TWO-DIMENSIONAL ARRAYS

Specifying the size at runtime

Delroy A. Brinkerhoff

CREATING AND USING ARRAYS

WORKS

- int table[20][12];
- double* scores = new double[size];

DOESN'T WORK

- double* scores = new double[rows][cols];

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

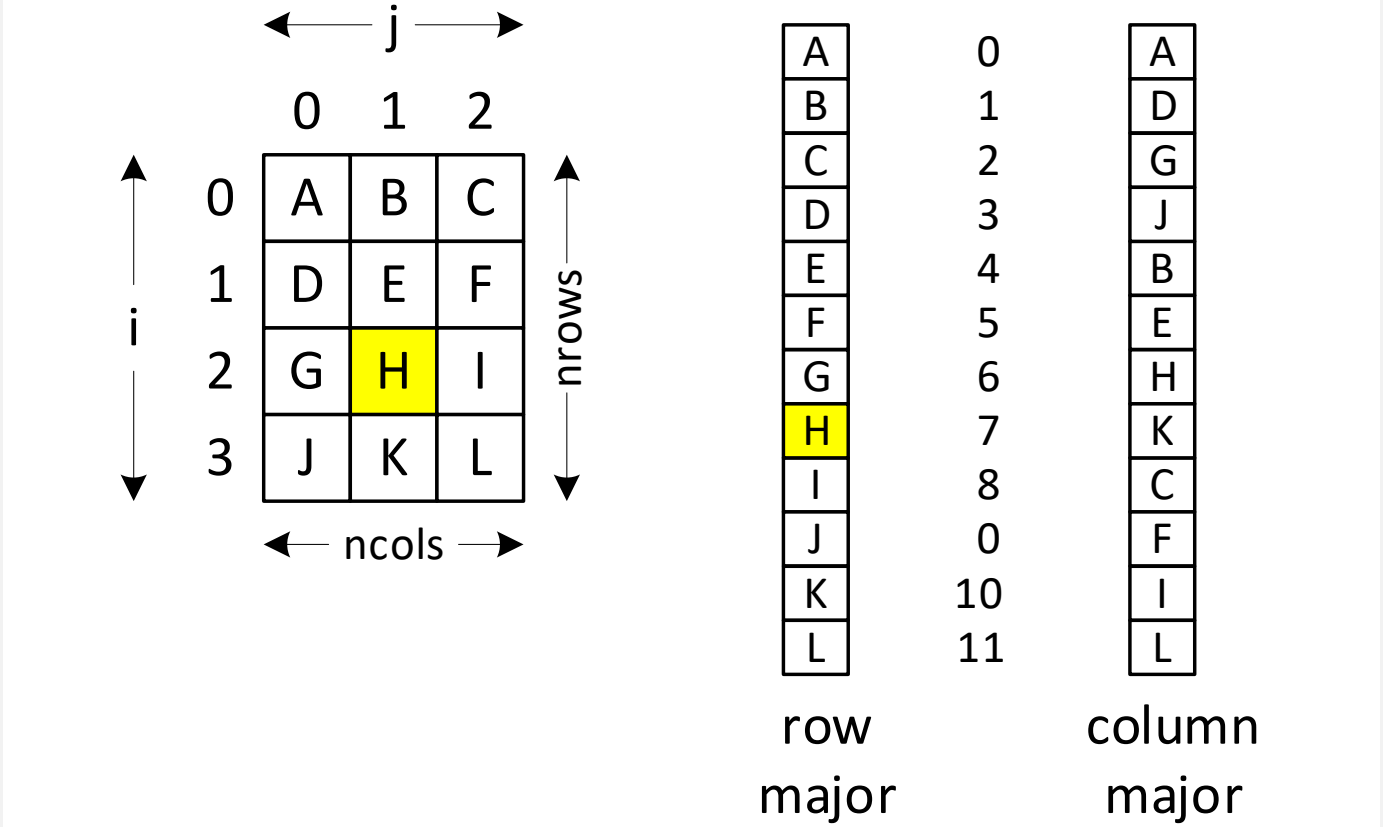
- void function(int table[ ][ ]);

PASSING TWO-DIMENSIONAL ARRAYS

```
char  a1[4][3] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L' };
char  a2[3][2] = { 'u', 'v', 'w', 'x', 'y', 'z' };
```

```
void print(char array[][3], int i, int j)
{
    cout << array[i][j] << endl;
}

void print(char array[][2], int i, int j)
{
    cout << array[i][j] << endl;
}
```

STORING 2D ARRAYS IN MEMORY



- Row-major mapping
  - i * ncols + j
- Example:
  - array[2][1]
  - 2 * 3 + 1 = 7

PROGRAMMER-IMPLEMENTED MAPPING

```
void print(char* array, int i, int j, int ncols)
{
    cout << array[i * ncols + j] << endl;
}
```

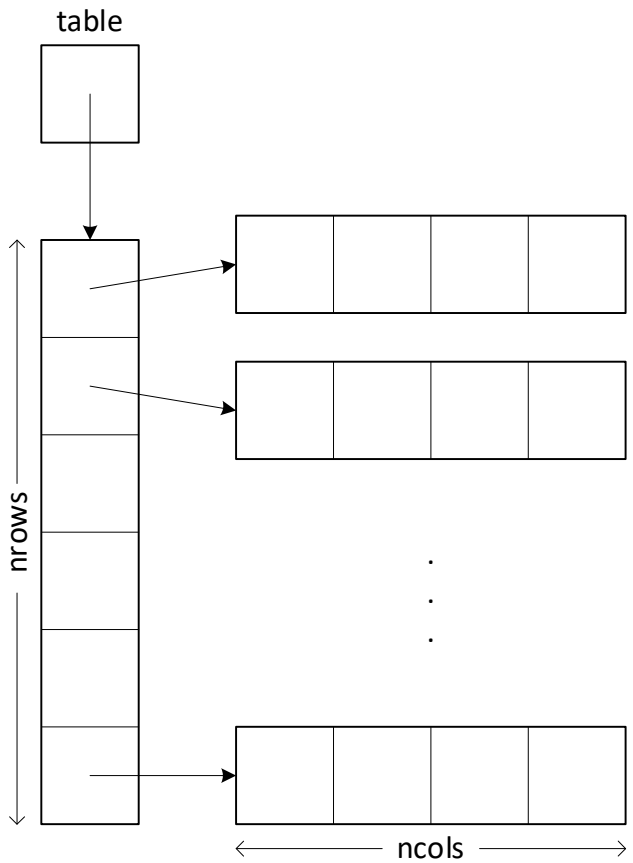
```
char a1[4][3] = { 'A', 'B', 'C', 'D', 'E', 'F',
                  'G', 'H', 'I', 'J', 'K', 'L' };
char a2[3][2] = { 'u', 'v', 'w', 'x', 'y', 'z' };

print((char *)a1, 2, 1, 3);
print((char *)a2, 1, 0, 2);
```

SYNTHESIZING A 2D ARRAY

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

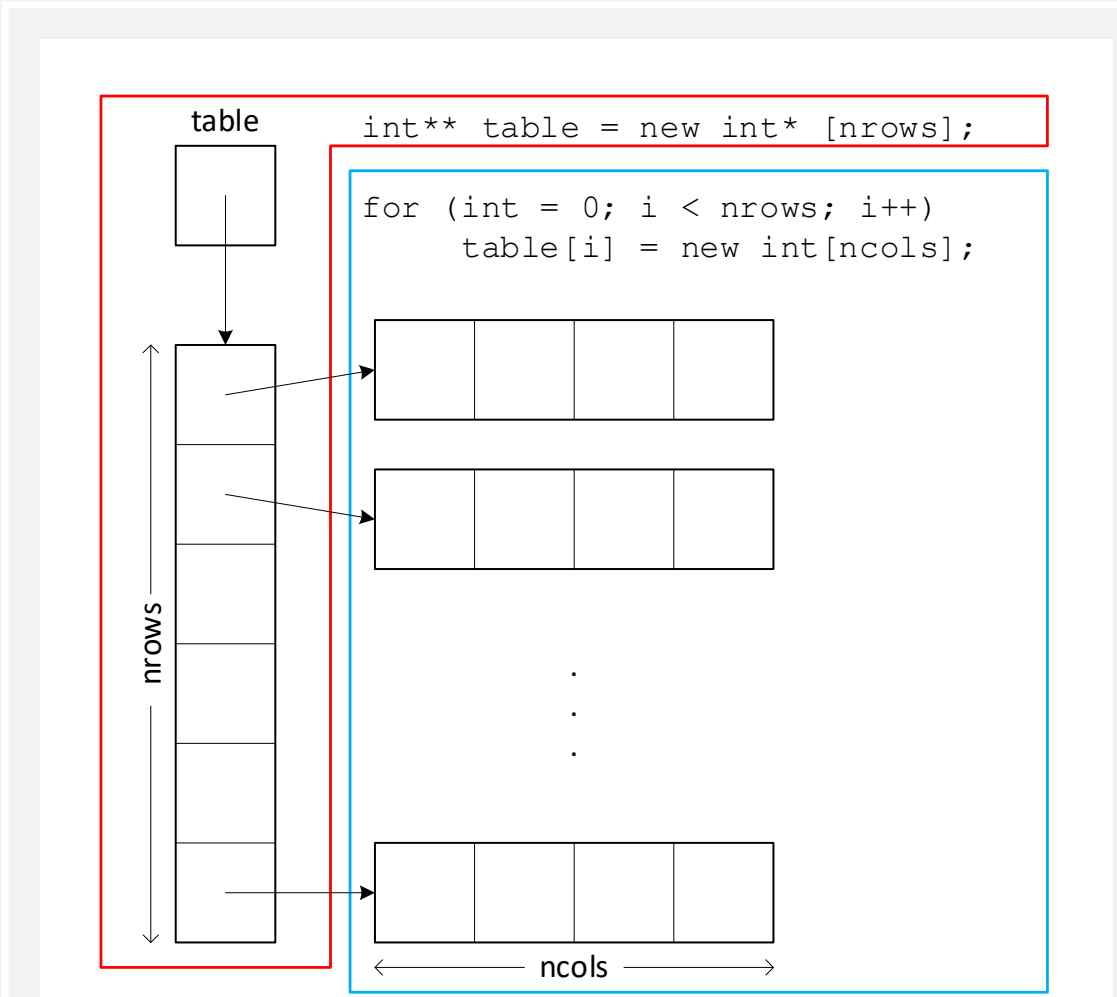
```
int*   table = new int[nrows * ncols];
```

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



CREATING A TWO-DIMENSIONAL ARRAY AS AN ARRAY OF ARRAYS

- Advantage: Element access uses a two-index notation
  - table[row][col]
- Disadvantages:
  - creating the array
  - destroying the array



CREATING & DESTROYING ARRAYS

```
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;
```