**1. Какие классы содержаться с System.IO?**

**BinaryReader, BinaryWriter**

Эти классы позволяют сохранять и извлекать элементарные типы данных (целочисленные, булевские, строковые и т.п.) в двоичном виде

**BufferedStream**

Этот класс предоставляет временное хранилище для потока байтов, которые могут затем быть перенесены в постоянные хранилища

**Directory, DirectoryInfo**

Эти классы используются для манипуляций структурой каталогов машины. Тип Directory представляет функциональность, используя статические члены. Тип DirectoryInfo обеспечивает аналогичную функциональность через действительную объектную ссылку

**DriveInfo**

Этот класс предоставляет детальную информацию относительно дисковых устройств, используемых данной машиной

**File, FileInfo**

Эти классы служат для манипуляций множеством файлов данной машины. Тип File представляет функциональность через статические члены. Тип FileInfo обеспечивает аналогичную функциональность через действительную объектную ссылку

**FileStream**

Этот класс обеспечивает произвольный доступ к файлу (т.е. возможности поиска) с данными, представленными в виде потока байт

**FileSystemWatcher**

Этот класс позволяет отслеживать модификации внешних файлов в определенном каталоге

**MemoryStream**

Этот класс обеспечивает произвольный доступ к данным, хранящимся в памяти, а не в физическом файле

**Path**

Этот класс выполняет операции над типами System.String, содержащими информацию о пути к файлу или каталогу в независимой от платформы манере

**StreamWriter, StreamReader**

Эти классы используются для хранения (и извлечения) текстовой информации из файла. Эти классы не поддерживают произвольного доступа к файлу

**StringWriter, StringReader**

Подобно классам StreamWriter/StreamReader, эти классы также работают с текстовой информацией. Однако лежащим в основе хранилищем является строковый буфер, а не физический файл

**2. Для чего используются классы Directory и DirectoryInfo? В чем отличие?**

**Directory, DirectoryInfo**

Эти классы используются для манипуляций структурой каталогов машины. Тип Directory представляет функциональность, используя статические члены. Тип DirectoryInfo обеспечивает аналогичную функциональность через действительную объектную ссылку

**3. Для чего используются классы File и FileInfo? Какие методы они содержат.**

**File, FileInfo**

Эти классы служат для манипуляций множеством файлов данной машины. Тип File представляет функциональность через статические члены. Тип FileInfo обеспечивает аналогичную функциональность через действительную объектную ссылку

Create()

Delete()

MoveTo()

**4. Для чего используются классы StreamReader и StreamWriter?**

**StreamWriter, StreamReader**

Эти классы используются для хранения (и извлечения) текстовой информации из файла. Эти классы не поддерживают произвольного доступа к файлу

**5. Для чего используются классы BinaryWriter и BinaryReader?**

Класс BinaryWriter служит оболочкой, в которую заключается байтовый поток, управляющий выводом двоичных данных. Ниже приведен наиболее часто употребляемый конструктор этого класса:

**BinaryWriter(Stream output)**

где output обозначает поток, в который выводятся записываемые данные. Для записи в выходной файл в качестве параметра output может быть указан объект, создаваемый средствами класса FileStream. Если же параметр output оказывается пустым, то генерируется исключение ArgumentNullException. А если поток, определяемый параметром output, не был открыт для записи данных, то генерируется исключение ArgumentException. По завершении вывода в поток типа BinaryWriter его нужно закрыть. При этом закрывается и базовый поток.

Класс BinaryReader служит оболочкой, в которую заключается байтовый поток, управляющий вводом двоичных данных. Ниже приведен наиболее часто употребляемый конструктор этого класса:

**BinaryReader(Stream input)**

где input обозначает поток, из которого вводятся считываемые данные. Для чтения из входного файла в качестве параметра input может быть указан объект, создаваемый средствами класса FileStream.

**6. Как можно сжимать и восстанавливать файлы?**

Кроме классов чтения-записи .NET предоставляет классы, которые позволяют сжимать файлы, а также затем восстанавливать их в исходное состояние.

Это классы **DeflateStream** и **GZipStream**, которые находятся в пространстве имен **System.IO.Compression** и представляют реализацию одного из алгоритмов сжатия Deflate или GZip.

**7. Расскажите алгоритм сжатия GZip.**

Сначала создается поток для чтения из исходного файла - FileStream sourceStream. Затем создается поток для записи в сжатый файл -FileStream targetStream. Поток архивации GZipStream compressionStream инициализируется потоком targetStream и с помощью метода CopyTo()получает данные от потока sourceStream.

Метод Decompress производит обратную операцию по восстановлению сжатого файла в исходное состояние. Он принимает в качестве параметров пути к сжатому файлу и будущему восстановленному файлу.

Здесь в начале создается поток для чтения из сжатого файла FileStream sourceStream, затем поток ля записи в восстанавливаемый файлFileStream targetStream. В конце создается поток GZipStream decompressionStream, который с помощью метода CopyTo() копирует восстановленные данные в поток targetStream.

Чтобы указать потоку GZipStream, для чего именно он предназначен - сжатия или восстановления - ему в конструктор передается параметр CompressionMode, принимающий два значения: Compress и Decompress.

Если бы захотели бы использовать другой класс сжатия - DeflateStream, то мы могли бы просто заменить в коде упоминания GZipStream на DeflateStream, без изменения остального кода. Их использование идентично.

В то же время при использовании этих классов есть некоторые ограничения, в частности, мы можем сжимать только один файл. Для архивации группы файлы лучше выбрать другие инструменты.

**8. Что такое тестовый и бинарный файл. Как можно выполнить чтение и запись в эти файлы.**

Для работы с бинарными файлами предназначена пара классов **BinaryWriter** и **BinaryReader**. Эти классы позволяют читать и записывать данные в двоичном формате.

**Основные метода класса BinaryWriter**

**Close()**: закрывает поток и освобождает ресурсы

**Flush()**: очищает буфер, дописывая из него оставшиеся данные в файл

**Seek()**: устанавливает позицию в потоке

**Write()**: записывает данные в поток

**Основные метода класса BinaryReader**

**Close()**: закрывает поток и освобождает ресурсы

**ReadBoolean()**: считывает значение bool и перемещает указатель на один байт

**ReadByte()**: считывает один байт и перемещает указатель на один байт

**ReadChar()**: считывает значение char, то есть один символ, и перемещает указатель на столько байтов, сколько занимает символ в текущей кодировке

**ReadDecimal()**: считывает значение decimal и перемещает указатель на 16 байт

**ReadDouble()**: считывает значение double и перемещает указатель на 8 байт

**ReadInt16()**: считывает значение short и перемещает указатель на 2 байта

**ReadInt32()**: считывает значение int и перемещает указатель на 4 байта

**ReadInt64()**: считывает значение long и перемещает указатель на 8 байт

**ReadSingle()**: считывает значение float и перемещает указатель на 4 байта

**ReadString()**: считывает значение string. Каждая строка предваряется значением длины строки, которое представляет 7-битное целое число

С чтением бинарных данных все просто: соответствующий метод считывает данные определенного типа и перемещает указатель на размер этого типа в байтах, например, значение типа int занимает 4 байта, поэтому BinaryReader считает 4 байта и переместит указать на эти 4 байта.

Класс FileStream не очень удобно применять для работы с текстовыми файлами. К тому же для этого в пространстве System.IO определены специальные классы: **StreamReader** и **StreamWriter**.

Чтение из файла и StreamReader

Класс StreamReader позволяет нам легко считывать весь текст или отдельные строки из текстового файла. Среди его методов можно выделить следующие:

**Close**: закрывает считываемый файл и освобождает все ресурсы

**Peek**: возвращает следующий доступный символ, если символов больше нет, то возвращает -1

**Read**: считывает и возвращает следующий символ в численном представлении. Имеет перегруженную версию: Read(char[] array, int index, int count), где array - массив, куда считываются символы, index - индекс в массиве array, начиная с которого записываются считываемые символы, и count - максимальное количество считываемых символов

**ReadLine**: считывает одну строку в файле

**ReadToEnd**: считывает весь текст из файла

Запись в файл и StreamWriter

Для записи в текстовый файл используется класс StreamWriter. Свою функциональность он реализует через следующие методы:

**Close**: закрывает записываемый файл и освобождает все ресурсы

**Flush**: записывает в файл оставшиеся в буфере данные и очищает буфер.

**Write**: записывает в файл данные простейших типов, как int, double, char, string и т.д.

**WriteLine**: также записывает данные, только после записи добавляет в файл символ окончания строки

**9. Что такое произвольный доступ к файлу? Приведите пример.**

Нередко бинарные файлы представляют определенную стрктуру. И, зная эту структуру, мы можем взять из файла нужную порцию информации или наоброт записать в определенном месте файла определенный набор байтов. Например, в wav-файлах непосредственно звуковые данные начинаются с 44 байта, а до 44 байта идут различные метаданные - количество каналов аудио, частота дискретизации и т.д.

С помощью метода Seek() мы можем управлять положением курсора потока, начиная с которого производится считывание или запись в файл. Этот метод принимает два параметра: offset (смещение) и позиция в файле. Позиция в файле описывается тремя значениями:

**SeekOrigin.Begin**: начало файла

**SeekOrigin.End**: конец файла

**SeekOrigin.Current**: текущая позиция в файле

Курсор потока, с которого начинается чтение или запись, смещается вперед на значение offset относительно позиции, указанной в качестве второго параметра. Смещение может отрицательным, тогда курсор сдвигается назад, если положительное - то вперед.

**10. Как применяется конструкция using (не директива) при работе с файловыми потоками? Для чего ее используют.**

И при чтении, и при записи используется оператор using. Не надо путать данный оператор с директивой using, которая подключает пространства имен в начале файла кода. Оператор using позволяет создавать объект в блоке кода, по завершению которого вызывается метод Dispose у этого объекта, и, таким образом, объект уничтожается. В данном случае в качестве такого объекта служит переменная fstream.

Объект fstream создается двумя разными способами: через конструктор и через один из статических методов класса File.

Здесь в конструктор передается два параметра: путь к файлу и перечисление FileMode. Данное перечисление указывает на режим доступа к файлу и может принимать следующие значения:

**Append**: если файл существует, то текст добавляется в конец файл. Если файла нет, то он создается. Файл открывается только для записи.

**Create**: создается новый файл. Если такой файл уже существует, то он перезаписывается

**CreateNew**: создается новый файл. Если такой файл уже существует, то он приложение выбрасывает ошибку

**Open**: открывает файл. Если файл не существует, выбрасывается исключение

**OpenOrCreate**: если файл существует, он открывается, если нет - создается новый

**Truncate**: если файл существует, то он перезаписывается. Файл открывается только для записи.