# Dynamic Two-dimensioned Arrays in C

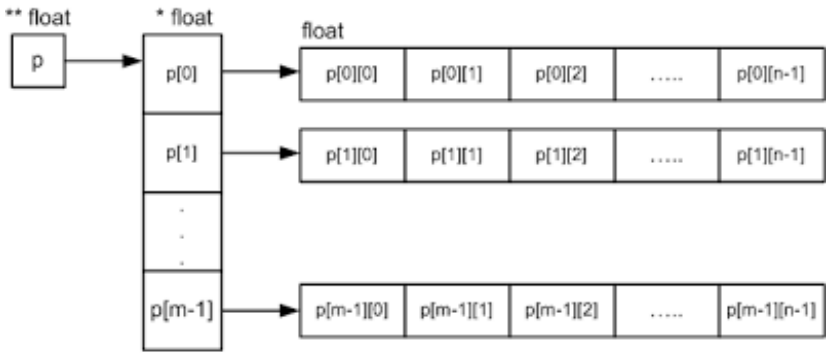
annigeri.in/2011/11/dynamic-two-dimensioned-arrays-in-c.html

Extending the concept discussed in dynamic one-dimensioned arrays to two-dimensioned arrays.

Let us now take the concept discussed in Static and Dynamic One-dimensioned Arrays in C forward. Here is the outline of what we intend to do:

1. Define 'p', a pointer to pointer to float, as the name of the 2D array.
2. To pointer 'p', let us allocate an array of 'm' elements of type 'pointer to float', Here, 'm' is the number of rows we wish the 2D array to have.
3. To each element p[i] allocated above, let us allocate an array of 'n' elements of type 'float'. Each such block will store one full row of the 2D array. Here, 'n' is the number of columns in each row of the array.

This is illustrated in the diagram below:




DYNAMIC ARRAY (2D) USING POINTER TO ARRAY OF POINTERS
- Pointer 'p' points to an array of pointers to float
- Each element of this array points to a block of memory sufficient to store one row
- The way memory is allocated, it is possible that the memory allocated to the different rows may not be contiguous

Note the following points:

1. Memory allocated to the rows of the array may not be contiguous. This deviates from the way memory is allocated to a static array in C. In a future iteration of this post, I intend to discuss how to remedy this shortcoming.
2. This dynamic 2D array requires more memory as compared to a static 2D array. The memory allocated to the rows (**sizeof(float *) * m**) and the memory allocated to the pointer 'p' itself are extra, as compared to a static 2D array in C.
3. Row and column indexing still starts with zero. I will discuss how to remedy this in a future post, but it follows the same concept that is discussed in the previous post C/C++ Arrays with Arbitrary Start Index .
4. The sequence of memory allocation is, allocate memory to pointer to row pointers first and then to the row pointers. While deallocating memory, deallocate memory allocated to the row pointers first, and then deallocate memory allocated to pointer to row pointers.
5. It would be possible to allocate a different number of columns to different rows, but it would deviate from the traditional definition of an array. However, this would be a good way to define sparse matrices (which contains only a few elements that are non-zero and remain non-zero during all subsequent operations on them).

Let us illustrate this with a C program:

```
#include
#include

int main() {
  float **p;
  int m, n, i, j;

  printf("Enter number of rows and columns: ");
  scanf("%d%d", &m, &n);

  /* Allocate memory */
  p = (float *) malloc(sizeof(float *) * m); /* Row pointers */
  for(i = 0; i < m; i++) {
    p[i] = (float) malloc(sizeof(float) * n); /* Rows */
  }

  /* Assign values to array elements and print them */
  for(i = 0; i < m; i++) {
    for(j = 0; j < n; j++) {
      p[i][j] = (i * 10) + (j + 1);
      printf("%6.2f ", p[i][j]);
    }
    printf("\n");
  }

  /* Deallocate memory */
  for(i = 0; i < m; i++) {
    free(p[i]); /* Rows */
  }
  free(p); /* Row pointers */
}
```

I have dispensed with error checking only for the purpose of brevity, however, it is a bad practice.