# Day 71/180 Dynamic Memory Allocation of 2D and 3D arrays

1: int arr[3][4],
cout<<arr;
cout<<arr[0]
cout<<arr[0][0]
What will be the output of the program and show us the proper reason behind it.

```
int arr[3][4];  // Declaration of a 2D array with 3 rows and 4 columns

    // Output the address of the array (equivalent to the address of the first
element)
    cout << "Address of the array: " << arr << endl;

    // Output the address of the first row (equivalent to the address of the first
element of the first row)
    cout << "Address of the first row: " << arr[0] << endl;

    // Output the value of the first element in the array
    cout << "Value of the first element: " << arr[0][0] << endl;
```

2: How memory is deallocated in case of dynamically created 3D arrays in c++.

```
#include <iostream>
using namespace std;
```

```
int main() {
    // Example: Dynamically allocate a 3D array with dimensions 2x3x4
    int*** dynamicArray3D = new int**[2];
    for (int i = 0; i < 2; ++i) {
        dynamicArray3D[i] = new int*[3];
        for (int j = 0; j < 3; ++j) {
            dynamicArray3D[i][j] = new int[4];
        }
    }

    // ... Perform operations with the 3D array ...

    // Deallocate the memory
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 3; ++j) {
            delete[] dynamicArray3D[i][j];
        }
        delete[] dynamicArray3D[i];
    }
    delete[] dynamicArray3D;

    return 0;
}
```

Explanation:

Memory Allocation: We use nested new operators to dynamically allocate memory for each dimension of the 3D array.

Memory Deallocation: The memory deallocation is performed in the reverse order of allocation. For each dimension, we use delete[]. The outer loop is responsible for deallocating the memory for the first dimension, and the inner loop is responsible for deallocating the memory for the subsequent dimensions.

Be Mindful of the Dimensions: Adjust the loop limits and array dimensions according to the actual size of your 3D array.

Prevent Memory Leaks: Ensure that you deallocate all the memory you allocated. Not doing so can result in memory leaks.

3: Dynamically create 4D arrays in C++.

```
#include <iostream>
using namespace std;

int main() {
    // Dimensions of the 4D array
    const int dim1 = 2;
    const int dim2 = 3;
    const int dim3 = 4;
    const int dim4 = 5;

    // Dynamically allocate a 4D array
    int**** dynamicArray4D = new int***[dim1];
    for (int i = 0; i < dim1; ++i) {
        dynamicArray4D[i] = new int**[dim2];
        for (int j = 0; j < dim2; ++j) {
            dynamicArray4D[i][j] = new int*[dim3];
            for (int k = 0; k < dim3; ++k) {
                dynamicArray4D[i][j][k] = new int[dim4];
            }
        }
    }

    // The dynamicArray4D is now a 4D array with dimensions dim1 x dim2 x dim3 x
dim4
    // Use dynamicArray4D for your operations

    // Don't forget to deallocate the memory when done
    // Deallocate memory for the innermost dimension
    for (int i = 0; i < dim1; ++i) {
        for (int j = 0; j < dim2; ++j) {
            for (int k = 0; k < dim3; ++k) {
                delete[] dynamicArray4D[i][j][k];
            }
            // Deallocate memory for the third dimension
```

```
            delete[] dynamicArray4D[i][j];
        }
        // Deallocate memory for the second dimension
        delete[] dynamicArray4D[i];
    }
    // Deallocate memory for the first dimension
    delete[] dynamicArray4D;

    return 0;
}
```