

COMP 722 E-commerce Fall 2006 Programming Assignment 4

Due in the digital drop box by Monday, Nov. 5 at 11:00 PM

1. The problem here is to write a program that, given the name of a text file as a command line argument, counts the number of definite articles (“the”) and the number of indefinite articles (“a” or “an”) in the text. So that these articles may be easily located, it echoes the text with articles converted to all uppercase.

Our solution is to loop through the text a line at a time and perform a substitution (operator `s`) on every (modifier `g`) occurrence of an article regardless of its case (modifier `i`). The position of the replacement string is occupied by a call to (execution of—modifier `e`) a function `count`, which takes one argument, viz., the article just matched. Function `count` converts its argument to all lowercase and checks whether the result is “the”. If so, it increments the variable `$def_count`, declared a my variable at the top level. If not, since `count` is called only when its argument is an article, its argument then must be a definite article, so it increments variable `$indef_count` (also a my variable at the top level). Function `count` returns the all-uppercase version of its argument since this is what is substituted for the occurrence that is matched.

The following is a listing of the program, `prob1.pl`, just described but with some gaps, labeled with Greek letters. Each label is repeated after the listing with a description of the code needed in place of the gap where it occurs. You can download a copy of this file from the location where you accessed this assignment. You can also download data file `prob1.dat`, whose contents are:

```
The cat is a cunning animal stalking a mouse.
A dog likes a mouse or an auck but hates a cat.
An eagle will eat the mouse.
```

When you execute the command

```
C:\someFolder>perl prob1.pl prob1.dat
```

the output should be

```
-----
THE cat is A cunning animal stalking A mouse.
A dog likes A mouse or AN auck but hates A cat.
AN eagle will eat THE mouse.

There were 2 definite articles
and 7 indefinite articles in the text.
-----
```

```

use English;

my $def_count; # the number of definite articles
my $indef_count; # the number of indefinite articles

sub count {
    if (  $\alpha$  ) {
        $def_count++;
    }
    else {
        $indef_count++;
    }

     $\beta$ ;
}

while ( <> ) {
    s/  $\gamma$  /  $\delta$  /  $\epsilon$ ;
     $\eta$ ;
}

print "\n\nThere were $def_count definite articles\n",
      "and $indef_count indefinite articles in the text.\n";

```

- α : An expression that counts as true if the lowercase version of the only argument of count is equal to “the”
- β : Return the uppercase version of the only argument of count.
- γ : A regular expression that matches an occurrence of “the”, “a”, or “an” that is not part of a larger word (Hint: consider the anchor `\b`.) The part of the regular expression that matches the article must be enclosed in parentheses so that it can be referenced in δ .
- δ : Invoke the function `count`, passing it the string that has just matched the article in the pattern.
- ϵ : Three modifiers, indicating that all possible matches are to be made, that matching is case insensitive, and that the position of the replacement string is occupied by a Perl expression that is to be executed.
- η : Output the line to which the substitution expression has just been applied.

2. The problem here is to write a Perl module that defines functions for binary operations on vectors, i.e., arrays of numbers. Each of these operations requires that the vector operands have the same length. We shall implement only vector sum and inner product. The sum of two vectors is the vector whose elements are the sums of the corresponding elements in the operand vectors. For example, $(1,2) + (3,4) = (4,6)$. The inner product of two vectors is the scalar that is the sum of the products of the corresponding elements in the operand vectors. For example, $(1,2) * (3,4) = 3 + 8 = 11$. We implement this module in the file `Vector.pm`. The functions that implement the operations, `v_sum` and `inner_product`, take references to arrays as arguments (so we don't end up copying two entire arrays, as would happen if the arrays themselves were passed); `v_sum` returns the resulting sum, and `inner_product` returns the number.

The following is a file that tests these functions.

```
use Vector;

my ( $ar_ref1, $ar_ref2, $ar_ref3 ) = ( [1,2], [3,4], [5,6,7] );
my @ar;
my $ip;

print "The vector sum of\n\t@$ar_ref1\n",
      "and\n\t@$ar_ref2\nis\n\t",
      ( @ar = v_sum( $ar_ref1, $ar_ref2 ) ) ?
      "@ar" : "undefined",
      "\n\n";

print "The inner product of\n\t@$ar_ref1\n",
      "and\n\t@$ar_ref2\nis\n\t",
      ( $ip = inner_product( $ar_ref1, $ar_ref2 ) ) ?
      "$ip" : "undefined",
      "\n\n";

print "The vector sum of\n\t@$ar_ref1\n",
      "and\n\t@$ar_ref3\nis\n\t",
      ( @ar = v_sum( $ar_ref1, $ar_ref3 ) ) ?
      "@ar" : "undefined",
      "\n\n";
```

The output of this program is shown on the following page.

```
The vector sum of
```

```
  1 2
```

```
and
```

```
  3 4
```

```
is
```

```
  4 6
```

```
The inner product of
```

```
  1 2
```

```
and
```

```
  3 4
```

```
is
```

```
 11
```

```
Arguments to v_sum have different lengths.
```

```
The vector sum of
```

```
  1 2
```

```
and
```

```
  5 6 7
```

```
is
```

```
undefined
```

In the file `Vector.pm`, we declare a package called `Vector` and arrange that it inherit from `Exporter`. We include the names of the two functions in the list of names exported. The two functions are quite similar. We first copy the arguments (references to arrays) into two `my` variables. If the arrays referenced by these variables have different lengths, we output a message to this effect on standard error output and return the empty list, `()`, for `vector_sum` and `undef` for `inner_product`. (The file handle for standard error output is `STDERR`. Standard error output, like standard output, is normally terminal output. If, for example, output is redirected to a file, however, standard error output will show up on the screen and will not be added to the file containing the standard output.) If the arrays have the same length, then we range a variable over the indices of these arrays. At each index, we add or multiply the elements of the two referenced arrays. For `vector_sum`, the sum is stored at the same index in the result array, and this array is returned. For `inner_product`, the product is added to the result scalar, and this scalar is returned.

The following is a listing of the module, `Vector.pm`, just described but with some gaps, labeled with Greek letters. Each label is repeated after the listing with a description of the code needed in place of the gap where it occurs. You can download a copy of this file from the location where you accessed this assignment. You can also download the test file shown above, called `prob2.pl`.

```
package α;
use English;

use β;
our @ISA = ( γ );

our @EXPORT = δ;

sub v_sum {

    my ( $ar_ref1, $ar_ref2 ) = ε;
    my @ar;

    if ( η ) {
        print STDERR "Arguments to v_sum have different lengths.\n";
        return ();
    }

    foreach ( 0 .. @$ar_ref1-1 ) {
        $ar[ $_ ] = ζ;
    }

    return @ar;
}

sub inner_product {
    my ( $ar_ref1, $ar_ref2 ) = ε;
    my $inner_p;

    if ( η ) {
        print STDERR "Arguments to inner_product have different lengths.\n";
        return undef;
    }

    foreach ( 0 .. @$ar_ref1-1 ) {
        $inner_p θ;
    }

    return $inner_p;
}

return 1;
```

- α : The name of this package is `Vector`.
- β : One part needed to make this package inherit from `Exporter`.
- γ : The other part needed to make this package inherit from `Exporter`.
- δ : Arrange that this module export the names of the two functions, `v_sum` and `inner_product`.
- ε (occurs twice): The array of arguments (Note that we have used the pragma `use English`.)
- η (occurs twice): An expression that is true if the arrays referenced by the two `my` variables have different lengths.
- ζ : The sum of the elements of the arrays referenced by the two `my` variables whose index is the default loop control variable
- θ : Increment `$inner_p` by the product of the elements of the arrays referenced by the two `my` variables whose index is the default loop control variable

3. Here we define a class `Point` with two “instance variables”, `x_coord` and `y_coord`. It has the following instance methods:

`x` (both a set and a get method): Return or update and return the value of the `x_coord` instance variable.

`y` (both a set and a get method): Return or update and return the value of the `y_coord` instance variable.

`set_cords`: This takes two arguments, the new value of `x_coord` and the new value of `y_coord`, and updates the point to have these coordinates. No validation is done.

`to_string`: Return the string `'(<x>, <y>)'`, where `<x>` is the value of `x_coord` and `<y>` is the value of `y_coord`.

`scalar_mult`: This takes one argument and multiplies both `x_coord` and `y_coord` by that argument to give their new values.

The following program, `prob3.pl`, is used to test class `Point`.

```
use Point;

my $pt = new Point;

$pt->set_coords( 3, 4 );
print "The point is ", $pt->to_string(), ".\n";
$pt->x( 1 );
$pt->y( 2 );
print "The new coordinates are ", $pt->x(), " and ",
      $pt->y(), ".\n";
$pt->scalar_mult( 2 );
print "The new point is ", $pt->to_string(), ".\n";
```

When executed, it produces the following output.

```
The point is (3, 4).
The new coordinates are 1 and 2.
The new point is (2, 4).
```

The following is a listing of the class file `Point.pm` but with some gaps, labeled with Greek letters. Each label is repeated after the listing with a description of the code needed in place of the gap where it occurs. You can download a copy of this file from the location where you accessed this assignment. You can also download the test file shown above, called `prob3.pl`.

```
package Point;

use English;

sub new {
    my $point = α_____
                _____;
    β_____;
    return $point;
}

sub x {
    my $self = shift;
    γ_____;
    return δ_____;
}

sub y {
    my $self = shift;
    ε_____;
    return η_____;
}

sub set_coords {
    my $self = shift;

    ζ_____;
    θ_____;
}

sub to_string {
    my $self = shift;

    return ι_____;
}

sub scalar_mult {
    my $self = shift;
    my $factor = λ_____;

    μ_____;
    ν_____;
}

return 1;
```

- α : A reference to a hash with keys `x_coord` and `y_coord`, both associated with value `undef`
- β : Convert `$point` into a reference to an instance of `Point`.
- γ : As long as this method was invoked with an argument, assign this argument to the `x_coord` “instance variable.”
- δ : The value of the `x_coord` “instance variable”
- ϵ : As long as this method was invoked with an argument, assign this argument to the `y_coord` “instance variable.”
- η : The value of the `y_coord` “instance variable”
- ζ : Use the `x` method to update the `x_coord` “instance variable” to the first argument.
- θ : Use the `y` method to update the `y_coord` “instance variable” to the second argument.
- ι : The string `'(<x>, <y>)'`, where `<x>` is the value of `x_coord` and `<y>` is the value of `y_coord`
- λ : The value of the argument passed to this method
- μ : Update the “instance variable” `x_coord` to `$factor` times its current value
- ν : Update the “instance variable” `y_coord` to `$factor` times its current value