

Server-side

C++ CGI programs
Server-side scripting



C++ CGI programs

Feasible, but not a good approach!
Just given for Assignment-1



CGI mechanism

- As previously described
 - Web server finding request was for CGI program would
 - start separate process which would “exec” appropriate executable
 - pass data to this program
 - would read the page generated by the CGI program
 - would add the necessary HTTP headers
 - would return the complete response to client



Web-server/CGI communications

- Web-server to CGI
 - Environment variables
 - “Pipe” to CGI’s stdin (cin)
- CGI to Web-server
 - Pipe taking CGI’s stdout (cout)
 - restriction
 - first lines of output must be text that are added to the HTTP header and **must** end with a content-type definition and a blank line



CGI programs

- Any language
 - C common when CGI first invented, soon lost popularity
 - C++ very rare
 - shell popular for a while, recognized as too vulnerable to hackers (and bad coding)
 - perl became most popular of true CGI systems
- Relatively easy
 - Nothing about networking!
 - Simply read from standard input (and environment) and write to standard output.



All CGI programs are the same ... (1)

1. Pick up input data
 - determine whether on stdin (cin) or all in environment variables
2. Read the data
 - get a string
name=value&name=value&name=value
 - names are the field names of the form
 - values are strings derived from form inputs
 - names and values “encoded” (space as + signs, other non-alphanumerics as escape sequences %xx)
3. Breakout the data
 - split up that string to get decoded name value pairs
4. ...

Actually, all web processing programs are the same!



All CGI programs are the same ... (2)



3. ...
4. Create an instance of a data structure with fields that correspond to form fields
5. Use name value pairs that have been read to fill in the fields in this data structure.
 - Ignore any unexpected inputs
 - Check for malicious inputs and discard
6. When all input data consumed, validate data in created data structure
7. ...

You don't have to collect all data into a structure, you can read the data into individual variables and check these separately.

All CGI programs are the same ... (3)



7. If data fail your validation tests (dumb user or hacker attack) return either:
 - canned HTML response saying user error
 - carefully generated form page with some fields filled in with the data you found acceptable and other fields highlighted, page also contains some general directive to user to re-enter the data that were missing or invalid
8. ...

All CGI programs are the same ... (4)



9. Validated data!
Happy to continue.

Data invariably either
 - input that defines some constraints on information to be retrieved from some persistent data store
*find all books with Java in title*or
 - input that will be added to persistent data store
add my email address to your mailing list

All CGI programs are the same ... (5)



- Persistent data
 - virtually always – relational database
 - very rarely – data file
 - Files have problems associated with
 - Access control
 - Concurrency
- CGI program with validated data
 - search
 - update

All CGI programs are the same ... (6)



10. CGI- search
 - open persistent data store
 - loop
 - get next record checked against constraints defined in input
 - if match
 - if first match generate HTML start of response page
 - generate next part of output with details of match
 - at end of available data
 - if no matches found – generate complete response page saying no matches
 - otherwise generate tail of response page listing matches

Of course, if the persistent data store is a database, it will do most or all of the data checking and simply return those records that meet requirements.

All CGI programs are the same ... (7)



11. CGI-update
 - add new record to persistent data store
 - generate a response page confirming that data have been stored.

Common coding issues ...

- All CGI programs need to
 - sort out where input data are
 - read string
 - split apart the string and decode the name/value pairs
- various libraries include functions for this

Simple example

- Form

```
<html>
<head>
<title>Demo form for minimum cgi</title>
</head>
<body>
<form method="get" action="/MinCGI.cgi">
<p>Name</p><input type="text" name="name_input" />
<p>Age</p><input type="text" name="age_input" />
<br />
<input type="submit" value="Invoke CGI" />
</form>
</body>
</html>
```

Very basic C++ program

```
#include <iostream>
#include <stdlib.h>
using namespace::std;
int main(int argc, char** argv)
{
    cout << "Content-type: text/html\n\n";
    cout << "<html><head><title>CGI demo, dynamic response"
         "page</title></head>\n\n";
    cout << "<body><h1 align='center'>Response</h1>\n\n";
    char* str = getenv("QUERY_STRING");
    if(str==NULL) {
        cout << "<p>No QUERY_STRING found!</p>\n\n";
    }
    else {
        cout << "<p>QUERY_STRING (in 'GET' request) was <br />\n\n";
        cout << str; cout << "</p>\n\n";
    }
    cout << "</body></html>\n\n"; return 0;
}
```

Compilation etc

- Would have to compile and link the code and then save the executable as MinCGI.cgi
 - (Web server will have been configured to run programs whose name ends in .cgi)
 - Web server configuration will restrict which directories can be used to hold such programs

Very basic C++ CGI program

- Must start output with content-type and blank line
 - Output from CGI program goes back to web server
 - Web server is checking for this content-type line, if it doesn't receive this information it will generate an error and user will get an automatically generated error page (505 server error)

Very basic C++ CGI program

- Because form specified GET, knew data would be in QUERY_STRING environment variable.
- Didn't bother to decode it – simply echoed it back to user

Your CGI exercise ...

- Exploiting similarity of CGI programs?
 - What is OO supposed to be good for?
 - Capturing similarity!
- You are given some C++ code defining classes that will simplify your implementation of CGI
<http://www.uow.edu.au/~nabg/399/Basics/CGIStuff.html>

CGI Helper classes supplied

- Two classes : Token, CGI_Helper
 - Token
 - owns two strings
 - one is field name
 - other is field value
 - Token objects passed to your code
 - Copy string values into fields of struct
 - If strings supposed to represent number try converting them (if string cannot be converted to number, catch any exception or error and just mark struct as invalid)

CGI_Helper

- Supposed to capture commonality of all simple CGI programs
 - some methods defined (e.g. finding input string and splitting it into Token objects)
 - some default methods defined
 - e.g. default header (content-type, blank line, html tags, title); you can use, but would modify if wanted cookies etc
 - some empty methods, most have to be defined in subclass
- You define subclass
 - provide implementation of those empty methods

```
class CGI_Helper {
public:
    CGI_Helper() { }
    virtual ~CGI_Helper() { }
    // Header, Trailer functions simply compose some
    // standard HTML code for the generated response page
    // (written to stdout, will be returned by Webserver)
    virtual void HTML_Header(const char* title);
    virtual void HTML_Trailer();
    // Handle request
    // determines if Get or Post, loads data appropriately
    // then using Process(), it loops through all tokens in
    // request
    virtual void Handle_Request();
};
```

virtual – developer can replace any of these if desired

```
class CGI_Helper {
protected:
```

```
    // these two for convenience, you might find it useful
    // to have something done by CGI_Helper before/after tokens
    // called from Process()
    virtual void StartTokens() { }
    virtual void EndTokens() { }
    // Process --- that loop through tokens
    virtual void Process(char *data);
    // Override this,
    // typically create a CGI_Helper with link to some other
    // object, ProcessToken can then pass each token to that object
    virtual void ProcessToken(Token *tok) { }
```

ProcessToken – really has to be defined in subclass; StartTokens, EndTokens – not necessarily defined (these functions not “abstract” as empty implementations acceptable)

```
class CGI_Helper {
    ...
private:
    int hexvalue(char ch);
    void ReadString(char *str, char*& data, char*endpt);
    Token *GetToken(char*& data);
    void HandleGet();
    void HandlePost();
};
```

You don't (can't) change these.

Usage

- Define and implement subclass that extends CGI_Helper

```
class EchoCGI : public CGI_Helper {
protected: virtual void StartTokens();
virtual void ProcessToken(Token *tok);
virtual void EndTokens();
};
```

- Little main program that creates and uses instance of your subclass.

```
int main() {
EchoCGI x;
x.HTML_Header("Echoing your name value pairs");
x.Handle_Request();
x.HTML_Trailer();
return 0;
}
```

Echo example

- Program is to process any input form
- Program is simply to list the name, value pairs that were read.
- Output is to be a HTML ordered list of names and values.
- Define EchoCGI as (public) subclass of CGI_Helper

Functions in EchoCGI

- StartTokens
 - Output the HTML for a start of ordered list
- ProcessToken
 - Output a HTML list item with name and value strings
- EndTokens
 - Output the HTML to end ordered list

StartTokens, EndTokens

```
void EchoCGI::StartTokens()
{
cout << "<ol>" << endl;
}
void EchoCGI::EndTokens()
{
cout << "</ol>" << endl;
}
```

ProcessToken

```
void EchoCGI::ProcessToken(Token *tok)
{
cout << "<li>" << tok->Name() << "\t: ";
if(tok->Value() == NULL)
cout << "(not specified)";
else cout << tok->Value();
cout << endl;
}
```

Server-side scripting

SSI
ASP, PHP etc

What is wrong with good old CGI?



- Web server
 - Receives http Get request
 - Is this a request for a file? Yes – return file.
 - Is this a request for processing? Yes:
 - Fork new process
 - New process “execs” program
 - Web server feeds data to new process (via pipe or environment variables)
 - Web server reads response from process, adds HTTP header and returns to client
- Costly

Why the fork(), exec() ...?



- Basic web server:
 - Can deliver unchanged pages itself
 - But has to ask a separate CGI program for any dynamically generated page.
- But often the processing needed to create a new page is “trivial” –
if we could do the work in web-server, we wouldn't need to launch a CGI program.

Doing “trivial” work in the web server ...



- Suppose
 - Most of the contents of a page are fixed
 - A few lines need to be generated dynamically:
 - **Include:** eg a large complex fixed page that includes a short regularly updated segment (travel agency page with current exchange rates in a dynamic page)
 - **File info:** eg file size, date file last modified (*could have a page with a list of files – each with a dynamically specified modification date*)
 - **Date:** eg for a page that is a printable invoice
 - **Parameters:** eg hostname, client IP address, ...

“Trivial” work on Web server ...



- Operations such as those listed could be performed very easily by Web server and would avoid need for separate CGI process.
- *The cost?*
 - Web server has to parse the text of a file that it is returning, checking to see whether it contains directives requiring other files to be included, or extra data such as dates or file sizes.

“Server side includes” – Mark 1



- Solution to cost problem:
 - Separate the pages that require processing by Web server from ordinary pages – eg by using a different file extension.
- Activate processing option in web server
- Server now acts as:
 - Agent for fetching static HTML pages
 - Generator for pages with limited, regular dynamic content
 - Agent for launching CGI

SSI 1



- Server Side Includes probably predates the CGI mechanism – included in NCSA web server (1993/4?).
- Files: .shtml or .shtm or .sht
- Configuration parameters for server (*Apache and similar have configuration files that control operations; to get SSI, you would have had to modify the contents of the configuration file*)
- Appear as special tags in modified HTML document:
`<!--#command parameter="argument"-->`

SSI examples

```
<hr>
Exchange rates:
<br>
<size -1>
As of:
<!--#flastmod file="exchange.txt"-->
<size +1>
<br>
<!--#include file="exchange.txt"-->
<hr>
```

SSI commands

- **config**
 - This command used to set script error messages, size and time formats.
- **echo**
 - Prints values of environment variables.
- **fsize**
 - Prints the size of the specified file, using sizefmt format specification.
- **flastmod**
 - Prints the last modification date of the specified file, using the timefmt format specification.
- **include**
 - This command inserts the text of another document or file into the parsed file (subject to the usual access controls).
- **exec**
 - The exec command executes a given shell command or CGI script.
- **cmd**
 - The server executes the given string using /bin/sh.

*When you've found a
new gimmick ...*

... flog it to death!

“exec”, “cmd” ? !

- SSI started with *trivial* processing.
- Uhm ...
- A C programmer can “*trivially*” launch another process to execute a command:
 - `system(...);`
 - `execl(..., ..., ..., ...);`
- So, *inevitably the scope of SSI grew.*
- Can sub-launch processes,
 - Environment variables are available, can be passed to process
 - Output is piped back and can form content of dynamic page.

Why sub-launch a process?

- *Isn't this simply duplicating CGI?*
!
What do you gain?
- *Cost is same as CGI. Doing the same things ...*
 - Pack environment variables (including those that came via HTTP GET request in a form)
 - Set up pipe communication
 - Launch process
 - Read response, incorporate in page

Why alternative to CGI?

- SSI advocates would probably argue along lines:
 - *CGI: need to put HTML code etc inside a program (Perl, C++, ...); this makes it hard for a "web designer" to tweak appearance of page; web designer has to alter program code (chance of error is high, cost of recompile etc)*
 - *SSI: HTML page designed by web designer, has some place where include output from a program (eg information formatted for a table with <td>, <tr> etc tags); web designer can change overall appearance of page fairly easily*

When you have found a new gimmick, flog it to death! Extended SSI

- So, now SSI scripts can launch processes and retrieve data.
- **Why not make it all programmable?**
 - Eg could have the web-server evaluate data to select which program we want to run.

XSSI

- Next feature added:
"variables" and conditional programming constructs for SSI.
- "set" "variable name" "value"

```
<!--#if expr="test_condition" -->
<!--#elif expr="test_condition" -->
<!--#else -->
<!--#endif -->
```

XSSI

- Test condition:
 - The usual: subexpressions combined using && or || or !
 - Subexpressions involved tests on strings
 - Equality tests
 - Also **pattern matching of regular expressions**
- So, could test QUERY_STRING and see if contained a particular pattern (allowing inspection of response from a Form submitted via GET)

XSSI conditional examples

```

      Name of environment   Regular expression
      variable              pattern
<!--#if expr="\$REMOTE_HOST" = /*.uk" -->
<p>Visit our new store in Oxford Street, London.
<!--#elif expr="\$REMOTE_HOST" = /*.au" -->
<p>Opening soon in Sydney and Melbourne.
<!--#else -->
<p>We have stores in Chicago, Los Angeles, New York,
Paris, London, ...
<!--#endif -->
```

(Tests are checking whether domain name of host ends in "uk" or "au")

XSSI conditional examples

```
<!--#if expr="\$QUERY_STRING" = /*gender=f.*" -->
<ul>
<li><a href="ballet.html">Culture</a></li>
<li><a href="gourmet.html">Food and Drink</a></li>
<li><a href="beauty.html">Beauty</a></li>
</ul>
<!--#else -->
<ul>
<li><a href="footy.html">Culture</a></li>
<li><a href="beer.html">Food and Drink</a></li>
<li><a href="porn.html">Beauty</a></li>
</ul>
<!--#endif -->
```

QUERY_STRING will hold values of fields in a Form that uses "GET";
So can customize response according to data in previous form.

XSSI

- NCSA's basic SSI commands were commonly copied by all web server providers.
- Extensions?
 - *Very much web-server dependent*
 - Either coded as part of web server
 - Or, as with Apache, provided as standard and alternative 3rd party "modules" that could be added to web server.
 - Non-standard

When you have found a new gimmick, flog it to death! Database access!!

- *What work is done by a typical CGI program?*
 - Extract some data from Query_string environment variable
 - Add a new row to a database table
 - Retrieve some data from another table
 - Print retrieved data
- **Why use CGI for that "trivial" work?**
Get the web-server to talk to the database!

Database access for web-server

- At least one of web-servers for PC platform included an "ODBC" command in its extended SSI dialect.

Cold Fusion? You may have seen reference to this product; it combines a web page editor and a web-server; the web-server part has a very extensive repertoire of SSI extensions. Adverts for jobs for Cold Fusion are (probably) mainly targeted at those who understand and can exploit the SSI extensions.

And loop constructs

- If you are going to retrieve data from a database, you are going to have to have loop constructs ...
For each entry in result_set do
Get data from selected columns
Format data into table
- (The code for loop constructs in SSI is too disgusting for public viewing)

Getting messy ...

- So, now have .shtml pages that may contain:
 - HTML directives
 - Content
 - SSI directives
 - Conditional code,
 - Loop constructs
 - Database access
 - Process sublaunch
 - Also Javascript to be run in client's browser once this page delivered
 - *And it all varies depending on which web server you use*
- These replaced (in Web server) by dynamically generated text before page is returned to client*

ASP

Active Server Pages

(an enormously successful Microsoft technology, now officially obsolete)

Microsoft's Internet Information Server

- Microsoft's IIS 4 came with a major advance on previous rather ad-hoc server side scripting features.
- **Active Server Pages:**
 - Same basic idea: web page source combines HTML, content, and script to run on server before page returned.
 - Script language now a full programming language(s) (default, VBScript, based on well known 'Visual Basic', other script languages can be used).
 - Database access standardized.

ASP technology

- ASP scripting:
 - "Visual Basic" VBScript preferred
 - JScript (an ECMAScript compliant scripting language may be used as alternative)
 - This is very confusing!
 - Source page can contain
 - HTML
 - Fixed content text
 - Jscript code that is to be sent to client and get run by browser
 - Jscript code that is to be run by server to generate dynamic content of the page

ECMAScript: a scripting language approved by a European standards body; basically a version of Javascript;

ASP - VBScript

- VBScript
 - has functions, conditionals, loops etc have full repertoire of programming constructs.
 - supports various scalar values (integers, floating-point numbers, strings, etc.), arrays,
 - quasi "object based" language (*no inheritance, but do have objects – instances of classes – that own data and perform actions*)
 - VBScript code will use objects – mainly the "Request" and "Response" objects
 - Request: *holds environment variables, form input data etc*
 - Response: *well, really it's the page that contains the VBScript, mostly write to the "Response" object when generate some dynamic data that are to be displayed in browser*

Just what you always wanted ... a "Hello World" program, ASP-style

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<TITLE>Hello Viper</TITLE>
</HEAD>
<BODY>
<P>
<% Response.Write("Hello World") %>
</BODY>
</HTML>
```

Looping in VBScript ...

```
<%
Dim I ' declare a loop index variable
%>
<%
For I = 1 To 5 Step 1
' Output HTML and text; font size varies
%>
<FONT SIZE="<%= I %>">Hello World</FONT><BR>
<% Next %>
```

ASP

- Wildly successful.
- Web-page editors exist that allowed
 - "graphics designers" to do their stuff and make an attractive page
 - Programmers to add script at appropriate points
- This is the advantage claimed by scripters who rate it as much more important than the efficiency gain
 - "HTML page with little escaped sections for code"
 - Not "code with output statements containing HTML strings"

HTML containing code



- Really the claim is only good for simple scripts.
- When the processing that is required gets complex, the page ends up either
 - having little fragments of HTML separated by huge chunks of scripting code
 - Starting with a huge set of script functions and in effect a script main program, then having a tiny HTML section that has a call to this main.

Alternative server scripting technologies ...



Alternative technologies



- ASP is probably (still!) the most common.
- PERL
 - Perl had rapidly established its role as a popular scripting language for CGI programs
 - "mod-Perl" was developed (just before ASP) as an Apache module that allows a Web server to execute a Perl script to generate a page
 - Still more in style of "content and layout embedded in code" as opposed to ASP ("code embedded in content and layout")
 - Same cost as ASP (avoids sub-launch of CGI program with Perl interpreter)
- ...

Alternative technologies : PHP3/4/5



- Pretty much a direct rival to ASP.
 - PHP code embedded in amongst all the HTML layout directives and content text of a page
 - Web-server incorporates interpreter; before a page is returned, it is processed by the interpreter – this results in changes in content (code replaced by output – eg table produced by code that gets information from database)
 - Fairly simple database access.
 - Ad hoc language; C origins, Perl influences

PHP origins



- Personal Home Page Tools, started late 1994, created by Rasmus Lerdorf
 - simple parser engine
 - a few special macros
 - utilities (guestbook, page visit counter, etc)
 - interface to mySQL engine
- Rapid growth; became "open source" cooperative project
- Several million web servers using this technology

Example



```
<html><header>
<title>Form Example</title>
</header>
<body><h1>Welcome</h1>
<?if($name):?>
Hi <?echo $name?>,
you are <?echo $age?> years old
<?endif?>
</body></html>
```

PHP scripting -



- Essentially same advantages and problems as ASP
- Seems more like parallel universes:
 - Microsofties
Internet Information Server => ASP
 - Everyone else:
Apache => PHP (integrated)
IIS => PHP generally as a CGI program
- (Windows/XP, Windows/Vista => IIS+.NET – a much more sophisticated approach than ASP)

PHP



- Origin – hacker community
- Today – professional
 - Zend
 - IBM
 - Oracle
 - ...