

Pointers in C Programming

① Understanding Pointers

- A pointer is a variable that stores the memory address of another variable.
- Instead of storing the actual value, it stores the address where the value is kept in memory.
- Pointers are widely used in C for dynamic memory allocation, arrays, functions, and more. They help in efficient memory management and allow indirect access to data.

Example:

```
int a = 10;  
a 10 → (stored at address 1000)  
  
int *p = &a;  
p 1000 → (stores address of a)
```

Here:

- a is a variable with value 10.
- &a gives the address of a (1000).
- p is a pointer that stores this address.

② Accessing the Address of a Variable

- In C, we use the & (address-of) operator to get the memory address of a variable.
- This address can be stored in a pointer variable.
- The * (dereference) operator is used to access the value stored at the address held by the pointer.

Output (example):

```
Address of a: 0x7ffeefbff4c  
Value of p: 0x7ffeefbff4c  
Value of a: 25  
Value using pointer: 25
```

Example:

```
#include <stdio.h>  
int main() {  
    int a = 25;  
    int *p = &a;    // p stores address of a  
    printf("Address of a: %p\n", &a);  
    printf("Value of p: %p\n", p);  
    printf("Value of a: %d\n", a);  
    printf("Value using pointer: %d\n", *p);  
    return 0;  
}
```

③ Declaration and Initialization of Pointer Variables

(a) Declaration

- To declare a pointer variable, we use the * symbol with the data type.
- Syntax:

```
data_type *pointer_name;
```

• Example:

```
int *p; // p is a pointer to an integer
```

Note:

- Always initialize pointers to avoid undefined behavior.
- The size of a pointer depends on the system (e.g., 4 bytes on 32-bit, 8 bytes on 64-bit).
- Use %p in printf() to print addresses.

(b) Initialization

- A pointer should be initialized with a valid address (using & operator) or NULL if it is not pointing to any variable.
- Syntax:

```
pointer_name = &variable;
```

or

```
pointer_name = NULL;
```

• Example:

```
int a = 10;  
int *p = &a; // p points to a  
int *q = NULL; // q is a null pointer
```

④ Accessing a Variable through its Pointer

- Once a pointer is initialized with the address of a variable, we can access the value of that variable using the dereference operator (*).
- The * operator gives the value stored at the address pointed by the pointer.

Syntax:

*pointer_name

Example:

```
#include <stdio.h>
int main() {
    int a = 10;
    int *p;
    p = &a;      // p stores address of a
    printf("Value of a = %d\n", a);
    printf("Value using pointer = %d\n", *p);
    return 0;
}
```

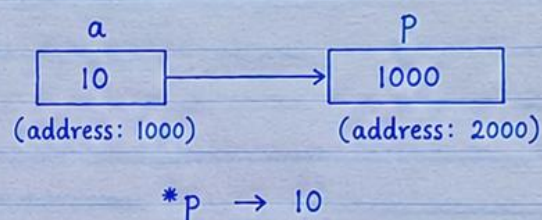
Output:

Value of a = 10
Value using pointer = 10

Explanation:

- p stores the address of a.
- *p gives the value stored at that address, which is 10.
- So, *p is another way to access the variable a.

Memory View (example):



⑤ Pointers and Arrays

- Arrays and pointers are closely related in C.
- The name of an array represents the address of its first element.
- We can use a pointer to access array elements using the same subscript notation.
- If p is a pointer to the first element of an array, then ***(p+i)** is equivalent to **a[i]**.

Syntax:

*(array_name + index) == array_name[index]

Example 1: Accessing array elements using a pointer

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int *p;
    p = arr;      // p points to the first element
    printf("Using pointer: %d %d %d %d %d\n",
           *(p+0), *(p+1), *(p+2), *(p+3), *(p+4));
    return 0;
}
```

Output:

Using pointer: 10 20 30 40 50

Explanation:

- arr is the base address of the array.
- p = arr; makes p point to the first element of arr.
- *(p+i) or p[i] gives the i-th element of the array.

Example 2: Using array and pointer together

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int *p = arr;
    for (int i = 0; i < 5; i++) {
        printf("%d ", *(p+i));
    }
    return 0;
}
```

Output:

10 20 30 40 50

File Handling in C Programming

① File Operations

File handling in C means performing operations on files such as creating, opening, reading, writing, and closing files.

Common File Operations:

1. Open a file - `fopen()`
2. Read from a file - `fscanf()` / `fgets()` / `fread()`
3. Write to a file - `fprintf()` / `fputs()` / `fwrite()`
4. Close a file - `fclose()`
5. End of file check - `feof()`
6. Error handling - `ferror()`

File Opening Modes:

Mode	Description
"r"	Open for reading (file must exist).
"w"	Open for writing (creates new file or truncates existing file).
"a"	Open for appending (creates new file if not exist).
"r+"	Open for reading and writing.
"w+"	Open for reading and writing (truncates).
"a+"	Open for reading and writing (appends).

Important Functions and Modes:

Function	Description
<code>fopen()</code>	Opens a file and returns a file pointer.
<code>fclose()</code>	Closes the opened file.
<code>fscanf()</code>	Reads formatted data from a file.
<code>fprintf()</code>	Writes formatted data to a file.
<code>fgets()</code>	Reads a line from a file.
<code>fputs()</code>	Writes a string to a file.
<code>fread()</code>	Reads binary data from a file.
<code>fwrite()</code>	Writes binary data to a file.
<code>feof()</code>	Checks end of file.
<code>ferror()</code>	Checks for file errors.

Example:

```
#include <stdio.h>
int main() {
    FILE *fp;
    fp = fopen("test.txt", "r"); // open file for reading
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    fclose(fp); // close file
    return 0;
}
```

② Processing a Data File in C Programming

We can process a data file by opening the file, reading data from it, performing some operation, and then writing the result to another file (or the same file).

Example: Read numbers from a file, calculate the sum, and display the result.

Sample Input File (data.txt):

```
10
20
30
40
50
```

Program:

```
#include <stdio.h>
int main() {
    FILE *fp;
    int num, sum = 0;
    fp = fopen("data.txt", "r");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    while (fscanf(fp, "%d", &num) != EOF) {
        sum += num;
    }
    fclose(fp);
    printf("Sum of numbers = %d\n", sum);
    return 0;
}
```

How it works:

- open the file data.txt in read mode.
- read each number using `fscanf()`.
- add the numbers to sum.
- close the file.
- print the total sum.

Output:

```
Sum of numbers = 150
```

Note:

- The file data.txt should be present in the same folder as the program.
- If the file is not found, an error message will be displayed.