Recursion in C

Recursion is the process by which a function calls itself. C language allows writing of such functions which call itself to solve complicated problems by breaking them down into simple and easy problems. These functions are known as recursive functions.
Syntax:

```
void recursive_function(){
  recursion();   // function calls itself
}
int main(){
  recursive_function();
}
```

Why is Recursion Used in C?

Recursion is commonly applied to handle complex tasks like traversing tree and graph structures. Well-known problems solved using recursion include factorial calculation, binary search, tree traversal, the Tower of Hanoi, and the eight queens problem in chess.

While recursive programs tend to be more concise, they are often harder to understand. Although the code may be shorter, recursion demands more processing resources due to the multiple function calls it requires.

Example:

```
#include <stdio.h>
int factorial(int n) {
    // Base case
    if (n == 0) {
        return 1;
    }
    // Recursive case
    return n * factorial(n - 1);
}
int main() {
    int num = 5;
    printf("Factorial of %d is %d\n", num, factorial(num));
    return 0;
}
```

Function with array in c

In C, functions can work with arrays by passing them as arguments. However, when passing an array to a function, only the pointer to the first element is passed (not the entire array), which allows for efficient memory usage.

Example:

```
#include <stdio.h>
// Function to print an array
void printArray(int arr[], int size) {
    for(int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

// Function to find the sum of elements in an array
int sumArray(int arr[], int size) {
    int sum = 0;
    for(int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}
int main() {
    int numbers[] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof(numbers[0]);
    printf("Array elements: ");
    printArray(numbers, size);
   int sum = sumArray(numbers, size);
    printf("Sum of array elements: %d\n", sum);
    return 0;
}
```

Function Call in C

A function call in C refers to invoking a function that is defined elsewhere in the code. Functions are used to perform specific tasks, and calling a function means transferring the control of the program to that function for execution. Once the function finishes, control is returned to the point after the function call.

Ways of Function call

Functions can be invoked in two ways:

- Call by Value
- Call by Reference.

1. Call by Value:

In call by value, a copy of the actual parameter's value is passed to the function. This means that changes made to the parameter inside the function do not affect the original argument in the calling function.

Example:

```
#include <stdio.h>
void changeValue(int x) {
    x = 100;  // Modifying x only affects the local copy
}
int main() {
    int num = 10;
    printf("Before function call: %d\n", num);

    changeValue(num);  // Call by value, passes a copy of num

    printf("After function call: %d\n", num);  // num is unchanged
    return 0;
}
```

2. Call by Reference:

In call by reference, instead of passing the value, we pass the address (or reference) of the variable to the function. This allows the function to modify the actual argument in the calling function.

To achieve a call by reference in C, we use pointers. A pointer stores the memory address of a variable, and by passing a pointer to a function, the function can directly access and modify the original variable's value.

Example:

```
#include <stdio.h>
void changeValue(int *x) {  // Accepting the pointer (reference) to the variable
    *x = 100;  // Modifying the actual value by dereferencing the pointer
}

int main() {
    int num = 10;
    printf("Before function call: %d\n", num);
   changeValue(&num);  // Passing the address of num to the function
    printf("After function call: %d\n", num);  // num is changed
```

```
    return 0;
}
```