NWEN 241
Arrays

School of Engineering and Computer Science

Victoria University of Wellington

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga
o te Ūpoko o te Ika a Māui

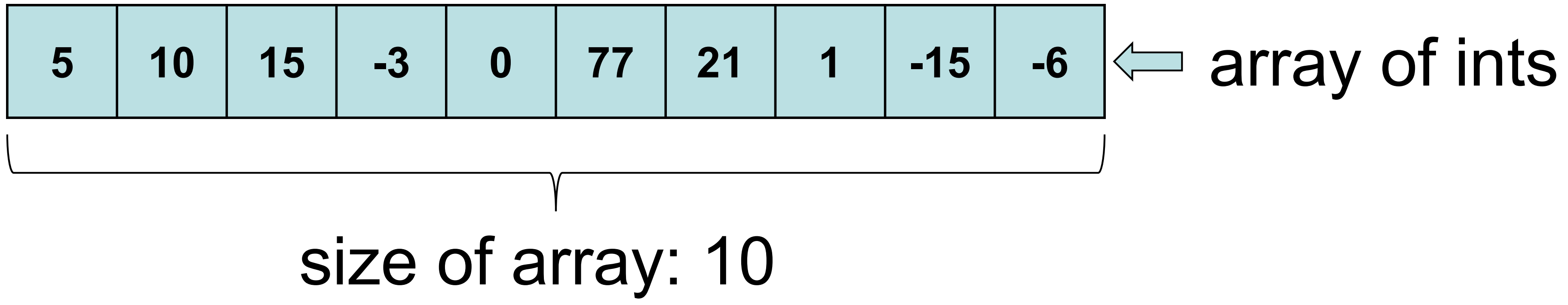
CAPITAL CITY UNIVERSITY

Arrays in General

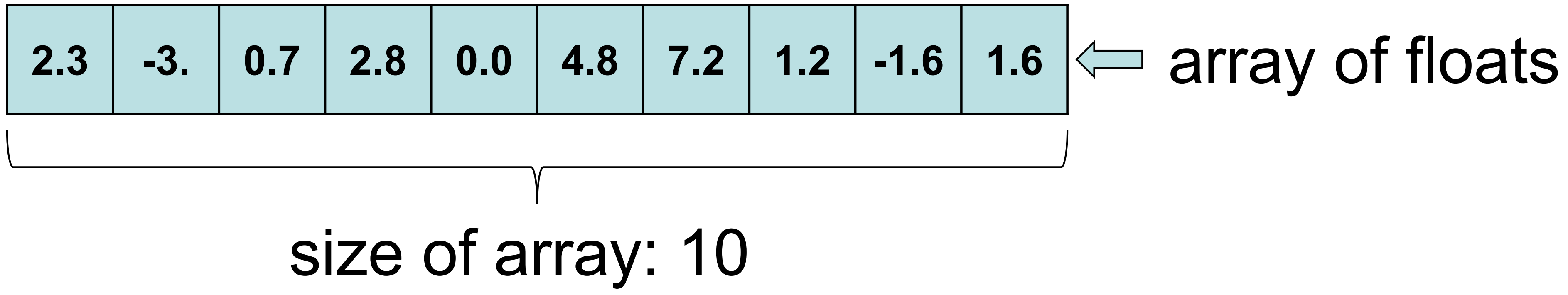
- An array is a collection of data that holds a **fixed** number of data (values) of the **same type**
- In C, arrays and pointers are closely related concepts
  - An array name by itself is treated as a constant pointer

- We distinguish between two types of arrays:
  - **One-dimensional arrays**
  - **Multi-dimensional arrays**
    - The C language places no limits on the number of dimensions in an array, though specific implementations may

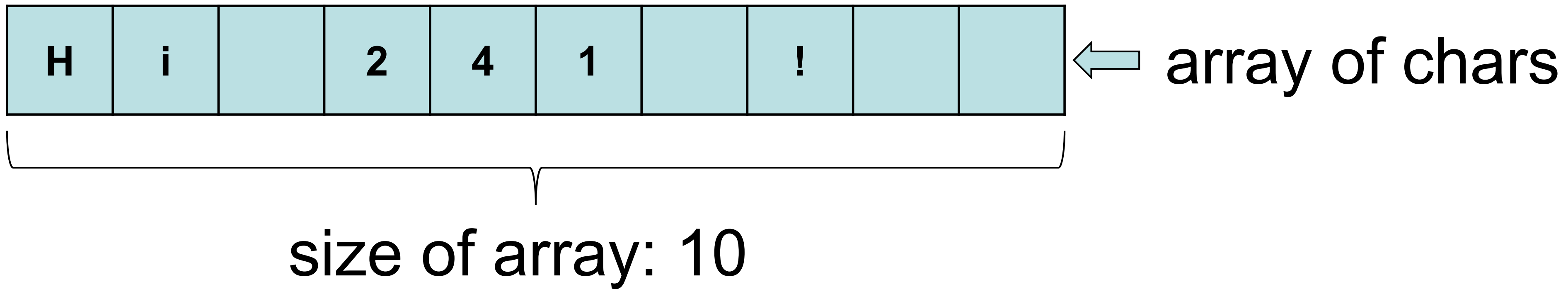
One-dimensional (1D) Array Overview



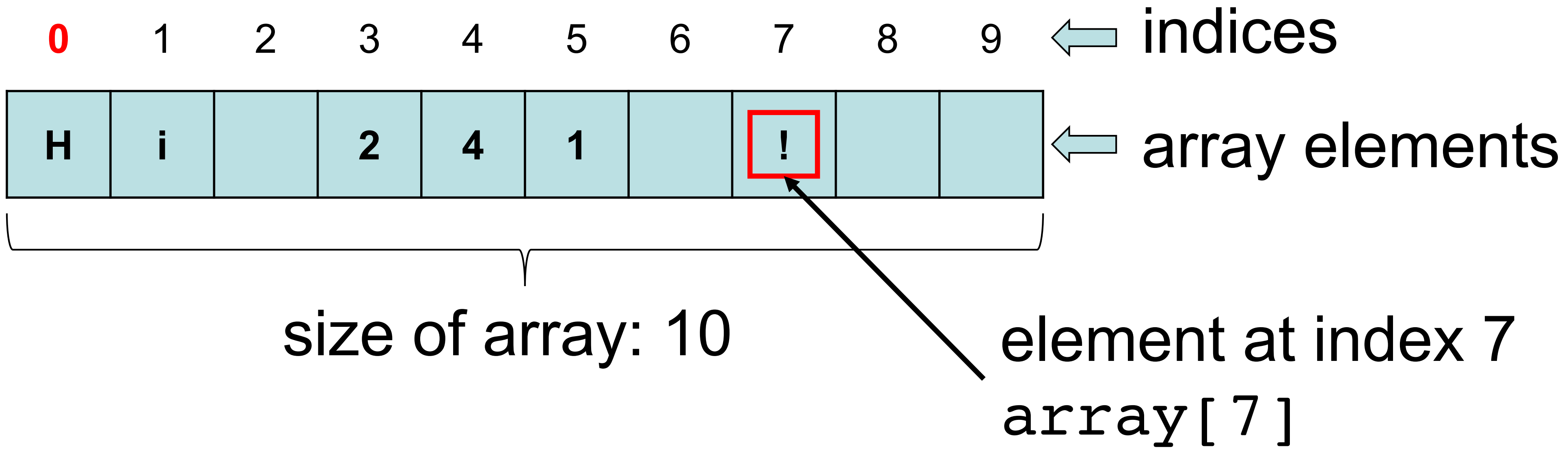
One-dimensional (1D) Array Overview



One-dimensional (1D) Array Overview



One-dimensional (1D) Array Overview



Arrays

- The simplest interpretation of an array is one-dimensional array, often referred to as a list
- The individual elements of the array can be accessed via **indices**
  - The first index of an array starts at **0**
  - If the size of an array is **n**, to access the last element the index **n-1** is used
  - This is because the index in C is actually an offset from the <u>beginning</u> of the array
    - The first element is at the beginning of the array, and hence has zero offset

Declaring 1D Arrays

- Syntax for **declaring** an array:

```
data_type array_name[array_size];
```

- Example:
  - We declare an array named **data** of **floating-point** type and size **4** as:

```
float data[4];
```

  - It can hold 4 floating-point values

- The **size** and **type** of arrays **<u>cannot</u>** be changed after their declaration!

Initializing 1D Arrays – Method 1

- Arrays can be initialized **one-by-one**
- For example:

```
float data[4];
data[0] = 22.5;
data[1] = 23.1;
data[2] = 23.7;
data[3] = 24.8;
```

- In the case of large arrays this method is <u>inefficient</u>

Initializing 1D Arrays – Method 2

- Arrays can be also initialized when they are **declared** (just as any other variables):

```
float data[4] = {22.5, 23.1, 23.7, 24.8};
```

- An array may be **partially initialized**, by providing fewer data items than the size of the array

```
float data[4] = {22.5, 23.1};
```

  - **The remaining array elements will be automatically initialized to zero**

- If an array is to be completely initialized, **the dimension (size) of the array is not required**

```
float data[] = {22.5, 23.1, 23.7, 24.8}
```

  - **The compiler will automatically size the array to fit the initialized data**

Arrays & Loops

- Arrays are commonly used in conjunction with **loops**
  - in order to perform the same calculations on all (or some part) of the data items in the array:
  - **While loop**:

```
int idx = 0;
while(idx < 10){
      // magic happens here
}
```

  - **For loop**:

```
for (int idx = 0; idx < 10; idx++){
      // magic happens here
}
```

Off-by-one Error

- The most common mistake when working with arrays in C is forgetting that indices start at zero and stop one less than the array size
- We often refer to this issue as ”Off-by-one Error”

```
int data[] = {1,2,3,4,5};//number of elements is 5
for (int idx = 0; idx <= 5; idx++){
        // magic happens here
}
```

- The compiler does not control the limits of the array!
- This type of error can be detected using static code analysis
  - For example using the **cppcheck** tool
- A Special type of Off-by-one error is the “Fencepost error”
  - A straight fence with *n* sections has *n+1* posts

Determining the size of an array

- The size of an array can be determined using the **sizeof()** operator
- It will return **the number of bytes the array "occupies" in the memory**

```
int data[] = {1,2,3,4,5};
printf("sizeof(data): %ld\n",sizeof(data)/sizeof(int));
```

- To determine the number of elements in the array, the **returned** value must be **divided** by the **number of bytes** reserved for the **data type** **!**

Passing 1D Arrays to Functions (1)

- Passing a **single array element** to a function
  - can be passed in a similar manner as passing a variable to a function

```
void display(int age) {
    printf("%d", age);
}

int main() {
    int age[] = { 18, 19, 20 };
    display(age[2]); //Passing array element age[2] only
    return 0;
}
```

Passing 1D Arrays to Functions (2)

- Passing an **entire array** to a function
  - When passing an array as an argument to a function, it is passed by its **memory address** (starting address of the memory area) and not its value!

```
float average(int age[]) {
   int sum = 0;
   for (int i = 0; i < 6; ++i) {
         sum += age[i];}
   float avg = ((float)sum / 6);
   return avg;
}
int main() {
   int age[] = {18,19,20,21,22,23};
   float avg = average(age);
   printf("Average age=%.2f\n", avg);
}
```

Searching Algorithms

- A search algorithm is an algorithm that retrieves information stored in some data structure
  - Data structures can include linked lists, arrays, hash tables, etc.
- In C programming searching through the data of an array is a common operation
  - Searching for a value in a large sized array is a resource and time demanding task
  - Search functions are usually evaluated on the basis of their complexity, or maximum theoretical run time.
  - The appropriate search algorithm often depends on the data structure being searched

→ → → Linear vs Binary search ← ← ←

Sorting Algorithms

- A sorting algorithm is an algorithm that puts elements of a list in a certain order
- The most-used orders are numerical order and lexicographical order
- Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists
- Since lists allow only sequential access, the data is often taken to be in an array

Bubble Sort

Starting from the beginning of the list, compare every adjacent pair, swap their position if they are not in the right order (the latter one is smaller than the former one).

After each iteration, one less element (the last one) is needed to be compared until there are no more elements left to be compared.

Unsorted: 4 1 5 9 8 2 3

Step 1 : 1 4 5 8 2 3 9
Step 2 : 1 4 5 2 3 8 9
Step 3 : 1 4 2 3 5 8 9
Step 4 : $\boxed{1\ 2\ 3\ 4\ 5\ 8\ 9}$

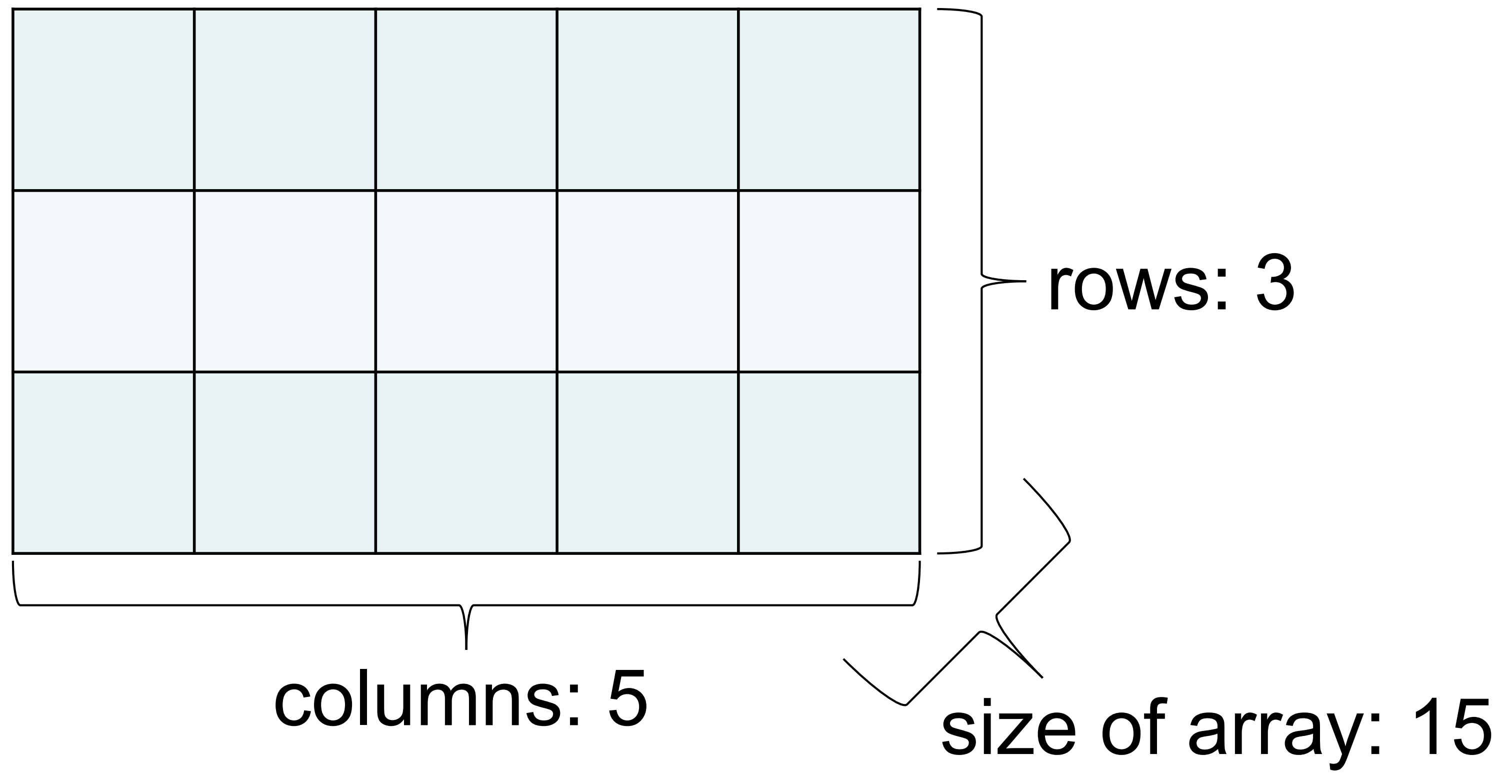
stdlib.h

- In practice you do not have to know how to implement these algorithms
  - The most efficient ones have been already implemented in C
  - In the **stdlib.h** pre-processor directive (header file)
    - **Binary search**
    - **Quick sort**

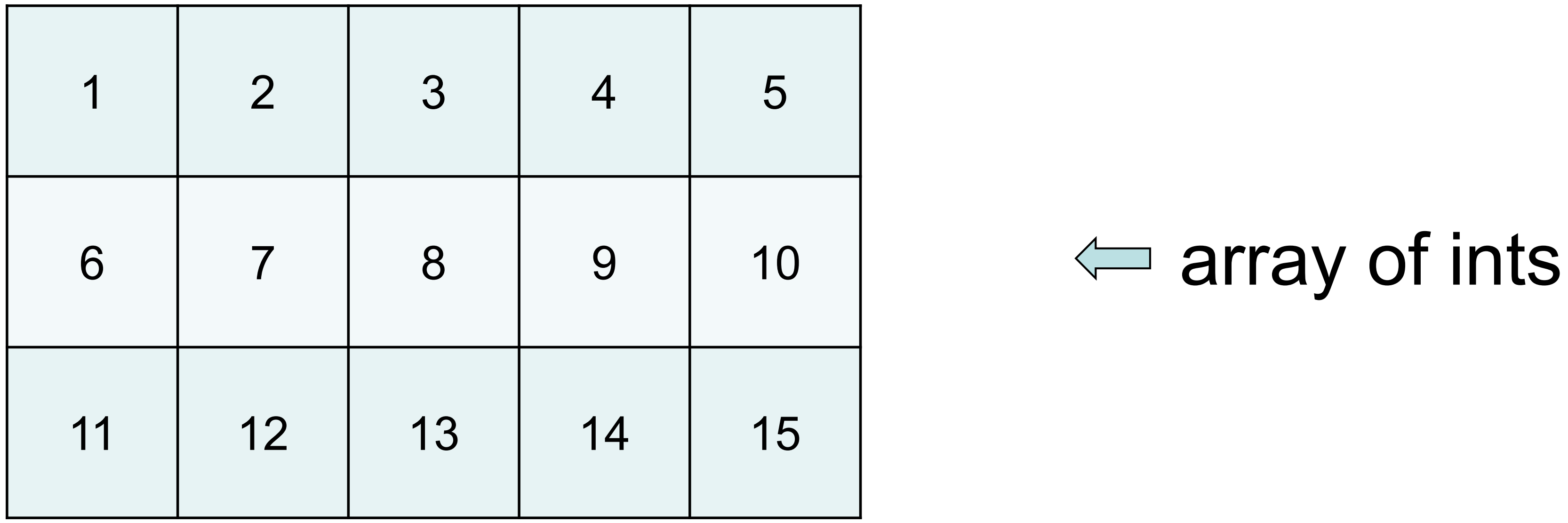
Multi-dimensional Arrays

- In C, you can create array of an array known as multidimensional array
- The simplest interpretation of a multi-dimensional array is a table, i.e. a **two-dimensional array**
  - each row has the same number of columns

Two-dimensional Arrays



Two-dimensional Arrays



Two-dimensional Arrays

1.0	2.0	3.0	4.0	5.0
6.0	7.0	8.0	9.0	10.0
11.0	12.0	13.0	14.0	15.0

⇦ array of floats

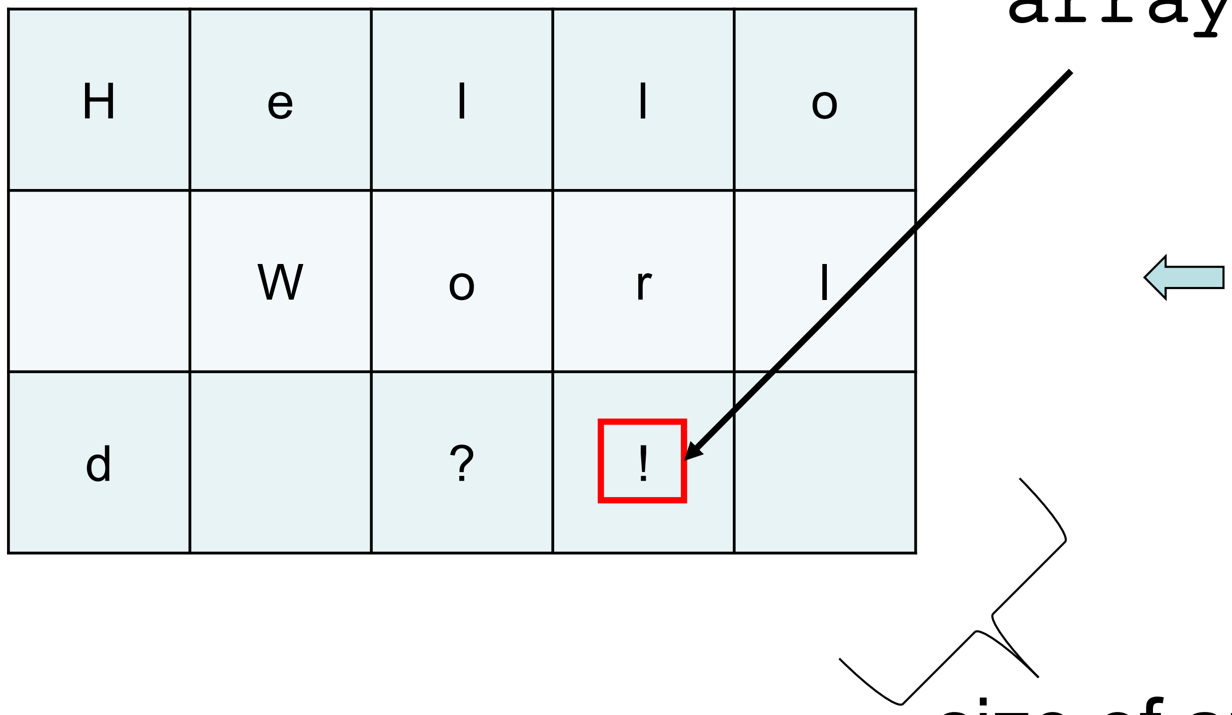
Two-dimensional Arrays

H	e	l	l	o
	W	o	r	l
d		?	!	

⇐ array of chars

Two-dimensional Arrays

element at row 3 column 4
`array[2][3]`



H	e	l	l	o
	W	o	r	l
d		?	!	

array elements

size of array: 15

Two-dimensional Arrays

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

2D Arrays

- Declaring a char array with 3 rows and 5 columns

```
char two_d[3][5];
```

  – The array can hold 15 elements

- Accessing a value

```
char ch;
ch = two_d[2][4];
```

- Modifying a value

```
two_d[0][0] = 'x';
```

- The array can be initialized in one of the following ways

```
int two_d[2][3] = {{ 5, 2, 1 }, { 6, 7, 8 }};
int two_d[2][3] = { 5, 2, 1 , 6, 7, 8 };
int two_d[][3] = {{ 5, 2, 1 }, { 6, 7, 8 }};
```

  – **The number of columns must be explicitly stated.** The compiler will find the appropriate amount of rows based on the initializer list.

Passing 2D Arrays to Functions

- Similarly to one-dimensional arrays, a two-dimensional array element or an entire two-dimensional array can be passed to a function.
- When passing a **multi-dimensional** array as an argument to a function, the array is passed to the function by its memory address (starting address of the memory area) and not its value!

```
void enterData(int firstMatrix[][10], int secondMatrix[][10])
{
   // code for reading and saving data into the 2D array
}

int main() {
       int firstMatrix[10][10], secondMatrix[10][10];
       enterData(firstMatrix, secondMatrix); // Calling
Function to take data
}
```

Three-dimensional Arrays

- Declaring a three-dimensional (3d) array

```
float three_d[2][4][3];
```

  - Here, three_d can hold 24 elements. Each 2 elements have 4 elements, which makes 8 elements and each 8 elements can have 3 elements.

- Initializing a three-dimensional array

```
int test[2][3][4]={{{3, 4, 2, 3},{0, -3, 9, 11},{23, 12, 23, 2}},
                    {{13, 4, 56, 3},{5, 9, 3, 5},{3, 1, 4, 9}} };
```