Pointers in C - A Complete Guide

1. Introduction to Pointers

What is a Pointer?

A pointer is a variable that stores the **memory address** of another variable.

Syntax:

```
int *ptr; // Declares a pointer to an integer
```

Basic Example:

```
#include <stdio.h>


int main() {
   int num = 10;
int[] numArray = {1,2,3,4}; // address of 1 => 10002
int **ptr = numArray;
//*ptr = address 2
//**ptr = value 2
   int *ptr = &num; // Pointer stores the address of num
 // ‘&’ reference operator to get the address of any variable
// and store it in the pointer variable
   printf("Value of num: %d\n", num);
   printf("Address of num: %p\n", &num);
   printf("Pointer ptr holds address: %p\n", ptr);
```

```
    printf("Value at pointer ptr: %d\n", *ptr); // Dereferencing
    return 0;
}
```

Key Takeaways:

✅ ptr stores the **address** of num ✅ *ptr (dereferencing) gives the **value** stored at that address

Exercise:

- Declare a pointer to a float and print its address and value.

---

2. Pointer Arithmetic

Incrementing Pointers

```
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30};
    int *ptr = arr;  // Points to the first element

    printf("First element: %d\n", *ptr); // 10
    ptr++; // Moves to the next element
    printf("Second element: %d\n", *ptr); // 20
    return 0;
}
```

Key Takeaways:

✅ ptr++ moves to the **next integer (4 bytes ahead)**

Exercise:

- Try decrementing (ptr--) and observe the changes.

---

3. Pointers and Arrays

Accessing an Array using Pointers

```
#include <stdio.h>

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int *ptr = arr;  // Points to first element

    for (int i = 0; i < 5; i++) {
        printf("Element %d: %d\n", i, *(ptr + i));
    }

    return 0;
}
```

Key Takeaways:

✅ arr[i] is the same as *(arr + i)

Exercise:

- Write a function that finds the sum of an array using pointers.

---

4. Pointers and Functions

Passing Pointers to Functions

```
#include <stdio.h>


void increment(int *p) {
   (*p)++; // Modify the actual value
}
int main() {
   int num = 10;
   increment(&num);
   printf("Updated value: %d\n", num);
   return 0;
}
```

Key Takeaways:

✅ Modifies the original value using a pointer

Exercise:

- Write a function to swap two numbers using pointers.

5. Dynamic Memory Allocation (malloc, free)

Allocating Memory Dynamically

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr = (int *)malloc(sizeof(int)); // Allocate memory for an integer
    *ptr = 100;
    printf("Dynamically allocated value: %d\n", *ptr);
    free(ptr); // Free allocated memory
    return 0;
}
```

Key Takeaways:

✅ malloc allocates memory dynamically ✅ Always use free(ptr) to prevent memory leaks

Exercise:

- Allocate memory for an array and take input from the user.

---

6. Pointers to Structures

Example:

```
#include <stdio.h>

struct Student {
   char name[20];
   int age;
};

int main() {
   struct Student s1 = {"Alice", 20};
// s1.name, s1.age
   struct Student *ptr = &s1;
   printf("Student Name: %s\n", ptr->name);
   printf("Student Age: %d\n", ptr->age);
   return 0;
}
```

Key Takeaways:

✅ ptr->field is the same as (*ptr).field

Exercise:

- Create an array of structures and use pointers to access them.

---

7. Advanced: Function Pointers

Example:

```
#include <stdio.h>

void greet1(string name) {
   printf("Hello %s\n", name);
}
void greet2(string name) {
   printf("Hello %s\n", name);
}

Void welcome(void *funcp){
        String name = “Alice”;
Funcp(name);
}

int main() {
   void (*funcPtr)(string) = greet; // Pointer to function
   funcPtr(“name”); // Calls the function
   funcPtr = greet2;
   funcPtr(“name”);
   return 0;
}
```

Key Takeaways:

✅ funcPtr holds the address of a function ✅ Useful in callback mechanisms

Exercise:

- Write a function pointer to call different functions dynamically.
