iii Design Document Extraganza 😮

By Josh Field (jfield02) and Vedant Modi (vmodi01) !!!!

```
/*
 * uarray2.h
 * Josh Field (jfield02) and Vedant Modi (vmodi01)
 *
* UArray2 is an unboxed 2D array implemented using a single Hanson UArray,
and
 * that is as long as the width * height of the UArray2.
 */

#ifndef __UARRAY2__H_
#define __UARRAY2__H_

#include <stdbool.h>
#include "uarray.h"

#define T UArray2_T

struct T {
    UArray_T *array;
    int height;
    int width;
};

typedef struct T *T;

/********** UArray2_new ********
 *
 * Initializes a new UArray2 struct and returns a pointer to the new
struct
 *
 * Parameters:
 *      int width: the number of columns in the array
 *      int height: the number of rows in the array
 *      int size: the number of bits each element take up
 *
 * Return: A pointer to the new UArray2 struct, with init'ed dimensions
 *
 * Expects
 *      all parameters to be positive integers
 * Notes:
 *      Will CRE if the integers do not match expectations
 *      Uses Hanson Alloc() to create the object
 ************************/
T UArray2_new(int width, int height, int size);
```

```
/********** UArray2_free ********
 *
 * Clears the UArray2 from memory
 *
 * Parameters:
 *      T *uarray2: address of the UArray2 that the client wishes to free
 *
 * Return: Nothing
 *
 * Expects
 *      Memory to not be freed, UArray2_ will not be null
 * Notes:
 *      Will CRE if free has been called on already free memory
 *      Relies on UArray_free() function
 ************************/
void UArray2_free(T *uarray2);

/********** UArray2_width ********
 *
 * Returns the width of the provided UArray2
 *
 * Parameters:
 *      T uarray2: the UArray2 of which the client desires the width
 *
 * Return: Number of columns in the array
 *
 * Expects
 *      UArray2 to not be NULL
 * Notes:
 *      Will CRE if UArray2 is NULL
 ************************/
int UArray2_width(T uarray2);

/********** UArray2_height ********
 *
 * Returns the height of the provided UArray2
 *
 * Parameters:
 *      T uarray2: the UArray2 of which the client desires the width
 *
 * Return: Number of rows in the array
 *
 * Expects
 *      UArray2 to be not NULL
 * Notes:
 *      Will CRE if UArray2 is NULL
 ************************/
int UArray2_height(T uarray2);

/********** UArray2_size ********
 *
 * Returns the size of the provided UArray2
 *
 * Parameters:
```

```
 *      T uarray2: the UArray2 of which the client desires the size
 *
 * Return: Size of each element in the array in bits
 *
 * Expects
 *      UArray2 to be not NULL
 * Notes:
 *      Will CRE if UArray2 is NULL
 ************************/
int UArray2_size (T uarray2);

/********** UArray2_at********
 *
 * Returns the height of the provided UArray2
 *
 * Parameters:
 *      T uarray2: the UArray2 of which the client desires the height
 *      int x: The distance from the top of the array
 *      int y: The distance from the left of the array
 *
 * Return: Number of rows in the array
 *
 * Expects
 *      UArray2 to be not NULL
 *      x = [0, width) and x is an integer
 *      y = [0, height) and y is an integer
 * Notes:
 *      Will CRE if UArray2 is NULL
 *      Will CRE if the range expectations are violated
 ************************/
void *UArray2_at(T uarray2, int x, int y);

/********** UArray2_map_col_major ********
 *
 * Traverses the array where columns increase faster than rows, applying
 * the provided function to each intialized index.
 *
 * Parameters:
 *      T uarray2: the UArray2 of which the client desires a traversal of
 *      void apply: The function that will be applied to each index
 *      bool *OK: Bool representing if the traversal succeeded
 *
 * Return: Nothing
 *
 * Expects
 *      UArray2 to be not NULL
 * Notes:
 *      Will CRE if UArray2 is NULL
 ************************/
void UArray2_map_col_major(T uarray2, void apply(int x, int y, T uarray2,
                                        void *elem, void *cl), bool *OK);

/********** UArray2_map_row_major ********
 *
```

```
 * Traverses the array where rows increase faster than columns, applying
 * the provided function to each intialized index.
 *
 * Parameters:
 *      T uarray2: the UArray2 of which the client desires a traversal of
 *      void apply: The function that will be applied to each index
 *      bool *OK: Bool representing if the traversal succeeded
 *
 * Return: Nothing
 *
 * Expects
 *      UArray2 to be not NULL
 * Notes:
 *      Will CRE if UArray2 is NULL
 ************************/
void UArray2_map_row_major(T uarray2, void apply(int x, int y, T uarray2,
                                        void *elem, void *cl), bool *OK);

/********** flattened_index ********
 *
 * Returns the corresponding 1 dimensional index for the provided two
 * dimensional index
 *
 * Parameters:
 *      T uarray2: the UArray2 of which the client desires to access
 *      int x: The distance from the top of the array
 *      int y: The distance from the left of the array
 *
 * Return: The 1 dimesional index for the given two dimensional index
 *
 * Expects
 *      UArray2 to be not NULL
 *      x = [0, width) and x is an integer
 *      y = [0, height) and y is an integer
 * Notes:
 *      Will CRE if UArray2 is NULL
 *      Will CRE if the range expectations are violated
 ************************/
int flattened_index(T uarray2, int x, int y);

#undef T
#endif
```

```
/*
 * bit2.h
 * Josh Field (jfield02) and Vedant Modi (vmodi01)
 *
 * Bit2 is an implementations of a 2D bit vector, implemented by using a
1D bit
 * vector with converted indices.
 */
```

```
#ifndef __BIT2__H_
#define __BIT2__H_

#include <stdbool.h>
#include <bit.h>

#define T Bit2_T

struct T {
    Bit_T *array;
    int height;
    int width;
};

typedef struct T *T;

/* Uses Hanson ALLOC() */

T Bit2_new(int width, int height, int size);
/* Relies on Bit_free */
void Bit2_free(T *uarray2);

/* It is a checked runtime error for n to be negative or to be equal to or
greater than the length of set, or for bit to be other than zero or one.
NOTE: n here would be the flattened index of the bit */

int Bit2_get(T set, int x, int y);
int Bit2_put(T set, int x, int y, int bit);

/* boring, just return the member variable in the struct */
int Bit2_width(T bit2);
/* boring, just return the member variable in the struct */
int Bit2_height(T bit2);
/* Uses UArray size */
int Bit2_size (T bit2);


void Bit2_map_col_major(T bit2, void apply(int x, int y, T bit2,
                                           void *elem, void *cl), bool *OK);
void Bit2_map_row_major(T bit2, void apply(int x, int y, T bit2,
                                           void *elem, void *cl), bool *OK);

/* Changes 2D index to 1D index */
int flattened_index(T bit2, int x, int y);

#undef T
#endif
```