

# CS193i Final Exam

---

- **SITN Students:** in theory, this exam has been emailed or faxed to your coordinator so expertly that you are able to take the exam on our regular day: Mon, which is good since we're doing the grading Mon night. Please fax it back as soon as you are done with it. -- fax to Nick Parlante, 650 723-6092. Make sure you only write on one side of each page, or the Fax won't pick it up. After faxing, keep the original in case of problems, but you do not need to courier it.
- This is a 3-hour, open book, open note, closed computer exam. There are 6 problems across 17 pages.
- **Please write only on the front sides of pages** — if you need to add pages, please attach them to your exam with your name on them.
- Things like syntax, exact method names, import statements, etc., are not important.
- There is some time pressure, so if a problem is getting away from you, skip it until later.
- Today's quote: Nothing shouts poor workmanship more than wrinkles in the duct tape!

Good luck, and have a great summer!

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

Please check here if you are graduating this quarter: \_\_\_\_\_

Please check here if you are a P/NC student: \_\_\_\_\_

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

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

- 1: \_\_\_\_\_/10 \_\_\_\_\_ Socket  
2: \_\_\_\_\_/30 \_\_\_\_\_ CGI  
3: \_\_\_\_\_/10 \_\_\_\_\_ HTTP  
4: \_\_\_\_\_/20 \_\_\_\_\_ Servlet  
5: \_\_\_\_\_/10 \_\_\_\_\_ JavaScript

6: \_\_\_\_\_/20 \_\_\_\_\_ Short Answer  
\_\_\_\_\_/100

## 1. Socket (10)

For this problem, you will write some Perl socket code. You may assume that all text lines use a simple "\n" end-line convention. Ignore all errors -- assume that all I/O operations succeed and that all the data is formatted correctly. To connect client sockets, assume that a Perl "conn" function is defined:

conn(SOCK, *host-name*, *port-num*). All connections for this problem use port 1000.

- Open a socket connection on port 1000 to the "A" host named in the command-line array \$ARGV[0]. Read the lines of text from the A host until EOF.
- Nested within each line, there may be one string that identifies a "B" host and looks like this: <foo.com>  
The B host is identified by <> brackets around a hostname, where the hostname is a non-empty string made of letters, digits, and periods. For each B host, perform the following operation.
- Open a connection to the B host on port 1000 and process the lines of text it sends as below until EOF. Some of the lines sent back are special, identifying a C host, looking like...  
send+ to host 12.56.171.2 ignore these words  
The line will begin with a 'send+' and there will be a single C hostname identified by a numeric IP address -- 4 numbers separated by periods. The C hostname is surrounded by a number of other words on the line which should be ignored. When a C host is found, open a connection to it on port 1000 and send to it all of the lines of text read from the B host, including the line that identified the C host, and then close the connection. Send the line "ok+\n" to the B host to confirm that the data has been sent, and then continue reading from the B host.

Your Perl code here...

```
#!/usr/bin/perl
```



## 2. CGI (30) (difficult)

For this problem, you will implement a CGI that serves up documents, but enforces that the client has a valid username/password before sending the documents. We will use the convention that the request path for a document served by our CGI will look like:

```
/cgi.pl?file=foo.html&id=5376
```

where the "foo.html" is the file to open, and "5376" is the session id that shows that the client is authorized to get documents.

On the server side, we keep a users file with 4 columns separated by tabs: username, password, session id, and time. The session id is a random number (e.g. 4567), and the time is the time that the session id was created, measured in seconds since the machine booted up (e.g. 1200034).

```
jsmith      secret    4567    1200034
misspiggy   igpay     3291    1200099
jane        password  2156    1200012
...
```

Here are the three main cases the CGI must deal with...

### 1. Check id.

A request comes in with the form "file=foo.html&id=5376". The file binding is always present, but the id may or may not be. If the id is not present, or if it is not valid, then return a login form that asks for a username and password. The session is valid if it matches a session in the users file, and that session is less than 3600 seconds old. Assume that the function time() returns the current time in seconds. If the request is valid, return the file as described by step (3) below.

### 2. Login.

When the login form is submitted, check the username/password against the local "users" text file. If the username and password match then: generate a random session id and store that id into users file along with the current time. Having logged in once, the session id should allow the user to look at files for 3600 seconds. Assume that the newid() function returns a random session id. If the login is successful, then return the original file that the user asked for in step (1) above. If the login is not successful, return the login page again.

### 3. Page processing.

Suppose we have a valid request for a file and a valid id. We assume that the file exists and is HTML. Consider simple relative URLs within the file -- so for example the file a.html might contain an "<a href=b.html>" url. The client gets the a.html file through a "file=a.html&id=123" request to the CGI. When returning the a.html file, the CGI must re-write the urls so that they will work with the CGI if clicked. We will only deal with very simple relative urls. It's sufficient to look for the "href=filename.html" pattern in the returned html (no quotes, no spaces, no slashes). Rewrite the url to the form "/cgi.pl?xxx" where xxx will load the right file.

```
#!/usr/bin/perl  
use CGI;  
$q = new CGI;
```





### 3. HTTP Quotes (10)

For this problem, you will write a little java code to act as a trivial HTTP quote server. The starter code below has the basic structure done for you. Write java code that waits for incoming HTTP connections. For a request path of the form `"/quote/word"`, it should search the given quotes array of strings, and produce HTML `<ul>` list output that lists all the quotes that include word. Requests that do not parse or have other problems can simply return zero quotes. So a request path of `"/quote/fish"` will return a page of quotes that include the word "fish". You may do a simple case-sensitive search, and you may omit the try/catch code. The HTTP response header should include both the content-type and content-length fields. You may use regular-expressions or the simple `s.indexOf(sub)` String message, which returns the index where sub begins or -1 if not found.

```
private String[] quotes = ....; // assume quotes is set

public void runServer() {
    ServerSocket serv = new ServerSocket(80);

    while (true) {
        Socket sock = serv.accept();
        BufferedReader in = new BufferedReader(
            new InputStreamReader(sock.getInputStream()));
        OutputStream out = sock.getOutputStream();

        String line = in.readLine();
```

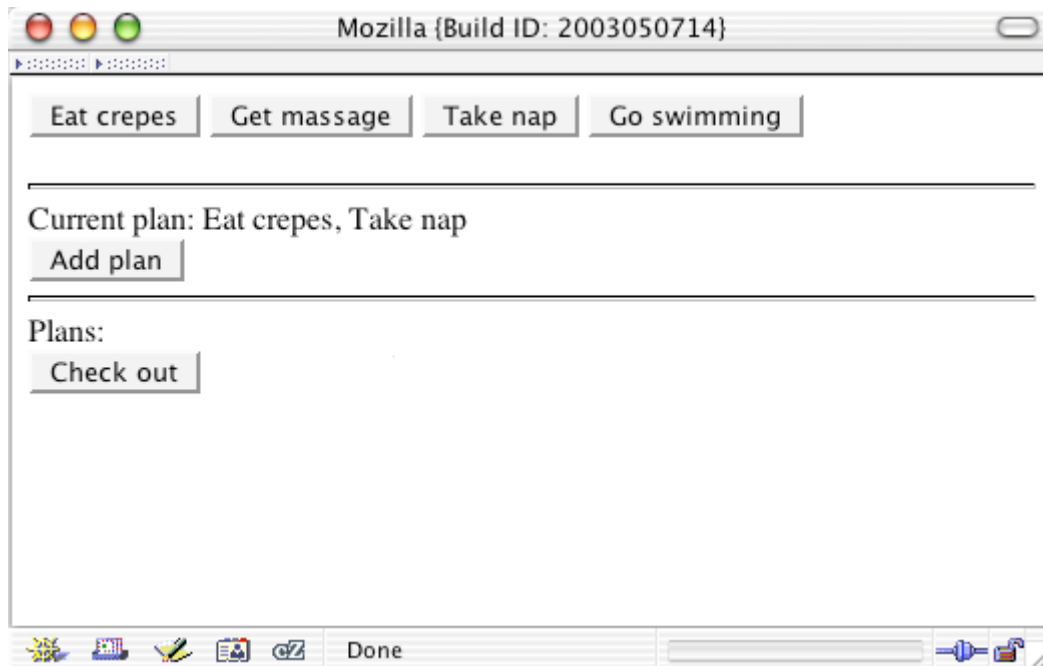


## 4. Servlet (20)

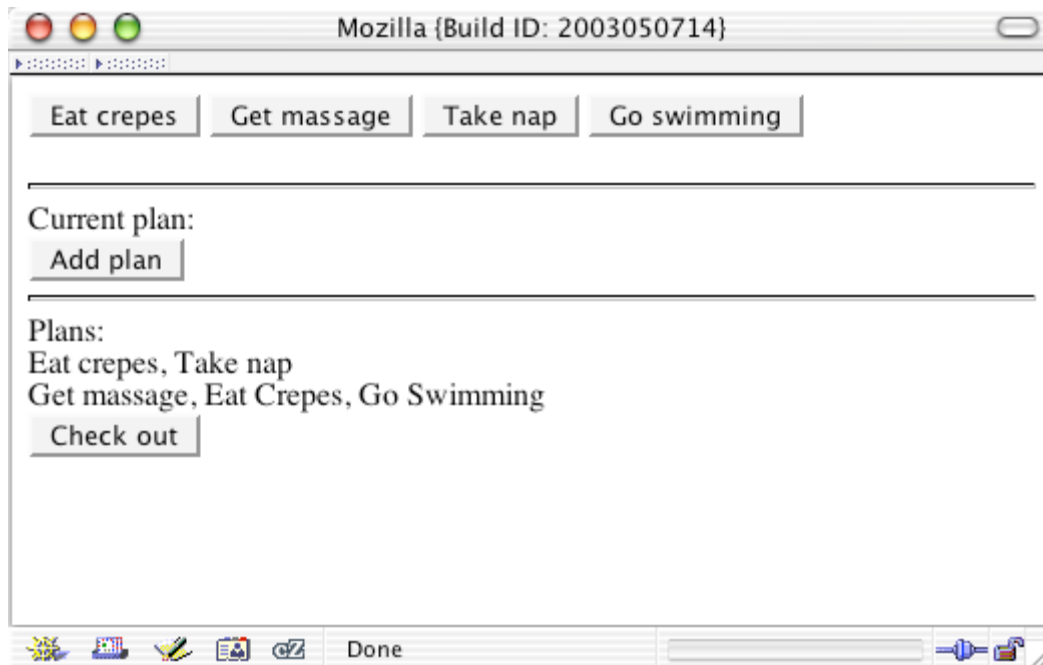
To be prepared for the summer, for this problem you will build a servlet that allows a user to set up a set of "plans" where each plan is a sequence of fun activities.

The page has three sections: the first section has a button for each fun activity. The second section lists the current plan that is being built up. Clicking a button in the first section appends that activity to the current plan. The third section lists the added plans.

For example, here the user has clicked "Eat crepes" followed by "Take nap" to build a 2-step plan...



The third section shows the list of added plans. Clicking the Add plan button adds the current plan to the list of plans and clears the current plan. Here the user has constructed and added two plans...



Clicking the "Check out" button should just clear out the list of plans, and we'll pretend the user's credit-card gets charged appropriately somehow.

We will provide partial support for the back button: suppose the user builds up a plan A, B, C. But then they hit the back button, so they are seeing the plan A, B. If they hit the Add plan button at that point, the plan added should be A, B.

Assume that an instance variable "fun" is already initialized in the servlet, and it is an array of the names of the fun activities. Each name will contain only letters and whitespace.

```
public class Fun extends HttpServlet {
    private String[] fun; // assume contains activity strings

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





## 5. JavaScript (10)

Suppose we are working on a bit of code similar to the fun activity servlet -- we have the `String[] fun` array of fun activities. Suppose that there is also a `String[] comment` array that contains a sentence for each of the fun activities.

Write a servlet that outputs an HTML page with one button for each fun activity and a text field at the bottom. There is JavaScript in the page so that when each button is clicked, the text field is updated to contain the corresponding comment.

For example, clicking the Eat crepes button might look like...



The servlet does not have any functionality other than that -- it just demonstrates the button-to-text-field Javascript feature.

```
public class Fun extends HttpServlet {
    private String[] fun; // assume contains activity strings

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





f. Suppose a NAT router has the IP address 1.2.3.4 on its WAN/Internet side, 192.168.1.1 on its LAN side, and there is a host on its LAN side with address 192.168.1.100. That host wants to communicate with foo.com 9.8.7.6 out on the Internet, and so sends out a packet of the form (From:192.168.1.100 To:9.8.7.6). True or false: Packets coming back from foo.com in response will be addressed as follows when they reach the NAT router: (From: 9.8.7.6 To: 192.168.1.100).

g. What is a SYN packet?

h. Draw a line connecting the word on the left with its area on the right. (right lines get +points, wrong lines get -points)

SMTP	Email reading
RFC	Packet lifetime counter
IMAP	Ethernet variant
SCP	Internet standard
10T	Email sending
TTL	Encrypted file copying

i. Suppose the page `http://foo.com/a.html` has the following link in it :  
`<a href=bar.com>click me!</a>`

For an HTTP/1.0 request, what is the exact GET line that will be sent to the server when the above link is clicked?