We are 183

L11: Week 7 - Monday

Reminders

- Exam 1 – tomorrow!
  - See the “About Exam 1” on the course website

Locations

If your UNIQNAME starts with…		go at 5:50 p.m. on Tue 2/16 to…
aa-ds	inclusive	Modern Languages Building 1400
dt-jj	inclusive	Chemistry 1210
jk-mw	inclusive	Chemistry 1400
mx-zz	inclusive	Chemistry 1800

- Project 3 out now!
  - Due February 26th

Last Time… on EECS 183

1D Arrays

declaration

initialization

with functions

Array Declaration

identifier

```
data    ;
```

```
// Allocate 5 consecutive
// memory locations suitable
// for storing double values
```

Array Declaration

(# of elements)
size

```
data[5];
```

```
// Allocate 5 consecutive
// memory locations suitable
// for storing double values
```

Array Declaration

type

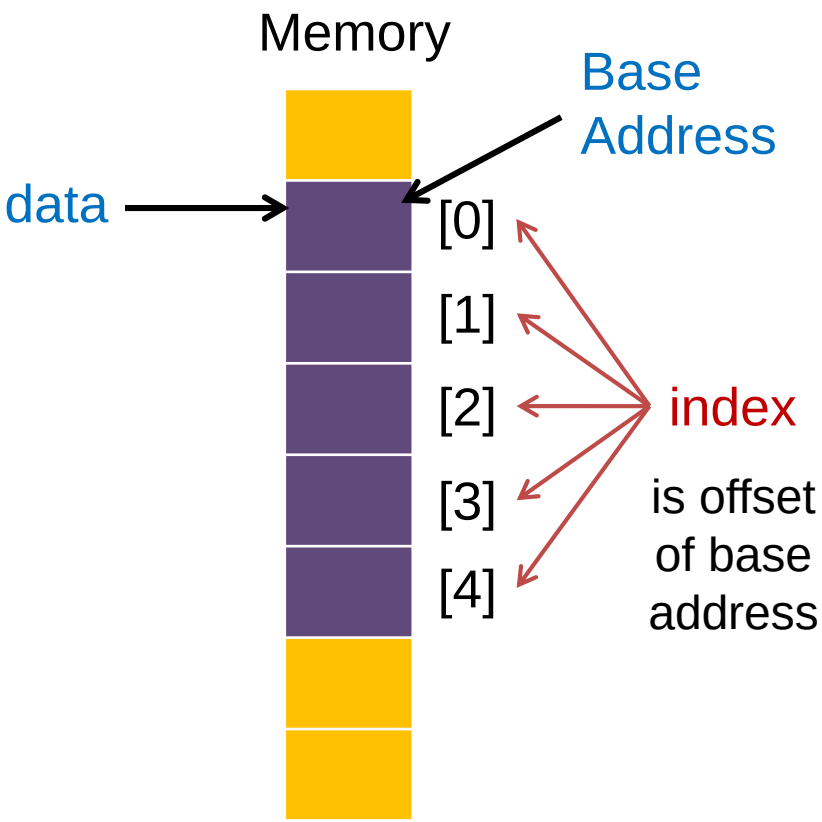
```
double data[5];

// Allocate 5 consecutive
// memory locations suitable
// for storing double values
```

Arrays in Memory

```
double data[5];

// Allocate 5 consecutive
// memory locations suitable
// for storing double values
```



Static Allocation

Array size MUST be

- known at compile time
- a constant integer value

```
int data[5];
```

```
const int SIZE = 5;
int data[SIZE];
```

```
int size = 5;
int data[size];
```

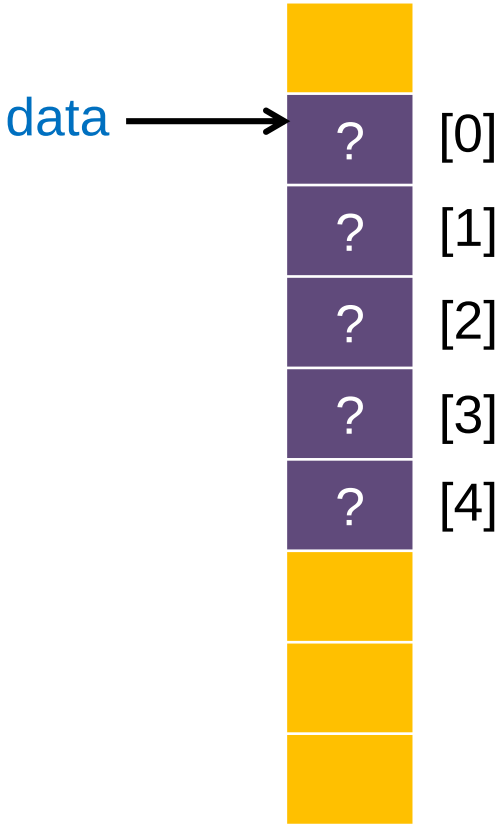
Compile error

Cannot use a variable

Array Initialization

```
int data[5];
```

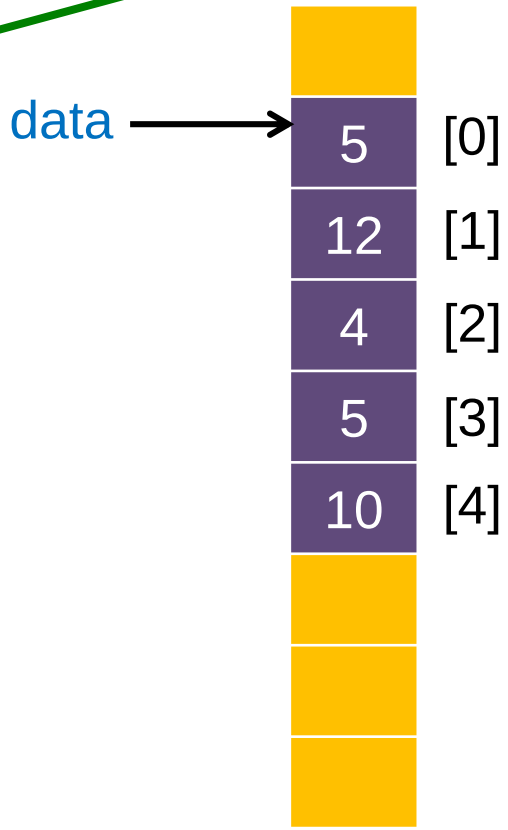
No initialization means the array contains **random** values.



Array Initialization

```
int data[5] = {5, 12, 4, 5, 10};
```

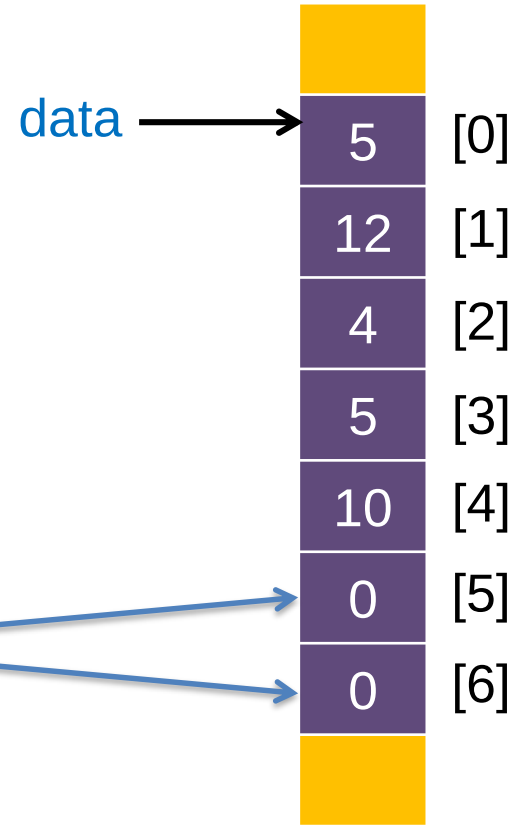
Braces can be used to initialize array values.



Array Initialization

```
int data[7] = {5, 12, 4, 5, 10};
```

When the **size** is **greater** than the number of **initial values**



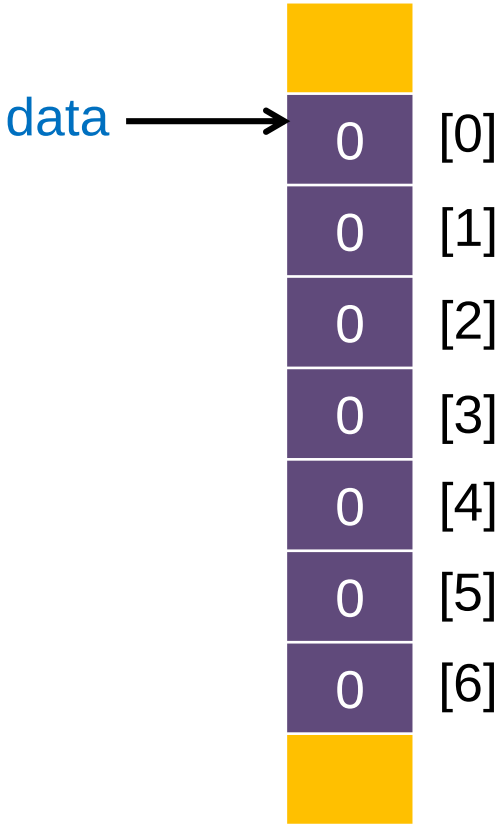
Compilers will insert 0's

Array Initialization

```
int data[7] = { };
```

When there are no values between the brackets

Compilers will fill 0's



i>Clicker #1

```
int main(void) {
    char message[5] = {'H'};
}
```

What is the value of `message[3]`?

A. `'\0'`
B. `'0'`
C. `'H'`
D. error
E. None of the above

i>Clicker #1

```
int main(void) {
    char message[5] = {'H'};
}
```

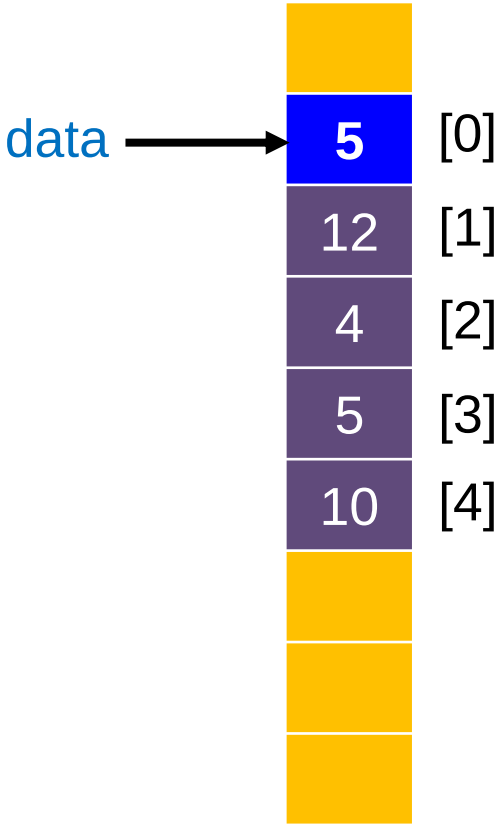
What is the value of `message[3]`?

A. `'\0'` -- ASCII 0
B. `'0'`
C. `'H'`
D. error
E. None of the above

Accessing array elements

```
int data[5] = {5, 12, 4, 5, 10};
cout << data[0];
// prints 5
```

Bracket access, just like a string!



Accessing array elements

Execution

```
int data [] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```

Accessing array elements

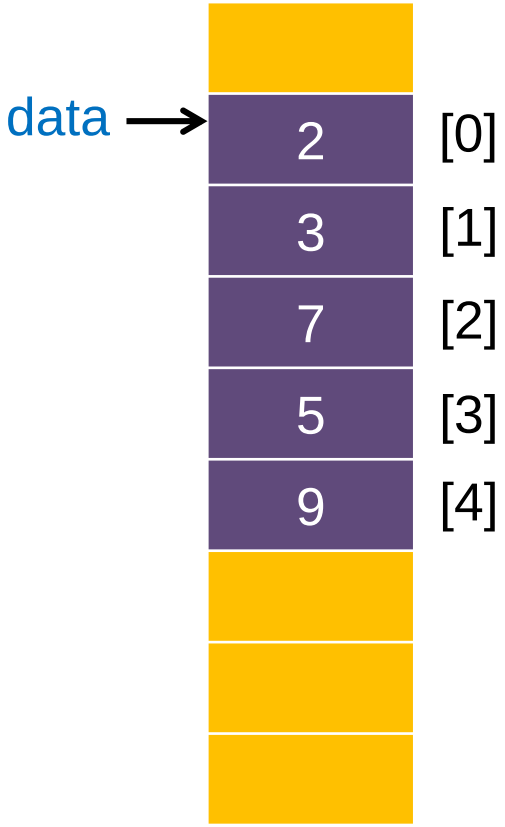
Execution

```
int data [] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```



Accessing array elements

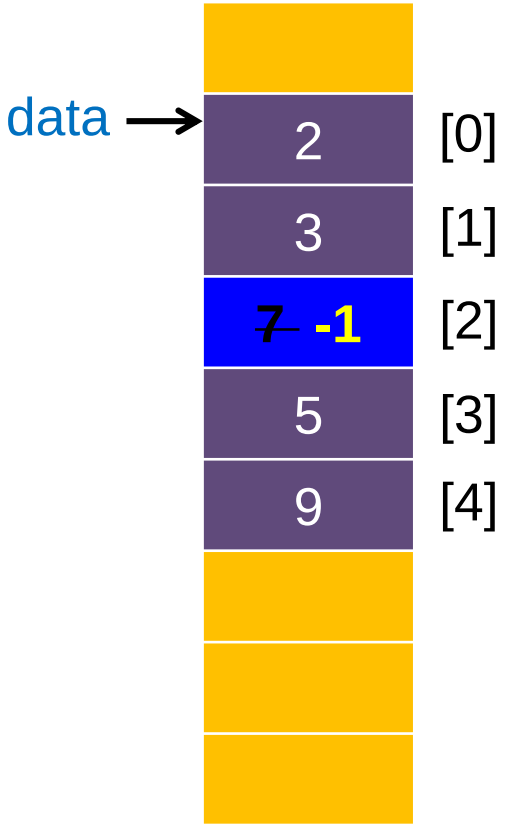
```
int data [] = {2, 3, 7, 5, 9};
```

Execution

```
data[2] = -1;

data[3] = data[4];

data[1]++;
```



Accessing array elements

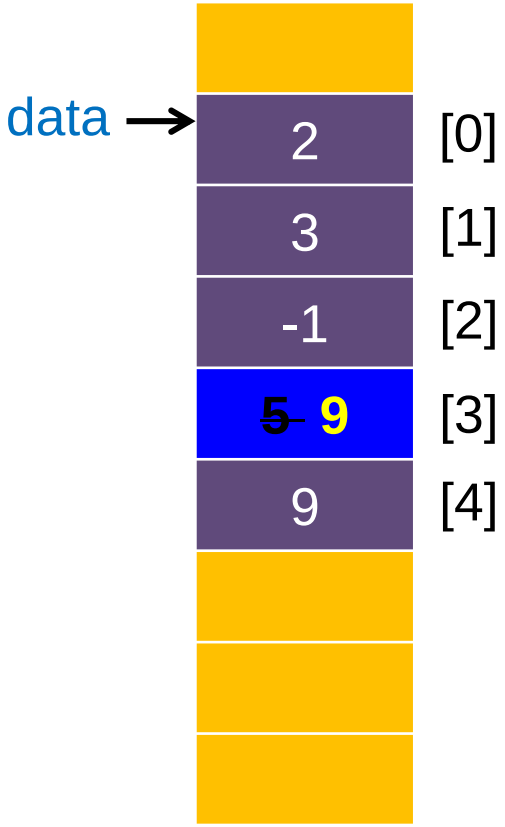
```
int data [] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```

Execution



Accessing array elements

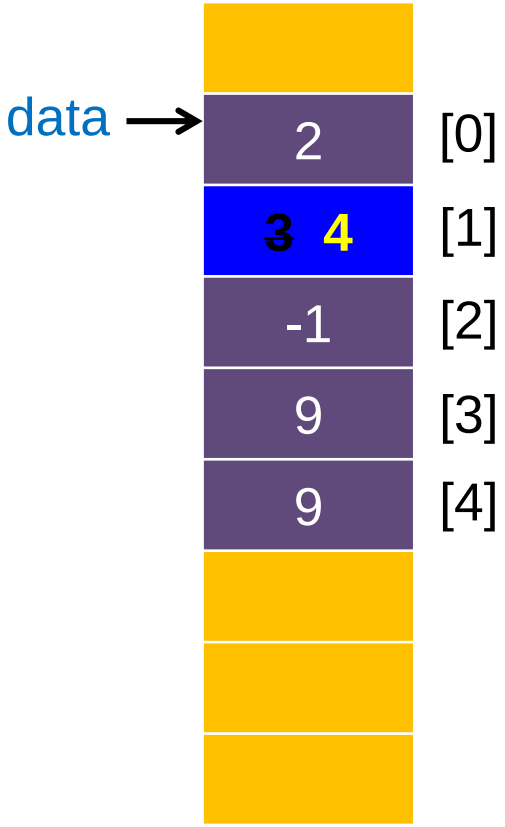
```
int data [] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```

Execution

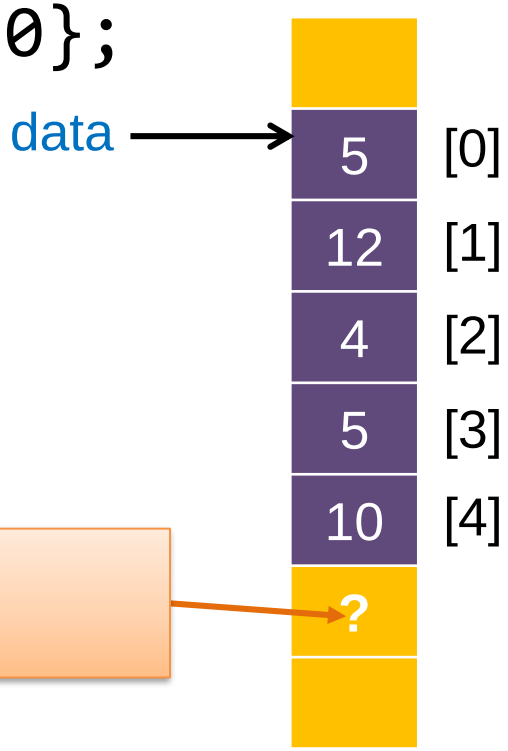


Common Errors: Out-of-Range

- C++ does **NOT** check array indices for <u>validity</u>

```
int data[5] = {5, 12, 4, 5, 10};

cout << data[5];
```



prints whatever random value happens to be stored here

Common Errors: Operating on Arrays

- I/O errors

```
const int SIZE = 10;

int data[SIZE];

cin >> data;
```

Common Errors: Operating on Arrays

- Assignment errors

```
const int SIZE = 10;

int arr1[SIZE], arr2[SIZE];

...

arr2 = arr1;
```

Compile Error

Common Errors: Operating on Arrays

- Arithmetic Operations on Arrays

```
const int SIZE = 10;

int arr1[SIZE], arr2[SIZE];

arr1 = arr2 + 6;
```

Compile Error

Common Errors: Operating on Arrays

- Comparing Elements of Arrays

```
const int SIZE = 10;

int arr1[SIZE], arr2[SIZE];

if (arr1 == arr2) {
    // Do something
}
```

Bad Idea

Compares ***addresses*** of **arr1** and **arr2**

Common Errors: Operating on Arrays

- Comparing Elements of Arrays

```
const int SIZE = 10;

int arr1[SIZE], arr2[SIZE];

if (arr1 < arr2) {
    // Do something
}
```

Bad Idea

Compares ***addresses*** of **arr1** and **arr2**

i>Clicker #2

```
int main(void) {
    int list1[5] = {1, 2, 3, 4, 5};
    int list2[5] = {5, 4, 3, 2, 1};

    if (list1[3] == list2[3]) {
        cout << "equal";
    } else if (list1[3] > list2[3]) {
        cout << "greater";
    } else {
        cout << "lesser";
    }
}
```

What prints?

A. equal
B. greater
C. lesser
D. nothing
E. error

i>Clicker #2

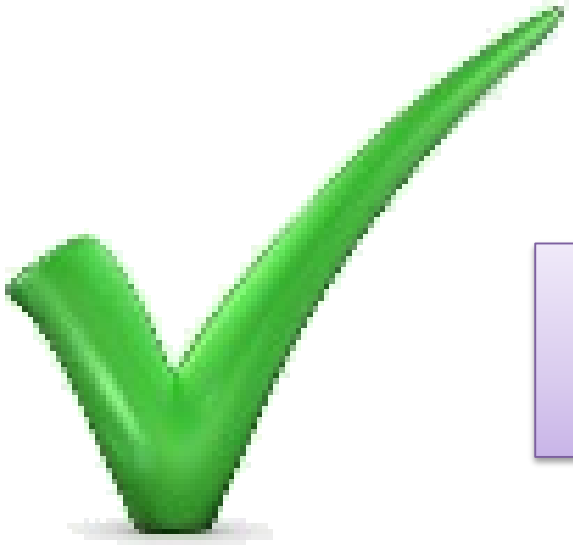
```
int main(void) {
    int list1[5] = {1, 2, 3, 4, 5};
    int list2[5] = {5, 4, 3, 2, 1};

    if (list1[3] == list2[3]) {
        cout << "equal";
    } else if (list1[3] > list2[3]) {
        cout << "greater";
    } else {
        cout << "lesser";
    }
}
```

What prints?

A. equal
B. greater
C. lesser
D. nothing
E. error

Working with arrays…



Use a loop!

It's best to deal with array elements **individually**.

Today

Passing Arrays to Functions

Multi-dimensional Arrays

declaration

initialization

with functions

Passing Arrays to Functions

Example: Print Array

- Write a function that accepts an array of integers and prints the content of the array

- Prototype:

```
printArray(                    );
```

Example: Print Array

- Write a function that accepts an array of integers and prints the content of the array

- Prototype:

```
void printArray(int arr[],  2  );
```

array

array size

Example: Print Array

- Write a function that accepts an array of integers and prints the content of the array

- Prototype:

```
void printArray(int arr[], int size);
```

Always passed by reference.
No **&** needed

Example: Print Array

- Write a function that accepts an array of integers and prints the content of the array

- Prototype:

```
void printArray(int arr[], int size);
```

Pass the size so you won't go out-of-range.

Example: Print Array

- Write a function that accepts an array of integers and prints the content of the array

- Prototype:

```
void printArray(int arr[], int size);
```

Array modified directly.
Nothing to return.

Example: Print Array

Test:

```
int main() {
    // test printArray
    const int SIZE = 5;
    int data[SIZE] = {1, 2, 3, 4, 5};
    printArray(data, SIZE);
}
```

Example: Print Array

Implementation:

```
void printArray(int arr[], int size) {

}
```

Example: Print Array

Implementation:

```
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << endl;
    }
}
```

Passing Arrays to functions

C++ arrays are:

- **Always** passed by **reference**
  - No **&** needed

- Saves memory space
  - array is not copied

- Saves processing time
  - array elements are not copied

i>Clicker #3

```
const int SIZE = 5;

void increment(int a[], int size) {
    for (int i = 0; i < size; i++) {
        a[i] = a[i] + 1;
    }
}

int main() {
    int a[SIZE] = {1, 2, 3, 4, 5};
    cout << a[1] << " ";
    increment(a, SIZE);
    cout << a[1];
}
```

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

i>Clicker #3

```
const int SIZE = 5;

void increment(int a[], int
    for (int i = 0; i < size; i++) {
        a[i] = a[i] + 1;
    }
}

int main() {
    int a[SIZE] = {1, 2, 3, 4, 5};
    cout << a[1] << " ";
    increment(a, SIZE);
    cout << a[1];
}
```

Arrays are **ALWAYS** passed by reference

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

Example: Square Array Elements

- Write a function that accepts an array of integers and squares all the elements of the array.

```
void squareArray(int arr[], int size);
```

Arrays are passed by reference – modified directly

Example: Square Array Elements

- Write a function that accepts an array of integers and squares all the elements of the array.

```
int[] squareArray(int arr[], int size);
```

Compile Error – Cannot return an array

Example: Square Array Elements

```
int main() {
    // test squareArray
    const int SIZE = 5;
    int data[SIZE] = {1, 2, 3, 4, 5}
    squareArray(data, SIZE);
    // array contents become 1, 4, 9, 16, 25
}
```

Example: Square Array Elements

```
void squareArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = arr[i] * arr[i];
    }
}
```

the Name of an Array ...

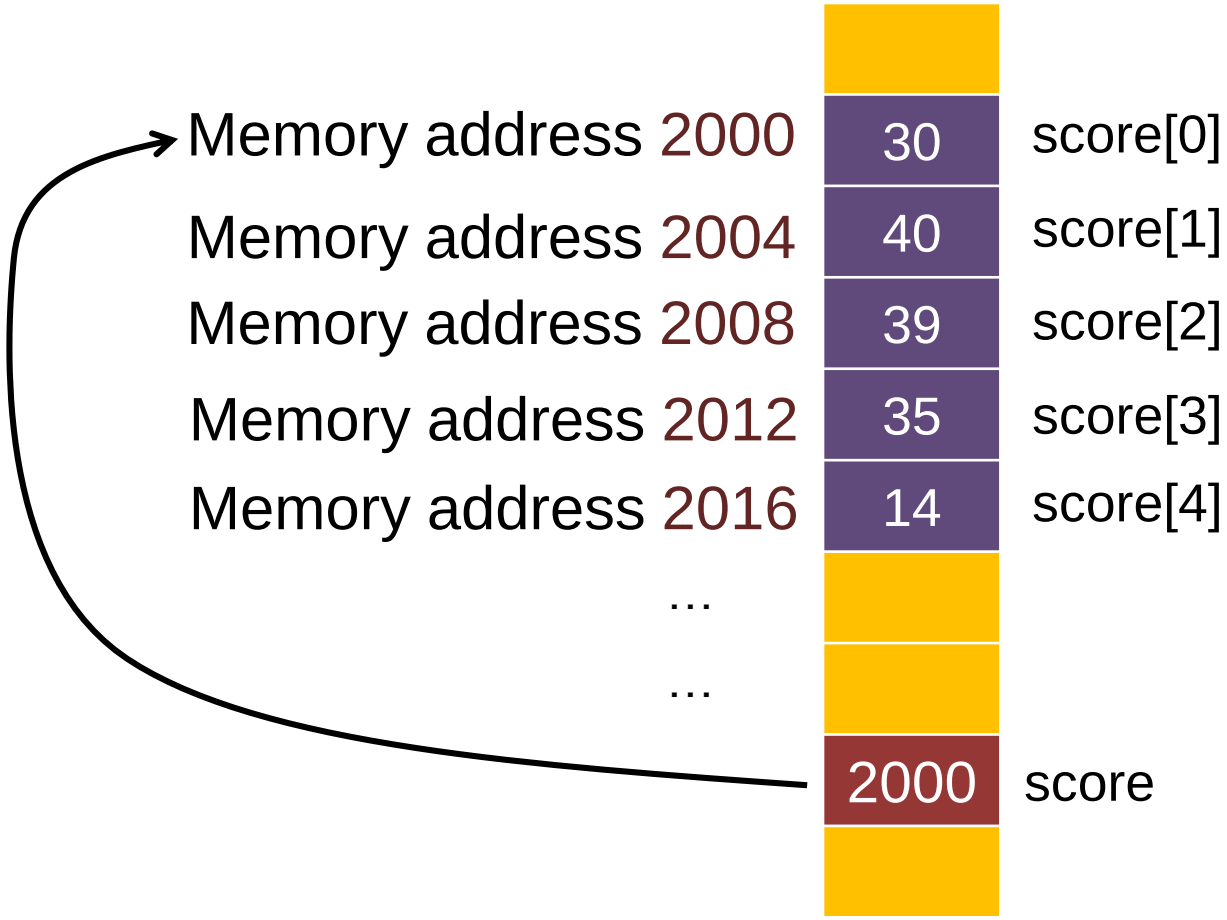
- Refers to the ***entire*** data structure

```
int score[5] = {30, 40, 39, 35, 14};
```

- **score** holds the memory address of the 1st array element (`score[0]`)

the Name of an Array ...

```
int score[5] = {30, 40, 39, 35, 14};
```



Prevent function from modifying an array – Similar to pass-by-value

```
// allows function to alter data
// all arrays are passed by reference
void printResults(int data[ ], int size);


// With const, function can't alter array
void printResults(const int data[ ], int size);
```

i>Clicker #4

```
void increment(int x) {
    x = x + 1;
}

int main() {
    const int SIZE = 3;
    int arr[SIZE] = {1, 2, 3};
    cout << arr[1] << " ";
    increment(arr[1]);
    cout << arr[1];
    return 0;
}
```

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

i>Clicker #4

```
void increment(int x) {
    x = x + 1;
}

int main() {
    const int SIZE = 3;
    int arr[SIZE] = {1, 2, 3};
    cout << arr[1] << " ";
    increment(arr[1]);
    cout << arr[1];
    return 0;
}
```

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

i>Clicker #4

```
void increment(int x) {
    x = x + 1;
}

int main() {
    const int SIZE = 3;
    int arr[SIZE] = {1, 2, 3};
    cout << arr[1] << " ";
    increment(arr[1]);
    cout << arr[1];
    return 0;
}
```

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

A single array element is ***also*** passed by value

Return from a function

```
const int SIZE = 5;
void foo(int arr[], int size);

int main() {
    int arr1[SIZE];
    foo(arr1, SIZE);
    // More code…
}
```

Return from a function

```
const int SIZE = 5;
void foo(int arr[], int size);

int main() {
    int arr1[SIZE];
    foo(arr1, SIZE);
    // More code…
}
```

Passed by **reference**
no need to "return" the array

Calls & Prototypes

Example Call (assume main is caller):

```
readScoreList(data, SIZE);
```

Possible Corresponding Prototypes:

```
void readScoreList(int data[SIZE], int size);
void readScoreList(int data[ ], int size);
void readScoreList(int * data, int size);
```

i>Clicker #5

What should be the prototype/signature of a function that gives back the absolute value of every element in an array?

```
A.  void arr_abs(int arr[], int size);
B.  void arr_abs(int arr[]);
C.  int[] arr_abs(int arr[], int size);
D.  int[] arr_abus(int arr[]);
```

i>Clicker #5

What should be the prototype/signature of a function that gives back the absolute value of every element in an array?

```
A.  void arr_abs(int arr[], int size);
B.  void arr_abs(int arr[]);
C.  int[] arr_abs(int arr[], int size);
D.  int[] arr_abus(int arr[]);
```

i>Clicker #6

```
void foo(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i]++;
    }
}

int main(void) {
    int arr[] = {0, 1, 2, 3};
    foo(arr, 3);
}
```

What are the final contents of `arr`?

A. {0, 1, 2, 3}
B. {1, 2, 3, 4}
C. {1, 2, 3, 3}
D. {0, 2, 3, 4}
E. None of the above

i>Clicker #6

```
void foo(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i]++;
    }
}

int main(void) {
    int arr[] = {0, 1, 2, 3};
    foo(arr, 3);
}
```

What are the final contents of arr?

A. {0, 1, 2, 3}
B. {1, 2, 3, 4}
C. {1, 2, 3, 3}
D. {0, 2, 3, 4}
E. None of the above

Multi-dimensional Arrays

declaration

initialization

with functions

(One-Dimensional) Arrays

- Basic array is one-dimensional

```
int arr[7] = {42, 10, -7, 15, -22, 10, 3};
```

42	10	-7	15	-22	10	3
[0]	[1]	[2]	[3]	[4]	[5]	[6]

(One-Dimensional) Arrays

- Basic array is one-dimensional

```
char chs[9] = {'Y','O','U',' ','R','O','C','K','!'};
```

Y	O	U		R	O	C	K	!
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

Two Dimensional Arrays

#columns

```
int arr[4][3];

...

...
```

#rows

	[0]	[1]	[2]
[0]	-15	12	13
[1]	12	21	4
[2]	2	-4	3
[3]	-15	23	11

Two Dimensional Arrays

- Used to implement table data

- Example uses:
  - Matrix
  - Image
  - Spreadsheet
  - Game board

Two Dimensional Arrays

- Used to implement table data

- Example uses:
  - Matrix
  - Image
  - Spreadsheet
  - Game board

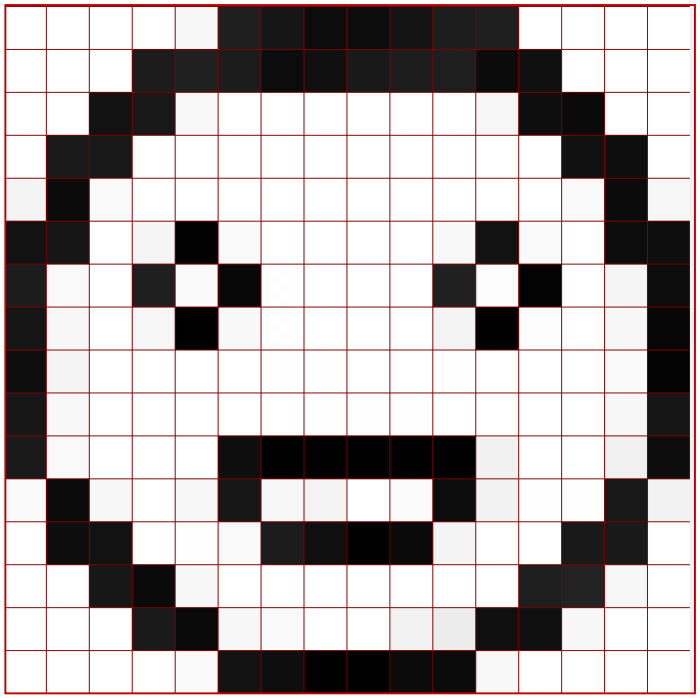
$$\begin{array}{cc} & \begin{array}{cc} [0] & [1] \end{array} \\ \begin{array}{c} [0] \\ [1] \\ [2] \end{array} & \begin{bmatrix} 10 & 2 \\ 1 & -4 \\ -5 & 3 \end{bmatrix} \end{array}$$

```
int matrix[3][2];
```

Two Dimensional Arrays

- Used to implement table data

- Example uses:
  - Matrix
  - Image
  - Spreadsheet
  - Game board



0 = black
1 = white

```
int myImage[16][16];
```

Two Dimensional Arrays

- Used to implement table data

- Example uses:
  - Matrix
  - Image
  - Spreadsheet
  - Game board

	Exam 1	Exam 2	Exam 3
	[0]	[1]	[2]
[0]	24	22	49
[1]	15	14	33
[2]	13	17	29
[3]	21	13	37
:	:	:	:
[49]	16	20	31

```
const int NUM_STUDENTS = 50;
const int NUM_EXAMS = 3;
int grades[NUM_STUDENTS][NUM_EXAMS];
```

Two Dimensional Arrays

- Used to implement table data

- Example uses:
  - Matrix
  - Image
  - Spreadsheet
  - Game board

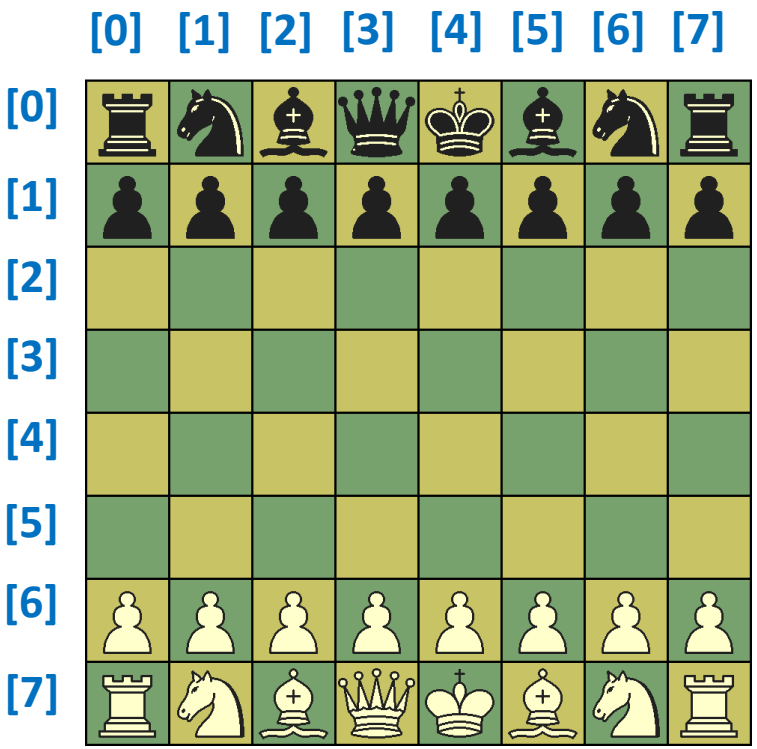
	[0]	[1]	[2]
[0]	O	X	O
[1]	O	X	X
[2]	X	O	O

```
char ticTacToe[3][3];
```

Two Dimensional Arrays

- Used to implement table data

- Example uses:
  - Matrix
  - Image
  - Spreadsheet
  - Game board

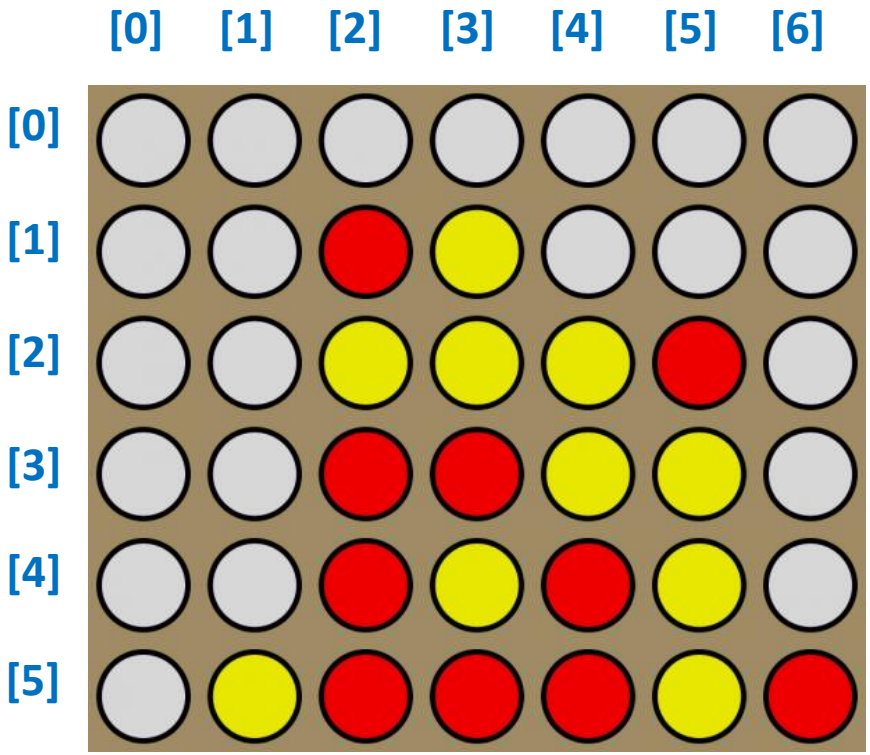



```
string chessBoard[8][8];
```

Two Dimensional Arrays

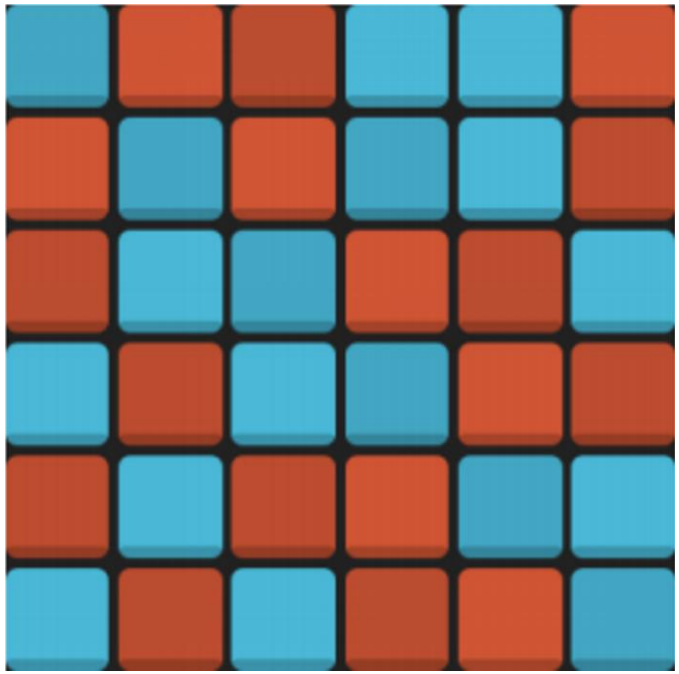
- Used to implement table data

- Example uses:
  - Matrix
  - Image
  - Spreadsheet
  - Game board



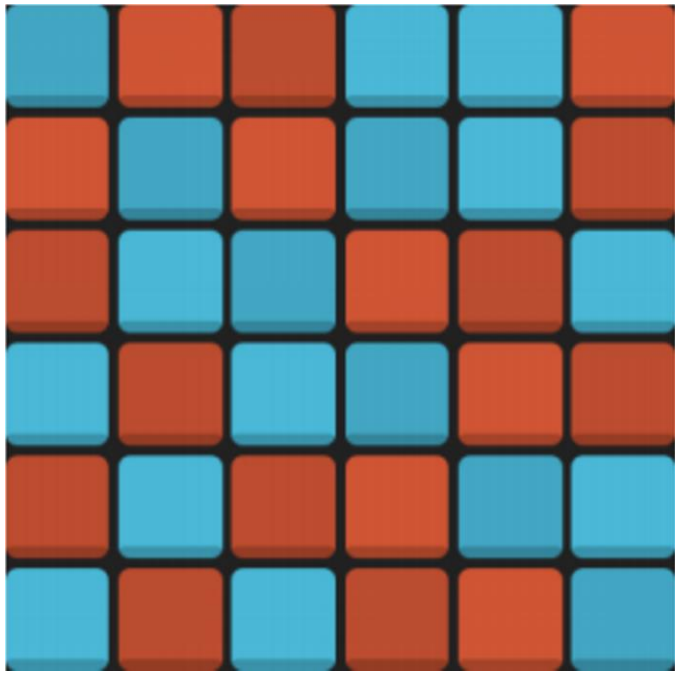

```
char connect4Board[6][7];
```

0h h1: 2D Array of `int`



```
int board[6][6];
```

0h h1: 2D Array of `int`

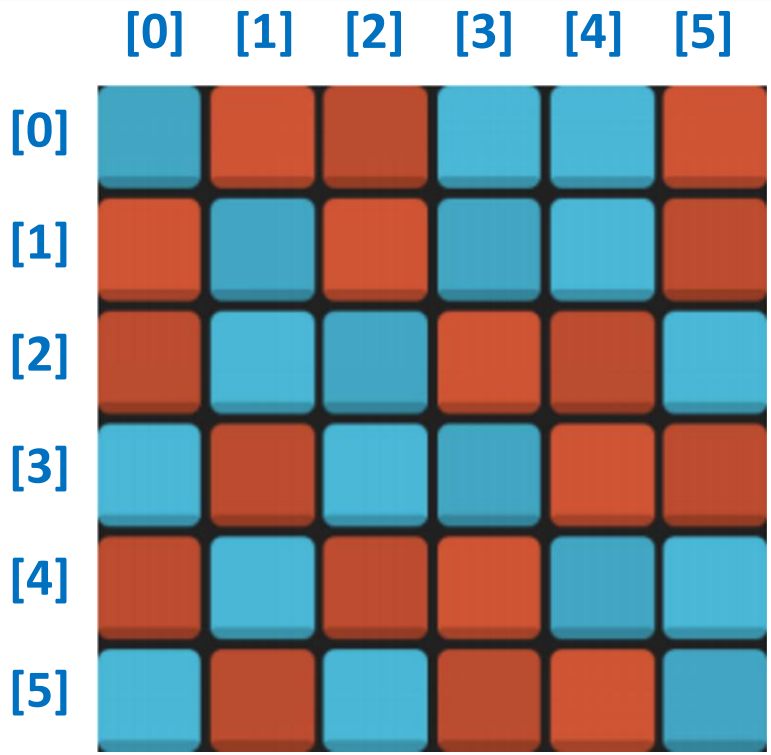


```
int board[6][6];
```

rows

columns

0h h1: 2D Array of `int`




```
int board[6][6];
```

rows **columns**

0h h1: 2D Array of `int`

	[0]	[1]	[2]	[3]	[4]	[5]
[0]						
[1]						
[2]						
[3]						
[4]						
[5]						

```
int board[6][6];
```

0h h1: 2D Array of `int`

	[0]	[1]	[2]	[3]	[4]	[5]
[0]						
[1]						
[2]						
[3]						
[4]						
[5]	1					

```
board[5][0] = 1;
```

0h h1: 2D Array of `int`

	[0]	[1]	[2]	[3]	[4]	[5]
[0]						
[1]						
[2]						
[3]						
[4]	2					
[5]	1					

```
board[4][0] = 2;
```

0h h1: 2D Array of `int`

	[0]	[1]	[2]	[3]	[4]	[5]
[0]						
[1]						
[2]						
[3]						
[4]	2					
[5]	1		1			

i>Clicker #7:
How do we place this 1?

```
A. board[2][5] = 1;
B. board[2][5] == 1;
C. board[5][2] = 1;
D. board[5][2] == 1;
```

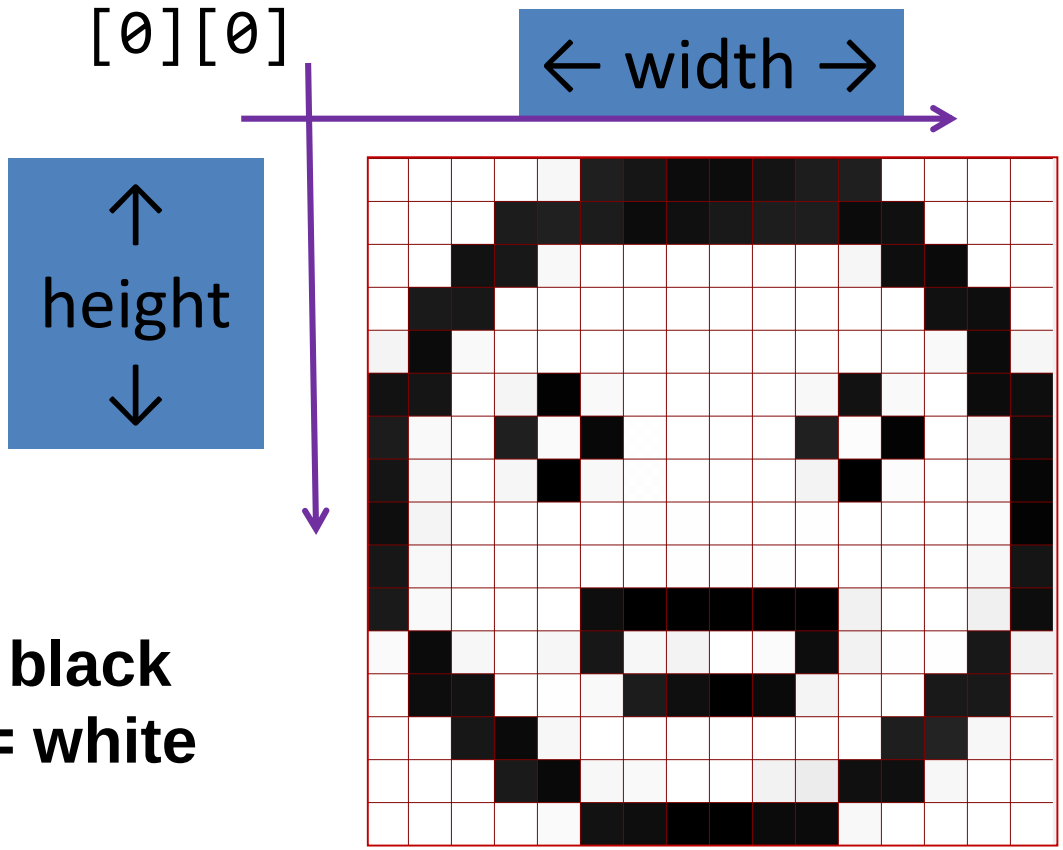
0h h1: 2D Array of `int`

	[0]	[1]	[2]	[3]	[4]	[5]
[0]						
[1]						
[2]						
[3]						
[4]	2					
[5]	1		1			

i>Clicker #7:
How do we place this 1?

```
A. board[2][5] = 1;
B. board[2][5] == 1;
C. board[5][2] = 1;
D. board[5][2] == 1;
```

Images: jpg, tiff, bmp

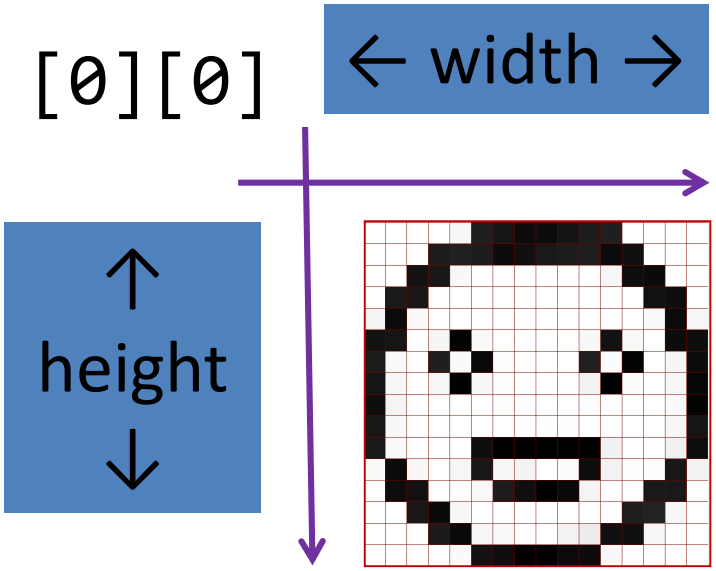


true = black
false = white

```
bool smiley[16][16];
```

Images: jpg, tiff, bmp

```
bool smiley[16][16];
// white-out the image
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        smiley[row][col] = false;
    }
}
```



Creating the Array

```
int height = 6;
int width = 6;


int board[height][width];
```

Creating the Array

```
int height = 6;
int width = 6;


int board[height][width];
```

Compile Error!

Why?

Creating the Array

```
const int HEIGHT = 6;
const int WIDTH = 6;
```

```
int board[HEIGHT][WIDTH];
```

Initializing the Array

```
const int HEIGHT = 6;
const int WIDTH = 6;

// like 1D arrays, the rest filled with 0
int board[HEIGHT][WIDTH] = { };
```

Initializing the Array

```
const int HEIGHT = 6;
const int WIDTH = 6;

// or we can initialize directly
int board[HEIGHT][WIDTH] = {
                            {1, 2, 3, 4},
                            {1, 4, 9, 16},
                            {1, 8, 27, 64},
                            {1, 16, 81, 256}
                           };
```

Initialize multi-dimensional arrays by grouping together initializers for each dimension

Initializing the Array

```
const int HEIGHT = 6;
const int WIDTH = 6;
const int UNKNOWN = 0;

// or we can initialize with loops!
int board[HEIGHT][WIDTH];

for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

i>Clicker #8

```
int main(void) {
    int message[3][3] = {{1, 2 , 3}, {4, 5}, { }};
}
```

What is the value of `message[2][0]`?

A. 0
B. 1
C. 2
D. 3
E. 4

i>Clicker #8

```
int main(void) {
    int message[3][3] = {{1, 2 , 3}, {4, 5}, { }};
}
```

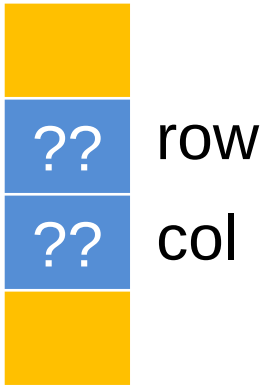
	0	1	2
0	1	2	3
1	4	5	0
2	0	0	0

What is the value of `message[2][0]`?

A. 0
B. 1
C. 2
D. 3
E. 4

Clearing a 0h h1 Board

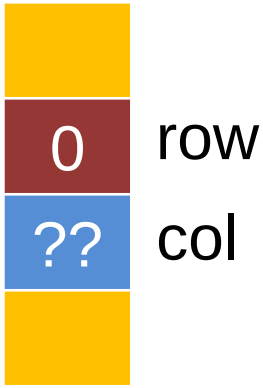
	[0]	[1]	[2]	[3]	[4]	[5]
[0]	??	??	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??




```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	??	??	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	??	??	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

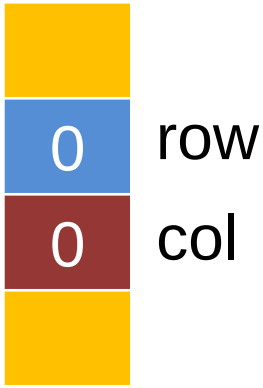
0	row
??	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	??	??	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	??	??	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

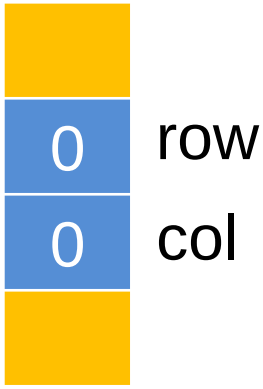
0	row
0	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	??	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

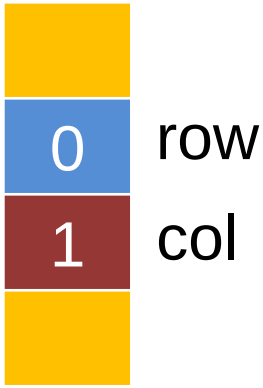


Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++){
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	??	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

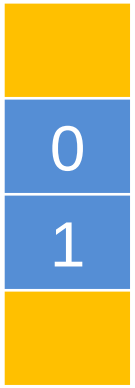


Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	??	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

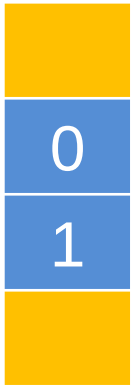


Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



row

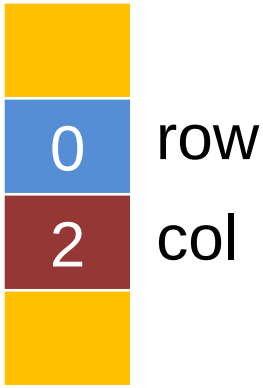
col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	??	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

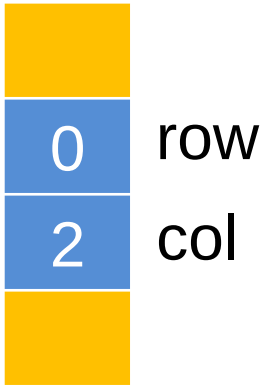
0	row
2	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

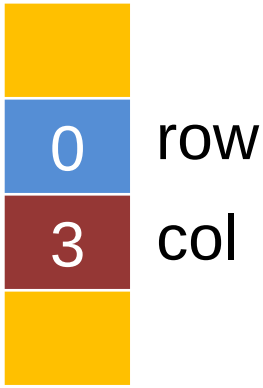


Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

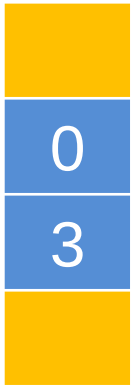


Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	??	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



row

col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	??	??
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

0	row
3	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

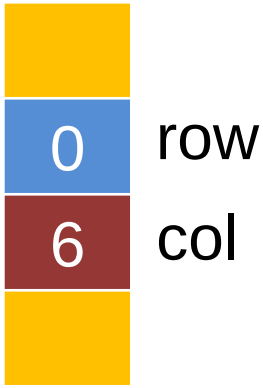
0	row
5	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

1	row
6	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

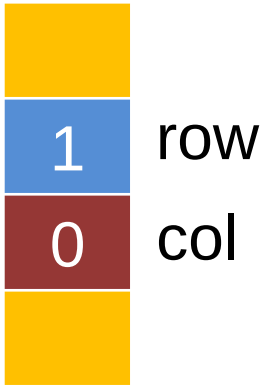
1	row
6	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	??	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

1	row
0	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	??	??	??	??	??
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

1	row
0	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	0	0	0	0	0
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

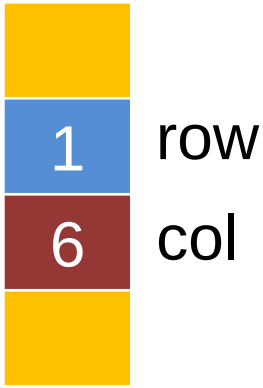
1	row
5	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	0	0	0	0	0
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	0	0	0	0	0
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

1	row
6	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	0	0	0	0	0
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

2	row
6	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	0	0	0	0	0
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

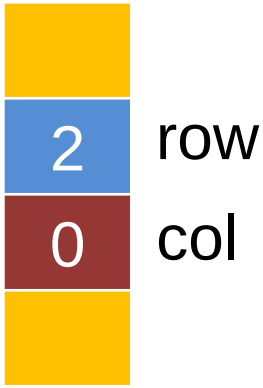
2	row
6	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	0	0	0	0	0
[2]	??	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??



Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	0	0	0	0	0
[2]	0	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

2	row
0	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

Parameters: 2D Arrays

```
void processArray(int data[50][3], ...);
```

Parameters: 2D Arrays

```
void processArray(int data[50][3], ...);
```

C++ does **not check** that your accesses are within range

Parameters: 2D Arrays

```
void processArray(int data[50][3], ...);
```

```
void processArray(int data[][3], ...);
```

Parameters: 2D Arrays

```
void processArray(int data[50][3], ...);
```

```
void processArray(int data[][3], ...);
```

```
void processArray(int data[][], ...);
```

Compile error

Parameters: 2D Arrays

```
void processArray(int data[50][3], ...);
```

```
void processArray(int data[][3], ...);
```

```
void processArray(int data[][], ...);
```

Second dimension required

Compile error

Parameters: 2D Arrays

```
void processArray(int data[50][3], ...);
```

```
void processArray(int data[][3], ...);
```

```
void processArray(int data[][], ...);
```

Compile error

Second dimension required

Actually, **ALL** but the first dimension is required

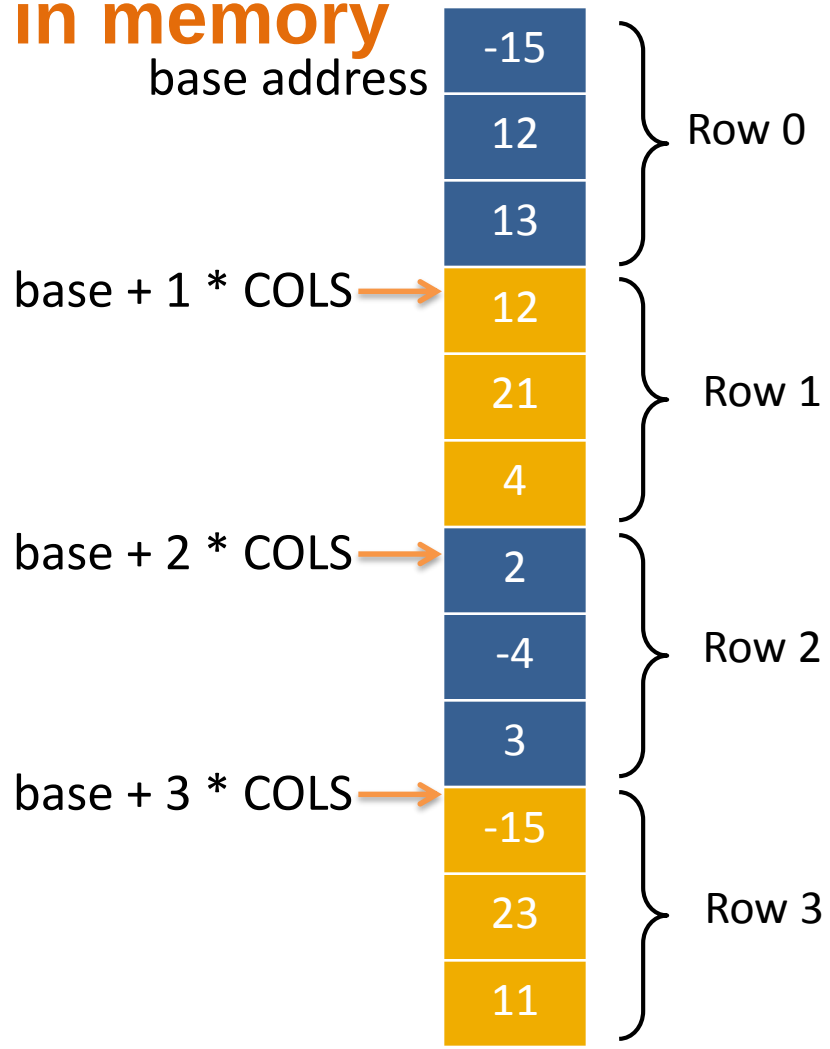
How 2D Arrays Are Stored in Memory

```
const int ROWS = 4;
const int COLS = 3;
int arr[ROWS][COLS];
```

logical view

	[0]	[1]	[2]
[0]	-15	12	13
[1]	12	21	4
[2]	2	-4	3
[3]	-15	23	11

in memory



How 2D Arrays Are Stored in Memory

```
const int ROWS = 4;
const int COLS = 3;
int arr[ROWS][COLS];
```

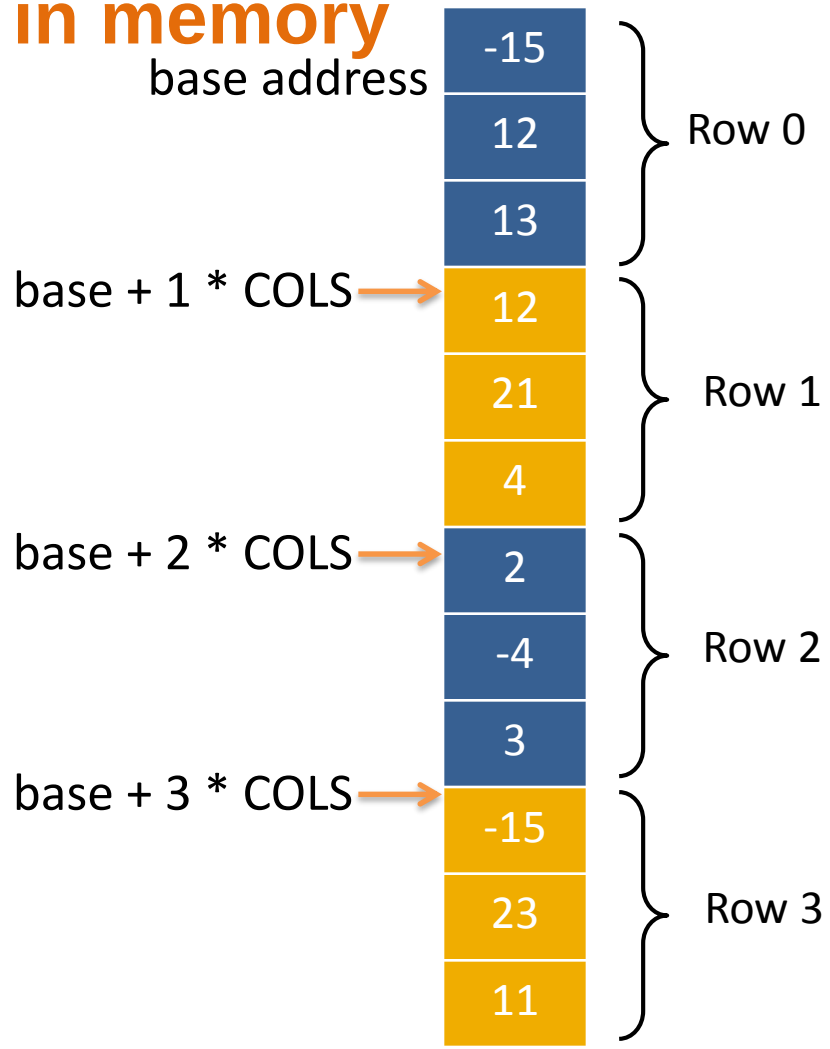
logical view

Known as Row-Major Form

stored row1, row2, etc

[1]	12	21	4
[2]	2	-4	3
[3]	-15	23	11

in memory



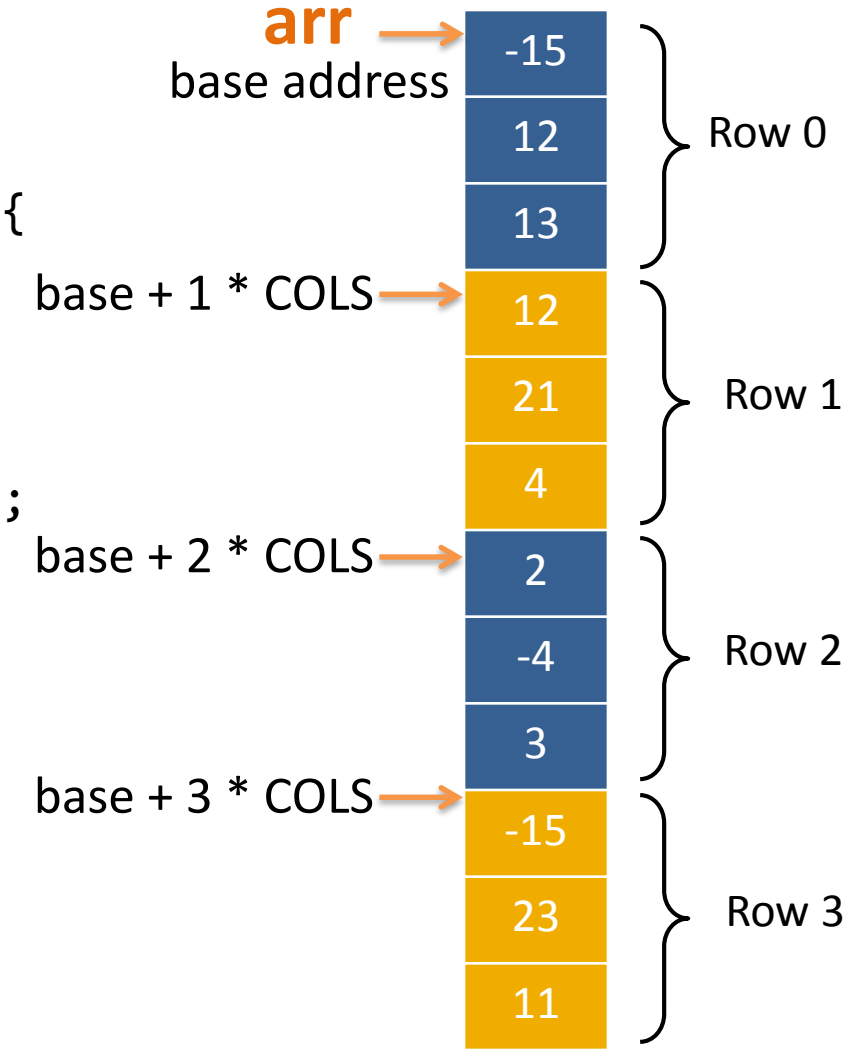
Why specify dimensions?

```
const int ROWS = 4;
const int COLS = 3;

void processArray(int arr[][]) {
    arr[1][1] = 0;
}

int main(void) {
    int arr[ROWS][COLS] = {...};
    arr[1][1] = 42;
    processArray(arr);
}
```

Execution



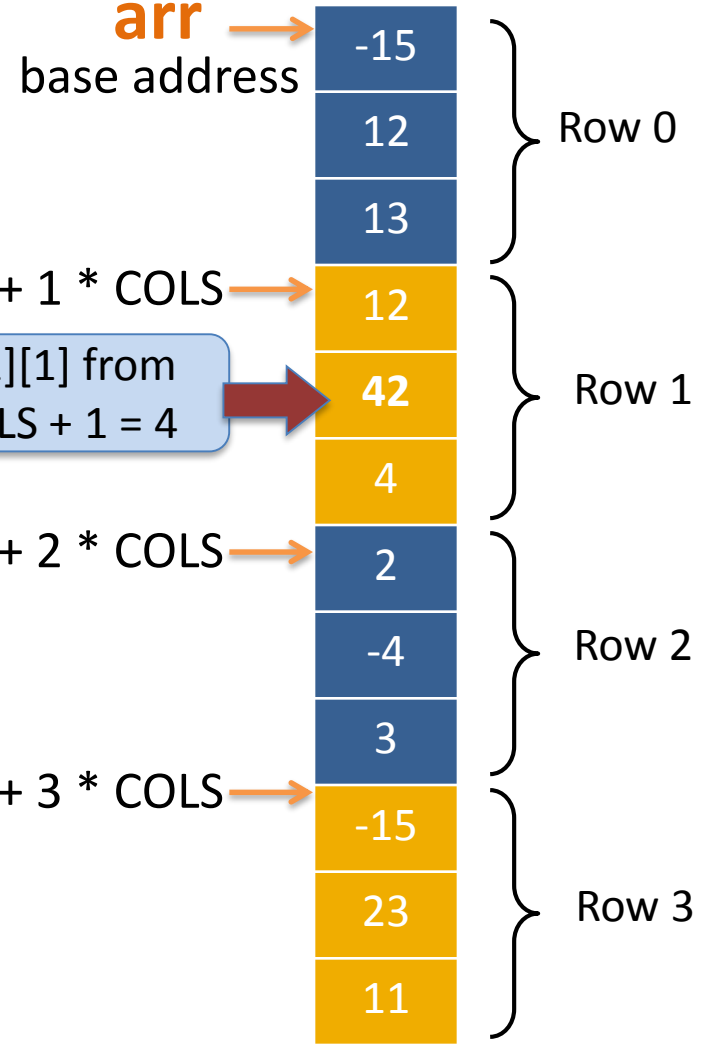
Why specify dimensions?

```
const int ROWS = 4;
const int COLS = 3;

void processArray(int arr[][]) {
    arr[1][1] = 0;
}

int main(void) {
    int arr[ROWS][COLS] = {...};
    arr[1][1] = 42;
    processArray(arr);
}
```

Execution



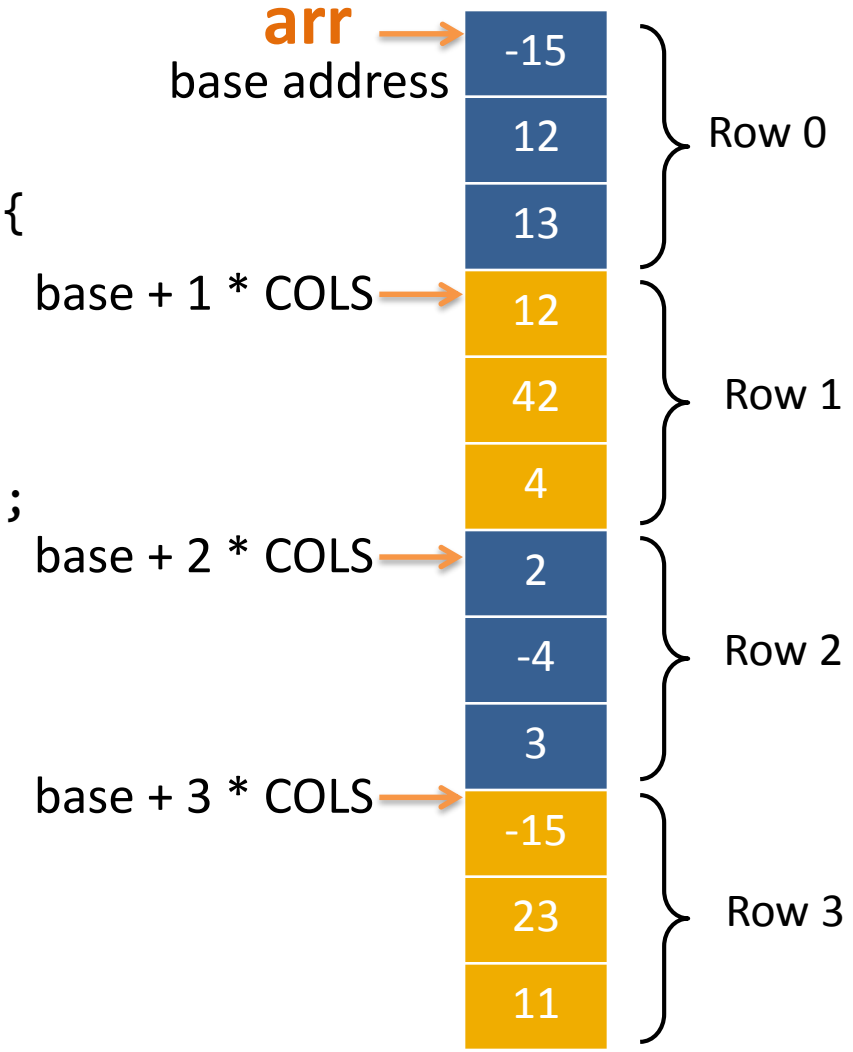
Why specify dimensions?

```
const int ROWS = 4;
const int COLS = 3;

void processArray(int arr[][]) {
    arr[1][1] = 0;
}

int main(void) {
    int arr[ROWS][COLS] = {...};
    arr[1][1] = 42;
    processArray(arr);
}
```

Execution



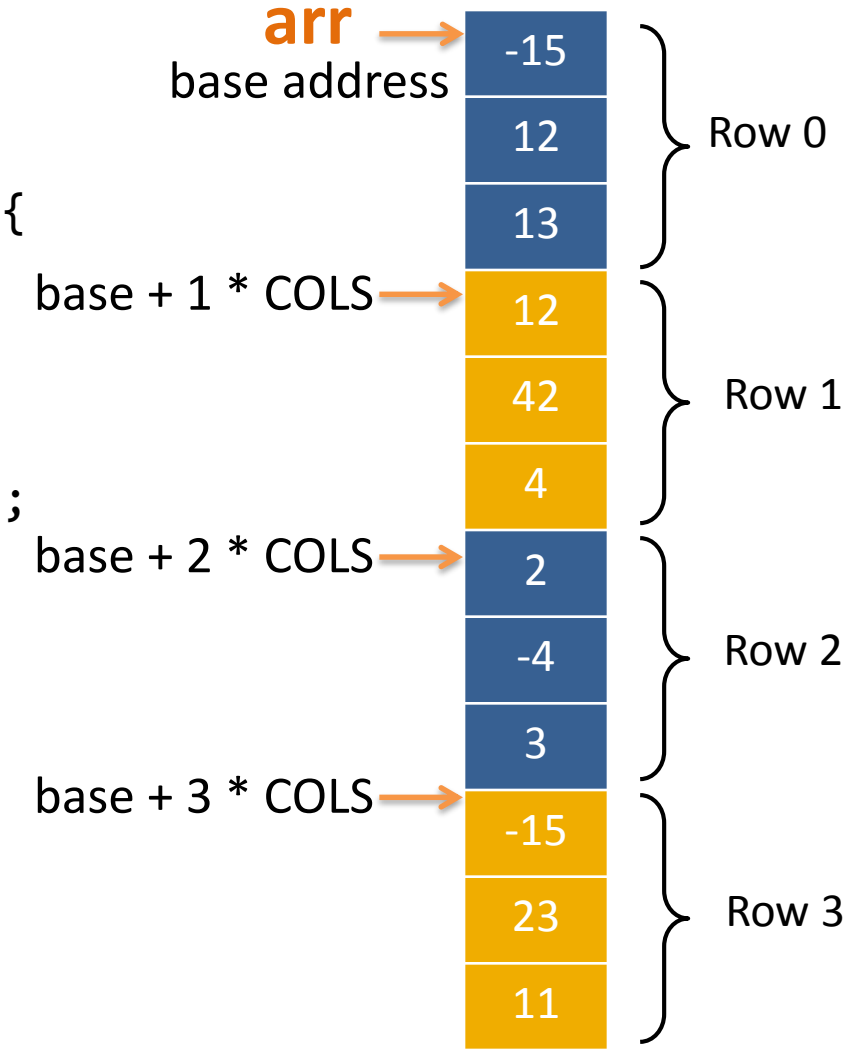
Why specify dimensions?

```
const int ROWS = 4;
const int COLS = 3;

void processArray(int arr[][]) {
    arr[1][1] = 0;
}

int main(void) {
    int arr[ROWS][COLS] = {...};
    arr[1][1] = 42;
    processArray(arr);
}
```

Execution



Why specify dimensions?

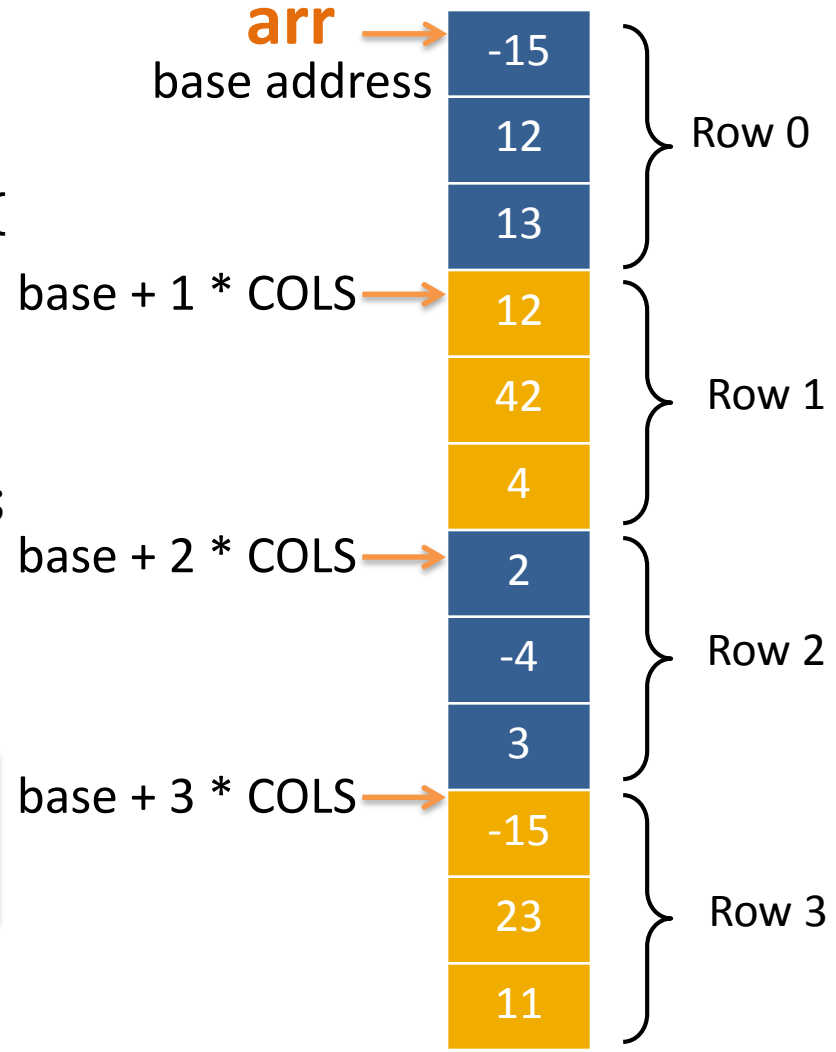
```
const int ROWS = 4;
const int COLS = 3;

void processArray(int arr[][]) {
    arr[1][1] = 0;
}

int main(void) {
    int arr[ROWS][COLS] = {...};
    arr[1][1] = 42;
    processArray(arr);
}
```

Execution

Offset of arr[1][1] is
1*??? + 1 = ???



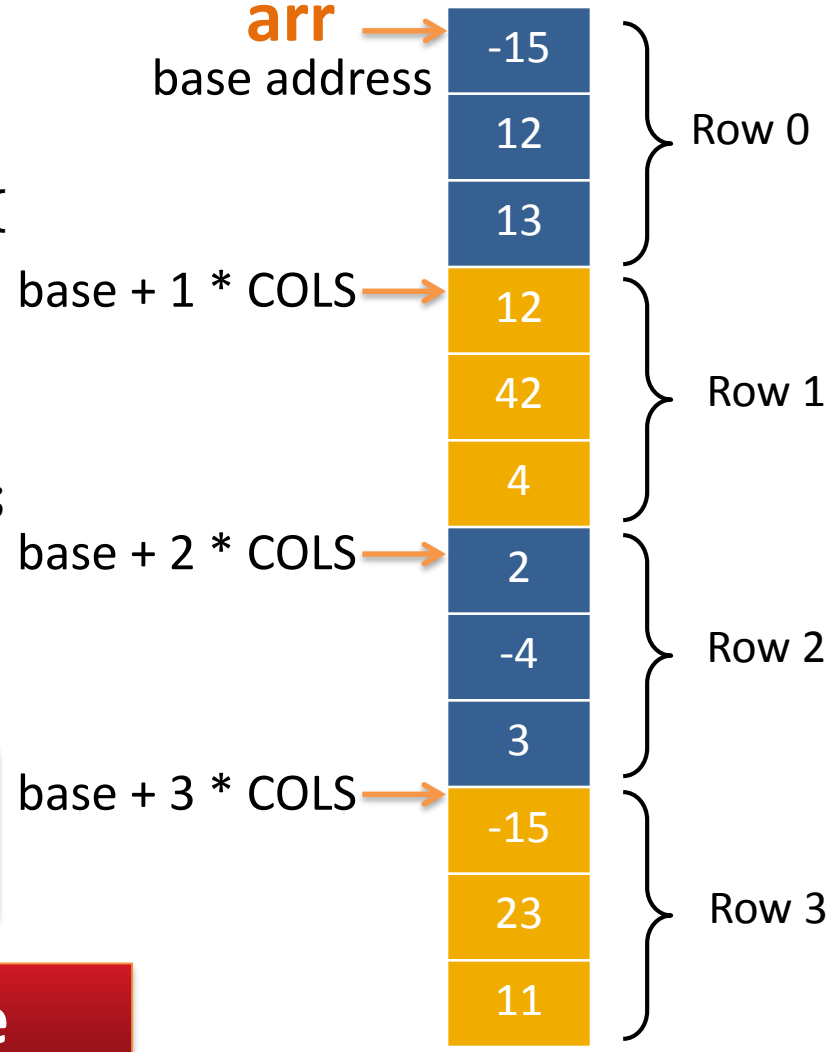
Tell the compiler how much memory to skip

```
const int ROWS = 4;
const int COLS = 3;

void processArray(int arr[][]) {
    arr[1][1] = 0;
}

int main(void) {
    int arr[ROWS][COLS] = {...};
    arr[1][1] = 42;
    processArray(arr);
}
```

Execution



Offset of arr[1][1] is
1 * ??? + 1 = ???

The code will not compile

i>Clicker #9

Which is a correct function prototype for a function that takes in a 2D array?

```
A.  int sumAll(int data[][], int rows, int cols);
B.  int sumAll(int data[9][2], int rows, int cols);
C.  int sumAll(int data[9][], int rows, int cols);
D.  int sumAll(int data[][2], int rows, int cols);
E.  B and D
```

i>Clicker #9

Which is a correct function prototype for a function that takes in a 2D array?

```
A.  int sumAll(int data[][], int rows, int cols);
B.  int sumAll(int data[9][2], int rows, int cols);
C.  int sumAll(int data[9][], int rows, int cols);
D.  int sumAll(int data[][2], int rows, int cols);
E.  B and D
```

Count Chips

```
// Requires: nothing
// Modifies: red, yellow
// Effects: sets red and blue to
//          the number of chips in
//          board of that color
void count_chips(int board[6][7], int &red,
                 int &blue) {
    // ???
}
```

Count Chips

```
void count_chips(int board[6][7],
                 int &red,
                 int &blue) {
    // initialize red and blue
    // loop over all squares
    //   if square is red, red++
    //   if square is blue, blue++
}
```

Count Chips

```
void count_chips(int board[6][7],
                 int &red,
                 int &blue) {
    red = 0;
    blue= 0;

}
```

Count Chips

```
void count_chips(int board[6][7],
                 int &red,
                 int &blue) {
    red = 0;
    blue= 0;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            }
    }
}
```

Count Chips

```
void count_chips(int board[6][7],
                 int &red,
                 int &blue) {
    red = 0;
    blue= 0;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (board[i][j] == RED) {
                red++;
            }
        }
    }
}
```

Count Chips

```
void count_chips(int board[6][7],
                 int &red,
                 int &blue) {
    red = 0;
    blue= 0;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (board[i][j] == RED) {
                red++;
            } else if (board[i][j] == BLUE) {
                blue++;
            }
        }
    }
}
```

Count Chips

```
void count_chips(int board[6][7],
                 int &red,
                 int &blue) {
    red = 0;
    blue= 0;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (board[i][j] == RED) {
                red++;
            } else if (board[i][j] == BLUE) {
                blue++;
            }
        }
    }
}
```

Next Time

More arrays
File I/O

Examples

On your own

Example 1: readGrades() function

Create the following global named constants:

```
const int NUM_STDNTS = 10;
const int NUM_EXAMS = 3;
```

Inside `main()`, create the array that will be passed to the functions:

```
int grades[NUM_STDNTS][NUM_EXAMS];
```

Read in exam grades interactively

```
// Requires: The size of array grades is
//        NUM_STDNTS x NUM_EXAMS
//        NUM_STDNTS > 0 && NUM_EXAMS > 0

// Modifies: The array grades

// Effects:  read from the standard input
//   NUM_EXAMS grades for each of NUM_STDNTS
//   students and store them in array grades
void readGrades(int grades[][NUM_EXAMS]);
```

Read in exam grades interactively

Console

```
Enter grade for student 1 exam 1:  17
Enter grade for student 1 exam 2:  22
Enter grade for student 1 exam 3:  35
Enter grade for student 2 exam 1:  15
Enter grade for student 2 exam 2:  19
….
```

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Read in exam grades interactively

```
void readGrades(int grades[][NUM_EXAMS]) {

    for (          ?          ) {

        for (          ?          ) {




        }
    }
}
```

Read in exam grades interactively

```
void readGrades(int grades[][NUM_EXAMS]) {

    for (int i = 0; i < NUM_STDNTS; i++)  {

        for (int j = 0; j < NUM_EXAMS; j++) {

                                ?

        }
    }
}
```

Read in exam grades interactively

```
void readGrades(int grades[][NUM_EXAMS]) {

    for (int i = 0; i < NUM_STDNTS; i++)  {

        for (int j = 0; j < NUM_EXAMS; j++) {

            cout << "Enter a grade for student "
                 << i + 1
                 << " exam " << j + 1 << ": ";

            cin >> grades[i][j];
        }
    }
}
```

Example 2: computeTotalGrades()

computeTotalGrades()

```
// Requires: The size of array grades is
//      NUM_STDNTS x NUM_STDNTS
//      The size of array totals is NUM_STDNTS
//      NUM_STDNTS > 0 && NUM_EXAMS > 0

// Modifies: The array totals

// Effects:  computes sum of the NUM_EXAMS
// grades for each student and stores total
// grade at corresponding index in totals
void computeTotalGrades(int grades[][NUM_EXAMS],
          int totals[]);
```

computeTotalGrades()

Exam 1	Exam 2	Exam 3		Total
17	22	35	→	74
15	19	33	→	67
:	:	:		:
16	20	31	→	67

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {

    for (int i = 0; i < NUM_STDNTS; i++) {



        for (int j = 0; j < NUM_EXAMS; j++) {



        }
    }
}
```

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {

    for (int i = 0; i < NUM_STDNTS; i++) {



        for (int j = 0; j < NUM_EXAMS; j++) {

            totals[i] += grades[i][j];

        }
    }
}
```

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {

    for (int i = 0; i < NUM_STDNTS; i++) {

        totals[i] = 0;

        for (int j = 0; j < NUM_EXAMS; j++) {

            totals[i] += grades[i][j];

        }
    }
}
```

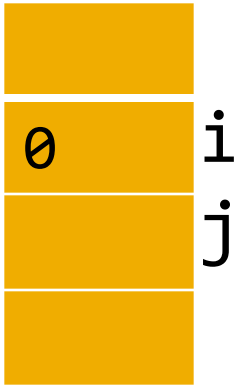
computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
:



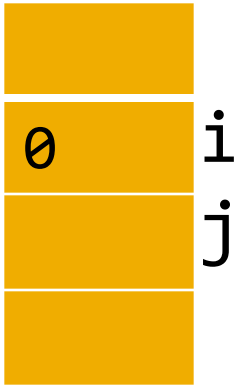
computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
:



computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
0
:

0	i
0	j

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
0
:

0	i
0	j

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
~~0~~ 17
:

0	i
0	j

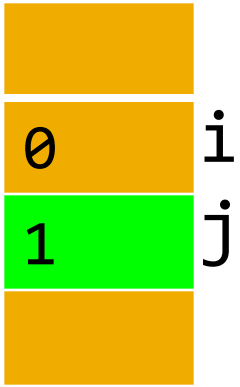
computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
17
:



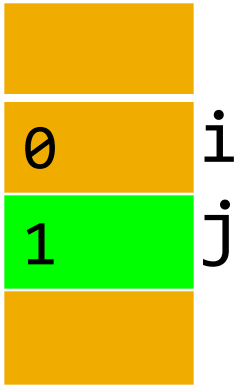
computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
17
:



computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
~~17~~ 39
:

0	i
1	j

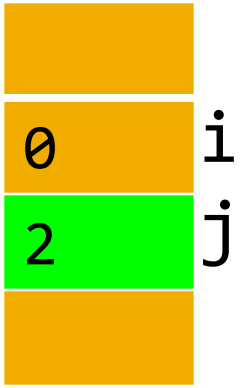
computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
39
:



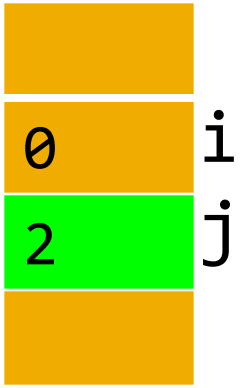
computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
39
:



computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
~~39~~ 74
:

0	i
2	j

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

sums row 'i'

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
74
:

0	i
2	j

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[i] += grades[i][j];
```

goes to next row

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
74
:

0	i
2	j

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int i = 0; i < NUM_STDNTS; i++) {
        totals[i] = 0;
        for (int j                                  +) {
            totals[i] += grades[i][j];
```

0's total for current row

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
74
:

0	i
2	j

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int j = 0; j < NUM_EXAMS; j++) {
        totals[j] = 0;
        for (int i = 0; i < NUM_STDNTS; i++) {
            totals[i] += grades[i][j];
```

swapped

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Total
:

0	i
2	j

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int j = 0; j < NUM_EXAMS; j++) {
        totals[j] = 0;
        for (int i = 0; i < NUM_STDNTS; i++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

A) sums rows
B) sums columns
C) sums entire array
D) non-deterministic

:

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    for (int j = 0; j < NUM_EXAMS; j++) {
        totals[j] = 0;
        for (int i = 0; i < NUM_STDNTS; i++) {
            totals[i] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

A) sums rows
B) sums columns
C) sums entire array
D) non-deterministic

:

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    int totals[0] = 0;
    for (int i = 0; i < NUM_STDNTS; i++) {
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[0] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

A) sums rows
B) sums columns
C) sums entire array
D) non-deterministic

:

computeTotalGrades()

```
void computeTotalGrades(int grades[][NUM_EXAMS],
                        int totals[]) {
    int totals[0] = 0;
    for (int i = 0; i < NUM_STDNTS; i++) {
        for (int j = 0; j < NUM_EXAMS; j++) {
            totals[0] += grades[i][j];
```

grades

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

A) sums rows
B) sums columns
C) sums entire array
D) non-deterministic

:

Example 3: printAverages()

Print Averages

```
// Requires: size of array grades is
//           NUM_STDNTS x NUM_STDNTS
//   size of array totals is NUM_STDNTS
//   NUM_STDNTS > 0 && NUM_EXAMS > 0
// Modifies: nothing
// Effects:  computes and prints the
//      average of each of the exams
void printAverages(int grades[][NUM_EXAMS])
```

Print Averages

Exam 1	Exam 2	Exam 3
17	22	35
15	19	33
:	:	:
16	20	31

Console

```
Exam 1 Average: 16.5
Exam 2 Average: 19.3
Exam 3 Average: 32.4
```

Print Averages

```
void printAverages(int grades[][NUM_EXAMS]) {
    for (int i = 0;          ?          ) {


        for (int j = 0;          ?          {



        }



    }
}
```

Print Averages

```
void printAverages(int grades[][NUM_EXAMS]) {
    for (int i = 0; i < NUM_EXAMS; i++) {


        for (int j = 0; j < NUM_STDNTS; j++) {




        }



    }
}
```

Print Averages

```
void printAverages(int grades[][NUM_EXAMS]) {
    for (int i = 0; i < NUM_EXAMS; i++) {

        double total = 0;
        for (int j = 0; j < NUM_STDNTS; j++) {




        }



    }
}
```

Print Averages

```
void printAverages(int grades[][NUM_EXAMS]) {
    for (int i = 0; i < NUM_EXAMS; i++) {

        double total = 0;
        for (int j = 0; j < NUM_STDNTS; j++) {

            total += grades[i][j];

        }



    }
}
```

Print Averages

```
void printAverages(int grades[][NUM_EXAMS]) {
    for (int i = 0; i < NUM_EXAMS; i++) {

        double total = 0;
        for (int j = 0; j < NUM_STDNTS; j++) {

            total += grades[i][j];

        }
        cout<< "Exam " << i + 1 << " Average: "
                       << total / NUM_STDNTS;

    }
}
```