H

Solutions to Exercises

Listed below are our solutions to the exercises given at the end of Chapters 1 - 9 and 11. Remember that 'There's more than one way to do it', so if you found a different way, that's equally valid too.

Chapter 1

Questions 1, 3, and 4 have no code-based solutions

Question 1_2

Hello Mum (the extended version)

```
#!/usr/bin/perl
#newline.plx
use warnings;

print "Hi Mum.\nThis is my second program. \n";
```

Chapter 2

Question 2_1

Getting some user interaction into our currency conversion program.

```
#!/usr/bin/perl
#Ex2_1.plx
use warnings;
use strict;

print "Currency converter\n\nPlease enter the Yen to pound exchange rate: ";
my $yen = <STDIN>;

#Prompt user to input amounts, and do the conversions:
print "Now please enter your first amount: ";
my $first = <STDIN>;
my $a = ($first/$yen);
```

```
print "Enter your second amount: ";
my $second = <STDIN>;
my $b = ($second/$yen);
print "Enter your third amount: ";
my $third = <STDIN>;
my $c = ($third/$yen);

#turn the results into values with no more than 2 decimal places, and rounded #to
2 places too.
my $mod1 = (int (100*$a)/100);
my $mod2 = (int (100*$b)/100);
my $mod3 = (int (100*$c)/100);

#Get rid of trailing newline character:
chomp ($first, $second, $third);

#print out answers with two decimal place accuracy:
printf ("$first Yen is %.2f pounds\n", $mod1);
printf ("$second Yen is %.2f pounds\n", $mod2);
printf ("$third Yen is %.2f pounds\n", $mod3);
```

Question 2_2

Converting hexadecimal and octal numbers to decimal.

```
#!usr/bin/perl
#Ex2_2.plx
use warnings;

#Convert hex to decimal;
print "Please enter a hexidecimal number to convert to a decimal number : \n";
my $hex = <STDIN>;

#remove newline character:
chomp ($hex);

print ("The decimal value of $hex is : ", (hex ("$hex")), "\n \n");

#Convert octal to decimal:
print "Please enter an octal number to convert to a decimal number : \n";
my $oct = <STDIN>;

chomp ($oct);

print ("The decimal value of $oct is : ", (oct ("$oct")), "\n");
```

Question 2_3

Converting decimal to binary with the bitwise AND operator.

```
#!usr/bin/perl
#Ex2_3.plx
use warnings;

print "Please enter the value(less than 256) you wish to be converted into binary
: \n";
my $bin = <STDIN>;

chomp ($bin);
```

```
print "The binary value of $bin is : ", "\n";

#Use the bitwise and operator to determine the binary value:

print ((128&$bin)/128);
print ((64&$bin)/64);
print ((32&$bin)/32);
print ((16&$bin)/16);
print ((8&$bin)/8);
print ((4&$bin)/4);
print ((2&$bin)/2);
print ((1&$bin)/1);
print "\n";
```

Question 2_4

Did you get them all? The correct answers were:

- $2 + (6/4) - (3*5) + 1 = -10.5$
- $17 + ((-3**3)/2) = -3.5$
- $26 + (3 \wedge (4*2)) = 37$
- $((4 + 3) >= 7) \,||\, (2 \,\&\, ((4*2) < 4)) = 1$

Chapter 3

Question 3_2

Running this program will show you that the elements in the first range are `aa ab ac ad ae ... az ba bb`. The second range holds `a0 a1 .. a9 b0 b1 .. b9`

```
#!usr/bin/perl
#Ex3_2.plx
use warnings;

print ('aa' .. 'bb');
print "\n";
print ('a0' .. 'b9');
print "\n";
```

Question 3

Questioning a hash for phone numbers.

```
#!usr/bin/perl
#Ex3_3.plx
use warnings;

#Create hash with important numbers:

my %numbers = (
                mum => "555-1111",
                dad => "555-2222",
                bro => "555-3333",
                sis => "555-4444"
        );
```

```
#Get persons name:

print "Please enter a name : \n";
my $Name = <STDIN>;
chomp ($Name);

#Find and print the name's number
print "$Name","'s number is $numbers{$Name}", "\n";
```

Question 4

Running the joke machine with a hash. The better jokes are up to you.

```
#usr/bin/perl
#Ex3_4.plx
use warnings;
use strict;

my $que_1 =
   "How many Java programmers does it take to change a light bulb? \n";
my $que_2 =
   "How many Python programmers does it take to change a light bulb? \n";
my $que_3 =
   "How many Perl programmers does it take to change a light bulb? \n";
my $que_4 = "How many C programmers does it take to change a light bulb? \n";

my $answer1 = "None. Change it once, and it's the same everywhere. \n\n";
my $answer2 =
   "One. He just stands below the socket and the world revolves around him.
   \n";
my $answer3 =
   "A million. One to change it, the rest to try and do it in fewer lines.
   \n";
my $answer4 = '"CHANGE?!! \n\n"';

my %jokes = (
             $que_1 => "$answer1",
             $que_2 => "$answer2",
             $que_3 => "$answer3",
             $que_4 => "$answer4"
);

print "$que_1";
sleep 5;
print "$jokes{$que_1}";

print "$que_2";
sleep 5;
print "$jokes{$que_2}";

print "$que_3";
sleep 5;
print "$jokes{$que_3}";

print "$que_4";
sleep 5;
print "$jokes{$que_4}";
```

Chapter 4

Question 1

Adding some error checking to the currency converter against user inconsistency.

```
#!/usr/bin/perl
#Ex4_1.plx
use warnings;
use strict;

my ($value, $from, $to, $rate, %rates);
%rates = (
    pounds          => 1,
    dollars         => 1.6,
    marks           => 3.0,
    "french francs" => 10.0,
    yen             => 174.8,
    "swiss francs"  => 2.43,
    drachma         => 492.3,
    euro            => 1.5
);

print "Enter your starting currency: ";
$from = <STDIN>;
print "Enter your target currency: ";
$to = <STDIN>;
print "Enter your amount: ";
$value = <STDIN>;

chomp($from,$to,$value);

#If this currency does not exist, then execute this subroutine:
while (not exists $rates{$to}) {

   print "I don't know anything about $to as a currency\n";
   print "Re-enter your target currency: ";
   $to = <STDIN>;
   chomp($to);
}

while (not exists $rates{$from}) {

   print "I don't know anything about $from as a currency\n";
   print "Re-enter your starting currency: ";
   $from = <STDIN>;
   chomp($from);
}

$rate = $rates{$to} / $rates{$from};

print "$value $from is ",$value*$rate," $to.\n";
```

Question 2

Looping the guess a number program until the user succeeds.

```
#!/usr/bin/perl
#Ex4_2.plx
use warnings;
use strict;

my $target = 73;
print "Guess my number!\n";
print "Enter your guess: ";
my $guess = <STDIN>;



while ($guess != $target) {

   if ($guess > $target){
      print "Your number is bigger than my number, guess again: \n";
   }

   if ($guess < $target){
      print "Your number is less than my number, guess again: \n";
   }

   $guess = <STDIN>;
}

print "That's it! You guessed correctly!\n";
```

Question 3

While it works, this solution is not very efficient as it tests a lot of numbers for primacy that couldn't possibly be primes. There are other reasons why it's not as fast as it could be. See if your solution is more efficient.

```
#!usr/bin/perl
#Ex4_3.plx
use warnings;
use strict;

print "To what number would you like to calculate the primes?", "\n";
my $n = <STDIN>;
my $i = 3;
print "1 2 ";

# We will keep executing our search until our
# number ($i) reaches the given value:

OUTER: while ($i <= $n){

   #Each time we iterate, we must begin division by 2:
```

```
    my $num = 2;

    #Do checks to see if our value $i is prime:
    for (1 .. $i){
       if (($i % $num == 0) and ($i != $num)){

          #If it is not, then check the next number:
          $i++;
          next OUTER;
       }

       if ($num > sqrt($i)) {
          print "$i ";
          $i++;
          next OUTER;
       }
       $num++;
    }
}
```

Chapter 5

Question 1

English descriptions of regular expressions.

- Matches if "one or more word characters are at the end of the string", referencing `$var`.
- Matches unless "a #-character is at the beginning of the string", referencing `$code`.
- Substitutes globally with '`#`' where "two or more #-characters are in string", referencing `$_`.

Question 2

Add individual solutions to the following code to produce functioning Perl programs:

```
#!/usr/bin/perl
# ex05_02.plx
use warnings;
use strict;

$_ = <<EOF;
            <put contents of gettsyburg.txt here>
EOF
```

- Counting occurrences of the word 'we':

```
my $count = 0;
while (/we/ig) { $count++; }
print $count;
```

- Reformat sentences as paragraphs:

```
s|(\w)\n|$1 |g;
s|(\w\.)\s|$1\n\n|g;
```

- Replace multiple spaces with single spaces:

```
s| {2,}| |g;
```

Question 3

Modification of `matchtest2.plx`, producing output directly from the return value of `//`:

```
#!/usr/bin/perl
# ex05_03.plx
use warnings;
use strict;

my @vars;
$_ = '1: A silly sentence (495,a) *BUT* one which will be useful. (3)';

print "Enter some text to find: ";
my $pattern = <STDIN>;
chomp($pattern);

if (@vars = /$pattern/) {
   print "The text matches the pattern '$pattern'.\n";
   foreach (@vars) {
      print "Group: $_\n";
   }
} else {
   print "'$pattern' was not found.\n";
}
```

Question 4

A bubble sort using regular expressions.

```
#!/usr/bin/perl
# ex05_04.plx
use warnings;
use strict;

$_ = <<EOF;
      <put your data in here>
EOF

my ($count, $swaps, $done) = (0,0,0);

until ($done) {
      m|^(.+\n){$count}|g;      # match first '$count' lines in $_

      if ( m|\G(.+)\n(.+)| ) {      # try matching the next pair
         if ($2 lt $1) {      # if they're in the wrong order
            s|\G(.+)\n(.+)|$2\n$1|; # swap 'em round
            $swaps++;
            } else {      # otherwise,
            pos()=0;      # reset the \G boundary for $_
            }
         $count++;
      } else {
         $done = 1 if ($swaps == 0);   # done if there were no swaps
         ($count, $swaps) = (0,0);   # reset for next time around
      }
}
print;
```

Chapter 6

Question 1

Search for specified string in all files in current directory.

```
#!/usr/bin/perl
# Ex6_1.plx
use warnings;
use strict;

print "What string would you like to search for?";
chomp (my $query = <STDIN>);

opendir DH, "." or die "Couldn't open the current directory: $!";
while ($_ = readdir(DH)) {
        next if $_ eq "." or $_ eq "..";

        eval {open FH, $_ or die "Couldn't open file $_";};
        my $file = $_;
        while (<FH>) {
         if (/$query/) {
              print "Found \"$query\" in file ";
              print $file, "." x (30-length($file));
              print "d" if -d $file;
              print "r" if -r _;
              print "w" if -w _;
              print "x" if -x _;
              print "o" if -o _;
              print "." x 10;
              print -s _ if -r _ and -f _;
              print "\n";

         }
   }
}
```

Question 2

Add overwrite warning to backup facility in `filetest.plx`.

```
#!/usr/bin/perl
# Ex6_2.plx
use warnings;
use strict;

my $target;
while (1) {
   print "What file should I write on? ";
   $target = <STDIN>;
   chomp $target;
   if (-d $target) {
      print "No, $target is a directory.\n";
      next;
   }
   if (-e $target) {
      print "File already exists. What should I do?\n";
      print "(Enter 'r' to write to a different name, ";
      print "'o' to overwrite or\n";
```

```
        print "'b' to back up to $target.old)\n";
        my $choice = <STDIN>;
        chomp $choice;
        if ($choice eq "r") {
           next;
        } elsif ($choice eq "o") {
           unless (-o $target) {
              print "Can't overwrite $target, it's not yours.\n";
              next;
           }
           unless (-w $target) {
              print "Can't overwrite $target: $!\n";
              next;
           }
        } elsif ($choice eq "b") {
        if (-e $target.".old") {
              print "Backup already exists. Overwrite?\n";
           my $choice = <STDIN>;
           chomp $choice;
           next unless ($choice eq "y");
        } elsif ( rename($target,$target.".old") ) {
           print "OK, moved $target to $target.old\n";
        } else {
           print "Couldn't rename file: $!\n";
           next;
        }
        } else {
           print "I didn't understand that answer.\n";
           next;
        }
     }
     last if open OUTPUT, "> $target";
     print "I couldn't write on $target: $!\n";
     # and round we go again.
  }
  print OUTPUT "Congratulations.\n";
  print "Wrote to file $target\n";
```

Chapter 7

Question 1

Constructing a multiplication tables in words

```
#!/usr/bin/perl
#ex7_1.plx
use warnings;


@one = qw(one two three four five six);
@two = qw(two four six eight ten twelve);
@three = qw(three six nine twelve fifteen eighteen);
@four = qw(four eight twelve sixteen twenty twenty-four);
@five = qw(five ten fifteen twenty twenty-five thirty);
@six = qw(six twelve eighteen twenty-four thirty thirty-six);

@mult_table=(\@one, \@two, \@three, \@four, \@five, \@six);
```

```
print "Enter a number between 1 and 6: ";
$i = <STDIN>;
print "Enter a number between 1 and 6: ";
$j = <STDIN>;
chomp ($i, $j);

print $i--, " multiplied by ", $j--," equals ", $mult_table[$i]->[$j], "\n";
```

Question 2

From the spot we begin at in the program, we need to first determine if the knight is actually moving and what color it is to make sure we're not moving it onto its own pieces. Then we can check if it has made a good move or not. Note the use of `abs()` in the `if` condition. This returns the absolute value of a number and so we can check for movement both forwards and backwards (or left and right) in one go.

All the other chess pieces (except the king) require further tests to make sure there are no pieces in the way of their move down a rank or diagonally. Note that these checks would do very well if kept in separate subroutines. See Chapter 8 for more on this.

```
#!/usr/bin/perl
# Ex7_2.plx
use warnings;


my @chessboard;
my @back = qw(R N B Q K N B R);
for (0..7) {
   $chessboard[0]->[$_] = "W" . $back[$_]; # White Back Row
   $chessboard[1]->[$_] = "WP";            # White Pawns
   $chessboard[6]->[$_] = "BP";            # Black Pawns
   $chessboard[7]->[$_] = "B" . $back[$_]; # Black Back Row
}

while (1) {
   # Print board
   for my $i (reverse (0..7)) { # Row
       for my $j (0..7) {       # Column
          if (defined $chessboard[$i]->[$j]) {
            print $chessboard[$i]->[$j];
          } elsif ( ($i %2) == ($j %2) ) {
             print "..";
          } else {
             print "  ";
          }
         print " ";  # End of cell
      }
      print "\n";     # End of row
   }

   print "\nStarting square [x,y]: ";
   my $move = <>;
   last unless ($move =~ /^\s*([1-8]),([1-8])/);
   my $startx = $1-1; my $starty = $2-1;

   unless (defined $chessboard[$starty]->[$startx]) {
      print "There's nothing on that square!\n";
      next;
   }

   print "\nEnding square [x,y]: ";
```

```
        $move = <>;
        last unless ($move =~ /([1-8]),([1-8])/);
        my $endx = $1-1; my $endy = $2-1;

        if ($chessboard[$starty]->[$startx] =~ /([WB])N/) {
           my $color = $1;
           print "$color Knight's move\n";

           #If knight is not moving to empty spot
           if (defined $chessboard[$endy]->[$endx]) {
              # Check not taking one of own pieces
              if ($chessboard[$endy]->[$endx] =~ /$color\w/)
              {
                 print "Don't take one of your own pieces silly!\n";
                 next;
              }
           }

           #Finally, check knight made an L-shape move
           if (((abs($endy - $starty)==2) && (abs($endx - $startx)==1))
              || ((abs($endx - $startx)==2) && (abs($endy - $starty)==1)))
           {
              print "Good move\n\n";
           }
           else
           {
              print "Knights move in a L-shape\n\n";
              next;
           }
        }

        # Put starting square on ending square.
        $chessboard[$endy]->[$endx] = $chessboard[$starty]->[$startx];
        # Remove from old square
        undef $chessboard[$starty]->[$startx];
     }
```

Chapter 8

Question 1

This slight extension of the `seconds1.plx` also does a quick check that the value the user has entered is a valid one for the question. That is, it keeps on asking for a value until a user enters a whole number only.

```
#!/usr/bin/perl
# ex8_1.plx
use warnings;
use strict;

my $num_of_seconds = "";
while ($num_of_seconds !~ /^\d+$/) {
   $num_of_seconds = getsecs();
}
```

```
my ($hours, $minutes, $seconds) = secs2hms($num_of_seconds);
print "3723 seconds is $hours hours, $minutes minutes and $seconds

seconds";
print "\n";

sub getsecs {
  print "How many seconds would you like converted? ";
  return <STDIN>;
}


sub secs2hms {
   my ($h,$m);
   my $seconds = shift;
   $h = int($seconds/(60*60)); $seconds %= 60*60;
   $m = int($seconds/60);      $seconds %= 60;

   return ($h,$m,$seconds);
}
```

Question 2

When you run this example, you'll get this message:

```
This is subroutine call 297
Deep recursion on subroutine "main::sub1" at ex8_2.plx line 25.
This is subroutine call 298
Deep recursion on subroutine "main::sub1" at ex8_2.plx line 11.
This is subroutine call 299
Deep recursion on subroutine "main::sub1" at ex8_2.plx line 18.
This is subroutine call 300
>
```

If your programs are looping within each other enough to get this message, you should try and analyze why and reduce it.

Note that in lines 11 and 18, we've had to append parentheses to `sub2()` and `sub3()` as the subroutines themselves haven't been defined yet as so need to be forward declared. The call to `sub1` in line 25 by contrast does not.

```
#!/usr/bin/perl
# ex8_2.plx
use warnings;
use strict;

my $count = 0;

sub sub1 {
   ++$count;
   print "This is subroutine call $count\n";
   sub2() unless $count==300;
   return;
}
```

```
sub sub2 {
   ++$count;
   print "This is subroutine call $count\n";
   sub3() unless $count==300;
   return;
}

sub sub3 {
   ++$count;
   print "This is subroutine call $count\n";
   sub1 unless $count==300;
   return;
}

sub1;
```

Question 3

For a little variety, this exercise also passes the call limit around to each subroutine as well. This time, the forward references we had to make in exercise two are made automatically, as we need to include parameters in the call to `sub2` and `sub3`.

```
#!/usr/bin/perl
# ex8_3.plx
use warnings;
use strict;

sub sub1 {
   my ($count, $limit) = @_;
   print "This is subroutine call ", ++$count, "\n";
   sub2($count, $limit) unless $count==$limit;
   return;
}

sub sub2 {
   my ($count, $limit) = @_;
   print "This is subroutine call ", ++$count, "\n";
   sub3($count, $limit) unless $count==$limit;
   return;
}

sub sub3 {
   my ($count, $limit) = @_;
   print "This is subroutine call ", ++$count, "\n";
   sub1($count, $limit) unless $count==$limit;
   return;
}

sub1(0, 300);
```

Question 4

```
#!/usr/bin/perl
# ex8_4.plx
use warnings;
use strict;

my @array = (1,1);
```

```
sub gen_fib (\@)
{

      my $array_r = shift;
      my $len = $#{$array_r};
      push (@{$array_r} , (@{$array_r}[$len-1]+@{$array_r}[$len]));

}

for( my $i=1; $i<=10; $i++)
{
   gen_fib (@array);
}

print "@array";
```

Chapter 9

Question 1

The debugged and properly laid out version of `buggy.plx` looks like this. How many things did you spot?

```
#!/usr/bin/perl
#fixed.plx

my %hash;

$_ = "";

until (/^q/i) {

print "What would you like to do? ('o' for options): ";
chomp($_ = <STDIN>);

if ($_ eq "o") { options() }
elsif ($_ eq "r") { readit() }
elsif ($_ eq "l") { listit() }
elsif ($_ eq "w") { writeit() }
elsif ($_ eq "d") { deleteit() }
elsif ($_ eq "x") { clearit() }
elsif ($_ eq "q") { print "Bye!\n"; }
else { print "Sorry, not a recognized option.\n"; }
}

sub options {
   print<<EOF;
      Options available:
      o - view options
      r - read entry
      l - list all entries
      w - write entry
      d - delete entry
      x - delete all entries
EOF
}
```

```
sub readit {
   my $keyname = getkey();

   if (exists $hash{"$keyname"}) {
      print "Element '$keyname' has value $hash{$keyname}";
   } else {
      print "Sorry, this element does not exist.\n"
   }
}

sub listit {
   foreach (sort keys(%hash)) {
      print "$_ => $hash{$_}\n";
   }
}

sub writeit {
   my $keyname = getkey();
   my $keyval = getval();

   if (exists $hash{$keyname}) {
      print "Sorry, this element already exists.\n"
   } else {
      $hash{$keyname}=$keyval;
   }
}

sub deleteit {
   my $keyname = getkey();

   if (exists $hash{$keyname}) {
      print "This will delete the entry $keyname.\n";
      delete $hash{$keyname} if besure();
   }
}

sub clearit {
   print "This will delete the entire contents of the current database.\n";
   undef %hash;
}

#*** Input Subs ***#

sub getkey {
   print "Enter key name of element: ";
   chomp($_ = <STDIN>);
   $_;
}

sub getval {
   print "Enter value of element: ";
   chomp($_ = <STDIN>);
   $_;
}
```

Chapter 11

Question 1

Filling in Dogbert's credentials.

```
#!/usr/bin/perl
use warnings;
use strict;
use Employee3;

my $dogbert = Employee->new (
   surname    => "Dogbert",
   employer   => "PHB",
   #salary     => "£1000",
   #address    => "3724 Cubeville",
   #forename   => "dogbert",
   #phone_no   => "555-5678",
   #occupation => "Overworked"
);

print $dogbert->occupation,"\n";
print $dogbert->address, "\n";
print $dogbert->forename, "\n";
print $dogbert->phone_no, "\n\n";

$dogbert -> occupation("Underpaid");
print $dogbert->occupation,"\n";
$dogbert -> address("31 Toon Street");
print $dogbert->address,"\n";
$dogbert -> forename("Bull");
print $dogbert->forename,"\n";
$dogbert -> phone_no("222-2-8");
print $dogbert->phone_no,"\n";
```

Question 2

Assuming we have the new method in place, we would call it from our main program like so.

```
#!/usr/bin/perl
use warnings;
use strict;
use Employee3;

my $dogbert = Employee->new (
   surname    => "Dogbert",
   employer   => "PHB",
   salary     => "£1000",
   address    => "3724 Cubeville",
   forename   => "dogbert",
   phone_no   => "555-5678",
   position => "Overworked"
);

$dogbert -> card();
```

The new `card()` method must go in the `Person` package, and it looks like this:

```
package Person;
# Class for storing data about a person
use warnings;
use strict;
use Carp;

my @Everyone
# Constructor and initialisation

# Object accessor methods

# Class accessor methods

# Utility methods


sub  card {
   my $self    = shift;
   my $name    = $self->fullname;
   my $address = $self->address;
   my $forename= $self->forename;
   my $surname = $self->surname;
   my $position = $self->position;
   my $phone_no = $self->phone_no;



  print <<EOF;

              $forename $surname

              $position

              $address

              Telephone: $phone_no

              Have a nice Day!

EOF
return $self;
}
1;
```