CISC 131
Introduction to Programming and Problem Solving
Spring 2020
Multi-Dimension Arrays

Due: **Tuesday, May 12, 2020, at start of class**
Points: **None**

Solution to Problems from Lecture 22

Problem 2:
Write a function with this function header:

```
function concatenate (twoDimensionArray)
```

The function is passed a reference to a two dimension array. The elements of the array could be *Strings* or numbers or anything. The function returns a *String* that is the result of concatenating all the elements of the array together.

```
function concatenate (twoDimensionArray)
{
 var i;
 var j;
 var result;

 result = "";
 for ( i=0; i<twoDimensionArray.length; i++ )
 {
  for ( j=0; j<twoDimensionArray[i].length; j++ )
  {
   result = result + twoDimensionArray [i][j];
  } // for ( j=0
 }  // for ( i=0

 return result;
}
```

Problem 3:
Write a function with this function header:

```
function create (numberOfRows, numberOfColumns, intialValue)
```

The function is passed an integer number of rows and and integer number of columns. Each of these is greater than zero. The function returns a rectangular array with that number of rows and than number of columns. The elements of the array are intialized to *initialValue*.

```
function create (numberOfRows, numberOfColumns, intialValue)
{
 var i;
 var j;
 var result;

 result = new Array ( numberOfRows );
 for ( i=0; i<result.length; i++ )
 {
  result [i] = new Array ( numberOfColumns );
```

```
  for ( j=0; j<result[i].length; j++ )
  {
   result [i][j] = intialValue;
  } // for ( j=0
 }  // for ( i=0

 return result;
}
```

Start of Today's Lecture

Today we will continue with the discussion of two dimension arrays. As mentioned last time, Javascript and most modern programming languages implement multidimension arrays using the array of arrays approach rather than the old style monolithic approach. Both have advantages but, for most applications, the array of array approach is often better.

- easier for the system to manage memory
  Imagine you had an array with ten thousand rows and each row had ten thousand columns. With the monolithic approach, there would have to be contiguous space in memory to hold ten thousand times ten thousands (one hundred million) values. With the array of arrays approach, the largest contiguous space in memory would be that required to hold ten thousand values since each row of the array is, itself, a one dimension array and can be stored separately. This is like having a six pack of soda. The whole six pack might not fit in the refrigerator but the six individual cans will fit.
- since each row may have a different number of columns, less memory is need to store the array for arrays that are not rectangular[1]
- since each row is a single dimension array, there are some applications where existing single dimension array functions can be used to process the two dimension array.

Last time, I used associated arrays as a way of introducing two dimension arrays. The example used is shown below but here the last name array is shown to the left of the first name array. The array element number is shown in the leftmost column.

	lastName	firstName
0	Hall	Susan
1	Cooper	George
2	Arkwrigth	Carol
3	Green	Leo
4	Adams	Alice

People's names are what people want them to be. Different cultures have had different rules for naming people. In our society, names must be letters of the alphabet and may include the hyphen as a separator and everyone must have a last name. Some people (Prince comes to mind) has only a last name. You cannot have a first name unless you have a last name and you cannot have a middle name unless you have a first name. With that in mind, we can create three associated arrays that contain the names of different people. Again, the leftmost column contains the array element number.

	lastName	firstName	middleName
0	Stevenson	Robert	Louis
1	Homer[2]		
2	Taft	William	Howard
3	Franklin	Aretha	
4	Houdini		

These names could easily be stored in a five row by three column array. Using the monolithic approach, this would require storage for five times three or fifteen values, If the array of arrays approach is used, only ten values need to be stored. Here is how these three associated arrays would be declared in Javascript:

[1] A rectangular array is one in which each row has the same number of columns
[2] The author of the *Iliad* and the *Odyssey*, not the Simpson's character

```
var bob;
var personName;

 personName = [
               [ "Stevenson", "Robert", "Louis" ],
               [ "Homer" ],
               [ "Taft", "William", "Howard" ],
               [ "Franklin", "Aretha" ],
               [ "Houdini" ],
              ];
```

The array can be visualized this way. The row numbers of *personName* are shown in the leftmost column and the column numbers are shown in the top row.

		0	**1**	**2**
0	-->	Stevenson	Robert	Louis
1	-->	Homer		
2	-->	Taft	William	Howard
3	-->	Franklin	Aretha	
4	-->	Houdinit		

For example, *personName[2]* points to the single dimension array that contains three elements: *Taft*, *William*, and *Howard*. The value at *personName[3][1*] is *Aretha*.

If you are using a language like Javascript, there is no direct way to determine how many elements are in the array because there may not be the same number of columns in each row. To find out how many elements are in a two dimension array, you must use the *length* properties like this:

```
function totalElements ( array2D )
{
 var i;
 var result;

 result = 0;
 for ( i=0; i<array2D.length; i++ )
 {
  result = result + array2D[i].length;
 }
 return result;
}
```

If you use only one subscript when accessing a two dimension array, you are actually referencing a one dimension array. For example, if this statement were executed after the creation of the *personName* array:

```
bob = personName[0];
```

the variable *bob* would now contain a reference to a one dimension array – the same array as referenced by *personName[0]*. The array is not copied. There is still only one array but now there are two variables that reference it. This is analogous to have two variables reference the same *String*. For example:

```
var x;
var y;
x = "hat";
y = x;
```

There is only one *hat* but two variables that point to it.

What will happen when the following statements are executed?

```
var hold;
hold           = personName [1];
personName [1] = personName [3];
personName [3] = hold;
```

Using the array of array approach can pose some challenges, though. When you wrote a function to find the element number that contained the minimum value in a one dimension array, you could always assume that, if the array contained at least one element, then element zero of the array existed, too, and the value in element zero could be used as a starting point when looking for the minimum. In a two dimension array using the array of arrays approach, there is no guarantee that element [0][0] exists even when the array contains many elements. Here is an example:

```
var a
a = [
     [],
     [1,7],
     [11,62,39],
     [5,68]
    ];
```

The array *a* contains seven elements but *a[0][0]* does not exist. The zero element of array *a* points to a zero element single dimension array. Here is a two dimension array that has three rows but no elements:

```
a = [
      [],
      [],
      [],
     ];
```

As in finding the number of elements, you will need a function to find the first element in a two dimension array. The first element could be in any row of the array but it must always be in column zero of that row. Do you know why?

The following function will return the row number that contains the first element in the array. If the array contains no elements, the function returns *NaN*.

```
function findFirstElement ( array2D )
{
 var i;

 for(i=0; i<array2D.length && array2D[i].length<1; i++) {}
 if ( i === array2D.length ) i = NaN;
 return i;
}
```

For the following problems, the functions you write should not modify the contents of the parameter array unless specifically told to do so,.

Problem 1:

Write a function with this function header:

```
function findLastElement (twoDimenstionArray)
```

The function is passed a reference to a two dimension array. You can assume that the reference is never *null*. The function returns the row number that contains the last element of the array. If the

array contains no elements, the function returns *NaN*. You may only use one loop and may not call any other functions that you have written.

Problem 2:
Write a function with this function header:

```
function sameShape ( array2Da, array2Db )
```

The function is passed a reference to two two-dimension arrays. You can assume that the references are never *null*. The function returns Boolean *true* if the two arrays have the same shape and Boolean *false* if they do not have the same shape. The shape is the same if the two arrays have the same number of rows and there are the same number of columns in the i_{th} row of each array for every valid subscript *i*.

Problem 3:
Write a function with this function header:

```
function function copy ( array2D )
```

The function is passed a reference to a two dimension array. You can assume that the referenceis never *null*. The function returns a copy of the array. The copy must have the same shape and values in the same locations as the parameter array.

Problem 4:
Write aanother version of the two dimension array *copy* function. In this version, make use of the single dimension array *copy* function you wrote when single dimension arrays were introduced.