3. Pointer Arithmetics

3. Pointer Arithmetic

Pointers can be incremented, decremented, and compared. When you perform arithmetic on pointers, the operation takes into account the size of the data type being pointed to.

3.1 Basic Pointer Arithmetic Operations

Operation	Description	Example
`ptr++`	Move to next element	`ptr = ptr + 1`
`ptr--`	Move to previous element	`ptr = ptr - 1`
`ptr + n`	Move n elements forward	`ptr = ptr + 3`
`ptr - n`	Move n elements backward	`ptr = ptr - 2`
`ptr2 - ptr1`	Distance between pointers	Number of elements

3.2 How Pointer Arithmetic Works

When you add 1 to a pointer, it doesn't increase by 1 byte—it increases by the size of the data type:

```
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int *ptr = arr;  // Points to first element

    printf("Address of ptr: %p, Value: %d\n", (void*)ptr, *ptr);

    ptr++;  // Move to next integer (adds sizeof(int) bytes)
    printf("Address of ptr: %p, Value: %d\n", (void*)ptr, *ptr);

    ptr++;  // Move to next integer
    printf("Address of ptr: %p, Value: %d\n", (void*)ptr, *ptr);
```

```
    return 0;
}

/* Output (addresses will vary):
Address of ptr: 0x7ffd5c8e4a20, Value: 10
Address of ptr: 0x7ffd5c8e4a24, Value: 20  (increased by 4 bytes for int)
Address of ptr: 0x7ffd5c8e4a28, Value: 30  (increased by 4 bytes again)
*/
```

Memory Layout:

```
Array: arr[] = {10, 20, 30, 40, 50}

        ┌────┬────┬────┬────┬────┐
        │ 10 │ 20 │ 30 │ 40 │ 50 │
        └────┴────┴────┴────┴────┘

         ↑    ↑    ↑
       ptr  ptr+1 ptr+2
      0x100 0x104 0x108  (assuming 4-byte int)
```

3.3 Pointer Arithmetic Examples

```
#include <stdio.h>

int main() {
    int numbers[] = {100, 200, 300, 400, 500};
    int *ptr = numbers;

    // Access elements using pointer arithmetic
    printf("First element: %d\n", *ptr);           // 100
    printf("Second element: %d\n", *(ptr + 1));    // 200
    printf("Third element: %d\n", *(ptr + 2));     // 300
    printf("Fifth element: %d\n", *(ptr + 4));     // 500

    // Equivalent array notation
    printf("\nUsing array notation:\n");
    printf("First element: %d\n", ptr[0]);         // 100
    printf("Second element: %d\n", ptr[1]);        // 200
```

```
    printf("Third element: %d\n", ptr[2]);         // 300

    // Distance between pointers
    int *start = &numbers[0];
    int *end = &numbers[4];
    printf("\nDistance between pointers: %ld elements\n", end - start);

    return 0;
}
```

3.4 Valid and Invalid Pointer Operations

Valid Operations:

```
int arr[5];
int *ptr = arr;

ptr++;           // Valid: increment pointer
ptr--;           // Valid: decrement pointer
ptr = ptr + 3;   // Valid: add integer to pointer
ptr = ptr - 2;   // Valid: subtract integer from pointer
int diff = ptr2 - ptr1;  // Valid: subtract two pointers (same type)
if (ptr1 < ptr2) { }     // Valid: compare pointers
```

Invalid Operations:

```
ptr = ptr * 2;    // INVALID: cannot multiply pointers
ptr = ptr / 2;    // INVALID: cannot divide pointers
ptr = ptr + ptr2; // INVALID: cannot add two pointers
```

Revision #1
Created 2025-10-01 03:26:29 UTC by DS
Updated 2025-10-01 03:26:40 UTC by DS