Using Single Pointer

In this approach we simply allocate memory of size M\*N dynamically and assign it to pointer. Even though the memory is linearly allocated, we can use pointer arithmetic to index 2D array.

#include <stdio.h>

#include <stdlib.h>

// M x N matrix

#define M 4

#define N 5

// Dynamically Allocate Memory for 2D Array

int main()

{

    // dynamically allocate memory of size M\*N

    int\* A = (int\*)malloc(M \* N \* sizeof(int));

    if (A == NULL)

    {

        fprintf(stderr, "Out of memory");

        exit(0);

    }

    // assign values to allocated memory

    for (int r = 0; r < M; r++)

        for (int c = 0; c < N; c++)

            \*(A + r\*N + c) = rand() % 100;

    // print the 2D array

    for (int r = 0; r < M; r++)

    {

        for (int c = 0; c < N; c++)

            printf("%d ", (A + r\*N)[c]);    // or \*(A + r\*N + c)

        printf("\n");

    }

    // deallocate memory

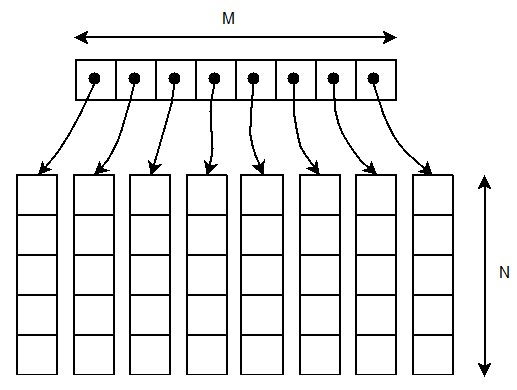
    free(A);

    return 0;

}

##### Using Array of Pointers

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



#include <stdio.h>

#include <stdlib.h>

// M x N matrix

#define M 4

#define N 5

// Dynamically Allocate Memory for 2D Array

int main()

{

    // dynamically create array of pointers of size M

    int \*\*A = (int \*\*)malloc(M \* sizeof(int \*));        // or int\* A[M];

    if (A == NULL)

    {

        fprintf(stderr, "Out of memory");

        exit(0);

    }

    // dynamically allocate memory of size N for each row

    for (int r = 0; r < M; r++)

    {

        A[r] = (int \*)malloc(N \* sizeof(int));

        if (A[r] == NULL)

        {

            fprintf(stderr, "Out of memory");

            exit(0);

        }

    }

    // assign values to allocated memory

    for (int r = 0; r < M; r++)

        for (int c = 0; c < N; c++)

            A[r][c] = rand() % 100;    // or \*(A[r] + c) or \*(\*(A + r) + c)

    // print the 2D array

    for (int r = 0; r < M; r++)

    {

        for (int c = 0; c < N; c++)

            printf("%d ", A[r][c]);    // or \*(A[r] + c) or \*(\*(A + r) + c)

        printf("\n");

    }

    // deallocate memory

    for (int r = 0; r < M; r++)

        free(A[r]);

    free(A);

    return 0;

}

##### Using Double Pointer

We start by creating array of pointers of size M as seen in 2nd method. Then we dynamically allocate memory of size M\*N and let \*A point to it. Finally, we position allocated memory across M pointers.

#include <stdio.h>

#include <stdlib.h>

// M x N matrix

#define M 4

#define N 5

// Dynamically Allocate Memory for 2D Array

int main()

{

    // dynamically create array of pointers of size M

    int \*\*A = (int \*\*)malloc(M \* sizeof(int \*));        // or int\* A[M];

    if (A == NULL)

    {

        fprintf(stderr, "out of memory\n");

        exit(0);

    }

    // dynamically allocate memory of size M\*N and let \*A point to it

    \*A = (int \*)malloc(sizeof(int) \* M \* N);

    // position allocated memory across M pointers

    for (int r = 0; r < M; r++)

        A[r] = (\*A + N\*r);

    // assign values to allocated memory

    for (int r = 0; r < M; r++)

        for (int c = 0; c < N; c++)

            A[r][c] = rand() % 100;    // or \*(A[r] + c) or \*(\*(A + r) + c)

    // print the 2D array

    for (int r = 0; r < M; r++)

    {

        for (int c = 0; c < N; c++)

            printf("%d ", A[r][c]);    // or \*(A[r] + c) or \*(\*(A + r) + c)

        printf("\n");

    }

    // deallocate memory

    free(\*A);

    free(A);

    return 0;

}