

# CS193i Final Exam

---

- **SITN students** -- hopefully you are getting this soon enough to take it on Tuesday. Please return it as soon as possible. If it's Thursday, and the exam is still at your site, then please fax it back to 650-723-6092. If you also send back your physical exam, write "faxed" on it so we don't grade it twice!
- This is a 2-hour, open book, open note exam. Your exam should have 18 pages.
- **Write only on the front sides of pages** — if you need to add pages, I have brought blank paper and a stapler.
- Syntax, exact method names, etc., are not important so long as the important and interesting points of your solution are clear.
- There is some time pressure. The answers should not need to be especially long — if you find yourself doing something very complex and time consuming, you may want to skip that question.

Good Luck!

Name (Last, First): \_\_\_\_\_  
(please try to print neatly!)

I agree to the letter and spirit of the Stanford honor code — that no illicit aid has been given or received on this exam

Signed: \_\_\_\_\_

1: \_\_\_\_/20 \_\_\_\_

2: \_\_\_\_/30 \_\_\_\_

3: \_\_\_\_/30 \_\_\_\_

4: \_\_\_\_/20 \_\_\_\_

The student's final exam prayer: Now I lay me down to rest. I pray to pass tomorrow's test. If from my sleep I should not wake. That's one less test I'll have to take.

# 1) Short Answer (20 points)

This section is made from a large number of small questions. Short answers are fine. Explanations are not necessary.

- a) When a computer wants to transmit on Ethernet, it first waits for \_\_\_\_\_ .
- b) What is the exact GET request line of text sent to an HTTP server by the client for the URL "http://cse.stanford.edu/class/cs193i"
- c) What is the exact GET request line of text sent to an HTTP server by the client for the URL "http://www-cs-faculty.stanford.edu/dumpform.pl?a=b"
- d) Suppose the page for "http://foo.com/bar/" contains the following ...  
`<a href = "/binky.com/bar.html">`  
 For that link, what HTTP server is contacted, and what is the exact GET request line of text sent to that host?
- e) True or False: All IP datagrams contain the DNS name of their destination?
- f) What does the HTTP "HEAD" request do?
- g) If a client is retrieving a message from a POP server, how is the end of the message marked?
- h) X is the superclass of Y. Is the assignment below a compile time error?
- ```
X x = new X();
Y y = new Y();

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

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

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

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

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

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

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

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

j) What is a "network effect" (we will accept any vaguely correct sentence).

k) Here is some HTML source.

```
<p>This dress exacerbates
the genetic
```

```
betrayal <p> that is my legacy.
```

```
It was the best of times.
```

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

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

l) What is an HTTP cookie, where is it created, where is it stored? (three sentences)

a cookie is:

created by:

stored by:

m) Suppose vendor A has 75% market share in some domain, and vendor B is one of many vendors dividing up the remaining 25%. Between A and B, which vendor is most likely to benefit from the development of a public standard in that domain?

n) Suppose a browser surfs over to a page for the first time ever. The page contains 10 different image tags (`<img src=url-of-image>`) for the 10 different images used in the page. How many separate GET requests will be made to download the page's HTML and images (in HTTP 1.0)?

o) What is inconvenient about private key cryptography (as opposed to public key cryptography)?

p) Consider a public key scenario where an email is encrypted with the private key before being sent. What is required to decrypt the message? Does this demonstrate privacy or authenticity?

## 2) Socket Programming (30 points)

This problem focuses on socket client programming - you may code your solution in C or Perl. For all the parts of this problem, you may assume that an EZSocket(host, portNum) function is available which takes care of establishing a connection to the given host and port number and returns the socket to use. EZSocket() returns false if the connection could not be opened, however you do not need to check that on this exam.

### Perl

```
$sock = EZSocket("binky.com", 345);
$line = <$sock>; // reading
print $sock "Foo!\r\n"; // writing
```

### C

```
int sock = EZSocket("binky.com", 345);

// You may assume that these read and write functions are defined...

// Write a C string to a socket
void WriteString(int sock, const char* string);

// Read one line of text ('\n') into the given buffer.
// Returns the number of chars read or 0 on EOF.
int ReadLine(int sock, char* buff, int buffSize);
```

For these problems we are not doing any error detection or worrying about end-of-line conventions. Your code may be written assuming that all connection succeed and the syntax of all returned text is correct.

a) HTTP. Write code that does the HTTP transaction to retrieve and print out the HTML body for the URL "http://cse.stanford.edu/". Your code may assume that no socket or HTTP error conditions happen – the content is always returned successfully and is HTML.

For this part of the problem, you will write a client for the "Binky" protocol. As before, we will keep this code short by ignoring error conditions. The end-of-line terminator used in the Binky protocol is the standard "\r\n". This first part of the problem describes the Binky protocol overall. The coding parts of the problem (b, c, d) will more specifically describe the code you need to write.

Here are the steps in the Binky protocol for one Binky URL...

- 1) A Binky URL has the form "binky:host/request" — a) *host* is a DNS name, b) the *request* is made of one or more non-whitespace characters, the first one of which must be a slash (/). The request will be sent to the server in step 4 below.
- 2) The client opens a connection to the host on port 1313.
- 3) The server sends back a "Hello" greeting to the client on a line by itself.
- 4) The client sends the request string on a line by itself.
- 5) First, the server sends back zero or more Binky URL's, each on a line by itself. The end of the sequence of Binky URLs is marked by a blank line.
- 6) The client stores the returned Binky URLs. The overall Binky protocol (below) uses a global array of Binky URLs. The URLs sent by the server in this step should be added to the end of that array, but not processed otherwise.
- 7) Second, after the blank line, the server sends HTML text terminated by the EOF.
- 8) The client prints the Binky URL one line, followed by all of the HTML. The client does not need to worry about end-of-line conversion within the HTML text — just print it out as it comes in.

The overall Binky process uses a global array of Binky URLs. The initial Binky URL is at index 0 in the array. Processing the initial URL prints out some HTML, and also adds more URLs to the end of the array. The client processes the URLs in the array in order (0, 1, 2, ...) until they have all been processed. Before processing each Binky URL, the client does a "visited" check to see if there has already been a connection to that DNS hostname. If there has already been a connection, then that Binky URL is skipped — no connection is made and nothing is printed.

b) Write code which given a full Binky URL as a string, parses out the host and request. You may write this as one function which returns both the host and request, or as two functions: one for host and one for request. Your code may assume that the URL is a Binky URL and it is syntactically correct.

c) Write a `oneBinky(url)` function that processes a single BinkyURL. The function should take as its argument the Binky URL string to process. The function should perform the client side of the Binky protocol for that URL. The function should add the returned URLs to the global array, named "urls", and print out the HTML. This function should not do the visited testing or any further processing on the URLs in the array (those are done in part (d) below).

d) Write an `allBinky(startUrl)` function that takes as its argument the starting Binky URL, initializes the global "urls" array, works its way through the array calling `oneBinky()` to process the URLs, and does the visited detection before each call to `oneBinky()`.

### 3) CGI Programming (30 points)

For the CGI problem, you will build a simple database browsing CGI. The first part of the question describes the overall structure of the CGI. The later parts (a b c) describe in more detail the code to write.

#### Database Browser

The database browser CGI lets a user see the rows in a simple two-column database. The first line of the database file is its title string, used in the title and h1 of every page. The second line in the database is its description sentence, which is printed below the h1 on every page. The rest of the lines in the file define a simple two-column database: each line is made of some "name" text, followed by a tab, followed by some description text. For this example, the "name" is the name of a movie, and the "description" is a quote from that movie, but the CGI should work for any two-column text data.

#### 1) Home Page

When invoked with a "db=filename" binding, the CGI shows its main page...



- 1) We are not doing any security checking the database filename. If the file cannot be opened, the CGI should return an error message. Other than that, we'll just assume the filename is ok, and we look for the file in the same directory as the CGI.
- 2) The title and description of the database are at the top of the page.
- 3) There are 26 buttons, one for each letter of the alphabet. Each button leads to a Letter page (below).

## 2) Letter Page

Choosing one of the 26 letters leads to a Letter Page. The top of the Letter page is the same as main page. The bottom of the Letter page is a table of the rows that begin with the chosen letter (not case-sensitive). The names (column 1) are shown, but the descriptions (column 2) are not. Here is what the table looks like having chosen the letter "t" ...



Each row has a "See" button that leads to a Detail page for that row in particular....

### 3) Detail Page

Choosing a row from the Letter page leads to the detail page for that row. The top of the Detail page is the same as for the other pages. The bottom of the Detail page shows the one selected row in a one-row, two-column table showing both the name and description.



### CGI Code

The following template has the basic structure of the CGI written for you. You need to fill in the missing code. You may write the solution as one big global piece of code. If you wish to separate out helper functions, please define them near where they are used. The @alpha array is provided for you as a simple way to iterate through the 26 letters of the alphabet.

```
#!/usr/bin/perl

require "193icgi.pl";

print "Content-type: text/html\r\n\r\n";

%bindings = ExtractBindings();

@alpha = ("a", "b", "c", "d", "e", "f", "g", "h", "i", "j",
"l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v",
"w", "x", "y", "z");
```

**##### a) Check for the db binding, open the file, print the main page**

**##### b) Detect that a letter has been chosen, print the table of rows  
##### starting with that letter**

**##### c) Detect that a row has been chosen, print its detail row**

## 4) Servlet Programming (20 points)

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

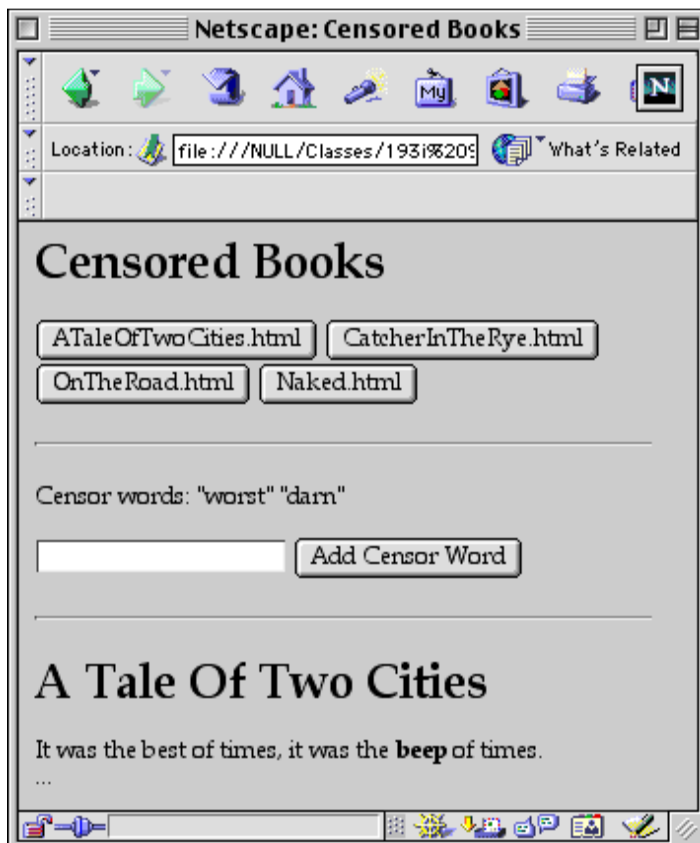
The top part of the page displays buttons, each of which can be used to choose a book. There's a Vector (below) that contains the book filenames which are used as the button titles.

### 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)
```

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

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

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