1. Цель и задачи лабораторной работы

Цель: изучить возможности применения делегатов в языке C#.

Задачи:

1. Освоить принципы работы с делегатами;
2. Освоить основные направления применения делегатов;
3. Изучить способы использования делегатов совместно с потоками.

2. Реализация индивидуального задания

Согласно варианту задания, требуется реализовать пользовательский тип делегата со следующей сигнатурой: ***Action<Func<int>, List<float>>***

Это означает, что делегат должен принимать два аргумента:

1. *Func<int>*
    a. Метод, принимающий *int* с непустым возвратом
2. *List<float>*
    a. Список значений с float

И ничего не возвращать.

Был объявлен делегат *CustomDelegate*.

2.1. Листинг программного кода

```
namespace lab1
{
    class Program
    {
        /// <summary>
        /// Кастомный делегат
        /// </summary>
        /// <param name="intGetter">Получатель цифры</param>
        /// <param name="floatList">Дефолтные цифры</param>
        public delegate void CustomDelegate(Func<int> intGetter, List<float>
floatList);

        /// <summary>
        /// Получатель случайного числа
        /// </summary>
        /// <returns>
        ///     <summary>
```

```
        ///     Случайное число
        ///     </summary>
        /// </returns>
        static int GetRandomNumber()
        {
            Random rnd = new();
            return rnd.Next(1, 100);
        }

        /// <summary>
        /// Получатель фиксированного числа
        /// </summary>
        /// <returns>
        ///     <summary>
        ///     Фиксированное число
        ///     </summary>
        /// </returns>
        static int GetFixedNumber()
        {
            return 42;
        }

        /// <summary>
        /// Получатель пользовательского числа
        /// </summary>
        /// <returns>
        ///     <summary>
        ///     Пользовательскоге число.
        ///     </summary>
        /// </returns>
        /// <exception cref="null">
        /// В случае ошибки ввода возвращается константа
        /// </exception>
        static int GetUserInput()
        {
            Console.Write("Введите число: ");
            string? input = Console.ReadLine();
            try
            {
#pragma warning disable CS8604
                return int.Parse(input);
#pragma warning restore CS8604
            }
            catch (FormatException)
            {
                const int default_number = 17;
                Console.WriteLine($"В следующий раз используйте цифры!\nВаш ввод
был заменён на: {default_number}");
                return default_number;
            }
```

```
    }

    /// <summary>
    /// Добавление чисел в список
    /// </summary>
    /// <param name="numberGenerator">Генератор цифр</param>
    /// <param name="numbers">Массив дефолтных фицр</param>
    static void ProcessList(Func<int> numberGenerator, List<float> numbers)
    {
        Console.WriteLine("=== Обработка списка ===");

        // Генерируем число с помощью переданного делегата
        int generatedNumber = numberGenerator();
        Console.WriteLine($"Сгенерировано число: {generatedNumber}");

        // Добавляем преобразованное число в список
        numbers.Add((float)generatedNumber / 2);
        numbers.Add((float)generatedNumber * 1.5f);

        // Выводим содержимое списка
        Console.WriteLine("Содержимое списка:");
        foreach (var num in numbers)
        {
            Console.WriteLine($"- {num:F2}");
        }
    }

    /// <summary>
    /// Анализатор списка
    /// </summary>
    /// <param name="numberGenerator">Генератор цифр</param>
    /// <param name="numbers">Массив дефолтных фицр</param>
    static void AnalyzeList(Func<int> numberGenerator, List<float> numbers)
    {
        Console.WriteLine("=== Анализ списка ===");

        int baseValue = numberGenerator();
        Console.WriteLine($"Базовое значение: {baseValue}");

        if (numbers.Count > 0)
        {
            float sum = 0;
            foreach (var num in numbers)
            {
                sum += num;
                Console.WriteLine($"Элемент: {num:F2}");
            }

            float average = sum / numbers.Count;
            Console.WriteLine($"Сумма: {sum:F2}, Среднее: {average:F2}");
        }
```

```
            else
            {
                Console.WriteLine("Список пуст!");
            }
        }

        /// <summary>
        /// Очистка и заполнение списка
        /// </summary>
        /// <param name="numberGenerator">Генератор цифр</param>
        /// <param name="numbers">Массив дефолтных фицр</param>
        static void ClearAndFillList(Func<int> numberGenerator, List<float>
numbers)
        {
            Console.WriteLine("=== Очистка и заполнение ===");

            numbers.Clear();

            for (int i = 0; i < 3; i++)
            {
                int value = numberGenerator() + i;
                numbers.Add(value);
                Console.WriteLine($"Добавлено: {value}");
            }
        }

        static void Main(string[] args)
        {
            CustomDelegate myDelegate;

            // Дефолтные цифры
            List<float> numbers = [1.5f, 2.8f, 3.2f];

            Console.WriteLine("Демонстрация работы делегатов:");
            Console.WriteLine("==============================");

            // 1. Используем делегат с разными методами

            // Первый вызов
            myDelegate = ProcessList;
            Console.WriteLine("\n1. Вызов ProcessList с GetRandomNumber:");
            myDelegate(GetRandomNumber, new List<float>(numbers));

            // Второй вызов
            myDelegate = AnalyzeList;
            Console.WriteLine("\n2. Вызов AnalyzeList с GetFixedNumber:");
            myDelegate(GetFixedNumber, new List<float>(numbers));

            // Третий вызов
            myDelegate = ClearAndFillList;
            Console.WriteLine("\n3. Вызов ClearAndFillList с GetRandomNumber:");
```

```
            myDelegate(GetRandomNumber, new List<float>(numbers));

            // 2. Демонстрация multicast делегата
            Console.WriteLine("\n4. Multicast делегат:");
            CustomDelegate multicastDelegate = ProcessList;
            multicastDelegate += AnalyzeList;
            multicastDelegate += ClearAndFillList;

            multicastDelegate(GetFixedNumber, new List<float>(numbers));

            // 3. Использование встроенного Action
            Console.WriteLine("\n5. Использование встроенного Action:");
            // Встроенный делегат вида Action
            Action<Func<int>, List<float>> builtInAction = ProcessList;
            builtInAction(GetUserInput, numbers);
        }
    }
}
```

2.2. Описание кода

В коде сначала показывается вызов делегатов с одинарными функциями.

Затем демонстрируется работа multicast делегата (когда передаётся несколько функций в один делегат)

В конце показывается, что можно было инициализировать делегат через встроенный Action.

Функции:

1. *GetRandomNumber()*
    a. Получает случайное число.
2. *GetFixedNumber()*
    a. Возвращает константу.
3. *ProcessList(Func<int> numberGenerator, List<float> numbers)*
    a. Получает число из *numberGenerator*, затем делит его на 2, умножает на полтора и добавляет в список, после чего выводит список в консоль
4. *AnalyzeList(Func<int> numberGenerator, List<float> numbers)*
    a. Анализирует сумму и среднее значение списка
5. *ClearAndFillList(Func<int> numberGenerator, List<float> numbers)*

a. Очищает и заполняет список случайными числами

2.3. Результат работы программы

Демонстрация работы делегатов:

==========

1. Вызов ProcessList с GetRandomNumber:

=== Обработка списка ===

Сгенерировано число: 75

Содержимое списка:

- 1,50

- 2,80

- 3,20

- 37,50

- 112,50

2. Вызов AnalyzeList с GetFixedNumber:

=== Анализ списка ===

Базовое значение: 42

Элемент: 1,50

Элемент: 2,80

Элемент: 3,20

Сумма: 7,50, Среднее: 2,50

3. Вызов ClearAndFillList с GetRandomNumber:

=== Очистка и заполнение ===

Добавлено: 55

Добавлено: 54

Добавлено: 41

4. Multicast делегат:

=== Обработка списка ===

Сгенерировано число: 42

Содержимое списка:

- 1,50

- 2,80

- 3,20

- 21,00

- 63,00

=== Анализ списка ===

Базовое значение: 42

Элемент: 1,50

Элемент: 2,80

Элемент: 3,20

Элемент: 21,00

Элемент: 63,00

Сумма: 91,50, Среднее: 18,30

=== Очистка и заполнение ===

Добавлено: 42

Добавлено: 43

Добавлено: 44

5. Использование встроенного Action:

=== Обработка списка ===

Введите число: 0

Сгенерировано число: 0

Содержимое списка:

- 1,50

- 2,80

- 3,20

- 0,00

- 0,00

3. Контрольные вопросы

1. **Что такое тип делегата? Какой аналог типа делегата существует в C++?**
    a. Делегат в C# — это тип, который хранит ссылку на метод с определённой сигнатурой. Он позволяет передавать методы как параметры, вызывать их позже или даже объединять несколько методов в цепочку. В C++ ближайший аналог — это std::function или указатель на функцию.
2. **Опишите основные направления использования делегатов.**
    a. Делегаты используются для обратных вызовов (callbacks), обработки событий (например, кликов в интерфейсе), реализации паттернов вроде Strategy или Observer, а также в LINQ и асинхронном программировании. Они делают код гибким и расширяемым.
3. **Какие механизмы технологии Windows Forms реализованы с использованием делегатов?**
    a. В Windows Forms вся система событий построена на делегатах. Например, событие Click у кнопки — это делегат типа EventHandler. При подписке через button.Click += MyHandler вы добавляете метод в список вызовов делегата.
4. **Для чего предназначен тип Action? Чем он отличается от Func?**
    a. Action — это готовый делегат для методов, которые ничего не возвращают (void). Func — для методов, которые возвращают

значение. Последний тип в Func всегда — тип возвращаемого результата.

5. **Чем пользовательские делегаты отличаются от библиотечных?**
    a. Пользовательские делегаты объявляются вручную с помощью ключевого слова delegate и имеют понятное имя (например, DataProcessor). Библиотечные (Action, Func) — универсальные и подходят для большинства случаев, но могут быть менее читаемыми в сложной логике. По сути, они делают одно и то же — просто разный стиль.