Structured Programming Language

Lecture 5

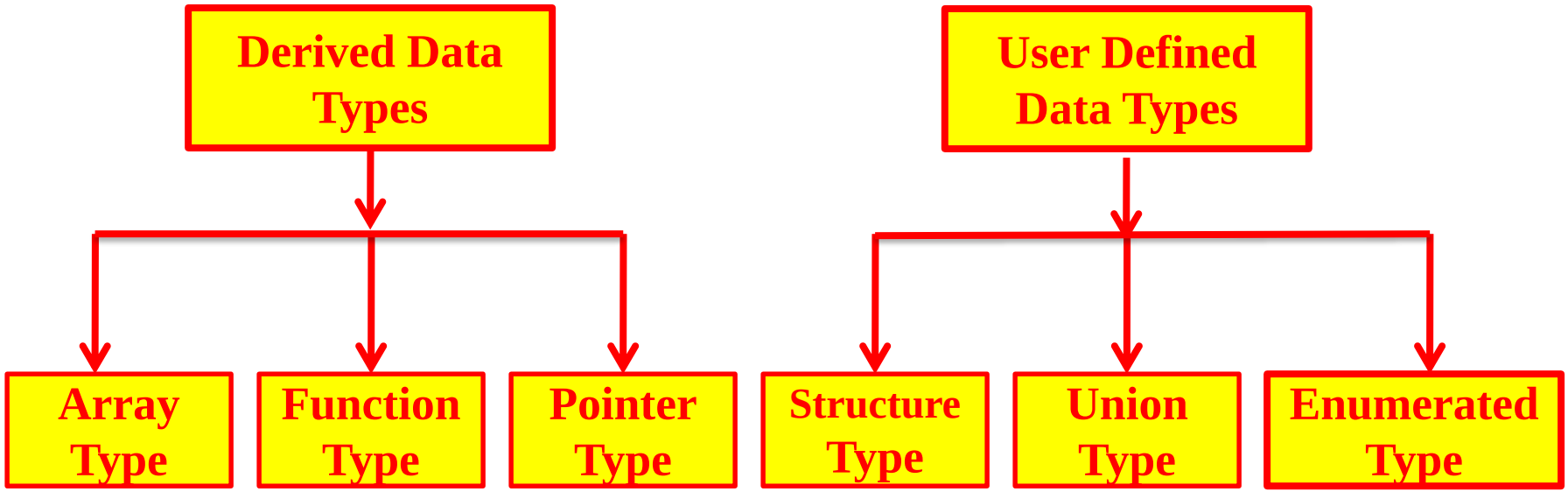
Overview

- ✓ One Dimensional Array.
- ✓ Two Dimensional Array.
- ✓ Inserting Elements in Array.
- ✓ Reading Elements from an Array.
- ✓ Different operations on Array.

What is an Array?

✓An *array* is a sequenced collection of elements that share the same data type.

✓Elements in an array share the same name



Array Types in C

C supports two types of arrays:

- ✓*Fixed Length Arrays* – The programmer "hard codes" the length of the array, which is fixed at compile-time.
  - ✓Example: int arr[50];

- ✓*Variable-Length Arrays* – The programmer doesn't know the array's length until run-time.
  - ✓Example: int size=50; int arr[size];

Declaring an Array

✓To declare an array, we need to specify its data type, the array's identifier and the size:

data_type arrayName [arraySize];

✓The arraySize can be a constant (for fixed length arrays) or a variable (for variable-length arrays).

✓Before using an array (even if it is a variable-length array), we must declare and initialize it!

Declaring an Array

- When declaring arrays, specify
  - Name
  - Type of array
  - Number of elements

    data_type arrayName [arraySize];
  - Examples:

    ```
    int c[ 10 ];
    float myArray[ 3284 ];
    ```
- Declaring multiple arrays of same type
  - Format similar to regular variables
  - Example:

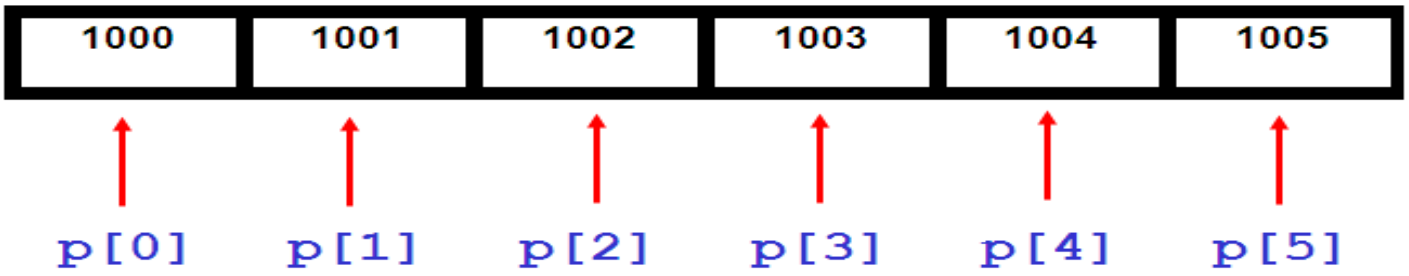
    ```
    int b[ 100 ], x[ 27 ];
    ```

Declaring an Array

So what is actually going on when you set up an array?

Memory

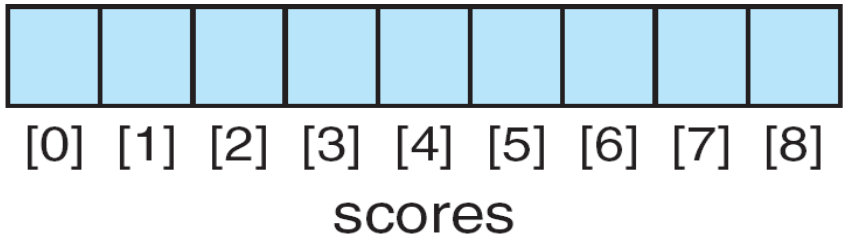
1. Each element is held in the next location along in memory
2. Essentially what the PC is doing is looking at the FIRST address (which is pointed to by the variable p) and then just counts along.



Declaration Examples

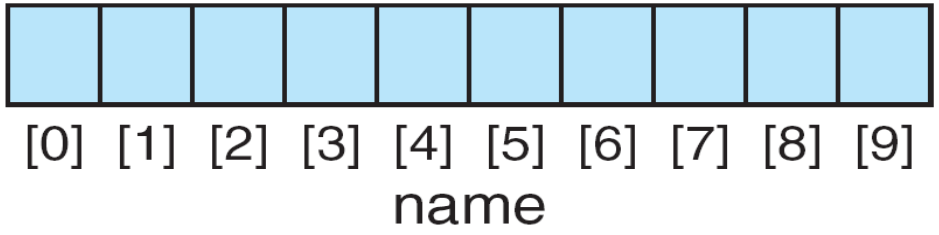
```
int scores [9];
```

type of each element



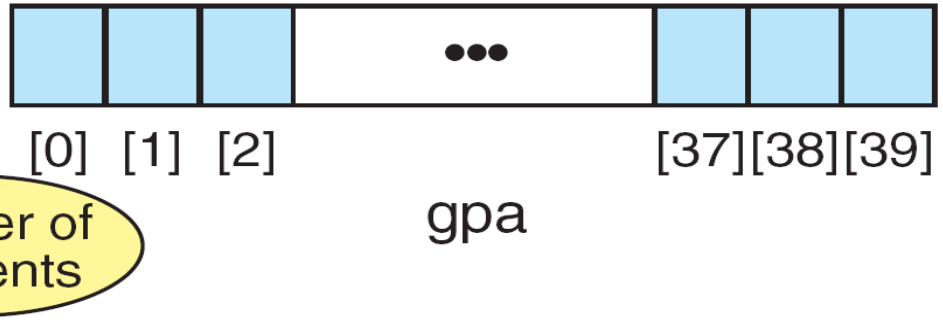

```
char name [10];
```

name of the array




```
float gpa [40];
```

number of elements



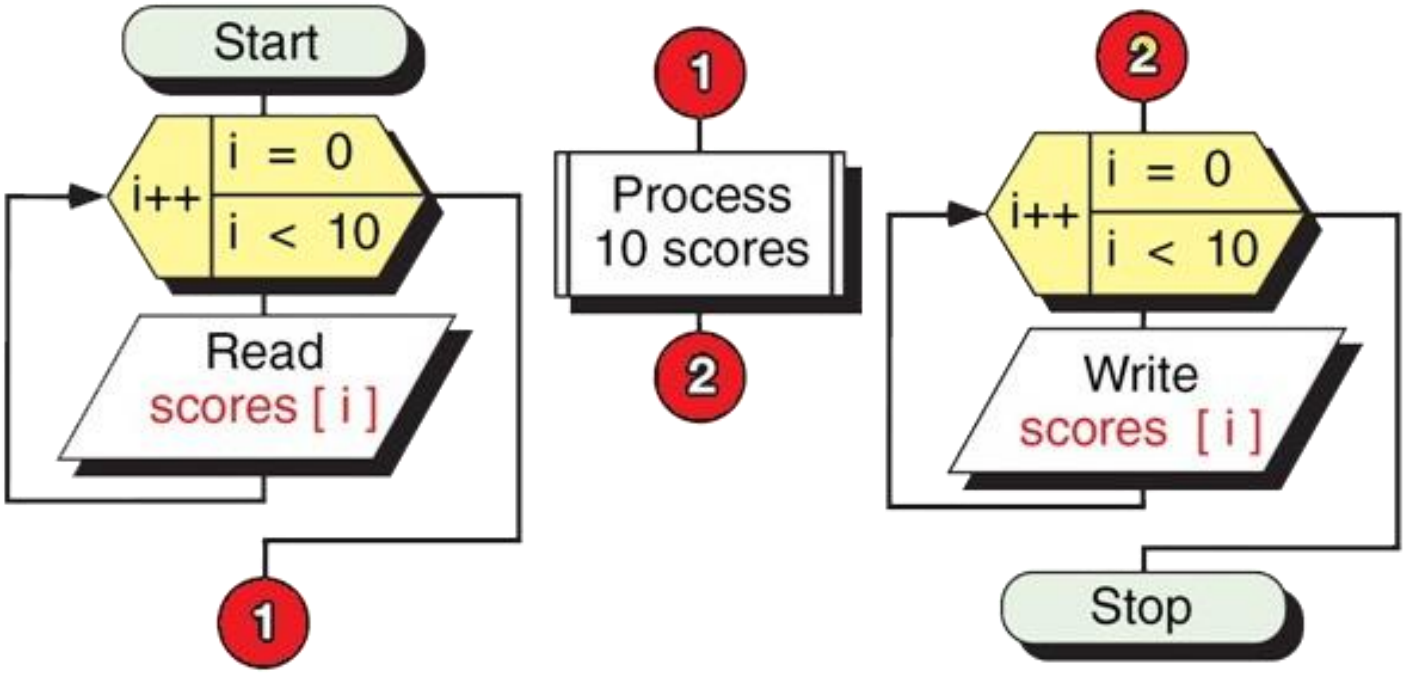
How to Refer Array Elements

- We can reference array elements by using the array's subscript/index.
- The first element has a subscript of 0.
- The last element in an array of length *n* has a subscript of *n-1*.
- To index a subscript, use the array name and the subscript in a pair of square brackets:

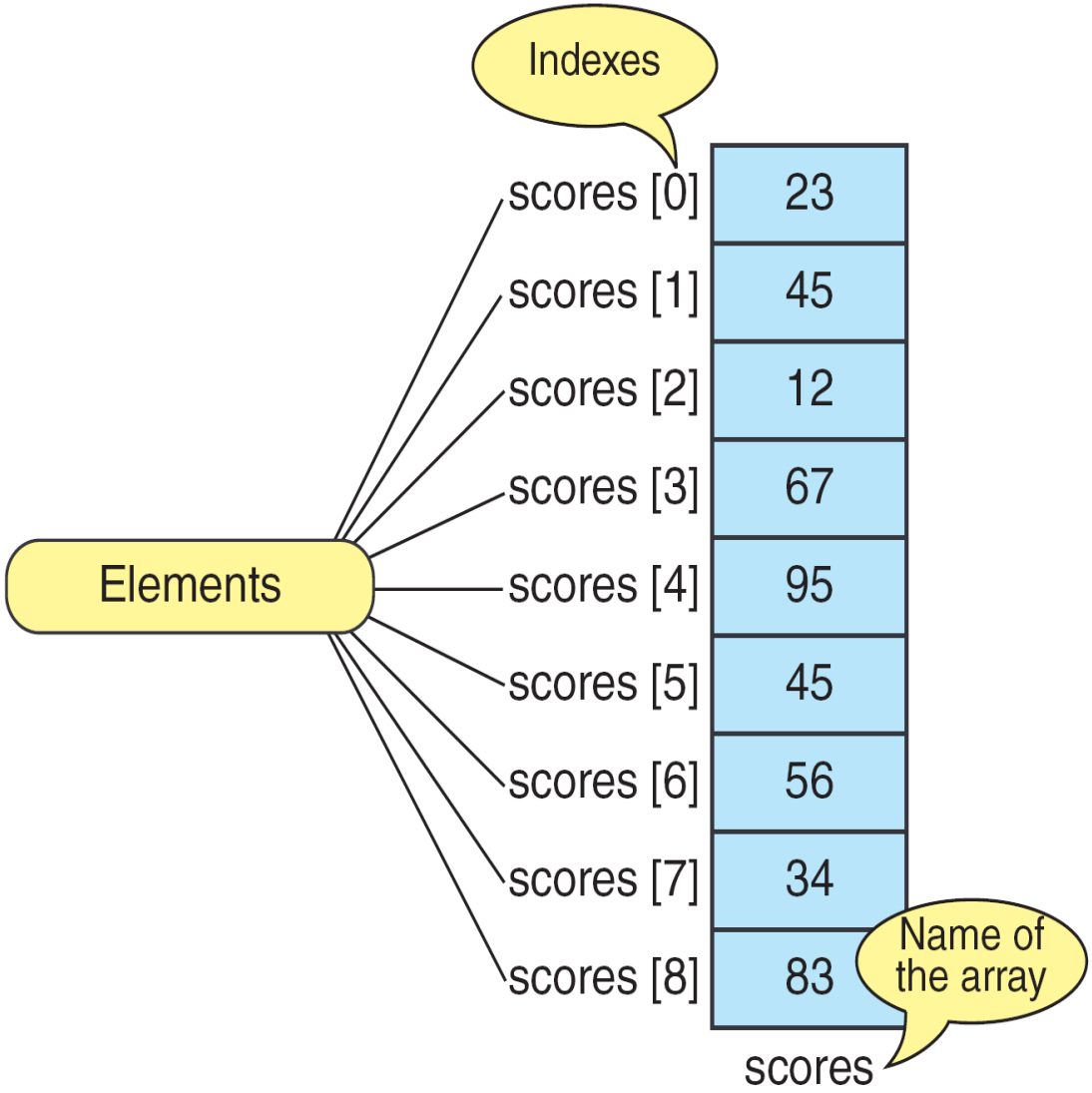
```
a[12];
```

Arrays and Loops

Since we can refer to individual array elements using numbered indexes, it is very common for programmers to use for loops when processing arrays.



Example : scores Array



Accessing Elements

✓To access an array's element, we need to provide an integral value to identify the index we want to access.

✓We can do this using a constant:

```
scores[0];scores[2];scores[5];...etc
```

✓We can also use a variable:

```
for(i = 0; i < 9; i++)
{
    scoresSum += scores[i];
}
```

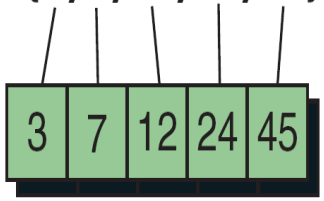
Array Initialization

- ✓ We can initialize only fixed-length array elements when we define an array.
- ✓ If we initialize fewer values than the length of the array, C assigns zeroes to the remaining elements.
- ✓ If we initialize more values than size then the compile throws an error.

Array Initialization Examples

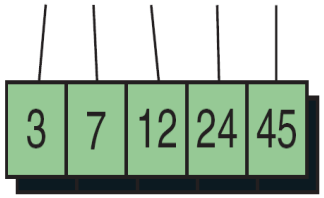
(a) Basic Initialization

```
int numbers[5] = {3,7,12,24,45};
```



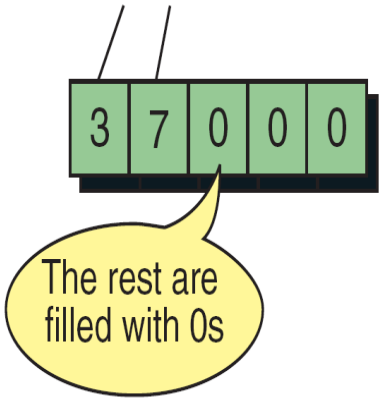
(b) Initialization without Size

```
int numbers[ ] = {3,7,12,24,45};
```



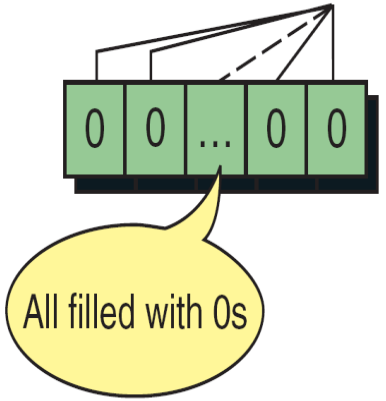
(c) Partial Initialization

```
int numbers[5] = {3,7};
```



(d) Initialization to All Zeros

```
int lotsOfNumbers [1000] = {0};
```



Assigning Values using a *for* Loop

✓Once we know the length of an array or the size of the array, we can input values using a for loop:

```
arrayLength = 9;
for (j = 0; j < arrayLength; j++)
{
   scanf("%d", &scores[j]);
}//end for
```

Assigning Values to Individual array Elements

✓We can assign any value that reduces to an array's data type:

```
scores[5] = 42;
scores[3] = 5 + 13;
scores[8] = x + y;
scores[0] = pow(7, 2);
```

Copying Entire Arrays

✓We cannot directly copy one array to another, even if they have the same length and share the same data type.

✓Instead, we can use a for loop to copy values:

```
for(m = 0; m < 25; m++)

{
    a2[m] = a1[m];
}//end for
```

Swapping Array Elements

✓To swap (or exchange) values, we must use a temporary variable.

✓A common novice's mistake is to try to assign elements to one another:

```
/*The following is a logic error*/
numbers[3] = numbers[1];
numbers[1] = numbers[3];

/*A correct approach …*/
temp = numbers[3];
numbers[3] = numbers[1];
numbers[1] = temp;
```

Printing Array Elements

✓To print an array's contents, we would use a for loop:

```
for(k = 0; k < 9; k++)
{
    printf("%d ",scores[k]);
}//end for
```

Range Checking

✓Unlike some other languages, C does not provide built-in range checking.

✓Thus, it is possible to write code that will produce "out-of-range" errors, with unpredictable results.

✓Common Error (array length is 9):

```
for(j = 1; j <= 9; j++)
{
    scanf("%d",&scores[j]);
}//end for
```

Programs on 1D Arrays

1. **Reversing an array**

```
j=N-1;
for(i=0;i<N/2;i++)
{       temp=a[i];
        a[i]=a[j];
        a[j]=temp;
        j--;
}
```

2. **Exchanging first half with second half** (array size is even)

```
 for(i=0,j=N/2;i<N/2;i++,j++)
 {      temp=a[i];
        a[i]=a[j];
        a[j]=temp;
 }
```

Programs on 1D Arrays

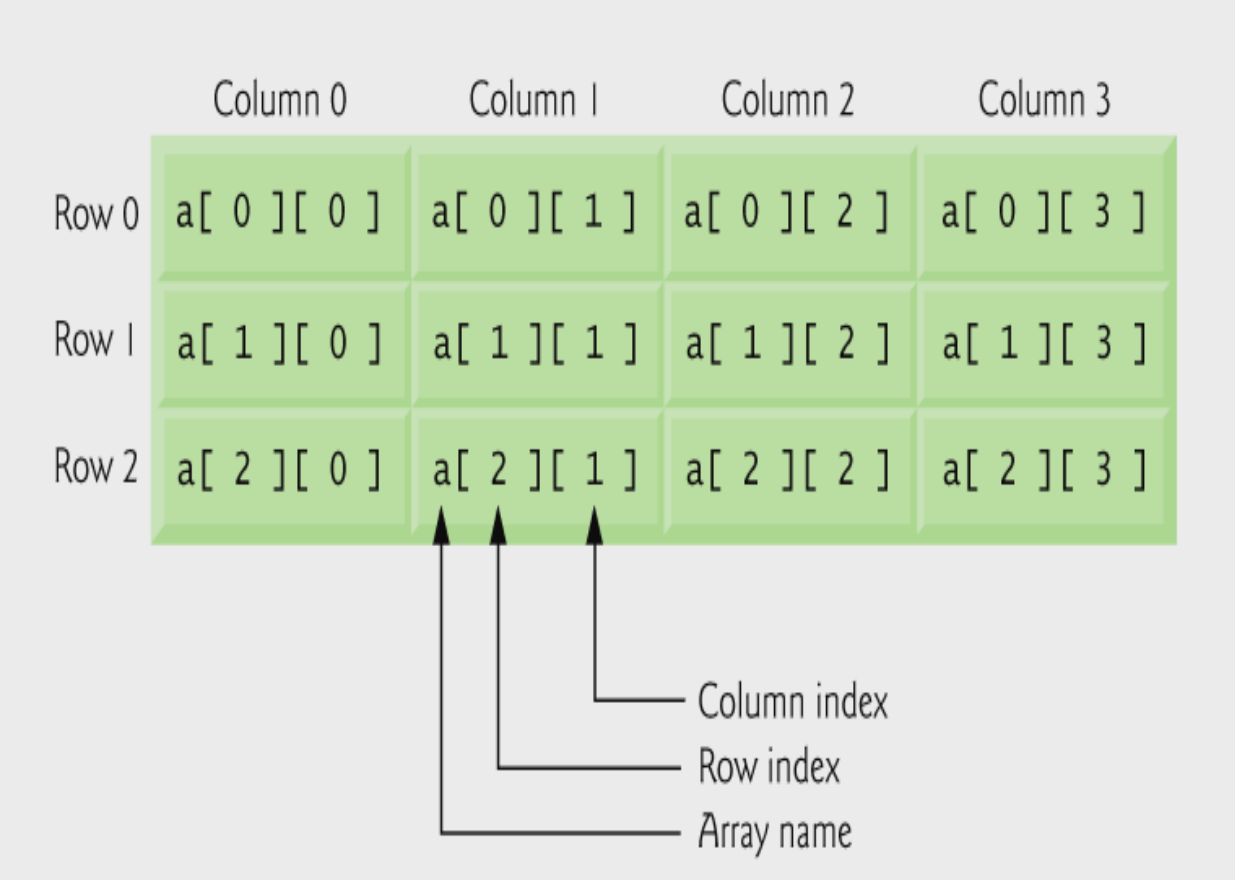
3. **Interchanging alternate locations**

(size of the array will be even)

```
for(i=0;i<N;i=i+2)
{
   temp= a[i]; //swapping the alternate
    a[i]=a[i+1]; //locations using third
    a[i+1]=temp;
} //variable
```

2D Array Definition

- ✓ To define a fixed-length two-dimensional array:
  int table[5][4];
- ✓ 2-D Array a[m][n];m Rows and n columns - int a[3][4];



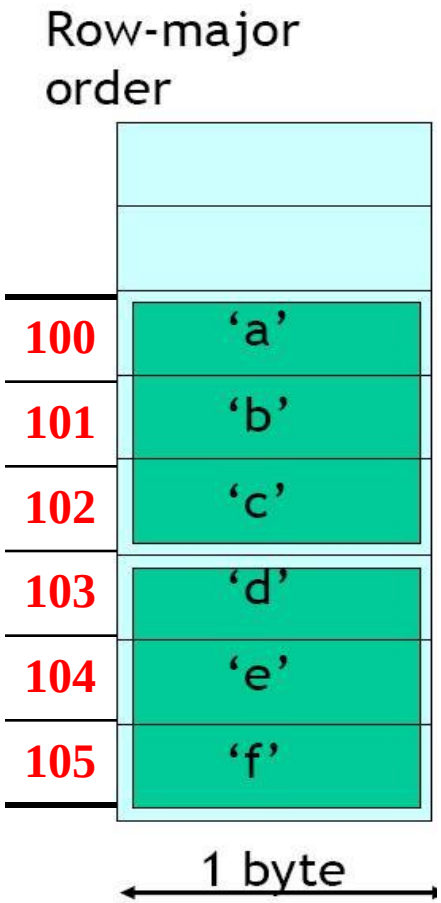
Implementation of 2-D Array in memory

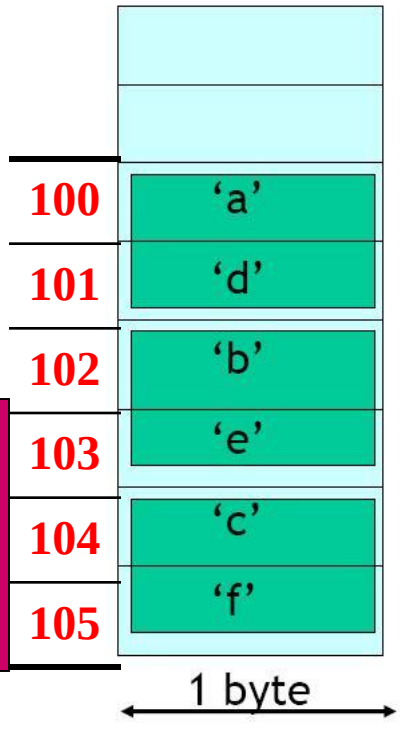
The elements of 2-D array can be stored in memory by two-Linearization method :

1. **Row Major**
2. **Column Major**

Here is a 2x3 array of characters to be stored:

‘a’	‘b’	‘c’
‘d’	‘e’	‘f’





Therefore,the memory address calculation will be different in both the methods.

Implementation of 2-D Array in memory

1. Row Major Order for a[m][n] or a[0…m-1][0…n-1]

   Address of a[I, J] element = B + w(N_c (I-L_r)+(J-L_c)

2. Column Major Order- a[m][n] or a[0…m-1][0…n-1]

   Address of a[I, J] element = B + w(N_r (J-L_c)+(I-L_r)

   B = Base Address, I = subscript (row), J = subscript (column), N_c = No. of column, N_r = No. of rows, L_r = row lower bound(0) , L_c = column lower bound (0), U_c=column upper bound(n-1) ,U_r =row upper bound (m-1), w = element size.

Memory Address Calculation

1. Row Major Order formula

$$X = B + W * [I * C + J]$$

2. Column Major Order formula

$$X = B + W * [I + J * R]$$

Memory Address Calculation

- ✓ An array S[10][15] is stored in the memory with each element requiring 4 bytes of storage. If the base address of S is 1000, determine the location of S[8][9] when the array is S stored by (i) Row major (ii) Column major.

ANSWER

- ✓ Let us assume that the Base index number is [0][0].
- ✓ Number of Rows = R = 10
- ✓ Number of Columns = C = 15
- ✓ Size of data = W = 4
- Base address = B = S[0][0] = 1000
- Location of S[8][9] = X

Memory Address Calculation

(i) When S is stored by Row Major

X = B + W * [8 * C + 9]

= 1000 + 4 * [8 * 15 + 9]

= 1000 + 4 * [120 + 9]

= 1000 + 4 * 129

= 1516

(ii) When S is stored by Column Major

X = B + W * [8 + 9 * R]

= 1000 + 4 * [8 + 9 * 10]

= 1000 + 4 * [8 + 90]

= 1000 + 4 * 98

= 1000 + 392

= 1392

Programs on 2D Arrays

1. sum of rows a[3][2]

```
for(row=0;row<3;row++)
{  sum=0;
      for(col=0;col<2;col++)
      sum+=a[row][col];
   printf("sum is %d", sum);
}
```

2. sum of columns a[2][3]

```
for(col=0;col<3;col++)
{sum=0;
      for(row=0;row<2;row++)
      sum+=a[row][col];
 printf("sum is %d", sum);
}
```

Programs on 2D Arrays

3. sum of left diagonals a[3][3]

```
sum=0;
for(row=0;row<3;row++)
sum+=a[row][row]
```

4. sum of right diagonals a[3][3]

```
sum=0;
for(row=0,col=2;row<3;row++,col--)
sum+=a[row][col]
```

5. Transpose of a matrix a[3][2] stored in b[2][3]

```
for(row=0;row<3;row++)
for(col=0;col<2;col++)
b[col][row]=a[row][col];
```

Programs on 2D Arrays

6. **Display the lower half** a[3][3]

```
for(row=0;row<3;row++)
for(col=0;col<3;col++)
     if(row<=col)
          printf("%d",a[row][col]);
```

7. **Display the Upper Half** a[3][3]

```
for(row=0;row<3;row++)
for(col=0;col<3;col++)
     if(row>=col)
          printf("%d",a[row][col]);
```

Programs on 2D Arrays

8. **ADD/Subtract matrix**

```
a[2][3]+/-b[2][3]=c[2][3]
for(row=0;row<2;row++)
for(col=0;col<3;col++)
c[row][col]=a[row][col] +/- b[row][col]
```

9. **Multiplication** a[2][3]*b[3][4]=c[2][4]

```
for(row=0;row<2;row++)
for(col=0;col<4;col++)
{
  c[row][col]=0;
     for(k=0;k<3;k++)
       c[row][col]+=a[row][k]*b[k][col]
}
```