

How to dynamically allocate a 2D array in C?

Following are different ways to create a 2D array on heap (or dynamically allocate a 2D array).

In the following examples, we have considered 'r' as number of rows, 'c' as number of columns and we created a 2D array with r = 3, c = 4 and following values

```
1  2  3  4
5  6  7  8
9 10 11 12
```

1) Using a single pointer:

A simple way is to allocate memory block of size r*c and access elements using simple pointer arithmetic.

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

int main()
{
    int r = 3, c = 4;
    int *arr = (int *)malloc(r * c * sizeof(int));

    int i, j, count = 0;
    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            *(arr + i*c + j) = ++count;





    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            printf("%d ", *(arr + i*c + j));

    /* Code for further processing and free the
       dynamically allocated memory */

    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12
```





 `#include <stdio.h>`
 `#include <stdlib.h>`
 `int main()`
 `{`
 `int r = 3, c = 4, i, j, count;`
 `int *arr[r];`
 `for (i=0; i<r; i++)`
 `arr[i] = (int *)malloc(c * sizeof(int));`
 `// Note that arr[i][j] is same as (*(arr+i)+j)`
 `count = 0;`
 `for (i = 0; i < r; i++)`
 `for (j = 0; j < c; j++)`
 `arr[i][j] = ++count; // Or (*(arr+i)+j) = ++count`
 `for (i = 0; i < r; i++)`
 `for (j = 0; j < c; j++)`
 `printf("%d ", arr[i][j]);`
 `/* Code for further processing and free the`
 `dynamically allocated memory */`
 `return 0;`
`}`

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12
```

3) Using pointer to a pointer

We can create an array of pointers also dynamically using a double pointer. Once we have an array pointers allocated dynamically, we can dynamically allocate memory and for every row like method 2.

 `#include <stdio.h>`
 `#include <stdlib.h>`
 `int main()`
 `{`
 `int r = 3, c = 4, i, j, count;`
 `int **arr = (int **)malloc(r * sizeof(int *));`
 `for (i=0; i<r; i++)`
 `arr[i] = (int *)malloc(c * sizeof(int));`

```

// Note that arr[i][j] is same as (*(arr+i)+j)
count = 0;
for (i = 0; i < r; i++)
    for (j = 0; j < c; j++)
        arr[i][j] = ++count; // OR (*(arr+i)+j) = ++count

for (i = 0; i < r; i++)
    for (j = 0; j < c; j++)
        printf("%d ", arr[i][j]);

/* Code for further processing and free the
   dynamically allocated memory */

return 0;
}

```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12
```

4) Using double pointer and one malloc call

```

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

int main()
{
    int r=3, c=4, len=0;
    int *ptr, **arr;
    int count = 0,i,j;

    len = sizeof(int *) * r + sizeof(int) * c * r;
    arr = (int **)malloc(len);

    // ptr is now pointing to the first element in of 2D array
    ptr = arr + r;

    // for loop to point rows pointer to appropriate location in 2D
    for(i = 0; i < r; i++)
        arr[i] = (ptr + c * i);

    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            arr[i][j] = ++count; // OR (*(arr+i)+j) = ++count

    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            printf("%d ", arr[i][j]);

    return 0;
}

```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12
```

A one dimensional array can be easily passed as a pointer, but syntax for passing a 2D array to a function can be difficult to remember. One important thing for passing multidimensional arrays is, first array dimension does not have to be specified. The second (and any subsequent) dimensions must be given

1) When both dimensions are available globally (either as a macro or as a global constant).

```
#include <stdio.h>
const int M = 3;
const int N = 3;

void print(int arr[M][N])
{
    int i, j;
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            printf("%d ", arr[i][j]);
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr);
    return 0;
}
```

1 2 3 4 5 6 7 8 9

2) When only second dimension is available globally (either as a macro or as a global constant).

```
#include <stdio.h>
const int N = 3;

void print(int arr[][N], int m)
{
    int i, j;
    for (i = 0; i < m; i++)
        for (j = 0; j < N; j++)
            printf("%d ", arr[i][j]);
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr, 3);
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9

The above method is fine if second dimension is fixed and is not user specified. The following methods handle cases when second dimension can also change.

3) If compiler is C99 compatible

From C99, C language supports variable sized arrays to be passed simply by specifying the variable dimensions (See [this](#) for an example run)

```
// The following program works only if your compiler is C99 compatible
#include <stdio.h>

// n must be passed before the 2D array
void print(int m, int n, int arr[][n])
{
    int i, j;
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            printf("%d ", arr[i][j]);
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int m = 3, n = 3;
    print(m, n, arr);
    return 0;
}
```

Output on a C99 compatible compiler:

```
1 2 3 4 5 6 7 8 9
```

If compiler is not C99 compatible, then we can use one of the following methods to pass a variable sized 2D array.

4) Using a single pointer

In this method, we must typecast the 2D array when passing to function.

```
#include <stdio.h>
void print(int *arr, int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            printf("%d ", *((arr+i*n) + j));
}
```

```
int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int m = 3, n = 3;

    // We can also use "print(&arr[0][0], m, n);"
    print((int *)arr, m, n);
    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9
```