**АРХАНГЕЛЬСКИЙ КОЛЛЕДЖ ТЕЛЕКОММУНИКАЦИЙ (ФИЛИАЛ) СПбГУТ**

**(АКТ (ф) СПбГУТ)**

# Отчеты по лабораторным и практическим работам

**по МДК.01.01**

Студент: Д.А. Капитанская

Группа: ИСПП-21

Преподаватель: Ю.С. Маломан

Архангельск 2024

# Лабораторная работа №1

**Оценка сложности алгоритмов сортировки**

1. **Цель работы**
   1. Научиться реализовывать и оценивать сложность алгоритмов сортировки массивов на C#.

я

1. **Ответы на контрольные вопросы**
   1. Что такое «массив»?

Ответ:

Массив - это структура данных, состоящая из набора элементов (значений или переменных) одинакового объема памяти, каждый из которых идентифицируется по крайней мере одним индексом или ключом массива.

* 1. Как описывается одномерный массив?

Ответ: тип\_переменной[] название\_массива;

* 1. Как обратиться к некоторому элементу одномерного массива?

Ответ: массив [индекс]

* 1. Как можно задать одномерный массив?

Ответ: тип\_переменной[] название\_массива = {переменные массива};

* 1. Что такое «сортировка»?

Ответ: Сортировка — это процесс упорядочивания элементов в определенном порядке (например, по возрастанию или убыванию).

* 1. Что такое «алгоритм сортировки»?

Ответ: Алгоритм сортировки — это последовательность действий, которая определяет порядок элементов в массиве.

* 1. Какие виды сортировки массивов существуют?

Ответ: Существует множество видов сортировки массивов, включая:

* Сортировка пузырьком
* Сортировка выбором
* Сортировка вставками
* Сортировка слиянием
* Быстрая сортировка
* Сортировка подсчетом
* Сортировка кучей
* И другие.

1. **Вывод**
   1. В ходе проделанной лабораторной работы я научился реализовывать и оценивать сложность алгоритмов сортировки массивов на C#.

# Лабораторная работа №2

**Оценка сложности алгоритмов поиска**

1. **Цель работы**
   1. Научиться реализовывать и оценивать сложность алгоритмов поиска элементов массивов на C#.
2. **Ответы на контрольные вопросы**
   1. Какие виды поиска элементов массивов существуют?

Ответ: Существует несколько видов поиска элементов в массивах, такие как линейный, двоичный, поиск прыжками и другие.

* 1. В чем особенность алгоритма линейного поиска и какова его временная сложность? Особенность алгоритма линейного поиска заключается в том, что он проверяет каждый элемент массива последовательно, пока не найдет искомый элемент либо не достигнет конца массива. В лучшем случае его временная сложность составляет O(1), а в худшем O(n), где n - количество элементов в массиве.

Ответ:

* 1. В чем особенность алгоритма двоичного поиска и какова его временная сложность?

Ответ: Алгоритм двоичного поиска работает только с отсортированными массивами. Его особенность заключается в том, что он делит массив пополам и сравнивает искомый элемент с элементом в середине массива. В зависимости от результата сравнения, он выбирает одну половину массива для дальнейшего поиска. Его временная сложность составляет O(log n).

* 1. В чем особенность алгоритма поиска прыжками и какова его временная сложность?

Ответ: Алгоритм поиска прыжками, или интерполяционный поиск, использует упорядоченный массив. Он делает предположение о расположении искомого элемента, основываясь на значениях самого массива. Затем он прыгает к приблизительному местоположению искомого элемента и, если необходимо, делает последовательные перемещения вперед или назад. В среднем его временная сложность составляет O(log log n), что делает его очень эффективным для больших упорядоченных массивов.

1. **Вывод**
   1. В ходе проделанной лабораторной работы я научился реализовывать и оценивать сложность алгоритмов поиска элементов массивов на C#.

# Лабораторная работа №3

**Оценка сложности рекурсивных алгоритмов**

1. **Цель работы**
   1. Научиться разрабатывать и оценивать сложность рекурсивных функций в программах на C#.
2. **Ответы на контрольные вопросы**
   1. Что такое «рекурсия»?

Ответ: Рекурсия - это прием программирования, при котором функция вызывает саму себя внутри своего тела.

* 1. Какие проблемы могут возникать при реализации рекурсивных алгоритмов на электронных вычислительных машинах?

Ответ: Проблемы, которые могут возникать при реализации рекурсивных алгоритмов на электронных вычислительных машинах, включают в себя потенциальный переполнение стека из-за слишком большой глубины рекурсии, что может привести к аварийному завершению программы.

Какое определение функции может быть названо рекурсивным? Привести примеры

Ответ:

* 1. . Функция может быть названа рекурсивной, если она вызывает саму себя внутри своего определения. Например, факториал числа можно реализовать рекурсивной функцией:

int Factorial(int n)

{

if (n == 0)

return 1;

else

return n \* Factorial(n - 1);

}

* 1. Что такое «глубина рекурсии»?

Ответ: Глубина рекурсии - это количество раз, которое функция вызывает саму себя.

* 1. Что такое «рекурсивный спуск»?

Ответ: Рекурсивный спуск - это метод построения синтаксических анализаторов, основанный на использовании рекурсии для обхода структуры входного текста с целью его анализа.

* 1. Что такое «рекурсивный подъём»?

Ответ: Рекурсивный подъем - это метод анализа и преобразования контекстно-свободных грамматик, который основан на использовании рекурсивных правил для построения дерева вывода.

1. **Вывод**
   1. В ходе проделанной лабораторной работы я научился разрабатывать и оценивать сложность рекурсивных функций в программах на C#.

# Лабораторная работа №4

**Оценка сложности эвристических алгоритмов**

1. **Цель работы**
   1. Научиться реализовывать и оценивать сложность эвристических алгоритмов на C#.
2. **Ответы на контрольные вопросы**
   1. Что такое «многомерный массив»?

Ответ: Многомерный массив — это массив, элементы которого сами являются массивами.

* 1. Как описывается двумерный массив?

Ответ: Двумерный массив в C# описывается с помощью объявления типа данных и указания двух размерностей массива. Например:

int[,] myArray = new int[3, 4];

* 1. Как обратиться к некоторому элементу двумерного массива?

Ответ: Чтобы обратиться к конкретному элементу двумерного массива в C#, нужно указать индексы этого элемента в квадратных скобках. Например:

int value = myArray[1, 2];

* 1. Как узнать количество строк двумерного массива?

Ответ: Для того, чтобы узнать количество строк двумерного массива в C#, можно использовать свойство GetLength(0).

* 1. Как узнать количество столбцов двумерного массива?

Ответ: Для того, чтобы узнать количество столбцов двумерного массива в C#, можно использовать свойство GetLength(1).

* 1. Как вывести двумерный массив на консоль в виде таблицы?

Ответ: Чтобы вывести двумерный массив на консоль в виде таблицы в C#, можно использовать вложенные циклы. Например:

for (int i = 0; i < myArray.GetLength(0); i++)

{

for (int j = 0; j < myArray.GetLength(1); j++)

{

Console.Write(myArray[i, j] + "\t");

}

Console.WriteLine();

}

1. **Вывод**
   1. В ходе проделанной лабораторной работы я научился реализовывать и оценивать сложность эвристических алгоритмов на C#.

# Лабораторная работа №5

**Работа с классами**

1. **Цель работы**
   1. Изучить процесс разработки и применения классов на языке C#,
   2. Изучить реализацию механизма инкапсуляции на языке C#.
2. **Ответы на контрольные вопросы**
   1. Какова общая форма объявления класса в C#?

Ответ:

class ClassName

{

// Поля, свойства, методы и т.д.

}

* 1. На какие виды делятся данные класса в C#?

Ответ: Данные класса в C# делятся на:

* Поля (fields)
* Свойства (properties)
  1. На какие виды делятся функции класса в C#?

Ответ: Функции класса в C# делятся на:

* Методы (methods)
* Конструкторы (constructors)
  1. Для чего применяются конструкторы классов?

Ответ Конструкторы классов применяются для инициализации новых объектов данного класса.

* 1. Что такое «цепочка конструкторов»?

Ответ: «Цепочка конструкторов» (constructor chaining) - это вызов одного конструктора из другого конструктора в том же классе.

* 1. Для чего применяются свойства классов?

Ответ: Свойства классов применяются для обеспечения доступа к данным объекта класса, контроля значения и выполнения дополнительных действий при чтении или записи.

* 1. Когда используются автоматически реализуемые свойства классов?

Ответ: Автоматически реализуемые свойства классов используются в случае, когда не требуется дополнительная логика при чтении или записи значения свойства.

* 1. В чем отличие вызова статические членов классов от членов экземпляра класса?

Ответ: Вызов статических членов класса происходит через имя класса, тогда как вызов членов экземпляра класса происходит через объект этого класса. Статические члены существуют на уровне самого класса, в то время как члены экземпляра существуют на уровне каждого конкретного объекта класса.

1. **Вывод**
   1. В ходе проделанной лабораторной работы я изучил процесс разработки и применения классов на языке C#,
   2. В ходе проделанной лабораторной работы я изучил реализацию механизма инкапсуляции на языке C#.

# Лабораторная работа №6

**Перегрузка методов**

1. **Цель работы**
   1. Изучить процесс выполнения перегрузки методов на языке C#;
   2. Изучить различные способы передачи параметров в метод.
2. **Ответы на контрольные вопросы**
   1. Что такое «перегрузка методов»?

Ответ: Перегрузка методов - это возможность в языках программирования определять несколько методов с одним и тем же именем, но различающимися параметрами (типами параметров или их количеством). Таким образом, в зависимости от передаваемых аргументов компилятор или интерпретатор выберет соответствующую версию метода для вызова.

* 1. К какому виду полиморфизма относится перегрузка методов?

Ответ: Перегрузка методов относится к полиморфизму времени компиляции (compile-time polymorphism), также известному как статический полиморфизм. Решение о том, какую версию метода вызывать, принимается на этапе компиляции, а не во время выполнения программы.

* 1. Какие типы функций класса можно перегружать?

Ответ: В C# можно перегружать как статические методы, так и методы экземпляра класса.

* 1. Какие требования предъявляются к сигнатуре перегружаемых функций класса?

Ответ: Перегружаемые функции класса должны иметь одинаковое имя, но различаться по количеству параметров или их типам. То есть сигнатуры методов должны отличаться друг от друга.

* 1. Какие типы функций класса нельзя перегружать?

Ответ: Нельзя перегружать функции класса, если их сигнатуры одинаковы и отличаются только типами возвращаемого значения или модификаторами доступа (например, public или private).

1. **Вывод**
   1. В ходе проделанной лабораторной работы я изучил процесс выполнения перегрузки методов на языке C#;
   2. В ходе проделанной лабораторной работы я изучить различные способы передачи параметров в метод.

# Лабораторная работа №7

**Определение операций в классе**

1. **Цель работы**
   1. Изучить процесс разработки дочерних классов на языке C#;
   2. Изучить реализацию механизма наследования на языке C#.
2. **Ответы на контрольные вопросы**
   1. Какое ключевое слово определяет операторный метод?

Ответ: Ключевое слово operator определяет операторный метод в C#.

* 1. Какие унарные операторы можно перегружать в C#?

Ответ: В C# можно перегружать унарные операторы, такие как унарный плюс (+), унарный минус (-), логическое отрицание (!) и так далее.

* 1. Какие бинарные операторы можно перегружать в C#

Ответ: В C# можно перегружать бинарные операторы, например, арифметические операторы (+, -, \*, /), операторы сравнения (==, !=, <, >) и так далее.

* 1. Какие операторы сравнения можно перегружать в C#?

Ответ: В C# можно перегружать операторы сравнения, такие как равенство (==), неравенство (!=), меньше (<), больше (>), меньше или равно (<=), больше или равно (>=).

* 1. Какие операторы требуется перегружать в C# попарно?

Ответ: В C# операторы == и != обязательно нужно перегружать в паре. Если перегружается оператор == , то также нужно перегрузить оператор != , чтобы обеспечить корректное поведение в сравнении объектов на неравенство.

1. **Вывод**
   1. В ходе проделанной лабораторной работы я изучил процесс разработки дочерних классов на языке C#;
   2. В ходе проделанной лабораторной работы я изучил реализацию механизма наследования на языке C#.

# Лабораторная работа №8

**Создание наследованных классов**

1. **Цель работы**
   1. Изучить процесс разработки дочерних классов на языке C#;
   2. Изучить реализацию механизма наследования на языке C#.
2. **Ответы на контрольные вопросы**
   1. Что такое «наследование»?

Ответ: Наследование в программировании - это механизм, который позволяет создавать новый класс на основе существующего класса (родительского класса), заимствуя его поля, свойства и методы. Дочерний класс наследует характеристики родительского класса и может добавлять к ним свои собственные члены.

* 1. Сколько родительских классов может быть у дочернего класса в C#?

Ответ: В C# дочерний класс может наследовать только один родительский класс, то есть поддерживается одиночное наследование.

* 1. Какое ключевое слово позволяет обратиться к реализации родительского класса из дочернего?

Ответ: Ключевое слово base позволяет обратиться к реализации родительского класса из дочернего в C#.

* 1. Что такое «переопределение метода» и как оно выполняется?

Ответ: Переопределение метода в C# - это процесс замены реализации унаследованного метода в дочернем классе своей собственной реализацией. Для выполнения переопределения метода в дочернем классе используется ключевое слово override .

* 1. Что такое «абстрактный класс»?

Ответ: Абстрактный класс в C# - это класс, который содержит хотя бы один абстрактный метод. Абстрактный метод не имеет тела и должен быть реализован в производных классах. Абстрактные классы не могут быть инстанциированы, они используются для предоставления базовой реализации для производных классов.

* 1. Для чего предназначены модификаторы virtual, override, abstract, new?

Ответ: virtual : позволяет методу быть переопределенным в дочерних классах.

override : указывает, что метод представляет переопределение унаследованного метода.

abstract : определяет абстрактный метод или класс.

new : предоставляет новую реализацию члена базового класса в производном классе.

1. **Вывод**
   1. В ходе проделанной лабораторной работы я изучил процесс разработки дочерних классов на языке C#;
   2. В ходе проделанной лабораторной работы я изучил реализацию механизма наследования на языке C#.

# Лабораторная работа №9

**Работа с объектами через интерфейс**

1. **Цель работы**
   1. Изучить процесс разработки и реализации интерфейсов на языке C#.
2. **Ответы на контрольные вопросы**
   1. Что такое «интерфейс»?

Ответ: В C# интерфейс – это контракт, который определяет набор методов, свойств, индексаторов и событий, которые должны быть реализованы любым классом, реализующим этот интерфейс. Интерфейсы не содержат никакой реализации, только декларации

* 1. Чем отличается интерфейс от абстрактного класса?

Ответ: Интерфейс: Описывает, что должен делать класс, без указания как.

Абстрактный класс: Описывает как делать что-то, частично реализуя функциональность.

* 1. Есть ли у класса ограничения по количеству реализуемых интерфейсов?

Ответ: Нет, класс может реализовывать неограниченное количество интерфейсов.

* 1. Какова общая форма объявления интерфейсов и их элементов?

Ответ:

public interface IMyInterface

{

// Метод

void MyMethod(string text);

// Свойство

int MyProperty { get; set; }

// Индексатор

string this[int index] { get; set; }

// Событие

event EventHandler MyEvent;

}

1. **Вывод**
   1. В ходе проделанной лабораторной работы я изучил процесс разработки и реализации интерфейсов на языке C#.

# Лабораторная работа №10

**Использование стандартных интерфейсов**

1. **Цель работы**
   1. Изучить процесс реализации стандартных интерфейсов на языке C#.
2. **Ответы на контрольные вопросы**
   1. Для чего используется интерфейс IComparable?

Ответ: Интерфейс IComparable предназначен для сравнения объектов. Он определяет единственный метод CompareTo, который принимает объект того же типа и возвращает:

* 1. Для чего используется интерфейс IEquatable?

Ответ: Интерфейс IEquatable предназначен для проверки равенства объектов. Он определяет единственный метод Equals, который принимает объект того же типа и возвращает true, если объекты равны, и false, если нет.

IEquatable используется, например, при проверке на наличие объекта в коллекции.

* 1. В чем отличие между обобщенным и необобщенным интерфейсами?

Ответ:

Необобщенный интерфейс:

Не использует дженерики (<T>).

Работает с любым типом данных.

Применяется к объектам, которые не должны иметь конкретный тип данных.

Обобщенный интерфейс:

Использует дженерики (<T>).

Работает с определенным типом данных, который указывается при реализации.

Применяется к объектам, которые должны иметь конкретный тип данных.

1. **Вывод**
   1. В ходе проделанной лабораторной работы я изучил процесс реализации стандартных интерфейсов на языке C#.

# Лабораторная работа №11

**Коллекции. Параметризованные классы**

1. **Цель работы**
   1. Изучить процесс создания и применения параметризованных классов коллекций (обобщенных списков и словарей) на языке C#.
2. **Ответы на контрольные вопросы**
   1. Коллекция - это структура данных, позволяющая хранить и управлять объединенными данными.
   2. В пространстве имен System.Collections.Generic описаны классы для работы с обобщенными коллекциями, такие как List, Dictionary<TKey, TValue>, Queue, Stack и другие.
   3. List - это обобщенный класс, представляющий список объектов одного типа T.
   4. К элементу списка можно обратиться по индексу или использовать методы Add, Remove, Find и другие.
   5. К элементу списка можно обратиться по индексу или использовать методы Add, Remove, Find и другие.
   6. К элементу словаря можно обратиться по ключу с помощью метода TryGetValue или индексатора.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс создания и применения параметризованных классов коллекций (обобщенных списков и словарей) на языке C#.

# Лабораторная работа №12

**Работа с типом данных структура**

1. **Цель работы**
   1. Изучить процесс создания и применения структур на языке C#.
2. **Ответы на контрольные вопросы**
   1. Типы значения хранятся в стеке.
   2. Структура – это пользовательский тип данных в C#, который позволяет объединять несколько разнородных переменных под одним именем.
   3. Основное отличие между структурой и классом заключается в том, что структуры являются значениями типа значения, в то время как классы являются ссылочными типами.
   4. Перечисление - это тип данных в C#, который позволяет создать набор именованных константных значений.
   5. Перечисления используются для создания набора именованных константных значений, чтобы упростить работу с данными и уменьшить вероятность ошибок.
   6. Общая форма объявления перечисления в C# представлена на рисунке 4.

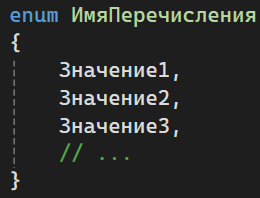


Рисунок 4 - Общая форма объявления перечисления в C#

1. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс создания и применения структур на языке C#.

# Лабораторная работа №13

**Обработка и форматирование строк**

1. **Цель работы**
   1. Изучить процесс обработки строк на языке C#;
   2. Научиться применять стандартные методы классов String, StringBuilder и Char для обработки строковых и символьных данных в программах на языке C#.
2. **Ответы на Ответы на контрольные вопросы**
   1. Переменные типа string относятся к ссылочным типам.
   2. Над строковыми данными допустимы операции конкатенации (сложения строк), сравнения, разделения на подстроки и другие операции со строками.
   3. Переменная типа string является неизменяемой последовательностью символов, в то время как массив символов представляет собой изменяемую последовательность символов.
   4. Интерполяция строк - это способ вставки значений переменных в строку с использованием символа $ и выражения в фигурных скобках.
   5. Методы класса String в C# не изменяют исходную строку, так как строки являются неизменяемыми.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс обработки строк на языке C#;
   2. В ходе проделанной лабораторной работы было сформировано умение применять стандартные методы классов String, StringBuilder и Char для обработки строковых и символьных данных в программах на языке C#.

# Лабораторная работа №14

**Использование регулярных выражений**

1. **Цель работы**
   1. Научиться составлять шаблоны регулярных выражений в программах на C#;
   2. Научиться применять регулярные выражения для поиска и замены подстрок в программах на языке C#.
2. **Ответы на контрольные вопросы**
   1. Регулярное выражение – это шаблон, описывающий набор возможных строк, которые соответствуют определенным правилам.
   2. Регулярные выражения используются для поиска, сопоставления и замены подстрок в тексте с использованием сложных шаблонов.
   3. Класс Regex в C# используется для работы с регулярными выражениями, включая компиляцию, поиск, сопоставление и замену подстрок.
   4. Для поиска подстроки при помощи регулярного выражения необходимо использовать методы класса Regex, такие как Regex.IsMatch или Regex.Match.
   5. Для замены подстроки при помощи регулярного выражения можно использовать методы класса Regex, такие как Regex.Replace.
   6. Escape-символы в регулярных выражениях используются для указания специального символа, который обычно имеет необычное значение, как обычный символ.
   7. Классы символов в регулярных выражениях используются для того, чтобы указывать определенные типы символов, например, цифры (\d), буквы (\w) и т.д.
   8. Квантификаторы в регулярных выражениях используются для указания количества повторений символов или групп символов.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение составлять шаблоны регулярных выражений в программах на C#;
   2. В ходе проделанной лабораторной работы было сформировано умение применять регулярные выражения для поиска и замены подстрок в программах на языке C#.

# Лабораторная работа №15

**Разработка делегатов**

1. **Цель работы**
   1. Научиться разрабатывать и применять делегаты на C#
2. **Ответы на контрольные вопросы**
   1. В C# делегат - это тип данных, который представляет ссылку на метод.
   2. Делегат на C# объявляется с помощью ключевого слова delegate, указывается тип возвращаемого значения и параметры метода.
   3. Метод можно вызвать через делегат с помощью оператора () или метода Invoke().
   4. В C# имеется несколько встроенных делегатов, таких как Action, Func, Predicate, которые используются для работы с методами и обратными вызовами.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение разрабатывать и применять делегаты на C#

# Лабораторная работа №16

**Разработка событий**

1. **Цель работы**
   1. Научиться создавать, вызывать и обрабатывать события на C#.
2. **Ответы на контрольные вопросы**
   1. Событие в C# - это механизм, который позволяет классам уведомлять другие классы о том, что произошло какое-то действие.
   2. Событие на C# объявляется с помощью ключевого слова event.
   3. Для создания обработчика события необходимо объявить метод с нужной сигнатурой и подписать его на событие с помощью оператора +=.
   4. Класс EventHandler является родительским для всех классов данных события.
   5. Стандартные классы делегатов для создания событий в C# - это EventHandler и EventHandler.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение создавать, вызывать и обрабатывать события на C#.

# Лабораторная работа №17

**Операции со списками**

1. **Цель работы**
   1. Изучить процесс обработки списков стандартными методами на языке C#.
2. **Ответы на контрольные вопросы**
   1. LINQ (Language Integrated Query) – это набор технологий, встроенных в язык программирования C#, которые позволяют писать запросы SQL-подобного стиля для выборки данных из различных источников (например, коллекций объектов, баз данных, XML-файлов), не выходя из языка программирования.
   2. Анонимный тип - это способ создания объекта с некоторым набором свойств без явного определения типа. Эти типы часто используются в LINQ для формирования результатов запроса, которые не соответствуют ни одному из предопределенных типов данных.
   3. Ключевое слово var используется для неявного определения типа переменной. Компилятор сам выводит тип на основе присваиваемого значения.
   4. Метод Select используется в LINQ для проекции каждого элемента источника данных (например, коллекции) в новую форму.
   5. Методы OrderBy и OrderByDescending используются для упорядочивания элементов последовательности в порядке возрастания или убывания соответственно.
   6. Методы ThenBy и ThenByDescending используются для дополнительной сортировки элементов последовательности, которые уже были упорядочены с помощью OrderBy или OrderByDescending, по вторичному критерию.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс обработки списков стандартными методами на языке C#.

# Лабораторная работа №18

**Проектирование диаграммы классов**

1. **Цель работы**
   1. Изучить процесс описания типов данных с применением диаграммы классов.
2. **Ответы на контрольные вопросы**
   1. Диаграмма классов используется для визуализации структуры системы, показывая классы системы, их атрибуты, методы и взаимосвязи между классами.
   2. Описание типа в диаграмме классов обычно состоит из трех частей: имени класса, атрибутов (или полей) и методов (или операций).
   3. Взаимосвязи в диаграмме классов делятся на ассоциации, агрегации, композиции, наследование и реализацию интерфейсов.
   4. Наследование означает, что класс наследует свойства и поведение от другого класса. Реализация означает, что класс должен реализовать все методы, определенные в интерфейсе.
   5. Агрегация и композиция - это типы ассоциаций, которые показывают отношения "часть-целое". Отличие в том, что композиция предполагает более сильную связь, где части не могут существовать независимо от целого.
   6. Ассоциация показывает, что объекты одного класса связаны с объектами другого класса.
   7. В диаграмме классов мощность отношений указывается числами или символами, показывающими, сколько экземпляров одного класса могут быть связаны с экземплярами другого класса. Например, "1" означает один-к-одному, "0..1" - ноль или один, "\*" или "n" - много.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс описания типов данных с применением диаграммы классов.

# Лабораторная работа №19

**Использование шаблонов проектирования**

1. **Цель работы**
   1. Научиться применять паттерны проектирования в разработке программ.
2. **Ответы на контрольные вопросы**
   1. Порождающие паттерны используются для создания объектов таким образом, чтобы они были разделены от систем, в которых они используются. Это увеличивает гибкость и повторное использование кода.
   2. К порождающим паттернам относятся:
      1. Абстрактная фабрика (Abstract Factory),
      2. Строитель (Builder),
      3. Фабричный метод (Factory Method),
      4. Прототип (Prototype),
      5. Одиночка (Singleton).
   3. Структурные паттерны используются для организации классов и объектов так, чтобы они формировали более крупные структуры, сохраняя при этом гибкость и эффективность.
   4. К структурным паттернам относятся:
      1. Адаптер (Adapter),
      2. Мост (Bridge),
      3. Компоновщик (Composite),
      4. Декоратор (Decorator),
      5. Фасад (Facade),
      6. Легковес (Flyweight),
      7. Заместитель (Proxy).
   5. Поведенческие паттерны используются для эффективной коммуникации и распределения обязанностей между объектами.
   6. К поведенческим паттернам относятся:
      1. Цепочка обязанностей (Chain of Responsibility),
      2. Команда (Command),
      3. Интерпретатор (Interpreter),
      4. Итератор (Iterator),
      5. Посредник (Mediator),
      6. Хранитель (Memento),
      7. Наблюдатель (Observer)Состояние (State),
      8. Стратегия (Strategy),
      9. Шаблонный метод (Template Method),
      10. Посетитель (Visitor).
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение применять паттерны проектирования в разработке программ.

# Лабораторная работа №20

**Проектирование API**

1. **Цель работы**
   1. Научиться проектировать API.
2. **Ответы на контрольные вопросы**
   1. REST (Representational State Transfer) - это архитектурный стиль разработки сетевых программ, который используется для создания веб-сервисов. REST позволяет запросить данные или функциональность от других программ через простые HTTP запросы, делая его легким и гибким в использовании.
   2. Метод GET используется для запроса данных от сервера. Он может быть использован для получения страниц, изображений или любого другого типа ресурсов.
   3. Метод POST используется для отправки данных на сервер для создания нового ресурса. Обычно используется в веб-формах и API для передачи пользовательских данных.
   4. Метод PUT используется для обновления существующего ресурса на сервере. Он предполагает полную замену ресурса.
   5. Метод DELETE используется для удаления существующего ресурса на сервере.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение проектировать API.

# Лабораторная работа №21

**Оптимизация кода**

1. **Цель работы**
   1. Изучить методы оптимизации программного кода.
2. **Ответы на контрольные вопросы**
   1. Оптимизация программного кода — это процесс изменения программы для улучшения её эффективности, например, для ускорения выполнения или уменьшения занимаемого объёма памяти.
   2. Цель оптимизации программного кода — повышение производительности программы, сокращение времени её выполнения, уменьшение потребления ресурсов (таких как память и пропускная способность сети) и улучшение общей эффективности.
   3. Методы оптимизации программного кода включают:
      1. Устранение лишних вычислений, таких как избыточные циклы или ненужные временные переменные;
      2. Использование более эффективных алгоритмов и структур данных;
      3. Минимизация обращений к памяти, например, путём улучшения локальности данных;
      4. Параллелизм, то есть выполнение нескольких операций одновременно;
      5. Компиляция с оптимизацией, используя специальные флаги или настройки компилятора.
3. **Вывод**
   1. В ходе проделанной лабораторной работы были изучены методы оптимизации программного кода.

# Лабораторная работа №22

**Асинхронное программирование**

1. **Цель работы**
   1. Научиться реализовывать и запускать асинхронные операции на C#;
   2. Научиться выполнять вычисления, используя асинхронные операции;
   3. Научиться выполнять ввод и вывод данных, используя асинхронные операции.
2. **Ответы на контрольные вопросы**
   1. В C# для работы с асинхронными вызовами используются ключевые слова async и await.
   2. Типы возврата асинхронных методов в C# могут быть:
      1. Task: используется, когда метод выполняет операцию асинхронно, но не возвращает значение.
      2. Task<T>: используется, когда метод возвращает значение типа T асинхронно.
      3. ValueTask или ValueTask<T>: используются для оптимизации производительности в сценариях, где результат может быть доступен синхронно или когда асинхронные операции выполняются нечасто.
   3. Чтобы вызвать метод в асинхронном режиме, используйте ключевое слово await перед вызовом метода, который возвращает Task или Task<T>.
   4. Чтобы указать, что в методе могут быть асинхронные вызовы, добавьте ключевое слово async перед типом возвращаемого значения метода.
   5. Для обработки исключений, возникших в асинхронных вызовах, используйте стандартный блок try-catch. Асинхронные исключения можно перехватывать и обрабатывать в блоке catch так же, как и синхронные исключения.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение реализовывать и запускать асинхронные операции на C#;
   2. В ходе проделанной лабораторной работы было сформировано умение выполнять вычисления, используя асинхронные операции;
   3. В ходе проделанной лабораторной работы было сформировано умение выполнять ввод и вывод данных, используя асинхронные операции.

# Лабораторная работа №23

**Документирование кода**

1. **Цель работы**
   1. Изучить процесс документирования программного кода.
2. **Ответы на контрольные вопросы**
   1. XML-документация – это стандартизированный способ документирования кода в .NET, который позволяет разработчикам добавлять описания к классам, методам и другим элементам кода непосредственно в исходные файлы. Эти комментарии могут быть автоматически извлечены и преобразованы в техническую документацию.
   2. Чтобы сгенерировать XML-комментарий, используйте символы // перед объявлением элемента кода. Visual Studio автоматически добавит шаблон комментария, который можно заполнить.
   3. Для того чтобы XML-документация была видна при подключении библиотеки, необходимо:
   4. Включить генерацию XML-файла документации в настройках проекта.
   5. Распространять сгенерированный XML-файл вместе с сборкой (.dll файлом).
   6. В разделе summary пишется краткое описание того, что делает класс, метод или другой элемент кода.
   7. В разделе param описываются параметры метода, их типы и назначение.
   8. В разделе returns описывается значение, которое возвращает метод.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс документирования программного кода.

# Лабораторная работа №24

**Рефакторинг кода**

1. **Цель работы**
   1. Изучить техники рефакторинга программного кода.
2. **Ответы на контрольные вопросы**
   1. Рефакторинг – это процесс изменения внутренней структуры программного обеспечения для улучшения его читаемости и упрощения его расширения, не влияя на внешнее поведение программы.
   2. Группы техник рефакторинга включают:
      1. Переименование (Rename): изменение имен переменных, методов, классов для улучшения понимания кода;
      2. Извлечение (Extract): создание новых методов, классов или интерфейсов из существующего кода;
      3. Встраивание (Inline): замена вызова метода его содержимым;
      4. Перемещение (Move): перенос методов и классов в другие классы или пакеты;
      5. Изменение сигнатуры (Change Signature): изменение списка параметров метода.
   3. Чтобы выполнить рефакторинг в Visual Studio нужно выполнить следующие действия:
      1. Выделить элемент кода, который нужно отрефакторить.
      2. Нажать правую кнопку мыши и выбрать нужную операцию рефакторинга из контекстного меню.
      3. Использовать меню "Refactor" или сочетания клавиш для доступа к инструментам рефакторинга.
3. **Вывод**
   1. В ходе проделанной лабораторной работы были изучены техники рефакторинга программного кода.

# Лабораторная работа №25

**Работа с системой контроля версий**

1. **Цель работы**
   1. Научиться применять систему контроля версий git в процессе разработки программного обеспечения.
2. **Ответы на контрольные вопросы**
   1. Репозиторий – место, где хранятся и поддерживаются какие-либо данные.
   2. В файле readme.md указывается название проекта и его описание.
   3. Файлы .gitignore содержат шаблоны, которые сопоставляются с именами. файлов в репозитории для определения необходимости игнорировать эти файлы.
   4. Для управления git-репозиторием применяется такое программное обеспечение, как GitHub Desktop.
   5. Репозиторий может быть локальным, то есть распологаться непосредственно в памяти компьютера разработчика, а может быть удаленным - находиться на сервере.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение применять систему контроля версий git в процессе разработки программного обеспечения.

# Лабораторная работа №26

**Разработка интерфейса пользователя: компоновка элементов**

1. **Цель работы**
   1. Изучить элементы-контейнеры, применяющиеся в приложениях WPF для компоновки.
2. **Ответы на контрольные вопросы**
   1. В WPF компоновка относится к способу организации элементов управления на пользовательском интерфейсе. Компоновка включает в себя размещение элементов в определенных позициях на экране, управление их размерами, расположение и взаимосвязи между элементами.
   2. Класс Panel является родительским для всех элементов-контейнеров в WPF.
   3. Для выравнивания элементов внутри контейнера по высоте и ширине можно использовать свойства HorizontalAlignment и VerticalAlignment, а также HorizontalContentAlignment и VerticalContentAlignment.
   4. Особенность компоновки с использованием Grid в том, что он позволяет разбивать контейнер на строки и столбцы, определяя расположение элементов с помощью сетки, а также он может создавать сложные макеты с использованием расположения в виде таблицы. Canvas в свою очередь позволяет свободно располагать элементы на плоскости контейнера, и через него можно задавать точное позиционирование и размеры элементов.
   5. StackPanel располагает элементы в одну строку или столбец в соответствии с заданным направлением, DockPanel позволяет привязывать элементы к краям контейнера, а WrapPanel автоматически переносит элементы на новую строку или столбец, если они не умещаются в текущей строке или столбце.
   6. Размеры элементов в приложениях WPF могут задаваться в единицах измерения, таких как пиксели (px), относительные величины (например, проценты), а также в единицах измерения WPF, таких как дюймы (in), пункты (pt) и расширения (зависят от разрешения экрана).
3. **Вывод**
   1. В ходе проделанной лабораторной работы были изучены элементы-контейнеры, применяющиеся в приложениях WPF для компоновки.

# Лабораторная работа №27

**Организация интерфейса пользователя**

1. **Цель работы**
   1. Изучить процесс настройки интерфейса и организации переходов в приложениях WPF.
2. **Ответы на контрольные вопросы**
   1. Элемент управления Frame используется для загрузки и отображения различных страниц в приложении WPF.
   2. Элемент управления Page используется для создания отдельных страниц с уникальным содержимым, которые могут быть загружены в элемент управления Frame.
   3. Для того чтобы перейти к определенной странице, используя фрейм, необходимо установить свойство Source фрейма на Uri страницы, к которой нужно перейти.
   4. Для проверки возможности вернуться к предыдущей странице из фрейма, можно использовать метод CanGoBack, который возвращает булевое значение - True, если есть возможность вернуться назад, и False, если не можем вернуться.
   5. Перейти к предыдущей странице, используя фрейм, можно с помощью метода GoBack().
   6. Для сгруппирования содержимого в WPF можно использовать элементы управления, такие как Panel (StackPanel, Grid, WrapPanel) и контейнеры (GroupBox, TabControl), которые позволяют организовать размещение и расположение других элементов внутри себя.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс настройки интерфейса и организации переходов в приложениях WPF.

# Лабораторная работа №28

**Разработка интерфейса пользователя: настройка стилей**

1. **Цель работы**
   1. Изучить процесс настройки интерфейса с использованием стилей в приложениях WPF
2. **Ответы на контрольные вопросы**
   1. В приложениях WPF стили используются для задания внешнего вида элементов управления, чтобы обеспечить единообразие и упростить обновление внешнего вида приложения.
   2. Общая форма локального определения стиля элемента управления представлена на рисунке 5.

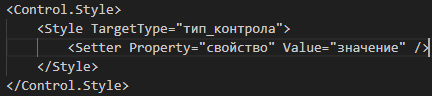


Рисунок 5 - Общая форма локального определения стиля элемента управления

* 1. Общая форма определения стиля приложения изображена на рисунке 6.

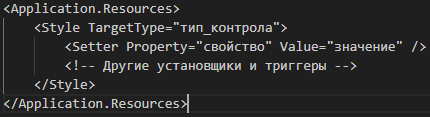


Рисунок 6 - Общая форма определения стиля приложения

* 1. Чтобы явно применить стиль, установите свойство Style элемента управления на ресурс стиля, используя ключ стиля.
  2. Для того, чтобы наследовать стиль можно использовать свойство BasedOn. Пример применения данного свойства показан на рисунке 7.



Рисунок 7 - Пример применения свойства BasedOn

* 1. Чтобы добавить новую тему в приложение нужно создать Resource Dictionary с определениями стилей и добавить тегов Application.Resources или Control.Resources.
  2. Для переключения тем во время выполнения можно динамически загружать и применять Resource Dictionaries, содержащие определения стилей темы.

1. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс настройки интерфейса с использованием стилей в приложениях WPF.

# Лабораторная работа №29

**Разработка интерфейса пользователя: применение триггеров**

1. **Цель работы**
   1. Изучить процесс применения триггеров в приложениях WPF;
   2. Закрепить навык применения стилей в приложениях на WPF.
2. **Ответы на контрольные вопросы**
   1. Триггеры в WPF позволяют автоматически изменять свойства элементов управления в ответ на определенные условия или события.
   2. В WPF можно разработать такие виды триггеров, как триггеры свойств (Property Triggers), триггеры данных (Data Triggers), триггеры событий (Event Triggers), мультитриггеры (MultiTriggers), триггеры валидации (Validation Triggers).
   3. Триггер свойств используется для автоматического изменения одного или нескольких свойств элемента управления, когда значение другого свойства изменяется. Он срабатывает, когда свойство достигает определенного значения.
   4. Триггер данных используется для изменения внешнего вида элемента управления в зависимости от значения свойства, связанного с данными. Он срабатывает, когда связанное с данными свойство соответствует определенному условию.
   5. Триггер данных используется для изменения внешнего вида элемента управления в зависимости от значения свойства, связанного с данными. Он срабатывает, когда связанное с данными свойство соответствует определенному условию.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс применения триггеров в приложениях WPF
   2. В ходе проделанной лабораторной работы был закреплен навык применения стилей в приложениях на WPF.

# Лабораторная работа №30

**Изучение особенностей элементов выбора в приложениях WPF**

1. **Цель работы**
   1. Изучить свойства и процесс обработки событий элементов выбора в приложениях WPF
2. **Ответы на контрольные вопросы**
   1. CheckBox — это элемент управления в WPF, который позволяет пользователю сделать выбор, обычно да/нет. Он используется для представления параметров, которые можно включать и выключать независимо друг от друга.
   2. RadioButton — это элемент управления, позволяющий пользователю выбрать один из нескольких вариантов. Радиокнопки обычно используются в группе, чтобы обеспечить взаимоисключающий выбор.
   3. Чтобы проверить, выбран ли CheckBox или RadioButton, можно использовать свойство IsChecked.
   4. Событие, которое срабатывает при выборе CheckBox или RadioButton, называется Checked.
   5. Событие, которое срабатывает при снятии выбора с CheckBox или RadioButton, называется Unchecked.
   6. CheckBox может принимать три значения: true (выбран), false (не выбран) и null (неопределенное состояние). RadioButton может принимать только два значения: true (выбран) и false (не выбран).
3. **Вывод**
   1. В ходе проделанной лабораторной работы были изучены свойства и процесс обработки событий элементов выбора в приложениях WPF.

# Лабораторная работа №31

**Разработка приложения с использованием текстовых компонентов**

1. **Цель работы**
   1. Изучить различные типы полей ввода, применяющихся в приложениях WPF;
   2. Изучить свойства полей ввода и процесс обработки событий полей ввода.
2. **Ответы на контрольные вопросы**
   1. Задать имя элементам управления можно с помощью атрибута x:Name
   2. Создать обработчик события можно, указав имя события и обработчика в XAML и затем определив обработчик в коде C#. Пример создания обработчика событий показан на рисунках 8 и 9.



Рисунок 8 - Пример создания обработчика событий в XAML

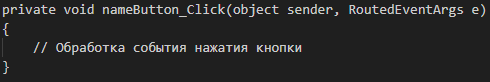


Рисунок 9 - Пример создания обработчика событий в коде C#

* 1. Slider используется для предоставления пользователю графического интерфейса для выбора значения из диапазона.
  2. TextBox предназначен для ввода и редактирования однострочного текста пользователем.
  3. TextBlock используется для отображения однострочного или многострочного текста, который не требует редактирования пользователем.
  4. Calendar предоставляет пользователю интерфейс для выбора даты из календаря.
  5. DatePicker позволяет пользователю вводить или выбирать дату через всплывающий календарь.
  6. PasswordBox используется для безопасного ввода паролей, скрывая введенные символы.

1. **Вывод**
   1. В ходе проделанной лабораторной работы были изучены различные типы полей ввода, применяющихся в приложениях WPF;
   2. В ходе проделанной лабораторной работы были изучены свойства полей ввода и процесс обработки событий полей ввода.

# Лабораторная работа №32

**Разработка приложения с использованием элементов отображения списков**

1. **Цель работы**
   1. Изучить свойства и процесс обработки событий элементов отображения списков в приложениях WPF.
2. **Ответы на контрольные вопросы**
   1. ComboBox — это элемент управления, который сочетает в себе текстовое поле и выпадающий список, позволяя пользователю выбрать значение из списка или ввести своё. Он используется для экономии места на форме, когда нужно представить множество вариантов выбора.
   2. ListBox — это элемент управления, который отображает список элементов, из которых пользователь может выбрать один или несколько. Он используется, когда необходимо предоставить доступ к коллекции элементов для выбора.
   3. При выборе элемента в селекторе, таком как ComboBox или ListBox, срабатывает событие SelectionChanged.
   4. Элементы селекторов хранятся в свойстве Items.
   5. В селекторе могут быть элементы различных типов, включая строки, числа, объекты данных и даже другие элементы управления WPF.
   6. Свойство ItemsSource позволяет привязать селектор к набору данных, например, к коллекции объектов.
   7. Свойство DisplayMemberPath используется в селекторе для указания имени свойства объекта, значение которого будет отображаться в списке элементов управления.

1. **Вывод**
   1. В ходе проделанной лабораторной работы были изучены свойства и процесс обработки событий элементов отображения списков в приложениях WPF.

# Лабораторная работа №33

**Разработка приложения для отображения данных в табличном виде**

1. **Цель работы**
   1. Изучить свойства и процесс настройки внешнего вида элемента DataGrid в приложениях WPF.
2. **Ответы на контрольные вопросы**
   1. DataGrid — это гибкий элемент управления, который предоставляет интерфейс для отображения, сортировки, редактирования и анализа табличных данных. Он используется для представления данных в виде таблицы, где строки соответствуют записям данных, а столбцы — свойствам этих данных.
   2. В DataGrid поддерживаются различные типы столбцов, включая:
      1. DataGridTextColumn для отображения и редактирования текста.
      2. DataGridCheckBoxColumn для отображения и редактирования булевых значений.
      3. DataGridComboBoxColumn для выбора значения из списка.
      4. DataGridTemplateColumn для создания пользовательского содержимого.
   3. Добавить кнопку в строки DataGrid можно с помощью DataGridTemplateColumn, определив в нём шаблон для кнопки. Пример реализации кнопки в DataGrid показан на рисунке 10.

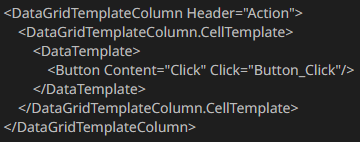


Рисунок 10 - Пример добавления кнопки в DataGrid

* 1. Указать источник данных для DataGrid можно с помощью свойства ItemsSource.
  2. Указать источник данных для выпадающего списка в DataGrid (например, в DataGridComboBoxColumn) можно также через ItemsSource.

1. **Вывод**
   1. В ходе проделанной лабораторной работы были изучены свойства и процесс настройки внешнего вида элемента DataGrid в приложениях WPF.

# Лабораторная работа №34

**Разработка приложения с меню и панелью инструментов**

1. **Цель работы**
   1. Изучить свойства и процесс настройки внешнего вида меню и панели инструментов в приложениях WPF.
2. **Ответы на контрольные вопросы**
   1. Menu - это элемент управления, который позволяет создавать меню с различными командами и подменю для управления приложением. Он используется для организации и предоставления пользователю доступа к функциям приложения.
   2. ContextMenu - это элемент управления, который показывает контекстное меню при правом клике на элемент интерфейса. Он используется для предоставления быстрого доступа к часто используемым командам или действиям в зависимости от контекста.
   3. ToolBar - это элемент управления, который позволяет создавать панели инструментов для размещения кнопок, команд и других элементов управления. Он используется для обеспечения быстрого доступа к функциям приложения и удобной работы с ними.
   4. StatusBar - это элемент управления, который позволяет отображать информацию о состоянии приложения или текущем действии. Он используется для отображения статусных сообщений, прогресса операций или другой информации, которая может быть полезна пользователю.
3. **Вывод**
   1. В ходе проделанной лабораторной работы были изучены свойства и процесс настройки внешнего вида меню и панели инструментов в приложениях WPF.

# Лабораторная работа №35

**Создание проекта с использованием компонентов стандартных диалогов**

1. **Цель работы**
   1. Изучить процесс создания и применения стандартных диалоговых окон в приложениях WPF.
2. **Ответы на контрольные вопросы**
   1. Диалоговое окно - это всплывающее окно, которое отображается пользователю для взаимодействия с программой или системой.
   2. OpenFileDialog используется для открытия файлов. Пользователь может выбрать файл из файловой системы, который затем можно открыть или использовать в программе.
   3. SaveFileDialog используется для сохранения файлов. Пользователю предоставляется возможность выбрать место сохранения файла и указать его имя.
   4. MessageBox - это диалоговое окно, которое используется для отображения информационных сообщений, предупреждений или ошибок пользователю.
   5. PrintDialog используется для настройки параметров печати, таких как выбор принтера, настройка количества копий, выбор страниц для печати и другие параметры печати. Пользователю предоставляется возможность настроить печать перед отправкой документа на печать.

1. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс создания и применения стандартных диалоговых окон в приложениях WPF.

# Лабораторная работа №36

**Разработка приложения с несколькими формами**

1. **Цель работы**
   1. Изучить процесс создания и применения пользовательских окон в приложениях WPF.
2. **Ответы на контрольные вопросы**
   1. Для того чтобы сделать данные доступными другим окнам, можно использовать свойство DialogResult. Это свойство позволяет установить результат выполнения диалогового окна и передать его обратно вызывающему окну.
   2. Переменная DialogResult может принимать значения true, false или null. Значение true указывает на успешное завершение диалога, false - на отмену операции, а null - если диалог был закрыт без выполнения каких-либо действий.
   3. Для открытия окна в диалоговом режиме можно использовать метод ShowDialog(), который блокирует вызывающее окно до того момента, пока диалоговое окно не будет закрыто.
   4. Для открытия окна в не диалоговом режиме можно использовать метод Show(), который открывает окно без блокировки вызывающего окна. В этом случае пользователь может взаимодействовать с несколькими окнами одновременно.
   5. В диалоговом режиме пользователь должен завершить работу с диалоговым окном, прежде чем вернуться к основному окну. В не диалоговом режиме пользователя можно взаимодействовать с несколькими окнами одновременно, не блокируя выполнение каких-либо операций.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс создания и применения пользовательских окон в приложениях WPF.

# Лабораторная работа №37

**Реализация фильтрации данных**

1. **Цель работы**
   1. Научиться применять LINQ-запросы для фильтрации данных по одному критерию и набору критериев.
2. **Ответы на контрольные вопросы**
   1. Метод Where используется для фильтрации элементов коллекции по определенному условию.
   2. При составлении условий могут применяться логические операторы && (логическое И), || (логическое ИЛИ), ! (логическое НЕ).
   3. Для выполнения регистронезависимого поиска можно использовать методы ToLower() или ToUpper() для приведения всех символов к нижнему или верхнему регистру.
   4. Для проверки, что строка начинается с определенного текста, можно использовать метод StartsWith().
   5. Для проверки, что строка содержит определенный текст, можно использовать метод Contains().
   6. Для фильтрации по нескольким критериям, указываемым пользователем, можно использовать условные конструкции в методе Where, а также комбинировать условия с помощью логических операторов.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение применять LINQ-запросы для фильтрации данных по одному критерию и набору критериев.

# Лабораторная работа №38

**Реализация постраничного вывода информации**

1. **Цель работы**
   1. Научиться применять LINQ-запросы для постраничного вывода данных.
2. **Ответы на контрольные вопросы**
   1. Вывод данных постранично может потребоваться для улучшения производительности системы, когда необходимо обработать большой объем данных. Пагинация позволяет разбить данные на отдельные страницы, что упрощает их отображение и позволяет пользователю выбирать нужную информацию.
   2. Пагинация - это разделение данных на страницы для более удобной навигации и отображения. К примеру, на сайтах поисковых систем или интернет-магазинах пагинация позволяет показывать результаты поиска или товары с определенным количеством на каждой странице.
   3. Метод Take используется для выбора указанного количества элементов из последовательности.
   4. Метод Skip используется для пропуска указанного количества элементов в последовательности.
   5. Метод TakeWhile используется для выбора элементов из последовательности, пока выполняется указанное условие.
   6. Метод SkipWhile используется для пропуска элементов в последовательности, пока выполняется указанное условие.
   7. Набор данных должен быть упорядоченным, чтобы можно было использовать методы Take и Skip. Например, при работе с базой данных можно отсортировать данные по какому-либо полю, чтобы затем правильно применить методы выбора и пропуска элементов.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение применять LINQ-запросы для постраничного вывода данных.

# Лабораторная работа №39

**Реализация группировки и соединения данных**

1. **Цель работы**
   1. Научиться применять LINQ-запросы для группировки и соединения данных.
2. **Ответы на контрольные вопросы**
   1. В LINQ поддерживаются следующие агрегатные функции:
      1. Count: возвращает количество элементов в коллекции.
      2. Sum: вычисляет сумму значений элементов.
      3. Average: вычисляет среднее значение элементов.
      4. Max: находит максимальное значение.
      5. Min: находит минимальное значение.
      6. Aggregate: применяет предоставленную функцию к элементам коллекции, выполняя итерацию по элементам.
   2. Метод Distinct возвращает новый набор, содержащий уникальные элементы из исходной коллекции, удаляя дубликаты.
   3. Метод GroupBy используется для группировки элементов коллекции на основе одного или нескольких ключей. Результатом является коллекция групп, каждая из которых содержит элементы с одинаковыми ключами.
   4. Метод Join используется для соединения двух коллекций на основе соответствия ключей. Он сопоставляет элементы из двух коллекций на основе ключевых выражений и объединяет их в один результирующий элемент.
   5. Метод GroupJoin используется для создания кортежей, где каждый кортеж содержит группу элементов из одной коллекции, соответствующих одному элементу из другой коллекции. Этот метод часто используется для реализации "левых соединений" в SQL.
   6. Различия между результатами, полученными при вызове LINQ-методов Concat, Union, Except, Intersect:
      1. Concat: объединяет две коллекции в одну, без удаления дубликатов.
      2. Union: объединяет две коллекции в одну, удаляя дубликаты.
      3. Except: возвращает элементы первой коллекции, которых нет во второй коллекции.
      4. Intersect: возвращает элементы, которые присутствуют в обеих коллекциях.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение применять LINQ-запросы для группировки и соединения данных.

# Лабораторная работа №40

**Разработка приложения для работы с графикой**

1. **Цель работы**
   1. Изучить процесс рисования и трансформации объектов в приложениях WPF.
2. **Ответы на контрольные вопросы**
   1. В приложениях WPF доступны следующие графические примитивы:
      1. Line - прямая линия,
      2. Rectangle – прямоугольник,
      3. Ellipse – эллипс,
      4. Polygon – многоугольник,
      5. Polyline - ломаная линия,
      6. Path - произвольная фигура, состоящая из линий и кривых,
      7. TextBlock - текстовое поле.
   2. Path в WPF используется для создания произвольных фигур, состоящих из линий и кривых. С помощью Path можно создавать сложные графические объекты, такие как ломаные линии, кривые Безье, дуги и другие.
   3. В приложениях WPF доступны следующие виды трансформаций объектов:
      1. ScaleTransform – масштабирование,
      2. RotateTransform – вращение,
      3. TranslateTransform – перемещение,
      4. SkewTransform – наклон,
      5. MatrixTransform - произвольная трансформация с помощью матрицы.
   4. Для указания заливки и контура графических объектов в WPF используются свойства Fill и Stroke соответственно. Fill задает цвет или изображение, которое будет использоваться для заливки объекта, а Stroke - цвет и толщину линии, которая будет отображаться вокруг объекта.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс рисования и трансформации объектов в приложениях WPF.

# Лабораторная работа №41

**Разработка приложения с анимацией**

1. **Цель работы**
   1. Изучить процесс анимации объектов в приложениях WPF.
2. **Ответы на контрольные вопросы**
   1. Для задания анимации размеров в приложениях WPF используется класс DoubleAnimation.
   2. Для задания анимации цвета в приложениях WPF используется класс ColorAnimation.
   3. Покадровая анимация (KeyFrame Animation) в WPF позволяет управлять анимацией на основе ключевых кадров, где каждый ключевой кадр определяет состояние анимации в определенный момент времени. Плавная анимация (Non-KeyFrame Animation) изменяет значение свойства от начального до конечного значения с определенной скоростью.
   4. В WPF есть несколько свойств, которые позволяют управлять анимацией:
      1. From, To, By: задают начальное, конечное и промежуточное значения анимации,
      2. Duration: определяет время, за которое анимация должна завершиться,
      3. AutoReverse: если установлено в true, анимация будет возвращаться к начальному значению после завершения,
      4. RepeatBehavior: определяет, сколько раз анимация должна повториться,
      5. SpeedRatio: изменяет скорость анимации относительно ее длительности,
      6. FillBehavior: определяет, как анимация будет вести себя после завершения (установится в конечное значение или останется в текущем состоянии).
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс анимации объектов в приложениях WPF.

# Лабораторная работа №42

**Разработка мультимедиа-приложения**

1. **Цель работы**
   1. Изучить процесс создания мультимедиа-приложений на WPF.
2. **Ответы на контрольные вопросы**
   1. Мультимедиа - это содержание, которое использует комбинацию различных форматов, таких как изображения, аудио и видео, для передачи информации.
   2. Для отображения изображений в приложениях WPF можно использовать элементы Image и ImageBrush.
   3. Для воспроизведения аудио в приложениях WPF можно использовать элементы MediaElement и MediaPlayer.
   4. Для воспроизведения видео в приложениях WPF также можно использовать элементы MediaElement и MediaPlayer.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс создания мультимедиа-приложений на WPF.

# Лабораторная работа №43

**Разработка игрового приложения**

1. **Цель работы**
   1. Изучить процесс разработки игровых приложений WPF, использующих графику, обработчики событий и таймеры.
2. **Ответы на контрольные вопросы**
   1. Для подключения таймера к приложению на WPF, можно использовать класс DispatcherTimer. Необходимо создать экземпляр этого класса и указать интервал, с которым будет срабатывать таймер.
   2. Для создания обработчика события таймера в приложении WPF, необходимо определить метод, который будет вызываться при каждом срабатывании таймера. Код для создания данного обработчика показан на рисунке 11.

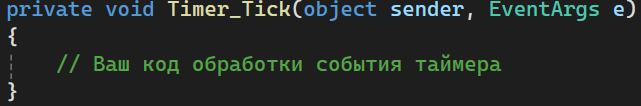


Рисунок 11 - Пример создания обработчика события для таймера

* 1. Для изменения интервала таймера в приложении WPF, можно просто изменить значение свойства Interval у объекта DispatcherTimer.

1. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс разработки игровых приложений WPF, использующих графику, обработчики событий и таймеры.

# Лабораторная работа №44

**Создание БД**

1. **Цель работы**
   1. Изучить процесс создания таблиц и связей между ними в реляционной СУБД.
2. **Ответы на контрольные вопросы**
   1. Система управления базами данных (СУБД) - это программное обеспечение, которое позволяет создавать, управлять, обновлять и анализировать базы данных.
   2. База данных - это организованный набор данных, которые хранятся в компьютере и могут быть легко доступны и управляемы через систему управления базами данных.
   3. Примерами реляционных СУБД являются MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, SQLite.
3. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс создания таблиц и связей между ними в реляционной СУБД.

# Лабораторная работа №45

**Создание приложения с БД для чтения данных**

1. **Цель работы**
   1. Научиться создавать приложения для чтения данных из БД.
2. **Ответы на контрольные вопросы**
   1. SqlConnection используется для установления соединения с базой данных MS SQL Server.
   2. Для чтения одного значения из базы данных можно использовать SqlCommand с методом ExecuteScalar(), который возвращает первую колонку первой строки в результирующем наборе.
   3. SqlDataReader используется для чтения данных из базы данных в виде потока данных, и предоставляет только чтение-вперед для выполнения быстрых операций.
   4. Для получения данных из БД с использованием SqlDataAdapter необходимо создать объект SqlDataAdapter, передать SQL запрос и объект SqlConnection, и заполнить DataSet или DataTable с помощью метода Fill().
   5. Для реализации подключения к СУБД MS SQL Server необходимо подключить пространства имен System.Data и System.Data.SqlClient.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение создавать приложения для чтения данных из БД.

# Лабораторная работа №46

**Создание приложения с БД для записи данных**

1. **Цель работы**
   1. Научиться создавать приложения для записи данных в БД.
2. **Ответы на контрольные вопросы**
   1. Компонент SqlDataAdapter используется для выполнения операций чтения данных из базы данных и их обновления, вставки и удаления данных в базе данных. Он является промежуточным звеном между набором данных и источником данных.
   2. Компонент SqlCommandBuilder используется для создания команд SQL автоматически на основе существующего запроса. Он позволяет генерировать команды для обновления, вставки и удаления данных без необходимости написания SQL запросов вручную.
   3. Чтобы изменить данные в базе данных с использованием SqlCommand, необходимо создать экземпляр объекта SqlCommand, указать SQL запрос для обновления данных и выполнить этот запрос с помощью метода ExecuteNonQuery.
   4. Чтобы изменить данные в базе данных с использованием SqlDataAdapter, необходимо связать SqlDataAdapter с объектом данных (например, DataTable), выполнить операции по чтению данных (Fill), вставке, обновлению и удалению данных, а затем применить изменения обратно в базу данных с помощью метода Update.
   5. Чтобы связать SqlCommandBuilder и SqlDataAdapter, необходимо использовать метод GetUpdateCommand объекта SqlCommandBuilder для получения команды обновления, после чего установить это значение в свойство UpdateCommand объекта SqlDataAdapter. Таким образом, SqlDataAdapter будет использовать команды, созданные SqlCommandBuilder, для синхронизации изменений с базой данных.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение создавать приложения для записи данных в БД.

# Лабораторная работа №47

**Создание запросов к БД**

1. **Цель работы**
   1. Научиться выполнять запросы к БД из клиентского приложения;
   2. Научиться передавать параметры в запросы.
2. **Ответы на контрольные вопросы**
   1. Параметры используются в командах для передачи значений в SQL запросы или хранения данных при вызове методов.
   2. Чтобы добавить параметр в команду с использованием метода Add, можно создать новый экземпляр параметра и добавить его в коллекцию параметров вашей команды. Пример кода представлен на рисунке 12.



Рисунок 12 - Пример добавления параметра в команду с использованием метода Add

* 1. Чтобы добавить параметр в команду с использованием метода AddWithValue, можно добавить параметр с указанием его имени и значения в одной строке. Пример кода показан на рисунке 13.



Рисунок 13 - Пример добавления параметра с помощью метода AddWithValue

* 1. Свойства параметра могут включать, но не ограничиваться следующими:
     1. ParameterName - имя параметра,
     2. Value - значение параметра,
     3. Direction - направление передачи параметра (Input, Output, InputOutput, ReturnValue),
     4. SqlDbType - тип данных SQL для параметра,
     5. Size - размер параметра,
     6. Precision - точность параметра,
     7. Scale - масштаб параметра,
     8. IsNullable - указывает, допускается ли параметр со значением NULL.

1. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение выполнять запросы к БД из клиентского приложения;
   2. В ходе проделанной лабораторной работы было сформировано умение передавать параметры в запросы.

# Лабораторная работа №48

# Создание хранимых процедур

1. **Цель работы**
   1. Научиться создавать и использовать хранимые процедуры в MS SQL Server;
   2. Закрепить навык создания запросов на выборку и модификацию данных в MS SQL Server.
2. **Ответы на контрольные вопросы**
   1. Хранимые процедуры – это наборы SQL-команд, хранящиеся на сервере базы данных и предназначенные для выполнения определенных задач. Они используются для повышения производительности, обеспечения целостности данных и упрощения управления данными. Хранимые процедуры могут содержать сложные логические конструкции, такие как условия, циклы и обработку ошибок, что делает их мощным инструментом для работы с базами данных.
   2. Для вызова выполнения хранимой процедуры в клиентском приложении, необходимо использовать соответствующий язык программирования или библиотеку, которая поддерживает взаимодействие с базой данных. Например, в приложении на языке C# можно использовать библиотеку ADO.NET. Вызов хранимой процедуры обычно представлен на рисунке 14.

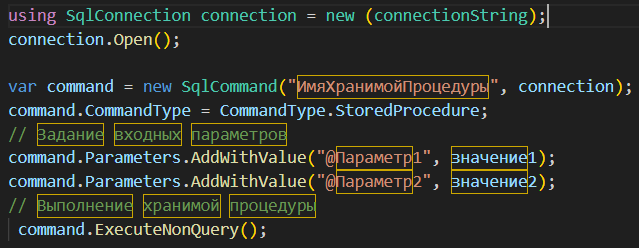


Рисунок 14 - Пример вызова хранимой процедуры

* 1. Для задания входных параметров хранимой процедуры необходимо добавить параметры к команде, вызывающей хранимую процедуру. В примере выше используется метод AddWithValue, который добавляет параметр с указанным именем и значением. В качестве имени параметра обычно используется символ "@", за которым следует имя параметра, а значение параметра передается вторым аргументом метода AddWithValue.

1. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение создавать и использовать хранимые процедуры в MS SQL Server;
   2. В ходе проделанной лабораторной работы был закреплен навык создания запросов на выборку и модификацию данных в MS SQL Server.

# Практическая работа №1

**Разработка приложений для обработки файлов**

1. **Цель работы**
   1. Научиться применять классы для работы с файлами в приложениях на C#.
2. **Контрольные вопросы**
   1. Directory – это статический класс, предоставляющий статические методы для работы с директориями (создание, перемещение, удаление и т.д). DirectoryInfo – представляет конкретную директорию на диске и позволяет работать с файлами и поддиректориями этой директории.
   2. File – это статический класс, предоставляющий статические методы для работы с файлами (чтение, запись, копирование и т.д). FileInfo – представляет конкретный файл на диске и позволяет получить информацию о нем и работать с его содержимым.
   3. Для получения списка файлов и папок определенного каталога можно воспользоваться методом GetFiles() и GetDirectories() соответственно.
   4. Некоторые из свойств класса FileInfo, которые позволяют получить информацию о файле:
      1. Name - имя файла,
      2. Extension - расширение файла,
      3. Length - размер файла в байтах,
      4. CreationTime - время создания файла,
   5. LastAccessTime - время последнего доступа к файлу.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение применять классы для работы с файлами в приложениях на C#.

# Практическая работа №2

**Работа с табличными файлами**

1. **Цель работы**
   1. Научиться выполнять создание и редактирование табличных документов на C#.
2. **Контрольные вопросы**
   1. Для работы с Excel в программе на C# требуется подключить пространство имен Microsoft.Office.Interop.Excel.
   2. Для создания объекта типа "приложение Excel" в программе на C# используется код, показанный на рисунке 1.

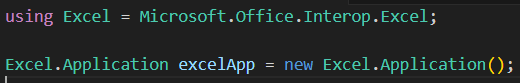


Рисунок 1 - Пример создания объекта типа "приложение Excel"

* 1. Workbooks представляет коллекцию всех книг (документов) Excel, открытых в приложении. Это свойство позволяет управлять книгами Excel.
  2. Worksheets представляет коллекцию всех листов в книге Excel. Каждый лист содержит данные, графики и другую информацию.
  3. Range представляет ячейки, ячейки диапазона или диапазон ячеек на листе Excel. С помощью Range можно работать с содержимым ячеек.
  4. Cells является частью Range и представляет собой коллекцию всех ячеек на листе Excel. С помощью Cells можно получить доступ к определенным ячейкам.
  5. Для доступа к значению конкретной ячейки можно воспользоваться свойством Value , а для диапазона - можно использовать метод get\_Range() и далее работать с этим диапазоном.

1. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение выполнять создание и редактирование табличных документов на C#.

# Практическая работа №3

**Работа с текстовыми файлами**

1. **Цель работы**
   1. Научиться выполнять создание и редактирование текстовых документов на C#.
2. **Контрольные вопросы**
   1. Для работы с Word в программе на C# требуется подключить пространство имен Microsoft.Office.Interop.Word.
   2. Для создания объекта типа "приложение Word" в программе на C# используется код, представленный на рисунке 2.

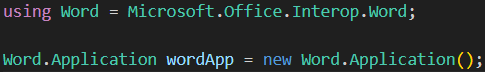


Рисунок 2 - Пример создания объекта типа "приложение Word"

* 1. Documents представляет коллекцию всех документов, открытых в приложении Word. Это свойство позволяет управлять документами Word.
  2. Range представляет диапазон текста в документе Word. С помощью Range можно выполнять операции над текстовыми данными, такими как форматирование, поиск и замена текста и т.д.
  3. Selection представляет выделенную область в документе Word. С помощью Selection можно работать с текущей выделенной частью документа, изменять форматирование текста и т.д.
  4. Paragraphs представляет коллекцию абзацев в документе Word.
  5. Tables представляет коллекцию таблиц в документе Word. С помощью Tables можно управлять структурой таблиц, добавлять и удалять строки и столбцы, форматировать таблицы и т.д.

1. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение выполнять создание и редактирование текстовых документов на C#.

# Практическая работа №4

**Сохранение настроек приложения**

1. **Цель работы**
   1. Научиться сохранять настройки в клиентском приложении на C#.
2. **Контрольные вопросы**
   1. Для добавления настроек в приложение на C# используется класс Properties.Settings. Для этого необходимо открыть проект в Visual Studio, щелкнуть правой кнопкой мыши по проекту в обозревателе решений, выбрать "Свойства", затем перейти на вкладку "Настройки". Здесь можно добавить нужные параметры и задать им значения по умолчанию.
   2. Для программного считывания значения параметра из настроек нужно обратиться к соответствующему свойству Properties.Settings.Default, указав имя параметра.
   3. Для программного изменения значения параметра в настройках также необходимо обратиться к свойству Properties.Settings.Default и присвоить новое значение параметру.
   4. Для сохранения изменений в параметрах в настройках нужно вызвать метод Save() у объекта Properties.Settings.Default.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение сохранять настройки в клиентском приложении на C#.

# Практическая работа №5

**Создание пользовательских элементов управления**

1. **Цель работы**
   1. Научиться создавать пользовательские элементы управления в приложении WPF.
2. **Контрольные вопросы**
   1. ControlTemplate применяется для определения визуального внешнего вида элемента управления в WPF приложениях.
   2. ControlTemplate может быть описан в ресурсах приложения (например, в файле XAML) или в коде приложения.
   3. UserControl применяется для создания пользовательских элементов управления, которые могут содержать в себе другие элементы управления, шаблоны и логику.
   4. Алгоритм создания пользовательского элемента управления включает определение его визуального внешнего вида с помощью ControlTemplate, добавление необходимых элементов управления, описание логики работы элемента.
   5. Обработчик события можно программно создать, определив метод, который будет обрабатывать событие, затем связать этот метод с нужным событием элемента управления.
   6. Свойство зависимости используется для определения свойств элементов управления, которые автоматически обновляются при изменении других свойств или привязываемых данных. Они также позволяют задать поведение свойств в различных контекстах приложения.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение создавать пользовательские элементы управления в приложении WPF.

# Практическая работа №6

**Привязка данных**

1. **Цель работы**
   1. Научиться выполнять привязку данных в приложении WPF.
2. **Контрольные вопросы**
   1. Привязка данных - это процесс связывания данных, например из источника данных (например, базы данных) с элементами пользовательского интерфейса (например, текстовым полям или элементам управления), чтобы данные автоматически отображались и обновлялись при изменениях в источнике данных.
   2. Шаблон настройки привязки данных включает в себя указание источника данных, целевого элемента управления и правил для обновления или отображения данных.
   3. Для создания конвертера значений необходимо реализовать интерфейс IValueConverter, который предоставляет методы для преобразования значений в привязке данных.
   4. Для валидации данных при привязке необходимо реализовать интерфейс IDataErrorInfo, который позволяет проверять и возвращать информацию об ошибках в привязанных данных.
3. **Вывод**
   1. В ходе проделанной лабораторной работы было сформировано умение выполнять привязку данных в приложении WPF.