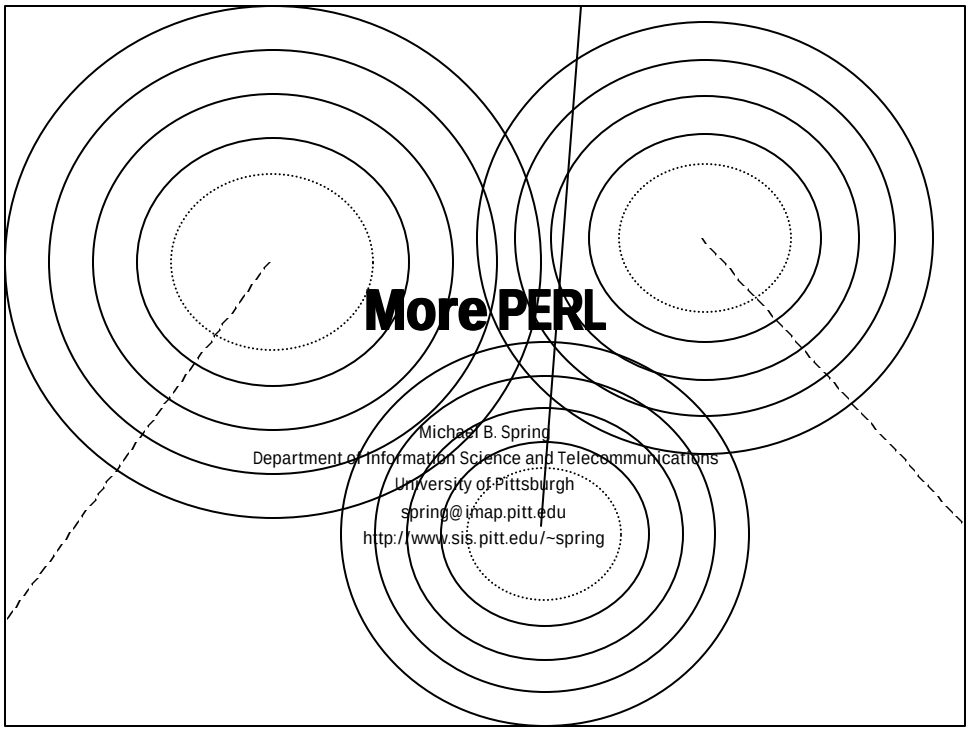

Overview

- File I/O
- Subroutines
- Perl Modules and Libraries
  - CGI
  - DBMS
- Pattern Matching and Parsing Example

File I/O

- Review
  - @var = <FILEHNDLE> reads an entire file
  - $var = <FILEHNDLE> reads one line
  - $var = <FH> -s <FH> reads entire file in one line
  - getc <FILEHANDLE> reads a single character
- Random access
  - seek <FILEHANDLE> offset position
  - offset is the offset from position – it can be negative
    - position is:
      - 0 == beginning
      - 1 == current position
      - 2 == end
  - Tell <FILEHANDLE> returns the current offset in a file

Formatted Output

- A format may be specified for a list of variables using the following form

```
format MYFORMAT
@<<<.< @|||| $@>>.>>
$var1, $var2, $var3
.
```

  - The . terminates the format
  - The "picture" defines the length and alignment of the variables. In the example above:
    - $var1 is output flush left in 4.1 format
    - $var2 is output centered as 5 chars
    - $var3 is output flush right at 5.2 preceded by a dollarsign

Formats Continued

- Once a format is defined, the current values of the variables are used when the write command is given as follows:

```
write MYFORMAT
```

- Headers and footers can be defined by creating formats with associated names
  - MYFORMAT_TOP would define the page header for MYFORMAT writes
  - There are also special variables that can be consulted to define current print status, e.g. $% is the current page number and $- is the number of lines left

Subroutines

- Subroutines are most easily declared prior to being referenced
- The arguments to a subroutine call are:
  - Accessible through the array @_
  - Each element can be accessed as $_[n] where n is the number of the array element
- A subroutine is normally invoked using an & preceding the name
- The next page shows a simple perl script that produces a directory list for a web server

A simple subroutine

```
#! /opt/bin/perl

sub print_element {
  chomp $_[0];
  $i=rindex "$_[0]","\/";
  $filename=substr $_[0],$i+1;
  $href="http://www2.sis.pitt.edu/~spring/".
   $filename;
  print "<LI>".
   "<A href=\"$href\">$filename</A></LI>\n";
}
```


Main body invoking subroutine

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

print "<HTML><HEAD><TITLE>Directory
  Listing</TITLE></HEAD>\n";
print "<BODY><H1>Directory Listing for Home
  Directory</H1>\n<UL>\n";
foreach $file (`ls
  /home/spring/public_html/*.html`)
  {
  &print_element($file)
  }
print "</UL></BODY></HTML>\n\n";
```

Libraries and Modules

- The PERL language is extended through the use of modules and libraries
- To include subroutines from a library – which has a default extension of pl:
  - require library.pl
- To include subroutines from a module – which has the default extension pm:
  - use module.pm
- You can use a module with the qw form to use standard function calls
  - use CGI qw /:standard/;

Some Libraries and Modules

- ActivePerl comes with more than a hundred libraries and modules which include code for debugging, system access, file control, etc.
- The website www.cpan.org contains hundreds of additional modules that allow, among other things:
  - Authentication, Security and Encryption
  - World Wide Web, HTML, HTTP, CGI, MIME
  - Images Manipulation, Drawing and Graphing
  - Mail and Usenet News
  - OS Interfaces Networking
  - Database Interfaces User Interfaces

CGI Module

- The CGI module provides tools for preparing web pages in response to form submissions
- This review only looks at the object oriented use
- The first step is to create a CGI object
  - q= new CGI;
- there are various forms which allow you to include a query object in the new CGI
  - q = new CGI ({'abc'=>'def'});
  - nb, use curly braces for any routines taking names arguments

Creating Headers and Document

- Create the one needed header
  - $q->header;
  - $q->header({-type=>'image/gif',-expires=>'+3d' });
- Create a simple normal html document
  - $q->start_html('hello world'); # start the HTML
  - $q->h1('hello world'); # level 1 header
  - $q->end_html; # end the HTML

Different forms of an h1 call;

- Code
  - Generated HTML
- $q->h1()
  - <H1>
- $q->h1('some','contents');
  - <H1>some contents</H1>
- $q->h1({-align=>left});
  - <H1 ALIGN="LEFT">
- $q->h1({-align=>left},'contents');
  - <H1 ALIGN="LEFT">contents</H1>

Parameters and Values

- To get the names of all the parameters
  - @names = $q->param
- To get all the values of a given parameter – note that the array is important for names that have multiple values
  - @values = $q->param('foo');
- To get just one value
  - $value = $q->param('foo');
- To set or append parameter values
  - $q->param(-name=>'foo',-values=>['an','array','of','values']);
  - $q->append(-name=>'foo',-values=>['yet','more','values']);
- To delete parameters and values
  - $q->delete('foo');
  - $q->delete_all();

Options for start_html

- The start function can be as simple as:
  - $q->start_html();
- Or more complex than:
  - $q->start_html(-title=>'Secrets of the Pyramids',
    -author=>'fred@capricorn.org',
    -base=>'true',
    -target=>'_blank',
    -meta=>{'keywords'=>
      'pharaoh secret mummy',
      'copyright'=>'copyright 1996 King Tut'},
    -style=>{'src'=>'/styles/style1.css'},
    -BGCOLOR=>'blue');

Obtaining the script's url

- There are several ways to get information about the URL of the script:
  - $full_url = $q->url();
  - $full_url = $q->url(-full=>1); #alternative syntax
  - $relative_url= $q->url(-relative=>1);
  - $absolute_url= $q->url(-absolute=>1);
  - $url_with_path = $q->url(-path_info=>1);
  - $path&query = $q->url(-path_info=>1,-query=>1);
  - $netloc= $q->url(-base => 1);

Creating embedded HTML Elements

- Elements can be embedded in the strings of a call

```
$q->blockquote(
    "Many years ago on the island of",
    $q->a({href=>"http://crete.org/"},"Crete"),
    "there lived a Minotaur named",
    $q->strong("Fred.")),
```

- Some elements have start and end forms

```
$q->start_form(        -method=>$method,
    -action=>$action,
    -enctype=>$encoding);
<... various form stuff ...>
$q->endform;
```


The form elements

- All of the form elements have calls like:
- $q->textfield(-name=>'field_name',
    -default=>'starting value',
    -override=>1,
    -size=>50,
    -maxlength=>80);
- $q->textarea(-name=>'foo',
    -default=>'starting value',
    -rows=>10,
    -columns=>50);

Some of the Others

- $q->popup_menu('menu_name',
  ['eenie','meenie','minie'],
  'meenie');
- $q->scrolling_list('list_name',
  ['eenie','meenie','minie','moe'],
  ['eenie','moe'],5,'true');
- $q->checkbox_group(
  -name=>'group_name',
  -values=>['eenie','meenie','minie','moe'],
  -default=>['eenie','moe'],
  linebreak=>'true',
  labels=>\%labels);

Buttons

- $q->submit(-name=>'button_name',
  -value=>'value');
- $q->image_button(-name=>'button_name',
  -src=>'/source/URL',
  -align=>'MIDDLE');
- $q->button(-name=>'button_name',
  -value=>'user visible label',
  -onClick=>"do_something()");

A simple Script

- use CGI;
- #create a new CGI object
- $q=new CGI;
- # write out our header
- print $q->header;
- # now we can write out an HTML page
- # print $q->starthtml("This is a form processor");

Continued

```
print $q->h1("Here are the parameters");
print $q->start_ul
foreach $key ($q->param)
   {
   print "<LI>The Variable: ".$key.
         " had the value ". $q->param{$key}."</LI>";
   }
print $q->end_ul;
print $q->end_html;
```

DBMS Access

```
use Win32::ODBC;

my ($db)=new Win32::ODBC('contacts');
$db->Sql("SELECT * from Contacts");
while ($db->FetchRow()){
my (%data)= $db->DataHash();
print "$data{'FirstName'}"." "."$data{'LastName'}"."\n";
}

$db->Close();
```


Some parsing

```
#! /opt/bin/perl
$mainfile="raw.dat";
open DATA, $mainfile;
read DATA, $filestring, -s DATA; # read whole file
print $filestring;
@lines = split /\n/, $filestring; # split on newlines
foreach (@lines)
    { print $_."\n";}
foreach (reverse @lines) # print lines in reverse order
   { print $_."\n";}
```

Parsing (continued)

```
print "\nTotal lines in the file: ".$#lines."\n";
$i=0;
foreach (@lines)
    { if (/\@image/i)  # if line contains @image, print
          { print  $_ ."\n";  $i++; }
    }
print "\nTotal lines with \@image: ".$i."\n";
$i=0;
foreach (@lines)
    { if (/the/i) # if line contains characters the, print
          { print  $_ ."\n"; $i++; }
    }
print "\nTotal lines with the characters 'the': ".$i."\n";
```


Parse (continued)

```
$i=0;
foreach (@lines)
    { if (/\bthe\b/i)
          { print  $_ ."\n"; $i++; }
    }
print "\nTotal lines with the word 'the': ".$i."\n";
$i=0;
foreach (@lines)
    { if (/\bthe\b/i)
          {foreach (split / /, $_)
               { if (/\bthe\b/i)
                    {# print  $_ ."\n"; $i++; }
               }
    }}
print "\nTotal number of 'the's: ".$i."\n";
```