

# CS193i Final Exam+Soln

---

Here is the final exam text with the solution code added. The median score on the exam was a 77/100. This was a 2-hour, open book, open note exam. We were not picky about syntax, method names, etc.

## 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).*

o. (2) For this problem, suppose X is the superclass of Y. Is the assignment below a compile time error?

```
X x = new X();
Y y = new Y();
```

```
y = x; // Is this a compile-time error?
```

YES

p. (2) Once again, X is the superclass of Y. What is the output of the Y.demo() method for the code below?

```
class X {
    void foo() {
        System.out.println("X foo");
    }

    void bar() {
        System.out.println("X bar");
        foo();
    }
}

class Y extends X {
    void foo() {
        System.out.println("Y foo");
    }

    static demo() {
        Y y = new Y();
        y.bar();
    }
}
```

*output:*  
*X bar*  
*Y foo*

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)

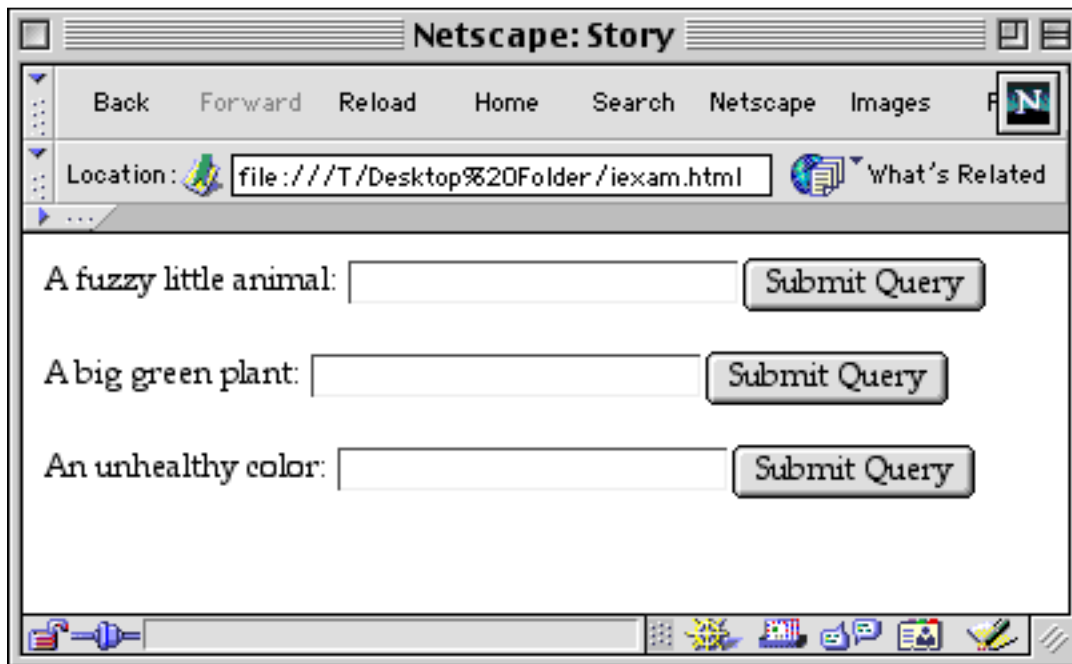
---

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...

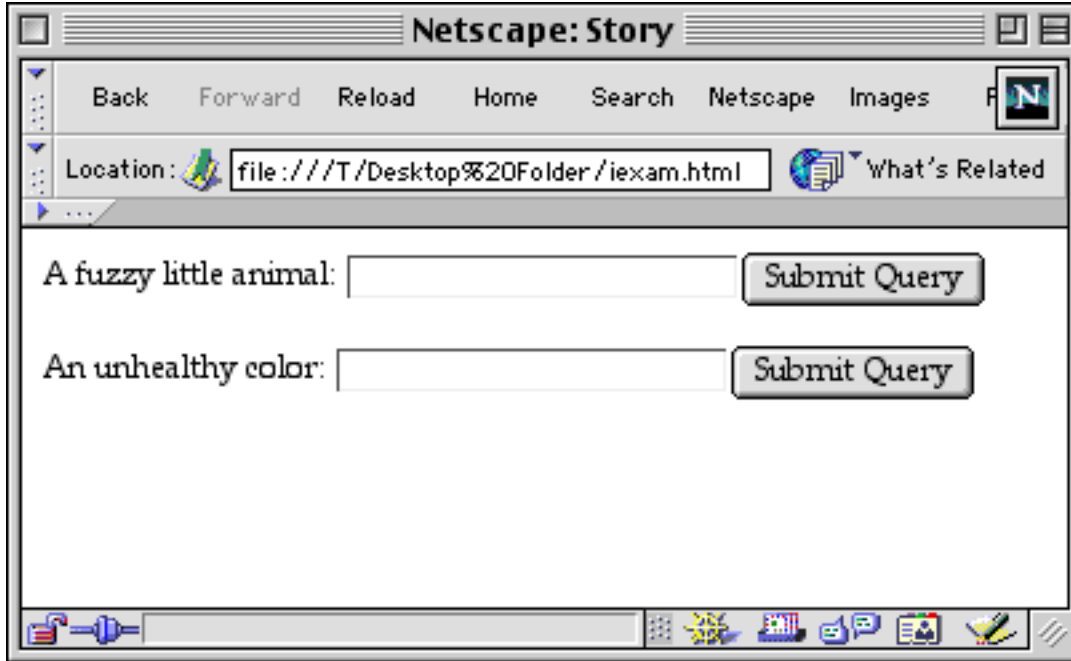
```
animal A fuzzy little animal
plant A big green plant
color An unhealthy color
```

```
<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 of times.

```

while (defined($line = <FILE>)) {
    if ($line =~ /^<(.*?)>$/) {
        if (defined($query->param($1))) {
            $line = $query->param($1);
        }
    }
    print $line;
}

```

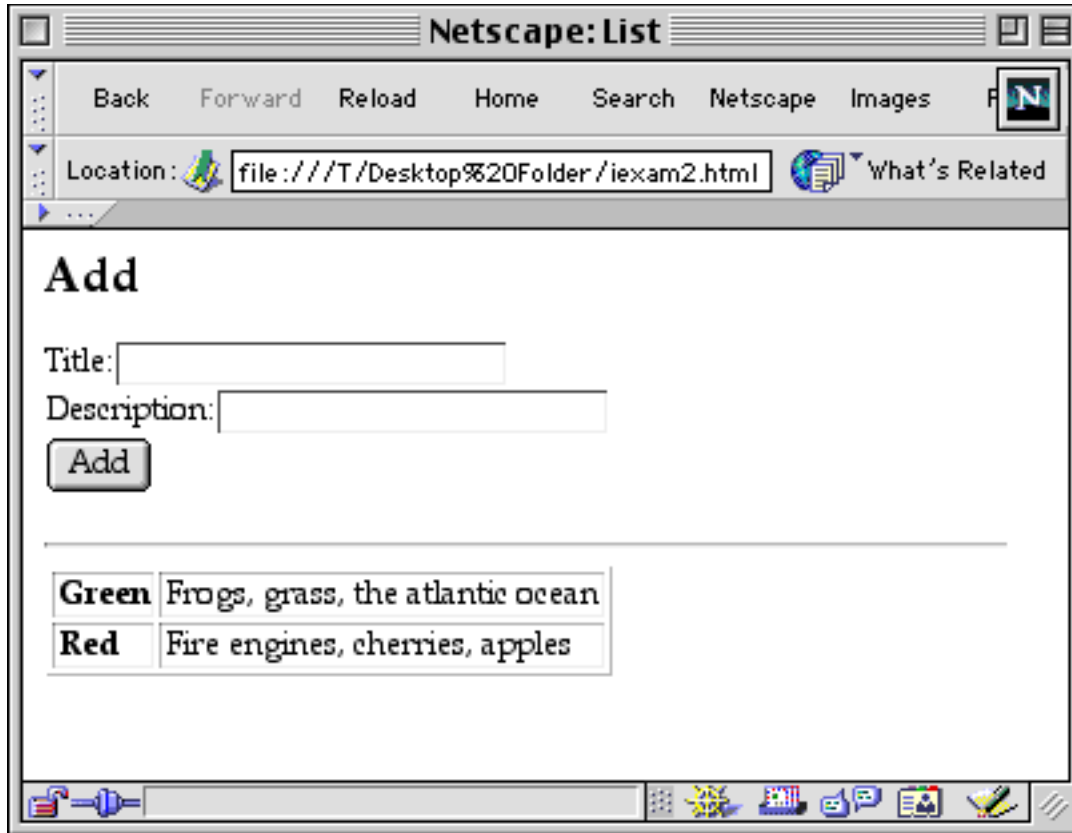
## 2. List Servlet (20 points)

---

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"

```

<% @page language="java" %>
<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>

```