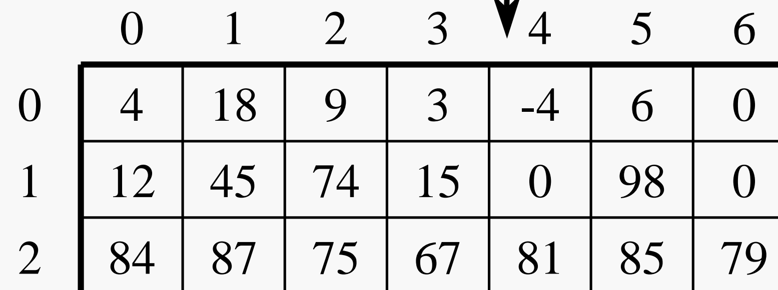
Multidimensional Arrays

C++ also allows an array to have more than one dimension.

For example, a two-dimensional array consists of a certain number of rows and columns:

```
const int NUMROWS = 3;
const int NUMCOLS = 7;
int Array[NUMROWS][NUMCOLS];
```



	0	1	2	3	4	5	6
0	4	18	9	3	-4	6	0
1	12	45	74	15	0	98	0
2	84	87	75	67	81	85	79

`Array[2][5]` 3rd value in 6th column

`Array[0][4]` 1st value in 5th column

The declaration must specify the number of rows and the number of columns, and both must be constants.

©1995-2000 Barnette ND, McQuain WD, Keenan MA

Processing a 2-D Array

A one-dimensional array is usually processed via a for loop. Similarly, a two-dimensional array may be processed with a nested for loop:

```
for (int Row = 0; Row < NUMROWS; Row++) {
    for (int Col = 0; Col < NUMCOLS; Col++) {
        Array[Row][Col] = 0;
    }
}
```

Each pass through the inner for loop will initialize all the elements of the current row to 0.

The outer for loop drives the inner loop to process each of the array's rows.

©1995-2000 Barnette ND, McQuain WD, Keenan MA

Initializing in Declarations

```
int Array1[2][3]  = { {1, 2, 3} , {4, 5, 6} };
int Array2[2][3]  = { 1, 2, 3, 4, 5 };
int Array3[2][3]  = { {1, 2} , {4 } };
```

If we printed these arrays by rows, we would find the following initializations had taken place:

Rows of Array1:

```
1 2 3
4 5 6
```

Rows of Array2:

```
1 2 3
4 5 0
```

Rows of Array3:

```
1 2 0
4 0 0
```

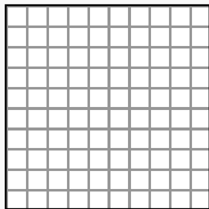
```
for (int row = 0; row < 2; row++) {
    for (int col = 0; col < 3; col++) {
        cout << setw(3)
             << Array1[row][col];
    }
    cout << endl;
}
```


©1995-2000 Barnette ND, McQuain WD, Keenan MA

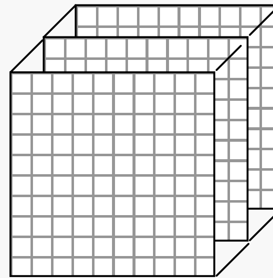
Higher-Dimensional Arrays

An array can be declared with multiple dimensions.

2 Dimensional

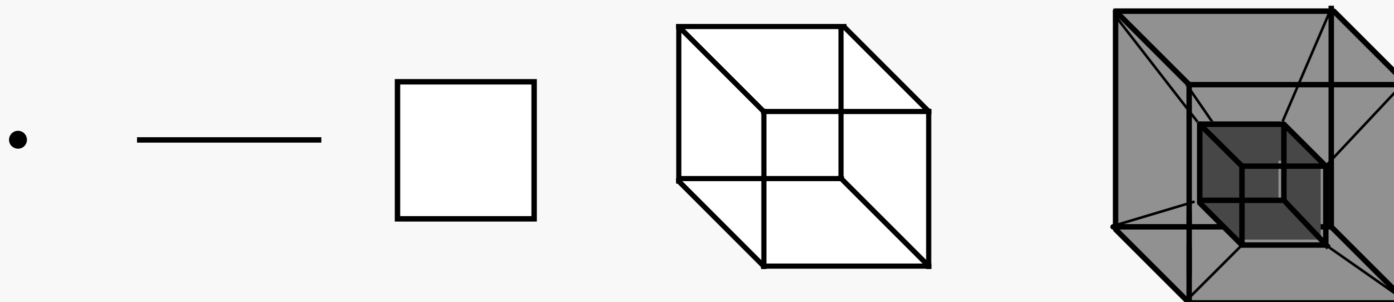


3 Dimensional



```
double Coord[100][100][100];
```

Multiple dimensions get difficult to visualize graphically.



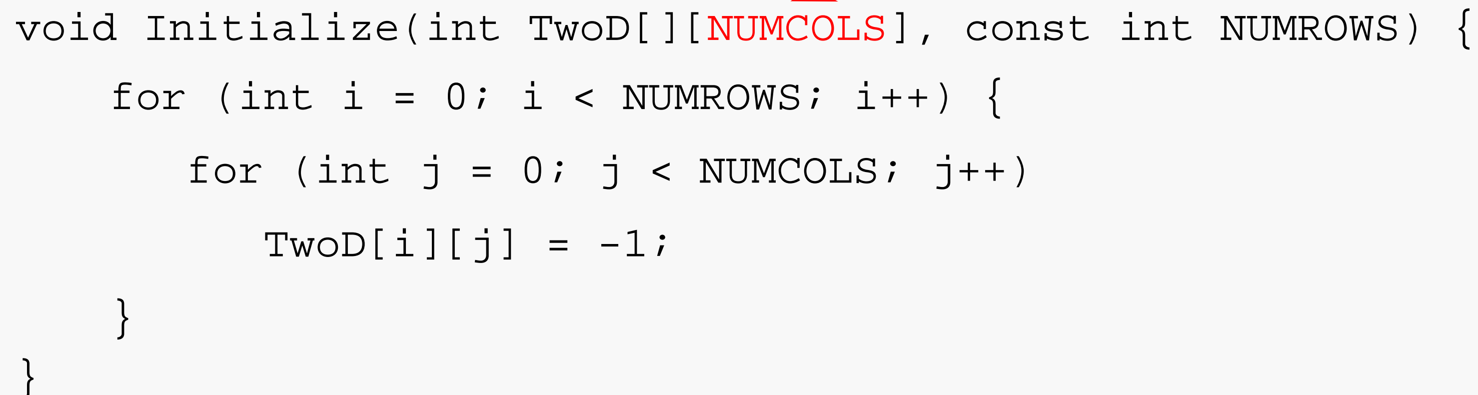
©1995-2000 Barnette ND, McQuain WD, Keenan MA

2-D Arrays as Parameters

When passing a two-dimensional array as a parameter, the base address is passed, as is the case with one-dimensional arrays.

But now the number of columns in the array parameter must be specified. This is because arrays are stored in row-major order, and the number of columns must be known in order to calculate the location at which each row begins in memory:

address of element (r, c) = base address of array
+ r*(number of elements in a row)*(size of an element)
+ c*(size of an element)



```
void Initialize(int TwoD[][NUMCOLS], const int NUMROWS) {
    for (int i = 0; i < NUMROWS; i++) {
        for (int j = 0; j < NUMCOLS; j++)
            TwoD[i][j] = -1;
    }
}
```


©1995-2000 Barnette ND, McQuain WD, Keenan MA