Pointers in C - Basic Notes

A pointer in C is a variable that stores the memory address of another variable. Pointers are a powerful feature in C, allowing direct memory access and manipulation, efficient array handling, and dynamic memory management.

---

1. Declaration of Pointers

A pointer is declared by specifying the data type it will point to, followed by an asterisk (`*`) before the pointer name.

```c
int *ptr;    // A pointer to an integer
char *chPtr; // A pointer to a character
```

- Here, `ptr` is a pointer to an integer and `chPtr` is a pointer to a character.

2. Pointer Initialization

Pointers are typically initialized by assigning them the address of another variable using the address-of operator (`&`).

```c
int num = 10;
int *ptr = &num; // Pointer 'ptr' stores the address of variable 'num'
```

- `ptr` now holds the address of `num`, and `*ptr` will refer to the value of `num`.

3. Dereferencing Pointers

Dereferencing a pointer means accessing the value stored at the memory address the pointer holds. This is done using the dereference operator (`*`).

c

```
printf("Value of num: %d\n", *ptr);  // Outputs 10
```

- `*ptr` gives the value stored at the address `ptr` points to, which is the value of `num`.

4. Pointer Arithmetic

You can perform arithmetic on pointers to move through memory locations, especially with arrays.

- Incrementing a pointer (`ptr++`) moves it to the next memory location, based on the data type size.

c

```
int arr[3] = {10, 20, 30};
int *ptr = arr;
ptr++;  // Now points to arr[1], which is 20
```

- Pointer arithmetic is scaled by the size of the data type the pointer is pointing to.

5. Pointers and Arrays

An array name acts as a pointer to its first element. For example:

c

```
int arr[3] = {10, 20, 30};
int *ptr = arr;
printf("%d\n", *ptr);   // Prints 10 (first element)
printf("%d\n", *(ptr + 1)); // Prints 20 (second element)
```

6. Null Pointer

A null pointer points to nothing. It is good practice to initialize pointers with `NULL` if they don’t have an address yet.

c

```
int *ptr = NULL;
if (ptr == NULL) {
    printf("Pointer is null.\n");
}
```

7. Void Pointer

A void pointer is a generic pointer that can point to any data type, but it cannot be dereferenced directly. It must be cast to another pointer type before dereferencing.

```c
void *ptr;
int num = 5;
ptr = &num;   // ptr can point to any data type
printf("%d\n", *(int *)ptr);  // Cast ptr to int* before dereferencing
```

8. Pointer to Pointer (Double Pointers)

A pointer to a pointer stores the address of another pointer. This can be useful for dynamically allocating memory for multidimensional arrays.

```c
int num = 10;
int *ptr = &num;
int pptr = &ptr;  // pptr is a pointer to ptr
printf("%d\n", pptr);  // Outputs 10
```

9. Function Pointers

A function pointer points to the address of a function and can be used to call functions dynamically.

c

```
void myFunc() {
    printf("Function called\n");
}
void (*funcPtr)() = myFunc;  // Pointer to function
funcPtr();  // Calls myFunc
```

10. Dynamic Memory Allocation

Pointers are essential in dynamic memory allocation. Functions like `malloc()`, `calloc()`, `realloc()`, and `free()` use pointers to manage memory at runtime.

c

```
int *ptr = (int *)malloc(sizeof(int));  // Allocates memory for an integer
*ptr = 100;
printf("%d\n", *ptr);  // Outputs 100
free(ptr);  // Frees the allocated memory
```

Summary:

- Pointers store memory addresses.
- The `*` operator is used for dereferencing, while `&` gives the address of a variable.
- Pointers allow dynamic memory management, pointer arithmetic, and efficient array handling.

These are foundational concepts for working with pointers in C. They are especially useful for advanced topics like data structures, dynamic arrays, and memory management.