

Contents

1	General Guidelines	1
1.1	To the Instructor	2
1.2	To the Student	2
2	Basic HTML	3
2.1	HTML Document Structure	3
2.2	Attributes and values	4
2.3	Font Formatting	4
2.4	Headings	5
2.5	Lists	5
2.6	Paragraphs, Breaks, and Pre-Formatted Text	6
2.7	Images: Size, Borders and Alignment	6
2.8	Links	7
2.9	Tables	7
2.10	Frames and Applets	7
2.11	HTML Forms	8
2.12	Validators and Other Resources	8
2.13	Network Protocols	9
3	Installing Apache	11
3.1	Building Apache	11
3.2	Running Apache	14
3.3	Minute Papers and Exercises	16
3.4	Homework Assignment One	16
4	Unix Commands	17
4.1	pwd, ls, man, mkdir, cd, cat (plus ~, . and ..)	17
4.2	tar, gzip, du, chmod, ps, grep	19
4.3	Startup and Configuration Files: ~/.bashrc	20
5	Introduction to Perl	21
5.1	A Review of Basic Concepts	21
5.2	What is CGI?	22
5.3	Simple CGI Scripts	23

5.4	User Input with CGI	24
5.5	%ENV, the Environment Hashtable	25
5.6	Keeping State	28
5.7	CGI and HTML Forms	29
5.8	Minute Papers and Exercises	32
5.9	Homework Assignment Two	32
6	User Authentication	33
6.1	The Password File	33
6.2	Server Configuration	34
7	HTTP	35
7.1	Experiment One	36
7.2	Experiment Two	37
7.3	Experiment Three	38
7.4	Experiment Four	38
7.5	Minute Papers and Exercises	40
8	Relational Databases	41
8.1	Storing and Retrieving Information	42
8.2	The Entity-Relationship Diagram	44
8.2.1	Entities	44
8.2.2	Relationships	45
8.3	Homework Assignment Three	46
9	MySQL	47
9.1	Installation of MySQL	47
9.2	Testing Your Installation	50
9.3	Minute Papers and Exercises	50
10	PHP	51
10.1	Installation of PHP	51
10.2	PHP Configuration	53
10.3	PHP Programming	55
10.3.1	Basic Concepts	55
10.3.2	User Input with PHP	56
10.3.3	HTML Forms in PHP	58
10.3.4	Keeping State on the Client Side	58
10.3.5	Keeping State on the Server Side	59
10.4	Homework Assignment Four	61
10.5	Program Transformations	61
10.6	Minute Papers and Exercises	65
10.7	Sample Midterm Exam	66

11 The DBI.pm Module	67
11.1 Generating Session IDs	67
11.2 Session Management	69
11.3 Program Transformations	74
11.4 Minute Papers and Exercises	82
11.5 Homework Assignment Five	82
12 Accidental Reloads	83
12.1 Minute Papers and Exercises	84
13 State Machines on the Web	85
13.1 The Chart	85
13.2 Nodes	86
13.3 Data	87
13.4 Data Access	88
13.5 Transitions	89
13.6 PHP Functions	91
13.7 Minute Papers and Exercises	93
13.8 Stage Five	96
13.9 Stage Six	99
13.10 Stage Seven (Last Stage)	103
13.11 Conclusion	108
14 Security Concerns in PHP and MySQL	109
14.1 PHP and Superglobals	109
14.2 MySQL Injection	111
15 Javascript and DHTML	113
15.1 Programming the Browser	113
15.2 Object-Oriented JavaScript	119
15.3 and .innerHTML	124
15.4 Homework Six	128
15.5 Minute Papers and Exercises	128
16 Introduction to Java	131
16.1 Basic Objects	131
16.2 The Class Extension Mechanism	132
16.3 Dynamic Method Lookup	132
16.4 Applets	133
16.5 Minute Papers and Exercises	134
17 Apache Tomcat	135
17.1 Installation of Tomcat	136
17.2 Tomcat Deployment: Contexts	140
17.3 A Simple Java Servlet	141
17.4 Minute Papers and Exercises	145

17.5 A Simple JSP	146
18 Server-Side Java	147
18.1 The Basic Servlet Template	147
18.2 The Manager Application	147
18.3 User Input, Sessions	148
18.4 Transformation to JSP	150
18.5 Homework Seven	152
18.6 A Note on Using Sessions	152
19 Java Database Connectivity	155
19.1 Basic Driver Installation	155
19.1.1 Using the CLASSPATH	155
19.1.2 The lib Folder	155
19.2 Standalone Database Access	155
19.2.1 Creating Tables	156
19.2.2 Inserting Data	157
19.2.3 Retriving Information	158
19.3 Servlet Database Access	159
20 Commencement	161

Chapter 1

General Guidelines

This is a one semester course consisting of about thirty lectures, and about half as many labs, whose aim is to present in a concise but consistent manner the essential concepts of web programming and using today's most popular development tools. To this end we have chosen to structure the course around four of the most successful open-source projects ever¹ and using a variety of languages and protocols² widely in use today.

Although no prior web programming experience is assumed, the reader is expected to have a reasonably good exposure to, and knowledge of, programming in general. Two semesters are the typical prerequisite for this course. Prior programming experience is important because it makes sure the student taking this class is aware of the very specific nature of this activity: it requires a fair amount of patience, diligence, a certain type of attention to detail and a very strong commitment to planning. But it also helps bring forth these abilities and traits in people.

Labs are a central feature of this course. There are two lectures and one lab each week, but a lab last about twice as much (115 minutes per lab compared to the 75 minutes used for each lecture). Programming is a very rewarding activity but it also requires practice. It definitely isn't a spectator's sport. There are seven homework assignments and twelve lab assignments, a semester project and two written exams in this course, each worth approximately a fifth of the grade (nineteen percent, to be more precise; the rest five percent is collected as minute paper points: in each lecture short (one-minute) questions are asked and student answers are submitted on paper to the instructor at the end of the lecture. Although their main purpose is to tally attendance points, minute papers provide instantaneous feedback to both the student and the lecturer and are usually a very useful and effective communication tool.) The grading scale is organized accordingly³.

This course differs from others in that it offers a truly unified approach to

¹Apache 2.2.2, MySQL 5.0.22, PHP 5.1.4 and Tomcat 5.5.17.

²Perl, Perl DBI, HTTP, HTML, CGI, SQL, PHP, JavaScript, DHTML, Java and JDBC

³Additional details can be found at <http://www.cs.indiana.edu/classes/a348>

all major technologies in use today. At first this might seem as yet another potentially challenging aspect of the course: a new pattern needs to be learned. However, the approach taken is such that students are gently guided to a stage in which, having seen what is available and what is required, they can in fact discover the pattern on their own. And once the template (or pattern) is established, with gentle reinforcement and by using it in the successive contexts and technologies presented in class the initial investment pays off.

1.1 To the Instructor

It has been said that in every big book there is a little book trying to get out. We have tried hard to make this text as comprehensive and concise as we could. To the motivated student the course has a lot to offer: a quick introduction to the Linux (Unix) operating system; installation and maintenance of the Apache web server; installation of MySQL; a quick introduction to RDBMS (relational database management systems); an introduction to Perl, CGI and Perl/DBI (database interface) programming; installation of PHP as an Apache module and PHP database programming; an introduction to JavaScript, DHTML and the W3C DOM; a quick introduction to Java; installation and maintenance of the Jakarta Tomcat web server; an introduction to server-side Java (Java servlets and JSP); an introduction to JDBC and Java database access; numerous examples of web programs using the technologies mentioned above.

It is our experience that students are strongly motivated by the variety of topics covered in this class. And while they appreciate the simplicity in the unified approach taken they also require a certain amount of assistance to master it. This type of assistance differs from student to student, and topic to topic but it all boils down to: discipline, confidence and patience. Most of the errors are going to be hard to find but easy to fix, so assistance is truly important especially in the early stages. We hope you like this text and that you will find it useful. But you are far more important to your students—and while we believe that the approach we took and the examples we prepared are sound and have instructional value, there's much more than an instructor using this text could add to it during a regular semester. Additional examples and larger projects can be found at the URL mentioned earlier.

1.2 To the Student

This is a web programming course which touches on a number of administrative issues. You have taken programming before; do not despair if at any time you find the course somewhat challenging. The web is a huge distributed platform and it functions in a purely event-driven manner. It may take time to understand this aspect thoroughly. Be patient and stay close to your instructor. The most important thing you will need to remember as we move ahead is: you can do it.

Now let's get started.

Chapter 2

Basic HTML

Do you Yahoo¹?

Behind every web address stands a web server, likely an Apache web server, perhaps running under Unix. For every browser request, the reply comes encoded in HTML: the hypertext markup language that allows seamless interconnection of information over the network.

Not all HTML is typed by humans, some HTML pages are the output of programs written by humans. But the encoding is always the same. This week's set of lectures and lab will introduce you to Linux (an open-source variety of Unix), HTML, and the Apache web server.

At the end of this week you should be able to:

- install, start, restart, stop, maintain a web server
- understand the Linux environment and the commands most needed
- code anything in HTML and publish it on your web server

We start with HTML.

To learn HTML you don't need a network connection. You only need a browser². We start with a review of basic HTML, which we summarize below.

2.1 HTML Document Structure

This is probably the simplest possible example.

```
<html>
  <head><title>This is a title</title></head>
  <body>
    This is my first HTML document.
  </body>
</html>
```

¹<http://www.yahoo.com>

²So one other key question becomes: do you *know* your browser?

Using a text editor type the code above into a file, call it anything you want³ but make sure its extension is `.html` and load the file in your browser.

HTML documents have a nested, tree-like structure. Try drawing a diagram for the document above: start with a root node, and label it `html`. There will be two children nodes below it, labeled `head` and `body` respectively. Text content hangs off the node for the document's body. A `title` node is the sole off-spring of the `head` node, and its contents is also text.

2.2 Attributes and values

Take a look how the syntax allows the specification of attributes for HTML nodes; below we specify a background color for the document's body:

```
<html>
  <head><title>This is a title</title></head>
  <body bgcolor=white>
    This is my first HTML document.
  </body>
</html>
```

Instead of `white` one could also try a different color. The next section uses this shade of blue: `#0066cc`. The digits (or characters) indicate, in pairs and in hexadecimal the amount of red, green and blue, in that order.

2.3 Font Formatting

A portion of text could change color by having its font attribute specified:

```
<html>
  <head><title>This is a title</title></head>
  <body bgcolor=white>
    This is my first HTML document. <p>

    This <font color=red>word</font> will appear in red.

    This one will be
    <font size=+6>bigger</font> and somewhat
    <font color="#0066ff">blue</font>.
  </body>
</html>
```

Changing the background color can be accomplished as follows:

```
The <span style="background-color: #FFCCCC">pink</span> panther.
```

The `` tag will be particularly useful in the JavaScript chapter.

³A good name would be `one.html` if you can't think of any.

2.4 Headings

There are six levels of headings represented by the pairs of tags `<h1>` `</h1>` through `<h6>` `</h6>`; we will not make much use of them. For the most part these tags will help us get a certain font size with minimum effort (typing).

```
<body bgcolor=white>
  Here are some headings.

  <h1> Heading One  </h1>
  <h2> Heading Two  </h2>
  <h3> Heading Three </h3>
  <h4> Heading Four </h4>
  <h5> Heading Five </h5>
  <h6> Heading Six  </h6>
```

```
  This is my first HTML document.
```

Note: to save space only a minimal HTML context is shown.

2.5 Lists

The following tags are useful with enumerations:

```
<body bgcolor=white>

  Some of the new things we will learn in this class:

  <ul>
    <li> designing a relational database
    <li> writing CGI scripts in Perl
    <li> installing and maintaining Tomcat
    <li> understanding HTTP
  </ul>

  The first four steps will be, in order:

  <ol>
    <li> HTML, Unix, Apache
    <li> Perl and CGI
    <li> Installing MySQL
    <li> Database access using Perl DBI
  </ol>
```

The `` tag is usually also used in pairs (`` ``). In fact it is recommended that we group in pairs all the tags that come in pairs, so that the resulting HTML can be easily understood (parsed) by a computer program. This brings it closer in spirit to XML and this version of HTML in which every element must be explicitly closed is called XHTML. XHTML is actually a lot more than that, but overall this is the most visible feature for us. Throughout

this text we will indicate additional references like this⁴, wherever appropriate and/or necessary.

2.6 Paragraphs, Breaks, and Pre-Formatted Text

The following example introduces the `<p>`, `
` and `<pre>` tags:

```
<html>
  <head><title>This is a title</title></head>
  <body bgcolor=white>

    <p> New paragraph.
    <p> New paragraph.
    <p> New paragraph. <br> A line break.
    <br> Another line break.

    This text
    will be rendered
    normally, on one line. <p>

    <pre>This text
    appears in between
    preformatting tags
    and therefore
    will stay
    as
    typed.</pre>

    You get the idea.

  </body>
</html>
```

Exercise: Take a pencil and make the document above (as well as the one in the preceding section) XHTML-compliant.

Since the `
` tag does not have a closing counterpart, the recommendation is, just like for images, to write it using a forward slash: `
`. This indicates that it ends right where it occurs and is thus, in effect, its own matching tag.

2.7 Images: Size, Borders and Alignment

Given the picture's URL we refer to it in an HTML document as follows:

```

```

Pictures could be aligned and resized on the page:

```

```

⁴<http://en.wikipedia.org/wiki/XHTML>

2.8 Links

The syntax for hyperlinks closely resembles the one for images, with the sole difference that the link syntax does provide a matching tag:

```
<body bgcolor=white>
Click
<a href="http://www.cs.indiana.edu/classes/a348">here</a>
for daily updates on the class notes.
```

While images are retrieved at the same time that the document is loaded, link tags tell the browser to render and wait for the user to initiate the transfer, which happens when the user finally clicks on the link.

2.9 Tables

We will need this tag only for formatting purposes: some formatting of our output will be desirable and this will be the extent to which we will be concerned with the appearance of our HTML documents.

The `table` tag is not complicated: it is in fact composed of a list of row tags, with each row tag grouping together a list of individual cell tags, which hold the actual data.

```
<body bgcolor=white>
<table border cellpadding=6>
  <tr> <td>(1, 1)</td> <td>(1, 2)</td> <td>(1, 3)</td> </tr>
  <tr> <td>(2, 1)</td> <td>(2, 2)</td> <td>(2, 3)</td> </tr>
</table>
```

Cells can span more than one column and more than one line.

```
<table border cellpadding=6>
  <tr> <td rowspan=2 align=center> One </td>
    <td colspan=2 align=center> Two </td>
  </tr>
  <tr> <td align=center bgcolor=lightgrey> Three </td>
    <td align=center> Four </td>
  </tr>
</table>

</body>
</html>
```

2.10 Frames and Applets

These two tags won't be illustrated until much later but we should say a few words about each one of them here.

Using frames allows you to display more than one web page at a time, although they may appear to be just one page. Frames divide a browser window

into sections, with each section being an HTML document. We will look at frames a bit later and in a very specific context.

Applet tags are somewhat similar to image tags: they instruct the browser to load a trusted small application (hence, applet) or program from a remote host and run it. We will look at applets towards the end of the course, during the larger, server-side Java Tomcat section.

2.11 HTML Forms

HTML forms are a very important tool used in collecting user input⁵.

Here is a form with a few basic elements that we will need in order to get started. Note that the form is missing some of the attributes needed for CGI⁶; they will be introduced in due time.

```
<body>
  <form>
    Username: <input type="text"> <p>
    Password: <input type="password"> <p>
    What is the capital of Italy?
    <blockquote>
      <input type="radio" name="question"> Milan
      <input type="radio" name="question"> Turin
      <input type="radio" name="question"> Rome
    </blockquote>
    Presidents of the United States (check all that apply):
    <blockquote>
      <input type="checkbox" name="q2"> Ross Perot
      <input type="checkbox" name="q2"> Bill Clinton
      <input type="checkbox" name="q2"> George Bush
      <input type="checkbox" name="q2"> Al Gore
    </blockquote>
    <select name="capital">
      <option> What
      <option> Milan
      <option> Rome
      <option> Turin
    </select> is the capital of Italy.
  </form>
```

2.12 Validators and Other Resources

The Bare Bones Guide to HTML⁷ contains a clear description of all the HTML tags in the standard and a few others. A great many tutorials about a great many things are available here⁸ including HTML, XHTML and XML. Valida-

⁵The emphasis is on the GUI aspect of the form.

⁶Common Gateway Interface (basically web scripting)

⁷<http://werbach.com/barebones/>

⁸<http://www.w3schools.com>

tors⁹ exist, they allow you to check your HTML for compliance to the standard.

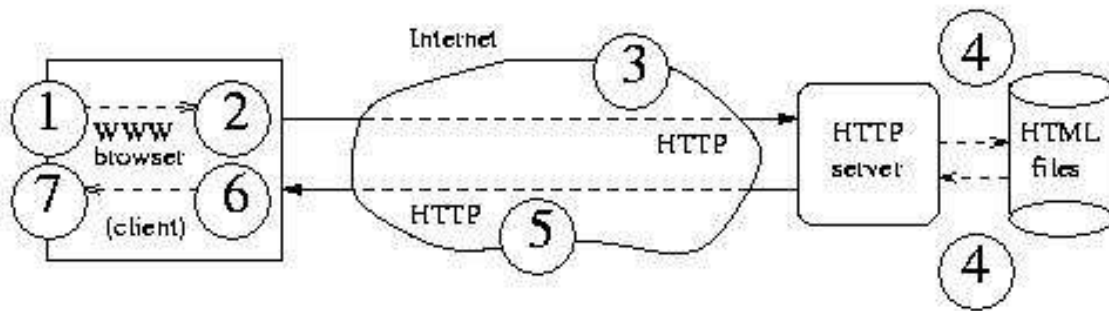
To practice with HTML all you need is a browser: create a file (using any editor, perhaps even `notepad`) and load it in Firefox. Your first assignment will ask you to set up Apache and put together a small web site, adding your picture to one of the pages on the web site. Instructions on how to set up the server are in the next chapter.

Before we get started we want to know how all of this works.

2.13 Network Protocols

There are thousands of web servers throughout the world (wide web) but they are all accessible from any browser because they have all agreed to use a common protocol—the Hypertext Transfer Protocol (HTTP). HTTP is based on an exchange of requests and responses.

Each request can be thought of as a command, or action, which is sent by the browser to the server to be carried out. The server performs the requested service and returns its answer in the form of a response.



The components of a simple WWW interaction are the user, the client, and the server. The client acts as an intermediary between the user and the server.

Steps 1-7 detail the basic information flow in a simple HTTP transaction. Essentially the client requests a file and the server delivers it. The entire HTTP process takes place as a result of simple transactions of requests and responses.

1. The user sees an interesting URL

`http://silo.cs.indiana.edu:13296`

and clicks the hyperlink or types the URL into the browser.

2. The browser interprets this command: It is different from printing, creating the bookmark, saving a file, changing any preferences, etc. This command says that the computer named `silo.cs.indiana.edu` needs to be contacted on port `13296`. For this the browser sends the HTTP GET

⁹<http://validator.w3.org/>

command to the server (not shown here—we will look at how this works when we simulate this request process using the old `telnet` program).

3. So the browser sends the `GET` request to the server, indicating what file it needs¹⁰. This request then travels over the Internet, going from computer to computer until it reaches the web server's host, which is the computer named `silo.cs.indiana.edu`, located on the IU CSCI network¹¹.
4. The server receives and parses the request. It uses the file extension¹² to determine the type of information in the file. This particular extension means that it will send back to the browser the file but it will first say (to the browser) this file's content type is `text/html`, so

`Content-type: text/html`

will be the first line the server will send. You do not have to write this in the file, it is inferred by the server from the file's extension. But the server does send this information to the browser as part of the header, followed by the data (the actual file) as explained below.

5. Thus an HTTP response goes from the server to the client. The headers that are part of the message indicate that the request was OK and that the data returned is of the type we discussed. The headers are then followed by (a blank line and then by) the HTML data itself, line by line.
6. The `Content-type: text/html` part of the header tells the browser that the data is text formatted in HTML, so the browser renders the text appropriately, highlighting hyperlinks, etc.
7. User views the HTML output and has the opportunity to select another hyperlink, starting the cycle over again .

That's the basic mechanism; the web server seems to be central to this mechanism. How do we install a web server? There are a number of things we need to do before installation:

1. Create a user account on `silo.cs.indiana.edu`
2. Familiarize ourselves with Linux/Unix commands.
3. Check the documentation steps regarding installation of Apache.

The lectures and the lab this week will clarify all these issues. Next chapter in particular takes us through the installation of your very own web server. You are about to become a webmaster.

¹⁰In this case and for this server the request translates to `/index.html`

¹¹There's a network security aspect here that we will need to address later.

¹²The file's name is `index.html` so the extension is `.html`

Chapter 3

Installing Apache

So far we have reviewed the basics of HTML and using a text editor and a web browser you have been able to create a few pages, perhaps interconnected with each other. It is time for you to learn the steps by which you can set up your own web server¹. As you will see, the steps are simple, but each step requires attention and careful typing. Don't rush through this installation: it's not long anyway. Take your time and review your accomplishments after each step.

3.1 Building Apache

The Apache HTTPD is a GNU licensed web server, developed by the Apache software foundation. It is highly configurable, extendable, multi-platform and easy to install. Apache's web server is the most popular web server to date. It is also extremely reliable and secure. This chapter describes the installation of the Apache web server. There are about 12-14 steps in all, that should not take you more than twenty minutes to complete.

Here's what you need to do:

1. Log into your `silو.cs.indiana.edu` account
2. Move into your `/nobackup` folder².
3. Go to `http://www.apache.org`, select HTTP server on the left, select the current release's download link and from the different versions available³, select the Unix source file in TAR (tarball) format and download it on your desktop (the file⁴ ending in `.tar.gz`). To simplify this process a bit copies

¹That way, everybody in the world can get access to your HTML pages.

²If you don't have one you must create it first, by running the `makenobackup` utility, at the command prompt. Read the information that appears on your screen and type `yes` to confirm, then change to the `/nobackup` folder created for you by typing: `cd /nobackup/[username]` where `[username]` is your IU (and `silو.cs.indiana.edu`) username.

³You should be on `http://httpd.apache.org/download.cgi` at this stage

⁴As of July 2006 the current version is `httpd-2.2.2.tar.gz`

of the necessary files (for this installation as well as for the later ones in the course) will be made available locally⁵ as well. Check the course web site (the **What's New?** page in particular) for latest information and potential updates. So at this point you should have a file like `httpd-2.2.2.tar.gz` on your desktop. You need to move it to your `/nobackup` folder now to get started with the installation steps.

4. Click the yellow folder on the SSH program window to open an SFTP window to the `sil0` server. In the address bar of the remote (right) side of the screen, type `/nobackup/[username]` and press enter, where `[username]` is in fact your actual username (for me: `dgerman`).
5. Drag the Apache HTTPD file you downloaded to the remote side, uploading the file. If you are taking this file from the shared software folder mentioned use the `cp` command to copy the file from that folder to yours:

```
cp /1/www/classes/a348/sum2006/software/httpd* /nobackup/[username]
```

Once the upload is done, unzip the file⁶. Here's how unzipping worked for me (my prompt is `bash-3.00$`):

```
bash-3.00$ pwd
/nobackup/dgerman
bash-3.00$ ls -ld htt*
-rw-r--r-- 1 dgerman faculty 6282043 Jul  7 23:03 httpd-2.2.2.tar.gz
bash-3.00$ gunzip http*.gz
bash-3.00$ ls -ld htt*
-rw-r--r-- 1 dgerman faculty 30371840 Jul  7 23:03 httpd-2.2.2.tar
bash-3.00$
```

Your specific circumstances might be slightly different.

6. Next, unarchive the file using `tar xvf [filename]`, where `[filename]` now ends in `.tar` not in `.gz`, in this case: `tar xvf httpd-2.2.2.tar`.

```
bash-3.00$ tar xvf httpd*.tar
[...]
httpd-2.2.2/include/ap_release.h
httpd-2.2.2/include/.indent.pro
httpd-2.2.2/include/util_cfgtree.h
bash-3.00$ which tar
/usr/local/gnu/bin/tar
bash-3.00$ pwd
/nobackup/dgerman
bash-3.00$ ls -ld http*
drwxr-xr-x 11 dgerman faculty    1024 Apr 21 23:54 httpd-2.2.2
-rw-r--r--  1 dgerman faculty 30371840 Jul  7 23:03 httpd-2.2.2.tar
```

⁵The folder is `/1/www/classes/a348/sum2006/software` available on-lin as well.

⁶By typing `gunzip [filename]`, where `[filename]` is the downloaded file.

```

bash-3.00$ rm httpd*.tar
bash-3.00$ ls -ld http*
drwxr-xr-x 11 dgerman faculty 1024 Apr 21 23:54 httpd-2.2.2
bash-3.00$ cd http*
bash-3.00$ pwd
/nobackup/dgerman/httpd-2.2.2
bash-3.00$

```

7. When you run `tar xvf` the output on the screen shows every single file that is being extracted from the archive, and as it is being created. Note that before each of the files is the name of the new folder that the archive program created. Move into it (with `cd`) then run the following commands⁷ (each one takes a bit to run) to install Apache:

```

./configure --prefix=/u/[username]/apache --enable-so
make
make install

```

8. Periodically check your quota (using `quota -v`) just to be sure everything is under control. Now we need to configure `httpd.conf` in the new server's `conf` directory⁸. This file was generated automatically by the installation process; copy it into `httpd.conf-backup` before making any changes. We need to edit this file (the `httpd.conf`) for this installation. It contains various configuration options for Apache.
9. Type `nano httpd.conf` to start the `nano` text editor. Find the line⁹ marked `Listen 80` and change the `80` to your assigned port number. For example, if my assigned port number is `13296`, the line should read: `Listen 13296` (individual port numbers will be distributed through the course web site, so you should use here the port you receive).
10. Find the line¹⁰ marked `ServerAdmin`, and change the default e-mail¹¹ to your e-mail address¹². Similarly, find the line¹³ marked `User daemon` and change `daemon` to your username (in my case: `dgerman`).
11. Press `Ctrl-x` to end the `nano` session and save the settings¹⁴. Some other settings we need to change are in the `httpd-mpm.conf` file in the `extra` directory. Type `cd extra` to go to the directory (from `~/apache/conf`) and then `nano httpd-mpm.conf` to start the text editor¹⁵.
- Once inside the file¹⁶ find the line marked `LockFile` and change it to:

⁷The usual convention applies to `[username]` which is to be replaced by *your* username

⁸`cd ~/apache/conf` is the command you need in this case.

⁹It's line 40 out of 407 for me in this version.

¹⁰It is line 85 out of 407 for me in this version.

¹¹The original line reads `ServerAdmin you@example.com`

¹²In my case that is: `dgerman@indiana.edu`, so yours should look similar.

¹³This is line 64 out of 407 for me in this version.

¹⁴You can now try `diff httpd.conf httpd.conf-backup` to verify your changes.

¹⁵If you are somewhere else type: `cd ~/apache/conf/extra`

¹⁶Feel free to make a backup first if you think things might go astray

```
LockFile /tmp/apache-2.2.lockfile.[username]
```

where instead of [username] you should put your own username. Save your changes and exit nano by typing Ctrl-x. You can now start the server.

3.2 Running Apache

Once Apache is installed you need to start and stop it, access it over the web and set your account up for automatic restart of your server.

1. To start the server¹⁷, move into the apache binaries directory¹⁸ either by typing `cd ../../bin` in the directory of your last change (`conf/extra`) or simply by typing `cd ~/apache/bin`.

Then run `./apachectl start`, and check to see your program is running:

```
ps -ef | grep [username]
```

Use your username where [username] appears in the instructions. You should see several processes marked `httpd` as illustrated below:

```
bash-3.00$ ~/apache/bin/apachectl start
bash-3.00$ ps -ef | grep dgerman
root      18351  2791  0 21:23 ?        00:00:00 sshd: dgerman [priv]
dgerman   18354 18351  0 21:23 ?        00:00:01 sshd: dgerman@pts/2
dgerman   18355 18354  0 21:23 pts/2    00:00:00 -csh
dgerman   18387 18355  0 21:23 pts/2    00:00:00 bash
dgerman   6257    1    0 23:47 ?        00:00:00 /u/dgerman/apache/bin/httpd -k start
dgerman   6258   6257  0 23:47 ?        00:00:00 /u/dgerman/apache/bin/httpd -k start
dgerman   6259   6257  0 23:47 ?        00:00:00 /u/dgerman/apache/bin/httpd -k start
dgerman   6260   6257  0 23:47 ?        00:00:00 /u/dgerman/apache/bin/httpd -k start
dgerman   6261   6257  0 23:47 ?        00:00:00 /u/dgerman/apache/bin/httpd -k start
dgerman   6262   6257  0 23:47 ?        00:00:00 /u/dgerman/apache/bin/httpd -k start
dgerman   6263 18387  0 23:47 pts/2    00:00:00 ps -ef
dgerman   6264 18387  0 23:47 pts/2    00:00:00 grep dgerman
bash-3.00$
```

Of course, this is just an example, your screen will have your processes if you type your username. Commands like `ps`, `grep` and the Unix pipe `|` will be explained in the next section. Suffice it to say here that `ps` shows process status, the switches `-ef` ask for all processes on the machine in full format, and the Unix pipe passes the resulting output (a list of all processes currently running, in full format) to the `grep` utility. `grep` is given the task to find all the lines in its input with a certain pattern (in this case we search for our username) and the result will be a list of processes that are currently running and belong to you. While the server is running, you can access it by pointing your browser to:

```
http://silo.cs.indiana.edu:[your port number]
```

¹⁷You could just issue: `~/apache/bin/apachectl start` or follow the steps described above.

¹⁸It is located in `~/apache/bin`

For me that would be: `http://silo.cs.indiana.edu:13296` and with the version of Apache that I am running at the time I am writing these notes the page that comes back to my browser looks like this:



It works!

- To stop the server, use: `~/apache/bin/apachectl stop`¹⁹. Similarly, you can start/restart the server with: `~/apache/bin/apachectl graceful`.
- Set your `crontab` for your server to be re-started (gracefully) every morning at 8:10am.

```
bash-3.00$ crontab -l
10 8 * * * /u/dgerman/apache/bin/apachectl graceful
bash-3.00$
```

You might need to set `nano` to be your default editor first.

- Don't forget the files in `~/apache/logs`: one of them keeps track of all the accesses, the other one records everything that didn't work well plus all the major events in the server's life (like the server being started, stopped, or restarted) and the third one lists the server's process id while the server is running. With respect to `crontab` you should be checking `error_log` to verify that the server's restart schedule works as expected²⁰.
- Type `man crontab` to learn more or search the IU Knowledge Base²¹.
- Your first homework assignment is to create a simple (it really is very simple) web site. The desired outcome is hinted at on the next page. Here's how you would get started:

```
bash-3.00$ clear
bash-3.00$ pwd
/u/dgerman/apache/htdocs
bash-3.00$ cat index.html
<html><body><h1>It works!</h1></body></html>bash-3.00$
```

¹⁹Note that this command is absolute, and can be issued from any folder.

²⁰Use `tail -10` or `tail -f` to see the last few lines of the file.

²¹<http://kb.iu.edu> and search for `cron` or `crontab`.

3.3 Minute Papers and Exercises

In class you will be asked to reproduce the steps for the installation of Apache, and to draw a picture of the Apache filesystem (the files and folders created by installation) and briefly characterize the important folders and files. Setting up the server for automatic restart should be done in the first lab and you should be familiar with all of these by the end of this week. You should have a good understanding of the basic HTML presented. Check the IU Knowledge Base for common Unix commands as you see them being used here, in lecture and lab.

3.4 Homework Assignment One

Create a web site with a few files accessible from your `index.html`. Make sure at least one of these files contains a picture. Describe the picture in that file. In another file create a table, then post the raw HTML code for the table and make sure both the table and the raw code show. (You will need to escape the special characters, e.g., use `<` instead of `<` and so on). As an example take a look at the sample page shown below, which shows both the image and the actual HTML of the image tag:



Post your picture on your server, then write the HTML for the image tag:

```

```

You now have a server that is accessible world-wide, and you can post HTML documents to it that everybody can check with a simple web browser: you should be happy with what you have accomplished so far—and it was a fair amount of work. In the next sections we will first take a look at the most popular Unix commands (from our point of view) then we will start explore Perl. Our goal will be to understand how Apache can be programmed (at first, using Perl) and what relative benefits this ability would present.

Chapter 4

Unix Commands

The purpose of this lab is to give you enough practice to install Apache on your own. You should be able to go through this exercise either before or after the Apache installation, as you prefer. We will create an archive and compress it (just like the Apache archive that we downloaded from the Apache website), then make the archive available for distribution locally on `sil0`. This will give you your first lab grade.

4.1 `pwd`, `ls`, `man`, `mkdir`, `cd`, `cat` (plus `~`, `.` and `..`)

There are two parts to this lab exercise, the first one of which is this section.

We start by logging into `sil0`, and creating a folder `lab1`. Inside `lab1` we will create another folder, `experiments`, which will contain two more folders: `documents` and `programs`. Use `nano` to create `doc1.txt` and `doc2.txt` in `documents`; the first file should describe you (name, username, major, reasons for taking this class and expectations), the second file should describe some of your hobbies(write at length about anything you like or dislike).

Create two folders in `programs`: `perl` and `Java`. Create and compile a simple Java program in `Java`. Write and test a simple perl script in the `perl` directory. I include below the commands that allowed me to accomplish all these tasks (the order in which these commands were issued matters for the most part, so you should study them in that context):

```
5 pwd
6 ls -l
7 mkdir lab1
8 cd lab1
9 ls -l
10 mkdir experiments
11 cd experiments
12 mkdir documents
13 mkdir programs
```

```

14 cd programs
15 mkdir perl
16 mkdir Java
17 cd Java
18 nano One.java
19 javac One.java
20 java One
21 pwd
22 cd ../perl
23 nano one
24 chmod 700 one
25 ./one
26 cd ..
28 pwd
29 cd ../documents
30 ls -l
31 nano doc1.txt
32 nano doc2.txt
38 cd
39 ls -l

```

Your job is to follow these steps and see if you can explain what they do and how they could help you achieve the same goal. Feel free to annotate them as you work on the assignment. The contents of `doc1.txt` and `doc2.txt` are entirely up to you. Here's the contents of the `One.java` file:

```

class One {
    public static void main(String[] args) {
        System.out.println("Oh, I am just typing something.");
    }
}

```

And here's `one`, our first perl program:

```

#!/usr/bin/perl

print "Three cheers for perl!"

```

Let's summarize the commands used above: the Unix file system has a hierarchical tree-like structure. When we first log in we are placed in our account's home directory, also denoted by `~`. The command `pwd` prints the working (current) directory's path. Paths can be relative (to the current folder) or absolute (relative to `/`, the root of the file system). Two important relative paths are `.` and `..` they indicate the current folder and the parent folder. Using the command `cd` we can move around the file system. New folders can be created with the command `mkdir`, new flat files (non-folder files) can be created with a text editor, such as `nano`. To see the contents of a folder we use the command `ls`. Switches are sometimes used on the command line, for example `ls -l` means "list files in long format". To see the contents of a regular file you can use the `cat` command which really means "concatenate" since it can be used to show the contents of more than one file at the same time (by listing their concatenated contents, one after the other). Another way we can look at the contents

of text files is with the commands `more` and `less`. File permissions can be set with the `chmod` command, which is explained in the next section.

The `man` command retrieves the manual page for a specific command.

4.2 tar, gzip, du, chmod, ps, grep

In the second part of this exercise we create an archive `whoa.tar` to contain all files in the `experiments` folder. Next, the archive is compressed and placed in a new folder (`~/public`) whose name indicates its purpose. We make the archive readable by everybody (`chmod 644`), make the `~/public` folder accessible to the world (`chmod 755`) and assign the correct permissions to the home directory (`chmod 711`) so the two files mentioned above can be accessed. Now the archive can be picked up by anybody, and in particular it will be accessed by the instructors, for grading.

Here are the commands, in order, that took me to this stage:

```
56 cd
57 clear
58 pwd
59 ls -l
60 du -a lab1
61 cd lab1
62 tar cvf whoa.tar experiments
63 ls -l
64 gzip whoa.tar
65 ls -l
66 pwd
67 cd ..
68 mkdir public
69 ls -ld public
70 cd public
71 cp ~/lab1/whoa.tar.gz .
72 pwd
73 ls -l
75 chmod 644 whoa.tar.gz
76 chmod 755 .
77 chmod 711 ~
```

Archiving is done with `tar` which means “tape archive”. The `cvf` switches are used when we create an archive, for extraction we use the switches `xvf`. Permissions are being set with `chmod`. The argument of this command is usually a three digit number where each digit is in base 8: the base 2 representation of each digit clearly indicates the file permissions. When you look at the long representation of a file, for example:

```
bash-3.00$ ls -ld alpha
-rw-r--r-- 1 dgerman faculty 46 Jul  2 22:16 alpha
bash-3.00$
```

the sequence of ten characters on the right indicates in order whether the file is a folder (`d`) or not (`-`), and then in groups of three we get the permissions for the file owner (`rw-` which in base 2 reads 110 and in base 8 it's 6) , group (`r--`, that is 100 in base

2, and 4 in base 8) and world (r--). Thus the octal representation for this example is 644 and this file is readable and writable by the owner, and readable by the group and the owner. Nobody can execute this file. Assuming this file is a perl script we can grant execution rights to everybody by issuing the following command:

```
bash-3.00$ chmod 755 alpha
bash-3.00$ ls -ld alpha
-rwxr-xr-x 1 dgerman faculty 46 Jul  2 22:16 alpha
bash-3.00$
```

The x indicates permission to execute (run) the file. Obviously execution makes sense only for executable files, for example perl scripts. Java files don't need to be executable. Java programs (extension .java) are being compiled with .javac and are "executed" (they are in fact interpreted) with java as shown earlier. For folders x means the ability to access files inside the folder. The r bit is different from x in that it allows the browsing of the contents without having the ability to cd into the folder or otherwise access files inside it. To see what processes you're running you can use the ps command ("process status", one can type man ps for a detailed description of the ps command):

```
bash-3.00$ ps -ef | grep dgerman
dgerman  30632      1  0 05:10 ?        00:00:01 /u/dgerman/apache/bin/httpd -k graceful
root     32338      2791  0 10:18 ?        00:00:00 sshd: dgerman [priv]
dgerman  32341  32338  0 10:19 ?        00:00:00 sshd: dgerman@pts/2
[...]
dgerman  6238  30632  0 22:39 ?        00:00:00 /u/dgerman/apache/bin/httpd -k graceful
dgerman  6239  30632  0 22:39 ?        00:00:00 /u/dgerman/apache/bin/httpd -k graceful
dgerman  6243  4025  0 22:39 pts/2    00:00:00 ps -ef
dgerman  6244  4025  0 22:39 pts/2    00:00:00 grep dgerman
bash-3.00$
```

We notice the use of grep which simply filters its input; in this case it is coming from ps -ef through a Unix pipe (the vertical bar between them |).

4.3 Startup and Configuration Files: ~/.bashrc

Every shell has a specific set of configuration files. We will be using bash. Here is a somewhat typical ~/.bashrc file that contains up to date settings for this course.

```
EDITOR=nano
export EDITOR

MYSQL_HOME=/nobackup/dgerman/mysql
export MYSQL_HOME

PATH=/usr/local/gnu/bin:/bin:$PATH
export PATH

JAVA_HOME=/1/jdk1.5
export JAVA_HOME
CATALINA_HOME=/u/dgerman/tomcat/apache-tomcat-5.5.17
export CATALINA_HOME
CLASSPATH=.:$CATALINA_HOME/common/lib/servlet-api.jar:~/programs/jdbc/mm.mysql-2.0.2-bin.jar
export CLASSPATH
```

You already have such a file; the PATH and EDITOR environment variables are needed in the beginning, they are likely already set correctly for you when the account is created.

Chapter 5

Introduction to Perl

5.1 A Review of Basic Concepts

Here now is a short, minimal introduction to Perl to get us started.

Perl is just a programming language. A variable in Perl is written like this:

```
$x
```

The name is `x` and the dollar sign indicates it's a scalar (has no dimension). A variable is just a location that is accessible by name. Not all data structures are that simple. You can have lists, sequences of locations, indexed by their position in the sequence. If the name of the list is `x` then I can refer to the *entire* list as follows:

```
@x
```

The list could be empty or could have one or more elements in it. Let's say `$i` is a variable that stores an integer, then

```
$x[$i]
```

means the element with index `$i` in the list. Remember that the first element in the list has index 0 (zero) while the last element in the list `@x` can be accessed as `#x[$#x]`. We discuss assignment statements. The symbol for assignment is

```
=
```

and it splits the assignment into two parts:

1. On the right hand side of the assignment we have expressions and *values*.
2. On the left hand side we have *locations*.

So in the following assignment

```
$i = $i + 1;
```

the variable `$i` is used for its *value* on the right, and for its *location* (or address) on the left. The result, of course, is that `$i` is incremented by one. It works just the same with elements of lists, since they're also variables. Their names are a bit more complex, since they are constructed from the name of the list and the index in the list and we need to use the brackets, but other than that they're names just as the names we're used to (the *identifiers*). So, this assignment:

```
$x[$#x] = $x[$#x] + 1;
```

will increment the value of the last element in the list `@x` by one. Hash tables (or hashes, or *association lists*) are just like lists, but indices are not numbers—instead strings of characters are being used to index the values stored. The indices must be unique and they are called *keys*. To refer to a hash table as a whole we use

```
%x
```

and to get the individual elements we index using a `$key`.

If `$key` contains a string, and if `%x` is a hashtable then if there is anything associated with the value of `$key` in `%x` it can be retrieved or indicated with

```
$x{$key}
```

while, if there is no association we will either obtain an **undefined** value for it or obtain the ability to store one for this key, depending on where this expression appears with respect to the `=` (assignment operator). Here's an example. Assume `%x` is empty to start with. Then

```
$x{"jordan"} = "bulls";
```

builds a first association.

```
$x{"miller"} = "pacers";
```

builds another, while

```
$x{"jordan"} = "wizards";
```

will change the value previously associated with `jordan`. In general you can obtain the list of all keys in a hash table this way:

```
@theKeys = keys %x;
```

where `%x` is the hashtable. Then you can use a `foreach` to go over all of them, for whatever processing purposes you may have in mind:

```
foreach $e (@theKeys) {
    $x{$e} = $x{$e} . " (nba)";
}
```

The code above will add `(nba)` to each one of the values stored in the hashtable (since the `.` (dot) operator is used for concatenation in Perl). So if you print `$x{"miller"}` now it would read `pacers (nba)`. That's the first part of the Perl review we need.

5.2 What is CGI?

We need two more things if we are to start making experiments: an understanding of how we run perl programs and the ability to print from inside our programs. Both these aspects have already been covered during the Unix lab; we created two programs then (one of them was in perl) and compiled and run them both. The perl program wasn't doing much but it was printing, a string.

To understand how Perl can be used to program the behavior of Apache we start by creating a file `hello.html` with the following contents, in the document root of our web server (the folder `/u/apache/htdocs`):

```

<html>
  <head><title>the hello page</title></head>
  <body bgcolor=white>
    <h1>Hello!</h1>
    
  </body>
</html>

```

We already discussed the server's reaction to an HTTP request for this file¹. One can easily write a program that implements that behavior:

```

#!/usr/bin/perl

print qq{Content-type: text/html\n\n<html>
  <head><title>the hello page</title></head>
  <body bgcolor=white>
    <h1>Hello!</h1>
    
  </body>
</html>
};

```

Note the use of `qq{` and `}` as generalized string delimiters. If we place this program in a file (call it `one`) in the `cgi-bin` folder (located in `~/apache`) and `chmod` it to 700 (so we can run it) accessing

```
http://silo.cs.indiana.edu:13296/cgi-bin/one
```

will have the exact same effect as

```
http://silo.cs.indiana.edu:13296/hello.html
```

to a web the browser. So a web script is at least as functional as a static web page.

5.3 Simple CGI Scripts

Web scripts are much more powerful than static web scripts, as our next example will show. To understand it we need two more pieces of information about perl: the `rand` function called with a positive argument returns a random real between 0 and its argument (without ever reaching the value of the argument). The `int` function simply performs casting from a real (floating-point) number to an integer, through truncation. With this (and what we learned in the first section of this chapter) the program below becomes very easy to understand: a list of picture names is created, it is called `images`; a random (valid) index in this list is generated and is stored in a variable called `index`; the string that corresponds to that index is calculated and stored in the variable `imgname`, which is later used in the HTML code printed by the program. The occurrence of any variable in a string of characters that gets printed is always replaced by its value, which is precisely what happens with the image name variable in the code (this is called interpolation). The net result is that each time the

¹As results by typing `http://silo.cs.indiana.edu:13296/hello.html` in a browser.

code is (re)loaded a new picture (more or less) will be shown, a feature that is simply not available with static HTML.

Here is the program. Call it `two` and place it in `cgi-bin`. Don't forget to make it executable, then access it from `http://silo.cs.indiana.edu:13269/cgi-bin/two` making sure to reload the page a few times. The program works as expected, and only has one minor drawback.

```
#!/usr/bin/perl

@images = ("lh08.gif", "lh07.gif", "lh09.gif", "lh01.gif");

$index = int(rand($#images + 1));

$imgname = $images[$index];

print qq{Content-type: text/html\n\n<html>
  <head><title>the hello four script</title></head>
  <body bgcolor=white>
    <h1>Hello!</h1>
    
  </body>
</html>
};
```

The drawback mentioned is simply a user interface issue: can we eliminate the need for the user to reload the page in the browser, and instead embed this functionality in the page itself? The answer is: yes, and the solution is to replace the `` tag by:

```
<p> The image below has index $index. <p> Click <a href=
"http://silo.cs.indiana.edu:13296/cgi-bin/two">here</a>
for a new random image. <p>


```

As we shall see improvements are still possible but for now we shall declare ourselves satisfied with this.

5.4 User Input with CGI

Web scripts are more powerful than static HTML since they can produce dynamic content. An example is our previous script; it randomly chooses a picture for display every time it runs. Is there any way in which we could choose what picture the program shows? In other words, is it possible to implement a program that offers a menu of four pictures, and allows us to select a picture for viewing much in the way the following picture illustrates?

Let's start by explaining the functionality of the program and the picture presented: there are four pictures and the program starts by presenting four links, one for each of the available pictures. Not shown is the default picture that accompanies the four links: the default picture is different from the four pictures for which the program is a menu. When one of the four links is pressed the program replies with an HTML page

in which the requested picture is shown, the links that don't refer to this picture are active, and the link that corresponds to the picture shown is turned inactive (plain text, with slightly bigger fonts). The illustration below shows the program's output when the picture requested was `lh09.gif`.



SouthWest of Lindley (<http://www.cs.indiana.edu/dept/img/lh09.gif>)

Careful investigation of the URL might give a clue of the approach taken.

5.5 %ENV, the Environment Hashtable

Suppose, with your already existing knowledge of Perl and CGI, that someone tells you that in each of your web scripts a hashtable by the name of `%ENV` magically will be provided. What would you do then? You would obviously print it out, to see what it contains, wouldn't you? That's exactly what the `printenv` script in your `cgi-bin` folder is doing already; the script came with your server and you should find this script in the `cgi-bin` folder after installation of your Apache server:

```
#!/usr/local/bin/perl
##
## printenv -- demo CGI program which just prints its environment
##

print "Content-type: text/plain; charset=iso-8859-1\n\n";
foreach $var (sort(keys(%ENV))) {
    $val = $ENV{$var};
    $val =~ s|\n|\\n|g;
    $val =~ s|"|\\"|g;
    print "${var}=\"${val}\"\\n";
}
```

What can we say by looking at the script? First, that iterating through the keys of a hashtable is done with `foreach`, which we knew already. That accessing the value for the entry that corresponds to the key `$key` is done with the expression `$ENV{$key}`. That there is some syntax that we haven't seen yet, too, yet for the better part the script is clear with what we said we knew above, already.

So here's a potential experiment after you look at the various entries that the `printenv` is listing: go on the URL and type what you saw in the previous picture's URL (the menu of four pictures program shown earlier): add a question mark and a few characters (no spaces) at the end of the URL then hit Enter. What do you notice?

I am sure you already have the answer, so I am just going to verify it for you now: one of the entries (`QUERY_STRING` suddenly carries the string that you have entered after the question mark.

But that's a wonderful discovery, then: it means we can use the URL to send any string we want to the program. Our problem is solved! The string we send can carefully be embedded in any link we write on the page (exactly like in a menu) and can easily be checked against known values. Here's a quick experiment to verify that the strategy works well:

```
#!/usr/bin/perl

print "Content-type: text/html\n\n<html><body><pre>";

$string = $ENV{'QUERY_STRING'};

foreach $key (keys %ENV) {

    if ($key eq $string) {
        print $key, " --> ";
        print $ENV{$key}, "\n";
    } else {
        print qq{<a href="/cgi-bin/circular?$key">$key</a>}, "\n";
    }

}

print "</pre></body></html>";
```

The name of this program is: `circular` and it prints all the entries in `%ENV` as links. Note the embedded functionality we were talking about earlier: each time a link is clicked the part following the question mark will be transmitted in `$ENV{QUERY_STRING}` and the program will store it in a variable called `string`. Consequently, for all the entries in the hashtable we test the key against the submitted value; only one such entry will pass the test and it will be printed as plain text, along with its value (the string itself). The others will stay as active links, awaiting user selection.

The one last thing we would like to point out is the wiring of the program's name into the program's code; looking carefully to the entries in the environment hashtable one notices that the name of the program is the value of the entry `SCRIPT_NAME`. We could use this to make the code independent of the program's name and this is the approach we take in the implementation of the Lindley portfolio²

²The four pictures show Lindley Hall, as one may have noticed already.

The code that follows then stores the URLs for the four pictures in a hashtable called `images`; it uses two subroutines called `printTop` and `printBottom` whose purpose is to print the top and the bottom parts of the HTML document that it prints. The approach is sound because these parts will always be the same regardless of the purpose and functionality of the program itself. Most of the program works as the `circular` script presented earlier:

```
#!/usr/bin/perl

%images = (
  "One"    => "http://www.cs.indiana.edu/dept/img/lh01.gif",
  "Seven"  => "http://www.cs.indiana.edu/dept/img/lh07.gif",
  "Eight"  => "http://www.cs.indiana.edu/dept/img/lh08.gif",
  "Nine"   => "http://www.cs.indiana.edu/dept/img/lh09.gif"
);

&printTop;

@lst = (keys \%images);

$input = $ENV{"QUERY_STRING"};

print "<table border cellpadding=6 width=100%><tr>";

$script = $ENV{"SCRIPT_NAME"};

foreach $i (@lst) {
  $name = $i;
  if ($input eq $name) {
    print qq{<td> $name </td> };
  } else {
    print qq{ <td> <a href="$script?$name">$name</a> </td> };
  }
}

print "</tr>";

if ($images{$input}) {
  $pic = $images{$input};
} else {
  $pic = "http://www.cs.indiana.edu/classes/a202-dger/sum99/a202.gif";
}

print qq{<tr>
<td align=center colspan=4> <p> <p> </td>
</tr> };

print "</table>";

&printBottom;

sub printTop {
  print qq{Content-type: text/html\n\n<html>
  <head><title>My Pictures Script</title></head>
  <body bgcolor=white>
  };
}

sub printBottom {
  print qq{</body></html>};
}
```

Worth mentioning in the program above are also the following: a table is used in the output, and the top line is printed in almost the same way as the list of entries in `%ENV` in `circular`; it does not matter what name we give to this program, the variable

`script` will be properly initialized with that name from the `SCRIPT_NAME` entry; there is a default picture, which is shown if the request does not match any of the existing four pictures offered; despite the program's structure as a menu it is still possible to formulate ill-formed requests³; and finally, to finish the program we need to print captions and URLs as in the demo shown earlier.

5.6 Keeping State

The developments in the previous section were a tour-de-force, undoubtedly. And while you should feel happy about the result a single question can point out that this is just the beginning of the road. So we will ask: what would it take for our program to show the pictures in succession, using only `Previous` and `Next` links in its interface? The answer is surprisingly simple: the script must remember. But there doesn't seem to be any simple mechanism that a script like ours can rely on, and indeed there isn't; we need to come up with a creative solution. What do we do?

Here's the solution:

```
#!/usr/bin/perl

%images = (
    "One"    => "http://www.cs.indiana.edu/dept/img/lh01.gif",
    "Seven"  => "http://www.cs.indiana.edu/dept/img/lh07.gif",
    "Eight"  => "http://www.cs.indiana.edu/dept/img/lh08.gif",
    "Nine"   => "http://www.cs.indiana.edu/dept/img/lh09.gif"
);

@keys = ("One", "Seven", "Eight", "Nine");

use CGI;
$q = new CGI;

print $q->header, $q->start_html;

$name = $ENV{SCRIPT_NAME};
$input = $ENV{QUERY_STRING};

$pic = $images{$keys[$input]};

$next = ($input + 1) % 4;
$prev = ($input + 3) % 4;

print qq{
<table>
<tr><td> <a href="$name?$prev">Previous</a>
<td align=right> <a href="$name?$next">Next</a>
<tr><td colspan=2 align=center> 
<tr><td colspan=2 align=center> The image above has index $input
</table>
};

print $q->end_html;
```

³This is a serious security issue and the first time we notice a major difference between programming as we knew it and the behavior of the scripts we deploy on the web. There are other differences, too, and we will cover them in due time.

This is a key stage in our development: we realize that state can be kept on the user side. Can we trust the user, though? Nevermind that, let's just assume that we can, for the time being⁴. State is embedded in the two links with which the user can control the program's behavior. Not bad, but we are only halfway there: can we use the same method to implement a simple ATM machine?

Before we answer that last question let's note the use of the `CGI.pm` perl library to accomplish what we were doing through the specialized subroutines `printTop` and `printBottom`. Use of this library must always leave the following two-line invocation behind, somewhere towards the top of the program:

- `use CGI;` which loads the library, and
- `$q = new CGI;` which creates an object with appropriate functionality

The second one requires additional qualifications: the name of the variable that points to the object (in this case `q`) is entirely up to you. It must however be used later to print the HTML header, top and bottom of the document. Each of these three instance methods return a string; to select these methods we use an arrow (as in `->`) between the variable and the instance method's name.

5.7 CGI and HTML Forms

Although our advances have been quite impressive in the previous sections there is no way we could use the same methods to implement a simple calculator of the type shown below. In the picture, if the `Compute` button is pressed the input (18) would be added to the current balance (38) making it 56, and reporting it. The interface would remain the same: a drop-down menu of actions offering addition and subtraction (deposit and withdraw, for a bank account) and a submit button.



Additional use of the `CGI.pm` library is made below: more instance methods are being used. HTML forms also show up for the first time: this one contains text and select fields, both visible to the user, and two hidden fields, which are not immediately noticeable in the page. There is also a submit button, and data collected is listed at

⁴We know, however, that this is not true in general.

the top of the URL⁵, right next to the usual question mark, and immediately following the submission. `CGI.pm` provides methods for reading the input values associated with a particular field: `$q->param('arg')` is the value the user enters for the amount of the transaction.

```
#!/usr/bin/perl

use CGI;
$q = new CGI;

print $q->header, $q->start_html;

$balance = $q->param('balance'); # retrieve the state
$message = $q->param('message');

$args = $q->param('arg'); # read user input
$fun = $q->param('fun');

if ($message) { # if there is state, update it
    if ($fun eq "add") {
        $balance += $q->param('arg');
    } elsif ($fun eq "sub") {
        $balance -= $q->param('arg');
    }
    $message = "The current balance is:" . $balance;
} else { # new state, initialize it
    $balance = 0;
    $message = "The starting balance is:" . $balance;
}

# report and save state, get ready for more input

print qq{
<form>
    $message <p>
    Please choose your transaction type: <select name="fun">
        <option value="non"> Click me!
        <option value="add"> Deposit
        <option value="sub"> Withdraw
    </select> <p>
    Please enter the amount: <input type="text" name="arg">
    <input type="hidden" name="balance" value="$balance"> <p>
    <input type="hidden" name="message" value="$message"> <p>
    When ready please press <input type="submit" value="Proceed">
</form>
};

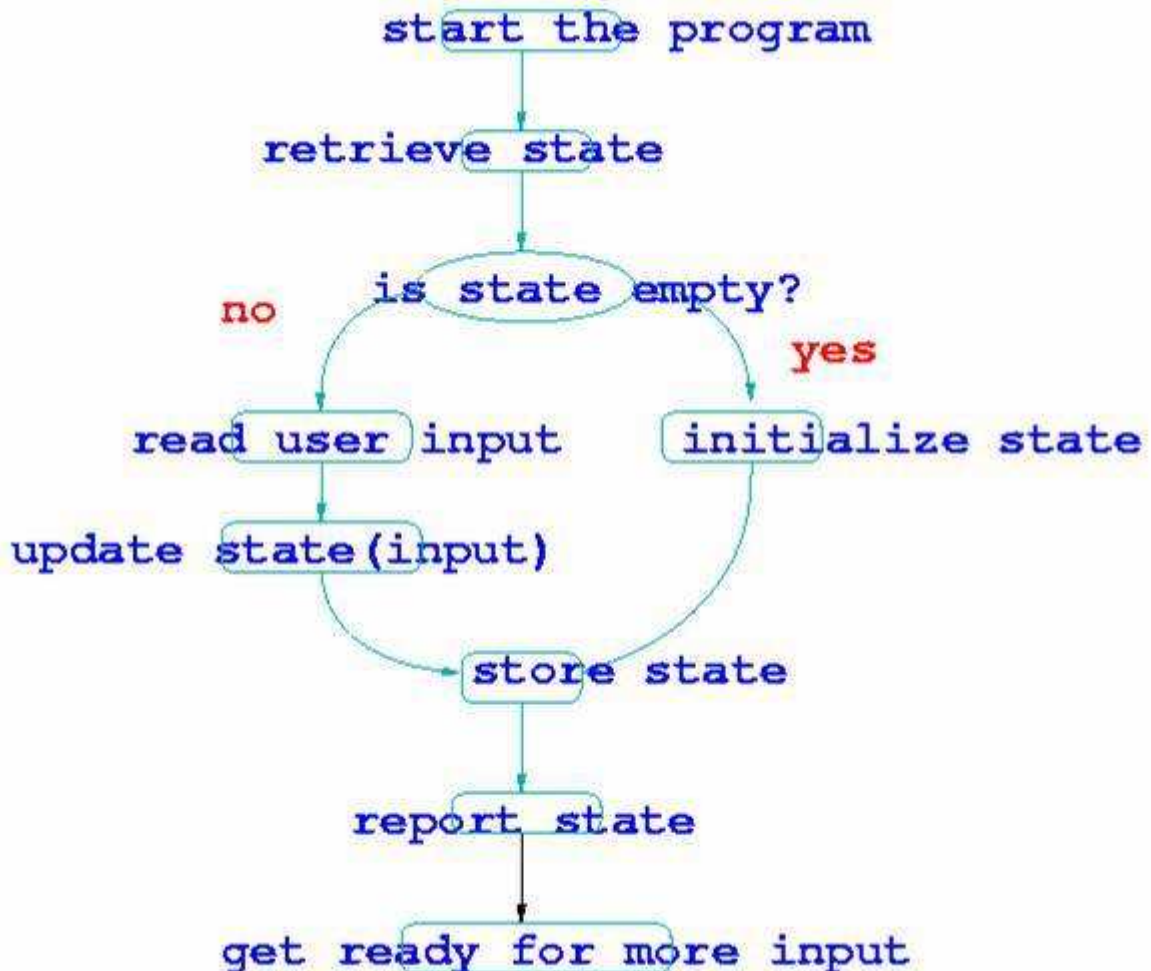
print $q->end_html;
```

More important than the new `CGI.pm` features used⁶ is the pattern that we see developing. Let's describe this pattern in terms that apply specifically to this problem: our state is composed of two variables (`balance` and `message`) one of which remembers

⁵The pattern is easy to spot, a list of strings with the format `name=value`, separated by ampersands (&), where names are of the fields in the HTML form and values are the data provided by the user for the specific input fields before pushing the submit button.

⁶Additional documentation can be found on-line or by typing `perldoc CGI` at the prompt.

what we said last to the user; were this to be empty we would know we're talking to a new user, and that's why the program makes this distinction first. Thus it appears that we first retrieve state, initialize the new state or update the existing one with the user input, then store, report the state and prepare for more input from the user:



It would be quite instructive at this stage to write a program like this:

```
#!/usr/bin/perl

while ($x = <STDIN>) {
    ($com, $arg) = split(/ /, $x);
    print "You have typed: $x";
    if ($com =~ /^bye/i) { print "Good-bye!\n"; exit; }
    elsif($com =~ /^add/i) { $acc += $arg; print "Acc is now $acc\n"; }
    elsif($com =~ /^sub/i) { $acc -= $arg; print "Acc is now $acc\n"; }
    else { print "Acc stays $acc\n"; }
}
```

It would achieve roughly the same purpose from the command prompt. For one thing, we would be able to introduce new features of Perl, such as `STDIN` and the diamond operator `< >`, and the `split` and pattern matching `~` operators. Second, the distinct difference between the data-driven approach of the web script and the busy-waiting existent in the program above (illustrated by the presence of the `while` loop) should also be evident. If we compare the two we realize: the web script only implements the body of the `while` loop, the looping behavior is in fact provided by the user (every time (s)he clicks, or sends a new request).

5.8 Minute Papers and Exercises

This chapter must be taught with care. The course URL contains additional materials and the development of the programs is done in slower motion. In class the portfolio and calculator programs could easily be given as minute paper tests; it's only by trying to develop a program from scratch that one realizes how much one really understands. Other topics for minute papers include: writing simple Perl programs, summarizing useful features of the `CGI.pm` module, etc.

5.9 Homework Assignment Two

Write a web script that acts as a flag quiz: it should present a sequence of eight flags to the user and the user needs to name the flags one by one. The game ends after all flags have been viewed⁷. The number of points for each good answer is up to you.



⁷The interface above belongs to Jun Yin, Spring 2005 A548 student.

Chapter 6

User Authentication

This short chapter shows how you can restrict access to a folder on your server to just those users who should have access to it, and only in exchange for a username and a password. The specific purpose is that of setting a **protected** folder in which assignments can be turned in on-line, which is password protected and to which instructors have access in a uniform way.

6.1 The Password File

The instructor will create a password file as follows:

```
bash-3.00$ pwd
/u/dgerman/public
bash-3.00$ /u/dgerman/apache/bin/htpasswd -c passwd dgerman
New password:
Re-type new password:
Adding password for user dgerman
bash-3.00$ ls -ld passwd
-rw-r--r-- 1 dgerman faculty 22 Jul  8 16:25 passwd
bash-3.00$ cat passwd
dgerman:UIaTPpE.ZpwHw
bash-3.00$
```

This file is made publicly available (note that the password is encrypted):

```
chmod 711 ~
chmod 755 ~/public
chmod 644 ~/public/passwd
```

The students then copy this file in their server root, then create a **protected** folder in their document root (`htdocs`) and switching to the `~/apache/conf` folder open the main configuration file:

```
cd ~/apache
cp /u/dgerman/public/passwd .
```

```

cd htdocs
mkdir protected
cd ../conf
nano httpd.conf

```

6.2 Server Configuration

In `nano` find the section of `httpd.conf` that refers to directory configuration (search for the first occurrence of the `<Directory />` tag, which is around line 125/416 for me. Add the following right after the default directory configuration, above the settings that refer to your document root. Instead of `[username]` the student's username should be written, while the instructor (or grader's) username (in this case `dgerman`) should appear in place of `[instructor]` below.

```

<Directory /u/[username]/apache/htdocs/protected>
    AuthName          Protected
    AuthType          Basic
    AuthUserFile      /u/[username]/apache/passwd
    <Limit GET POST>
        require user [instructor]
        require user lbird
    </Limit>
</Directory>

```

Additional users (such as `lbird` above) can be added, one per line. The password for the new user can be created by running the `passwd` utility in the student's account:

```

bash-3.00$ ~/apache/bin/htpasswd -b ~/apache/passwd lbird dribl
Adding password for user lbird
bash-3.00$ cat ~/apache/passwd
dgerman:UIaTPpE.ZpwHw
lbird:o8.5EkjWeFwG.
bash-3.00$

```

The server needs now to be stopped and restarted. Subsequently, every access to the `protected` folder¹ from now on will require authentication. The first such access will ask for a username and password. If these pieces of information are correctly specified then the browser assumes that identity and accesses that follow will belong all to the username that authenticated. For this reason, in general, it is good to instruct your customers to quit their browsers at the end of their session.

The setup described in this chapter allows the instructor to distribute the access password to all students without revealing it to anybody. Students can still control access to the folder, and can grant access to themselves or selected (other) users, but the instructor is now able to log into each and every protected folder with the same username and password, which greatly simplifies grading.

We now switch our attention to HTTP, the protocol used in the communication between servers and browsers.

¹Where homework assignments need to be placed for grading.

Chapter 7

HTTP

Communication between web servers and browsers follows a simple protocol, the details of which we plan to explore in this chapter. Some information has been presented already at the end of chapter 2; We use our newly acquired knowledge of Perl and CGI to deepen that understanding. The Hypertext Transfer Protocol (HTTP) is the language web clients and servers use to communicate with each other. It is essentially the backbone of the World Wide Web. All HTTP transactions follow the same general format. Each client request and server response has three parts:

1. the client request or server response line,
2. a header section, and
3. the entity body.

The client initiates a transaction as follows:

1. The client contacts the server at a designated port number (by default, 80).
2. It then sends a document request by specifying an HTTP command called a method, followed by a document address, and an HTTP version number. For example: `GET /index.html HTTP/1.0` This uses the `GET` method to request the document `index.html` using version 1.0 of HTTP.
3. Next the client sends optional header information to inform the server of its configuration, and the document formats that it will accept. All header information is given line by line, each with a header name and value. The client sends a blank line to end the header.
4. After sending the request and headers, the client may send additional data. (This data is mostly used by CGI programs that use the `POST` method).

The server responds in the following way to the client's request:

1. The server first sends a status line containing three fields: the HTTP version, a status code, and description. The HTTP version indicates the version of HTTP that the server is using to respond. the status code is a three-digit number that indicates the server's result of the client's request. The description following the status code is simply a human-readable text description of the status code. For example: `HTTP/1.0 200 OK` This status line indicates that the server uses version 1.0 of HTTP in its response. A status code of 200 means that the client's request was successful, and the requested data will be supplied after the headers.

2. After the status line, the server sends header information to the client about itself and the requested document. A blank line ends the header.
3. If the client's request is successful, the requested data is sent. This data may be a copy of a file, or the response from a CGI program. If the client's request could not be fulfilled, the additional data may be a human-readable explanation of why the server could not fulfill the request.

Being a stateless protocol, HTTP does not maintain any information from one transaction to the next, so the next transaction needs to start all over again. The advantage is that an HTTP server can serve a lot more clients in a given period of time, since there is no additional overhead for tracking sessions from one connection to the next. The disadvantage is that more elaborate CGI programs need to use hidden input fields, or external tools such as cookies, to maintain information from one transaction to the next.

7.1 Experiment One

Log onto `silو` and connect using `telnet` to your server. We will try to retrieve its home page. Type the host and the port number on the command line and hit enter; the server is available for your request. Type the line you see below (issuing the simplest `GET` command, and you will see that the server will continue to wait. Press enter one more time and only then the server will send back the `index.html`. Note that the line after the `GET` command cannot even have a blank space on it. If it has it's considered a header. An empty line ends the header section and prompts the server's response.

```
bash-3.00$ telnet silو.cs.indiana.edu 13296
Trying 129.79.247.5...
Connected to silو.cs.indiana.edu.
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 08 Jul 2006 23:39:47 GMT
Server: Apache/2.2.2 (Unix)
Last-Modified: Sat, 08 Jul 2006 04:25:46 GMT
ETag: "84dd47-14c-5ffd0680"
Accept-Ranges: bytes
Content-Length: 332
Connection: close
Content-Type: text/html

<html>
  <head><title>My first page</title></head>
  <body>
    This is my first HTML page.
  </body>
</html>
Connection closed by foreign host.
bash-3.00$
```

7.2 Experiment Two

Place the following program in `~/apache/cgi-bin/eTwo`

```
#!/usr/bin/perl

use CGI;
$q = new CGI;

print $q->header,
      $q->start_html,
      $q->Dump,
      $q->end_html;
```

Then experiment by trying to send it some data like this:

```
bash-3.00$ telnet silo.cs.indiana.edu 13296
Trying 129.79.247.5...
Connected to silo.cs.indiana.edu.
Escape character is '^]'.
GET /cgi-bin/eTwo?fun=add&arg=12&balance=34 HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 08 Jul 2006 23:58:04 GMT
Server: Apache/2.2.2 (Unix)
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head><title>Untitled Document</title></head><body>
<ul><li><strong>fun</strong></li>
  <ul><li>add</li></ul>
  <li><strong>arg</strong></li>
  <ul><li>12</li></ul>
  <li><strong>balance</strong></li>
  <ul><li>34</li></ul>
</ul>
</body></html>Connection closed by foreign host.
bash-3.00$
```

Try doing the same from a web browser; the behavior will be the same, the output definitely more readable. We see that when we issue a GET request the headers end with an empty line and the server does not expect anything else, so it proceeds with the response. Sending data with POST requires an additional header (which indicates the size of the data being transmitted) and the data itself is sent in a separate body section of the message.

7.3 Experiment Three

In the example below we send the exact same information to the script `eTwo`, only using method `POST` this time. You notice that data is no longer posted as part of the URL, following the question mark placed right after the name of the program. Instead, a new header indicates the content length: notice we indicated a 14 character content length then hit enter twice. The second time an empty line was transmitted to the server indicating the end of the headers. However, unlike before, this time the server keeps waiting. So we type our content and hit enter:

```
bash-3.00$ telnet silo.cs.indiana.edu 13296
Trying 129.79.247.5...
Connected to silo.cs.indiana.edu.
Escape character is '^]'.
POST /cgi-bin/eTwo HTTP/1.0
Content-length: 14

arg=12&fun=add&balance=32
HTTP/1.1 200 OK
Date: Sun, 09 Jul 2006 00:32:07 GMT
Server: Apache/2.2.2 (Unix)
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
  <head><title>Untitled Document</title></head>
  <body><ul><li><strong>arg</strong></li>
    <ul><li>12</li></ul>
    <li><strong>fun</strong></li>
    <ul><li>add</li></ul>
  </ul>
</body>
</html>Connection closed by foreign host.
bash-3.00$
```

It is worth pointing out the number of characters we indicated we would send: 14, and that we actually typed 12 more (including the newline); the extra characters have actually been discarded. The 14 characters we declared in the header are however read and processed and that can be seen in the script's output.

7.4 Experiment Four

We have one more experiment to make, but this experiment is in two parts. We would like to investigate the role played by the browser in the transmission of the data.

We start by creating a form in `~/apache/htdocs/eFour.html` and with the following contents¹:

```
<html>
  <head><title>Experiment Four</title></head>
  <body>
    <form method="POST" action="/cgi-bin/eFive">
      <table>
        <tr><td> Fun: <td> <input type="text" name="fun">
        <tr><td> Arg: <td> <input type="text" name="arg">
      </table> <p>
      Press <input type="submit" value="Proceed"> when done.
    </form>
  </body>
</html>
```

Notice the form is programmed to collect its data and send it using method `POST` to the script `/cgi-bin/eFive` whose contents appear like this:

```
#!/usr/bin/perl

if ($ENV{REQUEST_METHOD} eq 'GET') {
  $in = $ENV{QUERY_STRING};
} else {
  read(STDIN, $in, $ENV{CONTENT_LENGTH});
}

print "Content-type: text/html\n\n<html><body>";

print "<h1>$in</h1>";

print "</body></html>";
```

The first part of this experiment calls for the use of the form `eFour.html` as is, by typing some text in the fields and pushing the submit button. It can be noticed that the browser collects the names of and the values in the HTML fields and sends them all to the script in the format mentioned earlier. Thus, if we type `alpha` in the first field and `beta` in the second, the script will receive the following string as data: `fun=alpha&arg=beta` and it will display it back to the screen. This string of characters is submitted in the manner illustrated earlier, and the script receives the content length in a header with a similar name.

This example is particularly instructive if we want to know what happens under the hood of `CGI.pm` and we see that for `POST` transmissions the script receives the data in its standard input. Changing the method attribute of the form to `GET` won't make a visible difference, and the script will process incoming data differently.

The second part of this experiment involves sending what might look as meaningless data through this form and try to explain the result. Experiment with various values sent through the `eFour.html` form, using either `GET` and `POST`; I am sure you will

¹Which can be accessed through the URL: `http://silo.cs.indiana.edu:[port]/eFour.html` (where `[port]` is your port number.

eventually notice it. To indicate the nature of the behavior clearly try sending three equal signs through the `fun` field and `~%&` (tilde, the percent sign and the ampersand sign) through the other one. You will notice that the characters vanish, and the script prints back the following string:

```
fun=%3D%3D%3D&arg=%7E%25%26
```

From this we deduce that the equal sign is converted to a `%3D`, the ampersand to `%26`, and so on. The conclusion is correct, the reason is simple: a small set of characters (called *special* characters) are used in the encoding of data. Among them the equal sign, the percent sign and the ampersand². To avoid any ambiguities, the special characters typed by the user are being encoded. The CGI script is expected to decode them, and with `CGI.pm` this happens automatically. We used to have a special module on Perl pattern matching and developing a `readParse` method, which was able to do basic CGI processing, by hand. While instructive, the approach is now somehow outdated, and the use of `CGI.pm` is prevalent; as we said, there, decoding is done automatically along with parsing when the CGI object is created (`$q = new CGI;`).

7.5 Minute Papers and Exercises

You can be sure you will be asked what is the difference between `GET` and `POST`.

²A slightly different rule is applying to the blank space (character with ASCII code 32), but we need not be concerned with it here.

Chapter 8

Relational Databases

Consider the following problem.

You are interested in organizing a chess tournament. This is a big money opportunity for you so if you do it right you can retire a wealthy person. You're also in for the art of it, so you decide to make it a memorable event by inviting big names, and scheduling the events across the nation, in its finest venues, biggest halls, most exquisite establishments.

Here's the kind of data you will have to store:

Round one:

Jordan	-	Barkley	1—0	(Soldier Field)
Miller	-	Bird	0—1	(Madison Square Garden)
Kukoc	-	Duncan	$\frac{1}{2}$ — $\frac{1}{2}$	(Wrigley Field)

Statistics (and general information) related to round one:

- Soldier Field's capacity is: 2,800 seats. The game between Jordan and Barkley was watched by 100 students from IU Bloomington and 120 from the University of Illinois. The price of the ticket was \$5.50 and the cost of renting Soldier Field per game is \$800.
- Madison Square Garden's capacity is: 4,000 seats. It rents for \$1,200 per game. For the Miller vs. Bird game, the price of the ticket was \$8.00 and in audience were 140 students from the University of Illinois and 60 from Purdue University¹.
- Wrigley Field tickets were \$4.25 for the Kukoc—Duncan game. The lease was \$800.00 and capacity is 1,200. There were 20 students from the University of Minnesota and 220 from the University of Michigan.

Round two:

Duncan	-	Barkley	$\frac{1}{2}$ — $\frac{1}{2}$	(Woodburn Hall)
Kukoc	-	Bird	0—1	(Staples Center)
Miller	-	Jordan	$\frac{1}{2}$ — $\frac{1}{2}$	(RCA Dome)

- Woodburn Hall has a capacity of 2,000 seats and for the Duncan—Barkley game 80 tickets had been sold to the University of Minnesota and 300 to the University of Michigan. Ticket price at this game was \$6.50 and the rent is \$600.00/game.

¹Purdue is a little known university in West Lafayette, IN.

- The Staples Center rents 3,200 seats for \$500.00 per game and the tickets at the Kukoc—Bird game were \$5.50 each. University of Iowa was the only university to buy tickets to this game, 180 of them.
- The RCA Dome has 1,800 seats and rents for \$230.00 per night. Tickets at the Miller—Jordan game were \$5.75 each. Purdue bought 120, and IU bought 210.

Round three:

Bird	-	Jordan	1—0	(Rawles Arena)
Kukoc	-	Barkley	1—0	(Soldier Field)
Duncan	-	Miller	$\frac{1}{2}$ — $\frac{1}{2}$	(Madison Square Garden)

- Rawles Arena rents for \$740.00 (capacity of 3,400). 150 tickets were sold to IU Bloomington and 200 to Iowa. Tickets were \$7.35 at the Bird—Jordan game.
- 180 tickets (\$8.00 apiece) sold to U. of Illinois for the Kukoc—Barkley game.
- Tickets at the Duncan—Miller game were \$9.99 each and were sold as follows: 120 tickets to the University of Minnesota, and 40 to the University of Illinois.

Round four:

Miller	-	Barkley	$\frac{1}{2}$ — $\frac{1}{2}$	(Wrigley Field)
Duncan	-	Bird	0—1	(Woodburn Hall)
Jordan	-	Kukoc	1—0	(Staples Center)

- A Miller—Barkley ticket was \$9.99 and Purdue bought 200 tickets.
- The ticket at the Duncan—Bird match was \$8.50 (120: Iowa, and 80: Purdue).
- Jordan—Kukoc tickets were \$12.00 each (60 went to IU Bloomington, 60 to the University of Minnesota and 240 to the University of Michigan).

You need to design a database to store this data in it.

8.1 Storing and Retrieving Information

Storing data is extremely important; how else can we remember, if not by carefully storing and organizing data. Even more important is the information retrieval that follows: usually, we are not interested in just remembering the raw data. Information retrieval is a term mainly carrying the connotation that more than just the raw data is obtained through this process. Consider, for example, in the problem above potential questions one might ask, and compare these with the raw data listed above and stored in the database:

1. List all the players and their hometowns and rates.
2. List all the locations available for all events with the leasing rate.
List locations even if no games are scheduled at that location yet.
3. List all the players that charge more than \$250 per match.
4. List all the matches whose ticket prices are above \$7.00
5. List all the matches with number of tickets sold (per match).
6. List the most expensive player(s)?
(Assume Bird and Barkley cost as much as Jordan).

7. What's the venue with the smallest capacity?
(Same as above, assume Wrigley, Woodburn and RCA Dome are the same size).
8. List all the matches, number of tickets sold and venue capacity for each game.
9. List all the matches and the percentage of seats they sold.
What's the match that had the least percent of seats sold?
10. What is the total cost of the players (per round)?
11. What's the average cost of player (per match)?
(Does it matter that the roster could be bigger than the invited players?)
12. Which players cost above average?
(Take the average from the prices listed in players).
13. List the number of players that cost above average.
14. List all universities and the number of games they bought tickets to.
Can you identify universities that bought tickets to ALL/NONE of the games?
That's definitely an interesting question (with two possible answers).
15. List the games attended by Indiana University students.
16. List the average attendance of Indiana Univ. students per game.
17. List the total amount of money IU students paid for tickets.
18. Calculate entire amount spent on players up to (and including) round 4 (not 5).
19. Calculate income from tickets sold to matches up to (and including) round 4.
(What if the question were to ask up to an earlier round, or selected rounds?)
20. Calculate the total amount spent on leasing the rooms.
21. Calculate the net profit up to (and including) round four. (This will be the difference between income from tickets and the sum of expenses with players and rent. Perhaps it's better to calculate or take a look at the next query first.)
22. Calculate the profit per match and round (two queries).
23. Calculate the average profit per match and round (two queries).
24. List the matches and rounds with an above average profit (two queries).
25. How many students saw Bird playing?
(How many tickets were sold for games he was in?)
26. Who was the most watched player?
27. How much money did each player bring?
28. List the standings after the fourth round.
29. Which matches only had IN students in the audience?
(Purdue is in Indiana as is IU).
30. Which University did not purchase tickets to any game?
31. Which University did not purchase tickets to the Bird-Jordan game?
32. List the Universities sorted by the number of games they purchased tickets to.

Even a quick look at the list of questions above will help clarify the difference between raw data and information retrieval: most of these questions don't have any direct relationship with the actual data stored in the database. To answer these questions, each one of them, knowledge of SQL (Structured Query Language) is required. The SQL programs that retrieve the information requested by the questions listed above are called queries. Queries are a type of programming, and practice, knowledge and creativity play an important role in their development, as expected.

Before we can start retrieving information we must be concerned with storing data.

8.2 The Entity-Relationship Diagram

The key element we will discuss in this section is: database design. The model that we use is the relational one. That means that data will be stored in tables in a RDBMS (relational database management system). SQL is the language that allows us to communicate with the RDBMS: queries are formulated in SQL, as are all the instructions that result in the creation of tables and their population with data.

Relational database design has well established procedures and criteria; it will be slightly beyond the scope of this course to discuss the four normal forms, although they will be mentioned in class. But just like the situation in which if the initial decomposition is carefully done then the resulting third normal form database also satisfies the requirements of the fourth normal form (no non-trivial multivalued dependencies) we will present a simple recipe which can immediately produce a design that is in many respects in the neighborhood of being optimal².

8.2.1 Entities

The procedure starts by listing the entities, together with their attributes. Entities are self-sufficient; you don't need to refer to anything else to define them. In our problem players, venues, spectators qualify as entities (and so are the matches, if you want to think of them as events listed in the tournament schedule³). Once we have the entities we can create a table for each, with a number of columns equal to the number of features (attributes) for that entity, and we can start entering data in the tables.

Here's some data for the entities in this problem:

username	first name	last name	street	city	state	rate/match
cbarkley	Charles	Barkley		Phoenix	AZ	\$450.00
lbird	Larry	Bird		Naples	FL	\$350.00
mjordan	Michael	Jordan		Chicago	IL	\$500.00
rmiller	Reggie	Miller		Indianapolis	IN	\$150.00
tduncan	Tim	Duncan		Houston	TX	\$210.00
tkukoc	Toni	Kukoc		New Zagreb	PA	\$50.00

For venues and spectators the following considerations will apply: there are no individual spectators, universities will buy a certain number of tickets and they will be responsible for the distribution to individuals; the spectators will then be universities,

²These are all very well known things, no new breakthroughs presented here.

³Matches, for example those in the NBA playoffs, are completely specified and clear even before the season starts; we don't need to know the final standings before we can accept the idea of a playoff match.

listed in a table with that name; venues will be listed in a table of venues, with relevant attributes; all these tables will resemble rosters, as in class rosters. We simply list all the players, universities, venues and matches we have knowledge of. Here are the spectators and venues tables for this problem. The spectators table:

customer id	logo (picture)	address	city	state
Illinois				IL
Iowa				IO
IU Bloomington				IN
Michigan				MI
Purdue University				IN
Stanford University				CA
U of Minnesota				MN

The venues table:

location	address	city	state	leasing rate	capacity
Madison Square Garden		New York	NY	\$1,200.00	4,000
Rawles Arena		Las Vegas	NV	\$740.00	3,400
RCA Dome		Indianapolis	IN	\$230.00	1,800
Soldier Field		Chicago	IL	\$800.00	2,800
Staples Center		Los Angeles	LA	\$500.00	3,200
Woodburn Hall		Atlanta	GA	\$600.00	2,000
Wrigley Field		Chicago	IL	\$800.00	1,200

There is little that we need to say about the matches table. It basically lists 5 rounds of three games, for a total of 15 game entries. There's really no need to present this table here⁴ in more details. It will however come up again in our discussion pretty soon, and we will discuss it a bit more then.

8.2.2 Relationships

If all we had were the entities there would be no reason for us to call them as belonging to the same problem. What brings the entities together are the interconnections between them, also called relationships. Relationships are also entities, in the sense that each one could claim a table, with columns for features (attributes) etc. What distinguishes them from the entities listed earlier is that those entities are self-sufficient; relationships however can't be defined in entities' absence, so they are not self-contained. There are three relationships we can identify in our problem:

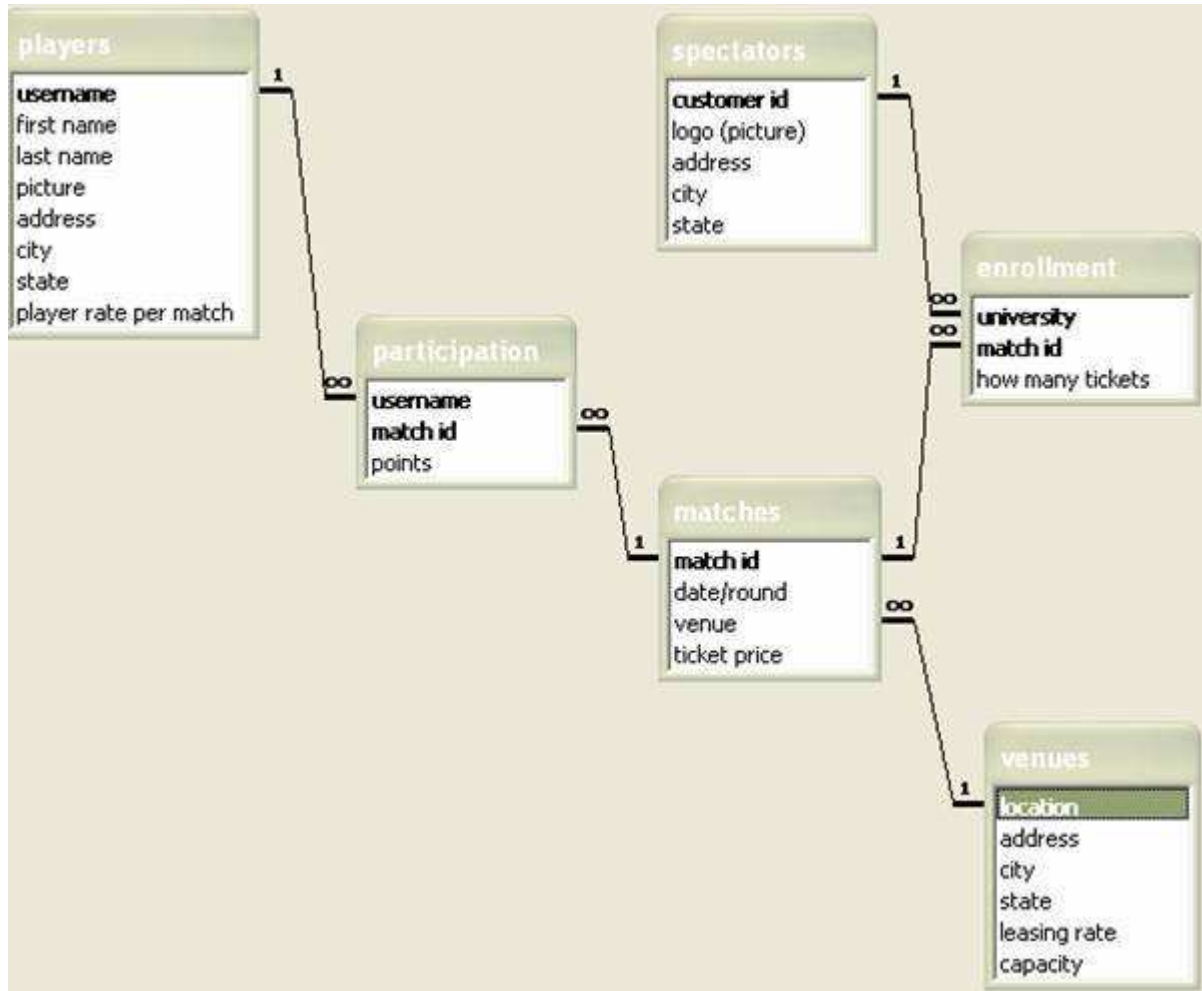
- players participate in matches
- spectators enroll (buy tickets) to see the matches
- matches are scheduled at venues

The recipe that we mentioned earlier asks that entities be listed first, and keys for the entities tables be determined. Next relationships must be identified, and for each relationship a table be created, containing all the keys of the entities that it relates, plus a few specific attributes, if any. In our problem two tables should be created

⁴Again, think of the match slots listed in the NBA playoffs or in the NCAA tournament.

for the the first two relationships listed above: we will call them **participation** and **enrollment**. For the third one, the special nature of this relationship (it is a one-to-many relationship from venues to matches) allows us to simply store it in an already existing table, the one for entity **matches**.

Here's the the entity relationship diagram for this database, as described earlier:



8.3 Homework Assignment Three

The assignment will be to set this database up, populate it with data, and write the SQL queries listed earlier. We need to describe a RDBMS with which we can work to accomplish all of the above. Our choice will be MySQL, and its installation will form the object of the next chapter.

Chapter 9

MySQL

MySQL is a GNU database server. A database is an information storing house, used by web and other applications to save and retrieve data. Databases allow fast and easy information retrieval and storage. MySQL is natively supported by many web programming languages. It's fast, easy to use, free for non-commercial use, and easy to set up. This last step is what we'll be covering here.

9.1 Installation of MySQL

As we will show, installation of MySQL is quite easy:

1. Download the MySQL source: go to <http://dev.mysql.com>, select **Downloads** on the page and select the current release marked **GA** (generally available)¹. From the different versions available, select the source file in **.tar.gz** (tarball) format and download it to your desktop. This file will also be available in the software repository folder² mentioned earlier when we installed Apache.
2. Connect to `silo.cs.indiana.edu` and move to your³ **nobackup** folder:

```
cd /nobackup/[username]
```

By now everybody has a `/nobackup` folder.

3. Drag the MySQL **.tar.gz** file you downloaded to the remote side, uploading the file (or copy it from the software repository folder:

```
bash-3.00$ pwd
/nobackup/dgerman
bash-3.00$ cp /1/www/classes/a348/sum2006/software/mysql* .
bash-3.00$ ls -ld mysql*
-rw-r--r-- 1 dgerman faculty 19542405 Jul  9 01:54 mysql-5.0.22.tar.gz
bash-3.00$
```

4. Once the upload is done, unzip and unarchive the file with **gunzip** and **tar xvf::**

¹As of July 2006 this means: MySQL 5.0.22 and file name `mysql-5.0.22.tar.gz`

²The folder is `/1/www/classes/a348/sum2006/software` and contains all software needed.

³Remember to write your own username in place of `[username]`

```

bash-3.00$ ls -ld mysql*
-rw-r--r-- 1 dgerman faculty 19542405 Jul  9 01:54 mysql-5.0.22.tar.gz
bash-3.00$ gunzip mysql*.gz
bash-3.00$ ls -ld mysql*
-rw-r--r-- 1 dgerman faculty 97884160 Jul  9 01:54 mysql-5.0.22.tar
bash-3.00$ tar xvf mysql*.tar
[...]
bash-3.00$

```

5. Move into the directory created by tar xvf:

```

bash-3.00$ ls -ld mysql*
drwxr-xr-x 38 dgerman faculty 1024 May 25 05:32 mysql-5.0.22
-rw-r--r-- 1 dgerman faculty 97884160 Jul  9 01:54 mysql-5.0.22.tar
bash-3.00$ rm mysql*.tar
bash-3.00$ ls -ld mys*
drwxr-xr-x 38 dgerman faculty 1024 May 25 05:32 mysql-5.0.22
bash-3.00$ cd mysql*2
bash-3.00$ pwd
/nobackup/dgerman/mysql-5.0.22
bash-3.00$

```

6. Next, run the following commands to install the program:

```

./configure --prefix=/nobackup/[username]/mysql
make
make install

```

7. In order to configure MySQL for use, run the following command:

```

/nobackup/[username]/mysql/bin/mysql_install_db \
--user=[username] \
--datadir=/nobackup/[username]/mysql

```

(Note that backslashes at the end must be immediately followed by newlines.)

8. Now with the server configured, you can write a script to easily start it. Create a file `~/bin/mysql_start` with the following contents:

```

/nobackup/[username]/mysql/bin/mysqld_safe \
--user=[username] \
--pid-file=/nobackup/[username]/mysql/mysqld.pid \
--log=/nobackup/[username]/mysql/mysqld.log \
--socket=/nobackup/[username]/mysql/mysql.sock \
--basedir=/nobackup/[username]/mysql \
--datadir=/nobackup/[username]/mysql \
--port=[your assigned mysql port] &

```

9. Make the file executable:

```

chmod 700 ~/bin/mysql_start

```

and test your server by running the newly created file:

```

~/bin/mysql_start

```

10. Change the root password:

```
/nobackup/[username]/mysql/bin/mysqladmin \  
--port=[your port] \  
--socket=/nobackup/[username]/mysql/mysql.sock \  
-u root password '[password]'
```

11. Connect as root, create a user and a database for the user:

```
bash-3.00$ cat ~/bin/root  
mysql --socket=/nobackup/[username]/mysql/mysql.sock \  
--port=[your port] -u root -p  
  
bash-3.00$ ./root  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 5 to server version: 5.0.22-log  
  
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.  
  
mysql> use mysql  
Reading table information for completion of table and column names  
You can turn off this feature to get a quicker startup with -A  
  
Database changed  
mysql> create user '[username]'@'silo.cs.indiana.edu' IDENTIFIED BY '[password]';  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> create database demoOne;  
Query OK, 1 row affected (0.03 sec)  
  
mysql> grant all on demoOne.* to '[username]'@'silo.cs.indiana.edu';  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> exit  
Bye  
bash-3.00$
```

12. To bring the server down:

```
bash-3.00$ cat ~/bin/mysql_stop  
/nobackup/[username]/mysql/bin/mysqladmin \  
--socket=/nobackup/[username]/mysql/mysql.sock \  
--port=[your port] -u root -p shutdown  
bash-3.00$ ~/bin/mysql_stop  
Enter password:  
STOPPING server from pid file /nobackup/[username]/mysql/mysqld.pid  
060709 03:14:20  mysqld ended  
  
bash-3.00$
```

9.2 Testing Your Installation

Database access from Perl is supported by the Perl/DBI module, already installed. Create a file `~/programs/one` with the following contents:

```
#!/usr/bin/perl

use DBI;

$DB      = "dbi:mysql:demoOne:silo.cs.indiana.edu;port=[your port]";
          # data source name (database)

$username = "[username]";           # username
$password = "[password you set]";   # password

$dbh = DBI->connect($DB,
                   $username,
                   $password, {PrintError => 1}) ||
      die "Couldn't open database: ", $DBI::errstr;

print "I have opened the database...\n";
```

Make the file executable and run it; if the program is able to open the database with no errors your installation has been successful.

9.3 Minute Papers and Exercises

It is important to remember the main points of the installation (settings, commands for starting, stopping the database server), and the need to set a new password for the root as soon as installation is completed; one should also remember the procedure for creating a new database, a new user, and of granting database rights to a user. There are two ways of accessing the database server: one from the command line (we only created a user, as root. from the command line), and through a Perl/DBI program. Our Perl/DBI program was very simple and its interaction with the database was minimal. The main goal for the immediate future is that of creating the tables and populating them with data in preparation for Homework Assignment Three.

Chapter 10

PHP

PHP (originally Personal Home Page, later PHP Hypertext Preprocessor) is a programming language which allows the mixture of HTML markup and code to produce web output. PHP allows for easy and fast coding of web programming by combining simple markup and code. PHP also easily connects to MySQL, as well as a host of other data sources.

10.1 Installation of PHP

Installation proceeds in about twelve steps which should take about twenty minutes of your time, if you are careful. Here's what you need to do:

1. Go to <http://www.php.net> and select **Downloads** on the top. Select the complete source link marked `.tar.gz`¹. Save the file on your desktop. As mentioned before the needed compressed archive will also be found in the class software repository folder: `/1/www/classes/a348/sum2006/software`.
2. Open SSH and connect to `silو.cs.indiana.edu`.
3. Move into your `nobackup` directory by typing `cd /nobackup/[username]` where instead of `[username]` you should write your real IUB (and `silو`) username. Either upload the PHP archive to your folder or copy it from the shared directory, then uncompress it:

```
bash-3.00$ cp /1/www/classes/a348/sum2006/software/php* .
bash-3.00$ ls -ld php*5*
-rw-r--r--  1 dgerman faculty 8109575 Jul  9 03:42 php-5.1.4.tar.gz
bash-3.00$ gunzip php*.gz
bash-3.00$ ls -ld php*5*
-rw-r--r--  1 dgerman faculty 45649920 Jul  9 03:42 php-5.1.4.tar
bash-3.00$
```

4. Unarchive the file with `tar xvf` and then remove the archive. The output on the screen shows all the files that are being created. Note that before each of the files is the name of a new folder (in my case `php-5.1.4`) that the archive program created. Move into that folder; here's how that worked for me:

¹As of July 2006 the latest PHP version is 5.1.4 and the file name `php-5.1.4.tar.gz`

```

bash-3.00$ tar xvf php*5*.tar
[...]
bash-3.00$ ls -ld php*5*
drwxr-xr-x 14 dgerman faculty      2048 May 12 10:41 php-5.1.4
-rw-r--r--  1 dgerman faculty 45649920 Jul  9 03:42 php-5.1.4.tar
bash-3.00$ rm php*5*.tar
bash-3.00$ cd php-5.1.4
bash-3.00$ pwd
/nobackup/dgerman/php-5.1.4
bash-3.00$

```

5. Configuration of PHP is complicated, so we'll make a shell script called `try` to allow for easy debugging. Set the following contents for `try`:

```

./configure \
--with-mysql=/nobackup/[username]/mysql \
--with-apxs2=/u/[username]/apache/bin/apxs \
--with-config-file-path=/u/[username]/apache/conf \
--with-xml --enable-track-vars \
--prefix=/u/[username]/apache

```

6. Next, run the following commands to install the program:

```

chmod 700 try
./try (this is the file you created)
make
make install

```

7. Move into the Apache configuration directory:

```

bash-3.00$ pwd
/nobackup/dgerman/php-5.1.4
bash-3.00$ ls -ld php*ini*
-rw-r--r--  1 dgerman faculty 41989 Feb  8 18:43 php.ini-dist
-rw-r--r--  1 dgerman faculty 46069 Feb  8 18:43 php.ini-recommended
bash-3.00$ cp ./php.ini-dist ~/apache/conf/php.ini
bash-3.00$ cd ~/apache/conf
bash-3.00$ nano httpd.conf

```

8. We need to edit the `httpd.conf` file for this installation. This file contains various configuration options for Apache. We typed `nano httpd.conf` to start the `nano` text editor. Find the line marked `AddType` in the conf file² and add the following two lines below it to tell Apache how to handle php:

```

AddType application/x-httpd-php .php .phtml
AddType application/x-httpd-php-source .phps

```

We also moved `php.ini` in the same folder:

9. We need to stop and start apache in order to allow the changes to take effect:

```

~/apache/bin/apachectl stop
~/apache/bin/apachectl start

```

²It's line 290/407 for me.

10. Create a file `~/apache/htdocs/one.html` with the following contents:

```
<? phpinfo() ?>
```

Access the file from `http://silo.cs.indiana.edu:[your apache port]/one.php` to verify that your installation was successful.

10.2 PHP Configuration

We need to make a couple of changes to `php.ini`, and then restart the server:

1. Create a folder `~/apache/phpsessions` in your server root.
2. Make the following two changes in `~/apache/conf/php.ini`:

```
session.save_path = "/u/[username]/apache/phpsessions"
```

and

```
register_globals = On
```

These are lines 892 and 399/1204 respectively, for me. Then restart Apache.

3. Create `~/apache/htdocs/two.php` with the following contents:

```
<html><head><title>Simple ATM</title></head><body bgcolor="white">

<? if ($fun == "add") {
    $acc += $arg;
  } else if ($fun == "sub") {
    $acc -= $arg;
  } else {

  }
?>
```

Current accumulator is: `<? echo $acc ?>` `<p>`

```
<form method="POST" action="<? echo $SCRIPT_NAME; ?>">
  Amount: <input type="text" name="arg"> <p>
  Action: <select name="fun">
    <option value="non"> Click Me!
    <option value="add"> Deposit
    <option value="sub"> Withdraw
  </select> <p>
  <input type="hidden" name="acc" value="<? echo $acc; ?>">
  Fill in the form then push: <input type="submit" value="Proceed">
</form>

</body></html>
```

Verify that this program works as expected.

4. Create another file `~/apache/htdocs/three.php` whose contents is as follows:

```

<? session_start();
    if (session_is_registered("acc")) {
        if ($fun == "add") $acc += $arg;
        else if ($fun == "sub") $acc -= $arg;
    } else {
        $acc = 0;
        session_register("acc");
    }
?>

<html>
<head>
    <title>Sessions Examples</title>
</head>
<body bgcolor=white>
    <form method="POST" action="<? echo $SCRIPT_NAME; ?>">
        The current value of the accumulator is: <? echo $acc ?> <p>
        <table cellpadding=2>
            <tr>
                <td> Amount: </td>
                <td> <input type="text" name="arg" size=4> </td>
                <td> Function: </td>
                <td> <select name="fun">
                    <option value="non"> Click Me!
                    <option value="add"> Deposit
                    <option value="sub"> Withdraw
                </select>
            </td>
        </tr>
    </table>
    <p> Enter the amount, select a function, then press
    <input type="submit" value="Proceed"> <p>
</form>
</body>
</html>

```

Verify that the program works as expected.

5. Look into `~/apache/phpsessions` to verify that sessions get created while working with the program listed at the previous step:

```

bash-3.00$ pwd
/u/dgerman/apache/phpsessions
bash-3.00$ ls -l
total 1
-rw----- 1 dgerman faculty 8 Jul  9 04:24 sess_e744af2e02425c4a23b5a9cfc170bad0
bash-3.00$

```

Note the difference between `two.php` and `three.php` with respect to reloading.

6. Finally we are ready to test the database connectivity from PHP.
Create a third file, `~/apache/htdocs/four.php` with the following contents:

```

<?php

$link = mysql_connect('silo.cs.indiana.edu:13297', '[username]', '[password]');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);

?>

```

Make sure MySQL is up, then access the script on-line. If the script reports connecting successfully your installation is in good shape.

10.3 PHP Programming

Let's discuss the basics of PHP; we will use a pattern already seen, in the hope that you will find the development somewhat familiar.

10.3.1 Basic Concepts

We start from a basic HTML file which, stored with extension `.php` in `htdocs`, is a valid PHP script.

```

<html>
  <head><title>One</title></head>
  <body bgcolor=white>
    <h1>Hello!</h1>
    
  </body>
</html>

```

Behind every PHP program there is a processor, existing like a parallel universe; the tags allow you to tunnel into that universe and program it. Below we define a variable, and use it in specifying the URL for the picture.

```

<html>
  <head><title>One</title></head>
  <? $name = "lh08.gif"; ?>
  <body bgcolor=white>
    <h1>Hello!</h1>
    
  </body>
</html>

```

Next we determine the picture to be one of many: an array of possible pictures is defined, a value for the index is chosen, and the name of the picture to be shown is produced by the combination of array name with the value of the index. Although there is virtually no visible difference with the program shown at the

previous step the one below has a distinct potential which will be exploited shortly³.

```
<html>
  <head><title>One</title></head>
  <? $images = array("lh08.gif", "lh07.gif", "lh09.gif", "lh01.gif");
    $index = 0;
    $name = $images[$index];
  ?>
  <body bgcolor=white>
    <h1>Hello!</h1>
    
  </body>
</html>
```

Generating a random⁴ number in PHP is a bit different from its Perl equivalent, and the length of the array is calculated in a very specific manner; but now we can generate a new random image with every access and a link in the page simplifies access for the user:

```
<html>
  <head><title>One</title></head>
  <? $images = array("lh08.gif", "lh07.gif", "lh09.gif", "lh01.gif");
    $index = rand(0, sizeof($images) - 1);
    $name = $images[$index];
  ?>
  <body bgcolor=white>
    <h1>Hello!</h1>

    The image below has index <?=$index?>. <p>

    Click <a href="<?=$SCRIPT_NAME?>">here</a> for a new random image.

    <p> 

  </body>
</html>
```

The one observation we should make is that access to the entries of the %ENV hashtable is somewhat simplified.

10.3.2 User Input with PHP

The basic idea when developing a PHP program is that the user interface could be developed first. Thus, if we were to develop the Lindley portfolio, we could start from a basic layout in which we would determine from the outset what will go where. As soon as the specific places have been determined we can

³The value of the index can be randomly chosen every time, thus producing a new picture with every new access.

⁴<http://www.php.net/manual/en/function.rand.php>

start writing the code for producing their specific values. Here's an example, below, in which only one of the four links in the portfolio program is being programmatically determined; likewise the picture to be shown is determined by the action of a small PHP scriptlet:

```
<html><head><title>Lindley portfolio</title></head><body>

<? echo $QUERY_STRING ?> <p>

<table width=100% cellpadding=2>
<tr> <td align=center>

<? if ($QUERY_STRING == "seven") {

    echo "Lindley 07";

    } else {

    echo "<a href=\"\$SCRIPT_NAME?seven\">Lindley 07</a>";

    }
?>
    </td> <td align=center> Lindley 08
    </td> <td align=center> Lindley 01
    </td> <td align=center> Lindley 09
    </td>
</tr>
<tr> <td colspan=4 align=center>

<? if ($QUERY_STRING == "seven") {
    echo
        "<img src=\"http://www.cs.indiana.edu\" .
        "/dept/img/lh07.gif\">";
    } else {
    echo
        "<img src=\"http://www.cs.indiana.edu\" .
        "/l/www/classes/a202-dger/sum99/a202.gif\">";
    }
?>

    </td>
</tr>
</table></body></html>
```

We notice that the concatenation operator is `.` (the dot, just as in Perl). We also remember that the portfolio relies on the value of the `QUERY_STRING` entry in the environment hashtable. Just as we noticed the use of `$SCRIPT_NAME` earlier, we point out the binding of a value to a variable whose name is the entry in the hashtable where the value is stored. The specific setting we have chosen for PHP (register globals is on) is responsible for the simplification; this setting is in fact a very sensitive one. Therefore at the end we should graduate to a more robust

use pattern. Until then, if we program with care, the simplifications induced by this set up will greatly enhance our understanding. We now look at the data bindings that occurs when a form is submitted to a PHP script.

10.3.3 HTML Forms in PHP

Earlier when we discussed Perl and CHGI we saw any web script receives a string of field names and their associated values in a certain pattern and that the browser is responsible for putting together this string and making sure that the special characters are being encoded, so they don't get lost. PHP has a similar action. In particular it even simplifies data binding a bit:

```
<html><head><title>Bank Accounts</title></head><body>

    Action selected: <?=$fun?> <p>
    Amount specified: <?=$arg?> <p>

    <form method="post" action="<?=$SCRIPT_NAME?>">
        Amount: <input type="text" name="arg"> <p>
        Action: <select name="fun">
            <option value="non"> Click Me!
            <option value="add"> Deposit
            <option value="sub"> Withdraw
        </select> <p>
        Fill in the form then push: <input type="submit"
                                   value="Proceed">
    </form></body></html>
```

The example above demonstrates the tag `<?=. . . ?>` which is used to include the value of arbitrary expressions on the page. It also demonstrates the data binding that we mentioned: the fields named `fun` and `arg` appear in the form and the user types (or selects from the provided drop-down) the data that corresponds to the named fields and submits the form. When the PHP script to which the form is submitted starts, a variable is created for each such field; thus, we find the value of the `fun` field in a variable called `$fun` and the value the user typed in the `arg` field, in a variable `$arg`.

You can experiment with the form above: in the beginning there won't be any values listed for the two variables. Set the fields in the form to whatever values you want and submit the form; the values you sent will be shown by the script, at the top of the page.

10.3.4 Keeping State on the Client Side

With the mechanism described in the previous section we can easily produce a simple version of the calculator program discussed in the Perl/CGI chapter. Let's say we keep the state at a minimum of one variable: the balance will be stored in a hidden field by the name of `acc` (standing for accumulator), and we drop the `message` state variable we used in CGI. Then there are only two changes to the script from the previous stage, and the calculator is finished,

namely: add a hidden field to the form and process the user input at the top before displaying a new form⁵.

```
<html><head><title>Bank Accounts</title></head><body>

<? if ($fun == "add") {
    $acc += $arg;
  } else if ($fun == "sub") {
    $acc -= $arg;
  } else {

  }
?>

Current accumulator is: <? echo $acc ?> <p>
<form method="post" action="<? echo $SCRIPT_NAME; ?>">
  Amount: <input type="text" name="arg"> <p>

  Action: <select name="fun">
    <option value="non"> Click Me!
    <option value="add"> Deposit
    <option value="sub"> Withdraw
  </select> <p>
  <input type="hidden" name="acc" value="<? echo $acc; ?>">

  Fill in the form then push: <input type="submit" value="Proceed">
</form></body></html>
```

10.3.5 Keeping State on the Server Side

PHP comes equipped with a mechanism by which session IDs can be associated with browser calls. For each such ID a file can be created on the server side in which information specific to that user ID can be stored. These files are called sessions and we set up PHP to store them in a folder `phpsessions` under `~/apache`. In this section we present an example of server-side state management using sessions. PHP provides the following methods:

- `session_start()`
Checks to see if an ID is submitted by the browser with the request, and if that ID is already available on the server. If it's not, a new one will be generated, and will be sent to the browser. This is the beginning of a new session. The browser has to keep it and submit it with every new request; if an ID matches what the server has then we know who we are talking to, so we refer to the corresponding file in the `session.save_path` directory.
- `session_register("varName")`
Is used to associate variable names with values. These are stored under the appropriate session ID file on the server side (see the example below).

⁵This, in fact, is our CGI/Perl pattern: retrieve state, name user input (PHP does that automatically for you) check state and initialize if empty or update state based on user input, save the new state, report it and get ready for more input.

They are retrieved from that file during subsequent calls in the same session (that is, for the same session ID).

- `session_is_registered("varName")`
Is used to check if a variable is registered already or not (that is, if it's stored on the server side in the file for this session ID).
- `session_unregister("varName")`
Is used to delete a variable from the file on the server associated with the session in which this instruction is run.
- `session_destroy()`
Deletes the session, that is, the file.

Here's the example:

```
<? session_start();
  if (session_is_registered("acc")) {
    if ($fun == "add") $acc += $arg;
    else if ($fun == "sub") $acc -= $arg;
  } else {
    $acc = 0;
    session_register("acc");
  }
?>
<html><head><title>Sessions Examples</title></head><body>

  <form method="POST" action="<? echo $SCRIPT_NAME; ?>"
    The current value of the accumulator is: <? echo $acc ?>

  <p> <table cellpadding=2>
    <tr>
      <td> Amount: </td>
      <td> <input type="text" name="arg" size=4> </td>
      <td> Function: </td>
      <td> <select name="fun">
        <option value="non"> Click Me!
        <option value="add"> Deposit
        <option value="sub"> Withdraw
      </select>
      </td>
    </tr>
  </table>

  <p> Enter the amount, select a function, then press

  <input type="submit" value="Proceed"> <p>

  </form>
</body>
</html>
```

It is worth pointing out that the `session_start()` invocation must be the very first line in the script. Checking whether the accumulator variable is registered or not is the same as retrieving state and asking if the state is empty or not. Initializing the state amounts to registering the desired variable (note that the name is passed to the session register function, that is, a string not a variable name). The connection between session entries and the variables in the program is transparent; just make sure you don't have a hidden field and a session entry (or variable) with the same name. Notice that storing the state is also transparent, every change to a variable associated (by name) with a session entry will also be automatic, transparent. Thus there is no need for the hidden fields, and the entire process becomes much more secure.

10.4 Homework Assignment Four

The assignment will be to implement the flag quiz using PHP; two implementations are needed, one with hidden fields, and the other one using sessions. The section below presents additional examples and shows how once an implementation of the problem exists (let's say, in PHP with sessions) the other two (CGI/Perl with hidden fields and PHP with hidden fields) can be immediately obtained through an almost automatic program transformation.

10.5 Program Transformations

Let's implement a simple game and in the process use the template that we have been mentioning. In this game the computer and the user take turns. The user starts first and enters a number. The number must be between 1 and 10, but in the code we develop below, for simplicity, we don't check that. The user's number is added to a common balance, which is 0 (zero) in the beginning. Then the computer randomly picks a number between 1 and 10, which gets added to the common balance, and then it's again the user's turn. Whoever makes the balance 100 or bigger first is the winner.

To write this program we apply our pattern, and will thus see how easy it is to develop the program regardless of the actually technology (PHP with sessions, PHP with hidden fields or CGI with hidden fields) we commit to. The pattern (we remember) looks like this:

1. retrieve state (this is somewhat technology dependent)
2. name (label) user input
3. if state is empty:
 - initialize state
- else:
 - update state based on user input
4. store state
5. report state
6. get ready for more input

So the first thing we need to do is to determine what our state will look like. We decide to have two variables in our state: a `message` and a balance (we will

call it `sum`). We also need to determine what implementation method we will be using, and we decide for PHP with sessions, first. This, then gives us the following program:

```
<?
    session_start();
    if ($message) {
        $sum = $sum + $user;
        if ($sum >= 100) {
            $message = "You enter: " . $user . "<p> You won: " . $sum;
        } else {
            $comp = rand(1, 10);
            $sum += $comp;
            if ($sum >= 100) {
                $message = "You entered: " . $user . "<p> Computer enters: " .
                    $comp . " <p> Computer won: " . $sum;
            } else {
                $message = "You entered: " . $user . "<p> Computer enters: " .
                    $comp . " <p> Game still going. <p> " .
                    "Current sum is: " . $sum;
            }
        }
    } else {
        session_register("message");
        session_register("sum");
        $sum = 0;
        $message = "Welcome, the sum is: " . $sum;
    }
?>

<form>
    <?=$message?> <p>
    Please enter your number: <input type="text" name="user"> <p>
    <input type="submit" name="action" value="Proceed"> when ready.
</form>
```

As we can see the interface is minimal. The state update is such that any user input is accepted, regardless. This is particularly useful since you can control the outcome of the game quickly. If the user input makes the sum reach 100 or above the user wins. Otherwise the computer gets a chance. The program does not have a reset button, and after one of the users reach 100 the score is not reset either; however with the user input you can enter anything you want including negative values. An easy way to reset the game would be to delete the sessions in the `phpsessions` folder.

Let's change this to hidden fields; please pay attention to the main differences:

- (a) the form now has two hidden fields for the two state variables (save state)
- (b) retrieving state is totally transparent (all session code has vanished too)
- (c) everything else is entirely, completely the same!

It really is very instructive to compare the two programs line by line; here's the PHP with hidden fields script, which you can compare with the program on the opposing page.

```
<?
  if ($message) {
    $sum = $sum + $user;
    if ($sum >= 100) {
      $message = "You enter: " . $user . "<p> You won: " . $sum;
    } else {
      $comp = rand(1, 10);
      $sum += $comp;
      if ($sum >= 100) {
        $message = "You entered: " . $user . "<p> Computer enters: " .
          $comp . " <p> Computer won: " . $sum;
      } else {
        $message = "You entered: " . $user . "<p> Computer enters: " .
          $comp . " <p> Game still going. <p> " .
          "Current sum is: " . $sum;
      }
    }
  }
  } else {
    $sum = 0;
    $message = "Welcome, the sum is: " . $sum;
  }
?>

<form>
  <?=$message?> <p>
  Please enter your number: <input type="text" name="user"> <p>
  <input type="submit" name="action" value="Proceed"> when ready.
  <input type="hidden" name="message" value="<?=$message?>">
  <input type="hidden" name="sum" value="<?=$sum?>">
</form>
```

And finally, into CGI with hidden fields. Let's enumerate the points of interest:

- there is additional overhead at the beginning in Perl: specifying the location of the interpreter, loading (importing) the library, reading and parsing the data and binding it into an object, printing the headers and the initial parts of the HTML document. PHP does all of these by default. (Nothing wrong with Perl either, just specific, just a bit different).
- the state is retrieved from the hidden fields; this is done in a specific way, by reading the values of the parameters with specific instance methods in the object created. It is important that we label the state and the user input, and store these values from the outset in variables, just like PHP did for us without asking. This way the code conversion is going to go much faster.
- the rest is almost unchanged, save for specific minor Perl to PHP differences, such as: generating a random number, having to use `print qq{` at the end, etc.

With this we have completed our first conversion. The program is presented below.

```
#!/usr/bin/perl

use CGI;

$q = new CGI;

print $q->header, $q->start_html;

$message = $q->param('message');
$sum = $q->param('sum');
$user = $q->param('user');

if ($message) {
    $sum = $sum + $user;
    if ($sum >= 100) {
        $message = "You enter: " . $user . "<p> You won: " . $sum;
    } else {
        $comp = int(rand(10));
        $sum += $comp;
        if ($sum >= 100) {
            $message = "You entered: " . $user . "<p> Computer enters: " .
                $comp . " <p> Computer won: " . $sum;
        } else {
            $message = "You entered: " . $user . "<p> Computer enters: " .
                $comp . " <p> Game still going. <p> " .
                "Current sum is: " . $sum;
        }
    }
} else {
    $sum = 0;
    $message = "Welcome, the sum is: " . $sum;
}

print qq{

<form>
    $message <p>
    Please enter your number: <input type="text" name="user"> <p>
    <input type="submit" name="action" value="Proceed"> when ready.
    <input type="hidden" name="message" value="$message">
    <input type="hidden" name="sum" value="$sum">
</form>

};
```

This conversion process is an excellent preparation for the midterm exam: in class you will be asked to submit a program, in writing, and in lab you will be asked to finish it and convert it to the two other ways of implementation.

10.6 Minute Papers and Exercises

You could be given the following code and asked to convert it twice.

```
<? session_start();

if ($message) {
    if ($action == "Reset") {
        $message = "Game reset, you lost the game The number was $secret";
        $lost += 1;
        $message .= "<p> The current score is: ($won - $lost) ";
        $secret = rand(1, 100);
        $message .= "<p> The new secret number is: $secret ";
        $attempts = 0;
        $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
    } else {
        if ($guess == $secret) {
            $message = "Good, you guessed the number ($secret) in " .
                ($attempts + 1) . " attempts. ";
            $won += 1;
            $message .= "<p> The current score is: ($won - $lost) ";
            $secret = rand(1, 100);
            $message .= "<p> The new secret number is: $secret ";
            $attempts = 0;
            $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
        } else {
            $attempts += 1;
            if ($attempts >= 10) {
                $message = "You are out of tries, you lost the game ($secret). ";
                $lost += 1;
                $message .= "<p> The current score is: ($won - $lost) ";
                $secret = rand(1, 100);
                $message .= "<p> The new secret number is: $secret ";
                $attempts = 0;
                $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
            } else {
                if ($guess > $secret) {
                    $message = "The number is ($secret). Try lower. ";
                } else {
                    $message = "The number is ($secret). Try higher. ";
                }
                $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
            }
        }
    }
} else {
    session_register("message");
    session_register("secret");
    session_register("attempts");
    session_register("won");
    session_register("lost");
    $message = "Welcome.";
    $secret = rand(1, 100);
    $message .= "<p> The secret number is $secret ";
    $attempts = 0;
    $message .= "<p> You have " . (10 - $attempts) . " attempts remaining.";
    $won = 0;
    $lost = 0;
    $message .= "<p> The score so far is: ($won - $lost)";
}
?>

<form>
<?=$message?> <p>
Please enter your guess here: <input type="text" name="guess"> <p>
Press <input type="submit" name="action" value="Proceed"> when ready to submit. <p>
Press <input type="submit" name="action" value="Reset"> if you want to start a new game. <p>
</form>
```

The program above implements the well know game: “guess the number in ten tries”. In our implementation, for debugging purposes the program shows the secret number; we truly want to be in control when we run it. The method of implementation is obviously PHP with sessions; when you start the conversion you should clearly identify the user input, the state variables, and the specific methodology involved in retrieving and storing state. The section below gives an example of a sample midterm exam.

10.7 Sample Midterm Exam

Implement a vending machine that sells three products and has the following interface: the three products are listed along with their prices, a text field helps collect a new coin, always one coin at a time, and a **Proceed** button is used for actually entering the coin into the machine. The picture below shows the vending machine after 75 cents have been entered already, and before (or, rather, just as) a new dime is about to be sent in. The credit is updated every time a valid coin is entered. Descriptions for the products also get updated, indicating how much more money is still needed for each product. For products whose price is below the current balance the amount needed is replaced by an active link, through which the product in question can be ordered. If instead of sending the dime (see the picture) we click the link for crackers the interface will change, showing a credit of 0 (zero) and all three products will go in the initial state (no links, just prices). The return screen in this case will also post this message: “Enjoy your crackers. Your change is: 20 cents.” Prices are as listed.

Your credit now is: 75 cents.

Please choose among the available products:

Cookies: (\$1.25) you need 50 cents.

Soda: (\$0.70) [dispense](#)

Crackers (\$.55) [dispense](#)

Or you may decide to enter more coins:

Push when you're ready to enter the coin.

Chapter 11

The DBI.pm Module

PHP provides the session mechanism that allows us to keep state on the server side. In Perl/CGI there is no preexisting such mechanism. Therefore if we want to store state on the server side using Perl/CGI we have to define our own tools. Necessarily we will need to be able to generate unique session IDs on the fly and that's how we will get started.

PHP sessions are flat files; while we could use flat files with Perl/CGI to implement a similar mechanism we follow a method first presented by Lincoln Stein and Doug MacEachern¹ and store state in a relational database management system (MySQL).

This chapter also represents an introduction to the Perl `DBI.pm` module.

11.1 Generating Session IDs

Our first program does not involve any data repository. It simply checks to see if the URL already contains a session ID (a string of eight characters). If a session ID is found the session ID is reported; if none is found, a very special redirection process is started: first a session ID is generated, randomly, and a redirection to a URL ending with this new session ID (and representing this script) is sent. The browser receives the redirection request, makes the request, and since this time around a session ID is found it is also reported and the process ends. Note that it is the server who creates the session ID; it does it randomly, but it could (if it wanted to) check to verify that no other user has the same session ID. Here's the program:

```
#!/usr/bin/perl

#-----(these are the modules we will be using)-----

use CGI;
use Digest::MD5 qw(md5 md5_hex md5_base64);

#-----(these are some constants)-----

$ID_LENGTH = 8; # length of session_id
```

¹Writing Apache Modules with Perl and C

```

#-----(let's get ready to process)-----

$q = new CGI;

my $session_id = &get_session_id(); # check get_session_id below though,
                                   # if we don't have one we create one
                                   # and then we redirect to this script
                                   # with the session added as path info

print $q->header(), $q->start_html(); # redirection is done at the level
                                   # of headers, and that's why this
                                   # script is somewhat difficult to
                                   # debug, and why the beginning of
                                   # the HTML page comes only here

print "Your session ID is: $session_id"; # note that we need to devise a
                                         # mechanism to keep the id's unique

print $q->end_html(); # end of script, helper procedures are defined below
#-----(sub get_session_id)---

sub get_session_id { # this subroutine tries to extract an id from the
                    # path information, and if it does not find one, or
                    # the format of the one it finds is not correct, it
                    # generates a session id with the right format and
                    # redirects the browser to the same script with the
                    # session id appended to the path

    my ($id) = $q->path_info() =~ m:~/([a-h0-9]{ $ID_LENGTH }):; # extract id

    return $id if $id;

    $id = &generate_id(); # if we reach this stage we didn't find
                        # a (valid) id, so we generate a one now

    print $q->redirect($q->script_name() . "/" . $id); # and we call ourselves
                                                    # right away with the id
                                                    # as added path info

    exit 0;
}

#-----(sub generate_id)---

sub generate_id {

    $SECRET = "some secret phrase";

    my $id = hash($SECRET . rand()); # note that hash is defined below

}

#-----(sub hash)---

sub hash {
    my $value = shift; # take the first argument and use it in hexhash
    return substr(md5_hex($value), 0, $ID_LENGTH);
}

```

The code is heavily commented, the redirection poses an interesting problem namely, nothing needs to be printed before the header (which could contain the redirection request) is sent. This is sometimes important when debugging. For the next section we can't stay away from MySQL any longer.

11.2 Session Management

We start this section by connecting to MySQL. MySQL must be started if it's not running already. One usually keeps MySQL running all the time, but you might want to shut it down at times. Verify that it's running and, if it isn't, start it as shown below. Let's now assume that MySQL is up and running.

```
bash-3.00$ ~/bin/mysql_start
Starting mysqld daemon with databases from /nobackup/dgerman/mysql
```

Log into MySQL with the correct username and password and create a table like so:

```
bash-3.00$ mysql --socket=/nobackup/dgerman/mysql/mysql.sock \
  --port=13297 --host=silo.cs.indiana.edu -u dgerman -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 5.0.22-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use demoOne
Database changed
mysql> create table dgerman_accumulator (
  ->   session_id char(8) primary key,
  ->   acc          int,
  ->   modified    timestamp
  -> );
Query OK, 0 rows affected (0.03 sec)
mysql> show tables;
+-----+
| Tables_in_demoOne |
+-----+
| dgerman_accumulator |
+-----+
1 row in set (0.01 sec)
mysql> describe dgerman_accumulator;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default      | Extra |
+-----+-----+-----+-----+-----+-----+
| session_id | char(8)   | NO   | PRI |               |       |
| acc        | int(11)   | YES  |     | NULL         |       |
| modified   | timestamp | YES  |     | CURRENT_TIMESTAMP |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
mysql>
```

As you can see the table has three columns: each session is a row in this table.

Of the three columns in the sessions table one is for the session ID, one for the accumulator (the state, the balance, we implement the simple calculator first), and a special field (called `modified`) will be used to establish the age of the session, in case we want to purge inactive sessions after a certain time.

```
#!/usr/bin/perl

use CGI;
use DBI;
use Digest::MD5 qw(md5 md5_hex md5_base64);

$DB = "dbi:mysql:demoOne:silo.cs.indiana.edu:port=13297";

$username = "dgerman";
$password = "[password]"; # use your own here

$DB_TABLE = "dgerman_accumulator";

$SECRET = "something secret";

$EXPIRE = 30 * 60 * 60 * 24; # one month
$MAX_TRIES = 10;
$ID_LENGTH = 8;
$q = new CGI;

$DBH = DBI->connect($DB, $username, $password, { PrintError => 0 }) ||
    die "Couldn't open database: ", $DBI::errstr;

my ($session_id) = &get_session_id();

my $acc          = &get_state($session_id);

# note: no need to initialize if it's not found

$acc            = &calculate($acc,
                           $q->param('fun'),
                           $q->param('arg'));

&save_state($acc, $session_id);

print $q->header, $q->start_html;

&status($acc);
&show_form();

print $q->end_html;

$DBH->disconnect;

#-----(end of main program)-----
```

```

sub show_form {
    print $q->start_form(),
        "Type an argument: ",
        $q->textfield(-name=>'arg',
                    -value=>'',
                    -override=>1),

        $q->p(),
        "Then please choose a function: ",
        $q->popup_menu(
            -name     => 'fun',
            -values  => ['non', 'add', 'sub'],
            -labels  => { 'non' => 'Click me!',
                        'add'  => 'Deposit',
                        'sub'  => 'Withdraw'
                      },
            -default => 'non'
        ),

        $q->p(),
        "When done please press ",
        $q->submit(-value=>'Proceed'), $q->end_form();
}
#-----(this was our basic form)-----

sub get_session_id {

    &expire_old_sessions();

    my ($id) = $q->path_info =~ m:~/([a-h0-9]{$ID_LENGTH}):o;
    return $id if $id and &check_id($id);

    my $session_id = &generate_id;
    die "Couldn't make a new session_id" unless $session_id;

    print $q->redirect($q->script_name() . "/" . $session_id);
    exit(0);

}
#-----(needed above)-----

sub expire_old_sessions {

    $DBH->do(<<END);
DELETE FROM $DB_TABLE
    WHERE (unix_timestamp() - unix_timestamp(modified)) > $EXPIRE
END

}
#-----(also needed above)-----

```

```

sub generate_id {

    my $tries = 0;

    my $id = &hash($SECRET . rand());

    while ($tries++ < $MAX_TRIES) {
        last if
            $DBH->do("INSERT INTO $DB_TABLE (session_id, acc) VALUES ('$id', 0)");
        $id = &hash($SECRET . rand());
    }

    return undef if $tries >= $MAX_TRIES;

    return $id;

}

sub hash {
    my $value = shift; # take the first argument and use it in hexhash
    return substr(md5_hex($value), 0, $ID_LENGTH);
}
#-----(last one needed)-----

sub check_id {
    my $id = shift;
    return $id
        if $DBH->do("SELECT 1 FROM $DB_TABLE WHERE session_id = '$id'") > 0;
    return $id
        if $DBH->do("INSERT INTO $DB_TABLE (session_id, acc) VALUES ('$id', 0)");
    return '';
}
#-----(retrieve acc)-----

sub get_state {
    my $id = shift;

    my $query = "SELECT * FROM $DB_TABLE WHERE session_id = '$id'";

    my $sth = $DBH->prepare($query) || die "Prepare: ", $DBH->errstr;

    $sth->execute || die "Execute: ", $sth->errstr;

    my $state = $sth->fetchrow_hashref;

    $sth->finish;

    return $state->{acc};
}

```

```

#------(calculate new acc)-----

sub calculate {
  my ($acc, $fun, $arg) = @_;
  return $acc + $arg if $fun eq 'add';
  return $acc - $arg if $fun eq 'sub';
  return $acc;
}

#------(store new acc)-----

sub save_state {
  my ($state, $id) = @_;
  my $sth = $DBH->prepare(<<END) || die "Prepare: ", $DBH->errstr;
UPDATE $DB_TABLE
  SET acc = ?
  WHERE session_id = '$id'
END
  $sth->execute($acc) || die "Execute: ", $DBH->errstr;
  $sth->finish;
}

#------(print current acc)-----

sub status {
  my ($acc) = @_;
  # $acc += 0;
  print "The accumulator is currently $acc. <p>";
}

```

Check the `dgerman_accumulator` table before and after we run this program²:

```

mysql> select * from dgerman_accumulator;
Empty set (0.00 sec)

mysql> select * from dgerman_accumulator;
+-----+-----+-----+
| session_id | acc | modified          |
+-----+-----+-----+
| 942b51f3   | 0   | 2006-07-09 22:49:04 |
| af08b4cc   | 22  | 2006-07-09 22:49:25 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

Everything goes very well.

²Twice: once from the command line, the other time from the web

The code above is not incomprehensible. It does require a certain amount of attention and research. We will leave it as an exercise for you to document the code, perhaps also based on the exercise we develop in the next section.

11.3 Program Transformations

Let's take the program presented in section 10.6 and transform it three times. As you will see, if the program is well developed it won't even take long to convert it. The first conversion is from PHP with sessions into PHP with hidden fields. Please take the look at the code below; as you will see it is the code from the section mentioned earlier with two notable changes: all session related statements have been commented out, and the form contains now five more fields, all of type hidden, for the storing of the state. Please use a pencil to mark these with your own hand in listing below:

```
<? // session_start();
if ($message) {
    if ($action == "Reset") {
        $message = "Game reset, you lost the game The number was $secret";
        $lost += 1;
        $message .= "<p> The current score is: ($won - $lost) ";
        $secret = rand(1, 100);
        $message .= "<p> The new secret number is: $secret ";
        $attempts = 0;
        $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
    } else {
        if ($guess == $secret) {
            $message = "Good, you guessed the number ($secret) in " .
                ($attempts + 1) . " attempts. ";
            $won += 1;
            $message .= "<p> The current score is: ($won - $lost) ";
            $secret = rand(1, 100);
            $message .= "<p> The new secret number is: $secret ";
            $attempts = 0;
            $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
        } else {
            $attempts += 1;
            if ($attempts >= 10) {
                $message = "You are out of tries, you lost the game ($secret). ";
                $lost += 1;
                $message .= "<p> The current score is: ($won - $lost) ";
                $secret = rand(1, 100);
                $message .= "<p> The new secret number is: $secret ";
                $attempts = 0;
                $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
            } else {
                if ($guess > $secret) {
                    $message = "The number is ($secret). Try lower. ";
                } else {
                    $message = "The number is ($secret). Try higher. ";
                }
            }
            $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
        }
    }
}
} else {
```

```

    // session_register("message");
    // session_register("secret");
    // session_register("attempts");
    // session_register("won");
    // session_register("lost");
    $message = "Welcome.";
    $secret = rand(1, 100);
    $message .= "<p> The secret number is $secret ";
    $attempts = 0;
    $message .= "<p> You have " . (10 - $attempts) . " attempts remaining.";
    $won = 0;
    $lost = 0;
    $message .= "<p> The score so far is: ($won - $lost)";
}
?>
<form>
  <?=$message?> <p>
  Please enter your guess here: <input type="text" name="guess"> <p>
  Press <input type="submit" name="action" value="Proceed"> when ready to submit. <p>
  Press <input type="submit" name="action" value="Reset"> if you want to start a new game. <p>
  <input type="hidden" name="message" value="<?=$message?>">
  <input type="hidden" name="secret" value="<?=$secret?>">
  <input type="hidden" name="attempts" value="<?=$attempts?>">
  <input type="hidden" name="won" value="<?=$won?>">
  <input type="hidden" name="lost" value="<?=$lost?>">
</form>

```

The transformation to CGI with hidden fields is just as simple; the pattern then is (considering this particular problem with its choice of state and input variables):

```

#!/usr/bin/perl
use CGI;
$q = new CGI;
print $q->header, $q->start_html;

$message = $q->param('message');
$secret = $q->param('secret');
$attempts = $q->param('attempts');
$won = $q->param('won');
$lost = $q->param('lost');

$guess = $q->param('guess');
$action = $q->param('action');

# [code as before]

print qq{<form>
  $message <p>
  Please enter your guess here: <input type="text" name="guess"> <p>
  Press <input type="submit" name="action" value="Proceed"> when ready to submit. <p>
  Press <input type="submit" name="action" value="Reset"> if you want to start a new game. <p>
  <input type="hidden" name="message" value="$message">
  <input type="hidden" name="secret" value="$secret">
  <input type="hidden" name="attempts" value="$attempts">
  <input type="hidden" name="won" value="$won">
  <input type="hidden" name="lost" value="$lost">
</form>}, $q->end_html;

```

The truth though is that the inner code is not really entirely identical (it actually seldom is *identical*, if ever, but it usually comes pretty close). Here's what it becomes in the program we're working on (the changed lines are indicated with a pound sign). Can you describe the changes we had to make?

```

if ($message) {
  if ($action eq "Reset") {
    $message = "Game reset, you lost the game The number was $secret";
    $lost += 1;
    $message .= "<p> The current score is: ($won - $lost) ";
    $secret = int(rand(100));
    $message .= "<p> The new secret number is: $secret ";
    $attempts = 0;
    $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
  } else {
    if ($guess == $secret) {
      $message = "Good, you guessed the number ($secret) in " .
        ($attempts + 1) . " attempts. ";
      $won += 1;
      $message .= "<p> The current score is: ($won - $lost) ";
      $secret = int(rand(100));
      $message .= "<p> The new secret number is: $secret ";
      $attempts = 0;
      $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
    } else {
      $attempts += 1;
      if ($attempts >= 10) {
        $message = "You are out of tries, you lost the game ($secret). ";
        $lost += 1;
        $message .= "<p> The current score is: ($won - $lost) ";
        $secret = int(rand(100));
        $message .= "<p> The new secret number is: $secret ";
        $attempts = 0;
        $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
      } else {
        if ($guess > $secret) {
          $message = "The number is ($secret). Try lower. ";
        } else {
          $message = "The number is ($secret). Try higher. ";
        }
        $message .= "<p> You have " . (10 - $attempts) . " attempts to do it.";
      }
    }
  }
} else {
  $message = "Welcome.";
  $secret = int(rand(100));
  $message .= "<p> The secret number is $secret ";
  $attempts = 0;
  $message .= "<p> You have " . (10 - $attempts) . " attempts remaining.";
  $won = 0;
  $lost = 0;
  $message .= "<p> The score so far is: ($won - $lost)";
}

```

So the changes we had to make are few, by any standard. Conversion to CGI with server-side state (sessions kept in database tables) is a process whose steps we are

about to describe, a process that greatly benefits from this last transformation. We start by creating a table to hold the sessions:

```

bash-3.00$ mysql --socket=/nobackup/dgerman/mysql/mysql.sock \
> --port=13297 --host=silo.cs.indiana.edu -u dgerman -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12 to server version: 5.0.22-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use demoOne
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> create table dgerman_guessTheNumber (
->  session_id char(8) primary key,
->  message varchar(120),
->  secret int,
->  attempts int,
->  won int,
->  lost int,
->  modified timestamp
-> );
Query OK, 0 rows affected (0.03 sec)

mysql> describe dgerman_guessTheNumber;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default      | Extra |
+-----+-----+-----+-----+-----+-----+
| session_id | char(8)       | NO   | PRI |               |       |
| message    | varchar(120) | YES  |     | NULL         |       |
| secret     | int(11)       | YES  |     | NULL         |       |
| attempts   | int(11)       | YES  |     | NULL         |       |
| won        | int(11)       | YES  |     | NULL         |       |
| lost       | int(11)       | YES  |     | NULL         |       |
| modified   | timestamp     | YES  |     | CURRENT_TIMESTAMP |       |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)

mysql> select * from dgerman_guessTheNumber;
Empty set (0.00 sec)

mysql>

```

The basic template for this program, based on what we have done already, starts by loading the three necessary modules (CGI, DBI and MD5), setting the data source, the username, the password, the name of the table to which we will be connecting to create or update the sessions plus a few more variable with a somewhat more minor role.

The program then creates a CGI query object and connects to the database (through `DBI->connect()`). If the connection succeeds the program prints a valid, well-formed web page indicating that the connection was successful. It then disconnects from the database.

```
#!/usr/bin/perl

use CGI;
use DBI;

use Digest::MD5 qw(md5 md5_hex md5_base64);

$DB = "dbi:mysql:demoOne:silo.cs.indiana.edu:port=13297";
$username = "dgerman";
$password = "[password]";

$DB_TABLE = "dgerman_guessTheNumber";

$SECRET = "something secret";
$EXPIRE = 30 * 60 * 60 * 24; # one month
$MAX_TRIES = 10;
$ID_LENGTH = 8;

$q = new CGI;

$DBH = DBI->connect($DB, $username, $password, { PrintError => 0 }) ||
    die "Couldn't open database: ", $DBI::errstr;

print $q->header, $q->start_html;

print "All is well.";

print $q->end_html;

$DBH->disconnect;
```

The next step is to get a session ID, and initialize state. A number of procedures will assist us. We will list them in stages; here's the first batch:

```
# get the state from the database -----get_state---
sub get_state {
    my $id = shift;
    my $query =
        "SELECT * FROM $DB_TABLE WHERE session_id = '$id'";
    my $sth = $DBH->prepare($query) || die "Prepare: ", $DBH->errstr;
    $sth->execute || die "Execute: ", $sth->errstr;
    my $state = $sth->fetchrow_hashref;
    $sth->finish;
    return $state;
}
```

```

# retrieve the session ID from the path info. if it's -----get_session_id---
# not already there, add it to the path info (more or less) with a redirect
sub get_session_id {

    my (@result);
    &expire_old_sessions();
    my ($id) = $q->path_info() =~ m:~/([a-h0-9]{ $ID_LENGTH }):o;
    return @result if $id and @result = &check_id($id);

    # if we get here, there's not already an ID in the path info
    my $session_id = &generate_id();
    die "Couldn't make a new session id" unless $session_id;
    print $q->redirect($q->script_name() . "/" . $session_id);
    exit 0;
}

# find a new unique ID and insert it into the database -----generate_id---
sub generate_id {
    # create a new session id
    my $tries = 0;
    my $id = &hash($SECRET . rand());

    while ($tries++ < $MAX_TRIES) {
        last if $DBH->do("INSERT INTO $DB_TABLE (session_id) VALUES ('$id')");
        $id = &hash($id);
    }

    return undef if $tries >= $MAX_TRIES; # we failed
    return $id;
}

# check to see that an old ID is valid -----check_id---
sub check_id {
    my $id = shift;

    return ($id, '')
        if $DBH->do("SELECT 1 FROM $DB_TABLE WHERE session_id = '$id'") > 0;

    return ($id, 'The record of your game may have expired. Restarting.')
        if $DBH->do("INSERT INTO $DB_TABLE (session_id) VALUES ('$id')");

    return ();
}

# generate a hash value -----hash---
sub hash {
    my $value = shift;
    return substr(md5_hex($value), 0, $ID_LENGTH);
}

```

```

sub expire_old_sessions { # -----expire_old_sessions---
    $DBH->do(<<END);
DELETE FROM $DB_TABLE
    WHERE (unix_timestamp() - unix_timestamp(modified)) > $EXPIRE
END
}

```

The code above supersedes any earlier definitions of these methods in this chapter; we are trying to be as general as possible. Here's what the code becomes then:

```

#!/usr/bin/perl
use CGI;
use DBI;
use Digest::MD5 qw(md5 md5_hex md5_base64);

$DB = "dbi:mysql:demoOne:silo.cs.indiana.edu:port=13297";
$username = "dgerman"; $password = "sp00n";
$DB_TABLE = "dgerman_guessTheNumber";

$SECRET = "something secret";
$EXPIRE = 30 * 60 * 60 * 24; # one month
$MAX_TRIES = 10; $ID_LENGTH = 8;

$q = new CGI;

# Open the database -----
$DBH = DBI->connect($DB, $username, $password, {PrintError => 0})
    || die "Couldn't open database: ", $DBI::errstr;

# get the current session ID, or make one -----
my ($session_id, $note) = &get_session_id();

# retrieve the state -----
my $state = &get_state($session_id) unless $q->param('clear');

# [1]

# start the page -----
print $q->header,
    $q->start_html(-title => 'Database Sessions with URL Rewriting',
                  -bgcolor => 'white');

# print "Your session ID is: ", $session_id;

# [2]

print $q->end_html;

$DBH->disconnect;

```

Retrieving the state is done in a very general way in the code above; the resulting state is a hashtable. Saving the state is not as problem-independent, since the sessions table has columns with names. Here's the code identified by # [1] above:

```

$message = $state->{'message'};
$secret  = $state->{'secret'};
$attempts = $state->{'attempts'};
$won     = $state->{'won'};
$lost    = $state->{'lost'};

$guess   = $q->param('guess');
$action  = $q->param('action');

# [3]

$state->{'message'} = $message;
$state->{'secret'}  = $secret;
$state->{'attempts'} = $attempts;
$state->{'won'}     = $won;
$state->{'lost'}    = $lost;

# save the modified state -----
&save_state($state, $session_id);

```

Here's the fragment labeled # [2]:

```

print qq{<form>
  $message <p>
  Please enter your guess here: <input type="text" name="guess"> <p>
  Press <input type="submit" name="action" value="Proceed"> when ready to submit. <p>
  Press <input type="submit" name="action" value="Reset"> if you want to start a new game. <p>
</form>};

```

And here's the `save_state` method that we need:

```

# save the state in the database -----save_state---
sub save_state {
  my ($state, $id) = @_;
  my $sth = $DBH->prepare(<<END) || die "prepare: ", $DBH->errstr;
  UPDATE $DB_TABLE
  SET message=?,secret=?,attempts=?,won=?,lost=?
  WHERE session_id='$id'
  END
  $sth->execute(@{$state}{qw(message secret attempts won lost)})
  || die "execute: ", $DBH->errstr;
  $sth->finish;
}

```

You will have noticed, of course, that the names of the columns in the session table are listed in the body of the procedure. This is the reason for which this procedure is not problem-independent.

The code labeled # [3] is our old acquaintance on page 76. There is absolutely no reason to reproduce it here, since it is exactly entirely the same.

You should clean the table of sessions once in a while and look at data as it is being collected while working with this program.

```
mysql> delete from dgerman_guessTheNumber;
Query OK, 9 rows affected (0.00 sec)
```

```
mysql> select * from dgerman_guessTheNumber;
Empty set (0.00 sec)
```

```
mysql> select * from dgerman_guessTheNumber;
+-----+-----+-----+-----+-----+-----+-----+
| session_id | message | secret | attempts | won | lost | modified |
+-----+-----+-----+-----+-----+-----+-----+
| 9af61826 | Welcome. | 79 | 0 | 0 | 0 | 2006-07-10 01:17:28 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql>
```

The message is really much longer, we cut it short to show the table, above.

11.4 Minute Papers and Exercises

Exercises now will be given asking for four implementations, in the styles studied so far. Typical problems include:

- play rock-paper-scissors with the computer, have the computer move first
- play the hangman game with the computer (secret word chosen in the beginning)
- implement the flag quiz discussed earlier
- implement the addition quiz³
- implement a simple dice game⁴

For all these problems you will first have to identify the state, the user input, then use the template we discussed to implement the game.

11.5 Homework Assignment Five

Implement the problem described on page 66 using CGI/DBI and server-side sessions.

³The computer asks random addition questions, numbers are in the range $[-50, 50]$, the user enters the answers and the computer scores them and reports the number of good and total answers. The game ends after 10 questions.

⁴For example a simplified version of the game of craps: computer rolls the dice, user wins, loses or has to match point. Computer rolls the dice again and again until the point is matched, and the user wins, or one of two outcomes lose the game for the user. Another simple game would be for the user to guess the parity of the sum of the two dice, and bet a number of points; computer keeps balance, takes the user's bet, scores rolls of dice against the bet and keeps the game going by rolling the dice again.

Chapter 12

Accidental Reloads

Take a look at this code:

```
<? session_start();

if ($message && !$reset) { // no new user

    if ($age == $ageCopy) {

        if ($n1 + $n2 == $answer) {
            $message = "Good answer";
            $g = $g + 1;
        } else {
            $message = "No way.";
        }
        $t = $t + 1;
        $message .= " Now your score is: " . $g . " out of " . $t;

        $n1 = rand(0, 100);
        $n2 = rand(0, 100);

        $age += 1;

    } else {
        $message = "I am sorry, you can't reload.";
        $message .= "<p>Your score is still: " . $g . " out of " . $t;
    }

} else { // no state, new user, initialize state

    session_register("g");
    session_register("t");
    session_register("n1");
    session_register("n2");
```

```

    session_register("message");

    session_register("age");
    $age = 1;

    $g = 0;
    $t = 0;
    $message = "Welcome, your score is: " . $g . " out of " . $t;
    $n1 = rand(0, 100);
    $n2 = rand(0, 100);
}
?>

<form>
  <?=$message?> <p>
  What is <?=$n1?> + <?=$n2?>? <input type="text" name="answer" size=4> <p>
  <input type="hidden" name="ageCopy" value="<?=$age?>">
  Click to <input type="submit" value="Proceed">
  <p> Click here to <a href="<?=$SCRIPT_NAME?>">reset</a>.
  <p> Click here to <input type="submit" name="reset" value="Reset">
</form>

```

Do you understand what is special¹ about it?

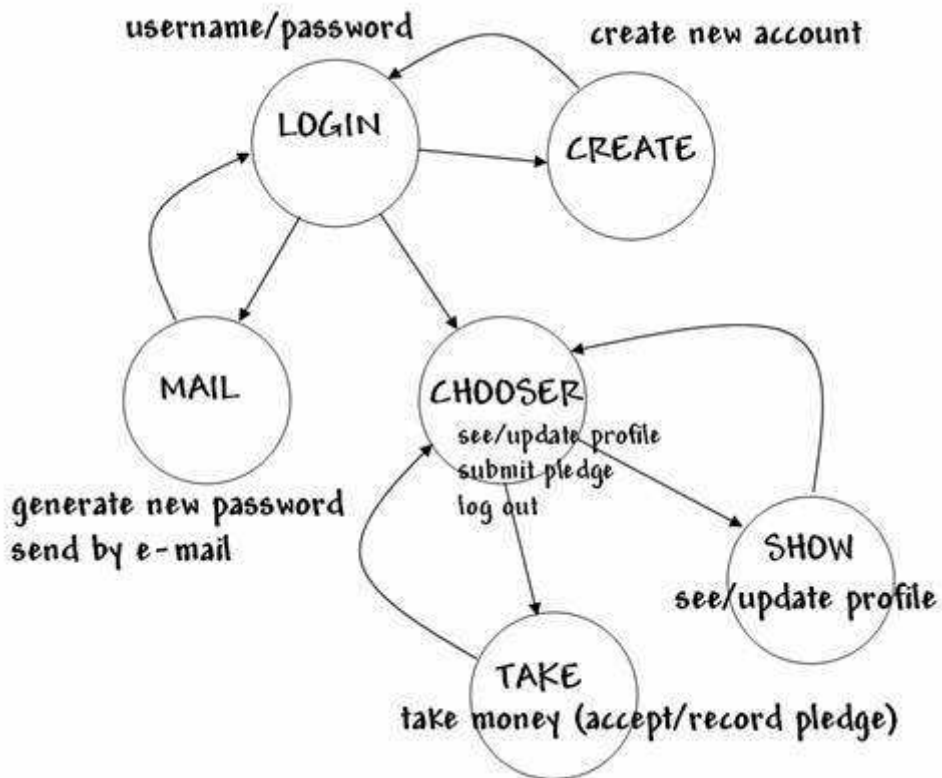
12.1 Minute Papers and Exercises

Make sure that all session-based programs that you have written thus far (and will be writing from now on) are protected against accidental reloads. Change them if you have to; this includes the CGI/DBI server-side state programs as well.

¹Sometimes accidental reloads are undesirable, especially when they update the state. We can prevent them from happening if we stamp the transactions with their age, keep a copy of this stamp on the server side and send another one to the client—and always compare the two before updating the state. If they don't match the state is not updated; if they do, the state is updated (and we include here the stamp itself, the copy kept on the server-side).

Chapter 13

State Machines on the Web



13.1 The Chart

Programs developed so far have one common feature: their user interface is a sole screen. Applications will have more than one screen in general. A methodology must

therefore be developed (or the one we have developed thus far must be extended) so we can accommodate a more dynamic user interface. We develop the template of such a general application, in stages. This is also an example of what the starting point of a semester project could look like.

13.2 Nodes

```
<? session_start();
  if ($source == "login") {

    } elseif ($source == "mail") {

    } elseif ($source == "chooser") {

    } elseif ($source == "create") {

    } elseif ($source == "take") {

    } elseif ($source == "show") {

    } else {
      _login();
    }

    function _login() { ?>
<form method="GET" action="<?=$PHP_SELF?>
  <table>

  <table>
</form> <? }

  function _mail() {

  }

  function _chooser() {

  }

  function _create() {

  }

  function _take() {

  }

  function _show() {
```

```

}
?>

```

So we start with a pencil and a piece of paper and we draw the user interface (screen by screen) for program we have in mind. Then we start developing a first sketch of the program; and the first thing we acknowledge are the nodes of the graph (the screens).

We are implementing a basic account management system. Users log in and update their account in some fashion, then they log out. In this particular case users simply come and pledge money. They can see their transactions if they choose to view their account. A user that does not have an account, or that misplaces the username or password at login will should be given the option to create an account, update the password, have the account information mailed to a different account and so on.

13.3 Data

Before we get into more code let's make sure the database tables are created:

```

bash-3.00$ mysql --socket=/nobackup/dgerman/mysql/mysql.sock \
> --port=13297 --host=silo.cs.indiana.edu -u dgerman -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 46 to server version: 5.0.22-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use demoOne
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> create table dgerman_users_draft (
->  username varchar(32) not null primary key,
->  password varchar(20) not null,
->  lastName varchar(20) not null,
->  firstName varchar(20) not null
-> );
Query OK, 0 rows affected (0.02 sec)

mysql> create table dgerman_transactions_draft (
->  username varchar(32) not null,
->  timestamp timestamp,
->  amount double not null,
->  primary key (username, timestamp)
-> );
Query OK, 0 rows affected (0.03 sec)

mysql> insert into dgerman_users_draft values
->  ('lbird@pacers.com', 'dribl', 'Bird', 'Larry'),

```

```

-> ('rmiller@pacers.com', 'clutch', 'Miller', 'Reggie');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

```

```

mysql> select * from dgerman_users_draft;
+-----+-----+-----+-----+
| username          | password | lastName | firstName |
+-----+-----+-----+-----+
| lbird@pacers.com  | dribl    | Bird     | Larry     |
| rmiller@pacers.com | clutch   | Miller   | Reggie    |
+-----+-----+-----+-----+
2 rows in set (0.01 sec)

```

```

mysql> show tables;
+-----+
| Tables_in_demoOne |
+-----+
| dgerman_accumulator |
| dgerman_guessTheNumber |
| dgerman_transactions_draft |
| dgerman_users_draft |
+-----+
4 rows in set (0.00 sec)

```

```
mysql>
```

13.4 Data Access

Let's test data access:

```

<?
$result = mysql_connect('silو.cs.indiana.edu:13297', 'dgerman', 'sp00n');

if ($result) {
  if (mysql_select_db("demoOne")) {
    echo "I can select the database. <p>";
    $query = "select * from dgerman_users_draft";
    $result = mysql_query($query);
    if (! $result) echo "I don't see anything in here. <p>";
    $num_cats = mysql_num_rows($result);
    echo "There are " . $num_cats . " records I can see. <p>";

    while ($row = mysql_fetch_row($result)) {
      echo "(" . $row[0] . ", " . $row[1] . ") <p> ";
    }

  } else {
    echo "I can connect but I can't select the database. <p>";
  }
}

```

```

    } else {
        echo "I cannot connect. <p>";
    }

?>

```

Running this should extract the two records with no problem.

13.5 Transitions

Once we have taken care of that we can move to stage two:

```

<? session_start();
    if ($source == "login") {
        do_login_request();
    } elseif ($source == "mail") {
        do_mail_request();
    } elseif ($source == "chooser") {
        do_chooser_requesr();
    } elseif ($source == "create") {
        do_create_request();
    } elseif ($source == "take") {
        do_take_request();
    } elseif ($source == "show") {
        do_show_request();
    } else {
        _login();
    }

    function _login() { ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> Username <td><input type="text" name="uname">
        <tr><td> Password <td><input type="password" name="pword">
    </table> <p>
    <input type="hidden" name="source" value="login">
    <input type="checkbox" name="create" value="true"> Check here
                                if you want to create a new account.
    <p> When done please press <input type="submit" value="Proceed">
</form>
<? }

function do_login_request() {
    // global $uname, $pword, $create;
    if ($uname != "" && $pword != "" && (! $create)) {
        $result = mysql_connect('sil0.cs.indiana.edu:13297', 'dgerman', 'sp00n');
        if ($result) {
            if (mysql_select_db("demo0ne")) {
                echo "I can select the database. <p>";
                $query = "select username, password from dgerman_users_draft " .

```

```

        "where username = '$uname' and password = '$pword'";
$result = @mysql_query($query);
if ($row = mysql_fetch_row($result)) {
    echo "(" . $row[0] . ", " . $row[1] . ")";
    session_register("username");
    $username = $row[0];
    _chooser();
} else {
    $query = "select username from dgerman_users_draft " .
            "where username = '$uname'";
    $result = @mysql_query($query);
    if ($row = mysql_fetch_row($result)) {
        _mail();
    } else {
        _create();
    }
}
} else {
    echo "I can connect but I can't select the database. <p>";
}
} else {
    echo "I cannot connect. <p>";
}
} else {
    _create();
}
}

function do_mail_request() { }
function do_chooser_request() { }
function do_create_request() { }
function do_take_request() { }
function do_show_request() { }

function _mail() { ?>
I will mail you a password?
<? }

function _chooser() { ?>
Change profile or pledge some more?
<? }

function _create() { ?>
Create a new account.
<? }

function _take() {

}

```

```
function _show() {
}
?>
```

If you don't uncomment the one line you won't see this work. Why?

13.6 PHP Functions

The reason is: variables in functions are always local in PHP functions, unless specified otherwise. Fortunately we specify `global` and add a bit of functionality too, and we get:

```
<? session_start();
    if ($source == "login") {
        do_login_request();
    } elseif ($source == "mail") {
        do_mail_request();
    } elseif ($source == "chooser") {
        do_chooser_request();
    } elseif ($source == "create") {
        do_create_request();
    } elseif ($source == "take") {
        do_take_request();
    } elseif ($source == "show") {
        do_show_request();
    } else {
        _login();
    }

    function _login() { ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> Username <td><input type="text" name="uname">
        <tr><td> Password <td><input type="password" name="pword">
    </table> <p>
    <input type="hidden" name="source" value="login">
    <input type="checkbox" name="create" value="true"> Check here
                                if you want to create a new account.
    <p> When done please press <input type="submit" value="Proceed">
</form>
<? }

function do_login_request() {
    global $uname, $pword, $create;
    if ($uname != "" && $pword != "" && (! $create)) {
        $result = mysql_connect('silo.cs.indiana.edu:13297', 'dgerman', 'sp00n');
        if ($result) {
            if (mysql_select_db("demoOne")) {
                echo "I can select the database. <p>";
            }
        }
    }
}
```

```

$query = "select username, password from dgerman_users_draft " .
        "where username = '$uname' and password = '$pword'";
$result = @mysql_query($query);
if ($row = mysql_fetch_row($result)) {
    echo "(" . $row[0] . ", " . $row[1] . ")";
    session_register("username");
    $username = $row[0];
    _chooser();
} else {
    $query = "select username from dgerman_users_draft " .
            "where username = '$uname'";
    $result = @mysql_query($query);
    if ($row = mysql_fetch_row($result)) {
        _mail();
    } else {
        _create();
    }
}
} else {
    echo "I can connect but I can't select the database. <p>";
}
} else {
    echo "I cannot connect. <p>";
}
} else {
    _create();
}
}

function do_mail_request() { }
function do_chooser_request() { }
function do_create_request() { }
function do_take_request() { }
function do_show_request() { }

function _mail() { ?>
I will mail you a password?
<? }

function _chooser() { ?>
<p>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> <input type="radio" name="choose" value="update"> <td> Update profile
        <tr><td> <input type="radio" name="choose" value="pledge"> <td> Pledge new amount
    </table> <p>
    <input type="hidden" name="source" value="chooser">
    <p> When done please press <input type="submit" value="Proceed">
</form>

```

```

<? }

    function _create() { ?>
Create a new account.
<? }

    function _take() {

    }

    function _show() {

    }
?>

```

You should be checking these listings for differences between a version and another.

13.7 Minute Papers and Exercises

What changes do you see in this code compared with the previous one?

```

<? session_start();
    if ($source == "login") {
        do_login_request();
    } elseif ($source == "mail") {
        do_mail_request();
    } elseif ($source == "chooser") {
        do_chooser_request();
    } elseif ($source == "create") {
        do_create_request();
    } elseif ($source == "take") {
        do_take_request();
    } elseif ($source == "show") {
        do_show_request();
    } else {
        _login(true);
    }

    function _login($val) { ?>
<form method="GET" action="<?=$PHP_SELF?>">
<table>
    <tr><td> Username <td><input type="text" name="uname">
    <tr><td> Password <td><input type="password" name="pword">
</table> <p>
<input type="hidden" name="source" value="login"> <?
    if ($val) { ?>
<input type="checkbox" name="create" value="true"> Check here
                                if you want to create a new account.

    <? } ?>
<p> When done please press <input type="submit" value="Proceed">

```

```
</form>
<? }
```

```
function do_login_request() {
    global $uname, $pword, $create;
    if ($uname != "" && $pword != "" && (! $create)) {
        $result = mysql_connect('silo.cs.indiana.edu:13297', 'dgerman', 'sp00n');
        if ($result) {
            if (mysql_select_db("demoOne")) {
                echo "I can select the database. <p>";
                $query = "select username, password from dgerman_users_draft " .
                    "where username = '$uname' and password = '$pword'";
                $result = @mysql_query($query);
                if ($row = mysql_fetch_row($result)) {
                    echo "(" . $row[0] . ", " . $row[1] . ")";
                    session_register("username");
                    $username = $row[0];
                    _chooser();
                } else {
                    $query = "select username from dgerman_users_draft " .
                        "where username = '$uname'";
                    $result = @mysql_query($query);
                    if ($row = mysql_fetch_row($result)) {
                        _mail();
                    } else {
                        _create();
                    }
                }
            } else {
                echo "I can connect but I can't select the database. <p>";
            }
        } else {
            echo "I cannot connect. <p>";
        }
    } else {
        _create();
    }
}

function do_mail_request() {
    echo "New password (_____) has been mailed.<p>";
    _login(false);
}

function do_chooser_request() {
    global $choose;
    if ($choose == "update") {
        _show();
    } elseif ($choose == "pledge") {
        _take();
    }
}
```

```

    } else {
        echo "You need to select at least one radio button ($choose).";
    }
}
function do_create_request() { }
function do_take_request() { }
function do_show_request() { }

function _mail() { ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <input type="checkbox"
        name="create"
        value="true"> I have mistyped the password. Please send me a new one.
    <input type="hidden" name="source" value="mail">
    <p> When done please press <input type="submit" value="Proceed">
</form>
<? }

function _chooser() { ?>
<p>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> <input type="radio" name="choose" value="update">
            <td> Update profile
        <tr><td> <input type="radio" name="choose" value="pledge">
            <td> Pledge new amount
    </table> <p>
    <input type="hidden" name="source" value="chooser">
    <p> When done please press <input type="submit" value="Proceed">
</form>

<? }

function _create() { ?>
Create a new account.
<? }

function _take() { ?>
Ready to take your pledge.
<?
    }

function _show() {?>
This is to show the profile.
<? }
?>

```

If you've been true to the requirement (determine the changes from the previous stage) you have just warmed up for the rest of the chapter.

13.8 Stage Five

From here on it will be progressively harder to identify the individual stages. Take a pencil and as you look at this code, for each of the remaining three stages, clearly identify what that particular stage brings new to, or changes in the previous stage. There is a fair amount of repetition since I truly wanted to present the entire stage, complete with changes without just focusing on the changes and letting you do the hard part of assembling the pieces (which sometimes is very prone to error).

You can still take the code apart and write the project in the most intuitive and convenient way for you.

```
<? session_start();
    if ($source == "login") {
        do_login_request();
    } elseif ($source == "mail") {
        do_mail_request();
    } elseif ($source == "chooser") {
        do_chooser_request();
    } elseif ($source == "create") {
        do_create_request();
    } elseif ($source == "take") {
        do_take_request();
    } elseif ($source == "show") {
        do_show_request();
    } else {
        _login(true);
    }

    function _login($val) { ?>
<form method="GET" action="<?=$PHP_SELF?>"
    <table>
        <tr><td> Username <td><input type="text" name="uname">
        <tr><td> Password <td><input type="password" name="pword">
    </table> <p>
    <input type="hidden" name="source" value="login"> <?
        if ($val) { ?>
    <input type="checkbox" name="create" value="true"> Check here
                                if you want to create a new account.

        <? } ?>
    <p> When done please press <input type="submit" value="Proceed">
</form>
<? }

function do_login_request() {
    global $uname, $pword, $create, $username;
    if ($uname != "" && $pword != "" && (! $create)) {
        $result = mysql_connect('silo.cs.indiana.edu:13297', 'dgerman', 'sp00n');
        if ($result) {
            if (mysql_select_db("demoOne")) {
                echo "I can select the database. <p>";
            }
        }
    }
}
```

```

$query = "select username, password from dgerman_users_draft " .
        "where username = '$uname' and password = '$pword'";
$result = @mysql_query($query);
if ($row = mysql_fetch_row($result)) {
    echo "(" . $row[0] . ", " . $row[1] . ")";
    session_register("username");
    $username = $row[0];
    _chooser();
} else {
    $query = "select username from dgerman_users_draft " .
            "where username = '$uname'";
    $result = @mysql_query($query);
    if ($row = mysql_fetch_row($result)) {
        _mail();
    } else {
        _create();
    }
}
} else {
    echo "I can connect but I can't select the database. <p>";
}
} else {
    echo "I cannot connect. <p>";
}
} else {
    _create();
}
}

function do_mail_request() {
    echo "New password (_____) has been mailed.<p>";
    _login(false);
}

function do_chooser_request() {
    global $choose;
    if ($choose == "update") {
        _show();
    } elseif ($choose == "pledge") {
        _take();
    } elseif ($choose == "logout") {
        session_destroy();
        echo "OK, now close your browser.";
    } else {
        echo "You need to select at least one radio button ($choose).";
    }
}

function do_create_request() { }
function do_take_request() {
    echo "...just a note from do_take_request(<hr>";
}

```

```

        _chooser();
    }
    function do_show_request() {
        // insert values in the table
        _chooser();
    }

    function _mail() { ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <input type="checkbox"
        name="create"
        value="true"> I have mistyped the password. Please send me a new one.
    <input type="hidden" name="source" value="mail">
    <p> When done please press <input type="submit" value="Proceed">
</form>
<? }

    function _chooser() { ?>
<p>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> <input type="radio" name="choose" value="update">
            <td> Update profile
        <tr><td> <input type="radio" name="choose" value="pledge">
            <td> Pledge new amount
        <tr><td> <input type="radio" name="choose" value="logout">
            <td> Log out (and close your browser window).
    </table> <p>
    <input type="hidden" name="source" value="chooser">
    <p> When done please press <input type="submit" value="Proceed">
</form>

<? }

    function _create() { ?>
Create a new account.
<? }

    function _take() { ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> Amount <td><input type="text" name="first">
    </table> <p>
    <input type="hidden" name="source" value="take">
    <p> When done please press <input type="submit" value="Proceed">
</form>
<?
    }

    function _show() {

```

```

    global $username;
?>
<form method="GET" action="<?=$PHP_SELF?>">
  <table>
    <tr><td> Username <td><?=$username?>
    <tr><td> Password <td><input type="password" name="pword">
    <tr><td> Confirmation:
      <td><input type="password" name="pword_"> (retype password)
    <tr><td> Last Name <td><input type="text" name="last" value="">
    <tr><td> First Name <td><input type="text" name="first" value="">
  </table> <p>
  <input type="hidden" name="source" value="show">
  <p> When done please press <input type="submit" value="Proceed">
</form>
<? }

?>

```

13.9 Stage Six

The section that follows this one is also the last one; so we're getting closer.

```

<? session_start();
  if (session_is_registered("age")) {

  } else {
    session_register("age");
    $age = 0;
  }

  if ($age == $ageCopy) {
    $age += 1;
    if ($source == "login") {
      do_login_request();
    } elseif ($source == "mail") {
      do_mail_request();
    } elseif ($source == "chooser") {
      do_chooser_request();
    } elseif ($source == "create") {
      do_create_request();
    } elseif ($source == "take") {
      do_take_request();
    } elseif ($source == "show") {
      do_show_request();
    } else {
      _login(true);
    }
  } else {
    echo "Please don't reload, it makes no difference.";
  }

```

```

    if ($source == "login") {
        _login(true);
    } elseif ($source == "mail") {
        _mail();
    } elseif ($source == "chooser") {
        _chooser();
    } elseif ($source == "create") {
        _create();
    } elseif ($source == "take") {
        _take();
    } elseif ($source == "show") {
        _show("", "", "");
    } else {
        _login(true);
    }
}

function _login($val) {
    global $age; ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> Username <td><input type="text" name="uname">
        <tr><td> Password <td><input type="password" name="pword">
    </table> <p>
    <input type="hidden" name="source" value="login"> <?
    if ($val) { ?>
    <input type="checkbox"
        name="create"
        value="true"> Check here if you want to create a new account.
    <? } ?>
    <p> When done please press <input type="submit" value="Proceed">
    <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>
<? }

function do_login_request() {
    global $uname, $pword, $create, $username;
    if ($uname != "" && $pword != "" && (! $create)) {
        $result = mysql_connect('silo.cs.indiana.edu:13297', 'dgerman', 'sp00n');
        if ($result) {
            if (mysql_select_db("demoOne")) {
                echo "I can select the database. <p>";
                $query = "select username, password from dgerman_users_draft " .
                    "where username = '$uname' and password = '$pword'";
                $result = @mysql_query($query);
                if ($row = mysql_fetch_row($result)) {
                    echo "(" . $row[0] . ", " . $row[1] . ")";
                    session_register("username");
                    $username = $row[0];
                    _chooser();
                }
            }
        }
    }
}

```

```

    } else {
        $query = "select username from dgerman_users_draft " .
            "where username = '$uname'";
        $result = @mysql_query($query);
        if ($row = mysql_fetch_row($result)) {
            _mail();
        } else {
            _create();
        }
    }
} else {
    echo "I can connect but I can't select the database. <p>";
}
} else {
    echo "I cannot connect. <p>";
}
} else {
    _create();
}
}

function do_mail_request() {
    echo "New password (_____) has been mailed.<p>";
    _login(false);
}

function do_chooser_request() {
    global $choose, $username;
    if ($choose == "update") {
        $query = "select username, lastName, firstName from dgerman_users_draft " .
            "where username = '$username'";
        $result = mysql_connect('sil0.cs.indiana.edu:13297', 'dgerman', 'sp00n');
        if ($result) {
            if (mysql_select_db("demoOne")) {
                $result = @mysql_query($query);
                if ($row = mysql_fetch_row($result)) {
                    _show($row[0], $row[1], $row[2]);
                } else {
                    echo "Something went wrong. But what? <p> ";
                }
            } else { echo "Can't select database."; }
        } else { echo "I can't connect."; }
    } elseif ($choose == "pledge") {
        _take();
    } elseif ($choose == "logout") {
        session_destroy();
        echo "OK, now close your browser.";
    } else {
        echo "You need to select at least one radio button ($choose).";
    }
}

```

```

    }
    function do_create_request() { }
    function do_take_request() {
        echo "...just a note from do_take_request()<hr>";
        _chooser();
    }
    function do_show_request() {
        // insert values in the table
        _chooser();
    }

    function _mail() {
        global $age; ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <input type="checkbox"
        name="create"
        value="true"> I have mistyped the password. Please send me a new one.
    <input type="hidden" name="source" value="mail">
    <p> When done please press <input type="submit" value="Proceed">
    <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>
<? }

    function _chooser() {
        global $age; ?>
<p>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> <input type="radio" name="choose" value="update">
            <td> Update profile
        <tr><td> <input type="radio" name="choose" value="pledge">
            <td> Pledge new amount
        <tr><td> <input type="radio" name="choose" value="logout">
            <td> Log out (and close your browser window).
    </table> <p>
    <input type="hidden" name="source" value="chooser">
    <p> When done please press <input type="submit" value="Proceed">
    <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>

<? }

    function _create() {
        global $age; ?>
    Create a new account.
<? }

    function _take() {
        global $age; ?>
<form method="GET" action="<?=$PHP_SELF?>">

```

```

<table>
  <tr><td> Amount <td><input type="text" name="first">
</table> <p>
<input type="hidden" name="source" value="take">
<p> When done please press <input type="submit" value="Proceed">
<input type="hidden" name="ageCopy" value="<?=$age?>">
</form>
<?
  }

  function _show($uname, $last, $first) {
    global $username, $age;
  ?>
<form method="GET" action="<?=$PHP_SELF?>">
  <table>
    <tr><td> Username <td><?=$username?>
    <tr><td> Password <td><input type="password" name="pword">
    <tr><td> Confirmation:
      <td><input type="password" name="pword_"> (retype password)
    <tr><td> Last Name <td><input type="text" name="last" value="<?=$last?>">
    <tr><td> First Name <td><input type="text" name="first" value="<?=$first?>">
  </table> <p>
  <input type="hidden" name="source" value="show">
  <p> When done please press <input type="submit" value="Proceed">
  <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>
<? }

  ?>

```

13.10 Stage Seven (Last Stage)

```

<? session_start();
  if (session_is_registered("age")) {

  } else {
    session_register("age");
    $age = 0;
  }

  if ($source == "login") {
    do_login_request();
  } elseif ($source == "mail") {
    do_mail_request();
  } elseif ($source == "chooser") {
    do_chooser_request();
  } elseif ($source == "create") {
    do_create_request();
  } elseif ($source == "take") {

```

```

if ($age == $ageCopy) {
    $age += 1;
    do_take_request();
} else {
    $conn = mysql_connect('silo.cs.indiana.edu:13297', 'dgerman', 'sp00n');
    if ($conn) {
        show_profile($conn);
        show_pledges($conn);
        _chooser();
    }
}
} elseif ($source == "show") {
    do_show_request();
} else {
    _login(true);
}

function _login($val) {
    global $age; ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> Username <td><input type="text" name="uname">
        <tr><td> Password <td><input type="password" name="pword">
    </table> <p>
    <input type="hidden" name="source" value="login"> <?
    if ($val) { ?>
    <input type="checkbox"
        name="create"
        value="true"> Check here if you want to create a new account.
    <? } ?>
    <p> When done please press <input type="submit" value="Proceed">
    <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>
<? }

function do_login_request() {
    global $uname, $pword, $create, $username;
    if ($uname != "" && $pword != "" && (! $create)) {
        $conn = mysql_connect('silo.cs.indiana.edu:13297', 'dgerman', 'sp00n');
        if ($conn) {
            if (mysql_select_db("demoOne")) {
                // echo "I can select the database. <p>";
                $query = "select * from dgerman_users_draft " .
                    "where username = '$uname' and password = '$pword'";
                $result = @mysql_query($query);
                if ($row = mysql_fetch_row($result)) {
                    # echo "(" . $row[0] . ", " . $row[1] .
                    #     ", " . $row[2] . ", " . $row[3] . ")<br>";
                    session_register("username");
                    $username = $row[0];
                }
            }
        }
    }
}

```

```

        show_profile($conn);
        show_pledges($conn);
        _chooser();
    } else {
        $query = "select username from dgerman_users_draft where " .
            "username = '$username'";
        $result = @mysql_query($query);
        if ($row = mysql_fetch_row($result)) {
            _mail();
        } else {
            _create();
        }
    }
} else {
    echo "I can connect but I can't select the database. <p>";
}
} else {
    echo "I cannot connect. <p>";
}
} else {
    _create();
}
}

function show_pledges($conn) {
    global $username;

    if (mysql_select_db("demoOne")) {
        // echo "I can select the database. <p>";
        echo "Your pledges thus far: <blockquote>";
        $query = "select * from dgerman_transactions_draft where " .
            "username = '$username'";
        $result = @mysql_query($query);
        while ($row = mysql_fetch_row($result)) {
            echo "(" . $row[0] . ", " . $row[1] . ", " . $row[2] . ") <br>";
        }
        echo "</blockquote>";
    } else {
        echo "I can connect but I can't select the database. <p>";
    }
}

function show_profile() {
    global $username;

    if (mysql_select_db("demoOne")) {
        // echo "I can select the database. <p>";
        echo "Your current profile: <blockquote>";
        $query = "select * from dgerman_users_draft " .
            "where username = '$username'";
    }
}

```

```

$result = @mysql_query($query);
while ($row = mysql_fetch_row($result)) {
    echo "(" . $row[0] . ", xxxxxxxxx, " .
        $row[2] . ", " . $row[3] . ") <br>";
}
echo "</blockquote>";
} else {
    echo "I can connect but I can't select the database. <p>";
}
}

function do_mail_request() {
    echo "New password (_____) has been mailed.<p>";
    _login(false);
}

function do_chooser_request() {
    global $choose, $username;
    if ($choose == "update") {
        $query = "select username, lastName, firstName from dgerman_users_draft " .
            "where username = '$username'";
        $result = mysql_connect('silo.cs.indiana.edu:13297', 'dgerman', 'sp00n');
        if ($result) {
            if (mysql_select_db("demoOne")) {
                $result = @mysql_query($query);
                if ($row = mysql_fetch_row($result)) {
                    _show($row[0], $row[1], $row[2]);
                } else {
                    echo "Something went wrong. But what? <p> ";
                }
            } else { echo "Can't select database."; }
        } else { echo "I can't connect."; }
    } elseif ($choose == "pledge") {
        _take();
    } elseif ($choose == "logout") {
        session_destroy();
        echo "OK, now close your browser.";
    } else {
        echo "You need to select at least one radio button ($choose).";
    }
}

function do_create_request() { }

function do_take_request() {
    global $username, $pledge;
    $conn = mysql_connect('silo.cs.indiana.edu:13297', 'dgerman', 'sp00n');
    if ($conn) {
        if (mysql_select_db("demoOne")) {

```

```

        // echo "I can select the database. <p>";
        $query = "insert into dgerman_transactions_draft " .
            "values ('$username', null, $pledge)";
        $result = @mysql_query($query);
        show_pledges($conn);
    } else {
        echo "I can connect but I can't select the database. <p>";
    }
} else {
    echo "I cannot connect. <p>";
}
_chooser();
}

function do_show_request() {
    // insert values in the table
    _chooser();
}

function _mail() {
    global $age; ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <input type="checkbox"
        name="create"
        value="true"> I have mistyped the password. Please send me a new one.
    <input type="hidden" name="source" value="mail">
    <p> When done please press <input type="submit" value="Proceed">
    <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>
<? }

function _chooser() {
    global $age; ?>
<p>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> <input type="radio" name="choose" value="update">
            <td> Update profile
        <tr><td> <input type="radio" name="choose" value="pledge">
            <td> Pledge new amount
        <tr><td> <input type="radio" name="choose" value="logout">
            <td> Log out (and close your browser window).
    </table> <p>
    <input type="hidden" name="source" value="chooser">
    <p> When done please press <input type="submit" value="Proceed">
    <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>

<? }

```

```

function _create() {
    global $age; ?>
    Create a new account.
<? }

function _take() {
    global $age; ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> Amount <td><input type="text" name="pledge">
    </table> <p>
    <input type="hidden" name="source" value="take">
    <p> When done please press <input type="submit" value="Proceed">
    <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>
<?
    }

function _show($uname, $last, $first) {
    global $username, $age;
    ?>
<form method="GET" action="<?=$PHP_SELF?>">
    <table>
        <tr><td> Username <td><?=$username?>
        <tr><td> Password <td><input type="password" name="pword">
        <tr><td> Confirmation:
            <td><input type="password" name="pword_"> (retype password)
        <tr><td> Last Name <td><input type="text" name="last" value="<?=$last?>">
        <tr><td> First Name <td><input type="text" name="first" value="<?=$first?>">
    </table> <p>
    <input type="hidden" name="source" value="show">
    <p> When done please press <input type="submit" value="Proceed">
    <input type="hidden" name="ageCopy" value="<?=$age?>">
</form>
<? }

    ?>

```

13.11 Conclusion

The strategy in general is simple: draw the diagram, including the transitions. For each screen come up with a label (a name). Store this in the screen in a hidden field called `source`. This way you will always know where you're coming from, who made the request. For each screen write a `do_screen_request` method, which determines where we're landing next: more than one transition is possible from a screen, and this method determines which one will eventually be chosen (this method also does real work. For each screen there should also be a `_screen` method, whose purpose is to display (or show) the target screen once a transition has been determined.

Chapter 14

Security Concerns in PHP and MySQL

14.1 PHP and Superglobals

PHP 4.2.0 saw a change in default settings which significantly improved the security of the language, but also made it more difficult to understand. In particular, the setting `register_globals` was changed from being on by default to off.

`register_globals` forces the PHP engine to registerize all global variables. This means that all global variables are not only available within their own scope, but also within the scope of functions and objects. The `register_globals` effect is better demonstrated through the annotated code snippet below:

```
$myVariable = true; // This is a global variable

function myFunction($myInput) { // This is a function
    if($myVariable) { // Because of register_globals,
                        // this variable is visible
        echo $myInput; // If register_globals is on,
                        // myVariable is true, and
                        // the input to the function
                        // displays the argument when called
    }
}
```

If `register_globals` is set to on, the above code creates a new function that prints whatever is passed to it. However, if it's set to off (which the latest version of PHP will do by default), the function below will not print anything. Why? Because if `register_globals` is off, the variable `$myVariable` is not visible within the scope of the function. There are several ways to solve this problem without turning `register_globals` on.

First, we can use the keyword `global`, which registerizes the variable following only within the given function:

```
$myVariable = true; // This is a global variable

function myFunction($myInput) { // This is a function

    global $myVariable; // The global keyword registerizes
                        // the variable for use only in this function

    if($myVariable) { // Because of the global keyword, this variable
        echo $myInput; // is visible. If global is used, myVariable is
                        // true, and the input to the function displays
    } // the argument when called
}
```

This solution is not preferable, mostly because registerizing a variable by its name doesn't guarantee its origin.

Even if `register_globals` is off, there are certain variables which are always accessible and registerized, called superglobals. One of these superglobals is the array `$GLOBALS`, which stores all of the global-scope variables under the keys of the same name. We can use the `$GLOBALS` superglobal in our previous code here:

```
$myVariable = true; // This is a global variable

function myFunction($myInput) { // This is a function
    if($GLOBALS['myVariable']) { // Because of the superglobal
                                // array, this variable is visible
        echo $myInput; // If the superglobal is used, myVariable
                        // is true, and the input to the function
    } // displays the argument when called
}
```

So, why are the superglobal arrays preferred to the `register_globals` method or global keyword? The arrays provide a mechanism for guaranteeing where our variables are coming from. More specifically, since a web program will supposedly be receiving input from the user (either in get or post requests), it's important to know whether those variables are uncontrolled data coming from the outside or globals we've defined ourselves. PHP has several other superglobal variables of note:

- `$_GET` - variables passed in through the URL query string
- `$_POST` - variables passed through the body of a post html request
- `$_SESSION` - variables stored in the client's session
(unlike the others, new variables can be added to this array)

- `$_SERVER` - server variables,
such as Apache CGI variables and PHP server variables

Using these variables guarantees that the variables come from their respective locations. If you registerize a global variable through the global keyword or `register_globals`, the global variable could be defined in any of these areas. For example, you could be counting on a variable defined as a server variable, the CGI variable `remote_addr`, which reports the IP address of the user. If the user passes the variable through a query string, in this way: `?remote_addr=127.0.0.1` registerizing the `remote_addr` variable through the global keyword or `register_globals` doesn't guarantee that the variable being used is the server variable and not the variable passed by the user. In essence, superglobals protect our code by ensuring that the variables being used cannot be from a dangerous and unprotected source, like internet users. The moral of the story? Whenever possible, use the superglobal arrays (listed above) to access variables not defined in your functions and classes.

14.2 MySQL Injection

Would we give an unknown user access to our MySQL account name and password? Since the user would be able to run such queries as

```
delete from users
```

giving him or her this access is a huge security risk. However, we've already given users of our programs this much access in the past through query concatenation. The code below shows our security risk:

```
$connect = mysql_connect($db_server, $db_user, $db_password);

if($connect){

    $query = "select password from users where username='" .

        $_POST['username'] . "'";

    if (mysql_select_db($database)) {

        $result = @mysql_query($query);

        $row = mysql_fetch_row($result);

        return $row[0];

    } else {

        echo "Bad Database...<br>";
        return false;
    }
}
```

```

    }

} else {
    echo "Can't connect...<br>";
    return false;
}

```

On the face of it, nothing seems wrong with this code.

However, imagine a malicious user entered the following as the username in a post request to this program:

```
'; delete from users where '' = '
```

Since we simply concatenated the strings together, our resulting query is:

```
select password from users where username=''; delete from users where '' = ''
```

This has changed our previously safe code into two queries. One retrieves all users where the username is the empty string, but the other deletes all of our user information! The username closed the quoted string and ended the query with a semicolon, which allows any following information to be parsed literally by MySQL.

Thus, our malicious user can enter any query he or she pleases, including deleting all of our users or inserting a special unauthorized account. Later versions of PHP automatically correct this issue by escaping single quotes (changing ' into \'). If this transformation had been applied above, the resulting dangerous query would not have been produced. If you're not using PHP, queries can be protected by making certain that single and double quotations are escaped. Escaping these characters prevents users from prematurely ending a variable string. If this tactic had been used on the previous input, the resulting query would have been:

```
select password from users where username='\'; delete from users where \'\' = \'\'
```

This is a query which searches for a password for a user with the username:

```
" "; delete from users where '' = '' "
```

Naturally, such an odd username will produce no results and the resulting query is completely benign.

In summation, to stay safe with MySQL queries, make sure you escape special characters in any concatenated user inputs.

Chapter 15

Javascript and DHTML

JavaScript is a lightweight interpreted programming language with rudimentary object-oriented capabilities, developed by Brendan Eich and his team at Netscape. General-purpose core of the language embedded in Navigator and other web browsers, embellished for web programming with the addition of objects that represent the web browser window and its contents. It allows programmatic control over content of web pages, browser, and HTML forms (through event handlers, pieces of JavaScript code that are executed when a particular event occurs).

Syntactically the core JavaScript language resembles C, C++, and Java. Untyped language, though, which means that variables do not need to have a type specified. Objects in JavaScript are more like Perl's associative arrays than they are like structures in C or objects in C++ or Java. Purely interpreted language. Has been with us since about 1994; has nothing in common with the Java programming language except perhaps that they were being developed at about the same time. JavaScript was called LiveWire before Java came about.

It is the addition of JavaScript that allows us to talk about dynamic HTML (DHTML) and the document object model (DOM), since it allows us to obtain programmatic access to the data structures created by the browser.

15.1 Programming the Browser

Here are two ways of adding PHP code to an HTML web page:

```
<html><head><title>PHP Tag Styles</title></head><body bgcolor=white>
```

```
It's <?=date("H:i, jS F")?> <p>
```

```
<? for ($i = 0; $i < 4; $i++) {  
    echo $i, ", ";  
}  
echo "... done! <hr>";
```

```

?>

<script language="php">

    for ($i = 0; $i < 4; $i++) {
        echo $i, ", ";
    }
    echo "... done! <hr>";

</script>

</body></html>

```

In both cases the result is the same: processing happens when the page is distributed, on the server side. Notice though that the tags are encoded differently. Here's a somewhat similar situation. What's different?

```

<html>
<body bgcolor=white>
<script language="javascript">
document.write("<h2>Table of Factorials</h2>\n");
for (i = 0, fact = 1; i < 10; i++, fact *= i) {
    document.write(i + "! = " + fact);
    document.write("\n<br>");
}
</script>
</body>
</html>

```

The difference is that the processing happens entirely on the client side, right before the page is loaded and rendered (or, rather, while the page is being loaded and rendered). But JavaScript is hardly ever being used that way. So here are some more examples to give us an idea of how it is being used. Basic, truly basic event handling:

```

<html>
<body bgcolor=white>
<form>
<input type="button"
    value="click here!"
    onClick="alert('You clicked the button!')">
</form>
</body>
</html>

```

Here's an example of a function being used to calculate the date and then display it; note that the values that result are from the client-side's environment:

```

<html>
<head>

```

```

<title> Today's Date </title>
  <script language="JavaScript">
    // Define a function for use later on
    function print_todays_date() {
      var d = new Date(); // today's date and time
      document.write(d.toLocaleString());
    }
  </script>
</head>
<body bgcolor=white>
<hr>The date and time are: <br> <b>
  <script language="JavaScript">
    // Now call the function we defined above
    print_todays_date();
  </script>
</b> <hr>
</body>
</html>

```

Like the HTML of Chapter 1, one doesn't need a network connection or a web server to learn JavaScript: a browser is enough.

```

<SCRIPT>
var _console = null;

function debug(msg)
{
  // Open a window the first time we are called, or after an existing
  // console window has been closed.
  if ((_console == null) || (_console.closed)) {
    _console = window.open("", "console", "width=600,height=300,resizable");
    // Open a document in the window to display plain text.
    _console.document.open("text/plain");
  }

  _console.document.writeln(msg);
}
</SCRIPT>

<!-- Here's an example of using this script. -->
<SCRIPT>var n = 0;</SCRIPT>
<FORM>
<INPUT TYPE="button" VALUE="Push Me"
  onClick="debug('You have pushed me:\t' + ++n + ' times.');">
</FORM>

```

The previous example was a simple debugging environment; in the next example special effects with images are being demonstrated. The only reason for which we need a network connection here is to get the images from some place. We would not need it if the images were on the desktop and the HTML would point to them. Note that the example below uses timers.

```

<html><head><title>Animation Example</title></head>
 <script>
  images = new Array(10);
  for (var i = 0; i < 10; i++) {
    images[i] = new Image();
    images[i].src =
      "http://www.cs.indiana.edu/classes/a202-dger/lectures/last/T" +
      (i + 1) + ".gif";
  }
  function animate() {
    document.animation.src = images[frame].src;
    frame = (frame + 1) % 10;
    timeout_id = setTimeout("animate()", 250);
  }
  var frame = 0;
  var timeout_id = null;
</script>
<body bgcolor=white>
<form>
  <input type=button value="Start"
        onClick="if (timeout_id == null) animate()">
  <input type=button value="Stop"
        onClick="if (timeout_id) clearTimeout(timeout_id); timeout_id=null;">
</form>
</body></html>

```

Here's another example of JavaScript use in a form:

```

<html><head><script>
  function setA (menu) {
    document.question.A.selectedIndex = menu.selectedIndex;
  }
  function setB (menu) {
    document.question.B.selectedIndex = menu.selectedIndex;
  }
</script><title>Test</title></head>
<body bgcolor=white>
<em>The two select buttons communicate with each other using JavaScript.
Thus they implement the radio button formatted question at the bottom of
this document in a more distributed manner.
</em>
<form name="question">
Early <select name="A" onChange="setB(this)">
<option> discovery
<option> development
<option> detection
<option> treatment
<option> incidence
</select>

```

```

of hearing loss is <select name="B" onChange="setA(this)">
<option> indicated
<option> prevented
<option> complicated
<option> facilitated
<option> corrected
</select> by the fact that
the other senses are able to compensate for moderate amounts of loss,
so that people frequently do not know that their hearing is imperfect.
<hr>
Early _____ of hearing loss is _____ by the fact that the other
senses are able to compensate for moderate amounts of loss, so that
people frequently do not know that their hearing is imperfect. <p>
A. <input type="radio" name="C"> discovery .. indicated <br>
B. <input type="radio" name="C"> development .. prevented <br>
C. <input type="radio" name="C"> detection .. complicated <br>

D. <input type="radio" name="C"> treatment .. facilitated <br>
E. <input type="radio" name="C"> incidence .. corrected <br>
<hr> <em>Note: the correct answer is (C)</em>.
</form></body></html>

```

The example above is meant to just illustrate the language's capability of extending the HTML's interactive (dynamic) features, much in the same way we demonstrated it with the first few CGI scripts. But although this won't be the main use of JavaScript in this chapter it does show how we can read data from various form elements.

One more example (also known as "The Big Wave"):

```

<html><head><title>The Big Wave</title></head>

  <!-- Here are all ten of the images. This is how they appear
        before the JavaScript function animate() is called on them. They have
        names so that we can refer to them later... -->









```

```





```

```

<!-- Now for the JavaScript, contained within the (script) tags -->

```

```

<script>
images = new Array(10); // New array; contains objects of type images
var k=0;                // Variable k is initialized to zero

for (var i = 0; i < 10; i++) { // A "for" loop...
  images[i] = new Image();    // Initializes all objects of type images

  images[i].src=

    "http://www.cs.indiana.edu/classes/a202-dger/lectures/last/T" +
    (i + 1) + ".gif";

} // This associates each of the images listed in (img src) tags with a
  // specific index in the images array. Each of the actual .gif images
  // are associated with one of the objects in the images array.

// This function is called on all the objects in the images array, when
// the user clicks the mouse on the "Start" button. When the function is
// first called, the source of object one, located in document, receives
// the value of images[(0+k)%10].src, that is: the object located at
// index [0] in the images array. Similar things happen with all the
// other images. So the first time animate() runs, the images do not
// change. However, at the end of the function definition, k is
// increased by one (mod 10), and animate() runs again (unless the user
// stops it by clicking the "Stop" button). This time, k is equal to
// one. So now, each image receives the value of the next one up. On the
// third cycle, k is equal to two, so each image object receives the
// value of the image that was two higher than it, and so on. The use of
// %10 when k is added to the index of an object of the images array is
// very important. If it were not there, the indexes of images would
// increase indefinitely (at least in principle) but as array images
// has 10 locations 0-9 using index 10 in it would result in a JavaScript
// error. %10 ensures that after the index of images reaches [9], the
// next index called would be zero. Thus, the cycle continues. The
// setTimeout is very important as well. After each of the objects is
// assigned a value in function animate(), there is 2.5 second delay
// before the images all "move" once more. Otherwise, animate() would
// continually run, exchanging images extremely fast without any pauses.

function animate() {

```

```

    document.one.src    = images[(0+k)%10].src;
    document.two.src    = images[(1+k)%10].src;
    document.three.src  = images[(2+k)%10].src;
    document.four.src   = images[(3+k)%10].src;
    document.five.src   = images[(4+k)%10].src;
    document.six.src    = images[(5+k)%10].src;
    document.seven.src  = images[(6+k)%10].src;
    document.eight.src  = images[(7+k)%10].src;
    document.nine.src   = images[(8+k)%10].src;
    document.ten.src    = images[(9+k)%10].src;

    timeout_id = setTimeout("animate()", 250);

    k=(k+1)%10;
}

var timeout_id = null;    // timeout_id set to null

</script>                <!-- End of script -->
<body bgcolor=white>     <!-- Some standard HTML.. -->

<form>                   <!-- Output is a form: two buttons -->

<input type=button
    value="Start"
    onClick="if (timeout_id == null) animate()">

<!-- If user clicks on Start button, and timeout_id is set to null,
    function animate() will be called -->

<input type=button
    value="Stop"
    onClick="
if (timeout_id) clearTimeout(timeout_id); timeout_id=null;">

<!-- If user clicks on Stop button, timeout_id is set to null -->

</form><hr></font></body></html>

```

In what follows we are looking for an extension to this mechanism. But first, let's briefly review the object-oriented features of JavaScript; it will be a review by example, in four steps.

15.2 Object-Oriented JavaScript

Objects in JavaScript are simply hashtables¹:

¹For this and the rest of examples presented type the code in and run it, then try to predict what it will produce if you load it in a browser. Afterwards, try to explain what you see.

```
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
<script language="Javascript">

    var account = new Object();

    account.balance = 20;

    function getBalance() {
        return this.balance;
    }

    account.getBalance = getBalance;

    function deposit(amount) {
        this.balance += amount;
    }

    account.deposit = deposit;

    document.write(account.getBalance() + "<br>");

    account.deposit(30);

    document.write(account.getBalance());
</script>
</body>
</html>
```

There is a nice project that the course website presents, inspired by a Lincoln Stein example, in the context of which these object-oriented features are particularly useful. But it is worthwhile reviewing them even outside of the context of that project.

We also see that functions are first-class in JavaScript. Let's take a look at constructor functions:

```
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
<script language="Javascript">

    function Account(initialBalance) {

        this.balance = initialBalance;

        this.deposit =
            function deposit(amount) {
                this.balance += amount;
            }
    }
</script>
</body>
</html>
```

```

        this.getBalance =
            function getBalance() {
                return this.balance;
            }
    }

    var account = new Account(20);

    account.deposit(300);

    document.write(account.balance + "<br>");

    document.write(account.getBalance()); // [1]

</script>
</body>
</html>

```

What do you think would happen if we changed the line marked // [1] to

```
document.write(account.getBalance);
```

Note that the parens are missing. Well, functions are first-class (that is, they can also be data). Here are a few more examples:

```

<html>
<head><title>Testing</title></head>
<body bgcolor=white>
  <script language="Javascript">

    function square(x) { return x * x; }

    a = square(4);

    b = square;

    c = b(5);

    document.write(a + "<hr>" + b + "<hr>" + c);

  </script>
</body>
</html>

```

Let's look now at objects as associative arrays (or hashes in Perl).

```
object.property
```

and

```
object["property"]
```

are one and the same thing.

Here's where the second approach would make a difference:

```
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
  <script language="Javascript">

    var one = new Object();

    for (i = 0; i < 4; i++)
      one["prop" + i] = "This is property " + i;

    document.write(one.prop0 + "<br>");
    document.write(one.prop1 + "<br>");
    document.write(one.prop2 + "<br>");
    document.write(one.prop3 + "<br>");

  </script>
</body>
</html>
```

How do we get the properties out in a more uniform manner? You need to know about the `for/in` construct.

```
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
  <script language="Javascript">

    var one = new Object();

    for (i = 0; i < 4; i++)
      one["prop" + i] = "This is property " + i;

    // document.write(one.prop0 + "<br>");
    // document.write(one.prop1 + "<br>");
    // document.write(one.prop2 + "<br>");
    // document.write(one.prop3 + "<br>");

    for (prop in one) {
      document.write(prop + ": " + one[prop] + "<br>");
    }

  </script>
</body>
</html>
```

Now we can put together these last two examples in a more complex one.

```

<html>
  <head><title>Testing</title></head>
  <body bgcolor=white>
    <script language="Javascript">

      // first define some simple functions

      function add(x, y) { return x + y; }
      function subtract(x, y) { return x - y; }
      function multiply(x, y) { return x * y; }
      function divide(x, y) { return x / y; }

      // this function takes three arguments
      //     a function (operator)
      //     and two operands (operand1, operand2)

      function operate(operator, operand1, operand2) {
        return operator(operand1, operand2);
      }

      var result = operate(add, 3, 4);

      document.write(result + "<br>");

      document.write(operate(add, operate(subtract, 3, 4), 5) + "<br>");

    </script>
  </body>
</html>

```

We used to present a slightly more involved example, which is a bit like this:

```

<html>
  <head><title>Testing</title></head>
  <body bgcolor=white>
    <script language="Javascript">

      var value = 3;

      function add(val, arg) { return val + arg; }

      function subtract(val, arg) { return val - arg; }

      function change(n) {
        if (n % 2 == 0) {
          eval("value = add(value, " + n + ")");
        } else {
          eval("value = subtract(value, " + n + ")");
        }
      }
    }

```

```

        change(2);

        document.write(value + "<br>");

        change(3);

        document.write(value + "<br>");

    </script>
</body>
</html>

```

And that concludes our very brief review.

15.3 and .innerHTML

Here's the first of the two examples we need to discuss:

```

<html>
  <head>
    <title>
      The Lindley Portfolio
    </title>
    <script language="javascript">

      function show(what) {

        document.getElementById("titleOne" ).innerHTML =

          "<a href=\"javascript:show('one' )\">Lindley One</a>";

        document.getElementById("titleEight").innerHTML =

          "<a href=\"javascript:show('eight')\">Lindley Eight</a>";

        document.getElementById("titleSeven").innerHTML =

          "<a href=\"javascript:show('seven')\">Lindley Seven</a>";

        document.getElementById("titleNine" ).innerHTML =

          "<a href=\"javascript:show('nine' )\">Lindley Nine</a>";

        if (what == 'one') {

          document.getElementById("titleOne" ).innerHTML = "Lindley One";

          url = "http://www.cs.indiana.edu/dept/img/lh01.gif";

```

```

    document.getElementById("pic"           ).innerHTML =
        "<img src=\"\" + url + \"\"/ >";

    document.getElementById("caption"      ).innerHTML =
        "Red October Lindley";

    document.getElementById("url"          ).innerHTML = url;

} else if (what == 'eight') {
    document.getElementById("titleEight").innerHTML = "Lindley Eight";

    url = "http://www.cs.indiana.edu/dept/img/lh08.gif";

    document.getElementById("pic"         ).innerHTML = "<img src=\"\" + url + \"\"/ >";
    document.getElementById("caption"     ).innerHTML = "Lindley Twilight Picture";
    document.getElementById("url"         ).innerHTML = url;

} else if (what == 'seven') {
    document.getElementById("titleSeven").innerHTML = "Lindley Seven";

    url = "http://www.cs.indiana.edu/dept/img/lh07.gif";

    document.getElementById("pic"         ).innerHTML = "<img src=\"\" + url + \"\"/ >";
    document.getElementById("caption"     ).innerHTML = "Lindley from Oz";
    document.getElementById("url"         ).innerHTML = url;

} else if (what == 'nine') {
    document.getElementById("titleNine" ).innerHTML = "Lindley Nine";

    url = "http://www.cs.indiana.edu/dept/img/lh09.gif";

    document.getElementById("pic"         ).innerHTML = "<img src=\"\" + url + \"\"/ >";
    document.getElementById("caption"     ).innerHTML = "SouthWest of Lindley";
    document.getElementById("url"         ).innerHTML = url;

} else {
    alert("Sorry, I don't quite understand...");
}
}
</script>
</head>
<body bgcolor=white>
<table cellpadding=6>
<tr><td align=center>
    <span id="titleOne" >
        <a href="javascript:show('one')">Lindley One</a>
    </span>
</td>
<td align=center>

```

```

        <span id="titleEight">
          <a href="javascript:show('eight')">Lindley Eight</a>
        </span>
      </td>
      <td align=center>
        <span id="titleSeven">
          <a href="javascript:show('seven')">Lindley Seven</a>
        </span>
      </td>
      <td align=center>
        <span id="titleNine" >
          <a href="javascript:show('nine')">Lindley Nine</a>
        </span>
      </td>
    <tr><td colspan=4 align=center>
      <span id="pic">
        
      </span>
    </td>
  </tr>
  <tr><td colspan=4 align=center>
    <span id="caption">
      Picture of Dilbert working like crazy in Lindley
    </span>
  </td>
</tr>
<tr><td colspan=4 align=center>
  <span id="url">
    <code>http://www.cs.indiana.edu/classes/a202-dger/sum99/a202.gif</code>
  </span>
</td>
</tr>
</table>
</body>
</html>

```

The second example would be the calculator; do you think you can implement it yourself? Furthermore, does the code above work in Internet Explorer as well as it does in Firefox? Why or why not? Here's the code for the calculator:

```

<html>
  <head>
    <title>
      The Calculator
    </title>
    <script language="javascript">
      acc = 0;

      function calculate() {
        arg = document.forms[0].arg.value;

```

```

ind = document.forms[0].fun.selectedIndex;
fun = document.forms[0].fun.options[ind].value;

if (fun == "add") {
    acc = eval(acc) + eval(arg);
} else if (fun == "sub") {
    acc = acc - arg;
} else {
    alert('Please choose either Deposit or Withdraw, then push Proceed.');
```

```

}

document.getElementById("acc").firstChild.nodeValue = acc;
document.forms[0].arg.value = "";

}
</script>
</head>
<body bgcolor=white>

<form>

<table cellpadding=6>

<tr> <td> The accumulator is currently <span id="acc"> 0 (zero). </span> </td> </tr>
<tr> <td> Please choose a function: <select name="fun">
    <option value="non"> Click me!
    <option value="add"> Deposit
    <option value="sub"> Withdraw
</select>

    </td>
</tr>
<tr> <td> Then type an amount: <input type="text" name="arg" size=4> </td> </tr>

<tr> <td> When ready, please press <input type="button"
                                value="Proceed"
                                onClick="calculate()"
                                >
    </td>
</tr>
</tr>

</form>

</body>
</html>

```

15.4 Homework Six

Implement the problem on page 66 using JavaScript, as shown above.

15.5 Minute Papers and Exercises

Here's a program; what does it do?

```
<html><head><title>First to 100</title>

<script language="javascript">

    var acc = 0; // state
    var message = ""; // also part of state

    function calculate() {

        arg = document.forms[0].arg.value; // read input

        var oldAcc = acc;
        // so it would look like fibonacci in the end (for fun and better reporting)

        if (arg > 10 || arg < 1) { message = arg + " is not a legal value."; }
        else { // process the state

            acc += eval(arg);

            if (acc >= 100) { message = "Game ends. You win."; } else {

                var comp = Math.round(1 + Math.random() * 10);

                if (acc >= 100) { message = "Game ends. Computer wins." } else {

                    acc += comp;

                    message = "Game is under way.";

                }

            }

        } // end of process the state
        // (we don't need to store (or retrieve it, for that matter))

        document.getElementById("acc").firstChild.nodeValue
            = oldAcc + " + " + arg + " + " + comp + " = " + acc + " (" + message + ") ";
        // print state

        document.forms[0].arg.value = ""; // get ready for new input
```

```

    }
</script>
</head><body bgcolor=white>

<form>

  <table cellpadding=6>

    <tr> <td> The value of the game is currently
      <span id="acc"> 0 (zero). </span> </td> </tr>

    <tr> <td> Please enter a number (between 1-10):
      <input type="text" name="arg" size=4> </td> </tr>

    <tr> <td> When ready, please press <input type="button"
      value="Proceed"
      onClick="calculate()"
      >

      </td>
    </tr>
  </form>
</body>
</html>

```

Here's another example, written in a slightly different style²:

```

<html>
  <head><title>Homework Five</title>
  <script>
    function Listener() {
      this.process = function process() {
        if (this.message) {
          var answer = document.forms[0].user.value;
          if (answer == this.n1 + this.n2) {
            this.message = "You won.";
            this.userScore += 1;
            this.n1 = Math.floor(Math.random() * 100 - 50);
            this.n2 = Math.floor(Math.random() * 100 - 50);
            this.message += "<p> User: " + this.userScore + ", " +
              " Computer: " + this.compScore + "<p>";
            this.message += " What is " + this.n1 + " + " + this.n2 + "?";
            this.attempts = 0;
          } else {
            this.attempts += 1;
            if (this.attempts == 3) {
              this.message = "You just lost.";
              this.compScore += 1;
            }
          }
        }
      }
    }
  </script>

```

²You should expect questions about all the problems listed before, such as those mentioned on page 82 and more.

```

        this.message += "<p> User: " + this.userScore + ", " +
            " Computer: " + this.compScore + "<p>";
        this.n1 = Math.floor(Math.random() * 100 - 50);
        this.n2 = Math.floor(Math.random() * 100 - 50);
        this.message += " What is " + this.n1 + " + " + this.n2 + "?";
        this.attempts = 0;
    } else {
        this.message = "Keep going. You have " +
            (3 - this.attempts) + " more attempts. <p>" +
            " What is " + this.n1 + " + " + this.n2 + "?";
    }
}
} else {
    this.message = "Welcome.";
    this.userScore = 0;
    this.compScore = 0;
    this.attempts = 0;
    this.n1 = Math.floor(Math.random() * 100 - 50);
    this.n2 = Math.floor(Math.random() * 100 - 50);
    this.message += " What is " + this.n1 + " + " + this.n2 + "?";
}
document.getElementById("message").innerHTML = this.message;
}
}
</script>
</head>
<body>
<form>
    <span id="message"></span> <p>
    Please enter your answer here: <input type="text" name="user"> <p>
    Press <input type="button"
        value="Proceed"
        onClick="listener.process()">
</form>
<script>
    listener = new Listener();
    listener.process();
</script>
</body>
</html>

```

I think this last example in particular is an excellent review of the pattern discussed in this course, in which the implementation combines the strategy with the object-oriented concepts described earlier (and which don't relate only with the shopping cart project presented on-line but are thus useful in a larger context).

Chapter 16

Introduction to Java

16.1 Basic Objects

```
class One {
    public static void main(String[] args) {
        Point a, b;
        a = new Point(-1, 3);
        b = new Point(2, -1);
        double distance = a.distanceTo(b);
        System.out.println (
            "The distance from " +
            a + " to " + b + " is " + distance +

            "\nThe distance from " + b + " to " + a +
            " should be the same, i.e. " + b.distanceTo(a)
        );
    }
}

class Point {
    double x, y;
    Point (int x, int y) {
        this.x = x; this.y = y;
    }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
    public double distanceTo (Point other) {
        return Math.sqrt(
            (this.x - other.x) * (this.x - other.x) +
            (this.y - other.y) * (this.y - other.y)
        );
    }
}
```

16.2 The Class Extension Mechanism

Compile and run this program, then explain:

```
class Horse {
    int numberOfLegs = 4;
    void talk() {
        System.out.println("Howdy!");
    }
}

class Unicorn extends Horse {
    Horn h;
}

class Ionesco {
    public static void main(String[] args) {
        Horse a;
        Unicorn b;
        a = new Horse();
        b = new Unicorn();
        Horse c = new Unicorn();
        /*
         Unicorn d = new Horse(); // this is incorrect
        */
        a.talk();
        b.talk();
        c.talk();
    }
}

class Horn {
    // whatever...
}
```

So we see that inheritance is just the set union of features.

16.3 Dynamic Method Lookup

What do we do in the case of a multiset?

```
class Horse {
    int numberOfLegs = 4;
    void talk() {
        System.out.println("Howdy!");
    }
}

class Unicorn extends Horse {
```

```

    Horn h;
    void talk() {
        System.out.println("Bonjour!");
    }
}

class Ionesco {
    public static void main(String[] args) {
        Horse a;
        Unicorn b;
        a = new Horse();
        b = new Unicorn();
        Horse c = new Unicorn();
        /*
            Unicorn d = new Horse(); // this is incorrect
        */
        a.talk();
        b.talk();
        c.talk();
    }
}

class Horn {
    // whatever...
}

```

This shows that it's the type of the object, not the type of the reference that matters.

16.4 Applets

Place the following program in a file called `Two.java` then compile it and run the `appletviewer` on it:

```
appletviewer Two.java
```

This demonstrates that applets are just like the examples discussed thus far:

```

/*
<applet code="Two.class" width=400 height=400>
</applet>
*/

import java.awt.*;
import java.applet.*;

public class Two extends Applet {
    int count = 0;
    public void paint(Graphics g) {

```

```
        this.count += 1;
        System.out.println("Paint called: " + this.count);
    }
}
```

16.5 Minute Papers and Exercises

Explain what happens in the program below and how this is related to applets:

```
class Horse {

    int numberOfLegs = 4;

    void greet() {
        talk();
    }

    void talk() {
        System.out.println("Howdy!");
    }

}

class Unicorn extends Horse {
    Horn h;
    void talk() {
        System.out.println("Bonjour!");
    }
}

class Ionesco {
    public static void main(String[] args) {
        Horse a = new Unicorn();
        a.greet();
    }
}

class Horn { }
```

Chapter 17

Apache Tomcat

The Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process.

Apache Tomcat (formerly under the Apache Jakarta Project; Tomcat is now a top level project) is a web container developed at the Apache Software Foundation. Tomcat implements the servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Because Tomcat includes its own HTTP server internally, it is also considered a standalone web server.

Tomcat is a web server that supports servlets and JSPs. Tomcat comes with the Jasper compiler that compiles JSPs into servlets.

The Tomcat servlet engine is often used in combination with an Apache webserver or other web servers. Tomcat can also function as an independent web server. Earlier in its development, the perception existed that standalone Tomcat was only suitable for development environments and other environments with minimal requirements for speed and transaction handling. However, that perception no longer exists; Tomcat is increasingly used as a standalone web server in high-traffic, high-availability environments.

Since its developers wrote Tomcat in Java, it runs on any operating system that has a JVM.

Tomcat started off as a servlet specification implementation by James Duncan Davidson, a software architect at Sun. He later helped make the project open source and played a key role in its donation by Sun to the Apache Software Foundation.

Davidson had initially hoped that the project would become open sourced and, since most open source projects had O'Reilly books associated with them featuring an animal on the cover, he wanted to name the project after an animal. He came up with Tomcat since he reasoned the animal represented something

that could take care of and fend for itself. His wish to see an animal cover eventually came true when O'Reilly published their Tomcat book with a tomcat on the cover.

17.1 Installation of Tomcat

Follow the steps below to install your Tomcat:

1. Log into your account on `silو.cs.indiana.edu`
2. Copy the zipped archive into your account:

```
cp /1/www/classes/a348/sum2006/software/apache-tomcat-5.5.17.tar.gz .
```

3. Unzip and unarchive the file there:

```
gunzip apache*tomcat*.gz
tar xvf apache*tomcat*.tar
```

4. Frequently check your quota to make sure all is in good order. When you're done remove the archive and move into the newly created folder:

```
bash-3.00$ ls -l apa*
total 13878
drwxr-xr-x 17 dgerman faculty      512 Jul  9 04:11 apache
drwxr-xr-x 11 dgerman faculty      512 Jul 12 05:33 apache-tomcat-5.5.17
-rw-r--r--  1 dgerman faculty 14192640 Jul 12 05:31 apache-tomcat-5.5.17.tar
bash-3.00$ rm apa*tom*.tar
bash-3.00$ ls -ld apa*
drwxr-xr-x 17 dgerman faculty 512 Jul  9 04:11 apache
drwxr-xr-x 11 dgerman faculty 512 Jul 12 05:33 apache-tomcat-5.5.17
bash-3.00$ cd apa*tom*
bash-3.00$ pwd
/u/dgerman/apache-tomcat-5.5.17
bash-3.00$
```

5. Move into the conf folder.
6. Save the `server.xml` and copy the `server-minimal.xml` into it. Then make changes to make it look like this (use your own ports):

```
<Server port="13299" shutdown="SHUTDOWN">

<GlobalNamingResources>
  <!-- Used by Manager webapp -->
  <Resource name="UserDatabase" auth="Container"
            type="org.apache.catalina.UserDatabase">
```

```

        description="User database that can be updated and saved"
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/tomcat-users.xml" />
</GlobalNamingResources>

<Service name="Catalina">
  <Connector port="13298" />

  <Engine name="Catalina" defaultHost="localhost">
    <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
      resourceName="UserDatabase" />
    <Host name="localhost" appBase="webapps" />
  </Engine>

</Service>
</Server>

```

7. Make sure you have these definitions loaded (in `~/ .bashrc`:

```

JAVA_HOME=/l/jdk1.5
export JAVA_HOME
CATALINA_HOME=/u/dgerman/apache-tomcat-5.5.17
export CATALINA_HOME
CLASSPATH=.:$CATALINA_HOME/common/lib/servlet-api.jar:~/programs/jdbc/mm.mysql-2.0.2-bin.jar
export CLASSPATH

```

8. Now you can start Tomcat:

```

bash-3.00$ source ~/.bashrc
bash-3.00$ $CATALINA_HOME/bin/startup.sh
Using CATALINA_BASE:   /u/dgerman/apache-tomcat-5.5.17
Using CATALINA_HOME:   /u/dgerman/apache-tomcat-5.5.17
Using CATALINA_TMPDIR: /u/dgerman/apache-tomcat-5.5.17/temp
Using JRE_HOME:        /l/jdk1.5
bash-3.00$ ps -ef | grep dgermanb
dgerman  9664  9478  0 05:43 pts/0    00:00:00 grep dgermanb
bash-3.00$ ps -ef | grep dgerman
root      9438  2797  0 05:25 ?        00:00:00 sshd: dgerman [priv]
dgerman   9441  9438  0 05:25 ?        00:00:00 sshd: dgerman@pts/0
dgerman   9442  9441  0 05:25 pts/0    00:00:00 -csh
dgerman   9478  9442  0 05:25 pts/0    00:00:00 bash
dgerman   9644      1 61 05:43 pts/0    00:00:03 /l/jdk1.5/bin/java -Djava.util.log[...]
dgerman   9665  9478  0 05:43 pts/0    00:00:00 ps -ef
dgerman   9666  9478  0 05:43 pts/0    00:00:00 grep dgerman

```

9. You can access it from the following URL:

<http://silo.cs.indiana.edu:13298>

You should, of course, use your own port.

Here's the result:



Now do the following:

```
bash-3.00$ cd $CATALINA_HOME/webapps/ROOT
bash-3.00$ ls -l
total 50
drwxr-xr-x  2 dgerman faculty   512 Jul 12 05:33 admin
-rw-r--r--  1 dgerman faculty  5866 Apr 14 14:09 asf-logo-wide.gif
-rw-r--r--  1 dgerman faculty 21630 Apr 14 14:09 favicon.ico
-rw-r--r--  1 dgerman faculty  8187 Apr 14 14:09 index.jsp
-rw-r--r--  1 dgerman faculty  6381 Apr 14 14:09 RELEASE-NOTES.txt
-rw-r--r--  1 dgerman faculty  1934 Apr 14 14:09 tomcat.gif
-rw-r--r--  1 dgerman faculty  2324 Apr 14 14:09 tomcat-power.gif
drwxr-xr-x  3 dgerman faculty   512 Jul 12 05:33 WEB-INF
bash-3.00$ cd WEB-INF
bash-3.00$ nano web.xml
```

In the file you opened remove:

```
<!-- JSPC servlet mappings start -->
```

```
<servlet>
```

```

        <servlet-name>org.apache.jsp.index_jsp</servlet-name>
        <servlet-class>org.apache.jsp.index_jsp</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>org.apache.jsp.index_jsp</servlet-name>
        <url-pattern>/index.jsp</url-pattern>
    </servlet-mapping>

<!-- JSPC servlet mappings end -->

```

This is responsible for the lack of response when editing the default page. If you change the page and reload, the updated information will be posted to your screen.

```

bash-3.00$ pwd
/u/dgerman/apache-tomcat-5.5.17/webapps/ROOT/WEB-INF
bash-3.00$ cd ..
bash-3.00$ pwd
/u/dgerman/apache-tomcat-5.5.17/webapps/ROOT
bash-3.00$ ls -l
total 50
drwxr-xr-x  2 dgerman faculty   512 Jul 12 05:33 admin
-rw-r--r--  1 dgerman faculty  5866 Apr 14 14:09 asf-logo-wide.gif
-rw-r--r--  1 dgerman faculty 21630 Apr 14 14:09 favicon.ico
-rw-r--r--  1 dgerman faculty  8187 Apr 14 14:09 index.jsp
-rw-r--r--  1 dgerman faculty  6381 Apr 14 14:09 RELEASE-NOTES.txt
-rw-r--r--  1 dgerman faculty  1934 Apr 14 14:09 tomcat.gif
-rw-r--r--  1 dgerman faculty  2324 Apr 14 14:09 tomcat-power.gif
drwxr-xr-x  3 dgerman faculty   512 Jul 12 05:33 WEB-INF
bash-3.00$ nano index.jsp

```

Before we should move any further you should click on the Tomcat Manager link. It requires a password. Here's what you need to do to be able to get in:

```

bash-3.00$ cd $CATALINA_HOME/conf
bash-3.00$ ls -ld *user*
-rw-r--r--  1 dgerman faculty  310 Jul 12 05:43 tomcat-users.xml
bash-3.00$

```

Edit the file to look something like this:

```

<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="t0mcat" password="t0mcat" roles="manager"/>
</tomcat-users>

```

The stop Tomcat, wait a bit, then restart:

```

bash-3.00$ $CATALINA_HOME/bin/shutdown.sh
Using CATALINA_BASE:   /u/dgerman/apache-tomcat-5.5.17
Using CATALINA_HOME:   /u/dgerman/apache-tomcat-5.5.17
Using CATALINA_TMPDIR: /u/dgerman/apache-tomcat-5.5.17/temp
Using JRE_HOME:        /l/jdk1.5
bash-3.00$ ps -ef | grep dgerman
root      9438  2797  0 05:25 ?          00:00:00 sshd: dgerman [priv]
dgerman   9441  9438  0 05:25 ?          00:00:00 sshd: dgerman@pts/0
dgerman   9442  9441  0 05:25 pts/0    00:00:00 -csh
dgerman   9478  9442  0 05:25 pts/0    00:00:00 bash
dgerman   9840  9478  0 06:03 pts/0    00:00:00 ps -ef
dgerman   9841  9478  0 06:03 pts/0    00:00:00 grep dgerman
bash-3.00$ $CATALINA_HOME/bin/startup.sh
Using CATALINA_BASE:   /u/dgerman/apache-tomcat-5.5.17
Using CATALINA_HOME:   /u/dgerman/apache-tomcat-5.5.17
Using CATALINA_TMPDIR: /u/dgerman/apache-tomcat-5.5.17/temp
Using JRE_HOME:        /l/jdk1.5

```

So now if we click on the Tomcat Manager link and provide the username and the password we see a screen as shown on the next page. We will soon see what the screen represents.



17.2 Tomcat Deployment: Contexts

All web applications are located in `$CATALINA_HOME/webapps`. For each application there is a context. To create a context follow the steps below:

```

bash-3.00$ pwd
/u/dgerman/apache-tomcat-5.5.17/webapps
bash-3.00$ ls -ld *
drwxr-xr-x  5 dgerman faculty  512 Apr 14 14:09 balancer
drwxr-xr-x 21 dgerman faculty  512 Jul 12 05:33 jsp-examples
drwxr-xr-x  4 dgerman faculty  512 Jul 12 05:33 ROOT
drwxr-xr-x  4 dgerman faculty  512 Jul 12 05:33 servlets-examples
drwxr-xr-x 12 dgerman faculty 1536 Jul 12 05:33 tomcat-docs
drwxr-xr-x  3 dgerman faculty  512 Jul 12 05:33 webdav
bash-3.00$ mkdir one
bash-3.00$ cd one
bash-3.00$ mkdir WEB-INF
bash-3.00$ cd WE*
bash-3.00$ mkdir classes
bash-3.00$ mkdir lib
bash-3.00$ cd ..
bash-3.00$ cat > index.html
<html><body>This is my new context.</body></html>
bash-3.00$

```

Tomcat Web Application Manager

Message: OK

Manager

[List Applications](#)
 [HTML Manager Help](#)
 [Manager Help](#)
 [Server Status](#)

Applications

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/balancer	Tomcat Simple Load Balancer Example App.	true	0	Start Stop Reload Undeploy
/host-manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/jsp-examples	JSP 2.0 Examples	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/servlets-examples	Servlet 2.4 Examples	true	0	Start Stop Reload Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy
/webdav	Webdav Content Management	true	0	Start Stop Reload Undeploy

Deploy

Deploy directory or WAR file located on server

Context Path (optional):

XML Configuration file URL:

WAR or Directory URL:

WAR file to deploy

Now if we reload the page we notice that the new context on is now listed (see next page). The links on the right of each context will prove very useful in the managing of each context's content.

17.3 A Simple Java Servlet

Let's describe what is needed to create and deploy a servlet. First you need to have the source code in the context's `WEB-INF/classes` folder. Then you need to compile the servlet and deploy it. Deployment is done through a `web.xml`

file, placed in the context's WEB-INF folder.

The Apache Software Foundation
http://www.apache.org/

Tomcat Web Application Manager

Message: OK

Manager

[List Applications](#) [HTML Manager Help](#) [Manager Help](#) [Server Status](#)

Applications

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/balancer	Tomcat Simple Load Balancer Example App	true	0	Start Stop Reload Undeploy
/host-manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/jsp-examples	JSP 2.0 Examples	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/one		true	0	Start Stop Reload Undeploy
/servlets-examples	Servlet 2.4 Examples	true	0	Start Stop Reload Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy
/webdav	Webdav Content Management	true	0	Start Stop Reload Undeploy

Deploy

Deploy directory or WAR file located on server

Context Path (optional):

XML Configuration file URL:

WAR or Directory URL:

Done

We show the source code of a typical servlet below followed by a `web.xml` file that contains more than one description. The servlet shown could have been much simpler, granted; however the one we present below has the advantage of being more useful in the longer run.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Three extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        String message, good, total, n1, n2, me;

        me = request.getContextPath() + request.getServletPath();

        message = request.getParameter("message");
        good = request.getParameter("good");
        total = request.getParameter("total");
        n1 = request.getParameter("n1");
        n2 = request.getParameter("n2");

        if (message == null) {

            n1 = (int)(Math.random() * 100 - 50) + "";
            n2 = (int)(Math.random() * 100 - 50) + "";
            good = "0";
            total = "0";
            message = "Welcome, score is: " + good + "/" + total +
                ". What is " + n1 + " + " + n2 + "?";

        } else {

            String answer = request.getParameter("answer");

            if (Integer.parseInt(answer) == Integer.parseInt(n1) + Integer.parseInt(n2)) {
                good = (Integer.parseInt(good) + 1) + "";
                message = "Good answer. ";
            } else {
                message = "Not good. ";
            }
            total = Integer.parseInt(total) + 1 + "";
            message += "score is: " + good + "/" + total + ". ";
            n1 = (int)(Math.random() * 100 - 50) + "";
            n2 = (int)(Math.random() * 100 - 50) + "";
            message += " What is " + n1 + " + " + n2 + "?";

        }

    }
}
```

```

        out.println(
            "<form action=\"" + me + "\">" +
            " " + message + "<p>" +
            " Type your answer here: <input type=\"text\" name=\"answer\"> <p> " +
            " Push <input type=\"submit\" value=\"Proceed\"> when ready. " +
            " <input type=\"hidden\" name=\"message\" value=\"" + message + "\">" +
            " <input type=\"hidden\" name=\"good\" value=\"" + good + "\">" +
            " <input type=\"hidden\" name=\"total\" value=\"" + total + "\">" +
            " <input type=\"hidden\" name=\"n1\" value=\"" + n1 + "\">" +
            " <input type=\"hidden\" name=\"n2\" value=\"" + n2 + "\">" +
            "</form>"
        );
    }
}

```

The web.xml file:

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <servlet>
        <servlet-name>Three</servlet-name>
        <servlet-class>Three</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Three</servlet-name>
        <url-pattern>/servlet/Three</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>Fv2_3</servlet-name>
        <servlet-class>Two</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Fv2_3</servlet-name>
        <url-pattern>/servlet/Fancy</url-pattern>
    </servlet-mapping>

</web-app>

```

Now the context can be accessed from:

```
http://silo.cs.indiana.edu:13298/one
```

The servlet can be accessed from:

```
http://silo.cs.indiana.edu:13298/one/servlet/Three
```

17.4 Minute Papers and Exercises

Deploy the following servlet:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class One extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        String message = request.getParameter("message");
        String you = request.getContextPath() + request.getServletPath();
        PrintWriter out = response.getWriter();
        // Use "out" to send content to browser
        response.setContentType("text/html");
        if(message == null) {
            message = "1";
            out.println(message);
        }
        else {
            message = "" + (Integer.parseInt(message) + 1);
            out.println(message);
        }

        out.println(
            "<html>"+
            "<head><title>Homework 6</title></head>"+
            "<body>"+
            "<form method=\"get\" action = \""+you+"\">"+

            "<input type=\"hidden\" name= \"message\" value= \""+message+"\">"+
            "<input type= \"submit\" value= \"proceed\">"+
            "</form>"+
            "</body>"+
            "</html>"
        );
    }
}
```

17.5 A Simple JSP

Here's the JSP that results. As you might be able to see already the translation is immediate. It's much easier to deploy a JSP: you just place it in the context (above WEB-INF and then access it.

```
<%
    String message = request.getParameter("message");

    String you = request.getContextPath() + request.getServletPath();

    if(message == null) {
        message = "1";
    } else {
        message = "" + (Integer.parseInt(message) + 1);
    }
%>

<html>
<head><title>Homework 6</title></head>
<body>
    <form method="get" action="<%=you%>">

        The counter is: <%=message%> <p>
        <input type="hidden" name="message" value="<%=message%>">
        <input type="submit" value="proceed">
    </form>

</body>
</html>
```

The URL in this case will be:

```
http://silo.cs.indiana.edu:13298/one/One.jsp
```

This, of course, assumes that the name of the file is `One.jsp`.

Chapter 18

Server-Side Java

18.1 The Basic Servlet Template

In this section we will discuss only the source code. We assume that the details of deploying the servlets are well understood; they don't change from servlet to servlet.

Here's the starting point:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Number extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
        throws ServletException,
            IOException {

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("Hola, how are ya (today)?");

    }
}
```

18.2 The Manager Application

Use the manager application to reload any server whose source code has been modified and recompiled.

18.3 User Input, Sessions

Here's basic information about sessions:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Number extends HttpServlet {
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException,
            IOException {

        res.setContentType("text/html");

        HttpSession who = req.getSession();

        PrintWriter out = res.getWriter();

        out.println("Session ID: " + who.getId() +
            "\n<br>Creation Time: " + who.getCreationTime() +
            "\n<br>Last Accessed: " + who.getLastAccessedTime());

    }
}
```

Here's how we can use sessions:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Number extends HttpServlet {
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException,
            IOException {

        res.setContentType("text/html");

        HttpSession who = req.getSession();

        Integer count = (Integer)who.getAttribute("count");

        if (count == null) {
            who.setAttribute("count", new Integer(1));
        } else {
```

```

        int aux = count.intValue();
        aux += 1;
        who.setAttribute("count", new Integer(aux));
    }

    PrintWriter out = res.getWriter();

    out.println("Session ID: " + who.getId() +
        "\n<br>Creation Time: " + who.getCreationTime() +
        "\n<br>Last Accessed: " + who.getLastAccessedTime() +
        "\n<br>Count: " + count);

    }
}

```

And here's basic information about user input:

```

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Number extends HttpServlet {
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException,
        IOException {

        res.setContentType("text/html");

        HttpSession who = req.getSession();

        Integer count = (Integer)who.getAttribute("count");

        if (count == null) {
            who.setAttribute("count", new Integer(1));
        } else {
            int aux = count.intValue();
            aux += 1;
            who.setAttribute("count", new Integer(aux));
        }

        String one = req.getParameter("one"),
            two = req.getParameter("two");

        PrintWriter out = res.getWriter();

        out.println("Session ID: " + who.getId() +

```

```

        "\n<br>Creation Time: " + who.getCreationTime() +
        "\n<br>Last Accessed: " + who.getLastAccessedTime() +
        "\n<br>Count: " + count +
        "\n<br>One: " + one +
        "\n<br>Two: " + two);
    }
}

```

Providing the user input to the servlet above should be a piece of cake for you now. Also, keeping state with hidden fields is now possible. (Why only now?)

18.4 Transformation to JSP

Here's stage one of a Java Server Page:

```

<html><head><title>JSP Stage One</title></head>
<body bgcolor="white">

    <%= new java.util.Date() %>

</body>
</html>

```

Recall that deployment is much easier than for servlets.

Discussing the following JSP will strengthen our understanding between a JSP and the underlying servlet.

```

<html><head><title>JSP Stage One</title></head>
<body bgcolor="white">

    <%= new java.util.Date() %>

    JSP has access to a number of predefined variables. <p>

    Here are some: <ol>

        <li> <code>session</code>

        <li> <code>request</code>

        <li> <code>response</code>

    </ol>

    <p> Let's use them. <p>

    Here's your session ID: <%= session.getId() %> <p>

```

Here's the value of parameter named `two`:

```
<%= request.getParameter("two") %> <p>
```

Here's an `<%! int count = 0; %>`

instance variable (`count`)

that's incremented every time: `<%= ++count %>` <p>

```
</body>
```

```
</html>
```

The next example relates to the last two servlets above:

```
<html><head><title>JSP Stage One</title></head>
```

```
<body bgcolor="white">
```

First we run the scriptlet. <p>

```
<% Integer count = (Integer)session.getAttribute("count");
```

```
    if (count == null) {
        session.setAttribute("count", new Integer(1));
    } else {
        int aux = count.intValue();
        aux += 1;
        session.setAttribute("count", new Integer(aux));
    }
```

```
%>
```

<p> Which you can't see. <p>

Then we print `<%= count %>`

```
</body>
```

```
</html>
```

And we wrap up with this:

```
<html><head><title>JSP Stage One</title></head>
```

```
<body bgcolor="white">
```

A declaration: `<%! int count = 6; %>` (invisible) <p>

A scriptlet: `<% int count = 3; %>` (invisible) <p>

An expression: `<%= count %>` (prints 3, doesn't it?) <p>

```

Another expression: <%= this.count %> (should print 6) <p>

A scriptlet printing them both: <%
    out.println(count + " + " + this.count + " = three + six = 9 (nine)");
%>

</body>
</html>

```

18.5 Homework Seven

Implement the program on page 66 in four ways:

1. as a Java servlet keeping state with hidden fields
2. Java servlet keeping state in a session
3. JSP keeping state with hidden fields
4. JPS with session-base state

18.6 A Note on Using Sessions

Here's an example of stamps with sessions in JSP:

```

<%
String me = request.getContextPath() + request.getServletPath();
// who am I?

String count = (String) session.getAttribute("count");
Integer stamp = (Integer) session.getAttribute("stamp");
// retrieve state (this part is new)

int index = (count == null) ? 0 : Integer.parseInt(count);
// initialize state

if (stamp == null) { stamp = new Integer(0); }

String whichWay = request.getParameter("fun");
// read input to change state (part I)

if (whichWay == null) { // protecting against bad input

} else {

    // the empty string is different from null!

String userStamp = request.getParameter("stamp");
// read user's stamp (part of input)

```

```
boolean match;
try {
    match = (stamp.intValue() == Integer.parseInt(userStamp));
} catch (Exception e) { match = false; } // comparing the stamps

if (match) {

    String arg = request.getParameter("arg");
    // reading input to change state (part II)

    if (whichWay.equals("up")) {
        // changing the state one way or another

        index += Integer.parseInt(arg);

    } else if (whichWay.equals("down")) {

        index -= Integer.parseInt(arg);

    } else {

        // something went wrong (we don't do anything)

    }

    stamp = new Integer(stamp.intValue() + 1);

    session.setAttribute("count", index + ""); // save state

    session.setAttribute("stamp", stamp); // save the stamp (it's part of the state)

} else {

}

}
%>

<html>
<head><title>My Last JSP</title></head>

<body bgcolor=white><table>

<form method="GET" action="<%=me%>">

    The result is currently: <%=index%> <p>
    <input type="text" name="arg" size=4>

<p>
```

```
Please choose an action: <select name="fun">
  <option value="nothing"> Click Me!
  <option value="up"> Addition
  <option value="down"> Substraction
</select> <p>

Then press <input type="submit" value="Proceed">

<input type="hidden" name="stamp" value="<%=stamp%>">

</form>

</body>

</html>
```

Frequently you will first write a servlet, which you will then translate into a JSP. If the two are using sessions you need to plan ahead to avoid potential conflicts. The best strategy would be to name the session variables in such a way that they could not be mistaken for the variables of the other program. So instead of `count` and `stamp` you could use `count_problemOne` and `stamp_problemOne` for the JSP, for example, and different names for the servlet, such as: `count_problemTwo` and `stamp_problemTwo`, although usually you would solve the servlet first.

Chapter 19

Java Database Connectivity

19.1 Basic Driver Installation

The driver is located in the common software folder:

```
bash-3.00$ cd /1/www/classes/a348/sum2006/software
bash-3.00$ pwd
/1/www/classes/a348/sum2006/software
bash-3.00$ ls -ld *
-rw-r--r-- 1 dgerman www 5994530 Jul  7 22:09 apache-tomcat-5.5.17.tar.gz
-rw-r--r-- 1 dgerman www 6282043 Jul  7 22:07 httpd-2.2.2.tar.gz
-rw-r--r-- 1 dgerman www  71328 Jul 12 08:59 mm.mysql-2.0.2-bin.jar
-rw-r--r-- 1 dgerman www 19542405 Jul  7 22:05 mysql-5.0.22.tar.gz
-rw-r--r-- 1 dgerman www 8109575 Jul  7 21:57 php-5.1.4.tar.gz
bash-3.00$
```

Just keep this in mind.

19.1.1 Using the CLASSPATH

Recall that CLASSPATH must point to it in some instances.

19.1.2 The lib Folder

Place the driver in the WEB-INF/lib folder of the context that will use JDBC.

19.2 Standalone Database Access

We will first look at basic database access. Create a `~/programs/jdbc` folder and move there. Also copy your driver here as well.

19.2.1 Creating Tables

This is an example of a program that creates tables in the database:

```
import java.sql.*;

public class One {
    public static void main(String[] args) {
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
        } catch (Exception e) {
            System.out.println("Can't load JDBC Driver. " +
                "Make sure classpath is correct.");
        }

        System.out.println("Loaded the JDBC driver.");

        String url = "jdbc:mysql://silo.cs.indiana.edu:13297/demoOne",
            username = "dgerman",
            password = "sp00n";
        Connection connection;

        try {
            connection = DriverManager.getConnection(url, username, password);

            Statement statement = connection.createStatement();
            ResultSet result;
            System.out.println("Creating tables...");

            statement.executeUpdate(
                " create table dgerman_person ( " +
                "   name      varchar(100) not null primary key, " +
                "   age       int unsigned not null           , " +
                "   lives_in  varchar(100) not null           " +
                " ) "
            );

            statement.close();
            connection.close();

        } catch (SQLException e) {
            System.out.println("An SQLException occurred: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}
```

You should check with your MySQL database to confirm.

```

bash-3.00$ mysql --socket=/nobackup/dgerman/mysql/mysql.sock \
--port=13297 --host=silo.cs.indiana.edu -u dgerman -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 5.0.22-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use demoOne
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_demoOne          |
+-----+
| dgerman_accumulator        |
| dgerman_guessTheNumber     |
| dgerman_person             |
| dgerman_transactions_draft |
| dgerman_users_draft        |
+-----+
5 rows in set (0.00 sec)

mysql>

```

19.2.2 Inserting Data

The programs in this and the next section simply insert data, and retrieve it.

```

import java.sql.*;

public class InsertData {
    public static void main(String[] args) {
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
        } catch (Exception e) {
            System.out.println("Can't load JDBC Driver. " +
                "Make sure classpath is correct.");
        }
        String url = "jdbc:mysql://localhost/a348",
            username = "a348",
            password = "a348AG";
        Connection connection;

        try {
            connection = DriverManager.getConnection(url, username, password);

```

```

Statement statement = connection.createStatement();
ResultSet result;
System.out.println("Inserting data...");

statement.executeUpdate(
    " insert into dgerman_person values          " +
    " ('David Beckham ', 24, 'England'),        " +
    " ('Roger Milla   ', 25, 'Africa'),          " +
    " ('George Weah   ', 24, 'Africa'),          " +
    " ('Tony Meola    ', 25, 'USA' ),            " +
    " ('Zinedine Zidane', 25, 'France')         "
);

statement.close();
connection.close();

} catch (SQLException e) {
    System.out.println("An SQLException occurred: " + e.getMessage());
} catch (Exception e) {
    System.out.println("Exception: " + e);
}
}
}

```

Note: they need to be updated to work. I am sure you can do that easily.

19.2.3 Retriving Information

Just like the program before three minor changes are needed here:

```

import java.sql.*;

public class SelectData {
    public static void main(String[] args) {
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
        } catch (Exception e) {
            System.out.println("Can't load JDBC Driver. " +
                "Make sure classpath is correct.");
        }
        String url = "jdbc:mysql://localhost/a348",
            username = "a348",
            password = "a348AG";
        Connection connection;

        try {
            connection = DriverManager.getConnection(url, username, password);

```

```
Statement statement = connection.createStatement();
ResultSet result;
System.out.println("Querying database...");

result = statement.executeQuery(
    " select name, age from dgerman_person    " +
    "      where lives_in = 'Africa'        "
);

while (result.next()) {
    System.out.println(" " + result.getString(1) +
        ", " + result.getString(2));
}

statement.close();
connection.close();

} catch (SQLException e) {
    System.out.println("An SQLException occurred: " + e.getMessage());
} catch (Exception e) {
    System.out.println("Exception: " + e);
}
}
}
```

19.3 Servlet Database Access

This will be a lab assignment: turn the last program into a servlet.

Chapter 20

Commencement

This is the end of our introductory tour of web programming essentials. You have acquired important information and fundamental programming skills that will enable you to understand virtually any web technology in use today.

There are a number of things that we didn't cover although we would have wanted; among them

- Java networking and especially Java RMI,
- XML processing and in particular
- the use of XML in language-independent networking
- such as XML-RPC or web services;
- web multiplayer game design¹;

All these topics proved to be popular in the past.

There are also others that could (and probably should) be covered in a sequel to this class:

- Hibernate
- Java Server Faces
- Struts
- Sakai
- ColdFusion
- ASP.NET 2.0

Perhaps a time when we will prepare a material about them as well.

In the meantime we congratulate you on your accomplishments and: keep practicing your newly acquired skills. Be well, do good work, and keep in touch

¹With Flash on the client side and a Java server on the server side

