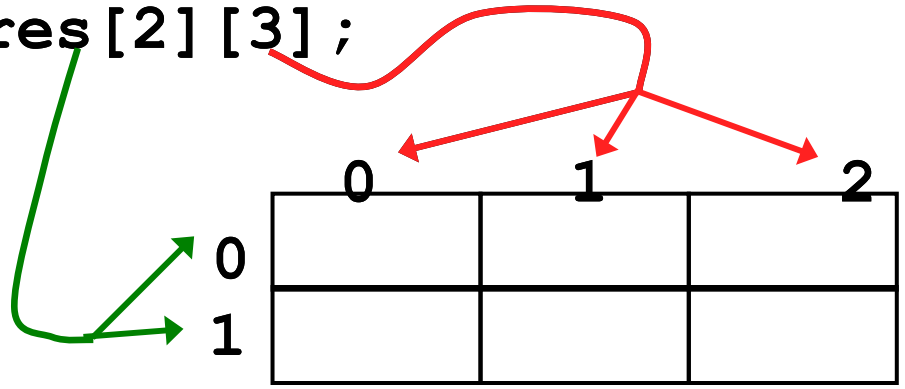
Multidimensional Arrays

Chapter 8

Two dimensional arrays (8.9)

- A grid of data!

```
int testScores[2][3];
```

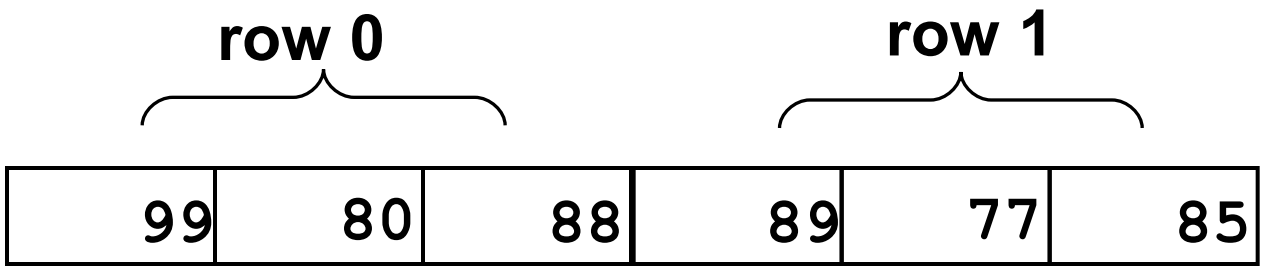



```
testScores[0][0] = 99;
testScores[0][1] = 80;
testScores[0][2] = 88;
testScores[1][0] = 89;
testScores[1][1] = 77;
testScores[1][2] = 85;
```

A 2D array in memory

- The 2D array in C++ is laid out by rows in memory

```
int testScores[2][3]= { {99, 80, 88},
                        {89, 77, 85}};
```



This is called row major order. Some languages use column major order.

More specifics about 2D arrays

- Creating and initializing 2D arrays

```
int vals1[2][3] = {{1, 2, 3}, {4, 5, 6}}; // OK

int vals2[][3] = {{1, 2, 3}, {4, 5, 6}}; // OK

int vals3[3][3] = {{1, 2, 3}, {4, 5, 6}}; // OK

int vals4[][] = {{1, 2, 3}, {4, 5, 6}}; // NOT OK
```

- Passing 2D arrays to a function

```
void fillarray (int vals[][3]);
```

- The number of columns MUST be specified in the function prototype while the number of rows is an optional specification.

Simple Problems

- Write a C++ program segment that creates a 2D array **temperatures** with 5 rows and 3 columns capable of storing temperatures.
- Write a C++ function **fillTemperatures** that accepts the array temperatures and allows the user the ability to enter values from the keyboard until the array is full.

More Practice

- Using the array below, calculate:
  - the average score on each assignment
  - the average score for each student
  - assume the array already contains data

```
const int NUM_STUDENTS = 100;
const int NUM_ASSIGNMENTS = 4;

int testScores[NUM_STUDENTS][NUM_ASSIGNMENTS];
```

N-Dimensional Arrays (8.10)

```
// cost of seats in a theatre
//
// 4 sections, each section has
// 20 rows with 30 seats each.

double seats[4][20][30];

seats[0][0][0] = 100.00;
seats[2][0][0] = seats[1][0][0] / 2;
seats[3][19][25] = 10.00;

// we can have as many dimensions as
// necessary in an array
```