Assignment 2 - Spencer Meldrum and Tim Alander
Design Document - Part A

1) *What is the abstract thing you are trying to represent?*

We are representing a 2-Dimensional set of data in a linear fashion. For this we will use a sequence; an array in C (Hanson's U-Arrays). We will call this UArray2_T in C.

We will implement this 2-D plane as row-major in linear memory.

2) *What functions will you offer, and what are the contracts of that those functions must meet?*

We will provide these functions:

1) Uarray2_T UArray2_new (int columns, int rows, int size);

-this function would create a new UArray that would hold data designated for a two-dimensional array. It would take in as arguments the height (columns) and width (rows) and size (a number, in bytes, that one element occupies) If the user does not supply the correct size, the array may fail at runtime.

2) void* UArray2_at(UArray2_T t, int column, int row);

- this function will return a pointer to element stored at [column][row]. Since we have implemented our table as a row major, to resolve [x][y] to a linear index k. our algorithm does as follows: k = y*(Number of cols) + x

3) void UArray2_map_row_major(UArray2_T t, void apply(UArray2_T t, void* cl), void *cl);

- this function will call an apply function on every element in the provided UArray2_T. Column indices will vary more quickly than row indices.

4) void UArray2_map_column_major(UArray2_T t, void apply(UArray2_T t, void* cl), void *cl);

- this function will call an apply function on every element in the provided UArray2_T. Row indices will vary more quickly than column indices.

5) void UArray2_free(UArray2_T t);

- this function deallocates all memory used by the UArray2_T instance created when calling UArray2_new. It cannot be given a NULL pointer, or a pointer that is not a UArray2_T pointer.

6) int UArray2_size(UArray2_T t);

- this function returns the total size of the UArray2_T it is given.

It cannot be given a NULL pointer, or a pointer that is not a UArray2_T pointer.

7) int UArray2_rows(UArray2_T t);

- this function returns the number of rows in the UArray2_T it is given.
It cannot be given a NULL pointer, or a pointer that is not a UArray2_T pointer.

8) int UArray2_columns(UArray2_T t);

- this function returns the number of columns in the UArray2_T it is given.
It cannot be given a NULL pointer, or a pointer that is not a UArray2_T pointer.

4) *What representation will you use, and what invariants will it satisfy?*

We will represent this implementation as a linear UArray_T, with the following invariants:

- The length of the UArray will be equal to the size of the 2-d plane (expressed by multiplying the row*column arguments) supplied by the user in UArray2_new function.

- The size of a single data element in bytes will be exactly equal to the the size argument supplied by the user in UArray2_new function.

- Any two dimensional index (in the format [x][y], x being column, y being row) will only map to one index in the linear implementation.

Design Document - Part B

1) *What is the abstract thing you are trying to represent?*

We will be representing a two dimensional array of bits (0 or 1) in a linear fashion. For this, we will build a new interface built upon Hanson's Bit Vector. It will be called Bit2_T in C.

This will be implemented as row major.

2) *What functions will you offer, and what are the contracts of that those functions must meet?*

1) Bit2_T Bit2_new (int width, int height);

- this function will create a two dimensional array used solely for the storage of bits. It takes, as arguments, the width and height of the desired two dimensional bit array.

2) int Bit2_put(Bit2_T t, int row, int column, int bit);

- this function sets the bit at [width][height] to the bit supplied to the function. It returns the previous value. Since we have implemented our array as a row major, to resolve [x][y] to a linear index k our algorithm does as follows: k = y*(width) + x

3) int Bit2_get(Bit2_T t, int row, int column);

- this function returns the value of the bit stored at [width][height]. Since we have implemented our array as a row major, to resolve [x][y] to a linear index k our algorithm does as follows: k = y*(width) + x

3) int Bit2_height(Bit2_T t);

- this function returns the height of the Bit2_T it is given.
It cannot be given a NULL pointer, or a pointer that is not a Bit2_T pointer.

4) int Bit2_width(Bit2_T t);

- this function returns the width of the Bit2_T it is given.
It cannot be given a NULL pointer, or a pointer that is not a Bit2_T pointer.

5) int Bit2_size(Bit2_T t);

- this function returns the total size of the Bit2_T it is given.
It cannot be given a NULL pointer, or a pointer that is not a Bit2_T.

6) void Bit2_map_column_major(Bit2_T t, void apply(void* element, void* cl),
  void *cl);

- this function calls the apply function for every bit in the two dimensional array. Height indices will vary more quickly than width indices.

7) void Bit2_map_row_major(Bit2_T t, void apply(Bit2_T t, void* cl),
  void *cl);

- this function calls the apply function for every bit in the two dimensional array. Width indices will vary more quickly than height indices.

8) void Bit2_free(Bit2_T t);

- this function deallocates all memory used by the Bit2_T instance created when calling Bit2_new. It cannot be given a NULL pointer, or a pointer that is not a Bit2_T pointer.

4) *What representation will you use, and what invariants will it satisfy?*

We will represent this implementation as a linear Bit_T, with the following invariants:

- The length of the Bit_T will be equal to the number of bits in the image (expressed by multiplying the width*height arguments) supplied by the user in Bit2_new function.

- The value of each bit stored in the Bit2_T implementation must either be 0 or 1.

- Any two dimensional index (in the format [x][y], x being width, y being height) will only map to one index in the linear implementation.