Лабораторная работа 2.

Работа с текстовыми файлами.

В рамках Framework .NET, независимо от характеристик того или иного устройства ввода/вывода, программист ВСЕГДА может узнать:

- можно ли читать из потока – `bool CanRead` (если можно – значение должно быть установлено в `true`);
- можно ли писать в поток – `bool CanWrite` (если можно – значение должно быть установлено в `true`);
- можно ли задать в потоке текущую позицию – `bool CanSeek` (если последовательность, в которой производится чтение/запись, не является жестко детерминированной и возможно позиционирование в потоке – значение должно быть установлено в `true`);
- позицию текущего элемента потока – `long Position` (возможность позиционирования в потоке предполагает возможность программного изменения значения этого свойства);
- общее количество символов потока (длину потока) – `long Length`.

В соответствии с общими принципами реализации операций ввода/вывода, для потока предусмотрен набор методов, позволяющих реализовать:

- чтение байта из потока с возвращением целочисленного представления СЛЕДУЮЩЕГО ДОСТУПНОГО байта в потоке ввода – `int ReadByte();`
- чтение определенного (`count`) количества байтов из потока и размещение их в массиве `buff`, начиная с элемента `buff[index]`, с возвращением количества успешно прочитанных байтов – `int Read(byte[] buff, int index, int count)`;
- запись в поток одного байта – `void WriteByte(byte b);`
- запись в поток определенного (`count`) количества байтов из массива `buff`, начиная с элемента `buff[index]`, с возвращением количества успешно записанных байтов – `int Write(byte[] buff, int index, int count)`;
- позиционирование в потоке – `long Seek (long index, SeekOrigin origin)` (позиция текущего байта в потоке задается значением смещения `index` относительно позиции `origin`);
- для буферизованных потоков принципиальна операция флэширования (записи содержимого буфера потока на физическое устройство) – `void Flush();`

закрытие потока – `void Close()`.

свойства и интерфейсы) на абстрактном классе `Stream`. При этом классы конкретных потоков обеспечивают собственную реализацию интерфейсов этого абстрактного класса.

Наследниками класса `Stream` являются, в частности, три класса байтовых потоков:

- `BufferedStream` – обеспечивает буферизацию байтового потока. Как правило, буферизованные потоки являются более производительными по сравнению с небуферизованными;

- `FileStream` – байтовый поток, обеспечивающий файловые операции ввода/вывода;
- `MemoryStream` – байтовый поток, использующий в качестве источника и хранилища информации оперативную память.

Манипуляции с потоками предполагают НАПРАВЛЕННОСТЬ производимых действий. Информацию из потока можно ПРОЧИТАТЬ, а можно ее в поток ЗАПИСАТЬ. Как чтение, так и запись предполагают реализацию определенных механизмов байтового обмена с устройствами ввода/вывода.

Свойства и методы, объявляемые в соответствующих классах, определяют специфику потоков, используемых для чтения и записи:

- `TextReader`,
- `TextWriter`.

Эти классы являются абстрактными. Это означает, что они не "привязаны" ни к какому конкретному потоку. Они лишь определяют интерфейс (набор методов), который позволяет организовать чтение и запись информации для любого потока.

В частности, в этих классах определены следующие методы, определяющие базовые механизмы символьного ввода/вывода. Для класса `TextReader`:

- `int Peek()` – считывает следующий знак, не изменяя состояние средства чтения или источника знака. Возвращает следующий доступный знак, не считывая его в действительности из потока входных данных;
- `int Read(...)` – несколько одноименных перегруженных функций с одним и тем же именем. Читает значения из входного потока. Вариант `int Read()` предполагает чтение из потока одного символа с возвращением его целочисленного эквивалента или `-1` при достижении конца потока. Вариант `int Read (char[] buff, int index, int count)` и его полный аналог `int ReadBlock(char[] buff, int index, int count)` обеспечивает прочтение максимально возможного количества символов из текущего потока и записывает данные в буфер, начиная с некоторого значения индекса;
- `string ReadLine()` – читает строку символов из текущего потока. Возвращается ссылка на объект типа `string`;
- `string ReadToEnd()` – читает все символы, начиная с текущей позиции символьного потока, определяемого объектом класса `TextReader`, и возвращает результат как ссылка на объект типа `string`;
- `void Close()` – закрывает поток ввода.

Для класса `TextWriter`, в свою очередь, определяется:

- множество перегруженных вариантов функции `void Write(...)` со значениями параметров, позволяющих записывать символьное представление значений базовых типов (смотреть список базовых типов) и массивов значений (в том числе и массивов строк);
- `void Flush()` – обеспечивает очистку буферов вывода. Содержимое буферов выводится в выходной поток;
- `void Close()` – закрывает поток вывода.

Эти классы являются базовыми для классов:

- StreamReader – содержит свойства и методы, обеспечивающие считывание СИМВОЛОВ из байтового потока,
- StreamWriter – содержит свойства и методы, обеспечивающие запись СИМВОЛОВ в байтовый поток.

Вышеуказанные классы включают методы своих "предков" и позволяют осуществлять процессы чтения-записи непосредственно из конкретных байтовых потоков. Работа по организации ввода/вывода с использованием этих классов предполагает определение соответствующих объектов, "ответственных" за реализацию операций ввода/вывода, с явным указанием потоков, которые предполагается при этом использовать.

Еще одна пара классов потоков обеспечивает механизмы символьного ввода– вывода для строк:

- StringReader,
- StringWriter.

В этом случае источником и хранилищем множества символов является символьная строка.

Интересно заметить, что у всех ранее перечисленных классов имеются методы, обеспечивающие закрытие потоков, и не определены методы, обеспечивающие открытие соответствующего потока. Потоки открываются в момент создания объекта-представителя соответствующего класса. Наличие функции, обеспечивающей явное закрытие потока, принципиально. Оно связано с особенностями выполнения управляемых модулей в Framework .NET. Время начала работы сборщика мусора заранее не известно.

Приведен пример программы по чтению/записи файла.

```
            private void menuItem2_Click(object sender, System.EventArgs e)
            {

openFileDialog1.ShowDialog() ;
string fileName = openFileDialog1.FileName;

FileStream stream = File.Open(fileName, FileMode.Open, FileAccess.Read);
                  if(stream != null)
                  {
                        StreamReader reader = new StreamReader(stream);
                        textBox1.Text = reader.ReadToEnd ();
                        stream.Close();
                              }
                  }

            private void menuItem3_Click(object sender, System.EventArgs e)
            {
                  saveFileDialog1.ShowDialog();
                  string fileName = saveFileDialog1.FileName;
                  FileStream stream = File.Open(fileName, FileMode.Create,
FileAccess.Write);
                  if(stream != null)
                  {
                        StreamWriter writer = new StreamWriter(stream);
```

```
                        writer.Write(textBox1.Text);
                        writer.Flush();
                        stream.Close();
                    }
                    }
        }
}
```

Задание.

1. Подсчитать количество чисел в тестовом файле и записать это значение в файл. (Текстовый файл отдельный на каждую бригаду).
2. Найти максимальное и минимальное значения.
3. Вывести на экран встречающиеся в тестовых файлах слова.