Recursive Functions

- Recursion is the process of repeating items in a self-similar way.
- In C programming recursive functions are the one where a function is called by itself repeatedly.
- Whenever a recursion function is written, there are 2 things should be followed:
    - Identification of base case or termination condition
    - Where the function should be called

- Recursion is often used in implementations of the Backtracking algorithm. For example Sudoku.
- Recursion is applied to problems (situations) where you can break it up (reduce it) into smaller parts, and each part(s) looks similar to the original problem.

Example:
Factorial of a number

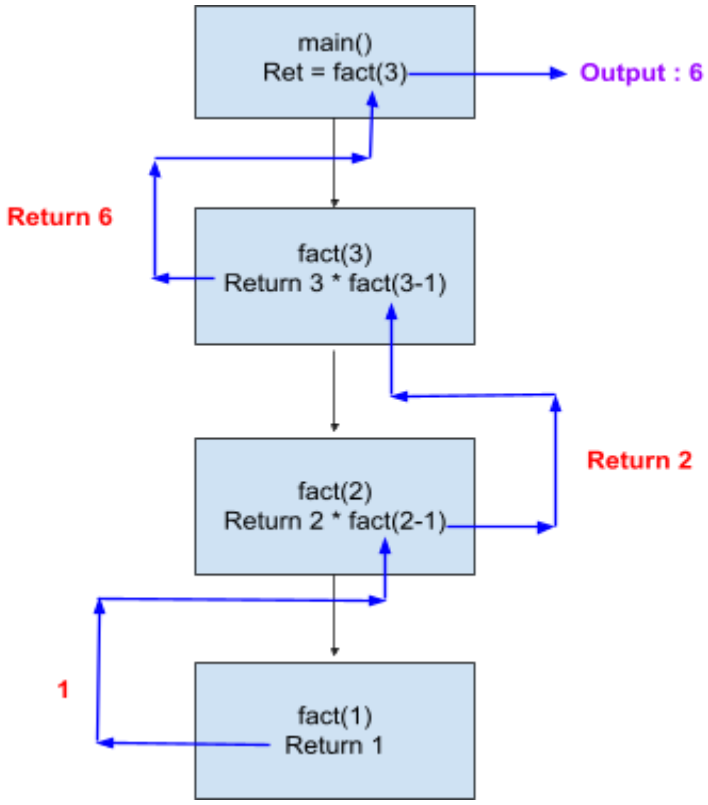
- Factorial of a number is found by multiplying the number by every number below it till 1.

- E.g, number = 3
  Factorial = 3 * 2 * 1 = 6

```
int main()
{
   int ret;
   ret = factorial(3);
   printf("Factorial of 3 is %d\n", ret);
   return 0;
}
```

```
int factorial(int number)
{
    if (number <= 1) /* Base Case */
    {
        return 1;
    }
    else /* Recursive Case */
    {
       return number * factorial(number – 1);
    }
}
```

- The recursion execution will looks as described in the below flow:



Recursion vs normal loop function

- The ultimate question here is when to use recursion? Or when to go for a normal loop function?
- The answer total depends on the resources and requirement, which is based on 2 factors: time and space
- For instance if you use recursion the space complexity will be for above factorial program:
    - Stackframe fact(3) will consume

      4 bytes of integer and 4 bytes of return value = 8bytes

      fact(2) = 8bytes

      fact(1) = 8bytes

- Total of 24 bytes will be used for finding factorials of 3. It will increase if the value increases

- Same program if its written using normal loop then

```
int factorial(int n)
{
    int fact = 1, i;
    for(i = 1; i <= n; i++)
    {
        fact *= i;
   }
   return fact;
}
```

Space complexity: 8(int fact + i) + 4(parameter) + 4(return) = 18 bytes
This is fixed despite passing the larger value.

- Time complexity will be:
- For recursion:
    - Creation of 3 stack frame : 3 machine cycle
    - Returning 3 values : 3 machine cycle
    - Total it will take 6 machine cycle

- For normal loop:
    - 1 stackframe + Loop (1 initialisation + 4 condition evaluation + 3 increment + 3 time statement execution + 1 return value)
    - Total it will take 13 machine cycles and this will increase if the value is larger.

- So as you can see in the above analysis if the requirement has memory constraints like an embedded system then normal loop function will be preferred.

- If the constraint is on time then recursion is preferred as recursion is faster than the loop.

Recursion with static variables

- In recursion function variables can be static variables but memory will be in the data segment.

```
void print()
{
        static int self_call = 0;
        if(self_call++ != 99)
        {
                printf("%d\n", self_call);
                print();
        }
        else
        {
                printf("End\n");
        }
}
```

- In the above example self_call will be in the data segment so one instance of memory will be created at the beginning, next call onwards it will make use of the same memory.
- Static can be used when the requirement says that all the recursive calls should make use of the same variable.
- Using auto local variables will result in stack overflow error, because its creating a new variable for each call which will have the same value each time.

Recursive main() function

- Main function can be called recursively, like below example

```
int main()
```

```
{
   static int self_Call = 0;

   if(self_call != 5)
   {
      printf(“%d\n”,self_call);
      main();
    }
   else
   {
      printf(“End\n”);
}
```

- Here, main is called recursively, but without passing any argument. The function which is called recursively without any argument such functions are not true recursive functions. Therefore main can be called recursively but its not true recursive function.

- And also Making use of static variables in the recursion will make the function as not a true recursive function.