CSC 551: Web Programming

Spring 2004

Server-side programming & CGI

- server-side vs. client-side
  - advantages, common applications
- Common Gateway Interface (CGI)
  - HTTP messages, GET vs. POST
  - CGI program ←→ HTML client
  - input: URL-encoding, form processing
  - output: HTTP headers, HTML formatting
- CGI examples
  - email database, online grades, NCAA pool

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
- server 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: Thu, 22 Jan 2004 18:35:24 GMT
Server: Apache/1.3.27 (Unix) PHP/4.1.2
Last-Modified: Tue, 13 Jan 2004 17:38:00 GMT
ETag: "155005-1a4-40042cf8"
Accept-Ranges: bytes
Content-Length: 420
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

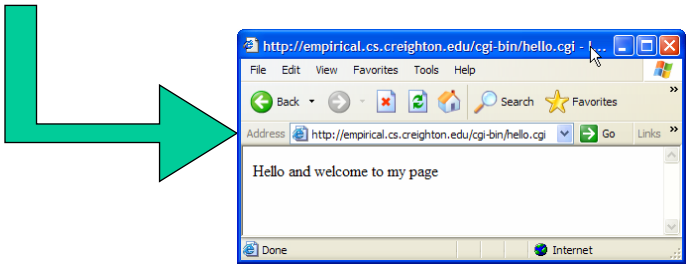
e.g., `Hello and welcome to my page`

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 the program



CGI file access

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

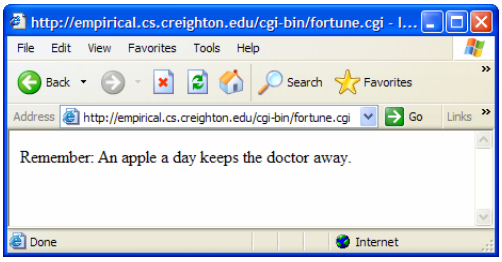
Here, `fortune.txt` contains various fortunes/cliches, one per line

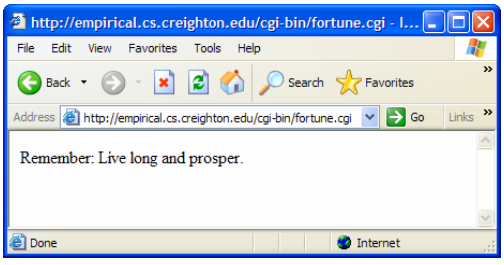
```
Live long and prosper.
An apple a day keeps the doctor away.
Don't do anything I wouldn't do.
Life is a bowl of cherries.
```

`fortune.cpp` reads the fortunes, picks one at random, and displays it in a page

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

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

  vector<string> fortunes;
  string line;
  while (getline(ffile, line)) {
    fortunes.push_back(line);
  }

  ffile.close();

  srand((unsigned)time(NULL));
  int pick = rand() % fortunes.size();

  cout << "Content-Type: text/html" << endl
       << endl;
  cout << "Remember:  " << fortunes[pick] << endl;

  return 0;
}
```


fortune.cgi

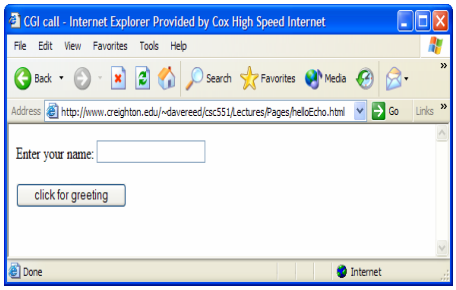




CGI input

CGI programs can accept input (provided via HTML forms)

```
// helloEcho.html
<html>   <head><title>CGI call</title></head>

<body>
<form action="http://empirical.cs.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+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: empirical.cs.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.,

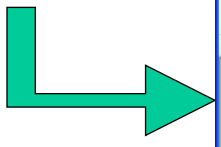
```
POST /cgi-bin/helloEcho.cgi HTTP/1.1
Host: empirical.cs.creighton.edu

yourName=Dave
```

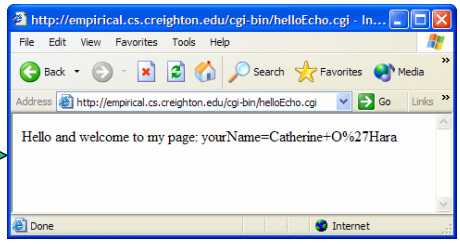

POST example

```
// 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 ('+' → ' ', %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
```


CGIinput class

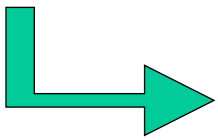
```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

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
    }
};
```

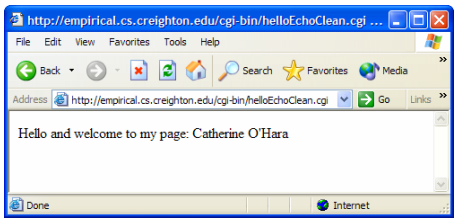
CGIinput class (cont.)

```
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)

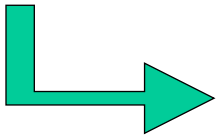
```
// helloEchoClean.cpp
#include <iostream>
#include <string>
#include "CGIinput.h"
using namespace std;

int main()
{
  CGIinput cgi;

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

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

  return 0;
}
```

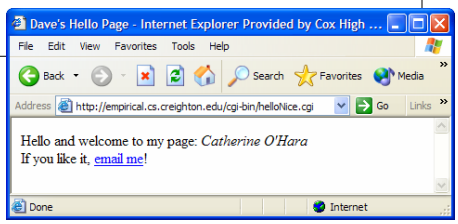




HTML formatted output

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

int main()
{
  CGIinput cgi;

  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 my page: <i>" << cgi.ElementValue("yourName")
       << "</i><br />" << endl
       << "If you like it, "
       << "<a href='mailto:davereed@creighton.edu'>email me</a>!" << endl
       << "</body></html>" << endl;

  return 0;
}
```

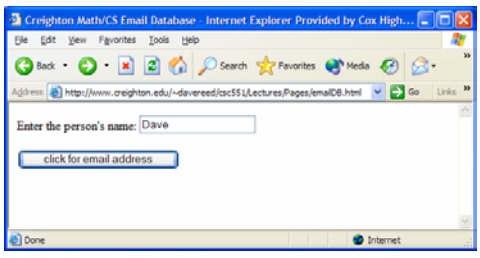


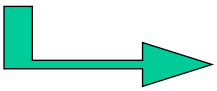


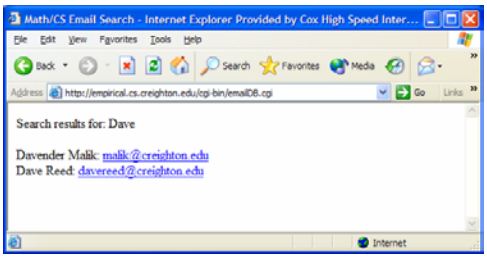
Database example

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

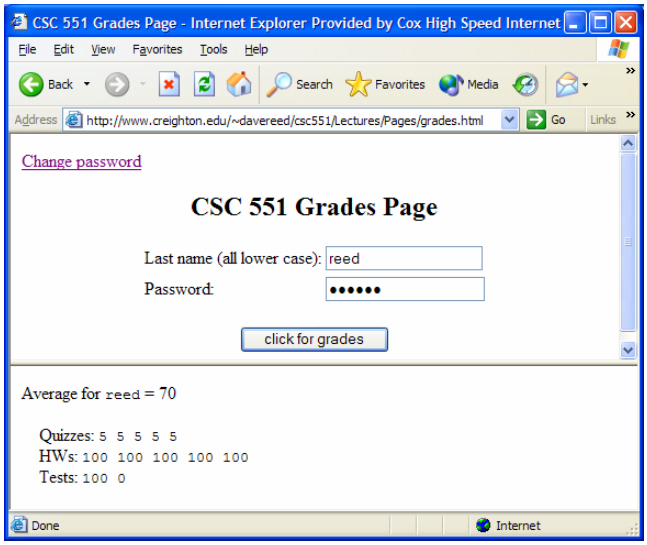
```
. . .

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://empirical.cs.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

Online grades access



want to:

- allow students to access grades
- hide password from view
- allow student to change password

requires server-side:

- must store grades & passwords on server
- allow access based on ID & password

Implementation

for simplicity, we will store grades in a file

- login ID & password on first line
- quiz, HW, test grades on subsequent lines

```
reed foobar
5 5 5 5 5
100 100 100 100 100
100 0
smith changeme
0 0 0 0 0 0 0 0
0 0 0 0 0 0
0 0
.
.
.
```

to look up grades:

- pass login ID and password to CGI program
- program must read lines from file:
  - look for matching login ID & password
  - if found, output individual grades
  - compute overall average and display
- keep a flag to determine if no match found
  - display "No match found" message

```
. . .
int main()
{
    CGIinput cgi;                                          // READ INPUT STRING
    string login = cgi.ElementValue("login");              // AND EXTRACT USER
    string passwd = cgi.ElementValue("passwd");            // LOGIN & PASSWORD

    cout << "Content-Type: text/html" << endl << endl;     // PRINT HEADER
    cout << "<html><head><title>CSC551 Grade Report</title></head><body>" << endl;

    ifstream ifstr("grades.txt");

    string fileLogin, filePasswd, fileQuizzes, fileHWs, fileTests;
    bool found = false;
    while (!found && ifstr >> fileLogin) {
      ifstr >> filePasswd;
      getline(ifstr, fileQuizzes); getline(ifstr, fileQuizzes);
      getline(ifstr, fileHWs);     getline(ifstr, fileTests);

      if (fileLogin == login && filePasswd == passwd) {
        cout << "Average for <tt>" << login << "</tt> = "
             << average(fileQuizzes, fileHWs, fileTests) << " <br><br>" << endl
             << "    Quizzes:  <tt>" << fileQuizzes << "</tt> <br>" << endl
             << "    HWs:  <tt>" << fileHWs << "</tt> <br>" << endl
             << "    Tests:  <tt>" << fileTests << "</tt> <br>" << endl;
        found = true;
      }
    }
    if (!found) {
      cout << "Invalid login.  Please try again.<BR>" << endl;
    }
    cout << "</body></html>" << endl;

    return 0;
}
```


Grades interface

in input frame, allow user to enter login ID and password

submit using POST

direct response to output frame

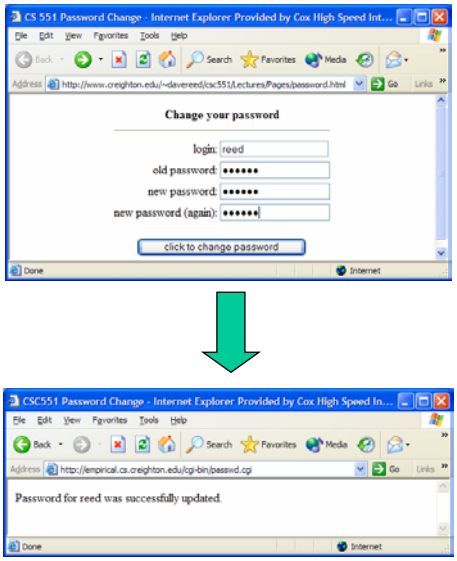
```
<html>
<head>
  <title>CSC 551 Grades Page</title>
</head>

<body>
  <a href="password.html" target="_blank">Change password</a><br>
  <div style="text-align:center">
    <h2>CSC 551 Grades Page</h2>

    <form action="http://empirical.cs.creighton.edu/cgi-bin/student.cgi"
          method="post" target="output">
      <table border=0>
        <tr><td>Last name (all lower case): <td><input type="text" name="login" />
        <tr><td>Password: <td><input type="password" name="passwd" />
      </table>
      <br /><br />
      <input type="submit" value="click for grades" />
    </form>
  </div>
</body>
</html>
```

Changing the password

to change the password, must be able to rewrite the file

- pass login ID, old & new passwords to CGI program
- as the program reads from "grades.txt", echo data to "temp.txt"
- when find matching login ID and password, substitute new password
- if password was changed, then copy "temp.txt" back to "grades.txt"

note: CGI file access presents serious security problems (later)



password program

```
. . .
int main()
{
    // CODE FOR READING CGI INPUTS, OUTPUTTING HTTP HEADER

    if (newPasswd1 != newPasswd2) {
      cout << "INPUT ERROR: you must enter the new password twice. <br>"
           << "<font color='red'><blink>PASSWORD NOT UPDATED</blink></font>" << endl;
    }
    else {
      ifstream ifstr("grades.txt");
      ofstream ofstr("temp.txt");

      string fileLogin, filePasswd, fileQuizzes, fileHWs, fileTests;
      bool found = false;
      while (ifstr >> fileLogin) {
        ifstr >> filePasswd;
        getline(ifstr, fileQuizzes); getline(ifstr, fileQuizzes);
        getline(ifstr, fileHWs);     getline(ifstr, fileTests);

        if (!found && fileLogin == login && filePasswd == oldPasswd) {
          ofstr << fileLogin << " " << newPasswd1 << endl << fileQuizzes << endl
                << fileHWs << endl  << fileTests << endl;
          cout << "Password for " << login << " was successfully updated. <br>" << endl;
          found = true;
        }
        else {
          ofstr << fileLogin << " " << filePasswd << endl << fileQuizzes << endl
                << fileHWs << endl << fileTests << endl;
        }
      }
      ifstr.close();
      ofstr.close();

    . . .
```

password (cont.)

```
      . . .

      if (!found) {
        cout << "INPUT ERROR: no match for login ID " << login << ". <br>"
           << "<font color='red'><blink>PASSWORD NOT UPDATED</blink></font>"
           << endl;
      }
      else {
        ifstream newIfstr("temp.txt");
        ofstream newOfstr("grades.txt");

        string line;
        while (getline(newIfstr, line)) {
          newOfstr << line << endl;
        }
        newIfstr.close();
        newOfstr.close();
      }
    }

    cout << "</body></html>" << endl;

    return 0;
}
```


Password interface

in separate window, allow user to enter login ID, old and new passwords

submit using POST

```
<html>
<head>
  <title>CSC 551 Password Change</title>
</head>

<body>
  <form action="http://empirical.cs.creighton.edu/cgi-bin/passwd.cgi" method="post">
    <div style="text-align:center">
    <table>
      <tr><th align="middle" colspan=2>Change your password
      <tr><td colspan=2><HR>
      <tr><td align="right">login: <td><input type="text" name="login" />
      <tr><td align="right">old password: <td><input type=password name="old" />
      <tr><td align="right">new password: <td><input type="password" name="new1" />
      <tr><td align="right">new password (again): <td><input type="password" name="new2" />
    </table>
    <br /><br />
    <input type="submit" value="click to change password" />
    </div>
  </form>
</body>
</html>
```

Serious security concerns

who owns the process when a cgi (or other server-side) program is executed?

- the process is owned by the Web server program (e.g., www)
- any file to be read/written by a cgi program must be readable/writable to www

as long as only trusted users can define server-side programs, this is fine

- can have other users on the system, but file protections will disallow direct access

but as soon as users are given the ability to define server-side programs, they can write a program that accesses any www accessible file!!!

- SOLUTION 1: don't allow users to define server-side programs (e.g., bluejay)
- SOLUTION 2: utilize a password-protected database application for secure data

NCAA tourney example

text boxes are arranged in a table

user selects teams by clicking on text boxes: ONCLICK event-handler copies box contents to next round box

when user clicks submit button, prompted for name (stored as hidden), then all data is sent to CGI program

NCAA implementation

CGI program parses input and saves in a file

- gets submitter's name, stores in a file by that name

`player=Dave+Reed` → `Entries/Dave_Reed`

- if a file with that name already exists, disallow entry
- could have CGI program generate HTML page for each entry

the generation of individual HTML pages & scoring is handled by UNIX scripts

NCAA CGI program

```
. . .

int main()
{
  CGIinput cgi;

  string name = cgi.ElementValue("player");
  for (int j = 0; j < name.length(); j++) {
    if (name[i] == ' ') {
      name[i] = '_';
    }
  }
  string fname = "Entries/" + name;

  cout << "Content-Type: text/html" << endl << endl;  // PRINT HEADER
  cout << "<html><head><title>NCAA Pool Submission</title></head><body>" << endl;

  ifstream ifstr(fname.c_str());
  if (ifstr) {
    ifstr.close();
    cout << "Submission failed: there is already a submission under that name." << endl;
  }
  else {
    ifstr.close();
    ofstream ofstr(fname.c_str());
    for (int i = 0; i < cgi.NumElements(); i++) {
      ofstr << cgi.Element(i) << "  " << cgi.Value(i) << endl;
    }
    ofstr.close();
    cout << "Submission accepted.  Good luck" << endl;
  }
  cout << "</body></html>" << endl;
  return 0;
}
```

CGI programming in Perl

while CGI programming can be done in any language (e.g., C++). most real-world CGI programming is done in Perl

Perl (Practical Extension and Reporting Language) was developed/implemented by Larry Wall at NASA's JPL in 1987

- originally designed as a UNIX shell-scripting language
- based on `awk`, it provided extensive text-processing operations
- it evolved to provide support for sockets, modules, objects, CGI, …

attractive features

- free language with lots of free applications
- simple scripting language (many of the advantages of JavaScript)
- applications are portable, requires Perl interpreter
- provides a CGI interface module: `CGI.pm`

hello.pl

in UNIX, can make the file executable by placing a directive at the top

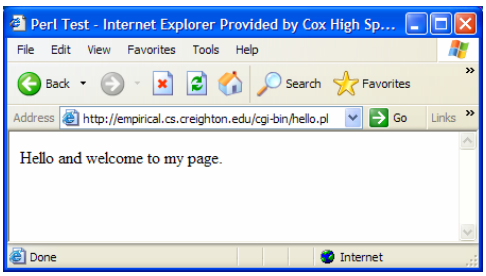
note that CGI format is the same as with C++

- Content-type: text/html
- blank line
- HTML to be displayed

Perl uses print to display text

- '\n' specifies newline
- string concatenation via '.'

```
#!/usr/bin/perl
#
# hello.pl        Dave Reed
# Silly program to display a greeting in a Web page.
########################################################

print("Content-type: text/html\n\n");
print("<html>\n<head><title>Perl Test</title>" .
      "</head>\n<body>\n");
print("Hello and welcome to my page.\n");
print("</body>\n<html>\n");
```



hello1.pl

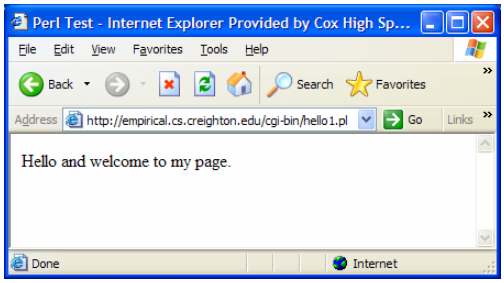
printing the HTML tags & content (& '\n's) can get tedious

- Perl provides a shorthand notation for displaying HTML

```
print <<LABEL
…
LABEL
```

anything between the LABELS is treated as HTML text to be displayed as is

```
#!/usr/bin/perl
#
# hello1.pl         Dave Reed
# Silly program to display a greeting in a Web page.
#####################################################

print("Content-type: text/html\n\n");
print(<<HTMLTEXT);
<html>
<head><title>Perl Test</title></head>
<body>
Hello and welcome to my page.
</body>
<html>
HTMLTEXT
```



helloEchoNice.pl

the CGI library contains useful functions

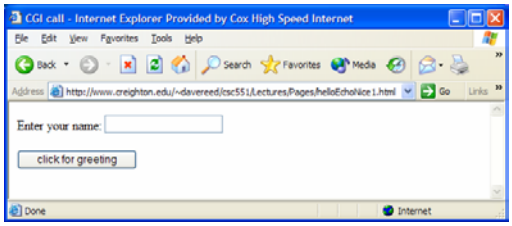
- the param function accesses the value of an element from the input string
- similar to our C++ function cgi.ElementValue("yourName")

Perl variables begin with $

- not typed (as in JavaScript)
- numbers & string types only

- variables can be used in the printed HTML text

```
#!/usr/bin/perl
#
# helloEchoNice.pl        Dave Reed
# Silly program to display a customized greeting.
#################################################

use CGI ":standard";
$name = param("yourName");

print("Content-type: text/html\n\n");
print(<<STARTHTML);
<html>
<head><title>Perl Test</title></head>
<body>
Hello and welcome to my page, <i>$name</i><br />
If you like it,
<a href='mailto:davereed@creighton.edu'>email
    me</a>!
</body>
<html>
STARTHTML
```



fortune.pl

Perl arrays begin with @ (when assigning)

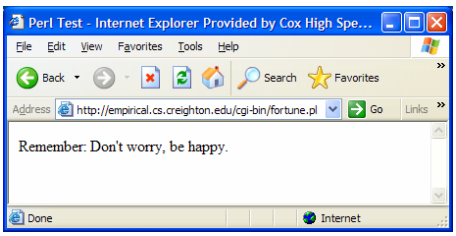
- access array element using `$ARRAY[index]`
- index of last element is `$#ARRAY`

file i/o is very simple

`open(HANDLE, "<fname")`
opens an input file and provides a handle

`@ARRAY = <HANDLE>;` reads all lines from the file and stores in the array

```
#!/usr/bin/perl
#
# fortune.pl        Dave Reed
# Reads fortunes from fortune.txt, randomly picks
# and displays one
#####################################################

open(INFILE, "<fortune.txt") || die("NO SUCH FILE.");
@fortunes = <INFILE>;
$chosen = $fortunes[int(rand($#fortunes+1))];
close(INFILE);

print("Content-type: text/html\n\n");
print(<<STARTHTML);
<html>
<head><title>Perl Test</title></head>
<body>
Remember: $chosen
</body>
<html>
STARTHTML
```



emailDB.pl

control statements are similar to C++/Java/JavaScript

`length(STR)` returns length

`uc(STR)` returns uppercase copy

`index(STR, SUB)` returns index of first occurrence

`substr(STR, START, LEN)` returns substring

view emailDB1.html

```
#!/usr/bin/perl
# emailDB.pl        Dave Reed
#####################################################

use CGI ":standard";
$name = param("person");
open(INFILE, "<email.txt") || die("NO SUCH FILE!");
@emails = <INFILE>;
close(INFILE);

print("Content-type: text/html\n\n");
print(<<STARTHTML);
<html>
<head><title>Perl Test</title></head>
<body>
STARTHTML

$found = 0;
for ($i = 0; $i < @emails; $i += 2) {
  if (index(uc($emails[$i]), uc($name)) != -1) {
    $found = 1;
    print(substr($emails[$i], 0, length($emails[$i])-1) .
          ": " . "<a href='mailto:" .
          substr($emails[$i+1], 0, length($emails[$i+1])-1) .
          "'>" .
          substr($emails[$i+1], 0, length($emails[$i+1])-1) .
          "</a><br />\n");
  }
}
if ($found == 0) {
  print "No matching names were found.  Please try again.\n";
}

print(<<ENDHTML);
</body>
<html>
ENDHTML
```