Multi-Dimensional Arrays

CS10003: PROGRAMMING AND DATA STRUCTURES

Two Dimensional Arrays

We have seen that an array variable can store a list of values.

Many applications require us to store a table of values.

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Student 1	75	82	90	65	76
Student 2	68	75	80	70	72
Student 3	88	74	85	76	80
Student 4	50	65	68	40	70

The table contains a total of 20 values, five in each line.

- The table can be regarded as a matrix consisting of four rows and five columns.

C allows us to define such tables of items by using two-dimensional arrays.

Declaring 2-D Arrays

General form:

```
type   array_name [row_size][column_size];
```

Examples:

```
int  marks[4][5];
float  sales[12][25];
double  matrix[100][100];
```

Accessing Elements of a 2-D Array

Similar to that for 1-D array, but use two indices.

- First indicates row, second indicates column.
- Both the indices should be expressions which evaluate to integer values.

Examples:

```
x[m][n] = 0;
c[i][k] += a[i][j] * b[j][k];
a = sqrt (a[j*3][k]);
```

How is a 2-D array is stored in memory?

Starting from a given memory location, the elements are stored row-wise in consecutive memory locations.

- x: starting address of the array in memory
- c: number of columns
- k: number of bytes allocated per array element

- a[i][j] is allocated memory location at address $x + (i * c + j) * k$

a[0]0] a[0][1] a[0]2] a[0][3] a[1][0] a[1][1] a[1][2] a[1][3] a[2][0] a[2][1] a[2][2] a[2][3]

Row 0 Row 1 Row 2

How to read the elements of a 2-D array?

By reading them one element at a time

```
for  (i=0; i<nrow; i++)
   for  (j=0; j<ncol; j++)
      scanf  ("%f", &a[i][j]);
```

- The ampersand (&) is necessary.
- The elements can be entered all in one line or in different lines.

We can also initialize a 2-D array at the time of declaration:

```
int a[MAX_ROWS][MAX_COLS] = { {1,2,3},  {4,5,6},  {7,8,9}  };
```

How to print the elements of a 2-D array?

By printing them one element at a time.

```
for  (i=0; i<nrow; i++)
   for  (j=0; j<ncol; j++) printf  (“%f  ”, a[i][j]);
```

- This will print all of them in one line

```
for  (i=0; i<nrow; i++) {
        for  (j=0; j<ncol; j++) printf  (“%f  ”, a[i][j]);
          printf(“\n”);
}
```

- The elements are printed with one row in each line.

Example: Matrix Addition

```
#include  <stdio.h>
int main( ) {

          int  a[100][100], b[100][100], c[100][100], p, q, m, n;
          scanf ("%d %d", &m, &n);

          for  (p=0; p<m; p++)
             for  (q=0; q<n; q++)  scanf ("%d", &a[p][q]);
          for  (p=0; p<m; p++)
             for  (q=0; q<n; q++)  scanf ("%d", &b[p][q]);

          for  (p=0; p<m; p++)
             for  (q=0; q<n; q++)  c[p][q] = a[p][q] + b[p][q];

          for  (p=0; p<m; p++)  {
            for  (q=0; q<n; q++) printf("%f   ", c[p][q]);
            printf("\n");
          }
}
```

Passing 2-D arrays to functions

Similar to that for 1-D arrays.

- The array contents are not copied into the function.
- Rather, the address of the first element is passed.

For calculating the address of an element in a 2-D array, we need:

- The starting address of the array in memory.
- Number of bytes per element.
- Number of columns in the array (that is, the size of each row).

The above three pieces of information must be known to the function.

Example:

```
#include  <stdio.h>
int main( ) {
    int  a[15][25], b[15][25], c[15][25];
    int m, n;
    scanf ("%d %d", &m, &n);

    for  (p=0; p<m; p++)
        for  (q=0; q<n; q++)  scanf ("%d", &a[p][q]);
    for  (p=0; p<m; p++)
        for  (q=0; q<n; q++)  scanf ("%d", &b[p][q]);

    add( a, b, m, n, c);
    for  (p=0; p<m; p++)  {
       for  (q=0; q<n; q++) printf("%f   ", c[p][q]);
       printf("\n");
    }
}
```

```
void add( int x[ ][25], int y[ ][25], int m, int n, int z[ ][25] )
{
        int p, q;
        for  (p=0; p<m; p++)
        for  (q=0; q<n; q++)  z[p]q] = x[p][q] + y[p][q];
}
```

Note that the number of columns has to be fixed in the function definition

- There is no difference between
  void add(int x[ ][25], …) and
  void add(int x[15][25], …)
- Specifying both dimensions is not necessary, but not a mistake

2D Arrays and Pointers

```
#define COL 5
int y[5][COL];
int x = *(y + 2*COL + 2);
```

This is not correct !!

```
#define COL 5
int y[5][COL];
int x = *((int *)y + 2*COL + 2);
```

This is correct!!

Data Type of 2-D Array

```
#include <stdio.h>
int main( )
{
    int matrix[4][3] = { {1, 2, 3},
                          {4, 5, 6},
                          {7, 8, 9},
                          {10, 11, 12}};
    int** pmat = (int **)matrix;

    printf("&matrix[0][0] = %u\n", &matrix[0][0]);
    printf("&pmat[0][0] = %u\n", &pmat[0][0]);
    return 0;
}
```

```
OUTPUT
=======
&matrix[0][0] = 1245016
&pmat[0][0] = 1
```

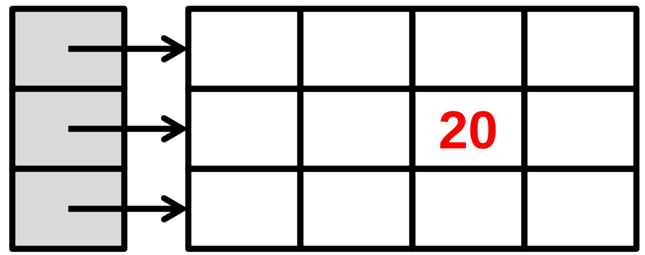
Why are they different?

Dynamic Allocation of 2D array

```
#include <stdio.h>
#include <stdlib.h>
#define ROW 3
#define COL 4
int main()
{
    int  count;
    int **arr = (int **) malloc(ROW * sizeof(int *));
    for (i=0; i<ROW; i++) arr[i] = (int *)malloc(COLUMN * sizeof(int));
    arr[2][3] = 20;  // Note that the style of accessing is the same
}
```

This creates a list of pointers

This creates each row

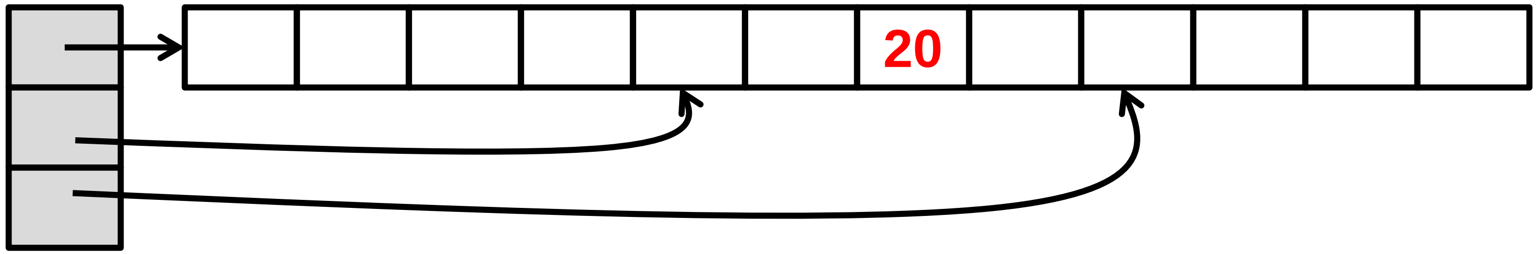


We could use one malloc( ) call for all the rows

```
#include<stdio.h>
#include<stdlib.h>
#define ROW 3
#define COL 4
int main( )
{
    int **arr;
    int i, j;

    arr  = (int **)malloc(sizeof(int *) * ROW);
    arr[0] = (int *)malloc(sizeof(int) * COL * ROW);

    for(i = 0; i < ROW; i++) arr[i] = (*arr + COL * i);
    arr[2][3] = 20;
}
```



Practice Problems

1. Write a function that takes a n x n square matrix A as parameter (n < 100) and returns 1 if A is an upper-triangular matrix, 0 otherwise.
2. Repeat 1 to check for lower-triangular matrix, diagonal matrix, identity matrix
3. Write a function that takes as parameter an m x n matrix A (m, n < 100) and returns the transpose of A (modifies in A only).
4. Consider a n x n matrix containing only 0 or 1. Write a function that takes such a matrix and returns 1 if the number of 1's in each row are the same and the number of 1's in each column are the same; it returns 0 otherwise
5. Write a function that reads in an m x n matrix A and an n x p matrix B, and returns the product of A and B in another matrix C. Pass appropriate parameters.

For each of the above, also write a main function that reads the matrices, calls the function, and prints the results (a message, the transposed matrix etc.)