадка, %: 70 - 95\n");
            // ввод контролируемых показателей
            double v = 0.2, osv_p = 4, p = 1.7, A = 75;
            if (v > 0.3)
               {
               Console.WriteLine("Нарушение работы: <ВЫШЕ НОРМЫ>");
               Console.WriteLine("Условие принятия решения: <Сверхнормативная скорость подачи воды>");
               Console.WriteLine("Характер нарушений: <Вынос больших количеств песка>");
               Console.WriteLine("Мероприятия по устранению: <Подключение резервной песколовки>");
               }
            else
            {
               if (v < 0.15)
               {
                  Console.WriteLine("Нарушение работы: <НИЖЕ НОРМЫ>");
                  Console.WriteLine("Условие принятия решения: <Низкая скорость протока>");
                  Console.WriteLine("Характер нарушений: <Содержание большого количества примесей>");
                  Console.WriteLine("Мероприятия по устранению: <Отключение одной или нескольких песколовок>");
               }
               else
                  Console.WriteLine("Нарушение работы: <В НОРМЕ>");
            }
         }
      }
   }
```

Запускаем программу в режиме ▷ *Запуск без отладки*.

5. Отладка и тестирование программы

Проект скомпилировался и запустился. Скомпилированное приложение появится в папке проекта *C:\Application_C#\Project2\bin\Debug* с именем проекта Project2 и расширением *exe* (приложение).

Результат выполнения программы с форматированным выводом представлен на рис. 17.

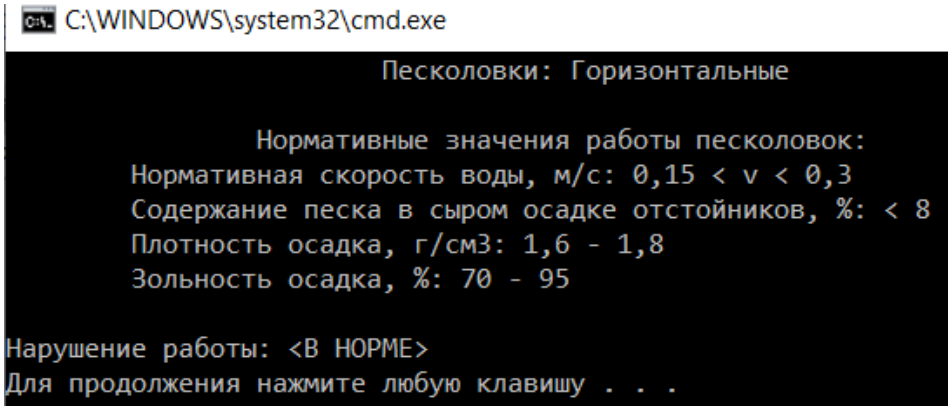


Рис. 17. Окно консольного приложения Проекта 2

Форматированный вывод строк в консольном приложении

```
Console.WriteLine("\t\t\tПесколовки: Горизонтальные\n");
```

осуществляют управляющие последовательности: \t – горизонтальная табуляция, т. е. длинный отступ (четыре пробела); \n – перевод на новую строку.

На этапе тестирования необходимо проверить работоспособность программы для различных наборов контролируемых параметров горизонтальной песколовки в соответствии с нормативными показателями технологического режима.

Тестирование программы можно определить как процесс испытания программного продукта на конечном наборе тестов. В табл. 9 представлены исходные данные для двух тестов.

Таблица 9

Входные данные для тестирования

Наименование величины	Тест 1	Тест 2
Нормативная скорость воды, м/с	0,1	0,6
Содержание песка в сыром осадке отстойников, %	4,0	7,0
Плотность осадка, г/см^3	1,7	1,7
Зольность осадка, %	65,0	80,0

Контрольные наборы тестов удобно хранить в файлах. Для тестирования создадим два файла данных с расширением **.dat*. В таких файлах хранится информация в двоичном (бинарном) или текстовом (символьном) виде.

Рассмотрим, как на языке программирования C# реализуется концепция файлового ввода и вывода данных, а именно считывание информации из файла и запись информации в файл. Определим такое понятие, как *поток* – это некоторая упорядоченная последовательность данных

(набор битов из нулей и единиц), которые обрабатываются блоком. Блок из восьми битов – *байт*. Соответственно, поток, который обрабатывается на уровне байтов, – байтовый. В символьных потоках данные обрабатываются блоками по два байта, и каждый такой блок интерпретируется как символ.

Для работы с файлами ввода/вывода используем класс *FileStream* из пространства имен *System.IO*. Конструктор *FileStream(string filename, FileMode mode)* в качестве аргументов принимает два параметра: путь к файлу и перечисление *FileMode*. Режим доступа определяет *FileMode.Create* – для создания нового файла, *FileMode.Open* – для открытия уже существующего файла.

Для работы с бинарными файлами используем бинарные потоки. Объекты таких потоков создаются на основе классов *BinaryWriter* (*bw* – поток для записи данных в бинарный файл) и *BinaryReader* (*br* – поток для считывания данных из бинарного файла):

```
20          Console.WriteLine("\tЗольность осадка, %: 70 - 95\n");
21          // ввод нормативных значений
22          double v, osv_p, p, A;
23          // данные для записи в файл
24          v = 0.1;
25          osv_p = 4.0;
26          p = 1.7;
27          A = 75.0;
28          // путь к файлу и его название
29          string file = "C:/Users/hp/Desktop/C# Examples/test1.dat";
30          Console.WriteLine("Запись данных в файл ...");
31          try
32          {
33          //бинарный поток для записи данных в файл
34          BinaryWriter bw = new BinaryWriter(new FileStream(file, FileMode.Create));
35          //запись данных в файл
36          bw.Write(v);
37          bw.Write(osv_p);
38          bw.Write(p);
39          bw.Write(A);
40          // поток закрывается
41          bw.Close();
42          }
43          // обработка исключений
44          catch(Exception e)
45          {
46          Console.WriteLine("Ошибка записи в файл!");
47          Console.WriteLine(e.Message);
48          return;
49          }
50          Console.WriteLine("Создан файл \"{0}\"", file);
```

Рис. 18. Фрагмент программы для записи данных в файл

Для записи значения в заданный файл, например, `string file = "C:/Users/hp/Desktop/C# Examples/test1.dat";` из объекта потока *bw* вызывается метод *Write()*, аргументом которому передается записываемое в файл значение, например, `bw.Write(v);`. При завершении записи данных в поток он закрывается командой *bw.Close()*.

Создадим файл *test1.dat* для контрольных показателей *Тесла 1*. Внесем изменения в текст *Project2.cs* в соответствии с кодом, приведенным на рис. 18.

Запускаем программу в режиме ▷ *Запуск без отладки*.

Результат выполнения программы с данными *Тесла 1* представлен на рис. 19.

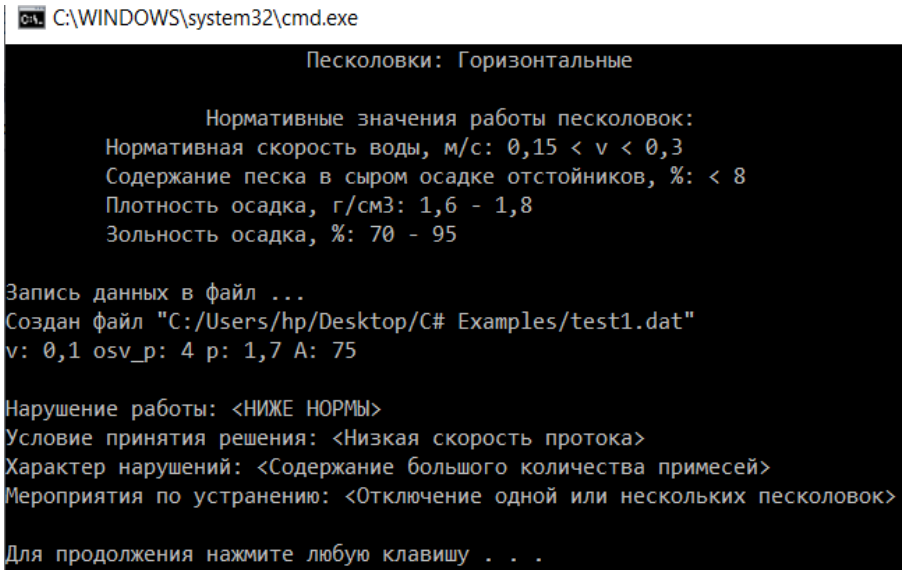


Рис. 19. Окно консольного приложения Тест 1

Аргументом конструктору класса *BinaryReader* передается анонимный объект байтового потока, который создается при помощи инструкции *new FileStream(file, FileMode.Open)* в режиме открытия уже существующего файла.

Бинарный поток для считывания данных из файла записывается следующей инструкцией: `BinaryReader br = new BinaryReader(new FileStream(file, FileMode.Open));`

Для считывания данных из заданного файла, например, `string file = "C:/Users/hp/Desktop/C# Examples/test2.dat";`, из объекта потока *br* вызывается метод *ReadDouble()*, который считывает действительное число, например, `v = br.ReadDouble();`.

Метод *ReadInt32()* считывает целое число, метод *ReadChar()* – символ, а *ReadString()* – текст. После завершения считывания данных поток закрывается командой *br.Close()*. Далее создаем файл *test2.dat* для контрольных показателей *Тесла 2*, считываем данные из файла

с отображением на консоли и вносим изменения в текст *Project2.cs* в соответствии с кодом, приведенным на рис. 20.

```
51          Console.WriteLine("Считывание данных из файла ...");
52          try
53          {
54          //бинарный поток для считывания данных из файла
55          BinaryReader br = new BinaryReader(new FileStream(file, FileMode.Open));
56          //BinaryReader br = new BinaryReader(File.Open(file, FileMode.Open));
57          //считывание данных из файла и отображение в консольном окне
58          v = br.ReadDouble();
59          osv_p = br.ReadDouble();
60          p = br.ReadDouble();
61          A = br.ReadDouble();
62          //Console.WriteLine(br.ReadDouble());
63          //Console.WriteLine(br.ReadDouble());
64          //Console.WriteLine(br.ReadDouble());
65          //Console.WriteLine(br.ReadDouble());
66          // поток закрывается
67          br.Close();
68          }
69          // обработка исключений
70          catch (Exception e)
71          {
72          Console.WriteLine("Ошибка чтения файла!");
73          Console.WriteLine(e.Message);
74          return;
75          }
76          Console.WriteLine($"v: {v} osv_p: {osv_p} p: {p} A: {A}\n");
```

Рис. 20. Фрагмент программы для чтения данных из файла

Запускаем программу в режиме ▷ *Запуск без отладки*.

Результат выполнения программы с данными *Теста 2* представлен на рис. 21.

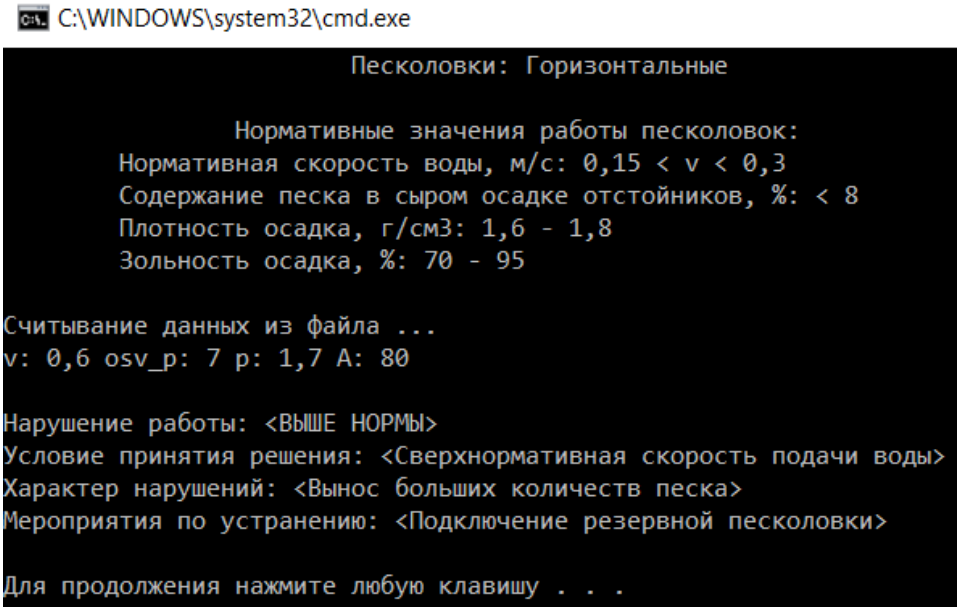


Рис. 21. Окно консольного приложения Тест 2

6. Проведение расчетов и анализ полученных результатов

Последний этап решения прикладной задачи – это проведение расчетов, анализ полученных результатов и корректировка математической модели (при необходимости).

Контрольные вопросы

1. Сформулируйте принципы записи логических выражений.
2. Охарактеризуйте такую структуру, как ветвление.
3. Какой алгоритм называется разветвляющимся?
4. Назовите основной блок разветвляющегося алгоритма.
5. Нарисуйте блок-схему разветвляющегося алгоритма в полной форме.
6. Нарисуйте блок-схему разветвляющегося алгоритма в неполной форме.
7. Опишите такую конструкцию, как ветвление.
8. Запишите шаблон оператора if в полной и не полной форме.
9. Запишите шаблон оператора if для вложенных условий.
10. Дайте определение логическим операторам *И* и *ИЛИ*.
11. Что представляют собой управляющие последовательности \t и \n?
12. Что представляет собой файл с расширением **.dat*?
13. Что представляет собой поток?
14. Какой класс используется для работы с файлами ввода/вывода?
15. Какие два параметра в качестве аргументов принимает конструктор *FileStream(string filename, Filemode mode)*?
16. Что определяет режим доступа *FileMode.Create*?
17. Что определяет режим доступа *FileMode.Open*?
18. На основе каких классов создаются объекты для бинарных потоков?
19. С помощью какой команды закрываются потоки при считывании и записи данных?
20. Какие методы позволяют считывать: целое число, символ и текст?

ПРОЕКТ 3. АЛГОРИТМИЧЕСКАЯ СТРУКТУРА «ПОВТОРЕНИЕ»

Цель: разработать алгоритм решения прикладной задачи с применением базовой структуры повторения (циклический алгоритм) и решить ее в соответствии с табл. 1.

1. Постановка задачи[6]

Вычислить численность населения Земного шара в заданный период. Для вычисления и прогноза численности населения земного шара используйте формулу, которую предложил С. П. Капица.

Полученные результаты по численности населения Земли вывести в табличной форме (табл. 10 и 11) за указанные промежутки по годам.

Таблица 10

Описание входных переменных

Наименование величины	Значение	Единицы измерения	Переменная		Тип данных
			в формуле	в программе	
Год, для которого вычисляется численность населения Земли	2001–2030	г	t	Year	data
Постоянная	172 ($172 \cdot 10^9$)	млрд чел.-лет	C	C	int
Средняя временная характеристика жизни человека	45	лет	τ	tau	int
Год от Рождества Христова	2000	г	T_1	T_1	data

Таблица 11

Описание выходных переменных

Наименование величины	Единицы измерения	Переменная		Тип данных
		в формуле	в программе	
Численность населения Земли	млрд чел.	N	N	int
Год, для которого вычисляется численность населения Земли	2001–2030	t	Year	data

[6] Капица С. П. Математическая модель роста населения мира // Математическое моделирование. 1992. Т. 4, № 6. С. 65–79.

2. Математическая формализация задачи

В 1996 г. советский и российский физик Сергей Петрович Капица представил оригинальную математическую модель для анализа и прогнозирования динамики мирового демографического процесса, основанную на идеях синергетики (изучения процессов самоорганизации в различных системах). Такое моделирование позволяет описать рост численности человечества на протяжении практически всего периода его существования. Отметим, что эта модель адекватна только в глобальном масштабе и не подходит для детального исследования локальных тенденций отдельных государств или регионов.

Население мира в момент времени *t* соотнесем с числом жителей земного шара как *N(t)* – это будет ведущей аддитивной переменной, подчиняющей все остальные. Такое выделение главной переменной характерно для системного подхода и получило свое обоснование в синергетике. В модели используется самое минимальное число параметров для описания всей истории человечества. Тем не менее она дает возможность не только описать развитие человечества за огромный период времени, но и сравнить результаты расчетов с данными демографических исследований[7].

Численность населения Земли определяем по формуле

$$N(t) = \frac{C}{\tau} \cdot arcctg\left(\frac{T_1 - t}{\tau}\right), \quad (4)$$

где t – год, для которого вычисляется численность населения; C – 172 млрд чел.-лет (постоянная, описывающая рост населения Земли в течение сотен и тысяч лет); T_1 – 2000 г. (год от Рождества Христова); τ – 45 лет (средняя продолжительность жизни человеческого поколения).

Функцию arcctg(x) выразим следующим образом:

$$arcctg(x) = \frac{\pi}{2} - arctg(x). \quad (5)$$

3. Построение алгоритма

1. *Представление расчетного соотношения (4) в виде инструкции линейного вида на языке программирования C#.* При записи тригонометрических функций используется класс *Math*, который подключается через пространство имен *using System*. В этом классе функция *arcctg()* отсутствует, поэтому выразим ее через *arctg()* по формуле (5). Для вычисления *arctg()* применяется метод *Math.Atan()*, а метод *Math.PI* возвращает число π.

[7] Котов, О. М. Язык C#: краткое описание и введение в технологии программирования: учеб. пособие. Екатеринбург, 2014. 208 с.

Инструкция кода для записи уравнения (4) в программе выглядит следующим образом:

```
N = (C / tau) * (Math.PI / 2 – Math.Atan((T1 – Year) / tau)
```

Необходимо рассчитать численность населения Земли по формуле (4) за период 2001–2030 гг., а также выполнить расчет прогнозных значений на 2030–2040 и 2050–2060 гг. Расчетные значения по этим трем периодам сохранить в различных текстовых файлах.

2. *Построение схемы циклического алгоритма.* Логическую структуру алгоритма любой сложности можно представить в виде комбинации трех базовых алгоритмических структур: следования, ветвления, повторения.

Повторение – это алгоритмическая конструкция, представляющая собой последовательность действий, выполняемых многократно. Алгоритм, основанный на данной конструкции, называется циклическим алгоритмом (циклом). Циклы являются управляющими конструкциями, в которых действие выполняется множество раз в зависимости от определенных условий. Действия алгоритма изображаются стандартными геометрическими фигурами в соответствии с ГОСТ 19.701-90 «Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».

Различают три типа циклических алгоритмов:

– цикл с предусловием (цикл с заданным условием продолжения работы или цикл-пока); тело цикла выполняется до тех пор, пока выполняется условие;

– цикл с постусловием (цикл с заданным условием окончания работы или цикл-до); тело цикла выполняется до тех пор, пока условие не выполняется;

– арифметический цикл (цикл с заданным числом повторений или цикл с параметром); тело цикла выполняется для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.

В циклическом алгоритме блок, который будет выполняться многократно, носит название *тело цикла*. Один цикл повторений называется *итерацией*. При конструировании циклов нужно соблюдать обязательное условие результативности (т. е. окончания) алгоритма за конечное число шагов. Это означает, что условие продолжения цикла должно зависеть от некоторой переменной, значение которой меняется непосредственно во время каждой отдельной итерации. Такая переменная называется управляющей переменной (параметром) цикла. Операторы с условием управляют процессом повторения вычислений, таких как инициа-

лизация переменной цикла, проверка условия продолжения цикла и изменение его переменной. Причем управляющая переменная изменяется таким образом, чтобы условие в конечном итоге перестало выполняться.

Блок-схемы циклического алгоритма для цикла с предусловием и с постусловием представлены на рис. 22, а для арифметического цикла и цикла в цикле (вложенного цикла) – на рис. 23.

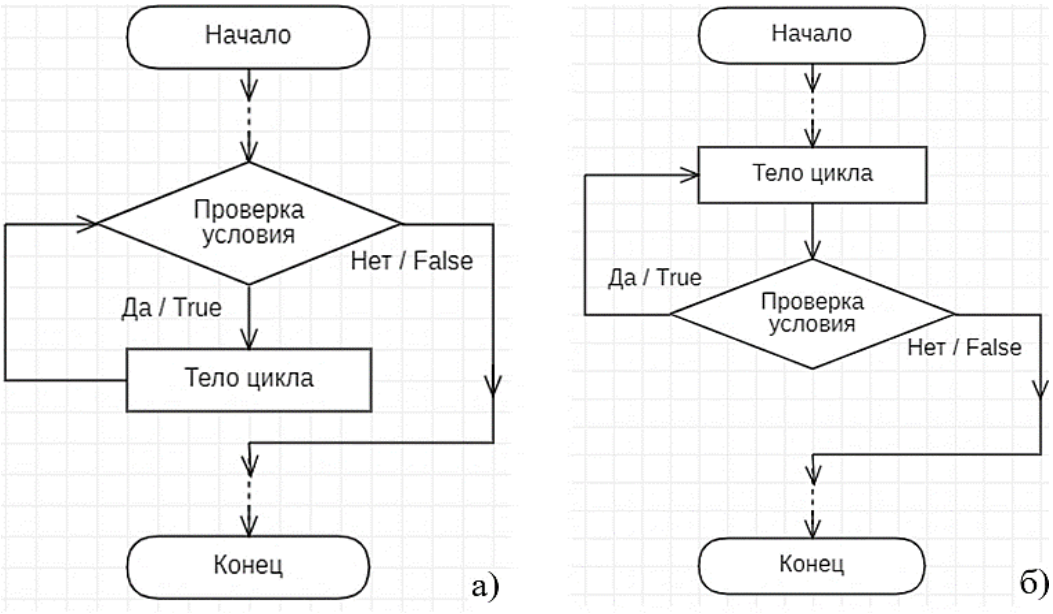
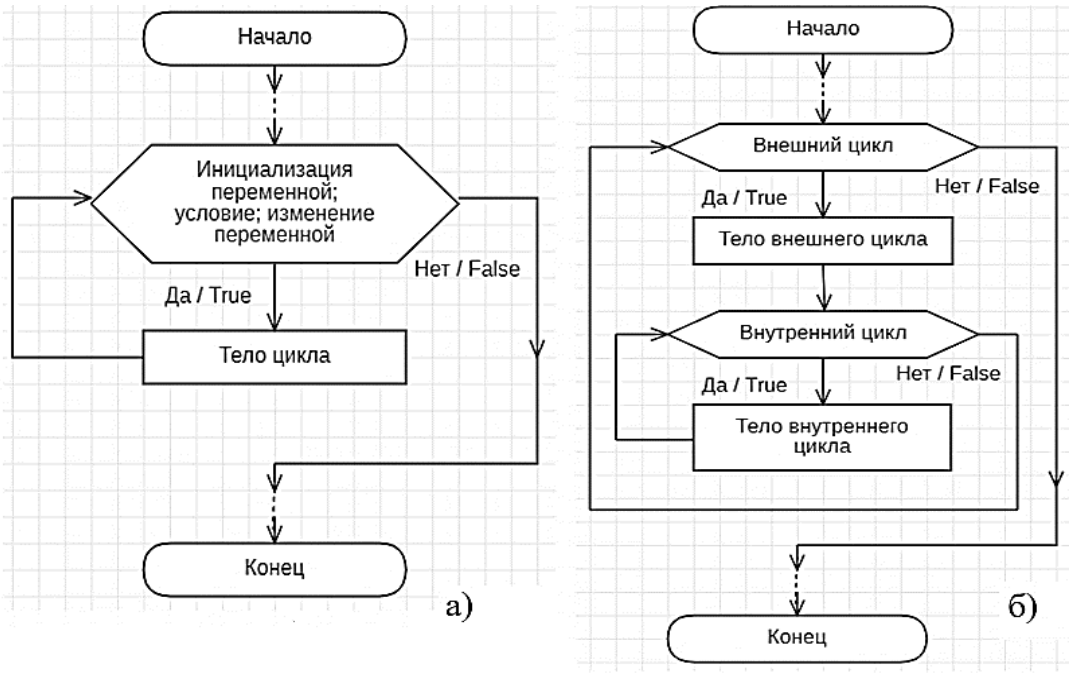


Рис. 22. Блок-схема циклического алгоритма:
а – цикл с предусловием; б – цикл с постусловием



бл7

Рис. 23. Блок-схема циклического алгоритма:
а – арифметический цикл; б – вложенный цикл

Описание операторов цикла представлено в табл. 12.

Таблица 12

Описание циклических операторов

<table>
<tr><td colspan="2">Шаблон оператора while</td></tr>
<tr><td>цикл с предусловием</td><td>цикл с постусловием</td></tr>
<tr><td>

```
while (условие)
{
 // Команды в теле цикла
}
```

</td><td>

```
do
{
 // Команды в теле цикла
}
while (условие);
```

</td></tr>
<tr><td colspan="2">Шаблон оператора for</td></tr>
<tr><td colspan="2">

```
for (инициализация переменной; условие; изменение переменной)
{
// Команды тела цикла
}
```

</td></tr>
<tr><td colspan="2">Пример оператора for для вложенного цикла</td></tr>
<tr><td colspan="2">

```
for (int i = 0; i < 5; i++)
 {
 // Команды внешнего цикла
    for (int j = 0; j < 5; j++)
    {
    // Команды внутреннего цикла
    }
 }
```

</td></tr>
</table>

Цикл, размещенный в теле другого цикла, называются вложенным, а цикл, вложенный в тело другого цикла, – внутренним. Внешним называется цикл, в теле которого существует вложенный цикл. Число итераций внутреннего цикла вычисляется путем умножения количества его собственных итераций на произведение чисел итераций всех внешних циклов.

3. *Текстуальное описание в программе*. Текстуальное описание в циклической программе содержит все три способа описания комментария: однострочный, многострочный; XML-комментарий (рис. 24).

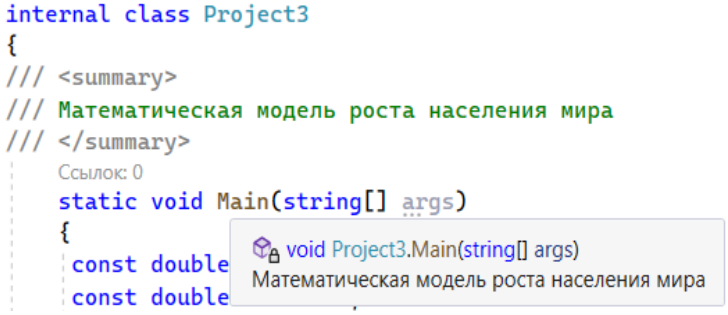


Рис. 24. XML-комментарий в Проекте 3

4. Составление программы на языке программирования

Составим программу для решения прикладной задачи с применением базовой структуры повторения (циклическая программа).

Создадим новый проект *Project3*. Запускаем Visual Studio2022 и открываем среду разработки при помощи клавиши *Esc* или команды *Продолжить без кода*. Выбираем *Файл* → *Создать* → *Проект*. Открываем  Консольное приложение (.NET Framework) Проект приложения для командной строки. После использования кнопки *Далее* в окне *Настроить новый проект* задаем имя проекта *Project3*.

Устанавливаем место расположения проекта в папке с именем *Application_C#*. Установив флажок *Поместить решение и проект в одном каталоге*, нажимаем кнопку *Создать*.

Изменяем имя текстового файла *Program.cs* на *Project3.cs*. С этой целью открываем контекстное меню, выбираем *Переименовать* и вводим новое имя файла программы *Project3*, подтверждая переименование в появившемся диалоговом окне.

На рис. 25 представлено окно *Visual Studio* с программой проекта *Project3*. Текст программы вводим в соответствии с рисунком. Для записи арифметических выражений используем класс *Math*, который подключаем через пространство имен *using System*.

```
Project3                                                    Project_3
1    using System;
2
3    namespace Project_3
4    {
         Ссылок: 0
5        internal class Project3
6        {
7        /// <summary>
8        /// Математическая модель роста населения мира
9        /// </summary>
             Ссылок: 0
10           static void Main(string[] args)
11           {
12            const double C = 172;
13            const double tau = 45;
14            const double T1 = 2000;
15            double Year = 2001;
16            double N;
17               // Цикл с предусловием
18               while (Year <= 2030)
19               {
20                   double t = (T1 - Year) / tau;
21                   N = C / tau * (Math.PI / 2 - Math.Atan(t));
22                   Console.WriteLine($"Year = {Year} \t" +
23                   $" N = {N.ToString("#.####")}");
24                   Year++;
25               }
```

Рис. 25. Окно Проекта 3

При программировании решения задачи по расчету численности населения Земли (по годам) используем все три вида циклических операторов: *while*, *do-while* и *for*.

Цикл с предусловием начинаем с ключевого слова *while*, после которого в круглых скобках приводим некоторое выражение со значением логического типа (*True/False*). Затем в фигурных скобках указываем блок из команд, формирующих тело цикла. Приступая к выполнению цикла *while*, сначала проверяем условие (*Year <= 2030*). Если оно истинно (*True*), то выполняем команды в теле цикла и снова проверяем условие. Если оно истинно, то опять выполняем команды в теле цикла, после чего повторно проверяем условие и т. д. Цикл длится до тех пор, пока при проверке условия оно не окажется ложным (*False*) – в этом случае работа его оператора завершается, а управление передается инструкции, следующей после оператора цикла. При логическом построении программы необходимо учитывать следующее: чтобы в операторе цикла начали выполняться команды, условие в самом начале должно соответствовать *True*, а для их завершения – стать равным *False*. Иначе получится бесконечный цикл.

Запускаем программу в режиме ▷ *Запуск без отладки*.

Результат выполнения программы представлен на рис. 26.

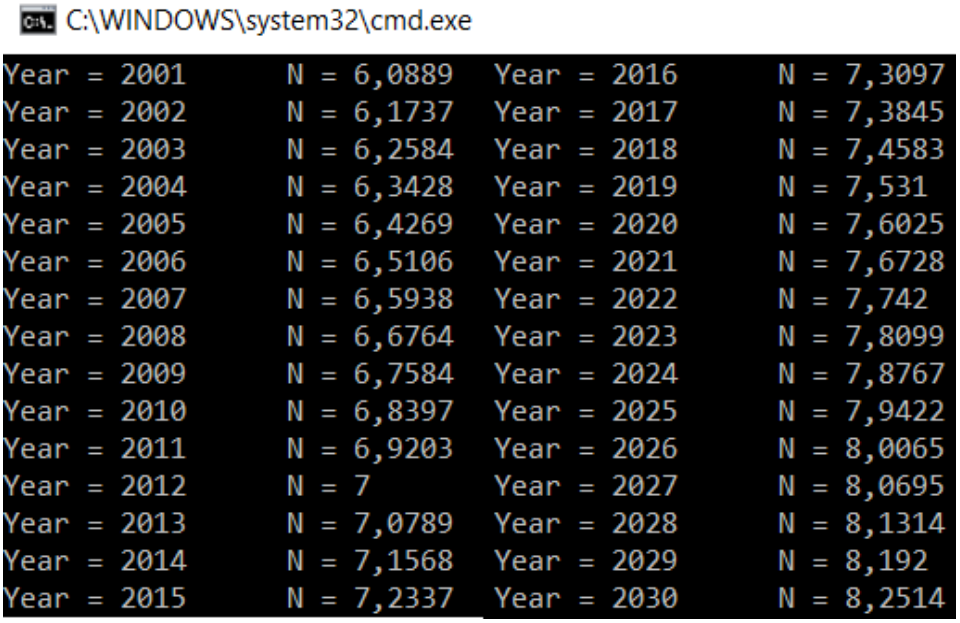


Рис. 26. Результат Проекта 3 с предусловием

Продолжим код в методе *Main* проекта *Project3* со строки 26 после фигурной скобки } для цикла с постусловием. Введем следующие инструкции:

```
// Цикл с постусловием
do
{
  double t = (T1 - Year) / tau;
  N = C / tau * (Math.PI / 2 - Math.Atan(t));
  Console.WriteLine($"Year = {Year} \t" +
      $" N = {N.ToString("#.####")}");
  ++Year;
}
while (Year <= 2030);
```

Цикл с постусловием начинаем с ключевого слова *do*, далее в фигурных скобках приводим инструкции тела цикла. После этого следует ключевое слово *while* с условием в круглых скобках (*Year <= 2030*). Последовательность выполнения цикла *do-while* следующая: сначала выполняем команды в теле цикла, затем проверяем условие, указанное после ключевого слова *while*. Если условие истинно (*True*), то выполняем команды в теле цикла и проверяем условие. Цикл завершится, если условие станет ложным (*False*).

Главное отличие операторов *while* и *do-while* в том, что выполнение оператора *while* начинается с проверки условия – если оно истинно (*True*), то выполняется тело цикла а в цикле *do-while* сначала выполняется тело цикла, и только после этого проверяется условие. Таким образом, в операторе *do-while* тело цикла по крайней мере выполняется один раз точно, а в операторе *while* оно может быть не выполнено ни разу в случае ложного условия.

Запускаем программу в режиме ▷ *Запуск без отладки*.

Результат выполнения программы представлен на рис. 27.

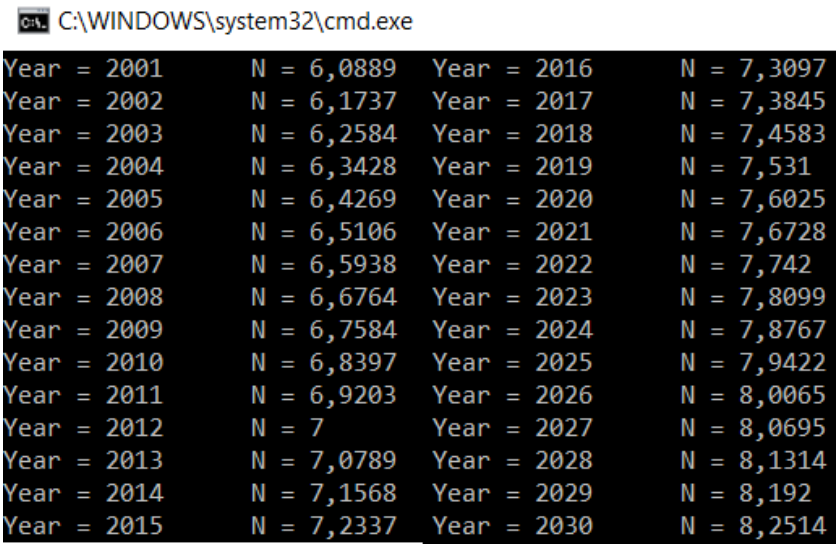


Рис. 27. Результат Проекта 3 с постусловием

Проанализировав полученный результат, обратим внимание на запись оператора *Year++* (в программе с предусловием) и *++ Year* (в программе с постусловием).

В языке C# присутствуют такие арифметические операторы, как инкремент и декремент. Они относятся к унарным операциям увеличения (++) и уменьшения (--) одного числа. Оператор инкремента выполняет сложение операнда с числом 1, т. е. инструкция x = x + 1 аналогична инструкции ++x. Оператор декремента выполняет вычитание операнда с числом 1, т. е. инструкция x = x - 1 аналогична инструкции --x. Операторы инкремента и декремента имеют два варианта использования: префиксная форма ++x (--x) и постфиксная форма x++ (x--). При использовании постфиксного варианта вначале происходит возврат значения переменной и только потом – выполнение операции, а для префиксного – наоборот.

Продолжим код в методе *Main* проекта *Project3* со строки 36 после фигурной скобки } для записи арифметического цикла. С этой целью введем следующие инструкции:

```
// Арифметический цикл
for (Year = 2001; Year <= 2030; Year++)
{
  double t = (T1 - Year) / tau;
  N = C / tau * (Math.PI / 2 - Math.Atan(t));
  Console.WriteLine($"Year = {Year} \t" +
     $" N = {N.ToString("#.####")}");
}
```

Арифметический цикл начинается с ключевого слова *for*, далее в круглых указываем трех операторов: инициализация переменной, условие и изменение переменной. Оператор инициализации переменной означает объявление и задание начального значения переменной (*Year = 2001*) и выполняется только один раз. Затем определяем условие (*Year <= 2030*), которое является логическим выражением и возвращает либо истину (*True*), либо ложь (*False*). Если условие истинно (*True*), то выполняется тело цикла. Затем происходит изменение инициализированной переменной на некоторую величину (*Year++*) и выполняется оценка условия. Процесс продолжается до тех пор, пока условие не станет ложным (*False*), после чего цикл *for* завершается.

Запускаем программу в режиме ▷ *Запуск без отладки*.

Проанализируем результат выполнения программы. В циклических программах иногда возникает ситуация, когда требуется выйти из

цикла, не дожидаясь его завершения. В таких случаях применяем оператор *break*, который завершает выполнение цикла и передает управление на инструкцию, следующую за циклом. Для пропуска текущей итерации применяем оператор *continue*, который завершает начатую итерацию и передает управление в начало цикла.

5. Отладка и тестирование программы

Проект скомпилировался и запустился. Скомпилированное приложение появится в папке проекта *C:\Application_C#\Project3\bin\Debug* с именем проекта Project3 и расширением *exe* (приложение). Выполняем пошаговую отладку и сравниваем значение *Year* в префиксной (*Year++*) и постфиксной форме (*++ Year*) для кода с циклическими операторами всех трех видов.

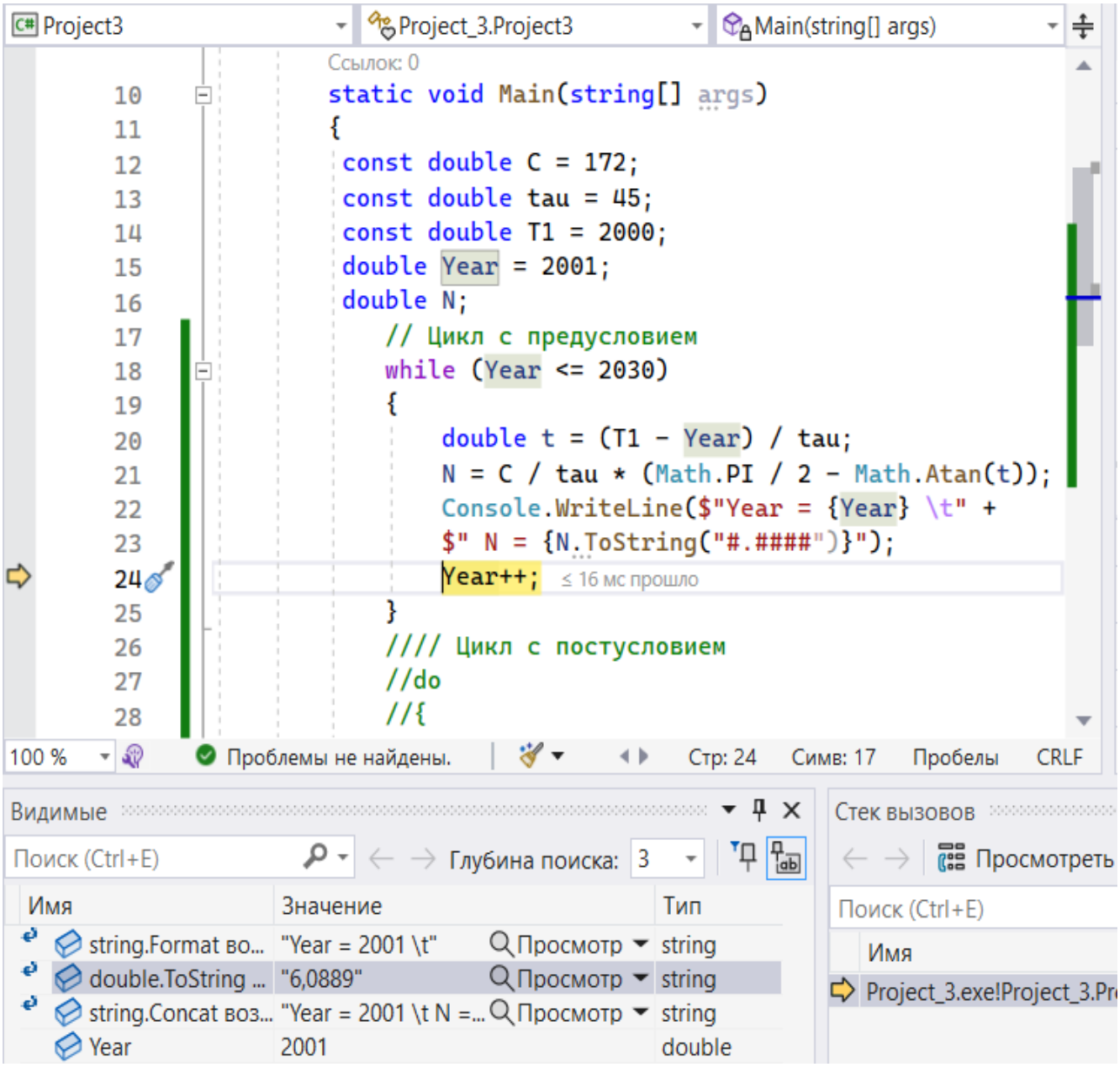


Рис. 28. Окно в режиме пошагового выполнения

Запускаем пошаговое выполнение кода с помощью функциональной клавиши *F11*. В строке появится специальный маркер в виде желтой стрелочки, а сама инструкция выделится желтым цветом. Стрелка-маркер указывает на следующую строку, которая будет выполняться. Последовательно перейдем по строкам программы с помощью клавиши *F11*. Одновременно рекомендуется наблюдать за окном *Видимые текущие значения вычисляемых переменных*, предназначенным для демонстрации текущих значений переменных, участвующих в расчетах. При запуске программы в режиме отладки значения переменных будут видны прямо в исходном коде (рис. 28).

На этапе тестирования необходимо выполнить расчет численности населения мира по формуле (4) для 2030–2040 гг. и 2050–2060 гг. Расчетные значения по трем периодам следует сохранить в различных текстовых файлах (табл. 13).

Таблица 13

Данные для тестирования

Наименование файла	Расчетный период
Test_41	2030–2040 гг.
Test_42	2050–2060 гг.

Изменим код арифметического цикла, добавив в него инструкции для выбора и печати расчетных периодов согласно рис. 29.

```
// Арифметический цикл
for (Year = 2001; Year <= 2060; Year++)
{
    double t = (T1 - Year) / tau;
    N = C / tau * (Math.PI / 2 - Math.Atan(t));
    //Console.WriteLine($"Year = {Year} \t" +
    //    $" N = {N.ToString("#.####")}");
    if (Year == 2030) Console.WriteLine("\n\tРасчет численности населения мира" +
        " \n\tза период 2030-2040гг.\n");
    if (Year == 2050) Console.WriteLine("\n\tРасчет численности населения мира" +
        " \n\tза период 2050-2060гг.\n");
    if (Year>=2030 &&  Year<=2040)
    {
        Console.WriteLine($"\tYear = {Year} \tN = {N.ToString("#.####")}");
    }
    else if (Year >= 2050 && Year <= 2060)
    {
        Console.WriteLine($"\tYear = {Year} \tN = {N.ToString("#.####")}");
    }
    else Console.WriteLine($"\tYear = {Year} \tN = {N.ToString("#.####")}");
```

Рис. 29. Окно программы тестирования выбора периодов

Запускаем программу в режиме ▷ *Запуск без отладки*.
Результат выполнения программы представлен на рис. 30.

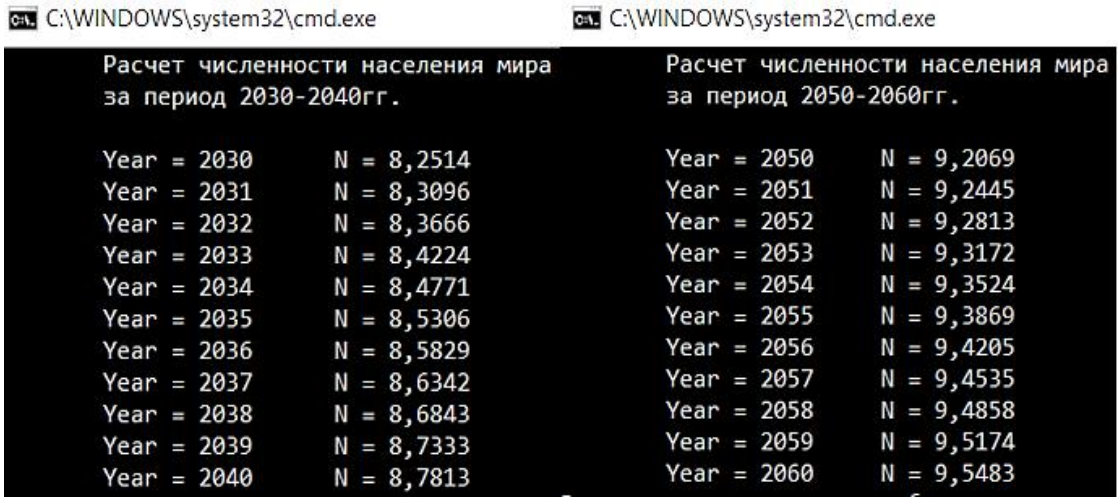


Рис. 30. Окно консольного приложения периодов

Ввиду специфики функционирования циклических алгоритмов, предполагающих обработку значительных объемов информационных массивов, данные необходимо вводить из множества разнородных файловых источников и впоследствии выводить результаты обработки также в разнообразные целевые файловые структуры. Все эти операции обеспечивает пространство имен *System.IO* (System Input/Output) – библиотека классов файлового ввода-вывода.

Выделим несколько классов, обеспечивающих функциональность пространства *System.IO*:

– *BinaryReader, BinaryWriter* – ввод/вывод базовых типов данных в двоичном виде;

– *File, FileInfo* – обеспечение работы с файлами;

– *FileStream* – обеспечение работы с файлом как с потоком байтов, поддержание произвольного доступа к файлу (поиска);

– *StreamWriter, StreamReader* – чтение-запись текстовой информации файла как потока байтов; отсутствие функции поддержки произвольного доступа (поиска).

Рассмотрим более подробно некоторые из приведенных классов, которые используются для ввода-вывода данных в *Проекте 3*.

Класс *FileInfo* позволяет получать характеристики файлов, расположенных на диске, а также создавать, копировать, перемещать и удалять файлы с помощью следующих инструментов:

– *AppendText()* – создает объект и добавляет текст в файл;

– *CopyTo()* – копирует существующий файл в новый файл;

– *CreateText()* – создает объект, который записывает новый текстовый файл;

– *Delete()* – удаляет файл.

Классы *StreamWriter* и *StreamReader* функционируют в кодировке Unicode с символьными данными. В рамках этих классов все данные представляются в виде потоков данных, т. е. последовательности байтов информации.

Рассмотрим запись результирующих данных в текстовый файл. Из типа *File* создаем объект с именем будущего текстового файла `StreamWriter w = File.AppendText("C:/Test41.txt");`. После этого с помощью ссылки на созданный объект вызываем метод *AppendText()*, который создает файл, если он не существует (поэтому метод *Create()* можно не использовать), и возвращает в переменную *w* ссылку на объект типа *Stream Writer*. В классе *StreamWriter* имеется метод *WriteLine()* для записи в файл текстовых строк: `w.WriteLine($"\tYear = {Year} \tN = {N.ToString("#.####")}");`. Завершив запись данных, для сохранения строк в файле его надо закрыть командой `w.Close();`.

Создадим текстовые файлы *Test41.txt*, *Test42.txt* и *Test43.txt* для записи расчетных значений численности населения мира в зависимости от заданного периода (см. табл. 13). Внесем изменения в текст программы *Project3.cs* в соответствии с приведенным кодом:

```
// Запись данных в файл
        Console.WriteLine("Запись данных в файл ...");
        // Арифметический цикл
        for (Year = 2001; Year <= 2060; Year++)
        {
           double t = (T1 - Year) / tau;
           N = C / tau * (Math.PI / 2 - Math.Atan(t));
           if (Year == 2030) Console.WriteLine("\nTest41 - Расчет численности населения мира" +
              " \n\tза период 2030-2040гг.\n");
           if (Year == 2050) Console.WriteLine("\nTest42 - Расчет численности населения мира" +
              " \n\tза период 2050-2060гг.\n");
           if (Year >= 2030 && Year <= 2040)
           {
              StreamWriter w = File.AppendText("C:/Users/hp/Desktop/C#Examples/Test41.txt");
              w.WriteLine($"\tYear = {Year} \tN = {N.ToString("#.####")}");
              w.Close();
           }
```

```
            else if (Year >= 2050 && Year <= 2060)
            {
               StreamWriter w = File.AppendText("C:/Users/hp/Desktop/C#Examples/
Test42.txt");
               w.WriteLine($"\tYear = {Year} \tN = {N.ToString("#.####")}");
               w.Close();
            }
            else
            {
               StreamWriter w = File.AppendText("C:/Users/hp/Desktop/C#Examples/
Test43.txt");
               w.WriteLine($"\tYear = {Year} \tN = {N.ToString("#.####")}");
               w.Close();
            }
         }
```

Запускаем программу в режиме ▷ *Запуск без отладки*.

На рис. 31 представлен результат работы консольного приложения, а также содержание текстового файла в блокноте *Notepad++*.

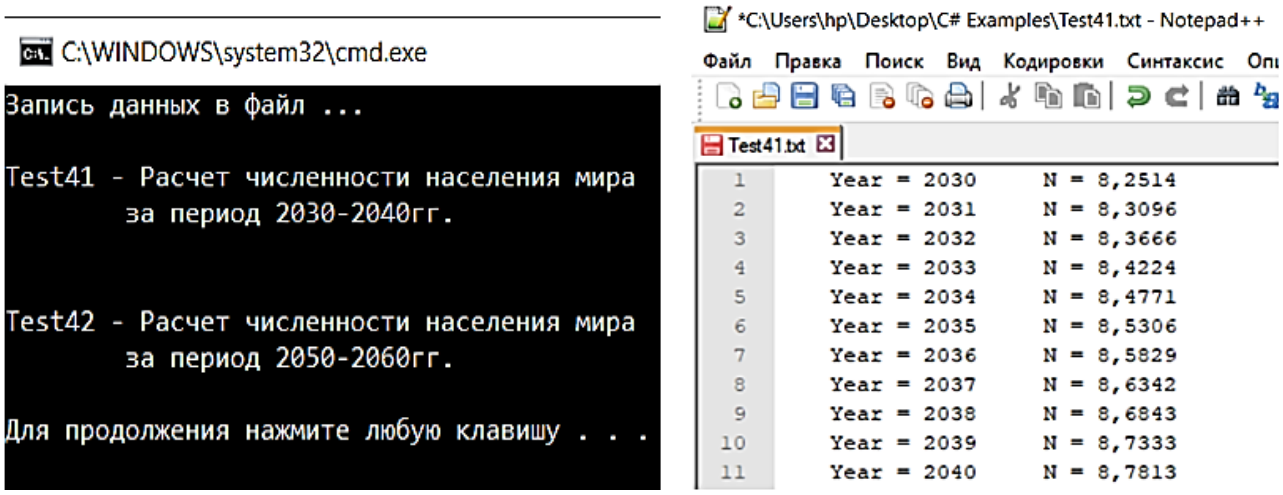


Рис. 31. Результат выполнения записи данных в файл

Для чтения данных из текстового файла его необходимо сначала открыть, чтобы получить к нему доступ. Эта операция выполняется с помощью метода *OpenText()*, формирующего в переменной *s* ссылку типа `StreamReader sr = File.OpenText("C:/Test41.txt");` *StreamReader*.

В классе *StreamReader* имеется метод *ReadLine()*, позволяющий считывать текстовые строки `s = sr.ReadLine();`. В ходе разового применения данного метода читается одна строка, а невидимый указатель файла устанавливается на начало следующей строки. Применение данного метода в цикле позволяет считывать все строки из файла.

Для закрытия файла класс *StreamReader* располагает собственным методом *Close()* `sr.Close();`. Если файл остался незакрытым, данные не потеряются, потому что все файлы закрываются операционной системой при ее выгрузке.

Для записи и чтения всех строк сразу в классе *File* предназначены следующие методы:

– *WriteAllLines()* – создает новый файл, записывает в него коллекцию строк данных и закрывает файл;

– *ReadAllLines()* – открывает файл, возвращает символьные данные в виде массива строк и закрывает файл.

Внесем изменения в текст программы *Project3.cs* в соответствии с приведенным кодом:

```
// Чтение данных из файла
        string s;
        Console.WriteLine("Содержимое введенного файла");
        StreamReader sr = File.OpenText("C:/Users/hp/Desktop/C# Examples/Test41.
txt");
        for (int i = 0; i <= 10; i++)
        {
           s = sr.ReadLine();
           Console.WriteLine(s);
        }
        sr.Close();
        Console.ReadKey(true);
```

Запускаем программу в режиме ▷ *Запуск без отладки*.

На рис. 32 представлены содержание текстового файла в блокноте *Notepad++*, а также результат работы консольного приложения.

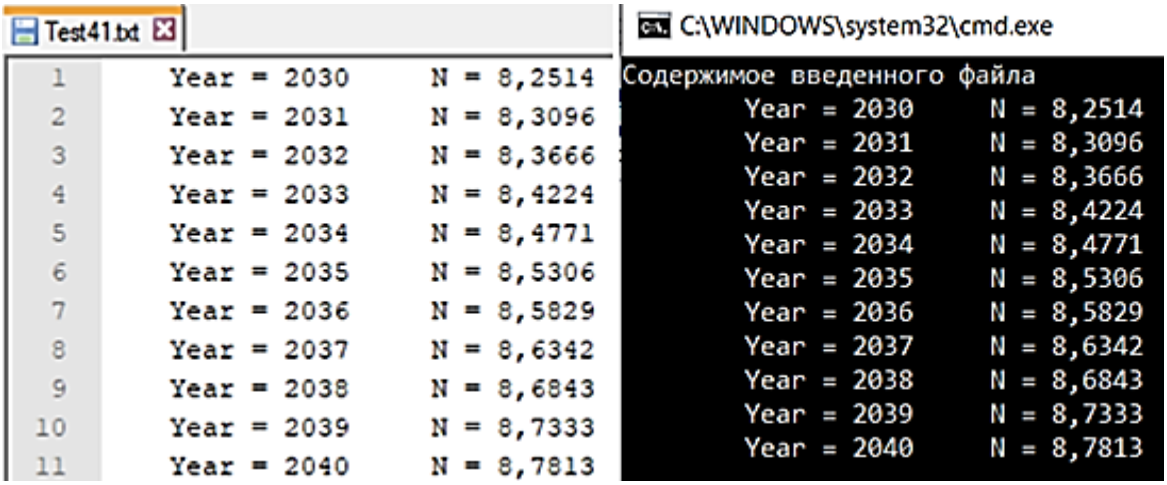


Рис. 32. Результат прочтения данных из файла

Необходимо проанализировать работу нашей программы в режиме записи/чтения данных и проверить правильность созданных файлов на диске в соответствии с табл. 13.

6. Проведение расчетов и анализ полученных результатов

Последний этап решения прикладной задачи заключается в проведении расчетов, анализе полученных результатов и корректировке математической модели (при необходимости).

Контрольные вопросы

1. Охарактеризуйте структуру «повторение».
2. Какой алгоритм называется циклическим?
3. Перечислите три типа циклических алгоритмов.
4. Как в циклическом алгоритме называется блок, который повторяется многократно?
5. Как называется один цикл повторений?
6. Какая переменная называется параметром цикла?
7. Нарисуйте блок-схему разветвляющегося алгоритма в полной форме.
8. Нарисуйте блок-схему цикла с предусловием.
9. Нарисуйте блок-схему цикла с постусловием.
10. Нарисуйте блок-схему арифметического цикла.
11. Дайте характеристику вложенному циклу, нарисуйте его блок-схему.
12. Запишите шаблон оператора *while* и объясните его работу.
13. Запишите шаблон оператора *do-while* и объясните его работу.
14. Запишите шаблон оператора *for* и объясните его работу.
15. Приведите пример записи оператора *for* для вложенного цикла.
16. Охарактеризуйте работу операторов инкремента и декремента.
17. Какой оператор позволяет выйти из цикла, не дожидаясь его завершения?
18. Как запустить пошаговое выполнение кода в режиме отладки?
19. Что представляет собой пространство имен *System.IO*?
20. Назовите инструменты класса *FileInfo*.
21. С помощью какой команды закрываются потоки при считывании и записи данных?
22. Назовите методы класса *File* для записи и чтения сразу всех строк.

ПРОЕКТ 4. ГРАФИЧЕСКИЙ ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

Основные принципы построения графического интерфейса

Понятие о графическом пользовательском интерфейсе в соответствии с требованиями и рекомендациями ГОСТ Р ИСО 9241-161-2016 «Эргономика взаимодействия человек-система. Часть 161. Элементы графического пользовательского интерфейса» уже были нами рассмотрены ранее.

В табл. 14 перечислены графические элементы интерфейса пользователя в зависимости от состояния системы и действий пользователя.

Таблица 14

Состояния графических элементов интерфейса пользователя

Состояние элемента	Описание
Видимый или невидимый	Элемент GUI является видимым/невидимым для пользователя
С фокусом или без фокуса	Нажатие кнопки клавиатуры передается/не передается к элементу GUI
Выбранный или невыбранный	Элемент GUI был выбран или нет
Помеченный или непомеченный	Значение элемента (группы элементов) было установлено или не было установлено
Заполненный или пустой	Элемент содержит некоторое содержимое или нет
Нажатый или не нажатый	Элемент (например, кнопка) отображает состояние вкл./выкл.
Свернутый или развернутый	Отображаемый элемент занимает максимальное или минимальное пространство

Необходимо учитывать, что каждое состояние визуально должно четко отличаться от другого состояния. Перечисленные в одном и том же пункте состояния являются взаимоисключающими.

На рис. 35 представлены основные принципы построения графического пользовательского интерфейса.

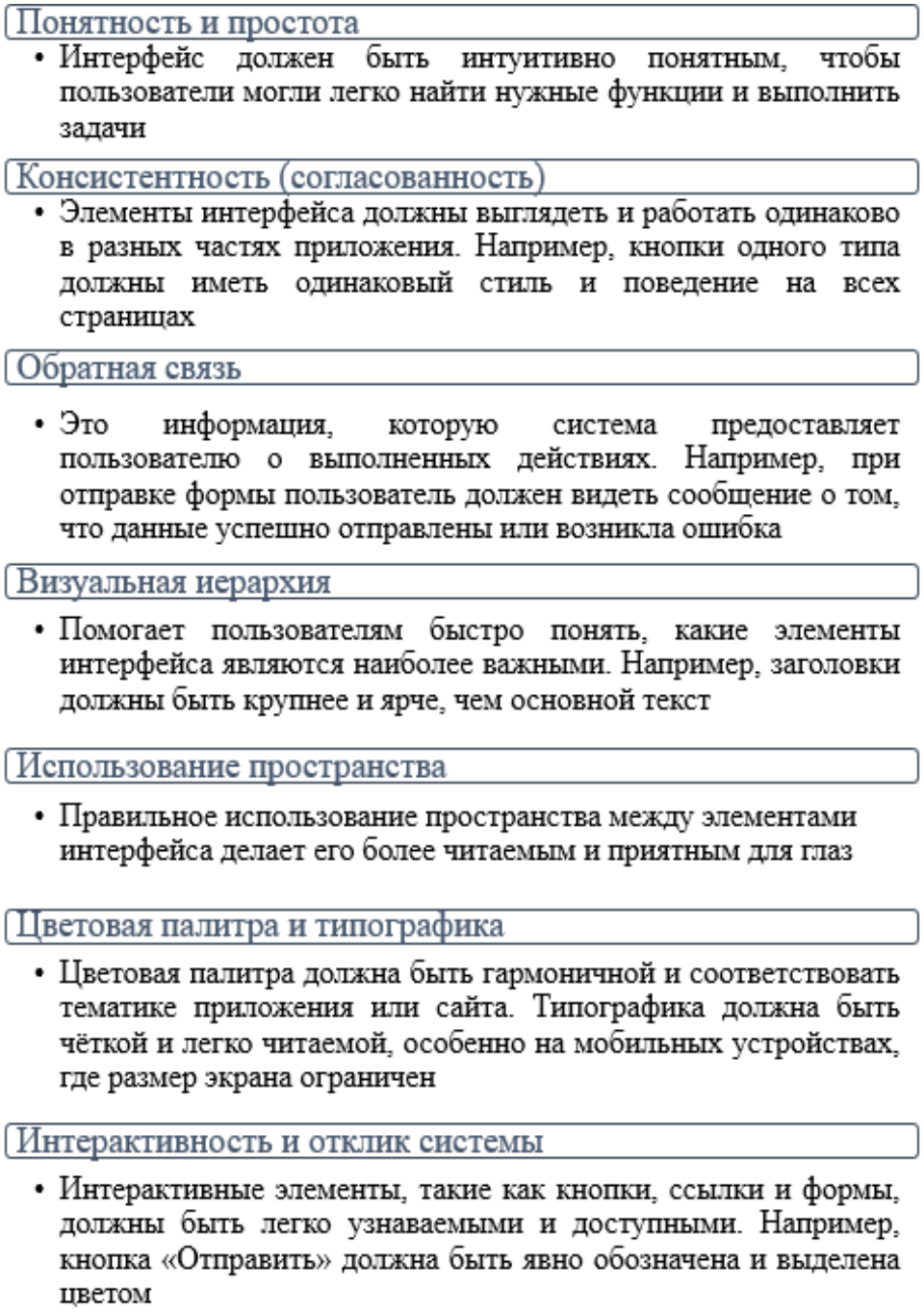


Рис. 35. Общие принципы построения GUI

Рассмотрим базовые принципы, которые лежат в основе технологии создания приложений с графическим интерфейсом пользователя.

Общая постановка задачи состоит в том, чтобы разработать графический интерфейс и написать программный код, при выполнении которого отображается окно с элементами управления. Графические элементы интерфейса подразделяются на статические (например, текст, графическое изображение или рамка) и функциональные (например, кнопки, раскрывающиеся списки, флажки). Статические элементы до-

бавляются в процессе разработки интерфейса и не связаны с действиями (например, активировать кнопку или выбрать пункт в списке). Функциональные элементы связаны с некоторыми манипуляциями, которые позволяют программе реагировать на действия, связанные с этими элементами управления.

Таким образом, можно разбить поставленную задачу на две составляющие:

1) создание графического интерфейса пользователя (формы);

2) реакция программы на действия пользователя с элементами управления (обработка событий).

Создадим проект графического интерфейса пользователя.

Запускаем Visual Studio Visual Studio 2022. Открываем среду разработки при помощи клавиши *Esc* или команды *Продолжить без кода*. Выбираем *Файл* → *Создать* → *Проект*. В окне *Создание проекта* (рис. 36) в поле поиска вводим *winforms* («Идет поиск, ждите»), после чего выбираем *Приложение Windows Forms* (Майкрософт) и нажимаем кнопку *Далее*.

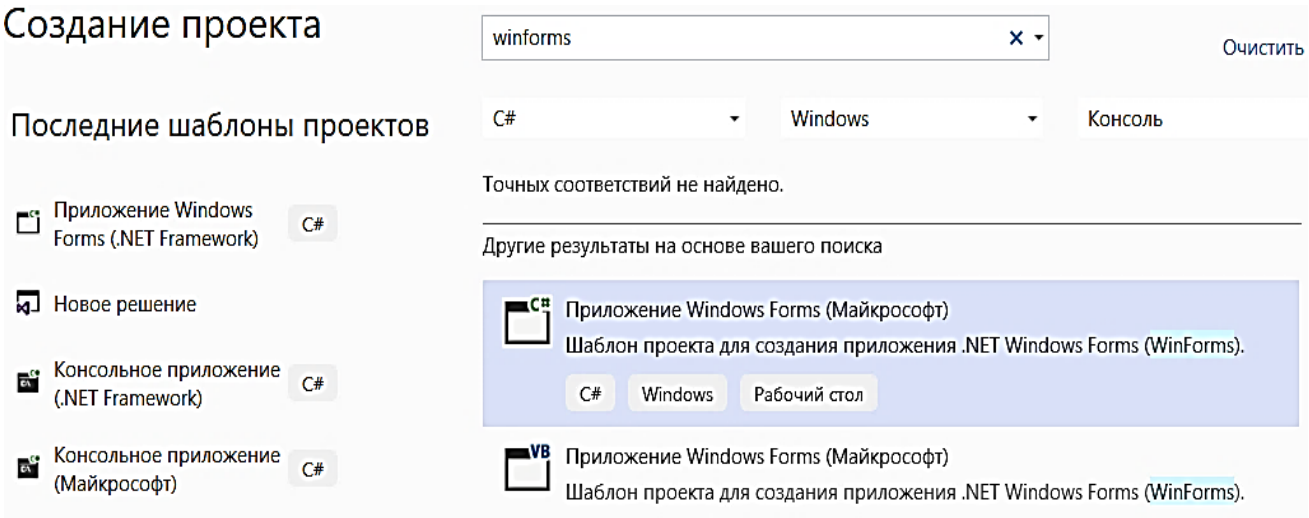


Рис. 36. Общие принципы построения GUI

Переходим в окно *Настроить новый проект* (рис. 37). Задаем имя проекта – *WinFormsApp1*, устанавливаем его место расположения в папке *Application C#*. Настраиваем окно для работы с формой (рис. 38). Если окна не присутствуют на экране, открываем:

– окно *Обозреватель решений* – *Вид* → Обозреватель решений;

– окно *Панель элементов* – *Вид* → Панель элементов;

– окно *Свойства* – *Вид* → Окно свойств (свойства отображаются для выделенного объекта).

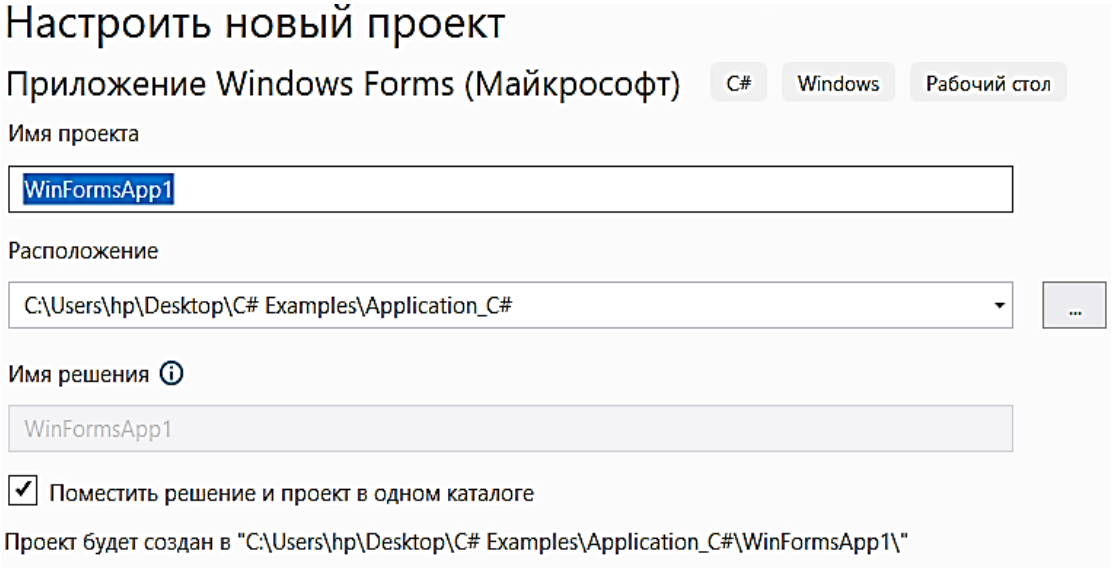


Рис. 37. Окно настройки Windows Forms

На рис. 38 выделенным объектом является форма *Form1*, а в окне *Свойства* отображаются все, связанные с ней, свойства.

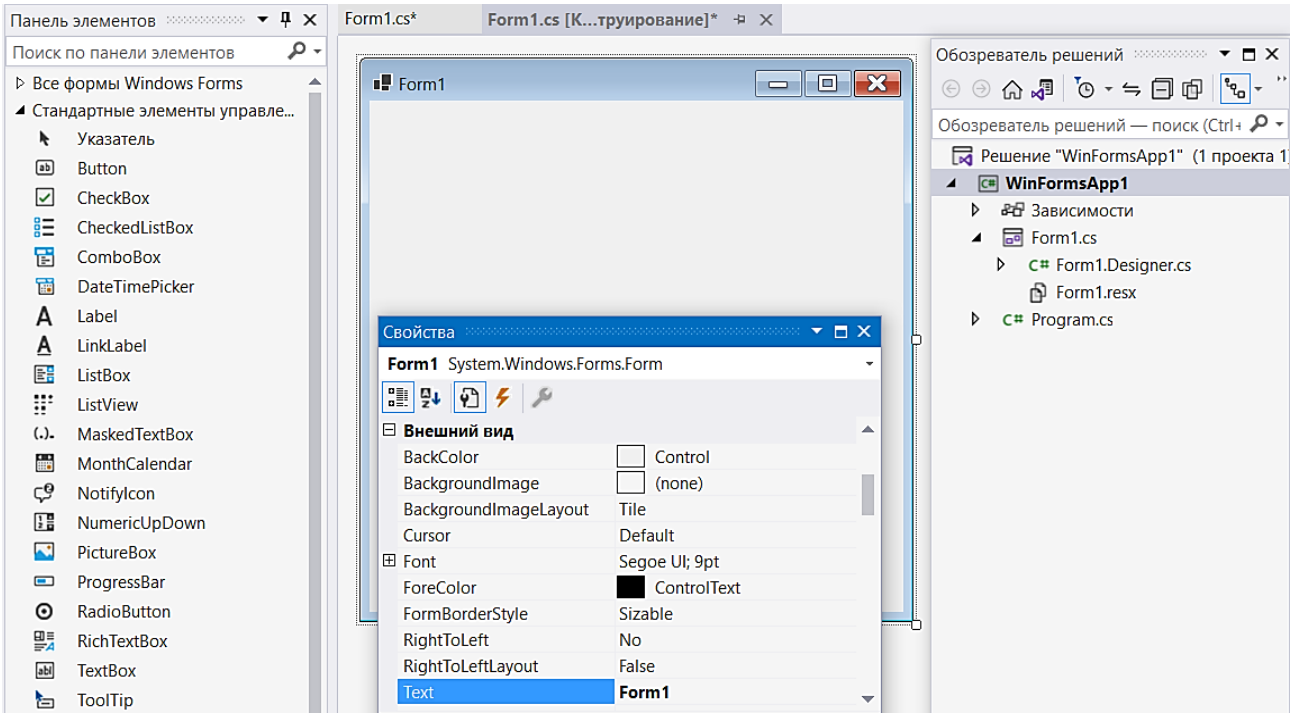


Рис. 38. Рабочее окно разработки Windows Forms

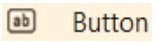
Окно *Обозреватель решений* содержит:

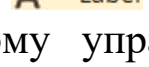
– файл формы *Form1.cs* – открыт в окне по умолчанию;

– файл *Form1.Desinger.cs* – содержит определение компонентов формы, добавленных на форму в графическом дизайнере;

– файл *Program.cs* – определяет точку входа в приложение.

Каждый элемент управления графического интерфейса реализуется с помощью объекта. Каждому типу элементов соответствует свой класс, например: для формы 

необходимо создать объект класса *Form*, для кнопки 

– объект класса *Button*, для метки, отображающей текст 

– объект класса *Label*.

Каждому управляющему элементу интерфейса соответствует определенное количество событий. Событие представляет собой объект, получающий уведомление о произошедшем активном воздействии и способный инициировать соответствующую реакцию системы посредством выполнения заданной последовательности операций. Разработчик может определить эти действия, добавив к событию обработчик. Например, у объектов класса *Button* есть событие *Click*. Инициация событий возможна посредством программного обеспечения либо вследствие активации элементов управления, таких как кнопка в графическом пользовательском интерфейсе (GUI), приводящей к формированию соответствующего сигнала или команды. События, в свою очередь, требуют инициирования соответствующей реакции, что подразумевает активацию специализированной программы, осуществляющей обработку информации, обусловленной данным событием. Подобная программа именуется обработчиком событий. Процедура обработки события определяется как метод, представляющий собой структурированный алгоритм действий, выполняемых системой при наступлении определенного события.

События обрабатываются экземплярами делегата *EventHandler*, который определен в пространстве имен *System*. Делегат – это тип данных, который позволяет передавать методы как аргументы другим методам. Делегат соответствует методам, не возвращающим результат (*void*), у которых два аргумента.

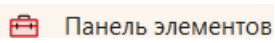
Приведем запись метода-обработчика событий на примере кнопки:

```
private void button1_Click(object sender, EventArgs e)
    {
     //код обработчика события
    }
```

Первый аргумент является объектной ссылкой класса *Object*. Через этот аргумент передается ссылка на объект элемента, на котором произошло событие. Объектной ссылкой класса *EventArgs* (пространство имен *System*) является второй аргумент, через который передается ссылка на объект, содержащий информацию о произошедшем событии.

Элементы управления. Свойства элементов

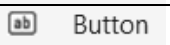
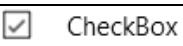
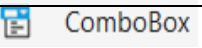
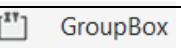
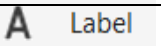
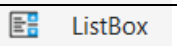
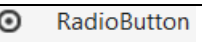
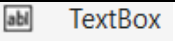
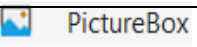
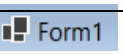
Как уже упоминалось, к элементам управления относятся кнопки, списки, надписи, флажки, переключатели и т. д. (см. рис. 5).

В табл. 15 перечислены некоторые классы из пространства имен *System.Windows.Forms*, которые позволяют создавать объекты для основных графических элементов управления и подключаются через `using System.Windows.Forms;`. Для размещения элементов управления на форме используем панель элементов 

.

Таблица 15

Классы для создания графических элементов

Класс	Описание	Элемент управления	Описание
Button	Класс для создания элемента кнопки		Кнопка, позволяющая реагировать на событие нажатия
CheckBox	Класс для создания элемента опционного типа		Флажок, переключатель или элемент выбора
ComboBox	Класс для создания элемента «список выбора»		Комбинированный список
GroupBox	Класс для создания элемента, используемого при группировке, в частности элементов выбора		Рамка группировки элементов
Label	Класс для создания объекта текстовой метки		Надпись с пояснительным текстом
ListBox	Класс для создания объекта элемента, представляющего собой раскрывающийся список		Список (элемент выбора)
Panel	Класс для создания объекта панели		Панель (контейнер для размещения других элементов управления)
RadioButton	Класс для создания объекта элемента, являющегося переключателем		Радиокнопка (переключатель или элемент выбора)
TextBox	Класс для создания объекта элемента, представляющего собой поле для ввода значений		Поле для ввода и вывода данных
PictureBox	Класс для создания объекта изображения		Окно для вывода изображения на экран
Form	Класс для создания объекта окна формы		Окно интерфейса пользователя

Таким образом, элементы управления представляют собой визуальные классы, которые получают введенные пользователем данные и могут инициировать различные события. Все элементы управления обладают рядом общих свойств, большинство из которых оказывает влияние на визуальное отображение формы. Свойства выделенного объекта отображаются в окне свойств 

и предоставляют пользователю удобный интерфейс для управления своими элементами. Задание свойств позволяет определять состояние элемента управления.

В табл. 16 перечислены некоторые общие свойства, которые можно изменять как в окне *Свойства* по категориям или в алфавитном порядке  , так и программно.

Таблица 16

Свойства графических элементов

Свойство	Описание
1	2
Name	Определяет имя элемента управления, через которое впоследствии можно будет обращаться к данному элементу
BackColor	Указывает на фоновый цвет формы
Background	Задает цвет фона
BackgroundImage	Указывает на фоновое изображение формы
ControlBox	Указывает, отображается ли меню формы
Cursor	Определяет тип курсора, который используется на форме
Enabled	Если данное свойство имеет значение *false*, то эта функция не сможет получать ввод от пользователя, т. е. нельзя инициировать кнопки, ввести текст в текстовые поля и т. д.
Font	Задает шрифт для всей формы и всех помещенных на нее элементов управления
FontStyle	Определяет наклон шрифта, принимает одно из трех значений (Normal, Italic, Oblique)
FontStretch	Определяет, как будет растягивать или сжимать текст: значение Condensed сжимает текст, а Expanded – расстягивает
Foreground	Задает цвет текста элемента управления
ForeColor	Определяет цвет шрифта на форме
FormBorderStyle	Указывает, как будет отображаться граница формы и строка заголовка
HelpButton	Указывает, отображается ли кнопка справки формы
Icon	Задает иконку формы

Окончание табл. 16

1	2
Location	Определяет положение по отношению к верхнему левому углу экрана
MaximizeBox	Указывает, будет ли доступна кнопка максимизации окна в заголовке формы
MinimizeBox	Указывает, доступна ли кнопка минимизации окна
MaximumSize	Задает максимальный размер формы
MinimumSize	Задает минимальный размер формы
Size	Определяет начальный размер формы
StartPosition	Указывает на начальную позицию, с которой форма появляется на экране
Text	Определяет заголовок формы
Visible	Устанавливает параметры видимости элемента и может принимать одно из трех значений: Visible (элемент виден и участвует в компоновке), Collapsed (элемент не виден и не участвует в компоновке), Hidden (элемент не виден, но при этом участвует в компоновке)
WindowState	Указывает, в каком состоянии форма будет находиться при запуске: нормальном, максимизированном или минимизированном

Решение с графическим пользовательским интерфейсом

Цель: создать решение IDE Visual Studio на языке программирования C# и объединить в нем три разработанных проекта (алгоритмы следования, ветвления, повторения), а также разработать графический интерфейс пользователя с названием формы *Мастер проектов решения прикладных задач*.

Запускаем Visual Studio  . Открываем среду разработки при помощи клавиши *Esc* или команды *Продолжить без кода→*, после чего появится окно с главным меню без кода. Затем выбираем *Файл → Создать → Проект*.

В окне создание проекта (рис. 39) в поле поиска вводим *Решение* («Идет поиск, ждите»).

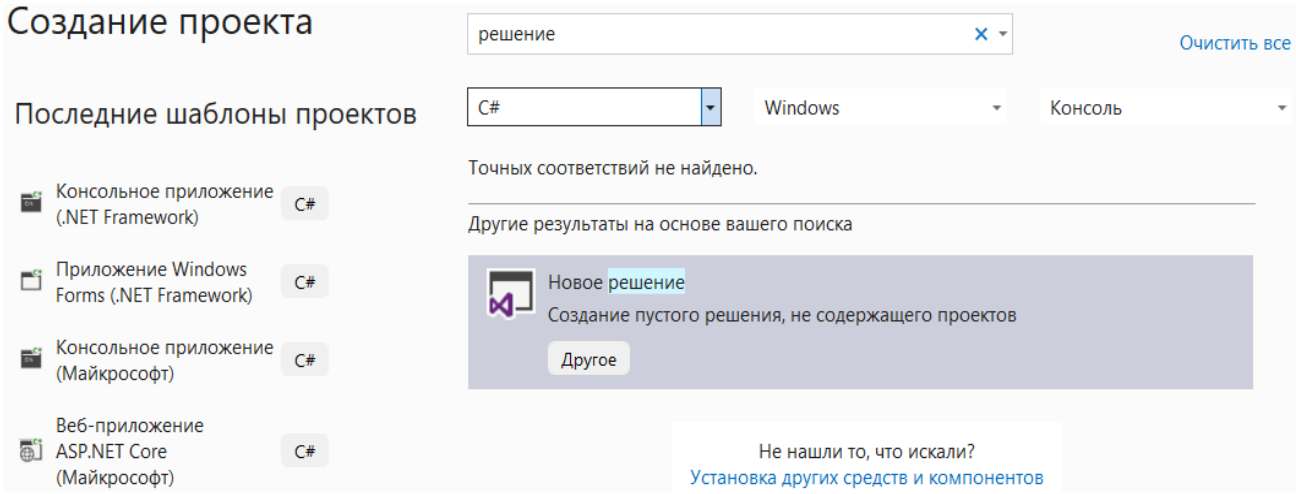


Рис. 39. Рабочее окно создания проекта

После этого выбираем шаблон пустого решения и активируем кнопку *Далее*. В окне *Настроить новый проект* вводим имя решения *Application_C#*, указываем место расположения и нажимаем кнопку *Создать* (рис. 40). В окне *Обозреватель решения* появится Решение "Application_C#" (проекты: 0).

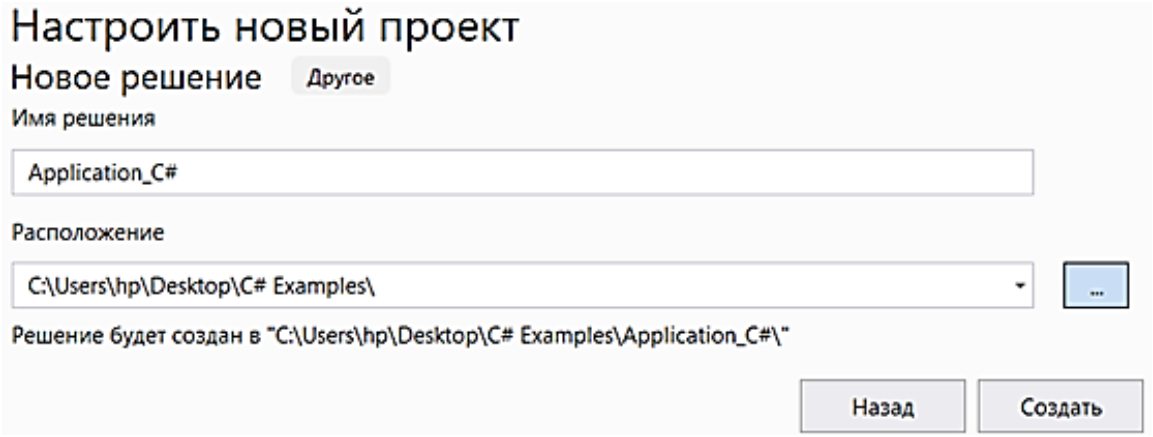


Рис. 40. Рабочее окно нового решения

Затем объединим три созданных ранее проекта в одном решении *Application_C#*. Для этого выделяем Решение "Application_C#" (проекты: 0), а в контекстном меню – *Добавить* → *Существующий проект* (рис. 41). В папке с *Проектом 1* Project_1 выбираем файл с именем *Project_1.csproj* и нажимаем кнопку *Открыть*. Папка проекта появится в окне *Обозреватель решений*.

После этого аналогично подключаем такие проекты, как *Project2* и *Project3*.

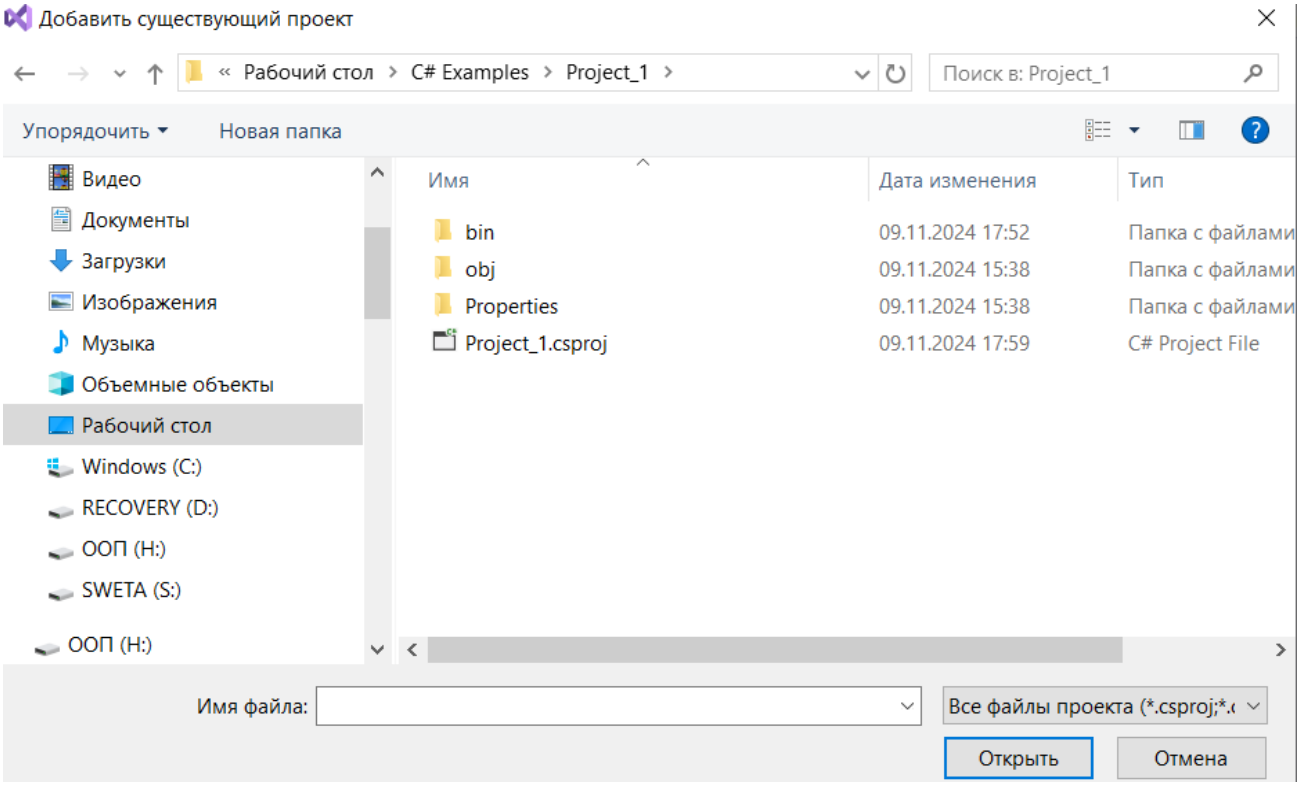


Рис. 41. Окно папки с Проектом 1

Результирующее окно *Обозреватель решений*, содержащее в одном решении три проекта, представлено на рис. 42.

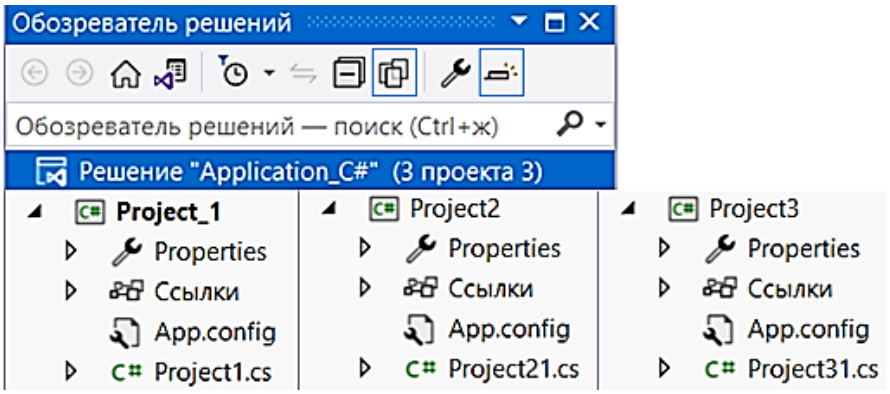


Рис. 42. Окно проектов в одном решении

Необходимо проверить выполнение каждого проекта. Запускаем проект с именем *Project 1*, выбрав в окне *Обозреватель решения* Project_1, а в контекстном меню – *Отладка* → *Запуск без отладки*. Также можно выбрать программу для запуска в раскрывающемся списке панели режима отладки Project_1 Пуск.

Аналогично запускаем на выполнение *Проект 2* и *Проект 3* (рис. 43).

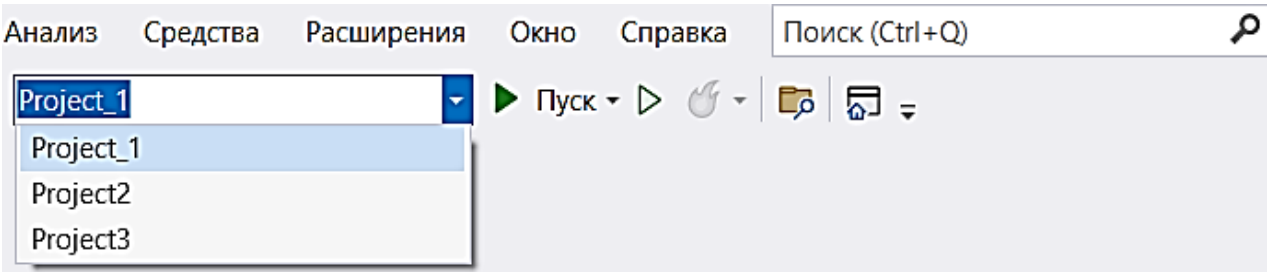


Рис. 43. Окно выбора проектов на выполнение

Активируем в окне *Обозреватель решения* кнопку *Показать все файлы*. Следует обратить внимание, что дополнительно отображаются папки bin и obj, созданные компилятором.

Просмотрев папку с решением на диске, выделим в окне *Обозреватель решений* решение *Application_C#*, после чего выберем в контекстном меню *Открыть папку в проводнике*. В папке решения *Application_C#* находятся файл решения Application_C#.sln и папки трех консольных проектов, каждая из которых содержит соответствующий файл проекта с расширением *.csproj (C# Project file)*.

Разработка оконных приложений – это создание программ с графическим интерфейсом пользователя (GUI). Для быстрой разработки настольных приложений с графическим интерфейсом в C# обычно используется Windows Forms (WinForms) – библиотека классов в .NET Framework. Она предлагает большое количество готовых элементов управления (кнопки, текстовые поля, метки и др.), что значительно упрощает создание классических настольных приложений.

Создадим графический интерфейс пользователя с названием формы *Мастер проектов решения прикладных задач*, содержащий три кнопки для запуска соответствующего проекта и надписи с пояснительным текстом для каждого из них. При этом каждый проект размещается в отдельной группе.

Создадим форму, выделив для этого в окне *Обозреватель решения* Решение "Application_C#" (3 проекта 3), а из контекстного меню выбираем *Добавить* → *Создать проект*. В окне поиска введем *winforms* («Идет поиск, ждите»), выберем приложение *Windows Forms* (Майкрософт) Приложение Windows Forms (Майкрософт) Шаблон проекта для создания приложения .NET Windows Forms (WinForms). Активируем кнопку *Далее*. В окне *Настроить проект* вводим имя проекта – *Master_Application_C#* и повторно нажимаем кнопку *Далее*.

Если создавать форму отдельно от решения, то в окне *Настроить проект* в поле *Решение* необходимо выбрать Добавить в решение. После этого

в рабочем окне появится форма *Form1*, а в окне *Обозреватель решений* – проект *Master_Application_C#* (рис. 44).

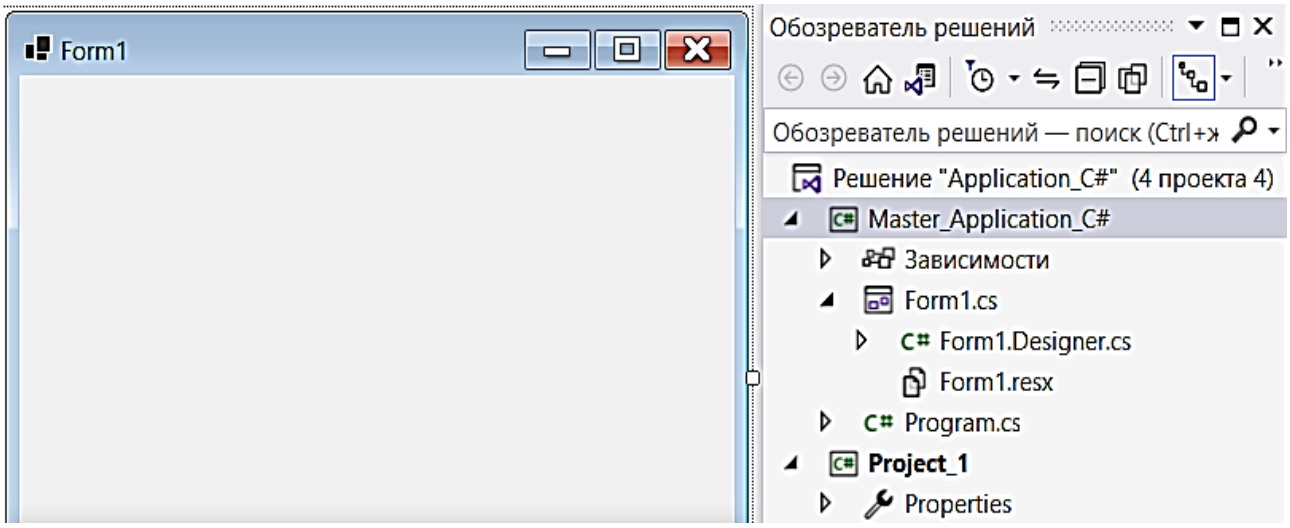


Рис. 44. Окно с формой для заполнения

Изменяем имя *Program.cs* в проекте *Master_Application_C#* (окно *Обозреватель решения*) на *Program_Form.cs* Program_Form.cs. Настраиваем окна для работы с формой:

– окно *Панель элементов* – *Вид* → Панель элементов;

– окно *Свойства* – *Вид* → Окно свойств.

Для открытия формы в режиме графического дизайнера нажимаем в окне *Обозреватель решений* на файл Form1.cs либо левой кнопкой мыши двойным кликом, либо правой кнопкой мыши и в появившемся контекстном меню выбираем Открыть в конструкторе.

Изменив заголовок окна формы *Form1* на *Мастер проектов решения прикладных задач*, в окне *Свойства* находим свойство формы *Text* Text Form1 и в поле с *Form1* вводим название формы *Мастер проектов решения прикладных задач*.

Открываем файл программы для формы и выбираем команду меню *Окно* → 1 Program_Form.cs* (рис. 45).

Программа для формы является главным модулем и содержит две инструкции, которые вызываются при помощи метода *Main()* в файле *Program_Form.cs*.

Метод *ApplicationConfiguration.Initialize()* устанавливает базовую конфигурацию приложения, а метод *Application.Run(new Form1())* запускает скрытый цикл обработки сообщений операционной системы (ОС) и делает видимой форму, которая передается в качестве параметра.

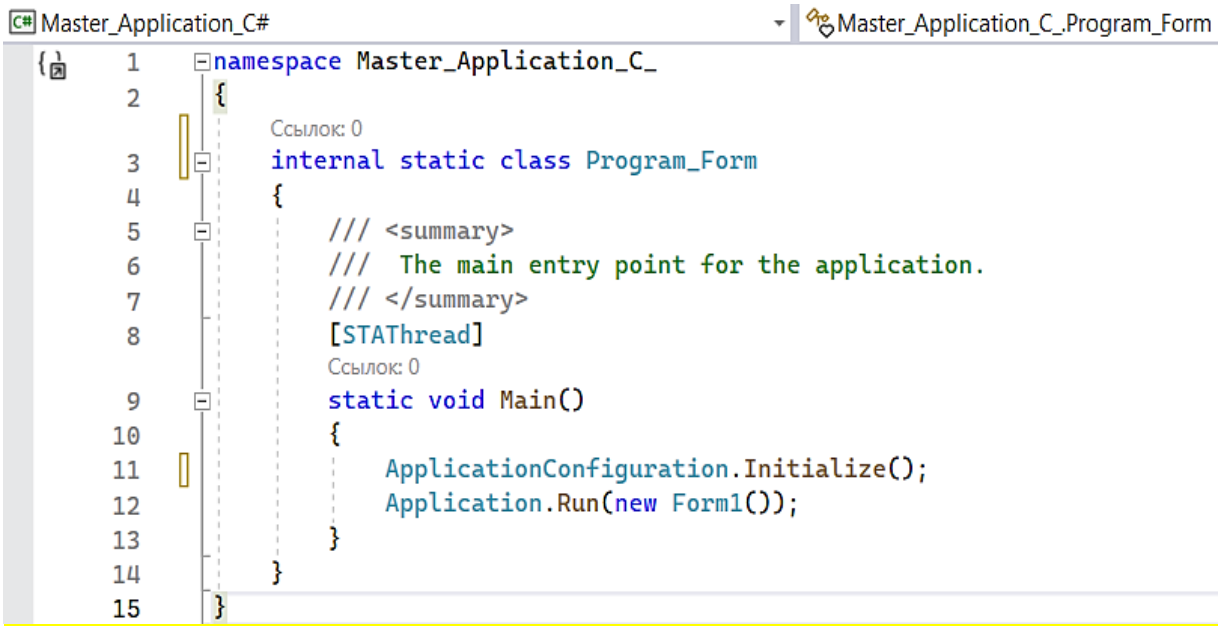


Рис. 45. Окно программы для формы

Разместим в поле формы три групповые рамки для проектов. Для добавления элементов управления на форму используем один из следующих способов:

1) в панели элементов кликнуть левой кнопкой мыши на нужном элементе и, не отпуская кнопки, перетянуть элемент в требуемое место формы;

2) выполнить двойной щелчок левой кнопкой мыши на нужном значке панели элементов; когда требуемый элемент появится в верхнем углу формы, его нужно переместить в заданное место формы, придав необходимые размеры и свойства; если же элемент является невидимым в процессе разработки (например, как окно OpenFileDialog), то он размещается в нижней части экрана под формой;

3) кликнуть правой кнопкой мыши на элементе, размещенном на форме, и выбрать *Копировать*, затем снова щелкнуть кнопкой на форме, выбрав *Вставить*, после чего будет создана копия данного элемента.

Установим на форме кнопку *Button1*. Найдем кнопку Button на панели элементов и, захватив ее указателем мыши, перенесем на форму. При выборе кнопки в окне *Свойства* отобразятся все свойства объекта *Button1* (рис. 46).

Заменим текст в окне кнопки *button1* на *Выполнить Проект 1*, изменив поле *Text* в окне *Свойства*. Выберем для кнопки в категории *Font* шрифт, размер и стиль: Font **Arial; 10,2pt; style=Bold**. Данная категория устанавливает шрифт, используемый для отображения текста на элементе управления.

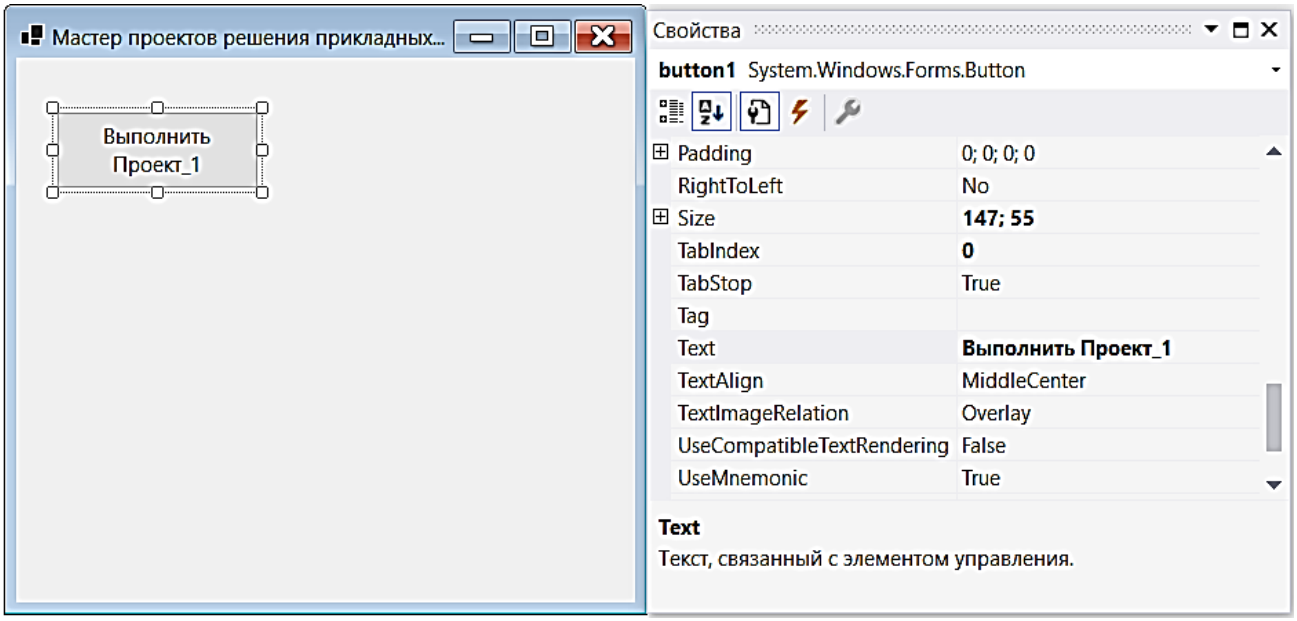


Рис. 46. Окно свойств кнопки Button1

Устанавливаем на форме метку *Label1*. Находим метку Label на панели элементов и, захватив ее указателем мыши, переносим на форму. При выборе метки в окне *Свойства* отображаются все свойства объекта *Label1*.

Заменим текст в окне метки *Label1* на *Алгоритмическая структура СЛЕДОВАНИЕ*, изменив поле *Text* в окне *Свойства*: Text Алгоритмическая структур. Раскрывающийся список открывает поле для ввода и редактирования текста. Выбираем для метки в категории *Font* шрифт, размер и стиль: Font Arial; 10,2pt; style=Bold.

Выбираем для метки поле *AutoSize*, установив в нем *False*: AutoSize False. *AutoSize* определяет, будет ли размер элемента управления автоматически изменяться в соответствии с размером его содержимого.

Для выравнивания текста метки по центру выбираем поле *TextAlign* TextAlign MiddleCenter, которое определяет положение текста в границах подписи. Разместим на форме кнопку и метку (рис. 47).

Разместим групповую рамку GroupBox из категории *Контейнеры панели элементов* в поле формы и, захватив ее указателем мыши, перенесем в заданное место формы. При выборе групповой рамки в окне *Свойства* отобразятся все свойства объекта *GroupBox1*, который предназначен для логической группировки коллекции элементов в форме.

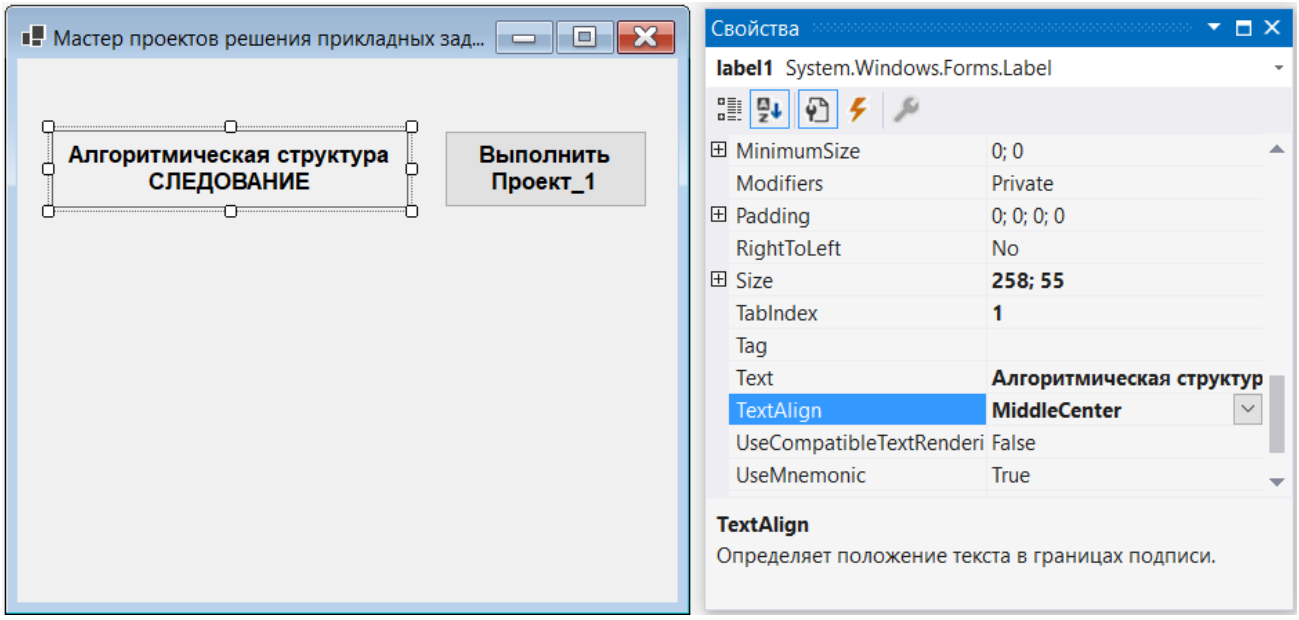


Рис. 47. Окно свойств метки Label1

Заменим текст в окне групповой рамки *groupBox1* на *Проект_1*, изменив поле *Text* в окне *Свойства*: Text **Проект_1**.

Раскрывающийся список открывает поле для ввода и редактирования текста. Выберем для групповой рамки в категории *Font* шрифт, размер и стиль Font **Arial; 10,8pt; style=Bold,** ... (рис. 48).

Графический дизайнер позволяет автоматически сгенерировать обработчик события элемента управления. Для этого в окне дизайнера нажимаем на элемент управления на форме двойным кликом мыши. Затем открываем файл *Form1.cs* (рис. 49).

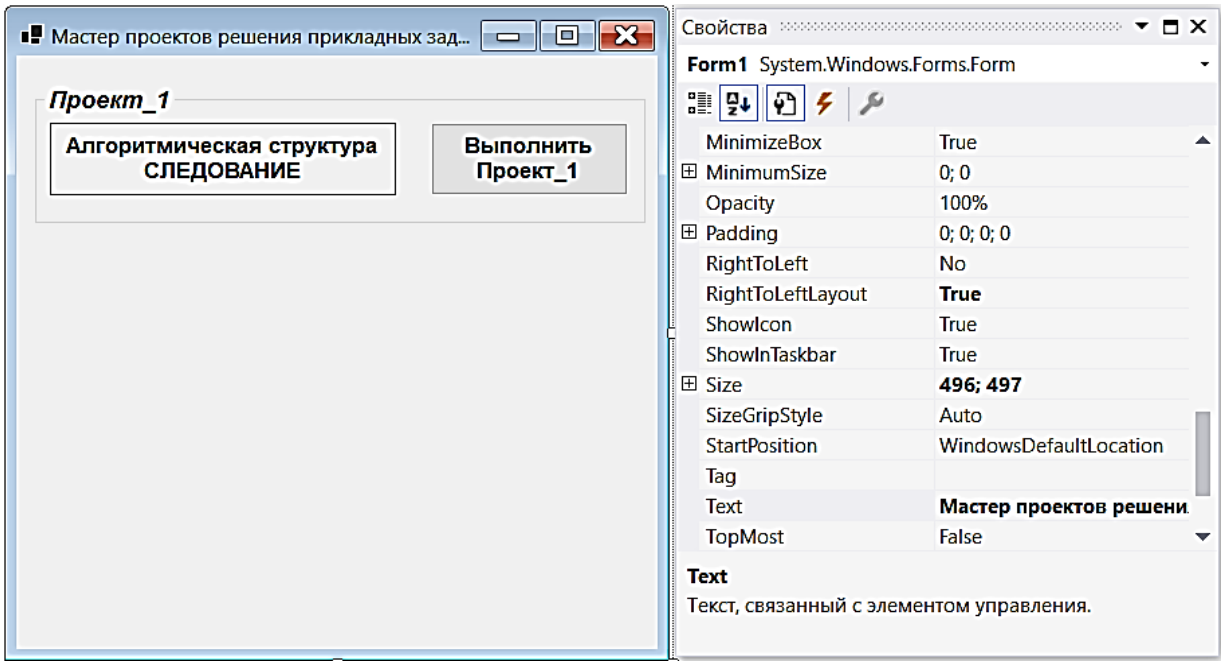


Рис. 48. Окно группировки коллекции

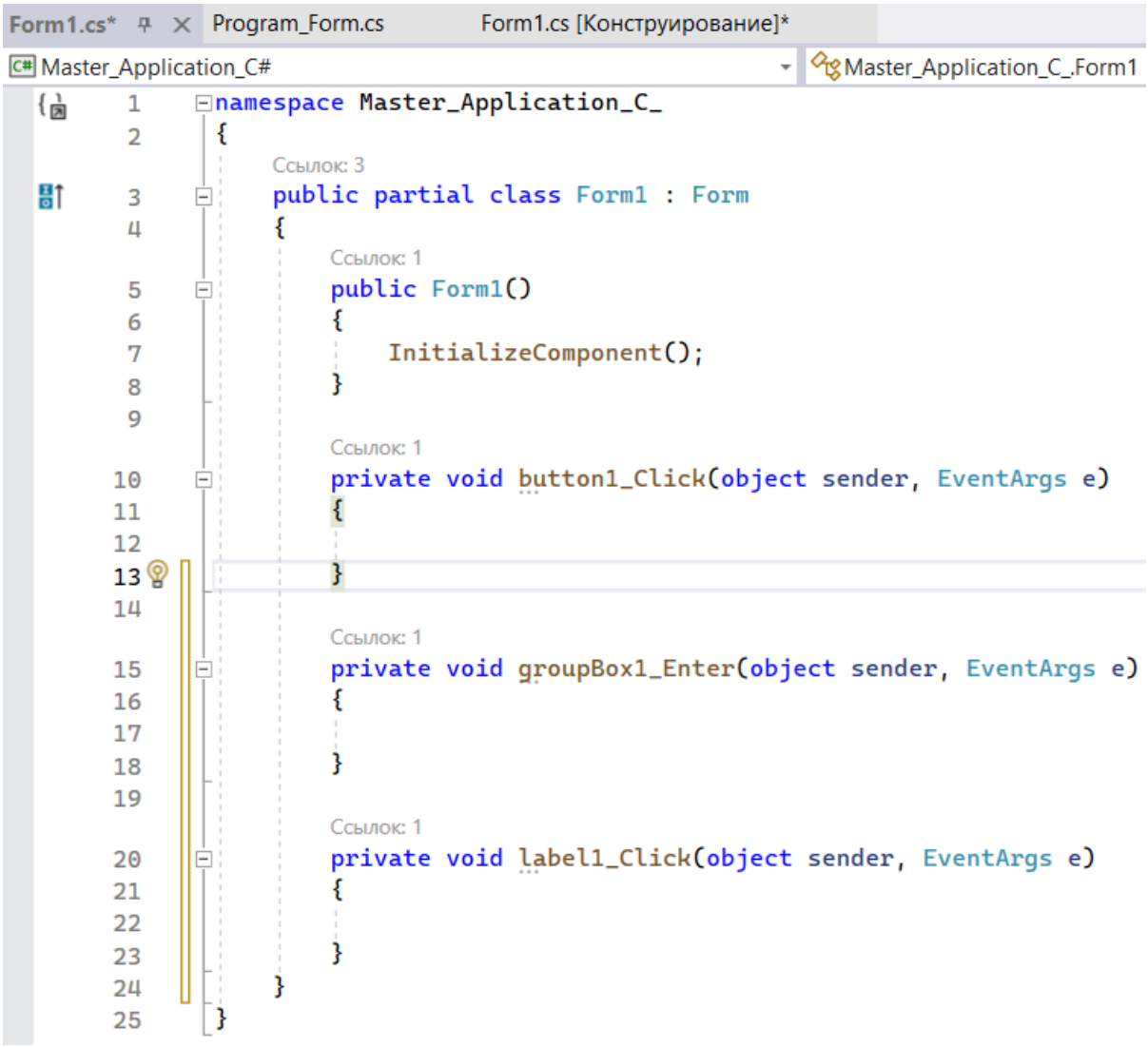


Рис. 49. Окно кода для элементов управления

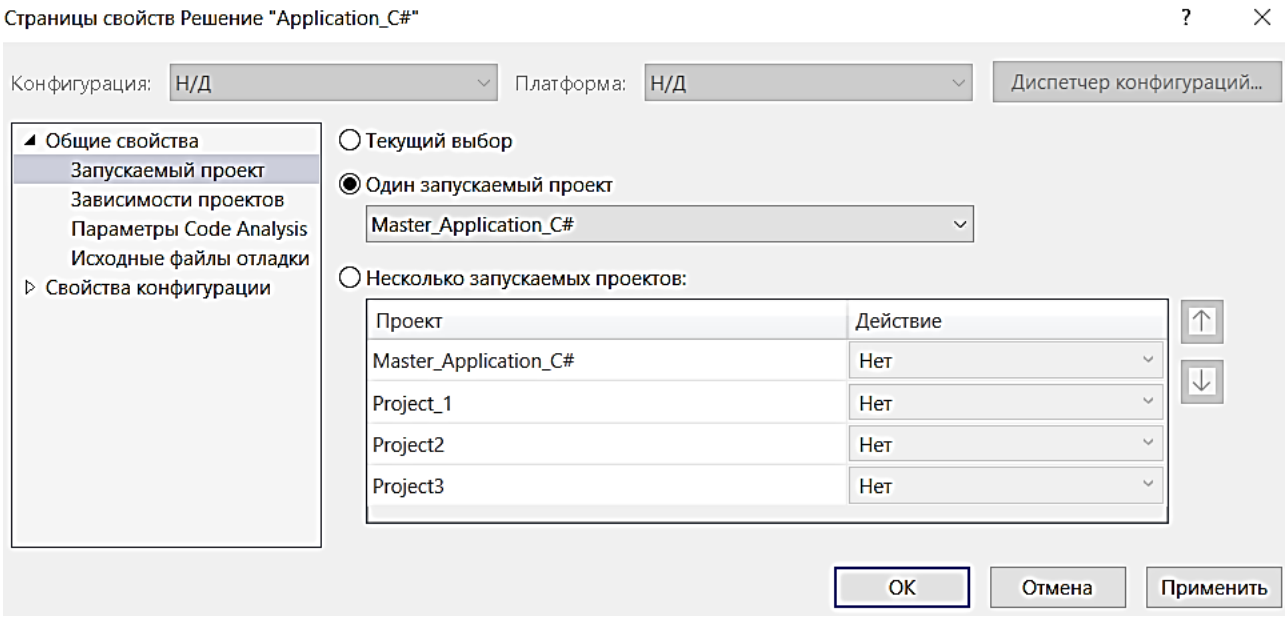


Рис. 50. Окно страницы свойств решения

Для каждого элемента управления, установленного на форму, в коде появится блок, сгенерированный обработчиком событий.

Проверяем работу нажатия кнопки *Выполнить Проект_1*, добавив в блок кода `private void button1_Click(object sender, EventArgs e)` инструкцию `MessageBox.Show("Здесь должен выполняться проект1");`.

Запускаем проект на выполнение. В окне *Обозреватель решений* выделим решение Решение "Application_C#" (4 проекта 4), а в контекстном меню выбираем *Свойства* (рис. 50).

Устанавливаем переключатель *Один запускаемый проект*, выбрав из списка *Master_Application_C#*. Нажимаем последовательно кнопки *Применить* и *ОК*, в меню выбираем *Отладка* → ▷ *Запуск без отладки*. После процесса сборки на экране откроется окно формы (рис. 51).

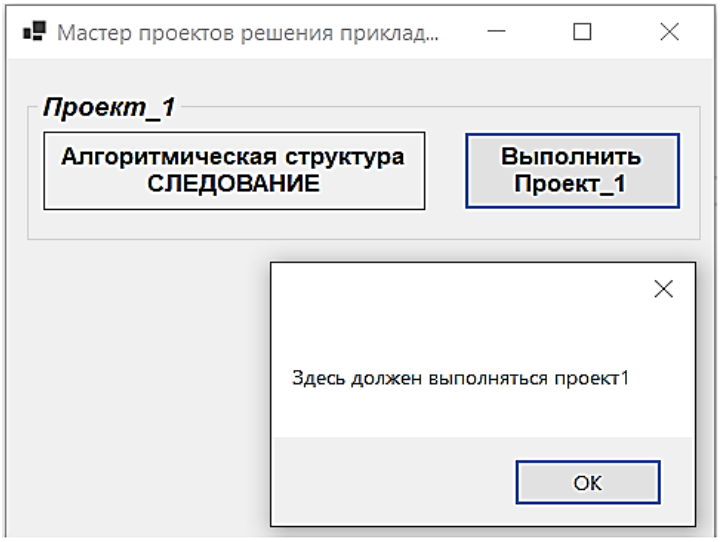


Рис. 51. Окно формы Мастер проектов

Опция *Один запускаемый проект* дублирует поле выбора в режиме отладки Master_Application_C# ▶ Master_Application_C# ▷, где в раскрывающемся списке выбираем программу для запуска.

Опция *Текущий выбор* означает, что в окне *Обозреватель решений* пользователем выбран проект для запуска и инициирована *Отладка*.

Опция *Несколько запускаемых проектов* позволяет запустить одновременно все проекты. Для этого выбираем действие для каждого проекта и нажимаем *Применить* и кнопку *ОК*. Для каждого проекта запускается свое консольное окно.

Проанализируем работу в режиме запуска нескольких проектов

Добавляем на форму *Мастер проектов решения прикладных задач* элементы управления для *Проекта 2* и *Проекта 3* (рис. 52).

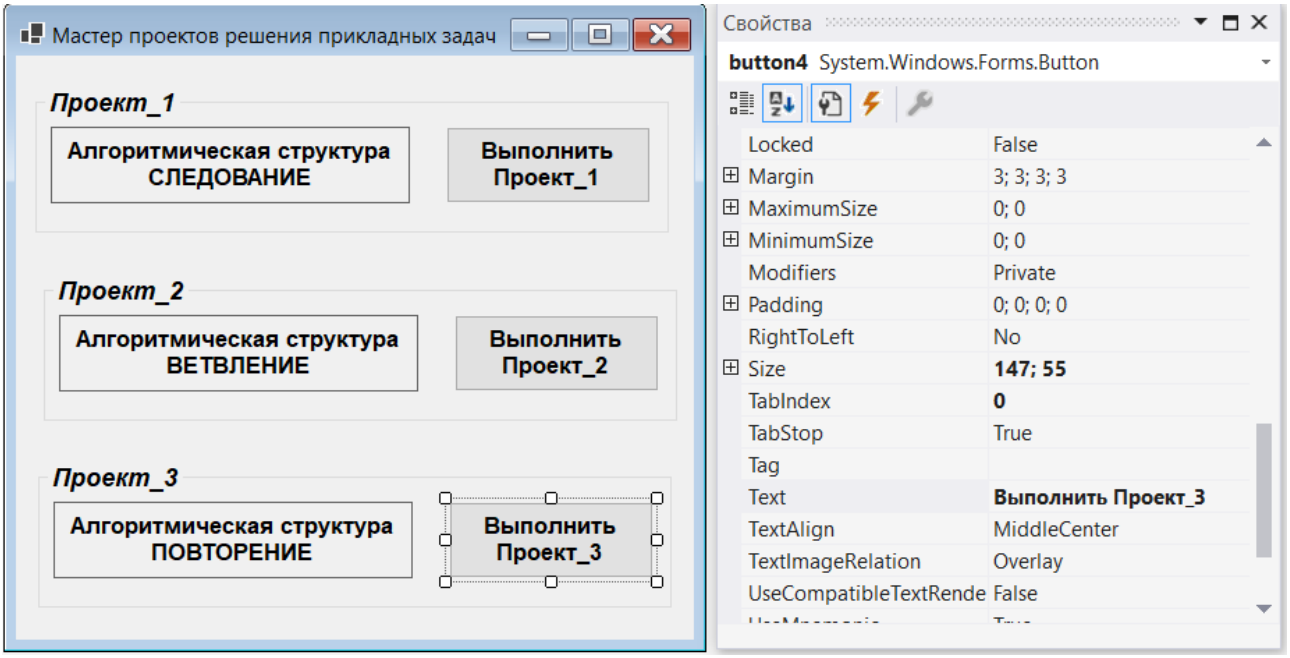


Рис. 52. Окно формы для проектов

Выделим все элементы для *Проекта 1*, для этого наведем указатель мыши на в рамке и, не отпуская клавишу *Ctrl*, переместим выделенную группу.

Для каждого проекта вносим изменения в окно *Свойства*. Так как при копировании элемента сохраняются все свойства форматирования, вносим изменения только в текст. С этой целью выделим кнопку *button2* и заменим текст в окне *Свойства* поле *Text* на *Выполнить проект_2*. Аналогично изменим поле *Text* для метки **label2** System.Windows.Forms.Label и групповой рамки **groupBox2** .

После этого выполним копирование для *Проекта 3* и внесем изменения.

Открыв *файл Form1.cs*, проанализируем его и добавим в код новые элементы управления. Для этого в окне формы каждый новый элемент управления дважды инициируем при помощи мыши. В код формы по умолчанию добавим метод сгенерированного события, например для кнопки `private void button1_Click(object sender, EventArgs e);` с параметрами *object sender* он инициирует событие, а с параметрами *EventArgs e* – хранит информацию о нем.

Для взаимодействия с пользователем в Windows Forms используется механизм событий. Все события для выбранного объекта отображаются в окне *Свойства*. Выделив форму *Form1*, в окне *Свойства* на панели инструментов нажимаем *Свойства*. На рис. 53 в поле *Load* представлены все свойства для рабочей формы *Мастер проектов*.

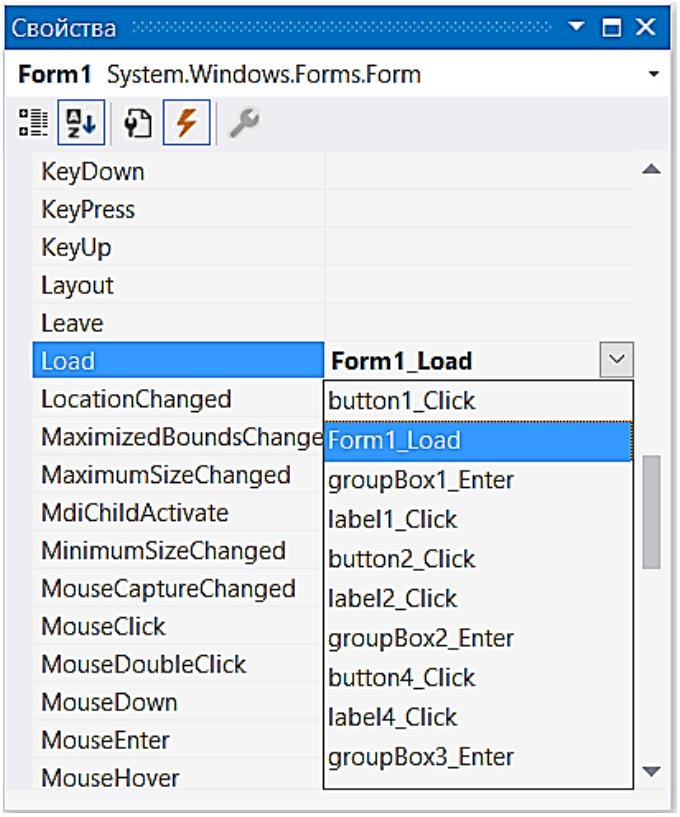


Рис. 53. Окно методов формы проекта

Разместим на форме *Form1* в каждой группе элемент управления TextBox, в который будет выводиться информация о пути доступа к файлу приложения для запуска соответствующего проекта. При нажатии кнопки *Выполнить* в поле *textBox* должно выдаваться сообщение и открываться приложение.

После установления на форме поля *TextBox1* захватываем указателем мыши на панели элементов кнопку TextBox и переносим ее на форму в группу *Проекта 1*. При выборе текстового поля в окне *Свойства* отобразятся все свойства объекта *TextBox1* (рис. 54).

Откроем файл кода *Form1.cs* и дополним его следующими инструкциями:

```
private void button1_Click(object sender, EventArgs e)
    {
      // Запуск на выполнение Проекта_1
      string fileName = "C:\\Users\\hp\\Desktop\\C# Examples\\" +
         "Project_1\\bin\\Debug\\Project_1.exe";
      textBox1.Text = fileName.ToString();
      Process process = new Process();
      process.StartInfo.FileName = fileName;
      process.Start();
    }
```

Для работы с файлами на языке C# можно применять абсолютный путь доступа. Полный путь доступа к файлу указывает инструкция string fileName = @"C:\\Users\\...\\Project_1.exe" или string fileName = @"C:\\Users\\...\\Project_1.exe". Инструкция `textBox1.Text = fileName.ToString()` выводит в поле информацию по размещению файла приложения на диске.

Запускаем проект на выполнение ▷ *Запуск без отладки*.

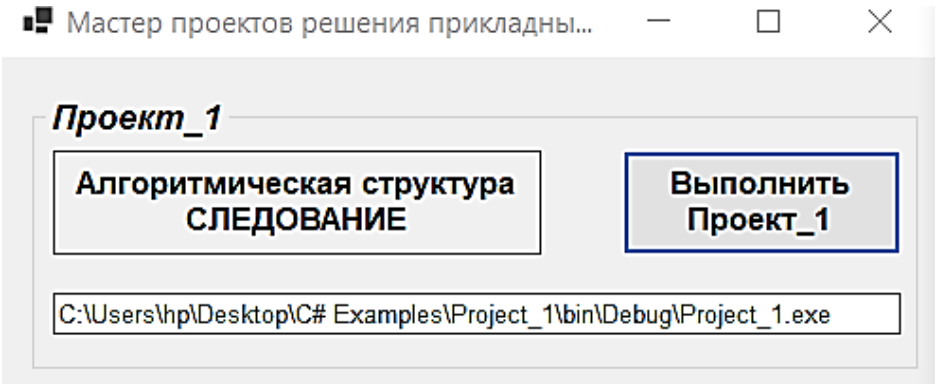


Рис. 54. Окно метода доступа к файлу

Устанавливаем для каждого проекта свой полный путь доступа к файлу. При запуске приложения операционная система создает для него отдельный процесс, представленный классом *Process* из пространства имен `using System.Diagnostics`. Этот класс предоставляет доступ к локальным и удаленным процессам, а также позволяет запускать и останавливать локальные системные процессы.

Новый объект класса *Process* в C# создается инструкцией `Process process = new Process()`.

Инструкция `process.StartInfo.FileName = fileName` применяется, чтобы задать исполняемый файл, для которого не был запущен связанный процесс.

Статический метод *process.Start()* запускает процесс. Запускаем решение на выполнение и проверяем все проекты.

Результат работы *Мастера проектов* для *Проекта 2* представлен на рис. 55.

Таким образом, в результате работы в интегрированной среде разработки Visual Studio разработано решение, содержащее четыре проекта: алгоритм следования, алгоритм ветвления, алгоритм повторения, а также графический интерфейс пользователя *Мастер проектов решения прикладных задач*.

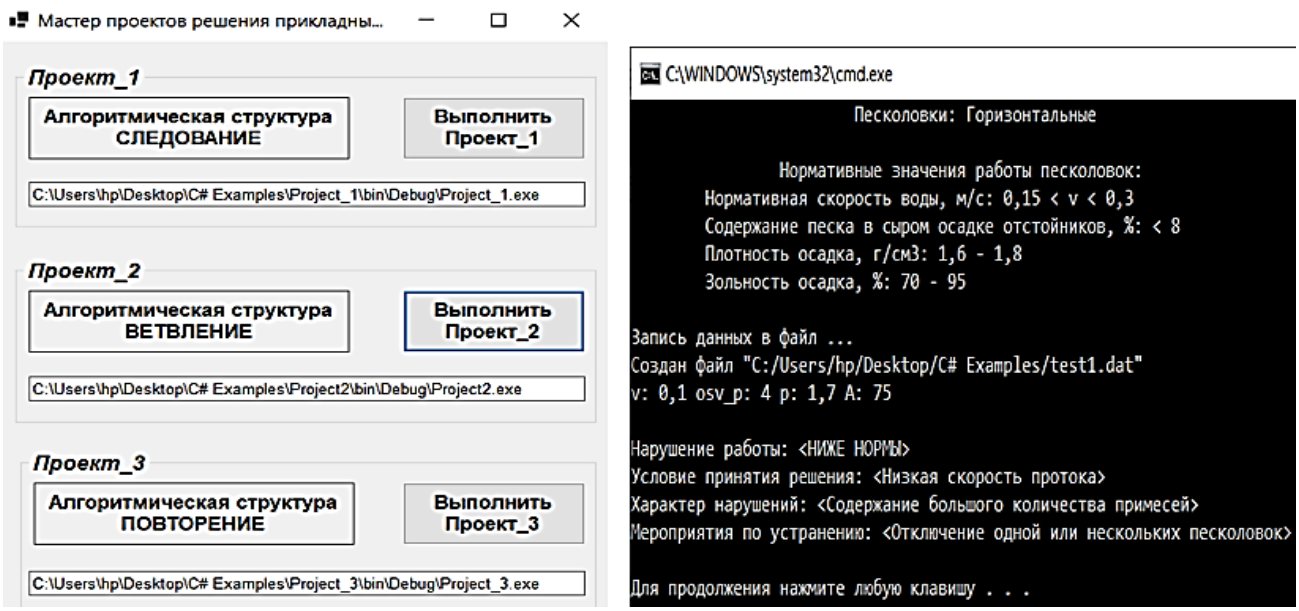


Рис. 55. Окно Мастера проектов

Контрольные вопросы

1. Что такое программирование?
2. Для чего предназначен язык программирования и что он определяет?
3. Что собой представляет графический интерфейс пользователя?
4. Как называются графические объекты пользовательского интерфейса?
5. Что представляет собой форма?
6. Перечислите наиболее часто используемые элементы управления и их свойства.
7. Назовите две задачи, при помощи которых реализуется создание приложений с графическим интерфейсом.
8. Дайте определение графического пользовательского интерфейса.
9. Какими нормативными документами регулируются требования к элементам графического интерфейса?
10. Перечислите состояния графических элементов интерфейса пользователя.
11. Перечислите общие принципы построения графического пользовательского интерфейса.
12. Сформулируйте постановку задачи для разработки приложения с графическим интерфейсом.
13. Какие элементы управления называются статическими?

14. Какие элементы управления называются функциональным?

15. Что представляет собой событие?

16. Что представляет собой обработчик событий?

17. Как обрабатываются события?

18. Приведите пример записи метода – обработчика событий.

19. Как называются графические объекты пользовательского интерфейса?

20. Назовите основные классы для создания графических элементов.

21. Какие элементы управления вы знаете?

22. Перечислите некоторые общие свойства элементов управления.

23. Как создать форму в приложении *Windows Forms*?

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Проект 1. Разработать алгоритм решения прикладной задачи с применением базовой структуры следования (линейный алгоритм) и решить ее в соответствии с этапами решения прикладной задачи с использованием интегрированной среды разработки Visual Studio 2022.

Задача 1[8]. Кислород объемом 5 м^3 при $p_1 = 3$ бар и $T_1 = 320°$ К находится в резервуаре. По трубопроводу в него подается 0,6 м^3 углекислого газа при $p_2 = 12$ бар и $T_2 = 400°$ К. В условиях постоянства удельной теплоемкости системы произвести расчет параметров состояния смеси газов.

Задача 2[9]. В модельном аппарате диаметром 500 мм, оснащенном четырьмя отражательными перегородками, проведены опыты по окислению сульфида аммония до тиосульфата посредством кислорода. В ходе исследований было выявлено, что при перемешивании закрытой турбинной мешалкой диаметром 125 мм, с частотой вращения 6 об/с и удельной производительностью $k_м = 1{,}04 \cdot 10^{-3}$ м^3 O_2/(м^3/с) при удельном расходе кислорода $u_м = 3{,}5 \cdot 10^{-3}$ м^3 O_2/(м^3/с) обеспечивается поглощение 1,04 дм^3 кислорода на 1 м^3 реакционной среды за 1 с.

На основании результатов модельных опытов проектируется промышленный аппарат диаметром 2 м с мешалкой диаметром 0,5 м, геометрически подобный модельному ($\Gamma_D = D/d = 4$). Необходимо рассчитать частоту вращения мешалки в промышленном аппарате, при которой будет обеспечена такая же удельная производительность, как и в модельных условиях. Расход кислорода в промышленном аппарате $u_п = 3{,}5 \cdot 10^{-3}$ м^3 O_2/(м^3/с); физические свойства реакционной среды в модельном и в промышленном аппаратах близки к свойствам воды. Температура реакции 80 °С.

Проект 2. Разработать алгоритм решения прикладной задачи с применением базовой структуры ветвление (разветвляющийся алгоритм) и решить ее в соответствии с этапами решения прикладной задачи с использованием интегрированной среды разработки Visual Studio 2022.

---

[8] Нащокин В. В. Техническая термодинамика и теплопередача. М.: Высш. шк., 1980. 469 с.

[9] Павлов К. Ф., Романков П. Г., Носков А. А. Примеры и задачи по курсу процессов и аппаратов химической технологии. Л.: Химия, 1981. 560 с.

Задача. Проверить технологическую эффективность работы решеток по показателям эффективности (нормативным значениям) их работы. На основании условия принятия решения указать характер и причины нарушений, а также определить мероприятия по устранению нарушений.

Решетки на очистных сооружениях применяются для задержания из сточных вод крупных и волокнистых материалов (остатки пищи, бумага, тряпки, упаковочные материалы и др.) и являются сооружениями предварительной очистки. Основным элементом решеток является рама с рядом металлических стержней, расположенных параллельно друг другу и создающих плоскость с прозорами, через которую процеживается вода. Решетки классифицируются следующим образом: решетки с ручной очисткой, механизированные решетки, решетки-дробилки.

Для обеспечения эффективного задержания крупного мусора решетки должны при своей работе обеспечивать следующие технологические параметры (контролируемые показатели):

– для решеток с ручной очисткой нормативная величина скорости движения сточных вод должна составлять не менее 0,4 м/с, при максимальном притоке – не более 1,0 м/с;

– для механизированных решеток нормативная величина скорости движения сточных вод должна составлять не менее 0,8 м/с, при максимальном притоке – не более 1,0 м/с;

– для решеток-дробилок нормативная величина скорости движения сточных вод должна превышать 1,2 м/с.

При эксплуатации решеток ежемесячно осуществляется технологический контроль их работы. Нарушение нормальной работы решеток связано с засорением или несоблюдением правил эксплуатации.

Одним из определяющих моментов при эксплуатации решеток является регулирование скоростей в зависимости от расхода поступающих сточных вод, которые определяются нормативными показателями для каждого вида решеток. Нарушение режима работы определяется выходом за нормативные пределы.

Определяем условия принятия решений по контролируемым показателям для разного вида решеток. Если контролируемое значение входит в интервал нормативных значений, то все решетки работают в нормальном режиме. Если контролируемое значение выходит из интервала нормативных значений, то возможны следующие алгоритмы действий для различных разновидностей решеток:

1. Решетки-дробилки: если сверхнормативная скорость подачи $\upsilon > 1{,}2$ м/с, то решетки работают с нарушением технологического режима; характер нарушений – образование плотных тромбов; забивание отверстий решеток, закупоривание пазух и рабочих зазоров, механизмов; мероприятия по устранению – установка дополнительных решеток.

2. Механизированные решетки:

а) если сверхнормативная скорость подачи $\upsilon > 1{,}0$ м/с, то решетки работают с нарушением технологического режима; характер нарушений – низкий эффект задержания отбросов в связи с продавливанием отбросов и их проскоком между прутьями; мероприятия по устранению – подвод сточных вод под углом к прямоугольным стержня;

б) если низкая скорость протока $\upsilon < 0{,}8$ м/с, то решетки работают с нарушением технологического режима; характер нарушений – задержание крупных фракций песка, что недопустимо; мероприятия по устранению – повышение скорости подачи воды.

3. Решетки с ручной очисткой:

а) если сверхнормативная скорость подачи $\upsilon > 1{,}0$ м/с, то решетки работают с нарушением технологического режима; характер нарушений – низкий эффект задержания отбросов в связи с продавливанием отбросов и их проскоком между прутьями; мероприятия по устранению – подвод сточных вод под углом к прямоугольным стержням;

б) если низкая скорость протока $\upsilon < 0{,}4$ м/с, то решетки работают с нарушением технологического режима; характер нарушений – задержание крупных фракций песка, что недопустимо; мероприятия по устранению – повышение скорости подачи воды.

Проект 3. Разработать алгоритм решения прикладной задачи с применением базовой структуры повторение (циклический алгоритм) и решить ее в соответствии с этапами решения прикладной задачи с использованием интегрированной среды разработки Visual Studio 2022.

Задача. Плотность воздуха убывает с высотой по закону $\rho = \rho_0 e^{-hz}$. Считая, что $\rho_0 = 1{,}29$ кг/м^3, $z = 1{,}25 \cdot 10^{-4}$ 1/м, составить таблицу зависимости плотности от высоты для значений от 0 до 1000 м с шагом 100 м.

ЗАКЛЮЧЕНИЕ

Интегрированная среда разработки (IDE) представляет собой ключевой инструментально-программный комплекс, предназначенный для осуществления процесса программирования и проектирования разнообразной прикладной функциональности, включая консольные приложения, сайты, веб-сервисы, настольные и мобильные приложения.

Язык программирования C# создан для программирования в Windows и вместе со средой разработки IDE Visual Studio 2022 позволяет разрабатывать эффективные приложения, имеющие удобный графический интерфейс для решения прикладных задач. В настоящее время язык C#, характеризующийся высокой производительностью, динамичным развитием и широким спектром применимости, занимает лидирующие позиции среди инструментальных средств разработки программного обеспечения в ИТ-отрасли. Текущей версией языка является версия C# 13, которая вышла в ноябре 2024 г. вместе с релизом .NET 9. Применение IDE Visual Studio 2022 помогает значительно упростить разработку приложений, повысить производительность и сделать продуктивность работы максимальной.

Авторы надеются, что предлагаемое учебно-методическое пособие поможет обучающимся в формировании следующих необходимых компетенций:

– навыки по применению к решению прикладных задач базовых алгоритмов обработки информации;

– способность работать в интегрированной среде разработки и реализовывать с применением языка программирования разработанные алгоритмы;

– умение применять справочную документацию и средства отладки среды разработки;

– знание структуры проекта и умение выполнять оценку его сложности.

Таким образом, полученные навыки в области программирования обеспечат студентам базу для дальнейшего профессионального роста и помогут им строить продуктивные деловые отношения.

ЛИТЕРАТУРА

1. Бельков, С. А. Прикладное программирование с использованием языка С-Шарп: учебно-метод. пособие / С. А. Бельков. – Екатеринбург: Изд-во Урал. ун-та, 2017. – 120 с.

2. Биллиг, В. А. Основы программирования на C#: учебное пособие / В. А. Биллиг. – Москва: ИНТУИТ, Ай Пи Ар Медиа, 2021. – 573 с.

3. Зыков, С. В. Программирование. Объектно-ориентированный подход : учебник и практикум для вузов / С. В. Зыков. – Москва: Юрайт, 2023. – 155 с.

4. Зыков, С. В. Программирование : учебник и практикум для вузов / С. В. Зыков. – Москва: Юрайт, 2023. – 320 с.

5. Казанский, А. А. Программирование на Visual C#: учебное пособие для вузов / А. А. Казанский. – 2-е изд., перераб. и доп. – Москва: Юрайт, 2024. – 192 с.

6. Котов, О. М. Язык C#: краткое описание и введение в технологии программирования: учебное пособие / О. М. Котов. – Екатеринбург : Изд-во Урал. ун-та, 2014. – 208 с.

7. Кудрина, Е. В. Основы алгоритмизации и программирования на языке C#: учебное пособие для вузов / Е. В. Кудрина, М. В. Огнева. – Москва: Юрайт, 2023. – 322 с.

8. Пахомов, Б. И. C# для начинающих / Б. И. Пахомов. – Санкт-Петербург: БХВ-Петербург, 2014. – 432 с.

9. Подбельский, В. В. Программирование. Базовый курс C#: учебник для вузов / В. В. Подбельский. – Москва: Юрайт, 2023. – 369 с.

10. Прайс, М. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Прайс. – Санкт-Петербург: Питер, 2023. – 848 с.

11. Тузовский, А. Ф. Объектно-ориентированное программирование: учебное пособие для вузов / А. Ф. Тузовский. – Москва: Юрайт, 2022. – 206 с.

12. Шилдт, Г. C# 4.0: полное руководство / Г. Шилдт. – Москва: Вильямс, 2011. – 1056 с.

УЧЕБНОЕ ИЗДАНИЕ

Светлана Алексеевна Понкратова
Антон Станиславович Понкратов
Лариса Юрьевна Кошкина
Денис Владимирович Тунцев

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ
ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

Редактор Е. И. Шевченко
Компьютерная верстка и макет – А. К. Рахманкулова

Подписано в печать 27.12.2024 | Формат 60×84 1/16
Бумага офсетная | Печать цифровая | 4,88 усл. печ. л.
5,25 уч.-изд. л. | Тираж 400 экз. | Заказ 134/24

Издательство Казанского национального исследовательского технологического университета

Отпечатано в офсетной лаборатории Казанского национального исследовательского технологического университета

420015, Казань, ул. К. Маркса, 68