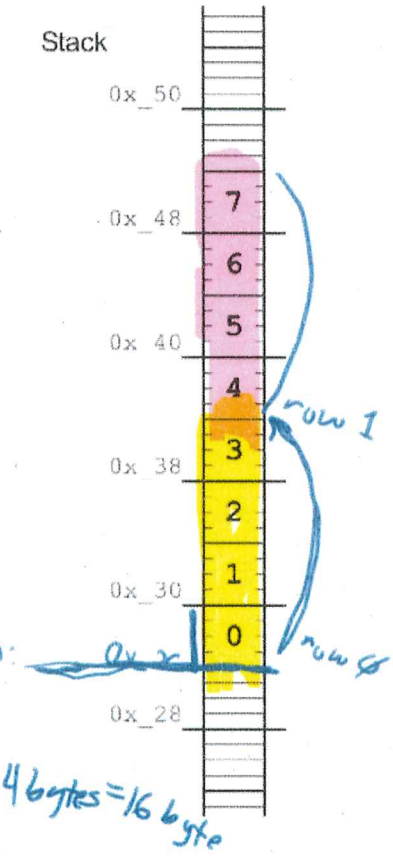
2D Arrays on the Stack

Stack Allocated 2D Arrays in C

```
void someFunction(){
  int m[2][4] = {{0,1,2,3},{4,5,6,7}};// SAA
```



* *2D arrays allocated on the stack* are laid out in row major order as a single contiguous block of memory with one row after the other

Stack & Heap 2D Array Compatibility

→ For each one below, what is provided when used as a source operand? What is its type and scale factor?

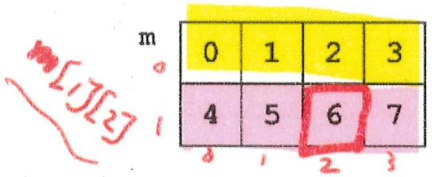
1. `**m`? ≡ *(*(m)) ≡ m[0][0] (ind element)
   type? int
   scale factor? no addr arithmetic
2. `*m`? `*(m+i)`? row 0, row i (a row) *(m+1)
   type? int *
   scale factor?
   SAA: #Colums * sizeof(element) = 4 cols * 4 bytes = 16 bytes
   AofA HEAP: sizeof(int*) = 4 bytes
3. `m[0]`? `m[i]`? row 0, row i, same as ?. (a row)
4. `m`? int ** (a 2D arr)
   type?
   scale factor?

For 2D STACK Arrays ONLY

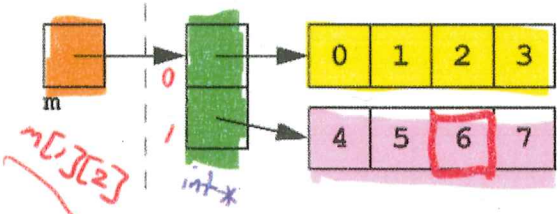
* `m` and `*m` are at same address, but not same type
* `m[i][j]` ≡ *(*(m+i)+j)) ≡ *(*m + COLS*i + j)
  SAA ONLY

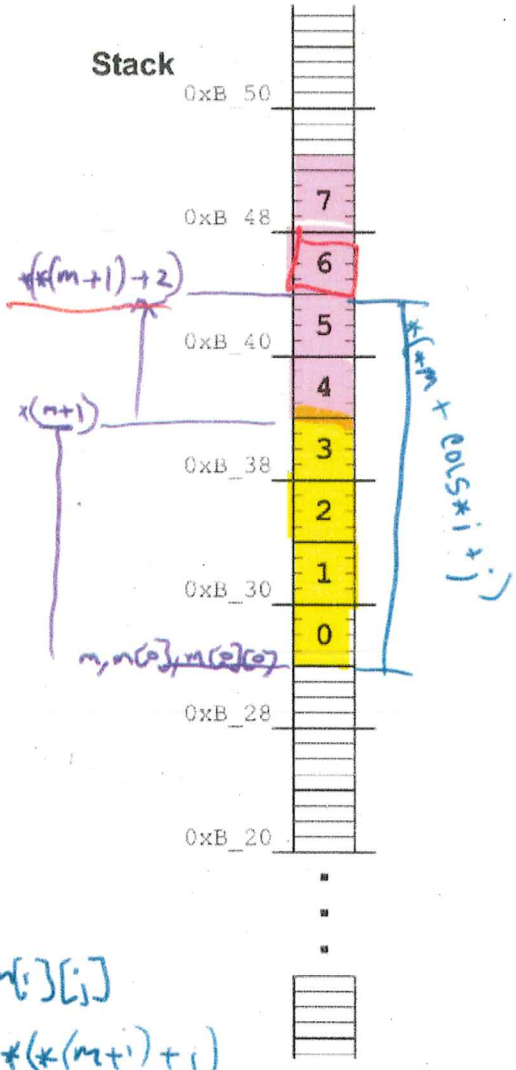
2D Arrays: Stack vs. Heap

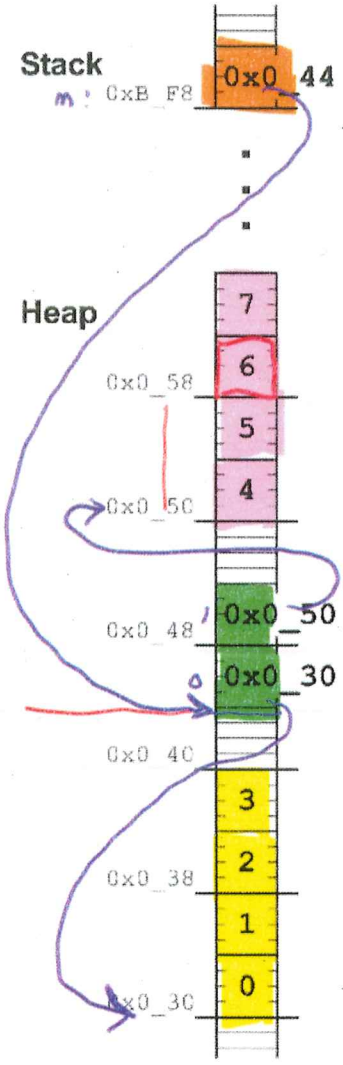
Stack: row-major order layout



Heap: array-of-arrays layout







SAA: $*(*m + \text{COLS} * i + j)$

$m[i][j] \equiv *(*(m+i)+j)$

Array Caveats

- *Arrays have no bounds checking!*

```
int a[5]; //SAA
for (int i = 0; i < 11; i++)
   a[i] = 0;
```

BUFFER OVERFLOW

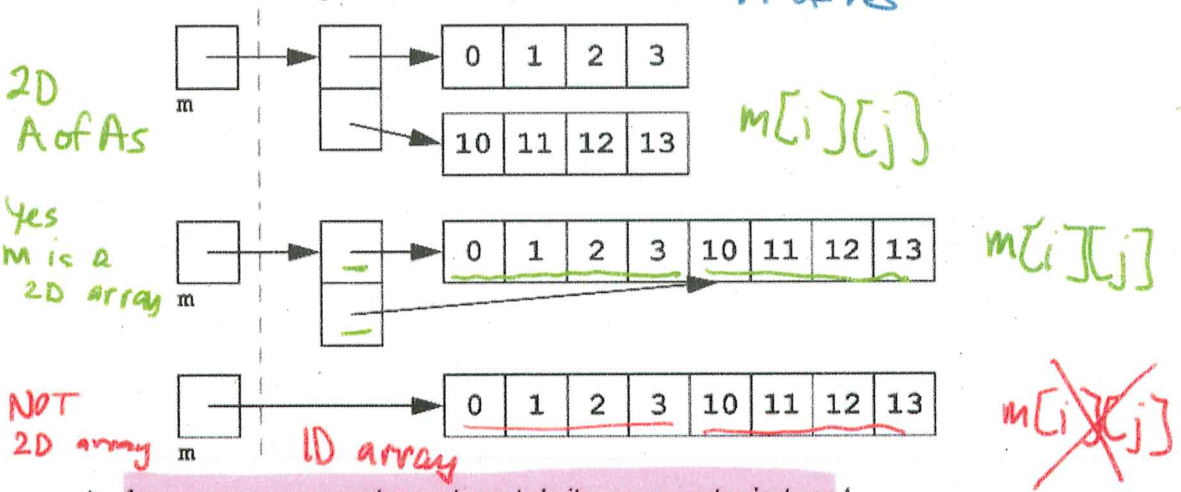
- *Arrays cannot be return types!*

```
int[] makeIntArray(int size) {
   return malloc(sizeof(int) * size);
}
```

int *

COMPILER ERROR

- *Not all 2D arrays are alike!*
  - → What is the layout for ALL 2D arrays on the stack? contiguous fixed size row-major order
  - → What is the layout for 2D arrays on the heap? A of As



- *An array argument must match its parameter's type!*
- *Stack allocated arrays require all but their first dimension specified!*

```
int a[2][4] = {{1,2,3,4},{5,6,7,8}}; //SAA
printIntArray(a,2,4); //size of 2D array must be passed in (last 2 arguments)
```

  - → Which of the following are type compatible with a declared above?

```
void printIntArray(int a[2][4],int rows,int cols)   OKAY
void printIntArray(int a[8][4],int rows,int cols)   OKAY
void printIntArray(int a[][4], int rows,int cols)   OKAY
void printIntArray(int a[4][8],int rows,int cols)   NO, cols must match array
void printIntArray(int a[][],  int rows,int cols)   NO
void printIntArray(int (*a)[4],int rows,int cols)   OKAY
void printIntArray(int **a,    int rows,int cols)   NO, cols must MATCH
```

okay for 2D AoAs (HEAP)

  - → Why is all but the first dimension needed?

For SAA, Compiler requires # of columns to find next row