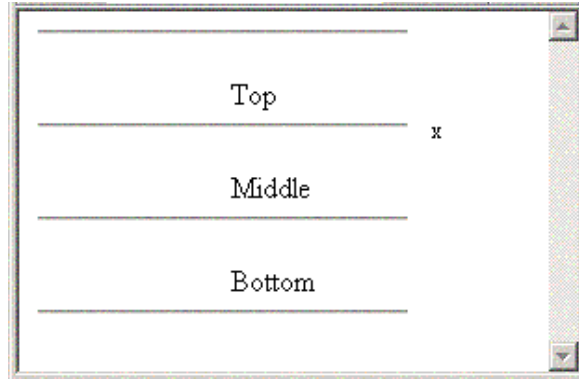


## COMP 322 Internet Systems Fall 2006 Exam 2—Solutions

50 points total

1 (20 points). The HTML document (with gaps) listed on the next two pages is rendered as shown below. The small `x` on the right travels from 10 pixels from the top to 160 pixels from the top and back, repeatedly, moving 5 pixels (down or up) every 10 milliseconds. The horizontal lines evenly divide the vertical path of the `x` into three segments; they lie at 10, 60, 110, and 160 pixels from the top. The paragraph elements (with content `Top`, etc.) are 35, 85, and 135 pixels from the top (i.e., the top of each element is this many pixels from the top). If the user clicks the paragraph with content `Top` when the `x` is between the top two lines, an alert box with content **Hit** pops up; if he clicks this paragraph when the `x` is outside this segment, an alert box with **Miss** pops up. The paragraph elements with content `Middle` and `Bottom` play analogous roles for their respective segments of the path of the `x`.



The `x` is a paragraph element positioned 20% of the way from the right margin. The horizontal rules extend 75% of the way across the windows, and the other paragraph elements begin 40% of the way from the left margin. The value of the `id` attribute of the `x` is "mover"; the values of the `id` attributes of the other three paragraph elements are "top", "middle", and "bottom".

As soon as the document is loaded, a function `start` is called. This makes function `attempt` the listener for `click` events in the `body` element and causes function `run` to be called every 10 msec. There are two global variables: `y` (the distance of the `x` from the top) and `incr` (the amount by which `y` is changed each 10 msec.). Function `run` first checks whether the `x` is at a turning point by checking whether `y modulo 150` is 10; if so, the sign of `incr` is changed. In any case, `incr` is added to `y`, variable `mv` is set to a reference to the object corresponding to the `x`, and the distance of the `x` from the top is set to `y`.

Function `attempt` first sets variable `mv` to a reference to the object corresponding to the `x` and assigns the distance of the `x` from the top to the variable `moverY` (after converting this distance to an integer, getting rid of the `pxl` unit label). If the `id` of the element where the `click` event happens is "top", it raises an alert box with **Hit** if `moverY` is between 10 and 60 and otherwise raises an alert box with **Miss**. If the `id` of this element is "middle", **Hit** is signaled if `moverY` is between 60 and 110. And, if this `id` is "bottom", **Hit** is signaled if `moverY` is between 110 and 160.

The following listing has gaps labeled with Greek letters. After this listing, the Greek letters are repeated with descriptions of what should go in the corresponding gaps. Supply the missing code after each description.

```
<html>
<head>
<title>Problem 1</title>
<script>
  var incr = -5,
      y = 10;

  function start()
  {
    α _____;
    β _____;
  }

  function run()
  {
    if ( y % 150 == 10 )
      incr *= -1;

    y += incr;
    var mv = γ _____;
    δ _____;
  }

  function attempt( event )
  {
    var mv = γ _____,
        moverY =
          parseInt( ε _____ );

    if ( ζ _____ == "top" )
      if ( 10 <= moverY && moverY <= 60 )
        window.alert( "Hit" );
      else
        window.alert( "Miss" );
    else if ( ζ _____ == "middle" )
      if ( 60 <= moverY && moverY <= 110 )
        window.alert( "Hit" );
      else
        window.alert( "Miss" );
    else if ( ζ _____ == "bottom" )
      if ( 110 <= moverY && moverY <= 160 )
        window.alert( "Hit" );
      else
        window.alert( "Miss" );
  }
}
```

Continued

Continued from previous page

```
</script>
</head>

<body η>
<hr style = "position : absolute; top : 10; width : 75%">

<p id = "top"
  style = "position : absolute; top : 35; left : 40%">
  Top
</p>

<hr style = "position : absolute; top : 60; width : 75%">

<p id = "middle"
  style = "position : absolute; top : 85; left : 40%">
  Middle
</p>

<hr style = "position : absolute; top : 110; width : 75%">
<p id = "bottom"
  style = "position : absolute; top : 135; left : 40%">
  Bottom
</p>

<hr style = "position : absolute; top : 160; width : 75%">

<p id = "mover"
  style =
    "position : absolute; right : 20%; top : 0; font-size : 6pt">
  X
</p>
</body>

</html>
```

$\alpha$  (3.5 pts.): Make function `attempt` the listener for `click` events in the `body` element. You may want to do this with two statements (introducing a variable).

**Answer**

```
document.body.addEventListener( "click", attempt, false );
```

Or, with two statements,

```
var bd = document.body;
bd.addEventListener( "click", attempt, false );
```

$\beta$  (3 pts.): Cause function `run` to be called every 10 msec.

**Answer**

```
window.setInterval( "run()", 10 );
```

$\gamma$  (occurs twice—3 pts.): an expression that evaluates to a reference to the object corresponding to the element with id “`mover`” (the `p` element with content `x`).

**Answer**

```
document.getElementById( "mover" );
```

$\delta$  (3 pts.): Set the CSS property `top` of the element referenced by `mv` to the value of variable `y`.

**Answer**

```
mv.style.setProperty( "top", y, "" );
```

Also acceptable (but not as good):

```
mv.style.top = y;
```

$\epsilon$  (3 pts.): an expression that evaluates to the value of the CSS property `top` of the element referenced by `mv`

**Answer**

```
mv.style.getPropertyValue( "top" )
```

Also acceptable (but not as good):

```
mv.style.top = y;
```

$\zeta$  (occurs 3 times—3 pts.): the value of the `id` attribute of the element where the currently handled event happened

**Answer**

```
event.target.id
```

$\eta$  (1.5 pts.): Arrange for function `start` to be called when the body of the document finishes loading.

**Answer**

```
onload = "start()"
```

2 (12 points). The following is a Perl script (with gaps) that repeatedly prompts the user for a name until the name Bob is entered. It keeps a count of the number of times the name Ed is entered. At the end, if Ed was entered more than once, the script outputs

More Eds than Bobs.

This is terminated by a newline. In any case, it outputs

The number of Eds is *ed\_count*.

where *ed\_count* is the number of times the name Ed was entered (maintained as a variable). This again is terminated by a newline.

The following listing has gaps labeled with Greek letters. After this listing, the Greek letters are repeated with descriptions of what should go in the corresponding gaps. Supply the missing code after each description.

```

$ed_count = 0;
do {
    print "Enter a name: ";
    chomp( $name = <STDIN> );
    α_____ ;
} β_____ ;

γ_____ ;
print "δ_____ " ;

```

α (3.5 pts.): Increment \$ed\_count if \$name contains “Ed”. (Use a statement modifier.)

**Answer**

```
$ed_count ++ if $name eq "Ed" ;
```

β (3 pts.): Complete this loop so that it iterates until \$name contains “Bob”.

**Answer**

```
( $name eq "Bob" )
```

γ (3.5 pts.): Output

More Eds than Bobs.

(terminated by a newline) as described above if \$ed\_count is greater than 1.

**Answer**

```
print "More Eds than Bobs\n" if $ed_count > 1 ;
```

δ (2 pts.): Fill in what’s between the double quotes to output

The number of Eds is *ed\_count*.

as described above

**Answer**

```
"The number of Eds is $ed_count.\n"
```

**3** (18 points). The following Perl program (with gaps) inputs a file whose name is given on the command line. Each line of the input file gives a person's name and a monetary amount. The unit for the amount is a three-letter abbreviation that follows immediately after the amount (with no intervening whitespace). The following is an example input file.

```
Bill 12.34USD
Fred 6.00GBP
Pierre 7.50EUR
Mike 5.15USD
```

(“USD” stands for “US dollars”, “GBP” stands for “UK pounds”, and “EUR” stands for euros.) The program splits each line (at the whitespace) into the name (variable `$name`) and the amount with the units (variable `$amount`). It then passes `$amount` to function `convert`, which returns the corresponding dollar amount (as a decimal number without the units). The converted value is assigned to the element of hash `%prices` with key `$name`.

After the loop that inputs the file, the program passes a reference to `%prices` to the function `sum_prices`, which returns the sum of the values in `%prices`, and this returned value is assigned to `$total`. The program also computes the average price. The sum and average are converted to formatted strings and output with identifying text. For example, the output given the above file contents is

```
Total prices: $38.76
Average price: $9.69
```

Function `convert` uses a regular expression to match the decimal value and the units. The decimal value consists of one or more digits, a decimal point (`.`), then exactly two digits; the unit designation consists of exactly three word characters. It remembers the decimal value and the unit designation. The function converts the decimal value to a dollar value. If the units are dollars, the decimal value is returned unchanged; if the units are pounds, the value is multiplied by 1.92; if the units are euros, it is multiplied by 1.30; if any other unit is given, 0 is returned.

Function `sum_prices` shifts the reference to `%prices` into `$ref_prices`, loops over the values in the hash referenced by `$ref_prices`, adding each to `$total`, whose final value it returns.

The following listing has gaps labeled with Greek letters. After this listing, the Greek letters are repeated with descriptions of what should go in the corresponding gaps. Supply the missing code after each description.

```

use English;
use strict 'vars';

my ( $name, $amount, %prices, $total, $print_total,
      $print_ave );

while ( <> ) {
    α _____;
    $prices{ $name } = convert( $amount );
}

$total = β _____;
$print_total = sprintf "\$%.2f", $total;
$print_ave = sprintf "\$%.2f", γ _____;
print "Total prices: $print_total\n",
      "Average price: $print_ave\n";

sub convert
{
    $ARG[0] =~ δ _____;

    ε _____
    _____
    _____
    _____
    _____
    _____
}

sub sum_prices
{
    my $ref_prices = shift;
    my $total;

    foreach ( ζ _____ ) {
        $total += $_;
    }

    return $total;
}

```

$\alpha$  (3 pts.): Split the input line at the whitespaces, giving a list of two elements; assign the first to `$name` and the second to `$amount`.

**Answer**

```
( $name, $amount ) = split;
```

$\beta$  (2.5 pts.): Call function `sum_prices`, passing it a reference to `%prices`.

**Answer**

```
sum_prices( \%prices )
```

$\gamma$  (2.5 pts.): Compute the average of the prices.

**Answer**

```
$total / keys( %prices )
```

(The parentheses aren't needed.) This could also be

```
$total / values( %prices )
```

$\delta$  (3.5 pts.): A regular expression that matches a decimal value followed by a unit designator. The decimal value consists of one or more digits, a decimal point (`.`), then exactly two digits; the unit designation consists of exactly three word characters. Remember the decimal value and the unit designator separately.

**Answer**

```
/(\d+\.\d{2})(\w{3})/
```

$\epsilon$  (multiple lines in the code—3.5 pts.): Return the decimal value (remembered from the match) converted according to the units (also remembered from the match). If the units are “USD”, return the value unchanged; if they are “GBP”, multiply it by 1.92; if they are “EUR”, multiply it by 1.30; if they are anything else, return 0.

**Answer**

```
if ( $2 eq "USD" ) {
    return $1;
} elsif ( $2 eq "GBP" ) {
    return 1.92 * $1;
} elsif ( $2 eq "EUR" ) {
    return 1.30 * $1;
} else {
    return 0;
}
```

$\zeta$  (3 pts.): The list of values in the hash referenced by `$ref_hash`

**Answer**

```
values %$ref_prices
```