COMP111: Software Tools

Final Exam – Dickson Chiu

Spring 2001 (May 31, 2001 Thursday 13:30-15:30)

Student Name: _____

Student Number: _____

Email: _____

Lab Section: _____

Instructions:

1. There are 8 problems worth 100 points total.
2. Check that you have all 10 pages.
3. Close book, close notes, work on your own, and you cannot use any computer.
4. Answer all questions in the space provided. Rough work can be done only on the back pages.
5. Leave all pages stapled together.
6. The examination period will last for **120 minutes.**
7. Stop writing immediately when the time is up.

For Grading Purposes Only:

Page 2: Problems 1	_____	/ 19
Page 3: Problems 2-4	_____	/ 20
Page 4: Problems 5-6	_____	/ 11
Page 5: Problems 7	_____	/ 9
Page 6: Problem 8a	_____	/ 6
Page 7: Problem 8b	_____	/ 4
Page 8: Problem 8c	_____	/ 6
Page 9: Problem 8d	_____	/ 14
Page 10: Problem 8e,f	_____	/ 11
Total:	_____	/100

1) (19 points) Output Identification. What are the output of the following Perl code fragments?

Perl Code Fragments	***Output***	***Marks***
`$a="b"; $b="a"; $aa="ab"; $ab="ba";` `print $$a, 1, $$$a, 2, ${a.$a};`	a1b2ba	3

`@a = (3,5,7,9);`
`$tmp = $a[0];`
`$a[0] = $a[-2];`
`$a[$#a] += $tmp;`
`$a[@a-3] = 6;`
`$a[-1] = 1;`
`print @a;`
	7671	2

`$n = 2;  @a = 1;`
`@b = ($n, @a, 4);`
`unshift(@a, 3);`
`push(@a, 3);`
`$n = pop(@a);`
`unshift(@a, @b);`
`@a = reverse(@a);`
`push(@a, @b);`
`@a = sort(@a);`
`pop(@a);`
`print @a;`
	1112234	3

`$bill{"Gates"} = "rich";`
`@bill{"Chan ","Mr."} = qw(very big);`
`@bill{4..7} =`
`  qw(very famous + nervous);`
`delete($bill{5});  delete($bill{6});`
`@a = sort values %bill;`
`%bill = reverse %bill;`
`print @a;`
	bignervousrichveryvery	2.5

`$n1 = 9; $n2 = 2; $n3 = 6;`
`$n1 = test($n2,$n3);`
`print "$n1 $n2 $n3";`
`sub test{`
`  my($n1,$n2) = @_;`
`  print "$n1 $n2 ";`
`  return $n2 -= $n1;`
`}`
	2 6 4 2 6	2.5

`$n = 2;`
`while($n--){`
`  if($n > 10){ last; }`
`  $n += 2;`
`  if($n <= 3){ next;}`
`  $n += 3;`
`  if($n <= 7){ redo;}`
`  $n += 1;`
`}`
`print $n;`
	12	2

`$_ = "03-5-2000";`
`s/(\d+)/($1)/g;`
`print;`
	(03)-(5)-(2000)	2

`$_ = "the quick brown fox";` `print /^(\w+)[\w\s]+\s+(\w+)$/;`	thefox	2

2) (10 marks) Regular Expression matching Below is a list of regular expressions and a list of strings. Each of the strings has a letter. Fill in the blank by each regular expression listing the letters for all of the strings for which that regular expression is true. **Scoring:** 2 points if answer is correct and complete; 1 point if answer contains some but not all of the correct letters; 0 points if *any* incorrect letters or if left blank. The first entry is an example: the expression `/o/` matches all of the strings except `"The answer is 42"`, so put `"A,B,C"` in the blank after `/o/`:

Regular Expressions	**Strings:**
`/o/` ____A,B,C	A. "Hello World"
1. `/\w$/` ___A,B,D,E_	B. "Perl is so much fun"
2. `/o\S/` _____A,C___	C. "http://www.perl.com/"
3. `/[^a-z]{3,}/` ____C,D,E__	D. "The answer is 24"
4. `/\d+/` _____D,E___	E. "20-Feb-2001"
5. `/\D/` __A,B,C,D,E_	

3) (2 marks) What does the acronym Perl stands for?

P__Practical____ E___Extraction___ and R____Report____ L____Language___

4) (8 marks) Substitution – Provide your regular expression to satisfy user's requirement.

Username requirement	*String ~= your regular expression;*
Strip leading spaces from the string	s/^ +//
Replace all non-alphabetic characters with '.' in the string	s/\w/./g
Swap the first word with the second word in the string. (Assume first word starts at the first character and there is only a single space between each word.) E.g. "I am John" -> "am I John"	s/(^\w+) (\w+)/$2 $1/
Strip HTML tags from the string (Hint: be careful of the greedy regular expressions deleting extra stuff…)	s/<(^>)*>//g

5) (5 marks) Mark Six. Write a Perl program to generate and print six **different** random numbers, in the range of 1 to 47. One way to do it is to initialize an array *a* of 47 elements. Then swap a random element from *a*[0..46] with *a*[46], another random element from the rest *a*[0..45] with *a*[45], another random element from the rest *a*[0..44] with the *a*[44] and so on, till you get all the six numbers.

```
#!/usr/local/bin/perl5
# 1 mark per line

@m = (1..47);

for ($i = 46; $i > 40; $i--) {
        $c = int(rand($i));
        ($m[$i],$m[$c]) = ($m[$c],$m[$i]);
         print "$m[$i] ";
}
```

6) (6 marks) Write a Perl subprogram "dist" that calculates the distance of two multi-dimensional points p_0 and p_1 given by the following formula: $\sqrt{\sum_i (p_{1i} - p_{0i})^2}$. Each point is passed to the subprogram as a pointer to a 1-dimensional list. Check that the two points have the same dimension (i.e., the two lists have the same number of elements) otherwise return *undef*. For example, dist([1,2,4,4],[2,3,3,5]) returns 2.

```
#!/usr/local/bin/perl5

sub dist {                                              # 0.5 mark
   @p1 = @{$_[0]};                                      # 0.5 mark
   @p2 = @{$_[1]};                                      # 0.5 mark
   return undef  if ($p1 != $p2);                       # 1 mark

   $sum = 0;                                            # optional, no mark
   for ($i=0;$i<@p1;$i++) {                             # 1 mark
      $sum += ($p2[$i] - $p1[$i]) * ($p2[$i] - $p1[$i]);     # 1.5 mark: ** 2 ok
   }
   return sqrt($sum);                                   # 1 mark: ** 0.5 ok
}
```

7) (9 marks) Write a Perl program (called “delr”) that recursively scans a user-specified sub-directory tree. Print out and then delete any file which ends with “.c” or “.h”. The directory name is specified by the first command-line parameter, while the printer name by the second parameter. If the user does not specify exactly 2 parameters, display a usage summary of your script. Display an appropriate error message if the first parameter is not a valid directory. (You must not use the “find” or “rm” utility, but you may use the printing utility in the form of “lpr –P<printer> <file>”.)

```
#!/usr/local/bin/perl5
# 1 mark each line unless otherwise stated

sub dirtree {                       #0.5
   my @files = <$_[0]/*>;     # local variable required
   foreach (@files){
        if (/\.c$|\.h$/) {
                system("lpr","-P$ARGV[1]","$_");
                unlink($_);              #0.5
        }
        dirtree($_) if (-d $_);  # recursion
   }
}
die(“Usage – prints out and delete files end with .c or .h: delr <dir> <printer> \n”)
        if (@ARGV <> 2);
die(“Invalid directory\n”) if (!(-d $ARGV[0]));
dirtree($ARGV[0]);
```

8) (41 marks) Web-based supplier database Perl cgi program

Based on the techniques you learnt from the class and labs, step by step according to the instructions below, build a Perl cgi program "supplier.cgi" for the following functions:

(A) **Always** display a drop down menu of the fields to be matched, a text box for the pattern and a query button.
(B) If the user pressed the "query" or "Update" button, read a disk file **"supplier.dat"** and initialize a Perl hash data-structure
(C) If the user pressed the "query" button, display the matching records in an editing mode.
(D) If the user pressed the "Update" button, change the internal Perl hash structure according to the parameters from the web, and writes back the data file **"supplier.dat"**.

(a) Main Program (6 marks)

Assume you have the following Perl subprograms (you have to write them later), write the required main Perl program by calling them:

- *readfile("filename")* – reads the file specified by a parameter "filename" (you may assume it is in the correct format) and initialize a Perl hash structure.
- *showmenu* – display an input menu as described by function (A) above.
- *editdata* – display the records and allows editing of editable fields with text box.
- *updatedata* – updates the internal Perl hash structure according to the parameters from the web
- *writefile("filename")* – writes data from the internal Perl hash structure to the file specified by a parameter "filename".

(You should use the CGI module in the main program and remember the header and ending of the web page.)

```
#!/usr/local/bin/perl5
# 0.5 mark each line

use CGI qw(:standard);

print header();
print start_html();
showmenu();

if (param("Query")) {
   readfile("supplier.dat");
   editdata();
}

if (param("Update")) {
   readfile("supplier.dat");
   updatedata();
   writefile("supplier.dat");
}
print end_html;
```

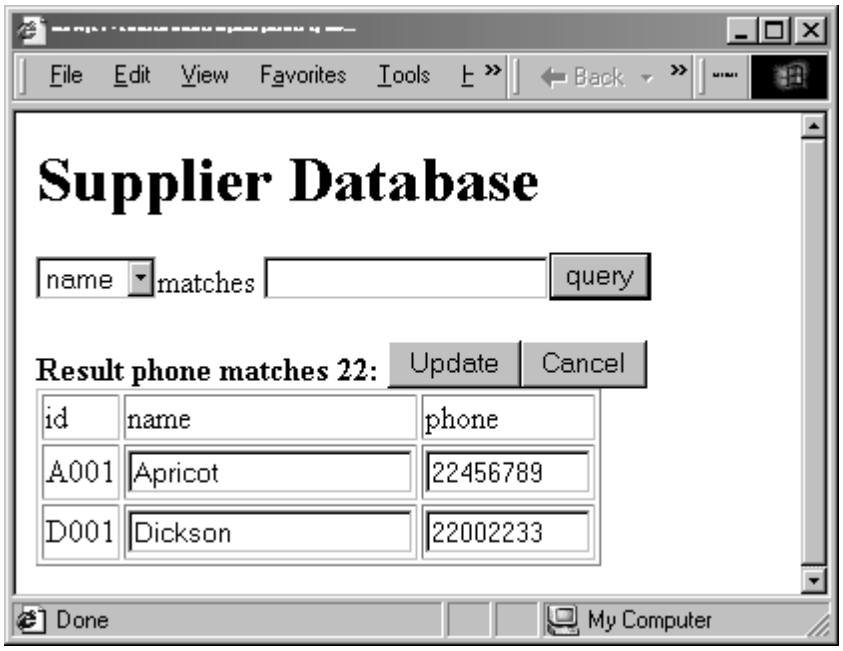

Fig. 1 - Sample Screen

(b) (4 marks) Display the Initial Screen Input Menu.

Write the Perl subprogram (called *showmenu*) to display a drop down menu of the fields to be matched, viz. “phone” and “name” (to be posted as parameter “field”), a text box for the matching pattern (parameter “pattern”) and a query submit button (parameter “query”).

```
sub showmenu {   # rest: 0.5 mark each line unless otherwise stated

   print start_form();
   print h1(“Supplier Database”);
   print popup_menu("field",["name","phone"]);              # 1 mark
   print " match ";
   print textfield("pattern");
   print submit("Query");
   print end_form;
}
```

(c) (6 marks) Reading the Data File and Initialize the Internal Perl Hash Structure.

Write the Perl subprogram (called *readfile*) to read the file specified by a parameter “filename” (you may assume it is in the correct format, but there can be any number of lines) and initialize the Perl hash structure.
File format - Each supplier record is in a separate line, and is the following format:

"supplier_number","supplier_name", “phone"

E.g.

“A001”,”Apricot”,”22456789”
“D001”,”Dickson”,”22002233”

Then, the internal Perl hash structure should contain:

supplier{‘A001’}{‘name’}=”Apricot”;
supplier{‘A001’}{‘phone’}=”22456789”;
supplier{‘D001’}{‘name’}=”Dickson”;
supplier{‘D001’}{‘phone’}=”22002233”;

```
sub readfile {              # rest 0.5 mark unless otherwise stated

    open(INFILE,$_[0]);
    while (<INFILE>) {
       @fields = split(/\,/,$_);   # 1 mark

       $fields[0] =~ /\w+/;
       $id = $&;

       $fields[1] =~ /[^"]+/;
       $name = $&;
       $supplier{$id}{'name'} = $name;

       $fields[2] =~ /[^"]+/;
       $phone = $&;
       $supplier{$id}{'phone'} = $phone;

    }
}
```

(d) (14 marks) Screen for Display and Editing the Data.

Write the Perl subprogram (called *editdata*) to display the selected records and allow editing of editable fields with textboxes. You should display only the records that contain text matching the field specified by parameter "field" to the pattern specified by parameter "pattern". Display them in ascending order of supplier number and in a table format. **For simplicity, only the fields "name" and "phone" are allowed for editing.** Include an "Update" button, so that pressing it triggers update of the data. Include a "Cancel" button, so that when pressing it, the web page returns to the initial screen. (See *Fig.1*)

When you display the records, add hidden fields like id1=A001, id2=D001, etc., to each record, so that these parameters are set when a user presses the "Update" button. The other corresponding parameters will be, name1, name2, etc.; phone1, phone2, etc., and so on. Then, you can process records with id1, id2, etc., until id*n* is not defined.

```
sub editdata {    # rest: 0.5 mark unless otherwise states

   $field = param("field");
   $pattern = param("pattern");

   print "Result $field matches $pattern :";
   print start_form();
   print submit("Update");
   print submit("Cancel");

   print "<TABLE border=1>";
   print "<TR><TD>ID</TD><TD>Name</TD><TD>Phone</TD></TR>\n";   # 1mark
# print tr(td(ID), td(Name), td(Phone));
   $j = 1;                                          # 0.5 mark combined with j++
   foreach $s (sort(keys(%supplier))) {             # 1 mark
      if ($supplier{$s}{$field} =~ /$pattern/) {    # 1 mark
         $name = $supplier{$s}{'name'};
         $phone = $supplier{$s}{'phone'};
         print "<TR><TD>$s";                        # 1 mark
         print hidden("id$j",$s);                   # 1 mark
         print "</TD><TD>";                         # no mark
         print textfield("name$j",$name);           # 1 mark
         print "</TD><TD>";                         # no mark
         print textfield("phone$j",$phone);         # 1 mark
         print "</TD></TR>\n";                      # 1 mark
         $j++;                                      # 0.5 mark combined with j =1
      }
   }
   print "</TABLE>";
   print end_form;
}
```

(e) (5 marks) Updating the data.

Write the Perl subprogram (called *updatedata*) to update the records in the internal Perl hash structure. The records are specified by the parameters posted from the web page, id1, id2, …, etc., whose corresponding field values are contained name1, name2, …, etc.; and, phone1, phone2, …, etc.

```
sub updatedata {   # rest: 0.5 mark each line
                   # total 1 mark for use of xxx$i
   $i = 1;
   while (param("id$i")) {

      $id = param("id$i");
      $name = param("name$i");
      $phone = param("phone$i");

      $supplier{$id}{'name'} = $name;
      $supplier{$id}{'phone'} = $phone;

      $i++;
   }
}
```

(f) (6 marks) Writing back the data file.

Write the Perl subprogram (called *writefile*) to save data from the internal Perl hash structure to the file specified by a parameter "filename". The format of the file is that for input as specified in part (c).

```
sub writefile {          # rest: 0.5 mark unless otherwise states

   $fn = $_[0];
   open (OUTF,">$fn");                    #  1 mark

   foreach $s (sort(keys(%supplier))) {     # 1 mark

      print OUTF "\"$s\",";
      print OUTF "\"$supplier{$s}{'name'}\",";              # 1 mark
      print OUTF "\"$supplier{$s}{'phone'}\",\"";           # 1 mark
      print OUTF "\n";
   }

   close(OUTF);
}
```