

2D Arrays in C++

Instructor: Andy Abreu

Why 2D Arrays?

- One dimensional arrays are great, but why makes such a fuzz and create two dimensional arrays?
- What do we really gain from using a two dimensional array?
- What are some sensible uses of two dimensional arrays?

Declaring an Array

- Model:

type name[row_size][column_size]

- *type: The data type, example: int*

- *name: The name of the array, example: grades*

- *row_size: The row capacity of the array, example: 10*

- *column_size: The column capacity of the array, ex. 5*

- `int grades[22][6];`

- `string students[2][22];`

Understanding parts of a 2D array

- Say we have the following array:

```
int grades[4][8];
```

- Here is the graphical representation:

grades[0]	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]	[0][6]	[0][7]
grades[1]	[1][0]							
grades[2]	[2][0]							
grades[3]	[3][0]							[3][7]

Understanding parts of a 2D array

- Same array:
 - `int grades[4][8];`
- In English the *grades* variable is describe as an **array of array of integers**
- While *grades[0]*, *grades[1]*, ... *grades[3]* are **array of integers**
- And *grades[0][0]* is simply an **integer**

Accessing Elements in the 2d Array

- If we have an 2D array declare as the following:
 - `int grades[5][10];`
- The elements of the array are as follows:
 - `grades[0]`
 - `grades[1]`
 - `grades[2]`
 - `grades[3]`
 - `grades[4]`
- Each “element” represent an array of 10 elements

Accessing Elements in the 2d Array

- We can assign values to **grades[0]** as follows:
 - `grades[0][0] = 89;`
 - `grades[0][1] = 93;`
 - `grades[0][2] = 85;`
 - `grades[0][3] = 88;`
 - `grades[0][4] = 100;`
 - `grades[0][5] = 89;`
 - `grades[0][6] = 83;`
 - `grades[0][7] = 85;`
 - `grades[0][8] = 78;`
 - `grades[0][9] = 99;`
- Likewise for `grades[1]`, `grades[2]`, `grades[3]`, `grades[4]`

Printing elements of the 2D array

- So can we print out all the grades in the following manner?
 - `cout << grades[0] << endl;`
 - `cout << grades[1] << endl;`
 - `cout << grades[2] << endl;`
 - `cout << grades[3] << endl;`
 - `cout << grades[4] << endl;`
- Why or why not?

Printing an element of a 2D array

- We can use a for loop to printing out elements of the array grades[0]

- Code would look like this:

```
for ( int i = 0 ; i < 10 ; ++i )  
    cout << grades[0][i] << " ";  
cout << endl;
```

Printing Entire 2D array

- So if we need an array to print out elements of grades[0], then naturally to print out all the grades[x] we will need to employ a second loop.

Code looks like this:

```
for( int r = 0 ; r < 5 ; ++r )  
{  
    for( int c = 0 ; c < 10 ; ++c )  
        cout << grades[r][c] << " ";  
    cout << endl;  
}
```

Initializing the 2D array

- Sometimes we want to pre-initialize the array, we can do the following:
 - `int lookup[3][2] = { {97, 93}, {87, 83}, {77, 73} };`
- Sometimes we want to initialize the entire array to zero, we can do the following:
 - `int sums[5][10] = {0};`
 - `{0}` is a special code to C++, `{1}` doesn't work.

2D Arrays and Functions

2D Arrays and Functions

- Like regular arrays, two dimensional arrays can be pass into sub functions, and they are always pass by reference.
- It is important to note:
 - If the function is trying to access the entire 2D array or
 - An element of the 2D array, the 1D array.

Example of passing 2D array

- To pass entire 2D array into the function
 - `int gradesSet[10][20];`
 - `printAllScore(gradesSet);`
 - `void printAllScore(int gradesSet[][20], int row, int col)`
- The **COLUMN SIZE** of the 2D array **MUST** be **provided** while row size is optional.

Passing one element of 2D array

- To pass 1 element of the 2D array into the function
 - `int gradesSet[10][20];`
 - `printRowScore(gradesSet[0]);`
 - `printRowScore(gradesSet[1]);`
 - `printRowScore(gradesSet[2]);`
 - `printRowScore(gradesSet[3]);`
 - `printRowScore(gradesSet[4]);`
 - `void printRowScore(int grades[], int col)`