**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

по дисциплине: «Объектно-ориентированное проектирование и программирование»

на тему: «Делегаты и события»

Выполнил: студент гр. ИТИ-21

Мусафиров А.Ю.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2024

**Цель работы**: разработать графическое приложение (WPF) без использования конструктора. Интерфейс должен генерироваться динамически в коде программы.

**Задание:**

Разработать графический редактор. Добавить панель синего цвета. Добавить слайдер, который изначально находится в начальном положении. При движении слайдера вращается панель. Конечное положение слайдера – поворот панели на 180 градусов. Вращение идет относительно центра панели.

|  |  |
| --- | --- |
| № | Условие |
| 14 | Разработать алгоритм бинарного поиска в массиве содержащем элементы любого типа. |

**Ход работы**

Создадим библиотеку классов и будем хранить самое важное в нашей программе − логику алгоритма бинарного поиска в массиве. После реализации нужных методов, приступим к реализации метода с итератором и метода бинарного поиска.

Для начала двоичный (бинарный) поиск − классический алгоритм поиска элемента в отсортированном массиве, использующий дробление массива на половины. Изначально берём левый и правый края массива. Далее находим среднее значение между ними и смотрим, больше/меньше/равно относительно искомого значения. Если меньше, то отбрасываем левую сторону массива, если больше, то отбрасываем правую. Если равно, то повезло.

На рисунке 1 представлен пример ввода массивов с разными типами.

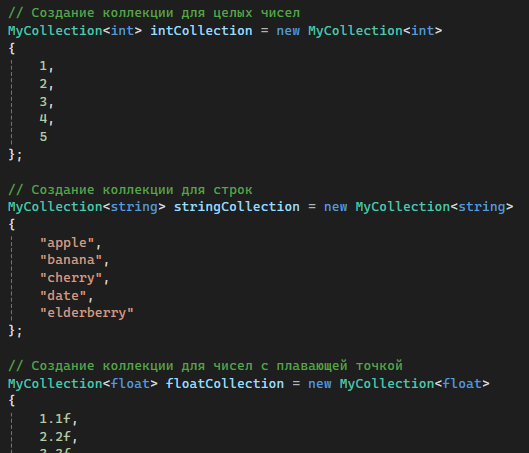


Рисунок 1 – Пример ввода массивов.

Для примера, будем искать в каждом массиве по одному элементу. В первом найдём позицию числа 1, во втором строки «*date*»*,* а в третьем значение 5.5f. На рисунке 2 представлена работа программы.

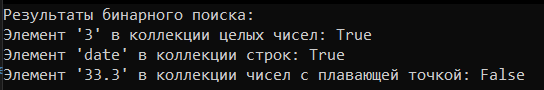


Рисунок 2 – Результат работы программы

В приложении А представлен код программы.

На рисунке 3 представлен репозиторий проекта.

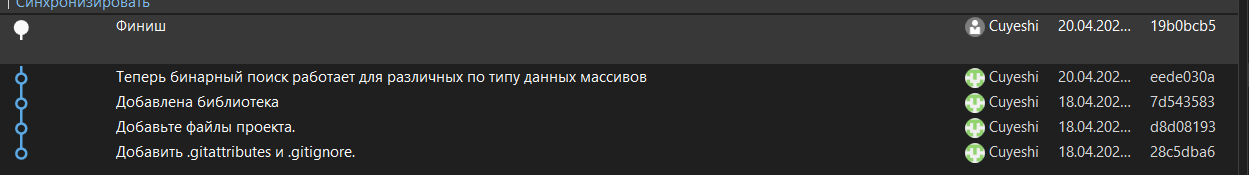


Рисунок 3 – Репозиторий проекта

**Вывод:** в результате выполнения лабораторной работы получили новые (и закрепили уже имеющиеся) знания работы со средой разработки *Visual Studio*, изучили новую концепцию итератор, разработали собственную коллекцию и реализовали бинарный поиск по массивам с разными типами данных.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

**Program.cs**

using ICollectionLibrary;

using System;

namespace Lab\_3

{

class Program

{

static void Main(string[] args)

{

// Создание коллекции для целых чисел

MyCollection<int> intCollection = new MyCollection<int>();

intCollection.Add(1);

intCollection.Add(2);

intCollection.Add(3);

intCollection.Add(4);

intCollection.Add(5);

// Создание коллекции для строк

MyCollection<string> stringCollection = new MyCollection<string>();

stringCollection.Add("apple");

stringCollection.Add("banana");

stringCollection.Add("cherry");

stringCollection.Add("date");

stringCollection.Add("elderberry");

// Создание коллекции для чисел с плавающей точкой

MyCollection<float> floatCollection = new MyCollection<float>();

floatCollection.Add(1.1f);

floatCollection.Add(2.2f);

floatCollection.Add(3.3f);

floatCollection.Add(4.4f);

floatCollection.Add(5.5f);

// Поиск элементов в каждой коллекции с помощью метода Contains

Console.WriteLine("Результаты бинарного поиска:");

Console.WriteLine($"Позиция элемента '3' в коллекции целых чисел: {intCollection.Contains(3)}");

Console.WriteLine($"Позиция элемента 'date' в коллекции строк: {stringCollection.Contains("date")}");

Console.WriteLine($"Позиция элемента '33.3' в коллекции чисел с плавающей точкой: {floatCollection.Contains(33.3f)}");

Console.ReadKey();

}

}

}

**BinarySearch.cs**

using System;

using System.Collections;

using System.Collections.Generic;

namespace ICollectionLibrary

{

public class MyCollection<T> : ICollection<T>, IEnumerable<T> where T : IComparable<T>

{

private T[] items;

private int count;

public MyCollection()

{

items = new T[100];

count = 0;

}

public int Count

{

get { return count; }

}

public bool IsReadOnly

{

get { return false; }

}

public void Add(T item)

{

if (count == items.Length)

{

throw new InvalidOperationException("Коллекция полна");

}

items[count] = item;

count++;

}

public void Clear()

{

count = 0;

}

public bool Contains(T item)

{

Array.Sort(items, 0, count);

int left = 0;

int right = count - 1;

while (left <= right)

{

int middle = left + (right - left) / 2;

int comparisonResult = Comparer<T>.Default.Compare(items[middle], item);

if (comparisonResult == 0)

{

return true;

}

else if (comparisonResult < 0)

{

left = middle + 1;

}

else

{

right = middle - 1;

}

}

return false;

}

public void CopyTo(T[] array, int arrayIndex)

{

Array.Copy(items, 0, array, arrayIndex, count);

}

public IEnumerator<T> GetEnumerator()

{

for (int i = 0; i < count; i++)

{

yield return items[i];

}

}

public bool Remove(T item)

{

int index = IndexOf(item);

if (index < 0)

{

return false;

}

Array.Copy(items, index + 1, items, index, count - index - 1);

count--;

return true;

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

private int IndexOf(T item)

{

for (int i = 0; i < count; i++)

{

if (Equals(items[i], item))

{

return i;

}

}

return -1;

}

}

}