

# Final Exam Info

---

The final exam is in our regularly scheduled exam time: **Fri June 7th, 8:30-11:30 am. in Terman Auditorium.**

SITN students have the two regular choices: 1) You are welcome to join us in person for the regular reading of the exam. This provides the most authentic and traditional stress/catharsis final exam experience. 2) You may wait at your site, and the your coordinator can administer the exam there Fri afternoon or Mon and fax it back immediately. SITN will have the exam and can fax or email it to the coordinator. The SITN site coordinators should know how to deal with this.

The final exam will cover all of the material we have seen this quarter. Most of the exam will be concerned with coding problems similar to the homeworks. A few of the problems will be essay/short-answer type on material from lecture. The short answer questions may be scored right=+2, blank=0 wrong=-2, in which case you should not guess if you don't know the answer.

The exam will be 3 hours long and will be open book, open note, etc. The exam will be short enough that you will have sufficient time to quickly review and retrieve details from your notes, but there will not be sufficient time to *learn* the material during the exam. To prepare for the exam, you should be familiar with all of the major technologies and issues, and plan to rely on your notes and solutions for details during the exam.

1. Understand all the example code from in lecture.
2. Understand your own solution to each homework. The exam will include smaller but related code-writing problems. I will provide basic interface information where necessary on the exam (e.g. what are the parameters to `doGet()`). Understand the code well enough to write small solutions starting with a blank sheet of paper.
3. Be conversant with the major technologies, issues, and examples presented in lecture.

For code writing problems, we will not be especially picky about syntax or other basically conceptually shallow ideas. Instead, the questions will focus on understanding the key issues for each of the technologies we have covered: TCP/IP, HTML, HTTP, CGI (and to a lesser extent Perl), Java, servlets, JSPs, and JavaScript.

## How To Study For A Code Writing Exam

A good way to study for the code writing questions is to take a coding problem for which you have a solution available (lecture example, homework problem fragment, sample exam problem) and try to write the solution as you would on an exam starting with a blank sheet of paper. Working through writing the code,

even with errors, will give you a much better understanding of the major concepts than a *passive* review of the solution code. I am providing a large number of exam problems you can practice with in this way.

The rest of this handout includes a patchwork of questions from the last few years of exams — I've selected questions which are most similar to this year's coverage. See our web page for a complete repository of old exams (but the older they are, the less relevant they are to our quarter.) Our exam will look a little different since we have covered somewhat different material, but these problems should give you a good idea of what a code writing exam looks like. On the real exam, each question will be on its own page so there will be room for you to write your solutions directly on the exam paper.

## Short Answer

2) Miscellaneous **Short Answer**. 1 word or 1 sentence answers— don't bog down with partial-credit explanations. If you know it great, otherwise leave it blank. There are 16 sub-parts. (25 points)

a) What is the difference between GET and HEAD in HTTP?

GET returns the HTTP header followed by the document. HEAD only returns the HTTP header.

b) Each of the following URL's is href'd in the page  
<http://hell.org/doc/punish.html>

For each URL, give its absolute form to the right...

- i) `/binky.html`
- ii) `ironic/homer.html#doh`
- iii) `donut.gif`
- iv) `/test.html#contents`

*Key points: understand how relative URLs work, realize that #doh is part of the URL.*

- i) *<http://hell.org/binky.html>*
- ii) *<http://hell.org/doc/ironic/homer.html#doh>*
- iii) *<http://hell.org/doc/donut.gif>*
- iv) *<http://hell.org/test.html#contents>*

c) What is the complete, exact string which the HTTP client sends to the HTTP server to request page (iv) above? Give your answer as a double quoted (") string constant as you would in C or Perl...

"GET /test.html HTTP/1.0\r\n\r\n"

Key points: remember how to put together an HTTP request which SiteSucker covered rather extensively.

d) How does an HTTP client know the type of the data returned by the server?

The Content-Type: field in the HTTP header returned by the server.

i) Most of our current Internet technology is based around public standards which have grown to have high quality implementations. Historically, many proprietary standards have not been widely adopted or well implemented. Why do public standards tend to eventually have better implementations?

Competition among multiple implementations. The marketplace of multiple implementors drives the quality of the implementations available up. There's a related virtuous cycle — because people expect the public standard to be better supported, they support it.

k) What are the two main things encoded in an IP address?

The host # and its network #.

l) Bob Metcalfe and Larry Wall are two very influential people for the content that has become CS193i. Which one of them invented Ethernet?

Bob Metcalfe. A little additional research suggests that there's an 'e' at the end of Metcalfe, although both spellings are used widely on the net!

m) What prevents IP datagrams from getting stuck in "infinite loops" in the Internet — forwarded from X, to Y, to X, to Y...

The Time-To-Live counter in each datagram. "Routers" or "TCP/IP" were not sufficiently specific.

n) True or False. When an IP router receives an IP datagram, it looks at the IP address of the final destination and figures out all the routers the datagram will need to traverse on its way.

False. It is largely concerned just with the "next hop".

o) Here is some HTML source.

```
<p>This dress <p>exacerbates the genetic betrayal
that is <p> my legacy
```

Please draw how the HTML might render in a browser window which is..

| <----- this wide -----> |

# This dress exacerbates the genetic betrayal that is my legacy

*Key points: realize that whitespace in the HTML source is ignored. Wrap the HTML appearance to the browser width, and put in vertical whitespace (wt) between the three paragraphs. The quote is from the embittered Janeane Garafolo in "Romy and Michele's High School Reunion "*

## 2) Miscellaneous Short Answer (1 or 2 sentence answers)

**a) With the current IP scheme, you cannot move a computer, say, across campus, plug it into a different physical net, and expect it to work. What information is encoded in an IP address and why does that scheme make it difficult to move computers around?**

*The IP address is broken into "net" and "host" numbers. The net number does not necessarily correspond exactly to a physical net, but it is related since it is a function of the overall topology of the Internet. Therefore, moving a computer from one physical net to another will very likely necessitate that it be "in" a different logical IP net number. Routers use the "net" part of an IP address to guide their routing decisions in real time. This is a basic tradeoff in the IP addressing scheme— encoding net numbers in the IP addresses makes routing easier, but it makes the IP addresses less flexible, and needing a new IP address when your computer changes nets is one example.*

**bc) CGI is a standard. Roughly speaking, what process does it control and what 3 software entities (programs) are involved?**

*CGI (Common Gateway Interface) mediates the dialog between three programs: an HTTP client, an HTTP server, and a CGI compliant program on the server. Typically the client sends a request, the HTTP server fields the request, maps the request to the CGI script, and sends the output (probably HTML) back to the client.*

**d) Name an advantage of client/server computing over the mainframe/terminal model.**

*Many possible answers. Many people quoted from lecture "computing power at client end allows a more responsive /*

*facile interface". "Can potentially exploit computing power at client end." "Allows efficient division of problem between client and server." Marginal answers included "less network bandwidth required" or "allows centralized resources" since neither of these contrasts with the mainframe/terminal model.*

**e) Name an advantage of client/server computing over the application+documents-all-on-the-local-machine model (e.g. Macintosh, Windows...).**

*Many possible answers. The most popular answer (from lecture somewhere no doubt) "allow centralization of things which should be centralized." Most answers included phrases like "centralization" "avoid duplication" "avoid synchronization problems" "don't have local copies of things" "exploit expensive resources available only on the server".*

### 3) CGI Coding

For this problem, you want to write a CGI script which allows users to pick a set of pictures to see. The CGI Perl code will use the same structure and the 193icgi.pl library as the homework.

You only need to write the "middle" part of the script which deals with generating and responding to the form. The context is..

- 1) (As on the homework) The %pairs associative array has been built by the 193icgi.pl library and the "content type: text/html" has already been written. Your code needs to write the entire HTML response. You do not need to deal with error cases gracefully.
- 2) There is an array called @names which contains the human readable names of a bunch of image files. e.g. ("Mona Lisa", "Hans Gruber", "Guernica", ...
- 3) There is an array called @urls, the same length as @names, which contains the corresponding URL for each of the images in @names.

a) Form case. For the case that the script is invoked without arguments, write the code which will generate a form page with the title "Image Picker" containing a single submit button and an unordered list containing a checkbox for each image in the @names array. Each list entry should look like...

```
<li><input name = "Mona Lisa" type = checkbox>Mona Lisa
```

```
# keep HTML header and trailer in vars (used below)
$html_head = << END_TEXT;
<html>
<head><title>Image Picker</title></head>
<body><h1>Image Picker</h1>
```

```

END_TEXT

$html_tail = "</body></html>";

*** Errors...
*** not having <form ...></form>
*** not having <input type=submit>
*** not having <ul></ul>

# Here the form HTML is in a var, but you could do it in print stmts

$form_head = << END_TEXT;
<form method=post> # as on the HW, could have URL or not
Please pick images to view:
<p>
<ul>
END_TEXT

$form_tail = << END_TEXT;
</ul>
<input type=submit>
</form>
END_TEXT

#### The main part ####
# "if" can be assumed...
if(scalar(%pairs) == 0) {
    # Invoked with no arguments. Return a form.
    print $html_head;
    print $form_head;

    *** Some people tried to put this loop inside a string!
    foreach $name (@names) {
        print "<li><input name = \"$name\" type = checkbox>$name\n";
    }
    print $form_tail;
    print $html_tail;
    exit(0);
}

b) Response case. Write the code which triggers when the part (a) form is submitted by the user. Each checked checkbox will create an entry in the %pairs associative array with the name as the key and the value "on". Unchecked checkboxes do not add anything in the associative array. Generate an HTML document titled "Selected Images" which embeds all of the checked images using the <img> tag. Use the values in the @urls array to get the url for each image. The HTML for each image should look like...

<p>
<img source-specifier><br>
image-name

```

```

# $html_tail is from part a).

$html_head2 = << END_TEXT;
<html>
<head><title>Selected Images</title></head>
<body><h1>Selected Images</h1>
END_TEXT

# This is the essentially "else" part of part a).

print $html_head2;

# There is more than one way to do this.
# An alternate way is to loop through the %pairs, but you will need a
# way to find the correct index into @names and @urls.
foreach $i (0..scalar(@names)-1) {# (or could write as a while loop)
    if ($pairs{$names[$i]} eq "on") {
        # or if($pairs{$names[$i]}) {
        # or if(defined($pairs{$names[$i]})) {
        # or if($pairs{$names[$i]} =~ /^on$/i) {

            *** Errors...
            *** not having "src="
            *** assuming @urls is an associative array
            *** not considering that items in %pairs might not be in the
            # same order as in @names or @urls.
            print "<p>\n<img src=$urls[$i]><br>\n$names[$i]\n";
        }
    }
}

print $html_tail;

exit(0);
}
}

```

## 1) TCP/IP and HTTP (25 points)

First, here is some reminder code for the socket interfaces you may want to consult for this problem. You do not need to check the error returns for reading and writing — just for hostname lookup and connecting.

### C Socket Code

```

struct hostent *hostInfo;
struct sockaddr_in inetAddr;
int sock;

inetAddr.sin_family = AF_INET;
inetAddr.sin_port = <port num>;
hostInfo = gethostbyname(<hostname>); // returns NULL on err
memcpy(&inetAddr.sin_addr, hostInfo->h_addr, hostInfo->h_length);
sock = socket(PF_INET, SOCK_STREAM, 0);
connect(sock, &inetAddr, sizeof(inetAddr)); // returns -1 on err

```

once the socket is setup, you can read and write with...

```
int write(int sock, char* buff, int buffLen); // you may ignore the return value
int read(int sock, char* buff, int buffLen); // returns 0 on EOF
```

## Perl Socket Code

```
$ipaddr = inet_aton(<hostname>); # returns false on error
$sockaddr = sockaddr_in(<port num> , $ipaddr);
socket(SOCK, PF_INET, SOCK_STREAM, 0);
connect(SOCK, $sockaddr); # returns false on error
```

Once connected, the socket can be read with <SOCK> and written to with:  
 print SOCK <string>;

For this problem, you need to write a small program which retrieves a series of web pages. Standard input (STDIN) to the program will consist of a series of absolute (not relative) HTTP URL with the leading "http://" omitted. Each URL ends with a complete path+file beginning with a single '/'. For simplicity, the path+file will not be the empty string and will not have a suffix. So the input will look like...

```
cse.stanford.edu/class/
www.whitehouse.gov/pets/socks.html
www.yahoo.com/
```

...  
 The program should print each URL to standard output and then try to do a single HTTP GET for the URL. If the hostname is bad or the connection fails, then print the error message "ERROR hostname" or "ERROR connection" after the URL with no further processing required for that URL. These are the only two error cases which the program must consider. Otherwise, if the connection is successful, the program should read all of the header and body of the HTTP response and simply print it all out as returned by the server. In any case, the program should then continue processing URLs until the input is exhausted. You may build your solution in C or Perl (just as on the homework), but you should not use any HTTP specialized libraries other than Socket. You do not need to worry about decomposition or style of your solution — we will grade only on the proper functioning of your code.

### Solution notes:

Read each line from standard input with <STDIN>

Parse each line with a regular expression such as `m"(.?)(/?.*)" == $1` is the host, `$2` is the path

Try to connect to the host on port 80

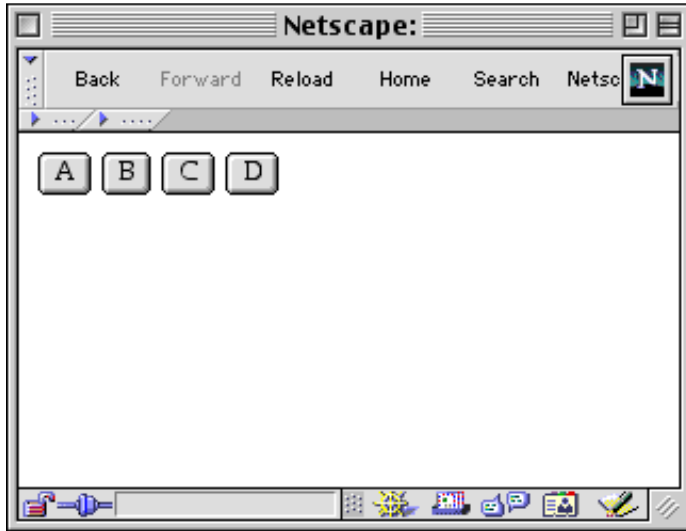
Send the GET request as `"GET $path HTTP/1.0\r\n\r\n"`

Use the standard while (defined(\$line = <SOCK>)) { ... to read all the lines returned from the server

### 3. CGI (30)

#### a. Simple Password CGI

For this problem, you will write a simple password CGI where the user clicks in the secret code to gain admission. The CGI should return a form that looks like this...



The user can press buttons, and the form returned looks the same every time. The user can press as many buttons as they like. After any number of wrong buttons, when the user finally enters the real button sequence "cab", then the form should return the "success" page...



Use hidden fields to keep track of the user input to detect when they have entered the "CAB" sequence. You may ignore error cases.

```
#!/usr/bin/perl
##
# 3a. Simple Password CGI
#
# (8 points total)
#
# There were two basic strategies to this problem. Either the hidden
# field stores a string of all the previous clicks, or it stores some
# indicator that says what portion of a correct password has already been
# clicked (e.g., hidden = 1 if the previous click was "c", 2 if the
# previous two clicks were "c" then "b", etc). The first solution is
# given below.
#

use CGI;
$query = new CGI;

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

$next = $query->param("next");
if (!defined($next)) {$next = "";}
$past = $query->param("past");
if (!defined($past)) {$past = "";}

$past = $past . $next;

$html = <<EOT;
<form action= CS193Iq3a.pl method=get>
<input type=submit name=next value = A>
<input type=submit name=next value = B>
<input type=submit name=next value = C>
<input type=submit name=next value = D>
<input type=hidden name=past value=$past>
</form>
</body></html>
EOT

$youwin = <<EOT;
<h1>You Win!</h1>
</body></html>
EOT

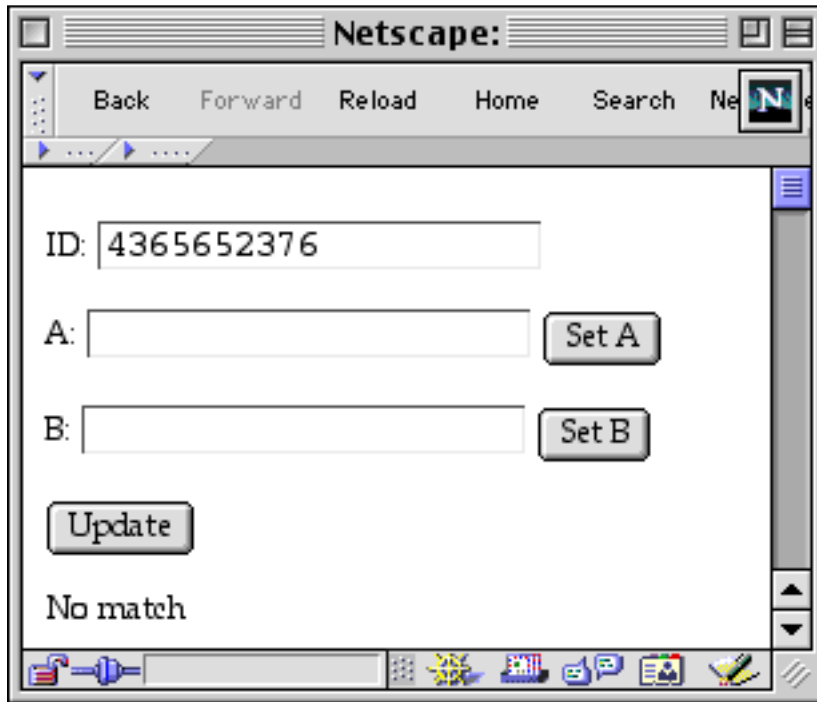
if ($past =~ m/cab/i) {
    print $youwin;
}
else {
    print $html;
}

```

## b. Shared Secret CGI (difficult)

Suppose there are two people who are on the phone to each other. They know a shared secret, but they do not know that the other person is who they say they

are. Each person needs to verify that the other person also knows the secret. This CGI will allow them to verify that they both know the secret.



The first connection to the CGI, returns the form with a random ID.  
The user may change the ID, or use the generated one. Use `rand()` to generate a large random ID number.

The user may use the Set A or Set B buttons to send the A or B string to the server where it is stored associated with the ID. On the server side, the CGI keeps a table where it stores the current A and B string for each ID. The A and B strings will not contain whitespace.

The Update buttons does not change A or B, it just updates the "match" string.

If both A and B have been set for this ID, and the two strings are the same, then the match string will be "Match". In all other cases, it will be "No match".

This should work for two different users on two different machines, so long as they use the same ID number. One user can set the A string, one user can set the B string, and they can both see if there is a match.

Multiple pairs of people should be able to use the CGI at the same time, so long as their IDs are different.

As with HW3, the CGI will need to use a file to store the id/A/B state between requests. Use the format `id\tA\tB\n` to store each id with its A and B strings on

a line. You may ignore all error cases. Use `open(F, "id.txt")` to open for reading, and `open(F, ">id.txt")` to open for writing. We will not worry about file locking.

```
#!/usr/bin/perl

##
# 3b. Shared Secret CGI
#

use CGI;
$query = new CGI;

if (!defined($query->param("id"))) {
    $id = rand();
    $id = "$id";
}
else {
    $id = $query->param("id");
}

$match = "No match";

$html = <<EOT;
Content-type: text/html
<html><body>
<form action=CS193Iq3b.pl method=get>
<p>ID: <input type=text name=id value=$id size=20>
<p>A: <input type=text name=passa size=20>
<input type=submit name=button value="Set A">
<p>B: <input type=text name=passb size=20>
<input type=submit name=button value="Set B">
<p><input type=submit name=button value="Update">
EOT

if (!defined($query->param("button"))) {
    print $html;
    print "<p>$match</body></html>";
}

else {
    open(F, "id.txt");
    @lines = <F>;
    close(F);

    $button = $query->param("button");
    $passa = $query->param("passa");
    $passb = $query->param("passb");

    ## the clicked a button -> update the file
    if ($button =~ m/Set\s([AB])) {
        $set = 0;
        foreach $line (@lines) {
            @arr = split(/\t/, $line);
            chomp($arr[2]);
            if ($arr[0] eq $id) {
                if ($button eq "Set A") {
                    $arr[1] = $passa;
                }
            }
        }
    }
}
```

```

    }
    else {
        $arr[2] = $passb;
    }
    $line = $arr[0] . "\t" . $arr[1] . "\t" . $arr[2] . "\n";
    if ($arr[1] eq $arr[2]) {$match = "Match";}
    $set = 1;
    last;
}
}
if ($set==0) {
    unshift(@lines, "$id\t$passa\t$passb\n");
}

open(F, ">id.txt");
foreach $line (@lines) {
    print F $line;
}
}
else {
    open(F, "id.txt");
    @lines = <F>;
    close(F);

    foreach $line (@lines) {
        if ($line =~ m/(.*?)\t(.*?)\t(.*?)\n/) {
            if ($id eq $1) {
                if ($2 eq $3) {
                    $match = "Match";
                    last;
                }
            }
        }
    }
}
print $html;
print "<p>$match</body></html>";
}

```

## 4) Servlet Programming (20 points)

(Note: the quarter this was given, the HW involved reading from a file. In a quarter where students did not do file reading on a homework, I would not ask for file reading on the exam).

For this problem, you will write a servlet that displays the text of books, where the user can censor out bad words. The servlet returns a page with three parts...

### 1) List Of Books

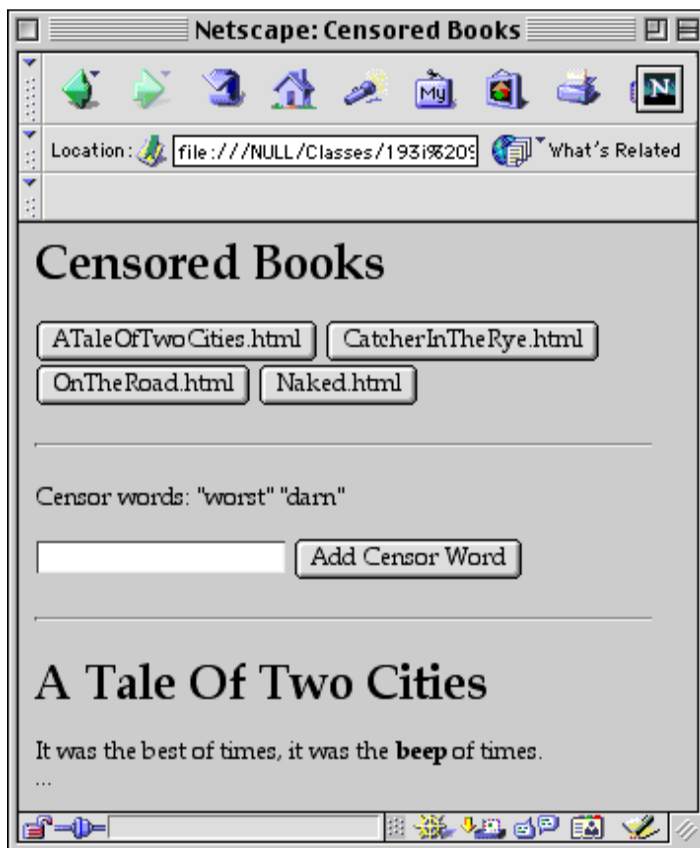
The top part of the page displays buttons, each of which can be used to choose a book. There's a Vector (below) that contains the book filenames which are used as the button titles.

## 2) Censor Section

The servlet keeps a list of words that it censors from the text of the books. The Censor section shows the words currently in the censor list. There is a text field and an Add button that allow the user to add a word to the list. Adding a word should return the same page, but with the word added to the list. Words can be added to the censor list, but they cannot be removed. Changing from one book to another, does not change the list of censored words. The list of censor words is used for the duration of the user's interaction with the servlet. Adding a censor word should not change which book is displayed in section (3) below.

## 3) Censored Book

If a book has been chosen, its HTML is pulled from its file and shown in the bottom of the page, except, any occurrence of a censored word in the HTML is replaced with a **beep**. Assume the file contains valid HTML that can be spliced right into the servlet HTML output. In this example, the words "worst" and "darn" have previously been added to the censor list. The servlet is currently showing the text of the book ATaleOfTwoCities.html. The opening sentence of this book begins "It was the best of times, it was the worst of times, ...". The word "worst" has been changed to **beep** since it is on the censor list.



## Code

You may assume that the following methods are already defined in your servlet for your convenience. Your code does not need to deal with exceptions or try/catch blocks at all (if you know what those are). Your code does not need to recognize or deal with any error conditions.

- 1) `Vector getBooks()` returns a `Vector` of `Strings` of the file names of the available books.
- 2) `FileReader openFile(String fileName)` returns an opened `FileReader` object for the given filename. We will assume this operation always succeeds.
- 3) This code will read each line of a file as a `String`...

```
String line;
FileReader reader = openFile(fileName);
while ((line = reader.readLine()) != null) {
    // do something with line
}
```

- 4) Assume that the following message has been defined: `String superReplace(String s, String old, String replacement)` that replaces all occurrences of the substring `old` in `s` with the substring `replacement`, and returns the new resulting `String`.

```
// Censor
// (assume the import directives are all done for you)
public class Censor extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

// a) Generate the top book buttons (leave the session for part b)

        out.println("<html><head><title>Censored Books</title></head><body>");
        Vector files = getBooks();
        int i;
        out.println("<form>");
        for (i=0; i<files.size(); i++) {
            String fname = (String) files.elementAt(i);
            out.println("<input type=submit name=book value=\"\" + fname + \"\"");
        }
        out.println("</form>");

// b) Deal with the session object. Detect if they are adding a word
// to the censor list and update the list

        Vector censor;

        HttpSession session = request.getSession(false);

        if (session==null) {
```

```

    censor = new Vector();
    session = request.getSession(true);
    session.setAttribute("censor", censor);
}
else {
    censor = (Vector) session.getAttribute("censor");
}

String word = request.getParameter("word");
if (word!=null and !word.equals("")) {
    censor.addElement(word);
}

// c) Echo the current state of the censor list and generate the
// form for adding censor words

out.println("<p>Censor words...");
for (i=0; i<censor.size(); i++) {
    String s = (String)censor.elementAt(i);
    out.println("\\" + s + "\\");
}

out.println("<p><form><input type=text name=word>");
out.println("<input type=submit value=\"Add Censor Word\"");

// d) If a book has been chosen, open its file, echo the HTML
// in the file, doing the censor replacement

String book = request.getParameter("book");
if (book!=null && !book.equals("")) {
    FileReader reader = openFile(book);
    String line;
    while ((line = reader.readLine()) != null) {
        int j;
        for (j=0; j<censor.size(); j++) {
            String cWord = (String)censor.elementAt(j);
            line = superReplace(line, cWord, "<b>beep</b>");
        }
        out.println(line);
    }
}
}

```

### 1) Short Answer (30 points)

These 19 little questions have simple, short answers — no more than 6 words, and often just one word. These are worth 1 or 2 points if correct, 0 point if left blank, and -1 or -2 if answered incorrectly. The 2 points questions are marked with a "(2)"; all the others are 1 point. Your best strategy is to go through quickly and answer the ones you know immediately, and leave the others blank. Crossing out 100% of your answer is equivalent to leaving the question blank. Your overall score for this question cannot go below 0.

Perfect answers got +1 or +2, wrong answers got -1 or -2. In rare cases, a slightly wrong answer would get a -0 instead of a -2.

a. True or false: "For IP communication with Ethernet, the Ethernet packet is transmitted inside of an IP datagram".

*false, it's the other way around*

b. True or false: the first router that gets a datagram computes the "route" -- the sequence of hops for that datagram to reach its final destination.

*false, the IP routers are typically concerned with just the next hop*

c. True or False: An IP datagram whose final destination is a particular computer contains the IP address of that computer.

*true. If people mentioned weird cases like "NAT" or "broadcast" we allowed true or false since in that case the phrase "address of the final destination" is ambiguous.*

For these three questions, suppose the Perl variable SOCK has just been connected to port 80 on the foo.com HTTP server, and you are initiating an HTTP1.0 connection.

*(The answers here depended on familiarity with the HTTP protocol and what parts of a URL are sent.)*

d. (2) Write a print statement that will initiate the retrieval of the document `http://foo.com/doc/bar.html`

```
print SOCK "GET /cgi-bin/doc/bar.html HTTP/1.0\r\n\r\n";
```

e. (2) Write a print statement that will initiate the retrieval of the document `http://foo.com/cgi-bin/a.pl?pi=3`

```
print SOCK "GET /cgi-bin/a.pl?pi=3 HTTP/1.0\r\n\r\n";
```

f. (2) Write a print statement that will initiate the retrieval of the document `http://foo.com/cgi-bin/a.pl#pi=3`

```
print SOCK "GET /cgi-bin/a.pl HTTP/1.0\r\n\r\n";
```

For these two questions, assume you are parsing HTML retrieved from the URL `http://foo.com/a/b.html`

g. (2) What is the full URL for `<a href=c.html>`

```
http://foo.com/a/c.html
```

h. (2) What is the full URL for `<a href=docs/c.html>`

```
http://foo.com/a/docs/c.html
```

i. True or false: when using the Location: redirect in a header, the HTTP server includes the text of the new destination page in the body section after the header.

*false*

j. (2) How does an HTTP client know that the HTTP server is sending back HTML data as opposed to, say, JPEG data?

*The "content-type" header field*

k. How is an HTTP cookie sent from the HTTP client to the HTTP server?

*as a field in the **request** header*

l. (2) What are the three reasonable techniques that an HTTP server application can use to give the "session" illusion of continuity across a sequence of pages?

*hidden fields, url re-writing, cookies*

m. (2) If a JSP sets a cookie on the response after having printed out all the HTML, the cookie does not work (this is why we do cookies before doing anything else in a JSP). Why?

*The cookie is transmitted on the HTTP **response**. Thus setting the cookie after the response has been sent does not work. Full credit required some mention of the time-ordering which is at the core of this problem.*

n. Why are servlets faster than CGI's (for the standard implementation of CGI)?

*Servlets are typically already loaded and running in RAM when the request comes in. (note: "servlets run in RAM" was not sufficient -- **everything** runs in RAM if you think about it).*

q. (2) Here is some HTML source....

```
<table border=1>
<tr>
  <td>
    <p> a
      b c d
    </td>
  <td>
    <p> a b
      c d
    </td>
</tr>
</table>
```

Please draw how the HTML table might render in the browser...

*The two "a b c d" cells should look the same. In particular, the placement of the \n's in the source has no effect at all.*

r. What is a "replay attack" and what is a simple way to guard against it?

*Record a message, and then replay it to the original recipient later. Solution: include serial numbers or other redundancy in each message. (Note: encryption is not sufficient -- the replay attack still works).*

s. What is one simple operation that a "firewall" does?

*A firewall blocks certain types of access across it -- typically disallowing certain port numbers in certain directions. "Separates two nets" was not sufficient -- a simple router does that.*

## 2. TCP/IP (25 points)

For this problem, you will write some client Perl code for the Binky protocol.

For this problem, a "binky id" is a hostname followed by a port number in angle brackets — "sunburn.stanford.edu<45>" or "yahoo.com<9000>". The port number and brackets may be omitted, in which case the port is assumed to be 100, so binky id "foo.com" would use port 100.

The paragraph below describes overall computation. Your solution will be divided into four little parts, and each part repeats the description of what it is supposed to do.

The file "binky.txt" contains binky id's, one per line. For each binky id in the file, do the following processing: connect to the host and port number and write "out\n" to the server. The server response will have two parts, (a) a series of "nested" ordinary DNS host names, one per line, followed by a blank line (simple "\n"), followed by (b) HTML text. For each of the nested host names, connect to the given host on port 999, send the request "in\n", and simply read and print out all that it sends without modification. Do this for each of the nested host names. Then print out the blank line and all of the HTML that follows it.

You do not need to include any error handling code -- assume all connections succeed. Also, you do not need to declare your variables or have the correct "use" clauses.

For reference, here is a simple subroutine that will make a socket connection. You may call it from your code, or copy code from inside it.

```
## Open a socket for a given host and port
## Call like this: simpleConnect(SOCK, "www.yahoo.com", 80);
sub simpleConnect {
    my($sock, $host, $port) = @_;

    my($ipaddr) = inet_aton($host);
    my($sockaddr) = sockaddr_in($port, $ipaddr);
    socket($sock, PF_INET, SOCK_STREAM, 0);
    connect($sock, $sockaddr);
}
```

a. Open the file "binky.txt", start the while loop to go through all of its lines, parse each binky id into the variables \$outhost and \$outport (leave the body of the while loop open so it contains parts (b) etc. below). Use 100 as the default port number. You may assume that the binky id's are properly formatted.

```
## Solution notes -- key points
## -open the file
## -do a little parsing to extract the host<port>
## -open the sockets
## -read-loop 'til blank line -or- end-of-file correctly

open(F, "binky.txt");
while(defined($line=<F>)) {
    if($line =~ m/(.+)<(\d+)>/) {
        $outhost = $1;
        $outport = $2;
    } else {
        chomp(line);
        $outhost = $line;
        $outport = 100;
    }
}
```

b. Given \$outhost and \$outport from above, make the connection and send the "out\n" request. Start a while loop that reads each nested hostname into the variable "\$inhost" which will be processed in the next step.

```
simpleconnect(SOCK1, $outhost, $outport);
print SOCK1 "out\n";

## get all the lines before the blank line
while (($inhost = <SOCK>) ne "\n") {
```

c. For each \$inhost, connect using port 999, send the "in\n" request, read and print out all that is sent back.

```
## just connect to inner host and print it all
simpleconnect(SOCK2, $inhost, 999);
print SOCK2 "in\n";
while (defined($x = <SOCK2>)) {
    print $x;
}
}
```

d. When done with the nested hostnames, read and print out the HTML that follows the nested hostnames. Loop back to (a) to continue processing the file.

```
## process the remaining lines
while (defined($x = <SOCK1>)) {
    print $x;
}

## loop back to process the next line
}
```

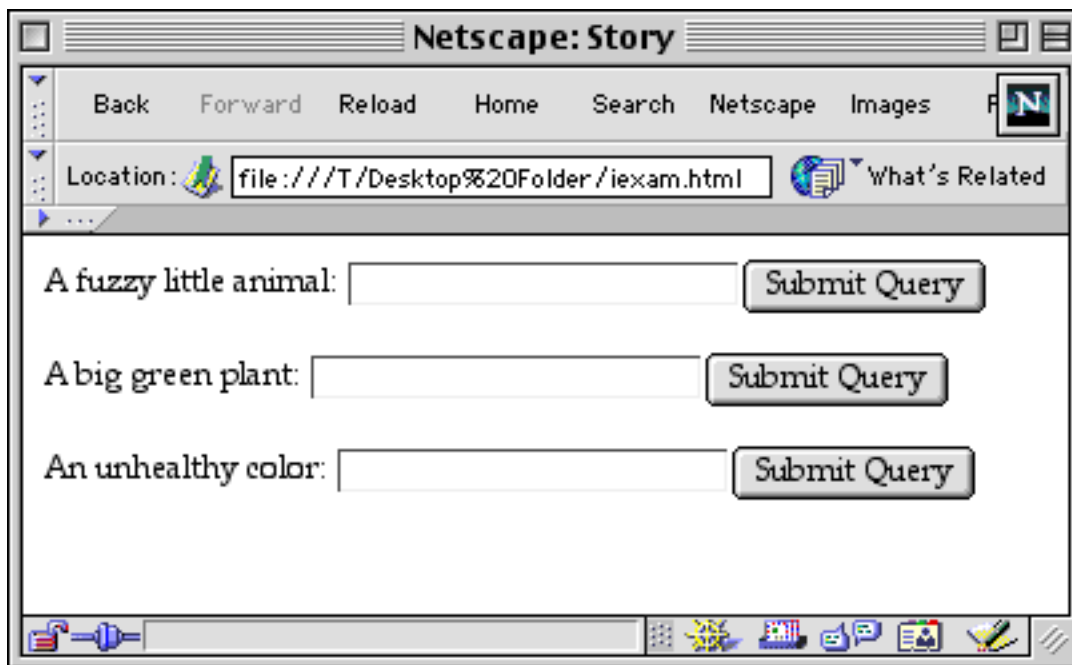
### 3. CGI (25 points)

For this problem, you will use the CGI Perl module to build a simple story-writing application. The file "story.txt" is divided into two parts separated by a blank line ("\n"). The top part is made of lines made of a single "name" word followed by a space, followed by a "description" phrase. The second part of the file is just regular HTML text, except some lines will contain only a single tag using one of the names from the first part...

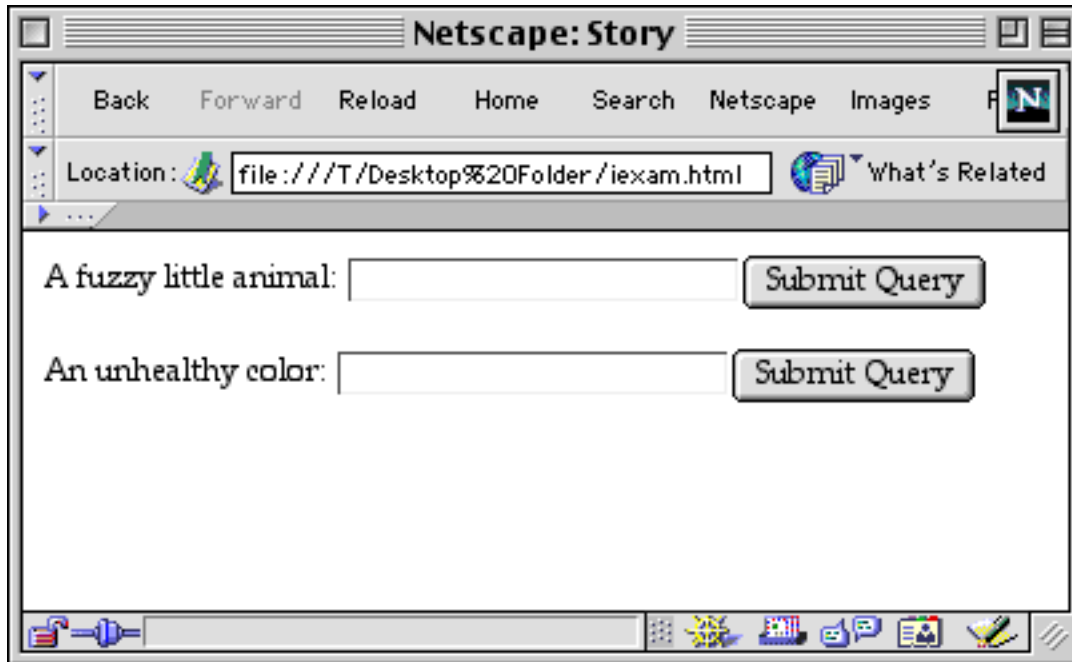
```
animal A fuzzy little animal
plant A big green plant
color An unhealthy color
```

```
<p>
Once upon a time
there was a cute little
<animal>
that enjoyed eating
<plant>
until it turned
<color>
```

When called with no bindings, the CGI produces one input field for each line in the first part of the file. The second part of the file is not used...



The user enters text in one of the fields and clicks the adjacent button to define the text for that name. This leads to a page with that input field no longer present. So entering "fennel" for "A big green plant" above results in the page...



The user will enter text for each field one at a time until all have been entered. The user may enter the fields in any order, however once one has been entered, the user does not get a chance to change it. We will not support the "back" button to go change old decisions — we will only go forward one entry at a time. Part (a) of the problem deals with generating the above pages as the user defines all the fields. Part (b) (below) deals with printing out the story once all the fields have been defined.

The routine header part of the CGI is given here — write your code to follow it. The CGI should re-read the "story.txt" file each time it is run, but should not change file.

```
#!/usr/bin/perl -w
use CGI;

$header = <<EOT;
<html><head>
<title>Story</title>
</head>
<body bgcolor=white>
EOT

$trailer = "</body></html>\n";

$| = 1;    ## set STDOUT to be unbuffered
```

a. Set things up and go through the file "story.txt" and produce form entries for each of the names that has not yet been given any text by the user. If all of the names have been entered, then do not produce any form inputs and execute (b)

below instead (in that case, it's ok to produce a form so long as it does not take up space on screen).

```
## Solution notes
## As you generate the next form, use hidden fields to "remember"
## which things the user has already entered.
## As on the homework, for full-credit the solution needed
## to work for files containing any bindings, not just
## "animal", "plant", and "color"

print "Content-type: text/html\r\n\r\n";
print $header;
print "<form>";
open (FILE, "story.txt");
$edited = 0;
## read until blank line
while (defined ($line = <FILE>) && ($line ne "\n")) {
    $line =~ m/(.*?) (.*)/;
    $name = $1;
    $description = $2;
    ## if defined, keep as hidden field
    if (!defined($query->param($name))) {
        print "$description: <input type=text name=$name> <input type=submit
value=\"Submit Query\">";
        $edited = 1;
    }
    ## otherwise make a text field for it
    else {
        print "<input type=hidden" name=$name value=\"${query->param($name)}\">";
    }
}
if ($edited == 0) {
    # do b
}
print "</form></body></html>";
```

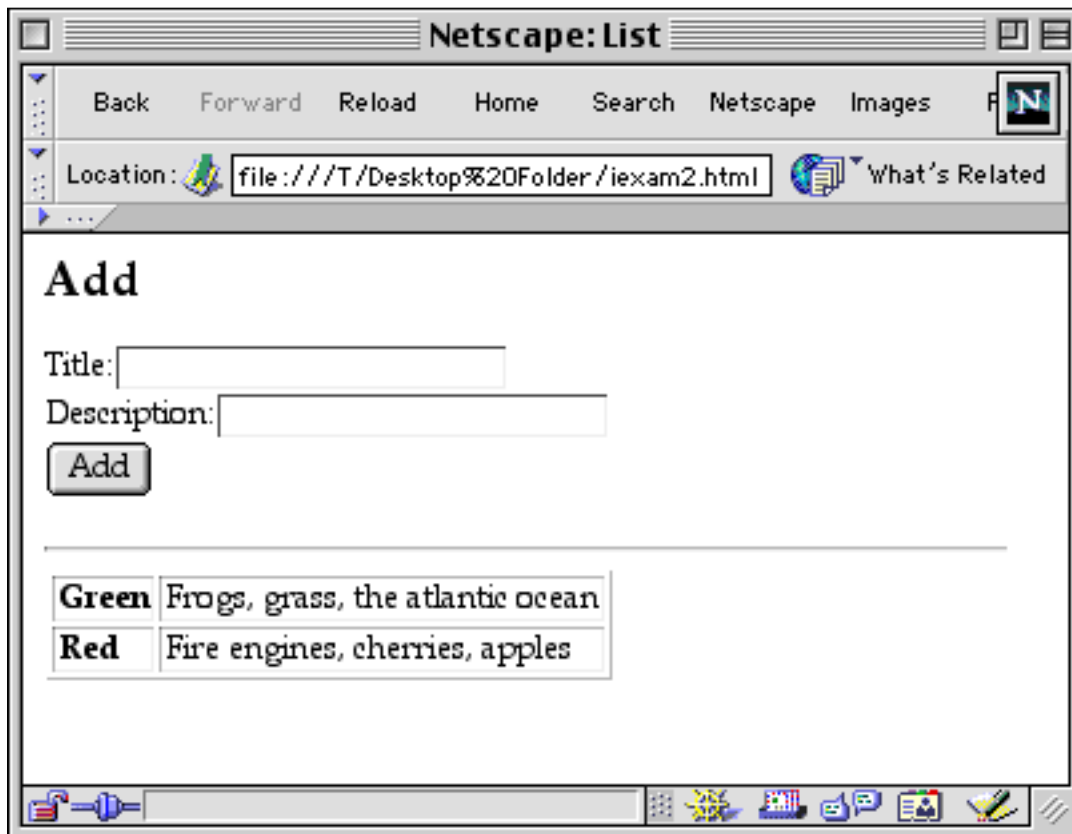
**b.** In the case that all the names have been defined, echo the HTML part of the file, using the text the user entered in place of each <name> tag. The name tags occur on lines by themselves in the HTML. The tags may appear in any order in the HTML and may each be used any number or times.

```
while (defined($line = <FILE>)) {
    if ($line =~ /^<(.*?)>$/) { ## look for a tag
        if (defined($query->param($1))) { ## see if it's defined
            $line = $query->param($1);
        }
    }
    print $line;
}
```

## 2. List Servlet (20 points)

For this problem, you will write a simple list servlet.

The top of the servlet output is always a little form which lets the user add a list item defined by a title and a description. The bottom part of the output shows the current list in a table. The title for each row is in bold. The list is maintained in a server side session, so if the user has two browser windows, they will manipulate the same underlying list. This is a very simple list application — items can be added, but they cannot be deleted or edited. Here's a screenshot after two entries have been added to the list...



There's an additional constraint that the lines inside the table must be generated using a JSP. The JSP will generate the HTML starting with `<tr>` and ending with `</tr>`. The servlet generates everything else. To help with the JSP, you have the `ItemBean` class which represents the "title" and "description" strings in the standard bean way...

```
// stores "title" and "description"
public class ItemBean {
    private String title;
    private String description;

    public ItemBean(String aTitle, String aDescription) {
        title = aTitle;
        description = aDescription;
    }

    public String getTitle() {
```

```

        return(title);
    }

    public String getDescription() {
        return(description);
    }
}

```

**Here is the standard header for a Servlet — write your code to follow it...**

```

import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ListServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {

        // your code starts on the next page

```

**a. Print out HTML and the "add" form. Locate the session object, notice if the add button has been clicked and if so change the session.**

```

## Solution notes:
## -get/create the session
## -use a Vector in the session to store the String pairs
## (putting an ItemBean in the vector works nicely)
## -loop through the vector
## -dispatch to the JSP for each vector element

response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html><head><title>List</title></head>");
out.println("<h1>Add</h1>");

// print the add form
out.println("<p><form>Title:<input type=text name=title>");
out.println("<br>Description:<input type=text name=descr>");
out.println("<br><input type=submit name=add value=Add></form>");

// deal with the session
// get a pointer to the vector
Vector data = null;
HttpSession s = getSession(false);
if (s==null) {
    s = getSession(true);
    data = new Vector();
    s.setAttribute("data", data);
}
else {

```

```

    data = s.getAttribute("data");
}

String title = request.getParamter("title");
String descr = request.getParamter("descr");
if (request.getParameter("add")!=null) {
    // we make an item bean and put it in
    // -or- we could store both strings somehow
    data.add(new ItemBean(title, descr));
}

```

### b. Print out the table, using "item.jsp" (part c) for each row

```

RequestDispatcher rd = getServletContext().
    getRequestDispatcher("/item.jsp");

out.println("<table>");
int i;
// iterate through all the items
for (i=0; i<data.size(); i++) {
    // pull the ith bean off the vector and put it on the request
    request.setAttribute("item",
        data.elementAt(i));
    rd.include(request, response);    // go over to the JSP
}
out.println("</table>");

```

### c. Define "item.jsp"

```

<jsp:usebean id="item" scope="page" class="ItemBean">
<tr>
    <td>
        <b><jsp:getProperty name="item" property="title"></b>
    </td>
    <td>
        <jsp:getProperty name="item" property="description">
    </td>
</tr>

```