NOTE BOLD: DO NOT COPY THE TEXT HIGHLIGHTED IN YELLOW COLOR.

DATE: LAB REPORT NO.: **1** SET: **D**

TITLE OF THE PROGRAM: **USER-DEFINED FUNCTIONS IN C**
PASS BY VALUE & PASS BY REFERENCE

OBJECTIVES

I. To understand the concept of user-defined functions and implement the swapping of two numbers using the following techniques in the C programming language.
   1. pass-by-value
   2. pass-by-reference

REQUIREMENTS

3. C Compiler (e.g., GCC)
4. Computer System
5. IDE or Text Editor
6. OS compatible with the software

THEORY

User-defined functions in C allow programmers to define their own functions, which can be called multiple times within a program. These functions can have parameters that can be passed by value or by reference.

1. **Pass By Value**

When a parameter is passed by value, a copy of the parameter's value is passed to the function. Any changes made to the parameter within the function do not affect the original value in the calling function.

2. **Pass By Reference**

When a parameter is passed by reference, the address of the parameter is passed to the function. This allows the function to directly modify the original value in the calling function.

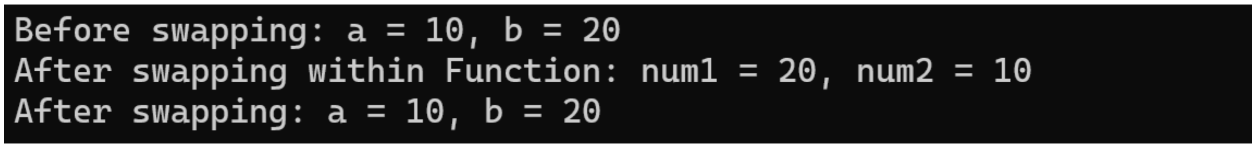
PROCEDURE (Program Code, Comment, and Output)

1. **Pass By Value**

Program Code:
```
#include <stdio.h>

void swapByValue(int, int);

int main()
{
```

```c
    int a = 10, b = 20;

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

    // The values of a and b will be passed by value
    swapByValue(a, b);

    printf("After swapping: a = %d, b = %d\n", a, b);

    return 0;
}

// This function swaps the values of num1 and num2
void swapByValue(int num1, int num2)
{
    int temp;

    // Swapping the values using a temporary variable
    temp = num1;
    num1 = num2;
    num2 = temp;

    // The swap is performed on local copies of the variables
    // The changes will not affect the original variables in the main
function
    printf("After swapping within Function: num1 = %d, num2 = %d\n",
num1, num2);

    // This is because the values are passed by value, not by
reference
}
```

Output:



```
Before swapping: a = 10, b = 20
After swapping within Function: num1 = 20, num2 = 10
After swapping: a = 10, b = 20
```

Explanation:

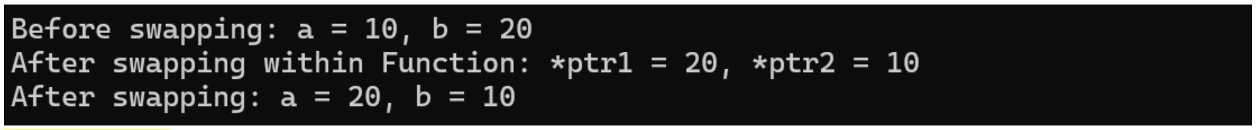
The code declares a main function and a swapByValue function. The main function initializes two variables, a and b, with the values 10 and 20 respectively. It then prints the values of a and b before swapping.

The swapByValue function takes two parameters, num1 and num2, representing the values to be swapped. Inside the function, a temporary variable temp is declared to hold the value of num1. Then, num1 is assigned the value of num2, and finally, num2 is assigned the value of temp.

However, this swap is performed on local copies of num1 and num2, so it doesn't affect the original variables in the main function.
After calling the swapByValue function, the main function prints the values of a and b again. Since the swap was performed on local copies within the swapByValue function, the values of a and b remain unchanged, resulting in the same output before and after the swap.

2. **Pass By Reference**

Program Code:
```
#include <stdio.h>

void swapByReference(int*, int*);

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

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

    // The address of a and b is passed to swapByReference()
    swapByReference(&a, &b);

    printf("After swapping: a = %d, b = %d\n", a, b);

    return 0;
}
    // The address of a and b is passed to pointers ptr1 and ptr2
void swapByReference(int* ptr1, int* ptr2)
{
    int temp;

    // Swap the values using pointers
    temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;

    printf("After swapping within Function: *ptr1 = %d, *ptr2 = %d\n",
*ptr1, *ptr2);
}
```
Output:


```
Before swapping: a = 10, b = 20
After swapping within Function: *ptr1 = 20, *ptr2 = 10
After swapping: a = 20, b = 10
```

Explanation:

The code defines a function swapByReference that takes two pointers to integers as arguments and swaps their values. In the main function, two variables a and b are declared and initialized with the values 10 and 20 respectively. The initial values of a and b are then printed.
Next, the swapByReference function is called with the addresses of a and b using the & operator. This allows the function to modify the values of a and b directly.
Inside the swapByReference function, a temporary variable temp is declared to hold the value of the first pointer (ptr1). The value of ptr1 is assigned to temp, then the value of the second pointer (ptr2) is assigned to ptr1, and finally, the value of temp is assigned to ptr2. This effectively swaps the values of the variables.
Back in the main function, after the call to swapByReference, the values of a and b have been swapped. The updated values are then printed, showing that the swap was successful.

CONCLUSION

The project involved implementing number swapping in C using user-defined functions and pass-by-value/pass-by-reference techniques. Pass-by-value preserved the original variable values, while pass-by-reference allowed direct modification within the function, offering flexibility for different program requirements.