Functions in C - Module - 4

Call by Value and Call by Reference in C

Actual Arguments and Formal Arguments

Actual Arguments:
1. Arguments which are mentioned in the function in the function call are known as calling function.
2. These are the values which are actual arguments called to the function. It can be written as constant , function expression on any function call which return a value . ex: funct (6,9) , funct (a,b)

Formal Arguments:
1. Arguments which are mentioned in function definition are called dummy or formal argument.
2. These arguments are used to just hold the value that is sent by calling function.
3. Formal arguments are like other local variables of the function which are created when function call starts and destroyed when end function.

In C programming, when a function is called, the arguments passed to it can be handled in two ways: **Call by Value** and **Call by Reference**. These two techniques define how the data is transferred from the calling function to the called function.

1.Call by Value

In **Call by Value**, the actual value of the argument is passed to the function. This means that any changes made to the parameter inside the function will not affect the actual argument in the calling function.

How it works:
- When you pass arguments to a function, a copy of the value is passed.
- Modifications made to the copy of the argument inside the function do not affect the original value outside the function.

Example:

```c
#include <stdio.h>

void addTen(int x) {
    x = x + 10;  // Changes only the local copy of x
}

int main() {
    int num = 5;
    addTen(num);  // Call by Value
    printf("Value of num: %d\n", num);  // Output will be 5
    return 0;
}
```

Explanation:

- The `num` variable holds the value `5`, which is passed to the function `addTen()`. However, since it's a copy of the value (not the original variable), the modification inside the function does not affect `num`. Therefore, the value of `num` remains `5` in `main()`.

Advantages:
- **Safety:** The original data is protected, as the function only works on a copy.
- **Simplicity:** Easy to use and understand for basic operations.

Disadvantages:
- **Inefficient for large data:** If large structures or arrays are passed, copying them can lead to inefficiency.

2. Call by Reference

In **Call by Reference**, the address (reference) of the actual argument is passed to the function. This means the function operates on the original data, so any changes made to the parameter inside the function will affect the actual argument in the calling function.

How it works:
- The function receives a reference to the original variable, not a copy.
- Any changes made to the reference affect the original variable directly.

Advantages:
- **Efficiency:** For large data structures or arrays, it's more efficient since no data is copied.

Introduction to 'C' Language

- **Direct Modification:** Changes made inside the function directly affect the original data.

Disadvantages:

- **Risk of unintended changes:** Since the function modifies the original data, it can lead to errors if not carefully handled.
- **Complexity:** Pointers can be harder to understand and use correctly for beginners.

Summary of Differences

Aspect	Call by Value	Call by Reference
What is passed?	A copy of the actual value	The address (reference) of the variable
Effect on original data	Does not modify the original data	Modifies the original data
Memory usage	More memory for creating a copy	More efficient, no need to copy large data
Complexity	Simpler to implement	Requires knowledge of pointers and memory addresses
Example use case	Small data, when you don't need to modify the original	Large data, when you need to modify the original data

In conclusion, **Call by Value** is simpler and safer for small and basic operations where you don't need to modify the original data. On the other hand, **Call by Reference** is more efficient and useful when you need to modify the actual arguments or deal with large datasets like arrays and structures.

Solved Example: Swapping of Two Numbers

Call by Reference - Swapping Two Numbers (modifies original):

Here is an example of using **Call by Reference** to swap two numbers (modifies the original numbers).

c

Copy code

```c
#include <stdio.h>

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

int main() {
    int x = 10, y = 20;
    swap(&x, &y);  // Call by reference, swap modifies original variables
    printf("After swap: x = %d, y = %d\n", x, y);
    return 0;
}
```

Explanation:

- The function `swap()` takes pointers to `x` and `y`, and since the function modifies the values at these memory addresses, the original variables in `main()` are changed.

Call by Value - Swapping Two Numbers (without modifying original):

Here is an example of using **Call by Value** to swap two numbers (although the original numbers will not be changed).

c Copy code

```c
#include <stdio.h>

void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
    printf("Inside function: a = %d, b = %d\n", a, b);
}

int main() {
    int x = 10, y = 20;
    swap(x, y);  // Call by value, swap inside function won't affect main variables
    printf("Inside main: x = %d, y = %d\n", x, y);
    return 0;
}
```

Explanation:

- Even though the function `swap()` changes `a` and `b` inside the function, it does not affect the values of `x` and `y` in the `main()` function because the values are passed by copy.