

# CSC 551: Web Programming

Fall 2001

## Server-side programming & CGI

- server-side vs. client-side
  - advantages
  - common applications
- Common Gateway Interface (CGI)
  - HTTP messages, GET vs. POST
  - CGI program  $\leftrightarrow$  HTML client
  - input: URL-encoding, form processing
  - output: HTTP headers, HTML formatting

## Client-side recap

### JavaScript provides for client-side *scripting*

- source code is downloaded with the Web page
- interpreted by the browser as the page is loaded
- simple execution model, language is closely integrated with HTML

### Java provides for client-side *programming*

- source code is compiled into Java byte code on server
- byte code is downloaded with the Web page
- interpreted by the Java Virtual Machine in the browser
- more complicated model, requires compiler on server
- (slightly) faster execution, full-featured with extensive library support

### both approaches yield platform independence

- requires JavaScript/Java enabled browser for desired platform

## Server-side vs. client-side programming

instead of downloading the program and executing on the client,

- have the client make a request
- execute the program on the server
- download the results to the client

### advantages

- cross-platform support  
browser variations/bugs yield differences with JavaScript & Java applets  
with server-side, only have to test & optimize program for server platform
- more options for applications  
server-side program not limited for security reasons, can access files & databases
- increased power  
server machines tend to be more powerful, better tools
- code integrity  
do not have to give client access to source code or data in order to execute

## Common server-side applications

### search engines

- must maintain a large database of links & documents
- must be able to index, sort data, perform complex searches
- requires lots of storage, optimum performance → server-side

### database access

- Web page can serve as front-end to a database
- make requests from browser, passed on to Web server, calls CGI program to access the database, sends the results back to the browser

### chat & bulletin board services

- user enters messages in a Web interface, passed on to server
- chat: CGI program distributes messages to all connected users
- bulletin board: CGI program adds to accessible database of messages

## CGI programming

### 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

### to execute a CGI 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 back

## HTTP messages

### recall the format of HTTP messages

- message headers contain specific information  
e.g., response headers include status code, date, last-modified, ...
- a blank line follows the headers
- the text of the HTML document follows

```
HTTP/1.1 200 OK
Date: Wed, 24 Jan 2001 20:29:50 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Tue, 23 Jan 2001 06:52:39 GMT
ETag: "a9260-18a-39b49837"
Accept-Ranges: bytes
Content-Length: 394
Content-Type: text/html

TEXT OF HTML DOCUMENT
```

## 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

- which is then passed on by the Web server as an HTTP header

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

At minimum, output can be plain text

- which is passed on by the Web server as the HTML document

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

## CGI example

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

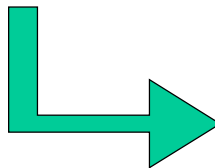
int main()
{
    cout << "Content-Type: text/html" << endl
          << endl;

    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



## CGI file access

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

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

```
4
Live long and prosper.
An apple a day keeps the doctor away.
Don't do anything I wouldn't do.
Life is a bowl of cherries.
```

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

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

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

    int numFortunes;
    ffile >> numFortunes;

    srand((unsigned)time(NULL));
    int find = rand() % numFortunes;

    string line;
    getline(ffile, line);
    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



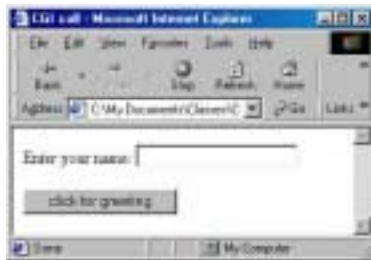
## CGI input

CGI programs can accept input (provided via HTML forms)

```
<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>
```



when a submit button is clicked,  
data in the form is submitted

- data 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+0%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;

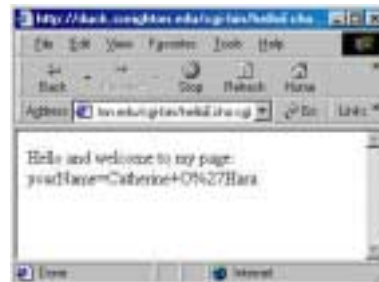
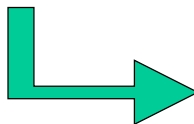
    cout << "Content-Type: text/html" << endl
          << endl;

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

    return 0;
}
```

reads URL-encoded data  
from cin

displays it (unaltered)



## Decoding URL-encoding

need to be able to

- separate elements & values
- replace special characters ('+' → '%20', '%27' → "'", ...)

can define a class to encapsulate CGI input routines

```
CGIinput(); // reads input string, parses, URL decodes,
            // and stores in private data fields

int NumElements(); // returns # of element/value pairs

string Element(int i); // returns ith element name
string Value(int i); // returns ith element value

string ElementValue(string element);
                    // returns value that corresponds to
                    // the specified element name
```

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class CGIinput
{
public:
    CGIinput() {
        // SEE NEXT SLIDE
    }

    int NumElements() {
        return elements.size();
    }

    string Element(int index) {
        return elements[index];
    }

    string Value(int index) {
        return values[index];
    }

    string ElementValue(string desiredElement) {
        for (int i = 0; i < elements.size(); i++) {
            if (elements[i] == desiredElement) {
                return values[i];
            }
        }
        return "NOT FOUND";
    }

private:
    vector<string> elements; // the names of the elements from the page
    vector<string> values; // the corresponding values for the elements

    string URLdecode(string input)
    {
        // SEE NEXT SLIDE
    }
};
```

## CGIinput class

## CGIinput class (cont.)

```
CGIinput()
// constructor, reads input string, parses & decodes,
// and stores element/values pairs in private vectors
{
    string input;
    cin >> input;

    input = URLdecode(input) + "&";

    while (input != "") {
        int equalPos = input.find("=");
        int ampPos = input.find("&");

        elements.push_back(input.substr(0, equalPos));
        values.push_back(input.substr(equalPos+1, ampPos-equalPos-1));

        input = input.substr(ampPos+1, input.length());
    }
}

string URLdecode(string input)
// returns input string with characters URL decoded
{
    string clean = "";
    for (int i = 0; i < input.length(); i++) {
        if (input[i] == '+') {
            clean += ' ';
        }
        else if (input[i] == '%') {
            const string digits = "0123456789ABCDEF";
            clean += (char)(digits.find(input[i+1])*16 + digits.find(input[i+2]));
            i += 2;
        }
        else {
            clean += input[i];
        }
    }
    return clean;
}
```

## POST example (cleaned up)

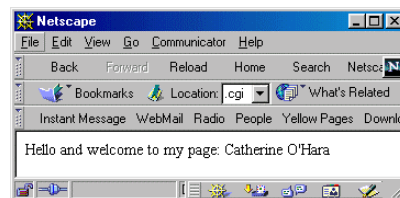
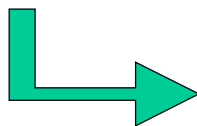
```
// helloEchoClean.cpp
#include <iostream>
#include <string>
#include "CGIinput.h"
using namespace std;

int main()
{
    CGIinput cgi;

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

    cout << "Hello and welcome to my page: "
         << cgi.ElementValue("yourName") << endl;

    return 0;
}
```



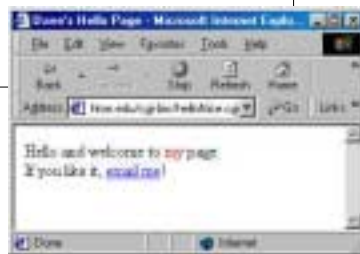
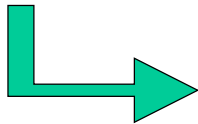
## HTML formatted output

```
// 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;
}
```



## Database example

suppose we want to store email addresses in a database

- Web page front-end allows user to enter desired name
- CGI program looks up name in database (here, a file)
- program returns the email addresses as HTML



## Email database example

```
. . .
int main()
{
    CGIinput cgi; // READ INPUT STRING
    string name = cgi.ElementValue("person"); // AND EXTRACT NAME

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

    cout << "<html>" << endl
         << "<head><title>Math/CS Email Search</title></head>" << endl
         << "<body>" << endl << "Search results for: " << name << "<br><br>" << endl;

    string nameLine, addressLine;
    bool found = false;
    ifstream efile("email.txt"); // OPEN FILE
    while (getline(efile, nameLine)) { // REPEATEDLY, READ NAME
        getline(efile, addressLine); // AND ADDRESS FROM FILE
        if (name == "" || toUpper(nameLine).find(toUpper(name)) != string::npos) {
            found = true; // IF MATCHES, THEN OUTPUT
            cout << nameLine << ": " << "<a href='mailto:"
                 << addressLine << "'>" << addressLine << "</a><br>" << endl;
        }
    }
    efile.close(); // CLOSE FILE

    if (!found) { // IF NO MATCHES FOUND
        cout << "No matching names were found. Please try again. <br>" << endl;
    }
    cout << "</body></html>" << endl;

    return 0;
}
```

## Email database example

```
<html>
<head>
  <title>Creighton Math/CS Email Database</title>
</head>

<body>
<form action="http://duck.creighton.edu/cgi-bin/emailDB.cgi" method="post">
  Enter the person's name: <input type="text" name="person">
  <br><br>
  <input type="submit" value="click for email address">
</form>
</body>
</html>
```

note: could improve user interface with frames

## Next week...

### CGI and other server-side technologies

- more CGI
- server-side tools

read Chapters 27-29

as always, be prepared for a quiz on

- today's lecture (moderately thorough)
- the reading (superficial)