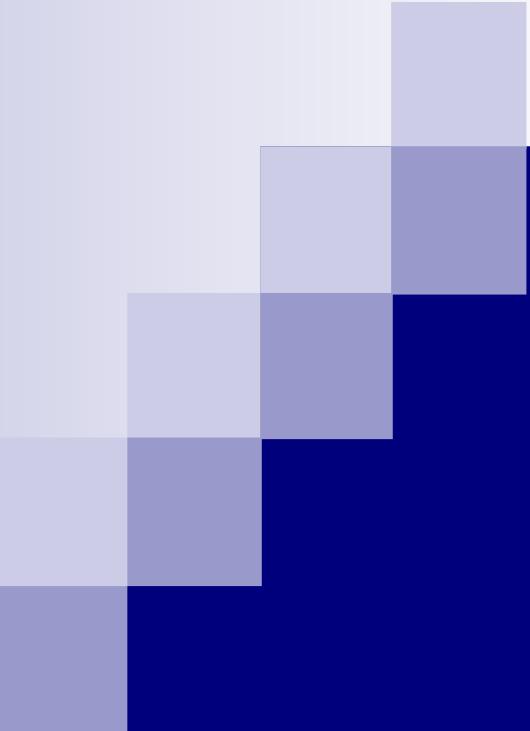


# **CS10003:**

# **Programming & Data Structures**

**Dept. of Computer Science & Engineering  
Indian Institute of Technology Kharagpur**

*Autumn 2020*



# Multi-dimensional arrays

# 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  $v=\text{sizeof}(T)$ , the array declaration will allocate space for  $s_1 * s_2 * \dots * s_k$  elements which is  $s_1 * s_2 * \dots * s_k * v$  bytes.

# 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]

# Pointers and multi-d arrays

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

# Pointers and multi-d arrays

- Int a[3][5];
- We can think of a[i] as the *i*th row of a.
- We can think of a[i][j] as the element in the *i*th row, *j*th 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.

# The storage mapping function

- (The mapping between pointer values and array indices.)
- int a[M][N];
  - The storage mapping function : **a[i][j]** is equivalent to **\*(&a[0][0] + N\*i + j)**

# 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[][][Z], int a[X][Y][Z], int (\*a)[Y][Z]**

# 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;  
}
```

In the header of the function definition, the following 3 parameter declarations are equivalent:

**int a[][5]**  
**int a[3][5]**  
**int (\*a)[5]**

# Initialization : multi-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}};`

# 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];  
            }  
        }  
    }  
}
```



**Thank You!**