# Topic 2 - Arrays

Chapter 9 in Malik

Multi-Dimensional Arrays

## Multi-Dimensional Arrays

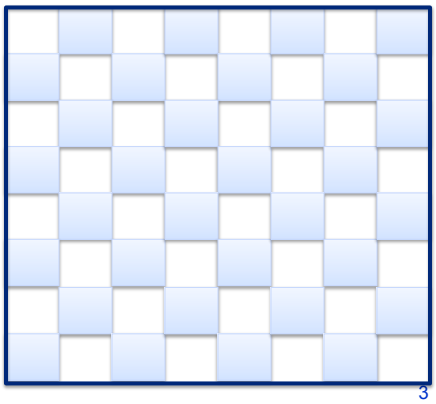
- So far, we've discussed one-dimensional arrays
- It is possible to have arrays of more than 1 dimension → multi-dimensional arrays
  - There is one subscript for each dimension
    - A 2-dimensional array has 2 subscripts
    - A 3-dimensional array has 3 subscripts … and so on
  - Think of them as an array of arrays
  - For example, an array of c-strings
    - c-strings are an array

# Example - Chessboard

- Creating a chessboard program
  - You want to track the pieces
- We could do a one dimensional array

  int board[64];
- A two-dimensional array would make more sense for this application

  int board[8][8];
- A 2-dimensional array for this application better corresponds to the real-world application





- This way we could think of the array in terms of (x, y) pairs.

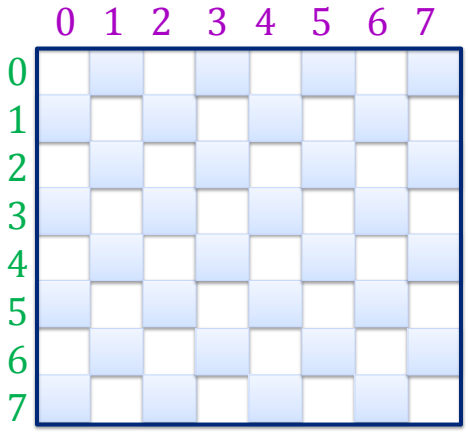
  x could represent the rows and

  y could represent the columns

int board[8][8];

# Of Rows

# Of Columns



# Chessboard Example

int b [8][8];

Rows Cols

This way we can reference our array with respect to the rows and columns

So we can say b[1][1] instead of b[9]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | b[0][0] | b[0][1] | b[0][2] | b[0][3] | b[0][4] | b[0][5] | b[0][6] | b[0][7] |
| 1 | b[1][0] | b[1][1] | | | | | | |
| 2 | b[2][0] | | | | | | | |
| 3 | b[3][0] | | | | | | | |
| 4 | b[4][0] | | | | | | | |
| 5 | b[5][0] | | | | | | | |
| 6 | b[6][0] | | | | | | | |
| 7 | b[7][0] | | b[7][2] | | b[7][4] | | | b[7][7] |

this is how we would access row 2, col 2



# Example 2 – Scores

- Let's say you are tracking a group of scores from different people
- Let's say we are tracking 2 people with 3 items to score

We declare our array like this → int scoresAr[2][3];

1st Person

scoresAr[0][0] = 75

scoresAr[0][1] = 65

scoresAr[0][2] = 95

2nd Person

scoresAr[1][0] = 45

scoresAr[1][1] = 85

scoresAr[1][2] = 100

| | 1st Score | 2nd Score | 3rd Score |
|---|---|---|---|
| 1st person | 75 | 65 | 95 |
| 2nd person | 45 | 85 | 100 |

# Initializing Multidimensional Arrays

- We initialize multidimensional arrays a little differently

```
int scoresAr[2][3] = { 75, 65, 95, 45, 85, 100 };
⇔ int scoresAr[2][3] = {  { 75, 65,  95 },
                          { 45, 85, 100 }   };
```

- Although these are equivalent the 2nd is easier to read
  - The compiler ignores the extra brackets, but needs the commas
- Or we can initialize all values to 0 like this:

```
int scoresAr[2][3] = {0};
```

Again.. We should use constants where we can:

```
const int TOTAL_PLAYERS = 2;
const int TOTAL_SCORES  = 3;

int scoresAr[TOTAL_PLAYERS][TOTAL_SCORES] = {0};
```

**Generally speaking we should always initialize arrays**



# Using For loops

```
int scoresAr[2][3];

for (int i = 0; i < 2; i++)
{
   cout << "Enter scores for player #" << i + 1 << " : ";
   for (int j = 0; j < 3; j++)
   {
      cout << "Enter score #" << j + 1 << " : ";
      cin   >> scoresAr [i][j];
   }
}
```

# Using For loops.

What does this do?
Do a desk check for the array
Containing
{50, 70, 90},
{60,80,100}

```
const int TOTAL_PLAYERS = 2;
const int TOTAL_SCORES = 3;

int scoresAr[TOTAL_PLAYERS][TOTAL_SCORES] = {0};
int sum, player, score;
float avg;

for (player = 0; player < TOTAL_PLAYERS;  player++)
{
   sum = 0;
   for (score = 0; score < TOTAL_SCORES; score++)
   {
     sum = sum + scoresAr[player][score];
   }
   avg = sum / TOTAL_SCORES;
   cout << "Average for player #" << player + 1 << " = " << avg << endl;
}
```



# Initializing an array of characters

```
char alphaAr[NUM_ROWS][NUM_COLS];
int rowCnt;
int colCnt;

for (rowCnt = 0; rowCnt < NUM_ROWS; rowCnt++)
{
   for (colCnt = 0; colCnt < NUM_COLS; colCnt++)
   {
        alphaAr[rowCnt][colCnt]=' ';
   }
}
```

# Passing 2-D arrays as Parameters

```
int scoresAr[TOTAL_PLAYERS][TOTAL_SCORES] = {0};
float avg;
int player, score;

for (player = 0; player < TOTAL_PLAYERS; player++)
{
    cout << "Average for player #" << player + 1 << " = ";
    cout << AverageArray(scoresAr, i);
}

float AverageArray(int arrayValues[][TOTAL_SCORES], int player)
{
    int sum;
    sum=0;
    for (score = 0; score < TOTAL_SCORES;  score++)
        sum += arrayValues[player][score];

    return (sum / NumVals);
}
```

How should TOTAL_SCORES be declared?

It should be passed by Const reference

You do not need to specify the 1st dimension. You do need to specify the 2nd dimension

C++ doesn't need to know how many rows, just the size of each row.

# Exercise

Write a function for a tic tac toe game. To determine which spot the user wants to play in they must type in the row, column. The function should obtain the input and verify that the row & column # are within range AND that the spot is not taken.

Assume that all elements in the array were initialized to a blank space. The array should be 3 x 3 2-dimensional array of type char.

```
const int NUM_ROWS = 3;
const int NUM_COLS  = 3;

char boardAr[NUM_ROWS][NUM_COLS];
```

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | [1][1] | [1][2] | [1][3] |
| 2 | [2][1] | [2][2] | [2][3] |
| 3 | [3][1] | [3][2] | [3][3] |

The following parameters will be used for the function.

- char boardAr[][NUM_COLS]
- char token

```
void GetAndCheckInp(char boardAr[][NUM_COLS], char token)
{
    // include the appropriate declaration section here
    do
    {

    }
```



# Get and Check Input - Modified

- Now, modify the previous code segment to obtain two players names and prompt the user by name
- Let's associate playerX with token 'X' and playerO with token 'O'

```
void GetAndCheckInp(char boardAr[][NUM_COLS],char token, string playerX, string playerO)
{
    // include the appropriate declaration section here
      do
      {



      }
      while (!valid);

}
```