10.1 The Common Gateway Interface

- Markup languages cannot be used to specify computations, interactions with users, or to provide access to databases

- CGI is a common way to provide for these needs, by allowing browsers to request the execution of server-resident software

- CGI is just an interface between browsers and servers

- An HTTP request to run a CGI program specifies a program, rather than a document

  - Servers can recognize such requests in two ways:

    1. By the location of the requested file (special subdirectories for such files)

    2. A server can be configured to recognize executable files by their file name extensions

- A CGI program can produce a complete HTTP response, or just the URL of an existing document

10.2 CGI Linkage

- CGI programs often are stored in a directory named `cgi-bin`

- Some CGI programs are in machine code, but Perl programs are usually kept in source form, so `perl` must be run on them

- A source file can be made to be "executable" by adding a line at their beginning that specifies that a language processing program be run on them first

  For Perl programs, if the `perl` system is stored in `/usr/local/bin/perl`, as is often is in UNIX systems, this is

```
#!/usr/local/bin/perl -w
```

- An HTML document specifies a CGI program with the hypertext reference attribute, `href`, of an anchor tag, <a>, as in

```
<a href =
   "http://www.cs.uccs.edu/cgi-bin/reply.pl>"
Click here to run the CGI program, reply.pl
</a>
```

10.2 CGI Linkage (continued)

```
<!-- reply.html - calls a trivial cgi program
     -->
 <html>
<head>
<title>
   HTML to call the CGI-Perl program reply.pl
</title>
</head>
<body>
This is our first CGI-Perl example
<a href =
      "http://www.cs.ucp.edu/cgi-bin/reply.pl">
Click here to run the CGI program, reply.pl
</a>
</body>
</html>
```

- The connection from a CGI program back to the requesting browser is through standard output, usually through the server

- The HTTP header needs only the content type, followed by a blank line, as is created with:

```
print "Content-type: text/html \n\n";
```

10.2 CGI Linkage (continued)

```
 #!/usr/local/bin/perl
# reply.pl – a CGI program that returns a
#            greeting to the user

print "Content-type: text/html \n\n",
      "<html> <head> \n",
      "<title> reply.pl example </title>",
      " </head> \n", "<body> \n",
      "<h1> Greetings from your Web server!",
      " </h1> \n </body> </html> \n";
```

10.3 Query String Format

- A query string includes names and values of widgets

- Widget values are always coded as strings

- The form of a name/value pair in a query string is: name=value

- If the form has more than one widget, their values are separated with ampersands

```
milk=2&payment=visa
```

- Each special character is coded as a percent sign and a two-character hexadecimal number (the ASCII code for the character)

- Some browsers code spaces a plus signs, rather than as `%20`

10.4 The `CGI.pm` Module

- A Perl module serves as a library

- The Perl `use` declaration is used to make a module available to a program

  - To make only part of a module available, specify the part name after a colon

    (For our purposes, only the `standard` part of the CGI module is needed)

    ```
    use CGI ":standard";
    ```

- Common `CGI.pm` Functions

  - "Shortcut" functions produce tags, using their parameters as attribute values

    - e.g., `h2("Very easy!");` produces
      `<h2> Very easy! </h2>`

    - In this example, the parameter to the function `h2` is used as the content of the `<h2>` tag

10.4 The `CGI.pm` Module (continued)

- Tags can have both content and attributes
  - Each attribute is passed as a name/value pair, just as in a hash literal
    - Attribute names are passed with a preceding dash

```
textarea(-name => "Description",
         -rows => "2",
        -cols => "35"
 );
```

Produces:

```
<textarea name ="Description" rows=2
          cols=35> </textarea>
```

10.4 The `CGI.pm` Module (continued)

- If both content and attributes are passed to a function, the attributes are specified in a hash literal as the first parameter

```
a({-href => "fruit.html"},
    "Press here for fruit descriptions");
```

Output:
```
<a href="fruit.html">
    Press here for fruit descriptions</a>
```

- Tags and their attributes are distributed over the parameters of the function

```
ol(li({-type => "square"},
        ["milk", "bread", "cheese"]));
```

Output:
```
<ol>
  <li type="square"milk</li>
  <li type="square"bread</li>
  <li type="square"cheese</li>
</ol>
```

- `CGI.pm` also includes non-shortcut functions, which produce output for return to the user

- A call to `header()` produces:

```
Content-type: text/html;charset=ISO-8859-1
```
-- blank line --

10.4 The CGI.pm Module (continued)

- The `start_html` function is used to create the head of the return document, as well as the <body> tag

  - The parameter to `start_html` is used as the title of the document

```
    start_html("Bill’s Bags");
```

```
DOCTYPE html PUBLIC
   "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml11-transitional.dtd">
<html xmlns=
  "http://www.w3.org/1999/xhtml lang="en-US">
<head><title>Bill’s Bags</title>
</head><body>
```

- The `param` function is given a widget’s name; it returns the widget’s value

  - If the query string has `name=Abraham` in it,

    `param("name")` will return `"Abraham"`

- The `end_html` function generates `</body></html>`

→SHOW `popcorn.html` , its display, and `popcorn.pl`

10.5 A Survey Example

- We will use a form to collect survey data from users

- The program needs to accumulate survey results, which must be stored between form submissions

    - Store the current results in a file on the server

    - Because of concurrent use of the file, it must be protected from corruption by blocking other accesses while it is being updated

        - Under UNIX, this can be done with the Perl function, `flock`, using the parameter value 2 to specify a lock operation and 8 to specify an unlock operation

--> SHOW `conelec.html` and its display

- Two CGI programs are used for this application, one to collect survey submissions and record the new data, and one to produce the current totals

- The file format is eight lines, each having seven values, the first four for female responses and the last four for male responses

10.5 A Survey Example (continued)

- *The program to collect and record form data must*:

1. Decode the data in the query string
2. Determine which row of the file must be modified
3. Open, lock, read, unlock, and close the survey data file
4. Split the affected data string into numbers and store them in an array
5. Modify the affected array element and join the array back into a string
6. Open, lock, write, unlock, and close the survey data file

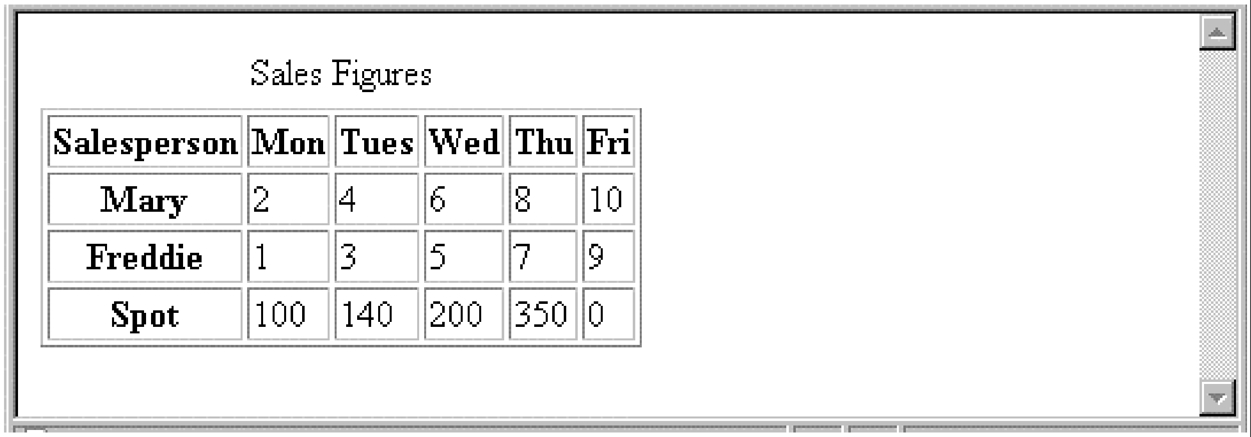
--> SHOW `conelec1.pl`

10.5 A Survey Example (continued)

- Tables are easier to specify with `CGI.pm`
    - The table is created with the `table` function
    - The `border` attribute is specified as a parameter
    - The table's caption is created with a call to `caption`, as the second parameter to table
    - Each row of the table is created with a call to `Tr`
    - A heading row is created with a call to `th`
    - Data cells are created with calls to `td`
    - The calls to `Tr`, `th`, and `td` require references as parameters
    - Suppose we have three arrays of sales numbers, one for each of three salespersons; each array has one value for each day of the work week
        - We want to build a table of this information, using `CGI.pm`

10.5 A Survey Example (continued)

```
table({-border => "border"},
       caption("Sales Figures"),
       Tr(
           [th(["Salesperson", "Mon", "Tues",
                "Wed", "Thu", "Fri"]),
            th("Mary").td(\@marysales),
            th("Freddie").td(\@freddiesales),
            th("Spot").td(\@spotsales),
                  ]
              )
       );
```



Sales Figures

Salesperson	Mon	Tues	Wed	Thu	Fri
Mary	2	4	6	8	10
Freddie	1	3	5	7	9
Spot	100	140	200	350	0

10.5 A Survey Example (continued)

- *The program that produces current results must*:

1. Open, lock, read the lines into an array of strings, unlock, and close the data file

2. Split the first four rows (responses from females) into arrays of votes for the four age groups

3. Unshift row titles into the vote rows (making them the first elements)

4. Create the column titles row with `th` and put its address in an array

5. Use `td` on each rows of votes

6. Push the addresses of the rows of votes onto the row address array

7. Create the table using `Tr` on the array of row addresses

8. Repeat Steps 2-7 for the last four rows of data (responses from males)

10.5 A Survey Example (continued)

--> SHOW `conelec2.pl`
--> SHOW Figure 10.7

10.6 Cookies

- A *session* is the collection of all of the requests made by a particular browser from the time the browser is started until the user exits the browser

- The HTTP protocol is stateless

  - But, there are several reasons why it is useful for the server to relate a request to a session

    - Shopping carts for many different simultaneous customers

    - Customer profiling for advertising

    - Customized interfaces for specific clients

  - *Approaches to storing client information:*

    - Store it on the server – too much to store!

    - Store it on the client machine - this works

10.6 Cookies (continued)

- A cookie is an object sent by the server to the client

- Cookies are created by some software system on the server (maybe a CGI program)

- Every HTTP communication between the browser and the server includes information in its header about the message

- At the time a cookie is created, it is given a lifetime

- Every time the browser sends a request to the server that created the cookie, while the cookie is still alive, the cookie is included

- A browser can be set to reject all cookies

- `CGI.pm` includes support for cookies

```
cookie(-name => a_cookie_name,
       -value => a_value,
       -expires => a_time_value);
```

  - The name can be any string
  - The value can be any scalar value
  - The time is a number followed by a unit code (`d`, `s`, `m`, `h`, `M`, `y`)

10.6 Cookies (continued)

- Cookies must be placed in the HTTP header at the time the header is created

```
header(-cookie => $my_cookie);
```

- To fetch the cookies from an HTTP request, call `cookie` with no parameters

  - A hash of all current cookies is returned

- To fetch the value of one particular cookie, send the cookie's name to the `cookie` function

```
$age = cookie( age );
```

- *Example:*

  A cookie that tells the client the time of his or her last visit to this site

  - Use the Perl function, `localtime`, to get the parts of time

```
($sec, $min, $hour, $mday, $mon, $year,
      $wday, $yday, $isdst) = localtime;
```

→ SHOW `day_cookie.pl`

10.7 Animation Using CGI

- CGI was once a good way to create animation, but now there are several better ways

- There are two ways to use CGI to create animation, neither of which requires user intervention

1. *Client-pull animation*

    - The client repeatedly requests images from the server, which it displays in sequence

        - *Problems*: Internet is not fast enough, and if the approach were widely used, it would pull down the speed of the whole Internet

2. *Server-push animation*

    - The server sends the sequence of images to the client, with delays between them

        - *Problems*: Also creates a huge load on the Internet, and it is supported only by Netscape