Introduction to Perl Programming: Summary of exercises

Thaddeus Aid
Christopher Yau
Sebastian Kelm
Department of Statistics
University of Oxford

for the Oxford University IT Services

April 17, 2013

A digital version of these notes, as well as a few data files, can be found on the course website. You will be referred to this URL whenever you need to obtain extra files to complete an exercise:

`http://www.stats.ox.ac.uk/~aid/perl/`

If you cannot complete all the exercises during the allotted time, you may wish to finish them at home. If you run into trouble you cannot solve by yourself, you can contact the course teacher by email. Please include the words "IT Services Perl" in the subject line.

`aid@stats.ox.ac.uk`

The purpose of this document is to make finding the exercises easier. The exercises are spread through the course booklet and are gathered here for simplicity.

EXERCISE 1: Hello World

1. Open a text editor (e.g. `gedit`).
2. Start a new text document.
3. Enter the source code for the Hello World program below:

```
#!/usr/bin/perl

use strict;

print "Hello World\n"; # prints "Hello World" to the screen
```

4. Save the file as `first.pl`.
5. Open a terminal window and change to the directory containing `first.pl` using the shell command `cd`.
6. Run the Hello World program by using the command `perl first.pl`. You should see the following output:

```
> perl first.pl
Hello World
```

EXERCISE 2: Numbers

1. Create a new text document and name it `addnumbers.pl`.
2. Enter the following code:

```perl
#!/usr/bin/perl

use strict;

my $x = 1; # assign the value 1 to variable $x
my $y = 3; # assign the value 1 to variable $y

my $z = $x + $y; # assign the sum of $x and $y to variable $z

print "The first number is: $x\n"; # print the value of $x
print "The second number is: $y\n"; # print the value of $y
print "The sum is: $z\n"; # print the value of $z
```

3. Run the program in the terminal. You should see the following output:

```
> perl addnumbers.pl
The first number is: 1
The second number is: 3
The sum is: 4
```

4. Extend the program to calculate and print the difference, product and quotient of `$x` and `$y`.

EXERCISE 3: Strings

1. Create a new text document and name it `addstrings.pl`.
2. Enter the source code below:

```
#!/usr/bin/perl

use strict;

my $x = "Jim";
my $y = "Hendrix";

print "My name is $x $y\n";
print 'My name is $x $y\n';
```

3. Switch to the terminal and run the program.

```
> perl addstrings.pl
My name is Jim Hendrix.
My name is $x $y\n
```

4. Create a new variable to store the string "is a legend" and print out all three strings.

EXERCISE 4: Using Input

1. Write a Perl script to read in a string from the console and print:
   (a) The length of the string
   (b) The reverse of the string
   (c) the upper and lower case version of the string
2. Modify your script to accept two string inputs and prints the concatenation of the two strings separated by a space.

EXERCISE 5: Array functions

1. Create a new script with the following code:

```
#!/usr/bin/perl

use strict;

my @array = ( 1 .. 10 ); # create an array of numbers 1-10

print "The array contains: @array\n";

my $first_element = shift(@array); # remove the first element
    and store in first_element
my $last_element = pop(@array); # remove the last element and
    store in last_element

print "The first and last elements of the array are
    $first_element and $last_element\n";

push(@array, ( -5 .. +5 ) ); # add the numbers -5 to +5 to the
    array

print "The array currently contains: @array\n";

my @sortedarray = sort{$a <=> $b}(@array); # sort the array
    numerically

print "The sorted array contains: @sortedarray\n";

my @new_array = qw(cat dog rabbit turtle fox badger); # create
    a new array using qw

print "@new_array\n";
```

2. Create a new script and the following array:

```
@array = qw( 99players b_squad a-team 1_Boy A-team B_squad 2_Boy);
```

3. Sort the array using the following sorting options:

   (a) Sort numerically in ascending order:

   ```
   @array = sort {$a <=> $b} @array;
   ```

   (b) Sort numerically in descending order (same as before but with `$a` and `$b` swapped):

   ```
   @array = sort {$b <=> $a} @array;
   ```

   (c) Sort alphabetically in a case-insensitive manner:

   ```
   @array = sort {lc $a cmp lc $b} @array;
   ```

4. Create a new script with the following array:

   ```
   @words = qw( The quick brown fox jumps over the lazy dog and runs away );
   ```

5. Using appropriate array access and join functions construct the following strings and store these in a single variable and print to screen:

   "The quick fox jumps over the dog"

   "The brown fox runs away"

   "The lazy dog runs"

   "The dog runs away quick"

   "The quick brown dog runs over the lazy fox"

6. Create a 2d array of people:

   ```
   my @people = (["Clark", "Kent"], ["Lois", "Lane"], ["Bruce", "Wayne"]);
   ```

   - Use `push` to add "Superman" to Clark Kent's sub-array.
   - Use `pop` to remove Bruce Wayne from the matrix.
   - Use a directly indexed scalar add "Reporter" to the third element of Lois Lane's sub-array.
   - Add a third sub-array with the values "Jimmy", "Olsen", "Photographer".
   - Print the resulting matrix to the screen.

- Print only the last names to the screen.

EXERCISE 6: If-else statements

1. Create a new text document and name it `ifthenelse.pl`.
2. Enter the source code below:

```
#!/usr/bin/perl

use strict;

my $x = 5.1;
my $y = 5;

if ( $x > $y )
{
    print "x is greater than y\n";
}
else
{
    print "y is greater than x\n";
}

$x = 5.0;
$y = 5.0;

if ( $x > $y )
{
    print "x is greater than y\n";
}
elsif ( $y > $x )
{
    print "y is greater than x\n";
}
elsif ( $y == $x )
{
```

```
    print "x is equal to y\n";
}
```

3. Switch to the terminal and run the program.

```
> perl ifthenelse.pl
x is greater than y
x is equal to y
```

4. Modify the program to accept the numbers entered by the user using `<STDIN>` and re-run the program to see the changes in the program behaviour.

5. Write a new program that computes the area of a circle with a radius that is specified by the user using `<STDIN>`. The area of a circle is π times the radius of the circle *squared* ($\pi \approx 3.141592654$).

6. Modify the program so that if radius is a negative number the program will print "The radius of a circle must be a positive number".

7. Add a conditional statement to print:

   (a) "This is a big circle" if the area of the circle is greater than 100.

   (b) "This is a small circle" if the area of the circle is less than 100.

EXERCISE 7: Nested and Compound If statements

1. Create a new text document and name it `nestedif.pl`.

2. Enter the source code below:

```
#!/usr/bin/perl

use strict;
```

```
my $x = 5.1;
my $y = 5.1;

if ( $x > 5.0 )
{
    if ( $y > 5.0 )
    {
        print "x and y are greater than 5\n";
    }
}

if ( ( $x > 5.0 ) and ( $y > 5.0 ) )
{
    print "x and y are greater than 5\n";
}
```

3. Switch to the terminal and run the program:

```
> perl nestedif.pl
x and y are greater than 5
x and y are greater than 5
```

4. Modify the program to accept numbers from `<STDIN>` and re-run the program to see the changes in the program behaviour.

5. Using a combination of `and`, `or` and `if` statements or nested statements, write a script to print out the following statements under these salary/bonus scenarios:

   (a) Salary $<$ 100000, Bonus $<$ 100000: "You are not a banker."

   (b) Salary $>$ 100000, Bonus $<$ 100000: "You are banker with no bonus."

   (c) Salary $>$ 100000, Bonus $>$ 100000: "You are banker with a big bonus."

   (d) Salary $<$ 100000, Bonus $>$ 100000: "You won the lottery."

   (e) Salary or Bonus $>$ 100000: "You are buying dinner tonight."

6. In Perl, we can use the `=~` operator to perform pattern matching. The statement `$x =~ /word/` is true if the variable `x` contains the phrase "word". Using this pattern matching operator and conditional statements, write a *case-insensitive* test to see if an input string `x` contains the following text:

Word to find	Print this if found
Chris	Found Chris!
Bells	Ding dong!
Wonder	I was wondering about that too
Land	Air and Sea

   Test your code using the following strings:

   (a) Christmas Time

   (b) The bells are ringing in Wonderland

   (c) Stevie Wonder

   (d) The land of hope and glory

   (e) Wondering about your day

EXERCISE 8:

1. Create a new text document and name it `loops.pl`.

2. Write a script that prints out the numbers 1980 to 2010 using a loop.

3. Modify your script to use a conditional statement and print out "This is a new decade!" for years ending in nought. HINT: Use `$year % 10 == 0` to test if a year is divisible by 10.

4. Use a `while` loop to count backwards from 10, print the numbers and print the line "We have lift off!" when the count reaches zero.

5. Create an array with the following strings as elements:

James Bond 007

Department of Statistics

University of Oxford

Fantastic 4

Use a loop to print "The string '`x`' contains numbers" if `$x` does contain numbers. Print the uppercase version of strings that do not contain numbers.

HINT: The test `$x =~ /[0-9]/` can be used to identify if `x` contains any (single digit) number from 0 to 9.

EXERCISE 9:

1. Download the file `fruit.csv` from the course website:

   `http://www.stats.ox.ac.uk/~aid/perl/`

2. Create a new text document called `readfile.pl` and enter the following code:

```
#!/usr/bin/perl

use strict;

my $infile = "fruit.csv";

open(FH, $infile) or die "Cannot open $infile\n";

    # this bit of code reads (skips) the header line
    <FH>;

    while ( my $line = <FH> )
    {
        chomp($line);
```

```
        my @linedat = split(/,/, $line); # splits the line at
            commas

        my $fruit = $linedat[0];
        my $quantity = $linedat[1];
        my $unitprice = $linedat[2];

        $unitprice = sprintf('%0.2f', $unitprice); # converts
            the unit price into 2 decimal places

        print "We have $quantity of $fruit at $unitprice pounds
             each\n";
    }

close(FH);
```

EXERCISE 10:

1. Create a new text document called `writefile.pl` and enter the following code:

```
#!/usr/bin/perl

use strict;

my $outfile = "myoutfile1.txt";

open(OUTFILE, "> $outfile") or die "Cannot write to $outfile\n"
    ;

    print OUTFILE "This is my first file\n";

close(OUTFILE);
```

2. Using a loop, add some extra code to print the numbers 1, .., 100 to the file.

3. Use conditional statements to print only odd numbers between 1 and 100 to the file.

EXERCISE 11:

1. Create a new text document called `myfilecache.pl` and enter the following code:

```
#!/usr/bin/perl

use strict;

no strict 'refs';

use FileCache maxopen => 16;

my $infile = "departments.csv"; # this is the file to read

# open a file handle
open(INFILE, $infile) or die "Cannot open $infile\n";

    # skip the header line
    <INFILE>;

    # read one line at a time
    while ( my $line = <INFILE> )
    {
        chomp($line);

        # extract data
        ( my $staffid, my $firstname, my $surname, my
            $department, my $employmentstatus ) = split(/,/,
            $line);

        my $name = $firstname . " " . $surname;
```

```
        print "$staffid\t$name\t$department\t$employmentstatus\
            n";
    }

# close file handle
close(INFILE);
```

2. Using the `cacheout` function, extend this program to write a set of files, one for each department, which contain the Staff ID and Names for each person working in that Department. The output files should be named after the department they represent.

3. Modify your program to only include full-time employees (FT).

EXERCISE 12:

1. Download the file `animals.zip` from the course website

   `http://www.stats.ox.ac.uk/~aid/perl/`

   Extract its contents to your home directory. This should create a directory called `animals` with four files in it.

2. Use `glob` to find all the text files in the `animals` directory and store these in an array.

3. Create a single summary file containing a list of all the types of foxes, badgers and rabbits and their numbers. The summary file should have the following headers: with the data following underneath

Species	Type	Number
Fox	Alopex	23

   You will need to:

   (a) Check that each file is a text file.

(b) Create a file handle for each data file and read in the data from each file.

(c) Create a file handle to a summary file and write the animal data to this file.

4. Write some code to delete the file called `sweets.dat` in the `animals` directory. WARNING! *BE CAREFUL WHAT YOU DELETE WHEN DOING THIS!*

EXERCISE 13:

1. Using regular expressions, test whether a string has a valid IP address (IPv4) format.

   Note: IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 172.16.254.1.

   Then using sub-string operations, verify that the address is valid.

2. Write a Perl program to read in an input file containing "Name Surname" lines and produce a second file with the format "Surname, Name" (note the comma after the surname). Use regular expressions to do the string conversion.

3. Write a Perl program to eliminate the blank lines from a text file, e.g. If the source file has the lines:

```
Line 1
Line 2

Line 4

Line 6
```

   Your program should modify this file to become:

```
Line 1
Line 2
Line 4
Line 5
Line 6
```

EXERCISE 14:

1. Download the file `counties.csv` from the course website:

   `http://www.stats.ox.ac.uk/~aid/perl/`

2. Read the contents of the `counties` file using Perl. The file has the following format:

FirstName	Surname	Email	County	Date of Birth
Sopoline	Carpenter	elit@semper.ca	Northumberland	21/04/1995

3. Find the number of people born since the year 2000. Hint: you will need to do some string manipulation to extract the year from the Date of Birth fields.

4. Find the number of people living in each county. Hint: Use a hash table.

EXERCISE 15:

1. Download the file `cars.csv` from the course website:

   `http://www.stats.ox.ac.uk/~aid/perl/`

2. Create a new text document called `hashtable.pl` and enter the following code:

```
#!/usr/bin/perl

use strict;

my $infile = "cars.csv";
my %customerTable = ();

open(INFILE, $infile) or die "Cannot open $infile\n";

     <INFILE>;

    while ( my $line = <INFILE> )
    {
        chomp($line);

        ( my $id, my $name, my $car, my $value) = split(/\t/,
            $line);

        $customerTable{$id} = {
            NAME => $name,
            CAR => $car,
            VALUE => $value,
        };
    }

close(INFILE);

foreach my $customer_id ( sort { $a <=> $b } keys(%
    customerTable) )
{
    my $customer_name = $customerTable{$customer_id}->{NAME};
    my $customer_car = $customerTable{$customer_id}->{CAR};
    my $car_value = $customerTable{$customer_id}->{VALUE};
    print "Customer ID $customer_id: $customer_name owns a
        $customer_car which costs £$car_value\n";
}
```

3. Modify the code to exclude cars with values below $15,000$.

4. Add the following customer cars to the hash table:
   (a) 310468389, Joshua Rankin, MG, 34,000
   (b) 550433311, Josephine Gould, Vauxhall, 4,500
5. Create a separate file for each make of car and write the customer details to each file.

EXERCISE 16:

1. Create a new text document called `subfunc.pl` and enter the following code:

```
#!/usr/bin/perl

use strict;

foreach my $number (1 .. 10)
{
    my $squarenumber = square($number);
    print "The square of $number is $squarenumber\n";
}

sub square
{
    my $ans = $_[0] * $_[0];
    return $ans;
}
```

2. Add a new sub-routine which calculates the cube of the numbers and print out the results to screen.
3. Add a new sub-routine that calculates both the square and cube of the numbers and returns two variables.
4. Write a sub-routine that returns the maximum of two numbers and extend your script to test its correct functionality.

5. Write a sub-routine that calculates the area of a circle and its circumference when passed a radius and extend your script to test its correct functionality.

6. Download the file `sales.zip` and extract its contents:

   `http://www.stats.ox.ac.uk/~aid/perl/`

7. Create a new text document called `subfunc2.pl` and enter the following code:

```
#!/usr/bin/perl

use strict;

my @filenames = ( "files/north2009.txt", "files/south2009.txt",
     "files/east2009.txt", "files/west2009.txt" );

foreach my $file ( @filenames )
{
    my $totalSales = readFile($file);

}

sub readFile
{

}
```

8. Complete the sub-routine `readFile` to read the sales data for each of the regions in 2009. The sub-routine should return the total sales across all individuals working in that region.

9. Repeat for the 2010 files.

10. Have sales improved in 2010?

EXERCISE 17:

1. Create a sub-routine that takes an array of numbers (passed by reference), finds all the unique numbers in the array and returns a sorted array (passed by reference) of those unique numbers, e.g.

```
# an array of numbers
my @numbers = qw( 1 1 2 2 3 3 5 5 7 7 7 9 9 9 9 15 );

# call to a sub-routine called uniqueValues
my $unique_numbers_ref = uniqueValues(\@numbers);

# print out array of unique values
print "@$unique_numbers_ref\n";
```

   should display the numbers `1 2 3 5 7 9 15`.

2. Download the file `cars.csv` from the course website:

   `http://www.stats.ox.ac.uk/~aid/perl/`

3. Create a new text document called `customer.pl` and enter the following code:

```
#!/usr/bin/perl

use strict;

my $infile = "cars.csv";
my %customerTable = ();

open(INFILE, $infile) or die "Cannot open $infile\n";
  <INFILE>;
  while ( my $line = <INFILE> )
  {
    chomp($line);
    ( my $id, my $name, my $car, my $value) = split(/\t/,
        $line);
    $customerTable{$id} = {
      NAME => $name,
      CAR => $car,
      VALUE => $value,
    };
  }
close(INFILE);
```

NOTE: You should re-use code from the previous chapter if you have already done this.

4. Write a sub-routine that takes a hash table as input (passed by reference) and returns an arrays (passed by reference) of all the unique types of car in the hash table, e.g.

```
my $unique_cars_ref = uniqueCars( \%customerTable );
```

5. Write a sub-routine that scans the hash table (passed by reference) and returns an array (passed by reference) of customer names whose cars greater than a value specified by a second input variable, e.g.

```
my $customers_ref = findCars( \%customerTable, $value );
```

6. Write a sub-routine that accepts a string and hash table (passed by reference) as inputs. The sub-routine should search the hash table to find the individual with the name specified in the input string and return the car type and value as two scalars, e.g.

```
( my $car, my $value ) = findCustomer( $name, \%
    customerTable );
```