Вот поэтому рекомендуется использовать специализированный в Delphi объект TFiiestream. Вы можете написать и свой класс, который будет делать то же самое, но TFiiestream делает это достаточно хорошо.

**СОВЕТ**. Объект TFileStream рекомендуется использовать по следующей причине. Если Microsoft снова введет какие-то нововведения, Borland учтет их в объекте, и вам нужно будет только перекомпилировать свои программы. Никаких изменений в код вносить не надо. Вы один раз изменяете объект (или это делает Borland) и компили­руете все свои программы с новыми возможностями. Это громадное преимущество ООП, и вы должны его использовать, потому что оно упрощает жизнь.

У TFileStream есть еще одно преимущество — вам не нужно отслеживать ново­введения от Microsoft. В этой корпорации постоянно появляются новые идеи, из-за которых появляются новые функции, а мы, программисты, должны постоянно изу­чать новинки и переписывать код. Лично я намучился уже с функциями работы с файлами. Их так много, что с первого взгляда даже не поймешь, какие новые и рекомендуются к использованию, а какие устарели. В любом случае, использова­ние этого объекта намного проще, чем WinAPI, поэтому желательно начинать изу­чение работы с файлами именно с него.

Итак, давайте взглянем на объект TFileStream. Первое, что надо сделать, — это объявить какую-нибудь переменную типа TFileStream:

var

f: TFileStream;

Вот так мы объявили переменную f типа объекта TFileStream. Теперь можно проинициапизировать переменную.

Инициализация — выделение памяти и установка значений по умолчанию. За эти действия отвечает метод create. Нужно просто вызвать его и результат выполнения присвоить переменной. Например, в нашем случае нужно вызвать TFileStream.create и результат записать в переменную f.

f := TFileStream.Create(параметры);

Давайте разберемся, какие параметры могут быть при инициализации объекта TFileStream. У метода create может быть три параметра, причем последний мож­но не указывать.

* Имя файла (или полный путь к файлу), который надо открыть. Этот параметр — простая строка.
* Режим открытия. Здесь вы можете указать один из следующих параметров открытия файла:
  + fmCreate— создать файл с указанным в первом параметре именем. Если файл уже существует, то он откроется в режиме для записи;
  + fmOpenRead — открыть файл только для чтения. Если файл не существует, то произойдет ошибка. Запись в файл в этом случае не возможна;
  + fmopenwrite — открыть файл для записи. При этом во время записи текущее содержимое уничтожается;
  + fmOpenReadWrite — открыть файл для редактирования (чтения и записи).
* Права, с которыми будет открыт файл. Здесь можно указать одно из следую­щих значений (а можно вообще ничего не указывать):
  + fmshareCompat — при этих правах другие приложения тоже имеют права ра­ботать с открытым файлом;
  + fmShareExciusive — другие приложения не смогут открыть файл;
    - fmShareDenyWrite — при данном режиме другие приложения не смогут от­крывать этот файл для записи. Файл может быть открыт только для чтения;
    - fmshareDenyRead— при данном режиме другие приложения не смогут от­крывать этот файл для чтения. Файл может быть открыт только для записи;
    - fmShareDenyNone — не мешать другим приложениям работать с файлом.

С первыми двумя параметрами все ясно. Но зачем же нужны права доступа к файлам. Допустим, что вы открыли файл для записи. Потом его открыло другое приложение и тоже для записи. Обе программы записали какие-то данные. После этого ваше приложение закрыло файл и сохранило все изменения. Тут же другое приложение перезаписывает ваши изменения, даже не подозревая о том, что они уже прошли. Вот так ваша информация пропадает из файла, потому что другая программа просто перезаписала ее.

Если вы пишете однопользовательскую программу, и к файлу будет иметь дос­туп только одно приложение, то про права можно забыть и не указывать их.

После того как вы поработали с файлом, достаточно вызвать метод Free, чтобы закрыть его:

f.Free;

Теперь давайте познакомимся с методами чтения, записи и внутренней структу­рой файла. Начнем со структуры. Когда вы открыли файл, позиция курсора уста­навливается в самое начало и любая попытка чтения или записи будет происходить в эту позицию курсора. Если вам надо прочитать или записать в любую другую по­зицию, то надо передвинуть курсор. Для этого есть метод seek. У него есть два па­раметра:

* Число, указывающее на позицию, в которую надо перейти. Если вам нужно пе­редвинуться на 5 байт, то просто поставьте цифру 5.
* Откуда надо двигаться. Тут возможны три варианта:
* soFromBeginning — двигаться на указанное количество байт от начала файла.
  + soFromCurrent — двигаться на указанное количество байт от текущей пози­ции в файле к концу файла.
  + soFromEnd — двигаться от конца файла к началу на указанное количество байт.

Не забывайте, что 1 байт— это один символ. Единственное исключение — файлы в формате Unicode. В них один символ занимает 2 байта. Так образом, надо учитывать, в каком виде хранится информация в файле.

Итак, если вам надо передвинуться на 10 символов от начала файла, можете на­писать следующий код:

f.Seek(10, soFromBeginning);

Метод seek всегда возвращает смещение курсора от начала файла. Этим можно воспользоваться, чтобы узнать, где мы сейчас находимся, а можно и узнать общий размер файла. Если переместиться в конец файла, то функция вернет количество байт от начала до конца, т. е. полный размер файла.

В следующем примере устанавливается позиция в файле на 0 байт от конца, т. е. в самый конец. Тем самым получается полный размер файла:

Размер файла := f.Seek(0, soFromEnd);

Вот таким небольшим трюком можно узнать размер. Правда, для этого придется ис­пользовать три операции: открыть, переместиться в конец файла и потом закрыть его.

Для чтения из файла нужно использовать метод Read. И снова у этого метода два параметра:

* переменная, в которую будет записан результат чтения;
* количество байт, которые надо прочитать.

Давайте взглянем на код чтения из файла, начиная с 15-й позиции. Этот код вы можете увидеть в листинге

var

f:TFileStream; //Переменная типа объект

TFileStream. buf: array[0..10] of char; // Буфер для хранения прочитанных данных begin

//В следующей строке я открываю файл filename.txt для чтения и записи,

f:= TFileStream.Create(1 с:\filename.txt', fmOpenReadWrite);

f.Seek(15, soFromCurrent); // Перемещаюсь на 15 символов вперед,

f.Read(buf, 10); // Читаю 10 символов из установленной позиции.

f.Free; // Уничтожаю объект и соответственно закрываю файл, end;

Обратите внимание, что в методе **seek** движение происходит на 15 символов не от начала, а от текущей позиции, хотя нужно от начала. Это потому, что после от­крытия файла текущая позиция и есть начало. Но лучше на это не рассчитывать.

Метод Read возвращает количество реально прочитанных байт (символов). Если не возникло никаких проблем, то это число должно быть равно количеству запро­шенных для чтения байт.

Есть только два случая, когда эти числа отличаются:

* при чтении был достигнут конец файла и дальнейшее чтение стало невозможным;
* ошибка на диске или любая другая проблема.

Осталось только разобраться с записью. Для этого мы будем использовать метод write. У него так же два параметра:

* переменная, содержимое которой нужно записать;
* число байт для записи.

Пользоваться этим методом можно точно так же, как и методом для чтения.

**ЗАМЕЧАНИЕ**. После чтения или записи текущая позиция в файле смещается на ко­личество прочитанных байт. То есть текущая позиция становится в конец прочитанно­го блока данных.

Пример пока рассматривать не будем, потому что в принципе и так все ясно. А если есть вопросы, то на практике мы закрепим этот материал буквально через один или два раздела, когда будем работать со структурами.