ОРГАНИЗАЦИЯ СИСТЕМЫ ВВОДА-ВЫВОДА В ЯЗЫКЕ C#

Иерархия потоков

Программы, написанные на языке C#, выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов. *Поток* (stream) – это абстракция, которая генерирует или принимает данные. С помощью потока можно читать данные из различных источников (клавиатура, файл), или записывать данные в различные источники (принтер, экран, файл). Несмотря на то, что потоки связываются с различными физическими устройствами, характер поведения всех потоков одинаков.

На самом низком уровне иерархии потоков ввода-вывода находятся байтовые потоки. Это объясняется тем, что многие устройства при выполнении операций ввода-вывода ориентированы на работу с «сырыми» байтами. Однако для человека привычнее оперировать символами, поэтому разработаны символьные потоки, которые фактически представляют собой оболочки, выполняющие преобразование байтовых потоков в символьные и наоборот.

Кроме этого, реализованы оболочки для работы со значениями размерных типов, которые выполняют преобразование байтовых потоков в целочисленные, вещественные и т.д. В действительности эти оболочки работают не с самими значениями, а с их внутренним представлением в виде двоичных кодов, поэтому они также называются двоичными.

Программист может создать собственные потоковые классы. Однако для подавляющего большинства приложений достаточно стандартных потоков.

Центральную часть потоковой системы языка C# занимает класс Stream пространства имен System.IO. Класс Stream представляет собой байтовый поток и является базовым для всех остальных потоковых классов. Из класса Stream выведен такой класс как FileStream, разработанный для организации файлового ввода-вывода. Для преобразования байтового потока в символьный разработаны такие классы как StreamWriter и StreamReader, представляющие собой оболочки для класса FileStream.

Далее мы подробно рассмотрим работу с классами FileStream, StreamWriter и StreamReader.

Байтовый поток

Чтобы создать байтовый поток, связанный с файлом, создается объект класса FileStream. В данном классе определено несколько конструкторов. Чаще всего используется конструктор, который открывает поток для чтения и/или записи:

```
FileStream(string filename, FileMode mode)
```

где:

1) параметр filename определяет имя файла, с которым будет связан поток ввода-вывода данных;
2) параметр mode определяет режим открытия файла, который может принимать одно из возможных значений, определенных перечислением FileMode:
    1) FileMode.Append – предназначено для добавления данных в конец файла;
    2) FileMode.Create – предназначено для создания нового файла, причем если существует файл с таким же именем, то он будет предварительно удален;
    3) FileMode.CreateNew – предназначено для создания нового файла, при этом файл с таким же именем не должен существовать;
    4) FileMode.Open – предназначено для открытия существующего файла;

5) FileMode.OpenOrCreate – если файл существует, то его необходимо открыть; в противном случае – создать новый;
6) FileMode.Truncate – предназначено для открытия существующего файла, с полным уничтожением его содержимого.

Если попытка открыть файл оказалась неуспешной, то генерируется исключение. Например, исключение FileNotFoundException говорит о том, что файл невозможно открыть по причине его отсутствия, ArgumentNullException – имя файла представляет собой null-значение, ArgumentOutOfRangeException – некорректен параметр mode, SecurityException – пользователь не обладает правами доступа, DirectoryNotFoundException – некорректно задан каталог.

Другая версия конструктора позволяет ограничить доступ к файлу только чтением, или только записью:

```
FileStream(string filename, FileMode mode, FileAccess how)
```

где:

1) параметры filename и mode имеют то же назначение, что и в предыдущей версии конструктора;
2) параметр how определяет способ доступа к файлу и может принимать одно из значений, определенных перечислением FileAccess:
   1) FileAccess.Read – только чтение;
   2) FileAccess.Write – только запись;
   3) FileAccess.ReadWrite – и чтение, и запись.

После установления связи байтового потока с физическим файлом внутренний указатель потока устанавливается на начальный байт файла.

Для чтения очередного байта из потока, связанного с физическим файлом, используется метод ReadByte(). После прочтения очередного байта внутренний указатель перемещается на следующий байт файла. Если достинут конец файла, то метод ReadByte() возвращает значение минус один.

Для побайтовой записи данных в поток используется метод WriteByte().

По завершении работы с файлом его необходимо закрыть. Для этого достаточно вызвать метод Close(). При закрытии файла освобождаются системные ресурсы, ранее выделенные для этого файла, что дает возможность использовать их для работы с другими файлами.

Рассмотрим пример использования класса FileStream для копирования одного файла в другой, но вначале создадим текстовый файл text.txt в папке текущего проекта. Это можно сделать, например, с помощью Блокнота, или с помощью VS. В нашем случае тестовый файл будет создан в папке d:/Example и полное имя файла будет выглядеть следующим образом: d:/Example/text.txt

Внесем в файл text.txt произвольную информацию, например:

```
12    456
Hello! Привет!!!
23,67  4: Message
```

Затем выполним следующую программу:

```
using System;
//подключаем пространство имен для работы с потоками
using System.IO;
namespace MyProgram
{
   class Program
   {
      static void Main()
      {
         FileStream fileIn = new FileStream("d:/Example/text.txt",
                                            FileMode.Open,
                                            FileAccess.Read);
         FileStream fileOut =
                           new FileStream("d:/Example/newText.txt",
                                          FileMode.Create,
                                          FileAccess.Write);
         int i;
         //читаем поток fileIn побайтно до тех пор,
         //пока не достигнем конца файла
         while ((i = fileIn.ReadByte()) != -1)
         {
            //записываем очередной байт в поток fileOut
            fileOut.WriteByte((byte)i);
         }
         fileIn.Close();
         fileOut.Close();
      }
   }
}
```

В результате выполнения данной программы в папке d:/Example создастся новый файл newText.txt, содержимое которого будет соответствовать файлу text.txt. Фактически, мы скопировали содержимое файла text.txt в файл newText.txt.

В рассмотренном примере сознательно допущена серьезная ошибка. Представьте себе ситуацию, при которой файл "d:/Example/newText.txt" невозможно открыть для записи, например, этот файл имеет атрибут «только для чтения». В этом случае, при запуске программы будет открыт файл "Text.txt", после чего программа попытается открыть файл "newText.txt" и произойдет исключение. При этом, так как перехват исключения не произведен (подробнее процесс обработки исключений будет рассмотрен позднее), программа завершит свою работу. Весь оставшийся код, включая вызов метода Close для потока fileIn, выполнен не будет и выделенные данному потоку системные ресурсы освободятся только спустя некоторое заранее неопределенное время. Если какая-то другая программа попытается открыть этот файл, она, возможно, не сможет этого сделать. Аналогичная ситуация произойдет, если мы по какой-либо причине забудем вызвать метод Close явно.

Чтобы решить эту проблему, в .NET используется оператор using. Данный оператор имеет вид:

```
using(контролируемый_ресурс)
{
     действия
}
```

Ресурс, объявленный внутри круглых скобок оператора using, будет существовать только внутри фигурных скобок этого оператора и будет гарантированно освобожден после выхода из них. Более подробно устройство оператора using будет рассмотрено в разделе, посвященном обработке исключений.

Правильный вариант примера будет иметь вид:

```
using System;
using System.IO;
namespace MyProgram
{
   class Program
   {
      static void Main()
      {
         using (FileStream fileIn =
                        new FileStream("d:/Example/text.txt",
                                  FileMode.Open, FileAccess.Read))
         {
            using (FileStream fileOut =
                           new FileStream("d:/Example/newText.txt",
                                 FileMode.Create, FileAccess.Write))
            {
               int i;
               while ((i = fileIn.ReadByte()) != -1)
               {
                  fileOut.WriteByte((byte)i);
               }
            }
         }
      }
   }
}
```


Задания

1. Подумайте, почему для переменной i *указан тип* int. *Можно ли было объявить типом данной переменной* byte*?*

2. Преобразуйте предложенный алгоритм копирования содержимого одного файла в другой в самостоятельный метод и, с помощью этого метода, выполните копирование файла f1 в файл f2, файла f2 в файл f3, файла f3 в файл f1.

Символьный поток

Чтобы создать символьный поток, нужно поместить объект класса FileStream «внутрь» объекта класса StreamWriter, или объекта класса StreamReader. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс StreamWriter предназначен для организации выходного символьного потока. В данном классе определено несколько конструкторов, один из поторых записывается следующим образом:

```
StreamWriter(Stream stream);
```

где параметр stream определяет имя уже открытого байтового потока. Например, создать экземпляр класса StreamWriter можно следующим образом:

```
FileStream file = new FileStream("d:/Example/text.txt",
                        FileMode.Create, FileAccess.Write);
StreamWriter fileOut = new StreamWriter(file));
```

Другой вид конструктора позволяет открыть поток сразу через обращение к файлу:

```
StreamWriter(string name);
```

где параметр name определяет имя открываемого файла. Например, обратиться к данному конструктору можно следующим образом:

```
StreamWriter fileOut = new StreamWriter("d:/Example/text.txt");
```

Еще один вариант конструктора StreamWriter имеет вид:

```
StreamWriter(string name, bool appendFlag);
```

где параметр name определяет имя открываемого файла, а параметр appendFlag может принимать значение true, если нужно добавлять данные в конец файла, или false, если файл необходимо перезаписать. Например, обратиться к данному конструктору можно следующим образом:

```
StreamWriter fileOut=new StreamWriter("d:/Example/text.txt", true);
```

После создания объекта, для записи данных в поток fileOut можно обратиться к методу WriteLine. Это можно сделать следующим образом:

```
fileOut.WriteLine("test");
```

В данном случае, в конец файла t.txt будет дописано слово test.

Задание

Используя дополнительную литературу, определите, какие исключения могут возникнуть при обращении к конструктору StreamWriter.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
StreamReader(Stream stream);
```

где параметр stream определяет имя уже открытого байтового потока. Например, создать экземпляр класса StreamReader можно следующим образом:

```
FileStream file = new FileStream("d:/Example/text.txt",
                           FileMode.Open, FileAccess.Read);
StreamReader fileIn = new StreamReader(file);
```

Как и в случае с классом StreamWriter, у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
StreamReader (string name);
```

где параметр name определяет имя открываемого файла. Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn = new StreamReader ("d:/Example/text.txt");
```

В C# символы записываются в кодировке Unicode. Для того чтобы можно было обрабатывать текстовые файлы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора StreamReader:

```
StreamReader fileIn = new StreamReader ("d:/Example/text.txt",
                                        Encoding.GetEncoding(1251));
```

где параметр Encoding.GetEncoding(1251) говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode. Если не использовать данный параметр, то возникнут проблемы при отображении русских символов. Класс Encoding определен в пространстве имен System.Text.

После того, как файл будет открыт, можно воспользоваться методом ReadLine для чтения данных из потока fileIn. Данный метод читает из файла очередную строку. Если будет достигнут конец файла, то метод ReadLine вернет значение null.

Рассмотрим пример, в котором данные из одного файла копируются в другой, но уже с использованием классов StreamWriter и StreamReader.

```
using System;
using System.Text;
using System.IO;
namespace MyProgram
{
   class Program
   {
      static void Main()
      {
         using (StreamReader fileIn =
                     new StreamReader("d:/Example/text.txt",
                                      Encoding.GetEncoding(1251)))
         {
            using (StreamWriter fileOut =
                   new StreamWriter("d:/Example/newText.txt", false))
            {
               string line;
               //читаем построчно до тех пор, пока поток
               //fileIn не пуст
               while ((line = fileIn.ReadLine()) != null)
               {
                  //записываем данные в выходной поток
                  fileOut.WriteLine(line);
               }
            }
         }
      }
   }
}
```

Задание

Выясните, для чего предназначен метод ReadToEnd() и когда имеет смысл его применять.

Данный способ копирования одного файла в другой даст нам тот же результат, что и при использовании байтовых потоков. Однако его работа будет менее эффективной, т.к. будет тратиться дополнительное время на преобразование байтов в символы.

Для простых операций типа копирования это может быть серьезным недостатком, но у символьных потоков есть свои преимущества. Символьные потоки удобно использовать тогда, когда необходимо анализировать информацию, хранящуюся в текстовом файле.

Пример

Даны текстовые файлы input.txt и output.txt. Переписать из файла input.txt в файл output.txt все слова, начинающиеся и заканчивающиеся на одну и ту же букву.

```
using System;
using System.Text;
using System.IO;
namespace MyProgram
{
   class Program
   {
      static void Main()
      {
         using (StreamReader fileIn =
                   new StreamReader("d:/Example/text.txt",
                                    Encoding.GetEncoding(1251)))
         {
            using (StreamWriter fileOut =
                     new StreamWriter("d:/Example/newText.txt",
                                      false))
            {
               string line = fileIn.ReadToEnd();
               StringBuilder a = new StringBuilder(line);
               // удаляем из строк все знаки пунктуации,
               // а также заменяем в ней все знаки
               // табуляции и перевод строки на пробел
               for (int i = 0; i < a.Length; )
               {
                  if (char.IsPunctuation(a[i]))
                  {
                     a.Remove(i, 1);
                  }
                  else
                  {
                     if (char.IsWhiteSpace(a[i]))
                     {
                        a[i] = ' ';
                     }
                     ++i;
                  }
               }
```

```
                string[] s = a.ToString().Trim().Split(' ',
                          StringSplitOptions.RemoveEmptyEntries);
                Console.WriteLine("Искомые слова: ");
                foreach (string str in s)
                {
                   if (str[0] == str[str.Length - 1])
                   {
                      fileOut.WriteLine(str);
                   }
                }
             }
          }
       }
    }
```

Результат работы программы:

input.txt

```
кол около колокола,
а шалаш у ручья…
```

output.txt

```
кол
около
а
шалаш
```

В текстовых файлах может храниться не только символьная информация, но и числовая, структурированная некоторым образом. Например, в текстовом файле может храниться двумерный массив. Рассмотрим, как можно считать двумерный массив из файла:

```
using System;
using System.IO;
using System.Text;
namespace MyProgram
{
   class Program
   {
      static void Main()
      {
         using (StreamReader file =
                   new StreamReader("d:/Example/text.txt"))
         {
            string line = file.ReadLine();
            string[] mas = line.Split(' ');
            int n = int.Parse(mas[0]);
            int m = int.Parse(mas[1]);
            int[,] MyArray = new int[n, m];
            for (int i = 0; i < n; i++)
            {
               line = file.ReadLine();
               mas = line.Split(' ',
                       StringSplitOptions.RemoveEmptyEntries);
```

```
                for (int j = 0; j < m; j++)
                {
                   MyArray[i, j] = int.Parse(mas[j]);
                }
             }
             Console.WriteLine("из файла прочитан массив:");
             Print(MyArray);
          }
       }
       static void Print(int[,] mas)
       {
          Console.WriteLine("{0} {1}", mas.GetLength(0),
                                       mas.GetLength(1));
          for (int i = 0; i < mas.GetLength(0); i++)
          {
             for (int j = 0; j < mas.GetLength(1); j++)
             {
                Console.Write("{0} ", mas[i, j]);
             }
             Console.WriteLine();
          }
       }
    }
}
```

Результат работы программы:

input.txt

```
3 4
1 2 3 4
5 6 7 8
9 0 1 2
```

Сообщение на экране

```
Из файла прочитан массив:
3 4
1 2 3 4
5 6 7 8
9 0 1 2
```

Задание

Измените программу так, чтобы из файла читался ступенчатый массив.

Мы рассмотрели примеры, в которых данные из файла читались построчно, или файл прочитывался полностью. Но бывают случаи, когда необходимо читать текстовый файл посимвольно. Например, нам необходимо подсчитать сколько раз встречается заданный символ в текстовом файле. Это можно сделать следующим образом:

```
using System;
using System.IO;
namespace MyProgram
{
   class Program
   {
```

```
        static void Main()
        {
           using (StreamReader fileIn =
                       new StreamReader("d:/Example/text.txt"))
           {
              char x = '@';
              int ch;
              int k = 0;
              //читаем поток посимвольно, до тех пор пока он не пуст
              while ((ch = fileIn.Read()) != -1)
              {
                 if ((char)ch == x)
                 {
                    k++;
                 }
              }
              Console.WriteLine(k);
           }
        }
     }
  }
```

Результат работы программы:

input.txt
Иванов А.С. ivanov@mail.ru
Петров В.Г. petrov@sgu.ru
Сергеев С.Ф.
Сообщение на экране
2

Практикум №9

Задание 1

1. Дан текстовый файл. Найти количество строк, которые начинаются с данной буквы.
2. Дан текстовый файл. Найти количество строк, которые начинаются и заканчиваются одной буквой.
3. Дан текстовый файл. Найти самую длинную строку и ее длину.
4. Дан текстовый файл. Найти самую короткую строку и ее длину.
5. Дан текстовый файл. Найти номер самой длинной строки.
6. Дан текстовый файл. Найти номер самой короткой строки.
7. Дан текстовый файл. Выяснить, имеется ли в нем строка, которая начинается с данной буквы. Если да, то напечатать ее.
8. Дан текстовый файл. Напечатать первый символ каждой строки.
9. Дан текстовый файл. Напечатать символы с k1 по k2 в каждой строке.
10. Дан текстовый файл. Напечатать все нечетные строки.
11. Дан текстовый файл. Напечатать все строки, в которых имеется хотя бы один пробел.
12. Дан текстовый файл. Напечатать все строки, длина которых равна данному числу.
13. Дан текстовый файл. Напечатать все строки, длина которых меньше заданного числа.
14. Дан текстовый файл. Напечатать все строки с номерами от k1 до k2.
15. Дан текстовый файл. Получить слово, образованное символами с номером k в каждой строке.

16. Дан текстовый файл. Переписать в новый файл все его строки, вставив в конец каждой строки ее номер.
17. Дан текстовый файл. Переписать в новый файл все его строки, вставив в конец каждой строки количество символов в ней.
18. Дан текстовый файл. Переписать в новый файл все его строки, длина которых больше заданного числа.
19. Дан текстовый файл. Переписать в новый файл все его строки четной длины.
20. Дан текстовый файл. Переписать в новый файл все его строки, удалив из них символы, стоящие на четных местах.

Задание 2

1. Дан файл f, компонентами которого являются целые числа. Переписать все четные числа в файл g, нечетные – в файл h.
2. Дан файл f, компонентами которого являются целые числа. Переписать все отрицательные числа в файл g, положительные – в файл h.
3. Даны два файла с числами. Поменять местами их содержимое (использовать вспомогательный файл).
4. Даны два файла с числами. Получить новый файл, каждый элемент которого равен сумме соответствующих компонентов заданных файлов (количество компонентов в исходных файлах одинаковое).
5. Даны два файла с числами. Получить новый файл, каждый компонент которого равен наибольшему из соответствующих компонентов заданных файлов (количество компонентов в исходных файлах одинаковое).
6. Даны два файла с числами. Получить новый файл, каждый компонент которого равен среднему арифметическому значению соответствующих компонентов заданных файлов (количество компонентов в исходных файлах одинаковое).
7. Даны два файла с числами. Получить новый файл, записав в него сначала все положительные числа из первого файла, потом все отрицательные числа из второго.
8. Даны два файла с числами. Получить новый файл, записав в него сначала все четные числа из первого файла, потом все нечетные числа из второго.
9. Даны два файла с числами. Получить новый файл, в котором на четных местах будут стоять компоненты, которые стоят на четных местах в первом файле, а на нечетных – компоненты, которые стоят на нечетных во втором (количество компонентов в исходных файлах одинаковое).
10. Дан файл f, компонентами которого являются символы. Переписать в файл g все знаки препинания файла f, а в файл h – все остальные символы файла f.
11. Дан файл f, элементами которого являются символы. Переписать в файл g все цифры файла f, а в файл h – все остальные символы файла f.
12. Даны два символьных файла. Выяснить, совпадают ли символы в файлах попарно. Если нет, получить номер первого элемента, в котором эти файлы отличаются.
13. Дан файл, компонентами которого являются целые числа. Переписать в новый файл сначала все отрицательные компоненты из первого, потом все положительные.
14. Дан файл, компонентами которого являются символы. Создать новый файл таким образом, чтобы на четных местах у него стояли компоненты, стоящие на нечетных в первом файле, и наоборот.
15. Дан файл, компонентами которого являются числа. Число компонент файла делится на два. Создать новый файл, в который будет записываться наименьшее из каждой пары чисел первого файла.

16. Дан файл, компонентами которого являются числа. Число компонент файла является четным. Создать новый файл, в который будет записываться среднее арифметическое из каждой пары чисел первого файла.
17. Дан файл, компонентами которого являются символы. Переписать все символы в новый файл в обратном порядке.
18. Даны два файла с одинаковым количеством компонент, компонентами которых являются натуральные числа. Создать новый файл, в который будут записываться числа по следующему правилу: берется первое число из первого файла и первое из второго; если одно из них делится нацело на другое, то их частное записывается в новый файл; затем берется второе число из первого и второе число из второго и т.д.
19. Дан файл, компонентами которого являются символы. Переписать в новый файл все символы, которым в первом файле предшествует данная буква.
20. Дан файл, компонентами которого являются символы. Переписать в новый файл все символы, за которыми в первом файле следует данная буква.

Самостоятельная работа №6

Используя Интернет и дополнительную литературу:

1) изучите возможности использования байтовых потоков: классы BufferedStream и MemoryStream;
2) изучите возможности использования двоичных потоков: классы BinaryWriter и BinaryReader.