TIMI DAYO-KAYODE
FAVOUR OKEREKE-MBA
02/05/2018
HW2 DESIGN CHECKLIST
COMP 40 SPRING 2018

## PART A: UARRAY2_T

- [*] What is the abstract thing you are trying to represent? Often the answer will be in terms of sets, sequences, finite maps, and/or objects in the world. Always the answer is client-oriented.
    - We are trying to represent a 2D array.

- What functions will you offer, and what are the contracts those functions must meet?
    - We will offer the following functions:
        - UArray2_new(int width, int height, int size); returns a pointer to the initialized array
        - UArray2_free(UArray2_T *myArray); returns nothing
        - UArray2_width(UArray2_T myArray); returns the length of the rows of the array
        - UArray2_height(UArray2_T myArray); returns the length of the columns of the array
        - UArray2_size(UArray2_T myArray); returns the size of each element of the array
        - UArray2_at(UArray2_T myArray, int row, int col); returns a pointer to the address of the location specified.
        - UArray2_map_col_major(UArray2_T myArray, void apply(int i, int j, UArray2_T a, void *p1, void *p2), void *cl)
        - UArray2_map_row_major(UArray2_T myArray, void apply(int i, int j, UArray2_T a, void *p1, void *p2), void *cl)

- What examples do you have of what the functions are supposed to do?
    - We have examples from the supplemental Hanson's chapter written with Norman.
    - UArray2_T myArray = UArray2_new(width, len, size) //len has value 10, and width has value 10, size has value 4.
    - int *myVal = UArray2_at(myArray, loc1, loc2) //loc1 has value 6 and loc2 has value 4
    - int mySize = UArray2_size(myArray)
    - Check that sizeof(*myVal) == mySize
    - UArray2_free(&myArray)

- What representation will you use, and what invariants will it satisfy? (This question is especially important to answer precisely.)
    - Our 2D representation of the unboxed array data structure is represented as a single instance of a UArray2_T of length row * column.
    - The invariants that our representation must satisfy are as follows:
        - The length of the 1D array must be equal to the value of the row * major; no more, no less.
        - The size of data stored in each index of the array must be equal to the initial size specified by the user when the UArray2_T was initialized
- How does an object in your representation correspond to an object in the world of ideas? That is, what is the inverse mapping from any particular representation to the abstraction that it represents. This question is also especially important to answer precisely.)
    - For any particular index (row,col) in the representation, it would be mapped to its location in our 1D array using the function; (row * width) + col.

- What test cases have you devised?
    - Our first tests will involve inserting integer values into the UArray using the UArray2_at function, and then printing out with both the UArray2_map_col_major and the UArray2_map_row_major and check that the values are printed in the right order
    - We will also use the UArray2_map_row_major to read and print out the pixels of a pgm file using the Pnmrdr interface since the pixels are stored in row major order. We will then compare the results with pnmtoplainpm to see if the results are the same
    - We will also then read the pgm image to our array with the UArray2_map_col_major function and print out the array using the UArray2_map_row_major. The result should be the same as calling pnmflip -transpose on the pgm image
    - Passing any non-positive number such as a value less than 1 to the UArray2_new function should end the program with a Checked Runtime Error.
    - If the pointer passed to the UArray2_free, UArray2_width, UArray2_height, or UArray2_size function is NULL, or the address it points to is NULL, the program should end with a Checked Runtime Error.
    - If the the pointer passed to the UArray2_at function is NULL, or points to NULL, or the multiplication of the row and col values it is passed is greater than the length of our 1D representation of the 2D array, the program should end with a checked runtime error.
    - We will also create an executable with our implementation of UArray2_T and the userarray2.c provided for us, and then compare the results of running the executable with the results of running the .c file with the implementation provided to us. The results should be identical.

- What programming idioms will you need?

- The idiom for creating an abstract type using incomplete structures(Hanson Style)
- The idioms for using unboxed arrays
    - Initializing array elements
    - Storing values in an unboxed array
        - We will not be using the idiom example of array of arrays as our implementation of a 2D array uses a different algorithm for simulating the 2D array experience on the client/interface end.

## PART B: TWO DIMENSIONAL ARRAY OF BITS

- [*] What is the abstract thing you are trying to represent? Often the answer will be in terms of sets, sequences, finite maps, and/or objects in the world. Always the answer is client-oriented.
    - We are trying to use a 2D array of bits to represent an image

- What functions will you offer, and what are the contracts those functions must meet?
    - We will offer the following functions:
        - Bit2_new(int width, int height); returns a pointer to the initialized array
        - Bit2_free(Bit2_T *myArray); returns nothing
        - Bit2_width(Bit2_T myArray); returns the length of the rows of the array
        - Bit2_height(Bit2_T myArray); returns the length of the columns of the array
        - Bit2_put(Bit2_T myArray, int row, int col, int bit); sets that (row,col) value to bit and returns the previous value of that location
        - Bit2_get(Bit2_T myArray, int row, int col); returns the value at position (row,col)
        - Bit2_map_col_major(Bit2_T myArray, void apply(int i, int j, Bit2_T a, int b, void *p1), void *cl)
        - Bit2_map_row_major(Bit2_T myArray, void apply(int i, int j, Bit2_T a, int b, void *p1), void *cl)

- What examples do you have of what the functions are supposed to do?
    - We have examples from Hanson's textbook.
    - Bit2_T myArray = Bit2_new(width, len) //len has value 10, and width has value 10
    - Bit2_put(myArray, loc1, loc2, bit) //loc1 has value 6, loc2 has value 4, and bit has a value of 0
    - int myVal = Bit2_get(myArray, loc1, loc2) //loc1 has value 6 and loc2 has value 4
    - if(myVal) should be false.
    - Bit2_free(&myArray)

- What representation will you use, and what invariants will it satisfy? (This question is especially important to answer precisely.)
    - Our 2D representation of the bit array is represented as a single instance of a Bit_T (Bit vector) of length row * column.
    - The invariants that our representation must satisfy are as follows:
        - The length of the bit vector must be equal to the value of the row * major; no more, no less.
        - The value of a position in the 2D array should be either 0 or 1 since it is a representation of bits

- How does an object in your representation correspond to an object in the world of ideas? That is, what is the inverse mapping from any particular representation to the abstraction that it represents. This question is also especially important to answer precisely.)
    - For any particular index (row,col) in the representation, it would be mapped to its location in our Bit vector using the function; (row * width) + col.

- What test cases have you devised?
    - Our first tests will involve inserting integer values of 0 or 1into the Bit2_array using the Bit2_put function, and then printing out with both the Bit2_map_col_major and the Bit2_map_row_major and checking that the values are printed in the right order
    - Passing any non-positive number such as a value less than 1 to the Bit2_new function should end the program with a Checked Runtime Error.
    - Passing an integer value less than 1, or greater than the length of our Bit vector representation of the 2D array to the Bit2_put or Bit2_get functions will result in a checked runtime error
    - If the pointer passed to the Bit2_free, Bit2_width, or Bit2_height function is NULL, or the address it points to is NULL, the program should end with a Checked Runtime Error.
    - If the the pointer passed to the Bit2_get or Bit2_put functions are NULL, or points to NULL, or the multiplication of the row and col values it is passed is greater than the length of our Bit vector representation of the 2D array, the program should end with a checked runtime error.
    - We will also create an executable with our implementation of Bit2_T and the usebit2.c provided for us, and then compare the results of running the executable with the results of running the .c file with the implementation provided to us. The results should be identical.

- What programming idioms will you need?
    - The idiom for creating an abstract type using incomplete structures(Hanson Style)
    - The idioms for using unboxed arrays
        - Initializing array elements

- Storing values in an unboxed array
    - We will not be using the idiom example of array of arrays as our implementation of a 2D array uses a different algorithm for simulating the 2D array experience on the client/interface end.