5. Pointers and Functions

Pointers are essential for functions to modify variables from the calling code and to work efficiently with arrays.

5.1 Pass by Value vs Pass by Reference

Pass by Value (Without Pointers):

```
#include <stdio.h>

void tryToChange(int x) {
    x = 100;  // Only changes local copy
    printf("Inside function: x = %d\n", x);
}

int main() {
    int num = 50;
    printf("Before function: num = %d\n", num);
    tryToChange(num);
    printf("After function: num = %d\n", num);  // num unchanged!
    return 0;
}

/* Output:
Before function: num = 50
Inside function: x = 100
After function: num = 50
*/
```

Pass by Reference (With Pointers):

```
#include <stdio.h>

void actuallyChange(int *x) {
    *x = 100;  // Changes the original variable
    printf("Inside function: *x = %d\n", *x);
}
```

```
int main() {
    int num = 50;
    printf("Before function: num = %d\n", num);
    actuallyChange(&num);  // Pass address
    printf("After function: num = %d\n", num);  // num changed!
    return 0;
}

/* Output:
Before function: num = 50
Inside function: *x = 100
After function: num = 100
*/
```

5.2 Why scanf() Requires &

Now we understand why `scanf()` needs the `&` operator:

```
int age;
scanf("%d", &age);  // Pass address so scanf can modify age

// What happens inside scanf (simplified):
void scanf(const char *format, int *ptr) {
    // Read input value
    int value = /* read from keyboard */;
    *ptr = value;  // Store in the address provided
}
```

5.3 Functions Returning Multiple Values

Since C functions can only return one value, pointers allow us to "return" multiple values:

```
#include <stdio.h>

// Function to find both quotient and remainder
void divide(int dividend, int divisor, int *quotient, int *remainder) {
    *quotient = dividend / divisor;
    *remainder = dividend % divisor;
```

```
}

int main() {
    int a = 17, b = 5;
    int q, r;

    divide(a, b, &q, &r);

    printf("%d divided by %d:\n", a, b);
    printf("Quotient: %d\n", q);
    printf("Remainder: %d\n", r);

    return 0;
}
```

5.4 Swapping Values Using Pointers

A classic example of pointer usage:

```
#include <stdio.h>

// WRONG: This doesn't swap the original variables
void wrongSwap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}

// CORRECT: This swaps the original variables
void correctSwap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int a = 10, b = 20;

    printf("Before swap: a = %d, b = %d\n", a, b);
```

```
    wrongSwap(a, b);
    printf("After wrongSwap: a = %d, b = %d\n", a, b);  // No change

    correctSwap(&a, &b);
    printf("After correctSwap: a = %d, b = %d\n", a, b);  // Swapped!

    return 0;
}
```

5.5 Passing Arrays to Functions

When passing arrays to functions, you're actually passing a pointer:

```
#include <stdio.h>

// These function declarations are equivalent:
void printArray1(int arr[], int size);
void printArray2(int *arr, int size);

void printArray1(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int numbers[] = {10, 20, 30, 40, 50};
    int size = sizeof(numbers) / sizeof(numbers[0]);

    printArray1(numbers, size);

    return 0;
}
```

Important Note:

```
void function(int arr[]) {
    // Inside function, sizeof(arr) gives size of pointer, not array!
    int size = sizeof(arr);  // This gives 8 (size of pointer on 64-bit)
    // WRONG! Does not give array size
```

```
}

int main() {
    int numbers[5] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof(numbers[0]);  // This is correct
    function(numbers);
    return 0;
}
```

Revision #1
Created 2025-10-01 03:27:29 UTC by DS
Updated 2025-10-01 03:27:49 UTC by DS