Tutorial 2: Arrays and Dynamic Arrays

ELEC 278: Fundamentals of Information Structures

The learning goals for Tutorial 2 are:

- Understand the differences between static arrays and dynamic arrays.
- Practice using pointers by creating a dynamic 2D array.

Problem 1. What is the difference between a static array and a dynamic array?

Solution. Static arrays are simpler and faster to use when the size is known and fixed, while dynamic arrays offer flexibility to adjust the size at runtime.

- Arrays
  - Fixed Size: The size of an array is fixed at compile time and cannot be changed during runtime.
  - Memory Allocation: Memory for arrays is allocated on the stack.
  - Syntax: Arrays are declared with a fixed size, e.g., int arr[10];. C assigns it a size even if you do not declare a size, e.g., `int arr[] = 1, 2, 3;`.
  - Initialization: Static arrays are initialized at the time of declaration, e.g., `int arr[3] = 1, 2, 3;`.
- Dynamic Arrays
  - Variable Size: The size of a dynamic array can be changed during runtime.
  - Memory Allocation: Memory for dynamic arrays is allocated on the heap using functions like malloc, calloc, or realloc.
  - Syntax: Dynamic arrays are declared using pointers, e.g., `int *arr = (int *)malloc(size * sizeof(int));`.
  - Initialization: Dynamic arrays are typically initialized after memory allocation.

Problem 2. In class, we looked at 1-dimensional arrays. Complete `main()` to create and print a 2-dimensional static array.

```c
#include <stdio.h>

int main() {
    // Define the size of the array
    // Feel free to change the size of the 2D array
    int rows = 3;
    int cols = 4;

    // To do: Declare and initialize the 2D array to hold int values
```

```
    printf("The 2D array is:\n");
    // To do: Print the array

    return 0;
}
```

Solution.

```
#include <stdio.h>

int main() {
    // Define the size of the array
    // Feel free to change the size of the 2D array
    int rows = 3;
    int cols = 4;

    // To do: Declare and initialize the 2D array to hold int values
    int array[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    // To do: Print the array
    printf("The 2D array is:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", array[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Problem 3. We will now create the 2D array and dynamically allocate memory for it on the heap.

Note that for a dynamically allocated 1D array, we create a pointer to a block of memory the size of the array on the heap.

For dynamic 2D arrays, we create a pointer to an array of pointers the size of the rows of the 2D array. This array of (row-)pointers will be on the heap and each of its elements (or pointers) will point to a memory block that is the size of each column.

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

int main() {
    // Feel free to change the size of the 2D array
    int rows = 3, cols = 3;

    // To do: allocate memory for the array of pointers to rows
    int **array = ;
    if (array == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    } // check if memory allocation failed

    // To do: allocate memory for each row
```

```
    for (int i = 0; i < rows; i++) {
        // To do
        if (array[i] == NULL) {
            fprintf(stderr, "Memory allocation failed\n");
            return 1;
        }
    } // check if memory allocation failed

    // to do: initialize the array with some values


    // To do: print the array


    // To do: free the allocated memory


    return 0;
}
```

Solution.

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

int main() {
    // Feel free to change the size of the 2D array
    int rows = 3, cols = 3;

    // Allocate memory for the array of pointers to rows
    int **array = (int **)malloc(rows * sizeof(int *)); // we use double pointers
   because it is a pointer to a pointer
    if (array == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    } // check if memory allocation failed

    // Allocate memory for each row
    for (int i = 0; i < rows; i++) {
        array[i] = (int *)malloc(cols * sizeof(int)); // note that you can also
   use *(array+i) instead of array[i]
        if (array[i] == NULL) {
            fprintf(stderr, "Memory allocation failed\n");
            return 1;
        }
    } // check if memory allocation failed

    // Initialize the array with some values
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            array[i][j] = i * cols + j; // note you can also use *(*(array+i)+j)
   instead of array[i][j]
        }
    }

    // Print the array
    printf("The 2D array is:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", array[i][j]);
```

```
        }
        printf("\n");
    }

    // Free the allocated memory
    for (int i = 0; i < rows; i++) {
        free(array[i]);
    }
    free(array);

    return 0;
}
```