Two-Dimensional Arrays

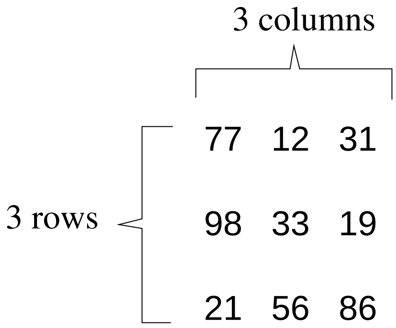
- What is a two-dimensional array ?

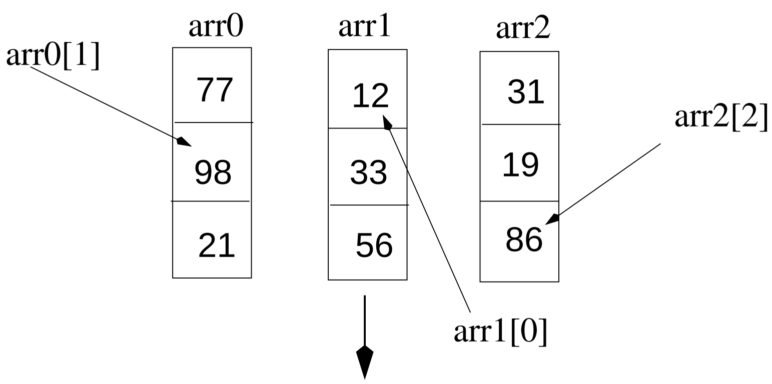
  - A collection of variables **having the same data type**

  - The variables are thought as being structured in two dimensions

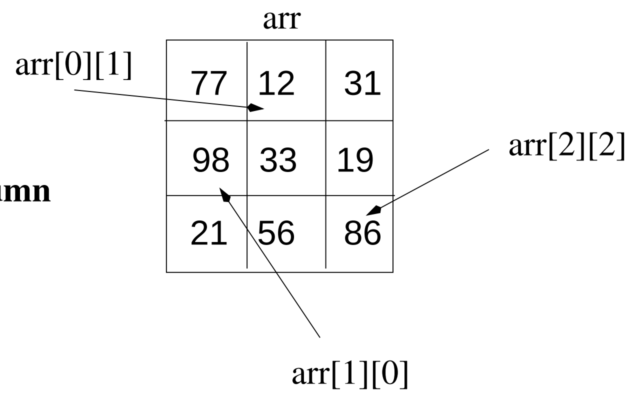
  - Each variable is accessed using:

  (i) a name which is the same for all of them
  (ii) a pair of indices that represent the position of the variable in each dimension





one two-dimensional array (3 x 3)



To access an element from a two-dimensional array, we need to specify both the row and column

a[i][j], 0<=i<3, 0<=j<3

  - In other words, **a two-dimensional array is an array of arrays !!**

- Declaring two-dimensional arrays

  - Specify the type of the variables followed by the size in each dimension

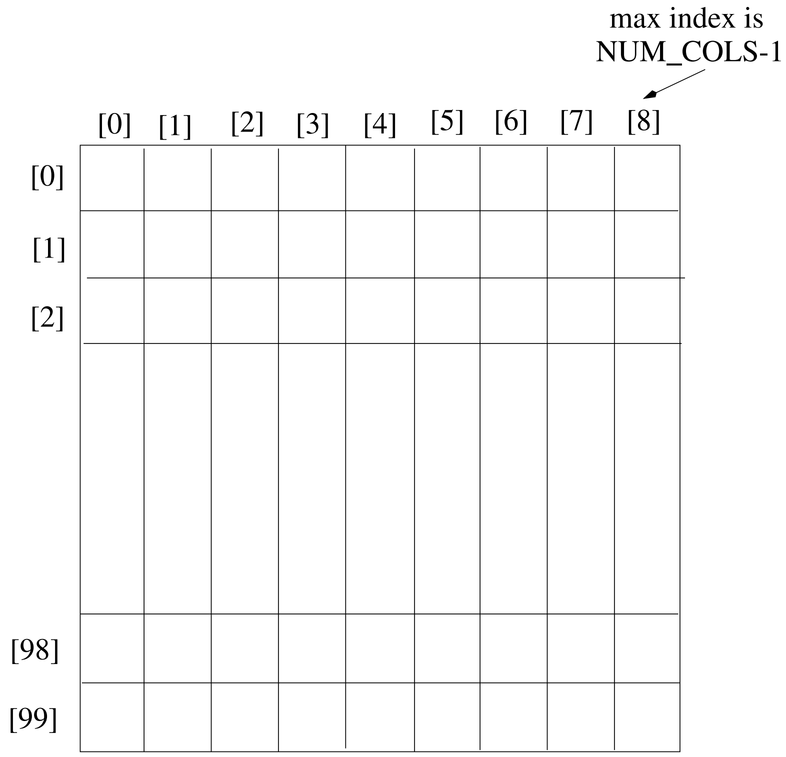
```
DataType ArrayName [ConstIntExpression][ConstIntExpression]
```

```
const int NUM_ROWS = 100;
const int NUM_COLS = 9;

int arr[NUM_ROWS][NUM_COLS];
```

must be int constants !!

max index is
NUM_ROWS-1



- Accessing individual components

ArrayName [IndexExpression][IndexExpression]

  - The *IndexExpression* may be just a constant, or a variable or even an expression

  - The result of *IndexExpression* **must be an integer value**

  - Examples: *value[4][1]*, *value[2*a+1][a-1]* (assuming *a* is an integer)

- Each array element can be treated the exactly the same way as a simple variable

  - *Assign* a value: *value[2][3]=100;*

  - Read a value into it: *cin >> value[2][3];*

  - Write its contents: *cout << value[2][3];*

  - Pass it as a parameter: *y=sqrt[value[2][3]);*

- Initializing two-dimensional arrays

  - Two-dimensional arrays can be initialized in a declaration

```
int value[2][3] =
{
  {10, 20, 12}
  {13, 33, 45}
};
```

  - Two-dimensional arrays can also be initialized using two loop statements:

data: 10 20 12 13 33 45

row by row

	[0]	[1]	[2]
[0]	10	20	12
[1]	13	33	45

```
for(i=0; i<2; i++)
  for(j=0; j<3; j++)
    cin >> arr[i][j];
```

column by column

	[0]	[1]	[2]
[0]	10	12	33
[1]	20	13	45

```
for(j=0; j<3; j++)
  for(i=0; i<2; i++)
    cin >> arr[i][j];
```

- Aggregate operations

  - **No** aggregate operations (assignment, comparison, arithmetic, etc.)

  - Passing a two-dimensional array to a function ? **Yes !!**

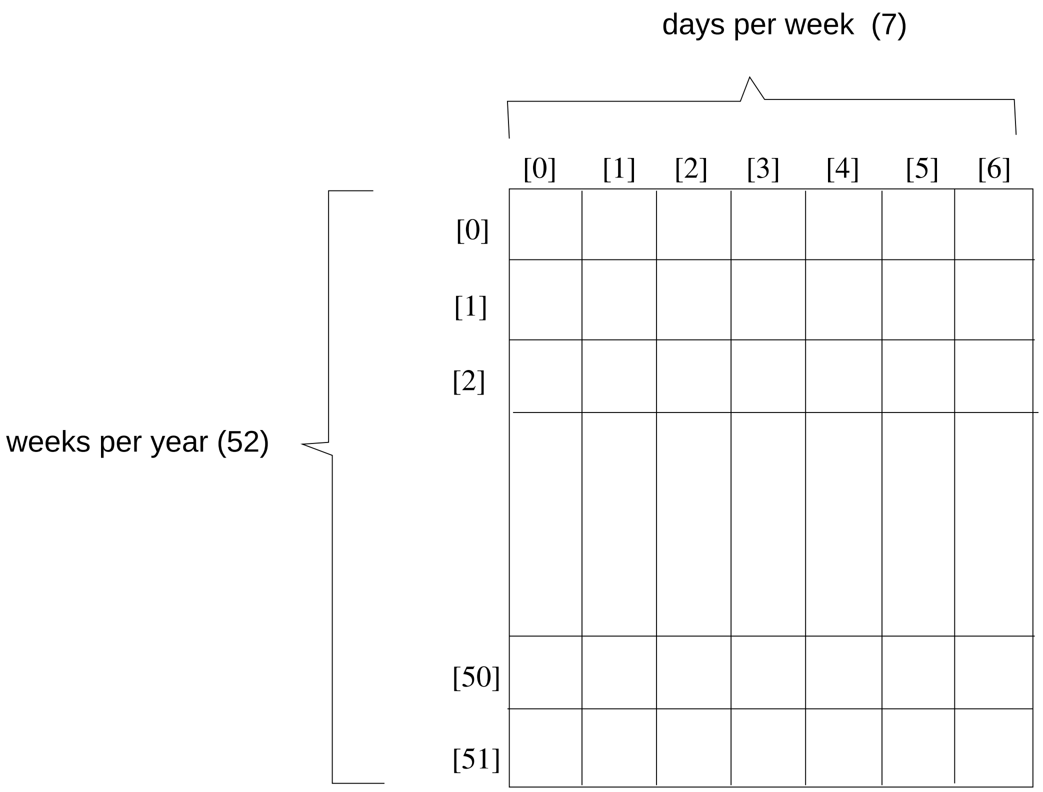
- Processing two-dimensional arrays

  - Sum the rows

  - Sum the columns

  - Sum all the elements

  - Print the table



temperature[i][j]: temperature on the (j+1)-th day of the (i+1)-th week

example: temperature[12][3] = 76 means that the temperature on the 4-th day of the 13-th week

- Sum the rows

  - Compute the average temperature per week

```
for(i=0; i<52; i++)
{
 sum = 0.0;
 for(j=0; j<7; j++)
   sum = sum + temperature[i][j];

 average = sum / 7;
 cout << "The average temperature of the " << i << "-th week is " << average << endl;
}
```

- Sum the columns

- Compute the average temperature for each day of the week

```
for(j=0; j<7; j++)
{
 sum = 0.0;
 for(i=0; i<52; i++)
   sum = sum + temperature[i][j];

 average = sum / 52;
 cout << "The average temperature of the " << j << "-th day is " << average << endl;
}
```

- Sum all the elements

- Compute the average temperature of the year

```
sum = 0.0;
for(i=0; i<52; i++)
 for(j=0; j<7; j++)
   sum = sum + temperature[i][j];

 average = sum / (7*52);
 cout << "The average temperature of the year is " << average << endl;
}
```

- Print the temperatures

- Week by week

```
for(i=0; i<52; i++)
{
 for(j=0; j<7; j++)
   cout << temperature[i][j] << " ";
 cout << endl;
}
```

- Day by day

```
for(j=0; j<7; j++)
{
  for(i=0; i<52; i++)
   cout << temperature[i][j] << " ";
 cout << endl;
}
```