**Лекция 13. Файлы с числовой информацией**

**Цель лекции:** Формирование у студентов знаний, связанных с работой с файлами с числовой информацией.

В результате изучения данной лекции студенты будут способны:

* Продемонстрировать знание способов записи и чтения из файлов числовой информации

Как известно, для долговременного хранения данных в компьютерах используются магнитные диски. Данные на дисках хранятся в виде файлов. *Файл* представляет собой единицу хранения данных на магнитных дисках компьютера, имеет уникальное имя в пределах дискового каталога. Файлы данных удобно хранить либо в том же самом каталоге, в котором находится исполняемый файл нашей программы. В этом случае при доступе к файлу достаточно указать лишь имя файла и не надо указывать каталог, содержащий файл.

При работе с файлами необходимо подключить пространство имен, в котором описываются стандартные классы для работы с файлами:

Using System.IO.

Выполнять обмен с внешними устройствами можно на уровне:

 двоичного представления данных (классы BinaryReader, BinaryWriter);

 байтов (класс FileStream);

 текста, то есть символов (классы StreamWriter, StreamReader).

В .NET используется кодировка Unicode, в которой каждый символ кодируется двумя байтами. Классы, работающие с текстом, являются оболочками классов, использующих байты, и автоматически выполняют перекодирование из байтов в символы и обратно.

Двоичные и байтовые потоки хранят данные в том же виде, в котором они представлены в оперативной памяти, то есть при обмене с файлом происходит побитовое копирование информации. Двоичные файлы применяются не для просмотра их человеком, а для использования в программах.

Доступ к файлам может быть *последовательным*, когда очередной элемент можно прочитать (записать) только после аналогичной операции с предыдущим элементом, и *произвольным*, или *прямым*, при котором выполняется чтение (запись) произвольного элемента по заданному адресу. Текстовые файлы позволяют выполнять только последовательный доступ, в двоичных и байтовых потоках можно использовать оба метода.

Для того чтобы использовать в программе файлы, необходимо:

1. Подключить пространство имен, в котором описываются стандартные классы для работы с файлами (using System.IO).

2. Создать файловый поток и связать его с физическим файлом.

3. Произвести операции обмена (ввод-вывод).

4. Закрыть файл

**Открытие и закрытие файла**

Для того, чтобы начать работать с файлом, нужно *открыть файловый поток*, создав для этого *объект библиотечного класса FileStrem* (определен в пространстве имен System.IO, поэтому в самом начале программы вводится строка using System.IO;).

FileStream("FileName", FileMode режим, FileAccess доступ)

где

"FileName" обозначает имя открываемого файла, включая и полный путь к нему;

режим – порядок открытия файла (FileMode.Open– открывает существующий файл,

FileMode.Create – создает новый выходной файл. Существующий файл с таким же именем будет разрушен)

доступ – обозначает конкретный способ доступа к файлу (FileAccess.Read, FileAccess.Write, FileAccess.ReadWrite)

*Пример*

FileStream fs = new FileStream("file1.dat", FileMode.Open, FileAccess.Read);

открывается существующий файл для чтения

После успешного открытия файла с ним можно выполнять операции чтения / записи данных.

При завершении работы с файлом его следует закрыть, вызвав метод Close():

fs.Close();

Если попытка открыть файл оказалась неудачной, то генерируется исключение FileNotFoundException и программа принудительно завершает работу. А если файл нельзя открыть из-за какой-нибудь ошибки ввода-вывода, то генерируется исключение IOException. Отсюда вытекает правило – *проверять факт существования файла* перед созданием объекта типа FileStream с помощью статического метода Exists() библиотечного класса File:

if (File.Exists (“FileName»))

{

…// здесь и создаем объект типа FileStream

}

Если файл с указанным именем не существует, а мы попытаемся открыть его на чтение, то результат будет отрицательным (в смысле возврата логического значения false методом Exists() класса File). В то же время, попытка *открыть файл на запись* (в режимах FileMode.Create и FileAccess.Write) всегда будет успешной: при существующем файле его прежнее содержимое аннулируется, а при несуществующем файле он автоматически создается.

*Пример*

FileStream fs = new FileStream("file1.dat", FileMode.Create, FileAccess.Write);

Мы открываем на запись файл с именем file1.dat, которого ранее не существовало.

Если при открытии файла путь к файлу не указан, то файл данных ищется или создается в том же каталоге, где находится исполняемый файл программы:

D:\ ProgramVS\ C95\ C95\ bin\ Debug\ file1.dat

При записи в тексте программы пути к файлу следует помнить, что символ \ в строковых константах обозначает начало специальной последовательности символов, например, \n. Поэтому при записи имени файла символ \ следует продублировать, например

“d:\\C#\\input.txt»

**Файловое хранение числовых данных**

Для чтения / записи числовых данных типа int или double лучше создать объекты классов BinaryWriter и BinaryReader, чьи методы Write() (перегружен для разных типов данных), ReadInt32() и ReadDouble() как нельзя лучше подходят для этой задачи.

Проиллюстрируем работу методов Write(), ReadInt32() (для чтения целых чисел типа int) и ReadDouble()(для чтения вещественных чисел типа double) на примере следующей программы.

**Задача 13.1.** Иллюстрация работы методов Write(), ReadInt32() (для чтения целых чисел типа int) и ReadDouble() (для чтения вещественных чисел типа double).

*Программный код:*

using System;

using System.IO;

namespace C95

{

class Program

{

static void Main(string[] args)

{

int m = 5;

double d = 8.34;

int iRes = 0;

double dRes = 0;

// Запись в файл

FileStream fsW = new FileStream("file1", FileMode.Create,

FileAccess.Write);

BinaryWriter bw = new BinaryWriter(fsW);

bw.Write(m);

bw.Write(d);

bw.Close();

fsW.Close();

// Чтение файла

if (File.Exists("file1"))

{

FileStream fsr = new FileStream("file1", FileMode.Open,

FileAccess.Read);

BinaryReader br = new BinaryReader(fsr);

iRes = br.ReadInt32();

dRes = br.ReadDouble();

br.Close();

fsr.Close();

}

Console.WriteLine("iRes i= {0}; dRes = {1}", iRes, dRes);

Console.ReadLine();

}

}

}

Результаты расчета см. рис. 13.1.



Рисунок 13.1 – Результаты решения задачи 13.1

В этой программе файл с именем file1 сначала открывается на запись, в результате чего объектная ссылка fsW типа FileStream получает актуальное значение. Поверх этой ссылки строится объект bw класса BinaryWriter:

BinaryWriter bw = new BinaryWriter(fsW);

С помощью перегруженного метода Write() класса BinaryWriter в открытый файл последовательно записываются значения переменных m и d. Затем пишущий объект и файл закрываются одноименными методами Close().

После этого файл снова открывается, но уже на чтение. Содержимое файла file1 вычитывается от имени объекта br типа BinaryReader методами ReadInt32() и ReadDouble() в переменные iRes и dRes, соответственно:

iRes = br.ReadInt32();

dRes = br.ReadDouble();

Для проверки результата значения переменных iRes и dRes выводятся в консольное окно программы с помощью статического метода WriteLine библиотечного класса Console.

**Задача 13.2.** Запись в файл и чтение из файла массивов

*Программный код:*

using System;

using System.IO;

namespace C96

{

class Program

{

static void Main(string[] args)

{

int[] iArr = {1,5,7,10,12 };

double[] dArr = {1.34, 3.56, 5.12, 7.45, 9.17};

int[] iArrRes = new int[5];

double[] dArrRes = new double[5];

int i;

// Запись в файл

FileStream fsW = new FileStream("file2", FileMode.Create,

FileAccess.Write);

BinaryWriter bw = new BinaryWriter(fsW);

for (i = 0; i <= (iArr.Length - 1); i++)

bw.Write(iArr[i]);

for (i = 0; i <= (dArr.Length - 1); i++)

bw.Write(dArr[i]);

bw.Close();

fsW.Close();

// Чтение файла

if (File.Exists("file2"))

{

FileStream fsR = new FileStream("file2", FileMode.Open,

FileAccess.Read);

BinaryReader br = new BinaryReader(fsR);

for (i = 0; i <= (iArrRes.Length - 1); i++)

iArrRes[i] = br.ReadInt32();

for (i = 0; i <= (dArrRes.Length - 1); i++)

dArrRes[i] = br.ReadDouble();

br.Close();

fsR.Close();

}

for (i = 0; i <= (iArrRes.Length - 1); i++)

Console.WriteLine("iArrRes[{0}]={1};" +

"\tdArrRes[{0}]={2}", i, iArrRes[i], dArrRes[i]);

Console.ReadLine();

}

}

}

Результаты расчета см. рис. 13.2.



Рисунок 13.2 – Результаты решения задачи 13.2

Здесь числовые массивы iArr и dArr циклически поэлементно записываются в файл file2. После закрытия файла и его повторного открытия для чтения содержимое файла file2 последовательно вычитывается в массивы - приемники iArrRes и dArrRes.

Бывают ситуации, когда число записанных в файл чисел неизвестно. Как поступить в этом случае, если надо прочитать все содержимое файла?

Ответ на этот вопрос дает задача 13.3.

**Задача 13.3.**

*Программный код:*

using System;

using System.IO;

namespace C97

{

class Program

{

static void Main(string[] args)

{

int[] iArr = { 1, 5, 7, 10, 12 };

int[] iArrRes = new int[8];

int i;

// Запись в файл

FileStream fsW = new FileStream("file3", FileMode.Create,

FileAccess.Write);

BinaryWriter bw = new BinaryWriter(fsW);

for (i = 0; i <= (iArr.Length - 1); i++)

bw.Write(iArr[i]);

bw.Close();

fsW.Close();

// Чтение файла

if (File.Exists("file3"))

{

FileStream fsR = new FileStream("file3", FileMode.Open,

FileAccess.Read);

BinaryReader br = new BinaryReader(fsR);

i = 0;

while (br.PeekChar() != -1 && i < 8)

{

iArrRes[i] = br.ReadInt32();

i++;

}

br.Close();

fsR.Close();

}

for (i = 0; i <= (iArrRes.Length - 1); i++)

Console.WriteLine("iArrRes[{0}]={1}" , i, iArrRes[i]);

Console.ReadLine();

}

}

}

Результаты расчета см. рис. 13.3.



Рисунок 13.3 – Результаты решения задачи 13.3

В данной программе мы сначала записываем в файл file3 содержимое массива iArr из пяти элементов типа int. Затем файл закрывается и снова открывается на чтение. Но прежде чем читать из файла методом ReadInt32() очередное целое значение, контролируем наличие невычитанных из файла данных методом PeekChar().

Пока метод PeekChar() возвращает не равные –1 значения, в файле file3 еще имеются невычитанные данные, которые мы и продолжаем циклически вычитывать методом ReadInt32(). Как только при очередной попытке чтения метод PeekChar() вернет –1, то условие цикла станет ложным и он перестанет выполняться (другое условие i<8 дополнительно страхует от переполнения принимающего буфера iArrRes). В принимающий массив iArrRes, размер которого достаточен для приема восьми целых чисел, из файла file3 было прочитано пять целых значений.