МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

Методические указания

по выполнению лабораторных работ
по дисциплине
«Введение в технологии высокопроизводительных вычислений вычислений»
для направления подготовки **09.03.02 Информационные системы и технологии**
направленность (профиль) **Информационные системы и технологии обработки цифрового контента**

Пятигорск
2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развитие информационных технологий порождает новые аппаратные и программные архитектуры информационных систем. Увеличени производительности серверов не ограничивается ростом тактовой частоты процессоров – это процесс многокомпонентный: основную роль в увеличении производительности информационных систем играют новые концептуальные схемы взаимодействия вычислителей. При этом учитывается характер вычислителя: узел кластера, процессор, ядро, абстрактный класс задачи или потока. Каждое направление порождает целые семейства технологий построения высокопроизводительных вычислительных систем.

Разработчику информационных систем доступны следующие технологии для построения параллельных и распределенных вычислительных систем:

– вычислительные системы с общей памятью; соответственно программные системы ориентированные на работу в данной модели;

– вычислительные системы, ориентированные на работу в модели распределенной памятью и програмнные компоненты данной архитектуры;

– кластерные технологии;

– высокопроизводительные системы, программным ядром которых являются базы данных;

– вычислительные системы, основанные на графических процессорах;

– программные комплексы функционирующие в многозадачной и многопоточной среде.

Данные направления не исчерпывают всех современных подходов построения высокопроизводительных систем, но образуют группы технологических решений, рассмотрение которых и работа с которыми позволяет изучить и получить практический опыт в сфере рпзработки и построения высокопроизводительных истем.

ЛАБОРАТОРНАЯ РАБОТА 1. ПРИМЕНЕНИЕ ДЕЛЕГАТОВ

1. Цель и содержание

Цель лабораторной работы: изучить возможности применения делегатов в языке C#.

Задачи лабораторной работы:

– освоить принципы работы с делегатами;

– освоить основные направления применения делегатов;

– изучить способы использования делегатов совместно с потоками.

2. Теоретическое обоснование

2.1 Назначение типов делегатов

Делегаты предназначены для ситуаций, когда нужно передать метод другому методу. Бывают случаи, когда методу потребуется обращение к другому методу, но определить к какому именно на этапе компиляции не представляется возможным. Эта информация доступна только во время выполнения, а потому должна быть передана первому методу в виде параметра.

Подобное поведение свойственно в следующих случаях:

1. Запуск потоков и задач. В C# существует возможность сообщить компилятору, что нужно запустить некоторую новую последовательность исполнения параллельно той, что работает в данный момент. Такая последовательность называется потоком (thread), а его запуск осуществляется с помощью метода Start одного из базовых классов – System.Threading.Thread. Если компьютеру сообщается о необходимости запуска новой последовательности исполнения, то также нужно сообщить, откуда именно ее следует начать, т.е. вызов какого метода должен ее

запускать. Другими словами, метод Thread.Start ( ) должен получить параметр, который указывает метод, подлежащий вызову потоком.

2. Обобщенные библиотечные классы. Многие библиотеки содержат код для выполнения разнообразных стандартных задач. Обычно такие библиотеки могут быть самодостаточными – в том смысле, что при их написании вы точно знаете, как следует решать задачи. Однако иногда задача может включать подзадачу, о которой знает только индивидуальный клиентский код, использующий эту библиотеку. Например, требуется написать класс, принимающий массив объектов и сортирующий их по возрастанию. Частью процесса сортировки должно быть повторяющееся сравнение двух объектов из массива, чтобы определить, какой из них нужно расположить первым. Если необходимо обеспечить классу возможность сравнивать любые объекты, то он не может знать заранее, как выполнять такие сравнения. Только клиентский код, использующий класс, может сообщить ему, как следует выполнять сравнение конкретных объектов, массив которых необходимо отсортировать. Клиентский код должен передать вашему классу подробности относительно того, какой метод вызвать для выполнения сравнения.

3. События. Главная идея здесь состоит в том, что часто приходится иметь дело с кодом, который должен быть проинформирован о возникновении каких-то событий. В программировании графического интерфейса пользователя полно подобных ситуаций. Когда событие происходит, исполняющая среда должна знать, какой метод необходимо вызвать. Это делается передачей методу, обрабатывающему события, параметра-делегата.

Как и в случае классов для использования делегата, его необходимо заранее объявить. Синтаксис объявления делегатов выглядит следующим образом:

Имя _ типа _ делегата ([*параметры*]) .

Например:

```
delegate void MyFirstDelegate(int pParam);
```

В данном примере определяется делегат, экземпляр которого принимает один параметр типа int и возвращает значение типа void.

Еще несколько примеров объявления делегатов:

```
delegate int Summ (int a, int v);
delegate string ConverterFromFloat (float D);
delegate int[] ToVector(int a, int b, int c);
delegate string GetAString();
```

Следует понимать, что при объявлении делегата с использованием ключевого слова delegate фактически создается специализированный класс. Делегаты создаются как классы, унаследованные от System.MulticastDelegate.

```
delegate string GetAString();

class Program
{
    static void Main(string[] args)
    {
        int x = 40;
        // переменная firstStringMethod
        //инициализируется значением x.ToString(),
        GetAString firstStringMethod = new GetAString(x.ToString);

        // приведенный оператор эквивалентен следующему:
        // Console.WriteLine("Строка равна " + x.ToString ());
        Console.WriteLine("Строка равна " + firstStringMethod());
    }
}
```

Применение скобок к экземпляру делегата это все равно что вызов метода Invoke ( ), то есть следующие сроки выполняют одно и тоже действие:

firstStringMethod ( );

firstStringMethod.Invoke ( );

Чтобы сократить код, в каждом месте, где требуется экземпляр делегата, можно просто передать имя адреса. Это называется выведением делегата (delegate inference). Это средство C# работает до тех пор, пока компилятор способен разрешить экземпляр делегата в специфический тип. В

предыдущем примере переменная firstStringMethod типа GetAsString инициализируется новым экземпляром делегата GetAsString:

GetAString firstStringMethod = new GetAString(x.ToString);

Сделать то же самое можно, просто передав переменной firstStringMethod имя метода в переменной x:

GetAString firstStringMethod = x.ToString;

В обоих случаях компилятор C# создает один и тот же код. Компилятор обнаруживает, что тип делегата требуется firstStringMethod, поэтому создает экземпляр делегата типа GetAString и передает конструктору адрес метода в объекте x.

Очевидно, что дописывать круглые скобки при передаче имени метода недопустимо.

Экземпляр делегата может ссылаться на любой метод — уровня экземпляра или статический – любого объекта любого типа, если сигнатура этого метода совпадает с сигнатурой делегата.

2.2 Делегаты Action<T> и Func<T> библиотеки .NET Framework.

Вместо определения нового типа делегата с каждым типом параметра и возврата можно использовать делегаты Делегаты Action<T> и Func<T>. Обобщенный делегат Делегаты Action<T> предназначен для ссылки на метод, возвращающий мщшв. Этот класс делегата существует в различных вариантах, так что ему можно передавать до 16 разных типов параметров.

Класс Action без обобщенного параметра предназначен для вызова методов без параметров, Action<in T> – для вызова метода с одним параметром, Action<in Tl, in T2> – для вызова метода с двумя параметрами и Action<in Tl, in T2, in T3, in T4, in T5, in T6, in T7, in T8> – для вызова метода с восемью параметрами.

Делегаты Func<T> могут использоваться аналогичным образом. Func<T> позволяет вызывать методы с типом возврата. Подобно Action<T>,

Func<T> определен в разных вариантах для передачи до 16 типов параметров и типа возврата.

Func<out Tresult> – тип делегата для вызова метода с типом возврата, но без параметров, Func<int Tl, out TResult> – для метода с одним параметром, а Func<in Tl, in T2, in T3, in T4, out TResult> – для метода с четырьмя параметрами.

2.3 Лямбда-выражения

Начиная с C# 3.0, доступен новый синтаксис для назначения реализации кода делегатам, называемый лямбда-выражениями (lambda expression). Лямбда-выражения могут использоваться везде, где есть параметр типа делегата. Ниже показан предыдущий пример, в котором применялись анонимные методы, адаптированный для использования лямбда-выражения.

```
static void Main(string[] args)
{
    string mid = ", средняя часть,";
    Func<string, string> lambda = param =>
    {
    param += mid;
    param += " а это добавлено к строке.";
    return param;
    };
    Console.WriteLine(lambda ("Начало строки")) ;
}
```

При использовании лямбда-выражений существует несколько способов определения параметров. Если параметр один, то достаточно его имени. В следующем лямбда-выражении используется параметр по имени s. Поскольку тип делегата определяет параметр string, то s имеет тип string. Реализация вызывает метод String.Format () для возврата строки, которая в конечном итоге выводится на консоль при вызове делегата – изменение регистра текстовой строки «TEST»:

```
static void Main(string[] args)
{
    Func<string, string> oneParam = s
        => String.Format("изменение регистра {0}",
        s.ToUpper());
    Console.WriteLine(oneParam("test"));
}
```

Если делегат использует более одного параметра, имена параметров можно комбинировать внутри скобок. Ниже представлены параметры x и y типа double, определенные делегатом Func<double, double, double>:

```
static void Main(string[] args)
{
    Func<double, double, double> twoParams = (x, y) => x * y;
    Console.WriteLine(twoParams(10, 2));
}
```

Для удобства к именам переменных внутри скобок можно добавлять типы параметров:

```
static void Main(string[] args)
{
    Func<double, double, double> twoParamsWithTypes =
    (double x, double y) => x * y;
    Console.WriteLine(twoParamsWithTypes(100, 2));
}
```

Если лямбда-выражение состоит из единственного оператора, блок метода с фигурными скобками и оператором return не требуются. Компилятор добавляет return неявным образом:

```
Func<double, double> square =
    x => x * x;
```

Можно добавить фигурные скобки, оператор return и точку с запятой. Просто обычно легче читать код без них:

```
Func<double, double> square =
    x =>
    {
        return x * x;
    };
```

Но если нужно включить в реализацию лямбда-выражения несколько операторов, то фигурные скобки и оператор return обязательны:

```
static void Main(string[] args)
{
    string mid = "Начало строки";

    Func<string, string> lambda = param =>
    {
        param += mid;
        param += " а это добавлено к строке.";
        return param;
    };
}
```

3. Методика и порядок выполнения работы

1. Создайте приложение C# в среде MS Visual Studio.

2. В соответствии с вариантом индивидуального задания реализуйте пользовательский тип делегата требуемой сигнатуры и выполните с его использованием вызов нескольких методов (с корректной сигнатурой).

Индивидуальное задание.

Вариант	Делегат (сигнатура)
1	Action<Func<float>, bool, List<float>>
2	Func<Action<char>, bool, double, double>
3	Action<Func<double>, double, double>
4	Func<Action<float>, int, float, bool>
5	Action<Func<bool, int>, char, string>
6	Func<Action<int>, bool, char, string>
7	Action<Func<bool>, double, double>
8	Func<Action<T>, float, float> : T object
9	Action<Func<int>, char, char>
10	Func<Action<int, int>, bool>
11	Action<Func<int>, char>
12	Func<Action<bool>, float, float>
13	Action<Func<char>, int, bool>
14	Func<Action<List<int>>, bool>
15	Action<Func<int>, List<float>>
16	Func<Action<string>, int, int>
17	Action<Func<int, int>, List<string>>
18	Func<Action<string, int>, int, int>
19	Action<Func<bool>, string, int>

20	Func<Action<int>, int, int>
21	Action<Func<double, double, double>, double>
22	Func<Action<int, int>, string>
23	Action<Func<char, int>, List<string>>
24	Func<Action<IEnumerable<T>>, int> : T object
25	Action<Func<int, int, int>, string>

4. Вопросы для защиты работы

1. Что такое тип делегата? Какой аналог типа делегата существует в C++?

2. Опишите основные направления использования делегатов.

3. Какие механизмы технологии Windows Forms реализованы с использованием делегатов?

4. Для чего предназначен тип Action<T>? Чем он отличается от Func<T>?

5. Чем пользовательские делегаты отличаются от библиотечных?

ЛАБОРАТОРНАЯ РАБОТА 2. ПРИМЕНЕНИЕ АСИНХРОННЫХ ДЕЛЕГАТОВ ДЛЯ РЕАЛИЗАЦИИ МНОГОПОТОЧНОСТИ

1. Цель и содержание

Цель лабораторной работы: научиться использовать делегаты для организации многопоточного приложения.

Задачи лабораторной работы:

– научиться объявлять, инициализировать и запускать потоки с использованием пользовательских делегатов;

– научиться запускать потоки с использованием библиотечных делегатов Action<T> и Func<T>;

– научиться запускать параллельные потоки с использованием лямбда-выражений.

2. Теоретическое обоснование

Наиболее простым способом для создания потока является определение делегата и его вызов асинхронным образом. Класс Delegate поддерживает и возможность асинхронного вызова методов. Для решения поставленной задачи класс Delegate создает отдельный поток.

Чтобы посмотреть в действии на асинхронные возможности делегатов, сначала нужно создать метод, выполнение которого занимает определенное время. Например, ниже показан метод TakesAWhile (), который благодаря вызову Thread.Sleep( ) выполняется минимум столько времени, сколько в миллисекундах задано во втором аргументе.

```
static int TakesAWhile(int data, int ms)
{
    Console.WriteLine("TakesAWhile запущен");
    Thread.Sleep(ms);
    Console.WriteLine("TakesAWhile завершен");
    return ++data;
}
```

Чтобы обеспечить вызов этого метода из делегата, понадобится определить тип делегата с тем де самым параметром и возвращаемым типом:

```
public delegate int TakesAWhileDelegate(int data, int ms);
```

Теперь можно применять различные приемы для асинхронного вызова данного делегата и возврата результатов.

Одним из таких приемов является опрос и проверка, завершил ли делегат свою работу. Созданный класс delegate предоставляет метод BeginInvoke ( ), в котором могут передаваться входные параметры, определенные вместе с типом делегата. Метод BeginInvoke ( ) всегда имеет два дополнительных параметра типа AsyncCallback и object, которые будут рассматриваться позже. Сейчас главный интерес представляет возвращаемый тип метода – IAsyncResult. С помощью IAsyncResult можно извлекать информацию о делегате и проверять, завершил ли он свою работу, что и демонстрируется в примере с применением свойства IsCompleted. Цикл while продолжает выполняться в главном потоке программы до тех пор, пока делегат не завершит работу.

```
static void Main(string[] args)
{
    // синхронный вызов метода
    // TakesAWhileA, 3000);

    // асинхронный вызов метода с применением делегата
    TakesAWhileDelegate dl = TakesAWhile;

    IAsyncResult ar = dl.BeginInvoke(1, 3000, null, null);
    while (!ar.IsCompleted)
    {
        // выполнение еще каких-нибудь Сепераций в главном потоке
        Console.Write(".");
        Thread.Sleep(50);
    }
    int result = dl.EndInvoke(ar);

    // вывод результата
    Console.WriteLine("result: {0}", result);
}
```

После запуска этого приложения можно увидеть, что главный поток и поток делегата выполняются параллельно, а после завершения работы потока делегата главный поток прекращает проход по циклу. Если главный поток завершает выполнение, не дожидаясь завершения работы делегата, поток делегата останавливается.

В данном разделе методических указаний рассмотрен самый простой метод запуска метода в отдельном потоке – с использованием асинхронного делегата. Рассмотрен также механизм передачи параметров в поток, выполняемый параллельно (асинхронно); возврат результата из асинхронно выполняемого метода. Необходимо обратить внимание на реализацию механизма ожидания завершения работы асинхронного метода.

3. Методика и порядок выполнения работы

1. Создайте приложение на языке C# в MS Visual Studio.

2. В соответствии с индивидуальным вариантом разработайте требуемый тип делегата (пользовательский, библиотечный или лямбда-выражение).

3. Реализуйте асинхронное выполнение метода на основе разработанного делегата с возможностью мониторинга процесса выполнения, передачи параметров в метод и получения результата работы метода.

Индивидуальное задание.

Вариант	Тип делегата	Решаемая задача (результат метода)	Входные параметры
1	пользовательский	Метод возвращает сумму элементов матрицы целых случайных чисел	Два параметра: размер матрицы
2	библиотечный	Метод возвращает разницу максимального и минимального элементов матрицы целых случайных чисел	Два параметра: размер матрицы
3	лямбда-выражение	Метод возвращает логическое значение, указывающее существует ли заданное число в массиве целых случайных чисел	Два параметра: размер массива и искомый элемент
4	пользовательский	Метод возвращает матрицу случайных битовых значений (0 или 1), формируемую случайным образом	Два параметра: размер матрицы
5	библиотечный	Метод возвращает строку максимальной длины на основе переданного массива строк	Один параметр: массив (или список) строк
6	лямбда-выражение	Метод возвращает подмножество элементов массива случайных чисел, которые делятся на 3	Один параметр: размер исходного массива
7	пользовательский	Метод возвращает число – количество элементов присутствующих в обоих массивах случайных чисел.	Два параметра: размеры массивов
8	библиотечный	Метод возвращает среднее арифметическое элементов матрицы случайных чисел.	Два параметра: размер матрицы
9	лямбда-выражение	Метод возвращает результат шифрования строки: каждый исходный символ строки заменяется шифрованным символом, код которого на n больше кода исходного символа.	Два параметра: исходная строка, число сдвига n
10	пользовательский	Метод возвращает статистику для строки: список пар (символ строки, количество вхождений)	Один параметр: анализируемая строка

11	библиотечный	Метод возвращает разницу максимального и минимального элементов матрицы.	Два параметра: размер матрицы
12	лямбда-выражение	Метод возвращает подмножество элементов массива случайных чисел, которые являются четными	Один параметр: размер исходного массива
13	пользовательский	Метод возвращает количество нечетных элементов в матрице случайных чисел	Два параметра: размер матрицы
14	библиотечный	Метод возвращает подмножество элементов массива случайных чисел, которые отличаются от заданного числа не более чем на 4.	Два параметра: размер массива, значеное число
15	лямбда-выражение	Метод возвращает логическое значение, указывающее существует ли заданный символ в строке	Два параметра: строка и искомый символ
16	пользовательский	Метод возвращает два числа – максимальные элементы массивов случайных чисел.	Два параметра: размеры массивов
17	библиотечный	Метод возвращает скадярное произведение двух случайных векторов.	Один параметр: размер векторов
18	лямбда-выражение	Метод возвращает сумму двух матриц случайных чисел	Два параметра: размер матриц
19	пользовательский	Метод возвращает количество вхождений заданного элемента в матрице вещественных чисел	Два параметра: матрица и целевой элемент
20	библиотечный	Метод возвращает подмножество элементов массива случайных чисел, которые делятся на 6	Один параметр: размер исходного массива
21	лямбда-выражение	Метод возвращает результат сложения двух случайных векторов целых чисел	Один параметр: размер векторов
22	пользовательский	Метод возвращает количество вхождений заданного символа в строке	Два параметра: строка и целевой символ
23	библиотечный	Метод возвращает подмножество четных элементов матрицы случайных чисел	Два параметра: размер матрицы
24	лямбда-выражение	Метод возвращает статистику массива случайных целых чисел: список пар (число массива, количество вхождений)	Один параметр: размер массива

25	пользовательский	Метод возвращает логическое значение, указывающее существует ли заданный элемент в матрице чисел double	Два параметра: матрица и искомый элемент

4. Вопросы для защиты работы

1. Поясните назначение типа IAsyncResult.

2. Для чего используется метод Thread.Sleep( )?

3. Поясните механизм возврата значения из метода асинхронного делегата.

4. Как произвести возврат более одного значения из метода?

5. Какая разница существует между библиотечными делегатами, пользовательскими типами делегатов и лямбда-выражениями? Являются ли эти делегаты взаимозаменяемыми при реализации асинхронного вызова методов?

Индивидуальное задание.

Вариант	Основная задача	Задачи продолжения
1	Генерация матрицы случайных чисел (размер задается пользователем)	2 задачи: расчет суммы элементов; поиск максимального элемента.
2	Генерация массива случайных чисел (размер задается пользователем)	2 задачи: вычисление количества элементов, делящихся на 3; поиск минимального элемента.
3	Генерация матрицы случайных чисел (размер определяется случайным образом)	2 задачи: расчет суммы всех четных элементов; поиск среднего арифметического элементов.
4	Генерация массива случайных чисел (размер определяется случайным образом)	2 задачи: расчет суммы нечетных элементов; подсчет количества элементов, кратных 3.
5	Генерация матрицы случайных чисел (размер задается пользователем)	2 задачи: вывод матрицы в консоль; подсчет количества элементов, кратных 7.
6	Генерация массива случайных чисел (размер задается пользователем)	2 задачи: расчет суммы квадратов четных элементов; поиск максимального элемента.
7	Генерация матрицы случайных чисел (размер определяется случайным образом)	2 задачи: поиск максимального элемента в каждом столбце; поиск минимального элемента матрицы.
8	Генерация массива случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы кубов нечетных элементов; поиск максимального элемента; поиск минимального элемента.
9	Генерация матрицы случайных чисел (размер задается пользователем)	2 задачи: поиск минимального элемента в каждом столбце; поиск элементов матрицы, отличающихся не более чем на 4 от заданного пользователем.
10	Генерация массива случайных чисел (размер задается пользователем)	2 задачи: поиск максимального элемента; подсчет количества элементов массива, делящихся на 5.

11	Генерация матрицы случайных чисел (размер определяется случайным образом)	3 задачи: вывод матрицы в консоль; подсчет количества элементов массива, которые делятся на 3; поиск минимума в каждой строке.
12	Генерация массива случайных чисел (размер определяется случайным образом)	2 задачи: поиск минимального элемента среди нечетных; подсчет количества элементов массива, делящихся на 7.
13	Генерация матрицы случайных чисел (размер задается пользователем)	3 задачи: расчет суммы кубов четных элементов; поиск среднего арифметического элементов; поиск минимального элемента.
14	Генерация массива случайных чисел (размер задается пользователем)	2 задачи: поиск максимального элемента среди четных; подсчет количества элементов массива, отличающихся от заданного пользователем не более чем на 3.
15	Генерация матрицы случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы четных элементов, делящихся на 4; поиск максимума среди элементов, делящихся на 3; поиск минимального элемента.
16	Генерация массива случайных чисел (размер определяется случайным образом)	3 задачи: вычисление квадрата суммы элементов; вычисление суммы кубов элементов; вывод массива в консоль.
17	Генерация матрицы случайных чисел (размер задается пользователем)	3 задачи: расчет суммы элементов, делящихся на 4; расчет количества элементов, делящихся на 3; поиск минимального элемента.
18	Генерация массива случайных чисел (размер задается пользователем)	3 задачи: расчет суммы нечетных элементов, делящихся на 7; поиск максиального элемента; вывод массива в консоль.

19	Генерация матрицы случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы нечетных элементов, делящихся на 7; поиск максимума среди элементов, делящихся на 2; вывод матрицы в консоль.
20	Генерация массива случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы четных элементов, делящихся на 4; поиск максимума среди элементов, делящихся на 3; поиск минимального элемента.
21	Генерация матрицы случайных чисел (размер задается пользователем)	3 задачи: расчет суммы кубов элементов, делящихся на 3; поиск максимального элемента; поиск минимального элемента в каждом столбце.
22	Генерация массива случайных чисел (размер задается пользователем)	3 задачи: расчет суммы нечетных элементов, делящихся на 5; поиск максимума среди элементов, делящихся на 9; поиск минимального элемента среди четных.
23	Генерация матрицы случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы элементов, делящихся на 5; поиск максимума среди элементов, делящихся на 9; вывод матрицы в консоль.
24	Генерация массива случайных чисел (размер определяется случайным образом)	3 задачи: расчет суммы квадратов четных элементов; поиск максимума среди элементов, делящихся на 9; вывод всех элементов массива.
25	Генерация матрицы случайных чисел (размер задается пользователем)	3 задачи: расчет суммы нечетных элементов, делящихся на 2; поиск максимума в каждом столбце; поиск минимального элемента в каждой строке.

4. Вопросы для защиты работы

1. Что такое функция продолжения? Для чего используется данный механизм?

2. Сколько задач продолжения может иметь каждая задача?

3. С помощью какого метода класса Task можно задать задачу продолжения?

4. Какие параметры принимает метод, представляющий задачу продолжения?

5. Поясните механизм определения условных задач продолжения.

6. Опишите методы класса TaskFactory, предназначенные для указания задач продолжения.

7. Опишите механизм, использующий задачи продолжения и позволяющий запустить большое количество задач (100, 1000, ...) одной командой Start( ).

СПИСОК ЛИТЕРАТУРЫ

1. Вилле К. Представляем C# / Вилле К. – М.: ДМК Пресс, 2008. – 183 с. Режим доступа: http://e.lanbook.com.

2. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. – М.: ДМК Пресс, 2007. – 368 с. Режим доступа: http://e.lanbook.com.

3. Рандольф, Н. Visual Studio 2010 для профессионалов / Н. Рандольф, Д. Гарднер. – М.: Диалектика, 2011. – 1184 с. – ISBN 978-5-8459-1683-9.

4. Уотсон, К. Visual C# 2010. Полный курс / К. Уотсон, К. Нагел. – М.: Вильямс, 2010. – 960 с. – ISBN 978-5-8459-1699-0, 978-0-470-50226-6.

5. Учебное пособие «Технологии программирования I» для студентов специальности 09.03.02 «Информационные системы и технологии» / сост. Николаев Е.И.; рец. Мочалов В.П, Маликов А.В. – Ставрополь : СКФУ, 2011. – 150 с.

ДОКУМЕНТ ПОДПИСАН
ЭЛЕКТРОННОЙ ПОДПИСЬЮ

Сертификат: 12000002A633E3D113AD425FB50002000002A6

Владелец: Шебзухова Татьяна Александровна

Действителен: с 20.08.2021 по 20.08.2022