Most of the time, pointer and array accesses can be treated as acting the same, the major exceptions being:

- the sizeof operator
  sizeof(array) returns the amount of memory used by all elements in array
  sizeof(pointer) only returns the amount of memory used by the pointer variable itself
- the & operator
  &array is an alias for &array[0] and returns the address of the first element in array

  &pointer returns the address of pointer

- a string literal initialization of a character array
  char array[] = “abc” sets the first four elements in array to ‘a’, ‘b’, ‘c’, and ‘\0’
  char *pointer = “abc” sets pointer to the address of the “abc” string (which may be stored in read-only memory and thus unchangeable)
- Pointer variable can be assigned a value whereas array variable cannot be.

int array[4] =

1	2	3	4
100	104	108	112

char arr[] =

a	b →z	c	\0
100	101	102	103

char *ptr =

a	b	c	\0
100	101	102	103

ptr = 100
1000

sizeof(array) → 16
array → 100 → &array → 100
char arr[] = “abc”
char *ptr = “abc”
arr → 100
(arr+1) → 101 → *(arr+1) = z
*(ptr+1) = z –> throw an error, stored in read only memory
arr = 98 → throw an error

- a string literal initialization of a character array
  char array[] = “abc” sets the first four elements in array to ‘a’, ‘b’, ‘c’, and ‘\0’
  char *pointer = “abc” sets pointer to the address of the “abc” string (which may be stored in read-only memory and thus unchangeable)
- Pointer variable can be assigned a value whereas array variable cannot be.

int array[4] =

1	2	3	4
100	104	108	112

char arr[] =

a	b →z	c	\0
100	101	102	103

char *ptr =

a	b	c	\0
100	101	102	103

ptr = 100
1000

sizeof(array) → 16
array → 100 → &array → 100
char arr[] = “abc”
char *ptr = “abc”
arr → 100
(arr+1) → 101 → *(arr+1) = z
*(ptr+1) = z –> throw an error, stored in read only memory
arr = 98 → throw an error

Thank you for watching!

Please leave us your comments.