# Цель и задачи лабораторной работы

**Цель:** научиться использовать задачи продолжения в многопоточных программах.

**Задачи:**

1. Научиться формировать последовательность запуска задач;
2. Научиться использовать задачи продолжения для формирования последовательности выполняемых работ;
3. Научиться использовать задачи продолжения для реализации массового запуска задач.

# Реализация индивидуального задания

Согласно варианту задания, требуется использовать использованием механизма задач продолжения для решения задачи:

Генерация матрицы случайных чисел (размер определяется случайным образом)

Задачи продолжения: 3 задачи: расчет суммы четных элементов, делящихся на 4; поиск максимума среди элементов, делящихся на 3; поиск минимального элемента

## Листинг программного кода

***namespace*** lab9

{

***class*** Program

    {

**async** **static** Task<**int**[,]> GenerateRandomMatrix()

        {

**await** Task.Delay(1000);

            Random rand **=** **new**();

**int** rows **=** rand.Next(3, 8);

**int** cols **=** rand.Next(3, 8);

**int**[,] matrix **=** **new** **int**[rows, cols];

*for* (**int** i **=** 0; i **<** rows; i**++**)

            {

*for* (**int** j **=** 0; j **<** cols; j**++**)

                {

                    matrix[i, j] **=** rand.Next(1, 101);

                }

            }

*return* matrix;

        }

**static** **int** CalculateSumOfEvenDivisibleBy4(**int**[,] matrix)

        {

**int** sum **=** 0;

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

            {

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

                {

*if* (matrix[i, j] **%** 2 **==** 0 **&&** matrix[i, j] **%** 4 **==** 0)

                    {

                        sum **+=** matrix[i, j];

                    }

                }

            }

*return* sum;

        }

**static** **int** FindMaxDivisibleBy3(**int**[,] matrix)

        {

**int** max **=** **int**.MinValue;

**bool** found **=** false;

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

            {

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

                {

*if* (matrix[i, j] **%** 3 **==** 0)

                    {

*if* (**!**found **||** matrix[i, j] **>** max)

                        {

                            max **=** matrix[i, j];

                            found **=** true;

                        }

                    }

                }

            }

*return* found **?** max **:** **-**1;

        }

**static** **int** FindMinElement(**int**[,] matrix)

        {

**int** min **=** matrix[0, 0];

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

            {

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

                {

*if* (matrix[i, j] **<** min)

                    {

                        min **=** matrix[i, j];

                    }

                }

            }

*return* min;

        }

**static** **void** PrintMatrix(**int**[,] matrix)

        {

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

            {

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

                {

                    Console.Write($"{matrix[i, j],3} ");

                }

                Console.WriteLine();

            }

        }

**static** (**int**, **int**, **int**) GenerateAndPrintMatrix()

        {

***var*** taskMatrix **=** GenerateRandomMatrix();

***var*** postTaskPrint **=** taskMatrix.ContinueWith(antecedent **=>**

            {

***var*** matrix **=** antecedent.Result;

                Console.WriteLine($"Сгенерированная матрица: {matrix.GetLength(0)}x{matrix.GetLength(1)}");

                PrintMatrix(matrix);

            });

***var*** evenSumTask **=** taskMatrix.ContinueWith(antecedent **=>**

                CalculateSumOfEvenDivisibleBy4(antecedent.Result));

***var*** maxDivisibleBy3Task **=** taskMatrix.ContinueWith(antecedent **=>**

                FindMaxDivisibleBy3(antecedent.Result));

***var*** minElementTask **=** taskMatrix.ContinueWith(antecedent **=>**

                FindMinElement(antecedent.Result));

            Task.WaitAll(postTaskPrint, evenSumTask, maxDivisibleBy3Task, minElementTask);

*return* (evenSumTask.Result, maxDivisibleBy3Task.Result, minElementTask.Result);

        }

**static** **void** Main(**string**[] args)

        {

**int** evenSumTaskRes;

**int** maxDivisibleBy3TaskRes;

**int** minElementTaskRes;

            (evenSumTaskRes, maxDivisibleBy3TaskRes, minElementTaskRes) **=** GenerateAndPrintMatrix();

            Console.WriteLine($"\nРезультаты:");

            Console.WriteLine($"Сумма чётных элементов, делящихся на 4: {evenSumTaskRes}");

            Console.WriteLine($"Максимальный элемент, делящийся на 3: {maxDivisibleBy3TaskRes}");

            Console.WriteLine($"Минимальный элемент: {minElementTaskRes}");

        }

    }

}

## Описание кода

**Ключевые механизмы**

**1. Основная асинхронная задача**

* Генерирует матрицу случайного размера (3-7 × 3-7)
* Заполняет случайными числами 1-100
* Имитирует долгую операцию через Task.Delay(1000)

**2. Задачи продолжения через ContinueWith**

* **Автоматический запуск** после завершения основной задачи
* **Доступ к результату** через antecedent.Result
* **Параллельное выполнение** нескольких продолжений

**3. Три типа вычислений-продолжений**

* evenSumTask - сумма четных элементов, делящихся на 4
* maxDivisibleBy3Task - максимальный элемент, делящийся на 3
* minElementTask - минимальный элемент матрицы

**4. Синхронизация продолжений**

* Ожидание завершения всех задач продолжения
* Возврат кортежа с результатами всех вычислений

## Результат работы программы

Сгенерированная матрица: 5x6

3 15 89 74 63 91

28 4 66 21 54 44

70 71 31 9 5 85

88 49 1 35 39 54

28 95 7 68 33 52

Результаты:

Сумма чётных элементов, делящихся на 4: 312

Максимальный элемент, делящийся на 3: 66

Минимальный элемент: 1

# Контрольные вопросы

1. **Что такое задача продолжения? Для чего используется данный механизм?**
   1. Задача, которая автоматически запускается после завершения предыдущей (antecedent) задачи.
   2. **Назначение:**
      1. Создание цепочек зависимых асинхронных операций
      2. Упрощение кода без вложенных колбэков
      3. Организация workflow с последовательными шагами
2. **Сколько задач продолжения может иметь каждая задача?**
   1. Неограниченное количество.
3. **С помощью какого метода класса Task можно задать задачу продолжения?**
   1. ContinueWith()
4. **Какие параметры принимает метод, представляющий задачу продолжения?**
   1. ContinueWith(  
       Action<Task> continuationAction, // Задача  
       CancellationToken cancellationToken, // Токен отмены задачи  
       TaskContinuationOptions continuationOptions, // Опции отмены  
       TaskScheduler scheduler // Расписание  
      )
5. **Поясните механизм определения условных задач продолжения.**
   1. Создание продолжений только при определенных условиях через TaskContinuationOptions
6. **Опишите методы класса TaskFactory, предназначенные для указания задач продолжения.**
   1. После всех задач
      1. TaskFactory.ContinueWhenAll(Task[] tasks, Action<Task[]> continuationAction);
   2. После любой задачи
      1. TaskFactory.ContinueWhenAny(Task[] tasks, Action<Task> continuationAction);
   3. С типизированными задачами
      1. TaskFactory.ContinueWhenAll<TResult>(Task<TResult>[] tasks, Action<Task<TResult>[]> continuation);
7. **Опишите механизм, использующий задачи продолжения и позволяющий запустить большое количество задач (100, 1000, …) одной командой Start( ).**

*// Создание массива задач*

Task[] tasks **=** **new** Task[1000];

*for* (**int** i **=** 0; i **<** 1000; i**++**)

{

    tasks[i] **=** **new** Task(() **=>** DoWork(i));

}

*// Запуск всех одной командой*

*foreach* (***var*** task *in* tasks)

{

    task.Start();

}

*// Продолжение после ВСЕХ задач*

Task.Factory.ContinueWhenAll(tasks, completedTasks **=>**

{

    Console.WriteLine($"Все {completedTasks.Length} задач завершены!");

});