Pointers in C – The Ultimate Guide

Table of Contents

1. Introduction to Pointers

1.1 What is a Pointer?

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

Example:

c
Copy
Download
```c
int num = 42;
int *ptr = &num;  // ptr stores the address of num
```

1.2 Why Use Pointers?

- Efficient memory access
- Dynamic memory allocation
- Passing by reference in functions
- Building complex data structures

1.3 Memory Addresses & Variables

- Every variable has a memory address.
- `&` retrieves the address.
- `*` dereferences (accesses the value at the address).

Example:

c
Copy
Download
```c
int x = 10;
printf("Address of x: %p\n", &x);
printf("Value of x: %d\n", *(&x));
```

1.4 Common Mistakes

- Uninitialized pointers → Undefined behavior
- Dangling pointers → Accessing freed memory
- Null dereference → Crash

2. Pointer Arithmetic and Arrays

2.1 Pointer Increment/Decrement

- `ptr + 1` moves to the next memory location (based on data type size).

Example:

c
Copy
Download
```c
int arr[3] = {10, 20, 30};
int *ptr = arr;

printf("%d\n", *ptr);      // 10
printf("%d\n", *(ptr+1));  // 20
```

2.2 Array-Pointer Relationship

- `arr[i]` is equivalent to `*(arr + i)`.
- Array names decay into pointers when passed to functions.

Example:

c
Copy
Download
```c
void printArray(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
}
```

3. Pointers to Pointers (Double Pointers)

3.1 Concept

A pointer that stores the address of another pointer.

Example:

c
Copy
Download
```c
int x = 5;
int *p = &x;
int **pp = &p;

printf("%d", **pp);  // Output: 5
```

3.2 Use Cases

- Dynamic 2D arrays
- Modifying pointers inside functions

4. Function Pointers

4.1 Definition

A pointer that points to a function instead of a variable.

Example:

c
Copy
Download
```c
int add(int a, int b) { return a + b; }
int (*funcPtr)(int, int) = add;

printf("%d", funcPtr(5, 3));  // Output: 8
```

5. Void Pointers (Generic Pointers)

5.1 Introduction

- Can hold any data type’s address.
- Must be typecasted before dereferencing.

Example:

c
Copy
Download
```c
int num = 10;
void *ptr = &num;
printf("%d", *(int *)ptr);  // Output: 10
```

6. Dynamic Memory Allocation

6.1 `malloc`, `calloc`, `free`

- `malloc` allocates uninitialized memory.
- `calloc` initializes memory to zero.
- `free` releases memory.

Example:

c
Copy
Download

```c
int *arr = malloc(5 * sizeof(int));
free(arr);
```

7. Common Pitfalls & Debugging

7.1 Memory Leaks

- Forgetting to `free()` allocated memory.

7.2 Tools

- Valgrind → Detects leaks.
- GDB → Debugs pointer issues.

8. Advanced Topics

8.1 Pointers and Structures

c
Copy

Download
```c
struct Node {
   int data;
   struct Node *next;
};
```

8.2 Linked Lists

- Singly, Doubly, Circular Lists.

9. Real-World Applications

9.1 Embedded Systems

- Memory-mapped I/O registers.

9.2 OS Development

- Kernel data structures.

10. Interview Questions & Exercises

10.1 Practice Problems

1. Reverse an array using pointers.
2. Implement a linked list.
3. Find memory leaks in given code.

Swap Two Numbers

c

Copy

Download

```
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5, y = 10;
    swap(&x, &y);
    printf("x = %d, y = %d\n", x, y); // x = 10, y = 5
    return 0;
}
```

2. Pointer Arithmetic

Sum of Array Elements

c

Copy

Download

```
#include <stdio.h>

int sumArray(int *arr, int size) {
    int sum = 0;
```

```c
    for(int i = 0; i < size; i++) {
        sum += *(arr + i);
    }
    return sum;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    printf("Sum: %d\n", sumArray(arr, 5)); // Sum: 15
    return 0;
}
```

3. String Operations

String Length (strlen)

c

Copy

Download

```c
#include <stdio.h>

int strLen(char *str) {
    char *ptr = str;
    while(*ptr != '\0') ptr++;
    return ptr - str;
}

int main() {
    char s[] = "Hello";
    printf("Length: %d\n", strLen(s)); // Length: 5
    return 0;
}
```

4. Dynamic Memory Allocation

Dynamic 2D Array

c

Copy

Download

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

int main() {
    int rows = 3, cols = 3;
    int **matrix = (int **)malloc(rows * sizeof(int *));

    for(int i = 0; i < rows; i++) {
        matrix[i] = (int *)malloc(cols * sizeof(int));
        for(int j = 0; j < cols; j++) {
            matrix[i][j] = i + j;
        }
    }

    // Print matrix
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    // Free memory
    for(int i = 0; i < rows; i++) free(matrix[i]);
    free(matrix);
    return 0;
}
```

5. Function Pointers

Calculator Using Function Pointers

c

Copy

Download

```c
#include <stdio.h>

int add(int a, int b) { return a + b; }
int subtract(int a, int b) { return a - b; }

int calculate(int (*op)(int, int), int x, int y) {
    return op(x, y);
}

int main() {
    printf("5 + 3 = %d\n", calculate(add, 5, 3));       // 8
    printf("5 - 3 = %d\n", calculate(subtract, 5, 3));  // 2
    return 0;
}
```

6. Advanced Pointer Concepts

Linked List Implementation

c

Copy

Download

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

struct Node {
    int data;
    struct Node *next;
};

void append(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if(*head == NULL) {
        *head = newNode;
        return;
    }

    struct Node *last = *head;
    while(last->next != NULL) last = last->next;
    last->next = newNode;
}

void printList(struct Node *node) {
    while(node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node *head = NULL;
    append(&head, 10);
    append(&head, 20);
    append(&head, 30);
    printList(head); // 10 -> 20 -> 30 -> NULL
    return 0;
}
```

7. Pointer Pitfalls

Dangling Pointer Example

c

Copy

Download

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

int main() {
    int *ptr = (int *)malloc(sizeof(int));
    *ptr = 5;
    free(ptr); // ptr is now dangling
    // printf("%d\n", *ptr); // Undefined behavior!
    ptr = NULL; // Proper fix
    return 0;
}
```

8. Embedded Systems Example

Memory-Mapped Register Access

c

Copy

Download

```c
#include <stdint.h>

#define GPIO_PORT ((volatile uint32_t *)0x40020000)

int main() {
    *GPIO_PORT = 0x1; // Turn on LED
    uint32_t status = *GPIO_PORT; // Read status
    return 0;
}
```

How to Compile & Run

1. Save each code snippet in a .c file (e.g., swap.c)
2. Compile:
   bash

   Copy

   Download

   ```bash
   gcc swap.c -o swap
   ```
3. Run:
   bash

   Copy

Download

```
./swap
```

Key Concepts Covered

Concept	Example Programs
Basic Pointers	**Swap, Sum Array**
String Operations	**strlen, strcpy**
Dynamic Memory	**2D Array, Linked List**
Function Pointers	**Calculator**
Embedded Systems	**Memory-Mapped I/O**