Dynamic Memory Allocation

Stack Memory Allocation

The memory is allocated on the function call stack. The memory gets deallocated as soon as the function call gets over. Deallocation is handled by the compiler.

Heap Memory Allocation

Allocation takes place on the pile of memory space available to programmers to allocated and de-allocate. The programmer has to handle the deallocation.

NOTE: It is different from the heap data structure.

Delete Operator

To de-allocate a memory p, we pass its address to the delete() function.

```
//to de-allocate a memory,
//pointed by pointer 'p'
delete(p)
```

New Operator:

New operator is used to allocate a block of memory of the given data type.

```
//Syntax
//myPointer = new <data_type>[size];
int *p = new int[10];
```

Dangling Pointer

If the memory location pointed by the pointer gets freed/ deallocated, then the pointer is known as the Dangling Pointer.

Practise Question:

1. Declare a 2D array Dynamically.
2. Declare a 3D array Dynamically.
3. MCQs on Dynamic Memory Allocation.

Dynamic memory

A dynamic array is quite similar to a regular array, but its size is modifiable during program runtime. Dynamic Array elements occupy a contiguous block of memory. Once an array has been created, its size cannot be changed. However, a dynamic array is different. A dynamic array can expand its size even after it has been filled. During the creation of an array, it is allocated a predetermined amount of memory. This is not the case with a dynamic array as it grows its memory size by a certain factor when there is a need.

new > C++ 138_DynamicMemoryAllocation_ex1.cpp > main()

```cpp
// DYNAMIC MEMORY ALLOCATION

#include <iostream>
// #include <new>          //header file for using (nothrow) exception
using namespace std;

// if (nothrow) is not used and allocation fails, ie. user enters a number too large for the computer to handle, an exception is thrown
// if (nothrow) is used and allocation fails, the failure can be detected by checking if pointer is a null pointer:

int main ()
{
    int a;
    cin>>a;
    int* q = new int(a);    //one way
    float *r = new float;   //other way
    *r = 84.98;
    cout<<"Pointer q with dynamic memory allocated: "<<*q<<endl;
    cout<<"Pointer r with dynamic memory allocated: "<<*r<<endl;
    delete(q);
    delete(r);


    int n;
    int * p;
    cout << "How many numbers would you like to type? ";
    cin >> n;
    p = new int[n];     //For 1D array

    //Since this memory allocation of 1D array is dependent on the user input n which is size, this is the use.

    // But for 1D array its not that useful as function doesn't need size, void func(int arr[]) is also correct.
    // This is more useful in 2D array.

    for (int i=0; i<n; i++){
        cout << "Enter number: ";
        cin >> p[i];
    }

    cout << "You have entered: ";
    for (int i=0; i<n; i++)
        cout << p[i] << ", ";

    delete[] p;     //for array
    return 0;
}
```

```
// DYNAMIC MEMORY ALLOCATION
// C++ program to dynamically allocate the memory for 2D array in C++ using new operator.
```

new > C++ 139_DynamicMemoryAllocation_ex2_2D-Array.cpp > main()

```cpp
// Method 1: using a single pointer - In this method, a memory block of size M*N is allocated
//  and then the memory blocks are accessed using pointer arithmetic

#include <iostream>
using namespace std;

void print2Darray(int* arr, int m, int n){

    // Traverse the 2D array
    for (int i = 0; i < m; i++){
        for (int j = 0; j < n; j++){
            // Print values of the memory block
            cout<< *(arr + i * n + j)<<" ";
        }
        cout<<endl;
    }
}

int main(){

    // Dimensions of the 2D array
    int m, n, val;
    cout<<"Enter the size of 2D array: ";
    cin>>m>>n;

    // Declare a memory block of size m*n, which is a pointer array of type int
    int* arr = new int[m * n];
    // Its actually a 1D array of size m*n, but here we want 2D array, so we will use it like 2D.

    // Traverse the 2D array
    for (int i = 0; i < m; i++){
        for (int j = 0; j < n; j++){
            val = 0;
            cout<<"Enter element "<<i<<j<<": ";
            cin>>val;

            // Assign values to the memory block
            *(arr + i * n + j) = val;
        }
    }

    print2Darray(arr, m, n);

    //Delete the array created
    delete[] arr;

    return 0;
}
```

new > C++ 140_DynamicMemoryAllocation_ex3_2D-Array.cpp > ...

```cpp
// DYNAMIC MEMORY ALLOCATION
// C++ program to dynamically allocate the memory for 3D array in C++ using new operator


// Method 2: using an array of pointer: Here an array of pointers is created and then to each memory block.

#include <iostream>
using namespace std;

void print2Darray(int** a, int m, int n){

    // Traverse the 2D array
    for (int i = 0; i < m; i++){
        for (int j = 0; j < n; j++){
            // Print the values of memory blocks created
            cout<<a[i][j]<<" ";
        }
        cout<<endl;
    }
}

int main()
{
    // Dimensions of the 2D array
    int m, n, val;
    cout<<"Enter the size of 2D array: ";
    cin>>m>>n;

    // Declare memory block of size M
    int** a = new int*[m];

    for (int i = 0; i < m; i++){
        // Declare a memory block of size n
        a[i] = new int[n];
    }

    // Traverse the 2D array
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            val=0;
            cout<<"Enter element "<<i<<j<<": ";
            cin>>val;

            // Assign values to the memory blocks created
            a[i][j] = val;
        }
    }

    print2Darray(a, m, n);

    //Delete the array created
    for(int i=0;i<m;i++)
        delete [] a[i];     //To delete the inner arrays

    delete [] a;    //To delete the outer array which contained the pointers of all the inner arrays

    return 0;
}
```