- **Nesting of Functions**.

In some applications, we have seen that some functions are declared inside another function. This is sometimes known as nested function, but actually this is not the nested function. This is called the lexical scoping. Lexical scoping is not valid in C because the compiler is unable to reach correct memory location of inner function.

Nested function definitions cannot access local variables of surrounding blocks. They can access only global variables. In C there are two nested scopes the local and the global. So nested function has some limited use. If we want to create nested function like below, it will generate error.

```
#include<stdio.h>
main(void)
 {
   printf("Main Function");
   int my_fun()
  {
      printf("my_fun function");
      // defining another function inside the first
function.
      int my_fun2()
      {
         printf("my_fun2 is inner function");
      }
   }
   my_fun2();
}
```

OUTPUT: **undefined reference to `my_fun2'**

- **RECURSION :**
  Recursion is the process of a function calling itself.
  repeatedly till the given condition is satisfied. A function that calls itself directly or indirectly is called a recursive function and such kind of function calls are called recursive calls.

Recursion Example :

Adding two numbers together is easy to do, but adding a range of numbers is more complicated. In the following example, recursion is used to add a range of numbers together by breaking it down into the simple task of adding two numbers:

```
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;
  }
}
```

0 + 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

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.

the function does not call itself when `k` is 0, the program stops there and returns the result.

- **Functions with arrays :**

Syntax for Functions with Arrays in C:

Passing an Array to a Function:

One-Dimensional Array Two-Dimensional Arrays.

To pass an array to a function, you can declare the function parameter as an array. However, unlike other data types, you need to specify the size of the array in the function parameter list.

```
void functionName(dataType arrayName[size]) {
   // Function body
}
```

Example: Function to Print Elements of an Array:

Here's a simple example of a function that prints the elements of an integer array.

```
#include <stdio.h>
// Function to print elements of an integer array
void printArray(int arr[], int size) {
   for (int i = 0; i < size; i++) {
       printf("%d ", arr[i]);
   }
      printf("\n");
}
   int main() {
   int myArray[] = {1, 2, 3, 4, 5};
   int arraySize = sizeof(myArray) / sizeof(myArray[0]);
  // Call the function to print array elements
   printArray(myArray, arraySize);
   return 0;
}
```

OUTPUT:

1 2 3 4 5

this example, the `printArray` function takes an integer array `arr` and its size as parameters. Inside the function, a `for` loop is used to iterate through the array and print each element.
The `Main` function declares an integer array `myArray` and calculates its size using the `sizeof` operator. It then calls the `printArray` function with the array and its size as arguments.

- **Three Rules to Pass an array to a Function.**
  1. **The Function must be Called by passing only the name of the array.**
  2. **In the function definition the formal parameter must be an array type the size of the array does not need to be specified.**
  3. **The function prototype must show that the argument is an array.**

***Write a program that uses a function to sort an array of integers.**

A program to sort an array of integers using the function sort() is given. Its output clearly shows that a function can change the values in an array passed as an argument.

```
void sort(int m, int x[]);
main()
{
   int i;
   int marks[5] = {40,90,73,81,35};
   printf("marks before sorting\n");
   for(i = 0; i < 5; i++)
     printf("%d", marks[i]);
   printf("\n\n");
  sort (5, marks);
 printf("marks after sorting\n");
 for(i = 0; i < 5; i++)
 printf("%4d", marks[i]);
```

```
    printf("\n");
}
  void sort(int m, int x[])
{
   int i, j, t;
    for(i = 1; i<=m-1; i++)
     for(j= 1; j<=m-i; j++)
      if(x[j-1] >= x[j])
    {
       t= x[j-1];
       x[j-1] = x[j];
       x[j] = t;
   }
}
```

OUTPUT:

Marks before sorting : 40 90 73 81 35

Marks after sorting : 35 40 73 81 90

Notes: The format specifier %4d is used for the third argument to indicate that the value should be printed using the %d format conversion with a minimum field width of 4 characters.