

CS193i Final Exam

- **SITN Students:** in theory, this exam has been carried or faxed to your coordinator so expertly that you are able to take the exam on our regular day: Wed. Or if that didn't work, you may take it on Thu. In any case, we need the exam back ASAP since we are starting the grading Wed. Please have your coordinator fax the completed exam back 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.
- This is a 3-hour, open book, open note, closed computer exam. There are 5 problems across 16 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.
- For Sale: Parachute. Only used once, never opened, small stain.

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 has been given or received on this exam.

Signed: _____

1: ____/20 ____

2: ____/20 ____

3: ____/30 ____

4: ____/15 ____

5: ____/15 ____

____/100

1. Socket (20)

a. Binky Socket

The Binky protocol operates as follows: The binky server listens on port 1010. When the client connects, the server sends a one line greeting that ends with '\n' such as "Greetings oh client\n". The client then sends the string "hello\n", the server responds "-nobody here\n", the client sends "hello\n" again, the server responds "-nobody here\n" again. The server will respond in this way with the "-nobody here\n" response an unknown number of times (including zero). Eventually, instead of "-nobody here\n", the server will send a single real response like this "+Klaatu Barada Nikto\n". The real response will begin with a '+' char, and it signals the end of the protocol.

Write Perl code that implements the Binky client. Your code should only print two things to standard output from the server: the initial greeting, and the real response. You may assume that the server implements the Binky protocol correctly and you may ignore all errors.

Assume that the hostname is in \$host, and the perl conn() function is already defined that creates and connects a socket like this...

```
conn(SOCK, $host, $port);          // now SOCK is connected
```

Your Perl code here...

```
#!/usr/bin/perl
## assume $host is set
```

b. Soze Socket

Here are the steps in the famous Soze protocol...

1. Connect to \$host on port 2000. We'll call this host "A".
2. Read a one line response from host A, which will be three words, separated by commas. The words will be composed of letters, digits, and periods -- like this: "www.foo.com,700,magic\n".
3. Connect to the host named in word-1 (we'll call this host "B") using the word-2 port, and send it word-3 on a line by itself.
4. Host B will send back a series of text lines ending with an EOF.
5. If one of the lines sent by host B includes word-3 at the very start of the line, then close the B connection, and send the matching line from host B to host A, and go to step 2 above. If none of the lines from host B match word-3, then we are done.

Assuming that the host A name is in \$host, write the Perl code for Soze protocol. You may assume the existence of a conn(SOCK, \$host, \$port) function and you may ignore all errors.

```
#!/usr/bin/perl
## Assume $host is set
```

2. HTTP (20)

a. Simple HTTP client

Suppose \$host contains a hostname, and \$field contains the name of a HTTP header field such as "Content-type". Write perl code that connects to an HTTP server running on the given host, requests the "/" path, and then searches the HTTP response header for \$field. If the \$field is defined in the HTTP response, then print that line of the HTTP response, which will look like this

Content-type: text/html

Otherwise do not print anything. Note that HTTP header field names are not case-sensitive. You may ignore all error cases and use the conn(SOCK, \$host, \$port) function.

```
#!/usr/bin/perl
## Assume $host and $field are set
```

b. HTTP Proxy

In China currently, the government attempts to control the web sites that its people can visit to prevent the spread of non-approved ideas. You are going to write a server that runs outside of China to allow people in China to visit web sites. Suppose that connections to "www.foo.com" are blocked from inside of China.

Write a Perl program that listens on port 80 for incoming HTTP/1.0 GET requests. The path in the HTTP request will look like this: "/www.foo.com/a/b.html". Make an HTTP connection to port 80 on the host given in the first part of the path ("www.foo.com") and pass along the rest of the request ("/a/b.html"). Read back the entire HTTP response, send it all to the original client, and close the connection.

You may ignore all error cases and use the `conn(SOCK, $host, $port)` function. Here's the standard server-side socket code started for you...

```
CreateServerSocket(SERVER, $port);
listen(SERVER, SOMAXCONN);

while (accept(CLIENT, SERVER)) {
    ## your code here
```

```
// more space
```

c. The proxy described above will get the HTML from `www.foo.com` and re-send it to the client. However, there can be problems with the URLs inside the HTML. Suppose that `www.foo.com` has documents `/a/b.html`, `/a/c.html`, and `/a/d.html`. Suppose the user in China has just accessed `/a/b.html` through your proxy, and `b.html` includes the following two URLs. Write "yes" to the right of each URL if it will work when clicked, or "no" if it will not, or leave each blank if you do not know. (+1 for each right, 0 for blank, -1 for each wrong)

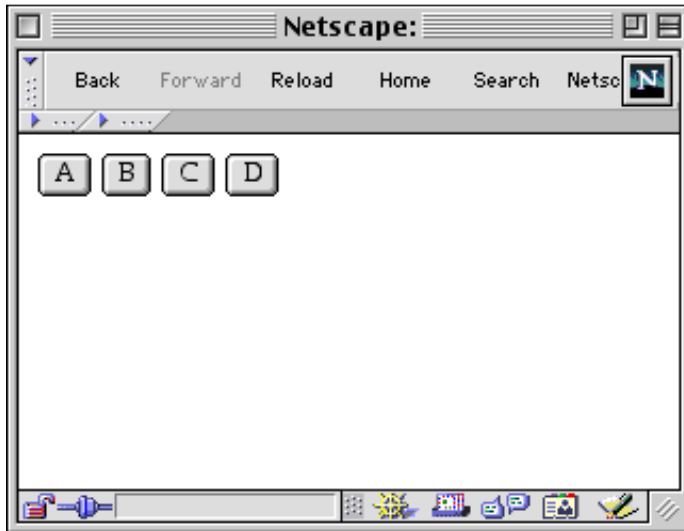
```
<a href="c.html">c</a>
```

```
<a href="/a/d.html">d</a>
```

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.

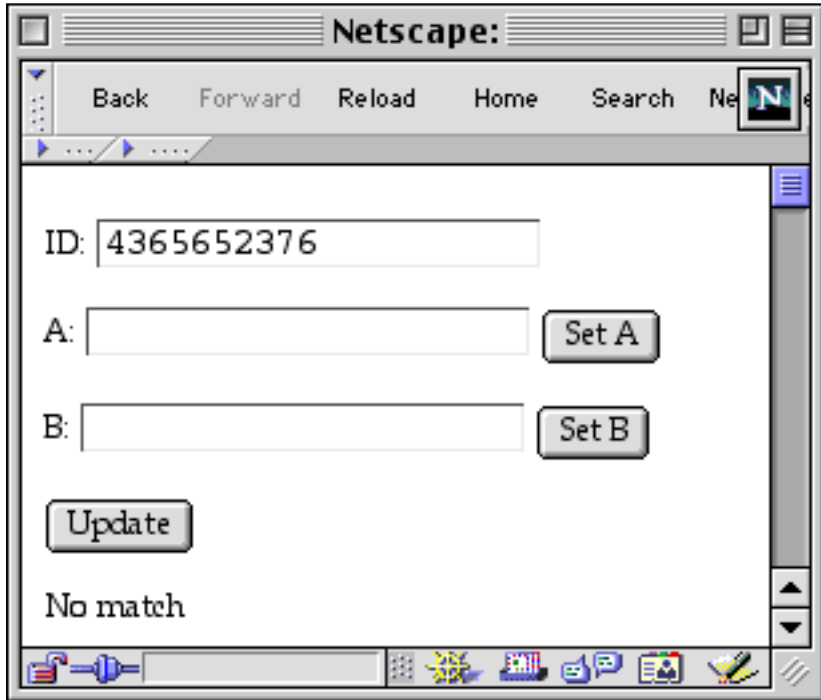
```
#!/usr/bin/perl
## (a) simple CGI

use CGI;

$query = new CGI;
```

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. They need 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
## (b) complex CGI

use CGI;

$query = new CGI;
```

```
// more space
```

4. Servlet (15)

This servlet will implement a simple game. On the first request, the user sees this form...



From here, the user can repeatedly select A, B, C or D, which returns this same form. As the user does this, they build up a sequence "C", "D", "A", etc. on the server side.

The "Set Greeting" button should return the same form, but with the given greeting at the top of the page ("Greetings" in this case). The greeting should be stored in a persistent cookie, so that it will re-appear the next time the browser is used with this servlet. Set Greeting with the empty string should delete the cookie.

When the user clicks the "Back to start" button, they get the same form again, but now they must repeat the same "C", "D", .. sequence they entered earlier. If at any point they click the wrong button -- return the form with "You lose" below the greeting. If they complete the sequence successfully, return the form with "You win" below the greeting. In either case, reset the sequence so the user may enter another.

As usual, we will not worry about error conditions. You may assume that the `findCookie(name)` method from the homework is already defined.

```
public class Game extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws IOException, ServletException  
    {  
        // your code here  
    }  
}
```

```
// more space
```

5. Short Answer (15)

We are looking for **very short** answers for these. Correct = +points, Wrong = -points, Blank = 0. Leave the question blank if you are not sure. You cannot get less than zero points for this question.

a. Suppose a computer is connected to the Internet through an NAT router. The computer has IP address 192.168.1.10 and the LAN side of the router is at 192.168.1.1 and the Internet side of the NAT router is 171.64.64.171. The computer establishes a connection to `www.yahoo.com` -- from yahoo's point of view, what IP address will be the source of the connection?

b. What is an RFC?

c. Name an advantage of a switched -circuit network connection.

d. True or false: for TCP/IP traffic between two computers connected by ethernet, the IP datagram is sent inside of an ethernet packet.

e. In the context of TCP -- what is "flow control"?

f. The traceroute utility maps the IP addresses between two hosts. How is it implemented?

g. What is a Distributed Denial of Service attack?

i. On Unix, the system does not store a user's password. Instead the system stores a one-way hash of the users's password. What problem is avoided by storing only the hash?

j. Suppose you are working on a JSP that takes a Java QuoteBean that responds to a getQuote() message. Suppose the usebean tag is correct, complete the JSP to splice in the quote from the bean...

```
<p>  
<jsp:usebean id="bean" scope="page" class="QuoteBean" />  
  
Quote:  
<!-- your code here -->
```

k. With public key cryptography (asymmetric cryptography), suppose Alice signs a doc by computing $h = \text{hash}(\text{doc})$, and then computing a signature $s = \text{crypt}(h, \text{alice_priv})$. Alice sends Bob doc and s. Bob obtains alice's public key (alice_pub) from a Certificate Authority. What steps does Bob use to verify Alice's signature on the document?