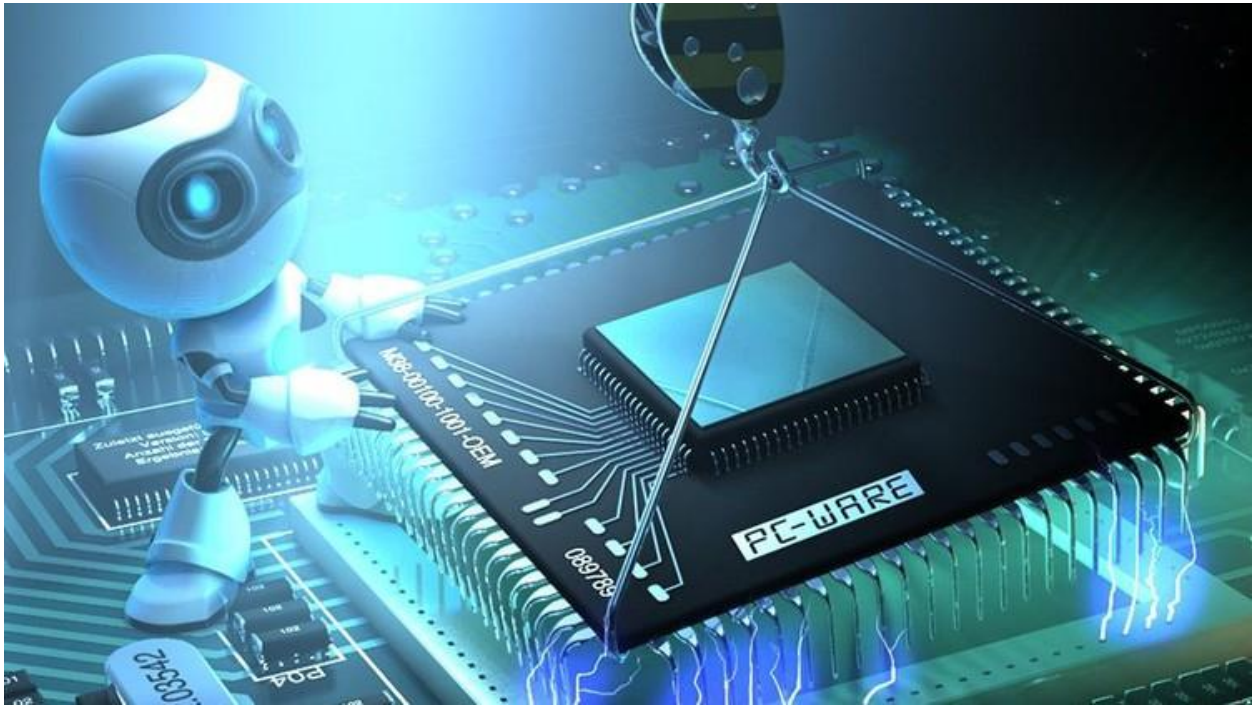
Practise on Matrices



Structured Data Types

Matrices

- **Declaration in C:**

```
type name[NumRows][NumColumns];      // indexed starting at (0,0)
```

- **Storage by rows in consecutive memory locations**
  - Access element `A[i][j]`: **@start A + (i*NumColumns + j) * size**

(size: size of the elements of A)

Structured Data Types

Matrices

- **Examples:**

Declaration in C	Size of Element	Size of Matrix	@element (i, j)
char A[80][25];	**1B**	2000B	@start A + i*25 + j
char *B[80][10];	**4B**	3200B	@start B + (i*10+j)***4**
double C[1024][100];	**8B**	800KB	@start C + (i*100+j)***8**
int *D[5][90];	**4B**	1800B	@start D + (i*90+j)***4**
int E[100][30];	**4B**	12000B	@start E + (i*30+j)***4**

Structured Data Types

Matrices 3-Dimensions

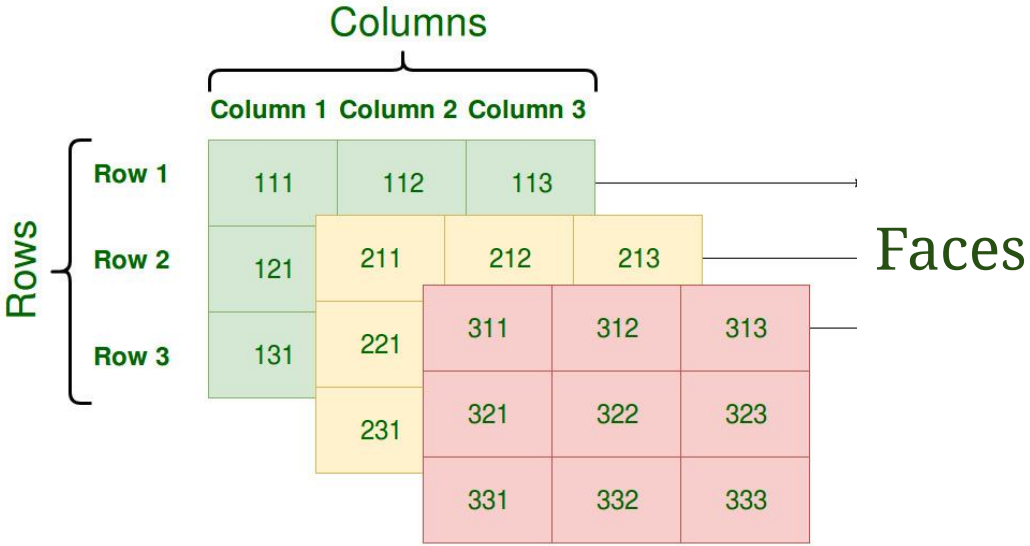
- **Example, 3-dimensional integer array:**

```
int M3D[10][64][48]     // each int occupies 4 bytes
```

- **The matrix is stored in consecutive memory locations: face to face and in each face by rows.**

- `M3D[face][row][column]`

3-Dimensional Matrix



Structured Data Types

Matrices 3-Dimensions

- **Example, 3-dimensional integer array:**

```
int M3D[10][64][48]     // each int occupies 4 bytes
```

- **The matrix is stored in consecutive memory locations: face to face and in each face by rows.**
- **Access to element** `M3D[face][row][column]`**:**
  - **@start M3D + (face·64·48 + row·48 + column)·4**

Structured Data Types

Matrices 3-Dimensions

- **Example, 3-dimensional integer array:**

```
int M3D[10][64][48]     // each int occupies 4 bytes
```

- **The matrix is stored in consecutive memory locations: face to face and in each face by rows.**
- **Access to element** `M3D[face][row][column]`**:**
  - **@start M3D + (face·64·48 + row·48 + column)·4**

Structured Data Types

Matrices 3-Dimensions

- **So, given a 3-dimensional matrix:**

```
type M3D[NumFaces][NumRows][NumCols]
```

- **The matrix is stored in consecutive memory locations: face to face and in each face by rows.**
- **Access to element** `M3D[face][row][column]`**:**
  - **@start M3D + (face·NumRows·NumCols + row·NumCols + column)·size**

Structured Data Types

Matrices 3-Dimensions

- **Example, 3-dimensional integer array:**

```
int M3D[10][64][48]      // each int occupies 4 bytes
```

- **The matrix is stored in consecutive memory locations: face to face and in each face by rows.**
- **Access to element** `M3D[face][row][column]`:
  - @start M3D + (face·64·48 + row·48 + column)·4
- **It is simple to figure out how N-dimensional matrices are stored/accessed.**

Structured Data Types

Matrices 4-Dimensions

- **So, given a 4-dimensional matrix:**
  `type M3D[NumFaces][NumRows][NumCols][NumSets]`

- **The matrix is stored in consecutive memory locations: face to face and in each face by rows.**

- **Access to element** `M3D[face][row][column][set]`**:**

@start M3D + (face·NumRows·NumCols·NumSets + row·NumCols·NumSets +

column·NumSets + set)·size

Structured Data Types

Matrices

- **Declaration in C:**

```
type name[NumRows][NumColumns];      // indexed starting at (0,0)
```

- **Storage by rows in consecutive memory locations**
  - Access element `A[i][j]`: **@start A + (i*NumColumns + j) * size**

(size: size of the elements of A)

Practise on Matrices

Example:

```
int Mfc(int M[50][80], int fil, int col) {
    return M[fil][col];
}
```

M[fil][col] → **@M + (fil·80 + col) · 4**

Matrices

Example:

```
int Mfc(int M[50][80], int fil, int col) {
   return M[fil][col];
 }
```

Traduction:

```
Mfc: pushl %ebp
     movl %esp, %ebp

     imull $80,12(%ebp),%eax # fil → 12[%ebp]
     addl 16(%ebp),%eax      # col → 16[%ebp]
     movl 8(%ebp),%ecx       # @M → 8[%ebp]
     movl (%ecx,%eax,4),%eax
     popl %ebp               # result in %eax
     ret
```

%eax ← fil·80
%eax ← fil·80 + col
%ecx ← @M