

```

/*
 * Name      : uarray2.h
 * Assignment : CS40 Homework 2 (iii)
 * Purpose   : Represents a 2-dimensional array of data of any type
 * Editors    : Matthew Wong (mwong14), Ivi Fung (sfung02)
 */

#ifndef UARRAY2_H_
#define UARRAY2_H_

typedef struct UArray2_T *UArray2_T;

/*
 * Name:      : UArray2_new
 * Purpose   : Create a rectangular 2D array (aka not jagged) that stores
 *             data of any type unboxed
 * Parameters : (int) The width of the 2D array;
 *             (int) The height of the 2D array;
 *             (int) The number of bytes each item takes
 * Return    : (UArray2_T) An array that can hold the new data
 * Notes     : Can only hold one type of data, but won't check
 *             Will CRE on failure to allocate memory
 *             Will CRE if width, height, or size are negative, or size is 0
 */
UArray2_T UArray2_new(int width, int height, int size);

/*
 * Name:      : UArray2_free
 * Purpose   : Free up the memory the 2D array after the client is done
 * Parameters : (UArray2_T *) The pointer to the 2D array to free up
 * Return    : None
 * Notes     : Sets the pointer data to NULL so no accidental references
 *             can be made;
 *             Does not free up the data in the 2D array;
 *             use UArray2_map_col_major or UArray2_map_row_major to
 *             map a free function to free up all of the data
 */
void UArray2_free(UArray2_T *uArray2Pointer);

/*
 * Name:      : UArray2_at
 * Purpose   : Get the pointer to the element at (col, row)
 *             for setting and getting
 * Parameters : (UArray2_T) The 2D array where the data is stored

```

```

*      (int) The column that the data is located at in the 2D array;
*      (int) The row that the data is located;
* Return   : (void *) The pointer to the data location at the given index
* Notes    : Does not check if data has previously been set there or not;
*           only gives the pointer to the data
*           Undefined behavior before setting the pointer to (col, row)
*           to any value;
*           CRE when out-of-bounds
*/

```

```
void *UArray2_at(UArray2_T uArray2, int col, int row);
```

```

/*
* Name:      : UArray2_width
* Purpose    : Get the width of the 2D array
* Parameters : (UArray2_T) The 2D array to get the width of
* Return     : (int) The width of the given 2D array
* Notes      : Width can't be changed after being initially set
*/

```

```
int UArray2_width(UArray2_T uArray2);
```

```

/*
* Name:      : UArray2_height
* Purpose    : Get the height of the 2D array
* Parameters : (UArray2_T) The 2D array to get the height of
* Return     : (int) The height of the given 2D array
* Notes      : Height can't be changed after being initially set
*/

```

```
int UArray2_height(UArray2_T uArray2);
```

```

/*
* Name:      : UArray2_size
* Purpose    : Get the byte size of the element type stored in the 2D array
* Parameters : (UArray2_T) The 2D array which contains the type of element
*           that the client is getting the size of
* Return     : (int) The size of the type in bytes
* Notes      : Only one type can be used in the UArray2_T,
*           but doesn't have type-checking
*/

```

```
int UArray2_size(UArray2_T uArray2);
```

```

/*
* Name:      : UArray2_map_col_major
* Purpose    : Use the apply function on each element of the 2D array going
*           column by column, from top of the column to the

```

```

*      bottom of the column
* Parameters : (UArray2_T) The 2D array with the elements;
*      (function) the function to apply to every element, which has
*      parameters that the column, row, 2D array,
*      pointer to the data value, and a pointer to an accumulator
*      variable
*      (void *) The pointer to the accumulator in its initial state
* Return    : None
* Notes     : This is a mapping function
*/
void UArray2_map_col_major(UArray2_T uArray2,
                          void (*apply)(int, int, UArray2_T, void *, void *),
                          void *cl);

/*
* Name:      : UArray2_map_row_major
* Purpose   : Use the apply function on each element of the 2D array going
*      row by row, from left of the row to the right of the row
* Parameters : (UArray2_T) The 2D array with the elements;
*      (function) the function to apply to every element, which has
*      parameters that the column, row, 2D array,
*      pointer to the data value, and a pointer to an accumulator
*      variable
*      (void *) The pointer to the accumulator in its initial state
* Return    : None
* Notes     : This is a mapping function
*/
void UArray2_map_row_major(UArray2_T uArray2,
                          void (*apply)(int, int, UArray2_T, void *, void *),
                          void *cl);

#endif

```

```

/*
 * Name      : bit2.h
 * Assignment : CS40 Homework 2 (iii)
 * Purpose   : Represents a 2-dimensional array of bits
 * Editors    : Matthew Wong (mwong14), Ivi Fung (sfung02)
 */

#ifndef BIT2_H_
#define BIT2_H_

typedef struct Bit2_T *Bit2_T;

/*
 * Name:      : Bit2_new
 * Purpose   : Create a rectangular 2D array that stores bits
 * Parameters : (int) The height of the 2D array;
 *              (int) The width of the 2D array;
 * Return    : (Bit2_T) An array that can hold the bits
 * Notes     : Will CRE on failure to allocate memory
 *              Will CRE if width, height, or size are negative
 */
Bit2_T Bit2_new(int width, int height);

/*
 * Name:      : Bit2_free
 * Purpose   : Free up the memory the 2D array after the client is done
 * Parameters : (Bit2_T *) The pointer to the 2D array to free up
 * Return    : None
 * Notes     : Sets the pointer data to NULL so no accidental references
 *              can be made
 */
void Bit2_free(Bit2_T *bit2Pointer);

/*
 * Name:      : Bit2_get
 * Purpose   : Get the bit at (col, row)
 * Parameters : (Bit2_T) The 2D array where the data is stored
 *              (int) The column that the data is located at in the 2D array;
 *              (int) The row that the data is located
 * Return    : (int) Returns 0 if the bit is 0 and 1 if the bit is 1
 * Notes     : Does not check if data has previously been set there or not;
 *              only gives the pointer to the data
 *              Undefined behavior before setting the pointer to (col, row)
 *              to any value;

```

```

*          CRE when out-of-bounds
*/
int Bit2_get(Bit2_T bit2, int col, int row);

/*
* Name:      : Bit2_put
* Purpose   : Set the bit at (col, row)
*           for setting and getting
* Parameters : (Bit2_T) The 2D array where the data is at
*              (int) The column that the data is located at in the 2D array;
*              (int) The row that the data is located
*              (int) The value to set the bit to
* Return    : None
* Notes     : Does not check if data has previously been set there or not;
*           only gives the pointer to the data
*           Undefined behavior before setting the pointer to (col, row)
*           to any value;
*           CRE when out-of-bounds
*           CRE when data is not 0 or 1
*/
void Bit2_put(Bit2_T bit2, int col, int row, int data);

/*
* Name:      : Bit2_width
* Purpose   : Get the width of the 2D array
* Parameters : (Bit2_T) The 2D array to get the width of
* Return    : (int) The width of the given 2D array
* Notes     : Width can't be changed after being initially set
*/
int Bit2_width(Bit2_T bit2);

/*
* Name:      : Bit2_height
* Purpose   : Get the height of the 2D array
* Parameters : (Bit2_T) The 2D array to get the height of
* Return    : (int) The height of the given 2D array
* Notes     : Height can't be changed after being initially set
*/
int Bit2_height(Bit2_T bit2);

/*
* Name:      : Bit2_map_col_major
* Purpose   : Use the apply function on each bit of the 2D array going
*           column by column, from top of the column to the

```

```

*           bottom of the column
* Parameters : (Bit2_T) The 2D array with the bits;
*             (function) the function to apply to every bit, which has
*             parameters that the column, row, 2D array,
*             the data value, and a pointer to an accumulator
*             variable
*             (void *) The pointer to the accumulator in its initial state
* Return    : None
* Notes     : This is a mapping function
*/

```

```

void Bit2_map_col_major(Bit2_T bit2,
                        void (*apply)(int, int, Bit2_T, int, void *),
                        void *cl);

```

```

/*
* Name:      : Bit2_map_row_major
* Purpose    : Use the apply function on each bit of the 2D array going
*             row by row, from left of the row to the right of the row
* Parameters : (Bit2_T) The 2D array with the bits;
*             (function) the function to apply to every bit, which has
*             parameters that the column, row, 2D array,
*             the data value, and a pointer to an accumulator
*             variable
*             (void *) The pointer to the accumulator in its initial state
* Return     : None
* Notes      : This is a mapping function
*/

```

```

void Bit2_map_row_major(Bit2_T bit2,
                        void (*apply)(int, int, Bit2_T, int, void *),
                        void *cl);

```

```

#endif

```