Замечания по поводу требований для выполнения данной лабораторной работы.

Студенты, выполняющую данную лабораторную работу должны успешно выполнить лабораторную работу, находящуюся в файле **«Технологии Интернет. Лабораторные работы. Раздел PHP. Работа с MySQL.pdf»** Кроме того, студенты должны обладать теоретическими знаниями протокола HTTP, уметь писать предложения на языке SQL-92.

Рекомендации по поводу настроек PHP и Apache не зависимо от версий и платформ:

Строго рекомендуется:

В **php.ini** раскомментировать одну из строк (только одну):

```
extension=php_gd.dll
extension=php_gd2.dll
```

необходимо для работы с функциями библиотеки GD

В **httpd.conf** добавить строку в раздел **Section 2: 'Main' server configuration**:

```
CharsetRecodeMultipartForms off
```

Необходимо для правильной загрузки бинарных файлов по протоколу http на Web-сервер

Работа с изображениями при использовании библиотеки GD.

Загрузка изображений на сервер. Создание галерей изображений.

Создание изображения

Как работать с картинками в GD? Для начала нужно картинку создать — пустую (при помощи imageCreate()) или же загруженную с диска (**imageCreateFromPng(), imageCreateFromJpeg()** или **imageCreateFromGif(**), в зависимости от того, какие форматы поддерживаются версиями PHP и GD). **int imageCreate(int $x, int $y)** Создает пустую картинку размером $x на $y точек и возвращает ее идентификатор. После того, как картинка создана, вся работа с ней осуществляется именно через этот идентификатор, по аналогии с тем, как мы работаем с файлом через его дескриптор.

```
int imageCreateFromPng(string $filename)
int imageCreateFromJpeg(string $filename)
int imageCreateFromGif(string $filename)
```

Эти функции загружают изображения из файла в память и возвращают его идентификатор. Как и после вызова imageCreate(), дальнейшая работа с картинкой возможна только через этот идентификатор. При загрузке с диска изображение распаковывается и хранится в памяти уже в неупакованном формате, для того чтобы можно было максимально быстро производить с ним различные операции, такие как масштабирование, рисование линий и т. д. При сохранении на диск или выводе в браузер функцией **imagePng()** (или, соответственно, **imageJpeg()** и **imageGif())** картинка автоматически упаковывается.

Стоит заметить, что не обязательно все три функции будут доступны в вашей версии PHP. Скорее всего, в ней не окажется функции **imageCreateFromGif()**, а возможно, и **imageCreateFromJpeg()**, потому что от первой разработчики GD просто отказались, а вторая появилась сравнительно недавно. Надеюсь, в скором времени ситуация нормализуется, но сейчас, к сожалению, это не так.

Определение параметров изображения

Как только картинка создана и получен ее идентификатор, GD становится совершенно все равно, какой формат она имеет и каким путем ее создали. То есть все остальные действия с картинкой происходят через ее идентификатор, вне зависимости от формата, и это логично — ведь в памяти изображение все равно хранится в распакованном виде (наподобие BMP), а значит, информация о ее формате нигде не используется. Так что вполне возможно открыть PNG-изображение с помощью

imageCreateFromPng() и сохранить ее на диск функцией **imageJpeg(),** уже в другом формате. В дальнейшем можно в любой момент времени определить размер загруженной картинки, воспользовавшись функциями **imageSX() и imageSY():**

int imageSX(int $im) Функция возвращает горизонтальный размер изображения, заданного своим идентификатором, в пикселах.

`int imageSY(int $im)` Возвращает высоту картинки в пикселах.

`int imageColorsTotal(int $im)` Эту функцию имеет смысл применять только в том случае, если вы работаете с изображениями, "привязанными" к конкретной палитре — например, с файлами GIF.

Она возвращает текущее количество цветов в палитре. Как мы вскоре увидим, каждый вызов **`imageColorAllocate()`** увеличивает размер палитры. В то же время известно, что если при небольшом размере палитры GIF-картинка сжимается очень хорошо, то при переходе через степень двойки (например, от 16 к 17 цветам) эффективность сжатия заметно падает, что ведет к увеличению размера (так уж устроен формат GIF).

Сохранение изображения

Для сохранения изображений на жесткий диск и для передачи изображений по протоколу HTTP используют следующие функции:

```
int imagePng(int $im [,string $filename])
int imageJpeg(int $im [,string $filename])
int imageGif(int $im [,string $filename])
```

Эти функции сохраняют изображение, заданное своим идентификатором и находящееся в памяти, на диск, или же выводят его в браузер. Разумеется, вначале изображение должно быть загружено или создано при помощи функции imageCreate(), т. е. мы должны знать его идентификатор $im. Если аргумент $filename опущен, то сжатые данные в соответствующем формате выводятся прямо в стандартный выходной поток, т. е. в браузер. Нужный заголовок Content-type при этом не выводится, ввиду чего нужно выводить его вручную при помощи Header(), как это было показано в примере данной лабораторной работы.

Некоторые браузеры не требуют вывода правильного Content-type, а определяют, что перед ними рисунок, по нескольким первым байтам присланных данных. *Ни в коем случае не полагайтесь на это! Дело в том, что все еще существуют браузеры, которые этого делать не умеют в частности MSIE. Кроме того, такая техника идет вразрез со стандартами HTTP. Фактически, вы должны вызвать одну из трех команд, в зависимости от типа изображения:*

`Header("Content-type: image/png")` для PNG

`Header("Content-type: image/jpeg")` для JPEG

`Header("Content-type: image/gif")` для GIF

Рекомендуется их вызывать не в начале сценария, а непосредственно перед вызовом **`imagePng()`**, **`imageGif()`** или **`imageJpeg()`**, поскольку иначе вы не сможете никак увидеть сообщения об ошибках и предупреждения, которые, возможно, будут сгенерированы программой.

Работа с цветом в формате RGB

Наверное, теперь вам захочется что-нибудь нарисовать на пустой или только что загруженной картинке. Но чтобы рисовать, нужно определиться, каким цветом это делать. Проще всего указать цвет заданием тройки RGB-значений (от red-green-blue) — это три цифры от 0 до 255, определяющие содержание красной, зеленой и синей составляющих в нужном нам цвете. Число 0 обозначает нулевую яркость соответствующей компоненты, а 255 — максимальную интенсивность. Например, (0,0,0) задает черный цвет, (255,255,255) — белый, (255,0,0) — ярко-красный, (255,255,0) — желтый и т. д.

Правда, GD не умеет работать с такими тройками напрямую. Она требует, чтобы перед использованием RGB-цвета был получен его специальный идентификатор. Дальше вся работа опять же осуществляется через этот идентификатор. Скоро станет ясно, зачем нужна такая техника.

Создание нового цвета: **`int imageColorAllocate(int $im, int $red, int $green, int $blue)`** Функция возвращает идентификатор цвета, связанного с соответствующей тройкой RGB. Обратите внимание, что первым параметром функция требует идентификатор изображения, загруженного в память или созданного до этого. Практически каждый цвет, который планируется в дальнейшем использовать, должен быть получен (определен) при помощи вызова этой функции. Почему "практически" — станет ясно после рассмотрения функции **`imageColorClosest()`**. Получение ближайшего цвета Давайте разберемся, зачем это придумана такая технология работы с цветами через промежуточное звено — идентификатор цвета. А дело все в том, что некоторые форматы изображений (такие как GIF и частично PNG) не поддерживают

любое количество различных цветов в изображении. А именно, в GIF количество одновременно присутствующих цветов ограничено цифрой 256, причем чем меньше цветов используется в рисунке, тем лучше он "сжимается" и тем меньший размер имеет файл. Тот набор цветов, который реально использован в рисунке, называется его палитрой. Представим себе, что произойдет, если все 256 цветов уже "заняты" и вызывается функция **imageColorAllocate()**. В этом случае она обнаружит, что палитра заполнена полностью, и найдет среди занятых цветов тот, который ближе всего находится к запрошенному — будет возвращен именно его идентификатор. Если же "свободные места" в палитре есть, то они и будут использованы этой функцией (конечно, если в палитре вдруг не найдется точно такой же цвет, как запрошенный — обычно дублирование одинаковых цветов всячески избегается).

int imageColorClosest(int $im, int $red, int $green, int $blue) Наверное, вы уже догадались, зачем нужна функция **imageColorClosest()**. Вместо того чтобы пытаться выискать свободное место в палитре цветов, она просто возвращает идентификатор цвета, уже существующего в рисунке и находящегося ближе всего к затребованному. Таким образом, нового цвета в палитру не добавляется. Если палитра невелика, то функция может вернуть не совсем тот цвет, который вы ожидаете. Например, в палитре из трех цветов "красный-зеленый-синий" на запрос желтого цвета будет, скорее всего, возвращен идентификатор зеленого — он "ближе всего" с точки зрения GD соответствует понятию "зеленый".

Копирование изображений

int imageCopyResized(int $dst_im, int $src_im, int $dstX, int $dstY, int $srcX, int $srcY, int $dstW, int $dstH, int $srcW, int $srcH)

Эта функция — одна из самых мощных и универсальных, хотя и выглядит просто ужасно. С помощью нее можно копировать изображения (или их участки), перемещать и масштабировать их и т.д.. Итак, $dst_im задает идентификатор изображения, в который будет помещен результат работы функции. Это изображение должно уже быть создано или загружено и иметь надлежащие размеры. Соответственно, $src_im — идентификатор изображения, над которым проводится работа. Впрочем, $src_im и $dst_im могут и совпадать. Параметры ($srcX, $srcY, $srcW, $srcH) (обратите внимание на то, что они следуют при вызове функции не подряд!) задают область внутри исходного изображения, над которой будет осуществлена операция — соответственно, координаты ее верхнеголевого угла, ширину и высоту. Наконец, четверка ($dstX, $dstY, $dstW, $dstH) задает то место на изображении $dst_im, в которое будет "втиснут" указанный в предыдущей четверке прямоугольник. Заметьте, что, если ширина или высота двух прямоугольников не совпадают, то картинка автоматически будет нужным образом растянута или сжата.

Таким образом, с помощью функции imageCopyResized() мы можем:

- копировать изображения;
- копировать участки изображений;
- масштабировать участки изображений;
- копировать и масштабировать участки изображения в пределах одной картинки.

В последнем случае возникают некоторые сложности, а именно, когда тот блок, из которого производится копирование, частично налагается на место, куда он должен быть перемещен. Убедиться в этом проще всего экспериментальным путем.

Работа с фиксированными шрифтами

Библиотека GD имеет некоторые возможности по работе с текстом и шрифтами. Шрифты представляют собой специальные ресурсы, имеющие собственный идентификатор, и чаще всего загружаемые из файла или встроенные в GD. Каждый символ шрифта может быть отображен лишь в моноцветном режиме, т. е. "рисованные" символы не поддерживаются. Встроенных шрифтов всего 5 (идентификаторы от 1 до 5), чаще всего в них входят моноширинные символы разных размеров. Остальные шрифты должны быть предварительно загружены.

Загрузка шрифта **int imageLoadFont(string $file)** Функция загружает файл шрифтов и возвращает идентификатор шрифта — это будет цифра, большая 5, потому что пять первых номеров зарезервированы как встроенные. Формат файла — бинарный, а потому зависит от архитектуры машины.

Параметры шрифта. После того как шрифт загружен, его можно использовать (встроенные шрифты, конечно же, загружать не требуется).

int imageFontHeight(int $font) Возвращает высоту в пикселах каждого символа в заданном шрифте.

int imageFontWidth(int $font) Возвращает ширину в пикселах каждого символа в заданном шрифте.

Вывод строки **int imageString(int $im, int $font, int $x, int $y, string $s, int $col)** Выводит строку **$s** в изображение **$im**, используя шрифт **$font** и цвет **$col**. Координаты **($x,$y)** будут координатами левого верхнего угла прямоугольника, в который вписана строка.

int imageStringUp(int $im, int $font, int $x, int $y, string $s, int $c) Эта функция также выводит строку текста, но не в горизонтальном, а в вертикальном направлении. Верхний левый угол, по-прежнему, задается координатами **($x,$y)**.

Работа со шрифтами TrueType

Библиотека GD поддерживает также работу со шрифтами PostScript и TrueType. Мы с вами рассмотрим только последние, т. к., во-первых, их существует великое множество (благодаря платформе Windows), а во-вторых, с ними проще всего работать в PHP.

Для того чтобы заработали приведенные ниже функции, PHP должен быть откомпилирован и установлен вместе с библиотекой FreeType, доступной по адресу http://www.freetype.org. В Windows-версии PHP она установлена по умолчанию. Всего существует две функции для работы со шрифтами TrueType. Одна из них выводит строку в изображение, а вторая — определяет, какое положение эта строка бы заняла при выводе. Вывод строки **list imageTTFText(int $im, int $size, int $angle, int $x, int $y, int $col, string $fontfile, string $text)**

Эта функция помещает строку **$text** в изображение **$im** цветом **$col**. Как обычно, **$col** должен представлять собой допустимый идентификатор цвета. Параметр **$angle** задает угол наклона в градусах выводимой строки, отсчитываемый от горизонтали против часовой стрелки. Координаты **($x,$y)** указывают положение так называемой базовой точки строки (обычно это ее левый нижний угол). Параметр **$size** задает размер шрифта, который будет использоваться при выводе строки. Наконец, **$fontfile** должен содержать имя TTF-файла, в котором, собственно, и хранится шрифт.

Параметр **$fontfile** должен всегда задавать абсолютный путь (от корня файловой системы) к требуемому файлу шрифтов. Функция возвращает список из 8 элементов. Первая их пара задает координаты **(x,y)** верхнего левого угла прямоугольника, описанного вокруг строки текста в изображении, вторая пара — координаты верхнего правого угла, и т. д. Так как в общем случае строка может иметь любой наклон **$angle**, здесь требуются 4 пары координат.

Пример работы с изображениями

Данный пример показывает, как можно работать с изображениями, делая так называемые «превьюшки»: Данный примет можно использовать при создании галерей изображений. На выходе скрипта изображение в формате PNG. Передача параметра осуществляется следующим способом:

```
Имя_файла_скрипта.php?Имя_файла_изображения
```

```
<?
// Получаем строку, которую нам передали в параметрах
$img=urldecode($QUERY_STRING);
// определяем расширение файла
$ext = strtolower(substr($img,-3));
// Загружаем рисунок с диска
if ($ext=='png')$im = imageCreateFromPng($img);
if ($ext=='gif')$im = imageCreateFromGif($img);
if ($ext=='jpg')$im = imageCreateFromJpeg($img);
// создаем переменную для хранения отношения сторон
$norma=(int)imageSY($im)/imageSX($im);
```

```
	//создаем новый рисунок для превьюшки шириной 100
	$resnew = imagecreate(100, 100*$norma);
	// копируем изображение в созданный файл
	imagecopyresized($resnew, $im, 0, 0, 0, 0, 100, $norma*100,
imageSX($im), imageSY($im));
	// Создаем в палитре новый цвет - черный
	$black = imageColorAllocate($resnew, 0, 0, 0);
	$string=date("d.m.Y H:i:s",filemtime($img));
	// Выводим строку поверх того, что было в загруженном изображении
	// getcwd() выводит абсолютный путь к файлу times.ttf
	// Шрифт используется times new roman
	imagettftext($resnew,10,0,5,imageSY($resnew)-
8,$black,getcwd()."/times.ttf",$string);
	// Сообщаем о том, что далее следует рисунок PNG
	// смотрите лекции по протоколу HTTP
	Header("Content-type: image/png");
	// Теперь - самое главное: отправляем данные картинки в
	// стандартный выходной поток, т. е. в браузер
	imagePng($resnew);
	// В конце освобождаем память, занятую картинками
	imageDestroy($im);
	imageDestroy($resnew);
	?>
```

Пример работы скрипта приведен на рисунке 1.

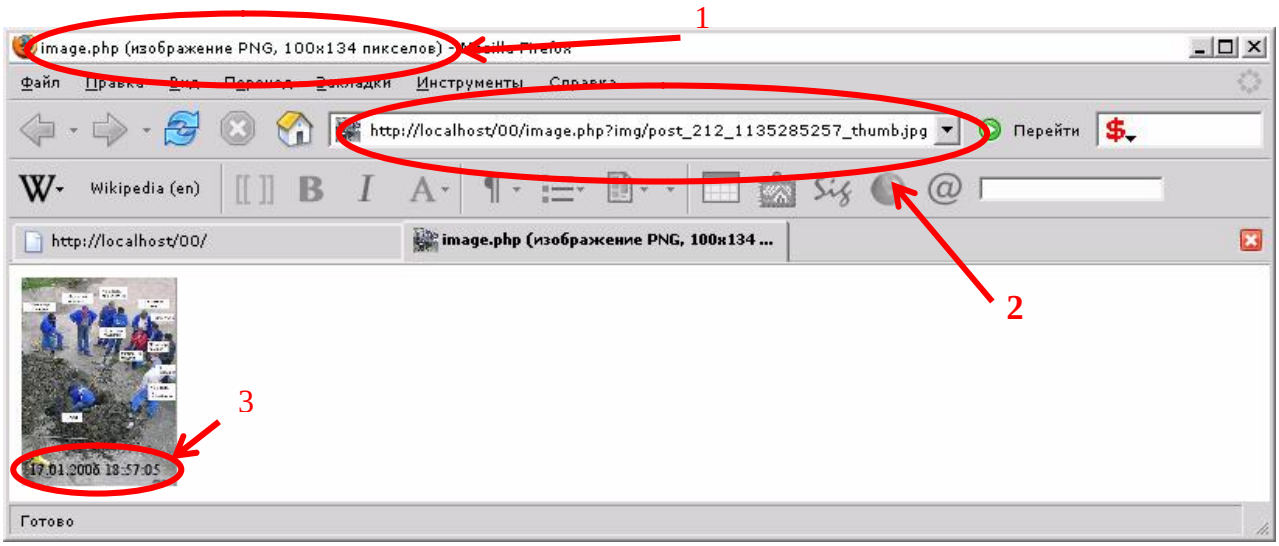


Рисунок 1

Как видно из стрелки №1 скрипт выдает рисунок, из стрелки №2 параметры запроса, стрелка №3 показывает текст созданный на лету (дата создания файла). Замечания в зависимости от версии библиотеки GD может не работать функция работы с gif изображениями. Предпочтительней использовать для вывода формат PNG. На рисунке №2 представлен оригинальный вид картинки

Рисунок 2

Загрузка файлов на сервер

Multipart-формы

Мы помним, что в большинстве случаев данные из формы в браузере, передающиеся методом GET или POST, приходят к нам в одинаковом формате: **`поле1=значение1&поле2=значение2&...`** При этом все символы, отличные от "английских" букв и цифр (и еще некоторых) URL-кодируются: заменяются на %XX, где XX — шестнадцатеричный код символа.Это сильно замедляет закачку больших файлов. В принципе, multipart-формы призваны одним махом решить эту проблему. Нам нужно в соответствующем тэге <form> задать параметр: enctype=multipart/form-data После этого данные, полученные от нашей формы, будут разбиты на несколько блоков информации (по одному на каждый элемент формы). Каждый такой блок очень похож на обычную посылку "заголовки-данные" протокола HTTP:

----------Идентификатор_начала\n
`Content-Disposition: form-data; name="имя" [;другие параметры]\n`
`\n`
`значение\n`

Браузер автоматически формирует строку **`Идентификатор_начала`** из расчета, чтобы она не встречалась ни в одном из передаваемых файлов (и ни в одном из других полей формы). Это означает, что сегодня идентификатор будет одним, а завтра, возможно, совсем другим.

Тэг выбора файла

Тэг **`<input>`** надо вставить в форму, чтобы в ней появился элемент управления загрузкой файла — текстовое поле с кнопкой Browse справа. **`<input type=file name=имя_элемента [size=размер_поля]>`** Сценарию вместе с содержимым файла передается и некоторая другая информация, а именно: размер файла; имя файла в системе клиента; тип файла.

Закачка файлов и безопасность

Возможно, вы обратили внимание на то, что у последнего приведенного тэга **`<input type=file>`** отсутствует атрибут **`value`**. То есть когда пользователь открывает страницу, он никогда не увидит в элементе закачки ничего, кроме пустой строки. Поначалу это кажется довольно неприятным ограничением: в самом деле, мы ведь можем задавать параметры по умолчанию для, скажем, текстового поля. Разработчики HTML пошли на такое исключение из общего правила специально. С их помощью JavaScript и можно создавать интерактивные страницы, реагирующие на действия пользователя в реальном времени. Например, можно написать код на JavaScript, который запускается, когда пользователь нажимает какую-нибудь кнопку в форме на странице, или когда он вводит текст в одно из текстовых полей.

Поддержка закачки в PHP

Так как PHP специально разрабатывался как язык для Web-приложений, то, естественно, он "умеет" работать как с привычными формами, так и с multipart-формами. Более того, он также поддерживает закачку файлов на сервер.

Интерпретатору совершенно все равно, в каком формате приходят данные из формы. Он умеет их обрабатывать и "рассовывать" по переменным в любом формате. Однако данные одного специального поля формы — а именно, поля закачки — он интерпретирует особым образом. Пусть у нас есть multipart-форма, а в ней — поле закачки файла:

```
<form action=index.php method=POST enctype=multipart/form-data>
<input type=file name=MyFile><br>
<input type=submit name=doUpload value="Закачать новую фотографию">
</form>
```

После выбора в этом поле нужного файла и отправки формы (и загрузки на сервер того файла, который был указан) PHP определит, что нужно принять файл, и сохранит его во временном каталоге на сервере. Кроме того, в программе создадутся несколько переменных. **`$MyFile`** — имя временного файла на машине сервера, который содержит данные, переданные пользователем. С этим файлом теперь можно вытворять все что угодно: удалять, копировать, переименовывать. Название переменной равно значению поля в форме отправки

Значения и имена переменных при передаче данных на сервер:

$MyFile_name — исходное имя файла, которое он имел до своей отправки на сервер.

$MyFile_size — размер закачанного файла в байтах.

$MyFile_type — тип загруженного файла, если браузер смог его определить. К примеру, **image/gif, text/html** и т. д.

Как видим, префикс у всех созданных переменных один и тот же — **MyFile_**. Этот префикс состоит из имени элемента закачки в форме, к которому присоединен знак _. Теперь мы можем, например, скопировать только что полученный файл на новое место, при помощи **Copy($MyFile,"имя_файла_на_сервере")**, проверив предварительно, не слишком ли он велик, основываясь на значении переменной **$MyFile_size**.

Если процесс окончится неуспешно, вы сможете определить это по отсутствию файла, имя которого задано в **$MyFile**, или же по отсутствию самой этой переменной в программе.

Настоятельно рекомендуется использовать функцию копирования, а не переименования/перемещения. Дело в том, что в большинстве случаев временный каталог, в котором PHP хранит только что закачанные файлы, находиться на другом носителе или логическом разделе, и в результате операция переименования завершится с ошибкой. Хотя мы и оставили копию полученного файла во временном каталоге, можно не заботиться о его удалении в целях экономии места: PHP сделает это автоматически, после завершения работы скрипта.

Обратите внимание, что в настройках php имеется ограничения на объем передаваемых методом POST данных по умолчанию это значение достаточно мало:

post_max_size = 8M

file_uploads = On разрешить загрузку файлов на сервер

upload_tmp_dir = указать папку для хранения временных файлов

upload_max_filesize = 2M максимальный объем одного загружаемого файла

При подобных ограничениях мы можем одновременно загрузить только 3 файла объемом 2 Мегабайта

Сложные имена полей

Как вы, наверное, помните, элементы формы могут иметь имена, выглядящие, как элементы массива: **A[10], B[1][text]** и т. д. До недавнего времени (в третьей версии PHP) это касалось только "обычных" полей, но не полей закачки файлов. К счастью, в PHP версии 4 все изменилось в лучшую сторону.

```
<form action="script.php" method=POST enctype=multipart/form-data>
<h3>Выберите тип файлов в вашей системе:</h3>
Текстовый файл: <input type=file name="File[text]"><br>
Бинарный файл: <input type=file name="File[bin]"><br>
Картинка: <input type=file name="File[pic]"><br>
<input type=submit name=Go value="Отправить файлы">
</form>
```

После того как программа **script.php** примет данные из формы, PHP создаст для нее следующие переменные: ассоциативный массив **$File**, ключи которого — **text, bin** и **pic**, а соответствующие значения — имена временных файлов на сервере, созданных PHP при загрузке; массив **$File_name** все с теми же ключами и значениями — именами файлов в системе пользователя; массив **$File_type** с теми же ключами и значениями — типами соответствующих файлов; массив **$File_size** со значениями — размерами этих файлов. Мы видим, информация об индексах в именах полей формы попала в ключи соответствующих массивов и сохранилась в них. Вы можете убедиться в том, что переменные действительно инициализированы, воспользовавшись вызовом функции

Пример работы для загрузки изображений

Исходный текст формы загрузки данных на сервер:

```
<html><head><title>Выберите файл картинки для загрузки на сервер
</title></head>
<body> <h1>Выберите изображение в формате png, jpeg или gif</h1>
<form action=upload.php method=POST enctype=”multipart/form-data”>
```

```
<input type=file name=File><br>
<input type=submit name=doUpload value="Закачать новую фотографию">
</form></body></html>
```

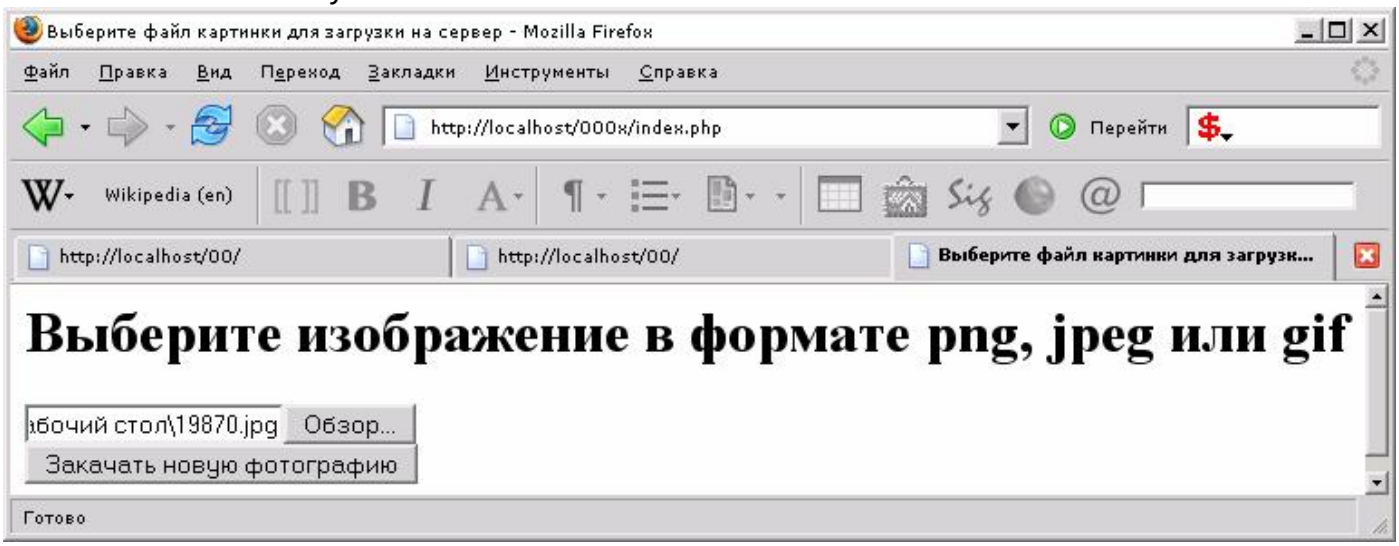


Рис. 3 Внешний вид формы загрузки изображений.

Исходный текст скрипта обработчика приема данных **upload.php**

```
<?
$ImgDir="img";       // Каталог для хранения изображений
@mkdir($ImgDir,666); // Создаем, если его еще нет с правом доступа на запись
    // Проверяем, принят ли файл
  if(file_exists($File)) {
     // Все в порядке - добавляем файл в каталог с фотографиями
     // Используем то же имя, что и в системе пользователя
     Copy($File,"$ImgDir/".basename($File_name));
  }
// Теперь считываем в массив наш фотоальбом
$d=opendir($ImgDir);  // открываем каталог
$Photos=array();      // изначально альбом пуст
// Перебираем все файлы
while(($e=readdir($d))!==false) {
  // Это изображение GIF, JPG или PNG?
  if(!ereg("^(.*)\\.(gif|jpg|png|JPG|GIF|PNG)$",$e,$P)) continue;
  // Если нет, переходим к следующему файлу,
  // иначе обрабатываем этот
  $path="$ImgDir/$e";      // адрес
  $sz=GetImageSize($path); // размер
  $tm=filemtime($path);    // время добавления
  // Вставляем изображение в массив $Photos
  $Photos[$tm] = array(
    'time' => filemtime($path), // время добавления
    'name' => $e,               // имя файла
    'url'  => $path,            // его URI
    'w'    => $sz[0],           // ширина картинки
    'h'    => $sz[1],           // ее высота
    'wh'   => $sz[3]             // "width=xxx height=yyy");
);
}
// Ключи массива $Photos - время в секундах, когда была добавлена
// та или иная фотография. Сортируем массив: наиболее "свежие"
// фотографии располагаем ближе к его началу.
```

```
krsort($Photos);
// Данные для вывода готовы. Дело за малым - оформить страницу.
?>
<body> <h1>Галерейка Изображений</h1>
<?foreach($Photos as $n=>$Img) {?>
<a href=<?=$Img['url']\?>>  <img
   src=<?=$Img['url']\?>
   alt="Добавлена <?=date("d.m.Y H:i:s",$Img['time'])?>"
  > </a></br>
<?}?>
</body>
```

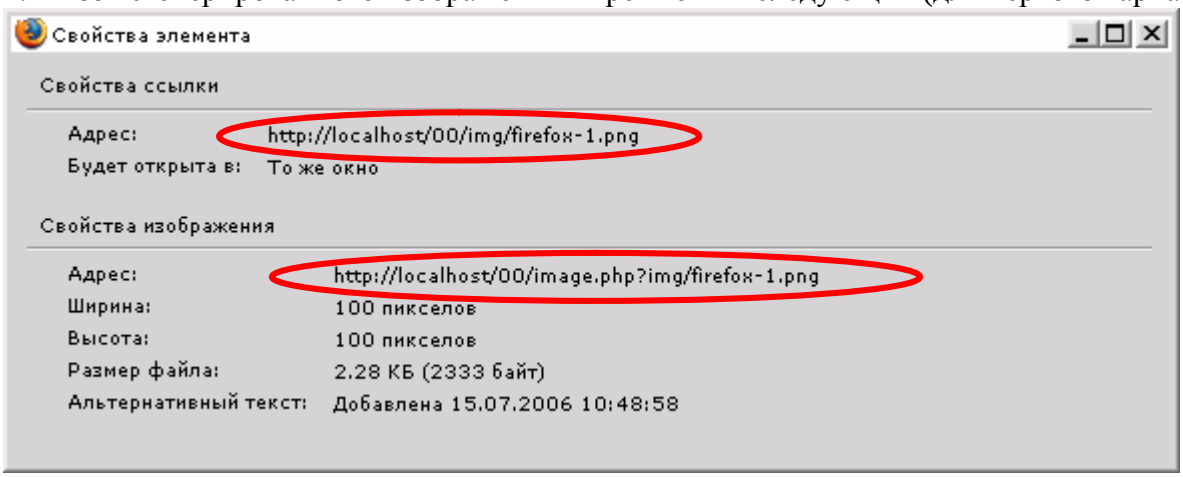
Пример взят из книги Дмитрия Котерова «Самоучитель по PHP4» (модифицированный)

Задания для самостоятельного выполнения

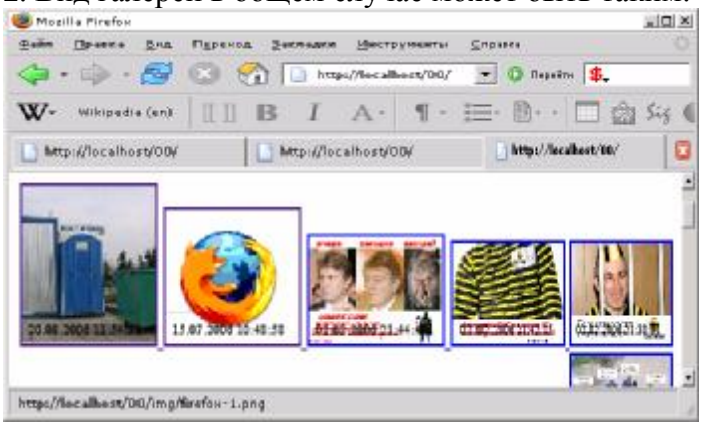
1. Доработать пример работы с изображениями таким образом, чтобы можно было с помощью параметров менять тип выдаваемого файла, текст на превью, размер превью файла, цвет шрифта.
2. Доработать скрипт загрузки таким образом, чтобы для загружаемых изображений автоматически создавались файлы «превью» и сохранялись в отдельную папку, например **`mini_img`**
3. Выполнить вторую версию скрипта обработки загруженных изображений и формы загрузки: Добавить поля название фотографии и краткое описание. Принимать данные необходимо с сохранением информации в таблице базы MySQL (имя файла, дата загрузки, название фотографии и краткое описание). Саму галерею разбить на несколько страниц, например по 10 фотографий, причем можно использовать модифицированный пример из задания 1

Замечания по ходу выполнения работы

1. Вызов сгенерированного изображения - превьюшки следующий (для первого варианта):



2. Вид галереи в общем случае может быть таким:



3. Лабораторная работа рассчитана на 4 академических часа с установкой серверного Программного обеспеченья.
4. Отличия между GD и GD2 могут быть в функциональности каждой из библиотек
5. Дополнительную информацию можно получить на сайте разработчика http://ru.php.net/
6. Обе версии GD с php 4.3.1 не поддерживают работу с форматом gif. Про более новые версии можете узнать экспериментальным путем.