SNS COLLEGE OF5 TECHNOLOGY

Coimbatore-35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

****

DEPARTMENT OF INFORMATION TECHNOLOGY

PROGRAMMING FOR PROBLEM SOLVING

UNIT – 4 CALL BY VALUE & CALL BY REFERENCE / RECURSION

Call by value in C

- **Call by value in C**
- In call by value method, the value of the actual parameters is copied into the formal parameters.
- In call by value method, we can not modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.

EXAMPLES

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and
      b in main
   swap(a,b);
   printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameters
     do not change by changing the formal parameters in call by value, a = 10, b = 20
}
void swap (int a, int b)
{
   int temp;
   temp = a;
   a=b;
   b=temp;
   printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a = 20,
     b = 10
}
```

Output

Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20

CALL BY REFERENCE IN C

In call by reference, the address of the variable is passed into the function call as the actual parameter.

The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.

In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

EXAMPLE

```
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b
      in main
   swap(&a,&b);
   printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual parameters d
      o change in call by reference, a = 10, b = 20
}
void swap (int *a, int *b)
{
   int temp;
   temp = *a;
   *a=*b;
   *b=temp;
   printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal parameters, a = 20, b
      = 10
}
```

Output

Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10

DIFFERENCE

Difference between call by value and call by reference in c

No.	Call by value	Call by reference
1	A copy of the value is passed into the function	An address of value is passed into the function
2	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

Recursion

- Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.
- Recursion may be a bit difficult to understand. The best way to figure out how it works is to experiment with it.

EXAMPLE

- ```
  int sum(int k);

  int main() {
    int result = sum(10);
    printf("%d", result);
    return 0;
  }

  int sum(int k) {
    if (k > 0) {
      return k + sum(k - 1);
    } else {
      return 0;
    }
  }
  ```
  ```

Example Explained

- When the sum() function is called, it adds parameter k to the sum of all numbers smaller than k and returns the result. When k becomes 0, the function just returns 0. When running, the program follows these steps:

Example Explained

- 10 + sum(9)
  10 + (9 + sum(8))
  10 + (9 + (8 + sum(7)))
  …
  10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)
  10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
- Since the function does not call itself when k is 0, the program stops there and returns the result

THANK YOU