

# **IIUSA-519: *CGI Programming in Perl***

## **IIUSA Official Curriculum**

# **IIUSA-519: *CGI Programming in Perl I***

**SYNOPSIS:** This course covers how to create server-side scripts that can be invoked via the World Wide Web.

**PREREQUISITE:** Basic knowledge of HTML.

# TOPICS

## Session 1

- Introduction to the World Wide Web
- Client-server interaction
- HyperText Markup Language (HTML) review
- Introduction to Common Gateway Interface (CGI)
- Introduction to the Perl Programming Language

## Session 2

- CGI forms
- CGI input and output
- File handles

## Session 3

- “Website in a file” concept
- Line oriented database

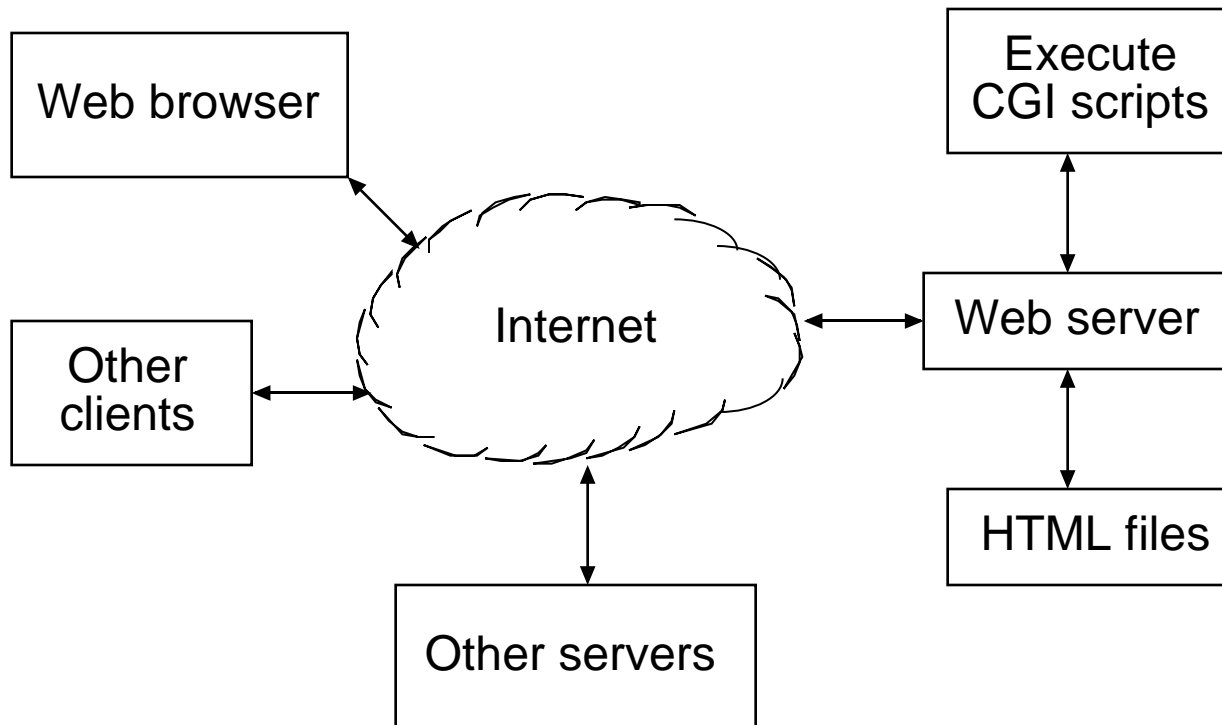
## Session 4

- Passing Arguments to/from subroutines
- Socket programming

# Session 1

- **Client-server interaction**
- **A simple Perl script**
- **MIME types**
- **Executing a command on the server**
- **Environment variables**
- **Query strings**
- **Fill-out forms**
- **GET method vs. POST method**
- **Setting up a CGI server script**

# Web Client-Server Interaction



# Uniform Resource Locators (URLs)

<http://internetinstituteusa.com:80/index.html>

## Protocol

http  
ftp  
file  
telnet  
gopher  
nntp  
wais  
smtp  
mailto

## Hostname

Can also be an IP  
address such as  
216.3.196.244

## Port

http default: 80  
ftp default: 21  
secure http default: 443

## Path

index.html or index.htm  
is generally the server  
default if the filename is  
not explicitly provided  
in the URL.

# A Simple Perl Script – plaintext.cgi

```
#!/usr/bin/perl
# The name of this file is "plaintext.cgi".
#
# NOTE: This file must be placed in the cgi-bin directory, and CGI
# must be enabled on the server machine, for this to work.
# Otherwise, the HTML code will be printed directly onto the
# screen.
#
# Don't forget to change the mode of this file to be readable and
# executable by all, so that it can be executed by a Web server.
# On a Unix machine, you can make this file readable and executable
# by all with:
#
#           chmod a+rx plaintext.cgi
#
# By the way, comments begin with a pound sign (#).
#

print <<END;
Content-type: text/plain

Here are a few lines of text. Notice that there
is no formatting, because the Content-type
is text/plain. If the Content-type is text/html
then we can put HTML tags here.

END
```

# MIME Types

- Multipurpose Internet Mail Extensions (MIME) types allow graphics, sound, video, and other information to be sent via email.

## Type/Subtype

`text/plain` -- Plain text  
`text/html` -- HTML text  
`image/gif` -- Graphics Interchange Format (GIF) image

# MIME Type text/html – `htmltext.cgi`

- An example using the text/html MIME type
- [http://internetinstituteusa.com/cgi-bin/519\\_scripts/htmltext.cgi](http://internetinstituteusa.com/cgi-bin/519_scripts/htmltext.cgi)

```
#!/usr/bin/perl
# The name of this file is "htmltext.cgi".
# Don't forget to put a blank line after the Content-type line.

print <<END;
Content-type: text/html

<HTML><HEAD>
<TITLE>This is HTML Formatting</TITLE>
</HEAD>
<BODY>
<H1>This is HTML Formatting</H1>
The Content-type is changed to text/html, which tells
the Web browser to look for HTML tags and
<B>format</B> the HTML text as appropriate <I>for the tags</I>.
<P>Here is a hyperlink<A HREF="http://internetinstituteusa.com">
http://internetinstituteusa.com</A>
</BODY></HTML>
END
```

# Creating a New MIME Type – `iiusatext.cgi`

```
#!/usr/bin/perl
# The name of this file is "iiusatext.cgi".
print <<END;
Content-type: text/x-iiusa00xx
```

The text that appears here is of type "text/x-iiusa00xx".  
It will be read by a word processor application on  
the Web browser system, after the end-user chooses  
a word processor.  
END

# Invoking a Command on the Server – Calendar Script `calendar.cgi`

```
#!/usr/bin/perl
# The name of this file is "calendar.cgi".
# This example comes from "How to Set Up
# and Maintain a Web Site,"
# 2/e, Lincoln D. Stein, Addison Wesley,
# (1997), p. 474.
```

```
$CAL='/usr/bin/cal';
$DATE='/bin/date';
```

```
# Fetch the current year using the Unix
# date command
$year = 2000 +`$DATE +%y`;
```

```
# Fetch the text of the calendar using the
# cal command
chop($calendar_text=`$CAL $year`);
```

```
# Print it all out now
print <<END;
Content-type: text/html
```

```
<HTML><HEAD>
<TITLE>Calendar for Year $year</TITLE>
```

```
</HEAD><BODY>
<H1>Calendar for Year $year</H1>
<PRE>
$calendar_text
</PRE> <HR>
<A HREF="http://
internetinstituteusa.com">Internet Institute
USA</A>
</BODY></HTML>
END
```

# Perl Data Types

- Perl has three data types: scalar, array, associative array.
- **scalar** – Holds a single item, prepended with (\$).  
**Ex:** `$year = 2001;`
- **array** – Holds a number of items, prepended with (@). First element is indexed at 0.  
**Ex:** `@baz = (10, 25, 44);`  
`$a = $baz[0];` **places 10** in \$a.
- **associative array** – An array that is indexed by a name instead of a number, prepended with (%). Made up of key-value pairs.  
**Ex:** `%teams = ('Scarlet Knights' => 'Rutgers',  
              'Huskies' => 'UCONN',  
              'Eagles' => 'Boston College');`  
`$name_of_college = $teams{'Huskies'};` **places**  
**'UCONN'** in \$name\_of\_college.

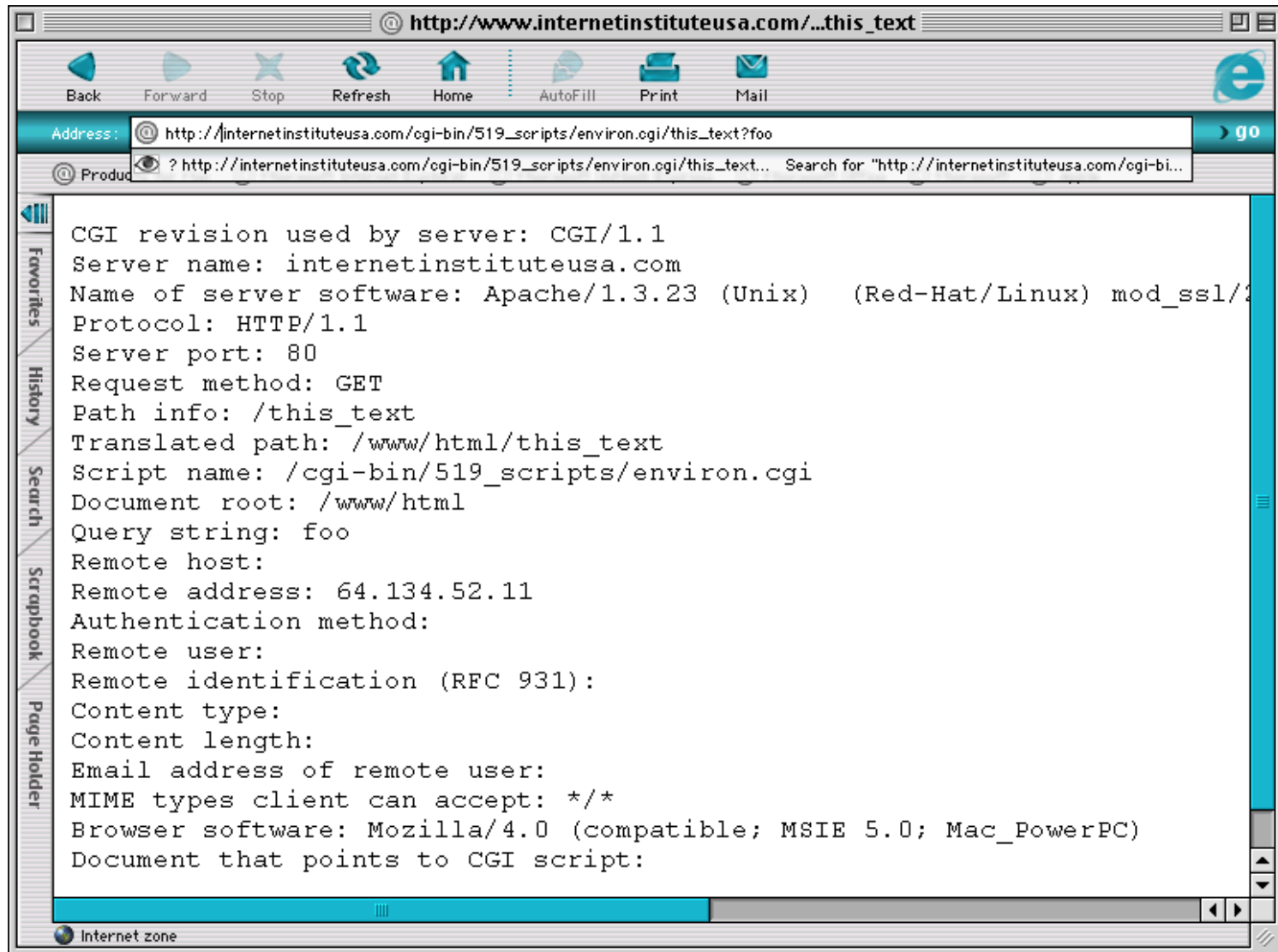
# Environment Variables – `environ.cgi`

```
#!/usr/bin/perl

print "Content-type: text/plain\n\n";

print "CGI revision used by server: ", $ENV{'GATEWAY_INTERFACE'}, "\n";
print "Server name: ", $ENV{'SERVER_NAME'}, "\n";
print "Name of server software: ", $ENV{'SERVER_SOFTWARE'}, "\n";
print "Protocol: ", $ENV{'SERVER_PROTOCOL'}, "\n";
print "Server port: ", $ENV{'SERVER_PORT'}, "\n";
print "Request method: ", $ENV{'REQUEST_METHOD'}, "\n";
print "Path info: ", $ENV{'PATH_INFO'}, "\n";
print "Translated path: ", $ENV{'PATH_TRANSLATED'}, "\n";
print "Script name: ", $ENV{'SCRIPT_NAME'}, "\n";
print "Document root: ", $ENV{'DOCUMENT_ROOT'}, "\n";
print "Query string: ", $ENV{'QUERY_STRING'}, "\n";
print "Remote host: ", $ENV{'REMOTE_HOST'}, "\n";
print "Remote address: ", $ENV{'REMOTE_ADDR'}, "\n";
print "Authentication method: ", $ENV{'AUTH_TYPE'}, "\n";
print "Remote user: ", $ENV{'REMOTE_USER'}, "\n";
print "Remote identification (RFC 931): ", $ENV{'REMOTE_IDENT'}, "\n";
print "Content type: ", $ENV{'CONTENT_TYPE'}, "\n";
print "Content length: ", $ENV{'CONTENT_LENGTH'}, "\n";
print "Email address of remote user: ", $ENV{'HTTP_FROM'}, "\n";
print "MIME types client can accept: ", $ENV{'HTTP_ACCEPT'}, "\n";
print "Browser software: ", $ENV{'HTTP_USER_AGENT'}, "\n";
print "Document that points to CGI script: ", $ENV{'HTTP_REFERER'}, "\n";
```

# Output Produced by `environ.cgi`



The screenshot shows a web browser window with the address bar containing `http://www.internetinstituteusa.com/...this_text`. The address bar also shows the full URL: `http://internetinstituteusa.com/cgi-bin/519_scripts/envIRON.cgi/this_text?foo`. The main content area displays the output of the `environ.cgi` script, which is a list of server and request details in a plain text format.

```
CGI revision used by server: CGI/1.1
Server name: internetinstituteusa.com
Name of server software: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2
Protocol: HTTP/1.1
Server port: 80
Request method: GET
Path info: /this_text
Translated path: /www/html/this_text
Script name: /cgi-bin/519_scripts/envIRON.cgi
Document root: /www/html
Query string: foo
Remote host:
Remote address: 64.134.52.11
Authentication method:
Remote user:
Remote identification (RFC 931):
Content type:
Content length:
Email address of remote user:
MIME types client can accept: */*
Browser software: Mozilla/4.0 (compatible; MSIE 5.0; Mac_PowerPC)
Document that points to CGI script:
```

# Environment Variables – `environ2.cgi`

```
#!/usr/bin/perl

print <<END;
Content-type: text/plain

CGI revision used by server: $ENV{'GATEWAY_INTERFACE'}
Server name: $ENV{'SERVER_NAME'}
Name of server software: $ENV{'SERVER_SOFTWARE'}
Protocol: $ENV{'SERVER_PROTOCOL'}
Server port: $ENV{'SERVER_PORT'}
Request method: $ENV{'REQUEST_METHOD'}
Path info: $ENV{'PATH_INFO'}
Translated path: $ENV{'PATH_TRANSLATED'}
Script name: $ENV{'SCRIPT_NAME'}
Document root: $ENV{'DOCUMENT_ROOT'}
Query string: $ENV{'QUERY_STRING'}
Remote host: $ENV{'REMOTE_HOST'}
Remote address: $ENV{'REMOTE_ADDR'}
Authentication method: $ENV{'AUTH_TYPE'}
Remote user: $ENV{'REMOTE_USER'}
Remote identification (RFC 931): $ENV{'REMOTE_IDENT'}
Content type: $ENV{'CONTENT_TYPE'}
Content length: $ENV{'CONTENT_LENGTH'}
Email address of remote user: $ENV{'HTTP_FROM'}
MIME types client can accept: $ENV{'HTTP_ACCEPT'}
Browser software: $ENV{'HTTP_USER_AGENT'}
Document that points to CGI script: $ENV{'HTTP_REFERER'}

END
```

# Query Strings – date\_or\_cal.cgi

```
#!/usr/bin/perl

$CAL='/usr/bin/cal';
$DATE='/bin/date';

$query_string = $ENV{'QUERY_STRING'};

if ($query_string eq "date") {
    print "Content-type: text/plain\n\n";
    print ` $DATE `;
}

else {

# Fetch the current year using the Unix date
# command

$year = 2000 + ` $DATE +%y `;

# Fetch the text of the calendar using the
# cal command

chop($calendar_text=` $CAL $year `);

# Print it all out now
print <<END;
Content-type: text/html
```

```
<HTML><HEAD>
<TITLE>Calendar for Year $year</TITLE>
</HEAD><BODY>
<H1>Calendar for Year $year</H1>
<PRE>
$calendar_text
</PRE>
<HR>
<A HREF="http://
internetinstituteusa.com">Internet Institute
USA</A>
</BODY></HTML>
END
}
```

# `if/elsif/else` Construct

```
if ($query_string eq "date") {  
    # date code goes here  
}
```

```
elsif ($query_string eq "cal") {  
    # cal code goes here  
    # There can be an arbitrary number of elsif clauses  
}
```

```
else {  
    # Default code goes here  
}
```

# Fill-Out Forms – calendar2.cgi

```
#!/usr/bin/perl
# Input looks like "year=2001"

$CAL='/usr/bin/cal';

# Get the year from the user
$query_string = $ENV{'QUERY_STRING'};
($field_name, $year) = split (/=/, $query_string);

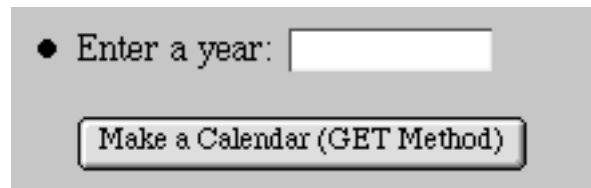
# Fetch the text of the calendar using the cal command
chop($calendar_text=`$CAL $year`);

# Print it all out now
print <<END;
Content-type: text/html

<HTML><HEAD>
<TITLE>Calendar for Year $year</TITLE>
</HEAD><BODY>
<H1>Calendar for Year $year</H1>
<PRE>
$calendar_text
</PRE>
<HR>
<A HREF="http://internetinstituteusa.com">Internet Institute USA</A>
</BODY></HTML>
END
```

# HTML Code for calendar2.cgi

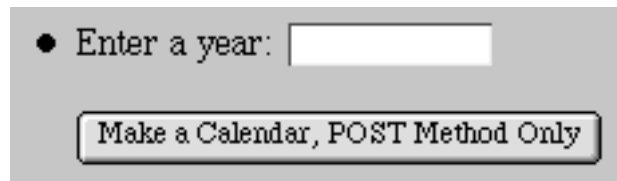
```
<FORM ACTION="http://internetinstituteusa.com/cgi-bin/519_scripts/calendar2.cgi"
METHOD="GET">
Enter a year: <INPUT TYPE="text" NAME="year" SIZE=10>
<P>
<INPUT TYPE="submit" VALUE="Make a Calendar (GET Method)">
</FORM>
```



● Enter a year:

# HTML Code for calendar3.cgi

```
<FORM ACTION="http://internetinstituteusa.com/cgi-bin/519_scripts/calendar3.cgi" METHOD="POST">
Enter a year: <INPUT TYPE="text" NAME="year" SIZE=10>
<P>
<INPUT TYPE="submit" VALUE="Make a Calendar, POST Method Only">
</FORM>
```



● Enter a year:

# GET vs. POST – calendar3.cgi

```
#!/usr/bin/perl

$CAL='/usr/bin/cal';

# Get the year from the user
$size_of_form_information = $ENV{'CONTENT_LENGTH'};

read (STDIN, $form_info, $size_of_form_information);

($field_name, $year) = split (/=/, $form_info);

# Fetch the text of the calendar using the cal command
chop($calendar_text=`$CAL $year`);

# Print it all out now
print <<END;
Content-type: text/html

<HTML><HEAD>
<TITLE>Calendar for Year $year</TITLE>
</HEAD><BODY>
<H1>Calendar for Year $year</H1>
<PRE>
$calendar_text
</PRE> <HR>
<A HREF="http://internetinstituteusa.com">Internet Institute USA</A>
</BODY></HTML>

END
```

# GET or POST - calendar4.cgi

```
#!/usr/bin/perl

$CAL='/usr/bin/cal';
$request_method = $ENV{'REQUEST_METHOD'};

# Get the year from the user

if ($request_method eq "GET") {
    $form_info = $ENV{'QUERY_STRING'};
}
else {
    $size_of_form_information =
$ENV{'CONTENT_LENGTH'};
    read (STDIN, $form_info,
$size_of_form_information);
}

# At this point, $form_info has the input
# data from the browser.
($field_name, $year) = split ( /=/,
$form_info);

# Fetch the text of the calendar using the
cal command

chop($calendar_text=`$CAL $year`);
```

```
# Print it all out now
print <<END;
Content-type: text/html
<HTML><HEAD>
<TITLE>Calendar for Year $year</TITLE>
</HEAD><BODY>
<H1>Calendar for Year $year</H1>
<PRE>
$calendar_text
</PRE>
<HR>
<A HREF="http://
internetinstituteusa.com">Internet Institute
USA</A>
</BODY></HTML>
END
```

# Session 2

- **More on fill-out forms: radio buttons, checkboxes, textfields, password fields, scrolled lists, and menus**
- **Exercise: madlibs**
- **Server redirection**
- **Reading from a file**
- **Writing to a file**
- **Subroutines**

# Radio Buttons, Checkboxes, and More

Enter your username:

Enter your password:

Select days you are available: Monday  Tuesday  Wednesday  Thursday  Friday

Select a semester:

Fall:

Spring:

Summer:

Pay by:

Which courses? 

Beginning HTML	▲
Intermediate HTML	☰
Advanced HTML	■
CGI Programming	■
Web Technology	▼

Give us any special instructions:

# Radio Buttons, Checkboxes, and More

```
<FORM ACTION="http://
internetinstituteusa.com/cgi-bin/519_scripts/
sampleform.cgi"
METHOD="GET">
Enter your username: <INPUT TYPE="text"
NAME="Username" SIZE=30>
<P>
Enter your password: <INPUT TYPE="password"
NAME="Password" SIZE=10>
<P>
Select days you are available:
Monday <INPUT TYPE="checkbox" NAME="Monday">
Tuesday <INPUT TYPE="checkbox"
NAME="Tuesday">
Wednesday <INPUT TYPE="checkbox"
NAME="Wednesday">
Thursday <INPUT TYPE="checkbox"
NAME="Thursday">
Friday <INPUT TYPE="checkbox" NAME="Friday">
<P>
Select a semester:<BR>
Fall: <INPUT TYPE="radio" NAME="Semester"
VALUE="Fall" CHECKED><BR>
Spring: <INPUT TYPE="radio" NAME="Semester"
VALUE="Spring"><BR>
Summer: <INPUT TYPE="radio" NAME="Semester"
VALUE="Summer"><BR>
<P>
```

Pay by:

```
<SELECT NAME="paymethod" SIZE=1>
<OPTION SELECTED>Pay by check
<OPTION>Pay by P.O.
<OPTION>Bill me
</SELECT>
<P>
```

Which courses?

```
<SELECT NAME="course" SIZE=5 MULTIPLE>
<OPTION>Beginning HTML
<OPTION>Intermediate HTML
<OPTION>Advanced HTML
<OPTION>CGI Programming
<OPTION>Web Technology
<OPTION>Java for Beginners
<OPTION>Java for Programmers
<OPTION>Graphic Java
</SELECT>
<P>
```

Give us any special instructions:

```
<TEXTAREA ROWS=10 COLS=40 NAME="instructions">
Type a message here.
</TEXTAREA>
<P>
<CENTER>
<INPUT TYPE="submit" VALUE="Submit the Form">
<INPUT TYPE="reset" VALUE="Clear All Fields">
</CENTER>
</FORM>
```

# Fill-Out Forms – sampleform.cgi

```
#!/usr/bin/perl
# The name of this file is "sampleform.cgi".

$request_method = $ENV{'REQUEST_METHOD'};

# Get the form data

if ($request_method eq "GET") {
    $form_info = $ENV{'QUERY_STRING'};
}
else {
    $size_of_form_information =
$ENV{'CONTENT_LENGTH'};
    read (STDIN, $form_info,
$size_of_form_information);
}

print <<END1;
Content-type: text/html

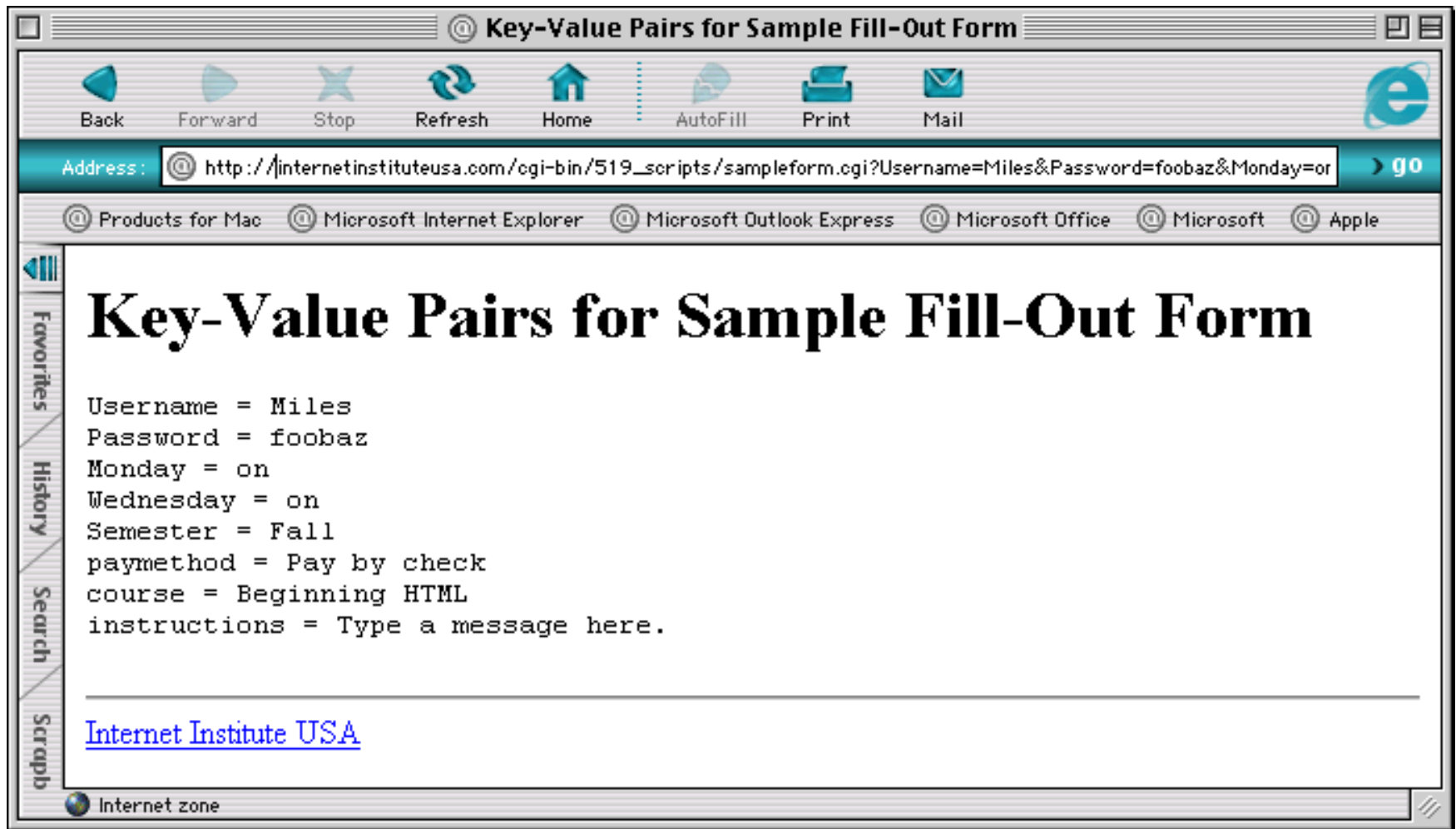
<HTML><HEAD>
<TITLE>Key-Value Pairs for Sample Fill-Out
Form</TITLE>
</HEAD><BODY>
<H1>Key-Value Pairs for Sample Fill-Out
Form</H1>
<PRE>
END1
```

```
@key_value_pairs = split(/&/, $form_info);

foreach $key_value (@key_value_pairs) {
    ($key, $value) = split (/=/,
$key_value);
    $value =~ tr/+// ;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack
("C", hex ($1))/eg;
    print $key, " = ", $value, "\n";
}

print <<END2;
</PRE>
<HR>
<A HREF="http://
internetinstituteusa.com">Internet Institute
USA</A>
</BODY></HTML>
END2
```

# Output from sampleform.cgi



# Fill-Out Form Example – madlibs.html

```
<html>
<head>
<title>A Form To Try</title>
</head>

<body bgcolor="#FFFFCC">

<form action="http://internetinstituteusa.com/cgi-bin/519_scripts/madlibs.cgi"
method="POST">

<h3 align="center">This form will result in a custom madlibs Web page.</h3>

<pre>
    Your Name: <input type="text" name="username">
    Your Age: <input type="text" name="age">
    Your Shoe Size: <input type="text" name="shoesize">
    Favorite Smell: <input type="text" name="favesmell">
</pre>

<input type="submit" value="Send it On!"><br><br>
<input type="reset" value="Start Over!">

</form>

</body>
</html>
```

# Fill-Out Form Example – madlibs.cgi

```
#!/usr/bin/perl
# The name of this file is "madlibs.cgi".

$request_method = $ENV{'REQUEST_METHOD'};

# Get the form data

if ($request_method eq "GET") {
    $form_info = $ENV{'QUERY_STRING'};
}
else {
    $size_of_form_information =
$ENV{'CONTENT_LENGTH'};
    read (STDIN, $form_info,
$size_of_form_information);
}

print <<END1;
Content-type: text/html

<HTML><HEAD>
<TITLE>Madlibs Form Results</TITLE>
</HEAD><BODY>
<PRE>
END1

@key_value_pairs = split(/&/, $form_info);
```

```
foreach $key_value (@key_value_pairs) {
    ($key, $value) = split ( /=/, $key_value);
    $value =~ tr/+// ;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack
("C", hex ($1))/eg;
    $FORM{$key} = $value;
}

print "<h1>What a Great Day to be you,
$FORM{'username'} !</h1>";

print "<p>You could have an IQ less than
$FORM{'shoesize'}, but you don't!</p>";

print "<p>You could be 122 instead of
$FORM{'age'}, but you aren't!</p>";

print "<p>Is that $FORM{'favesmell'} I
smell?</p>";

print <<END2;
</PRE>
<HR>
<A HREF="http://
internetinstituteusa.com">Internet Institute
USA</A>
</BODY></HTML>
END2
```

# Server Redirection

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";
print "<META HTTP-EQUIV=REFRESH CONTENT=\"0; URL=http://
www.yahoo.com\">\n\n";
```

# Input File directory.dat

Jim Thomas|Communication|SCILS 415|husatonic@scils.rutgers.edu|732-555-8826  
Andrew Mayer|Library and Information Studies|DeWitt 415|jahlov@scils.rutgers.edu|732-555-7501  
Jennifer Chromos|Journalism and Mass Media|DeWitt 417|ante@scils.rutgers.edu|732-555-7369  
Bell J. Starr|Library and Information Studies|SCILS 418|belken@scils.rutgers.edu|732-555-8585  
Sumner Sampson|Communication|SCILS 419|bigboy@scils.rutgers.edu|732-555-7910  
Lulu Babinski|Communication|SCILS 516|lulu@scils.rutgers.edu|732-555-7878  
MaryLou Krieger|Library and Information Studies|662|maryl@scils.rutgers.edu|732-555-6262  
Laurie L.|Library and Information Studies|202|lmarceau@yahoo.com|555-8965

# Reading from a File – read.cgi

```
#!/usr/bin/perl

$filename = "directory.dat";

print <<MARKER_A;
Content-type:text/html

<html><head><title>Staff Directory</
title></head>

<body bgcolor="#FFFFCC">

MARKER_A

open(INF,$filename);
@indata = <INF>;
close(INF);

print "<table border=1>";
print "<tr><th>Faculty Name</
th><th>Department</th><th>Room Number</th>
<th>Email</th><th>Phone Number</th></
tr>\n";

foreach $i (@indata) {
    chop($i);
    ($facultyname,$department,$roomnumber,
    $email,$phonenumber) = split(/\|/, $i);

    print "<tr>";
    print "<td>$facultyname</td>";
    print "<td>$department</td>";
    print "<td>$roomnumber</td>";
    print "<td><a
href=\"mailto:$email\">$email</a></td>";
    print "<td>$phonenumber</td>";
    print "</tr>\n";
}
print "</table>";
print "</body></html>";
```

# Writing to a File – administer.html

```
<html><head>
<title>Administer your Database</title>
</head>

<body bgcolor="#FFFFCC">

<form action="http://internetinstituteusa.com/cgi-bin/admindirectory.cgi" method="post">

<h2>This form will add people to our directory</h2>

<pre>
Faculty Name: <input type="text" name="facultyname">
  Department: <select name="department">
<option>Communication</option>
<option>Library and Information Studies</option>
<option>Journalism and Mass Media</option>
</select>
      Room: <input type="text" name="roomnumber">
      Email: <input type="text" name="email">
Phone Number: <input type="text" name="phonenumber">
</pre>

<input type="submit" value="add to database">

</form>

</body></html>
```

# Writing to a File – admindirectory.cgi

```
#!/usr/bin/perl                                     # write everything into the directory file

# Get the form data                                open(OUTF, ">>directory.dat");

$request_method = $ENV{'REQUEST_METHOD'};

if ($request_method eq "GET") {
    $form_info = $ENV{'QUERY_STRING'};
}
else {
    $size_of_form_information =
$ENV{'CONTENT_LENGTH'};
    read (STDIN, $form_info,
$size_of_form_information);
}

@pairs = split(/&/, $form_info);

foreach $pair (@pairs) {
    ($name, $value) = split(/=/, $pair);
    $value =~ tr/+//;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/
pack ("C", hex ($1))/eg;
    $FORM{$name} = $value;
}
```

# SMTP telnet Session with Mail Server

```
%telnet internetinstitute.com 25
```

```
Trying 216.3.196.244
```

```
Connected to internet.rutgers.edu
```

```
220 ESMTP spoken here
```

```
MAIL FROM:<me@here.com>
```

```
250 Sender OK
```

```
RCPT TO:<iiusa21xy@internetinstituteusa.com>
```

```
250 Recipient OK
```

```
DATA
```

```
354 Enter mail, end with a "." on a line by itself
```

```
Testing...
```

```
.
```

```
250 Message accepted
```

```
QUIT
```

# Subroutines – sampleform2.cgi

```
#!/usr/bin/perl
# The name of this file is
"sampleform2.cgi".

$request_method = $ENV{'REQUEST_METHOD'};

# Get the form data

if ($request_method eq "GET") {
    $form_info = $ENV{'QUERY_STRING'};
}
else {
    $size_of_form_information =
$ENV{'CONTENT_LENGTH'};
    read (STDIN, $form_info,
$size_of_form_information);
}

@key_value_pairs = split(/&/, $form_info);

$has_password = 0;
$has_username = 0;

foreach $key_value (@key_value_pairs) {
    ($key, $value) = split (/=/,
$key_value);
    $value =~ tr/+/ /;
```

```
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack
("C", hex ($1))/eg;

    # The Username/Password pair is
hardwired to: "Guest"/"baz"
    if ($key eq 'Username') {
        $has_username = 1;
        if (!($value eq 'Guest')) {
            &return_error(500, "Bad
username", $value);
        }
    }
    elsif ($key eq 'Password') {
        $has_password = 1;
        if (!($value eq 'baz')) {
            &return_error(500, "Bad pass-
word", $value);
        }
    }
}

# Did someone submit a bad form?
if (($has_username == 0) ||
($has_password == 0)) {
    &return_error(500, "Bad form",
"Missing username or password");
}
```

# Subroutines (continued #1)

```
&print_all_OK;
exit(0);

# Here is the error subroutine

sub return_error {
local ($status, $keyword, $message) = @_ ;

print "Content-type: text/html", "\n";
print "Status: ", $status, " ", $keyword,
"\n\n";

print <<End_Of_Error;

<HTML><HEAD>
<TITLE>CGI Program - Unexpected Error</TITLE>
</HEAD><BODY>
<H1>$keyword</H1>
<HR>$message<HR>
</BODY></HTML>

End_Of_Error
exit(1);
}

# If the username and password check OK, then
we invoke this routine
```

```
sub print_all_OK {
print <<END1;
Content-type: text/html

<HTML><HEAD>
<TITLE>Processing a Fill-Out Form</TITLE>
</HEAD><BODY>
<H1>Processing a Fill-Out Form</H1>
This confirms that your form was processed,
with the following
parameters:
<PRE>
END1

foreach $key_value (@key_value_pairs) {
    ($key, $value) = split (/,/,
$key_value);
    $value =~ tr/+/ /;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack
("C", hex ($1))/eg;
    if (!(($key eq 'Username') && !($key eq
'Password'))) {
        print $key, " = ", $value, "\n";
    }
}
}
```

# Subroutines (continued #2)

```
print <<END2;
</PRE><HR>
<A HREF="http://internetinstituteusa.com">Internet Institute USA</A>
</BODY></HTML>
END2
# Replace your_address@company.com with your email address, and you will
# receive an email message for each form submitted. Be mindful of
# the backslash that appears before the commercial at sign (@), which
# is needed so that it does not look like an array, which
# also starts with (@). You will need to replace your_address@company.com
# in two places, and also change the "From:" line to correspond to
# your account on internet.rutgers.edu.

open(SENDMAIL, "| /usr/lib/sendmail -fyour_address\@company.com -t -n -oi");
print SENDMAIL <<End_Of_Mail;
From: iiusa00xx <iiusa00xx\@internetinstituteusa.com>
To: your_address\@company.com
Subject: Confirmation of processed form
This confirms that your form was processed, with the following parameters:

End_Of_Mail
  foreach $key_value (@key_value_pairs) {
    ($key, $value) = split (/=/, $key_value);
    $value =~ tr/+// /;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack ("C", hex ($1))/eg;
    if (!($key eq 'Username') && !($key eq 'Password')) {
      print SENDMAIL $key, " = ", $value, "\n";
    } } }
```

# Session 3

- **Line oriented image database format**
- **Converting multiline records to single line records**
- **The Sprite module**
- **“Website in a file” concept**
- **Database Perl code**

# Flat, Text-Oriented Database Format

- **Microsoft Excel comma delimited (.csv) format has a header record followed by data records:**

Name , Course , Grade

John , ITI-519 , B

Mary , ITI-460 , A

- **Each record is delimited by a newline for this case, and each field is delimited by a comma.**

# Image Database Format

- We would like to create a line oriented database that keeps track of images in terms of their location (a URL), a description, and keywords used for searching.
- We start with an easier to edit multiline format, with records delimited by “-=-” and fields separated by newlines, and then convert it into a comma-delimited format.
- Example source format:

```
ImageLocation
```

```
Description
```

```
Keywords
```

```
--=
```

```
http://waldo.wi.mit.edu/WWW/examples/Ch9/pictures/camel.gif
```

```
This is a picture of camels in the foreground, loaded with supplies.+  
There are snow covered mountains in the background. This comes from+  
Lincoln Stein\'s collection.
```

```
camel mountain desert
```

```
--=
```

```
http://waldo.wi.mit.edu/WWW/examples/Ch9/pictures/alps.gif
```

```
A skier and a dog are in the foreground, looking over the peaks of the Alps.+  
From Lincoln Stein\'s collection.
```

```
mountain Alp dog ski
```

# Converting the Multiline Format to Comma Delimited Format

```
#!/usr/bin/perl

open (INFILE, 'PictureSource.db') || die "Cannot open PictureSource.db\n";
open (OUTFILE, '>PictureFinal.db') || die "Cannot create PictureFinal.db\n";

$/ = "\n--\n"; # The delimiter between records, which contain newlines.

while (<INFILE>) {
    chomp;          # Remove ---
    s/,/\\",/g;     # Convert any ',' to '\,'
    s/\n/,/g;      # Convert newline to ','
    s/,$/;/;       # Remove final ','

    # A plus sign (+) at the end of the line is a continuation character.
    # The plus sign is converted to a space character, and a newline is
    # not produced at the end because the field has not yet ended.
    # This allows a single field to extend over several lines.

    s/\+,/ /g;     # Convert '+' to ' '
    print OUTFILE "$_\n"; # Print converted line to OUTFILE
}

close (INFILE);
close (OUTFILE);
```

# The Converted Database Format

ImageLocation,Description,Keywords

<http://waldo.wi.mit.edu/WWW/examples/Ch9/pictures/camel.gif>,This is a picture of camels in the foreground\, loaded with supplies. There are snow covered mountains in the background. This comes from Lincoln Stein\'s collection.,camel mountain desert

<http://waldo.wi.mit.edu/WWW/examples/Ch9/pictures/alps.gif>,A skier and a dog are in the foreground\, looking over the peaks of the Alps. From Lincoln Stein\'s collection.,mountain Alp dog ski

<http://waldo.wi.mit.edu/WWW/examples/Ch9/pictures/cactus.gif>,A cactus flower is blooming. From Lincoln Stein\'s collection.,cactus flower

<http://www.internet.rutgers.edu/ITI/Press/Focus/248a-1.jpg>,The Hill 248 laboratory.,workstation computer desk

<http://www.cs.rutgers.edu/~murdocca/Executive.gif>,An image that Miles scanned in from an issue of Scientific American. Forgot who the artist is.,executive robot automaton

# The Database Code – Main Routine

```
#!/usr/bin/perl

# The Sprite library comes from the CPAN archive
# at ftp://ftp.cis.ufl.edu/pub/perl/CPAN in
# modules/by-authors/Shishir_Gundavaram.  Examples
# of how to use Sprite are in "CGI Programming on
# the World Wide Web" by Shishir Gundavaram, O'Reilly
# & Associates, (1996).

# This comes from the Sprite author, in a README file in the distribution:
# Sprite, v3.21
# -----
#
# What is it? No, it's not the soda that claims, "Image is Nothing, Thirst
# is Everything" :-) However, it is a useful module that allows you to access
# text-delimited databases, such as those exported from applications like
# MS Excel, using a _small_ subset of SQL.
#
# Sprite has some cool advantages, like allowing you to embed Perl's powerful
# regular expressions -- not those wimpy ones that come with your commercial
# relational database engine -- into your queries.
#
# [...]
#
# Shishir Gundavaram
# April 21, 1998

#
```

# Main Routine (cont' #1)

```
# The following code is written by Miles Murdocca, October 2002
#
#
# The next line tells the Perl interpreter to look in the Sprite-3.21
# directory to find libraries, in addition to the default library path.
# You will need to change this path to correspond to the place where
# you install Sprite on your system.  On internet.rutgers.edu at Rutgers
# University, Sprite is already installed at this location so there is
# nothing more for you to do if you are using that system.
#
use lib '/home/iiusa0004/public_html/cgi-bin/Sprite-3.21';

# The next line selects a specific library we will use (there are
# two Sprite libraries in the library path above.  We will use Sprite.pm,
# which is indicated by simply specifying "Sprite" as below.)

use Sprite;

# The PATH_INFO environment variable extracts information
# from the URL.  So, if the name of this file is
# "Pictures.cgi", then a URL such as:
# http://internetinstituteusa.com/~iiusa00xy/cgi-bin/Pictures.cgi/Update
# will put "/Update" into the PATH_INFO environment variable.
#
# Here are the supported PATH_INFO fields:
#
```

# Main Routine (cont' #2)

```
# / -- (No additional path information) Show the initial page.
# /ShowAll -- Show everything in the database.
# /Query -- Execute a query
# /Update -- Show the update page
# /(Anything else) -- Show the initial page
#

# Change the next line to correspond to your cgi-bin
$DOCUMENT_ROOT = "http://internetinstituteusa.com/~iiusa0004/cgi-bin";

# Change the next line to correspond to your login id
$webmaster = "iiusa2xyz\@internetinstituteusa.com";

# Change the next line if you change the name of the database.
# *** VERY IMPORTANT *** The way we are doing it, *you* are the owner of
# this file, and so, the Web server cannot modify it unless you change
# the permissions to be (somewhat dangerously) writable by all.  From the
# Unix command line, use:

# internet> chmod a+rw PictureFinal.db

# A slightly better solution has the Web server create the file, but
# that is not a lot better.  A better solution is to run the server with
# your permissions and also check for a password.
#
$DATABASE_LOCATION = "PictureFinal.db";

# The password is used in process_update, for adding entries to the database
```

# Main Routine (cont' #3)

```
#
$PASSWORD = "foo";

$path_info = $ENV{'PATH_INFO'};

# Delete leading "/" from $path_info, so "/Update" becomes "Update". Notice
# that "/" is a special character used in Perl patten matching, and so we need
# to escape it with a backslash.

($discard, $path_info) = split(/\/, $path_info);

# Associative array (a.k.a. hash) "FORM" below will hold the key/value pairs

# For PATH_INFO = "/Query"
if ($path_info eq "Query") {
    &parse_form_data(*FORM);
    &print_query();
}
# For PATH_INFO = "/Update"
elsif ($path_info eq "Update") {
    &parse_form_data(*FORM);
    &process_update();
}
# For PATH_INFO = "/ShowAll"
elsif ($path_info eq "ShowAll") {
    &parse_form_data(*FORM);
    &print_showall();
}

# For PATH_INFO = all other cases.
else {
    &print_main_form();
}

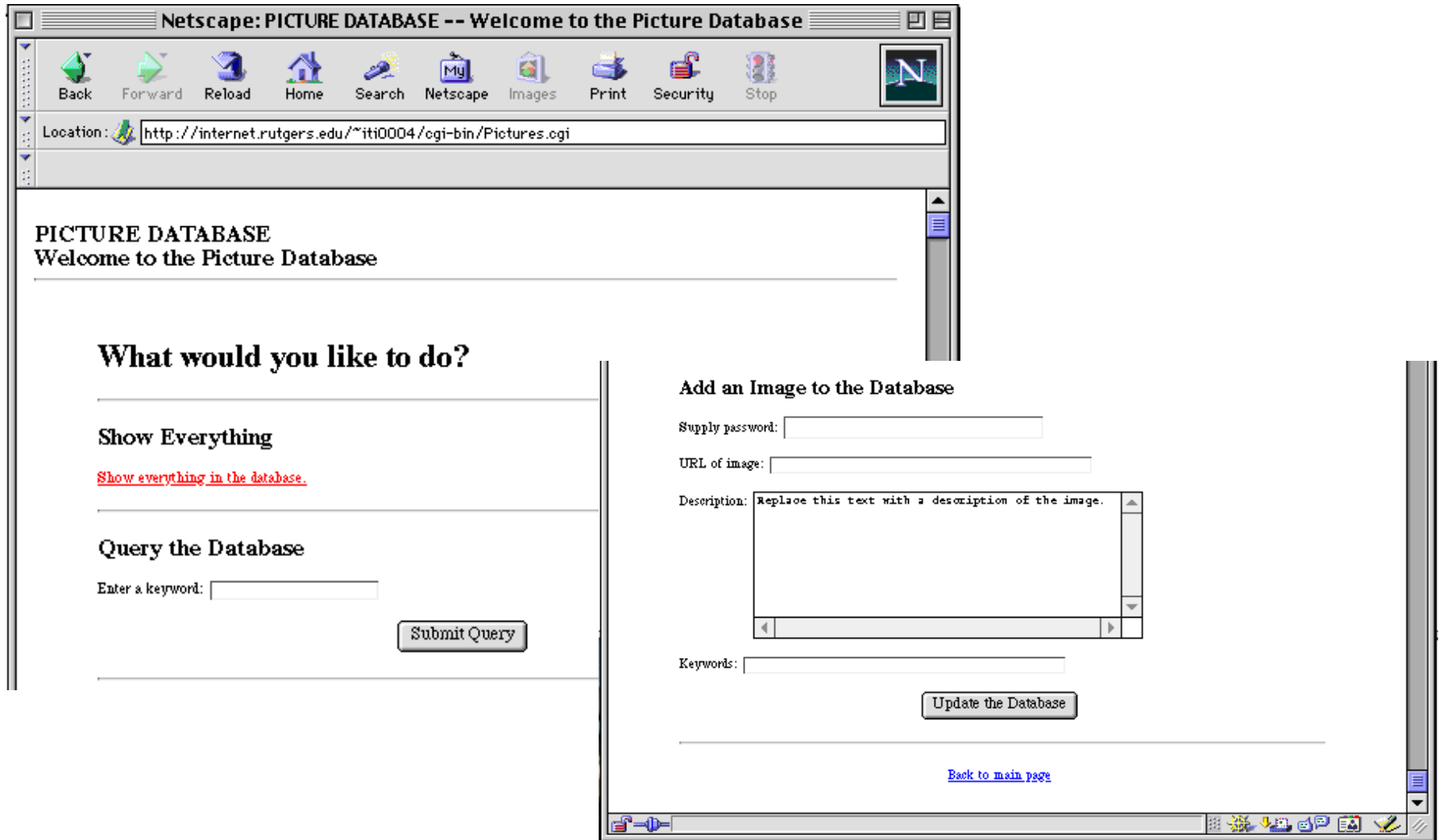
exit(0);

##### THIS IS THE END OF THE MAIN PROGRAM.
##### SUBROUTINES START HERE. #####
```

# Database Code – Summary of Subroutines

- `print_main_form` – **Creates the top-level Web page.**
- `print_showall` – **Creates a Web page that shows the database contents.**
- `print_query` – **Creates a Web page that allows the user to query the database.**
- `process_update` – **Creates a Web page that allows the user to update (add a record to) the database.**
- `parse_form_data` – **Reads data from the Web browser and breaks it into key-value pairs.**
- `return_error` – **Creates a Web page that indicates an error.**
- `print_HTML_Header` – **Prints the leading HTML text for a Web page.**
- `print_HTML_Trailer` – **Prints the trailing HTML text for a Web page.**
- `database_error` – **Prints an error message.**

# Top Level Database Web Page



# Database Code – print\_main\_form

```
#
# ----- SUBROUTINE print_main_form
#
# Create an HTML page for the main page.
#

sub print_main_form {
&print_HTML_Header("Welcome to the Picture
Database");

    print <<HTML_Middle;
<H1>What would you like to do?</H1>
<P><HR>
<H2>Show Everything</H2>
<A HREF="$DOCUMENT_ROOT/Pictures.cgi/ShowAll">
Show everything in the database.</A>
<P>
<HR>
<P>
<H2>Query the Database</H2>
<FORM ACTION="Pictures.cgi/Query"
METHOD="POST">
<P>Enter a keyword:
<INPUT TYPE="text" NAME="keyword" SIZE=20>
<P>
<CENTER>
<INPUT TYPE="submit" VALUE="Submit Query">
</CENTER>
```

```
</FORM>
<P><HR><P>
<H2>Add an Image to the Database</H2>
<P>
<FORM ACTION="Pictures.cgi/Update"
METHOD="POST">
Supply password:
<INPUT TYPE="password" NAME="Password"
SIZE=30>
<P>URL of image:
<INPUT TYPE="text" NAME="URL" SIZE=45>
<P>Description:
<TEXTAREA ROWS=10 COLS=50 NAME="Description">
Replace this text with a description of the
image.
</TEXTAREA>
<P>Keywords:
<INPUT TYPE="text" NAME="Keywords" SIZE=45>
<P>
<CENTER>
<INPUT TYPE="submit" VALUE="Update the Data-
base">
</CENTER>
</FORM>
<P>
HTML_Middle

    &print_HTML_Trailer;
}
```

# Database Code – print\_showall

```
#
# ----- SUBROUTINE print_showall
#
# Create an HTML page that lists full pic-
# ture info.
#
sub print_showall {
    # Create a new Sprite database "object"
    $rdb = new Sprite();

    # Read in all records in the database,
    # which are separated by commas.
    $rdb->set_delimiter("Read", ",");
    @data = $rdb->sql("select * from
$DATABASE_LOCATION");
    $rdb->close();

    $status = shift(@data); # status, which
# is set by rdb->sql, is OK if nonzero
    $no_elements = scalar(@data); # Find
    how many records were retrieved.

    if (!$status) {
        &return_error(500, "Database Error",
"Sprite database error.");
    } elsif (!$no_elements) {
        &return_error(500,
```

```
"Database Error", "No records.");
    }

    &print_HTML_Header("Show Entire Database");

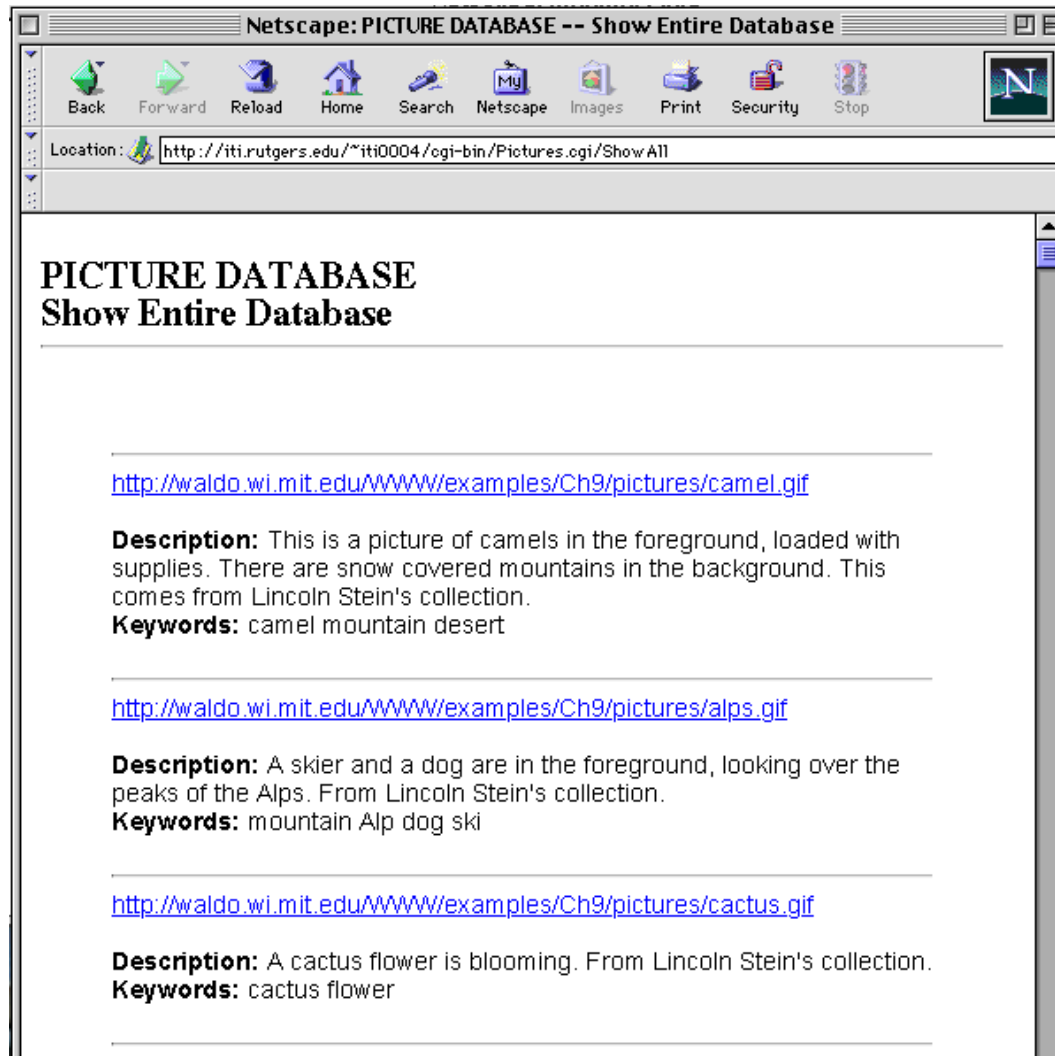
    # Each database record has three fields:
    # ImageLocation,Description,Keywords

    foreach $record (@data) {
        ($ImageLocation, $Description, $Keywords)
= @$record;

        print "<HR>\n";
        print "<A
href=\" $ImageLocation\">\n";
        print "$ImageLocation</A>";
        print "<P><B>Description:</B>
$Description<BR>\n";
        print "<B>Keywords:</B>
$Keywords<BR><P>\n";
    }

    &print_HTML_Trailer;
}
```

# The ShowAll Web Page



# Database Code – print\_query

```
#
# ----- SUBROUTINE print_query
#
# Create an HTML page that results from a
# query.
#

sub print_query {
    # Create a new Sprite database "object"
    $rdb = new Sprite();

    # Read in all records in the database,
    # which are separated by commas.
    $rdb->set_delimiter("Read", ",");

    @data = $rdb->sql(
        "select * from $DATABASE_LOCATION
where (Keywords =~ /$FORM{'keyword'}/)");
    $rdb->close();

    $status = shift(@data); # status, which
    # is set by rdb->sql, is OK if nonzero
    $no_elements = scalar(@data); # Find
    # how many records were retrieved.

    if (!$status) {
        &return_error(500, "Database Error",
            "Sprite database error.");
    }
    elsif (!$no_elements) {
        &return_error(500, "Database Error",
            "No records.");
    }

    &print_HTML_Header("Show Query Re-
sults");

    # Each database record has three fields:
    # ImageLocation,Description,Keywords

    foreach $record (@data) {
        ($ImageLocation, $Description, $Keywords)
        = @$record;

        print "<HR>\n";
        print "<A
        HREF=\"\$ImageLocation\">\n";
        print "$ImageLocation</A>";
        print "<P><B>Description:</B>
        $Description<BR>\n";
        print "<B>Keywords:</B>
        $Keywords<BR><P>\n";
    }

    &print_HTML_Trailer;
}
```

# Session 4

- **Database Perl code (continued from Session 3)**
- **Introduction to socket programming in Perl**

# Database Code – process\_update

```
#
# ----- SUBROUTINE process_update
#
# Add a record to the database.
#

sub process_update {

    # Check for the password:
    if (!($FORM{'Password'} eq $PASSWORD)) {
        &return_error(500, "Password error",
            "You are not authorized to make
changes to the database.");
    }

    # Create a new Sprite database "object"
    $rdb = new Sprite();

    # Prepare the database for writing
    $rdb->set_delimiter("Read", ",");
    $rdb->set_delimiter("Write", ",");

    $rdb->sql(<<End_of_Insert) ||
&database_error();

insert into $DATABASE_LOCATION
    (ImageLocation, Description, Keywords)
values

        ('$FORM{'URL'}',
        '$FORM{'Description'}',
        '$FORM{'Keywords'}')
End_of_Insert

    $rdb->close($DATABASE_LOCATION);

    &print_main_form();
}
```

# Database Code – parse\_form\_data

```
#
# ----- SUBROUTINE parse_form_data
#
# Read in the parameters uploaded from the browser to the server.  If
# there are multiple entries with the same name, like:
#     FILENAME=file1.gif
#     FILENAME=file2.gif
# then the entries are catenated like this:
#     file1.gif\0file2.gif
# and the catenated entries are assigned to key FILENAME.
#
sub parse_form_data {
    # "@_" is a magic variable that holds argument passed to the subroutine.
    local(*FORM_DATA) = @_;

    # When a variable is declared "local", its scope is limited to
    # the subroutine.
    local($request_method, $query_string, @key_value_pairs,
          $key_value, $key, $value);

    # This same code will work for either the GET or POST methods
    $request_method = $ENV{'REQUEST_METHOD'};

    if ($request_method eq "GET") {
        $query_string = $ENV{'QUERY_STRING'};
    } elsif ($request_method eq "POST") {
        read (STDIN, $query_string, $ENV{'CONTENT_LENGTH'});
    } else {
```

# Database Code – parse\_form\_data

```
&return_error (500, "Server Error",
    "Browser uses unsupported method");
}

# The browser uploads information like this:
# key1=value1&key2=value2&key3=value3
# The next line breaks the input into key/value pairs.
@key_value_pairs = split(/&/, $query_string);

# Now, break each key/value pair into a key and a value. Also, undo
# any damage the browser had to do encoding special characters like
# spaces, plus signs, etc.
foreach $key_value (@key_value_pairs) {
    ($key, $value) = split(/=/, $key_value);
    $value =~ tr/+/ /;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack ("C", hex ($1))/eg;

    if (defined($FORM_DATA{$key})) {
        $FORM_DATA{$key} = join("\0", $FORM_DATA{$key}, $value);
    } else {
        $FORM_DATA{$key} = $value;
    }
}
}
```

# Database Code – return\_error

```
#
# ----- SUBROUTINE return_error
#
sub return_error {
    local($status, $keyword, $message) = @_;

    print"Content-type: text/html", "\n";
    print"Status: ", $status, " ", $keyword, "\n\n";

    print <<End_of_Error;

<HTML>
<HEAD>
    <TITLE>CGI Program - Unexpected Error</TITLE>
</HEAD>
<BODY>
<H1>$keyword</H1>
<HR>$message<HR>
Please contact $webmaster for more information.
</BODY>
</HTML>

End_of_Error

    exit(1);
}
```

# Database Code – print\_HTML\_Header

```
#
# ----- SUBROUTINE print_HTML_Header
#
# Print the header for a Web page
#
sub print_HTML_Header {
    local ($title) = @_ ;
    print <<HTML_Header;
Content-type: text/html

<HTML>
<HEAD>
<TITLE>PICTURE DATABASE -- $title</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000"
LINK="#0000FF" VLINK="#FF0000"
ALINK="#808080">
<BASEFONT FACE="Verdana,Arial,Helvetica">
<P>
<br>
<P>
<table border="0" width="550">
  <tr>

  <td align="left">
  <FONT size="+2"><B>PICTURE
DATABASE<BR>$title</B>
```

```
<HR>
<P>

<P>
</FONT>
</td>
<P>
  </tr>
  <tr>
    <td><blockquote>
      <p align="left"><font
face="Arial,Helvetica" size="+0>

HTML_Header
  }
```

# Database Code – print\_HTML\_Trailer

```
#
# ----- SUBROUTINE print_HTML_Trailer
#
# Print the trailer for a Web page
#
sub print_HTML_Trailer {

    print <<HTML_Trailer;
<P>
<HR>
<P>
<CENTER>

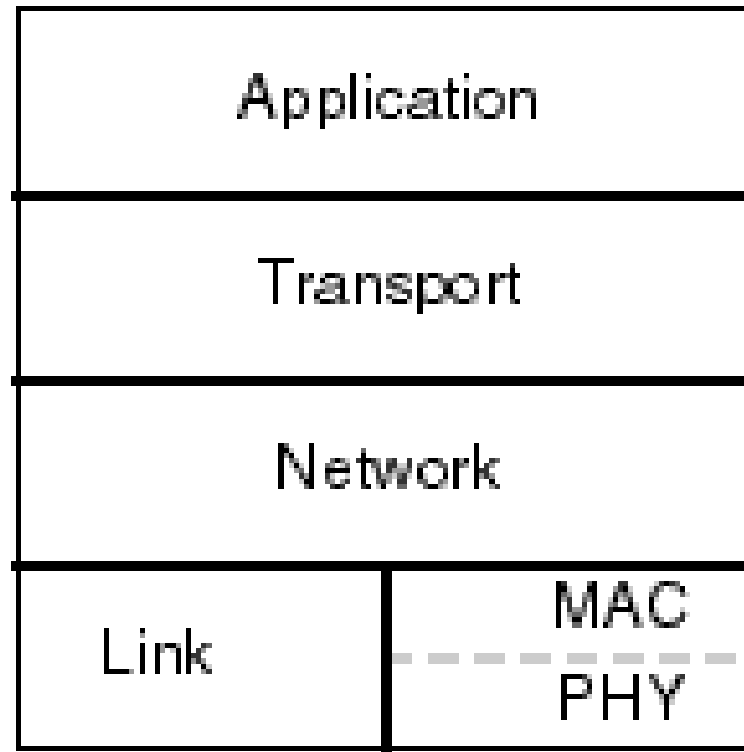
<A HREF="$DOCUMENT_ROOT/Pictures.cgi">Back to main page</A>
</CENTER>
    </td>
    </tr>
</table>
<P>
</BODY></HTML>
HTML_Trailer
}
```

# Database Code – database\_error

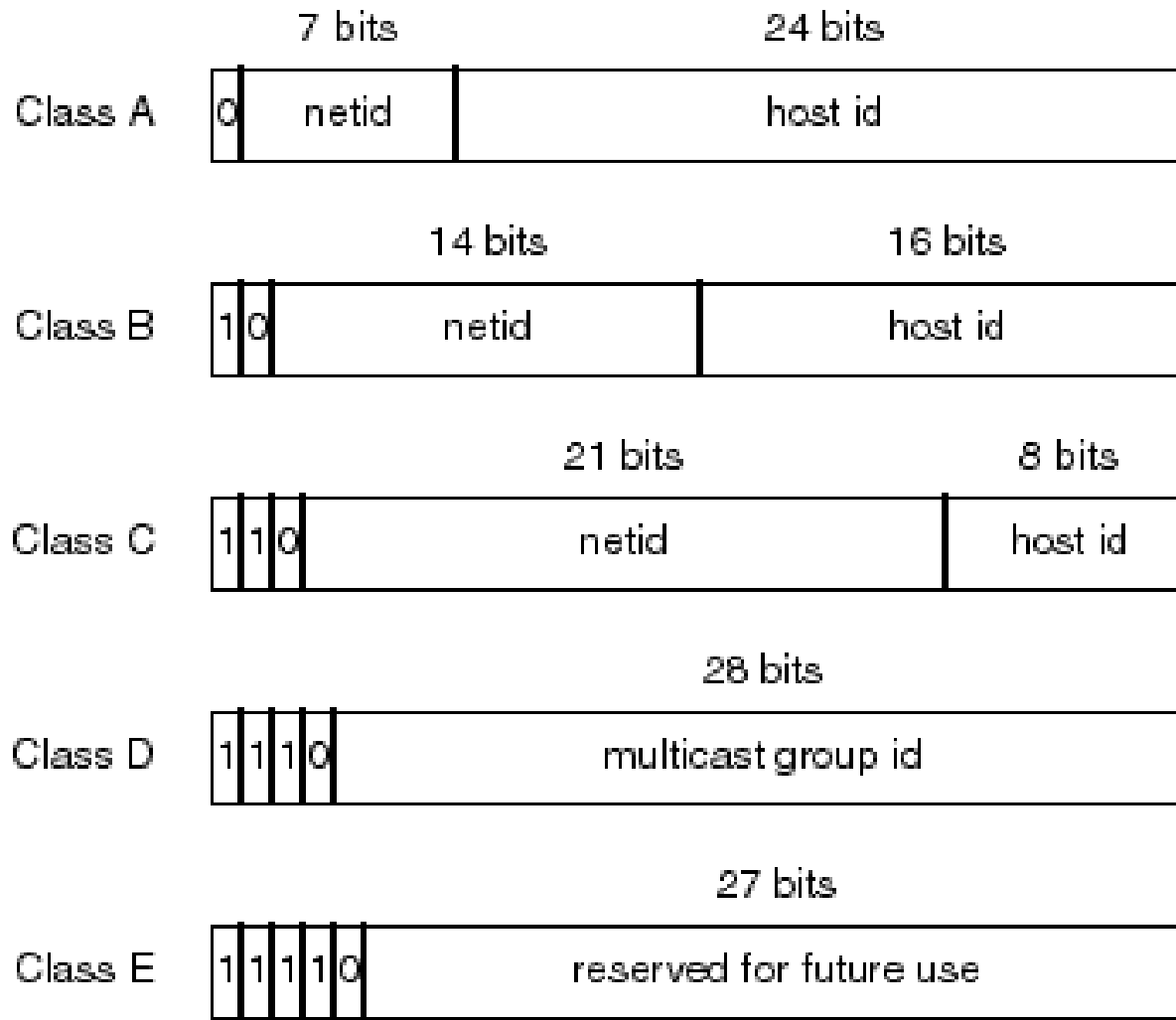
```
#
# ----- Subroutine database_error
#
# Simple error routine.
#

sub database_error {
    &return_error(500, "Database Error", "Sprite (Insert) database error.");
}
```

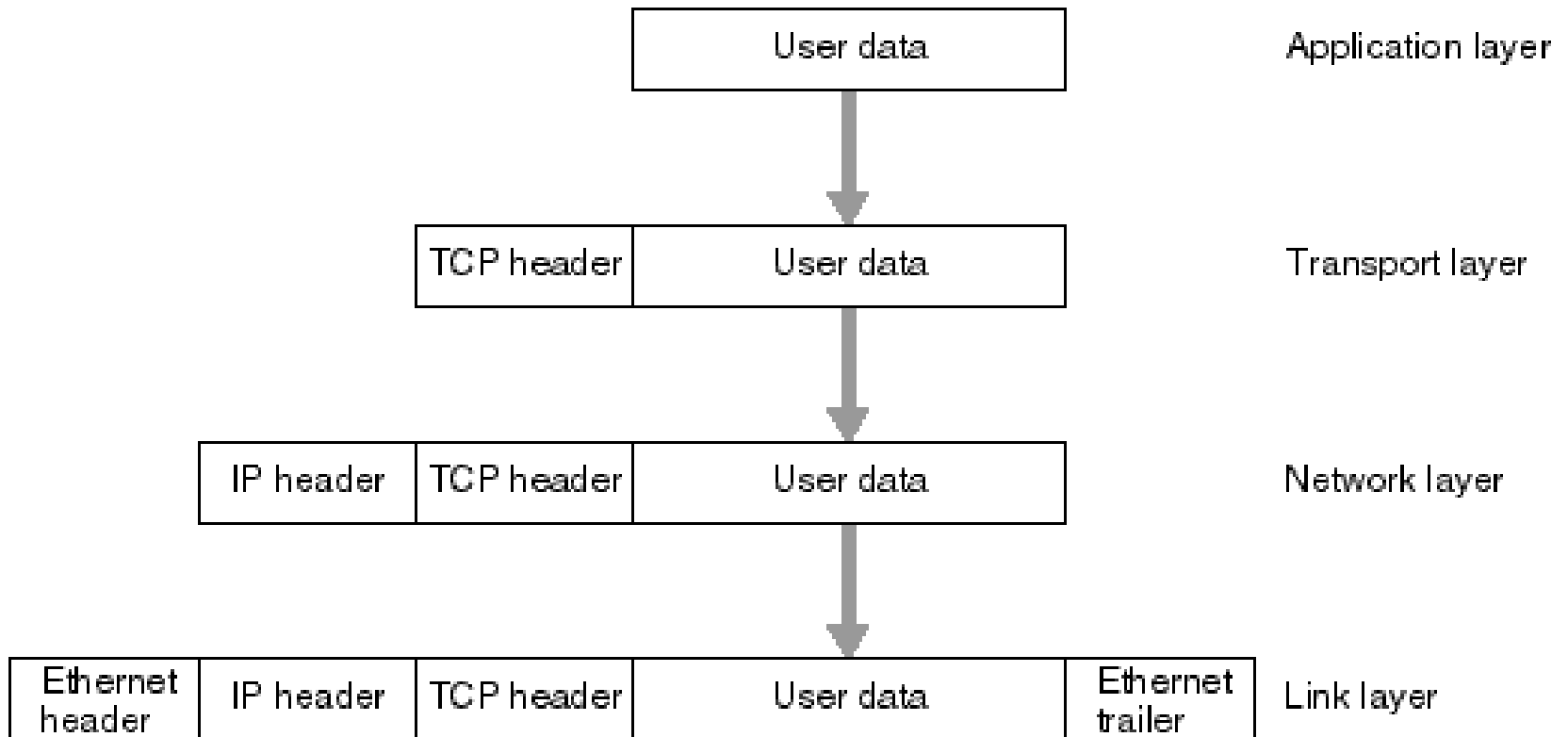
# Layering in the TCP/IP Protocol Suite



# Classes of IPv4 Addresses



# Data Encapsulation



# Simple Client

```
#!/usr/bin/perl -w

# Source: page 349 of "Programming Perl", Larry Wall, Tom Christiansen, and
# Randal L. Schwartz, O'Reilly, (1996).

require 5.002;
use strict;
use Socket;
my ($remote, $port, $iaddr, $paddr, $proto, $line);

$remote = shift || 'localhost';
$port = shift || 2345; # random port
if ($port =~ /\D/) {$port = getservbyname($port, 'tcp')}
die "No port" unless $port;
$iaddr = inet_aton($remote) or die "no host: $remote";
$paddr = sockaddr_in($port, $iaddr);

$proto = getprotobyname('tcp');
socket(SOCK, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
connect(SOCK, $paddr) or die "connect: $!";
while (defined($line = <SOCK>)) {
    print $line;
}

close (SOCK) or die "close: $!";
exit;
```

# Simple Server

```
#!/usr/bin/perl -Tw

# Source: page 349 of "Programming Perl", Larry Wall, Tom Christiansen, and
# Randal L. Schwartz, O'Reilly, (1996).

require 5.002;
use strict;
BEGIN { $ENV{PATH} = '/usr/ucb:/bin' }
use Socket;
use Carp;

sub logmsg {print "$0 $$: @_ at ", scalar localtime, "\n" }
my $port = shift || 2345;
my $proto = getprotobyname('tcp');
socket(Server, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
setsockopt(Server, SOL_SOCKET, SO_REUSEADDR, pack("l", 1)) or die "setsockopt: $!";
bind(Server, sockaddr_in($port, INADDR_ANY)) or die "bind: $!";
listen(Server, SOMAXCONN) or die "listen: $!";

logmsg "server started on port $port";
my $paddr;
$SIG{CHLD} = \&REAPER;
for ( ; $paddr = accept(Client, Server); close Client) {
    my($port, $iaddr) = sockaddr_in($paddr);
    my $name = gethostbyaddr($iaddr, AF_INET);
    logmsg "connection from $name [", inet_ntoa($iaddr), "] at port $port";
    print Client "Hello there, $name, it's now ", scalar localtime, "\n";
}
```

# Multi-Threaded Server

```
#!/usr/bin/perl -Tw
# Source: page 350 of "Programming Perl",
# Larry Wall, Tom Christiansen, and
# Randal L. Schwartz, O'Reilly, (1996).

require 5.002;
use strict;
BEGIN { $ENV{PATH} = '/usr/ucb:/bin' }
use Socket; use Carp; use FileHandle;

sub spawn; #forward declaration
sub logmsg {print "$0 $$: @_ at ", scalar
localtime, "\n" }
my $port = shift || 2345;
my $proto = getprotobyname('tcp');
socket(Server, PF_INET, SOCK_STREAM, $proto)
or die "socket: $!";
setsockopt(Server, SOL_SOCKET, SO_REUSEADDR,
pack("l", 1)) or die "setsockopt: $!";
bind(Server, sockaddr_in($port, INADDR_ANY))
or die "bind: $!";
listen(Server,SOMAXCONN) or die "listen: $!";
logmsg "server started on port $port";
my $waitedpid = 0; my $paddr;

sub REAPER {
$waitedpid = wait;
$SIG{CHLD} = \&REAPER; # if you don't have
# sigaction(2)
```

```
logmsg "reaped $waitedpid" . ($? ? " with
exit $?" : ""); }
$SIG{CHLD} = \&REAPER;
for ( ; $paddr = accept(Client,Server); close
Client) {
my($port,$iaddr) = sockaddr_in($paddr);
my $name = gethostbyaddr($iaddr,AF_INET);
logmsg "connection from $name [",
inet_ntoa($iaddr), "]" at port $port";
spawn sub {
print Client "Hello there, $name, it's now ",
scalar localtime, "\n";
exec '/usr/games/fortune' or confess
"can't exec fortune: $!"; };}

sub spawn {
my $coderef = shift;
unless (@_ == 0 && $coderef &&
ref($coderef) eq 'CODE') {
confess "usage: spawn CODEREF";} my $pid;
if (!defined($pid = fork)) {
logmsg "cannot fork: $!"; return; }
elsif ($pid) { logmsg "begat $pid";
return;} # I'm the parent
# else I'm the child -- go spawn
open(STDIN, "<&Client") or die "can't dup
client to stdin";
open(STDOUT, ">&Client") or die "can't dup
client to stdout";
STDOUT->autoflush(); exit &$coderef();}
```