CS 354 - Machine Organization & Programming
Thursday, September 19, 2019

Project p1 (3%): DUE at 10 pm on Monday, September 23rd

Project p2A (3%) DUE: 10 pm, Monday, September 30th

Homework hw1 (1.5%) DUE: 10 pm, Friday, September 27th

Last Time

Passing Addresses
1D Arrays on the Heap
Pointer Caveats
Meet C Strings
Meet `string.h`

Today

Meet `string.h` (from last time)
Recall 2D Arrays
2D Arrays on the Heap
2D Arrays on the Stack
2D Arrays: Stack vs. Heap
Array Caveats

Next Time

Structures

Read:

K&R Ch. 6.1: Basic Structures
K&R Ch. 6.2: Structures and Functions
K&R Ch. 6.3: Arrays and Structures
K&R Ch. 6.4: Pointers to Structure

Recall 2D Arrays

2D Arrays in Java

```
int[][] m = new int[2][4];
```

→ Diagram the resulting 2D array:

```
for (int i = 0; i < 2; i++)
   for (int j = 0; j < 4; j++)
      m[i][j] = i + j;
```

➢ What is output by this code fragment?

```
for (int i = 0; i < 2; i++) {
   for (int j = 0; j < 4; j++)
      printf("%i", m[i][j]);
   printf("\n");
}
```

→ What memory segment does Java use to allocate 2D arrays?

→ What technique does Java use to layout a 2D array?

→ What does the memory allocation look like for `m` in the code at the top of the page?

2D Arrays on the Heap

2D "Array of Arrays" in C

→ **1. Make a 2D array pointer named `m`.**
Declare a pointer to an integer pointer.

→ **2. Assign `m` an "array of arrays".**
Allocate of a 1D array of integer pointers of size 2 (the number of rows).

→ **3. Assign each element in the "array of arrays" it own row of integers.**
Allocate for each row a 1D array of integers of size 4 (the number of columns).

➢ What is the contents of `m` after the code below executes?

```
for (int i = 0; i < 2; i++) {
   for (int j = 0; j < 4; j++)
      m[i][j] = i + j;
```

→ Write the code to free the heap allocated 2D array.

❋ *Free the components of your heap memory*

Address Arithmetic

→ Which of the following are equivalent to `m[i][j]`?

a.) `*(m[i]+j)`
b.) `(*(m+i))[j]`
c.) `*(*(m+i)+j)`

❋ *`m[i][j]`*

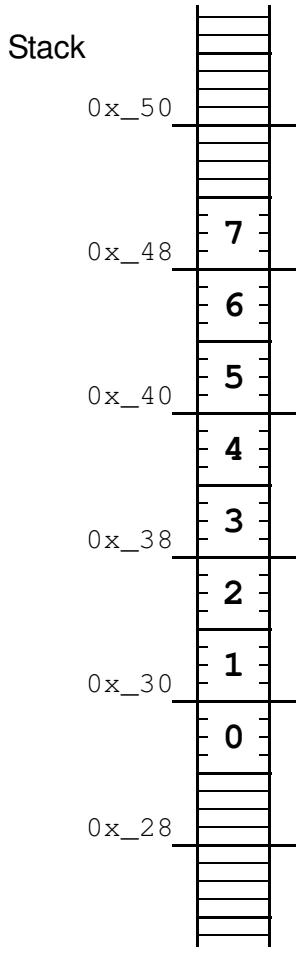
compute row `i`'s address
dereference address in 1. gives
compute element `j`'s address in row `i`
dereference the address in 3. to access element at row `i` column `j`

❋ *`m[0][0]`*

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}};
```



❋ *2D arrays allocated on the stack*

For 2D Arrays on the Stack only:

→ What is m?

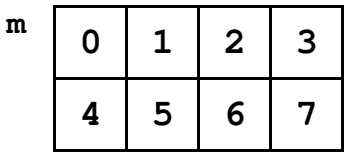
→ What is *m?

❋ `m` *and*

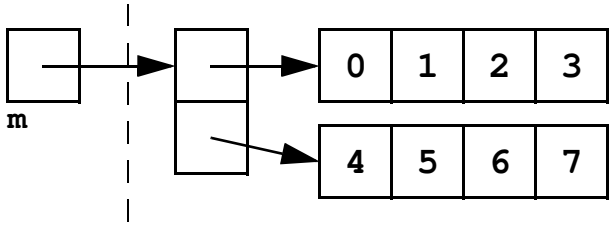
❋ `m[i][j]`

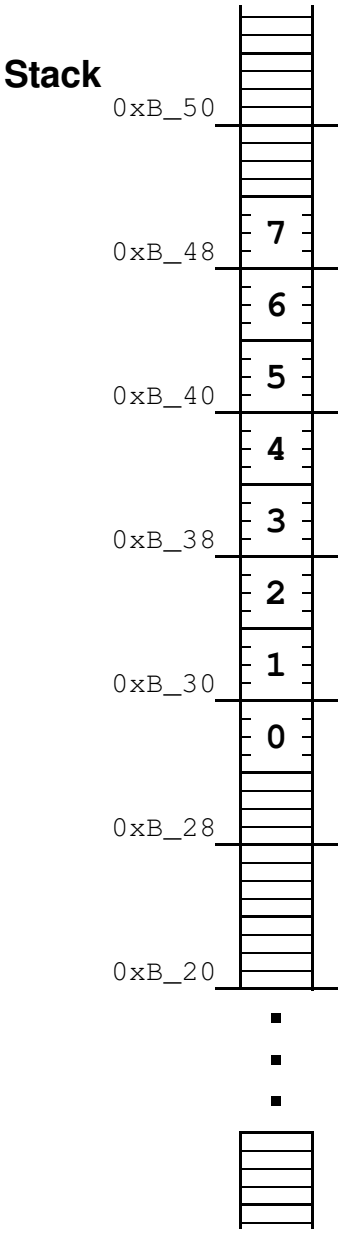
2D Arrays: Stack vs. Heap

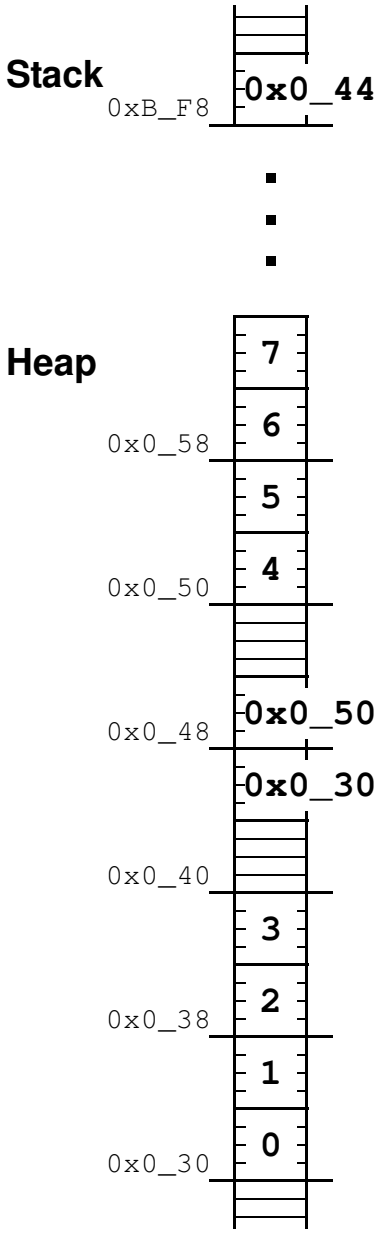
Stack: row-major order layout



Heap: array of arrays layout







Array Caveats

❋ *Arrays have no bounds checking!*

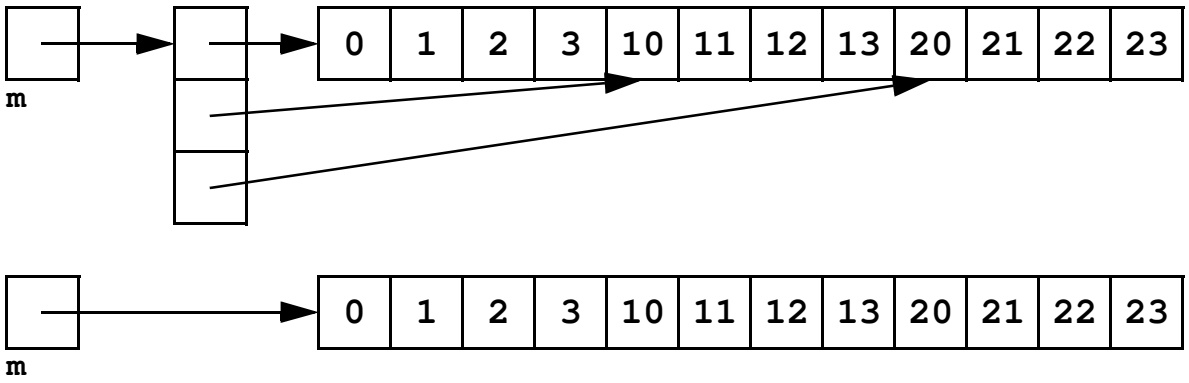
```
int a[5];
for (int i = 0; i < 11; i++)
   a[i] = 0;
```

❋ *Arrays cannot be return types!*

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

❋ *Not all 2D arrays are alike!*

→ What is the layout for all 2D arrays on the stack?
→ What is the layout for 2D arrays on the heap?



❋ *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}};
printIntArray(a,2,4); //size of 2D array is passed via last 2 arguments
```

→ Which of the following are type compatible with `a`?

```
void printIntArray(int a[2][4],int rows,int cols)
void printIntArray(int a[8][4],int rows,int cols)
void printIntArray(int a[][4], int rows,int cols)
void printIntArray(int a[4][8],int rows,int cols)
void printIntArray(int a[][],  int rows,int cols)
void printIntArray(int (*a)[4],int rows,int cols)
void printIntArray(int **a,    int rows,int cols)
```

→ Why is all but the first dimension needed?