Arrays- V

CS10001: Programming & Data Structures

Sudeshna Sarkar

Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur

Two-dimensional Arrays

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

2-dimensional Arrays

- **It is convenient to think of a 2-d array as a rectangular collection of elements .**
- **int a[3][5]**

	col0	col1	col2	col3	col4
row0	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
row1	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
row2	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
row3	a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

Contd.

- 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.
- The computer memory is an 1-dimensional sequence of bytes.
- A 2-d array is stored by the C compiler in row major order.

Row major memory mapping

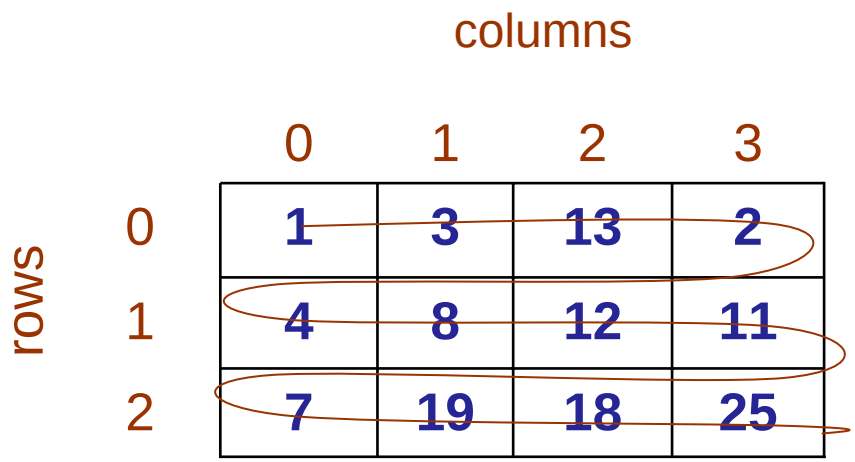
columns

rows

	0	1	2	3
0	1	3	13	2
1	4	8	12	11
2	7	19	18	25

M[0][0]	1
M[0][1]	3
M[0][2]	13
M[0][3]	2
M[1][0]	4
M[1][1]	8
M[1][2]	12
M[1][3]	11
M[2][0]	7
M[2][1]	19
M[2][2]	18
M[2][3]	25

Row major memory mapping

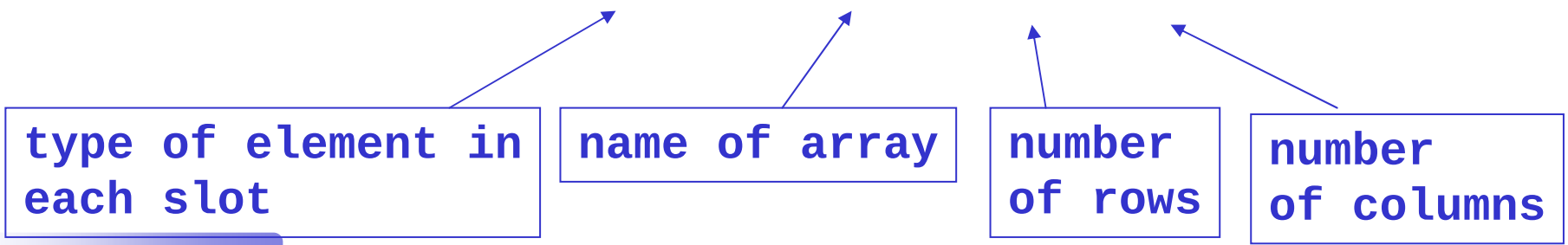


M[0][0]	1
M[0][1]	3
M[0][2]	13
M[0][3]	2
M[1][0]	4
M[1][1]	8
M[1][2]	12
M[1][3]	11
M[2][0]	7
M[2][1]	19
M[2][2]	18
M[2][3]	25

Declaring 2D Arrays

'J'	'o'	'h'	'n'
'M'	'a'	'r'	'y'
'I'	'v'	'a'	'n'

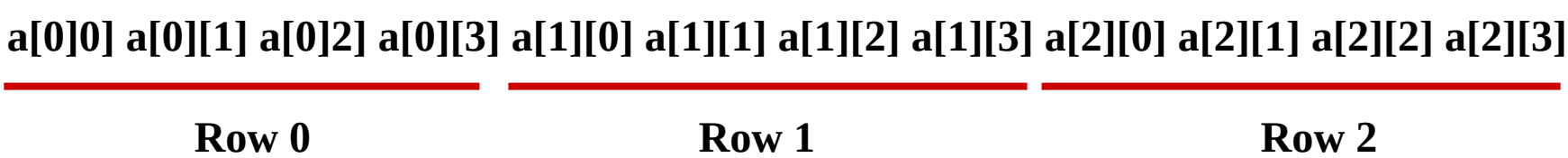
- **This is**
  an array of size 3 **names[3]**
  whose elements are arrays of size 4 **[4]**
  whose elements are characters **char**
- **Declare it like this:** `char names[3][4];`



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**
  - **s: number of bytes allocated per array element**

a[i][j] → is allocated memory location at address x + (i * c + j) * s



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];

Multidimensional Arrays

double a[100];

int b[4][6];

char c[5][4][9];

A k-dimensional array has a size for each dimensions.

Let s_i be the size of the ith dimension. If array elements are of type T and v=sizeof(T), the array declaration will allocate space for $s_1*s_2*...*s_k$ elements which is $s_1*s_2*...*s_k*v$ bytes.

Initialization : 2-d arrays

- int a[2][3] = {1,2,3,4,5,6};
- int a[2][3] = {{1,2,3}, {4,5,6}};
- int a[][3]   = {{1,2,3}, {4,5,6}};

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 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]);
```

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  ("\n %f", a[ i ][ j ]);
```

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

Contd.

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

Passing 2-D Arrays

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

Formal parameter declarations

- When a multi-dimensional array is a formal parameter in a function definition, all sizes except the first must be specified so that the compiler can determine the correct storage mapping function.

```
int sum ( int a[ ][5] )  {
        int i, j, sum=0;
        for (i=0; i<3; i++)
                for (j=0; j<5; j++)
                        sum += a[i][j];
        return sum;
}
```

Example: Matrix Addition

```
#include  <stdio.h>
#define N 100
int main() {
   int  a[N][N], b[N][N],
        c[N][N], 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++)  {
      printf  (“\n”);
      for  (q=0; q<n; q++)
         printf (“%f   ”, a[p][q]);
    }
}
```

Example Usage

```
#include  <stdio.h>

int main() {
    int  a[15][25],  b[15]25];
    :
    :
    add (a, b, 15, 25);
    :
}

void  add (int x[ ][25], int y[ ][25], int rows, int cols) {

    : : :

}
```

Pointers and multi-d arrays

int a[3][5]

- **We can think of a[i] as the ith row of a.**
- **We can think of a[i][j] as the element in the ith row, jth column.**
- **The array name, a (&a[0]) is a pointer to an array of 5 integers.**
- **The base address of the array is &a[0][0].**
- **Starting at the base address the compiler allocates contiguous space for 15 ints.**

Passing 2-d arrays to functions as pointers

We can use

f (int a [ ][5]) {…….}

or

f (int (*a) [5]) {.........}

We need parenthesis (*a) since [ ] have a higher precedence than *

Note:

int (*a)[5] declares a pointer to an array of 5 ints.

int *a[5] declares an array of 35 pointers to ints.

The storage mapping function

- **(The mapping between pointer values and array indices.)**

T mat[M][N];

  - **The storage mapping function : a[i][j] is equivalent to *(&a[0][0] + N*i + j)**

address (mat[i][j]) = address(mat[0][0]) + (i * N + j) * size(T)

= address (mat [0][0]) + i * N * size(T) + j * size(T)

= address (mat [0][0]) + i * size(row of T) + j * size(T)

Pointers and multi-d arrays

- **There are numerous ways to access elements of a 2-d array.**
- **a[i][j] is equivalent to:**
  - *(a[i]+j)
  - (*(a+i)[j])
  - *((*(a+i))+j)
  - *(&a[0][0] + 5*i + j)

Exercise

- Write a function **int maxinrow (..)** which takes as parameters a two dimensional matrix ***M*** declared with ***N*** columns having ***r*** rows and ***c*** columns, the values of ***r*** and ***c*** , a 1-d array ***rarr***, and fills up each of the elements in the 1-d array with the maximum element in the corresponding columns of ***M***. The function must return the size of the array ***rarr.***

```
int maxinrow (int M[ ][N], int r, int c, int rarr) {
        int i, j, max;
        for (i=0; i<c; i++) {
              max = M[i][0] ;
              for (j=1; j<r; j++ {
                    if (M[i][j] > max)
                          max = M[i][j] ;
              }
              rarr[i] = max;
         }
         return c;
}
```

3-dimensional arrays

- int a[X][Y][Z];
- **The compiler will allocate** X*Y*Z **contiguous ints.**
- **The base address of the array is &a[0][0][0]**
- **Storage mapping function :**

  a[i][j][k] ≡*(&a[0][0][0] + Y*Z*i +Z*j + k)

- **In the header of the function definition, the following 3 parameter declarations are equivalent:**
  - int a[][Y][Z], int a[X][Y][Z], int (*a)[Y][Z]

The use of typedef

```
#define N 4
typedef double scalar;
typedef scalar vector[N];
typedef scalar matrix[N][N];
    or typedef vector matrix[N];
```

```
void add (vector x, vector y, vector z) {
        int i;
        for (i=0; i<N; i++)
                x[i] = y[i]+z[i];
}

scalar dot_product (vector x, vector y) {
    int i;
    scalar sum = 0.0;
    for (i=0; i<N; i++)
        sum += x[i]*y[i];
    return sum;
}
```

```
void multiply (matrix x, matrix y, matrix z) {
        int i, j, k;
        for (i=0; i<N; i++)  {
                for (j=0; j<N; j++) {
                        x[i][j] = 0.0;
                        for (k=0; k<N; k++)  {
                                x[ i ][ j ] += y[ I ][ k ]*z[ k ][ j ];
                }
        }
}
```