

**LAB 15**

# Dynamic Memory Allocation:

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. 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 Keyword:**

In C++, we can create a dynamic array using the new keyword. The number of items to be allocated is specified within a pair of square brackets. The type name should precede this. The requested number of items will be allocated.

**Delete Keyword:**

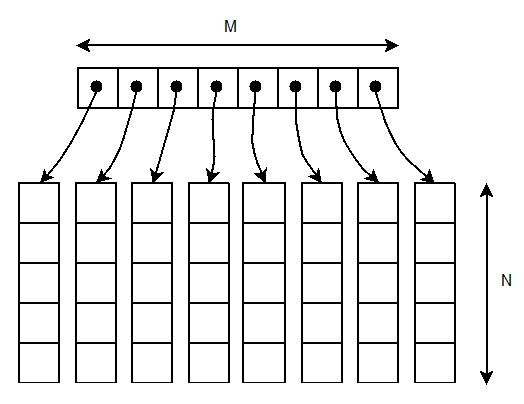
A dynamic array should be deleted from the computer memory once its purpose is fulfilled. The delete statement can help you accomplish this. The released memory space can then be used to hold another set of data. However, even if you do not delete the dynamic array from the computer memory, it will be deleted automatically once the program terminates.

# 1D Dynamic Memory Allocation:

|  |
| --- |
| #include <iostream>  #define N 10  // Dynamically allocate memory for 1D Array in C++  int main()  { // dynamically allocate memory of size `N`  int\* A = new int[N];  // assign values to the allocated memory  for (int i = 0; i < N; i++) {  A[i] = i + 1; }  // print the 1D array  for (int i = 0; i < N; i++) {  std::cout << A[i] << " "; // or \*(A + i) }  // deallocate memory  delete[] A; } |

# 2D Dynamic Memory Allocation:

We can dynamically create an array of pointers of size M and then dynamically allocate memory of size N for each row, as shown below:

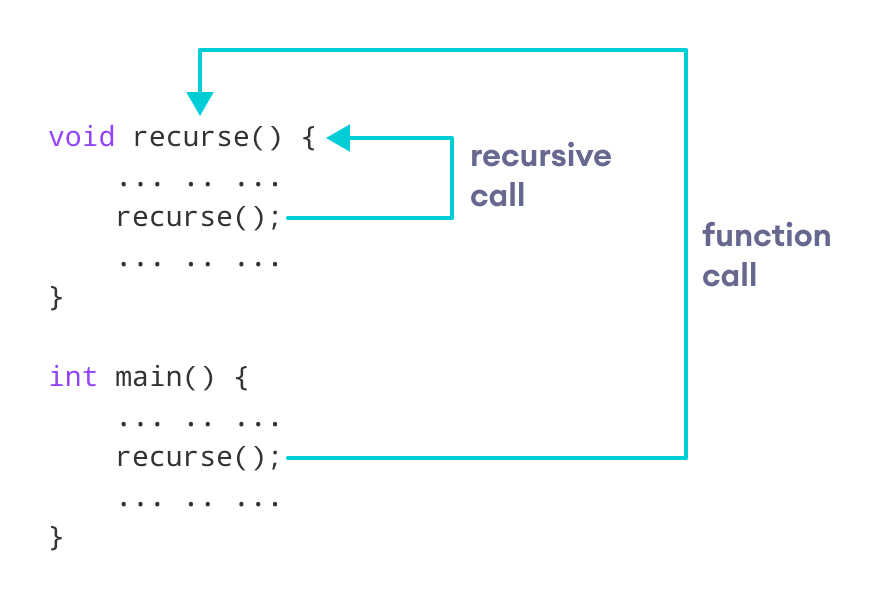


|  |
| --- |
| #include <iostream>  // `M × N` matrix  #define M 4  #define N 5  // Dynamic Memory Allocation in C++ for 2D Array  int main(){  // dynamically create an array of pointers of size `M`  int\*\* A = new int\*[M];  // dynamically allocate memory of size `N` for each row  for (int i = 0; i < M; i++) {  A[i] = new int[N]; }  // assign values to the allocated memory  for (int i = 0; i < M; i++){  for (int j = 0; j < N; j++) {  A[i][j] = rand() % 100;  } }  // print the 2D array  for (int i = 0; i < M; i++) {  for (int j = 0; j < N; j++) {  std::cout << A[i][j] << " "; }  std::cout << std::endl; }  // deallocate memory using the delete operator  for (int i = 0; i < M; i++) {  delete[] A[i]; }  delete[] A; } |

# Recursion:

A function that calls itself is known as a recursive function. And, this technique is known as recursion.

# Working of Recursion in C++:



The recursion continues until some condition is met.

To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and the other doesn't.

# Example 1: Factorial of a Number Using Recursion

// Factorial of n = 1\*2\*3\*...\*n

#include <iostream>

using namespace std;

int factorial(int);

int main() {

int n, result;

cout << "Enter a non-negative number: ";

cin >> n;

result = factorial(n);

cout << "Factorial of " << n << " = " << result;

return 0;

}

int factorial(int n) {

if (n > 1) {

return n \* factorial(n - 1);

} else {

return 1;

}

}

# Output

Enter a non-negative number: 4

Factorial of 4 = 24

# Lab Task

Write a program that creates a two-dimensional dynamic array initialized with test data. Use any data type you wish. The program should have the following functions:

* **GetHighestInRow**. This function should accept a two-dimensional array as its first argument and an integer as its second argument. The second argument should be the subscript of a row in the array. The function should return the highest value in the specified row of the array.
* **GetLowestInRow**. This function should accept a two-dimensional array as its first argument and an integer as its second argument. The second argument should be the subscript of a row in the array. The function should return the lowest value in the specified row of the array.
* Transpose: Find Transpose of array.
* **LeftDiagonalTotal**: Calculates total/sum of the values in the left Diagonal of array.
* **RightDiagonalTotal**: Calculates total/sum of the values in the right Diagonal of array.
* **Multiply**: Take another 3 X 3 array as input from user and multiply both.

**Note**: Make all code separately and then merge them all in a menu. Use switch statement for menu.

|  |
| --- |
| **Submission Instructions:**   1. Save all .cpp files and screenshots with task number   e.g. Task01.cpp   1. Now create a new folder with name ROLLNO\_LAB015 e.g. i22XXXX\_LAB15 2. Move all your .cpp files to this newly created directory and compress it into .zip file. |