CSC 546: Client/Server Fundamentals

Fall 2000

World Wide Web as a client/server system

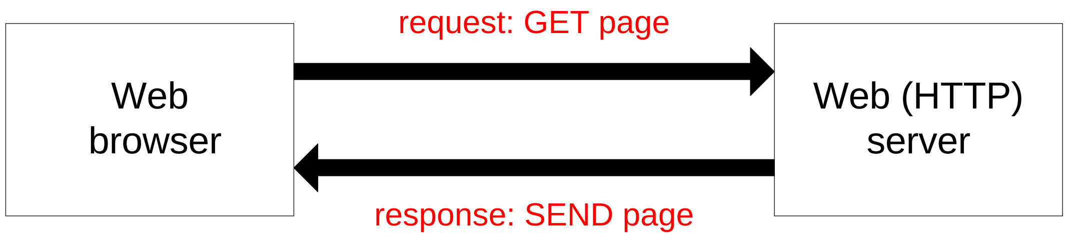
- HTTP
- caching
- cookies
- server-side programming

World Wide Web

the Web is the world's largest client/server system

communication occurs via message passing

- within browser, select URL of desired page
- browser requests page from server
- server responds with message containing
  - type of page (HTML, gif, pdf, zip, …)
  - page contents
- browser uses type info to correctly display page
- if page contains other items (images, applets, …), browser must request each separately



HTTP

Hypertext Transfer Protocol (HTTP):

application-level protocol for distributed, collaborative, hypermedia information systems

- generic, stateless, object-oriented
- can be used for many tasks, such as name servers & distributed object management systems
- underlying language of the Web

Note:

HTTP is an application-level protocol

TCP/IP is the underlying transport-level protocol of the Internet

Connectionless vs. connection-oriented

HTTP/1.0 allowed only connectionless message passing

- each request/response required a new connection
- to download a page with images required multiple connections

  can overload the server, require lots of overhead

HTTP/1.1 provides persistent connection by default

- once client and server connect, remains open until told to close it

  usually implemented via timeouts

  reduces number of connections, saves overhead

- client can send multiple requests without waiting for responses

  e.g., can request all images in a page at once

GET request

most URL's have the form: `protocol://serverName URI`

e.g., `http://www.creighton.edu/~davereed/index.html`

to retrieve a document via HTTP from the server, issue a GET request

```
GET URI HTTP/1.1
Host: serverName
```

Web server only knows the contents of the GET request message

- automatically generated by browser when you select a URL
- could also come from a link checker, a search engine robot, …
- can come directly from a telnet connection using port 80

GET example

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to parrot.creighton.edu.
Escape character is '^]'.
GET /~davereed/index.html HTTP/1.1
Host: www.creighton.edu
```

```
HTTP/1.1 200 OK
Date: Wed, 13 Sep 2000 20:29:50 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Tue, 05 Sep 2000 06:52:39 GMT
ETag: "a9260-18a-39b49837"
Accept-Ranges: bytes
Content-Length: 394
Content-Type: text/html

<HTML>
<!--- Dave Reed          index.html         8/17/00 -->
<!------------------------------------------------------>

<HEAD>
<TITLE>Dave Reed's Home Page</TITLE>
<SCRIPT LANGUAGE="JavaScript">
    if (self!=top)
    top.location.href=self.location.href;
</SCRIPT>
</HEAD>

<FRAMESET COLS="170,*">
    <FRAME SRC="menu.html" NAME="menu">
    <FRAME SRC="info.html" NAME="main">
</FRAMESET>

</HTML>
```

server response has assorted
header information,
followed by the page

Response header fields

the first line of the server's response contains a status code

200 OK	request was processed successfully
301 Moved permanently	document has been moved
304 Not modified	if cached version is up-to-date
400 Bad request	syntax error in client's request
403 Forbidden	client is not allowed access (e.g., protected)
404 Not found	file could not be found
500 Internal server error	server failed
503 Service unavailable	server is overloaded

Other response header fields

in addition to the status code, the server's response may include

▪ Date	response time (in GMT)
▪ Server	identification info on the server
▪ Last-modified	time document was last changed (in GMT)
▪ Content-length	size of document, in bytes
▪ Content-type	file format (e.g., html, gif, pdf)
▪ Expires	prevents browser from caching beyond date

File not found

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to parrot.creighton.edu.
Escape character is '^]'.
GET /~davereed/foo.html HTTP/1.1
Host: www.creighton.edu
```

```
HTTP/1.1 404 Not Found
Date: Wed, 13 Sep 2000 21:03:02 GMT
Server: Apache/1.3.12 (Unix)
Content-Type: text/html

<head>
<META HTTP-EQUIV=refresh
    CONTENT="50;URL=http://www.creighton.edu/">
<title>Requested Page Not Found!</title>
</head>
<body bgcolor=white>
<font face="Arial">
<h1>Requested Page Not Found!</h1>
<hr>

<p><b>
The URL you requested was not found on <a
href="http://www.creighton.edu">this server</a>.
    (Error 404)
</p>
<p>
.
.
.
```

if file not found, response includes 404 status code and generic error page

Directories as URI's

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to parrot.creighton.edu.
Escape character is '^]'.
GET /~davereed/ HTTP/1.1
Host: www.creighton.edu
```

```
HTTP/1.1 200 OK
Date: Wed, 13 Sep 2000 20:29:50 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Tue, 05 Sep 2000 06:52:39 GMT
ETag: "a9260-18a-39b49837"
Accept-Ranges: bytes
Content-Length: 394
Content-Type: text/html

<HTML>
<!--- Dave Reed          index.html        8/17/00 -->
<!------------------------------------------------------>

<HEAD>
<TITLE>Dave Reed's Home Page</TITLE>
<SCRIPT LANGUAGE="JavaScript">
    if (self!=top)
    top.location.href=self.location.href;
</SCRIPT>
</HEAD>

<FRAMESET COLS="170,*">
    <FRAME SRC="menu.html" NAME="menu">
    <FRAME SRC="info.html" NAME="main">
</FRAMESET>

</HTML>
```

if a directory is specified,
will look for a file
named `index.html`

Redirection

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to parrot.creighton.edu.
Escape character is '^]'.
GET /~davereed HTTP/1.1
Host: www.creighton.edu
```

```
HTTP/1.1 301 Moved Permanently
Date: Wed, 13 Sep 2000 20:58:58 GMT
Server: Apache/1.3.12 (Unix)
Location: http://www.creighton.edu/~davereed/
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

166
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A
    HREF="http://www.creighton.edu/~davereed/">here</A
    >.<P
>
<HR>
<ADDRESS>Apache/1.3.12 Server at <A
    HREF="mailto:webmaster@creighton.edu">www.creighto
    n.edu</A> Port 80</ADDRESS>
</BODY></HTML>


0
```

since URI is missing / at end, browser must do 2 requests

Request header fields

the client can specify additional information in the request

▪ User-Agent	specifies the browser version
▪ Referer	tells server where the user came from *useful for logging and customer tracking*
▪ From	contains email address of user *generally not used for privacy reasons*
▪ Authorization	can send username & password *used with documents that require authorization*
▪ If-Modified-Since	only send document if newer than specified date *used for caching*

Conditional GET

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to parrot.creighton.edu.
Escape character is '^]'.
GET /~davereed/ HTTP/1.1
Host: www.creighton.edu
If-Modified-Since: Wed, 13 Sep 2000 14:00:00 GMT
```

```
HTTP/1.1 304 Not Modified
Date: Wed, 13 Sep 2000 20:36:55 GMT
Server: Apache/1.3.12 (Unix)
ETag: "a9260-18a-39b49837"
```

since the document has not been modified since the specified date, the page is not sent by the server (status code 304)

Other request methods

HEAD	similar to GET, but requests header information only *useful for checking to see if a document exists, how recent*
POST	similar to GET, but encodes inputs differently *useful for submitting form contents to a CGI program*
PUT	upload a document to the server *new in HTTP/1.1*
DELETE	delete a document from the server *new in HTTP/1.1*

HEAD example

```
bluejay> telnet www.creighton.edu 80
Trying...
Connected to parrot.creighton.edu.
Escape character is '^]'.
HEAD /~davereed/index.html HTTP/1.1
Host: www.creighton.edu
```

```
HTTP/1.1 200 OK
Date: Wed, 13 Sep 2000 20:54:23 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Tue, 05 Sep 2000
    06:52:39 GMT
ETag: "a9260-18a-39b49837"
Accept-Ranges: bytes
Content-Length: 394
Content-Type: text/html
```

server does not send the page, only the header information

Caching

browsers cache pages to save downloading

- maintain temporary storage (cache) for recent pages
- when a page is requested, check to see if already in cache
- if not in the cache, issue GET request
  - when response message arrives,
    - display page and store in cache (along with header info)
- if already stored in the cache, send GET request with If-Modified-Since header set to the data of the cached page
  - when response message arrives,
    - if status code 200, then display and store in cache
    - if status code 304, then display cached version instead

Cookies

HTTP message passing is transaction-based, stateless

- many e-commerce apps require persistent memory of customer interactions

e.g., amazon.com
    remembers your name, credit card, past purchases, interests

Netscape's solution: cookies

- a *cookie* is a collection of information about the user

- server can download a cookie to the client's machine using the "Set-cookie" header in a response

```
Set-cookie: CUSTOMER=Dave_Reed; path=/; expires=Thursday, 14-Sep-00 12:00:00
```

- when user returns to URL on the specified path, the browser returns the cookie data as part of its request

```
Cookie: CUSTOMER=Dave_Reed
```

Programming the Web

client-side programs (executed on client's machine)

- JavaScript: embed code in HTML text, interpreted by browser
- Java: place applet on server, downloaded and executed by browser

server-side programs (executed on server's machine)

- CGI programs
- server-side includes
- active server pages (Microsoft), Java server pages (Sun)

useful when program requires large amounts of data, infrequent interaction

Server-side programming

To execute a server-side program,

- server receives a request
- must recognize that the URL maps to a program, not a document
- server executes program
  - feeds data from request message to program as output
  - takes program output, adds appropriate HTTP headers, and sends

CGI (Common Gateway Interface)

protocol for input/output of a server-side program

- program can be written in any language as long as it accepts input
  and produces output as specified by CGI

- server must be able to recognize a URL as being a CGI program
  generally done by placing program in special `cgi-bin` directory

CGI output

The output of a CGI program consists of

- HTTP headers
- blank line
- program output to be displayed/downloaded

At minimum, HTTP header must specify content type

e.g., `Content-Type: text/html`

At minimum, output can be plain text

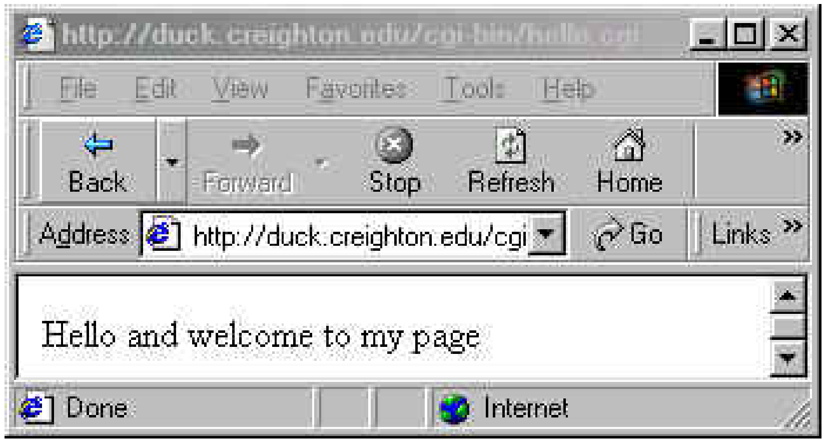
e.g., `Hello and welcome to my page`

CGI example

```
// hello.cpp
#include <iostream>
using namespace std;

int main()
{
  cout << "Content-Type: text/html" << endl
       << endl;

  cout << "Hello and welcome to my page " << endl;

  return 0;
}
```

executable is stored in the cgi-bin under the name hello.cgi

GET request executes program


http://duck.creighton.edu/cgi-bin/hello.cgi

Address: http://duck.creighton.edu/cgi

Hello and welcome to my page

CGI file access

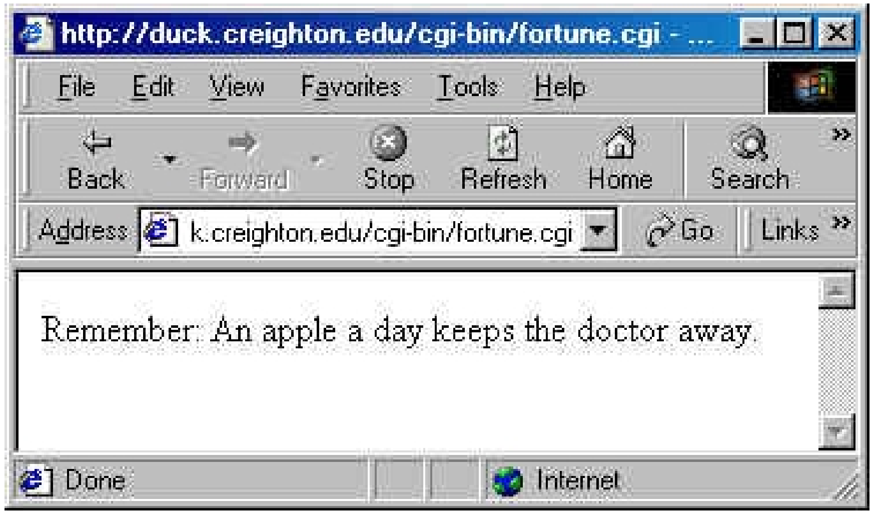
CGI programs can access local files (e.g., databases)

Here, `fortune.txt` contains various fortunes/cliches (1st line specifies #)

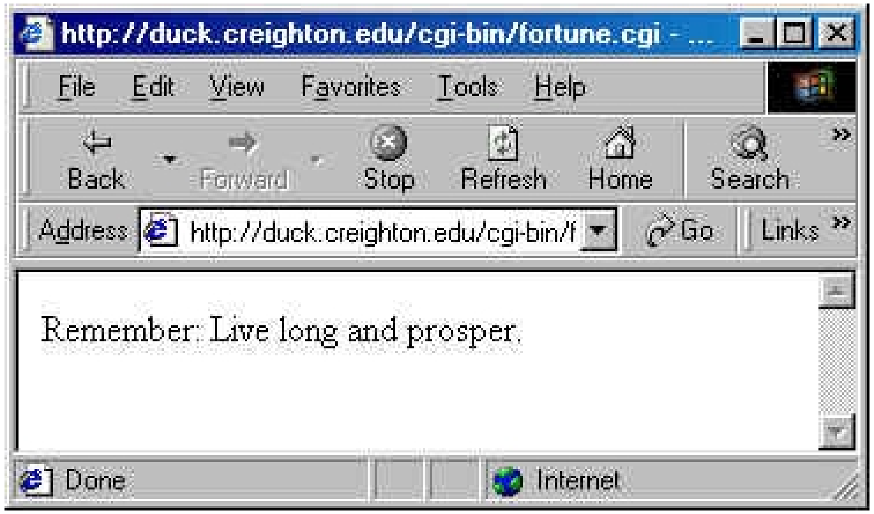
`fortune.cpp` reads a random fortune from the file and displays it in a page

```
// fortune.cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
  ifstream ffile("fortune.txt");

  int numFortunes;
  string line;
  ffile >> numFortunes;
  getline(ffile, line);

  srand((unsigned)time(NULL));
  int find = rand() % numFortunes;
  for (int i = 0; i <= find; i++) {
    getline(ffile, line);
  }
  ffile.close();

  cout << "Content-Type: text/html" << endl
       << endl;
  cout << "Remember:  " << line << endl;

  return 0;
}
```

fortune.cgi


http://duck.creighton.edu/cgi-bin/fortune.cgi - ...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address k.creighton.edu/cgi-bin/fortune.cgi
Go Links
Done
Internet

Remember: An apple a day keeps the doctor away.


http://duck.creighton.edu/cgi-bin/fortune.cgi - ...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://duck.creighton.edu/cgi-bin/f
Go Links
Done
Internet

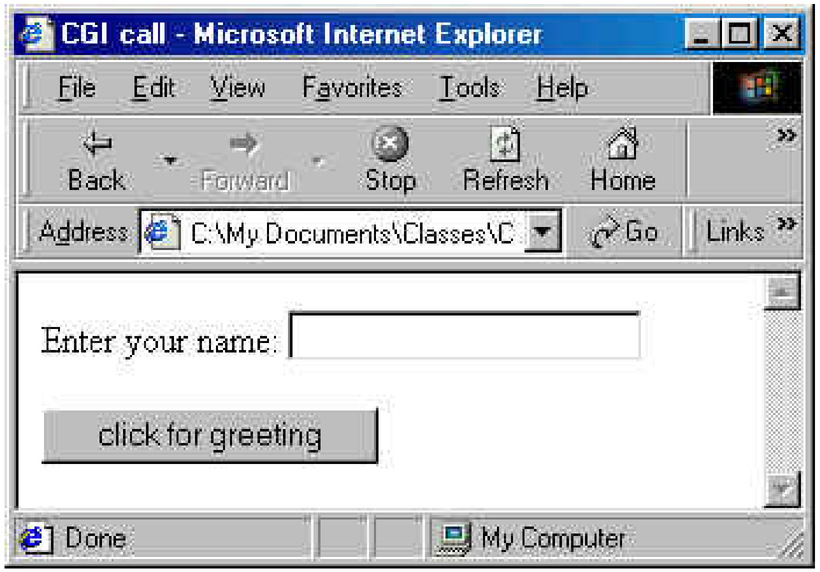
Remember: Live long and prosper.

CGI input

CGI programs can accept input

- usually provided via form elements in a Web page

```
<HTML>
<HEAD><TITLE>CGI call</TITLE></HEAD>

<BODY>
<FORM ACTION="http://duck.creighton.edu/cgi-
    bin/helloEcho.cgi" METHOD="POST">

  Enter your name: <INPUT TYPE="text"
    NAME="yourName">
<BR><BR>
  <INPUT TYPE="submit" VALUE="click for
    greeting">
</FORM>
</BODY>
</HTML>
```


CGI call - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Address
C:\My Documents\Classes\C
Go
Links
Enter your name:
click for greeting
Done
My Computer

When user clicks on button, data in the form is submitted

- arrives as part of the request message, read as input by CGI program

URL-encoding

input data from a page is sent *URL-encoded*

```
name1=value1&name2=value2&name3=value3…
```

e.g., `yourName=Dave`

special characters are translated

- space is represented using +
- non-letters digits are represented using ASCII code (preceded by %)

e.g.,

```
yourName=Dave+Reed
yourName=Catherine+O%27Hara
```

GET vs. POST

form data can be submitted using either GET or PUT

GET form data is appended to the URI in the request

must be accessed by CGI program via environment variables

e.g.,

```
GET /cgi-bin/helloEcho.cgi?yourName=Dave HTTP/1.1
Host: duck.creighton.edu
```

POST form data is appended to end the request (after headers + blank line)

can be accessed by CGI program via standard input

e.g.,

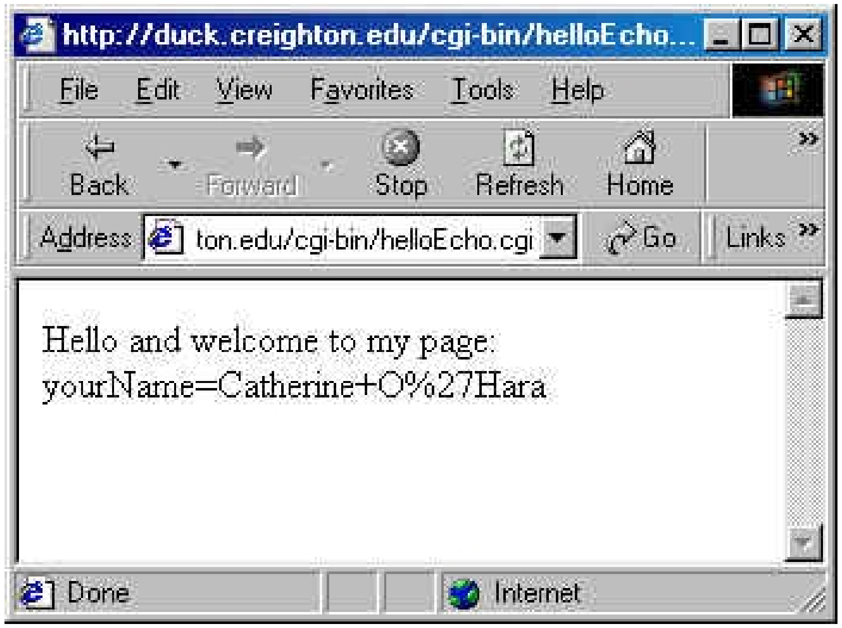
```
POST /cgi-bin/helloEcho.cgi HTTP/1.1
Host: duck.creighton.edu

yourName=Dave
```

POST example

```
// helloEcho.cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string inputString;
  cin >> inputString;

  cout << "Content-Type: text/html" << endl
       << endl;

  cout << "Hello and welcome to my page: " <<
    inputString << endl;

  return 0;
}
```

reads URL-encoded data from cin

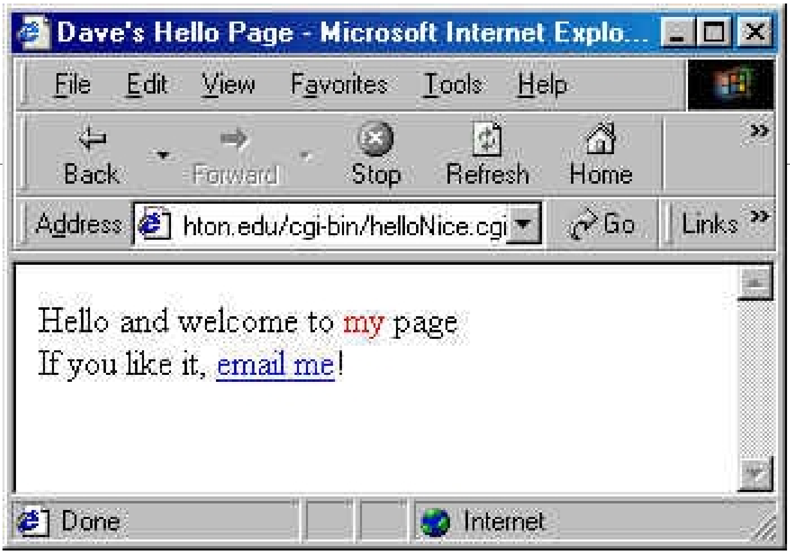
displays it (unaltered)



```
Hello and welcome to my page:
yourName=Catherine+O%27Hara
```

HTML output

```
// helloNice.cpp
#include <iostream>
using namespace std;

int main()
{
  cout << "Content-Type: text/html" << endl
       << endl;
  cout << "<HTML>" << endl
       << "<HEAD><TITLE>Dave's Hello Page</TITLE></HEAD>" << endl
       << "<BODY>" << endl
       << "Hello and welcome to <FONT COLOR='red'>my</FONT> page<BR> " << endl
       << "If you like it, "
       << "<A HREF='mailto:davereed@creighton.edu'>email me</A>!" << endl
       << "</BODY></HTML>" << endl;

  return 0;
}
```


Dave's Hello Page - Microsoft Internet Explo...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address hton.edu/cgi-bin/helloNice.cgi Go Links
Done
Internet

Hello and welcome to my page
If you like it, email me!

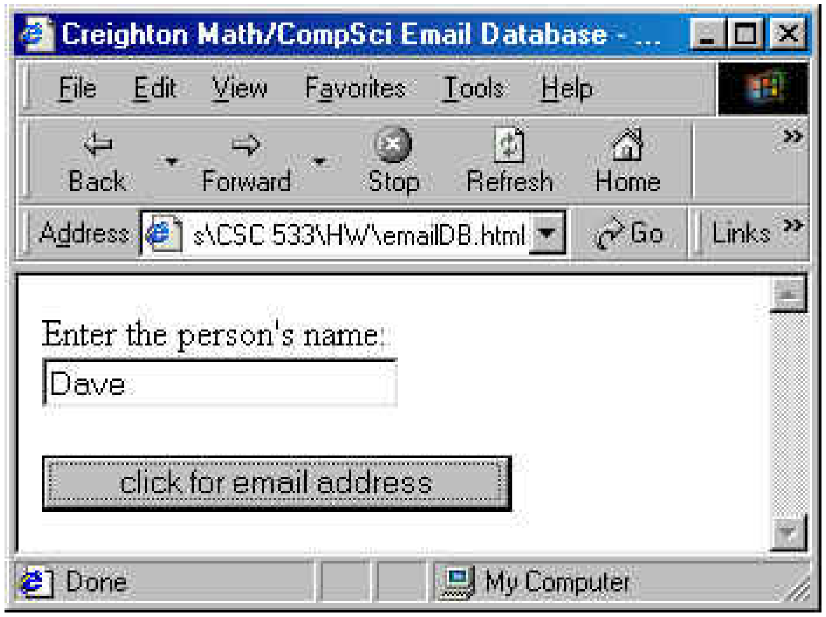
HW1 & HW2

you will develop a client/server based email address search engine

- specify name (or partial name) of person in a Web form
- CGI program will search a database file, return all matches

HW1: simple substring matching, text output

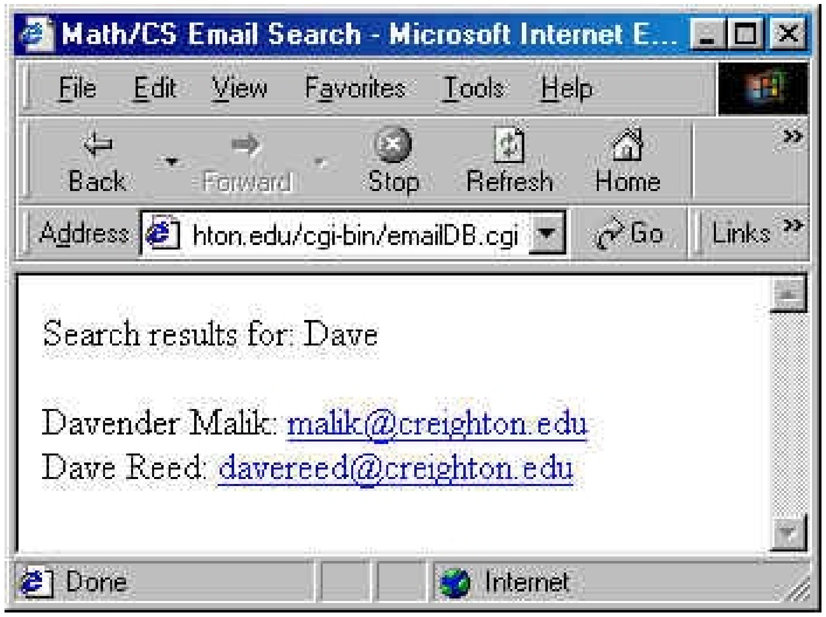
HW2: more complex matching, rich HTML output



Creighton Math/CompSci Email Database - ...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address s:\CSC 533\HW\emailDB.html
Go Links
click for email address
Done
My Computer

Enter the person's name:

Dave



Math/CS Email Search - Microsoft Internet E...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address hton.edu/cgi-bin/emailDB.cgi
Go Links
Done
Internet

Search results for: Dave

Davender Malik: malik@creighton.edu
Dave Reed: davereed@creighton.edu

Next week…

Major client/server protocols

- microcomputer: NetBEUI, SPX/IPX, named pipes
- UNIX: TCP/IP, Sun RPC, DCE RPC
- sockets
- SNA protocols

Read Chapter 10

As always, be prepared for a short quiz