

Pointers in C

Md. Aminul Islam Shazid

24 Nov 2025

Outline

- 1 Introduction
- 2 Pointer Arithmetic
- 3 Pointers with Function

Introduction

What is a Pointer?

- Every variable has an address in memory
- A pointer stores the memory address of another variable
- A pointer itself is a variable and has its own memory address

Defining a Pointer

Pointers are defined by putting a `*` before the pointer's name:

```
int x = 10;  
int *p = &x;    // p stores the address of x
```

Different Types of Pointers

```
int *p;    // pointer to int  
char *c;   // pointer to char  
float *f;  // pointer to float
```

Dereferencing a Pointer

Dereferencing means accessing the value stored at an address:

```
int x = 10;  
int *p = &x;  
  
printf("%d", *p);    // prints 10  
scanf("%d", p);     // takes user input into x
```

In the last line, the `&` sign is not needed because `p` holds the memory address of `x`.

Pointers and Memory Addresses

```
int x = 10, *px = &x;
```

The following are equivalent:

```
scanf("%d", &x);  
scanf("%d", px);
```

The following are equivalent:

```
printf("%d", x);  
printf("%d", *px);
```

This is due to `px` holding the memory address of `x`.

Example: Pointer Basics

```
1  #include <stdio.h>
2
3  int main(){
4      int x = 10;
5      int *p = &x;
6
7      printf("x = %d\n", x);
8      printf("Address of x = %d\n", p);
9      printf("Value through pointer = %d\n", *p);
10
11     return 0;
12 }
```

Pointer Arithmetic

Pointer Arithmetic

Adding or subtracting integers moves a pointer by multiples of the pointed type's size. For example, in the case of integers, it moves by 4.

```
1  #include <stdio.h>
2
3  int main(){
4      int x, *p = &x;
5
6      printf("Address of x = %d\n", p);
7      printf("Size of a single integer = %d\n",
8             ↪ sizeof(x));
9      printf("Address after adding 1 = %d\n", p+1);
10     printf("Address after adding 2 = %d\n", p+2);
11     return 0;
12 }
```

Array Name as Pointer

The name of an array holds the memory address to the first element of that array:

```
1  #include <stdio.h>
2
3  int main(){
4      int a[4] = {1,2,3,4}, *p = a;
5
6      printf("%d\n", *p);    // prints 1
7      printf("%d\n", *(p + 1)); // prints 2
8
9      printf("%d\n", *a);    // prints 1
10     printf("%d\n", *(a + 1)); // prints 2
11 }
```

Printing an Array

```
int arr[] = {1, 2, 3, 4, 5}
```

The following two snippets of code are equivalent:

```
for(int i=0; i<5; i++){  
    printf("%d", arr[i]);  
}
```

```
for(int i=0; i<5; i++){  
    printf("%d", *(arr+i));  
}
```

Pointer Difference

```
int a[5];  
int *p = &a[0];  
int *q = &a[3];  
  
int diff = q - p;    // diff = 3
```

Pointers with Function

Example: Pointer as Function Parameter

```
1  #include <stdio.h>
2
3  void increment(int *p){
4      *p = *p + 1;    // modify the original variable
5  }
6
7  int main(){
8      int x = 10;
9      increment(&x);
10
11     printf("x = %d\n", x);    // 11
12     return 0;
13 }
```


Example: Swap Using Pointers

```
1  #include <stdio.h>
2
3  void swap(int *a, int *b) {
4      int temp = *a;
5      *a = *b;
6      *b = temp;
7  }
8
9  int main() {
10     int x = 3, y = 7;
11     swap(&x, &y);
12     printf("x = %d, y = %d\n", x, y);
13     return 0;
14 }
```

Example: Array Passed to Function

```
1  #include <stdio.h>
2
3  void printArray(int arr[], int size){
4      for (int i = 0; i < size; i++){
5          printf("%d ", arr[i]);
6      }
7  }
8
9  int main(){
10     int a[5] = {10, 20, 30, 40, 50};
11     printArray(a, 5);
12 }
```

In line 3, `int arr[]` can be replaced with `int *arr`. They are equivalent.

Example: Modify Array in Function

```
1  #include <stdio.h>
2
3  void doubleValues(int arr[], int size){
4      for (int i = 0; i < size; i++){
5          arr[i] = arr[i] * 2;
6      }
7  }
8
9  int main() {
10     int a[4] = {1, 2, 3, 4};
11     doubleValues(a, 4);
12     for (int i = 0; i < 4; i++) {
13         printf("%d ", a[i]);    // 2 4 6 8
14     }
15 }
```