# Task 1

Файл **in.csv** (текст с разделителями «;») содержит строки с числами. Самый первый элемент строки имеет индекс 0. Его значение определят то, с каким элементом строки будет вестись работа.

1 Вычислить сумму значений из всех строк, на которые указывает первый элемент строки (с индексом 0).

2 Подсчитать и вывести кол-во строк с ошибками.

Пример 1 in,csv: Output:

```
3;qw;4;5,2;2,7          result(5,2 - 3,14 + 0,0) = 2,06
15;;;k;5                error-lines = 3
1;-3,14;fgh;5
0;;e1;2;3
-2,3;a;b;c
b;d;e
```

Пример 2 in,csv: Output:

```
3;qw;4;-3,1;2,7         result(-3,1 – 1,0) = -4,1
1;-1;fgh;5              error-lines = 0
```

Пример 3 in,csv: Output:

```
1;7,5e-1;2,7            result(0,75) = 0,75
                        error-lines = 0
```

Пример 4 in,csv: Output:

```
1;0                     result(0,0) = 0,0
                        error-lines = 0
```

Пример 5 in,csv: Output:

```
1;da                    result() = 0,0
                        error-lines = 1
```

Замечания к задаче 1

– Перевод строки в число можно осуществлять через метод Parse( ) классов Int32, Double.

– Вывод результата – на консоль.

– Информационных классов не создавать. Реализовать исключительно в методе Main( ) класса Program.

– В алгоритме запрограммировать ход работы программы в случае корректных данных, ошибочные ситуации (не число, выход за границы массива) обрабатывать через исключения.

– Строки csv–файла разбивать на элементы методом Split( ).

– Не использовать регулярные выражения.

– Не выводить строку с результатом отдельными фрагментами. Согласно условию нужно сформировать ответ в виде строки. Обратите внимание на наличие пробелов в выходной строке. В частности, если первое число отрицательное, то после знака минус нет пробела. Далее каждый знак (плюс, минус, равно) окружен пробелами. Для формирования строки нужно использовать класс StringBulder.

– Обязательно проверить тест, в котором отсутствует входной файл.

– Самый распространенный антипаттерн – magic numbers, т.е. константы (особенно повторяющиеся), смысл которых очевиден исключительно автору кода. Числа или строки, которые используются в алгоритме, обязательно нужно инициализировать в константных локальных переменных либо вообще выносить в отдельный класс статических констант.

## Task 2

Текстовый файл **in.txt** содержит текст на английском языке, в котором присутствуют:

– **корректные** даты в формате день**X**месяц**X**год, где день и месяц – одна или две цифры, год – две или четыре цифры. **X** – разделитель: точка, прямой слеш или минус (одинаковый в каждой дате, но может быть разным в разных датах).

– **корректные** денежные суммы в формате **сумма** belarusian roubles или **сумма** blr, где **сумма** – число, разбитое на разряды *произвольным* числом пробелов (например, 32  671 000  blr).

Программа должна сделать следующее:

1 Преобразовать даты в тексте в американский формат Month day, year (например, October 3, 2010).

2 Убрать пробелы между разрядами в суммах (например, 32671000 blr).

Результат записать в файл **out.txt**.

Пример in.txt

```
I was 2  300  530 belarusian roubles and 2  351 dollars 12/9/2010.
After shopping 15.09.10
I was left with 1 700 253 blr and 2 000$.
After shopping 16.09.10
I was left with 1 7  00    2 500 blr.
232     500 blr and 10     blr.
```

Результат out.txt

```
I was 2300530 belarusian roubles and
2  351 dollars September 12, 2010.
After shopping September 15, 2010
I was left with 1700253 blr and 2 000$.
After shopping September 16, 2010
I was left with 1 7  00    2500 blr.
232500 blr and 10 blr.
```

Замечания к задаче 2

– Тест не полный, добавить текст для полной проверки решения.

– Замену найденной регулярным выражением подстроки удобно производить с использованием класса MatchEvaluator.

– Как переводить дату в другой формат есть в ссылках в задании.

– Имейте в виду, что, например, текст 15/15/15 похож на дату, но ей не является. Текст такого вида должен остаться без изменений.

– Чтобы упростить использование регулярных выражений, используйте два символа:

1. \b – граница слова. Это даже не символ, а метка начала или конца слова. Ограничители – пробел, табуляция, начало/конец строки, начало/конец файла, точка, запятая, тире, … Они не попадают в сравниваемую/выделяемую последовательность;

2. \положительное_число – ссылка на найденную группу. Например, @"(\d)0\1" – после 0 та же цифра, что и перед 0.

– Алгоритм анализа и редактирования текста исходного файла состоит из двух циклов. Сначала строка прогоняется через первое регулярное выражение (даты), а затем через второе (деньги). Структура циклов одинаковая, и код отличается лишь отдельными строками.

Хорошее решение (в стиле ООП) предполагает создание абстрактного класса, в котором есть универсальный метод с реализацией цикла для поиска и замены в переданной строке по регулярному выражению. Детали обработки, уникальные для регулярного выражения, должны быть вынесены в подклассы.

## Task 3

Файл input.txt содержит корректный C# код. Написать программу, которая удаляет все комментарии из исходного файла и сохраняет его в output.txt.

Пример:

Input.txt

```
/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication3
{  /// class Program here
  class Program
  {
    static void Main(string[] args)
    {
      /// TODO code application logic here
      string ss="sdfsdf\"\\*sdfsdf*\\\"";
      int i /*jjjjj*/ = 0;
      string s = "\t";
      char c = '\t';
      char cc = '/';
      string str = "'";//COMMENT
      char ccc = '*';
      char cccc = '"'; //"hjhkjhkj
      Console.WriteLine("/*dfsdf*////****/**//**Hello, world!\\");/**/ //
      int ccccc = 100/'*';
      char f = '\\';
      char f1 = '\n';
      char f2 = '\"';
      "".ToString();
    }
  }
}
```

Output.txt

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication3
{
  class Program
  {
    static void Main(string[] args)
    {

      string ss="sdfsdf\"\\*sdfsdf*\\\"";
      int i  = 0;
      string s = "\t";
      char c = '\t';
      char cc = '/';
      string str = "'";
      char ccc = '*';
      char cccc = '"';
      Console.WriteLine("/*dfsdf*////****/**//**Hello, world!\\");
```

```
        int ccccc = 100/'*';
        char f = '\\';
        char f1 = '\n';
        char f2 = '\"';
        "".ToString();
    }
  }
}
```

Обработку достаточно реализовать в методе Main( ).

– из примера видно, что нужно выделить три типа текста в исходном тексте:

1 строки – они беспрепятственно переносятся в результат, независимо, есть ли в них текст, совпадающий по синтаксису с комментариями, нужно только учесть, что в строках могут быть символы `\"`.

2 комментарии `//` и до конца строки – они заменяются от `//` и до конца строки на перевод строки `\r\n`.

3 комментарии `/*` … `*/` – они удаляются.

Разработанное регулярное выражение обязательно прокомментируйте, какая часть выражения для чего предназначена.