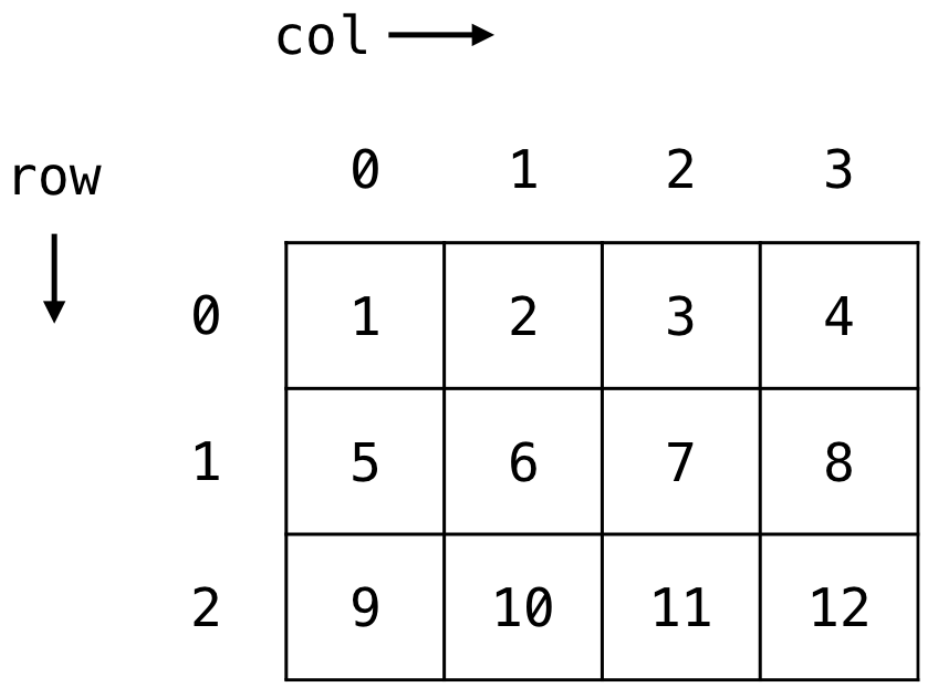
CSE1142 – Review Questions

Sanem Arslan Yılmaz

An Example 2D Array

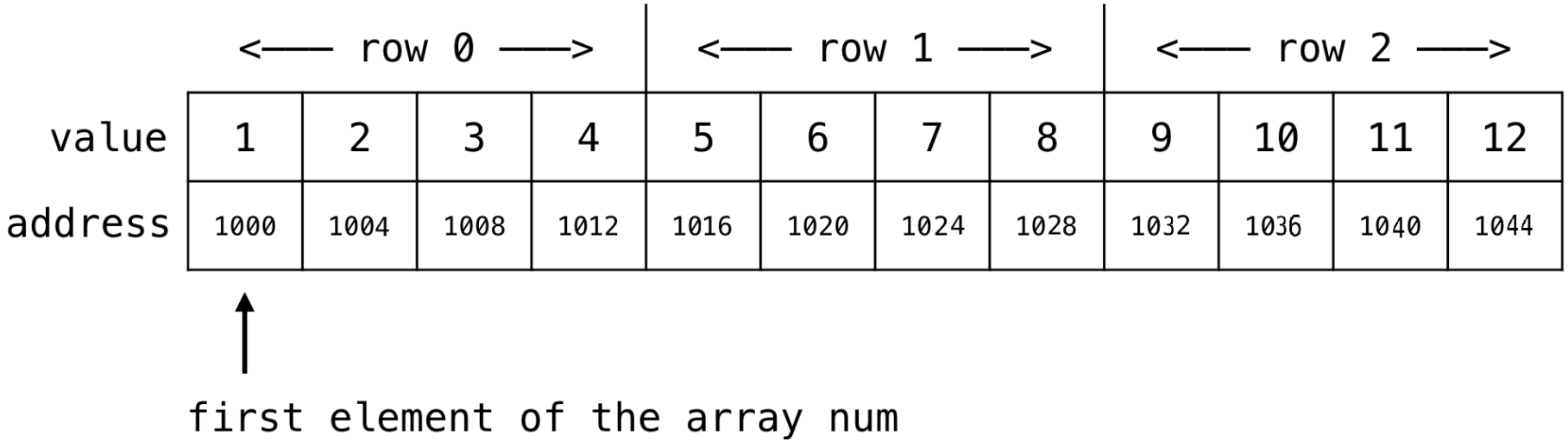
```
int num[3][4] = {
  {1, 2,  3,  4},
  {5, 6,  7,  8},
  {9, 10, 11, 12}
};
```



Memory representation of a 2D array

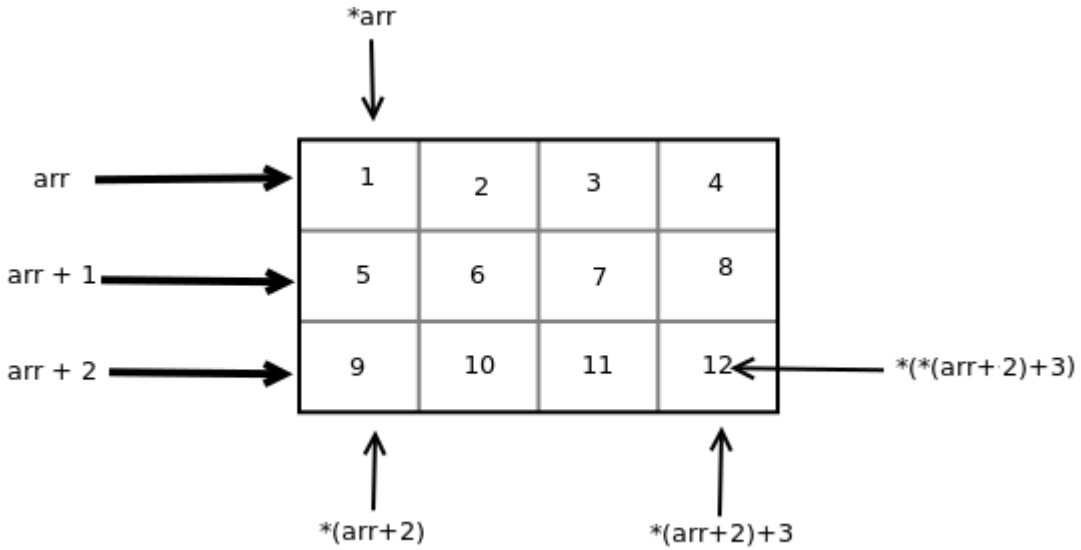
```
int num[3][4] = {
  {1, 2,  3,  4},
  {5, 6,  7,  8},
  {9, 10, 11, 12}
};
```

row-wise memory allocation



In general..

int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };



arr	**Points to 0^{th} 1-D array**
***arr**	**Points to 0^{th} element of 0^{th} 1-D array**
(arr + i)	**Points to i^{th} 1-D array**
***(arr + i)**	**Points to 0^{th} element of i^{th} 1-D array**
***(arr + i) + j)**	**Points to j^{th} element of i^{th} 1-D array**
***(*(arr + i) + j)**	**Reprents the value of j^{th} element of i^{th} 1-D array**

How to dynamically create a 2D array in C ?

- Method 1: Using a single pointer
- Method 2: Using an array of pointers
- Method 3: Using a pointer to a pointer

Method 1: Using a single pointer

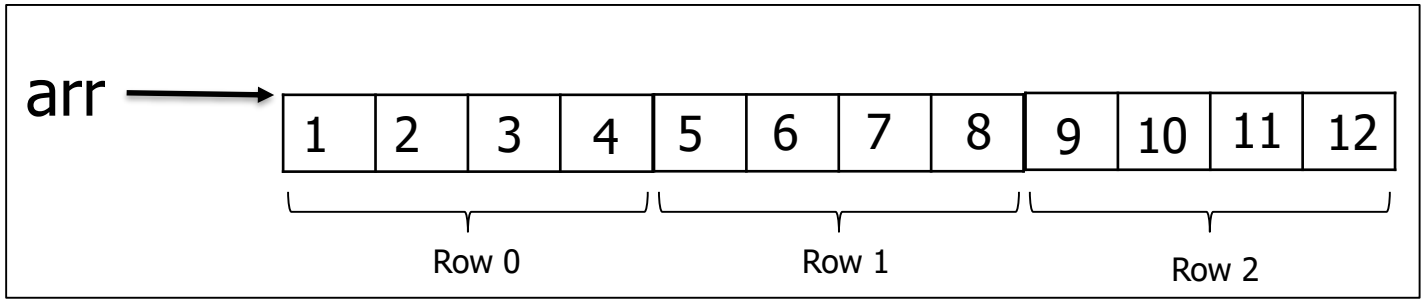
```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int rows = 3, columns = 4;
    int *arr = (int *)malloc(rows * columns * sizeof(int));

    int i, j, count = 0;
    for (i = 0; i < rows; i++)
      for (j = 0; j < columns; j++)
         *(arr + i*columns + j) = ++count;

    for (i = 0; i < rows; i++)
      for (j = 0; j < columns; j++)
         printf("%d ", *(arr + i*columns + j));

   free(arr);
   return 0;
}
```



Method 2: Using an array of pointers

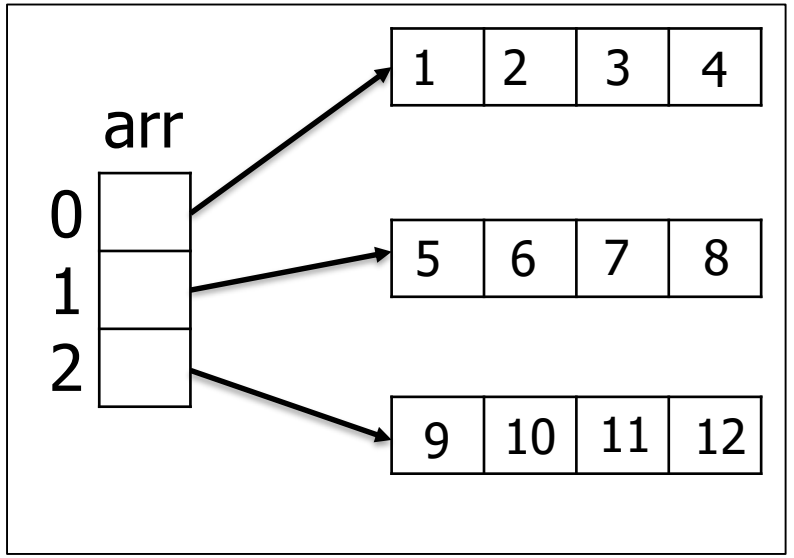
```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int rows = 3, columns = 4, i, j, count;
    int *arr[rows];
    for (i=0; i<rows; i++)
         arr[i] = (int *)malloc(columns * sizeof(int));

    // Note that arr[i][j] is same as *(*(arr+i)+j)
    count = 0;
    for (i = 0; i <  rows; i++)
      for (j = 0; j < columns; j++)
         arr[i][j] = ++count;   // Or *(*(arr+i)+j) = ++count

    for (i = 0; i <  rows; i++)
      for (j = 0; j < columns; j++)
         printf("%d ", arr[i][j]);

    for (i=0; i<rows; i++)  free(arr[i]);
    free(arr);
   return 0;
}
```



Method 3: Using a pointer to a pointer

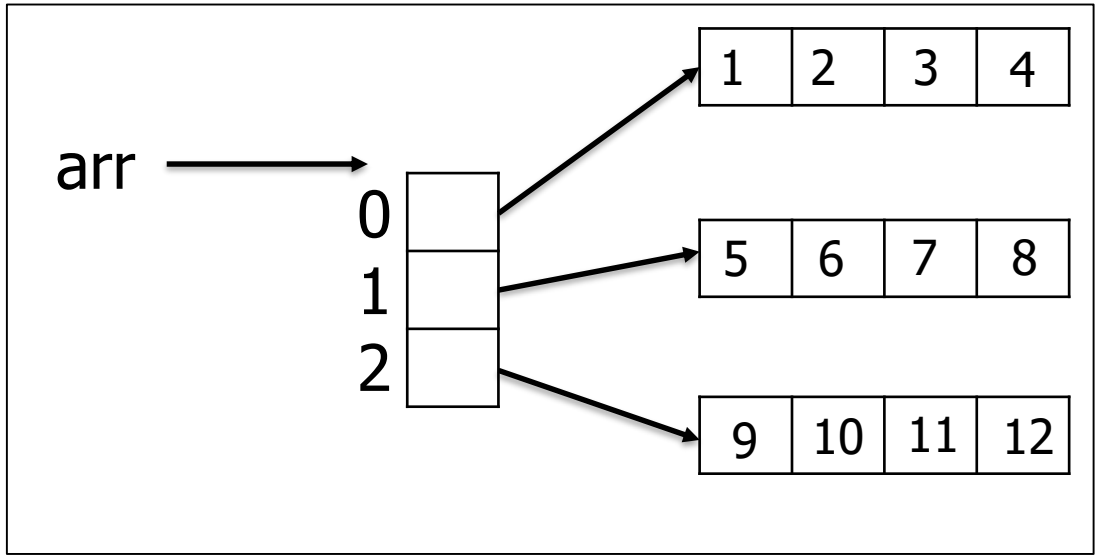
```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int rows = 3, columns = 4, i, j, count;

    int **arr = (int **)malloc(rows * sizeof(int *));
    for (i=0; i<rows; i++)
         arr[i] = (int *)malloc(columns * sizeof(int));

    // Note that arr[i][j] is same as *(*(arr+i)+j)
    count = 0;
    for (i = 0; i < rows; i++)
      for (j = 0; j < columns; j++)
         arr[i][j] = ++count;  // OR *(*(arr+i)+j) = ++count

    for (i = 0; i <  rows; i++)
      for (j = 0; j < columns; j++)
         printf("%d ", arr[i][j]);

   for (i=0; i<rows; i++)  free(arr[i]);
   free(arr);
   return 0;
}
```

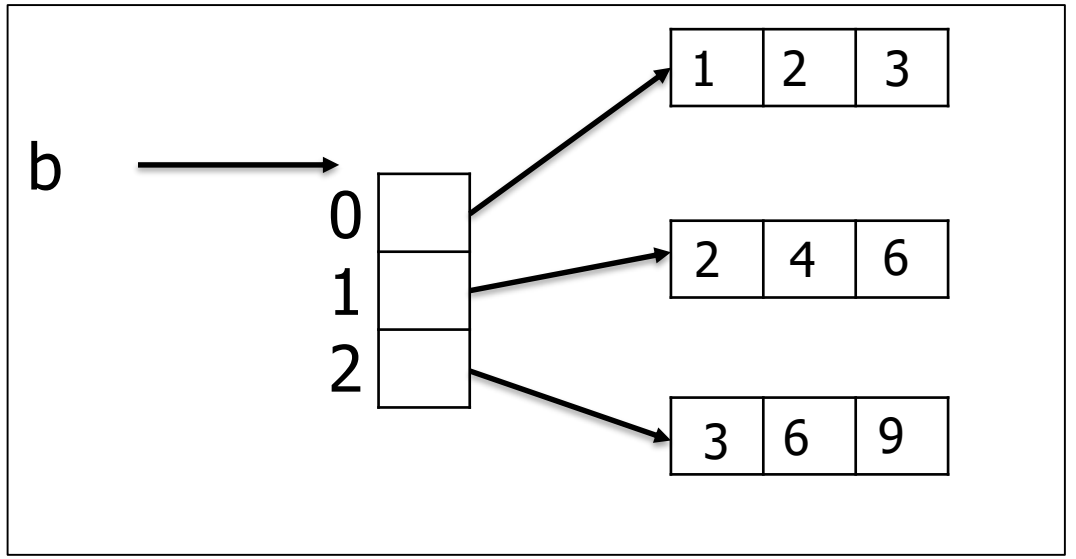


Example – Show the output of the following code

```
#include <stdio.h>
#include <stdlib.h>
int main(){
   int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
   int **b, i, j;
   b = (int **)malloc(3 * sizeof(int *));

   for(i=0;i<3;i++){
        b[i]=(int *)malloc(3 * sizeof(int));
   }

   for(i=0; i<3; i++){
      for(j=0; j<3; j++){
         b[i][j]=(i+1)*(j+1);
      }
   }
   printf("%d %d %d %d\n", *(*arr+4), **arr+4, *(arr[2]-2), **(arr+2));
   printf("%d %d\n", **b+4, **(b+2));
   return 0;
}
```



Output:

5 5 5 7

5 3

Past Exam Question I

- What is the output of the following program?

```
int main(){
        int a=2, *b, c=3, d=5;
        b = &a;

        if((d > 1) && (c = 2)){
                d++;
                *b = *&c;
        }
        printf("%d %d \n", *b, d);
}
```

Past Exam Question II

- What is the output of the following program?

```
int main(){
    int x[5] ={ -4, 9, 12, -3, 7 };
    int a, b;

    int *p = x+1;
    *(p+3)=9;
    a = *(x + 3);
    b = *(--p);
    printf("%d %d \n", a, b);
    printf("%d %d \n", *p, x[4]);

    return 0;
}
```

Past Exam Question III

- What is the output of the following program? Write down the output of the following C program. If a given output is unpredictable, write '**#**' in the corresponding place.

```
int main(){
    int a[3][3]={{0,1},{2,3},{5,6}}, **b, i, j;
    b = (int **)calloc(3, sizeof(int *));

    for(i=0; i<3; i++){
        b[i] = (int *) calloc(3, sizeof(int));
    }
    for(i=0; i<3; i++) {
        for(j=0; j<3; j++){
            b[i][j] = (i+1)*(j+1);
        }
    }
    printf("%d %d %d %d\n", *(*a+5), **a+3, *(*b+2), **b+5);
    printf("%d %d %d %d\n", *(a[2]-3), **(a+1), *(b[2]-2), **(b+1));
    return 0;
}
```

Linked List Examples - 1

- Assume that we have a linked list as the following:

  28 -> 9 -> 2 -> 1 -> 13 -> 7 -> 0 -> 3 -> 4

- Trace the following program segment, for the given lists. Assume that **currentPtr** points to the first node of the list.

```
while (currentPtr->data > 1)
    currentPtr = currentPtr->next->next;
printf ("%d\n", currentPtr->data + currentPtr->next->data);
```

Linked List Examples - 2

- Consider a singly linked list of nodes as the following:

  p -> q -> r -> s -> t -> NULL

- Assume that the following method is called on the head node of the list:

```
void mystery(Node *node) {
    if(node == null)
      return;
    mystery(node->next);
    printf(" %c", node->data);
  }
```

What will the above program print?

Linked List Examples - 3

Assume you are using the `struct listNode` definition given in below:

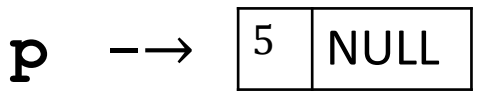
```
struct listNode{
        int data;
        struct listNode *nextPtr;
};

typedef struct listNode Node;
typedef Node *NodePtr;
```

Example: If the code segment is

```
struct listNode *p;
p = malloc(sizeof(struct listNode));
p->data = 5;
p->nextPtr = NULL;
```

You should draw the following:



Linked List Examples – 3.a

Draw a picture that shows the final result of the execution of the code segment.

```
NodePtr p; NodePtr q;
p = malloc(sizeof(Node));
p->data = 5;
p->nextPtr = NULL;
q = malloc(sizeof(Node));
q->data = 10;
q->nextPtr = p;
p->nextPtr = p;
```

Linked List Examples – 3.b

Draw a picture that shows the final result of the execution of the code segment.

```
NodePtr p;
NodePtr q = NULL;
p = malloc(sizeof(Node));
p->data = 5;
p->nextPtr = q;
q = malloc(sizeof(Node));
q->data = 10;
q->nextPtr = p->nextPtr;
```

Queue example

- Consider the following operations on a Queue data structure that stores int values.

```
QueueNodePtr q = NULL;
enqueue(&q, 3);
enqueue(&q, 5);
enqueue(&q, 9);
printf(dequeue(&q ));          // d1
enqueue(&q, 2);
enqueue(&q, 4);
printf(dequeue(&q ));          // d2
printf(dequeue(&q ));          // d3
enqueue(&q, 1);
enqueue(&q, 8);
```

Q1. After the code executes, how many elements would remain in q?

Q2. What value is returned by the last dequeue operation (denoted with a d3 in comments)?

Q3. If we replace the **printf** statements (denoted in comments as d1, d2 and d3) with the statement **enqueue(&q, dequeue(&q));**
q would contain which order of int values after all instructions have executed?

Past Exam Question IV

- Suppose you have three stacks s1, s2, s2 with starting configuration shown on the left, and finishing condition shown on the right. Give a *minimum sequence* of push and pop operations that take you from start to finish.
- For example, to pop the top element of **s1** and push it onto **s2**, you would write **`push(s2, pop(s1));`**