DYNAMIC AND MULTI-DIMENSIONAL ARRAYS

Specifying array size at runtime

Delroy A. Brinkerhoff

CREATING AND USING ARRAYS

WORKS

- `int scores[15];`
- `int scores[15][10];`
- `int* scores = new int[15];`
- `int* scores = new int[size];`

- `void function(int table[ ][12]);`

DOESN’T WORK

- `int* scores = new int[15][10];`
- `int* scores = new int[rows][cols];`

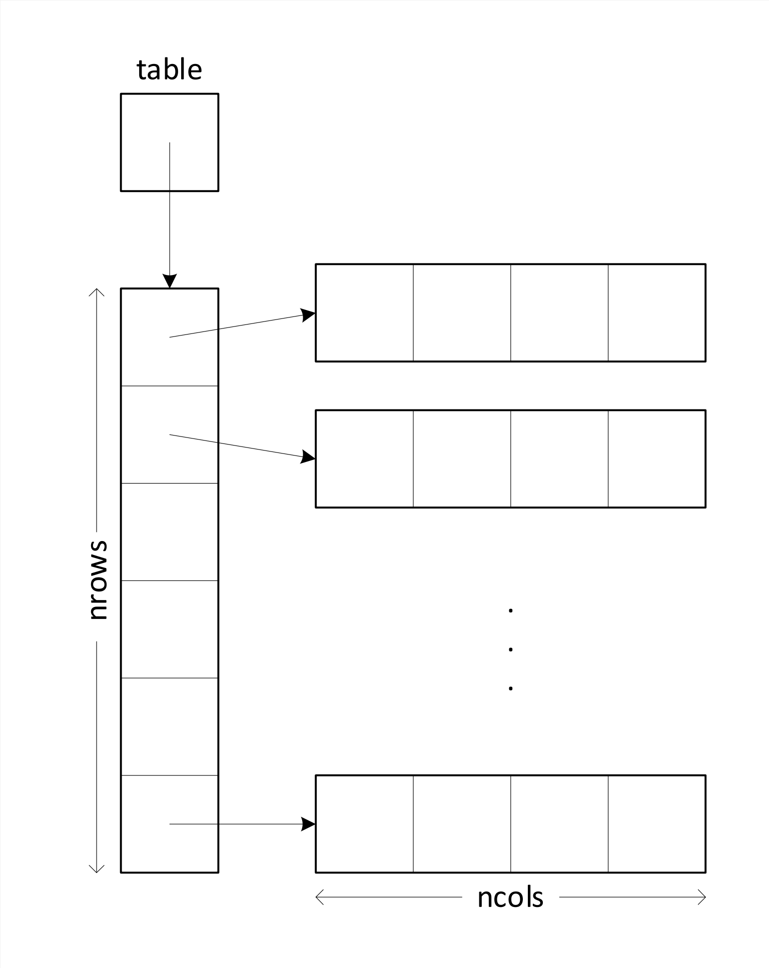
- `void function(int table[ ][ ]);`

AUTOMATIC TYPE DEDUCTION

```
int nrows = 15;
const int ncols = 10;

auto scores = new int[nrows][ncols];
```

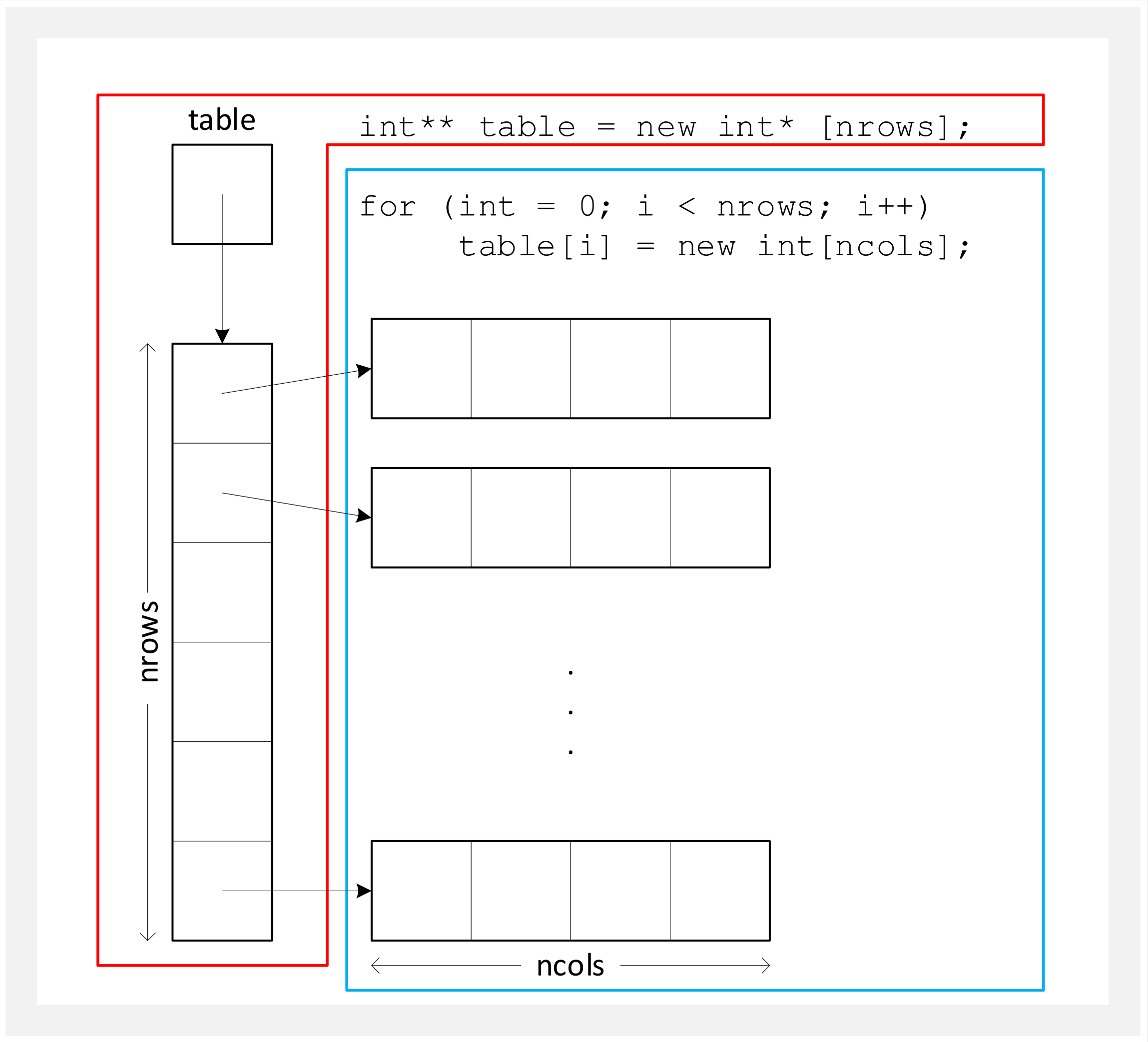
- The number of rows is dynamic:
  - Input
  - Calculated
- The number of columns is static:
  - Must be a compile-time constant
  - For two or more dimensions, only the first may be a variable



CREATING A TWO-DIMENSIONAL ARRAY AS AN ARRAY OF ARRAYS

- Advantages
  - Array sizes match a specific problem
  - Element access uses a two-index notation: `table[row][col]`
  - May be extended to higher dimensions
- Disadvantages:
  - creating the array
  - destroying the array

CREATING & DESTROYING ARRAYS



```
int** table = new int* [nrows];
```

```
for (int = 0; i < nrows; i++)
     table[i] = new int[ncols];
```

```
int** table = new int* [nrows];
for (int i = 0; i < nrows; i++)
     table[i] = new int[ncols];
```

```
for (int i = 0; i < nrows; i++)
     for (int j = 0; j < ncols; j++)
          ...table[i][j]...
```

```
for (int i = 0; i < nrows; i++)
        delete[] table[i];
delete[] table;
```

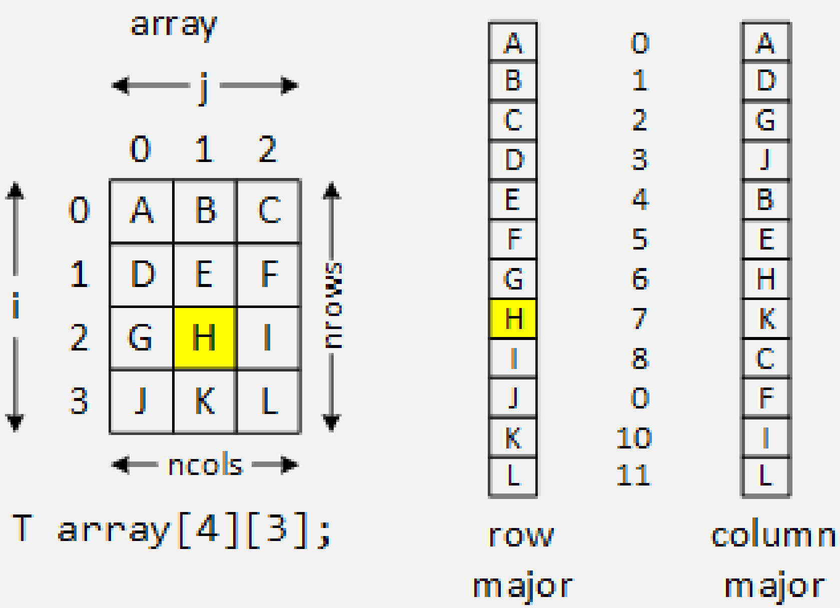
SYNTHESIZING A 2D ARRAY

```
inline int index(int row, int col, int ncols)
{
   return row * ncols + col;
}
```

```
int*   table = new int[nrows * ncols];
```

```
table[index(row, col, ncols)]
table[row * ncols + col]
```

TWO-DIMENSIONAL INITIALIZER LISTS AND ROW-MAJOR ORDERING



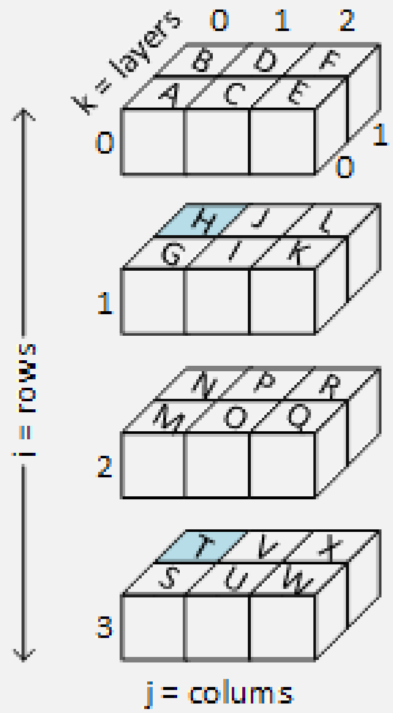
	0	1	2
0	A	B	C
1	D	E	F
2	G	H	I
3	J	K	L

```
T array[4][3];
```

row major		column major
A	0	A
B	1	D
C	2	G
D	3	J
E	4	B
F	5	E
G	6	H
H	7	K
I	8	C
J	0	F
K	10	I
L	11	L

```
char  array[4][3] = {
    'A', 'B', 'C', 'D', 'E', ‘F’,
    'G', 'H', 'I', 'J', 'K', 'L’
};
```

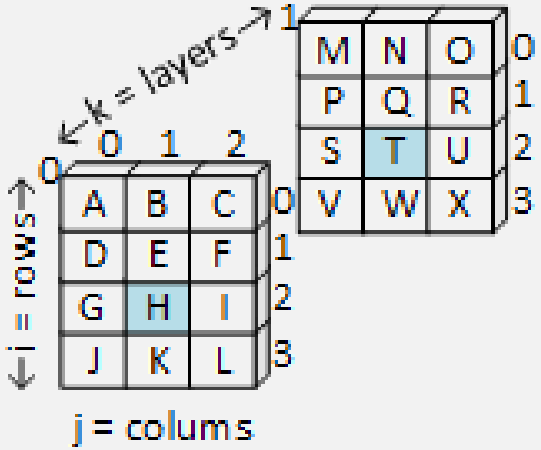
THREE-DIMENSIONAL INITIALIZATION ORDER



```
{
   'A', 'B', 'C', 'D', 'E', ‘F’,
   'G', 'H', 'I', 'J', 'K', 'L’
};
```

- `char  array[4][3][2]`
- `char*  array = new char[nrows * ncols * nlayr]`
- `inline int index(int i, int j, int k, int ncols, int nlayr)`
  `{ return k + nlayr * (j + ncols * i); }`
- `array[index(1, 0, 1, ncols, nlayr)]`
- `array[index(3, 0, 1, ncols, nlayr)]`

THREE-DIMENSIONAL
ROWS X COLUMNS X LAYERS ORDER



- `char  array[4][3][2]`
- `char*  array = new char[nrows * ncols * nlayr]`

-
  ```
  inline int index(int i, int j, int k, int ncols, int nlayr)
    { return nrows * ncols * k + (j + ncols * i); }
  ```
- `array[index(1, 0, 1, ncols, nlayr)]`
- `array[index(3, 0, 1, ncols, nlayr)]`