Camren Woodstock
Jesse Hazard
Design Document

1. [*] What is the abstract thing you are trying to represent? Often the answer will be in terms of sets, sequences, and finite maps.

For part A, a two-dimensional, polymorphic, unboxed array. For part B a two dimensional array of bits.

2. What functions will you offer, and what are the contracts of that those functions must meet?
Functions for UArray2:
extern UArray2_new(int length, int width, int size) - create and allocate memory for a new array given length and size

extern UArray2_at(T UArray2,, int x, int y) - will get the piece of data at the given row and column index.

extern UArray2_map_row_major(T UArray2, apply(int xIndex, int yIndex, T UArray2, void indexPtr, void *cl), void *cl) - calls an apply function for each element in the array. Column indices vary more rapidly than row indices.

extern UArray2_map_col_major(T UArray2, apply(int xIndex, int yIndex, T UArray2, void indexPtr, void *cl), void *cl)- calls an apply function for each element in the array. Row indices vary more rapidly than column indices.

extern UArray2_free(T *UArray2) - free the memory of the two dimensional array.

Functions for Bit2:

extern T Bit2_new(int length, int width); // create 2d array;
extern int Bit2_length(T Bit2); // returns the length
extern int Bit2_width(T Bit2); // returns the width
extern int Bit2_area(T Bit2); // returns the area
extern void Bit2_free(T *Bit2); // frees the memory of the 2D array
extern void Bit2_size(T Bit2); // returns the size of array
extern int Bit2_get(T Bit2, int x, int y); // return bit value
extern int Bit2_put(T Bit2, int x, int y, int bit); // put into 2d array
// Call apply function on every element
extern Bit2_map_rowl_major(T Bit2, apply(int xIndex, int yIndex, T Bit2, void idxPtr, void *cl), void *cl)

extern Bit2_map_col_major(T Bit2, apply(int xIndex, int yIndex, T Bit2, void idxPtr, void *cl), void *cl)

3. What examples do you have of what the functions are supposed to do?

There are many examples in Hanson's Interfaces and Implementations. Chapter 10 describes how the interface for a one dimensional array is implemented. Also chapter 13 describes how the bit interface is implemented. We will use examples from both chapters to create our own two dimensional array interface.

4. What representation will you use, and what invariants will it satisfy? (This question is especially important to answer precisely.)

We will represent UArray2_T as an array of UArray_Ts. This satisfies three invariants. First, when Uarray2_T is extended, a new UArray_T is allocated of length equal to the width of the double array. Second, the only way to extend any of the UArray_T rows is to extend all of them. Third, because there is only one base object, it is impossible to have pointer inconsistency.

5. When a representation satisfies all invariants, what abstract thing from step [<-] does it represent? (This question is also especially important to answer precisely.)

A NULL pointer represents the empty set. A pointer T that isn't null represents the union of sets that T->1 and T->r, together with the set containing the event that T points to.

6. What test cases have you devised?
To test our UArray2 mapping functions we will write a program that reads and writes a graymap by calling UArray2_map_row_major or UArray2_map_col_major with a function argument that calls pnmrdr_get from the interface. Once we do this we can test our output to see if it matches the output as pnmtoplainpnm. We will also use the same technique to test our Bit2_map_col_major with a function argument that calls a pnmrdr function from the interface.

7. What programming idioms will you need?

We will need the idiom for void** pointers.
The idiom for allocating and initializing pointers to structures.
The idiom for using unboxed arrays. We will also be looking at the example of the array of arrays in the idioms document.