

COMP 722 E-commerce Fall 2006 Programming Assignment 3

Due in the digital drop box on Wed., Oct. 25 by 11:00

1. (This problem covers material on the DOM.) Create an HTML document that initially displays just a paragraph element whose content is “Start.” When this element is clicked, its text disappears (i.e., is changed to the empty string), a short (two of three sentences) description about you appears at the top left of the screen, and a short (two of three sentences) description about A&T appears at the top somewhere to the right. The description of you moves slowly right and the description of A&T moves left at the same speed. They move across each other, and both disappear (i.e., their text becomes the empty string) when they reach the positions originally occupied by each other.

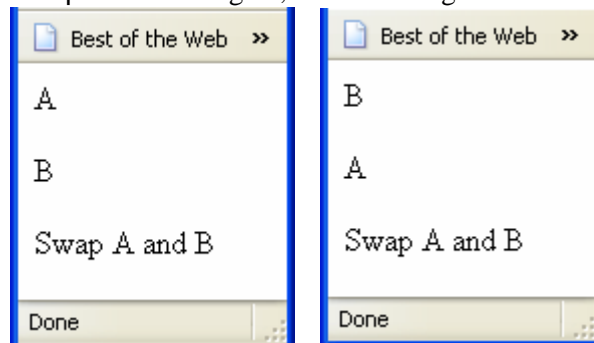
Hints: Use paragraph elements whose widths are certain percentages of the screen (see Part 3). The paragraphs’ contents should initially be empty. Use `setInterval` and `clearInterval` with a global variable. See the example program that uses `setInterval`. Note that you can use `clearInterval` in the function that is repeatedly executed by virtue of the `setInterval` call. Give values for the `id` attributes of the paragraphs and use the `getElementById` method to obtain references to the nodes corresponding to these elements. These references should be in global variables. You will have to manipulate the `innerHTML` attribute values of all three nodes. Since `setAttribute` does not work with “`innerHTML`”, you have to use constructs of the form

```
obj.innerHTML = "A string";
```

For this problem, you have to define one or two global variables whose values are changed and are assigned to the appropriate properties of objects representing paragraph elements. Use the `setProperty` method of the `style` object. You must also use one of these variables in the test that checks whether one of the paragraphs has reached its final position (and so `clearInterval` should be called).

2. (This problem also covers material on the DOM. It is a small problem intended to give you some experience with manipulating nodes in the document tree and with event listeners.) Create an HTML document whose body contains three paragraphs, with content “A”, “B”, and “Swap A and B”, respectively. Be sure to have `id` attributes for these paragraphs. After the document is loaded, it should execute a function that adds the function `swapAB` to the node for the third paragraph as a listener for `click` events. Function `swapAB` swaps the node for the paragraph with content “A” with the node for the paragraph with content “B”. Thus, when the document is loaded, the rendering is as on the left below. After the user clicks **Swap A and B**, the rendering is as on the right below. If the user clicks **Swap A and B** again, the rendering returns to the original form. Further clicking toggles the rendering between these two forms.

Hints: The paragraphs are all children of the body; `document.body` gives a reference to the node for this. Get references to the nodes for the first two paragraphs and use the `replaceChild` method of the body node. Note that the first argument of this method is the new child node and the second argument is the node it replaces. You’ll want to create an additional `Element` node as a placeholder among the



children of the body. The straightforward approach is to invoke `replaceChild` three times in a pattern reminiscent of the standard three-assignment-statement pattern for swapping the contents of two variables.

3. The following Perl code (with fragments missing) is in a file `prob3.pl`. When executed with a command-line argument specifying a file that has one integer per line, it first inputs each integer and stores it in the next cell in array `@ar`. It then repeatedly prompts the user for an integer. This input is assigned to the variable `$factor` and its input record separator is removed. Each element of `@ar` is multiplied by `$factor` and the result is stored in the corresponding cell of array `@ar1`. The values of `@ar`, `$factor`, and `@ar1` are then output, and the user is prompted again.

Supply the missing code; each gap is labeled with a Greek letter, and the description of that gap, labeled with the same letter, is below the listing. You can download this file (with the fragments missing) from the page where you accessed this assignment. A data file (to specify on the command line), `prob1.dat`, can also be downloaded.

```
# Read in the array of numbers

while ( α ) {
    chomp( $ar[ $i++ ] = $_ );
}

print "Enter an integer (Control-Z to end): ";

while ( β ) ) {

    # Multiply each element of @ar by $factor,
    # storing the result in @ar1.

    for ( γ ) {
        δ;
    }

    print "ε";
    print "Enter an integer (Control-Z to end): ";
}

```

α : Input a value from the file specified on the command line.

β : Input a value from standard input, assigning it to `$factor` and removing the input record separator.

γ : Range variable `$i` over all the indices of array `@ar`.

δ : Set the element of array `@ar1` at index `$i` to `$factor` times the corresponding element of `@ar`.

ϵ : Output the contents of `@ar`, `$factor`, and `@ar1`. Don't use a loop.

The following are the contents of the data file, prob3.dat:

```
2
4
6
```

The following is an example run:

```
C:\someDirectory>perl prob3.pl prob3.dat
Enter an integer (Control-Z to end): 2
Array 2 4 6 multiplied by 2 is 4 8 12
Enter an integer (Control-Z to end): -1
Array 2 4 6 multiplied by -1 is -2 -4 -6
Enter an integer (Control-Z to end): ^Z
```

4. The following Perl code (with fragments missing) is in a file prob4.pl. When executed with a command-line argument specifying a file that has one name per line, it first assigns each name to a successive element of array @ar. It then iterates over the names in @ar, prompting the user for the points for each. The points are recorded in hash %points, whose keys are the names. It then echoes back all the key-value pairs in %points. Finally, it repeatedly prompts the user for names, outputting the associated points in reply, until the user types Control-Z.

Supply the missing code. You can download this file (with the fragments missing) from the page where you accessed this assignment. A data file (to specify on the command line), prob4.dat, is also available there.

```
# Read the names from the file
while ( <> ) {
    chomp( $ar[ $i++ ] = $_ );
}

# Get the points from the user; create the hash
_α_____ {
    print "Enter the points for $_: ";
    chomp( _β_____ = <STDIN> )
        or die "No value entered for $_";
}

# Echo the points
print "The points are:\n";

while ( ($name, $pts) = _γ_____ ) {
    print "$name has $pts points\n";
}

# Let the user find the points for specified individuals
print "Enter a name (Control-Z to stop): ";

while ( chomp( $name = <STDIN> ) ) {
    print "$name has _δ_____ points\n";
    print "Enter a name (Control-Z to stop): ";
}
}
```

- α : Iterate over all the elements in `@ar`. Don't give a loop control variable. Rather, the implicit variable `$_` will successively be an alias for each element of `@ar`.
- β : Assign the input to the element of hash `%points` with key `$_`.
- γ : Assign the next key-value pair in hash `%points`.
- δ : Output the value in `%points` with key `$name`.

The following are the contents of the data file, `prob4.dat`:

```
Bob
Fred
Bill
```

The following is a run:

```
C:\someDirectory>perl prob4.pl prob4.dat
Enter the points for Bob: 2
Enter the points for Fred: 3
Enter the points for Bill: 4
The points are:
Fred has 3 points
Bill has 4 points
Bob has 2 points
Enter a name (Control-Z to stop): Bill
Bill has 4 points
Enter a name (Control-Z to stop): Fred
Fred has 3 points
Enter a name (Control-Z to stop): ^Z
```

5. The following Perl code (with fragments missing) is in a file `prob5.pl`. It is intended to be executed with a command-line argument that is the name of a file that contains on its first line the number of rows (= the number of columns) in a square matrix. The following lines contain the rows of the matrix. After reading in the lines and constructing a two-dimensional array representing the square matrix, it outputs the transpose of the matrix. Recall that the transpose of a matrix M is the matrix whose rows are the columns of M (and whose columns are the rows of M). For example, if M is

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

then its transpose is

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Supply the missing code. You can download this file (with the fragment missing) from the location where you got this assignment. A data file (to specify on the command line), `prob5.dat`, can also be downloaded. Look at the example in sec. 14.3 of the notes.

```

# Input the number of rows
# (= the number of columns)

$dim = <>;

# Get the matrix
foreach $row ( 0 .. $dim-1 ) {
    α
    @arow = split / +/, $inrow;
    $mat[ $row ] = β;
}

# Output the transpose of the matrix
foreach $col ( 0 .. $dim-1 ) {
    foreach $row ( 0 .. $dim-1 ) {
        γ;
    }
    print "\n";
}

```

α: Input the next line, assign it to \$inrow, and remove the input record separator.

β: Complete the assignment so that a reference to the list of elements in @arow is assigned to \$mat[\$row].

γ: Output the appropriate element of the matrix.

The following are the contents of the data file.

```

2
1 2
3 4

```

The following is an example run.

```

C:someDirectory>perl prob5.pl prob5.dat
1 3
2 4

```

6. The following Perl code (with fragments missing) is in a file `prob6.pl`. It does not expect a command-line argument. It defines a function `inc_array`, which is supposed to be passed a number and a reference to an array of numbers. (Essentially, the array is passed by reference.) It updates the array by adding the number parameter to each element of it.

Supply the missing code. You can download this file (with the fragment missing) from the location where you got this assignment.

```
use English;

sub inc_array {
    my $increment = α;
    my $ref_array = β;

    foreach γ {
        $element += $increment;
    }
}

@ar = ( 1, 3, 5 );
δ;

print "The new contents of the array: @ar \n";
```

α : The first parameter (a scalar)

β : The second parameter (a reference to an array)

γ : Range the variable `$element` over the elements in the array that is referenced by the second parameter.

δ : Call `inc_array` with 3 as the first parameter and a reference to array `@ar` as the second parameter.