

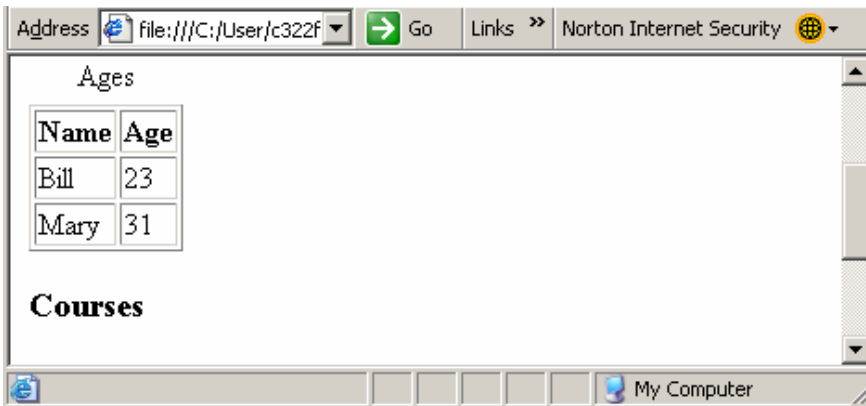
# COMP 322 Internet Systems Fall 2007 Exam 1—Solutions

50 points total

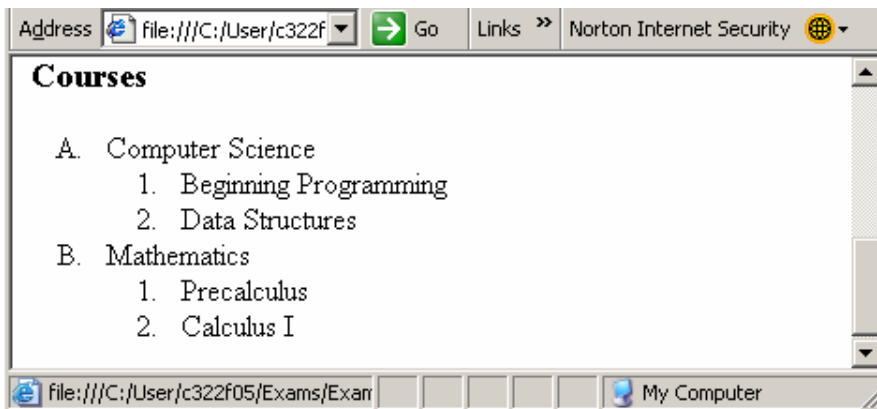
1 (18 pts.). The rendering below is of the top part of `probl.html`, which is listed with gaps after this description. The image shown has two hotspots, one occupying the left half, the other occupying the right half of the image.



When the hotspot on the left is clicked, the browser follows an internal link to the location associated with the name `ages`, and the rendering becomes



When the hotspot on the right is clicked, the browser follows an internal link to the location associated with the name `courses`, and the rendering becomes





$\alpha$  (3 lines, 4 pts.): A hotspot with an internal link to the location named `ages`. This is a rectangle whose upper left corner is at (0,0) and whose lower right corner is at (176,106) in pixels in the image coordinates. When the mouse cursor moves over this area in the image, the text “go to ages” appears in a small box.

**Answer**

```
<area href = "#ages" shape = "rect"
      coords = "0,0,176,106"
      alt = "go to ages">
```

$\beta$ : (3 lines, 4 pts.): A hotspot with an internal link to the location named `courses`. This is a rectangle whose upper left corner is at (177,0) and whose lower right corner is at (353,106) in pixels in the image coordinates. When the mouse cursor moves over this area in the image, the text “go to courses” appears in a small box.

**Answer**

```
<area href = "#courses" shape = "rect"
      coords = "177,0,353,106"
      alt = "go to courses">
```

$\delta$  (10 lines, 7 pts.): Write the entire table as shown in the screenshot.

**Answer**

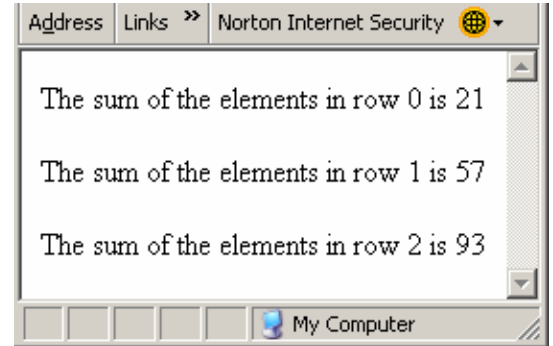
```
<table border="1px">
  <caption>Ages</caption>
  <thead>
    <tr><th>Name</th><th>Age</th></tr>
  </thead>
  <tbody>
    <tr><td>Bill</td><td>23</td></tr>
    <tr><td>Mary</td><td>31</td></tr>
  </tbody>
</table>
```

$\varepsilon$  (3 pts.): Associate the name `courses` (a *fragment identifier*) with this location (point—no content is included).

**Answer**

```
<a name="courses">
```

2 (14 pts.). The JavaScript file listed below with gaps begins execution with the function `start()`, which adds two  $3 \times 3$  arrays in matrix fashion then finds and outputs the three row sums in the result. The rendering of the output is shown at right. Function `addArrays()` takes three array arguments; the third is the result of adding the first two. The arrays are not checked to determine whether they are compatible for addition. Function `vectorSum()` takes an (one-dimensional) array and returns the sum of its elements. Note that the operand arrays are hard-coded into the document.



The following is the listing with gaps. The gaps in the listing are labeled with Greek letters. The letters are repeated after the listing with descriptions of the code that should go in the corresponding gaps. You there supply the missing code.

```
function start()
{
    var ar0 = [ [1, 3, 5],
                [7, 9, 11],
                [13,15,17] ],
        ar1 = [ [2, 4, 6],
                [8, 10,12],
                [14,16,18] ],
        ar2 = new Array(3);

    α
    _____
    _____

    addArrays( ar0, ar1, ar2 );

    for ( var i in ar2 )
        document.writeln( "<p>The sum of the elements in row "
                           + i + " is " + vectorSum( ar2[i] ) + "</p>" );
}

function addArrays( a0, a1, a2 )
{
    β
    _____
    _____
}

function vectorSum( vec )
{
    γ
    _____
    _____
    _____
}

```

$\alpha$  (3 lines, 3 pts.): Allocate three cells for each of the three rows in `ar2`.

**Answer**

```
ar2[0] = new Array(3);
ar2[1] = new Array(3);
ar2[2] = new Array(3);
```

$\beta$  (3 lines, 5 pts.): Write the body of `addArrays()`, which computes two-dimensional array `ar2` as the matrix sum of two-dimensional arrays `ar0` and `ar1`. Use `for/in` loops and assume that the arrays are compatible for addition.

**Answer**

```
for ( var i in a0 )
  for ( var j in a0[0] )
    a2[i][j] = a0[i][j] + a1[i][j];
```

$\gamma$  (4 lines, 6 pts.): Write the body of `vectorSum()`, which returns the sum of the elements in the one-dimensional array `vec`. Use a `for/in` loop.

**Answer**

```
var sum = 0;

for ( var i in vec )
  sum += vec[i];

return sum;
```

**3** (18 pts.). The following listing shows a constructor `Person` with instance variables `name` (a string), `gender` (a string, expected to be “male” or “female”), `mother` (an instance of `Person` or the string “unknown”), `father` (an instance of `Person` or the string “unknown”), and `children` (a possibly empty array of instances of `Person`). There are five parameters to the constructor, with the same names as the instance variables. The defaults for the instance variables are all “unknown”. The `toString()` method returns the value of the `name` instance variable. There are the expected `get` and `set` methods except that, instead of a `setChildren()` method, there is an instance method `addChild()`.

Instance method `addChild()` is defined outside the constructor as the function `Person_adChild()` and assigned to the `addChild` property of the `Person` prototype. This function takes one argument, `child`, expected to be an instance of `Person`. This function obviously adds `child` to the `children` array instance variable of the current instance. The current instance, however, must be the father or mother of the child just added. So, if the `gender` of the current instance is “male”, it (the current instance) must be assigned as the father of `child`, and, if the `gender` of the current instance is “female”, then it must be assigned as the mother of `child`. (If the `gender` of the current instance is “unknown”, then we cannot make the current instance either the father or the mother of `child`.)

We also define a zero-parameter instance method `brothers()`, again defined outside the constructor. This returns an array without duplicates of instances of `Person` that are brothers of the current instance. This counts among a person’s brothers his or her half brothers. The idea is first to build the array `broMom` from the children of the mother of the current instance that are male and distinct from the current instance itself. Next, we build the array `broDad` from the children of the father of the current instance that again are male and distinct from the current instance but also now are not in the array `broMom`. (When checking whether to include a `Person` instance in `broDad`, we use the function `member()`, not shown in this listing.) Finally, we return the result of concatenating one array onto the other; to be definite, we concatenate `broDad` onto `broMom`.

The major problem here is that either the father or the mother of the current instance may be unknown. We then consider that no brothers come from the unknown parent. So the loops constructing `broMom` and `broDad` work with the arrays `momsChildren` and `dadsChildren`, respectively, where `momsChildren` is the empty array if the mother is unknown (and otherwise is the array of the mother’s children), and `dadsChildren` is constructed similarly (but regarding the father, not the mother).

After the listing, you are asked (1) to supply the body of the `addChild()` instance method and (2) to write the loop that constructs the array `broMom` in instance method `brothers()`. The code for the loop constructing the array `broDad` has been erased (since it is similar to the code you are asked to write), but you do not have to write this missing code.

```
function Person(name, gender, mother, father, children)
{
  this.name = name || "unknown";
  this.gender = gender || "unknown";
  this.mother = mother || "unknown";
  this.father = father || "unknown";
  this.children = children || [];

  this.getName = function() { return this.name; };
  this.setName = function( name ) { this.name = name; };
  this.getgender = function() { return this.gender; };
  this.setgender = function( gender ) { this.gender = gender; };
  this.getmother = function() { return this.mother; };
  this.setmother = function( mother ) { this.mother = mother; };
  this.getfather = function() { return this.father; };
  this.setfather = function( father ) { this.father = father; };
  this.getchildren = function() { return this.children; };

  this.toString = function() { return this.name; };
}

function Person_addChild( child )
{
  

You write the body of this function.


}

Person.prototype.addChild = Person_addChild;

function Person_brothers()
{
  var brosMom = new Array(),
      brosDad = new Array(),
      mom = this.mother,
      dad = this.father,
      momsChildren,
      dadsChildren;

  if ( mom == "unknown" )
    momsChildren = [];
  else
    momsChildren = mom.children;
}
```

Continued

Continued from previous page

**You write the loop that constructs the array `broMom`**

```
if ( dad == "unknown" )
    dadsChildren = [];
else
    dadsChildren = dad.children;
```

**The loop constructing the array `broDad` has been erased. You do not have to write this loop.**

```
return broMom.concat( broDad );
}
Person.prototype.brothers = Person_brothers;
```

**A (8 pts.).** Write the body of `addChild()` as described in the second paragraph above.

**Answer**

```
if ( this.gender == "male" )
    child.father = this;
else if ( this.gender == "female" )
    child.mother = this;

this.children.push( child );
```

**b (10 pts.).** Write the loop in the body of the definition of `brothers()` that constructs the array `broMom` (the array of instances of `Person` that are brothers of the current instance by virtue of his/her mother). Loop over the array `momsChildren` and include only those that are male and distinct from the current instance.

**Answer**

```
for ( var i=0; i<momsChildren.length; i++ ) {
    var kidM = momsChildren[i];

    if ( kidM != this && kidM.gender == "male" )
        broMom.push( kidM );
}
```