Лабораторная работа №3.
Файлы.

Время: 180 мин.
Что нужно освоить:
1) особенности доступа к файлам из программы;
2) подпрограммы для работы с файлами;
3) порядок и особенности обработки текстовых и типизированных файлов;
4) обработка ошибок ввода/вывода.

Создайте папку, в которую будете сохранять разрабатываемые приложения. Для каждого приложения в дальнейшем в главной папке следует создавать отдельный каталог.

1. ТЕКСТОВЫЕ ФАЙЛЫ

1.1. Доступ к файлам

Текстовые файлы рассматриваются как совокупность строк переменной длины. Доступ к каждой строке доступен лишь последовательно. Начать чтение строк файла можно только с первой строки. Чтение строки организуется с помощью процедуры `ReadLn`. Данная процедура имеет два аргумента. Первый аргумент используется для обозначения конкретного файла, из которого осуществляется чтение. Второй аргумент – переменная в которую будет помещена очередная строка из файла. Пример: `ReadLn(f,s)` – здесь `f` – имя файловой переменной типа `textfile`, ассоциированной с конкретным файлом, а `s` – имя строковой переменной, в которую помещается считанная из файла строка.

Для того чтобы можно было обращаться к файлу через имя файловой переменной следует сначала их связать процедурой `AssignFile(f,n)`, где `f` – имя файловой переменной, а `n` – имя файла. Имя файла можно задать непосредственно через строковую константу:

```
AssignFile(f,'start.txt')
```

или через строковую переменную:

```
n:='start.txt';
AssignFile(f,n);
```

Организация доступа к файлу этим не исчерпывается. Кроме *связывания файловой переменной* следует также *инициализировать файл* одной из процедур: `Reset`, `Rewrite` или `Append`. Процедура `Reset` открывает файл для чтения, `Rewrite` – для записи, `Append` – для добавления записей.

После работы с файлом его следует закрыть процедурой `CloseFile(f)`, где `f` – имя файловой переменной.

Создайте новое приложение в среде `Delphi`. Разместите на форме компоненты: `Memo` и `Button`. Создайте на кнопке обработчик события `Button1Click` и заполните его кодом по образцу:

```
procedure TForm1.Button1Click(Sender: TObject);
var f: textfile;
    s: string;
begin
  AssignFile(f,'start.txt');
  Reset(f);
  readln(f,s);
  memo1.Lines.Add(s);
  closefile(f);
End;
```

Данная процедура при отсутствии файла start.txt, естественно, работать не будет. Сохраните проект в заранее подготовленную папку и в ней же разместите файл start.txt, в котором должны быть несколько строк произвольного текста.

Если вы создаете сложное приложение и в основной папке программы впоследствии будут располагаться различные файлы и каталоги, то, скорее всего, возникнет необходимость читать файлы не только из основного каталога программы, но и из подкаталогов.

Создайте в основном каталоге вашей программы подкаталог INI и в него скопируйте файл start.txt. Переименуйте его в start.ini. Для того чтобы ассоциировать его с файловой переменной можно поступить так: AssignFile(f,'ini/start.ini');

Доработайте самостоятельно процедуру, чтобы она читала строку из файла start.ini находящегося в подкаталоге INI. Апробируйте программу.

Другой способ состоит в том, чтобы сформировать полное имя файла, включая наименование диска и путь к файлу. Для этого сначала следует определить текущую папку программы с помощью процедуры GetDir(0,t), где 0 – указывает на текущий диск, а t – переменная строкового типа, которая возвращает путь к основному каталогу программы (откуда она была запущена на исполнение).

Доработайте процедуру следующим образом и оцените результат её работы:

```
procedure TForm1.Button1Click(Sender: TObject);
var f: textfile;
    s: string;
begin
  getdir(0,s);
  memo1.Lines.Add(s);
  AssignFile(f,'ini/start.ini');
  Reset(f);
  readln(f,s);
  memo1.Lines.Add(s);
  closefile(f);
end;
```

В поле Memo выводится путь к основному каталогу программы. Путь возвращает переменная строкового типа s. Имя файла также может быть переменной или константой строкового типа, поэтому к ним можно применить операцию конкатенации строк. Например, если файл находится в подкаталоге INI, то можно взять путь к основному каталогу программы, к нему добавить имя подкаталога INI и имя файла:

```
procedure TForm1.Button1Click(Sender: TObject);
var f: textfile;
    s,n,k: string;
begin
  getdir(0,k);          // определяем основной каталог
  n:='start.ini';       // указываем имя файла
  s:=k+'/ini/'+n;       // определяем полное имя файла
  AssignFile(f,s);
  Reset(f);
  readln(f,s);
  memo1.Lines.Add(s);
  closefile(f);
end;
```

До сих пор мы только читали файл, теперь попробуем записывать – доработайте процедуру следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var f: textfile;
    s,n,k: string;
begin
  getdir(0,k);          // определяем основной каталог
  n:='start.ini';       // указываем имя файла
  s:=k+'/ini/'+n;       // определяем полное имя файла
  AssignFile(f,s);
  Reset(f);
  readln(f,s);
  memo1.Lines.Add(s);
  closefile(f);

  n:='start2.txt';      // указываем имя файла
  s:=k+'/ini/'+n;       // определяем полное имя файла
  AssignFile(f,s);
  Rewrite(f);           // открываем файл для записи
  writeln(f,s);
  closefile(f);
end;
```

Испытайте её и посмотрите на содержимое нового файла.

Доработайте её так, чтобы первая строка из исходного файла переписывалась во второй файл.

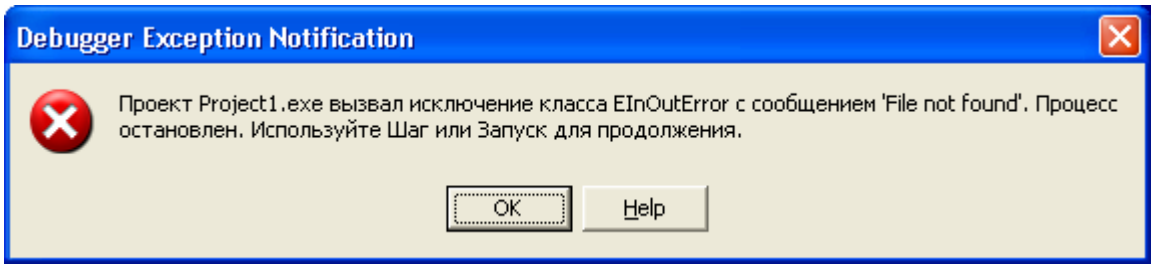
1.2. Обработка ошибок обращения к файлу.

Пришло время обсудить *порядок обработки ошибок* ввода/вывода.

Рассмотрим два способа. **Первый** основан на использовании стандартного оператора обработки исключительных ситуаций: **`try except`**. Напомню кратко синтаксис этого оператора: после ключевого слова `try` (проверить) должны располагаться операторы, при выполнении которых возможно появление исключительной ситуации (в нашем случае, например, отсутствие файла); после ключевого слова `except` (исключить) – операторы, которые по замыслу программиста должны выполняться при наступлении исключительной ситуации.

```
Procedure TForm1.Button1Click(Sender: TObject);
var f: textfile;
    s,n,k: string;
begin
  getdir(0,k);          // определяем основной каталог
  n:='start.ini';       // указываем имя файла
  s:=k+'/ini/'+n;       // определяем полное имя файла
  try
    AssignFile(f,s);
    Reset(f);
    readln(f,s);
    memo1.Lines.Add(s);
    closefile(f);
  except
    on EInOutError do memo1.Lines.Add('файл '+s+' не найден');
  end;
end;
```

Апробируйте эту процедуру при наличии файла start.ini и при его отсутствии. Если вы запускаете программу из среды Delphi, то сама среда будет обрабатывать ошибки и сообщит вам об этом:



Нажмите OK и Delphi укажет вам место в программном коде где случилось исключение – для продолжения работы программы в пошаговом режиме нажимайте F8.

В структурном операторе try except не обязательно указывать тип исключительной ситуации (более подробно об операторе можно посмотреть в справке Delphi). В нашем случае процедуру можно несколько сократить:

```
Procedure TForm1.Button1Click(Sender: TObject);
var f: textfile;
    s,n,k: string;
begin
  getdir(0,k);          // определяем основной каталог
  n:='start.ini';       // указываем имя файла
  s:=k+'/ini/'+n;       // определяем полное имя файла
  try
    AssignFile(f,s);
    Reset(f);
    readln(f,s);
    memo1.Lines.Add(s);
    closefile(f);
  except
    memo1.Lines.Add('файл '+s+' не найден');
  end;
end;
```

Организовать работу программы можно и иным способом. Второй способ основан на использовании стандартной функции FileExists, которая возвращает True, если файл существует, и False – если не существует:

```
Procedure TForm1.Button1Click(Sender: TObject);
var f: textfile;
    s,n,k: string;
begin
  getdir(0,k);          // определяем основной каталог
  n:='start.ini';       // указываем имя файла
  s:=k+'/ini/'+n;       // определяем полное имя файла
  if FileExists(s) then
  begin
    AssignFile(f,s);
    Reset(f);
    readln(f,s);
    memo1.Lines.Add(s);
```

```
    closefile(f);
  end
  else
    memo1.Lines.Add('файл '+s+' не найден');
end;
```

1.3. Стандартные процедуры работы с файлами.

Приведу в качестве справки еще несколько стандартных процедур работы с файлами и каталогами:

1) `procedure Rename(var f; NewName: String);` – файл, ассоциированный с файловой переменной `f`, переименовывается в `NewName`, перед выполнением файл должен быть закрыт;

2) `function EOF(var f): Boolean;` – возвращает `True`, если достигнут конец файла;

3) `procedure Flush(var f);` – очищает буфер файла, обеспечивая сохранность последних изменений;

4) `procedure MkDir(Dir: String);` – создает новый каталог;

5) `procedure RmDir(Dir: String);` – удаляет каталог (если он пустой);

6) `procedure Erase(var f);` – удаляет файл, перед выполнением файл должен быть закрыт;

7) `function DeleteFile(const FileName: string): Boolean;` – удаляет файл без связывания с файловой переменной, перед выполнением файл должен быть закрыт;

8) `function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer;` – функция возвращает 0 при успешном поиске, используется для инициализации переменной f, используемой при последующем поиске функцией `FindNext`.

Параметр `Path` может содержать путь к каталогу, в котором организуется поиск, и должен содержать маску выбора файлов. Маска формируется с помощью символов-заменителей:

? – означает, что на этой позиции может находиться один из разрешенных символов;

* – означает, что на этой позиции могут находиться несколько (включая 0) разрешенных символов.

Например:

`*.*` – любой файл из текущего каталога;

`a*.*` – любой файл из каталога, начинающийся с символа `a`;

`c:\pas\a?.pas` – файлы с расширением `pas` из каталога `c:\pas\`, имя которых начинается с символа `a` и содержит два символа (`a0.pas, a1.pas, aa.pas, …`).

Параметр `Attr` формируется битами разрядной сетки и может приобретать значения из списка констант:

`faReadOnly` – только чтение;

`faHidden` – скрытый файл;

`faSysFile` – системный файл;

`faVolumeID` – идентификатор тома;

`faDirectory` – имя вложенного каталога;

`faArchive` – архивный файл;

`faAnyFile` – любой файл.

Переменная `f` возвращает имя найденного файла.

9) `function FindNext(var F: TSearchRec): Integer;` – функция при успешном поиске возвращает 0.

Переменная `f` возвращает имя найденного файла.

10) `procedure FindClose(var F: TSearchRec);` – освобождает память, выделенную для поиска файлов.

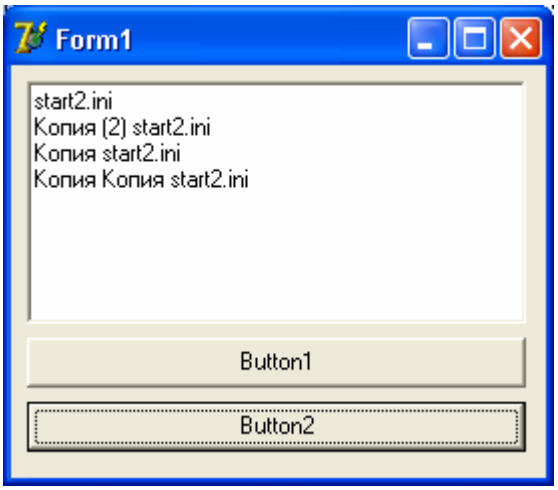
1.4. Обсуждение примеров работы с файлами.

Разберем на простых примерах некоторые из приведенных стандартных подпрограмм.

Пример №1. Написать процедуру построения списка файлов с расширением `ini`, находящихся в подкаталоге `INI`.

Программу выполним на основе предыдущей, для чего добавьте на форму еще одну кнопку – `Button2`. Создайте следующий обработчик события:

```
Procedure TForm1.Button2Click(Sender: TObject);
var sr: TSearchRec;
    m,k,p: string;
begin
  getdir(0,k);          // определяем основной каталог
  k:=k+'/ini/';         // назначаем текущий каталог
  m:='*.ini';           // назначаем маску
  p:=k+m;               // назначаем параметр Path
  memo1.Lines.Clear;    // очищаем memo1
  if FindFirst(p,faAnyFile,sr)=0 then // если удалось найти файл
  repeat
    memo1.Lines.Add(sr.Name);         // выводим имя файла в memo1
  until FindNext(sr)<>0;              // пока не закончатся файлы
  FindClose(sr);        // освобождаем память
end;
```



Комментарии помогут вам разобраться с особенностями организации процедуры. В процедуре организован цикл с помощью структурного оператора `repeat until`. Особенность его состоит в том, что тело цикла выполнится хотя бы один раз, который наступит, если будет найден хотя бы один файл:

`if FindFirst(p,faAnyFile,sr)=0 then`,

соответствующий заданным в маске условиям.

Тело цикла будет повторяться пока в каталоге функцией `FindNext(sr)` будет находиться очередной файл, соответствующий заданным в маске условиям. Цикл закончится, когда очередной файл не будет обнаружен – `FindNext(sr)<>0`.

На скриншоте показан пример работы программы. Апробируйте процедуру.

Пример №2. Написать процедуру удаления файлов с расширением `ini`, находящихся в подкаталоге `INI`.

Программу выполним на основе предыдущей, для чего добавьте на форму еще одну кнопку – `Button3`. Создайте следующий обработчик события:

```
procedure TForm1.Button2Click(Sender: TObject);
var sr: TSearchRec;
    m,k,p,s: string;
    f: textfile;
begin
  getdir(0,k);           // определяем основной каталог
  k:=k+'/ini/';          // назначаем текущий каталог
  m:='*.ini';            // назначаем маску
  p:=k+m;                // назначаем параметр Path
  if FindFirst(p,faAnyFile,sr)=0 then // если удалось найти файл
  repeat
    s:=k+sr.Name;         // формируем полное имя файла
    AssignFile(f,s);      // ассоциируем имя с файловой переменной
    erase(f);             // удаляем файл
  until FindNext(sr)<>0;  // пока не закончатся файлы
  FindClose(sr);          // освобождаем память
end;
```

Комментарии помогут вам разобраться с особенностями организации процедуры. Изменено тело цикла – вместо вывода имени файла в memo1, производится удаление файла. Апробируйте процедуру.

Можно несколько упростить данную процедуру, используя процедуру `DeleteFile` вместо Erase. При этом не требуется проводить ассоциацию имени файла с файловой переменной – `AssignFile(f,s);` и, соответственно, нет необходимости объявлять ее в разделе `var` (`f: textfile;`). Произведите эту доработку самостоятельно.

Пример №3. Написать процедуру переименования файлов с расширением `ini`, находящихся в подкаталоге `INI`, в файлы с расширением txt.

Программу выполним на основе предыдущей, для чего добавьте на форму еще одну кнопку – `Button4`. Создайте следующий обработчик события:

```
procedure TForm1.Button2Click(Sender: TObject);
var sr: TSearchRec;
    m,k,p,s,sn: string;
    f: textfile;
begin
  getdir(0,k);           // определяем основной каталог
  k:=k+'/ini/';          // назначаем текущий каталог
  m:='*.ini';            // назначаем маску
  p:=k+m;                // назначаем параметр Path
  if FindFirst(p,faAnyFile,sr)=0 then // если удалось найти файл
  repeat
    s:=k+sr.Name;         // формируем полное имя файла
    AssignFile(f,s);      // ассоциируем имя с файловой переменной
    sn:=copy(s,1,length(s)-3)+'txt';  // меняем расширение файла
    Rename(f,sn);         // переименовываем файл
  until FindNext(sr)<>0;  // пока не закончатся файлы
  FindClose(sr);          // освобождаем память
end;
```

Комментарии помогут вам разобраться с особенностями организации процедуры.

Следует заметить, что для визуальной (и не только) работы пользователя с файлами и каталогами удобнее использовать специальные компоненты из палитры Delphi – компоненты `FileListBox`, `DirectoryListBox`, `DriveComboBox` и `FilterComboBox` с вклад-

ки Win 3.1. Однако рассмотрение свойств, событий и методов этих компонентов не входит в содержание данной работы.

1.5. Задания для самостоятельного исполнения.

1.5.1. В примере №1 разработана процедура построения списка файлов с расширением ini, находящихся в подкаталоге INI. Доработайте её таким образом, чтобы она выдавала список файлов ini, имеющих размер не менее заданного. Для задания размера файла в килобайтах разместите на форме поле Edit. Размер файла легко узнать, используя поле size типизированной переменной sr из примера №1. Пример обращения к полю size:

```
Repeat
   memo1.Lines.Add(IntToStr(sr.Size)+#9+sr.Name);
until FindNext(sr)<>0;
```

Это модификация цикла из процедуры примера №1. Добавление к строке кода #9 осуществляет вставку символа табуляции. Апробируйте процедуру, после чего выполните самостоятельно задание 1.5.1.

1.5.2. Создайте простейший текстовый редактор на основе компонента Memo. Основная особенность редактора состоит в том, что в нем предусмотрена возможность автосохранения резервной копии (*.bak) редактируемого файла. Должна быть предусмотрена возможность отключения опции «автосохранение» через меню программы. Частоту автосохранения ($T_{авт}$ раз в 1 мин., 5 мин., 10 мин. и т.д.) можно менять. В момент открытия файла (или в момент включения опции «автосохранение») при включенной опции «автосохранение» включается отсчет времени и при наступлении $T_{авт}$ происходит сохранение резервной копии. В программе должна быть предусмотрена возможность отката к резервной копии (к *.bak файлу), то есть возможность загрузить последний сохраненный резервный файл – файл с именем редактируемого файла, но с расширением bak.

2. ТИПИЗИРОВАННЫЕ ФАЙЛЫ

2.1. Организация доступа к типизированным файлам.

Организация доступа к типизированному файлу аналогична организации доступа к текстовому файлу: необходимы процедуры связывания файла с файловой переменной (`AssignFile`) и инициализации файла (для чтения, записи или перезаписи). Однако есть и отличия – к записям типизированного файла можно обращаться как последовательно, так и в любом порядке. Это обстоятельство связано с тем, что длина любого компонента типизированного файла постоянна и, собственно, зависит от объявленного типа записей файла. Последовательное обращение к записям типизированного файла организуется стандартными процедурами `Read` и `Write`. Если есть необходимость сменить порядок чтения/записи, то можно воспользоваться процедурой `Seek(f,k)`, которая смещает указатель текущего положения в файле `f` (`f` – файловая переменная) к требуемому компоненту `k` (`k` – переменная типа `LongInt`).

Для того чтобы узнать размер файла в компонентах его типа можно воспользоваться стандартной функцией `FileSize(f)`. Текущую позицию в файле (порядковый номер компонента файла, который будет обрабатываться следующей операцией ввода/вывода) возвращает функция `FilePos(f)`.

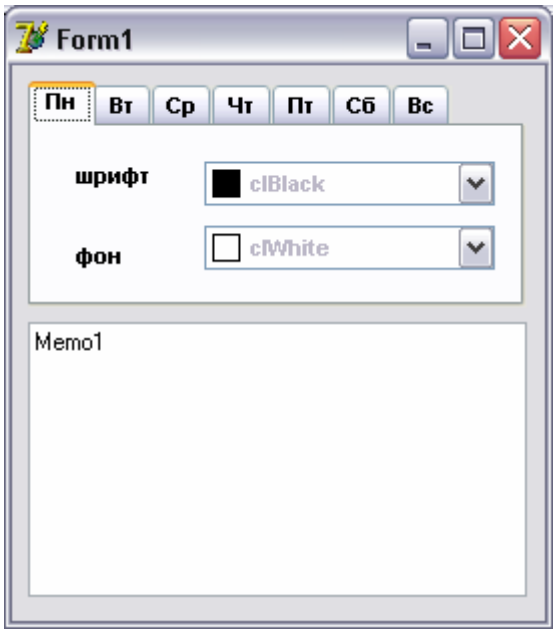
2.2. Пример работы с типизированными файлами.

Рассмотрим организацию работы с типизированными файлами на примере.

Разработаем программу – заготовку под текстовый редактор. Основная особенность редактора состоит в том, что цвета фона и шрифта зависят от дня недели и есть возможность эти цвета настроить.

Создайте новое приложение, на главной форме разместите компоненты: `Memo1` и `TabControl1` (с вкладки `Win32`).

В компоненте `TabControl1` (в разрабатываемом приложении он нужен для настройки цветов по дням недели) добавьте вкладки (Пн, Вт, Ср, Чт, Пт, Сб, Вс) через свойство `Tabs`, набрав соответствующие строки. Затем измените ширину ярлычков через свойство `TabWidth` (30 пикселей будет достаточно).



На компонент `TabControl1` положите две метки `Label` (с надписями – шрифт и фон) и два компонента `ColorBox` (с вкладки `Additional`). Для цвета шрифта у `ColorBox1` оставьте выделенный цвет `clBlack` (свойство `Selected`), а для цвета фона у `ColorBox2` измените свойство `Selected` на `clWhite`.

Примерный вид формы смотрите на рисунке.

После визуальной настройки компонентов можно приступать к созданию обработчиков событий. Всего их будет пять:

1) при изменении цвета в `ColorBox1` будет меняться цвет шрифта в `Memo1`;
2) при изменении цвета в `ColorBox2` будет меняться цвет фона в `Memo1`;
3) при нажатии на ярлычок `TabControl1` будут происходить изменения: устанавливаться цвета в `ColorBox1`, `ColorBox2` и `Memo1` для выбранного дня недели;
4) при запуске приложения подгружаются цветовые настройки из `ini` файла и устанавливается цветовая настройка `Memo1`, соответствующая текущему дню недели;
5) при закрытии приложения цветовые настройки по дням недели сохраняются в `ini` файл.

Первые два обработчика достаточно просты. Через инспектор объектов для компонентов `ColorBox1` и `ColorBox2` создайте заготовки под обработчики событий `OnChange` и заполните их следующим кодом:

```
procedure TForm1.ColorBox1Change(Sender: TObject);
begin
  memo1.Font.Color:=ColorBox1.Selected;
end;

procedure TForm1.ColorBox2Change(Sender: TObject);
begin
  memo1.Color:=ColorBox2.Selected;
end;
```

Апробируйте работу созданных процедур.

Переходим к *третьему обработчику* – он должен менять цветовые настройки многострочного текстового поля memo1 в зависимости от выбранного пользователем дня недели, то есть по клику на соответствующем ярлычке `TabControl1`. Сами цвета должны где то храниться, для чего создадим массив записей с двумя полями – одно поле для хранения цвета шрифта, а другое для хранения цвета фона.

В разделе глобальных переменных модуля добавим описание массива:

```
m: array[1..7] of dn;
```

где `dn` нестандартный тип данных, подразумевающий наличие двух полей записи: цвет шрифта и цвет фона. Идентификаторы полей у типа `dn` придумаем сами – например, `cfont` и `cback`. Нестандартный тип данных следует описать в соответствующем разделе модуля – в разделе `Type`:

```
Type
    dn = record
    cfont,cback: TColor;
  end;
var
  Form1: TForm1;
      m: array[1..7] of dn;
```

Итак, массив `m` описан как глобальная переменная и доступен в любой процедуре модуля. На начальном этапе, пока еще нет `ini` файла программы, при запуске приложения установим значения для всех дней недели одинаковыми: цвет фона – белый, цвет шрифта – черный. Чтобы реализовать эту возможность, создайте через инспектор объектов для главной формы приложения заготовку под обработчик события `OnCreate` и заполните его следующим кодом (*это 4-ый из 5-ти разрабатываемых обработчиков*):

```
procedure TForm1.FormCreate(Sender: TObject);
var i: byte;
begin
  for i:=1 to 7 do
  begin
    m[i].cfont:=clBlack;
    m[i].cback:=clWhite;
  end;
end;
```

Теперь уже есть возможность переключаться между цветовыми настройками дней недели. Через инспектор объектов для компонента `TabControl1` создайте обработчик события `OnChange` и заполните его следующим кодом (*это 3-ий из 5-ти разрабатываемых обработчиков*):

```
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  ColorBox1.Selected:=m[TabControl1.TabIndex+1].cfont;
  ColorBox2.Selected:=m[TabControl1.TabIndex+1].cback;
  memo1.Font.Color:=ColorBox1.Selected;
  memo1.Color:=ColorBox2.Selected;
end;
```

Первые две строчки обеспечивают изменение выбранных цветов шрифта и фона в соответствующих компонентах `ColorBox1` и `ColorBox2` в соответствии с выбранным днем недели (ярлычок `TabControl1`). Значения соответствующих цветов берутся из массива `m`. Тут следует пояснить, что, как и везде в Паскале, счет ярлычков в компоненте `TabControl` ведется с нуля, поэтому, для приведения в соответствие номеров ярлычков с номерами элементов массива `m` необходимо увеличивать номер `TabIndex` нажатого ярлычка на единицу (`m[TabControl1.TabIndex+1].cfont`). Например, при нажатии на самый правый ярлычок (с надписью ‘вс’) `TabIndex` возвращает значение 6.

Следующие две строчки устанавливают цветовые параметры поля `memo1`.

Апробируйте работу программы.

Осталось реализовать сохранение цветовой настройки по дням недели в `ini` файле.

Добавим в раздел описания глобальных переменных модуля еще одну строчку – описание файловой переменной `f` типа `dn`, описанного нами ранее:

```
Var
  Form1: TForm1;
      m: array[1..7] of dn;
      f: file of dn;
```

Теперь можно организовать работу с типизированным файлом. Каждый компонент этого файла будет представлять собой запись с двумя полями: cfont, cback: TColor.

При закрытии приложения текущие цветовые настройки должны быть сохранены в `ini` файл. Через инспектор объектов создайте заготовку под обработчик события `OnClose` и заполните следующим кодом (*это 5-ый из 5-ти разрабатываемых обработчиков*):

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var i: byte;
begin
  assignfile(f,'week.ini');
  rewrite(f);
  for i:=1 to 7 do write(f,m[i]);
  closefile(f);
end;
```

Здесь все достаточно очевидно и не требует дополнительных комментариев. Апробируйте программу: измените цветовые настройки разных дней. При выходе из приложения они будут сохранены в `ini` файл.

Если вы сразу же запустите программу повторно, то ваши настройки будут уничтожены процедурой `FormCreate`, так как в ней, при запуске приложения, устанавливаются цветовые значения по умолчанию – черный и белый для всех семи дней недели. Пришло время убрать цикл (`for i:=1 to 7 do`) и заменить его на цикл чтения записей из типизированного файла.

Было так:

```
procedure TForm1.FormCreate(Sender: TObject);
var i: byte;
begin
  for i:=1 to 7 do
  begin
    m[i].cfont:=clBlack;
    m[i].cback:=clWhite;
  end;
end;
```

А следует сделать так:

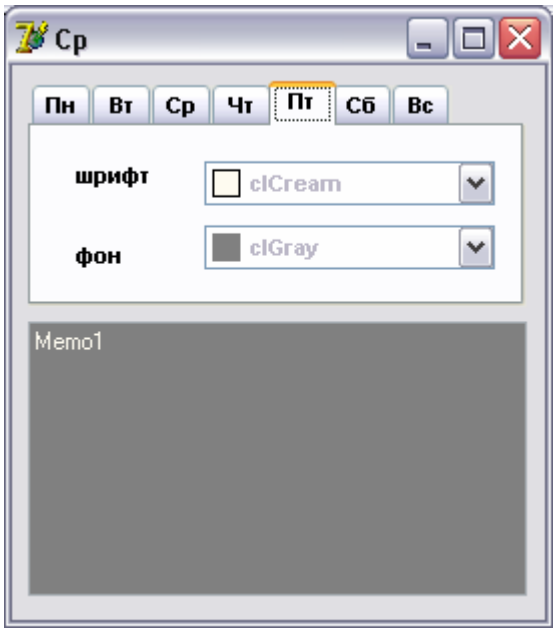
```
procedure TForm1.FormCreate(Sender: TObject);
var i: byte;
begin
  assignfile(f,'week.ini');
  reset(f);
  for i:=1 to filesize(f) do read(f,m[i]);
  closefile(f);
end;
```

Итак, к настоящему моменту программа уже функционально закончена и разработанные пять процедур реализуют основные задуманные возможности. Однако, до сих пор не реализовано основное предназначение `ini` файла: программа должна не только хранить в типизированном файле цветовые настройки, но и пользоваться ими, то есть приводить цветовые настройки поля `memo1` в соответствие с цветовыми настройками дня недели. При запуске программы (процедура `FormCreate`) следует определять текущий день недели и устанавливать соответствующие цвет шрифта и фона поля `memo1`.

Текущий день недели возвращает в строковом формате стандартная функция `FormatDateTime('ddd',Now);` извлекая его из переменной `Now`, содержащей системное время. Давайте попробуем её применить, добавив в процедуру `FormCreate`:

```
procedure TForm1.FormCreate(Sender: TObject);
var i: byte;
begin
  assignfile(f,'week.ini');
  reset(f);
  for i:=1 to filesize(f) do read(f,m[i]);
  closefile(f);
  Form1.Caption:=FormatDateTime('ddd',Now);
end;
```

Апробируйте работу программы. Текущий день недели в сокращенном формате (Пн, Вт, Ср, Чт, Пт, Сб, Вс) при запуске приложения будет выводиться в заголовок главной формы.



Осталось только по дню недели определять какой порядковый номер использовать для извлечения цветовых настроек из массива `m`. Сделать это довольно просто, так как список дней недели в сокращенном формате уже находится в свойстве `Tabs` компонента `TabControl1`. Свойство `Tabs` представляет собой список строк (Пн, Вт, Ср, Чт, Пт, Сб, Вс), каждая из которых имеет свой порядковый номер и номера строк начинаются с нуля. Например, `TabControl1.Tabs[0]` содержит строку `'Пн'`.

Дополним процедуру `FormCreate` циклом, последовательно проверяющим все строки свойства `Tabs` компонента `TabControl1` на совпадение с текущим днем недели. При совпадении цикл прерывается и имитируется нажатие на ярлычок компонента `TabControl1`, соответствующий текущему дню недели:

```
procedure TForm1.FormCreate(Sender: TObject);
var i: byte;
begin
  assignfile(f,'week.ini');
  reset(f);
  for i:=1 to filesize(f) do read(f,m[i]);
  closefile(f);
  Form1.Caption:=FormatDateTime('ddd',Now);
  for i:=0 to 6 do
    if TabControl1.Tabs[i]=Form1.Caption then break;
  TabControl1.TabIndex:=i;
  TabControl1Change(TabControl1);
end;
```

На этом разработку программы можно считать законченной. Создано пять несложных и небольших по объему процедур. Программа может служить шаблоном для разработки приложений, использующих в своей работе файлы конфигурации, содержащие необходимые настройки.

2.3. Задание для самостоятельного исполнения.

Дополните разработанное приложение таким образом, чтобы от дня недели зависели не только цвета шрифта и фона компонента `memo1`, но и сам шрифт (например, его имя, размер или стиль).