

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Саратовский государственный технический
университет имени Гагарина Ю.А.»

Филиал федерального государственного бюджетного образовательного
учреждения высшего образования
«Саратовский государственный технический университет
имени Гагарина Ю.А.» в г. Петровске



УТВЕРЖДАЮ
Директор филиала СГТУ
имени Гагарина Ю.А. в г.Петровске
Е.А.Бесшапошникова
«30» июня 2025 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

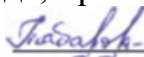
по дисциплине

ОП.04 «Основы алгоритмизации и программирования»

направление подготовки

09.02.07 «Информационные системы и программирование»

Методические указания рассмотрены
на заседании предметной (цикловой) комиссии
общепрофессиональных дисциплин и
профессиональных модулей
«16» июня 2025 года, протокол №13

Председатель ПЦК  /Ю.А.Табарова/

Петровск 2025

Пояснительная записка

Методические указания по выполнению практических работ подготовлены на основе рабочей программы учебной дисциплины ОП.04 «Основы алгоритмизации и программирования», разработанной на основе ФГОС СПО по специальности 09.02.07 «Информационные системы и программирование» и соответствующих общих (ОК) компетенций:

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста.

ОК 09. Пользоваться профессиональной документацией на государственном и иностранном языках

При выполнении практических работ студент должен **знать**:

- понятие алгоритмизации, свойства алгоритмов, общие принципы построения алгоритмов, основные алгоритмические конструкции;
- классификацию языков программирования; понятие системы программирования;
- основные элементы языка, структура программы;
- методы реализации типовых алгоритмов;
- операторы и операции, управляющие структуры, структуры данных, классы памяти;
- понятие подпрограммы, библиотеки подпрограмм;
- объектно-ориентированная модель программирования, основные принципы объектно-ориентированного программирования на примере алгоритмического языка: понятие классов и объектов, их свойств и методов, инкапсуляции и полиморфизма, наследования и переопределения.

При выполнении практических работ студент должен **уметь**:

- разрабатывать и анализировать алгоритмы для решения поставленных задач;
- определять сложность алгоритмов;
- реализовывать типовые алгоритмы в виде программ на актуальных языках программирования;
- использовать средства проектирования для создания и графического отображения алгоритмов;
- оформлять код программ в соответствии со стандартом кодирования;
- выполнять проверку, отладку кода программы

Содержание практических занятий определено рабочей программой и тематическим планированием, соответствует теоретическому материалу изучаемых разделов учебной дисциплины.

Объем практических занятий по дисциплине определяется учебным планом по данной специальности.

Продолжительность практического занятия – 2 академических часа. Перед проведением практического занятия преподавателем организуется инструктаж, а по его окончании – обсуждение итогов. Комплект методических указаний по выполнению практических работ по дисциплине ОП.04 «Основы алгоритмизации и программирования» содержит 57 практических занятий.

Перечень практических работ

по дисциплине ОП.04 «Основы алгоритмизации и программирования»

ПРАКТИЧЕСКАЯ РАБОТА № 1

Тема: Знакомство со средой программирования

ПРАКТИЧЕСКАЯ РАБОТА № 2

Тема: Составление программ линейной структуры

ПРАКТИЧЕСКАЯ РАБОТА № 3

Тема: Составление программ разветвляющей структуры

ПРАКТИЧЕСКАЯ РАБОТА № 4

Тема: Составление программ разветвляющей структуры

ПРАКТИЧЕСКАЯ РАБОТА № 5

Тема: Составление программ циклической структуры

ПРАКТИЧЕСКАЯ РАБОТА № 6

Тема: Составление программ циклической структуры

ПРАКТИЧЕСКАЯ РАБОТА № 7

Тема: Обработка одномерных массивов

ПРАКТИЧЕСКАЯ РАБОТА № 8

Тема: Обработка одномерных массивов

ПРАКТИЧЕСКАЯ РАБОТА № 9

Тема: Обработка двумерных массивов

ПРАКТИЧЕСКАЯ РАБОТА № 10

Тема: Обработка двумерных массивов

ПРАКТИЧЕСКАЯ РАБОТА № 11

Тема: Работа со строками

ПРАКТИЧЕСКАЯ РАБОТА № 12

Тема: Работа с данными типа множество

ПРАКТИЧЕСКАЯ РАБОТА № 13

Тема: Составление программ с использованием текстовых файлов

ПРАКТИЧЕСКАЯ РАБОТА № 14

Тема: Создание программ с использованием типизированных и нетипизированных файлов

ПРАКТИЧЕСКАЯ РАБОТА № 15

Тема: Организация процедур

ПРАКТИЧЕСКАЯ РАБОТА № 16

Тема: Организация функций

ПРАКТИЧЕСКАЯ РАБОТА № 17

Тема: Применение рекурсивных функций

ПРАКТИЧЕСКАЯ РАБОТА № 18

Тема: Программирование модуля

ПРАКТИЧЕСКАЯ РАБОТА № 19

Тема: Использование указателей для организации связанных списков

ПРАКТИЧЕСКАЯ РАБОТА № 20

Тема: Изучение интегрированной среды разработчика

ПРАКТИЧЕСКАЯ РАБОТА № 21

Тема: Создание проекта с использованием компонентов для работы с текстом

ПРАКТИЧЕСКАЯ РАБОТА № 22

Тема: Создание проекта с использованием компонентов для работы с текстом

ПРАКТИЧЕСКАЯ РАБОТА № 23

Тема: Создание проекта с использованием компонентов для работы с текстом

ПРАКТИЧЕСКАЯ РАБОТА № 24

Тема: Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени

ПРАКТИЧЕСКАЯ РАБОТА № 25

Тема: Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени

ПРАКТИЧЕСКАЯ РАБОТА № 26

Тема: Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени

ПРАКТИЧЕСКАЯ РАБОТА № 27

Тема: Создание проекта с использованием полос прокрутки для ввода информации

ПРАКТИЧЕСКАЯ РАБОТА № 28

Тема: Создание проекта с использованием полос прокрутки для ввода информации

ПРАКТИЧЕСКАЯ РАБОТА № 29

Тема: Создание проекта с использованием полос прокрутки для ввода информации

ПРАКТИЧЕСКАЯ РАБОТА № 30

Тема: Создание проекта с использованием группы зависимых переключателей

ПРАКТИЧЕСКАЯ РАБОТА № 31

Тема: Создание проекта с использованием группы зависимых переключателей

ПРАКТИЧЕСКАЯ РАБОТА № 32

Тема: Создание проекта с использованием группы зависимых переключателей

ПРАКТИЧЕСКАЯ РАБОТА № 33

Тема: Создание процедур на основе событий

ПРАКТИЧЕСКАЯ РАБОТА № 34

Тема: Создание процедур на основе событий

ПРАКТИЧЕСКАЯ РАБОТА № 35

Тема: Создание проекта с использованием кнопочных компонентов

ПРАКТИЧЕСКАЯ РАБОТА № 36

Тема: Создание проекта с использованием кнопочных компонентов

ПРАКТИЧЕСКАЯ РАБОТА № 37

Тема: Создание проекта с использованием кнопочных компонентов

ПРАКТИЧЕСКАЯ РАБОТА № 38

Тема: Создание проекта с использованием компонентов стандартных диалогов и системы меню

ПРАКТИЧЕСКАЯ РАБОТА № 39

Тема: Создание проекта с использованием компонентов стандартных диалогов и системы меню

ПРАКТИЧЕСКАЯ РАБОТА № 40

Тема: Создание проекта с использованием компонентов стандартных диалогов и системы меню

ПРАКТИЧЕСКАЯ РАБОТА № 41

Тема: Разработка функциональной схемы работы приложения

ПРАКТИЧЕСКАЯ РАБОТА № 42

Тема: Разработка функциональной схемы работы приложения

ПРАКТИЧЕСКАЯ РАБОТА № 43

Тема: Разработка функциональной схемы работы приложения

ПРАКТИЧЕСКАЯ РАБОТА № 44

Тема: Разработка оконного приложения с несколькими формами

ПРАКТИЧЕСКАЯ РАБОТА № 45

Тема: Разработка оконного приложения с несколькими формами

ПРАКТИЧЕСКАЯ РАБОТА № 46

Тема: Разработка оконного приложения с несколькими формами

ПРАКТИЧЕСКАЯ РАБОТА № 47

Тема: Разработка игрового приложения

ПРАКТИЧЕСКАЯ РАБОТА № 48

Тема: Разработка игрового приложения

ПРАКТИЧЕСКАЯ РАБОТА № 49

Тема: Разработка игрового приложения

ПРАКТИЧЕСКАЯ РАБОТА № 50

Тема: Разработка многооконного приложения. Компиляция и запуск приложения

ПРАКТИЧЕСКАЯ РАБОТА № 51

Тема: Разработка многооконного приложения. Компиляция и запуск приложения

ПРАКТИЧЕСКАЯ РАБОТА № 52

Тема: Разработка многооконного приложения. Компиляция и запуск приложения

ПРАКТИЧЕСКАЯ РАБОТА № 53

Тема: Разработка интерфейса приложения

ПРАКТИЧЕСКАЯ РАБОТА № 54

Тема: Разработка интерфейса приложения

ПРАКТИЧЕСКАЯ РАБОТА № 55

Тема: Разработка интерфейса приложения

ПРАКТИЧЕСКАЯ РАБОТА № 56

Тема: Тестирование, отладка приложения

ПРАКТИЧЕСКАЯ РАБОТА № 57

Тема: Тестирование, отладка приложения

ИНСТРУКЦИИ ДЛЯ ОБУЧАЮЩИХСЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

Прежде чем приступить к выполнению заданий, внимательно прочитайте данные рекомендации. Практические работы включают в себя задания следующих видов.

1. Работа за компьютером

В ходе выполнения практических работ студент должен:

- выполнять требования по охране труда
- соблюдать инструкцию по правилам и мерам безопасности в кабинете информационных технологий
- строго выполнять весь объем работы, указанный в задании
- соблюдать требования эксплуатации компьютерной техники (правила включения и выключения)
- предоставить отчет о проделанной работе по окончании выполненной работы, который должен содержать:

1. Название работы.
2. Цель работы.
3. Задание и его решение.
4. Вывод о проделанной работе.

Текст отчета по практической работе должен быть набран на компьютере шрифтом Times New Roman размером 14 пт. (при оформлении текста используется текстовый редактор MS Word). Шрифт, используемый в иллюстративном материале (таблицы и рисунки), рекомендуется уменьшить до 12 пт. Межстрочный интервал в основном тексте - полуторный. В иллюстративном материале межстрочный интервал рекомендуется сделать одинарным. Поля страницы должны быть: левое поле - 30 мм; правое поле – 15 мм; верхнее и нижнее поле - 20 мм. Каждый абзац должен начинаться с красной строки. Отступ абзаца – 1,25 см от левой границы текста.

Студент должен выполнить практическую работу самостоятельно (или в группе, если это предусмотрено заданием). Практическая работа выполняется согласно заданию и методическим рекомендациям. После выполнения практической работы обучающийся самостоятельно себя контролирует путем ответов на вопросы. Результат работы представляется преподавателю в виде файла (файлов) в личном каталоге, защищается обучающимися.

По ходу выполнения работы при возникновении вопросов обучающийся может получить консультацию у преподавателя или самостоятельно воспользоваться лекционным материалом, рекомендуемой литературой.

1. Составление программы на языке программирования

Правила оформления кода:

1. Используйте разумные имена для переменных и функций

Программа должна быть хорошо понятна человеку при чтении. Если при чтении программы приходится понимать назначение переменных и функций по тому, как они используются, то читать код становится гораздо сложнее. Неудачно выбранные имена могут привести к тому, что смысл программы может быть неправильно понят. Выбирайте такие имена, которые бы объясняли смысл

переменных и функций, тогда код станет гораздо понятнее, и не потребуется писать множество комментариев.

При написании составных слов, например, в именах переменных, пишите их слитно без пробелов, при этом каждое новое слово пишется с большой буквы.

2. Не дублируйте код. Если в программе есть одинаковые выражения или фрагмента кода, вынесите этот код в отдельную функцию. Верным признаком необходимости создания новой функции является желание скопировать фрагмент кода из одного места программы в другое. В таком случае сразу перенесите этот фрагмент в отдельную функцию.

Если фрагменты кода похожи, но не идентичны, подумайте, не получится ли и их вынести в одну функцию, возможно добавив параметры или условия.

3. Не используйте «магические константы». Использование неименованных «магических» констант в коде нежелательно:

- при чтении кода может быть не понятно, что это за число, и почему оно именно такое;
- чаще всего одно и то же число потребуется написать в нескольких местах кода. Если его придётся изменять, можно пропустить одно из использований, что приведёт к ошибке.

Если в коде нужно использовать константу, дайте ей имя, используя `const`.

4. Расставляйте пробелы вокруг бинарных операторов. Это улучшает читаемость формул.

5. Всегда выделяйте блоки условных операторов и циклов скобками. В любой блок условия или цикла может захотеться добавить новое выражение. При этом можно забыть добавить скобки. Лучше сразу добавить скобки, чтобы потом не было с этим проблем.

6. Расставляйте скобки одинаково. Выберите и используйте для себя один из стилей расстановки скобок. Это улучшает читаемость структуры программы.

7. Не делайте строки слишком длинными. Строка программы должна помещаться на экране. Обычно рекомендуют ограничить максимальную ширину строки в 80 символов. Если определение или вызов функции получается слишком широким поместите по одному параметру на каждой строке. Длинные математические выражения разбивайте на несколько строк, разбивая по границам логических блоков выражения.

8. Объявляйте переменные непосредственно перед использованием. Обязательно указывайте начальное значение для переменных. Значение неинициализированной переменной может быть любым. Использование (чтение) такого значения приведёт к недетерминированной работе программы, а в некоторых случаях является неопределённым поведением. Чтобы избежать проблем всегда инициализируйте переменные прямо в момент их создания.

9. Единый стиль оформления кода во всем проекте;

10. Визуальное выделение наиболее значимых частей – используя *вертикальное форматирование*, мы выделяем объявление переменных, цикл заполнения массива случайными числами и цикл обработки по формуле. Если ваша функция выполняет несколько действий — то разумно разделить соответствующие блоки кода пустыми строками

ПРАКТИЧЕСКАЯ РАБОТА № 1

Тема: Знакомство со средой программирования

Цель работы: научиться устанавливать среду разработки программ Visual Studio 2022, познакомиться со средой Visual Studio 2022 и получить навыки работы с ней.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Visual Studio 2022 - интегрированная среда разработки (IDE) от Microsoft, предназначенная для создания приложений для Windows, Android, iOS, веба и облачных сервисов. В ней можно разрабатывать программы на языках C#, C++, JavaScript и Python, а также адаптировать проекты для различных платформ

Системные требования

Перед установкой Visual Studio 2022 убедитесь, что ваш компьютер соответствует минимальным системным требованиям:

- Операционная система: Windows 10, Windows 11
- Процессор: 1.8 ГГц или быстрее, 2 ядра или более
- Оперативная память: 4 ГБ (рекомендуется 8 ГБ и более)
- Место на диске: от 20 ГБ до 50 ГБ в зависимости от установленных компонентов
- Видеокарта: с поддержкой DirectX 11

Содержание работы:

Задание 1. Загрузить установщик программы

Для загрузки Visual Studio 2022 выполните следующие действия:

1. Перейдите на официальный сайт Microsoft Visual Studio: <https://visualstudio.microsoft.com/ru/downloads/>

Visual Studio 2022 | Windows

Наиболее полная интегрированная среда разработки для разработчиков .NET и C++ в Windows для создания веб-приложений, облачных, классических и мобильных приложений, а также служб и игр.

Предварительная версия

Получите ранний доступ к последним функциям, которые еще не доступны в основном выпуске

[Скачать](#) [Подробнее →](#)

Community	Professional	Enterprise
Мощная интегрированная среда разработки, бесплатная для студентов, участников проектов с открытым кодом и отдельных пользователей	Профессиональная интегрированная среда разработки, оптимально подходящая для небольших команд	Комплексное масштабируемое решение для команд любого размера
Скачать бесплатно	Бесплатная пробная версия	Бесплатная пробная версия

2. Выберите версию Community (бесплатная) и нажмите Скачать бесплатно.
3. Сохраните установочный файл на компьютер

Задание 2. Установить программу.

После загрузки установщика выполните следующие шаги:

1. Запустите скачанный файл (обычно называется vs_community.exe)
2. Примите условия лицензионного соглашения
3. На этапе выбора рабочих нагрузок отметьте нужные вам варианты.

Например:

- Разработка классических приложений .NET
- ASP.NET и разработка веб-приложений
- Разработка для Python

Можно выбрать и больше опций или вообще все опции, однако стоит учитывать свободный размер на жестком диске - чем больше опций будет выбрано, соответственно тем больше места на диске будет занято.

4. На вкладке "Отдельные компоненты" можно выбрать дополнительные инструменты, такие как Git, GitHub Extension, SQL Server Data Tools и другие

5. На вкладке "Языковые пакеты" выберите русский или другие нужные вам языки

6. Укажите место установки (по умолчанию C:\Program Files\Microsoft Visual Studio\2022\Edition)

7. Нажмите кнопку "Установить" и дождитесь завершения процесса.

8. И при инсталляции Visual Studio на компьютер будут установлены все необходимые инструменты для разработки программ, в том числе фреймворк .NET.

Задание 3. Настроить программу.

Первоначальная настройка:

После завершения установки Visual Studio 2022 предложит вам выполнить первоначальную настройку:

1. Войдите в свою учетную запись Microsoft (это необязательно, но рекомендуется для синхронизации настроек)
2. Выберите цветовую тему (темная, светлая, синяя или классическая)
3. Укажите предпочитаемые параметры среды разработки (например, "Общие", "Visual C++", "C#" и т.д.)
4. Настройте горячие клавиши в соответствии с вашими предпочтениями

Настройка параметров разработки

Для оптимальной работы с Visual Studio 2022 рекомендуется выполнить следующие настройки:

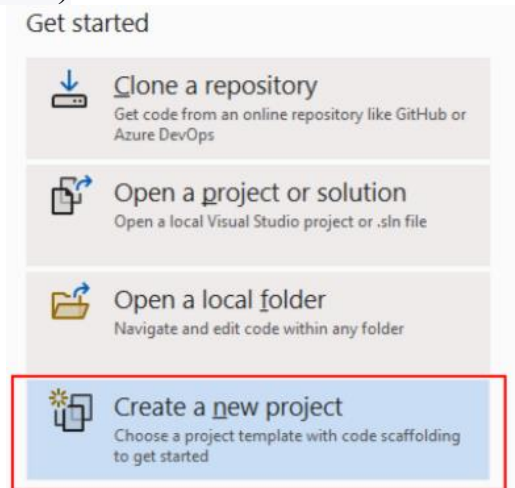
1. Настройка шрифтов и цветов. Перейдите в меню "Сервис" - "Параметры" - "Среда" - "Шрифты и цвета". Здесь вы можете изменить шрифт редактора, его размер и цветовую схему для различных элементов кода.

2. Настройка параметров текстового редактора. В разделе "Текстовый редактор" вы можете настроить такие параметры, как отступы, перенос строк, подсветка синтаксиса и многое другое для различных языков программирования.

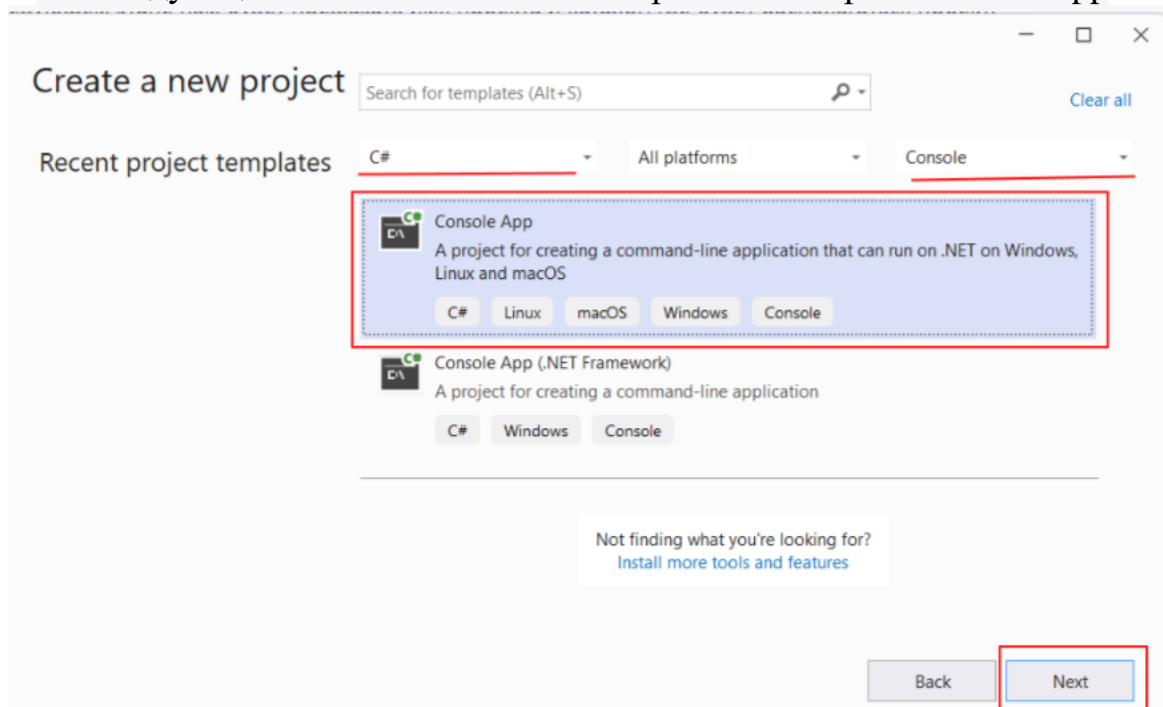
3. Настройка системы контроля версий. В разделе "История управления версиями" - "Git" вы можете настроить интеграцию с Git, указать путь к исполняемому файлу Git и настроить параметры фиксации.

Задание 4. Создать первый проект

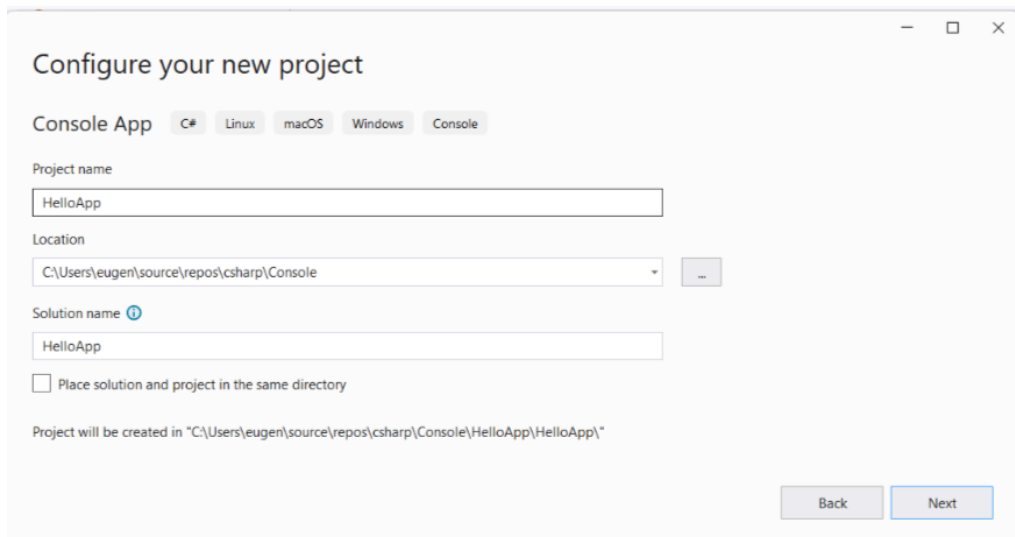
1. Откроем Visual Studio. На стартовом экране выбрать Create a new project (Создать новый проект)



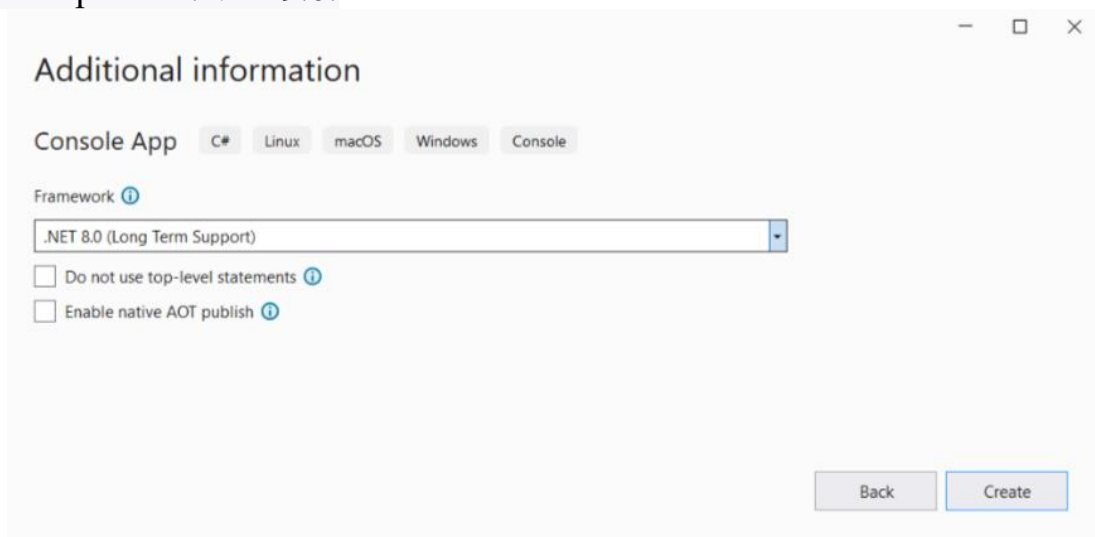
2. На следующем окне в качестве типа проекта выберем Console App



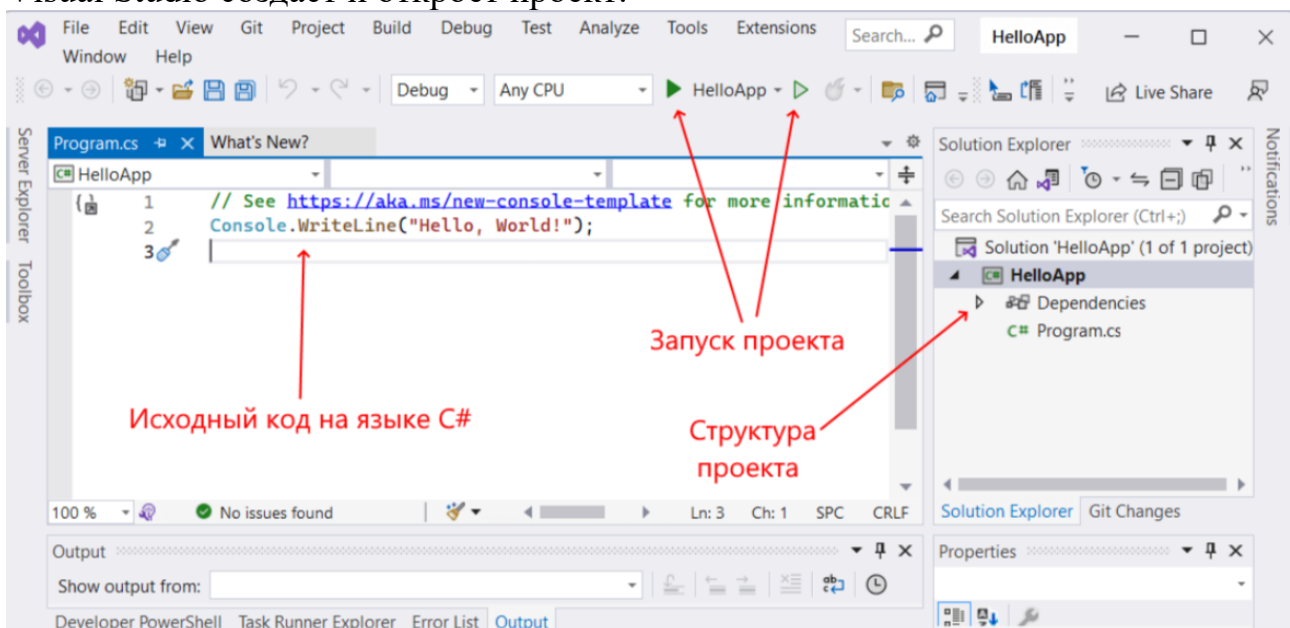
3. На следующем этапе будет предложено указать имя проекта и каталог, где будет располагаться проект, выбираем свою папку. В поле Project Name дадим проекту какое-либо название. В данном случае это HelloApp.



4. На следующем окне Visual Studio предложит выбрать версию .NET, которая будет использоваться для проекта. Выберем последнюю на данный момент версию - .NET 9.0:



5. Нажмем на кнопку Create (Создать) для создания проекта, и после этого Visual Studio создаст и откроет проект:



6. В большом поле в центре, которое по сути представляет текстовый редактор, находится сгенерированный по умолчанию код C#. Справа находится окно Solution Explorer, в котором можно увидеть структуру проекта. В данном случае у нас сгенерированная по умолчанию структура: узел **Dependencies** - это узел содержит сборки dll, которые добавлены в проект по умолчанию. Эти сборки как раз содержат классы библиотеки .NET, которые будет использовать C#. Однако не всегда все сборки нужны. Ненужные потом можно удалить, в то же время если понадобится добавить какую-нибудь нужную библиотеку, то именно в этот узел она будет добавляться.

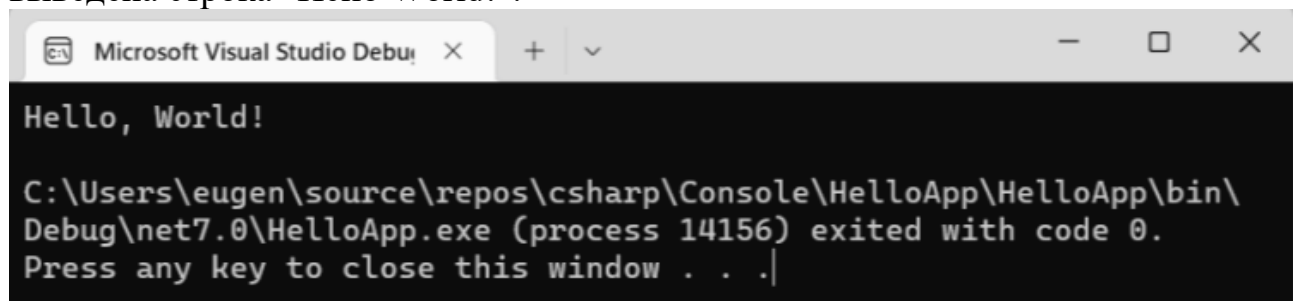
Далее идет непосредственно сам файл кода программы **Program.cs**, который по умолчанию открыт в центральном окне и который имеет всего две строки:

```
1 // See https://aka.ms/new-console-template for more information
2 Console.WriteLine("Hello, World!");
```

Первая строка предваряется символами // и представляет комментарии - пояснения к коду.

Вторая строка собственно представляет собой код программы: `Console.WriteLine("Hello World!");`. Эта строка выводит на консоль строку "Hello World!".

Несмотря на то, что программа содержит только одну строку кода, это уже некоторая программа, которую можем запустить. Запустить проект можно с помощью клавиши F5 или с панели инструментов, нажав на зеленую стрелку. И если вы все сделали правильно, то при запуске приложения на консоль будет выведена строка "Hello World!".



7. Теперь изменим весь этот код на следующий:

```
Console.Write("Введите свое имя: ");
var name = Console.ReadLine(); // вводим имя
Console.WriteLine($"Привет {name}"); // выводим имя на консоль
```

Первой строкой выводится приглашение к вводу: `Console.Write("Введите свое имя: ");`

Метод **Console.Write()** выводит на консоль некоторую строку. В данном случае это строка "Введите свое имя: ".

На второй строке определяется строковая переменная `name`, в которую пользователь вводит информацию с консоли: `var name = Console.ReadLine();`

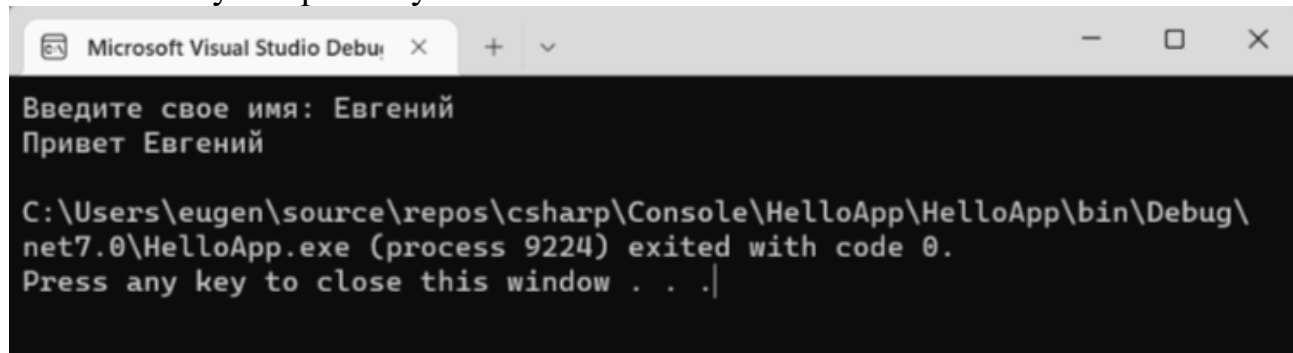
Ключевое слово **var** указывает на определение переменной. В данном случае переменная называется `name`. И ей присваивается результат метода **Console.ReadLine()**, который позволяет считать с консоли введенную

строку. То есть мы введем в консоли строку (точнее имя), и эта строка окажется в переменной `name`.

Затем введенное имя выводится на консоль: `Console.WriteLine($"Привет {name}");`

Чтобы ввести значение переменной `name` внутрь выводимой на консоль строки, применяются фигурные скобки `{}`. То есть при выводе строки на консоль выражение `{name}` будет заменяться на значение переменной `name` - введенное имя. Однако, чтобы можно было вводить таким образом значения переменных внутрь строки, перед строкой указывается знак доллара `$`.

Теперь протестируем проект, запустив его на выполнение, также нажав на F5 или зеленую стрелочку.



```
Microsoft Visual Studio Debug Console
Введите свое имя: Евгений
Привет Евгений

C:\Users\eugen\source\repos\csharp\Console\HelloApp\HelloApp\bin\Debug\net7.0\HelloApp.exe (process 9224) exited with code 0.
Press any key to close this window . . .
```

Скомпилированное приложение можно найти в папке проекта в каталоге **bin\Debug\net8.0**. Оно будет называться по имени проекта и иметь расширение `exe`. И затем этот файл можно будет запускать без Visual Studio, а также переносить его на другие компьютеры, где установлен .NET9.

ПРАКТИЧЕСКАЯ РАБОТА № 2

Тема: Составление программ линейной структуры

Цель работы: изучить основные принципы построения программ на языке программирования C#, изучить порядок действий при вычислении выражений; приобрести навыки в записи выражений и использовании стандартных функций; овладеть практическими навыками в программировании линейных алгоритмов и отладке программ.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

В тексте программы вначале перечисляются в виде списка все подключаемые директивы препроцессора, необходимые для корректной работы программы. В частности, директива `using System` разрешает в дальнейшем использование имен стандартных классов из пространства имен `System`. Далее в тексте программы представлена строка, начинающаяся с ключевого слова `namespace`, которое создает для проекта собственное пространство имен (в нашем случае – это `ConsoleApplication1`).

У нас используется только один метод с именем Main. Именно с этого метода должно начинаться выполнение программы.

Добавим в текст программы операции, демонстрирующие ввод-вывод информации на экран:

```
Console.WriteLine("*****");
Console.WriteLine("Ваштекст");
Console.WriteLine("*****");
Console.ReadLine();
```

Метод WriteLine предназначен для вывода информации (числа, строки) на консоль (экран, файл).

Помимо метода `WriteLine` в языке программирования `C#` допускается использование и метода `Write`. Например: `Console.Write(153);`

Допускается также использование форматированного вывода на консоль: `Console.WriteLine ("Сумма чисел {0} и {1} равна {2}",2,4,6);` В этом случае при выводе на экран мы получим: Сумма чисел 2 и 4 равна 6.

При использовании левого или правого выравниваний возможно задать также ширину вывода:

```
Console.WriteLine("\nЛевое выравнивание {0,-10} \\", 153);
Console.WriteLine("\nПравое выравнивание {0,10} \\", 153);
```

Последний используемый в программе метод `Console.ReadLine()` применен для задержки экрана. Когда в программе управление передается на этот метод, что система будет ждать нажатия любой клавиши на клавиатуре.

Объявление переменных:

```
int x; //объявление переменной x
x=500; //инициализация переменной x
long x, y=100; //объявление переменных x и y и инициализация y
long a= 5 *y; //объявление переменной с динамической инициализацией
```


string z= "Программирование на C#"; // объявление строковой переменной и ее инициализация

bool w=true; // объявление логической переменной с инициализацией

Неявно типизированная переменная объявляется с помощью ключевого слова var и должна быть обязательно инициализирована. Для определения типа этой переменной компилятору служит тип ее инициализатора, т.е. значения, которым она инициализируется:

var a= 123; // переменная a инициализируется целочисленным литералом

var b= 12.3; // переменная b инициализируется литералом с плавающей точкой, имеющему тип double

var f = 1.23F; // переменная f имеет тип float

Считывание данных с клавиатуры

Считывание данных с клавиатуры реализуется с помощью методов ReadLine, ReadKey и Read, объявленных в классе System.Console. Причем метод Read аналогичен методу ReadKey. Особенность метода ReadLine заключается в том, что он всегда возвращает строку (типа string).

string s;

s=Console.ReadLine();

Console.WriteLine("Ваш текст-"+s);

С помощью метода ReadKey можно считать с клавиатуры один символ.

Математические функции

В языке программирования C#представлено большое количество встроенных математических функций и константы (PI, E). Все функции и константы реализованы в классе Math пространства имен System:

- тригонометрические функции - Sin, Cos, Tan;
- обратные тригонометрические функции - ASin, ACos, ATan, ATan2 (sinx, cosx);
- экспоненту и логарифмические функции - Exp, Log, Log10;
- модуль, корень, знак - Abs, Sqrt, Sign;
- функции округления - Ceiling, Floor, Round;
- минимум, максимум, степень, остаток - Min, Max, Pow, IEEERemainder.

Содержание работы:

Задание 1. Написать программу, которая выводит на экран свою фамилию, имя, отчество и дату рождения.

using System;

namespace ConsoleApp1{

class Program {

static void Main(string[] args) {

Console.Write ("Введите имя: ");

string name = Console.ReadLine();

Console.Write ("Введите фамилию: ");

string fam = Console.ReadLine();

Console.Write ("Введите Отчество: ");


```

string otch = Console.ReadLine();
Console.Write ("Введите дату рождения: ");
string date = Console.ReadLine();
    string S = fam + " " + name + " " + otch + " " + date + " ";
    Console.WriteLine(S);
    Console.ReadLine();    }}}

```

Задание 2. Используя стандартные математические операции, вычислить сумму, разность, произведение двух чисел.

```

using System;
namespace ConsoleApp1{
    class Program    {
        static void Main(string[] args)    {
Console.Write ("Введите первое число ");
int a = Convert.ToInt32(Console.ReadLine());
Console.Write ("Введите второе число: ");
int b = Convert.ToInt32(Console.ReadLine());
int c = a + b;        Console.WriteLine ("сумма равна " + c);
int d = a-b;        Console.WriteLine ("разность равна " + d);
int k = a * b;        Console.WriteLine ("произведение равно "+ k);
        Console.ReadLine();    }}}

```

$$y = \sqrt{\frac{x+3}{x-3}}.$$

Задание 3. Написать программу для расчета функции

```

using System;
namespace ConsoleApp1{
    class Program    {
        static void Main(string[] args)    {
double x, y;
Console.WriteLine("Введите значение x: ");
x = Convert.ToDouble(Console.ReadLine());
y = Math.Sqrt((x+3)/(x-3));
Console.WriteLine("Результат: {0}", y);
        Console.ReadLine();    }}}

```

Задания для самостоятельного выполнения:

1. По известному радиусу круга определить его длину и площадь.
2. Даны стороны треугольника. Найдите периметр и площадь треугольника, используя формулу Герона
3. Даны стороны прямоугольника. Определить его периметр, площадь.
4. Даны длины сторон прямоугольного параллелепипеда. Найти его объем и площадь боковой поверхности.
5. Напишите программу, которая обрабатывает данные о двух человек. С клавиатуры вводятся ФИО, возраст и вес заданы в коде программы. На экран выводится строка, например, Иванов Иван Иванович, возраст 25, вес 78

ПРАКТИЧЕСКАЯ РАБОТА № 3

Тема: Составление программ разветвляющей структуры

Цель работы: овладение практическими вычислительного процесса разветвляющейся структуры на языке C#, получить опыт работы с условными операторами языка C#.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Условные операторы позволяют осуществить ветвление алгоритма и делают возможность выбрать для выполнения один из операторов.

Синтаксис оператора if можно представить следующим образом:

if (выражение_лог_типа) оператор; //сокращенная форма

if (выражение_лог_типа) оператор1; else оператор2; // полная форма

В выражении должен получаться результат, имеющий логический тип. Если результатом выражения является истинное значение (True), то выполняется оператор, следующий за условием в скобках. Если результатом выражения является значение false и присутствует ключевое слово else, то выполнятся оператор, следующий за ключевым словом else. Если ключевое слово else отсутствует, то никакой оператор не выполняется.

В предшествующем else операторе точка с запятой указывается. В общем случае ключевое слово else связывается с ближайшим ключевым словом if, которое еще не связано с ключевым словом else. Если вместо указанных операторов 1,2 требуется выполнить несколько операторов используются операторные скобки { }.

Содержание работы:

Задание 1. Напишите консольную программу, в которую пользователь вводит с клавиатуры два числа. А программа сравнивает два введенных числа и выводит на консоль результат сравнения (два числа равны, первое число больше второго или первое число меньше второго).

```
using System;
namespace _2{
    class Program    {
        static void Main(string[] args)    {
            Console.WriteLine("Введите первое число: ");
            int num1 = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Введите второе число: ");
            int num2 = Convert.ToInt32(Console.ReadLine());
            if(num1 > num2)    {
                Console.WriteLine("Первое число больше второго"); }
            else if (num1 < num2)    {
                Console.WriteLine("Первое число меньше второго"); }
            else    {
                Console.WriteLine("Оба числа равны"); }
            Console.ReadKey();    }    } }
```

Задание 2. В банке в зависимости от суммы вклада начисляемый процент по вкладу может отличаться. Напишите консольную программу, в которую пользователь вводит сумму вклада. Если сумма вклада меньше 100, то начисляется 5%. Если сумма вклада от 100 до 200, то начисляется 7%. Если сумма вклада больше 200, то начисляется 10%. В конце программа должна выводить сумму вклада с начисленными процентами.

Для получения вводимого с клавиатуры числа используйте выражение `Convert.ToDouble(Console.ReadLine())`

```
using System;
namespace _4 {
    class Program {
        static void Main(string[] args) {
            Console.WriteLine("Введите сумму вклада: ");
            double sum = Convert.ToDouble(Console.ReadLine());
            if (sum < 100) {
                sum += sum * 0.05; }
            else if (sum <= 200) {
                sum += sum * 0.07; }
            else {
                sum += sum * 0.1; }
            Console.WriteLine($"Сумму вклада после начисления процентов:
sum}");
            Console.ReadKey();    } } }
```

Задания для самостоятельного решения:

1. Написать программу для нахождения корней квадратного уравнения.
2. Написать программу для проверки является ли заданное число четным или нечетным.
3. Написать программу, которая проверяет существует ли треугольник со сторонами a, b, c,
4. Написать программу для проверки является ли год високосным.
5. Написать программу, которая вычисляет значение функции y:

$$y = \begin{cases} 4, & x > 20 \\ x^2 + x - 1, & x < 20 \end{cases}$$

ПРАКТИЧЕСКАЯ РАБОТА № 4

Тема: Составление программ разветвляющей структуры

Цель работы: овладение практическими вычислительного процесса разветвляющейся структуры на языке C#, получить опыт работы с условными операторами языка C#.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Оператор варианта (switch)

Оператор варианта switch состоит из выражения (переключателя) и списка операторов, каждому из которых предшествует одна или более констант (они называются константами выбора) или ключевое слово default. Все константы выбора предваряются ключевым словом case, должны быть уникальными и иметь тип, совместимый с типом переключателя.

Управление передается оператору case, совпадающему со значением оператора switch. Оператор switch может включать любое количество экземпляров case, но два оператора case не могут иметь одинаковое значение. Выполнение текста оператора начинается с выбранного оператора и продолжается до тех пор, пока оператор break не передаст управление за пределы текста case. Оператор перехода, такой как break, требуется после каждого блока case, включая последний блок, вне зависимости от того, какой из двух операторов — case или default — там использован.

Содержание работы:

Задание 1. Выдать введенное число в словесной интерпретации using System;

```
namespace _2{
    class Program {
        static void Main(string[] args) {
            int A; string s;
            Console.WriteLine("Введите A=");
            s = Console.ReadLine();
            A = Convert.ToInt32(s);
            switch (A) {
                case 1: Console.WriteLine("Один"); break;
                case 2: Console.WriteLine("Два"); break;
                case 3: Console.WriteLine("Три"); break;
                case 4: Console.WriteLine("Четыре"); break;
                default: Console.WriteLine("Остальные числа"); break;
            }
        }
    }
}
```

Задание 2. Напишите консольную программу, которая выводит пользователю сообщение "Введите номер операции: 1.Сложение 2.Вычитание 3.Умножение". Рядом с названием каждой операции указан ее номер, например, операция вычитания имеет номер 2. Пусть пользователь вводит в программу номер операции, в зависимости от номера операции программа выводит ему

название операции и с введенными числами выполняются определенные действия (например, при вводе числа 3 числа умножаются).

Для определения операции по введенному номеру используйте конструкцию switch...case.

```
using System;
```

```
namespace _3{
```

```
class Program {
```

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

```
    Console.WriteLine("Введите первое число: ");
```

```
    int num1 = Convert.ToInt32(Console.ReadLine());
```

```
    Console.WriteLine("Введите второе число: ");
```

```
    int num2 = Convert.ToInt32(Console.ReadLine());
```

```
        Console.WriteLine("Введите номер операции:  
1.Сложение 2.Вычитание 3.Умножение");
```

```
    int operation = Convert.ToInt32(Console.ReadLine());
```

```
    int result = 0;
```

```
    switch (operation) {
```

```
        case 1:
```

```
            result = num1 + num2;
```

```
            Console.WriteLine($"Результат операции {result}");
```

```
            break;
```

```
        case 2:
```

```
            result = num1 - num2;
```

```
            Console.WriteLine($"Результат операции {result}");
```

```
            break;
```

```
        case 3:
```

```
            result = num1 * num2;
```

```
            Console.WriteLine($"Результат операции {result}");
```

```
            break;
```

```
        default:
```

```
            Console.WriteLine("Неизвестная операция");
```

```
            break;    }
```

```
    Console.ReadKey();    } }
```

Задания для самостоятельного выполнения:

1. Составить программу, которая в зависимости от порядкового номера дня недели (1, 2, ..., 7) выводит на экран его название (понедельник, вторник, ..., воскресенье).

2. Составить программу, которая в зависимости от порядкового номера месяца (1, 2, ..., 12) выводит на экран время года, к которому относится этот месяц.

3. Мастям игральных карт условно присвоены следующие порядковые номера: масти «пики» — 1, масти «трефы» — 2, масти «бубны» — 3, масти «червы» — 4. По заданному номеру масти m ($1 \leq m \leq 4$) определить название соответствующей масти

4. Составить программу, которая в зависимости от порядкового номера дня месяца (1, 2, ..., 12) выводит на экран его название (январь, февраль, ..., декабрь).
5. Составить программу, которая в зависимости от номера пальца на руке выводит на экран его название.

ПРАКТИЧЕСКАЯ РАБОТА № 5

Тема: Составление программ циклической структуры

Цель работы: научиться создавать, выполнять и исправлять простейшие программы с циклическим оператором `for`.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Цикл `for` – цикл со счетчиком. Цикл `for` имеет следующее формальное определение: `for ([действия_до_выполнения_цикла]; [условие]; [действия_после_выполнения])`

Объявление цикла **for** состоит из трех частей. Первая часть объявления цикла - некоторые действия, которые выполняются один раз до выполнения цикла. Обычно здесь определяются переменные, которые будут использоваться в цикле.

Вторая часть - условие, при котором будет выполняться цикл. Пока условие равно `true`, будет выполняться цикл.

И третья часть - некоторые действия, которые выполняются после завершения блока цикла. Эти действия выполняются каждый раз при завершении блока цикла.

После объявления цикла в фигурных скобках помещаются сами действия цикла.

Содержание работы:

Задание 1. Найти сумму первых `N` членов арифметической прогрессии с использованием цикла **for**.

```
using System;
```

```
class Program{
```

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

```
        Console.WriteLine("n = ");
```

```
        int n = Convert.ToInt32(Console.ReadLine());
```

```
        int sum = 0;
```

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

```
            sum += i; }
```

```
        Console.WriteLine("Сумма первых {0} членов арифметической прогрессии  
равна {1}", n, sum);
```

```
        Console.ReadLine(); } }
```

Задание 2. Найти среднее арифметическое всех целых чисел от `a` до `b` (значения `a` и `b` вводятся с клавиатуры)

```
using System;
```

```
class Program {
```

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

```
        double a, b;
```

```
        double sum=0;
```

```
        double result = 0;
```

```
        Console.Write("Введите a: ");
```

```

a = Convert.ToDouble(Console.ReadLine());
Console.Write("Введите b: ");
b = Convert.ToDouble(Console.ReadLine());
for (double i = a; i <= b; i++)
    { sum += i;
result += i / ((b-a)+1);      }
Console.WriteLine("Сумма чисел от {0} до {1} равна {2}, среднее
арифметическое равно {3}: ",a,b,sum,result );
Console.ReadLine();      }      }

```

Задание 3. За каждый месяц банк начисляет к сумме вклада 10% от суммы. Напишите консольную программу, в которую пользователь вводит сумму вклада и количество месяцев. А банк вычисляет конечную сумму вклада с учетом начисления процентов за каждый месяц.

Для вычисления суммы с учетом процентов используйте цикл for. Для ввода суммы вклада используйте выражение `Convert.ToDecimal(Console.ReadLine())` (сумма вклада будет представлять тип decimal).

```

using System;
namespace _1{
class Program {
static void Main(string[] args) {
Console.WriteLine("Введите сумму вклада: ");
decimal sum = Convert.ToDecimal(Console.ReadLine());
Console.WriteLine("Введите срок вклада в месяцах: ");
int period = Convert.ToInt32(Console.ReadLine());
for(int i = 1; i <= period; i++) {
sum += sum * 0.1M;      }
Console.WriteLine($"После {period} месяцев сумма вклада составит
{sum}");
Console.ReadKey();      }      }}

```

Задания для самостоятельного выполнения:

Написать программы вывода на экран следующей информации с использованием цикла for:

1. Напишите программу, которая выводит на консоль таблицу умножения
2. Выведите на экран, все четные числа от 35 до 77
3. Написать программу, которая выводит таблицу значений функции $y = -2.4x^2 + 5x - 3$ в диапазоне от -2 до + 2 с шагом 0,5.

ПРАКТИЧЕСКАЯ РАБОТА № 6

Тема: Составление программ циклической структуры

Цель работы: научиться создавать, выполнять и исправлять простейшие программы с циклическим оператором `while` и `do while`.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Оператор цикла `while` предназначен для организации циклического процесса, в котором выполнение каждой следующей итерации определяется на основе истинности некоторого условия. Оператор цикла `while` еще называют оператором цикла с предусловием.

Циклы `while` бывают двух видов – собственно цикл `while` и `do-while`.
`while` (условие_продолжения) оператор;
`do` оператор; `while` (условие продолжения);
здесь

- условие – некоторое условие согласно синтаксису языка C#. Инструкция оператор выполняется до тех пор, пока значение условие = `true`. Как только значение условие становится равным `false`, то циклический процесс прекращается и выполняются следующие после `while` операторы;
- оператор – один или несколько операторов. Если в цикле `while` нужно выполнить несколько операторов одновременно, то эти операторы берутся в фигурные скобки `{ }`.

Тело цикла (оператор) выполняется, пока значение условие есть истинным (`true`). Оператор цикла должен быть организован таким образом, чтобы в конечном счете значение условия стало равно `false`. Иначе, программа «зависнет», так как выйдет бесконечный цикл.

Оба эти цикла используются, как правило, тогда, когда точно неизвестно, сколько раз цикл должен выполниться. Например, при вводе пользователем пароля или при подсчете чего-либо с определенной точностью. Оба эти цикла будут выполняться до тех пор, пока условие в круглых скобках после слова `while` будет истинно. Как только условие станет равным `false`, выполнение цикла прекращается. Самое важное отличие между `while` и `do-while` в том, что `while` может не выполниться ни одного раза, тогда как `do-while` по крайней мере один раз выполнится.

Содержание работы:

Задание 1. Получить таблицу температур по Цельсию t_c от -50 до +50 градусов а также их эквивалентов по шкале Фаренгейта t_f , используя

$$t_f = \frac{9}{5}t_c + 32$$

соотношение

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
namespace ConsoleApplication3 {
```

```
class Program {
static void Main(string[] args) {
    // шкала температур Цельсий => Фаренгейт
    int tc; // текущее значение температуры по Цельсию
    double tf; // значение температуры по Фаренгейту
    tc = -50;
    Console.WriteLine("Шкала температур: Цельсий - Фаренгейт");
    while (tc <= 50) {
        tf = 9 / 5 * tc + 32;
        tc++;
        Console.WriteLine("{0} C => {1} F", tc, tf);    } } }
```

Задания для самостоятельного выполнения:

Написать программы вывода на экран следующей информации с использованием цикла while:

1. Напишем программу, которая выводит в консоль числа от 0 до 10
2. Найти сумму всех целых чисел от n_{\min} до n_{\max}
3. Выведите последовательность 3 5 7 9 ... 21 (от 3 до 21 с шагом = 2).

ПРАКТИЧЕСКАЯ РАБОТА № 7

Тема: Обработка одномерных массивов

Цель работы: изучить принципы работы с одномерными массивами, научить писать программы с одномерными массивами

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Массив – множество однотипных элементов. Любой массив является производным от класса `System.Array`. В отличие от других языков программирования, при объявлении массива в C# нельзя указать его размер, т.к. при объявлении не создается сам массив, а только ссылка на будущий массив. Поэтому после объявления необходима инициализация массива.

Объявление одномерного массива выглядит следующим образом:

`<тип>[] <имя массива>;`

Квадратные скобки приписаны не к имени переменной, а к типу. Они являются неотъемлемой частью определения класса, так что запись `T[]` следует понимать как класс одномерный массив с элементами типа `T`. В данном случае речь идет об отложенной инициализации. При объявлении с отложенной инициализацией сам массив не создается, а создается только ссылка на массив, имеющая неопределенное значение `Null`. Поэтому пока массив не будет реально создан и его элементы инициализированы, использовать его в вычислениях нельзя.

`int[] a, b, c; // пример объявления трех массивов с отложенной инициализацией`

Чаще всего, при объявлении массива используется имя с инициализацией. В случае простых переменных, могут быть два варианта инициализации:

а) инициализация является явной и задается константным массивом:

`double[] x= {5.5, 6.6, 7.7};`

Синтаксически, элементы константного массива следует заключать в фигурные скобки.

б) массив создается и инициализируется (выделяется место в памяти с указанным числом элементов массива) массив из 5 элементов типа `int`: `int[] d= new int[5];`

При создании массива можно указывать число элементов, которое хранит переменная, инициализируемая в результате работы программы.

Содержание работы:

Задание 1. Ввести размер массива и его элементы с клавиатуры. Найти сумму всех элементов.

`using System;`

`class Program {`

`static void Main() {`

`Console.Write("Введите размер массива: ");`

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

`int[] array = new int[n];`

`Console.WriteLine("Введите элементы массива:");`

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

```

        array[i] = int.Parse(Console.ReadLine());    }
        int sum = 0;
        foreach (int num in array)    {
            sum += num;    }
        Console.WriteLine("Сумма элементов: " + sum);    } }

```

Задание 2. Создать массив из 10 целых чисел. Заполнить его случайными числами от 1 до 100. Найти сумму всех элементов.

```

using System;
class Program {
    static void Main()    {
        Random rnd = new Random();
        int[] array = new int[10];
        int sum = 0;
        for(int i = 0; i < array.Length; i++) {
            array[i] = rnd.Next(1, 101);
            sum += array[i]; }
        Console.WriteLine($"Сумма элементов: {sum}");    } }

```

Задание 3. Ввести размер массива и его элементы. Найти максимальный элемент.

```

using System;
class Program {
    static void Main()    {
        Console.Write("Введите размер массива: ");
        int n = int.Parse(Console.ReadLine());
        int[] array = new int[n];
        Console.WriteLine("Введите элементы массива:");
        for (int i = 0; i < n; i++)    {
            array[i] = int.Parse(Console.ReadLine());    }
        int max = array[0];
        for (int i = 1; i < n; i++)    {
            if (array[i] > max)
                max = array[i];    }
        Console.WriteLine("Максимальный элемент: " + max);    } }

```

Задания для самостоятельного выполнения:

1. Ввести размер массива и его элементы. Найти максимальный элемент.
2. Требуется найти сумму и произведение N элементов массива, используя цикл while.
3. В массив записаны заработные платы работников некоторого предприятия. Определить количество работников, заработная плата которых ниже средней заработной платы по предприятию, и вывести на экран номера элементов массива, которые соответствуют таким работникам.

ПРАКТИЧЕСКАЯ РАБОТА № 8

Тема: Обработка одномерных массивов

Цель работы: изучить принципы работы с одномерными массивами, научить писать программы с одномерными массивами

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Содержание работы:

Задание 1. Ввести размер массива и его элементы. Посчитать количество четных чисел.

```
using System;
class Program {
    static void Main()    {
        Console.WriteLine("Введите размер массива: ");
        int n = int.Parse(Console.ReadLine());
        int[] array = new int[n];
        Console.WriteLine("Введите элементы массива:");
        for (int i = 0; i < n; i++)    {
            array[i] = int.Parse(Console.ReadLine());    }
        int count = 0;
        foreach (int num in array)    {
            if (num % 2 == 0)
                count++;    }
        Console.WriteLine("Количество четных чисел: " + count);    } }
```

Задание 2. Дан массив, состоящий из 15 элементов целого типа. Получить новый массив, как разность между элементами исходного массива и его среднего арифметического.

```
using System;
namespace Tasks2{
class Program{
static void Main(string[] args){
    int[] mass = new int[15]; //создаем массив
    Random rnd = new Random();
    int summ = 0; //переменная для суммы всех элементов массива
    string s=null; //строка для вывода массива на экран
    for (int i = 0; i < 15; i++)    {
        mass[i] = rnd.Next(1, 100); //задаем рандомное значение элементу
        массива, можно самому вручную, разницы нет
        s = s + mass[i].ToString() + " "; //дополняем строку
        summ = summ + mass[i]; } //суммируем элементы массива
    Console.WriteLine("Начальный массив = " + s); //выводим элементы
    массива
    Console.WriteLine("Среднее арифметическое = " + (summ/15));
    s = null;
    for (int i = 0; i < 15; i++)    {
```

```
        mass[i] = mass[i] - (summ / 15); //теперь вычитаем из элементов массива  
среднее арифметическое  
        s = s + mass[i].ToString() + " ";    }  
        Console.WriteLine("Новый массив = " + s);  
        Console.ReadKey();}}}
```

Задания для самостоятельного решения:

1. Ввести размер массива и его элементы. Посчитать количество нечетных чисел.
2. Написать программу, которая определяет, состоят ли два массива из одинаковых элементов.
3. Написать программу, которая определяет, сколько элементов с равными значениями содержится в массиве. Например:
 Задайте количество элементов массива: 7
 Введите элементы массива: 3 6 -1 3 7 3 7
 В данном массиве два элемента равны 7, три элемента равны 3

ПРАКТИЧЕСКАЯ РАБОТА № 9

Тема: Обработка двумерных массивов

Цель работы: изучить принципы работы с двумерными массивами, научить писать программы с одномерными массивами

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Объявление многомерного массива в общем случае:

`<тип>[, ... ,] <имя массива>;`

Число запятых, увеличенное на единицу, и задает размерность массива. Можно лишь отметить, что хотя явная инициализация с использованием многомерных константных массивов возможна, но применяется редко из-за громоздкости такой структуры.

Простейший многомерный массив — двумерный. В двумерном массиве позиция любого элемента определяется двумя индексами. Если представить двумерный массив в виде таблицы данных, то один индекс означает строку, а второй — столбец.

Чтобы объявить двумерный массив целочисленных значений размером 10x20 с именем `table`, достаточно записать следующее:

```
int[,] table = new int[10, 20];
```

Обратите особое внимание на то, что значения размерностей отделяются запятой. Синтаксис первой части этого объявления означает, что создается ссылочная переменная двумерного массива. Для реального выделения памяти для этого массива с помощью оператора `new` используется более конкретный синтаксис: `int[10, 20]`

Чтобы получить доступ к элементу двумерного массива, необходимо указать оба индекса, разделив их запятой. Например, чтобы присвоить число 10 элементу массива `table`, позиция которого определяется координатами 3 и 5, можно использовать следующую инструкцию: `table [3 , 5] = 10;`

Содержание работы:

Задание 1. Создать программу, которая создает двумерный массив 3x3, заполняет его случайными числами от 1 до 100 и выводит на консоль.

```
using System;
```

```
class Program {  
    static void Main()    {  
        Random rnd = new Random();  
        int[,] array = new int[3, 3];  
        // Заполнение массива  
        for (int i = 0; i < 3; i++)    {  
            for (int j = 0; j < 3; j++)    {  
                array[i, j] = rnd.Next(1, 101);    }    }  
        // Вывод массива  
        for (int i = 0; i < 3; i++)    {  
            for (int j = 0; j < 3; j++)    {  
                Console.Write($"{array[i, j]}t");    }  
        }  
    }  
}
```

```
Console.WriteLine();    }    }
```

Задание 2. Найти максимальный элемент в двумерном массиве и его позицию.

```
using System;
class Program {
    static void Main() {
        int[,] array = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
        int max = array[0, 0];
        int maxI = 0, maxJ = 0;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (array[i, j] > max) {
                    max = array[i, j];
                    maxI = i;
                    maxJ = j;
                }
            }
        }
        Console.WriteLine($"Максимальный элемент: {max} (позиция [{maxI},{maxJ}]);    }    }
```

Задания для самостоятельного решения:

1. Найдите сумму и произведение элементов квадратной матрицы размерности 10x10. Размерность и элементы массива задаются пользователем.
2. Задана квадратная матрица целых чисел. Подсчитайте количество отрицательных и положительных элементов, а также выведите на печать координаты нулевых элементов (номер строки и номер столбца).

ПРАКТИЧЕСКАЯ РАБОТА № 10

Тема: Обработка двумерных массивов

Цель работы: изучить принципы работы с двумерными массивами, научить писать программы с одномерными массивами

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Содержание работы:

Задание 1. Создать программу, которая транспонирует квадратную матрицу.

```
using System;
class Program {
    static void Main() {
        int[,] matrix = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
        int[,] transposed = new int[3, 3];
        // Транспонирование
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                transposed[j, i] = matrix[i, j];
            }
        }
        // Вывод результата
        Console.WriteLine("Исходная матрица:");
        PrintMatrix(matrix);
        Console.WriteLine("Транспонированная матрица:");
        PrintMatrix(transposed);
    }
    static void PrintMatrix(int[,] matrix) {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                Console.Write($"{matrix[i, j]}\t");
            }
            Console.WriteLine();
        }
    }
}
```

Задание 2. Найти сумму элементов главной и побочной диагоналей квадратной матрицы.

```
using System;
class Program {
    static void Main() {
        int[,] matrix = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
        int mainDiagSum = 0;
        int sideDiagSum = 0;
        for (int i = 0; i < 3; i++) {
            mainDiagSum += matrix[i, i];
            sideDiagSum += matrix[i, 2 - i];
        }
        Console.WriteLine($"Сумма главной диагонали: {mainDiagSum}");
        Console.WriteLine($"Сумма побочной диагонали: {sideDiagSum}");
    }
}
```

Задания для самостоятельного выполнения:

1. В двумерном массиве переставьте попарно соседние строки, т.е. 1-ю со 2-ой, 3-ю с 4-й и т.д. Результат выведите на экран.
2. Дан целочисленный двумерный массив, размерности $n \times m$, найти сумму всех положительных элементов массива.
3. Дан целочисленный двумерный массив, размерности $n \times m$, найти наибольший элемент массива и номер строки, в которой он находится.

ПРАКТИЧЕСКАЯ РАБОТА № 11

Тема: Работа со строками

Цель работы: приобретение навыков алгоритмизации и программирования задач, оперирующих строковыми типами данных; научиться обрабатывать строковые данные.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Строка – это последовательная коллекция символов, используемая для представления текста.

За представление строк в C# отвечает класс `System.String`. В коде, для объявления переменной соответствующего типа, предпочтительно использовать следующий вариант написания: `string` – с маленькой буквы. Это ключевое слово языка, используя которое можно объявлять строковые переменные, также как `int` является псевдонимом для `System.Int32`, а `bool` – для `System.Boolean`.

Класс `string` обеспечивает множество встроенных методов для сравнения, поиска и управления строковыми значениями.

`Empty` — свойство, определяющее, пустая ли строка;

`Compare()` — функция сравнения двух строк;

`Concat()` — создает новую строку из двух и более исходных строк;

`CopyO` — создает дубликат исходной строки;

`Equals()` — определяет, содержат ли две строки одинаковые значения;

`Join()` — добавляет новую строку в любое место уже существующей строки;

`Length` — количество символов в строке;

`CompareToO` — сравнивает одну строку с другой;

`CopyToQ` — копирует определенное число символов строки в массив `Unicode` символов;

`EndsWithQ` — определяет, заканчивается ли строка определенной последовательностью символов;

`EqualsQ` — определяет, имеют ли две строки одинаковые значения;

`InsertO` — вставляет новую строку в уже существующую;

`LastIndexOfQ` — возвращает индекс последнего вхождения элемента в строку;

`Remove()` — удаляет необходимое число символов из строки;

`Split()` — возвращает подстроку, отделенную от основного массива определенным символом;

`StartsWith()` — определяет, начинается ли строка с определенной последовательности символов;

`Substring()` — возвращает подстроку из общего массива символов;

`ToLower()` — преобразует строку к нижнему регистру;

`ToUpper()` — преобразует строку к верхнему регистру;

`TrimQ` — удаляет все вхождения определенных символов в начале и в конце строки;

`TrimEndO` — удаляет все вхождения определенных символов в конце строки;

`TrimStartf)` — удаляет все вхождения определенных символов в начале строки.

Содержание работы:

Задание 1. Написать программу, для определения – является ли введенное слово палиндромом

// Строка - палиндром пример: "чем нежен меч"

```
using System;
namespace палиндром {
    class Program {
        static void Main(string[] args) {
            Console.Write("Введите слово: ");
            string s = Console.ReadLine();
            if (ItIsPalindrom(s))
                Console.Write("это - палиндром ");
            else
                Console.Write("не палиндром ");
            Console.ReadKey(); }
        // Проверка. Возврат ДА/НЕТ
        static bool ItIsPalindrom(string s) {
            bool b = true;
            for (int k = 0; k < s.Length / 2; k++) {
                if (s[k] != s[s.Length - 1 - k])
                    b = false; }
            return b; } } }
```

Задание 2. Написать программу, которая определяет совпадение подстроки в начале и в конце строки

```
using System;
namespace строки{
    class Program {
        static void Main(string[] args) {
            Console.WriteLine("Введите строку текста");
            string s1 = Console.ReadLine();
            Console.WriteLine("Введите образец совпадения");
            string s2 = Console.ReadLine();
            if (s1.StartsWith(s2)) Console.WriteLine("Найдено совпадение подстроки в начале строки");
            if (s1.EndsWith(s2)) Console.WriteLine("Найдено совпадение подстроки в конце строки");
            else Console.WriteLine("Совпадений не найдено");
            Console.ReadKey(); } } }
```

Задание 3. Написать программу для вставки одной подстроки в другую строку. Строка и подстрока вводятся с клавиатуры.

```
using System;
```

```
namespace строки{
```

```
class Program {
```

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

```
        Console.Write("Введите первую строку текста: ");
```

```
        string s1 = Console.ReadLine();
```

```
        Console.Write("Введите вторую строку текста: ");
```

```
        string s2 = Console.ReadLine();
```

```
        string s = s1.Insert(6, s2);
```

```
        Console.WriteLine(s);
```

```
        Console.ReadKey();    }    }    }
```

Задания для самостоятельного решения:

1. Написать программу, объединяющую две введенные с клавиатуры строки текста
2. Написать программу, которая сравнивает две введенные с клавиатуры строки текста
3. Написать программу, в которой производится замена символов. Все данные вводятся с клавиатуры

ПРАКТИЧЕСКАЯ РАБОТА № 12

Тема: Работа с данными типа множество

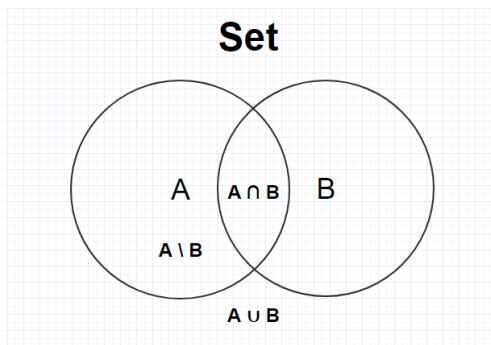
Цель работы: получение практических навыков работы с правилами объявления и использования переменных типа множество на языке программирования C#; а также с операциями и процедурами при работе с ними.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Множество (Set) - это совокупность однотипных объектов, рассматриваемая как одно целое. Коллекция, содержащая только отличающиеся элементы, называется *множеством*. Для работы с множествами в библиотеке классов .NET Framework имеется целых два обобщённых класса **HashSet** и **SortedSet**, которые находятся в пространстве имён *System.Collections.Generic*. Различие между ними в том, что **SortedSet** представляет упорядоченное множество.

Операции над множествами



Add - добавление элемента. Если такой элемент уже присутствует, то он не будет добавлен.

Remove - удаление элемента из множества.

Union - объединение множеств. Создается новое множество, включающее в себя все элементы из множества A и множества B. Если элемент содержится в обоих множествах, он будет добавлен однократно.

Разность множеств

С помощью метода **Except** можно получить разность двух множеств:

```
string[] soft = { "Microsoft", "Google", "Apple"};  
string[] hard = { "Apple", "IBM", "Samsung"};  
// разность множеств  
var result = soft.Except(hard);  
foreach (string s in result)  
    Console.WriteLine(s);
```

В данном случае из массива *soft* убираются все элементы, которые есть в массиве *hard*. Результатом операции будут два элемента: Microsoft и Google

Пересечение множеств

Для получения пересечения множеств, то есть общих для обоих наборов элементов, применяется метод:

```
string[] soft = { "Microsoft", "Google", "Apple"};
string[] hard = { "Apple", "IBM", "Samsung"};
// пересечение множеств
var result = soft.Intersect(hard);
foreach (string s in result)
    Console.WriteLine(s);
```

Так как оба набора имеют только один общий элемент, то соответственно только он и попадет в результирующую выборку: Apple

Объединение множеств

Для объединения двух множеств используется метод **Union**. Его результатом является новый набор, в котором имеются элементы, как из одного, так и из второго множества. Повторяющиеся элементы добавляются в результат только один раз:

```
string[] soft = { "Microsoft", "Google", "Apple"};
string[] hard = { "Apple", "IBM", "Samsung"};
// объединение множеств
var result = soft.Union(hard);
foreach (string s in result)
    Console.WriteLine(s);
```

Содержание работы:

Задание 1. Сформировать множество, содержащее 5 целых чисел из диапазона 1..10 и вывести это множество на экран.

```
using System;
using System.Collections.Generic;
using System.Linq;
class Program {
    static void Main() {
        // Создаем множество (HashSet)
        HashSet<int> numbers = new HashSet<int>();
        // Заполняем множество случайными числами
        Random random = new Random();
        while (numbers.Count < 5) {
            numbers.Add(random.Next(1, 11)); } // 11 - потому что верхний
предел не включается
        // Выводим результат
        Console.WriteLine("Сформированное множество:");
        foreach (int number in numbers) {
            Console.Write(number + " "); } }
```

Объяснение работы программы:

1. Используется HashSet<int> для хранения уникальных чисел
2. Цикл продолжается, пока в множестве не будет 5 элементов
3. random.Next(1, 11) генерирует числа от 1 до 10 включительно
4. Метод Add() автоматически проверяет уникальность элементов
5. Результат выводится в консоль

При каждом запуске программы будет создаваться новое множество с разными числами.

Задание 2. Дана строка. Содержит ли она и строчные символы?

```
using System;
class Program {
    static void Main() {
        // Получаем строку от пользователя
        Console.Write("Введите строку: ");
        string input = Console.ReadLine();
        // Проверяем наличие строчных символов
        bool hasLowercase = false;
        foreach (char c in input) {
            if (char.IsLower(c)) {
                hasLowercase = true;
                break;
            }
        }
        // Выводим результат
        if (hasLowercase) {
            Console.WriteLine("Строка содержит строчные символы");
        }
        else {
            Console.WriteLine("Строка не содержит строчных символов");
        }
    }
}
```

Задание 3. Дана строка символов. Требуется построить и вывести множество, элементами которого являются гласные буквы, которые входят в текст

```
using System;
using System.Collections.Generic;
using System.Linq;
class Program {
    static void Main() {
        // Исходная строка
        string text = "Введите здесь ваш текст";
        // Множество для хранения гласных букв
        HashSet<char> vowels = new HashSet<char>();
        // Множество всех возможных гласных букв
        HashSet<char> allVowels = new HashSet<char>
        {
            'a', 'e', 'ё', 'и', 'o', 'y', 'ы', 'э', 'ю', 'я',
            'A', 'E', 'Ё', 'И', 'O', 'Y', 'Ы', 'Э', 'Ю', 'Я'
        };
        // Проходим по каждому символу в строке
        foreach (char c in text) {
            // Если символ является гласной, добавляем его в множество
            if (allVowels.Contains(c)) {
                vowels.Add(c);
            }
        }
        // Выводим результат
    }
}
```



```
Console.WriteLine("Найденные гласные буквы:");  
foreach (char vowel in vowels)    {  
    Console.Write(vowel + " ");    }    }
```

Объяснение работы программы:

1. Используем `HashSet<char>` для хранения гласных, так как множество автоматически исключает дубликаты
2. Создаём множество `allVowels` со всеми возможными гласными буквами (включая заглавные)
3. Проходим по каждому символу входной строки
4. Проверяем, является ли символ гласной буквой
5. Если да — добавляем его в результирующее множество
6. В конце выводим все найденные гласные

Программа учитывает как строчные, так и заглавные гласные буквы. Вы можете изменить исходную строку `text` на любую другую для тестирования.

Задания для самостоятельного решения:

1. Дана строка. Содержит ли она только латинские символы?
2. Дана строка. Содержит ли она только русские символы?
3. Дана строка. Содержит ли она и прописные символы?
4. Дана строка. Содержит ли она только арабские цифры?
5. Даны 2 строки. Входят ли все символы одной строки в другую?
6. Дана строка. Содержит ли она символы “№ ! » ; , . % : ? @ # \$ ^ & * ()” ?
7. Разработать программу, какому алфавиту (русскому или латинскому), принадлежит введенный с клавиатуры символ.
8. Даны два множества, состоящие из 10 целых чисел из диапазона 1..100. Из них выделить одно подмножество, делящееся на 2 без остатка, и другое, делящееся на 3 без остатка.
9. Дана строка символов. Требуется построить и вывести множество, элементами которого являются буквы, входящие в текст не менее двух раз.
10. Дана строка символов. Требуется построить и вывести множество, элементами которого являются встречающиеся в строке знаки арифметических операций и знаки препинания.

ПРАКТИЧЕСКАЯ РАБОТА № 13

Тема: Составление программ с использованием текстовых файлов

Цель работы: научиться писать программы с использованием базовых операций работы с текстовыми файлами в языке программирования C#.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы.

Справочный материал:

Для работы непосредственно с текстовыми файлами в пространстве System.IO определены специальные классы: **StreamReader** и **StreamWriter**.

Запись в файл и StreamWriter

Для записи в текстовый файл используется класс **StreamWriter**. Некоторые из его конструкторов, которые могут применяться для создания объекта StreamWriter:

- **StreamWriter(string path):** через параметр path передается путь к файлу, который будет связан с потоком
- **StreamWriter(string path, bool append):** параметр append указывает, надо ли добавлять в конец файла данные или же перезаписывать файл. Если равно true, то новые данные добавляются в конец файла. Если равно false, то файл перезаписывается заново
- **StreamWriter(string path, bool append, System.Text.Encoding encoding):** параметр encoding указывает на кодировку, которая будет применяться при записи

Свою функциональность StreamWriter реализует через следующие методы:

- **int Close():** закрывает записываемый файл и освобождает все ресурсы
- **void Flush():** записывает в файл оставшиеся в буфере данные и очищает буфер.
- **Task FlushAsync():** асинхронная версия метода Flush
- **void Write(string value):** записывает в файл данные простейших типов, как int, double, char, string и т.д. Соответственно имеет ряд перегруженных версий для записи данных элементарных типов, например, Write(char value), Write(int value), Write(double value) и т.д.
- **Task WriteAsync(string value):** асинхронная версия метода Write. Обратите внимание, что асинхронные версии есть не для всех перегрузок метода Write.
- **void WriteLine(string value):** также записывает данные, только после записи добавляет в файл символ окончания строки
- **Task WriteLineAsync(string value):** асинхронная версия метода WriteLine

Чтение из файла и StreamReader

Класс StreamReader позволяет нам легко считывать весь текст или отдельные строки из текстового файла.

Некоторые из конструкторов класса StreamReader:

- **StreamReader(string path):** через параметр path передается путь к считываемому файлу
- **StreamReader(string path, System.Text.Encoding encoding):** параметр encoding задает кодировку для чтения файла

Среди методов StreamReader можно выделить следующие:

- void Close(): закрывает считываемый файл и освобождает все ресурсы
- int Peek(): возвращает следующий доступный символ, если символов больше нет, то возвращает -1
- int Read(): считывает и возвращает следующий символ в численном представлении. Имеет перегруженную версию: Read(char[] array, int index, int count), где array - массив, куда считываются символы, index - индекс в массиве array, начиная с которого записываются считываемые символы, и count - максимальное количество считываемых символов
- Task<int> ReadAsync(): асинхронная версия метода Read
- string ReadLine(): считывает одну строку в файле
- string ReadLineAsync(): асинхронная версия метода ReadLine
- string ReadToEnd(): считывает весь текст из файла
- string ReadToEndAsync(): асинхронная версия метода ReadToEnd

Содержание работы:

Задание 1. Создать файл и записать в него текст.

```
using System;
using System.IO;
class Program {
    static void Main()    {
        string path = "example.txt";
        string text = "Привет, мир!";
        // Создание и запись в файл
        File.WriteAllText(path, text);
        Console.WriteLine("Файл создан и записан");    } }
```

Задание 2. Чтение из файла

```
class Program {
    static void Main()    {
        string path = "example.txt";
        // Чтение файла
        using (StreamReader reader = new StreamReader(path))    {
            string line;
            while ((line = reader.ReadLine()) != null)    {
                Console.WriteLine(line);    }    } }
```

Задание 3. Добавить текст в конец файла.

```
using System;
using System.IO;
class Program {
    static void Main()    {
        string path = "example.txt";
        string newText = "\nДополнительная строка";
        // Добавление текста в конец файла
        File.AppendAllText(path, newText);
```

```
Console.WriteLine("Текст добавлен");    } }
```

Задание 4. Записать массив строк в файл и прочитать его.

```
class Program {  
    static void Main()    {  
        string path = "array.txt";  
        string[] lines = { "Строка 1", "Строка 2", "Строка 3" };  
        // Запись массива  
        File.WriteAllLines(path, lines);  
        // Чтение массива  
        string[] readLines = File.ReadAllLines(path);  
        foreach (string line in readLines)    {  
            Console.WriteLine(line);    }    } }
```

Контрольные вопросы

1. Какие классы используются для работы с файлами в C#?
2. В чем отличие File от FileInfo?
3. Как осуществляется запись в файл?
4. Какие режимы открытия файла существуют?
5. Как прочитать файл построчно?

Задания для самостоятельного решения:

1. Написать программу, которая читает текстовый файл и подсчитывает количество слов, строк и символов.
2. Дан файл, содержащий текст, набранный строчными русскими буквами. Получить в другом файле такой же текст, записанный прописными буквами
3. Прочитать файл с числами, отфильтровать четные числа и записать их в новый файл

ПРАКТИЧЕСКАЯ РАБОТА № 14

Тема: Создание программ с использованием типизированных и нетипизированных файлов

Цель работы: научиться писать программы на языке C# с использованием типизированных и нетипизированных файлов.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Каждая величина в C# должна иметь тип. Из-за разнообразия имеющихся в C# типов часто приходится объединять величины различных типов в одних и тех же выражениях исходной программы. Чтобы помочь программисту правильно применить этот диапазон типов, ускорить разработку и увеличить устойчивость конечного приложения к ошибкам, в C# имеются четко определенные правила совместимости между различными типами. Компилятор действует как полиция — он следит за исполнением этих правил, и при их нарушении сообщает об ошибках и дает предупреждения.

Процедура, выполняемая компилятором C# для проверки строгого исполнения правил совместимости в программе, называется проверкой соответствия типов.

Типичные элементы кода, при наличии которых компилятор C# делает проверку соответствия типов:

1. Операция присваивания.
2. Обращение метода к другому методу с передачей ему аргументов.
3. Выражение.

Операция присваивания. Всякий раз, когда переменной присваивается какое-либо значение в операции присваивания, компилятор проверяет совместимость между типом переменной и типом этого значения. При несоответствии выводится сообщение об ошибке.

Передача аргументов при вызове метода. Передача аргументов формальным параметрам при обращении к методу — другой случай потенциальных коллизий типов, и поэтому тщательно проверяется компилятором.

Выражение. Любое выражение в C# имеет конкретный тип. Когда выражения вложены друг в друга, компилятор выбирает тип, подходящий для выражений более высоких уровней, и, естественно, проверяет совместимость.

Неявные преобразования — это преобразования одного типа в другой, которые автоматически выполняются компилятором C#.

Для выполнения неявного преобразования необходимо, чтобы оно не приводило к потере каких-либо данных и не требовало дальнейшего детального исследования участвующих в этом преобразовании конкретных значений.

Типизированные файлы (определяются типом данных File of Тип) используются для хранения однородной информации, например, только числовых данных определённого типа.

Нетипизированные файлы (определяются типом данных File) используются для быстрого доступа к файлу с любыми данными и пересылки данных с высокой скоростью без потерь времени на преобразования данных

Содержание работы:

Задание 1. Работа с типизированным файлом

```
using System;
using System.IO;
public class Person {
    public string Name { get; set; }
    public int Age { get; set; } }
class Program {
    static void Main() {
        // Создание объекта
        Person person = new Person { Name = "Иван", Age = 25 };
        // Запись в бинарный файл
        using (BinaryWriter writer = new BinaryWriter(File.Open("people.dat",
        FileMode.Create))) {
            writer.Write(person.Name);
            writer.Write(person.Age); }
        // Чтение из бинарного файла
        using (BinaryReader reader = new BinaryReader(File.Open("people.dat",
        FileMode.Open))) {
            string name = reader.ReadString();
            int age = reader.ReadInt32();
            Console.WriteLine($"Имя: {name}, Возраст: {age}"); } } }
```

Задание 2. Работа с нетипизированными файлами в C#

```
using System;
using System.IO;
class Program {
    static void Main() {
        // Создание массива байтов
        byte[] data = { 0x48, 0x65, 0x6C, 0x6C, 0x6F }; // "Hello" в ASCII
        // Запись в нетипизированный файл
        using (FileStream fs = new FileStream("data.bin", FileMode.Create)) {
            fs.Write(data, 0, data.Length); }
        // Чтение из нетипизированного файла
        using (FileStream fs = new FileStream("data.bin", FileMode.Open)) {
            byte[] buffer = new byte[fs.Length];
            fs.Read(buffer, 0, buffer.Length);
            // Преобразование байтов в строку
            string result = System.Text.Encoding.ASCII.GetString(buffer);
            Console.WriteLine($"Прочитанные данные: {result}"); } } }
```

Задания для самостоятельного решения:

1. Записываем текст, а старые данные стираются (если файл не существует создается новый файл и записывается текст)
2. Запись и чтение массива целых чисел

ПРАКТИЧЕСКАЯ РАБОТА № 15

Тема: Организация процедур

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием стандартных функций и процедур для работы со строками.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Подпрограмма - это отдельная часть программы, имеющая имя и решающая свою отдельную задачу. Располагается подпрограмма в начале основной программы и может быть запущена (вызвана) из основной программы по указанию имени. Использование подпрограмм позволяет избежать дублирования кода, в случае если необходимо один и тот же код писать в разных местах программы.

Каждая подпрограмма должна решать только одну задачу, либо что-то вычислять, либо выводить какие-либо данные, либо делать что-то еще.

Подпрограммы, или методы, бывают двух типов - функции (те, которые возвращают результат работы) и процедуры (те, которые не возвращают). Процедуры и функции связываются с классом, они обеспечивают функциональность данных класса и называются методами класса. Главную роль в программной системе играют данные, а функции лишь служат данным. В C# процедуры и функции существуют только как методы некоторого класса, они не существуют вне класса. В данном контексте понятие класс распространяется и на все его частные случаи - структуры, интерфейсы, делегаты. В языке C# нет специальных ключевых слов - `procedure` и `function`, но присутствуют сами эти понятия. Синтаксис объявления метода позволяет однозначно определить, чем является метод - процедурой или функцией.

Функция отличается от процедуры двумя особенностями:

1. Она всегда вычисляет некоторое значение, возвращаемое в качестве результата функции;
2. И вызывается в выражениях.

Процедура C# имеет свои особенности:

1. Она возвращает формальный результат `void`, указывающий на отсутствие результата;
2. Вызов процедуры является оператором языка;
3. И она имеет входные и выходные аргументы, причем выходных аргументов - ее результатов - может быть достаточно много.

Предположим, что нам нужно выводить на экран строку "Error" каждый раз, когда в коде может возникнуть ошибка по вине пользователя (например, когда он вводит неверные данные).

Это можно сделать, написав оператор `Console.WriteLine("Error");`

А теперь представим, что такую строчку нужно вставить во многих местах программы. Конечно, можно просто везде ее написать. Но это решение имеет два недостатка.

1) данная строка будет храниться в памяти много раз;

2) если мы захотим изменить вывод при ошибке, то придется менять эту строку по всей программе, что достаточно неудобно.

Для таких случаев и нужны методы и процедуры. Программа с процедурой может выглядеть следующим образом:

```
using System;
class Program {
    static void PrintError() {
        Console.WriteLine("Error"); }
    static void Main() {
        PrintError(); } }
```

Процедура начинается со слова void. После имени процедуры записаны пустые скобки.

Все операторы, которые выполняются в процедуре, записываются с отступом.

Модификатор Static означает, что данное поле, метод или свойство будет принадлежать не каждому объекту класса, а всем им вместе.

Методы и процедуры записываются до главного метода Main().

Чтобы обратиться к процедуре, в основной программе необходимо вызвать ее по имени и не забыть написать скобки.

Вызывать процедуру в программе можно сколько угодно раз.

А теперь представим, что нам необходимо в ответ на ошибку пользователя вывести разные сообщения, в зависимости от того, какую именно ошибку он сделал. В этом случае можно для каждой ошибки написать свою процедуру:

```
void printErrorZero() {
    Console.WriteLine("Error. Division by zero!"); }
void printErrorInput() {
    Console.WriteLine("Error in input!"); }
```

А если возможных ошибок будет намного больше? Тогда такое решение нам не подойдет.

Надо научиться управлять процедурой, указывая ей, какое сообщение на ошибку нужно вывести.

Для этого нам понадобятся параметры, которые мы будем записывать в круглых скобках, после имени процедуры

```
void printError(string s) {
    Console.WriteLine(s); }
```

В данной процедуре s - это параметр - специальная переменная, которая позволяет управлять процедурой.

Параметр - это переменная, от значения которой зависит работа подпрограммы. Имена параметров перечисляются через запятую в заголовке подпрограммы. Перед параметром записывается его тип.

Теперь при вызове процедуры нужно в скобках указывать фактическое значение, которое будет присвоено параметру (переменной s) внутри нашей процедуры printError("Error! Division by zero!");

Такое значение называется аргументом.

Аргумент - это значение параметра, которое передается подпрограмме при ее вызове. Аргументом может быть не только постоянное значение, но и переменная или арифметическое выражение.

Содержание работы:

Задание 1. В программе необходимо добавить вызовы процедуры таким образом, чтобы при вводе значения 0 выводилась на экран ошибка "Error: division by zero!", а при вводе любого другого числа выводилась ошибка "Error in input!". Ваша задача - оформить правильный вызов процедуры. Запрещенные операторы: return.

```
using System;
class Program {
static void printError(string s) {
Console.WriteLine(s); }
static void Main() {
int N = Convert.ToInt32(Console.ReadLine());
if (N == 0) {
// оформите вызов процедуры и укажите значение параметра равным "Error:
division by zero!"
}
else {
// оформите вызов процедуры и укажите значение параметра равным "Error in
input!"
} } }
```

Задание 2. Напишите процедуру, которая выводит прямоугольный треугольник, на сторонах которого n символов 'm'.

Пример.

Входные данные	Выходные данные
3	m mm mmm

Задание 3. Напишите процедуру, которая выводит на экран все делители числа N в одну строку через пробел.

Пример.

Входные данные	Выходные данные
10	1 2 5 10

Задание 4. Напишите процедуру, которая принимает два параметра – натуральное число N , и строку S – и выводит на экран строку S , повторив ее N раз.

Пример.

Входные данные	Выходные данные
3	okokok

ПРАКТИЧЕСКАЯ РАБОТА № 16

Тема: Организация функций

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием стандартных функций и процедур для работы со строками.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Функции можно назвать небольшими подпрограммами, куда можно вынести повторяющийся код и обращаться к нему, когда это будет нужно. Функции значительно облегчают построение программ, так как нам не надо копировать однотипный код множество раз, а можно просто воспользоваться одной общей функцией.

Многие путают функции и методы и не понимают отличий между ними. На самом деле отличий нет, так как что методы, что функции являются одним и тем же. Функции что записаны вне классов называют функциями, а функции что записаны внутри классов называются методами. Поскольку **C#** это объектно-ориентированный язык, то лучше говорить методы, хотя это не имеет никакого значения.

Точно такая же ситуация обстоит с переменным. В классах переменные называются полями, а вне классов - переменными.

Для создания функций необходимо указать возвращаемый тип данных, указать название и параметры. В случае, когда функция ничего не возвращает, то указывается тип данных `void`.

Перед типом данных всегда прописывается модификатор доступа

Поскольку мы хотим обращаться к функциям напрямую без создания объекта, то мы прописываем ключевое слово `static`. Оно говорит компилятору что функция принадлежит всему классу, а не конкретным объектам.

Модификатор доступа — откуда этот метод может быть вызван.

Некоторые из них:

- *private* — наиболее ограничительный, допускающий доступ к методу только из содержащего его класса или структуры
- *public* — доступ из любого фрагмента кода в рамках приложения
- *protected* — позволяет получить доступ из содержащего класса или из производных классов
- *internal* — доступ из файлов внутри одной сборки
- *static* — указывает, что метод является статическим членом класса, а не членом экземпляра конкретного объекта

Тип возвращаемого значения — используется для указания типа возвращаемого значения. Используйте `void`, если метод не возвращает значение

Идентификатор (имя) метода — все методы должны иметь идентификатор (имя), чтобы была возможность вызывать метод в коде. Правила идентификаторов применяются и к именам методов

Список аргументов (параметров) — разделенный запятыми список аргументов, передаваемых в метод

```
public Boolean StartService(string serviceName)
{
    // код функции
}
```

В примере:

public модификатор доступа,
Boolean тип возвращаемого значения,
StartService идентификатор (имя),
string serviceName параметр (аргумент).

Чтобы вернуть значение из функции, используйте оператор:

return выражение;

Если вместо возвращаемого типа у нас присутствует ключевое слово **void**, то это означает, что функция не вернет никакого значения.

```
static void SayHello()
{
    Console.WriteLine("Hello");
}
```

Содержание работы:

Задание 1. Создайте метод **Sum()**, который принимает два целочисленных аргумента и суммирует их. Метод не возвращает никакого значения (именно поэтому вы должны использовать ключевое слово **void**).

Пример выполнения:

Введите два числа

20 40

Сумма 20 + 40 = 60

1. Запустите Visual Studio.

2. Создайте консольное приложение, назовите проект **Lesson_5Lab1**.

3. В окне Solution Explorer (Обозреватель решений) найдите файл **Program.cs** и переименуйте его в **L5Lab1.cs**.

4. В главной функции запросите пользователя ввести два числа:

```
...
Console.WriteLine("Введите два числа");
int a = int.Parse(Console.ReadLine());
int b = int.Parse(Console.ReadLine());
...
```

5. Поместите курсор после закрывающей фигурной скобки функции **Main()** и нажмите клавишу *enter*, чтобы вводить новую функцию уже после функции **Main()** (Мы делаем это потому, что нельзя помещать методы/функции внутри другой функции).

6. Объявите новый метод **Sum()**, который будет использоваться для суммы переданных в него значений:

```
...
static void Sum(int first, int second)
{
    int sum = first + second;
    Console.WriteLine($"Сумма {first} + {second} = {sum}");
}
...
```

Метод не возвращает никакого значения основной программе, поэтому мы должны использовать ключевое слово `void`.

7.Теперь мы можем вызвать этот метод из основной функции. Введите следующий код в фигурные скобки `Main()`:

```
static void Main(string[] args)
{
    ...
    Sum(a, b);
}
```

8.Щелкните `CTRL+F5`, чтобы запустить приложение без отладки.

9.Созданная нами функция `Sum()` не возвращает значения. Теперь мы изменим этот метод так, чтобы он возвращал результат вызывающему методу (основному методу, откуда он был вызван).

10.Закомментируйте код функции `Sum()`, используя горячие клавиши `[CTRL]+k+c`:

```
//static void Sum(int first, int second)
//{
//    int sum = first + second;
//    Console.WriteLine($"The sum of {first} and {second} is: {sum}");
//}
```

11.Установите курсор после комментариев и введите следующий код:

```
...
static int Sum(int first, int second)
{
    int sum = first + second;
    return sum;
}
...
```

Метод возвращает целочисленное значение, именно поэтому мы используем `int` в сигнатуре (`static int Sum(...)`).

Обратите внимание, что имена параметров, которые мы указали здесь в сигнатуре метода, могут не совпадать с именами аргументов, которые мы передали. Эти параметры становятся локальными переменными в рамках этого метода.

12.Затем нам нужно изменить способ вызова метода. Объявите целочисленную переменную (в функции `Main()`), чтобы получить возвращаемое значение. Выведите результат в окно консоли

```

...
static void Main(string[] args)
{
    int result = Sum(a, b);
    Console.WriteLine($"Сумма {a} + {b} = {result}");
}
...

```

13. Запустите приложение. Результаты должны быть одинаковыми.

14. Перегрузка метода / функции:

Перегрузка метода означает наличие метода с тем же именем, но разными сигнатурами (обычно с разным количеством параметров).

15. Давайте теперь перегрузим наш метод Sum(). Для этого мы создадим два дополнительных метода с одинаковыми названиями.

16. Поместите курсор после метода Sum(). Сначала создайте метод, который принимает три целых числа:

```

static int Sum(int first, int second, int third)
{
    int sum = first + second + third;
    return sum;
}

```

17. Этот метод использует то же имя, что и предыдущий метод Sum(), который принимает два целых числа, но параметры здесь указывают, что метод ожидает три целых числа в качестве аргументов. Компилятор «узнает», какой метод следует вызвать, основываясь на количестве переданных аргументов.

18. Затем введите следующий код для создания метода Sum(), принимающего два вещественных аргумента (типа double):

```

...
static double Sum(double first, double second)
{
    double result = first + second;
    return result;
}
...

```

19. Наконец, измените код в Main(), который вызывает данные методы:

```

static void Main(string[] args)
{
    ...
    int result = Sum(a, b);
    Console.WriteLine($"Вызов Sum() с двумя аргументами: {result}");

    int result3 = Sum(10, 50, 80);
    Console.WriteLine($"Вызов Sum() с тремя аргументами: {result3}");

    double dblResult = Sum(20.5, 30.6);
    Console.WriteLine($"Вызов Sum() с вещественными аргументами: {dblResult}");
}

```

20. Запустите приложение еще раз и проверьте выходные данные. Вы должны увидеть правильные результаты суммы для всех трех различных вызовов.

Несмотря на то, что у них всех идентификатор Sum, компилятор выбирает правильный метод для вызова на основе сигнатуры метода. Именно так работает перегрузка метода.

Задания для самостоятельного решения:

1. Вводятся три числа — длины трех сторон треугольника. Создайте функцию Perimeter(), которая вычисляет периметр треугольника по длинам трех его сторон.

Указание 1: Метод Perimeter() должен принимать в качестве аргументов три целых числа.

Указание 2: Метод не должен возвращать никакого значения, поэтому вы должны использовать ключевое слово void в сигнатуре:

static void Perimeter(...);

Указание 3: Не забудьте преобразовать введенные значения в целые числа.

Например: `int a = int.Parse(Console.ReadLine());`

Пример выполнения:

Введите значения для трех сторон треугольника:

3 5 6

Периметр: 14

2. Создать функцию, которая выводит 10 четных чисел, начиная с 2: 2, 4, 6, 8 и т.д., используя цикл с постусловием.

3. Создать функцию, которая выводит 10 нечётных чисел в обратной последовательности, начиная с 19: 19, 17, 15 и т. д., используя цикл с предусловием.

ПРАКТИЧЕСКАЯ РАБОТА № 17

Тема: Применение рекурсивных функций

Цель работы: Получение практических навыков по организации функций при построении алгоритмов; решение задач с использованием рекурсивных алгоритмов.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Рекурсия – это одна из фундаментальных концепций в математике и программировании и является мощным средством, позволяющим строить элегантные и выразительные алгоритмы. Рекурсия — это такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения ее операторов обращается сама к себе. Если процедура **p** содержит явное обращение к самой себе, то она называется явно рекурсивной. Если процедура **p** содержит обращение к некоторой процедуре **q**, которая в свою очередь содержит прямое или косвенное обращение к **p**, то **p** – называется косвенно рекурсивной.

Но рекурсивная программа не может вызывать себя бесконечно, иначе она никогда не остановится, таким образом в программе (функции) должен присутствовать еще один важный элемент – так называемое терминальное (граничное) условие, то есть условие при котором программа прекращает рекурсивный процесс.

Рекурсивная функция Фибоначчи

Другим распространенным показательным примером рекурсивной функции служит функция, вычисляющая числа Фибоначчи. n -й член последовательности Фибоначчи определяется по формуле: $f(n)=f(n-1) + f(n-2)$, причем $f(0)=0$, а $f(1)=1$. То есть последовательность Фибоначчи будет выглядеть так 0 (0-й член), 1 (1-й член), 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, Для определения чисел этой последовательности определим следующий метод:

```
int Fibonacci(int n) {  
    if (n == 0 || n == 1) return n;  
    return Fibonacci(n - 1) + Fibonacci(n - 2); }  
int fib4 = Fibonacci(4);  
int fib5 = Fibonacci(5);  
int fib6 = Fibonacci(6);  
Console.WriteLine($"4 число Фибоначчи = {fib4}");  
Console.WriteLine($"5 число Фибоначчи = {fib5}");  
Console.WriteLine($"6 число Фибоначчи = {fib6}");
```

Здесь базовый вариант выглядит следующий образом:

```
if (n == 0 || n == 1) return n;
```

То есть, если мы ищем нулевой или первый элемент последовательности, то возвращается это же число - 0 или 1. Иначе возвращается результат выражения $Fibonacci(n - 1) + Fibonacci(n - 2)$;

Рекурсии и циклы

Это простейшие пример рекурсивных функций, которые призваны дать понимание работы рекурсии. В то же время для обеих функций вместо рекурсий можно использовать циклические конструкции. И, как правило, альтернативы на основе циклов работают быстрее и более эффективны, чем рекурсия. Например, вычисление чисел Фибоначчи с помощью циклов:

```
static int Fibonacci2(int n) {  
    int result = 0;  
    int b = 1;  
    int tmp;  
    for (int i = 0; i < n; i++) {  
        tmp = result;  
        result = b;  
        b += tmp;    }  
    return result;    }
```

В то же время в некоторых ситуациях рекурсия предоставляет элегантное решение, например, при обходе различных древовидных представлений, к примеру, дерева каталогов и файлов.

Содержание работы:

Задание 1: Написать программу для вывода n первых членов арифметической прогрессии 1, 2, 3... с использованием рекурсивного метода.

```
using System;  
class Program{  
    static uint ArithmeticProgression(uint n){  
        if (n == 0){  
            return 1;}  
        return ArithmeticProgression(n - 1) + 1;}  
    static void Main(string[] args){  
        Console.Write("N = ");  
        var n = Convert.ToInt32(Console.ReadLine());  
        for (uint i = 0; i < n; i++){  
            Console.WriteLine(ArithmeticProgression(i));}  
        Console.ReadLine();}}
```

Задание 2. Напишите программу для поиска индекса максимального элемента массива с использованием рекурсии.

```
using System;  
class Program{  
    static int IndexOfMax(int[] array, int len) {  
        if (len == 0){  
            return len;}  
        var i = IndexOfMax(array, len - 1);  
        return array[len] > array[i] ? len : i;}  
    static void Main(string[] args){  
        var a = new[] { 0, 5, 6, 1, 9, 7, 8, 3, 2, 4 };
```

```
Console.WriteLine("Индекс максимального элемента массива: " + IndexOfMax(a,  
a.Length - 1));  
Console.ReadLine();    }}
```

Задание для самостоятельного решения:

1. Напишите программу для получения суммы n первых членов арифметической прогрессии. Разность прогрессии d задается в качестве параметра.

2. Напишите программу для переворота строки с использованием рекурсии. Строка “abc” переворачивается в “cba”.

ПРАКТИЧЕСКАЯ РАБОТА № 18

Тема: Программирование модуля

Цель работы: научиться писать отдельные модули на языке с#

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Модуль — функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом или поименованной непрерывной её части, предназначенный для использования в других программах. Модули позволяют разбивать сложные задачи на более мелкие в соответствии с принципом модульности. Обычно проектируются таким образом, чтобы предоставлять программистам удобную для многократного использования функциональность (интерфейс) в виде набора функций, классов, констант. Модули могут объединяться в пакеты и, далее, в библиотеки. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы. В большинстве случаев различные модули могут запускаться как на одном сервере, так и на разных, для распределения нагрузки и создания распределенной архитектуры.

Все свойства и методы классов имеют права доступа. По умолчанию, все содержимое класса является доступным для чтения и записи только для него самого. Для того, чтобы разрешить доступ к данным класса извне, используют модификатор доступа `public`. Все функции и переменные, которые находятся после модификатора `public`, становятся доступными из всех частей программы. Закрытые данные класса размещаются после модификатора доступа `private`. Если отсутствует модификатор `public`, то все функции и переменные, по умолчанию являются закрытыми (как в первом примере).

Обычно, приватными делают все свойства класса, а публичными — его методы. Все действия с закрытыми свойствами класса реализуются через его методы.

Содержание работы:

Задание 1. Написать программу для определения класса, описывающего некоторого студента вуза.

1. Запустить Visual Studio

2. Создать консольное приложение.

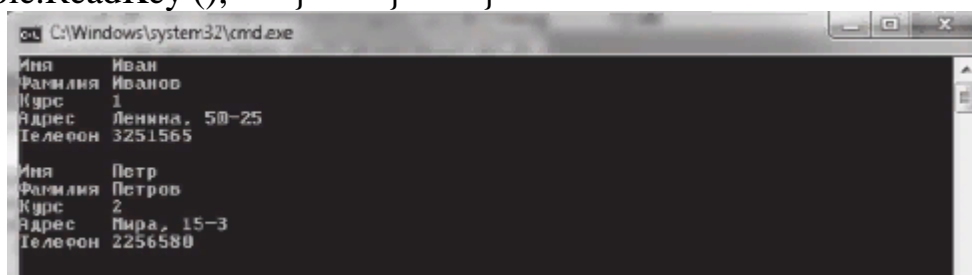
3. Код будет выглядеть следующим образом:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication12{
class Student    {
//описываем характеристики объекта класса
//характеристики доступны только внутри данного класса
private string name;
```

```

private string sename;
private int kurs;
private string adress;
private int tel;
//определяем конструктор класса для создания объектов данного класса
//в скобках указаны параметры, которые передаются методу для
создания//объекта с //указанными свойствами
public Student (string name, string sename, int kurs, string adress, int tel) {
this.name = name;
this.sename = sename;
this.kurs = kurs;
this.adress = adress;
this.tel = tel;      }
//определяем метод для объекта класса (его поведение)
//метод будет выводить на экран информацию о студенте
public void Print ()      {
Console.WriteLine ("Имя\t"+name+"\nФамилия\t"+sename+"\nКурс\
t"+kurs+"\nАдрес\t"+adress+"\nТелефон\t"+tel+"\n");      }      }
class Program      {
static void Main (string [] args) {
//создаем первый объект нашего класса
Student St1 = new Student («Иван»,»Иванов»,1,»Ленина, 50–25»,3251565);
//создаем второй объект нашего класса
Student St2 = new Student («Петр», «Петров», 2, «Мира, 15–3», 2256580);
//используем метод вывода на экран информации о студентах,
//определенный ранее в классе
St1.Print ();
St2.Print ();
Console.ReadKey ();      }      }      }

```



```

C:\Windows\system32\cmd.exe
Имя      Иван
Фамилия  Иванов
Курс     1
Адрес    Ленина, 50–25
Телефон  3251565

Имя      Петр
Фамилия  Петров
Курс     2
Адрес    Мира, 15–3
Телефон  2256580

```

Задания для самостоятельного решения:

1. Описать класс «Домашняя библиотека».
2. Описать класс «Записная книжка».
3. Описать класс «Студенческая группа».

ПРАКТИЧЕСКАЯ РАБОТА № 19

Тема: Использование указателей для организации связанных списков

Цель работы: получить навыки работы с указателями для организации связанных списков.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Указатели позволяют получить доступ к определенной ячейке памяти и произвести определенные манипуляции со значением, хранящимся в этой ячейке. В языке C# указатели очень редко используются, однако в некоторых случаях можно прибегать к ним для оптимизации приложений. Код, применяющий указатели, еще называют небезопасным кодом. Однако это не значит, что он представляет какую-то опасность. Просто при работе с ним все действия по использованию памяти, в том числе по ее очистке, ложится целиком на нас, а не на среду CLR. И с точки зрения CLR такой код не безопасен, так как среда не может проверить данный код, поэтому повышается вероятность различного рода ошибок.

Чтобы использовать небезопасный код в C#, надо первым делом указать проекту, что он будет работать с небезопасным кодом. Для этого надо установить в настройках проекта соответствующий флаг - в меню Project (Проект) найти Свойства проекта. Затем в меню Build установить флажок Allow unsafe code (Разрешить небезопасный код). Теперь мы можем приступать к работе с небезопасным кодом и указателями.

Указатели позволяют получить доступ к определённой ячейке памяти и произвести манипуляции со значением, хранящимся в этой ячейке.

В C# указатели очень редко используются, но в некоторых случаях к ним прибегают для оптимизации приложений.

Код, применяющий указатели, называют небезопасным, так как при работе с ним все действия по использованию памяти, в том числе по её очистке, выполняет разработчик, а не среда CLR.

Чтобы использовать небезопасный код в C#, нужно установить в настройках проекта соответствующий флаг.

Связный список (Linked List) представляет набор связанных узлов, каждый из которых хранит собственно данные и ссылку на следующий узел. Таким образом, если в массиве положение элементов определяется индексами, то в связанном списке - указателями на следующий и (или) на предыдущий элемент.

Связные списки могут различаться. Есть односвязные списки, в которых каждый узел хранит указатель только на следующий узел. Есть двусвязные списки: в них каждый элемент хранит ссылку как на следующий элемент, так и на предыдущий. Есть кольцевые замкнутые списки. В данном случае мы рассмотрим создание односвязного списка

Содержание работы:

Задание 1. Создать список

1. Перед созданием списка нам надо определить класс узла, который будет представлять одиночный объект в списке:

```

public class Node<T> {
    public Node(T data)    {
        Data = data;    }
    public T Data { get; set; }
    public Node<T>? Next { get; set; } }

```

Класс Node является обобщенным, поэтому может хранить данные любого типа. Для хранения данных предназначено свойство Data. Для ссылки на следующий узел определено свойство Next.

2. Далее определим сам класс списка:

```

using System.Collections;
using System.Collections.Generic;
public class LinkedList<T> : IEnumerable<T> { // односвязный список
    Node<T>? head; // головной/первый элемент
    Node<T>? tail; // последний/хвостовой элемент
    int count; // количество элементов в списке
    // добавление элемента
    public void Add(T data)    {
        Node<T> node = new Node<T>(data);
        if (head == null)    head = node;
        else    tail!.Next = node;
        tail = node;
        count++;    }
    // удаление элемента
    public bool Remove(T data)    {
        Node<T>? current = head;
        Node<T>? previous = null;
        while (current != null && current.Data != null)    {
            if (current.Data.Equals(data))    {
                // Если узел в середине или в конце
                if (previous != null)    {
                    // убираем узел current, теперь previous ссылается не на current, а на current.Next
                    previous.Next = current.Next;
                    // Если current.Next не установлен, значит узел последний,
                    // изменяем переменную tail
                    if (current.Next == null)
                        tail = previous;    }
                else    {
                    // если удаляется первый элемент переустанавливаем значение head
                    head = head?.Next;
                    // если после удаления список пуст, сбрасываем tail
                    if (head == null)    tail = null;    }
                count--;
                return true;    }
            previous = current;
            current = current.Next;    }
    }

```

```

    return false; }
public int Count { get { return count; } }
public bool IsEmpty { get { return count == 0; } }
// очистка списка
public void Clear() {
    head = null;
    tail = null;
    count = 0; }
// содержит ли список элемент
public bool Contains(T data) {
    Node<T>? current = head;
    while (current != null && current.Data != null) {
        if (current.Data.Equals(data)) return true;
        current = current.Next;
    }
    return false; }
// добавление в начало
public void AppendFirst(T data) {
    Node<T> node = new Node<T>(data);
    node.Next = head;
    head = node;
    if (count == 0) tail = head;
    count++; }
IEnumerator<T> IEnumerable<T>.GetEnumerator() {
    Node<T>? current = head;
    while (current != null) {
        yield return current.Data;
        current = current.Next;
    } }
// реализация интерфейса IEnumerable
IEnumerator IEnumerable.GetEnumerator() {
    return ((IEnumerable<T>)this).GetEnumerator(); }}

```

Разберем основные моменты. В зависимости от конкретных задач реализация списков может отличаться, но для всех реализаций характерны прежде всего два метода: добавление и удаление.

3. Но прежде чем выполнять различные операции с данными, в классе списка определяются три переменные:

```

Node<T>? head; // головной/первый элемент
Node<T>? tail; // последний/хвостовой элемент
int count; // количество элементов в списке

```

4. Метод добавления:

```

public void Add(T data) {
    Node<T> node = new Node<T>(data);
    if (head == null) head = node;
    else tail!.Next = node;
    tail = node;
}

```

```
count++; }
```

5. Если у нас не установлена переменная head (то есть список пуст), то устанавливаем head и tail. После добавления первого элемента они будут указывать на один и тот же объект.

Если же в списке есть как минимум один элемент, то устанавливаем свойство tail.Next - теперь оно хранит ссылку на новый узел. И переустанавливаем tail - теперь она ссылается на новый узел.

6. Важно отметить наличие переменной tail, которая указывает на последний элемент. Ряд реализаций не используют подобную переменную и добавляют иным образом:

```
public void AddWithoutTail(T data) {  
    Node<T> node = new Node<T>(data);  
    if (head == null) {  
        head = node;    }  
    else {  
        Node<T> current = head;  
        // ищем последний элемент  
        while (current.Next != null) {  
            current = current.Next;    }  
        //устанавливаем последний элемент  
        current.Next = node;    }  
    count++; }
```

Данный способ вполне рабочий и нередко встречается, однако необходимость перебора элементов для нахождения последнего увеличивает время на поиск и сложность алгоритма. Она равна $O(n)$.

7. Особняком стоит метод добавления в начало списка, где нам достаточно переустановить ссылку на головной элемент:

```
public void AppendFirst(T data) {  
    Node<T> node = new Node<T>(data);  
    node.Next = head;  
    head = node;  
    if (count == 0)    tail = head;  
    count++; }
```

8. Удаление элемента

```
public bool Remove(T data)    {  
    Node<T>? current = head;  
    Node<T>? previous = null;  
    while (current != null && current.Data != null)    {  
        if (current.Data.Equals(data))    {  
            // Если узел в середине или в конце  
            if (previous != null)    {  
                // убираем узел current, теперь previous ссылается не на current, а на current.Next  
                previous.Next = current.Next;  
                // Если current.Next не установлен, значит узел последний,  
                // изменяем переменную tail
```



```

        if (current.Next == null)    tail = previous;    }
    else    {
        // если удаляется первый элемент  переустанавливаем значение head
        head = head?.Next;
        // если после удаления список пуст, сбрасываем tail
        if (head == null)    tail = null;    }
    count--;
    return true;    }
    previous = current;
    current = current.Next;    }
return false;    }

```

Алгоритм удаления элемента представляет следующую последовательность шагов:

1. Поиск элемента в списке путем перебора всех элементов
2. Установка свойства Next у предыдущего узла (по отношению к удаляемому) на следующий узел по отношению к удаляемому.

Для отслеживания предыдущего узла применяется переменная previous. Если элемент найден, и переменная previous равна null, то удаление идет сначала, и в этом случае происходит переустановка переменной head, то есть головного элемента.

Если же previous не равна null, то реализуются шаги выше описанного алгоритма.

9. Чтобы проверить наличие элемента, используется метод Contains:

```

public bool Contains(T data) {
    Node<T>? current = head;
    while (current != null && current.Data != null)    {
        if (current.Data.Equals(data))
            return true;
        current = current.Next;    }
    return false;    }

```

10. И для того, чтобы список можно было бы перебрать во внешней программе с помощью цикла for-each, класс списка реализует интерфейс **IEnumerable**:

// реализация интерфейса IEnumerable

```

IEnumerator IEnumerable.GetEnumerator(){
    return ((IEnumerable<T>)this).GetEnumerator();    }
IEnumerator<T> IEnumerable<T>.GetEnumerator()    {
    Node<T>? current = head;
    while (current != null)    {
        yield return current.Data;
        current = current.Next;    }    }

```

11. Реализация данного интерфейса не является неотъемлемой частью односвязных списков, однако предоставляет эффективный метод для перебора коллекции в цикле foreach. Иначе нам бы пришлось реализовать какие-то собственные конструкции по перебору списка.

12. Применение LinkedList:

```

LinkedList<string> linkedList = new LinkedList<string>    {
    // добавление элементов
    "Tom",
    "Alice",
    "Bob",
    "Sam"    };
// выводим элементы
foreach (var item in linkedList) {
    Console.WriteLine(item);    }
// удаляем элемент
linkedList.Remove("Alice");
Console.WriteLine("\nПосле удаления Alice");
foreach (var item in linkedList) Console.WriteLine(item);
// проверяем наличие элемента
bool isPresent = linkedList.Contains("Sam");
Console.WriteLine(isPresent ? "Sam присутствует" : "Sam отсутствует");
// добавляем элемент в начало
linkedList.AppendFirst("Bill");
Console.WriteLine("\nПосле добавления Billa");
foreach (var item in linkedList) Console.WriteLine(item);
13. Консольный вывод:

```

```

Tom
Alice
Bob
Sam

После удаления Alice
Tom
Bob
Sam
Sam присутствует

После добавления Billa
Bill
Tom
Bob
Sam

```

Задание 2. Создать связный мультисписок на основе указателей

ПРАКТИЧЕСКАЯ РАБОТА № 20

Тема: Изучение интегрированной среды разработчика

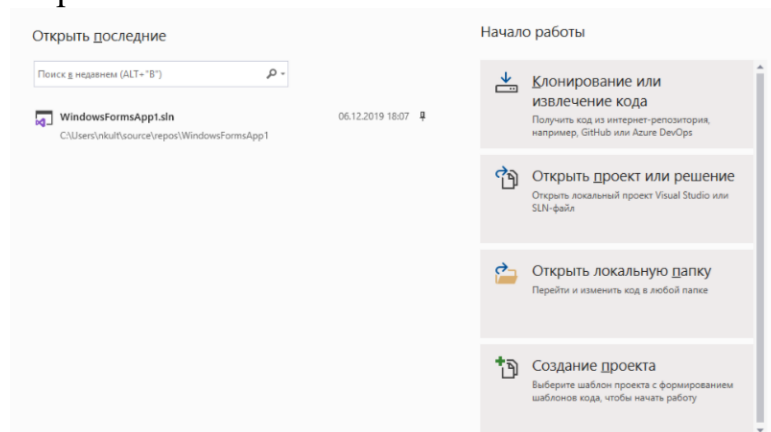
Цель работы: изучить среду быстрой разработки приложений MS Visual Studio; научиться создавать простейшие приложения типа Windows Forms, содержащие базовые элементы управления.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

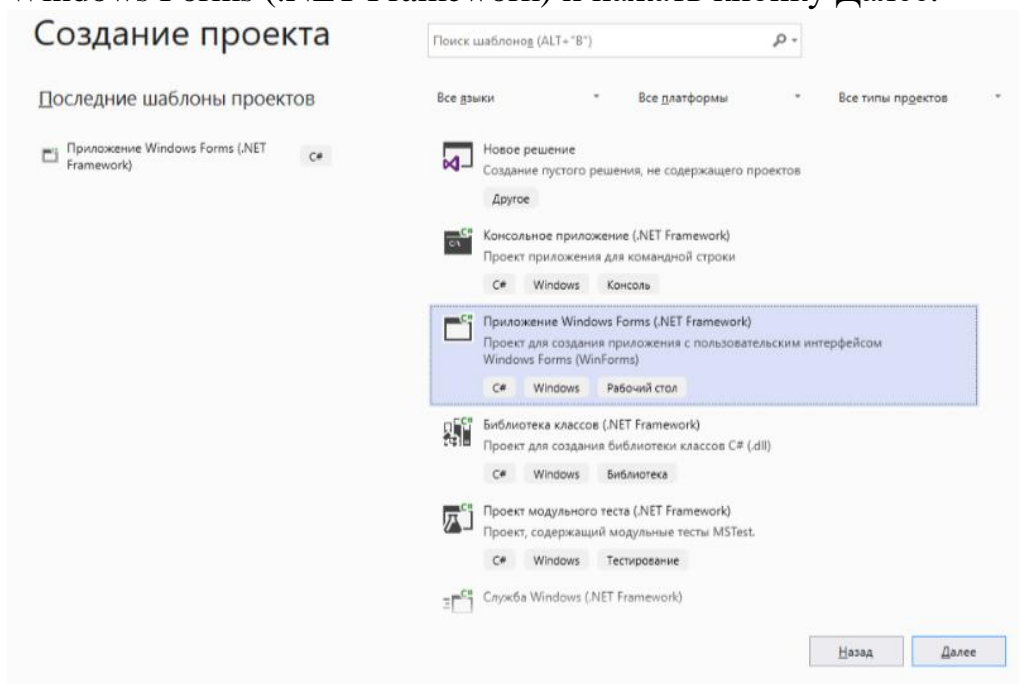
Справочный материал:

Среда Visual Studio визуально реализуется в виде одного окна с несколькими панелями инструментов. Количество, расположение, размер и вид панелей может меняться программистом или самой средой разработки в зависимости от текущего режима работы среды или пожеланий программиста, что значительно повышает производительность работы.

1. Запустить Visual Studio или, если Visual Studio уже запущена, в меню Файл выбрать команду Создать проект.
2. В открывшемся окне Visual Studio 2022, в списке Начало работы, нажать кнопку Создание проекта.



3. В открывшемся окне Создание Проекта в списке выбрать C# Приложение Windows Forms (.NET Framework) и нажать кнопку Далее.



4. В окне Настроить новый проект в поле Имя проекта надо ввести название проекта и, если необходимо, изменить расположение папки проекта. Затем нужно нажать кнопку Создать.

Настроить новый проект

Приложение Windows Forms (.NET Framework) C# Windows Рабочий стол

Имя проекта
WindowsFormsApp1

Расположение
C:\Users\vnkult\source\repos

Имя решения
WindowsFormsApp1

☐ Поместить решение и проект в одном каталоге

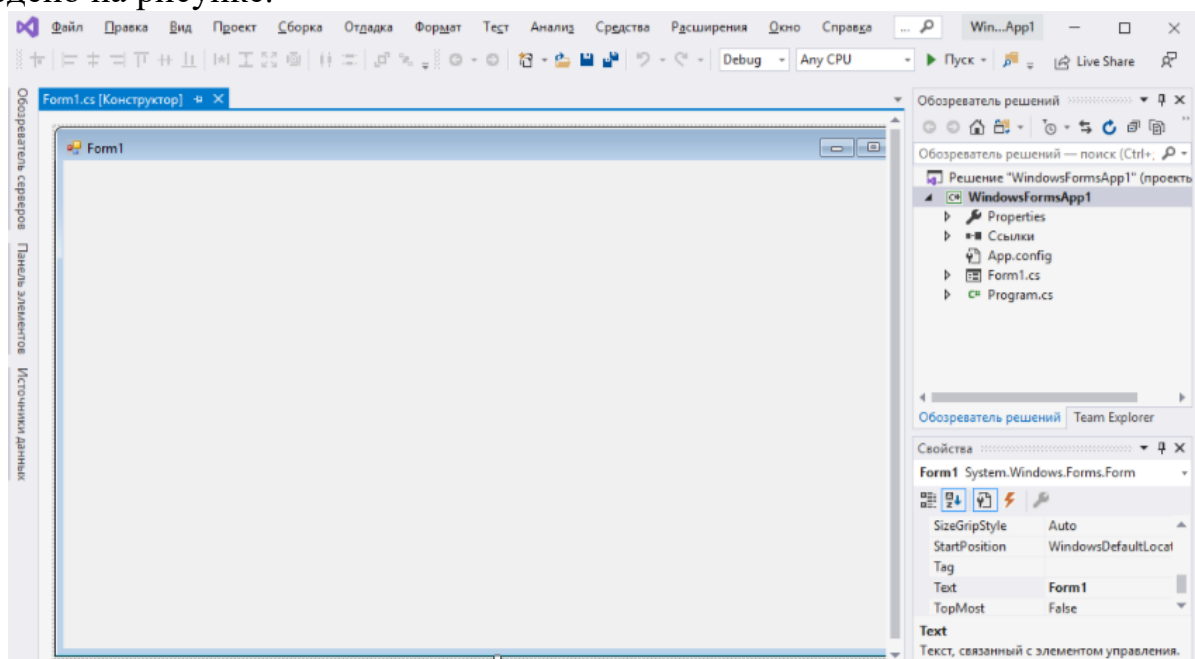
Платформа
.NET Framework 4.7.2

Назад Создать

В результате описанных действий будет создан *проект* — совокупность папок и файлов, содержащих всю информацию, необходимую для создания выполняемого (exe) файла программы.

Приложение Windows Forms — данный тип проекта позволяет создать полноценное приложение с окнами и элементами управления (кнопками, полями ввода и пр.)

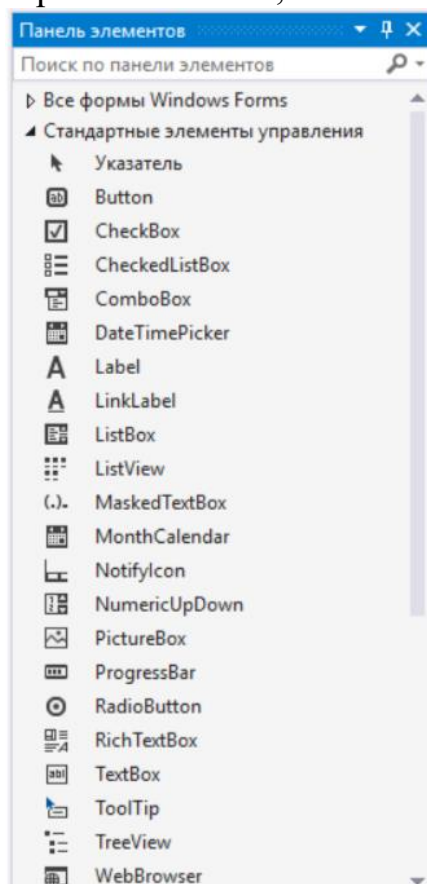
Главное окно Microsoft Visual Studio в начале работы над новым проектом приведено на рисунке.



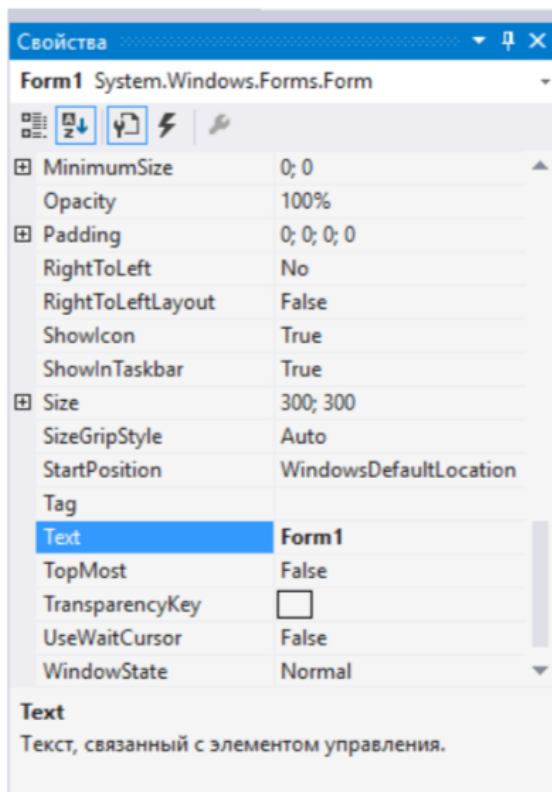
В его заголовке отображается имя проекта, над которым в данный момент идет работа. В верхней части главного окна находится строка меню и область отображения панелей инструментов. По умолчанию в области панелей инструментов отображаются панели Стандартная и Макет. Чтобы сделать доступными другие панели инструментов, надо в меню Вид выбрать команду Панели инструментов и в раскрывшемся списке сделать щелчок на имени нужной панели.

Центральную часть окна Visual Studio занимает окно конструктора (Designer) формы. В нем находится *форма* — заготовка главного окна приложения. Окно программы во время ее разработки принято называть формой. В левой вертикальной области главного окна находятся кнопки Источники данных, Обозреватель серверов и Панель элементов. Кнопка Панель элементов раскрывает окно Панель элементов.

В панели элементов находятся *элементы*. Элемент — это объект, реализующий некоторую функциональность. Например, в группе Стандартные элементы управления находятся элементы, реализующие пользовательский интерфейс (Label — область отображения текста; TextBox — поле редактирования текста; Button — командная кнопка).

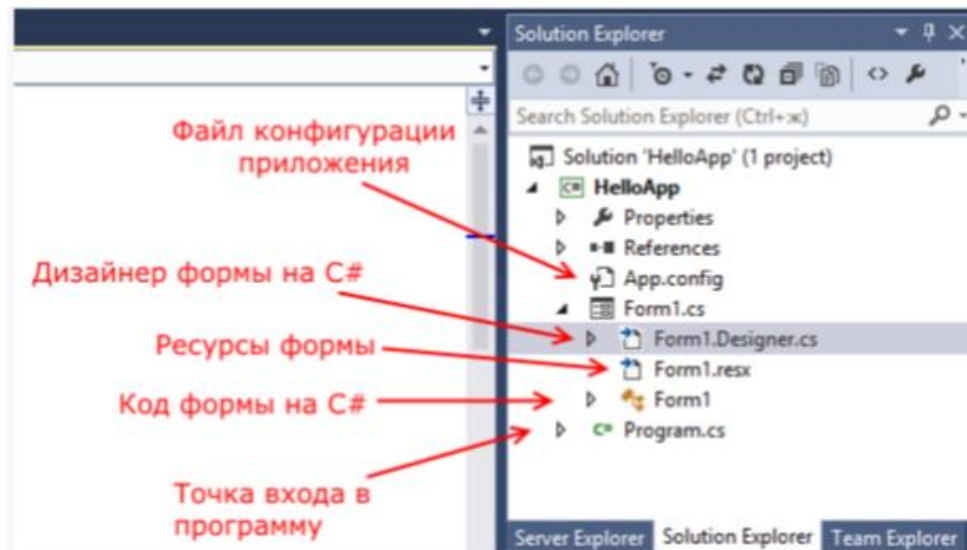


В окне Свойства, которое находится в правой нижней области окна Visual Studio, отображаются свойства объекта, выбранного в окне Конструктор, — элемента или, если ни один из элементов не выбран, формы. Обратите внимание, в верхней части окна Свойства отображается имя выбранного объекта и его тип (Form1 — имя формы (объекта), Form - тип).



Сделав в окне **Свойства** щелчок на кнопке **События** (на ней нарисована молния), можно увидеть *события*, которые способен воспринимать выбранный объект (компонент или форма).

Событие — это то, что происходит во время работы программы. Например, командная кнопка может реагировать на щелчок кнопкой мыши — событие Click.



Содержание работы:

Задание 1. Настроить форму.

Настройка формы начинается с настройки размера формы. С помощью мыши, «захватывая» одну из кромок формы или выделенную строку заголовка, отрегулируйте нужные размеры формы. Также для установки размеров формы можно использовать такие свойства как Width/Height или Size. Width/Height принимают числовые значения.

Для настройки будущего окна приложения задаются свойства формы. Для задания любых свойств формы и элементов управления на форме используется окно свойств. Новая форма имеет одинаковые имя (Name) и заголовок (Text) – Form1.

Для изменения заголовка щелкните кнопкой мыши на форме, в окне свойств найдите и щелкните мышкой на строчке с названием Text. В выделенном окне наберите «Тестовая форма». Для задания цвета окна используйте свойство BackColor, где выберите понравившийся цвет.


Задание 2. Размещение элементов управления

Для размещения различных элементов управления на форме используется панель элементов. Панель элементов содержит элементы управления, сгруппированные по типу. Каждую группу элементов управления можно свернуть, если она в настоящий момент не нужна.

Щелкните на элементе управления, который хотите добавить, а затем щелкните в нужном месте формы – элемент появится на форме. Элемент можно перемещать по форме, схватившись за него левой кнопкой мышки (иногда это можно сделать лишь за появляющийся при нажатии на элемент квадрат со стрелками). Если элемент управления позволяет изменять размеры, то на соответствующих его сторонах появятся белые кружки, ухватившись за которые и можно изменить размер. После размещения элемента управления на форме, его можно выделить щелчком мыши и при этом получить доступ к его свойствам в окне свойств.


Задание 3. Размещение строки ввода (TextBox)

Если необходимо ввести из формы в программу или вывести на форму информацию, которая вмещается в одну строку, используют окно однострочного редактора текста, представляемого элементом управления TextBox.

Выберите на панели элементов пиктограмму с названием TextBox  , щелкните мышью в том месте формы, где вы хотите ее поставить. Вставьте три элемента TextBox в форму. Захватывая их —мышью, отрегулируйте размеры окон и их положение. Обратите внимание на то, что теперь в тексте программы можно использовать переменные textBox1, textBox2 и textBox3, которые соответствуют каждому добавленному элементу управления. В каждой из этих переменных в свойстве Text будет содержаться строка символов (тип string) и отображаться в соответствующем окне TextBox.

С помощью инспектора объектов установите шрифт и размер символов, отражаемых в строке TextBox (свойство Font).

Задание 4. Размещение надписей (Label)

На форме могут размещаться пояснительные надписи. Для нанесения таких надписей на форму используется элемент управления Label. Выберите на панели элементов пиктограмму  , щелкните на ней —мышью. После этого в нужном месте формы щелкните мышью, появится надпись label1.

Проделайте это для четырех надписей. Для каждой надписи, щелкнув на ней мышью, отрегулируйте размер и, изменив свойство Text в окне свойств, введите строку, например «Введите значение X:», а также выберите размер символов (свойство Font).

Обратите внимание, что в тексте программы теперь можно обращаться к четырем новым переменным типа Label. В них хранятся пояснительные строки, которые можно изменять в процессе работы программы.

Задание 5. Разместите на форме четыре кнопки (Button). Сделайте на кнопках следующие надписи: «красный», «зеленый», «синий», «желтый».

ПРАКТИЧЕСКАЯ РАБОТА № 21

Тема: Создание проекта с использованием компонентов для работы с текстом

Цель работы: сформировать умения по использованию компонентов для работы с текстом в среде программирования Visual Studio, сформировать умения по созданию приложений с компонентами для работы с текстом в среде программирования Visual Studio

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Компоненты ввода — вывода данных можно условно разделить на несколько различных блоков: компоненты вывода текстовой информации на экран; однострочные поля ввода текстовой и числовой информации; многострочные поля ввода.

Для вывода определенной информации на экран, кроме уже ранее используемого компонента label, есть и другие компоненты. Текст, который будет отображен, можно задавать как на этапе разработки формы, так и в процессе выполнения программы, присвоив значение свойству Text.

Написание программы обработки события

С каждым элементом управления на форме и с самой формой могут происходить события во время работы программы. Например, с кнопкой может произойти событие – нажатие кнопки, а с окном, которое проектируется с помощью формы, может произойти ряд событий: создание окна, изменение размера окна, щелчок мыши на окне и т. п. Эти события могут обрабатываться в программе. Для обработки таких событий необходимо создать обработчики события – специальный метод.

Для создания обработчика события существует два способа. Первый способ – создать обработчик для события по умолчанию (обычно это самое часто используемое событие данного элемента управления). Например, для кнопки таким образом создается обработчик события нажатия.

Динамическое изменение свойств

Свойства элементов на окне могут быть изменены динамически во время выполнения программы. Например, можно изменить текст надписи или цвет формы. Изменение свойств происходит внутри обработчика события (например, обработчика события нажатия на кнопку). Для этого используют оператор присвоения вида:

```
<имя элемента>.<свойство> = <значение>;  
label1.Text = "Привет";
```

<Имя элемента> определяется на этапе проектирования формы, при размещении элемента управления на форме. Например, при размещении на форме ряда элементов TextBox, эти элементы получают имена textBox1, textBox2, textBox3 и т. д., которые могут быть заменены в окне свойств в свойстве (Name) для текущего элемента. Допускается использование латинских или русских символов, знака подчеркивания и цифр (цифра не должна стоять в начале идентификатора). Список свойств для конкретного элемента можно

посмотреть в окне свойств, а также в приложении к данным методическим указаниям.

Если требуется изменить свойства формы, то никакое имя элемента перед точкой вставлять не нужно, как и саму точку. Например, чтобы задать цвет формы, нужно просто написать: `BackColor = Color.Green;`

Содержание работы:

Задание 1. Написание программы обработки события нажатия кнопки

1. Запустите Visual Studio 2022. Создайте новый проект.

2. Поместите на форму кнопку, которая описывается элементом управления `Button`. С помощью окна свойств измените заголовок (`Text`) на слово «Привет» или другое по вашему желанию. Отрегулируйте положение и размер кнопки.


3. После этого два раза щелкните мышью на кнопке, появится текст программы:

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Это и есть обработчики события нажатия кнопки. Вы можете добавлять свой код между скобками `{ }`. Например, наберите:

```
MessageBox.Show("Привет, " + textBox1.Text + "!");
```

Задание 2. Написание программы обработки события загрузки формы

Второй способ создания обработчика события заключается в выборе соответствующего события для выделенного элемента на форме. При этом используется окно свойств и его закладка . Рассмотрим этот способ. Выделите форму щелчком по ней, чтобы вокруг нее появилась рамка из точек. В окне свойств найдите событие `Load`. Щелкните по данной строчке дважды мышью. Появится метод:

```
private void Form1_Load(object sender, EventArgs e)
{
}
```

Между скобками `{ }` вставим текст программы:
`BackColor = Color.AntiqueWhite;`


Каждый элемент управления имеет свой набор обработчиков событий, однако некоторые из них присущи большинству элементов управления.

Наиболее часто применяемые события описаны ниже:

- **Activated:** форма получает это событие при активации.
- **Load:** возникает при загрузке формы. В обработчике данного события следует задавать действия, которые должны происходить в момент создания формы, например установка начальных значений.
- **KeyPress:** возникает при нажатии кнопки на клавиатуре. Параметр `e.KeyChar` имеет тип `char` и содержит код нажатой клавиши (клавиша `Enter` клавиатуры имеет код `#13`, клавиша `Esc` – `#27` и т. д.). Обычно это событие используется в том случае, когда необходима реакция на нажатие одной из клавиш.
- **KeyDown:** возникает при нажатии клавиши на клавиатуре. Обработчик этого события получает информацию о нажатой клавише и состоянии клавиш `Shift`, `Alt`

и Ctrl, а также о нажатой кнопке мыши. Информация о клавише передается параметром `e.KeyCode`, который представляет собой перечисление `Keys` с кодами всех клавиш, а информацию о клавишах-модификаторах Shift и др. можно узнать из параметра `e.Modifiers`.

- **KeyUp**: является парным событием для **KeyDown** и возникает при отпускании ранее нажатой клавиши.
- **Click**: возникает при нажатии кнопки мыши в области элемента управления.
- **DoubleClick**: возникает при двойном нажатии кнопки мыши в области элемента управления.

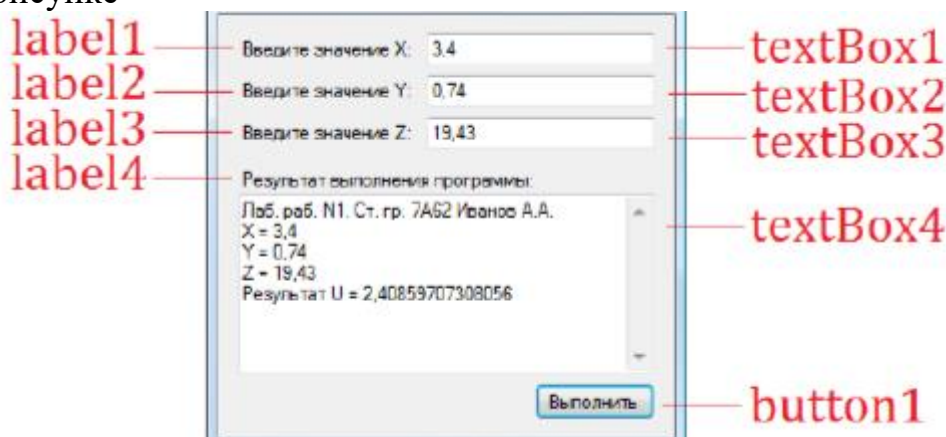
Важное примечание! Если какой-то обработчик был добавлен по ошибке или больше не нужен, то для его удаления нельзя просто удалить программный код обработчика! Сначала нужно удалить строку с именем обработчика в окне свойств на закладке . В противном случае программа может перестать компилироваться и даже отображать форму в дизайнера Visual Studio.

Задание 3. Составить программу вычисления для заданных значений x , y ,

$$u = \operatorname{tg}^2(x + y) - e^{y-z} \sqrt{\cos x^2 + \sin z^2}$$

z арифметического выражения:

Панель диалога программы организовать в виде, представленном на рисунке



label1 — Введите значение X: 3,4
label2 — Введите значение Y: 0,74
label3 — Введите значение Z: 19,43
label4 — Результат выполнения программы:
Лоб. раб. N1. Ст. гр. 7A62 Иванов А.А.
X = 3,4
Y = 0,74
Z = 19,43
Результат U = 2,40859707308056
Выполнить

textBox1
textBox2
textBox3
textBox4
button1

Для вывода результатов работы программы в программе используется текстовое окно, которое представлено обычным элементом управления. После установки свойства `Multiline` в `true` появляется возможность растягивать элемент управления не только по горизонтали, но и по вертикали. А после установки свойства `ScrollBars` в значение `Both` в окне появится вертикальная, а при необходимости и горизонтальная полосы прокрутки.

Информация, которая отображается построчно в окне, находится в массиве строк `Lines`, каждая строка которого имеет тип `string`. Однако нельзя напрямую обратиться к этому свойству для добавления новых строк, поскольку размер массивов в C# определяется в момент их инициализации. Для добавления нового элемента используется свойство `Text`, к текущему содержимому которого можно добавить новую строку:

```
textBox4.Text += Environment.NewLine + "Привет";
```

В этом примере к текущему содержимому окна добавляется символ перевода курсора на новую строку (который может отличаться в разных операционных системах и потому представлен свойством класса Environment) и сама новая строка. Если добавляется числовое значение, то его предварительно нужно привести в символьный вид методом ToString().

Работа с программой происходит следующим образом. Нажмите (щелкните мышью) кнопку «Выполнить». В окне textBox4 появляется результат. Измените исходные значения x, y, z в окнах textBox1– textBox3 и снова нажмите кнопку «Выполнить» – появятся новые результаты.

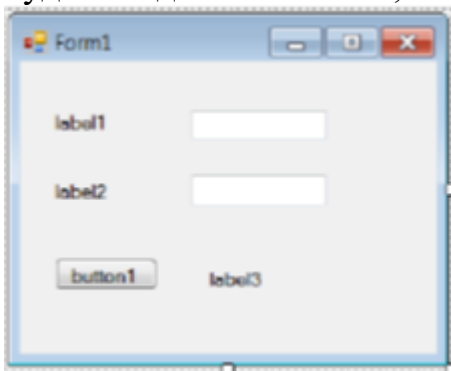
Полный текст программы имеет следующий вид:

```
using System;
using System.Windows.Forms;
namespace MyFirstApp {
public partial class Form1 : Form {
public Form1() {
InitializeComponent(); }
private void Form1_Load(object sender,
EventArgs e) {
// Начальное значение X
textBox1.Text = "3,4";
// Начальное значение Y
textBox2.Text = "0,74";
// Начальное значение Z
textBox3.Text = "19,43"; }
private void button1_Click(object sender, EventArgs e) {
// Считывание значения X
double x = double.Parse(textBox1.Text);
// Вывод значения X в окно
textBox4.Text += Environment.NewLine + "X = " + x.ToString();
// Считывание значения Y
double y = double.Parse(textBox2.Text);
// Вывод значения Y в окно
textBox4.Text += Environment.NewLine + "Y = " + y.ToString();
// Считывание значения Z
double z = double.Parse(textBox3.Text);
// Вывод значения Z в окно
textBox4.Text += Environment.NewLine + "Z = " + z.ToString();
// Вычисляем арифметическое выражение
double a = Math.Tan(x + y) * Math.Tan(x + y);
double b = Math.Exp(y - z);
double c = Math.Sqrt(Math.Cos(x*x) + Math.Sin(z*z));
double u = a - b * c;
// Выводим результат в окно
textBox4.Text += Environment.NewLine + "Результат U = " + u.ToString(); }}}}
```

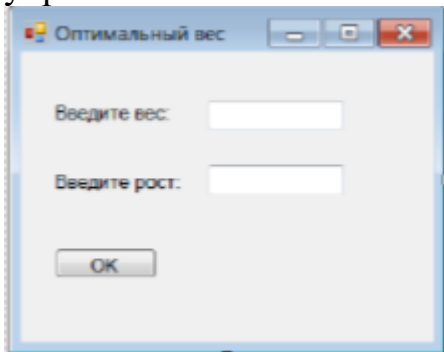
Если просто скопировать этот текст и заменить им то, что было в редакторе кода Visual Studio, то программа не заработает. Правильнее будет создать обработчики событий Load у формы и Click у кнопки и уже в них вставить соответствующий код.

Задание 4. Разработать программу с помощью, которой пользователь, введя свой рост и фактический вес, мог бы определить худой он или полный, и на сколько ему нужно поправиться или похудеть. Для разработки программы воспользуйтесь тем, что оптимальный вес человека определяется так: рост человека минус 100. Если фактический вес человека меньше оптимального, то человек худой, и наоборот, если больше, то нужно похудеть.

1. Создайте в папке своей группы новую папку и назовите её Weight.
2. Создайте новое Приложение Windows Forms. Имя проекта и приложения – Weight.
3. Разместите на форме компоненты в соответствии с рисунком. В TextBox1 будет вводиться вес в кг, а в TextBox 2 – рост в см.



4. Задайте для элементов управления значение свойства Text в соответствии с рисунком. Для label3 значение свойства Text оставьте пустым, для элементов управления TextBox1 и TextBox2 значение свойства Text также оставьте пустым.



5. Выделите кнопку button1, дважды щелкните по ней мышью. Оказавшись в коде программы, а точнее, в заготовке процедуры кнопки button1 заполните её следующим кодом:

```

private void button1_Click(object sender, EventArgs e)
{
    int faktW = Convert.ToInt32(textBox1.Text);
    int Rost = Convert.ToInt32(textBox2.Text);
    int optW = Rost - 100;
    int Delta = Math.Abs(faktW - optW);
    if (optW == faktW)
    {
        label3.Text = "Ваш вес оптимален";
    }
    else
    {
        if (optW > faktW)
        {
            label3.Text = "Вам надо поправиться на " + Delta + " кг";
        }
        else
        {
            label3.Text = "Вам надо похудеть на " + Delta + " кг";
        }
    }
}
}

```

Вес и рост заданы в объектах textBox1 и textBox2 через свойство Text, а это строковое значение – тип string. Следовательно, эти значения надо преобразовать в целое число. Встроенный класс Convert содержит нужные методы преобразования, в частности метод ToInt32.

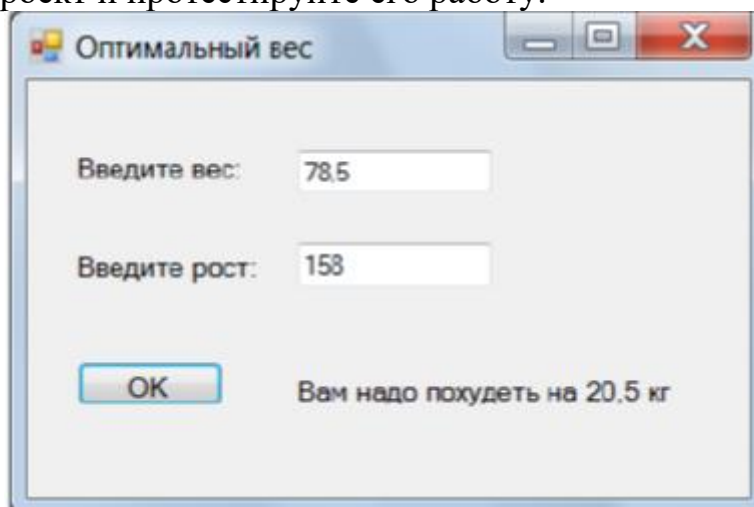
6. Сохраните проект и протестируйте работу приложения.

7. Усовершенствуйте программу так, чтобы можно было бы вводить десятичные величины. Для этого задайте тип переменных не Int, а Double.

8. Метод ToDouble встроенного класса Convert преобразует помещенную в скобки переменную типа string в переменную вещественного типа. Замените в тексте программного кода ToInt32 на ToDouble.

9. Озаглавьте окно проекта Оптимальный Вес.

10. Сохраните проект и протестируйте его работу.



ПРАКТИЧЕСКАЯ РАБОТА № 22

Тема: Создание проекта с использованием компонентов для работы с текстом

Цель работы: сформировать умения по использованию компонентов для работы с текстом в среде программирования Visual Studio, сформировать умения по созданию приложений с компонентами для работы с текстом в среде программирования Visual Studio

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Разместите на форме две кнопки (Button) и одну метку (Label). Сделайте на кнопках следующие надписи: «привет», «до свидания». Создайте обработчики события нажатия на данные кнопки, которые будут менять текст метки на слова, написанные на кнопках. Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы».

Задание 2. Разместите на форме ряд кнопок (Button) напротив каждой поле ввода (TextBox) и одну метку (Label). Создайте обработчики события нажатия на данные кнопки, которые будут менять текст в метке. Текст в метке берется из поля ввода напротив нажимаемой кнопки.

Задание 3. Разработка проекта, содержащего форму, на которую помещены две кнопки и три элемента управления label. При нажатии на первую кнопку в элементе управления label1 выдается ваша фамилия, в label2 – имя, в label3 – отчество, при нажатии на вторую – надписи меняются местами.

Задание 4. Разработка проекта, содержащего форму, на которую помещена кнопка управления, label и 2 элемента textBox. При нажатии на кнопку содержимое элементов управления textBox1 и textBox2 выдается в элементе управления label1.

ПРАКТИЧЕСКАЯ РАБОТА № 23

Тема: Создание проекта с использованием компонентов для работы с текстом

Цель работы: сформировать умения по использованию компонентов для работы с текстом в среде программирования Visual Studio, сформировать умения по созданию приложений с компонентами для работы с текстом в среде программирования Visual Studio

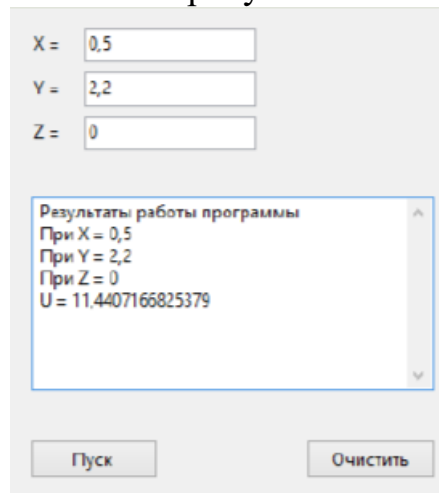
Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Ввести три числа – x, y, z. Вычислить

$$U = \begin{cases} y \cdot \sin(x) + z, & \text{при } z - x = 0 \\ y \cdot e^{\sin(x)} - z, & \text{при } z - x < 0 \\ y \cdot \sin(\sin(x)) + z, & \text{при } z - x > 0 \end{cases}$$

Создайте форму, в соответствии с рисунком



Разместите на форме элементы Label, TextBox и Button. Поле для вывода результатов также является элементом TextBox с установленным в true свойством Multiline и свойством ScrollBars установленным в Both. Текст обработчика события нажатия на кнопку «Пуск» приведен ниже.

```
private void button1_Click(object sender, EventArgs e) {  
    // Получение исходных данных из TextBox  
    double x = Convert.ToDouble(textBox2.Text);  
    double y = Convert.ToDouble(textBox1.Text);  
    double z = Convert.ToDouble(textBox3.Text);  
    // Ввод исходных данных в окно результатов  
    textBox4.Text = "Результаты работы программы " + Environment.NewLine;  
    textBox4.Text += "При X = " + textBox2.Text + Environment.NewLine;  
    textBox4.Text += "При Y = " + textBox1.Text + Environment.NewLine;  
    textBox4.Text += "При Z = " + textBox3.Text + Environment.NewLine;  
    // Вычисление выражения  
    double u;
```



```

if ((z - x) == 0)
    u = y * Math.Sin(x) * Math.Sin(x) + z;
else
    if ((z - x) < 0)
        u = y * Math.Exp(Math.Sin(x)) - z;
    else
        u = y * Math.Sin(Math.Sin(x)) + z;
// Вывод результата
textBox4.Text += "U = " + u.ToString() +
    Environment.NewLine; }

```

Запустите программу и убедитесь в том, что все ветви алгоритма выполняются правильно.

Задание 2. Разработать программу, которая при нажатии на кнопку «Нажать» выводит сообщение «Я студент специальности Информационные системы и программирование», а затем при повторном нажатии на эту же кнопку сообщение исчезает. При повторном выводе цвет надписи должен принять любой цвет, кроме черного.

Задание 3. Разработка проекта, содержащего форму, на которую помещена кнопка и элементы управления label. При нажатии на одну кнопку меняется цвет фона, а на другую – цвет текста в label.

ПРАКТИЧЕСКАЯ РАБОТА № 24

Тема: Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени

Цель работы: получить навыки управления данными даты и времени.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Для работы с датами в Windows Forms имеются элементы `DateTimePicker` и `MonthCalendar`. Использование элемента управления, отображающего календарь, значительно упрощает для пользователя выбор даты. Кроме того, такие элементы управления гарантируют, что дата будет отформатирована правильно. Календарь можно отобразить с помощью элемента управления `MonthCalendar` или `DateTimePicker`.

Элемент управления `MonthCalendar` позволяет отображать календарь для одного или нескольких месяцев. При этом пользователи могут выбирать отдельную дату или диапазон дат.

Элемент управления `DateTimePicker` имеет два состояния. По умолчанию элемент управления `DateTimePicker` выглядит как текстовое поле с раскрывающимся списком в виде стрелки. Когда пользователь нажимает на стрелку раскрывающегося списка, появляется календарь. При использовании этого элемента управления пользователь может выбрать только одну дату. Элемент управления `DateTimePicker` также позволяет отображать время вместо дат.

Процесс, используемый для извлечения даты из этих элементов управления, зависит от конкретного используемого элемента. Используйте свойство `Start` для элемента управления `MonthCalendar` и свойство `Value` для элемента управления `DateTimePicker`.

Элемент управления `MonthCalendar` позволяет отображать на экране одновременно до 12 месяцев. По умолчанию в этом элементе управления отображается только один месяц, однако имеется возможность указать количество месяцев, которые будут отображаться на экране, и их размещение в данном элементе управления. Чтобы обеспечить достаточное количество места в форме для новой размерности, при изменении диапазона календаря изменяются размеры элемента управления.

Константы для указания формата даты

Константа	Описание	Пример
<code>DateFormat.GeneralDate</code>	Отображает дату, время или оба значения. Если присутствует дата, она отображается в кратком формате. Если присутствует время, оно отображается в полном формате. Если присутствует и время, и дата, отображаются обе части.	22/11/1963 12:00:00 PM
<code>DateFormat.LongDate</code>	Отображает дату в полном формате, который	Пятница, 22

	определяется установленными на компьютере региональными параметрами.	ноября, 1963
DateFormat.ShortDate	Отображает дату в кратком формате, который определяется установленными на компьютере региональными параметрами.	11/22/1963
DateFormat.LongTime	Отображает время в полном формате, который определяется установленными на компьютере региональными параметрами.	12:00:00 PM
DateFormat.ShortTime	Отображает время в 24-часовом формате (чч:мм).	12:00

Содержание работы:

Задание 1. Создать календарь.

1. Создать новый проект командой Создать проект
2. Выберите элемент Приложение WindowsForms и нажмите кнопку ОК.
3. Добавьте в форму элемент Label, оставив имя по умолчанию Label1.
4. Удалите текст из свойства Text элемента управления Метка.
5. Добавьте в форму элемент управления MonthCalendar, оставив имя по умолчанию MonthCalendar1.
6. Дважды щелкните элемент управления MonthCalendar, чтобы открыть обработчик событий по умолчанию в редакторе кода.
7. В обработчике событий MonthCalendar1_DateChanged добавьте следующий код для добавления элементов в список.

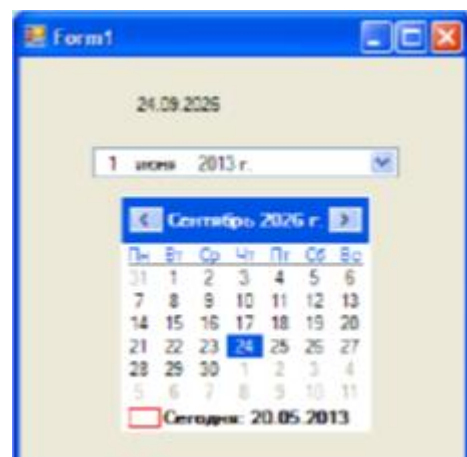
```
Me.Label1.Text = CStr(Me.MonthCalendar1.SelectionRange.Start)
```

8. Вернитесь в режим конструктора и добавьте в форму элемент управления DateTimePicker, оставив имя по умолчанию DateTimePicker1.
9. Дважды щелкните элемент управления DateTimePicker, чтобы открыть обработчик событий по умолчанию в редакторе кода.
10. В обработчике событий DateTimePicker1_ValueChanged добавьте следующий код для добавления элементов в список.

```
Me.Label1.Text = CStr(Me.DateTimePicker1.Value)
```

11. Нажмите клавишу F5 для запуска программы.
12. Когда появится форма, выберите дату в элементе управления MonthCalendar и убедитесь, что она отображается в метке.
13. Щелкните стрелку раскрывающегося списка элемента управления DateTimePicker и выберите дату.

Дата и время отображаются в метке.



Задание 2. Извлечение диапазона дат из элемента управления календарем месяца.

Диапазон дат, выбранных в элементе управления MonthCalendar, можно извлечь с помощью свойств Start и End свойства SelectionRange. По умолчанию максимальное число дней, которые можно выбрать, равно 7, но при необходимости этот параметр можно изменить, установив значение свойства MaxSelectionCount. Чтобы определить, выбран ли диапазон дат, просто проверьте, совпадают ли даты начала и конца.

1. Замените код в обработчике событий MonthCalendar1_DateChanged следующим. Этот код устанавливает максимальное число дней (две недели), которые могут быть выбраны в элементе управления. Он отображает дату начала в метке, если выбран только один день, и отображает диапазон дат при выборе диапазона дней в элементе управления MonthCalendar.

```
Me.MonthCalendar1.MaxSelectionCount = 14

If Me.MonthCalendar1.SelectionRange.Start = _
    Me.MonthCalendar1.SelectionRange.End Then

    Me.Label1.Text = CStr(Me.MonthCalendar1.SelectionStart)

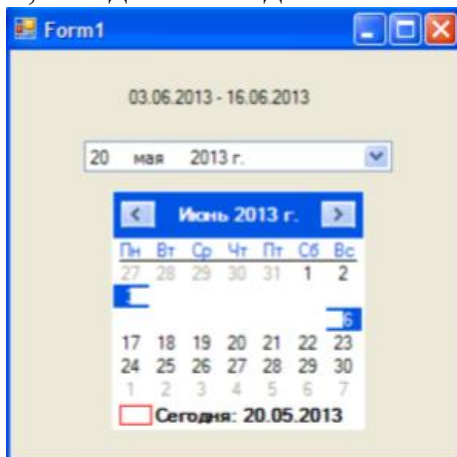
Else

    Me.Label1.Text = Me.MonthCalendar1.SelectionRange.Start & _
        " - " & Me.MonthCalendar1.SelectionRange.End

End If
```

2. Запустите проект.

3. Когда появится форма, выберите диапазон дат в элементе управления MonthCalendar и убедитесь, что диапазон дат появился в метке.



Задание 3. Форматирование даты в метке

Даты, возвращаемые элементами управления MonthCalendar и DateTimePicker, можно форматировать с помощью функции FormatDateTime. Существует несколько констант, которые можно использовать для указания формата даты.

1. Замените код в обработчике событий MonthCalendar1_DateChanged следующим. Этот код форматирует дату, возвращаемую в полном формате.

```

Me.MonthCalendar1.MaxSelectionCount = 14

If Me.MonthCalendar1.SelectionRange.Start = _
    Me.MonthCalendar1.SelectionRange.End Then

    Me.Label1.Text = FormatDateTime( _
        Me.MonthCalendar1.SelectionStart, _
        DateFormat.LongDate)
Else
    Me.Label1.Text = FormatDateTime( _
        Me.MonthCalendar1.SelectionRange.Start, _
        DateFormat.LongDate) & " - " & FormatDateTime( _
        Me.MonthCalendar1.SelectionRange.End, DateFormat.LongDate)
End If

```

2. Замените код в обработчике событий DatePicker1_ValueChanged следующим. Этот код форматирует дату, возвращаемую в полном формате.

```

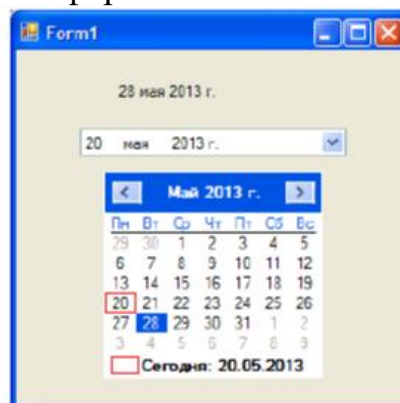
Me.Label1.Text = FormatDateTime(Me.DateTimePicker1.Value, _
    DateFormat.LongDate)

```

3. Запустите программу.

4. Когда появится форма, выберите дату или диапазон дат в элементе управления MonthCalendar. Убедитесь, что дата или диапазон дат отображается в метке в полном формате.

5. Выберите дату в элементе управления DateTimePicker и убедитесь, что дата в метке отображается в полном формате.



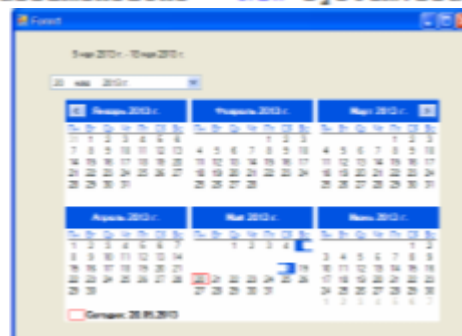
Задание 4. Отобразить несколько месяцев в календаре

Задайте для свойства CalendarDimensions значение, равное числу месяцев, отображаемых по горизонтали и вертикали.

```

MonthCalendar1.CalendarDimensions = New System.Drawing.Size(3, 2)

```



ПРАКТИЧЕСКАЯ РАБОТА № 25

Тема: Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени

Цель работы: получить навыки управления данными даты и времени.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Написать программу Birthday

В программе Birthday элементы управления DateTimePicker и Button используются, чтобы выяснить у пользователя дату его рождения и показать эту информацию в окне сообщения.

1. В области элементов выберите элемент управления Button, и ниже объекта выбора даты и времени добавьте объект кнопки. Эта кнопка будет использована для показа дня рождения и для проверки правильности работы объекта выбора даты и времени.

2. В окне Свойства(Properties) измените свойство Text объекта кнопки на Показать день моего рождения.

3. Дважды щелкните мышью на объекте кнопки, а потом наберите следующий фрагмент программы между операторами Private Sub и End Sub в процедуре события Button1_Click:

```
MsgBox("Ваш день рождения " & DateTimePicker1.Text)
MsgBox("День года: " & DateTimePicker1.Value.DayOfYear.ToString())
MsgBox("Сейчас: " & DateTimePicker1.Value.TimeOfDay.ToString())
```

Этот фрагмент программы показывает три последовательных окна сообщения (небольшие диалоговые окна), которые содержат информацию из объекта календаря. В первой строке используется свойство Text календаря для вывода информации о дате рождения, которую пользователь выберет в этом объекте после запуска программы.

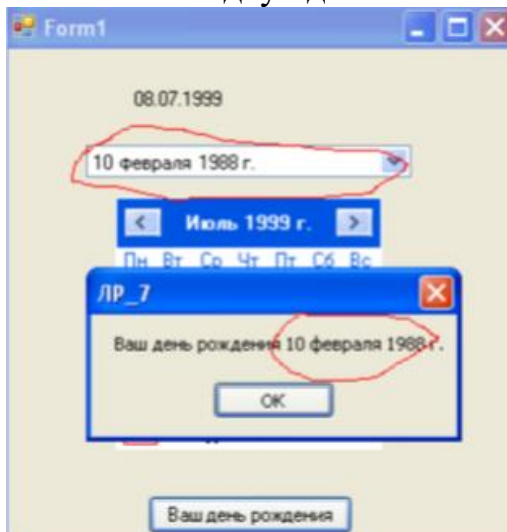
Функция MsgBox кроме текстового значения из свойства Text календаря показывает строку "Ваш день рождения". Эти два текстовых элемента объединяются в строку с помощью оператора конкатенации (слияния) строк &. Во второй строке `DateTimePicker1.Value.DayOfYear.ToString()` объект календаря используется для вычисления дня года, отсчитывая с 1 января. Это делается с помощью свойства DayOfYear и метода ToString, который переводит числовой результат вычисления даты в текстовое значение, которое гораздо проще показать с помощью функции MsgBox. В третьей строке фрагмента, после перевода значения в строковое (или текстовое) представление, в окне сообщения показывается информация о точном времени.

Запуск программы Birthday

1. На стандартной панели инструментов нажмите кнопку Start (Начать). Программа Birthday запустится в среде разработки. В окне объекта выбора даты и времени появится текущая дата.

2. Нажмите стрелку раскрывающегося списка, чтобы вывести на экран представление этого объекта в виде календаря. Форма будет выглядеть как на следующей иллюстрации.

3. Выберите в элементе DateTimePicker1 число, месяц и год Вашего рождения, пользуясь стрелками прокрутки. Нажмите кнопку Показать день моего рождения. Программа исполнит введенный вами код и покажет окно с сообщением, содержащим день и дату вашего рождения. Обратите внимание на соответствие двух дат.

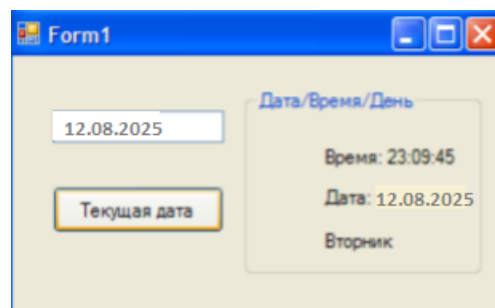


4. В окне сообщения нажмите ОК. Появится второе окно сообщения, указывающее, в какой день года вы родились.

5. Нажмите ОК, чтобы показать последнее окно сообщения. Появятся текущие дата и время. Вы обнаружите, что объект выбора даты и времени очень удобен - он не только помнит новую, введенную вами информацию о дате или времени, но также отслеживает текущие дату и время и может показывать эту информацию в различных форматах.

Совет. Чтобы настроить объект выбора даты и времени для показа времени, а не даты, установите свойство Format этого объекта равным Time.

Задание 2. Реализовать программу, которая будет узнавать текущую дату и время.



ПРАКТИЧЕСКАЯ РАБОТА № 26

Тема: Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени

Цель работы: получить навыки управления данными даты и времени.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Разработка проекта, содержащего форму, на которую помещена кнопка и элементы управления `textBox` и `dateTimePicker`. При нажатии на кнопку содержимое элемента управления `dateTimePicker` выдается в `textBox`.

Задание 2. Разработка проекта, содержащего форму, на которую помещена кнопка, элементы управления `label` и `monthCalendar`. При нажатии на кнопку в элементе управления `label1` выдается выбранная дата, а в `label2` день недели.

Задание 3. Разработка проекта, содержащего форму, на которую помещена кнопка и элементы управления `label`, в котором содержится слово «сегодня» и `dateTimePicker`. При нажатии на кнопку по выбранной дате в `dateTimePicker` в `label1` добавляется день недели, а в `label2` текущее время.

ПРАКТИЧЕСКАЯ РАБОТА № 27

Тема: Создание проекта с использованием полос прокрутки для ввода информации

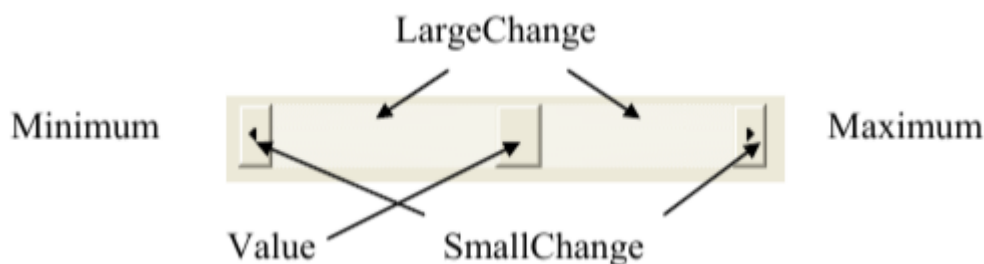
Цель работы: получить навыки добавления в программы полос прокруток для ввода данных на языке C#

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Горизонтальные и вертикальные полосы прокрутки широко используются в приложениях Windows. Они обеспечивают интуитивный способ передвижения по спискам информации и позволяют делать поля для ввода данных очень большими. Полоса прокрутки состоит из трех областей, которые нажимаются или перемещаются для изменения значения полосы прокрутки.

Рассмотрим работу элемента на примере горизонтальной полосы прокрутки. Нажатие левой стрелки уменьшает значение положения бегунка на минимальное. Нажатие правой стрелки увеличивает значение положения бегунка на минимальное. При нажатии курсором мыши в области полосы прокрутки значение положения бегунка изменяется на величину, большую, чем значение при нажатии на кнопки «влево» и «вправо». При использовании реквизитов полос прокрутки мы можем полностью определять, как работает каждый из них. Позиция бегунка — единственная выходная информация из полосы прокрутки.



LargeChange — значение, которое добавляется или вычитается из значения текущего положения бегунка; это величина, на которую изменяется положение бегунка при нажатии курсора в области полосы прокрутки.

Maximum — значение горизонтальной полосы прокрутки в крайнем правом положении и значение вертикальной полосы прокрутки в крайнем нижнем положении. Может принимать значения от -32 768 до 32 767

Minimum — другое предельное значение — для горизонтальной полосы прокрутки в крайнем левом положении и для вертикальной полосы прокрутки в крайнем верхнем. Может принимать значения от -32 768 до 32 767

SmallChange — значение, которое добавляется или вычитается из значения текущего положения бегунка; значение, на которое изменяется положение полосы прокрутки, когда нажата любая из стрелок прокрутки.

Value — текущая позиция бегунка внутри полосы прокрутки. При изменении значения свойства Value бегунок перемещается в соответствующую позицию.

События полосы прокрутки:

Change — событие, генерируемое после того, как позиция бегунка изменилась. Это событие используется для получения нового значения, чтобы считать новое значение положения бегунка после любых действий на полосе прокрутки.

Scroll — событие, вызываемое непрерывно всякий раз, когда бегунок передвигается.

Содержание работы:

Задание 1. Создание полос прокруток на форме.

1. Запустите Visual Studio. Создайте новый Windows Forms проект с именем ScrollApp. Поместите на форму элемент VScrollBar и четыре элемента Label. Сделайте это так, как показано на рисунке



2. Установите для компонент свойства в соответствии с приведенным ниже списком:

Form1: Text – Измерение температуры

vScrollBar1: LargeChange – 10; Maximum – 60; Minimum – 20; SmallChange -1; Value - 32

label1: Text – Фаренгейт; Font Size – 10; Font Bold – true

label2: BackColor – White; Text – 32; Font Size – 14; Font Bold – true; Name - labelFarTemp

label3: Text – Цельсий; Font Size – 10; FontBold - true

label4: BackColor – White; Text – 0; Font Size – 14; Font Bold – true; Name – labelCTemp

Обратите внимание, что значения температур проинициализированы 32 и 0. Когда вы сделаете свою форму, она должна иметь вид, представленный на рисунке



3. Добавим обработчик события Scroll в код программы. Для этого в окне свойств на закладке событий для элемента vScrollBar1 щелкните два раза по событию Scroll. В код программы добавится обработчик события Scroll:

```
private void vScrollBar1_Scroll(object sender,
System.Windows.Forms.ScrollEventArgs e) {
}
```

4. Добавьте в функцию vScrollBar1_Scroll код:

```
private void vScrollBar1_Scroll(object sender,
System.Windows.Forms.ScrollEventArgs e) {
    labelFarTemp.Text =
vScrollBar1.Value.ToString() ;
    labelCTemp.Text =Convert.ToString(((int)(((double)vScrollBar1.Value - 32)/9*5));
}
```

Этот код переводит значение Value полосы прокрутки во время изменения положения бегунка и определяет значение температуры по шкале Фаренгейта. Затем вычисляется значение температуры по Цельсию и отображаются оба значения.

5. Откомпилируйте и запустите программу. Вы увидите, что при изменении положения бегунка полосы прокрутки изменяются значения температур. Попробуйте найти точку, в которой значение температуры по Цельсию равно значению температуры по Фаренгейту.

Задание 2. Создайте приложение, на форме которого разместите три полосы прокрутки для управления цветом (красным, зеленым, синим), который используется как фон формы. Начальные положения бегунков полос прокруток должны соответствовать заданному в дизайнера цвету фона формы.

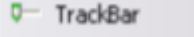
ПРАКТИЧЕСКАЯ РАБОТА № 28

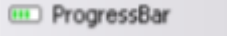
Тема: Создание проекта с использованием полос прокрутки для ввода информации


Цель работы: получить навыки добавления в программы бегунок, индикатор прогресса и регулятора числовых значений на языке C#

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Бегунок (TrackBar) . Типичным примером применения элемента TrackBar является регулятор уровня громкости в панели Windows. TrackBar может использоваться в различных режимах: в горизонтальном или вертикальном положении, с включенными черточками или без.

Индикатор прогресса (ProgressBar)  чаще всего используют для отображения степени завершенности той или иной задачи.

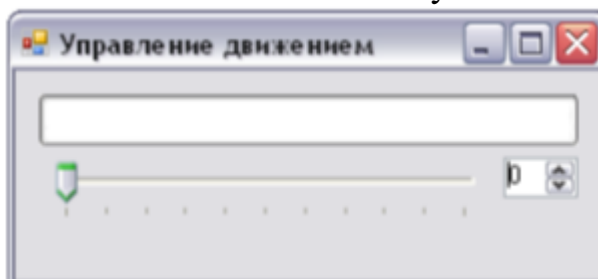
Регулятор численных значений (NumericUpDown)  позволяет без помощи клавиатуры вводить численные значения в поле для ввода. Данный элемент управления имеет три возможности для ввода данных: щелчок мышкой на указатели вверх-вниз, использование кнопок вверх-вниз на клавиатуре или ввод данных в поле ввода.

Содержание работы:

Задание 1. Написать приложение, в котором бегунок и элемент управления NumericUpDown управляют индикатором прогресса. Дополнительное условие: бегунок и NumericUpDown должны работать синхронно. То есть, при изменении значения одного элемента, значение другого должно изменяться автоматически на ту же величину.

- 1.Создайте новый Windows Application проект под названием TestIndicator. Сохраните его в вашу папку.
2. Переименуйте файл Form1.cs в TestIndicatorForm.cs.
- 3.Теперь добавьте на вашу форму следующие элементы управления: TrackBar, ProgressBar, NumericUpDown.

Примерное размещение элементов изображено на рисунке. Вы можете расположить элементы по-своему.



- 4.Измените свойства элементов управления. Свойства элемента TrackBar1: Maximum —100; TickStyle — Both. При этом TrackBar изменит свой вид. Бегунок примет прямоугольную форму, а полосы появятся и сверху, и снизу от него. Это результат изменения свойства TickStyle. Данное свойство определяет месторасположение черточек элемента управления. В этом случае было выбрано значение Both (с обеих сторон). Кроме того, возможны расположения только

сверху, только снизу или вообще без черточек. Свойства `Minimum` и `Maximum` задают минимальное и максимальное значения, до которых может изменяться `TrackBar`. В данном случае мы задали максимальное значение 100, а минимальное 0 (оставили по умолчанию). То есть, когда бегунок будет находиться в крайнем левом положении, значение его свойства `Value` будет равно 0, а при нахождении бегунка в крайнем правом положении свойство `Value` будет иметь значение 100.

5. Свойства объекта `numericUpDown1` оставим по умолчанию. Элемент управления `NumericUpDown` также имеет свойства `Minimum` и `Maximum`. И по умолчанию, свойство `Minimum` равно 0, свойство `Maximum` равно 100. Это соответствует параметрам, установленным для объекта `trackBar1`. Очень важное свойство компонента `NumericUpDown` — `DecimalPlaces`. Оно определяет количество знаков после запятой. В нашем примере это свойство необходимо оставить по умолчанию равным 0, однако при необходимости получить большую точность, чем целое значение, следует устанавливать значение свойства в соответствии с заданной точностью.

6. Измените значение свойства `Text` формы на «Управление движением». Проанализируйте код программы.

7. Элементы `numericUpDown1` и `trackBar1` являются управляющими, а элемент `progressBar1` — управляемым. Давайте зададим обработчики событий для управления индикатором прогресса. Итак, компонент `TrackBar` имеет событие `Scroll`, которое предназначено для обработки перемещения указателя бегунка. Создайте функцию обработчик для события `Scroll`, щелкнув два раза указателем мыши по имени события в окне свойств. В код программы добавится функция с именем `trackBar1_Scroll`. Измените ее код так, как показано ниже:

```
private void trackBar1_Scroll(object sender, System.EventArgs e) {  
    int Value = trackBar1.Value;  
    numericUpDown1.Value = Value;  
    progressBar1.Value = Value; }  
}
```

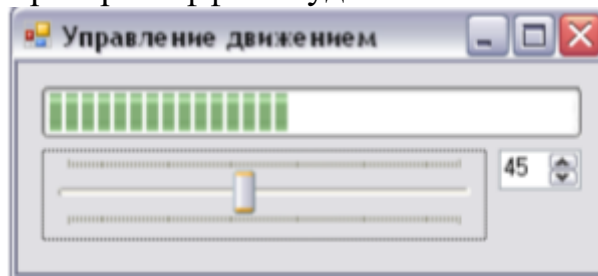
Теперь при движении курсора бегунка будут изменяться положение индикатора прогресса и значение элемента `numericUpDown1`. Однако это еще не полная синхронность работы элементов, потому что управление должно вестись из двух элементов: бегунка и регулятора числовых значений (`NumericUpDown`), а у нас сейчас управление возможно лишь от бегунка.

8. Давайте добавим обработчик события `ValueChanged` для элемента `numericUpDown1`. Для этого щелкните два раза указателем мыши по имени события `ValueChanged` в окне свойств. В код программы добавится функция с именем `numericUpDown1_ValueChanged`. Измените ее содержимое аналогично функции `trackBar1_Scroll`.

```
private void numericUpDown1_ValueChanged(object sender, System.EventArgs e){  
    int Value = (int)numericUpDown1.Value;  
    trackBar1.Value = Value;  
    progressBar1.Value = Value; }  
}
```

9. Откомпилируйте и запустите ее. Попробуйте изменить положение бегунка. При этом индикатор прогресса и числовой итератор изменят свои значения на

соответствующие величины. Попробуйте управлять индикатором прогресса при помощи числового итератора. Эффект будет аналогичный работе с бегунком.



Задание 2. Разработать проект, который позволяет пользователю вычислить факториал числа. Число, для которого рассчитывается факториал, выбирается с помощью элемента управления TrackBar. При щелчке по кнопке «Расчет», меняется надпись «Число n» на «N!» и в строке ввода выводится значение факториала числа.

ПРАКТИЧЕСКАЯ РАБОТА № 29

Тема: Создание проекта с использованием полос прокрутки для ввода информации

Цель работы: получить навыки добавления в программы полос прокруток для ввода данных на языке C#

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Написать простое приложение, которое будет рисовать на главной форме 4000 квадратных клеток, каждая из которых будет пронумерована для удобства. Максимальное количество клеток по горизонтали будет 80, по вертикали - 50 штук.

При запуске программы главная форма приложения будет выглядеть так:



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256
321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336
401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416
481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496
561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576
641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656

Как видим, слева-вверху номер самой первой клетки равен 1, затем нумерация клеток идёт по горизонтали и вниз. Также можно видеть, что при первом запуске формы оба скроллбара - горизонтальный и вертикальный - видны, и их позиция равна 0, т.е. они находятся в своём начальном положении.

Если мы передвинем бегунок горизонтального скроллбара в крайнее правое положение, то форма преобразится следующим образом:

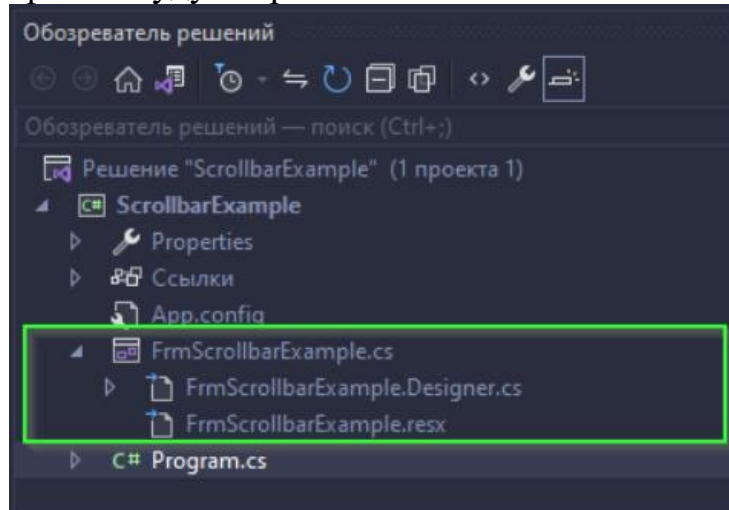


Аналогичным образом, если мы сдвинем бегунок вертикального скроллбара вниз до крайней позиции, то мы увидим самую последнюю клетку нашего "поля", номер которой равен 4000.



1. Откройте среду разработки Microsoft Visual Studio и создайте новый проект с типом Приложение Windows Forms (.NET Framework). В качестве имени нового проекта задайте ScrollbarExample и выберите местоположение, куда будут сохранены файлы нового проекта. После создания проекта будет создана главная форма по умолчанию с именем Form1 и соответствующий ей класс Form1.cs.

2. Выберите в окне "Обозреватель решений" эту форму и в окне "Свойства" измените название класса для формы на FrmScrollbarExample.cs. В диалоговом окне, запрашивающем подтверждение переименования класса формы и связанных с ней файлов, нужно согласиться. В итоге должно получиться следующее - в окне "Обозреватель решений" форма и зависящие от неё файлы будут переименованы:

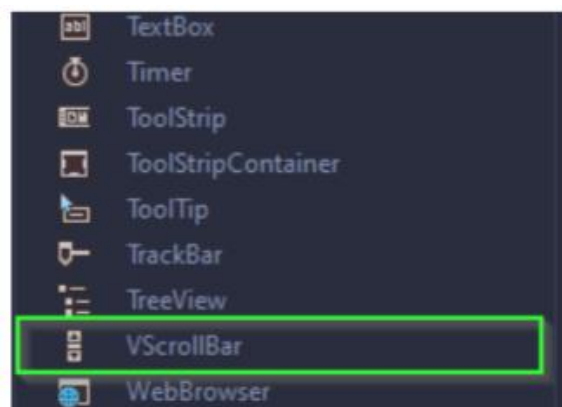
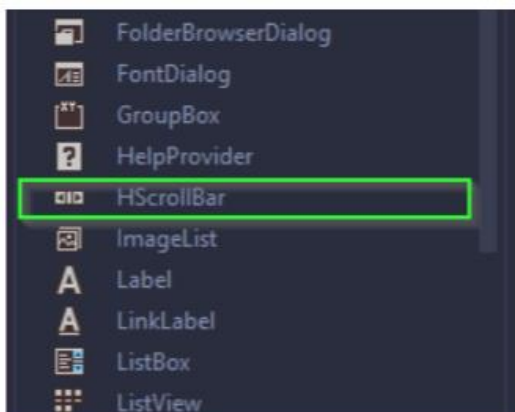


3. Измените свойства главной формы. Теперь в окне "Обозреватель решений" дважды кликнем на файл для формы FrmScrollbarExample.cs. В результате главная форма приложения будет выбрана в визуальном конструкторе, а справа-внизу появится список всех свойств формы. Установите для формы следующие значения свойств:

- Text - Пример работы с элементами HScrollBar и VScrollBar
- StartPosition - CenterScreen
- Size - 816; 489
- Name – FrmScrollbarExample

4. Добавить горизонтальный и вертикальный скроллбары на главную форму. Теперь нужно по очереди добавить горизонтальный скроллбар и вертикальный скроллбар на форму. Для этого слева в окне "Панель элементов" нужно найти нужные элементы, представляющие собой полосы прокрутки.

Горизонтальный скроллбар представлен элементом управления HScrollBar. Вертикальный скроллбар представлен элементом управления VScrollBar.



По очереди перетащите каждый из них на форму и установите следующие свойства:

Для горизонтального скроллбара: Dock – Bottom, Name - hScrollBar1

Для вертикального скроллбара: Dock – Right, Name - vScrollBar1

5. Напишите код для главной формы приложения. Дважды кликнув по главной форме из представления визуального конструктора. Откроется редактор кода для класса главной формы, в начало класса добавьте константы, задающие количество клеток по горизонтали и вертикали, а также словарь colorMap, который будет содержать номера клеток в качестве ключей и конкретный цвет клетки в качестве значения для указанного ключа:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
namespace ScrollbarExample {
    public partial class FrmScrollbarExample: Form {
        /// <summary>
        /// Количество клеток по горизонтали
        /// </summary>
        const int MAX_CELLS_HORIZONTAL = 80;
        /// <summary>
        /// Количество клеток по вертикали
        /// </summary>
        const int MAX_CELLS_VERTICAL = 50;
        private Dictionary<int, Color> colorMap = new Dictionary<int, Color>();
        /// ... остальной код формы
    }
```

6. Теперь напишите следующий код для события загрузки главной формы (метод FrmScrollbarExample_Load уже должен существовать в редакторе после двойного клика на форме, поэтому в него нужно добавить лишь три строки кода):

```
/// <summary>
    /// Загрузка главной формы приложения
    /// </summary>
    /// <param name="sender">объект главной формы, сгенерировавший
событие</param>
    /// <param name="e">параметры события</param>
    private void FrmScrollbarExample_Load(object sender, EventArgs e) {
        DoubleBuffered = true;
        hScrollBar1.Maximum = MAX_CELLS_HORIZONTAL * 50 - this.Width + 50;
        vScrollBar1.Maximum = MAX_CELLS_VERTICAL * 50 - this.Height + 80; }
    }
```

Разберём три строки, что добавили в метод загрузки формы: DoubleBuffered=true устанавливает в значение true свойство DoubleBuffered для главной формы - это позволяет

главной форме использовать двойную буферизацию, за счёт чего отрисовка всех клеток на форме не будет мерцать, а будет плавной. В двух других строках устанавливаем свойство `Maximum` для горизонтальной и вертикальной полос прокрутки, вычисляя их как произведение максимального числа клеток по горизонтали/вертикали на размер клетки в пикселях (50), затем вычитаем ширину/высоту самой формы и прибавляем числа 50 и 80 для горизонтального и вертикального скроллбара, соответственно. Числа 50 и 80 были подобраны экспериментально, они позволяют увидеть целиком ряд последних клеток по горизонтали и вертикали, когда позиция бегунка скроллбара будет максимальной. При желании вы можете увеличить эти числа и посмотреть на результат - вы просто будете видеть больше серого фона самой формы.

7. Теперь добавим в код формы следующий метод `GetCellColor`:

```
/// <summary>
/// Получает цвет для очередной клетки
/// </summary>
/// <param name="cellNumber">текущий номер клетки</param>
/// <param name="random"></param>
/// <returns>Возвращает цвет для клетки поля с заданным номером</returns>
private Color GetCellColor(int cellNumber, Random random) {
    Color cellColor;
    if (colorMap.ContainsKey(cellNumber)) {
        cellColor = colorMap[cellNumber];    }
    else {
        int red = random.Next(128, 256);
        int green = random.Next(128, 256);
        int blue = random.Next(128, 256);
        cellColor = Color.FromArgb(red, green, blue);
        colorMap.Add(cellNumber, cellColor); }
    return cellColor;    }
```

Метод будет получать цвет конкретной клетки поля по её номеру. В качестве второго аргумента для метода будем впоследствии передавать заранее созданный экземпляр класса `Random`, который умеет генерировать случайные числа в заданном диапазоне.

Логика метода довольно проста: сначала проверяем, есть ли уже в словаре `colorMap` ключ с номером клетки `cellNumber`. Если да, то просто возвращаем заранее сохранённый в словаре цвет клетки по ключу. Если же в словаре ещё нет ключа с номером клетки `cellNumber`, то с помощью переданного в метод экземпляра `random` сгенерируем значение для красного, зелёного и синего цветов в диапазоне от 128 до 255 (граничное значение 256 не включено в диапазон генерируемых значений). Далее мы генерируем цвет для клетки поля и сохраняем его в словаре. Метод всегда возвращает цвет клетки с заданным номером - независимо от того, был ли он уже сохранён в словаре или только что был добавлен в словарь.

8. Далее нужно написать метод с именем **`GetFontXDelta`**, который вычислит смещение шрифта внутри клетки при отрисовке номера клетки внутри её границ:

```
/// <summary>
```

```
/// Получить смещения шрифта по оси X. В зависимости от величины  
номера клетки
```

```
/// нужно добавлять дополнительное смещение для центрирования текста с  
её номером внутри самой клетки
```

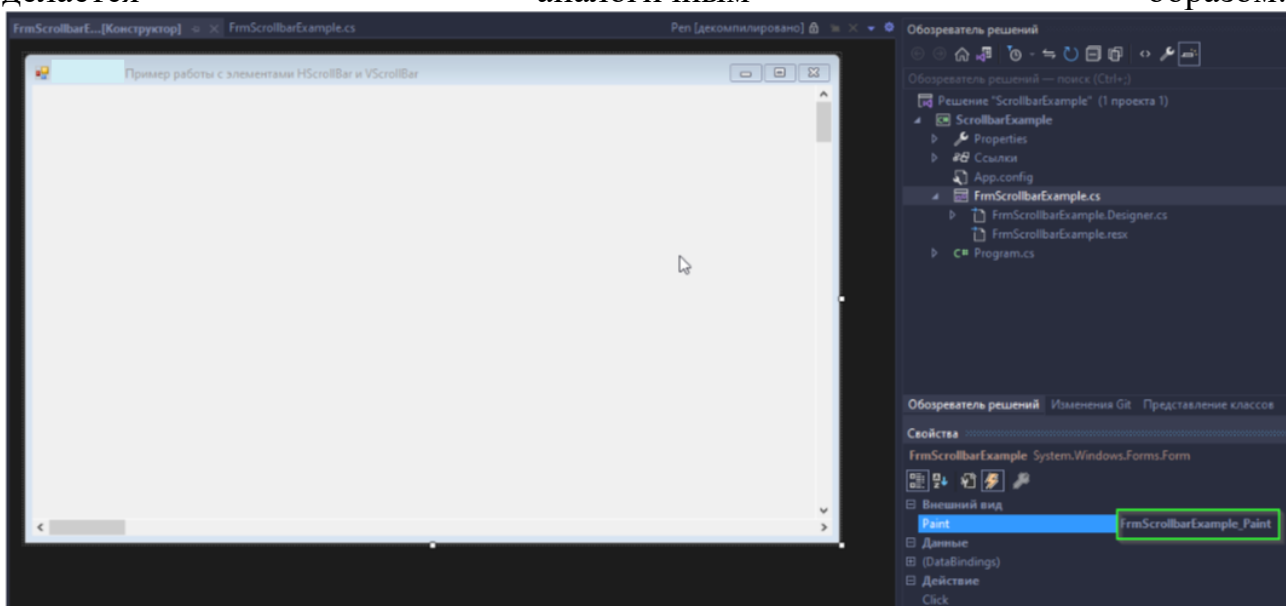
```
/// </summary>
```

```
/// <param name="figureNumber">номер клетки, для которого вычислить  
смещение шрифта по оси </param>
```

```
/// <returns>значение смещения по оси X для шрифта с заданным номером  
клетки</returns>
```

```
private int GetFontXDelta(int figureNumber) {  
    if (figureNumber >= 1 && figureNumber < 10) {  
        return 18;    }  
    else if (figureNumber >= 10 && figureNumber < 100) {  
        return 14;    }  
    else if (figureNumber >= 100 && figureNumber < 1000) {  
        return 8;     }  
    else {  
        return 2;     }    }  
}
```

9. Теперь нужно сгенерировать два метода-обработчика для событий Paint и SizeChanged главной формы. Для этого вернитесь к визуальному конструктору формы и выделите саму форму. В нижнем правом окне "Свойства" нажмите на иконку молнии для просмотра всех событий формы и напротив событий Paint и SizeChanged сделайте двойной клик. Ниже показан пример, как это делается для событий Paint, для события SizeChanged всё делается аналогичным образом:



10. Теперь напишите следующий код для сгенерированных обработчиков:

```
/// <summary>
```

```

    /// Отрисовка главной формы
    /// </summary>
    /// <param name="sender">объект главной формы, сгенерировавший
событие</param>
    /// <param name="e">параметры события</param>
    private void FrmScrollbarExample_Paint(object sender, PaintEventArgs e) {
        Graphics g = e.Graphics;
        Font font = new Font("Tahoma", 14, FontStyle.Regular, GraphicsUnit.Point);
        Random random = new Random();
        Brush blackBrush = new SolidBrush(Color.Black);
        Pen blackPen = new Pen(blackBrush, 1);
        int cellNumber = 1;
        for (int nofFiguresVert = 1; nofFiguresVert <= MAX_CELLS_VERTICAL;
nofFiguresVert++) {
            for (int nofFiguresHoriz = 1; nofFiguresHoriz <=
MAX_CELLS_HORIZONTAL; nofFiguresHoriz++) {
                Color cellColor = GetCellColor(cellNumber, random);
                Brush fillBrush = new SolidBrush(cellColor);
                int x = hScrollBar1.Visible ? (nofFiguresHoriz - 1) * 50 - hScrollBar1.Value :
(nofFiguresHoriz - 1) * 50;
                int y = vScrollBar1.Visible ? (nofFiguresVert - 1) * 50 - vScrollBar1.Value :
(nofFiguresVert - 1) * 50;
                Rectangle rect = new Rectangle(x, y, 50, 50);
                Rectangle fillRect = new Rectangle(x + 1, y + 1, 49, 49);
                g.DrawRectangle(blackPen, rect);
                g.FillRectangle(fillBrush, fillRect);
                Point pFont = new Point(x + GetFontXDelta(cellNumber), y + 15);
                g.DrawString(cellNumber.ToString(), font, blackBrush, pFont);
                cellNumber++;
                fillBrush.Dispose();          }          }
            blackPen.Dispose();
            blackBrush.Dispose();        }
    /// </summary>
    /// Обработка события изменения размера главной формы
    /// </summary>
    /// <param name="sender">объект главной формы, сгенерировавший
событие</param>
    /// <param name="e">параметры события</param>
    private void FrmScrollbarExample_SizeChanged(object sender, EventArgs e) {
        int width = MAX_CELLS_HORIZONTAL * 50 - this.Width + 50;
        int height = MAX_CELLS_VERTICAL * 50 - this.Height + 80;
        if (width < 0) {
            hScrollBar1.Visible = false;          }
        else {
            hScrollBar1.Visible = true;

```

```

        hScrollBar1.Maximum = width;
        hScrollBar1.Minimum = 0;          }
    if (height < 0) {
        vScrollBar1.Visible = false;      }
    else {
        vScrollBar1.Visible = true;
        vScrollBar1.Maximum = height;
        vScrollBar1.Value = 0;            }
    Invalidate();    }

```

11. Разбор метода-обработчика `FrmScrollbarExample_Paint`. В первой строке метода получаем в переменную `g` ссылку на объект класса `Graphics`, который уже присутствует в переменной `e`, представляющей собой параметры события `Paint`:
`Graphics g = e.Graphics;`

Это сделано просто для удобства, чтобы потом каждый раз для обращения к объекту класса `Graphics` нам не приходилось писать `e.Graphics`.

Далее создаём объект шрифта, которым будем "писать" внутри нарисованной клетки:

```
Font font = new Font("Tahoma", 14, FontStyle.Regular, GraphicsUnit.Point);
```

Следующей строкой создаём экземпляр класса `Random`, который будем использовать для генерации случайного цвета очередной клетки:

```
Random random = new Random();
```

Далее создаём сначала экземпляр класса `SolidBrush` и сохраняем его в переменную `blackBrush`. Он представляет собой сплошную "кисть" для рисования границ наших клеток, в аргументе для конструктора указываем, что цвет кисти - чёрный. Также создаём объект `Pen` (представляет собой "ручку") с заданными параметрами кисти и толщины штриха в 1 пиксель:

```
Brush blackBrush = new SolidBrush(Color.Black);
```

```
Pen blackPen = new Pen(blackBrush, 1);
```

Рисование границ клетки будет осуществляться с помощью экземпляра `blackPen`, объект кисти, как видим, нужен лишь для создания объекта класса `Pen`.

Далее, перед входом в циклы рисования поля из клеток инициализируем текущий номер клетки единицей:

```
int cellNumber = 1;
```

Затем организуем два цикла: внешний для отрисовки клеток по вертикали и вложенный в него цикл для отрисовки клеток по горизонтали:

```

for (int nofFiguresVert = 1; nofFiguresVert <= MAX_CELLS_VERTICAL;
    nofFiguresVert++) {
    for (int nofFiguresHoriz = 1; nofFiguresHoriz <=
        MAX_CELLS_HORIZONTAL; nofFiguresHoriz++) {
        // ... код внутри цикла ...
    }
}

```

Для каждой итерации вложенного цикла мы будем рисовать очередную клетку поля.

12.Разберём, что происходит во вложенном цикле. Сначала мы получаем цвет клетки по её номеру - через вызов ранее созданного метода GetCellColor. И создаём кисть с полученным цветом для окрашивания содержимого клетки:

```
Color cellColor = GetCellColor(cellNumber, random);
```

```
Brush fillBrush = new SolidBrush(cellColor);
```

Теперь нужно вычислить координаты для левого верхнего угла рисуемой клетки:

```
int x = hScrollBar1.Visible ? (nofFiguresHoriz - 1) * 50 - hScrollBar1.Value :
```

```
(nofFiguresHoriz - 1) * 50;
```

```
int y = vScrollBar1.Visible ? (nofFiguresVert - 1) * 50 - vScrollBar1.Value :
```

```
(nofFiguresVert - 1) * 50;
```

В случае, когда горизонтальный скроллбар видимый, то учитывается текущая позиция его бегунка hScrollBar1.Value. Если же горизонтальный скроллбар скрытый, то считается, что горизонтальные клетки полностью поместились на форме, поэтому просто используется формула (nofFiguresHoriz - 1) * 50 для вычисления координаты x левого верхнего угла клетки. Эта формула используется потому что размерность клеток - 50x50 пикселей, и для переменной внешнего цикла nofFiguresHoriz, равной 1, получим (1 - 1) * 50, т.е. 0, значит самая первая клетка будет рисоваться в самом первом пикселе формы с индексом 0 по горизонтали.

То же самое делаем и для координаты y, которая показывает текущее смещение по вертикали для левого верхнего угла рисуемой клетки.

Далее идёт код, который рисует внешнюю границу клетки, используя прямоугольник rect и осуществляет заполнение прямоугольной области fillRect клетки заданным цветом:

```
Rectangle rect = new Rectangle(x, y, 50, 50);
```

```
Rectangle fillRect = new Rectangle(x + 1, y + 1, 49, 49);
```

```
g.DrawRectangle(blackPen, rect);
```

```
g.FillRectangle(fillBrush, fillRect);
```

Наконец, вложенный цикл заканчивается следующими строками кода:

```
Point pFont = new Point(x + GetFontXDelta(cellNumber), y + 15);
```

```
g.DrawString(cellNumber.ToString(), font, blackBrush, pFont);
```

```
cellNumber++;
```

```
fillBrush.Dispose();
```

Здесь в переменную pFont сохраняются координаты той точки, в которой должен быть отрисован текст, содержащий номер самой клетки. По оси X требуется дополнительное смещение, поскольку числа от 0 до 9 имеют одну "ширину", числа от 10 до 99 - другую ширину и так далее. Для того, чтобы номер клетки всегда был отрисован по центру клетки нужен метод GetFontXDelta, который вернёт нужную величину смещения шрифта, которую прибавляем к координате x.

13.Для рисования текста используется доступный метод DrawString в классе Graphics. Как видно, в него передаётся сама строка с номером клетки,

шрифт, которым необходимо нарисовать надпись в клетке, объект кисти и, наконец, точку `pFont` для левого верхнего угла области отрисовки текста.

После отрисовки надписи с номером клетки увеличиваем номер текущей клетки на единицу и освобождаем ресурсы, занимаемые кистью `fillBrush`, поскольку её создаём каждый раз во вложенном цикле.

Метод завершается двумя строками, освобождающими ресурсы, выделенные под объекты классов `SolidBrush` и `Pen`, созданные до входа в циклы:

```
blackPen.Dispose();
```

```
blackBrush.Dispose();
```

14. Разбор метода-обработчика `FrmScrollbarExample_SizeChanged`.

Метод-обработчик вызывается при изменениях размера главной формы. Его предназначение в том, чтобы отрегулировать состояние полос прокрутки в зависимости от текущих размеров формы. Поэтому сначала вычисляем ширину (`width`) и высоту (`height`), которые затем должны стать максимальным значением горизонтального и вертикального скроллбара - но в том случае, если они неотрицательны. Ведь может случиться и так, что ширина/высота формы (к примеру, в развёрнутом виде) превысит произведения `MAX_CELLS_HORIZONTAL * 50` и `MAX_CELLS_VERTICAL * 50`.

```
int width = MAX_CELLS_HORIZONTAL * 50 - this.Width + 50;
```

```
int height = MAX_CELLS_VERTICAL * 50 - this.Height + 80;
```

Следующим шагом мы проверяем - отрицательны или нет получившиеся значения. Если отрицательны, то мы скрываем конкретный скроллбар (признак того, что все клетки поля видимы). Если неотрицательны - устанавливаем свойства `Maximum` и `Minimum` для конкретного скроллбара и делаем его видимым:

```
if (width < 0) {
```

```
    hScrollBar1.Visible = false;    }
```

```
else {
```

```
    hScrollBar1.Visible = true;
```

```
    hScrollBar1.Maximum = width;
```

```
    hScrollBar1.Minimum = 0;    }
```

```
if (height < 0) {
```

```
    vScrollBar1.Visible = false;    }
```

```
else {
```

```
    vScrollBar1.Visible = true;
```

```
    vScrollBar1.Maximum = height;
```

```
    vScrollBar1.Value = 0; }
```

Последней строкой метода вызываем метод `Invalidate()` - для того, чтобы принудительно сделать недействительной всю поверхность главной формы и вызвать её перерисовку: `Invalidate()`;

15. Програмируем реакцию скроллбаров на изменение позиции бегунка. Осталось сделать - это вернуться на главную форму и для каждого из скроллбаров сгенерировать их методы-обработчики для события **Scroll**. Это событие отвечает за "скроллинг", т.е. прокрутку бегунка для конкретного

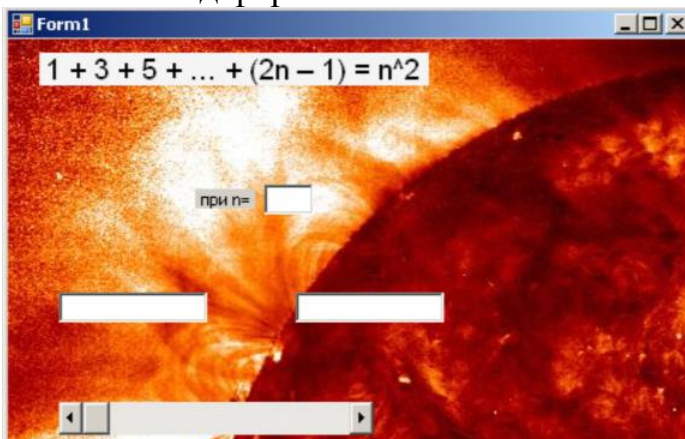
скроллбара. Сгенерировать методы можно также через двойной клик напротив имени события, как делалось для Paint и SizeChanged. После генерации методов-обработчиков просто вызовите Invalidate() из каждого из них, это заставит главную форму перерисоваться в момент "скроллинга":

```
/// <summary>
/// Изменение позиции бегунка для горизонтального скроллбара
/// </summary>
/// <param name="sender">объект горизонтального скроллбара,
сгенерировавший событие</param>
/// <param name="e">параметры события</param>
private void hScrollBar1_Scroll(object sender, ScrollEventArgs e) {
    Invalidate();
}
/// <summary>
/// Изменение позиции бегунка для вертикального скроллбара
/// </summary>
/// <param name="sender">объект вертикального скроллбара,
сгенерировавший событие</param>
/// <param name="e">параметры события</param>
private void vScrollBar1_Scroll(object sender, ScrollEventArgs e) {
    Invalidate();
}
```

16. Запустить приложение и протестируйте самостоятельно работу горизонтальной и вертикальной полос прокрутки, а также посмотрите на реакцию формы на события прокрутки бегунков для скроллбаров.

Задание 2. В приводимых ниже заданиях организовать вычисление с помощью полосы прокрутки для различных n . Причём, предусмотреть вычисление, как в цикле, так и по формуле, приведенной в правой части выражения: $1 + 3 + 5 + \dots + (2n - 1) = n^2$

Внешний вид формы:



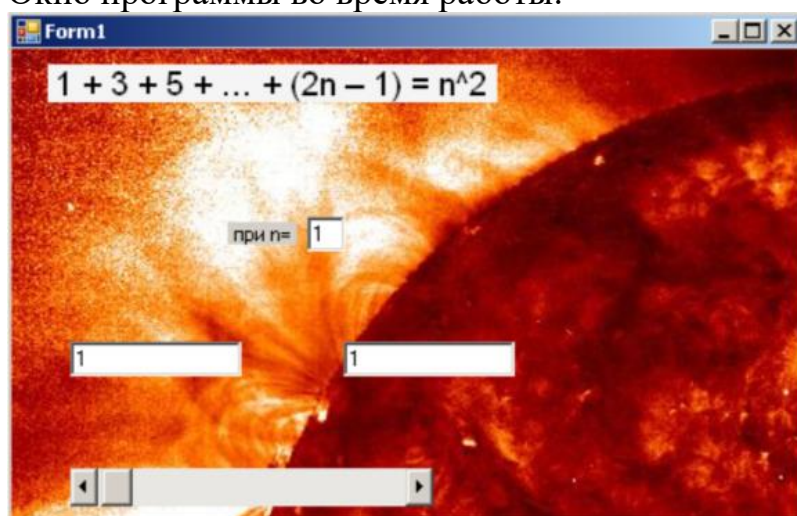
На форме расположены следующие компоненты:

Label – 2 штуки. Присутствует для отображения какой-то информации. В данной программе отображается значение n получаемой в результате прокручивания полосы прокрутки, а также значения, полученные в результате вычислений по формуле.

TextBox – 3 штуки. Присутствует для ввода данных.

Во время работы программы сущность подсчёта не видна. Для этого следует посмотреть блок-схему алгоритма программы. Так как при прокручивании бегунка видим лишь результат вычислений, а нам необходимо узнать, как именно шел подсчёт данных.

Окно программы во время работы:



ПРАКТИЧЕСКАЯ РАБОТА № 30


Тема: Создание проекта с использованием группы зависимых переключателей


Цель работы: получить навыки использовать группы переключателей на языке C#


Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы


Справочный материал:

Элементы управления «Кнопки».


Нажимаемой кнопкой (Button)  Button называется элемент управления, все взаимодействие пользователя с которым ограничивается одним действием — нажатием. Все, что вам необходимо сделать при работе с кнопкой, — это поместить ее в нужном месте формы и назначить ей соответствующий обработчик. Обработчик назначается для события Click.


Флажки (CheckBox)  CheckBox являются кнопками отложенного действия, т. е. их нажатие не должно запускать какое-либо немедленное действие. С их помощью пользователи вводят параметры, которые скажутся после, когда действие будет запущено иными элементами управления. Элемент CheckBox может иметь 3 состояния — помеченное, непомеченное и смешанное. Чаще всего этот элемент применяется для определения значений, которые могут иметь только два состояния.

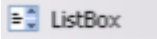
Радиокнопки (RadioButton)  RadioButton по своим свойствам немного похожи на флажки. Их главное различие заключается в том, что группа флажков позволяет выбрать любую комбинацию параметров, радиокнопки же дают возможность выбрать только один параметр. Из этого различия проистекают и все остальные. Например, в группе не может быть меньше двух радиокнопок. Кроме того, у радиокнопок не может быть смешанного состояния.


Блок группировки (GroupBox)  GroupBox помогает визуально объединить несколько элементов управления в одну группу. Это бывает особенно полезно, когда надо придать вашему приложению более понятный пользовательский интерфейс. Например, объединить группу радиокнопок.


Элементы управления «Поля ввода» и «Списки».

Поле ввода (TextBox)  TextBox является основным элементом, предназначенным для ввода пользователем текстовых данных. Использовать TextBox можно в однострочном или многострочном режиме. Однако данный элемент управления имеет ограничение — до 64 кБайт текста. Если вам необходимо обрабатывать большие объемы информации, лучше использовать элемент RichTextBox.

Расширенное поле ввода (RichTextBox)  RichTextBox дает возможность пользователю вводить и обрабатывать большие объемы информации (более 64 кБайт). Кроме того, RichTextBox позволяет редактировать цвет текста, шрифт, добавлять изображения. RichTextBox включает все возможности текстового редактора Microsoft Word.

Список (ListBox)  представляет собой простейший вариант пролистываемого списка. Он позволяет выбирать один или несколько хранящихся в списке элементов. Кроме того, ListBox имеет возможность отображать данные в нескольких колонках. Это позволяет представлять данные в большем объеме и не утомлять пользователя скроллингом.

Помечаемый список (CheckedListBox)  является разновидностью простого списка. Его дополнительное достоинство — в наличии флажков рядом с каждым элементом списка. Пользователь имеет возможность отметить один или несколько элементов списка, выставив напротив его флажок.

Выпадающий список (ComboBox)  удобен тем, что не занимает много пространства на форме. Постоянно на форме представлено только одно значение этого списка. При необходимости пользователь может раскрыть список и выбрать другое интересующее его значение. Кроме того, режим DropDown дает пользователю возможность вводить собственное значение при отсутствии необходимого значения в списке.

Содержание работы:

Задание 1. Создание формы с различными видами кнопок. Программа должна будет выполнять следующие функции: радиокнопки задают текст сообщения, которое будет выводиться по нажатию на обычную кнопку. Флажок должен определять — выводить сообщение или нет.



1. Создайте новый Windows Application проект под названием TestButtons. Сохраните его в созданную папку. Измените некоторые свойства созданной формы: Name — «TestButtonsForm»; Text — «Тест для кнопок»

2. Теперь добавьте на форму один элемент управления GroupBox, три элемента RadioButton, один элемент CheckBox и один элемент Button. На рисунке показано примерное расположение элементов. Можете разместить элементы произвольным образом. Однако заметьте, что все три радиокнопки должны быть помещены в один GroupBox. Иначе они не будут связаны между собой.

3. Теперь измените некоторые свойства добавленных элементов:

Button1: Text — Показать сообщение

groupBox1: Text — Выберите текст сообщения

radioButton1: Text — Первое сообщение

radioButton2: Text — Второе сообщение

radioButton3: Text — Третье сообщение

checkBox1: Text — Показывать сообщение, Checked — true

4.Теперь давайте обратимся к коду программы, который создала среда Visual Studio. Среда создает два основных файла с кодом – код дизайнера форм (TestButtons.Designer.cs) и основной код программы (Program.cs).

Файл TestButtons.Designer.cs содержит:

```
namespace TestButtons {
partial class TestButtonsForm {
private
System.ComponentModel.IContainer
components = null;
protected override void Dispose(bool disposing) {
if (disposing && (components != null))
components.Dispose(); }
base.Dispose(disposing); }
#region Windows Form Designer generated code
private void InitializeComponent() {
System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(TestButtonsForm));
this.groupBox1 = new System.Windows.Forms.GroupBox();
this.radioButton3=new System.Windows.Forms.RadioButton();
this.radioButton2=new System.Windows.Forms.RadioButton();
this.radioButton1=new System.Windows.Forms.RadioButton();
this.checkBox1= new System.Windows.Forms.CheckBox();
this.button1 = new System.Windows.Forms.Button();
this.groupBox1.SuspendLayout();
this.SuspendLayout();
// groupBox1
this.groupBox1.Controls.Add(this.radioButton3);
this.groupBox1.Controls.Add(this.radioButton2);
this.groupBox1.Controls.Add(this.radioButton1);
resources.ApplyResources(this.groupBox1, "groupBox1");
this.groupBox1.Name = "groupBox1";
this.groupBox1.TabStop = false;
// radioButton3
resources.ApplyResources(this.radioButton3, "radioButton3");
this.radioButton3.Name = "radioButton3";
this.radioButton3.TabStop = true;
this.radioButton3.UseVisualStyleBackColor = true;
// radioButton2
resources.ApplyResources(this.radioButton2, "radioButton2");
this.radioButton2.Name = "radioButton2";
this.radioButton2.TabStop = true;
this.radioButton2.UseVisualStyleBackColor = true;
// radioButton1
resources.ApplyResources(this.radioButton1, "radioButton1");
```

```

this.radioButton1.Name = "radioButton1";
this.radioButton1.TabStop = true;
this.radioButton1.UseVisualStyleBackColor = true;
// checkBox1
resources.ApplyResources(this.checkBox1, "checkBox1");
this.checkBox1.Name = "checkBox1";
this.checkBox1.UseVisualStyleBackColor = true;
// button1
resources.ApplyResources(this.button1, "button1");
this.button1.Name = "button1";
this.button1.UseVisualStyleBackColor = true;
// TestButtonsForm
resources.ApplyResources(this, "$this");
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.Controls.Add(this.button1);
this.Controls.Add(this.checkBox1);
this.Controls.Add(this.groupBox1);
this.Name = "TestButtonsForm";
this.groupBox1.ResumeLayout(false);
this.groupBox1.PerformLayout();
this.ResumeLayout(false);
this.PerformLayout();          }
#endregion
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.RadioButton radioButton3;
private System.Windows.Forms.RadioButton radioButton2;
private System.Windows.Forms.RadioButton radioButton1;
private System.Windows.Forms.CheckBox checkBox1;
private System.Windows.Forms.Button button1; } }

```

Ваш код может отличаться от приведенного координатами расположения элементов управления и их размером. То есть свойствам Location и Size объектов в функции InitializeComponent могут присваиваться другие значения. Это никак не влияет на работоспособность программы.

5. Если строго следовали всем инструкциям, то программа должна построиться без ошибок. Откомпилируйте ее и запустите. Пока программа не способна выполнять какие-либо действия.

Добавьте функцию-обработчик для кнопки button1. Используйте имя по умолчанию для этой функции — button1_Click, создаваемое средой VisualStudio. При этом в файл TestButtons.cs добавится функция:

```

private void button1_Click(object sender, System.EventArgs e){
}

```

Также появится обработчик этого события в функции InitializeComponent:

```

this.button1.Click += new System.EventHandler(this.button1_Click);

```

6. Добавьте в тело функции button1_Click следующий код:

```

private void button1_Click(object sender, EventArgs e) {

```



```

//объявляем строковую переменную для хранения выбранного сообщения
string strMessage = "";
//определяем какая именно радиокнопка отмечена и выбираем в соответствии
// с этим текст выводимого сообщения проверяем первую радиокнопку
if (radioButton1.Checked == true) {
//если отмечена именно эта кнопка, то копируем текст кнопки в переменную
strMessage = radioButton1.Text;    }
//проверяем вторую радиокнопку
else if (radioButton2.Checked == true) {
//если отмечена именно эта кнопка, то копируем текст кнопки в переменную
strMessage = radioButton2.Text;    }
//проверяем третью радиокнопку
else if (radioButton3.Checked == true)    {
//если отмечена именно эта кнопка, то копируем текст кнопки в переменную
strMessage = radioButton3.Text;    }
//проверяем, установлен ли флажок, разрешающий вывод сообщения
//если да, то выводим выбранное сообщение на экран
if (checkBox1.Checked == true)
MessageBox.Show("Вы выбрали " + strMessage);    }

```

7.Откомпилируйте и запустите программу. Выберите первую радиокнопку «первое сообщение». Нажмите кнопку Показать сообщение. На экране появится надпись: Вы выбрали первое сообщение.

Выбрав иную радиокнопку, вы получите другой текст сообщения.

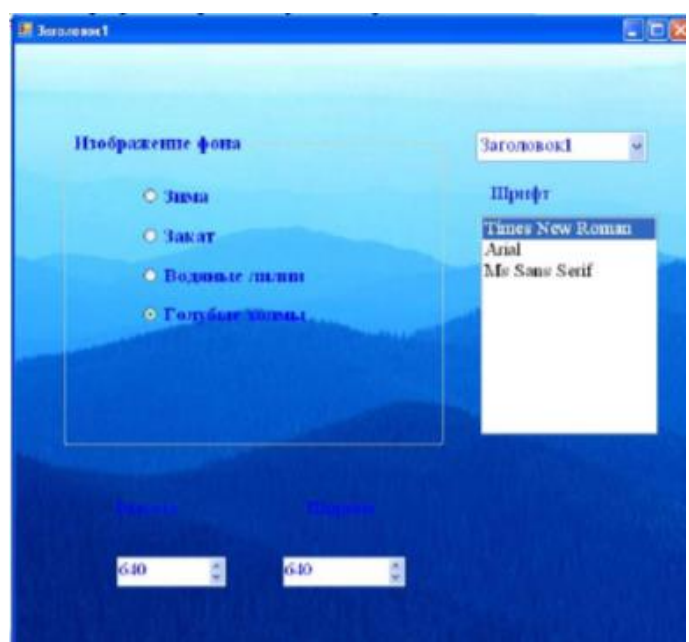
Теперь уберите флажок Показывать сообщение. Нажмите кнопку Показать сообщение. На экране ничего не должно появиться.

Если программа работает в строгом соответствии с данным описанием, значит, вы все сделали верно.

Задание 2. Создайте Windows-приложение, на форме которого определите следующие элементы:

- радиокнопки, которые управляют выбором изображения фона формы (4 варианты). При выборе радиокнопки сразу изменяется фон формы.
- в комбинированном списке выбирается заголовок формы (4 варианта).
- список с единичным выбором позволяет выбрать шрифт на всех элементах управления.

Во всех элементах должны быть выбраны значения по умолчанию, именно те, которые используются на форме при запуске приложения.



ПРАКТИЧЕСКАЯ РАБОТА № 31

Тема: Создание проекта с использованием группы зависимых переключателей

Цель работы: получить навыки использовать группы переключателей на языке C#

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

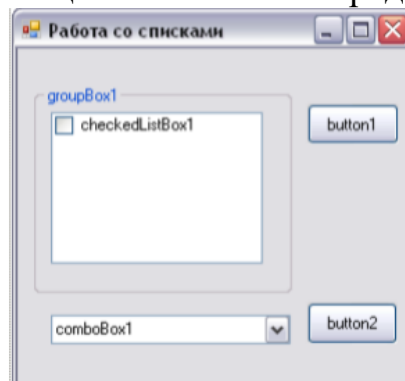
Задание 1. Создание формы с использованием списков. Необходимо написать программу, предназначенную для учета данных об участниках соревнований. Программа будет содержать два списка — ComboBox, для ввода информации об участниках, и CheckedListBox, для хранения и обработки данных. С помощью списка ComboBox пользователь будет выбирать фамилии лиц, которых необходимо добавить в список участников. Две кнопки на форме будут добавлять или удалять участников из списка.

1.Создайте новый Windows Applications проект под названием TestLists. Сохраните его в созданную папку. Переименуйте файл Form1.cs в TestListsForm.cs.

2.Теперь добавьте на форму следующие элементы управления:

- GroupBox, и поместите в него CheckedListBox
- ComboBox
- два элемента Button.

Приблизительное размещение элементов представлено на рисунке



3.Измените некоторые свойства созданной формы: Text —«работа со списками»

4.Теперь измените свойства элементов управления:

groupBox1: Text — «список участников»

checkedListBox1: Name — memberList

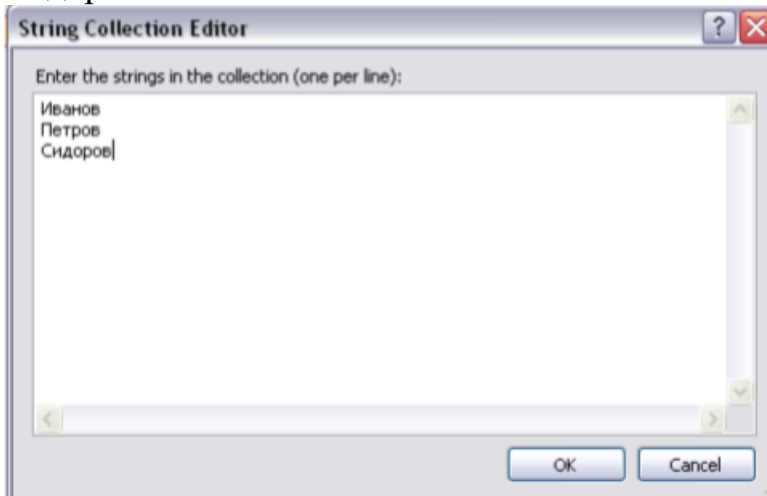
comboBox1: Name — peopleList, Text —«»

button1: Name — buttonAdd, Text — «Добавить»

button2: Name — buttonDelete, Text — «Удалить»

5.Элементы управления ComboBox и CheckedListBox могут быть проинициализированы с помощью дизайнера среды VisualStudio. Для хранения элементов списков данные компоненты имеют свойство Items. Свойство Items само по себе является массивом строк. Проинициализируйте элемент управления ComboBox, который имеет имя peopleList, списком фамилий предполагаемых участников соревнований. Для этого в окне свойств peopleList выберите свойство

Items. Откройте окно StringCollectionEditor, нажав на кнопку с тремя точками в поле Items. Добавьте в предложенный список три фамилии: Иванов, Петров, Сидоров



6. Добавьте обработчики для кнопок Добавить и Удалить, два раза щелкнув левой кнопкой мыши по каждой из кнопок. Подготовительный этап к написанию программы завершен. Сохраните сделанные вами изменения. Проанализируйте сгенерированный средой код программы (файлы TestListsForm.cs и TestListsForm.Designer.cs).

7. Откомпилируйте и запустите проект.

8. Сейчас необходимо добавить обработчики для кнопки «Добавить» и «Удалить». Как известно из исходных данных, кнопка «Добавить» должна заносить строку, выбранную в комбобоксе, в список участников. Для этого измените функцию buttonAdd_Click так, как показано ниже:

```
private void buttonAdd_Click(object sender, System.EventArgs e) {  
    //работаем со списком для ввода фамилий проверяем выбран ли элемент в списке  
    if(peopleList.Text.Length != 0)    {  
        //если элемент выбран, то переносим его в список участников  
        memberList.Items.Add(peopleList.Text);    }  
    else {  
        //если элемент не выбран, то выдает информационное сообщение  
        MessageBox.Show("Выберите элемент в списке для ввода или введите новый.");  
    }  
}
```

9. Функция memberList.Items.Add добавляет новый элемент в список memberList. При этом параметром функции является значение свойства peopleList.Text, которое выбирает пользователь. Теперь осталось реализовать удаление элементов из списка. Для этого введите код для функции buttonDelete_Click.

```
private void buttonDelete_Click(object sender, System.EventArgs e) {  
    //пока список помеченных элементов не пуст  
    while(memberList.CheckedIndices.Count > 0)  
        //удаляем из общего списка участников по одному элементу  
        //при этом список помеченных элементов автоматически обновляется  
        //каждый раз нулевой элемент из CheckedIndices  
        //будет содержать индекс первого помеченного в списке объекта
```

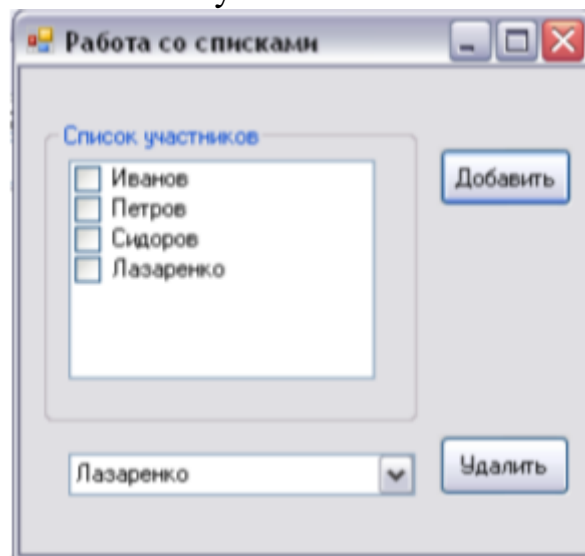
```
memberList.Items.RemoveAt(memberList.CheckedIndices[0]);  
//при удалении из списка последнего помеченного элемента  
//CheckedIndices.Count станет равным нулю и цикл автоматически завершится  
}
```

10. Функция `memberList.Items.RemoveAt` удаляет из списка элемент по его индексу. При этом элементы списка, идущие за удаленным, уменьшают свой индекс на единицу. Это обязательно нужно учитывать при дальнейшем обходе списка.

11. Класс `CheckedListBox` содержит свойство `CheckedIndices`, которое представляет собой массив индексов всех помеченных элементов списка. Этот массив тоже изменяется, если из списка был удален помеченный элемент. А поскольку удаляем из списка только помеченные элементы, то `CheckedIndices` будет изменяться всегда: место удаленного элемента займет следующий за ним.

12. Цикл продолжит работать до тех пор, пока в списке `CheckedIndices` будет оставаться хоть один элемент.

13. Откомпилируйте и запустите приложение. Заполните список участников так, как показано на рисунке. Для того чтобы добавить элемент в список, выберите его в комбобоксе и нажмите кнопку `Add`.

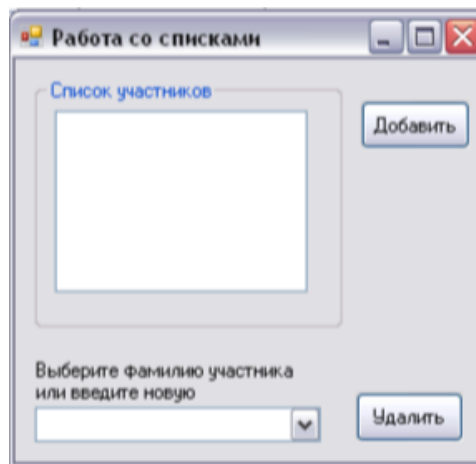


14. По умолчанию, элемент управления `ComboBox` имеет стиль `DropDown` (свойство `ComboBox.DropDownStyle.DropDown`). Этот стиль дает возможность пользователю не только выбирать элементы из списка, но и вводить данные с клавиатуры. Поэтому для добавления в список фамилии «Лазаренко», наберите ее на клавиатуре и нажмите кнопку `Add`.

15. Теперь давайте разберемся с логикой работы программы при удалении элементов из списка. Для этого выделите в списке фамилии «Петров» и «Сидоров», которые в списке имеют индексы 1 и 2 соответственно (начиная с нулевого). Поэтому массив `CheckedIndices` будет содержать два элемента — 1 и 2. При нажатии кнопки `Delete` программа по циклу начинает удалять элементы из списка. Нулевым элементом в массиве `CheckedIndices` стоит число 1 (индекс фамилии «Петров»), поэтому фамилия «Петров» первой удаляется из списка. При этом фамилии «Сидоров» и «Лазаренко» изменяют свои индексы с 2 и 3 на 1 и 2 соответственно. Массив `CheckedIndices` тоже модифицируется. Во-первых,

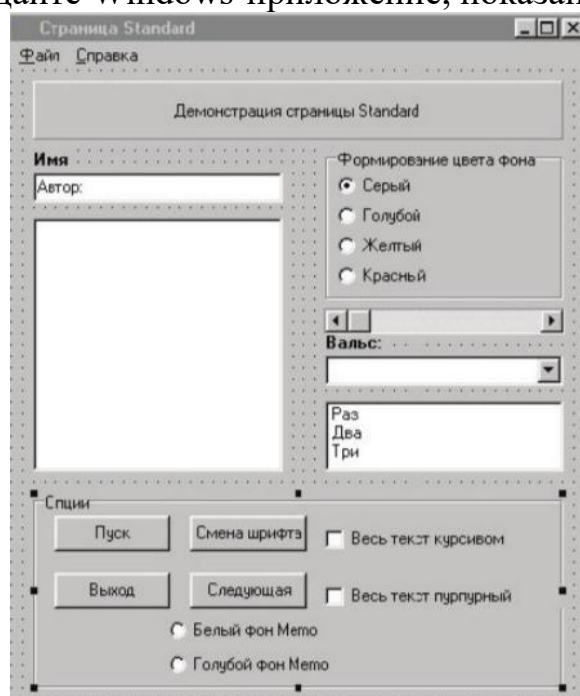
из него удалится нулевой элемент, индекс фамилии «Петров». Во-вторых, место нулевого элемента в массиве `CheckedIndices` теперь займет индекс фамилии «Сидоров». А поскольку «Сидоров» теперь имеет индекс 1 в общем списке, то `CheckedIndices [0]` будет содержать число 1. На второй итерации цикла удаления из списка исчезнет фамилия «Сидоров», а «Лазаренко» переместится на позицию 1. В итоге, коллекция `CheckedIndices` окажется пустой, и цикл завершится.

16. Элемент управления `Label` предназначен для создания подписей к другим элементам управления или для вывода информационных сообщений прямо на поверхности формы. Например, на форму `TestListsForm` можно добавить элемент управления `Label` с надписью «Выберите фамилию участника или введите новую». Надписи повышают уровень восприятия программы пользователем.



Задание 2. Создайте Windows-приложение, используя переключатели или флажки, для выбора цвета формы. При выборе цвета появляется сообщение с выбранным цветом.

Задание 3. Создайте Windows-приложение, показанное на рисунке



ПРАКТИЧЕСКАЯ РАБОТА № 32

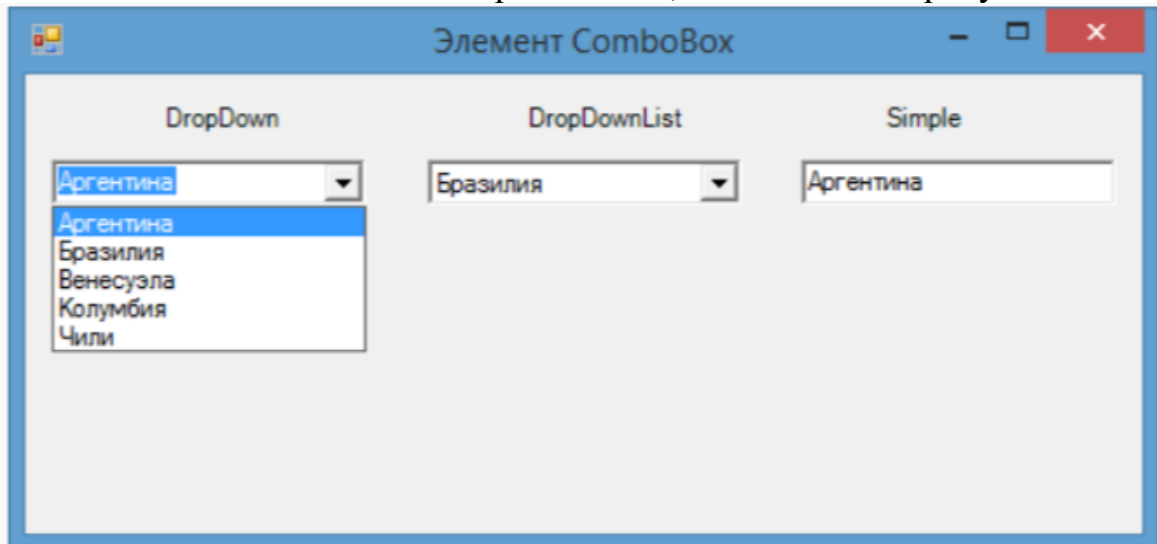
Тема: Создание проекта с использованием группы зависимых переключателей

Цель работы: получить навыки использовать группы переключателей на языке C#

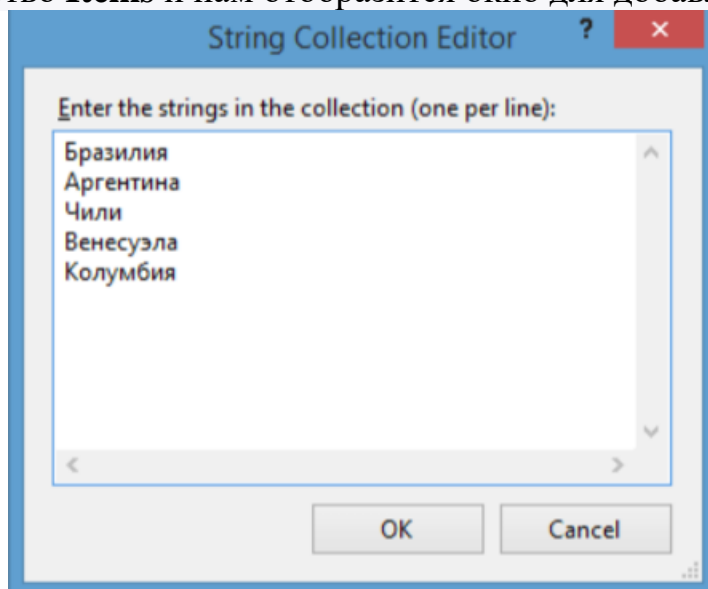
Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Создать Windows-приложение, показанное на рисунке



Для хранения элементов списка в ComboBox также предназначено свойство **Items** и нам отобразится окно для добавления элементов ComboBox.



С помощью ряда свойств можно настроить стиль оформления компонента. Так, свойство `DropDownWidth` задает ширину выпадающего списка. С помощью свойства `DropDownHeight` можно установить высоту выпадающего списка. Еще одно свойство `MaxDropDownItems` позволяет задать число видимых элементов списка - от 1 до 100. По умолчанию это число равно 8. Другое свойство `DropDownStyle` задает стиль ComboBox. Оно может принимать три возможных значения:

- **Dropdown:** используется по умолчанию. Мы можем открыть выпадающий список вариантов при вводе значения в текстовое поле или нажав на кнопку со стрелкой в правой части элемента, и нам отобразится собственно выпадающий список, в котором можно выбрать возможный вариант

- **DropDownList:** чтобы открыть выпадающий список, надо нажать на кнопку со стрелкой в правой стороне элемента

- **Simple:** **ComboBox** представляет простое текстовое поле, в котором для перехода между элементами мы можем использовать клавиши клавиатуры **вверх/вниз**

Задание 2. Создайте форму, которая позволит ввести имя, адрес, род занятий и возраст. На форме должны располагаться кнопки **OK** и **Help**. Данные из формы считываются и заносятся в результирующее текстовое поле, расположенное на этой же форме. При нажатии на кнопку **Help** выводится краткое описание каждого текстового окна в результирующем текстовом поле. В программе должна быть реализована проверка вводимого текста по следующим критериям:

- а) поле с именем пользователя не должно быть пустым;
- б) возраст пользователя должен представлять собой число, большее или равное 0;
- в) род занятий пользователя должен описываться как программист или оставаться пустым (можно использовать флажок или текстовое поле);
- г) поле с адресом пользователя не может оставаться пустым;
- д) поле для вывода результирующей информации должно быть только для чтения.

Подсказка: Для проверки значения возраста необходимо обработать сообщение **KeyPress**. Нажатие кнопки **OK** невозможно до введения всей необходимой информации.

ПРАКТИЧЕСКАЯ РАБОТА № 33

Тема: Создание процедур на основе событий

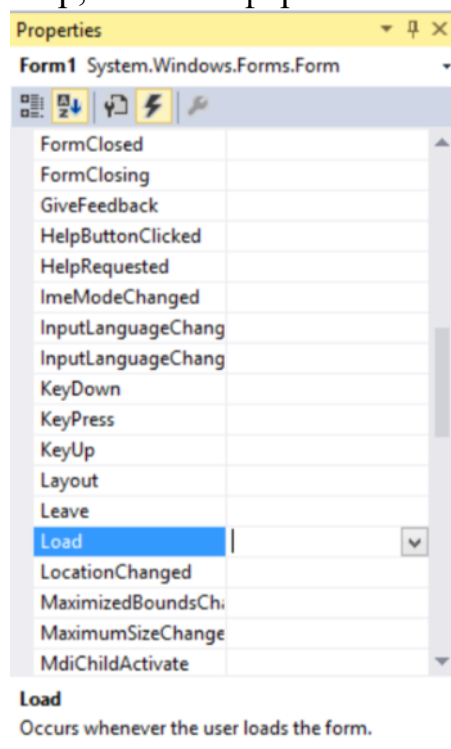
Цель работы: научиться создавать процедуры на основе событий, используя язык С#, при разработке Windows-приложений.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

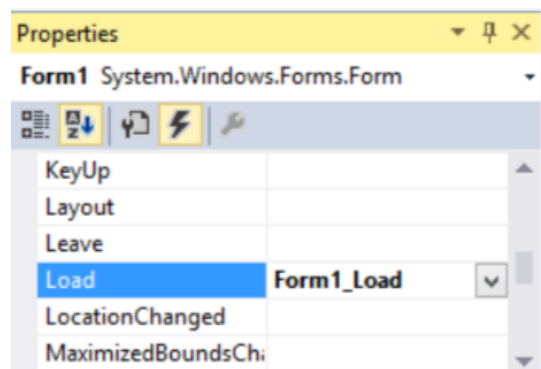
Справочный материал:

Для взаимодействия с пользователем в Windows Forms используется механизм событий. События в Windows Forms представляют стандартные события на С#, только применяемые к визуальным компонентам и подчиняются тем же правилам, что события в С#. Но создание обработчиков событий в Windows Forms все же имеет некоторые особенности.

Прежде всего в WinForms есть некоторый стандартный набор событий, который по большей части имеется у всех визуальных компонентов. Отдельные элементы добавляют свои события, но принципы работы с ними будут похожие. Чтобы посмотреть все события элемента, нам надо выбрать этот элемент в поле графического дизайнера и перейти к вкладке событий на панели форм. Например, события формы:



Чтобы добавить обработчик, можно просто два раза нажать по пустому полю рядом с названием события, и после этого Visual Studio автоматически сгенерирует обработчик события. Например, нажмем для создания обработчика для события Load:



И в этом поле отобразится название метода обработчика события Load. По

умолчанию он называется Form1_Load. Если мы перейдем в файл кода формы Form1.cs, то увидим автосгенерированный метод Form1_Load:

```
public partial class Form1 : Form{  
    public Form1() {  
        InitializeComponent(); }  
    private void Form1_Load(object sender, EventArgs e) {  
    } }  
}
```

И при каждой загрузке формы будет срабатывать код в обработчике Form1_Load. Как правило, большинство обработчиков различных визуальных компонентов имеют два параметра: sender - объект, инициировавший событие, и аргумент, хранящий информацию о событии (в данном случае EventArgs e).

Но это только обработчик. Добавление же обработчика, созданного таким образом, производится в файле Form1.Designer.cs:

```
namespace HelloApp {  
    partial class Form1 {  
        private System.ComponentModel.IContainer components = null;  
        protected override void Dispose(bool disposing) {  
            if (disposing && (components != null)) {  
                components.Dispose();  
            }  
            base.Dispose(disposing);  
        }  
        private void InitializeComponent() {  
            this.SuspendLayout();  
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);  
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;  
            this.ClientSize = new System.Drawing.Size(284, 261);  
            this.Name = "Form1";  
            // добавление обработчика  
            this.Load += new System.EventHandler(this.Form1_Load);  
            this.ResumeLayout(false); } }  
}
```

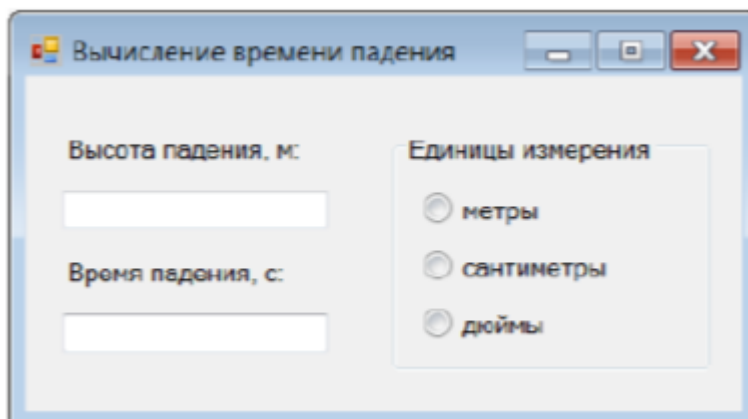
Для добавления обработчика используется стандартный синтаксис C#: this.Load += new System.EventHandler(this.Form1_Load)

Поэтому если мы захотим удалить созданный подобным образом обработчик, то нам надо не только удалить метод из кода формы в Form1.cs, но и удалить добавление обработчика в этом файле.

Содержание работы:

Задание 1. Разработать приложение, с помощью которого можно вычислить время падения тела с некоторой высоты при условии, что высота может задаваться в метрах, сантиметрах и дюймах.

1. Создайте новое Приложение Windows Forms. Имя проекта и приложения Padenie tela.
2. Разместите на форме компоненты в соответствии с рисунком.



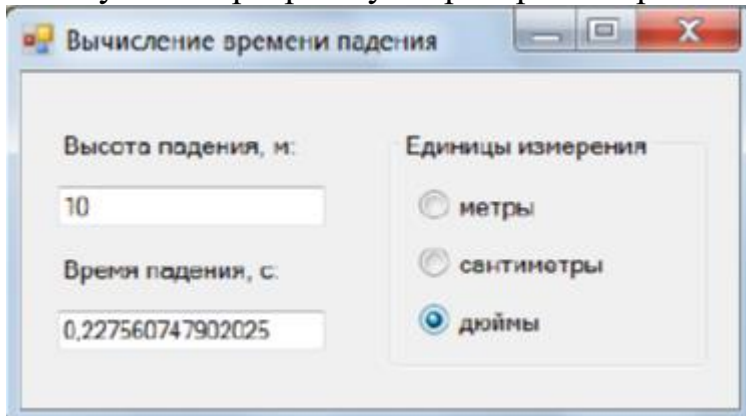
3. Для группы переключателей создайте свой собственный обработчик события `CheckedChanged`. Это событие возникает, когда пользователь изменяет состояние флажка, устанавливая или снимая отметку. Для того чтобы создать обработчик событий `rbChanged` для группы переключателей, отвечающих за выбор единицы измерения в расчете, необходимо выделить первый переключатель в группе. Затем на панели Свойства открыть вкладку События, в поле `CheckedChanged` ввести строку `rbChanged` и нажать клавишу `Enter` на клавиатуре. В результате будет создано тело обработчика событий `rbChanged`.

4. Введите следующий код в построенную заготовку обработчика:

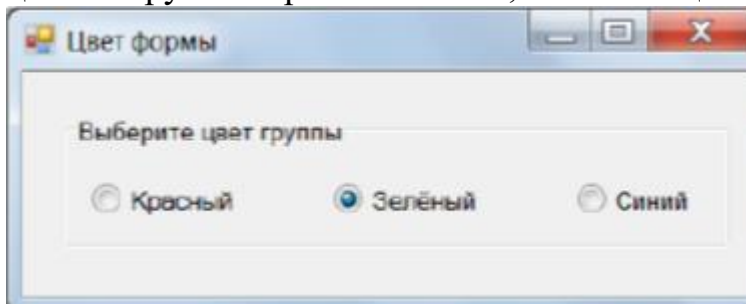
```
{
    double g = 9.81;
    double h;
    double t;
    h = Convert.ToDouble(textBox1.Text);
    RadioButton rb = (RadioButton)sender;
    switch (rb.TabIndex)
    {
        case 0:
        {
            t = Math.Sqrt(2 * h / g);
            textBox2.Text = Convert.ToString(t);
            break;
        }
        case 1:
        {
            t = Math.Sqrt(2 * h / 100 / g);
            textBox2.Text = Convert.ToString(t);
            break;
        }
        case 2:
        {
            t = Math.Sqrt(2 * h * 2.54 / 100 / g);
            textBox2.Text = Convert.ToString(t);
            break;
        }
    }
}
```

Данный код работает следующим образом: вначале он сохраняет в переменной `rb` идентификатор переключателя, состояние которого было изменено. Затем обработчик извлекает номер переключателя в группе. В зависимости от этого номера, обработчик выполняет расчет времени падения.

5. Выделите по очереди остальные переключатели и на вкладке События панели Свойства для события CheckedChanged выберите из списка событие rbChanged.
6. Запустите программу и проверьте её работоспособность.



Задание 2. Разработайте приложение, которое при выборе определенного цвета в группе переключателей, изменяет цвет формы.



ПРАКТИЧЕСКАЯ РАБОТА № 34

Тема: Создание процедур на основе событий

Цель работы: научиться создавать процедуры на основе событий, используя язык C#, при разработке Windows-приложений.

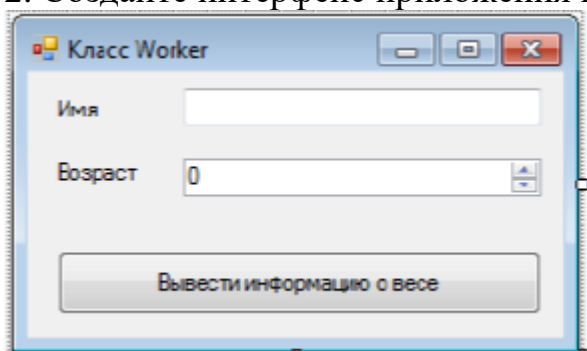
Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Создайте проект, для решения задачи с использованием классов: класс «Worker» поля: имя; возраст; вес (начальное значение 60). методы: «GetEat» - отображается вес; «SetEat» - если человек что-то съедает, то его вес увеличивается на количество съеденного.

1. Запустите среду программирования VisualStudio. Создайте новое Приложение Windows Forms. Имя проекта и приложения ClassApp1.

2. Создайте интерфейс приложения в соответствии с образцом:



3. Добавьте в проект новый класс Worker, для этого перейдите в Обозреватель решений и в контекстном меню проекта выполните команду Добавить – Класс. В окне Добавление нового элемента введите имя Worker.cs и нажмите кнопку Добавить.

Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В C# используется спецификация класса для построения объектов, которые являются экземплярами класса. При этом очень важно подчеркнуть, что класс является логической абстракцией. Физическое представление класса появится в оперативной памяти лишь после того, как будет создан объект этого класса.

При определении класса объявляются данные, которые он содержит, а также код, оперирующий этими данными. Если самые простые классы могут содержать только код или только данные, то большинство настоящих классов содержит и то, и другое. Объект можно рассматривать как совокупность свойств, методов и событий, работающий при этом как единое целое.

4. В класс добавьте два общедоступных поля: Имя и Возраст, а также одно скрытое поле: Вес.

```
class Worker
{
    public string имя;
    public int возраст;
    double вес = 60.00;
}
```

5. Для записи и чтения данных из скрытых полей используйте методы. Метод – это фрагмент программного кода, имеющий собственное имя и позволяющий значительно облегчить процесс программирования. Добавьте в класс Worker метод SetEat, который будет отвечать за еду: если человек что-то съест, то его вес должен будет увеличиться на количество съеденного. Если поле вес скрытое, то запись в него возможна и чтение из него тоже невозможно. Для чтения данных из скрытого поля необходимо использовать еще один метод GetEat.

```
class Worker
{
    public string имя;
    public int возраст;
    double вес = 60.00;
    public void SetEat(double eda)
    {
        вес += eda;
    }

    public double GetEat()
    {
        return вес;
    }
}
```

6. Далее используйте данные два метода в программе. Заставьте рабочего съесть 2 кг, а затем 3 кг пищи, а потом проверьте его вес. Для этого перейдите к окну формы и выполните двойной щелчок по кнопке Вывести информацию о весе. В обработчик события вставьте следующий код:

```
private void button1_Click(object sender, EventArgs e)
{
    Worker worker = new Worker();

    worker.имя = textBox1.Text;
    worker.возраст = (int)numericUpDown1.Value;

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.GetEat()));
    worker.SetEat(2);
    worker.SetEat(3);

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.GetEat()));
}
```

7. Запустите программу на выполнение. Проверьте её работоспособность.

8. Усовершенствуйте метод SetEat таким образом, что если рабочий за раз съедает более, чем 10 кг, то его возраст увеличивается на год, а вес увеличивается только наполовину от съеденного. Для этого измените код модуля следующим образом:

```
public void SetEat(double eda)
{
    if (eda > 10)
    {
        возраст++;
        вес += eda / 2;
    }
    else
        вес += eda;
}
```

9. Попросите рабочего съесть 15 кг. Для этого измените программный код кнопки следующим образом:

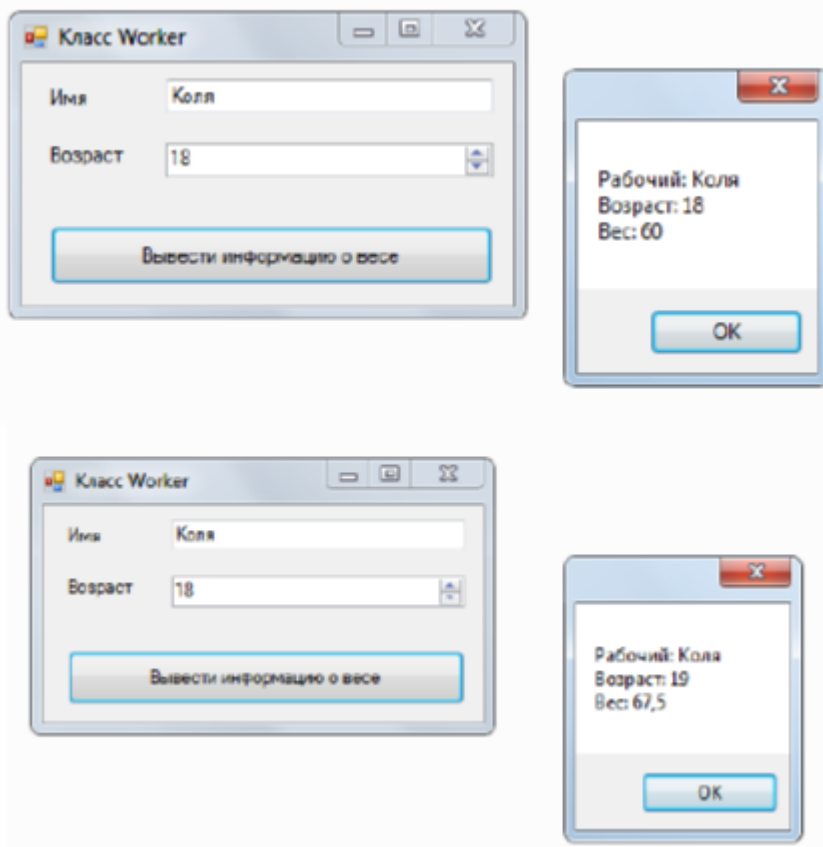
```
private void button1_Click(object sender, EventArgs e)
{
    Worker worker = new Worker();

    worker.имя = textBox1.Text;
    worker.возраст = (int)numericUpDown1.Value;

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.GetEat()));
    worker.SetEat(15);

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.GetEat()));
}
```

10. Запустите программу на выполнение. Проверьте её работоспособность.



Задание 2. Создайте новый проект ClassApp2 для решения следующей задачи с использованием классов.

Класс «Человек»

Поля: имя; возраст; вес, настроение (начальное значение=10).

Методы:

- «Информация о человеке» - отображаются имя, возраст и вес.
- «Есть» - если человек что-то съедает, то его вес увеличивается на количество съеденного.
- «Гулять» - настроение увеличивается на 1
- «Танцевать» - настроение увеличивается на 2

– «Работать» - настроение уменьшается на 2

1. Усовершенствуйте метод «Есть» так, чтобы при съедании за один раз более 5 кг, его возраст увеличивался бы на год, а вес увеличивался бы только на половину от съеденного.
2. Усовершенствуйте метод так, чтобы настроение не могло становиться отрицательным числом.
3. Дополните основную программу так, чтобы человек после еды четыре раза погулял и два раза потанцевал.
4. Добавьте в класс метод, который будет возвращать текущее настроение человека.
5. Добавьте в основную программу метод работать 9 раз и выведите настроение пользователя на экран. Измените метод работать таким образом, чтобы настроение никогда не было меньше нуля (т.е. если настроение было 1 и человек поработал, то оно должно стать не меньше 0).
6. Запустите программу на выполнение. Проверьте её работоспособность.

ПРАКТИЧЕСКАЯ РАБОТА № 35

Тема: Создание проекта с использованием кнопочных компонентов

Цель работы: сформировать навыки разработки приложений с использованием кнопочных компонентов в среде программирования Visual Studio, изучить особенности их использования.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

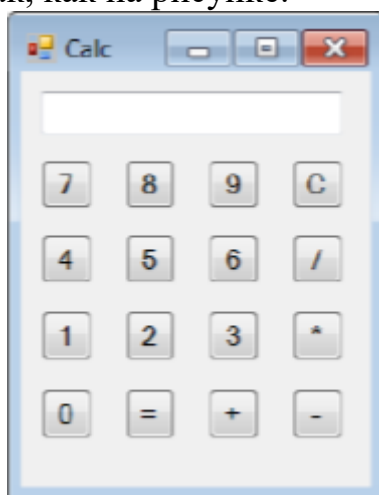
Задание 1. Используя кнопочные компоненты `button`, разработать программу – калькулятор, выполняющий простейшие действия.

1. Запустите среду программирования Visual Studio. Создайте новое Приложение Windows Forms. Имя проекта и приложения – Калькулятор.

2. Задайте для формы следующие свойства: `Text` – `Calc`, `Font - Size` – 10

3. Разместите на форме одно поле ввода, дайте ему имя `Disp` и очистите свойство `Text`.

4. Добавьте на форму 12 кнопок `button` для цифр, арифметических действий, знака «равно» и операции «сброс». Установите для всех кнопок размеры 30 на 30 пикселей и разместите их так, как на рисунке.



5. Дайте кнопкам-действиям имена `btnPlus`, `btnMinus`, `btnMul`, `btnDiv` (сложение, вычитание, умножение и деление), а кнопке «равно» — имя `btnCalc`.

6. Для действий и кнопки `C` установите жирный шрифт (свойство `Font - Bold`), а для кнопки `C` дополнительно — красный цвет шрифта.


7. Кнопка `C` должна просто стирать содержимое поля ввода `Disp`. То есть, нужно вызвать метод `Disp.Clear()`. Добавьте обработчик события `OnClick` для кнопки `C`.

8. Понятно, что когда пользователь щелкнул по кнопке-цифре, нужно добавить эту цифру в конец текста поля ввода. Добавьте обработчик события `OnClick` для кнопки `0`. Добавьте следующий код:

```
{  
    Disp.Text += 0;  
}
```

9. Добавьте аналогичный код для всех кнопок-цифр.

10. Сохраните программу и протестируйте работу кнопок.

11. При работе программы вы увидели, что с клавиатуры можно ввести буквы, которые нам совсем не нужны. Когда пользователь нажмет клавишу в поле ввода, возникает событие OnKeyPress, которое можно перехватить, установив соответствующий обработчик. Создайте обработчик события OnKeyPress для поля ввода Disp. Для этого перейдите на вкладку Form1 [Конструктор], выделите поле ввода Disp, на панели Свойства перейдите на вкладку События, щелкнув по значку . Найдите событие KeyPress и дважды щелкните мышью в поле справа от него, будет сгенерирована заготовка метода. В тело обработчика события добавьте следующий код:

```
private void Disp_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!char.IsDigit(e.KeyChar))
    {
        e.Handled = true;
    }
}
```

Для проверки ввода используется метод char.IsDigit, который возвращает true, если введенный символ является десятичной цифрой и false, если нет. Свойство Handled используется для определения того, было ли событие обработано. Установив значение Handled в true, событие ввода не будет передано операционной системе для обработки по умолчанию.

12. Запустите программу и попробуйте вводить буквы.

13. Далее необходимо организовать вычисления. Нам нужны две переменных для хранения чисел и одна символьная переменная, в которую будем записывать тип операции. Поскольку при расчетах могут получиться числа с дробной частью, для хранения чисел будем использовать вещественные переменные. В простейшем случае для хранения операции можно использовать переменную типа Char (один символ), но мы объявим ее как символьную строку, так как при доработке программы могут понадобиться и многосимвольные названия операций. Объявите в начале программы две вещественных переменные x1 и x2 типа Double и одну символьную строку oper.

```
public partial class Form1 : Form
{
    double x1;
    double x2;
    string oper;
```

14. Когда мы нажимаем на одну из кнопок-операций, нужно запомнить введенное число в переменной x1 и тип операции в переменной oper. Тип операции (надпись на кнопке) легко узнать, обратившись к свойству Text. Выделите кнопку + и создайте для нее обработчик события OnClick:

```
private void btnPlus_Click(object sender, EventArgs e)
{
    x1 = Convert.ToDouble(Disp.Text);
    oper = btnPlus.Text;
}
```

15. Создайте аналогичный обработчик для всех остальных кнопок-операций.

16. При нажатии на кнопку = нужно прочитать из поля ввода второе число и выполнить операцию. При этом первое число и тип операции уже должны находиться в переменных `x1` и `oper`. Поскольку в результате деления может получиться число с дробной частью, переменная для хранения результата (назовем ее `res`) тоже должна быть вещественной. Объявите в начале программы данную переменную.

17. Введите обработчик события `OnClick` для кнопки =:

```
private void btnCalc_Click(object sender, EventArgs e)
{
    double x2 = Convert.ToDouble(Disp.Text);
    if (oper == "+")
    {
        res = x1 + x2;
    }
    if (oper == "-")
    {
        res = x1 - x2;
    }
    if (oper == "*")
    {
        res = x1 * x2;
    }
    if (oper == "/")
    {
        res = x1 / x2;
    }
    Disp.Text = Convert.ToString(res);
}
```

18. Запустите программу и проверьте ее работу. Учтите, что для ввода второго числа нужно сначала очистить экран кнопкой «С».

19. Конечно, очень неудобно, что перед вводом второго числа нужно очищать поле ввода, нажимая на кнопку С. Хотелось бы делать это автоматически. Запустите стандартную программу Калькулятор и посмотрите, в какой момент стирается первое число. Наверное, вы увидели, что число из поля ввода автоматически стирается, когда после нажатия на кнопки-действия или кнопку = пользователь набирает новое число. Мы введем логическую переменную `newNumber`, которой будем присваивать значение `true` в том случае, если нужно начинать вводить новое число. Объявите логическую переменную `newNumber`.

20. В конце обработчиков события `OnClick` для кнопок-действий и кнопки равно добавьте строку: `newNumber = true`;

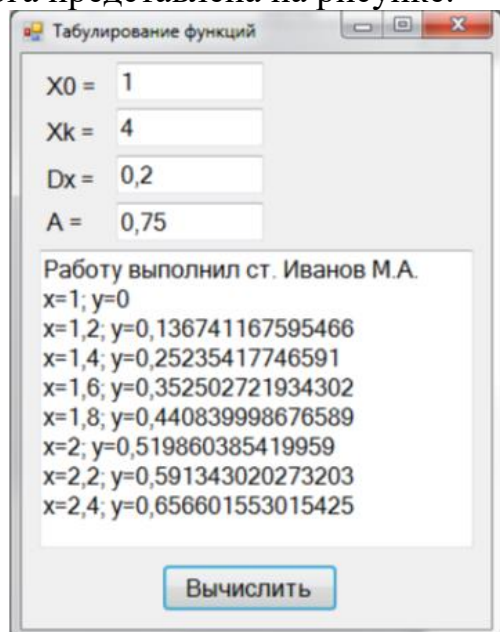
21. Очистку экрана будем делать перед вводом нового числа. Дополните обработчики кнопок-цифр. В начало обработчика события `OnClick` для кнопок-цифр добавьте код:

```
if (newNumber)
{
    Disp.Clear();
    newNumber = false;
}
```


22. Запустите программу и проверьте ее работу. После этого закройте проект.

Задание 2. Составьте программу, которая переводит суммы из рублей в доллары.

Задание 3. Вычислить и вывести на экран таблицу значений функции $y = a \cdot \ln(x)$ при x , изменяющемся от x_0 до x_k с шагом dx , a – константа. Панель диалога представлена на рисунке.



Текст обработчика нажатия кнопки Вычислить приведен ниже.

```
private void button1_Click(object sender, EventArgs e) {  
    // Считывание начальных данных  
    double x0 = Convert.ToDouble(textBox1.Text);  
    double xk = Convert.ToDouble(textBox2.Text);  
    double dx = Convert.ToDouble(textBox3.Text);  
    double a = Convert.ToDouble(textBox4.Text);  
    textBox5.Text = "Работу выполнил ст. Иванов М.А." + Environment.NewLine;  
    // Цикл для табулирования функции  
    double x = x0;  
    while (x <= (xk + dx / 2)) {  
        double y = a * Math.Log(x);  
        textBox5.Text += "x=" + Convert.ToString(x) + "; y=" + Convert.ToString(y) +  
            Environment.NewLine;  
        x = x + dx; } }  
}
```

После отладки программы следует проверить правильность работы программы с помощью контрольного примера. Установите точку останова на оператор перед циклом и запустите программу. После попадания на точку останова, выполните пошагово программу и проследите, как меняются все переменные в процессе выполнения.

ПРАКТИЧЕСКАЯ РАБОТА № 36

Тема: Создание проекта с использованием кнопочных компонентов

Цель работы: сформировать навыки разработки приложений с использованием кнопочных компонентов в среде программирования Visual Studio, изучить особенности их использования.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

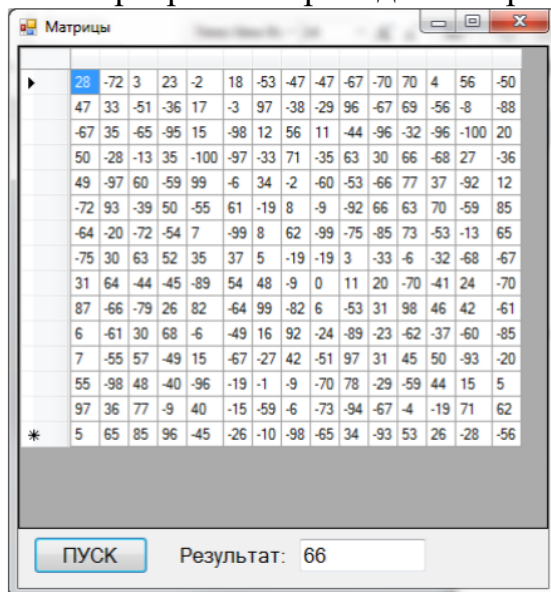
Справочный материал:

При работе с двумерными массивами ввод и вывод информации на экран удобно организовывать в виде таблиц. Элемент управления DataGridView может быть использован для отображения информации в виде двумерной таблицы. Для обращения к ячейке в этом элементе необходимо указать номер строки и номер столбца. Например: `dataGridView1.Rows[2].Cells[7].Value = "*";`

Этот код запишет во вторую строку и седьмой столбец знак звездочки.

Содержание работы:

Задание 1. Создать программу для определения целочисленной матрицы 15×15 . Разработать обработчик кнопки, который будет искать минимальный элемент на дополнительной диагонали матрицы. Результат вывести в текстовое поле. Окно программы приведено на рисунке.



Текст обработчика события нажатия на кнопку:

```
private void button1_Click(object sender, EventArgs e) {  
    dataGridView1.RowCount = 15; // Кол-во строк  
    dataGridView1.ColumnCount = 15; // Кол-во столбцов  
    int[,] a = new int[15,15]; // Инициализируем массив  
    int i,j;  
    //Заполняем матрицу случайными числами  
    Random rand = new Random();  
    for (i = 0; i < 15; i++)  
        for (j = 0; j < 15; j++)  
            a[i,j] = rand.Next(-100, 100);  
    // Выводим матрицу в dataGridView1
```

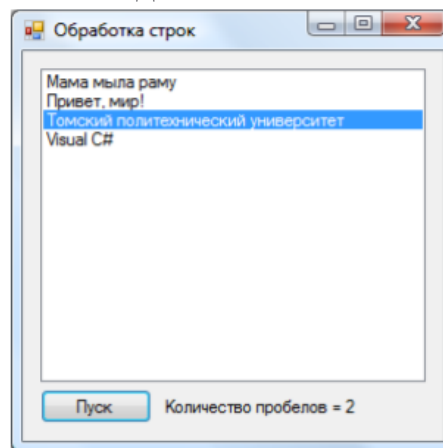
```

for (i = 0; i < 15; i++)
for (j = 0; j < 15; j++)
dataGridView1.Rows[i].Cells[j].Value = a[i, j].ToString();
// Поиск максимального элемента на дополнительной диагонали
int m = int.MinValue;
for (i = 0; i < 15; i++)
if (a[i, 14 - i] > m) m = a[i, 14 - i];
// выводим результат
textBox1.Text = Convert.ToString(m); }

```

Задание 2. Написать программу подсчета числа слов в произвольной строке. В качестве разделителя может быть любое число пробелов. Для ввода строк использовать ListBox. Строки вводятся на этапе проектирования формы, используя окно свойств. Вывод результата организовать в метку Label.

Панель диалога будет иметь вид:



Текст обработчика нажатия кнопки «Пуск»:

```

private void button1_Click(object sender, EventArgs e) {
// Получаем номер выделенной строки
int index = listBox1.SelectedIndex;
// Считываем строку в переменную str
string str = (string)listBox1.Items[index];
// Узнаем количество символов в строке
int len = str.Length;
// Считаем, что количество пробелов равно 0
int count = 0;
// Устанавливаем счетчик символов в 0
int i = 0;
// Организуем цикл перебора всех символов в строке
while (i < len) {
// Если нашли пробел, то увеличиваем счетчик пробелов на 1
if (str[i] == ' ')
count++;
i++; }
label1.Text = "Количество пробелов = " + count.ToString(); }

```

ПРАКТИЧЕСКАЯ РАБОТА № 37

Тема: Создание проекта с использованием кнопочных компонентов

Цель работы: сформировать навыки разработки приложений с использованием кнопочных компонентов в среде программирования Visual Studio, изучить особенности их использования.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Разместите на форме ряд кнопок (Button), и одно поле ввода (TextBox). Создайте обработчики события нажатия на данные кнопки, которые будут менять текст на нажатой кнопке. Текст на кнопке берется из поля ввода.

Задание 2. Разместите на форме две кнопки (Button) и одну метку (Label). Сделайте на кнопках следующие надписи: «скрыть», «показать». Создайте обработчики события нажатия на данные кнопки, которые будут скрывать или показывать метку. Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы».

Задание 3. Разместите на форме поле ввода (TextBox), метку (Label) и кнопку (Button). Создайте обработчик события нажатия на кнопку, который будет копировать текст из поля ввода в метку. Создайте обработчик события нажатия кнопки мышки на форме (Click), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы» и очищать поле ввода.

ПРАКТИЧЕСКАЯ РАБОТА № 38

Тема: Создание проекта с использованием компонентов стандартных диалогов и системы меню

Цель работы: овладение навыками создания меню различного типа и обработчиков сообщений пунктов меню; владение навыками создания и практического использования панели инструментов и строки состояния.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Создание главного меню приложения является фактически альтернативой выполнения команд элементами управления на форме в соответствии с программным кодом процедур, написанных для этих элементов. Однако, в отличие от элементов управления, которые осуществляют выполнение различных процедур при совершении различных событий над элементом управления, главное меню приложения позволяет создавать иерархию вложенных друг в друга меню команд любой степени сложности. Особенно эффективно создание главного меню приложения в том случае, когда в приложении необходимо выполнять множество различных команд. При этом команды, выполняющиеся из главного меню, будут собраны в одной строке главного меню, которое можно раскрыть в любой нужный момент для выполнения требуемой команды, не занимая много места в окне приложения на экране дисплея, где должна происходить основная обработка информации.

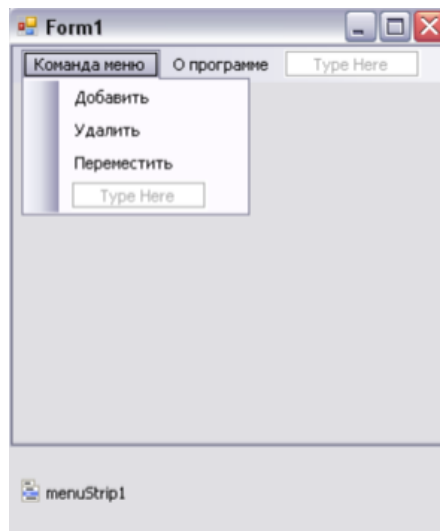
Главное и контекстное меню — это абсолютно различные вещи с точки зрения функционального назначения. В главное меню выносят все функции, которые выполняет программа. В любой момент пользователь может воспользоваться нужным пунктом меню для совершения какого-либо действия. С контекстным меню все по-другому. Оно должно включать лишь те пункты, которые соответствуют позиции вызова контекстного меню. Контекстное меню появляется при нажатии правой кнопки мыши.

Содержание работы:

Задание 1. Создать главное меню.

Для создания главного меню приложения Visual Studio имеет в панели ToolBox компонент MenuStrip. Создайте новое Windows Application C# приложение с именем MenuApp. Добавьте на форму компонент MenuStrip. В панели компонентов ниже основной формы приложения появится объект menuStrip1. В верхней части формы появится проект меню с единственным полем «Type Here».

Поле является редактируемым, если вы измените надпись в поле, то справа и снизу от него появятся дополнительные поля. Добавьте в меню пункты так, как показано на рисунке.



Пункт меню «Команда меню» содержит три подпункта: Добавить, Удалить, Переместить.

Кроме пункта «Команда меню» основное меню содержит пункт «О программе». Такой пункт обычно присутствует во всех коммерческих приложениях и предоставляет пользователю информацию о версии программы, разработчиках и т.д. Пункт меню «О программе» не имеет подпунктов.

Для лучшей читаемости программы измените свойство Name каждого пункта меню.

Команда меню — menuItemCommand;

Добавить — menuItemAdd;

Удалить — menuItemDel;

Переместить — menuItemMove;

О программе — menuItemAbout.

Компонент MenuStrip представлен в коде программы классом System.Windows.Forms.MenuStrip.

```
private System.Windows.Forms.MenuStrip menuStrip1;
```

Каждому пункту меню в коде программы соответствует объект класса ToolStripMenuItem.

```
private System.Windows.Forms.ToolStripMenuItem menuItemCommand;
```

```
private System.Windows.Forms.ToolStripMenuItem menuItemAdd;
```

```
private System.Windows.Forms.ToolStripMenuItem menuItemDel;
```

```
private System.Windows.Forms.ToolStripMenuItem menuItemMove;
```

```
private System.Windows.Forms.ToolStripMenuItem menuItemAbout;
```

Вот как происходит добавление элементов главного меню:

```
this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[]  
{ this.menuItemCommand, this.menuItemAbout });
```

Эти строки кода добавляют в объект menuStrip1 два пункта меню: «Команда меню» и «О программе». Свойство Items объекта MenuStrip имеет функцию AddRange, которая добавляет массив элементов ToolStripItem[] в меню.

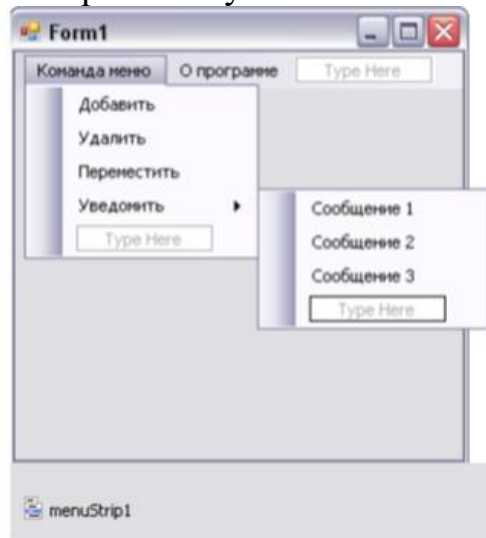
```
this.MenuItemCommand.DropDownItems.AddRange(new  
System.Windows.Forms.ToolStripItem[] {  
this.menuItemAdd,
```

```
this.menuItemDel,  
this.menuItemMove});
```

Затем, в пункт меню `menuItemCommand` («Команда меню») добавляется три пункта `menuItemAdd`, `menuItemDel`, `menuItemMove`. Теперь становится все очень понятно: для того чтобы внести изменения в какой-либо пункт меню, вам необходимо будет изменить соответствующий объект класса `ToolStripItem`.

Задание 2. Создать вложенное меню

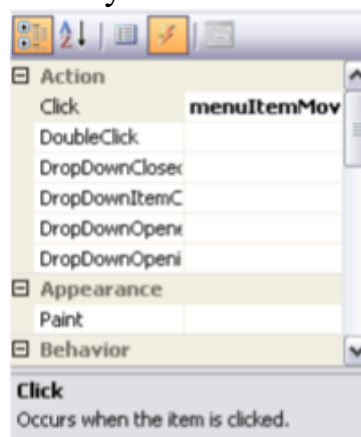
Подпункты меню тоже могут содержать вложенные элементы. Для того чтобы «вложить» элемент в пункт меню, необходимо добавить название в поле «Type Here», находящееся справа от пункта меню.



При этом с правой стороны пункта меню появится указательная стрелка, свидетельствующая о наличии вложенных пунктов меню. Добавьте в наше приложение пункт меню «Уведомить» и вложите в него пункты «Сообщение 1», «Сообщение 2», «Сообщение 3».

Задание 3. Написать обработчики сообщений меню

Основным сообщением пункта меню является `Click`. Это сообщение приходит, когда пользователь выбирает пункт меню. Давайте добавим обработчики к пунктам меню. Для этого необходимо щелкнуть два раза по полю с именем сообщения в окне свойств пункта меню.



Добавьте обработчики для пунктов меню «Добавить», «Удалить» и «Переместить». При этом в код программы добавятся три новых метода. Измените код этих методов так, как показано ниже:

```
private void menuItemAdd_Click(object sender, System.EventArgs e) {  
    // код для добавления  
    MessageBox.Show("Добавление"); }  
private void menuItemDel_Click(object sender, System.EventArgs e) {  
    // код для удаления  
    MessageBox.Show("Удаление"); }  
private void menuItemMove_Click(object sender, System.EventArgs e) {  
    // код для перемещения  
    MessageBox.Show("Перемещение"); }
```

Теперь при выборе одного из указанных выше пунктов меню на экране появится соответствующее сообщение. Кроме того, C# позволяет нескольким пунктам меню использовать один и тот же обработчик сообщения. Для того чтобы поэкспериментировать с такой возможностью, выделите сразу три пункта меню: «Сообщение 1», «Сообщение 2», «Сообщение 3», удерживая нажатой клавишу Ctrl.

Теперь в окне свойств щелкните два раза по событию Click. Таким образом, вы присвоите всем трем пунктам меню один и тот же обработчик. Посмотрите, как это выглядит в коде программы:

```
this.сообщение1ToolStripMenuItem.Click += new System.EventHandler  
(this.сообщение1ToolStripMenuItem_Click);  
this.сообщение2ToolStripMenuItem.Click += new System.EventHandler  
(this.сообщение1ToolStripMenuItem_Click);  
this.сообщение3ToolStripMenuItem.Click += new System.EventHandler  
(this.сообщение1ToolStripMenuItem_Click);
```

Для всех трех событий различных пунктов меню устанавливается один и тот же обработчик — `сообщение1ToolStripMenuItem_Click`. Напишите следующий код для функции `сообщение1ToolStripMenuItem_Click`:

```
private void сообщение1ToolStripMenuItem_Click(object sender, EventArgs e) {  
    ToolStripMenuItem item = (ToolStripMenuItem)sender;  
    string message = item.Text;  
    MessageBox.Show(message); }
```

Один из параметров любого обработчика события `object sender` — это объект, который послал сообщение. Поскольку все объекты в C# являются наследниками класса `System.Object`, то использование класса `object` в качестве параметра «универсализирует» использование обработчиков событий. В данном случае обработчик `сообщение1ToolStripMenuItem_Click()` получает события только от объектов класса `ToolStripMenuItem`. Поэтому можно смело приводить объект `sender` к классу `ToolStripMenuItem`.

```
ToolStripMenuItem item = (ToolStripMenuItem)sender;
```

Для того чтобы различить, от какого именно пункта меню пришло событие, считывается значение свойства `Text` пункта меню.

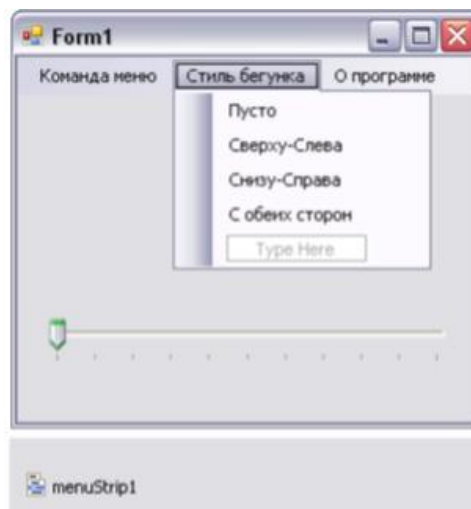
```
string message = item.Text;
```


Это свойство различно у всех пунктов меню. Поэтому можно вычислить, какой именно объект прислал сообщение. В данном примере просто выводится на экран сообщение с текстом пункта меню.

```
MessageBox.Show(message);
```

Задание 4. Создать контекстное меню

Поместите на форму приложения MenuApp компонент TrackBar. Расположите его по своему усмотрению. В главное меню приложения между пунктами «Команда меню» и «О программе» добавьте пункт «Стиль бегунка». Вложите в этот пункт четыре подпункта: «Пусто», «Сверху-слева», «Снизу-справа», «С обеих сторон».



Переименуйте пункты меню, изменив свойство Name каждого элемента:

- Стиль бегунка — menuItemTrackBar;
- Пусто — menuItemNone;
- Сверху -слева — menuItemTopLeft;
- Снизу -справа — menuItemBottomRight;
- С обеих сторон — menuItemBoth.

Для всех вышеописанных элементов меню присвойте один и тот же обработчик сообщения. Для этого выделите по очереди все четыре пункта меню, удерживая нажатой кнопку Ctrl. В окне свойств, на закладке Properties, щелкните два раза указателем мыши по событию Click. При этом ко всем пунктам меню добавится обработчик события menuItemNone_Click:

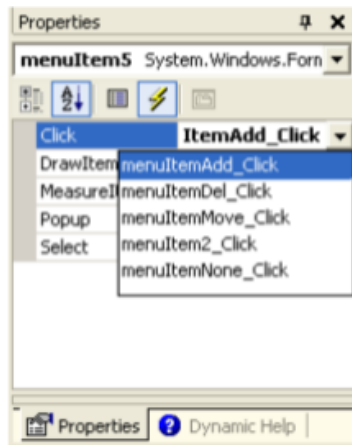
```
this.menuItemNone.Click += new System.EventHandler (this.menuItemNone_Click);  
this.menuItemTopLeft.Click += new System.EventHandler  
(this.menuItemNone_Click);  
this.menuItemBottomRight.Click+=new System.EventHandler  
(this.menuItemNone_Click);  
this.menuItemBoth.Click+= new System.EventHandler(this.menuItemNone_Click);
```

В конец кода программы добавится тело самой функции. Теперь давайте создадим два контекстных меню. Одно из них будет частично дублировать пункт основного меню «Команда меню», другое — пункт «Стиль бегунка». Для этого поместите на форму два компонента ContextMenuStrip: contextMenuStrip1 и

contextMenuStrip2. Выберите в панели компонент программы объект contextMenu1.

При этом в верхней части формы появится редактируемое поле. Выделите его указателем мыши. Ниже выделенного поля появится поле ввода с надписью «Type Here». Добавьте в контекстное меню пункты «Добавить», «Удалить» и «Переместить». Это контекстное меню будет соответствовать пункту основного меню «Команда меню».

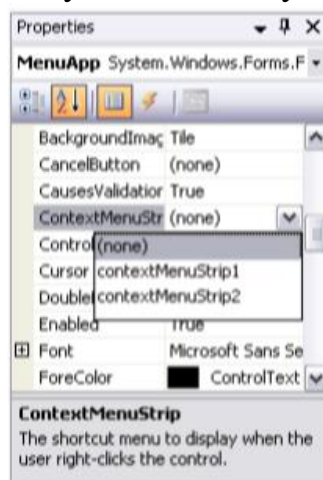
Теперь необходимо присвоить обработчики всем добавленным пунктам меню. Для пункта контекстного меню «Добавить» выберите из списка обработчиков события Click функцию menuItemAdd_Click.



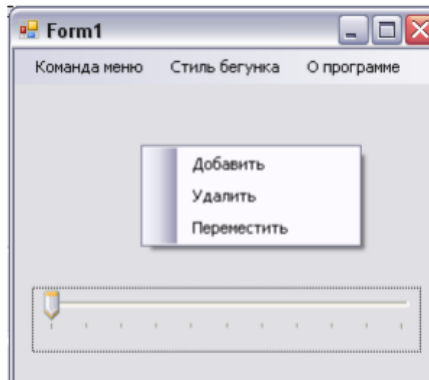
Таким образом, событию Click пункта контекстного меню «Добавить» присвоится обработчик события Click пункта основного меню «Добавить».

```
this.добавитьToolStripMenuItem.Click+= new System.EventHandler  
(this.menuItemAdd_Click);
```

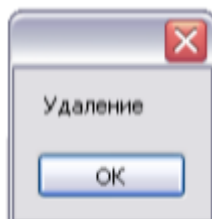
Тело функции-обработчика уже существует, поэтому при выборе любого из этих двух пунктов меню будет выполняться одно и то же действие. Установите для пунктов контекстного меню contextMenuStrip1 «Удалить» и «Переместить» обработчики menuItemDel_Click и menuItemMove_Click соответственно. После того необходимо установить contextMenuStrip1 контекстным меню для формы Form1. Для этого у объекта Form1 существует свойство ContextMenuStrip. Среда сама добавит в список ContextMenuStrip все необходимые элементы, которые могут быть присвоены этому свойству. В нашем случае список будет состоять из двух элементов.



Установите значение свойства ContextMenuStrip в contextMenuStrip1. Откомпилируйте и запустите программу. Щелкните правой кнопкой мыши по любому свободному месту на форме. На экране появится контекстное меню, которое вы создали.



Если вы выберете любой из пунктов контекстного меню, выполнится то же действие, которое соответствует аналогичным пунктам основного меню. Выберите пункт «Удалить» контекстного меню. На экране появится сообщение, изображенное на рисунке. Такое же сообщение появится, если вы выберете пункт «Удалить» основного меню.



Контекстное меню — это удобная возможность выполнять различные действия над объектами, не загромождая основное окно программы. Допустим, нам необходимо в программе изменять свойства элемента `TrackBar`. Предоставим пользователю возможность делать это как из основного меню программы, так и из контекстного меню. В основном меню программы у нас уже имеется пункт «Стиль бегунка», который содержит подпункты с названиями стилей. Давайте добавим в контекстное меню `contextMenuStrip2` те же подпункты. Для этого выделите объект `contextMenuStrip2` и добавьте в него поля: «Пусто», «Сверху-слева», «Снизу-справа», «С обеих сторон».

Теперь добавьте обработчик события `Click` для пунктов контекстного меню. Для этого выделите все четыре пункта меню, удерживая клавишу `Ctrl`, и выберите из предложенного списка для события `Click` обработчик `menuItemNone_Click`. Теперь все пункты `contextMenuStrip2` и соответствующие пункты основного меню имеют один и тот же обработчик — `menuItemNone_Click`.

Добавьте к этому обработчику код, который представлен ниже:

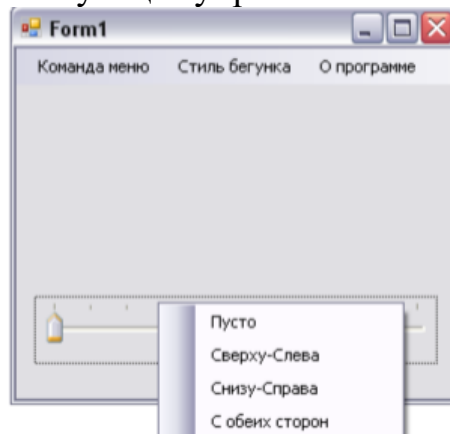
```
private void menuItemNone_Click(object sender, System.EventArgs e) {  
    ToolStripMenuItem item = (ToolStripMenuItem)sender;  
    string text = item.Text;  
    switch(text){  
    case "Пусто":  
        trackBar1.TickStyle = TickStyle.None; break;
```

```

case "Сверху-Слева":
trackBar1.TickStyle = TickStyle.TopLeft; break;
case "Снизу-Справа":
trackBar1.TickStyle = TickStyle.BottomRight; break;
case "С обеих сторон":
trackBar1.TickStyle = TickStyle.Both; break; } }

```

Для завершения функциональности программы установите свойство ContextMenuStrip компонента trackBar1 как contextMenuStrip2. Все, программа закончена. Откомпилируйте и запустите программу. Щелкните правой кнопкой мыши по изображению бегунка на форме. Вместо контекстного меню, которое появляется при нажатии правой кнопкой мыши на форме, вы увидите контекстное меню, соответствующее управлению стилями бегунка.



Попробуйте изменить стиль бегунка путем выбора различных пунктов контекстного меню. Местоположение черточек бегунка будет изменяться в зависимости от выбранного пункта меню. Те же самые операции можно выполнить из основного меню приложения. Давайте детальнее рассмотрим код функции-обработчика.

```
ToolStripMenuItem item = (ToolStripMenuItem)sender;
```

Присваиваем переменной item объект sender, от которого пришло сообщение. Поскольку заранее известно, что сообщения всегда посылаются от пунктов меню, выполняем приведение типа к ToolStripMenuItem. Объекты класса ToolStripMenuItem имеют свойство Text, которое в нашем случае характеризует пункт меню. Инструкция switch в языке C# поддерживает использование строк в качестве элементов для сравнения. Поэтому можно получить значения свойства Text и по нему идентифицировать пункт меню.

```

string text = item.Text;
switch(text)
//...

```

Далее выполняются несколько инструкций case, и по тексту пункта меню выставляется стиль элемента управления trackBar.

Задание 5. Добавить пометки пунктов меню

Очень удобная возможность пунктов меню — их пометка. Обычно это флажок слева от надписи пункта меню. Для пометки пункта меню используется свойство Checked класса ToolStripMenuItem. Это свойство типа bool. Если

значение `Checked` установлено в `true`, то флажок присутствует, если в `false` — то отсутствует. Давайте допишем программу так, чтобы текущий стиль элемента управления `trackBar1` в меню всегда был отмечен флажком. Для этого необходимо изменить код программы. Поместите в конец функции `menuItemNone_Click` следующие строки кода:

```
// проходим по всем подпунктам изменяющим стиль бегунка
// расположенным в основном меню программы
foreach (ToolStripMenuItem item1 in menuItemTrackBar.DropDownItems) {
// если текст меню совпадает с переданным параметром, то помечаем пункт меню
if (item1.Text == text)
item1.Checked = true;
// если текст меню не совпадает с переданным параметром,
// то снимаем пометку с пункта меню
else item1.Checked = false; }
// проходим по всем подпунктам, изменяющим стиль бегунка и
// расположенных в контекстном меню программы
foreach (ToolStripMenuItem item1 in contextMenuStrip2.Items) {
// если текст меню совпадает с переданным параметром, то помечаем пункт меню
if (item1.Text == text)
item1.Checked = true;
// если текст меню не совпадает с переданным параметром,
// то снимаем пометку с пункта меню
else item1.Checked = false; }
```

Для того чтобы программа с самого начала правильно функционировала, установите для пунктов меню «Снизу-Справа» свойство `Checked` в `true` по умолчанию, потому как именно этот стиль установлен по умолчанию для объекта `trackBar1`. Вы можете сделать это, выбрав нужный пункт меню и установив для него свойство `Checked` как `true` в окне свойств.

Запустите программу. Выберите в контекстном меню стиль «С обеих сторон». Откройте еще раз контекстное меню — пункт меню «С обеих сторон» будет помечен. Можете изменить стиль на любой другой, пометка всегда будет соответствовать выбранному стилю. То же самое произойдет и с пунктами основного меню: их пометка будет строго соответствовать пунктам контекстного меню.

Задание 2. Добавьте в приложение пункт меню «Ориентация» с двумя подпунктами «Горизонтальная» и «Вертикальная». С помощью данных пунктов изменяется расположение элемента `trackbar1`. Аналогичные команды добавьте и в контекстное меню для элемента `trackbar1`. Пометка пунктов меню должна соответствовать выбранному стилю элемента управления.

ПРАКТИЧЕСКАЯ РАБОТА № 39

Тема: Создание проекта с использованием компонентов стандартных диалогов и системы меню

Цель работы: овладение навыками создания меню различного типа и обработчиков сообщений пунктов меню; владение навыками создания и практического использования панели инструментов и строки состояния.









Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Панель инструментов ведет к отдельным функциональным возможностям приложения посредством одного щелчка мышью, что, вне всякого сомнения, требует меньших усилий, чем при работе с меню — пользователю намного легче щелкать мышью на постоянно видимой кнопке, чем осуществлять поиск по иерархии различных меню. Панелей инструментов в одном окне может быть несколько, они могут располагаться в любой части окна, их можно передвигать и скрывать.

Кнопки, расположенные на панели инструментов, обычно содержат рисунок без текста, хотя и существует возможность использовать кнопки, на которых будет изображено и то, и другое. В качестве панели с кнопками без текста можно привести панели в MS Word, а в качестве примера панелей с текстом — используемые в MS Internet Explorer. Если вы наведете курсор мыши на какую-либо кнопку и остановите его, то будет выведено некоторое пояснение относительно предназначения данной кнопки, особенно в тех случаях, когда на ней имеется только изображение и отсутствует какой-либо текст.

В отличие от управляющих элементов типа "меню", управляющий элемент ToolStrip не является просто контейнером для других объектов. Можно задать непосредственно некоторые его свойства, например, координаты его расположения на экране. Элементами на панели инструментов могут быть объекты следующих классов:

- 1) Кнопки  класса ToolStripButton;
- 2) Надписи  класса ToolStripLabel;
- 3) Кнопки с разделителем  класса ToolStripSplitButton;
- 4) Кнопки с выпадающим списком  ToolStripDropDownButton;
- 5) Разделительная линия  класса ToolStripSeparator;
- 6) Комбинированный список  класса ToolStripComboBox;
- 7) Текстовое поле  класса ToolStripTextBox;
- 8) Индикатор прогресса  класса ToolStripProgressBar.

Свойства элемента ToolStrip позволяют управлять тем, как и где этот элемент будет выводиться на экран. Они также регулируют некоторые установки, касающиеся вывода элементов панели инструментов на экран; такие установки являются едиными для всех элементов, содержащихся на данной панели.

Управляющий элемент StatusStrip (линейка состояния) обычно используется для предоставления дополнительных данных о выбранном

элементе или информации о действии, выполняемом в настоящий момент в рамках диалога. Обычно линейка состояния располагается у нижней границы экрана, как, например, в приложениях Ms Office, хотя ее можно расположить и в любом другом месте. Линейка состояния может использоваться для вывода следующих элементов: надписей, индикатора прогресса, кнопок.

Содержание работы:

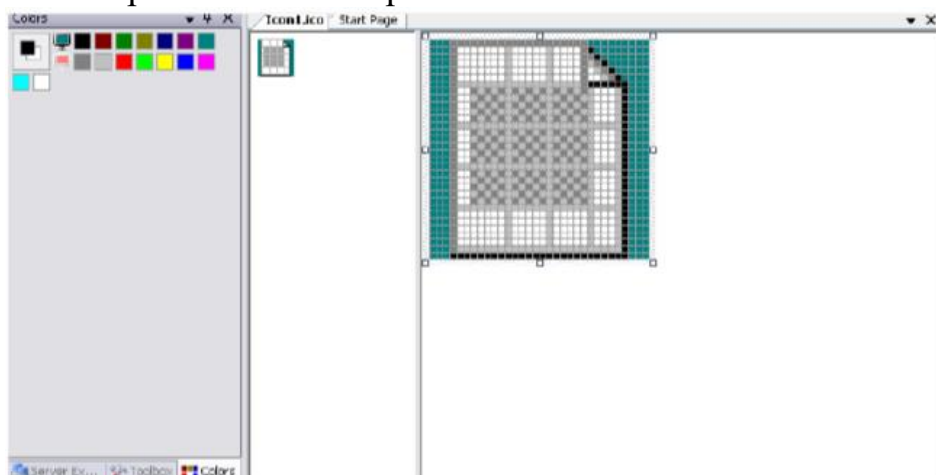
Задание 1. Работа с редактором изображений

Создайте новое Windows Application приложение с именем GraphEditorApp. Измените свойство Text формы на «Графический редактор».

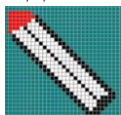
Начнем проектирование панели инструментов приложения. Панель инструментов представлена в C# классом ToolStrip. Панель инструментов может содержать в себе различные элементы управления: кнопки, списки и даже диалоги.

Основной характеристикой кнопок является их изображение. Как правило, кнопка в панели инструментов хранит пиктографическое изображение того действия, которое совершается по нажатию кнопки. Для того чтобы создать ToolStrip, вам необходимо предварительно подготовить изображения для его кнопок.

Для работы с изображениями в среде разработки существует Image Editor. Он позволяет создать изображения с использованием простейших инструментов. Для создания файла с изображением вам необходимо воспользоваться пунктом меню File/New/File... В появившемся окне New File выберите тип файла Icon File. Перед вами появится пустое изображение с дополнительной панелью управления. Кроме того, в основном меню появится новый пункт меню Image. Выберите в меню пункт Image/Show Colors Window. На экране вы увидите панель с отображением палитры компонент.

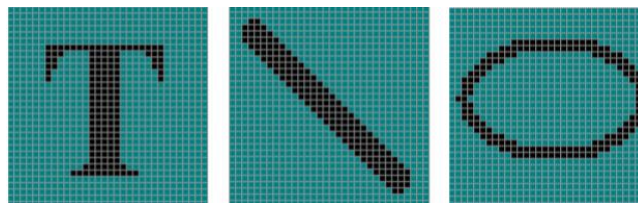


Перед вами появится пустое изображение с дополнительной панелью управления. Кроме того, в основном меню появится новый пункт меню Image. Выберите в меню пункт Image/Show Colors Window. На экране вы увидите панель с отображением палитры компонент. Создайте изображение элемента «Карандаш».



По умолчанию Visual Studio создает пиктограмму с двумя изображениями 16x16 пикселей и 32x32 пикселей. Вы рисовали изображение размером 32x32. Для того чтобы избежать неприятностей при дальнейшем использовании пиктограммы, необходимо удалить изображение 16x16 из файла. Для этого выберите из меню Image/Current Icon Image Types/16x16. В рабочей области появится новое изображение. Удалите его, используя меню Image/Delete Image Type.

Сохраните изображение в файл с именем «Карандаш.ico». Кроме элемента «Карандаш» вам необходимо создать пиктограммы для инструментов «Текст», «Линия» и «Эллипс». Примеры пиктограмм представлены ниже. Не забудьте удалить пиктограммы размером 16x16 и из этих файлов. Сохраните созданные изображения в файлы с именами «Текст.ico», «Линия.ico» и «Эллипс.ico», соответственно.

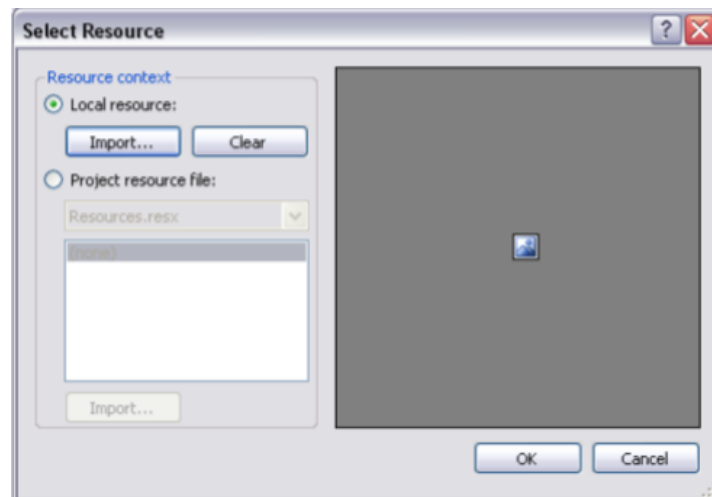


Задание 2. Добавить на форму панель инструментов

Добавьте на форму компонент ToolStrip. На форме появится пустая панель инструментов.



Добавьте к панели инструментов четыре кнопки класса ToolStripButton. Для этого в окне свойств объекта toolStrip1 необходимо настроить свойство Items. Нажмите на кнопку с тремя точками в поле Items. Откроется окно Items Collection Editor. Добавьте четыре кнопки и назовите их toolStripButtonPen, toolStripButtonText, toolStripButtonLine, toolStripButtonEllipse, изменив свойство Name каждого элемента в правом окне редактора. Установите изображения для каждой кнопки панели инструментов. Для этого выберите свойство кнопки Image, нажмите на кнопку с тремя точками в этом поле. Появится окно выбора изображения.



Нажмите кнопку Import, и выберите нужный файл с иконкой. Прделайте это для всех четырех кнопок. В итоге у вас получится панель инструментов, соответствующая рисунку.



Вы уже можете запустить программу и посмотреть результаты своей работы. Программа будет содержать только панель инструментов, которая не выполняет пока никаких функций.

Добавьте в программу перечисление (enum), которое будет содержать режимы работы программы (инструменты). Для этого выше описания класса Form1 добавьте описание перечисления.

```
public partial class Form1 : Form {
    public enum Tools{
        PEN=1, TEXT, LINE, ELLIPSE, NULL=0 }
    public Form1()
```

В классе Form1 создайте объект перечисления. Tools curentTool;

Этот объект будет содержать выбранный в панели инструментов инструмент для рисования. Впоследствии программа будет использовать эту переменную для определения того, какой инструмент выбран текущим. Для того чтобы завершить работу с панелью инструментов приложения, добавьте обработчик события ItemClicked объекта toolStrip1, щелкнув два раза указателем мыши по имени события ItemClicked на закладке событий в окне свойств. При этом в программу добавится функция toolStrip1_ItemClicked как обработчик события, происходящего при нажатии кнопки на панели инструментов.

Измените метод toolStrip1_ItemClicked и добавьте дополнительный код, как показано ниже:

```
private void toolStrip1_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    switch (e.ClickedItem.Name) {
        case "toolStripButtonPen":
            curentTool = Tools.PEN; break;
        case "toolStripButtonText":
```

```

        curentTool = Tools.TEXT; break;
case "toolStripButtonLine":
    curentTool = Tools.LINE; break;
case "toolStripButtonEllipse":
    curentTool = Tools.ELLIPSE; break; }
SetToolStripButtonsPushedState(e.ClickedItem); }
private void SetToolStripButtonsPushedState(ToolStripItem button){
foreach (ToolStripButton btnItem in toolStrip1.Items) {
if (btnItem == button){btnItem.Checked =true; }
else {btnItem.Checked = false;
}} }

```

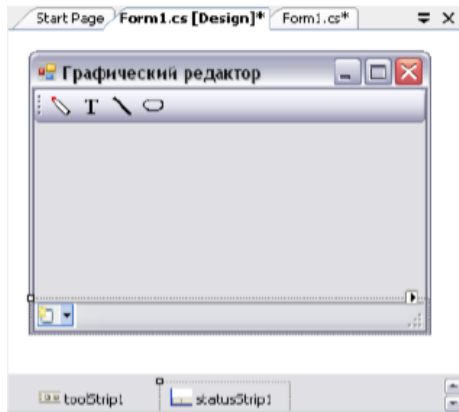
Обработчик нажатия кнопок на панели инструментов — функция `toolStrip1_ItemClicked` — включает в себя инструкцию `switch`, которая позволяет определить, какая именно кнопка была нажата. Вторым параметром функции-обработчика является переменная типа `ToolStripItemClickedEventArgs`.

Класс `ToolStripItemClickedEventArgs` имеет свойство `ClickedItem`, которое соответствует нажатой кнопке. Если свойство `Name` кнопки совпадает с искомым в инструкции `case`, то будет выполняться соответствующая операция. В соответствии с нажатой кнопкой выставляется режим работы программы (текущий выбранный инструмент).

Функция `SetToolStripButtonsPushedState` предназначена для контроля состояния кнопок панели инструментов. Если вы запускали программу до изменения вышеприведенного кода, то при нажатии кнопок на панели инструментов они нажимались независимо друг от друга. Но для нашего приложения свойственно существование только одного режима в один момент времени. Поэтому может быть нажатой только одна из кнопок. Функция `SetToolStripButtonsPushedState` следит за состоянием нажатия кнопок. При помощи цикла `foreach` функция просматривает все кнопки панели инструментов. Та из кнопок, которая совпадает с переданным параметром, делается вдавленной, свойство `Checked` устанавливается в `true`. Для всех остальных кнопок свойство `Checked` устанавливается в `false`. Вызов функции `SetToolStripButtonsPushedState` происходит из обработчика нажатия кнопок `toolStrip1_ItemClicked` с параметром, соответствующим нажатой кнопке. Поэтому нажатая кнопка будет вдавлена, а все остальные нет. Теперь при запуске программы вы увидите, что только одна из кнопок может быть вдавлена в один момент времени. Как только вы нажмете другую кнопку, предыдущая вдавленная примет свое обычное состояние.

Задание 3. Добавить на форму строку состояния

Пусть информацию о выбранном инструменте рисования необходимо отображать в строке состояния. Добавьте на форму компонент `StatusStrip`. На форме появится пустая строка состояния.



Добавьте к строке состояния надпись — элемент класса ToolStripStatusLabel. Установите свойство Text этого элемента в значение «Не выбрано ни одного графического инструмента». При нажатии на кнопку панели инструментов необходимо изменить свойство Text надписи. Для этого внесите такие изменения в функцию, обрабатывающую нажатия кнопок на панели инструментов.

```
private void toolStrip1_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    switch (e.ClickedItem.Name) {
        case "toolStripButtonPen":
            currentTool = Tools.PEN;
            statusStrip1.Items[0].Text = "Выбран карандаш";           break;
        case "toolStripButtonText":
            currentTool = Tools.TEXT;
            statusStrip1.Items[0].Text = "Создание надписей";         break;
        case "toolStripButtonLine":
            currentTool = Tools.LINE;
            statusStrip1.Items[0].Text = "Рисование линий";           break;
        case "toolStripButtonEllipse":
            currentTool = Tools.ELLIPSE;
            statusStrip1.Items[0].Text = "Рисование эллипса";         break; }
    SetToolStripButtonsPushedState(e.ClickedItem); }

```

Задание 4. Создайте на форме графического редактора меню с пунктом «Инструмент», в который добавьте пункты Карандаш, Текст, &Линия, Эллипс. Свойство Name для них установите, как:

Карандаш — menuItemPen;

Текст — menuItemText;

Линия — menuItemLine;

Эллипс — menuItemEllipse.

Эти пункты меню будут предназначены для выбора режима работы программы, точно так же, как это делается из панели инструментов. Сделайте подпункты меню Инструмент помечаемыми (Checked), аналогично кнопкам на панели инструментов. Нужно учесть, что выбранным может быть только один пункт меню. Сохраните проект для дальнейшей работы с ним.

ПРАКТИЧЕСКАЯ РАБОТА № 40

Тема: Создание проекта с использованием компонентов стандартных диалогов и системы меню

Цель работы: овладение навыками создания меню различного типа и обработчиков сообщений пунктов меню; владение навыками создания и практического использования панели инструментов и строки состояния.

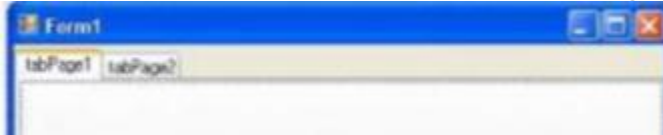
Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

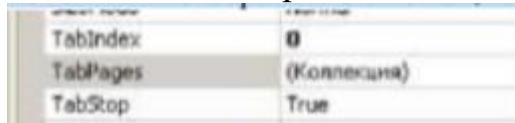
Задание 1. Создайте Windows-приложение для изменения цвета фона формы через пункты меню (например, Красный, Синий, Белый) и кнопки панели инструментов. Название цвета фона должно выводиться в строке состояния.

Задание 2. Создать программу «Тест»

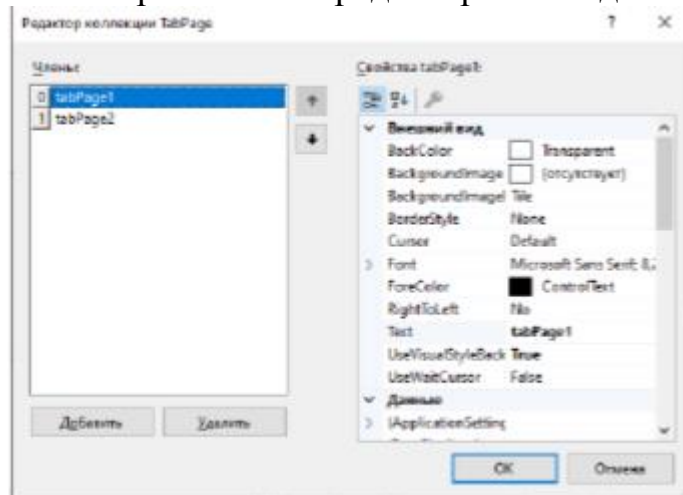
Чтобы настроить вкладки элемента TabControl используем свойство TabPages. При переносе элемента TabControl с панели инструментов на форму по умолчанию создаются две вкладки - tabPage1 и tabPage2.



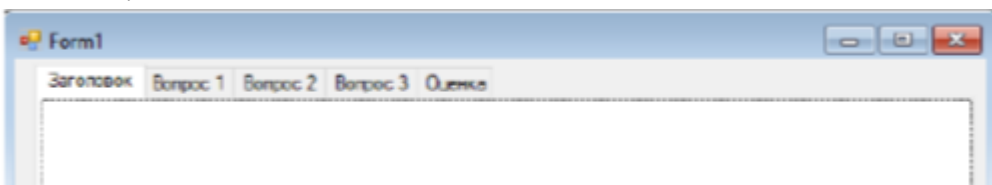
Изменим их отображение с помощью свойства TabPages:



Нам откроется окно редактирования/добавления и удаления вкладок:



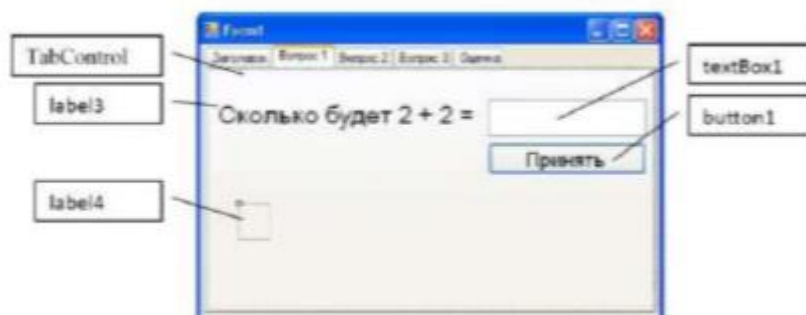
Каждая вкладка представляет своего рода панель, на которую мы можем добавить другие элементы управления, а также заголовок, с помощью которого мы можем переключаться по вкладкам. Текст заголовка задается с помощью свойства Text.



На основе этих вкладок создадим простейшую программу тест.

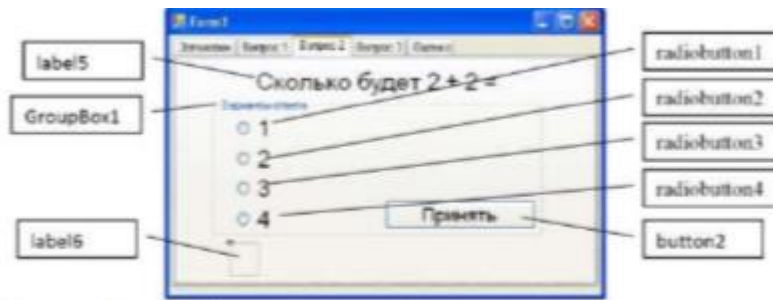
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApp5
{
    public partial class Form1 : Form
    {
        // Опишем глобальную переменную для хранения количества правильных
        // ответов b
        int b = 0;
        public Form1()
        {
            InitializeComponent();
            // Добавим команду, вывода формы в центре экрана.
            this.StartPosition = FormStartPosition.CenterScreen;
        }
        private void tabPage2_Click(object sender, EventArgs e)
        {
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text == "4")
            {
                label1.Text = "Правильно";
                b = b + 1;
            }
            else label1.Text = "Неправильно";
            // прячем кнопку что бы исключить повторный ввод (угадывание ответа)
            button1.Visible = false;
        }
    }
}
```

Разместить на вкладке следующие элементы. Номера элементов могут быть другие. Все зависит от количества ранее установленных элементов.



Процедура «Принять» будет иметь следующий вид (см. функцию button1_Click).

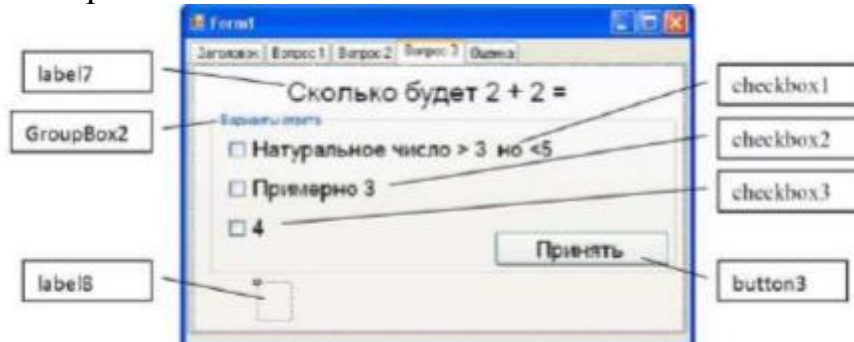
Выбирается один вопрос при помощи радиокнопок.



Процедура «Принять» будет иметь следующий вид:

```
private void button2_Click(object sender, EventArgs e) {
    if (radioButton4.Checked == true) {
        label6.Text = "Правильно";
        b = b + 1;
    }
    else label6.Text = "Неправильно";
    button2.Visible = false;
}
```

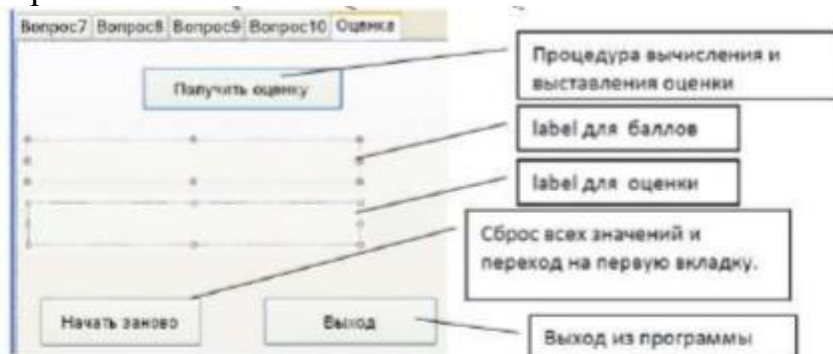
Выбираем несколько ответов.



Процедура «Принять» будет иметь следующий вид:

```
private void button3_Click(object sender, EventArgs e) {
    if (checkBox1.Checked == true && checkBox3.Checked == true &&
        checkBox2.Checked == false) {
        label8.Text = "Правильно";
        b = b + 1;
    }
    else label8.Text = "Неправильно";
    button3.Visible = false;
}
```

Так как галочки можно поставит на всех checkbox, то проверяем их все, не поставлено ли лишних галочек. На вкладке «Оценка» предлагаем три возможных варианта:



Процедура «Получить» оценку может выглядеть так:

```
label9.Text = "Набрано баллов=" + Convert.ToString(b);
```



```

if (b == 10) label10.Text = "Оценка 5 (отлично)";
if (b == 9 || b == 8 || b == 7) label10.Text = "Оценка 4(хорошо)";
if (b == 6 || b == 5) label10.Text = "Оценка 3(удовлетворительно)";
if (b == 4 || b == 3) label10.Text = "Оценка 2(плохо)";
if (b == 2 || b == 1) label10.Text = "Оценка 1(все ужасно)";

```

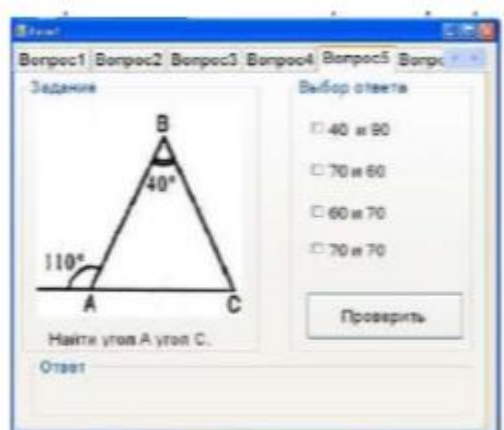
Переход в начало потребует очистки всех введенных пользователем данных:

```

// переходим на первую вкладку
tabcontrol1.SelectedIndex = 0;
// показываем все кнопки «Проверить» обратно.
button1.Visible = true;
button2.Visible = true;
...
button10.Visible = true;
// скрываем надписи правильно неправильно
label20.text = "";
...
label2.text = "";
// убираем все галочки кнопочки и надписи в полях ответа
checkBox1.checked= false;
...
checkBox12.checked:= false;
textBox1.Text = "";
...
textBox3.Text = "";
radioButton1.checked= false;
...
radioButton14.checked= false;
// обнуляем балы и изменяем соответствующие надписи
b = 0;
label21.Text = "Набрано баллов= ";
label22.Text = "";

```

В вопросе можно использовать картинки. Пример оформления вопроса:



Придумайте 10 вопросов по любой теме и создайте тест.

ПРАКТИЧЕСКАЯ РАБОТА № 41

Тема: Разработка функциональной схемы работы приложения

Цель работы: изучить процесс разработки функциональной схемы работы приложения на языке C#, освоить основные принципы проектирования и реализации программных модулей, а также научиться составлять диаграммы и схемы, отражающие логику работы приложения

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Функциональная схема – это схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств. Для изображения функциональных схем используют специальные обозначения.

Сначала определяются состав и подчинённость функций, а затем — набор программных модулей, реализующих эти функции. Однотипные функции реализуются одними и теми же модулями, функция верхнего уровня обеспечивается главным модулем, который управляет выполнением нижестоящих функций

Виды схем:

- *Обобщённая схема алгоритма* — раскрывает общий принцип функционирования алгоритма и основные логические связи между отдельными модулями на уровне обработки информации (ввод и редактирование данных, вычисления, печать результатов и т. п.).
- *Детальная схема алгоритма* — представляет содержание каждого элемента обобщённой схемы с использованием управляющих структур в блок-схемах алгоритма, псевдокода либо алгоритмических языков высокого уровня.

Содержание работы:

Задание 1. Разработать программу, которая рассчитывает стоимость с учётом скидки. Пользователь вводит информацию о скидке, наименовании материалов, количестве и стоимости, а программа рассчитывает общую стоимость по каждому пункту, стоимость с учётом скидки и суммарный итог.

Нарисовать блок-схему, отражающую логику работы приложения, используя любую онлайн-программу для создания диаграмм. В схеме должны быть обозначены основные модули: интерфейс пользователя, обработка данных, хранение информации. Обозначить потоки данных между модулями

Задание 2. Начертите функциональную блок-схему для игры в крестики-нолики, отражающую логику работы приложения, используя любую онлайн-программу для создания диаграмм. В схеме должны быть обозначены основные модули: интерфейс пользователя, обработка данных, хранение информации. Обозначить потоки данных между модулями

ПРАКТИЧЕСКАЯ РАБОТА № 42

Тема: Разработка функциональной схемы работы приложения

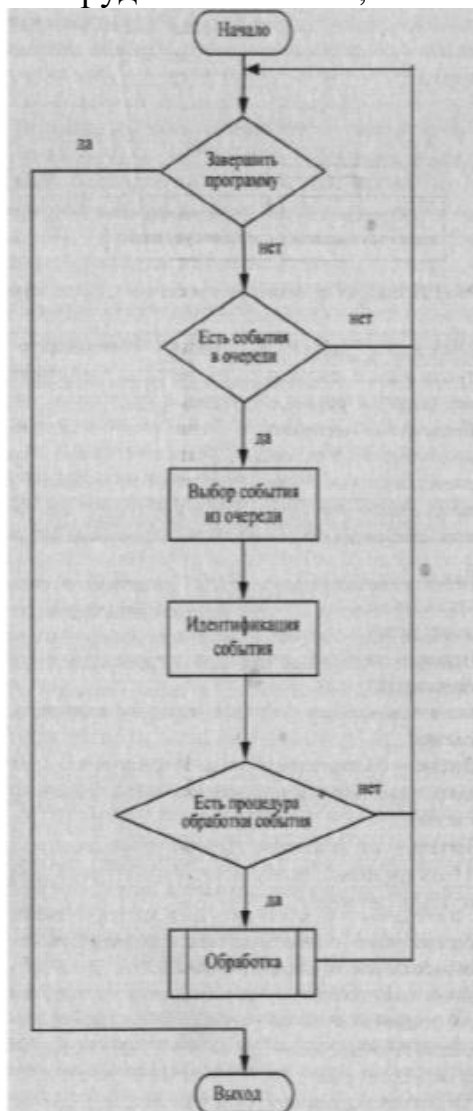
Цель работы: изучить процесс разработки функциональной схемы работы приложения на языке C#, освоить основные принципы проектирования и реализации программных модулей, а также научиться составлять диаграммы и схемы, отражающие логику работы приложения

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Доработайте функциональную схему системы комплексной автоматизации лечебно-профилактических учреждений (поликлиника).

Состав: Регистратура, Врач поликлиники, Отделение стационара, – Параклиника - это служба, входящая в состав лечебно-профилактического учреждения и обеспечивающая проведение различных исследований (клинических и других видов анализов и процедур), Рецепт, Аптека, Ведение листов нетрудоспособности, Расчеты / Статистика



Задание 2. Разработать функциональную схему работы приложения «Библиотека». Обязательные блоки: БД книг, БД читателей, Подсистема поиска, Справочная подсистема и т.д.

ПРАКТИЧЕСКАЯ РАБОТА № 43

Тема: Разработка функциональной схемы работы приложения

Цель работы: изучить процесс разработки функциональной схемы работы приложения на языке C#, освоить основные принципы проектирования и реализации программных модулей, а также научиться составлять диаграммы и схемы, отражающие логику работы приложения

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Разработать функциональную схему работы приложения «Продажа бытовой техники». Обязательные блоки: БД товара, БД покупателей, Подсистема поиска, Поставщики товара и т.д. В схеме должны быть обозначены основные модули: интерфейс пользователя, обработка данных, хранение информации. Обозначить потоки данных между модулями

Задание 2. Разработать функциональную схему работы приложения «Автосервис». Обязательные блоки: БД услуг, БД клиентов (регистрационные данные транспорта), Склад запчастей и т.д. В схеме должны быть обозначены основные модули: интерфейс пользователя, обработка данных, хранение информации. Обозначить потоки данных между модулями

ПРАКТИЧЕСКАЯ РАБОТА № 44

Тема: Разработка оконного приложения с несколькими формами

Цель работы: научиться создавать приложения с несколькими формами в среде Visual Studio на языке C#.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Создание проекта. Нужно открыть Visual Studio, выбрать «Создать новый проект», выбрать «Windows Forms App (.NET Framework)» и нажать «Далее».

Работа с формой. После создания проекта откроется главная форма приложения (Form1) — основное окно. Можно добавить элементы управления на форму (Button, Label, TextBox) и настроить их свойства в окне «Свойства».

Взаимодействие между формами. Например, можно реализовать, что первая форма по нажатию на кнопку вызывает вторую форму. Для этого нужно создать объект второй формы и вызвать метод Show для её отображения на экране.

Добавить ещё одну форму в проект. Для этого нужно нажать на имя проекта в окне Solution Explorer (Обозреватель решений) правой кнопкой мыши и выбрать «Добавить» → «Windows Form...». Новой форме можно дать имя, например, Form2.cs.

Реализовать взаимодействие между формами. Например, если вторая форма не знает о существовании первой, нужно передать сведения о первой форме в конструкторе. Для этого можно использовать ключевое слово `this` — ссылку на текущий объект (объект Form1).

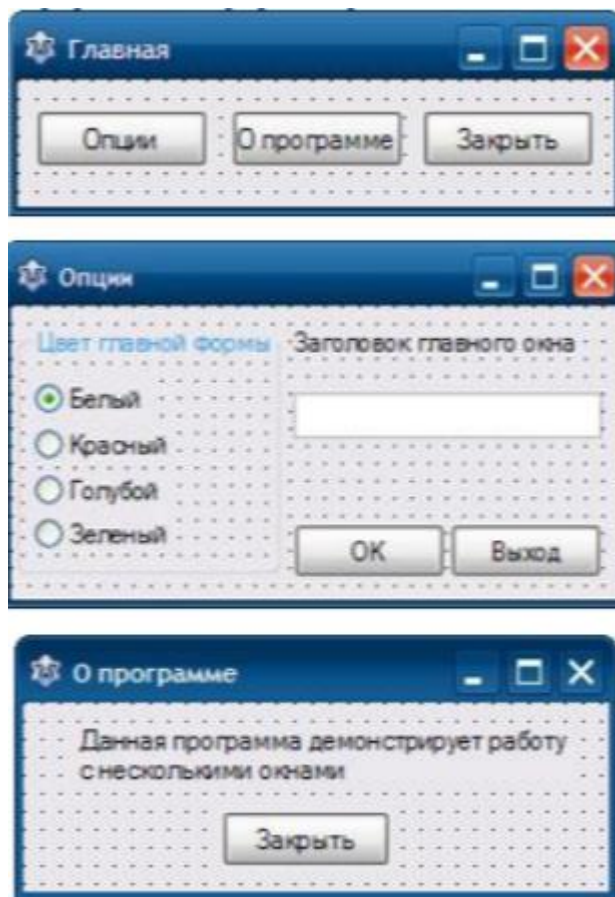
Учитывать, что одна из форм — главная — она запускается первой в файле Program.cs. Если одновременно открыта несколько форм, то при закрытии главной закрывается всё приложение и вместе с ним все остальные формы.

Код вызова второй формы в обработчике события нажатия кнопки на первой форме. Например, если вторая форма называется Form2, код вызова:
`Form2 newForm = new Form2(); newForm.Show()`

Код, в котором вторая форма получает ссылку на первую форму в конструкторе. Например, если нужно, чтобы вторая форма воздействовала на первую, код первой формы, где вызывается вторая форма, можно изменить на:
`Form2 newForm = new Form2(this); newForm.Show()`

Содержание работы:

Задание 1. Создать приложение с тремя формами: Главная, Опции и О программе. Форму Опции вызывать в обычном окне. Для вызова формы О программе использовать модальное окно. На рисунке показаны главная форма и подформы проекта.



Задание 2. В зрительном зале 25 рядов, в каждом из которых 36 мест (кресел). Информация о проданных билетах хранится в двумерном массиве, номера строк которого соответствуют номерам рядов, а номера столбцов — номерам мест. Если билет на то или иное место продан, то соответствующий элемент массива имеет значение 1, в противном случае — 0 Составить программу, определяющую число проданных билетов на места в 12-м ряду.

ПРАКТИЧЕСКАЯ РАБОТА № 45

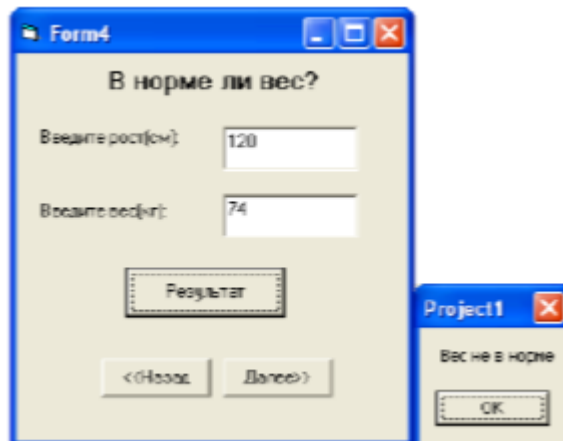
Тема: Разработка оконного приложения с несколькими формами

Цель работы: научиться создавать приложения с несколькими формами в среде Visual Studio на языке C#.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Составить программу определения, в норме ли вес обследуемого пациента (нормой считается вес, равный $(\text{рост(см)} - 100) \pm 5 \text{ кг}$).



Задание 2. Создать приложение с несколькими формами.

- Главная форма с меню.
- Форма для определения режимов работы программы.
- Форма «О программе» (About).
- Форма для ввода исходных данных и вывода результата.

В момент запуска приложения на экране появляются первые две формы, остальные появляются при выборе соответствующего пункта меню.

ПРАКТИЧЕСКАЯ РАБОТА № 46

Тема: Разработка оконного приложения с несколькими формами

Цель работы: научиться создавать приложения с несколькими формами в среде Visual Studio на языке C#.

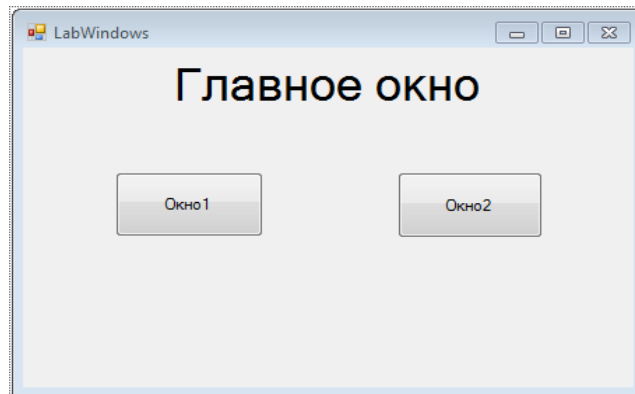
Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

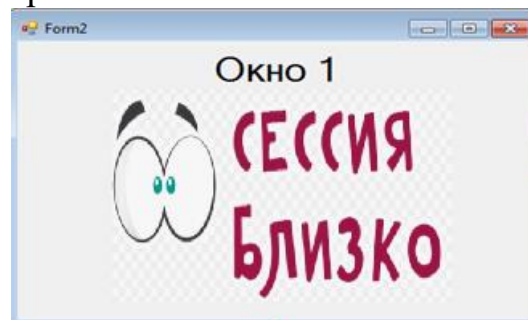
Задание 1. Создайте программу, в которой предусмотрена работа с несколькими окнами, организация парольного доступа к скрытым окнам и вывод сообщений об ошибках через MessageBox.

Например, программа, демонстрирующая работу с многооконными программами и организацию парольного доступа к окнам, а также работу с MessageBox. В программе используются компоненты Button, TextBox и PictureBox. Все эти компоненты можно найти в "Панели элементов" вкладка "Стандартные элементы управления".

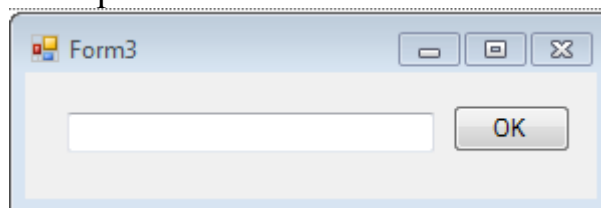
При запуске программы открывается "Главное окно" (форма 1) с двумя кнопками.



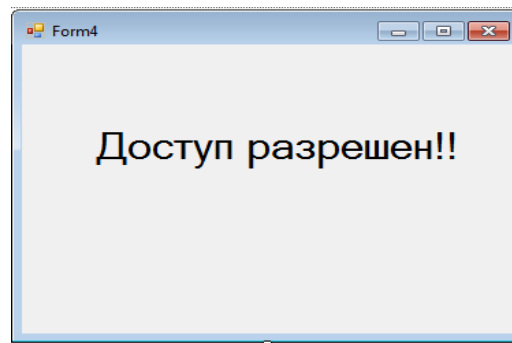
При нажатии на кнопку с именем "Окно 1" открывается "Окно 1" (форма 2) с картинкой. Например



Если пользователь нажимает кнопку "Окно 2", то вызывается форма 4, окно которой не отображается, а из нее вызывается форма 3. Появляется "Окно 3" (форма 3) с запросом пароля.



Если пароль введен правильно, то пользователь получает доступ к “Окну 4” (форма 4).



Окно с запросом пароля автоматически закрывается. Если пользователь закрывает окно ввода пароля, то приложение завершается.

ПРАКТИЧЕСКАЯ РАБОТА № 47

Тема: Разработка игрового приложения

Цель работы: научиться создавать 2D игровые приложения на языке C#, продумывая логику и интерфейс игры.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Создать игру «Подбери пару», в которой игрок должен подобрать пару скрытым значкам.

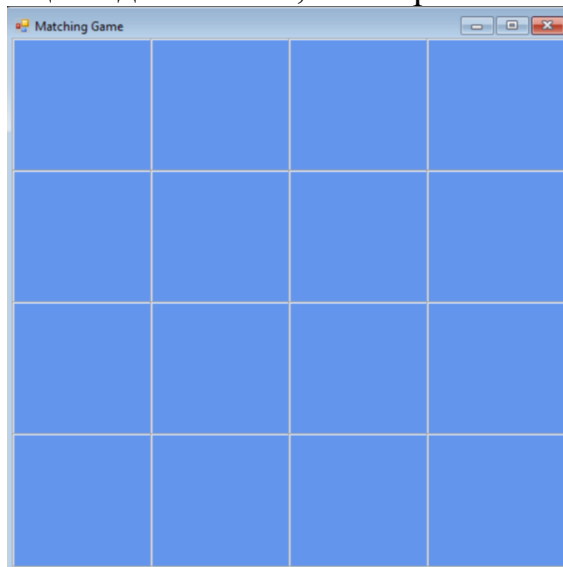
1. Откройте Visual Studio.
2. В окне запуска выберите Создание нового проекта.
3. Выберите шаблон Приложение Windows Forms (.NET Framework) для C# и нажмите Далее
4. В окне Настроить новый проект назовите проект *MatchingGame*, а затем выберите Создать.
5. Создайте сетку 4*4. Щелкните форму, чтобы выбрать конструктор Windows Forms. На этой вкладке для C# считывается файл Form1.cs [Design]. В окне Свойства задайте следующие значения свойств формы.
 - Измените свойство Text с Form1 на Matching Game. Этот текст отображается в верхней части окна игры.
 - Задайте размер формы. Вы можете изменить его либо задав для свойства Size значение 550, 550, либо перетягивая угол формы до тех пор, пока вы не увидите правильный размер в нижней части IDE Visual Studio.
6. Выберите вкладку Панель элементов в левой части интегрированной среды разработки. Если она не отображается, выберите Представление – Панель элементов в строке меню или нажмите сочетание клавиш Ctrl+Alt+X.
7. Перетащите элемент управления TableLayoutPanel из категории Контейнеры на панель элементов или дважды щелкните его. В окне Свойства задайте следующие свойства для панели.
 - Задайте для свойства BackColor значение CornflowerBlue. Чтобы задать это свойство, щелкните стрелку рядом со свойством BackColor. В диалоговом окне BackColor выберите Интернет. Выберите CornflowerBlue в списке названий доступных цветов.
 - Выберите для свойства Dock значение Заливка из раскрывающегося списка, нажав большую кнопку, расположенную посередине. Этот параметр позволяет растянуть таблицу по всей форме.
 - Для свойства CellBorderStyle установите значение Inset. Задание этого значения приведет к тому, что между ячейками поля появятся видимые границы.
 - Нажмите треугольную кнопку в правом верхнем углу элемента управления TableLayoutPanel для отображения меню задач этой панели. В меню задачи щелкните команду Добавить строку дважды, чтобы добавить еще две строки. Затем дважды щелкните пункт Добавить столбец, чтобы добавить еще два столбца.

- В меню задач выберите команду Правка строк и столбцов, чтобы открыть окно Стили столбцов и строк. Для каждого столбца выберите параметр Процент, а затем задайте для каждого столбца ширину 25 процентов.

- Затем в списке в верхней части окна выберите пункт Строки и задайте высоту каждой строки равной 25 процентам.

- Задав все параметры, нажмите кнопку ОК, чтобы сохранить изменения.

Элемент управления `TableLayoutPanel` теперь представляет собой сетку "четыре на четыре" с 16 квадратными ячейками одинакового размера. Эти строки и столбцы задают места, в которых позже появятся значки.



8. Создайте и отформатируйте метки, которые будут отображаться во время игры. Убедитесь, что `TableLayoutPanel` выбран в редакторе формы. Вы должны увидеть элемент управления `tableLayoutPanel1` в верхней части окна Свойства. Если он не выбран, выберите элемент управления `TableLayoutPanel` в форме или из списка в верхней части окна Свойства.

9. Откройте панель элементов, как и прежде, а затем — категорию Стандартные элементы управления. Добавьте элемент управления `Label` в верхнюю левую ячейку `TableLayoutPanel`. Теперь элемент управления `label` выбран в интегрированной среде разработки. Задайте для него следующие свойства:

- Задайте для свойства `BackColor` метки значение `CornflowerBlue`.
- Задайте свойству `AutoSize` значение `False`.
- Задайте для свойства `Dock` значение `Fill`.
- Задайте для свойства `TextAlign` значение `MiddleCenter`, нажав кнопку раскрывающегося списка рядом со свойством, а затем щелкнув среднюю кнопку.

Это значение необходимо, чтобы значок отображался в середине ячейки.

- Выберите свойство `Font`. Появится кнопка с многоточием (...). Нажмите многоточие и задайте для параметра `Font` значение `Webdings`, для параметра `Font Style` — значение `Bold`, а для параметра `Size` — значение 48.

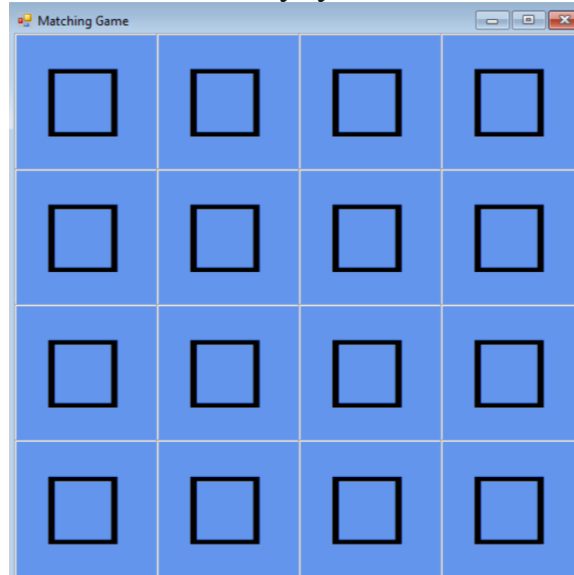
- Установите свойство `Text` равным букве с.

Теперь в левой верхней ячейке `TableLayoutPanel` располагается черный квадрат на синем фоне, который выравнивается по центру.

10. Выберите элемент управления `Label` и скопируйте его в следующую ячейку `TableLayoutPanel`. (Нажмите сочетание клавиш `Ctrl+C` или в строке меню

выберите Правка>Копировать.) Затем вставьте его с помощью комбинации клавиш CTRL+V или выберите Правка – Вставить.

11. Во второй ячейке элемента управления `TableLayoutPanel` появится копия первого элемента управления `Label`. Вставьте его снова, и в третьей ячейке появится еще один элемент управления `Label`. Продолжайте вставлять элементы управления `Label`, пока все ячейки не будут заполнены.



12. Создайте набор совпадающих символов для игры. Каждый символ добавляется к двум случайным ячейкам в таблице `TableLayoutPanel` в форме. Инструкции `new` используются для создания двух объектов. Первым является объект `Random`, который случайным образом выбирает ячейки в `TableLayoutPanel`. Второй объект является объектом `List<T>`. Он хранит случайные выбранные символы.

13. Выберите `Form1.cs`. Затем выберите Просмотреть код>. В качестве альтернативы выберите клавишу F7 или дважды щелкните `Form1`. Интегрированная среда разработки Visual Studio отображает модуль кода для `Form1`.

В существующем коде добавьте следующий код:

```
public partial class Form1 : Form {  
    // Use this Random object to choose random icons for the squares  
    Random random = new Random();  
    // Each of these letters is an interesting icon  
    // in the Webdings font,  
    // and each icon appears twice in this list  
    List<string> icons = new List<string>() {  
        "!", "!", "N", "N", ",", ",", "k", "k",  
        "b", "b", "v", "v", "w", "w", "z", "z" };  
}
```

Объекты списка можно использовать для отслеживания различных типов элементов. Список может содержать числа, значения `true/false`, текст или другие объекты. В соответствующей игре объект списка имеет 16 строк, по одному для каждой ячейки на панели `TableLayoutPanel`. Каждая строка — это одна буква,

соответствующая значкам в метках. Эти символы отображаются в шрифте Webdings как автобус, велосипед и другие

14. При каждом запуске программы он назначает значки случайным образом элементам управления Label в форме с помощью метода AssignIconsToSquares(). Этот код использует ключевое слово foreach. Добавьте метод AssignIconsToSquares() для Form1.cs.

```
/// <summary>
/// Assign each icon from the list of icons to a random square
/// </summary>
private void AssignIconsToSquares() {
    // В панели TableLayoutPanel есть 16 меток, а в списке значков - 16 пиктограмм,
    // поэтому значок выбирается случайным образом из списка
    // и добавляется в каждый ярлык
    foreach (Control control in tableLayoutPanel1.Controls) {
        Label iconLabel = control as Label;
        if (iconLabel != null) {
            int randomNumber = random.Next(Icons.Count);
            iconLabel.Text = Icons[randomNumber];
            // iconLabel.ForeColor = iconLabel.BackColor;
            Icons.RemoveAt(randomNumber); } } }
```

Метод AssignIconsToSquares() выполняет итерацию по каждому элементу управления метки в TableLayoutPanel. Он выполняет одни и те же инструкции для каждого из них. Утверждения извлекают случайный значок из списка.

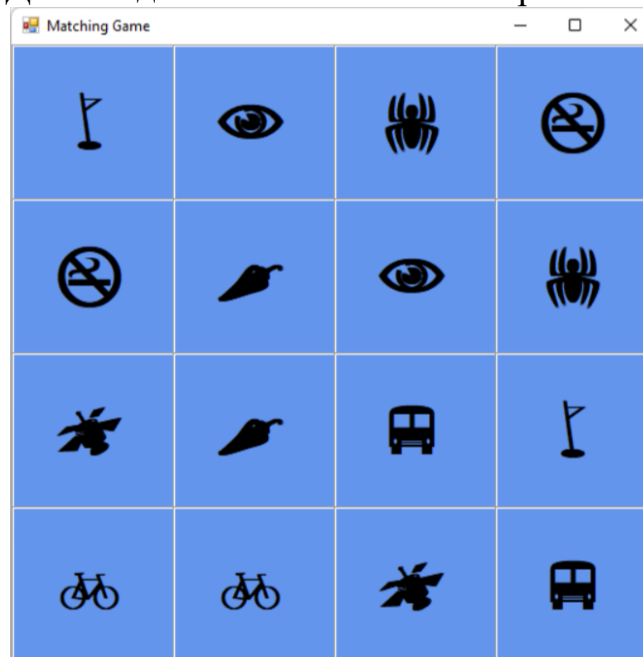
- Первая строка преобразует переменную элемента управления в метку с именем iconLabel.
- Вторая строка — это оператор if, который проверяет, работает ли преобразование. Если преобразование работает, инструкции в инструкции if выполняются.
- Первая строка в инструкции if создает переменную с именем randomNumber, содержащую случайное число, соответствующее одному из элементов в списке значков. Он использует метод Next() объекта Random. Метод Next возвращает случайное число. Эта строка также использует свойство Count значков для определения диапазона, от которого следует выбрать случайное число.
- Следующая строка назначает один из значков элементов списка свойству Text метки.
- Следующая строка скрывает значки. Строка закомментирована здесь, чтобы проверить остальную часть кода перед продолжением.
- Последняя строка в инструкции if удаляет значок, добавленный в форму из списка.

15. Добавьте вызов метода AssignIconsToSquares() в конструкторе Form1 в Form1.cs. Этот метод заполняет игровую доску иконками. Конструкторы вызываются при создании объекта.

```
public Form1() {
    InitializeComponent();
```

```
AssignIconsToSquares(); }
```

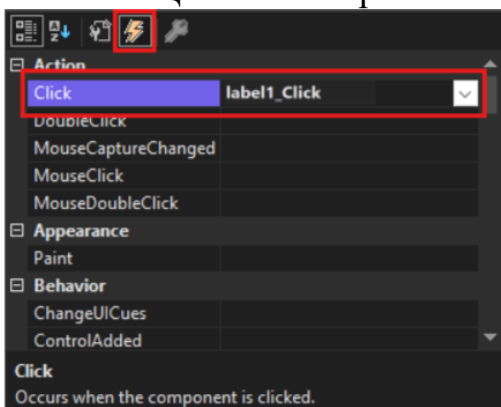
16. Сохраните программу и запустите ее. Он должен отображать форму со случайными значками, назначенными каждой метки. Закройте программу и снова запустите ее. Для каждой метки назначаются разные значки.



Теперь значки видны, так как они не скрыты. Чтобы скрыть их от проигрывателя, можно задать для каждого свойства ForeColor тот же цвет, что и свойство BackColor.

17. Остановите программу. Удалите метки комментариев для закомментированной строки кода внутри цикла в методе AssignIconsToSquares().
`iconLabel.ForeColor = iconLabel.BackColor;`

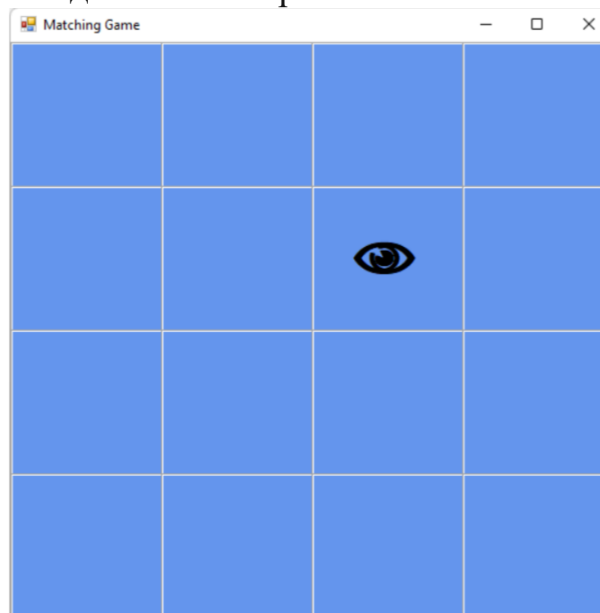
18. Чтобы игра работала, добавьте обработчик события Click, который изменяет цвет выбранной метки в соответствии с фоном. Для этого, откройте форму в конструкторе Windows Forms. Выберите Form1.cs, а затем выберите View - Designer. Выберите первый элемент управления меткой, чтобы выбрать его и дважды щелкните его, чтобы добавить обработчик событий Click с именем label1_Click() в код. Затем удерживайте клавишу CTRL при выборе каждой из других меток. Убедитесь, что выбрана каждая метка. В окне "Свойства" нажмите кнопку "События", которая является молнией. Для события Щелчок выберите в поле label1_Click.



Нажмите клавишу ВВОД. Интегрированная среда разработки автоматически добавляет обработчик событий Click с именем label1_Click() в код в Form1.cs. Так как вы выбрали все метки, обработчик подключается к каждой из меток. Заполните остальную часть кода.

```
/// <summary>
/// Every label's Click event is handled by this event handler
/// </summary>
/// <param name="sender">The label that was clicked</param>
/// <param name="e"></param>
private void label1_Click(object sender, EventArgs e) {
    Label clickedLabel = sender as Label;
    if (clickedLabel != null) {
        //Если метка, на которую был сделан щелчок, черная, это означает, что игрок
        // нажал на значок, который уже был показан проигнорируйте щелчок
        if (clickedLabel.ForeColor == Color.Black)
            return;
        clickedLabel.ForeColor = Color.Black; } }
```

19. Выберите Отладка – Запуск отладки, чтобы выполнить программу. Вы увидите пустую форму с синим фоном. Выберите любую ячейку в форме. Один из значков должен стать видимым. Продолжайте выбирать разные места в форме. При выборе значков они должны отображаться.



20. Добавьте две ссылочных переменных *в код*. Они отслеживают объекты Label или ссылаются на объекты Label. Добавьте ссылки на метки в форму с помощью следующего кода в Form1.cs

```
public partial class Form1 : Form {
    //firstClicked указывает на первый элемент управления меткой,
    // который игрок нажимает, но он будет равен нулю
    // если игрок еще не нажимал на метку
    Label firstClicked = null;
    // secondClicked указывает на второй элемент управления меткой,
```

```
// который игрок нажимает
```

```
Label secondClicked = null;
```

Эти инструкции не приводят к отображению элементов управления Label в форме, так как ключевое слово new отсутствует. При запуске программы firstClicked и secondClicked заданы null. Измените обработчик событий Click в Form1.cs, чтобы использовать новую ссылочную переменную firstClicked. Удалите последнюю инструкцию в методе обработчика событий label1_Click() (clickedLabel.ForeColor = Color.Black;) и замените его инструкцией if следующим образом.

```
/// <summary>
```

```
/// Every label's Click event is handled by this event handler
```

```
/// </summary>
```

```
/// <param name="sender">The label that was clicked</param>
```

```
/// <param name="e"></param>
```

```
private void label1_Click(object sender, EventArgs e) {
```

```
    Label clickedLabel = sender as Label;
```

```
    if (clickedLabel != null) {
```

```
// Если метка, на которую был сделан щелчок, черная, это означает, что игрок
```

```
// нажал на значок, который уже был показан -- проигнорируйте щелчок
```

```
    if (clickedLabel.ForeColor == Color.Black)
```

```
        return;
```

```
// Если значение firstClicked равно null, то это первая иконка в паре,
```

```
// по которой щелкнул игрок, поэтому установите firstClicked на метку,
```

```
// по которой щелкнул игрок, измените ее цвет на черный и верните
```

```
if (firstClicked == null) {
```

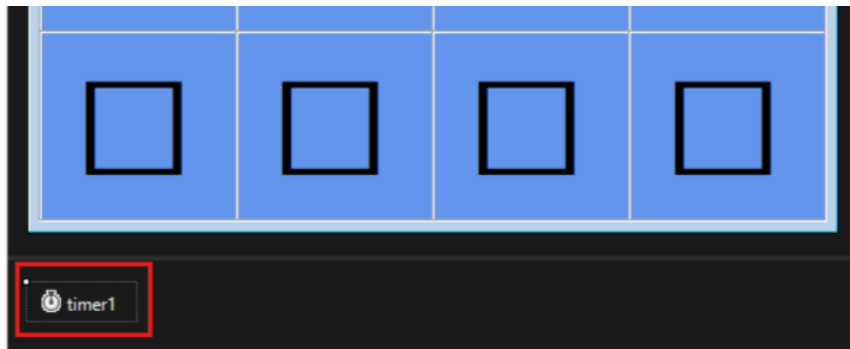
```
    firstClicked = clickedLabel;
```

```
    firstClicked.ForeColor = Color.Black;
```

```
    return; } }
```

21. Сохраните и запустите программу. Выберите один из элементов управления меткой и появится его значок. Выберите следующий элемент управления меткой и заметьте, что ничего не происходит. Появится только первый значок, выбранный. Другие значки невидимы. Программа уже отслеживает первую метку, которую выбрал игрок. Ссылка firstClicked не null в C#. Когда оператор if находит, что firstClicked не равно null, он выполняет инструкции.

22. Добавьте таймер. Таймер ожидает, а затем запускает событие, называемое *галочки*. Таймер может запускать действие или регулярно повторять действие. В программе таймер позволяет игроку выбрать два значка. Если значки не совпадают, он скрывает два значка снова после короткого периода времени. Выберите вкладку панели элементов в категории компонентов, дважды щелкните или перетащите компонент таймера в форму. Значок таймера с именем таймер1 отображается в пространстве под формой.



Щелкните значок Таймер1, чтобы выбрать таймер. В окне "Свойства" нажмите кнопку "Свойства", чтобы просмотреть свойства. Задайте для свойства интервал значение 750, что составляет 750 миллисекунда. Свойство Interval сообщает таймеру, как долго ждать между *тиков*, когда он вызывает событие Tick. Программа вызывает метод Start(), чтобы запустить таймер после того, как проигрыватель выберет вторую метку.

Выберите значок элемента управления таймером и нажмите клавишу ВВОД или дважды щелкните таймер. Интегрированная среда разработки добавляет пустой обработчик событий Tick в Form1.cs. Замените код следующим кодом.

```
/// <summary>
```

```
/// Этот таймер запускается, когда игрок нажимает два не совпадающих значка,
```

```
/// так что это засчитывается на три четверти секунды
```

```
/// а затем отключается и скрывает оба значка
```

```
/// </summary>
```

```
/// <param name="sender"></param>
```

```
/// <param name="e"></param>
```

```
private void timer1_Tick(object sender, EventArgs e) {
```

```
    // Остановить таймер
```

```
    timer1.Stop();
```

```
    // Скрыть оба значка
```

```
    firstClicked.ForeColor = firstClicked.BackColor;
```

```
    secondClicked.ForeColor = secondClicked.BackColor;
```

```
    // Сбросьте первый и второй щелчки чтобы при следующем нажатии на ярлык
```

```
    // программа знала, что это был первый щелчок
```

```
    firstClicked = null;
```

```
    secondClicked = null; }
```

Обработчик событий Tick выполняет три действия:

- Он гарантирует, что таймер не запущен, вызывая метод Stop().
- Он использует две ссылочные переменные, firstClicked и secondClicked, чтобы снова сделать невидимыми значки двух меток, которые выбрал игрок.
- Он перезагружает ссылочные переменные firstClicked и secondClicked на null

23.Перейдите в редактор кода и добавьте код в верхнюю и нижнюю часть метода обработчика событий label1_Click() в Form1.cs. Этот код проверяет, включен ли

таймер, задайте secondClicked ссылочной переменной и запустите таймер. Теперь метод обработчика событий label1_Click() выглядит следующим образом:

```
/// <summary>
/// Every label's Click event is handled by this event handler
/// </summary>
/// <param name="sender">The label that was clicked</param>
/// <param name="e"></param>
private void label1_Click(object sender, EventArgs e) {
    //Таймер включается только после того, как игроку будут показаны две
    //несовпадающие иконки, поэтому не обращайтесь внимания на щелчки, если
    //таймер запущен
    if (timer1.Enabled == true)
        return;
    Label clickedLabel = sender as Label;
    if (clickedLabel != null) {
        //Если метка, на которую был сделан щелчок, черная, это означает, что игрок
        //нажал на значок, который уже был показан -- проигнорируйте щелчок
        if (clickedLabel.ForeColor == Color.Black)
            return;
        // Если игрок зашел так далеко, значит, таймер не запущен, а значение
        //firstClicked не равно нулю, так что это, должно быть, второй значок, на
        //который нажал игрок установите его цвет на черный
        if (firstClicked == null) {
            firstClicked = clickedLabel;
            firstClicked.ForeColor = Color.Black;
            return;
        }
        // Если игрок зашел так далеко, значит, он нажал на две разные иконки,
        //поэтому запустите таймер (который будет ждать три четверти
        //секунды, а затем скроет значки).
        secondClicked = clickedLabel;
        secondClicked.ForeColor = Color.Black;
        // If the player gets this far, the player
        // clicked two different icons, so start the
        // timer (which will wait three quarters of
        // a second, and then hide the icons)
        timer1.Start();
    }
}
```

Код в верхней части метода проверяет, был ли таймер запущен путем проверки значения свойства Enabled. Если игрок выбирает элементы управления первой и второй метки и начинается таймер, то выбор третьей метки ничего не

изменит. Код в нижней части метода присваивает ссылочной переменной `secondClicked` значение для отслеживания второго элемента управления типа `Label`. Затем он задает цвет значка метки черным, чтобы сделать его видимым. Затем он запускает таймер в однократном режиме, чтобы ожидать 750 миллисекунд, после чего произойдет один тик. Обработчик события "Tick" таймера скрывает два значка и сбрасывает ссылочные переменные `firstClicked` и `secondClicked`. Форма готова, чтобы игрок мог выбрать другую пару значков.

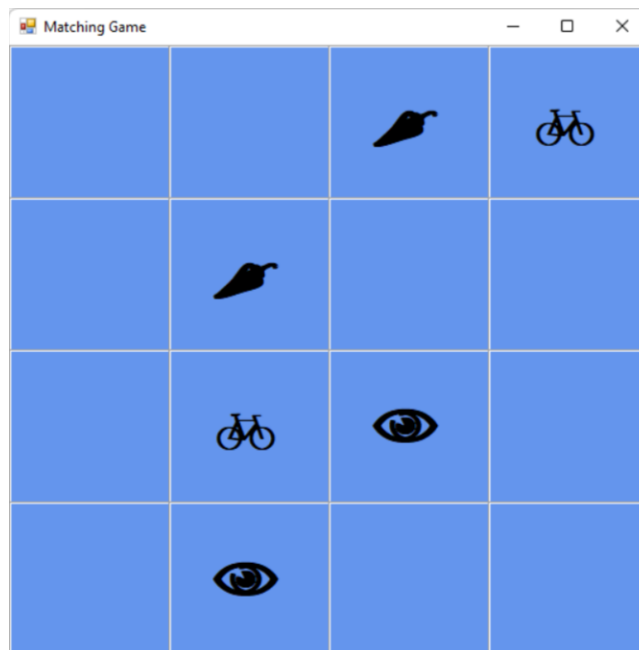
24. Сохраните и запустите программу. Выберите квадрат и значок становится видимым. Выберите еще один квадрат. Значок отображается кратко, а затем оба значка исчезают. Теперь программа отслеживает первый и второй значки, которые вы выбираете. Он использует таймер для приостановки перед исчезновением значков.

25. Когда игрок находит пару, игра должна сброситься, чтобы больше не отслеживать любые метки, использующие ссылочные переменные `firstClicked` и `secondClicked`. Он не должен сбрасывать настройки цветов для двух ярлыков, которые были сопоставлены. Эти метки продолжают отображаться. Добавьте следующую инструкцию `if` в метод обработчика событий `label_Click()`. Поместите его в конец кода чуть выше инструкции, в которой начинается таймер.

```
// Если игрок зашел так далеко, значит, таймер не запущен, а значение
//firstClicked не равно нулю, так что это, должно быть, второй значок, на
//который нажал игрок установите его цвет на черный
    secondClicked = clickedLabel;
    secondClicked.ForeColor = Color.Black;
// Если игрок нажал на две одинаковые иконки, оставьте их черными и
//сбросьте первый и второй клики, чтобы игрок мог нажать на другую иконку
if (firstClicked.Text == secondClicked.Text)    {
    firstClicked = null;
    secondClicked = null;
    return;    }
// Если игрок зашел так далеко, значит, он нажал на две разные иконки,
//поэтому запустите таймер (который будет ждать три четверти
// секунды, а затем скроет значки).
    timer1.Start();    } }
```

Оператор `if` проверяет, совпадает ли значок в первой метке, которую выбирает игрок, с значком во второй метке. Если значки одинаковы, программа выполняет три инструкции. Первые два оператора сбрасывают референсные переменные `firstClicked` и `secondClicked`. Они больше совсем не следят за никакими метками. Третья инструкция — это оператор `return`, который пропускает остальные инструкции в методе без их выполнения.

26. Запустите программу, а затем начните выбирать квадраты в форме.



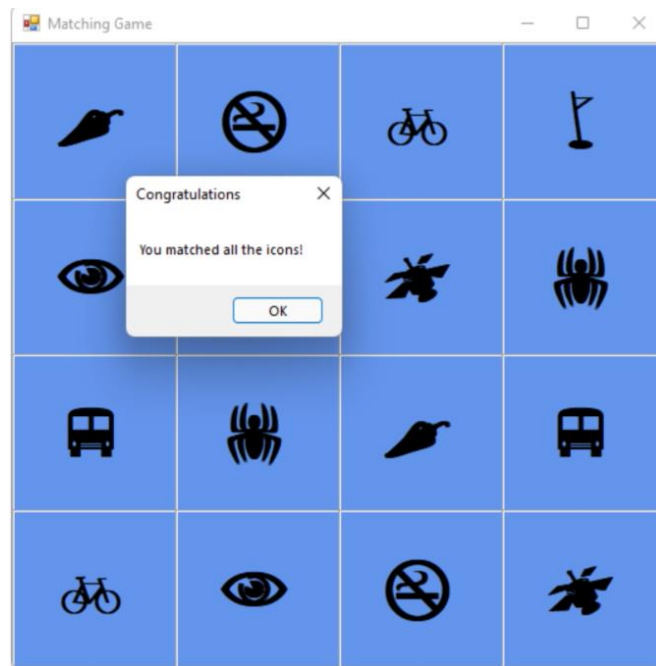
Если вы выбираете пару, которая не соответствует, срабатывает событие Tick таймера. Оба значка исчезают. Если выбрать соответствующую пару, запускается новая инструкция if. Оператор return заставляет метод пропустить код, запускающий таймер. Значки остаются видимыми.

27. Добавьте метод CheckForWinner() в нижней части кода под обработчиком событий timer1_Tick().

```

/// <summary>
/// Check every icon to see if it is matched, by
/// comparing its foreground color to its background color.
/// If all of the icons are matched, the player wins
/// </summary>
private void CheckForWinner() {
// Просмотрите все ярлыки в панели TableLayoutPanel,
// проверяя каждый из них на соответствие его значку
    foreach (Control control in tableLayoutPanel1.Controls) {
        Label iconLabel = control as Label;
        if (iconLabel != null) {
            if (iconLabel.ForeColor == iconLabel.BackColor)
                return;
        }
    }
// Если цикл не вернулся, значит, он не нашел ни одной несоответствующей
// иконки это означает, что пользователь выиграл. Отобразите сообщение и
// закройте форму
    MessageBox.Show("Вы угадали все картинки!", "Поздравляю");
    Close();
}

```



ПРАКТИЧЕСКАЯ РАБОТА № 48

Тема: Разработка игрового приложения

Цель работы: научиться создавать 2D игровые приложения на языке C#, продумывая логику и интерфейс игры.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Разработать 2D-игру «Змейка»

Некоторые важные ограничения в игре, которые предусмотрим:

- если "Змейка" уже движется вправо, то мгновенно развернуть её влево (т.е. в обратном направлении) у нас не получится: "Змейка" в игре *не будет уметь* менять направление резко на 180 градусов.
- никаких ограничений на скорость нажатия игроком клавиш. Это может привести к тому, что для *очень быстрых* игроков, если они нажмут, к примеру, шустро клавиши "стрелка вправо" и "стрелка вверх" при текущем движении "Змейки" вниз, то "Змейка" тут же "съест сама себя", и игра тут же закончится.
- нет функции "пауза", т.е. можно только играть или перезапускать игру.

1. Создадим новый проект в среде разработки. В качестве имени проекта укажем SnakeGameExample, местоположение проекта выбираем на свой вкус. После того, как проект создан, в окне "*Обозреватель решений*" увидите дерево проекта и созданную по умолчанию форму Form1.cs. Переименуем её сразу в FrmSnakeGame.cs и при выдаче диалогового окна с запросом на переименование всех связанных ссылок - соглашаемся.

Сразу выставим некоторые свойства для главной формы:

- FormBorderStyle: FixedSingle
- Text: Allineed.Ru - Пример игры "Змейка" на C#
- Size: 700; 521
- StartPosition: CenterScreen
- MaximizeBox: False

Также нам потребуется всего один элемент управления на форме - с типом Timer. Добавим его из панели элементов, перетащив на главную форму проекта, после чего установим ему следующие значения свойств:

- Name: TimerGameLoop
- Interval: 300

2. Дважды кликнем по главной форме проекта, находясь в режиме визуального конструктора. Это приведёт к генерации пустого обработчика для события *Load* формы, который будет срабатывать при загрузке главной формы игры. Пропишем все необходимые импорты

```
using System;  
using System.Collections.Generic;  
using System.Drawing;  
using System.Linq;  
using System.Windows.Forms;
```

3. Далее мы напишем пока пустой закрытый (*private*) метод с именем StartGame в классе формы и добавим в код обработчика загрузки формы - FrmSnakeGame_Load - следующий код:

```
private void FrmSnakeGame_Load(object sender, EventArgs e) {  
    DoubleBuffered = true;  
    BackColor = Color.Black;  
    StartGame();  
}  
private void StartGame() {  
}
```

Свойство DoubleBuffered устанавливаем в true, чтобы при отрисовке всех элементов игры включить режим двойной буферизации для формы. Это необходимо для исключения неприятных эффектов "мерцания" клеток игрового поля и других рисуемых элементов игры. Также установили свойство BackColor формы равным Color.Black, это значит, что фон игры будет чёрным. При желании можете потом поменять на любой другой цвет из доступных в стандартной структуре *Color*, который вам понравится.

Теперь поднимемся в самое начало класса FrmSnakeGame для главной формы и добавим все необходимые поля, которые нам потребуются в игре:

```
// Размер клетки игрового поля, в пикселях  
private const int CELL_SIZE_PIXELS = 30;  
// Количество рядов в игровом поле  
private const int ROWS_NUMBER = 15;  
// Количество столбцов в игровом поле  
private const int COLS_NUMBER = 15;  
// Отступ в пикселях от левого края формы  
private const int FIELD_LEFT_OFFSET_PIXELS = 40;  
// Отступ в пикселях от правого края формы  
private const int FIELD_TOP_OFFSET_PIXELS = 15;  
// Задержка (свойство "Interval") для основного игрового таймера  
TimerGameLoop  
private const int INITIAL_SNAKE_SPEED_INTERVAL = 300;  
// На сколько миллисекунд увеличить скорость "Змейки" при очередном  
//поглощении змейкой "Еды"  
private const int SPEED_INCREMENT_BY = 5;  
private enum SnakeDirection { Left, Right, Up, Down }  
// Текущее направление движения "Змейки"  
private SnakeDirection snakeDirection = SnakeDirection.Up;  
// Список точек, содержащих координаты всего "тела Змейки"  
private LinkedList<Point> snake = new LinkedList<Point>();  
// Точка, содержащая координаты "Еды" для "Змейки"  
private Point food;
```

```
// Генератор псевдослучайных чисел. нужен для генерации очередной "Еды" в
//произвольном месте игрового поля
private Random rand = new Random();
private bool isGameEnded; // Признак: игра завершена?
    Внутренний закрытый enum-тип SnakeDirection будет определять
возможные направления движения "Змейки":
```

- Left - "Змейка" движется влево
- Right - "Змейка" движется вправо
- Up - "Змейка" движется вверх
- Down - "Змейка" движется вниз

4. Полный вид класса главной формы FrmSnakeGame:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
namespace SnakeGameExample {
    public partial class FrmSnakeGame : Form {
        private const int CELL_SIZE_PIXELS = 30;
        private const int ROWS_NUMBER = 15;
        private const int COLS_NUMBER = 15;
        private const int FIELD_LEFT_OFFSET_PIXELS = 40;
        private const int FIELD_TOP_OFFSET_PIXELS = 15;
        private const int INITIAL_SNAKE_SPEED_INTERVAL = 300;
        private const int SPEED_INCREMENT_BY = 5;
        private enum SnakeDirection { Left, Right, Up, Down }
        private SnakeDirection snakeDirection = SnakeDirection.Up;
        private LinkedList<Point> snake = new LinkedList<Point>();
        private Point food;
        private Random rand = new Random();
        private bool isGameEnded;
        public FrmSnakeGame() {
            InitializeComponent();
        }
        private void FrmSnakeGame_Load(object sender, EventArgs e) {
            DoubleBuffered = true;
            BackColor = Color.Black;
            StartGame();
        }
        private void StartGame() {
        }
    }
}
```

5. При создании формы для нас был автоматически добавлен метод конструктора формы с именем `FrmSnakeGame()`. Добавим сразу после этого метода конструктора следующий новый метод с именем `InitializeSnake()`:

```
private void InitializeSnake() {  
    snakeDirection = SnakeDirection.Up;  
    snake.Clear();  
    snake.AddFirst(new Point(ROWS_NUMBER - 1, COLS_NUMBER / 2 - 1));    }
```

Он отвечает за первичную инициализацию "Змейки":

- устанавливает ей начальное направление движения - "вверх": `snakeDirection = SnakeDirection.Up`;
- очищает связанный список точек с координатами всех клеток "Змейки" (очистка изначально пустого списка здесь нужна, т.к. метод будет вызываться не только при старте игры, но и при её перезапуске, когда нужно будет вновь создать только "голову" змейки, без "тела"): `snake.Clear()`;
- добавляет в связанный список `snake` первую клетку "Змейки". Это голова "Змейки", и её координаты - это *самый низ игрового поля*, т.е. самый нижний ряд (его индекс равен `ROWS_NUMBER - 1`), а расположение "головы" по горизонтали - примерно посередине игрового поля (индекс столбца равен `COLS_NUMBER / 2 - 1`).

Сделаем так, чтобы инициализация "Змейки" происходила сразу при старте игры, т.е. при загрузке главной формы. Для этого добавляем в метод `StartGame()` вызов нового метода `InitializeSnake()`:

```
private void StartGame() {  
    InitializeSnake();    }
```

6. Теперь давайте создадим в коде главной формы метод по генерации "Еды" для "Змейки" - это, по сути, будет какая-то произвольная клетка игрового поля, которая не может пересекаться с клетками поля, где расположена сама "Змейка". Назовём этот метод **`GenerateFood()`**:

```
private void GenerateFood() {  
    bool isFoodClashWithSnake;  
    do {  
        food = new Point(rand.Next(0, ROWS_NUMBER), rand.Next(0,  
COLS_NUMBER));  
        isFoodClashWithSnake = false;  
        foreach (Point p in snake) {  
            if (p.X == food.X && p.Y == food.Y) {  
                isFoodClashWithSnake = true;  
                break;            }        }  
    } while (isFoodClashWithSnake);  
    TimerGameLoop.Interval -= SPEED_INCREMENT_BY;    }
```

Алгоритм метода по генерации "Еды" довольно простой:

- В начале метода объявляем булеву переменную `isFoodClashWithSnake`. Когда в ней будет значение `true`, это будет означать, что очередная произвольно

сгенерированная клетка поля пересеклась с телом "Змейки". Соответственно, значение `false` скажет о том, что пересечения "Еды" со "Змейкой" нет, и мы можем поместить "Еду" на игровое поле в нужных, произвольно сгенерированных, координатах.

- Далее идет цикл *do-while*. Условием выхода из этого цикла является признак того, что "Еду" можно разместить на поле, и нет пересечения со "Змейкой".

1. Внутри цикла получаем очередной экземпляр структуры *Point* с произвольными координатами в пределах игрового поля. Рандомизацию ряда и столбца клетки поля для "Еды" осуществляем вызовами `rand.Next(0, ROWS_NUMBER)` и `rand.Next(0, COLS_NUMBER)`, которые всегда сгенерируют допустимое значение ряда и столбца для клетки поля.

2. Далее мы поворачиваем флаг `isFoodClashWithSnake` в значение `false`, тем самым утверждая, что пересечения сгенерированной клетки "Еды" со "Змейкой" пока нет.

3. Внутренний цикл *foreach* нужен для перебора всех клеток поля, где расположена "Змейка", и внутри этого цикла мы проверяем: совпадают ли координаты текущей клетки "Змейки" с координатами только что сгенерированной клетки "Еды"? Если совпадают, мы взводим флажок `isFoodClashWithSnake` в значение `true` (т.е. пересечение "Еды" и "Змейки" возникло) и при помощи оператора *break* тут же выходим из цикла *foreach*.

Таким образом, перед циклом *foreach* флажок `isFoodClashWithSnake` устанавливается так, чтоб выйти из внешнего цикла *do-while*, если нет пересечений. Если пересечения возникли, флаг будет взведён, и мы будем продолжать "крутить" цикл *do-while* до тех пор, пока не будет пересечения.

- Перед выходом из метода `GenerateFood()` ускоряем игру: поскольку в методе генерируем новую "Еду", это значит, что "Змейка" перед этим "скушала" предыдущую "Еду". А значит, можно ускорить "Змейку", делая процесс игры более увлекательным и одновременно более сложным при увеличении длины тела "Змейки".

7. Добавим теперь вызов этого нового метода в метод `StartGame()`, а также сбросим признак окончания игры `isGameEnded` в значение `false`, запустим игровой таймер - `TimerGameLoop.Start()` - и выставим значение интервала для таймера в исходное - в значение константы `INITIAL_SNAKE_SPEED_INTERVAL`:

```
private void StartGame() {  
    GenerateFood();  
    InitializeSnake();  
    isGameEnded = false;  
    TimerGameLoop.Start();  
    TimerGameLoop.Interval = INITIAL_SNAKE_SPEED_INTERVAL; }  
}
```

8. Теперь перейдем к рисованию игрового поля, отрисовке самой "Змейки" и отрисовке "Еды". Вернёмся к представлению визуального конструктора главной формы игры и в окне "Свойства" перейдем к событиям для формы (иконка "молнии"). Сгенерируем обработчик для события *Paint* - двойным кликом

напротив имени события. В результате снова откроется редактор кода главной формы, и мы увидим следующий сгенерированный пустой метод-обработчик:

```
private void FrmSnakeGame_Paint(object sender, PaintEventArgs e) {  
}
```

Первым делом получим в новую локальную переменную *g* экземпляр класса *Graphics* из аргументов этого события (параметр *e* с типом *PaintEventArgs*):

```
private void FrmSnakeGame_Paint(object sender, PaintEventArgs e) {  
    Graphics g = e.Graphics;  
}
```

Этот экземпляр класса *Graphics* будем использовать для отрисовки всех игровых объектов ("Змейка" и "Еда"), а также игрового поля и передавать в качестве аргумента для вспомогательных методов отрисовки игровых объектов. Создадим три метода - *DrawGrid*, *DrawSnake*, *DrawFood*, - каждый из которых на вход принимает параметр с типом *Graphics*:

```
private void DrawGrid(Graphics g) {  
    for (int row = 0; row <= ROWS_NUMBER; row++) {  
        g.DrawLine(Pens.Cyan,  
            new Point(FIELD_LEFT_OFFSET_PIXELS,  
FIELD_TOP_OFFSET_PIXELS + row * CELL_SIZE_PIXELS),  
            new Point(FIELD_LEFT_OFFSET_PIXELS + CELL_SIZE_PIXELS *  
ROWS_NUMBER, FIELD_TOP_OFFSET_PIXELS + row * CELL_SIZE_PIXELS)  
);  
        for (int col = 0; col <= COLS_NUMBER; col++) {  
            g.DrawLine(Pens.Cyan,  
                new Point(FIELD_LEFT_OFFSET_PIXELS + col *  
CELL_SIZE_PIXELS, FIELD_TOP_OFFSET_PIXELS),  
                new Point(FIELD_LEFT_OFFSET_PIXELS + col *  
CELL_SIZE_PIXELS, FIELD_TOP_OFFSET_PIXELS + CELL_SIZE_PIXELS *  
COLS_NUMBER)  
);  
        }  
    }  
}  
private void DrawSnake(Graphics g) {  
    foreach (Point p in snake) {  
        g.FillRectangle(Brushes.Lime, new Rectangle(  
            FIELD_LEFT_OFFSET_PIXELS + p.Y * CELL_SIZE_PIXELS + 1,  
            FIELD_TOP_OFFSET_PIXELS + p.X * CELL_SIZE_PIXELS + 1,  
            CELL_SIZE_PIXELS - 1,  
            CELL_SIZE_PIXELS - 1));  
    }  
}  
private void DrawFood(Graphics g) {  
    g.FillRectangle(Brushes.Red, new Rectangle(  
        FIELD_LEFT_OFFSET_PIXELS + food.Y * CELL_SIZE_PIXELS + 1,  
        FIELD_TOP_OFFSET_PIXELS + food.X * CELL_SIZE_PIXELS + 1,  
        CELL_SIZE_PIXELS - 1,  
        CELL_SIZE_PIXELS - 1));  
}
```

По названиям методов нетрудно догадаться, за что они отвечают:

- *DrawGrid(Graphics g)* - метод рисует всё игровое поле в виде клеток. Это делается двумя циклами *for*, сначала по рядам (внешний цикл), затем - по

столбцам (внутренний цикл). Пробегая по рядам, мы просто рисуем горизонтальные линии, зная отступ от левого края формы (константа класса `FIELD_LEFT_OFFSET_PIXELS`), а также зная размер каждой клетки игрового поля в пикселях (константа класса `CELL_SIZE_PIXELS`). Аналогичным образом во внутреннем цикле *for* рисуются вертикальные линии "сетки" игрового поля. В итоге мы получаем поле из клеток размером `ROWS_NUMBER` x `COLS_NUMBER`.

- `DrawSnake(Graphics g)` - метод рисует саму "Змейку". Алгоритм рисования очень прост: всего один цикл *foreach* по всем точкам (экземпляры структуры *Point*) "Змейки". Внутри этого цикла рисуем заполненный прямоугольник (при помощи метода `FillRectangle`, доступного в классе `Graphics`), который на самом деле является квадратом с длиной стороны, равной (`CELL_SIZE_PIXELS - 1`) пикселей. В качестве начальной координаты квадрата по оси X указываем выражение `FIELD_LEFT_OFFSET_PIXELS + p.Y * CELL_SIZE_PIXELS + 1`. Оно означает, что берём отступ игрового поля слева от главной формы, к нему прибавляем произведение индекса столбца на размерность клетки игрового поля: `p.Y * CELL_SIZE_PIXELS`. Именно столбца (он у нас в `p.Y`), поскольку нужен расчёт координаты X, который хранится как раз в точке "текущего звена Змейки". Затем прибавляем ещё единицу, чтобы клетка "Змейки" не "налезала" на границы клетки, а отрисовка была *внутри границ* клетки (по этой же причине от длины стороны квадрата отнимали единицу: `CELL_SIZE_PIXELS - 1`). Тот же принцип используется для вычисления начальной координаты квадрата по оси Y, только там индекс ряда берём из `p.X`. Наконец, в качестве кисти для заполнения клетки поля используем кисть с цветом лайма (`Brushes.Lime`) - чтобы "Змейка" в игре была зелёной. Если захотите перекрасить "Змейку" в какой-то другой цвет, то здесь самое место для того, чтобы выбрать любую другую кисть, которая по душе.

- `DrawFood(Graphics g)` - метод рисует "Еду" для "Змейки", которая представляется одной клеткой поля, закрашенной красным цветом (кисть `Brushes.Red`). Метод очень похож по структуре на внутреннее наполнение цикла *foreach*, который только что рассмотрели в методе `DrawSnake`, здесь разница лишь в том, что никакого цикла нам не нужно - координаты клетки с "Едой" у нас уже хранятся в переменной `food` класса главной формы.

9.Теперь всего лишь нужно вызвать все эти методы из обработчика события *Paint*, передавая в аргументах каждого вызова подготовленную переменную `g` класса `Graphics`:

```
private void FrmSnakeGame_Paint(object sender, PaintEventArgs e) {  
    Graphics g = e.Graphics;  
    DrawGrid(g);  
    DrawFood(g);  
    DrawSnake(g);  
}
```

Таким образом делегировали отрисовку разных игровых объектов соответствующим отдельным методам. И когда теперь главная форма будет вызывать событие *Paint* для своей перерисовки, у нас будут рисоваться сразу все игровые объекты: сетка игрового поля, "Змейка" и "Еда".

10. Напишем отдельный небольшой метод с именем `GameOver()`. Несложно догадаться, что он отвечает за отработку действий, когда произошёл конец игры. Конец игры может случиться в двух ситуациях: "Змейка" попыталась съесть саму себя или её голова дошла до одной из границ игрового поля, а игрок не успел развернуть "Змейку" в одном из направлений движения, не приводящих к поражению. Метод будет взводить флаг `isGameEnded`, отвечающий за признак завершения игры, останавливать таймер `TimerGameLoop` для игрового цикла, а также выводить диалоговое окно для игрока с вопросом о желании начать игру заново:

```
private void GameOver() {
    isGameEnded = true;
    TimerGameLoop.Stop();
    if (MessageBox.Show("Конец игры! Начать заново?", "Конец игры",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes) {
        StartGame();
    }
}
```

Если игрок нажмёт "Да" в диалоговом окне, то игра будет перезапущена - за счёт вызова метода `StartGame()`.

11. Теперь давайте в классе формы напишем метод, который будет отвечать за передвижение "Змейки". Этот метод так и назовём - `MoveSnake()`, а ниже представлен его код:

```
private void MoveSnake() {
    LinkedListNode<Point> head = snake.First;
    Point newHead = new Point(0, 0);
    switch (snakeDirection) {
        case SnakeDirection.Left:
            newHead = new Point(head.Value.X, head.Value.Y - 1); break;
        case SnakeDirection.Right:
            newHead = new Point(head.Value.X, head.Value.Y + 1); break;
        case SnakeDirection.Down:
            newHead = new Point(head.Value.X + 1, head.Value.Y); break;
        case SnakeDirection.Up:
            newHead = new Point(head.Value.X - 1, head.Value.Y); break;
    }
    if (snake.Any(point => point.X == newHead.X && point.Y == newHead.Y))
    {
        // "Змейка" съела саму себя! Конец игры!
        Invalidate();
        GameOver();
        return;
    }
    snake.AddFirst(newHead);
    if (newHead.X == food.X && newHead.Y == food.Y) {
        GenerateFood();
    }
    else {
        snake.RemoveLast();
    }
}
```

Разберём алгоритм этого метода:

- Вначале получаем в переменную `head` самую первую точку из списка с именем `snake`, который хранит все точки с координатами игрового поля, где расположено тело "Змейки". Самая первая точка - это голова "Змейки", поэтому и переменная называется `head`.

- Далее, заготавливаем в переменной `newHead` новую точку, по умолчанию инициализируя её координатами (0, 0). Это та точка, в которой будет следующая ближайшая позиция головы "Змейки" после осуществления ей движения в текущем направлении.

- В операторе `switch` проверяем текущее направление движения "Змейки" и устанавливаем реальные координаты следующей позиции головы "Змейки". Если "Змейка" движется влево, то от координаты `Y` текущей головы "Змейки" отнимаем единицу, тем самым получая столбец игрового поля левее текущего столбца, где расположена голова "Змейки". Все остальные развилки оператора `switch` выстроены по этому же принципу - разница лишь в изменяемой координате (`X` - строки игрового поля, `Y` - столбцы) для текущего направления движения.

- После оператора `switch` идёт оператор `if`, в котором проверяем: *"а не съела ли только что Змейка саму себя?"*. Для этого пытаемся найти среди всех точек (*Point*) тела "Змейки" ту, которая полностью по координатам совпадает с координатами новой головы "Змейки" (`newHead`). Если нашлась такая точка, у которой `X` и `Y` полностью совпадает с новой позицией головы "Змейки", то это значит, что игра окончена, т.к. произошло "самосъедание Змейки". В этом случае мы вызываем перерисовку всей формы при помощи вызова метода **`Invalidate()`**, который, в свою очередь, вызовет событие *Paint* и перерисует всю видимую область главной формы. Далее вызываем уже рассмотренный выше метод `GameOver()`, после чего выходим из метода `MoveSnake()` - за счёт оператора `return`.

- Если "Змейка" не съела саму себя, то в оператор `if` не входим, и вызывается `snake.AddFirst(newHead)`. Это добавляет "новую голову" для "Змейки" - в самое начало списка `snake`.

- В самом конце - также оператор `if`, где идёт проверка: *"Не произошло ли поедание Змейкой текущей еды?"*. Если произошло, пора сгенерировать новую "Еду", и вызываем метод `GenerateFood()`, при этом ничего не делаем со списком `snake`, т.е. не удаляем из него "хвост". За счёт этого "Змейка" увеличивается в размерах на одну клетку. Если же *не произошло* поедание "Еды", то попадём в ветку `else`, где мы удаляем последний элемент списка `snake`: это "хвост" нашей "Змейки", который за счёт удаления из списка очистит клетку игрового поля, из которой он "уполз".

12. Напишем метод `IsGameOver()`, которые вернет `true`, если произошёл конец игры, и "Змейка" врезалась в края игрового поля, не успев развернуться, и `false` - в противном случае (т.е. игра продолжается):

```
private bool IsGameOver() {  
    LinkedListNode<Point> head = snake.First;  
    switch (snakeDirection) {  
        case SnakeDirection.Left:
```

```

        return head.Value.Y - 1 < 0;
    case SnakeDirection.Right:
        return head.Value.Y + 1 >= COLS_NUMBER;
    case SnakeDirection.Down:
        return head.Value.X + 1 >= ROWS_NUMBER;
    case SnakeDirection.Up:
        return head.Value.X - 1 < 0;    }
    return false;    }

```

Алгоритм этого метода очень прост: из списка snake получаем голову "Змейки", т.е. первый элемент списка. Далее смотрим на текущее направление движения "Змейки" и делаем расчёт: *"не выйдет ли голова Змейки ближайшим следующим ходом за пределы игрового поля?"*. Для направлений движения Left и Right смотрим, чтобы координата Y (отвечает за столбцы) у головы "Змейки" не выходила за пределы размеров игрового поля. Если это случилось, метод возвращает true. Аналогично для направлений движения Up и Down: если голова "Змейки" попытается следующим ходом выйти за пределы первого или последнего ряда, то также возвращаем true из метода. Если же все проверки прошли, и мы не вышли из метода, то в самом конце возвращаем false, что означает: *"игра продолжается, Змейка продолжает движение"*.

13. Теперь давайте вернёмся к представлению конструктора главной формы, найдем элемент управления TimerGameLoop, привязанный к форме на первых шагах, и дважды кликнем по нему. В результате у нас сгенерируется обработчик события Tick для нашего таймера, и его заполним следующим образом:

```

private void TimerGameLoop_Tick(object sender, EventArgs e) {
    if (IsGameOver()) {
        GameOver();    }
    else {
        MoveSnake();
        Invalidate();    }    }

```

Т.е. таймер срабатывает исходно (при старте игры) с периодичностью 300 миллисекунд. И с этой периодичностью сперва проверяем - не произошёл ли конец игры только что? (вызов метода IsGameOver() в операторе if). Если случился конец игры, вызываем метод GameOver(). В противном же случае игра продолжается, и вызываем всего два метода: MoveSnake() и Invalidate(). Первый, как помним, передвигает "Змейку" в направлении её движения, а второй - вызывает перерисовку всей области главной формы.

14. Реализуем управление в игре поскольку пока не умеем "заставлять" нашу "Змейку" поворачивать в разные стороны. Напишем следующий короткий метод ChangeSnakeDirection, который будет отвечать за смену направления движения "Змейки":

```

private void ChangeSnakeDirection(SnakeDirection restrictedDirection,
SnakeDirection newDirection) {
    if (snakeDirection != restrictedDirection) {
        snakeDirection = newDirection;    }    }

```

В первом параметре `restrictedDirection` метода будем передавать некоторое значение направления движения "Змейки", которое будет *запрещённым* для дальнейшей смены направления "Змейки", т.е. это значение, при котором не хотим, чтобы "Змейка" меняла направление движения.

Вторым же параметром `newDirection` будем передавать желаемое новое направление движения "Змейки". Это желаемое направление чуть ниже будем регулировать нажатиями определённых клавиш на клавиатуре.

Таким образом, алгоритм этого нового метода таков: если текущее направление движения "Змейки" не равно некоторому запрещённому (`restrictedDirection`), то поменять направление "Змейки" на новое, заданное параметром `newDirection`.

15. Давайте реализуем обработку нажатия клавиш, чтобы управлять "Змейкой" в игре. Для этого вновь вернёмся на главную форму (режим конструктора) и перейдем к событиям формы. Двойным кликом напротив события *KeyDown* генерируем для него пустой метод-обработчик в коде формы. И заполняем этот пустой метод следующим образом:

```
private void FrmSnakeGame_KeyDown(object sender, KeyEventArgs e) {  
    switch (e.KeyCode) {  
        case Keys.Left:  
        case Keys.A:  
            ChangeSnakeDirection(SnakeDirection.Right, SnakeDirection.Left);  
            break;  
        case Keys.Right:  
        case Keys.D:  
            ChangeSnakeDirection(SnakeDirection.Left, SnakeDirection.Right);  
            break;  
        case Keys.Down:  
        case Keys.S:  
            ChangeSnakeDirection(SnakeDirection.Up, SnakeDirection.Down);  
            break;  
        case Keys.Up:  
        case Keys.W:  
            ChangeSnakeDirection(SnakeDirection.Down, SnakeDirection.Up);  
            break;  
        case Keys.Escape:  
            TimerGameLoop.Stop();  
            Close();  
            break;  
        case Keys.Space:  
            if (isGameEnded && !TimerGameLoop.Enabled) {  
                StartGame();  
            }  
            break;  
    }  
}
```

Можно видеть, что логика этого обработчика заключена в единственном операторе *switch*, где проверяем код текущей нажатой клавиши: Для клавиш "стрелка влево" или "А" вызываем смену направления движения

"Змейки" на "движение влево" (SnakeDirection.Left). При этом *запрещённым* значением для смены направления движения будет являться признак, что "Змейка" *уже движется вправо* (это регулирует первый аргумент SnakeDirection.Right при вызове метода ChangeSnakeDirection). Это сделано для запрета поворота "Змейки" на 180°, т.е. влево, ведь если её голова *уже* движется вправо, повернуть резко налево нельзя. По такому же принципу происходит обработка нажатия клавиш для остальных направлений движения:

- Для клавиш "стрелка вправо" или "D" вызываем смену направления движения "Змейки" на "движение вправо" (SnakeDirection.Right)
- Для клавиш "стрелка вниз" или "S" вызываем смену направления движения "Змейки" на "движение вниз" (SnakeDirection.Down)
- Для клавиш "стрелка вверх" или "W" вызываем смену направления движения "Змейки" на "движение вверх" (SnakeDirection.Up)

В случае нажатия клавиши Escape, останавливаем таймер игрового цикла - TimerGameLoop.Stop() - и просто закрываем главную форму приложения при помощи вызова Close(), что приводит к завершению игры. Здесь нет никаких диалоговых окон *"Вы уверены, что хотите выйти из игры?"*.

16.Последнее - обработка нажатия на клавишу пробела (Keys.Space): здесь проверим, что если игра закончена, и игровой таймер выключен, то пробел запустит игру заново, вызвав метод StartGame().

17.Запустите игру.

ПРАКТИЧЕСКАЯ РАБОТА № 49

Тема: Разработка игрового приложения

Цель работы: научиться создавать 2D игровые приложения на языке C#, продумывая логику и интерфейс игры.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Разработайте любую игру на выбор:

1. «Угадай число»
2. «Угадай слово»
3. «Крестики-нолики»

ПРАКТИЧЕСКАЯ РАБОТА № 50

Тема: Разработка многооконного приложения. Компиляция и запуск приложения

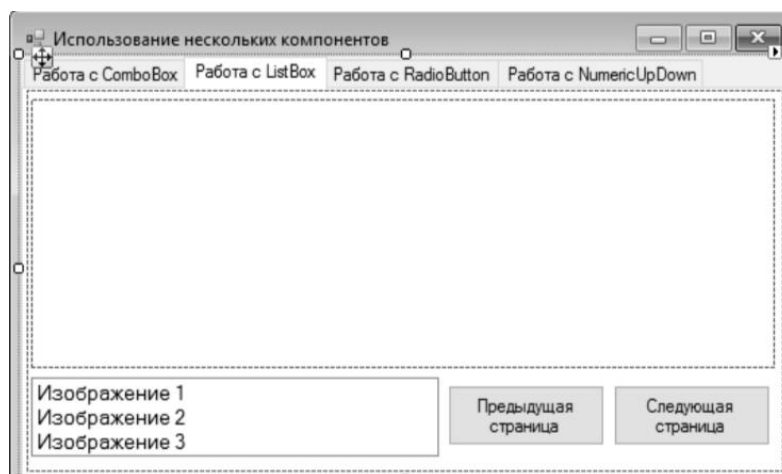
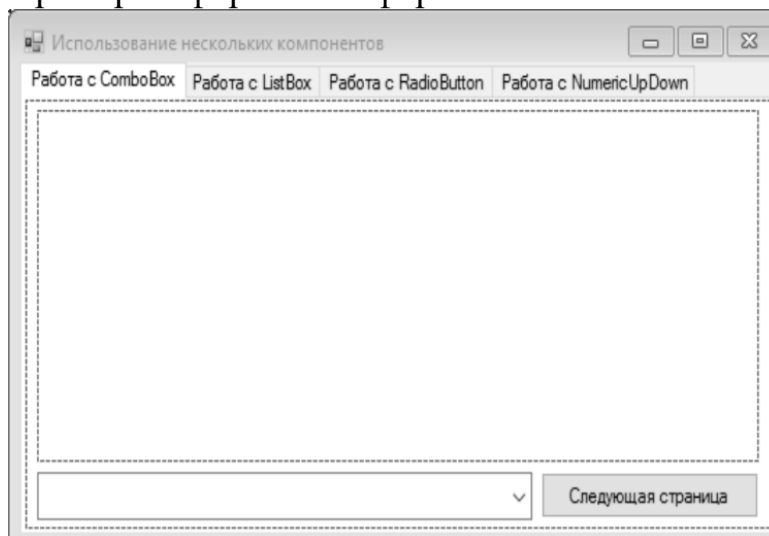
Цель работы: изучить основные способы разработки многооконных приложений, получить практические навыки в создании многооконных приложений.

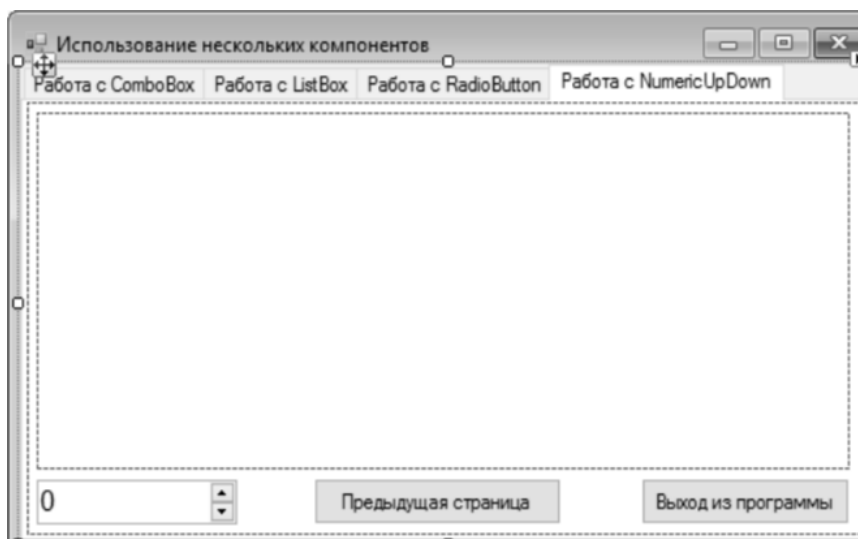
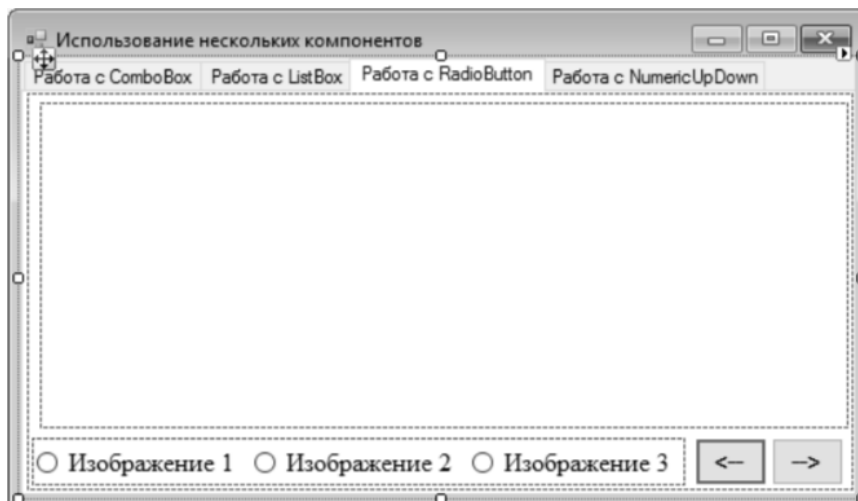
Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. При помощи компонентов ComboBox, ListBox, RadioButton, NumericUpDown выводить изображения через pictureBox. Компоненты ComboBox, ListBox, RadioButton, NumericUpDown будут разделены через компонент TabControl, содержащий несколько вкладок. На каждой вкладке будет свой компонент pictureBox и один из компонентов управления. Для удобства перехода между вкладками будут созданы кнопки перехода на следующую и/или предыдущую вкладку, а на последней вкладке будет кнопка закрытия программы.

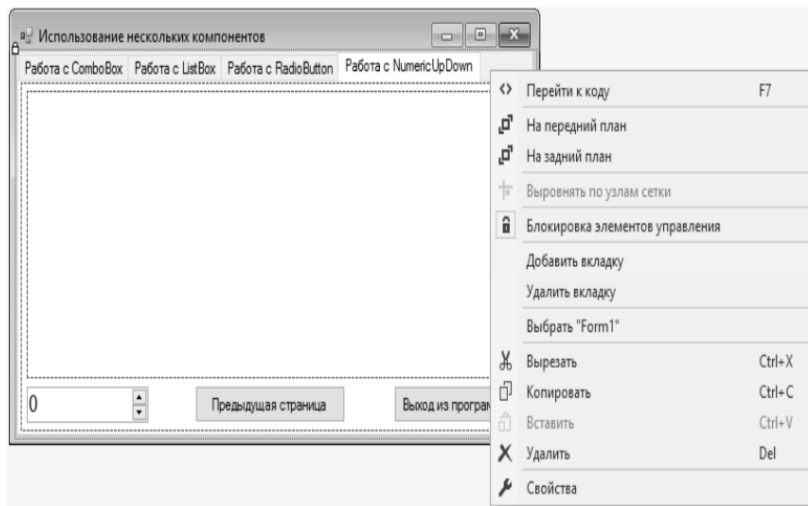
Примеры оформления формы



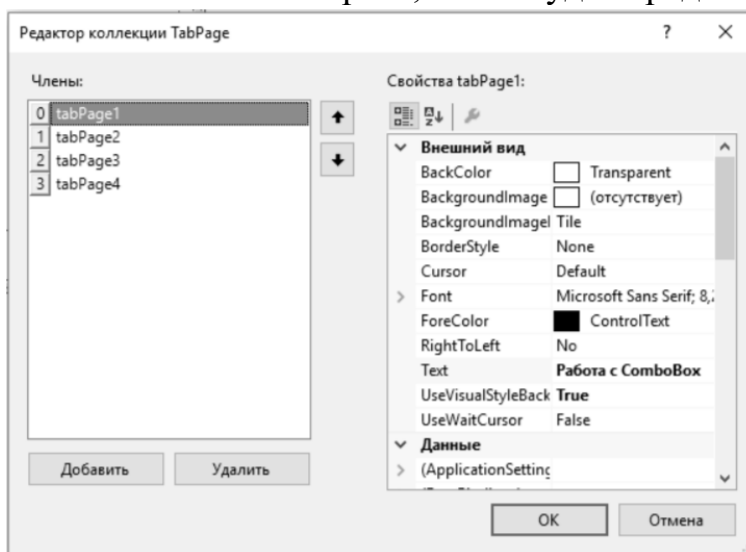


2. Разработка формы. Выберите форму, установите свойство text формы в значение Использование нескольких компонентов, свойство size в значение 537; 313, свойство Locked установите в значение True. Выберите компонент TabControl, расположите её на форму и установите свойства этого компонента: Size – 537; 313. Locked – True.

В компоненте TabControl, по умолчанию, уже есть 2 вкладки, это компоненты tabPage. Добавьте еще две дополнительные вкладки, это можно сделать многими способами однако тут будут описаны два способа: 1) Щелкните правой кнопкой мыши в область, где находятся данные вкладки и в контекстном меню выберите пункт добавить вкладку



2) Выберите компонент TabControl и выберите свойство TabPages. Нажмите на многоточие справа, и вам будет представлено диалоговое окно



Данное окно дает нам возможность отредактировать должным образом созданные tabPage. Обратите внимание, что нумерация tabPage начинается с нуля, это нам пригодится в будущем.

Отредактируйте свойство text у каждого tabPage: Tabpage1 - Работа с ComboBox, Tabpage2 - Работа с ListBox, Tabpage3 - Работа с RadioButton, Tabpage4 - Работа с NumericUpDown.

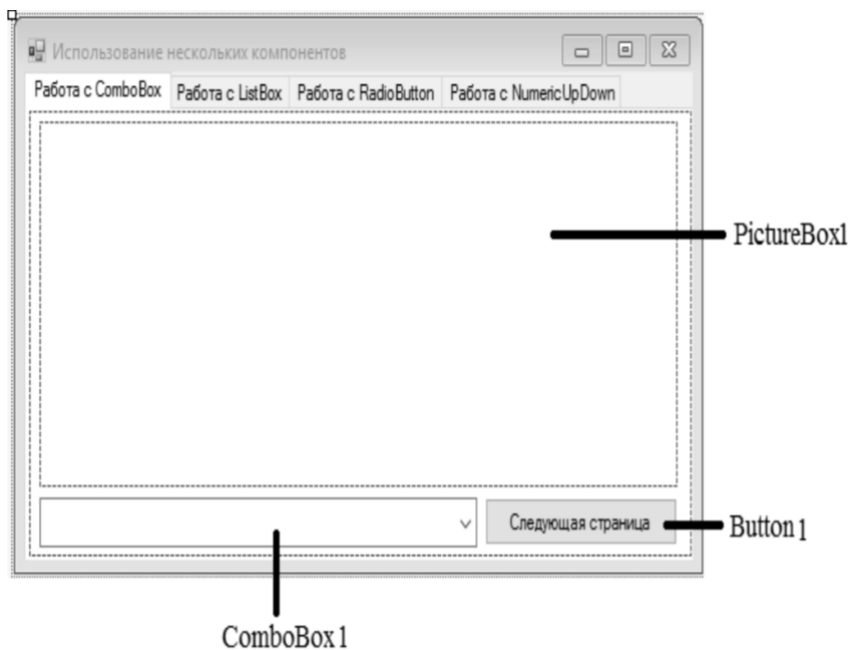
3. Выберите вкладку Работа с ComboBox. Расположите компоненты ComboBox, Button и PictureBox в соответствии с образцом.

Выберите компонент ComboBox1, найдите свойство Items нажмите на кнопку многоточия справа, и в отрывшееся окно введите следующее:

Изображение 1

Изображение 2

Изображение 3



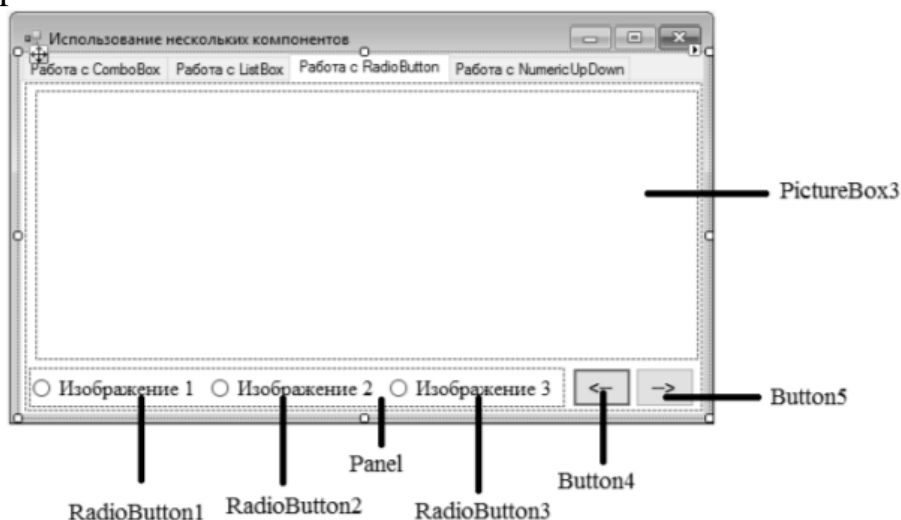
4. Перейдите на вкладку работа с ListBox. Оформите в соответствии с образцом. Выберите компонент ListBox1, найдите свойство Items нажмите на кнопку многоточия справа, и в отрывшееся окно введите следующее:

Изображение 1

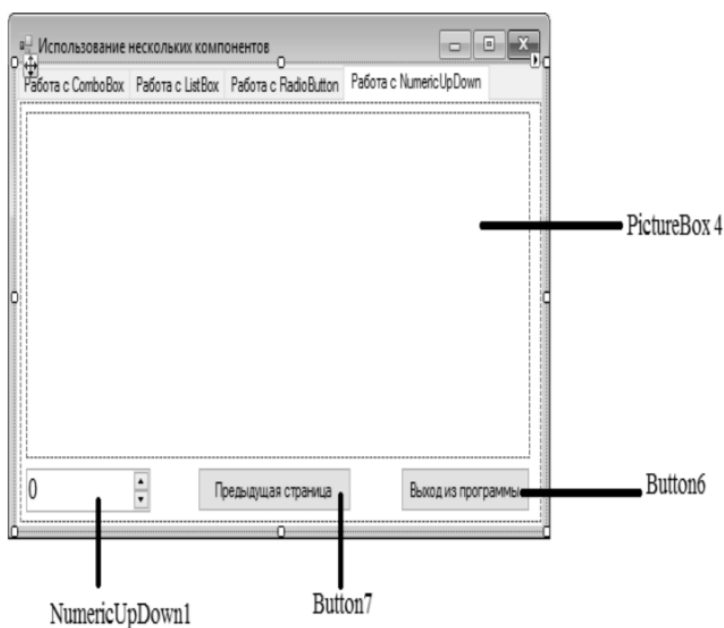
Изображение 2

Изображение 3

Обратите внимание, на компонент Panel, который служит для группировки различных компонентов. Прежде, чем установить RadioButton нужно установить panel.



5. Перейдите на вкладку работа с NumericUpDown. Оформите в соответствии с образцом. Выберите компонент NumericUpDown1 и установите свойство Maximum в значение 2.



6. Теперь, когда форма готова, нужно заполнить её кодом. • Перейдите на вкладку работа с ComboBox и дважды щелкните по компоненту ComboBox1. Автоматически создается событие SelectedIndexChanged. Введите следующий код:

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e) {
    switch (comboBox1.SelectedIndex) // Определяем значение comboBox1
    { case 0:
      { pictureBox1.Visible = true;
        //в данном примере мы загружаем картинку из определенного места на диске
        pictureBox1.Image = Image.FromFile(@"C:\Users\Admin\Documents\Visual Studio
        2022\pic\1.jpg");
        break; }
      case 1:
      { pictureBox1.Visible = true;
        pictureBox1.Image = Image.FromFile(@"C:\Users\Admin\Documents\Visual Studio
        2022\pic\2.jpg");
        break; }
      case 2:
      { pictureBox1.Visible = true;
        pictureBox1.Image = Image.FromFile(@"C:\Users\Admin\Documents\Visual Studio
        2022\pic\3.jpg"); break; } } }
```

7. Теперь дважды щелкните по компоненту button1 и наберите следующий код:

```
private void button1_Click(object sender, EventArgs e){
    tabControl1.SelectedIndex = 1; }
```

8. Перейдите на вкладку работа с ListBox. Дважды щелкните по компоненту listBox1. Автоматически создается событие SelectedIndexChanged. Введите следующий код:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e) {
    switch (listBox1.SelectedIndex) { // Определяем значение listBox1
    case 0: {
```

```

pictureBox2.Visible = true;
//в данном примере мы загружаем картинку из определенного места на диске
pictureBox2.Image = Image.FromFile(@"C:\Users\Admin\Documents\Visual Studio
2022\pic\4.jpg");
break; }
case 1:
{ pictureBox2.Visible = true;
pictureBox2.Image = Image.FromFile(@"C:\Users\Admin \Documents\Visual Studio
2022\pic\5.jpg");
break; }
case 2:
{ pictureBox2.Visible = true;
pictureBox2.Image = Image.FromFile(@"C:\Users\Admin \Documents\Visual Studio
2022\pic\6.jpg");
break; } } }

```

Дважды щелкните по компоненту button2 и наберите следующий код:

```

private void button2_Click(object sender, EventArgs e) {
tabControl1.SelectedIndex = 0; }

```

Дважды щелкните по компоненту button3 и наберите следующий код:

```

private void button2_Click(object sender, EventArgs e){
tabControl1.SelectedIndex = 2; }

```

9.Перейдите на вкладку работа с RadioButton. Дважды щелкните по компоненту RadioButton1. Автоматически создается событие CheckedChanged. Введите следующий код:

```

private void radioButton1_CheckedChanged(object sender, EventArgs e) {
// проверяем radioButton1, если он выбран загружаем картинку.
if (radioButton1.Checked == true)
{ pictureBox3.Visible = true;
pictureBox3.Image = Image.FromFile(@"C:\Users\Rinat\Documents\Visual Studio
2022\pic\7.jpg"); } }

```

Самостоятельно напишите код для radioButton2 и radioButton3, учтите, что они должны загружать разные изображения. Также исходя из примеров выше, напишите код для button4 и button5.

10. Перейдите на вкладку работа с NumericUpDown1. Дважды щелкните по компоненту NumericUpDown1. Автоматически создается событие ValueChanged. Введите следующий код:

```

private void numericUpDown1_ValueChanged(object sender, EventArgs e) {
if (numericUpDown1.Value == 0) {
pictureBox4.Visible = true;
pictureBox4.Image = Image.FromFile(@"C:\Users\Rinat\Documents\Visual Studio
2022\pic\10.jpg"); }
if (numericUpDown1.Value == 1) {
pictureBox4.Visible = true;
pictureBox4.Image = Image.FromFile(@"C:\Users\Rinat\Documents\Visual Studio
2022\pic\11.jpg"); }

```

```
if (numericUpDown1.Value == 2) {  
    pictureBox4.Visible = true;  
    pictureBox4.Image = Image.FromFile(@"C:\Users\Rinat\Documents\Visual Studio  
2022\pic\12.jpg"); } } •
```

Напишите самостоятельно код для button7 (Предыдущая страница).

11. Наберите для кнопки button6 (Выход из программы) следующий код, который закрывает форму и завершает работу программы:

```
private void button6_Click(object sender, EventArgs e) {  
    Close();}
```

12. Протестируйте программу. Запустите на выполнение. Обратите внимание, что размеры изображения «скачут», самостоятельно исправьте это.

ПРАКТИЧЕСКАЯ РАБОТА № 51

Тема: Разработка многооконного приложения. Компиляция и запуск приложения

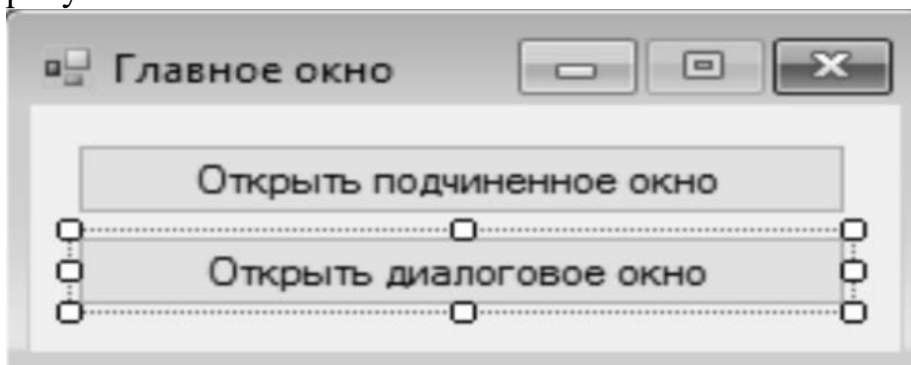
Цель работы: изучить основные способы разработки многооконных приложений, получить практические навыки в создании многооконных приложений.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Создать проект с тремя формами. Первая, она же главная, будет иметь две кнопки открыть подчиненное окно и открыть диалоговое окно, вторая форма будет считать, сколько раз её уже открыли, а при попытке повторно открыть её с главной формы, будет появляться сообщение о закрытии формы. Третья форма (диалоговое окно) будет изменять заголовки главного окна и подчинённого окна.

1. Создайте проект, назовите его Forms.
2. Перенесите два компонента button на форму и расположите их как показано на рисунке



3. Задайте форме и двум кнопкам свойства в соответствии с таблицей

Компонент	Свойство	Значение
Form1	Text	Главное окно
	MaximizeBox	False
	FormBorderStyle	FixedSingle
Button1	Text	Открыть подчиненное окно
Button2	Text	Открыть диалоговое окно

4. Создать дополнительные формы с помощью команды Проект – Добавить форму Windows и в появившемся окне выберите Форма Windows Forms. Создайте подобным образом две формы. Перейдите в конструктор второй формы и добавьте на форму компонент label, он нам понадобится в дальнейшем. Измените свойство второй формы в соответствии с таблицей

Свойство	Значение
Text	Подчиненное окно
StartPosition	Manual
ShowInTaskBar	False

5. Оформите вторую форму по образцу

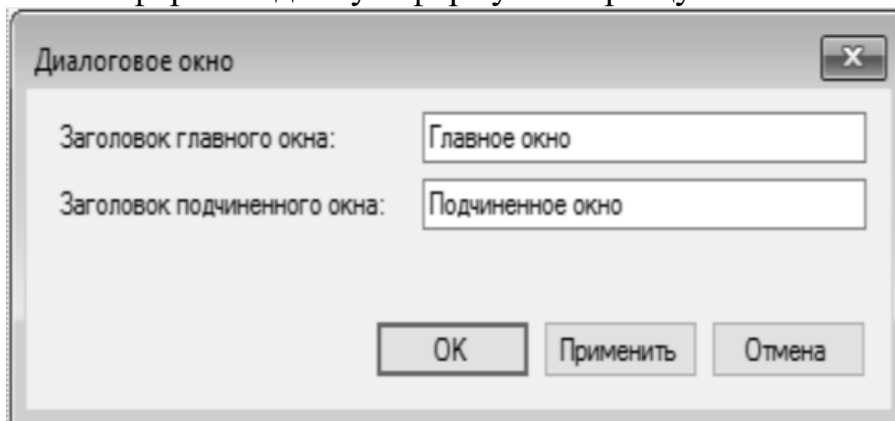


Задайте для label1 свойство AutoSize в значение False, свойство Dock в значение Fill, а свойство TextAlign установите в значение MiddleCenter.

6. Перейдите в конструктор третьей формы и добавьте на форму два компонента label, два компонента TextBox и 3 компонента Button. Измените свойства формы и компонентов в соответствии с таблицей

Компонент	Свойство	Значение
Form3	Text	Диалоговое окно
	MaximizeBox	False
	MinimizeBox	False
	FormBorderStyle	FixedDialog
	StartPosition	CenterScreen
	ShowInTaskbar	False
	AcceptButton	Button1
	CancelButton	Button3
Label1	Text	Заголовок главного окна
Label2	Text	Заголовок подчиненного окна
TextBox1	Text	Главное окно
TextBox2	Text	Подчиненное окно
Button1	Text	ОК
	DialogResult	ОК
Button2	Text	Применить
Button3	Text	Cancel
	DialogResult	Cancel

Оформите данную форму по образцу



7. Заполнение первой формы кодом. Перейдите на первую форму и нажмите клавишу F7. После строк:
namespace FORMS {

```
public partial class Form1 : Form {
```

Введите следующее:

```
private Form2 form2 = new Form2();
```

```
private Form3 form3 = new Form3();
```

Также после строк:

```
public Form1() { InitializeComponent();
```

Введите:

```
AddOwnedForm(form2);
```

```
AddOwnedForm(form3);
```

8. Перейдите вновь в конструктор, выберите форму и на панели свойств и событий выберите событие Shown. Дважды щелкните по событию. Впишите в данное событие следующую строку кода:

```
form2.Location = new Point(Right - 10, Bottom - 10);
```

Для события нажатия на кнопку button1 впишите следующее:

```
if (form2.Visible) form2.Close();
```

```
else form2.Show();
```

Впишите для события нажатия на кнопку button2 следующее:

```
if (form3.ShowDialog() == DialogResult.OK)
```

```
form3.button2_Click(this, EventArgs.Empty);
```

С учетом проделанной работы полный листинг для первой формы должен выглядеть вот так:

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.ComponentModel;
```

```
using System.Data;
```

```
using System.Drawing;
```

```
using System.Text;
```

```
using System.Windows.Forms;
```

```
namespace FORMS {
```

```
public partial class Form1 : Form {
```

```
private Form2 form2 = new Form2();
```

```
private Form3 form3 = new Form3();
```

```
public Form1() {
```

```
InitializeComponent();
```

```
AddOwnedForm(form2);
```

```
AddOwnedForm(form3); }
```

```
private void Form1_Shown(object sender, EventArgs e) {
```

```
form2.Location = new Point(Right - 10, Bottom - 10); }
```

```
private void button1_Click(object sender, EventArgs e) {
```

```
if (form2.Visible) form2.Close();
```

```
else form2.Show(); }
```

```
private void button2_Click(object sender, EventArgs e) {
```

```
if (form3.ShowDialog() == DialogResult.OK)
```

```
form3.button2_Click(this, EventArgs.Empty); } } }
```

9. Заполнение второй формы кодом. Перейдите на вторую форму и нажмите клавишу F7. После строк:

```
namespace FORMS {  
public partial class Form2 : Form {
```

Введите:

```
private int count;
```

Измените у второй формы два события, а именно FormClosing и VisibleChanged. Листинг для события VisibleChanged:

```
private void Form2_VisibleChanged(object sender, EventArgs e) {  
Owner.Controls["button1"].Text = Visible ? "Заккрыть подчиненное окно" :  
"Открыть подчиненное окно";  
if (Visible)  
label1.Text = "Окно открыто " + (++count) + "-й раз."; }
```

Листинг для события FormClosing:

```
private void Form2_FormClosing(object sender, FormClosingEventArgs e) {  
if (e.CloseReason == CloseReason.UserClosing) {  
e.Cancel = true;  
if (MessageBox.Show("Заккрыть подчиненное окно?", "Подтверждение",  
MessageBoxButtons.YesNo, MessageBoxIcon.Question,  
MessageBoxDefaultButton.Button2) == DialogResult.Yes)  
{ Hide();  
Owner.Activate(); } } }
```

С учетом проделанной работы полный листинг для второй формы должен выглядеть вот так:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Text;  
using System.Windows.Forms;  
namespace FORMS {  
public partial class Form2 : Form {  
private int count;  
public Form2() {  
InitializeComponent(); }  
private void Form2_FormClosing(object sender, FormClosingEventArgs e) {  
if (e.CloseReason == CloseReason.UserClosing) {  
e.Cancel = true;  
if (MessageBox.Show("Заккрыть подчиненное окно?", "Подтверждение",  
MessageBoxButtons.YesNo, MessageBoxIcon.Question,  
MessageBoxDefaultButton.Button2) == DialogResult.Yes) {  
Hide();  
Owner.Activate(); } } }  
private void Form2_VisibleChanged(object sender, EventArgs e) {
```

```
Owner.Controls["button1"].Text = Visible ? "Закрыть подчиненное окно" :  
"Открыть подчиненное окно";
```

```
if (Visible) label1.Text = "Окно открыто в " + (++count) + "-й раз."; } } }
```

10. Заполнение третьей формы кодом. Перейдите на третью форму. Выберите её. Найдите событие `VisibleChanged` дважды щелкните на событие и вставьте следующий код:

```
if (Visible) ActiveControl = textBox1;
```

Выберите `Button2`. У данного компонента нас интересует событие `Click`, дважды щелкните на событие, Visual Studio сформирует следующий код:

```
private void button2_Click(object sender, EventArgs e) {  
}
```

Измените свойство `private` на `internal`, чтобы получилось так:

```
internal void button2_Click(object sender, EventArgs e) {  
}
```

И вставьте следующий код:

```
Owner.Text = textBox1.Text;
```

```
Owner.OwnedForms[0].Text = textBox2.Text;
```

С учетом проделанной работы полный листинг для третьей формы должен выглядеть так:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Text;  
using System.Windows.Forms;  
namespace FORMS {  
    public partial class Form3 : Form {  
        public Form3() {  
            InitializeComponent(); }  
        internal void button2_Click(object sender, EventArgs e) {  
            Owner.Text = textBox1.Text;  
            Owner.OwnedForms[0].Text = textBox2.Text; }  
        private void Form3_VisibleChanged(object sender, EventArgs e) {  
            if (Visible) ActiveControl = textBox1; } } }
```

11. Проверка программы и самостоятельная работа. Запустите и проверьте работоспособность программы. Доработайте третье окно (диалоговое окно) так, чтобы с помощью этого окна можно было изменять не только заголовки первых двух форм, но и другие свойства (3 на выбор).

ПРАКТИЧЕСКАЯ РАБОТА № 52

Тема: Разработка многооконного приложения. Компиляция и запуск приложения

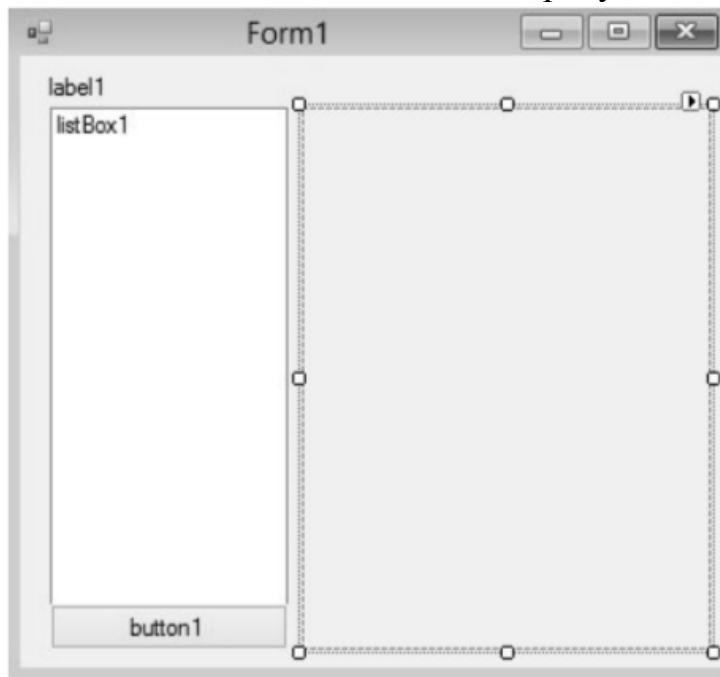
Цель работы: изучить основные способы разработки многооконных приложений, получить практические навыки в создании многооконных приложений.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Создать простую программу для просмотра изображений.

1. Поместите в форму несколько компонентов: ListBox, PictureBox, Button и Label. Расположите их как показано на рисунке



2. Измените у кнопки свойство Text на Указать директорию. Измените у формы свойство Text на Галерея. Измените у компонента PictureBox1 свойство.SizeMode на StretchImage. Компонент label1 будет показывать директорию, которую указал пользователь. ListBox1 будет отображать список названий всех изображений, PictureBox1 будет отображать выбранное в ListBox1 изображение. Кнопка button1 будет вызывать диалог FolderBrowserDialog, через который в свою очередь пользователь будет указывать каталог с изображениями.

3. Редактируем код программы. Добавим пространство имен

`using System.IO;`

в начало нашего кода, т.к. нам необходимо работать с файлами и директориями. И объявим новую переменную типа string для хранения в ней пути к директории

```
string path; //Путь к директории с файлами перед конструктором класса
public Form1() {
    InitializeComponent(); }
```

Заполняем список названиями изображений:

Для этого необходимо добавить и реализовать новую функцию, которая в качестве входного параметра будет принимать путь указанный нами:

```

private Boolean FillListBox(string patch) {
//Объявляем экземпляр класса DirectoryInfo используя конструктор,
принимаящий в качестве параметра путь к файлам
DirectoryInfo directory = new DirectoryInfo(patch);
//Считываем все файлы в папке с расширением .jpg
FileInfo[] files = directory.GetFiles("*.jpg");
listBox1.Items.Clear();//Очищаем список - если в нем что-то уже было f
foreach (FileInfo Files in files) {
listBox1.Items.Add(Files.Name);//Добавляем новые компоненты в список
}
label1.Text = patch;//Указываем путь выбранной нами директории
//Если папка не содержит таких файлов возвращаем false
if (files.Length == 0) return false;
else {
//Выбираем 1 файл из полученного списка
listBox1.SelectedIndex = 0;
return true; } }

```

4. Выводим изображения на экран. Для события SelectedIndexChanged компонента listBox1 сделаем такую реализацию:

```

private void listBox1_SelectedIndexChanged(object sender, EventArgs e) {
pictureBox1.Image = new Bitmap(patch + "\\" + listBox1.SelectedItem.ToString()); }

```

Создание диалогового окна для выбора директории: пишем для компонента button1 на событие Click такую реализацию:

```

private void button1_Click(object sender, EventArgs e) {
FolderBrowserDialog fbd = new FolderBrowserDialog();
fbd.Description = "Выберите папку"; //Описание внутри диалога
//После того, как указали директорию с файлами
if (fbd.ShowDialog() == DialogResult.OK) {
patch = fbd.SelectedPath; //Записываем текущий путь в переменную
label1.Text = patch; //Указываем его в качестве текста компонента label
//Если в директории нет изображений, то соответственно картинка
//показываться не будет
if (!FillListBox(fbd.SelectedPath))
pictureBox1.Image = null; } }

```

5. Полный код программы и результат:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

```

```

namespace ImageViewer {
public partial class Form1 : Form {
string patch;          //Путь к директории с файлами
public Form1() {
InitializeComponent(); }
private Boolean FillListBox(string patch) {
DirectoryInfo directory = new DirectoryInfo(patch) files =
directory.GetFiles("*.jpg");
listBox1.Items.Clear();          //Очищаем список - если в нем что-то уже было
foreach (FileInfo Files in files) {
listBox1.Items.Add(Files.Name) }
label1.Text = patch;
if (files.Length == 0) return false;
else {          //Выбираем 1 файл из полученного списка
listBox1.SelectedIndex = 0;
return true; } }
private void listBox1_SelectedIndexChanged(object sender, EventArgs e) {
pictureBox1.Image = new Bitmap(patch + "\\" + listBox1.SelectedItem.ToString()); }
private void button1_Click(object sender, EventArgs e) {
//Экземпляр класса FolderBrowserDialog
FolderBrowserDialog fbd = new FolderBrowserDialog();
fbd.Description = "Выберите папку";          //Описание внутри диалога
//После того, как указали директорию с файлами
if (fbd.ShowDialog() == DialogResult.OK) {
patch = fbd.SelectedPath;          //Записываем текущий путь в переменную
label1.Text = patch;          //Указываем его в качестве текста компонента label
//Если в директории нет изображений, то соответственно картинка показываться
//не будет
if (!FillListBox(fbd.SelectedPath))
pictureBox1.Image = null; } } } }

```

6. Запустите программу. Пример работы программы



ПРАКТИЧЕСКАЯ РАБОТА № 53

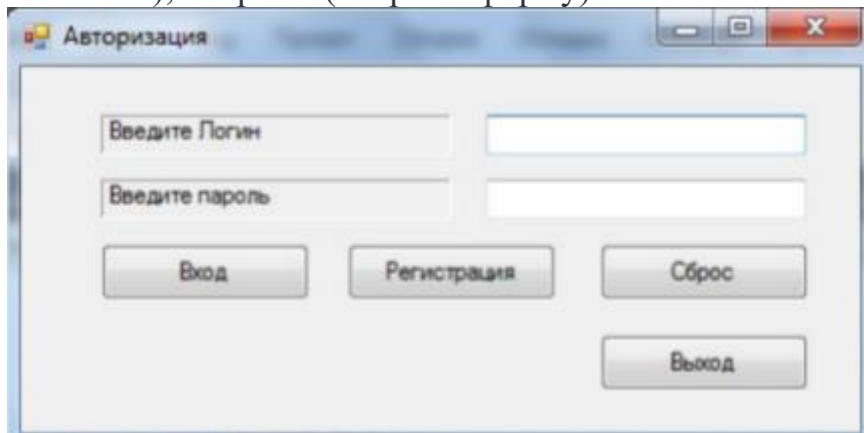
Тема: Разработка интерфейса приложения

Цель работы: получить практический опыт разработки пользовательского Windows-интерфейса с использованием WindowsForm.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Необходимо построить приложение, содержащее форму входа пользователя в систему и форму регистрации нового пользователя. Необходимо также написать процедуры обработки событий для кнопок Регистрация (вызов формы Регистрация нового пользователя), Сброс (очистка окна), Выход (выход из приложения), Закреть (Закреть форму).



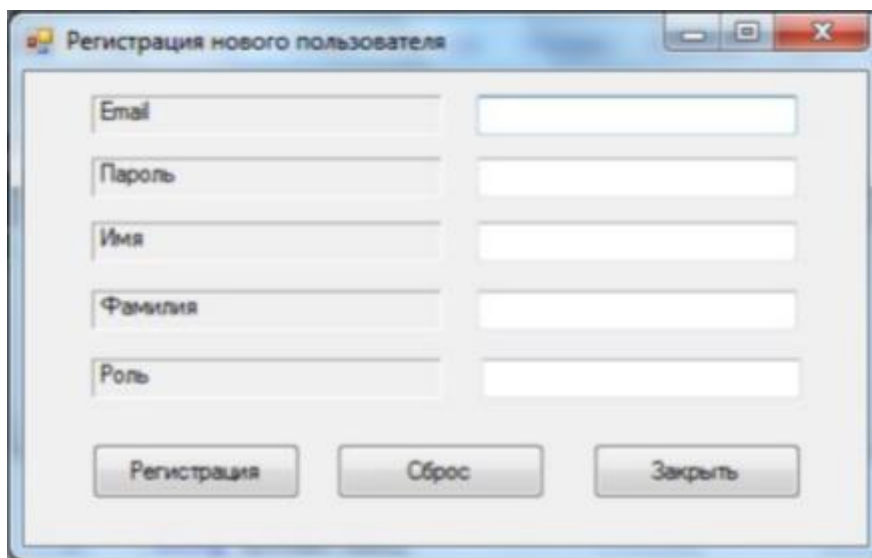
Авторызация

Введите Логин

Введите пароль

Вход Регистрация Сброс

Выход



Регистрация нового пользователя

Email

Пароль

Имя

Фамилия

Роль

Регистрация Сброс Закреть

ПРАКТИЧЕСКАЯ РАБОТА № 54

Тема: Разработка интерфейса приложения

Цель работы: получить практический опыт разработки пользовательского Windows-интерфейса с использованием WindowsForm.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. В соответствии с вариантом задания разработать интерфейс основных программных модулей с использованием WindowsForms.

Пример интерфейса пользователя для приложения Библиотека:



Необходимо разработать главную кнопочную форму. На нее поместить надписи, изображения и кнопки. Одна из кнопок должна обеспечивать окончание работы с информацией. Оформить с помощью свойств созданные объекты. Добавить формы в проект в соответствии с темой. Организовать взаимодействие между двумя формами, добавив код вызова форм.

Варианты заданий:

1. Автотранспортное предприятие. Основные объекты предметной области: автомобили, марки автомобилей, водители, заказчики, грузы. Основные функции: учет заказов по перевозке грузов и подбор автомобилей. Выходные документы: накладная на перевозку груза для водителя.
2. Агентство недвижимости. Основные объекты предметной области: квартиры, продавцы, покупатели, агенты, сделки. Основные функции: учет заказов и подбор вариантов в соответствии с требованиями. Выходные документы: договор на осуществление сделок.
3. Банк. Основные объекты предметной области: вкладчики, типы счетов, счета, поступления на счёт, снятия со счета. Основные функции: учет вкладов и начисление процентов по вкладам. Выходные документы: договор на открытие вклада.
4. Поликлиника. Основные объекты предметной области: пациенты, посещения, врачи, специальности, диагнозы. Основные функции: учет посещений и ведение медицинских карт. Выходные документы: журнал посещений врача.

5. Юридическое агентство. Основные объекты предметной области: клиенты, услуги, номенклатура услуг. Основные функции: учет предоставленных услуг. Выходные документы: ведомость оказания услуг.
6. Отдел продаж. Основные объекты предметной области: товары, продажи, продавцы, покупатели. Основные функции: учет продаж. Выходные документы: товарная накладная.
7. Отель. Основные объекты предметной области: номерной фонд, типы номеров, занятость номеров, клиенты, заказы, сотрудники. Основные функции: учет занятости номеров. Выходные документы: журнал учета проживающих.
8. Фитнес-клуб. Основные объекты предметной области: клиенты, тренеры, абонементы. Основные функции: учет продажи абонементов. Выходные документы: журнал продажи абонементов.
9. Заказ такси. Основные объекты предметной области: клиенты, поездки, автомобили, водители. Основные функции: Учет заказов. Выходные документы: журнал заказов.
10. Турфирма. Основные объекты предметной области: клиенты, продажи, страны, туры. Основные функции: подбор туров. Выходные документы: журнал продаж.

ПРАКТИЧЕСКАЯ РАБОТА № 55

Тема: Разработка интерфейса приложения

Цель работы: получить практический опыт разработки пользовательского Windows-интерфейса с использованием WindowsForm.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. В соответствии с вариантом задания разработать интерфейс основных программных модулей с использованием WindowsForms.

Пример интерфейса пользователя для приложения Библиотека:



Необходимо разработать главную кнопочную форму. На нее поместить надписи, изображения и кнопки. Одна из кнопок должна обеспечивать окончание работы с информацией. Оформить с помощью свойств созданные объекты. Добавить формы в проект в соответствии с темой. Организовать взаимодействие между двумя формами, добавив код вызова форм.

Варианты заданий:

1. Центр детского творчества. Объекты предметной области: учащиеся, кружки, занятия. Основные функции: учет учащихся центра. Выходные документы: журнал учащихся.
2. Аптека. Объекты предметной области: лекарства, показания, поставщики, пациенты. Основные функции: учет заказов лекарств. Выходные документы: журнал заказов
3. Магазин бытовой техники. Объекты предметной области: категории товаров, условия кредитов, условия доставки. Выходные документы: прайс-лист.
4. Ателье пошива одежды. Объекты предметной области: портные, услуги, материалы. Выходные документы: прайс-лист.
5. Театр. Объекты предметной области: актеры, спектакли. Выходные документы: афиша.
6. Зоопарк. Объекты предметной области: животные, мероприятия. Выходные документы: расписание мероприятий.

7. Станция техобслуживания. Объекты предметной области: оказываемые услуги, обслуживающий персонал. Выходные документы: прайс-лист.

8. Кредитный отдел банка. Основные объекты предметной области: клиенты, типы кредитов, счета, график платежей. Основные функции: учет выплат и начисление процентов по вкладам. Выходные документы: договор на открытие кредита.

9. Пункт обмена валют. Объекты предметной области: курс валют, клиенты, заказ валют, наличие. Выходные документы: журнал заказа.

10. Отдел кадров. Основные объекты предметной области: должности, отделы, сотрудники, квалификации сотрудников. Основные функции: начисление зарплаты. Выходные документы: ведомость сотрудников.

ПРАКТИЧЕСКАЯ РАБОТА № 56

Тема: Тестирование, отладка приложения

Цель работы: изучить процессы тестирования и отладки программного обеспечения; научиться проводить тестирование приложения, исправлять ошибки.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Справочный материал:

Отладка – это процесс локализации и исправления ошибок, обнаруженных при тестировании программного обеспечения.

Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, т.е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты.

Классификация ошибок

В соответствии с этапом обработки, на котором появляются ошибки, различают:

- *синтаксические ошибки* – ошибки, фиксируемые компилятором (транслятором, интерпретатором) при выполнении синтаксического и частично семантического анализа программы;
- *ошибки компоновки* – ошибки, обнаруженные компоновщиком (редактором связей) при объединении модулей программы;
- *ошибки выполнения* – ошибки, обнаруженные операционной системой, аппаратными средствами или пользователем при выполнении программы.

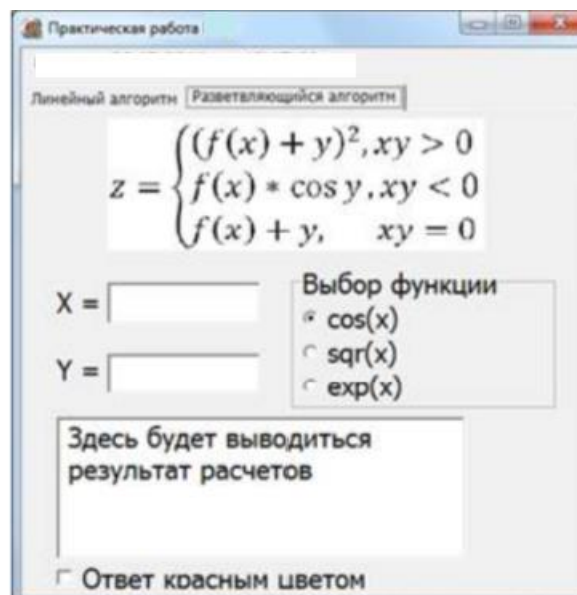
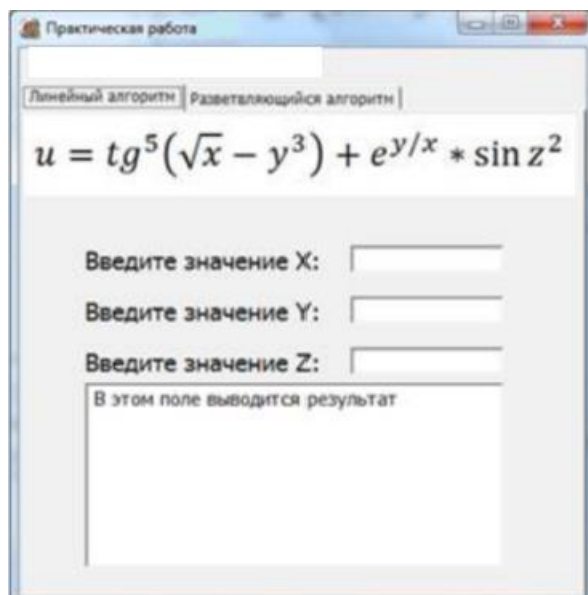
Методы отладки программного обеспечения

Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации. При этом используют различные методы:

- ручного тестирования;
- индукции;
- дедукции;
- обратного прослеживания.

Содержание работы:

Задание 1. Создать Windows-приложение, реализующие линейный и разветвляющийся алгоритмы, которые размещены на разных вкладках окна формы. На вкладке линейного алгоритма предусмотреть поля ввода значений переменных и поле вывода результата вычисления. На вкладке разветвляющегося алгоритма предусмотреть поля для ввода значений переменных, поле вывода результатов расчета по одной из трех формул в зависимости от результата выполнения условия. В качестве $f(x)$ использовать по выбору: $\cos(x)$ или x^2 или e^x .



Протестируйте созданное приложение, проверив весь функционал программы, вводя разные входные данные. В случае нахождения ошибок, исправьте их. Заполните таблицу:

№ теста	Исходные данные	Результат	Описание ошибки

Задание 2. Протестируйте созданное приложение в Практической работе № 55, проверив весь функционал программы, вводя разные входные данные. В случае нахождения ошибок, исправьте их. Заполните таблицу:

№ теста	Исходные данные	Результат	Описание ошибки

ПРАКТИЧЕСКАЯ РАБОТА № 57

Тема: Тестирование, отладка приложения

Цель работы: изучить процессы тестирования и отладки программного обеспечения; научиться проводить тестирование приложения, исправлять ошибки.

Оборудование: ПК, программное обеспечение – Visual Studio 2022, инструкции по выполнению работы

Содержание работы:

Задание 1. Протестируйте созданное приложение в Практической работе № 53, проверив весь функционал программы, вводя разные входные данные. В случае нахождения ошибок, исправьте их. Заполните таблицу:

№ теста	Исходные данные	Результат	Описание ошибки

Задание 2. Протестируйте созданное приложение в Практической работе № 54, проверив весь функционал программы, вводя разные входные данные. В случае нахождения ошибок, исправьте их. Заполните таблицу:

№ теста	Исходные данные	Результат	Описание ошибки

Информационное обеспечение обучения

Основные учебные издания:

1. Лебеденко, Л. Ф. Технологии программирования: учебно-методическое пособие для СПО / Л. Ф. Лебеденко, О. И. Моренкова. — 2-е изд. — Саратов: Профобразование, 2024. — 108 с. — ISBN 978-5-4488-1204-0. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/139115>
2. Медведев, М. А. Программирование на СИ#: учебное пособие для СПО / М. А. Медведев, А. Н. Медведев; под редакцией А. В. Присяжного. — 3-е изд. — Саратов, Екатеринбург: Профобразование, Уральский федеральный университет, 2024. — 62 с. — ISBN 978-5-4488-0471-7, 978-5-7996-2833-8. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/139593>

Дополнительные учебные издания:

3. Биллиг, В. А. Основы программирования на С#: учебное пособие / В. А. Биллиг. — 4-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2025. — 573 с. — ISBN 978-5-4497-0893-9. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/146368>
4. Зыков, С. В. Введение в теорию программирования. Объектно-ориентированный подход: учебное пособие для СПО / С. В. Зыков. — 2-е изд. — Саратов: Профобразование, 2024. — 187 с. — ISBN 978-5-4488-0995-8. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/139748>
5. Маляров, А. Н. Объектно-ориентированное программирование: учебник для СПО / А. Н. Маляров. — 2-е изд. — Саратов: Профобразование, 2022. — 334 с. — ISBN 978-5-4488-1561-4. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/132418>

Интернет-ресурсы:

Электронно-библиотечная система:

6. ЭБС «Znanium»
7. ЭБС «PROФобразование»
8. ЭБС «Book.ru»