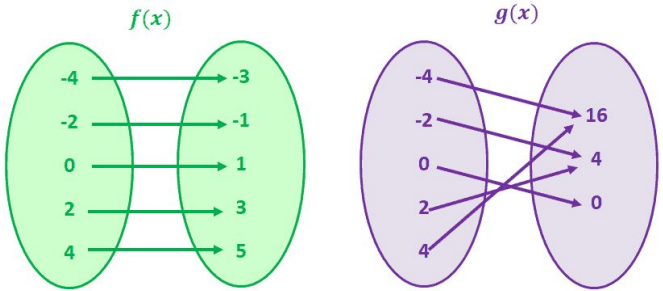



# Functions

**Sumaiyah Zahid**

https://www.hackerrank.com/domains/c

https://leetcode.com/problemset/

# Function Definition

A function is a black box which relates input to output.

INPUT

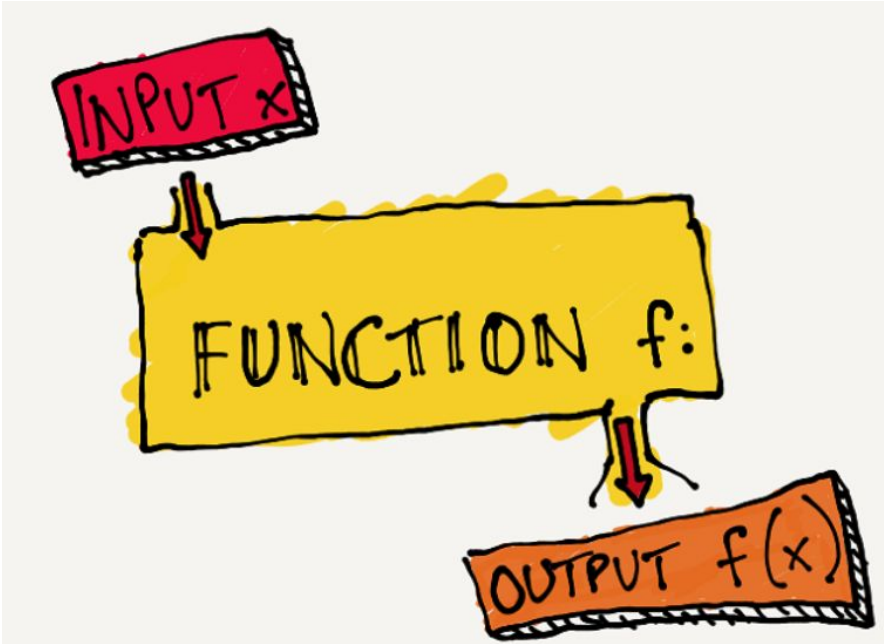
MYSTERY

OUTPUT

Outer World don’t know what happens inside.
Your relatives think it’s fun but in actual it isn’t.

# Function

```
Output  Function_Name ( Input )
int main(void) //void = input
{
        return 0; // Output
}
Function name = main
Output datatype = int
```



# Input / Arguments / Parameters

```
printf(“Hello World”);

scanf (“%d”, &a);
```

printf and scanf are functions.

Inputs are separated by commas.

Inputs are also known as arguments or parameters.

# Output / Return Values

```
a=printf(“Hello World”);

b=scanf (“%d”, &a);

printf(“%d %d”, a,b);   // Try this
```

printf returns an integer value, which is the total number of printed characters.

scanf returns an integer value, which is the total number of inputs.

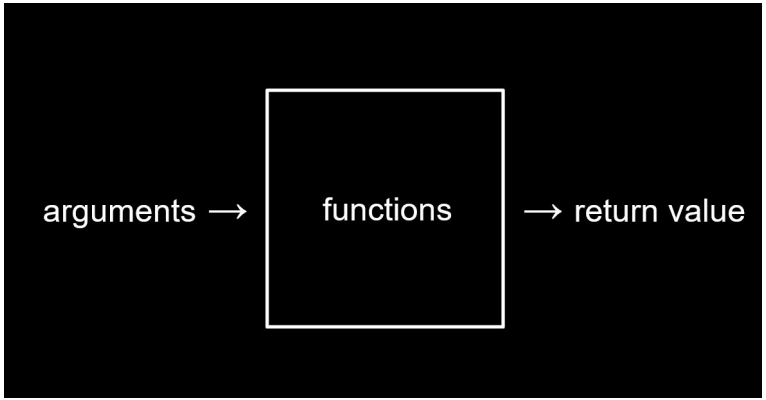
# Function Definition

A block of code that performs a specific task.

They avoid duplicating codes.

Function help you break down a big project.

- Library Functions
- User - Defined Functions

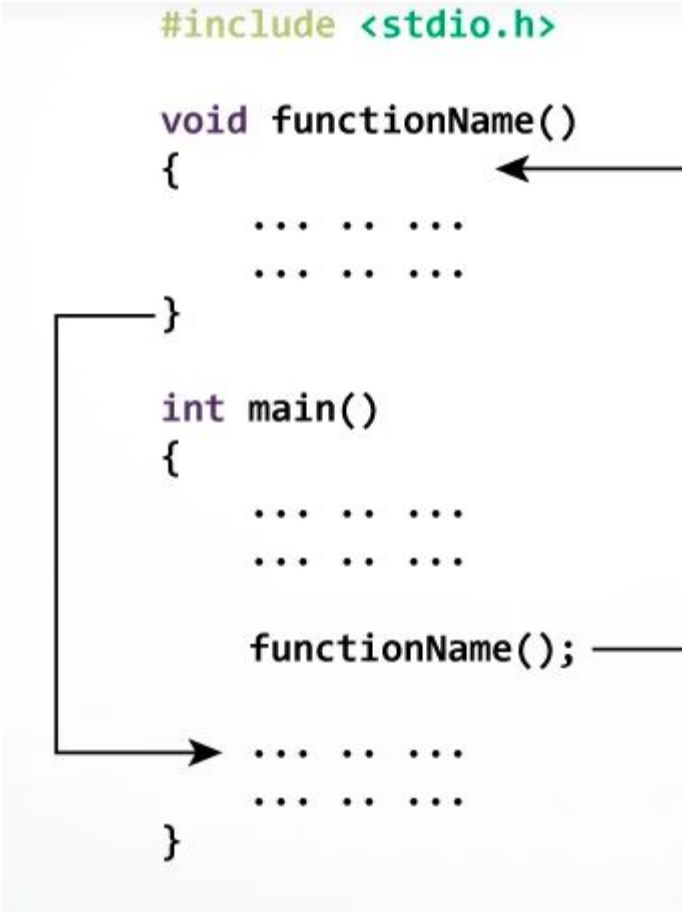


# User - Defined Functions

```
ReturnType  FunctionName (Arguments);

int add (int a, int b)
```

3 important things:

- Function declaration or prototype
- Function call
- Function definition



```c
#include <stdio.h>
void smile();     // Function Declaration or Prototype
int main()
{
   smile();       // Function Call
   return 0;
}

void smile()      // Function Definition
{
   printf("\nSmile, and the world smiles with you...");
}
```

```c
#include <stdio.h>
void smile()// Function Declaration & Definition
{
   printf("\nSmile, and the world smiles with you...");
}
int main()
{
    smile();       // Function Call
    return 0;
}
```

# Practice Question

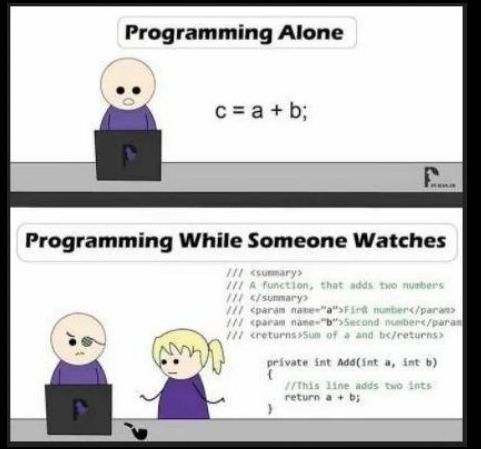
Make a function in C which prints following pattern.

```
****

****

****

****
```

```c
#include <stdio.h>
void star();    // Function Declaration or Prototype
int main()
{
   star();         // Function Call
   return 0;
}

void star()    // Function Definition
{
   int i;
   for(i=0; i<4; i++)
      printf("\n****");
}
```

```c
#include <stdio.h>
void add(int a, int b); // Function Prototype
int main()
{
   add(5,4);          // Function Call
   return 0;
}

void add(int a, int b)     // Function Definition
{
   printf("%d", a+b);
}
```

```c
#include <stdio.h>
int add(int a, int b);
int main()
{
   int sum;
   sum=add(5,4);
   printf("%d", sum);   // you can't access a and b here
   return 0;
}

int add(int a, int b)
{
    int c=a+b;          // you can't access sum here
    return c;
}
```



# Why Functions?

To hide irrelevant detail at the main() level, so the the program's primary purpose is clearer.

To divide a complex problem into a series of simple problems

To make subsequent modification of the program easier.

To reduce the errors that inevitably come with a single large complex program

# Practice Question

Make a function in C which takes one int argument and check whether it’s an even or odd.

```c
#include <stdio.h>
int even_odd(int x);
int main()
{
    int num=6;
    even_odd(num);
    return 0;
}
int even_odd(int x)
{
    if (x%2==0)
        printf("It's an even number!");
    else
        printf("It's an odd number!");
}
```

# Practice QUestion

Make a function in C which takes int as a parameter and return it’s cube value.

```c
#include <stdio.h>
int cube(int a);
int main()
{
    int num=4, result;
    result=cube(num);
    printf("%d", result);
    return 0;
}

int cube(int a)
{
     return a*a*a;
}
```

# Practice Question

Write a function to converts a hexadecimal number to its decimal equivalent and prints it.

# Passing Array Elements

Passing array elements to a function is similar to passing variables to a function.

```
void display(int age1, int age2) {

}

int main() {
  int ageArray[] = {2, 8, 4, 12};
  display(ageArray[1], ageArray[2]);
  return 0;
}
```

# Passing Array

1D array:

```
void myFunction(int arr[10]);
void myFunction(int arr[]);
```

2D array:

```
void myFunction(int arr[10][10]);
void myFunction(int arr[][10]);
```

```c
#include <stdio.h>
float calculate_sum(float arr[])   ;
int main()
{
   float array[5]={1.5, 2.3,4.5,6.7,1.3};

   printf("%f", calculate_sum(array));
   return 0;
}

float calculate_sum(float arr[])
{
     float sum=0; int i;
     for(i=0; i<5; i++)
        sum=sum+arr[i];
     return sum;
}
```

```c
#include <stdio.h>
float calculate_sum(float arr[], int size);
int main()
{
   float array[5]={1.5, 2.3,4.5,6.7,1.3};

   printf("%f", calculate_sum(array,5));
   return 0;
}

float calculate_sum(float arr[], int size)
{
     float sum=0; int i;
     for(i=0; i<size; i++)
        sum=sum+arr[i];
     return sum;
}
```

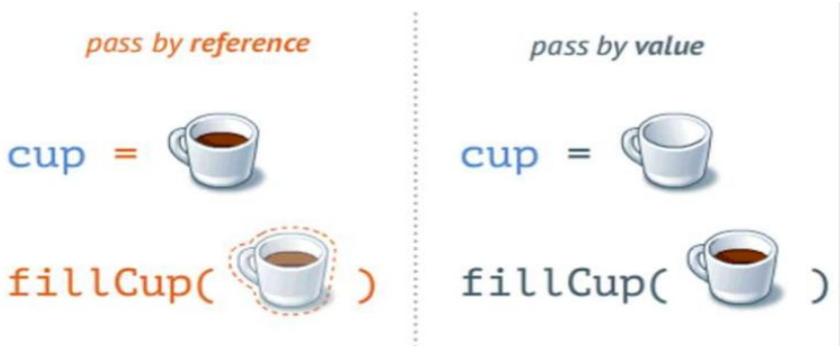
```c
#include <stdio.h>
float calculate_sum(float arr[2][5])   ;
int main()
{
   float array[2][5]={{1.5, 2.3,4.5,6.7,1.3},
                      {10.3,6.5,3.9,6.7,8.3}};

   printf("%f", calculate_sum(array));
   return 0;
}

float calculate_sum(float arr[2][5])
{
     float sum=0; int i,j;
     for(i=0; i<2; i++)
        for (j=0; j<5; j++)
            sum=sum+arr[i][j];
     return sum;
}
```

# Practice Question

Write a function declaration for  the below task:

You have a 3*3 char matrix. You need to  check if any row or column forms a palindrome.
Matrix:
m a d
a m a
d a m
Row 1: "ama" (Palindrome)
Column 1: "ama" (Palindrome)

# Calling Functions




Call by value

- Copy of argument passed to function
- Changes in function do not effect original
- Use when function does not need to modify argument
- Avoids accidental changes

All the examples mentioned before are using call by value.

Call by reference

- Passes original argument
- Changes in function effect original
- Only used with trusted functions

# Pass By Reference

```
void function( int *p);
```

```
int *p,num=6;
p = &num;
function( p );
```

```c
#include <stdio.h>
int cube(int *a);
int main()
{
    int num=4, *p;
    p=&num;
    cube(p); // cube(&num);
    printf("%d", num);
    return 0;
}
int cube(int *a)   // a is an alias or nickname of p
{
     *a= (*a) * (*a) *(*a);
}
```

# Passing Arrays By Reference

Array name is already a pointer, so passing an array name is itself passing by reference.

That means, if you update array element in function, it also changes in main.

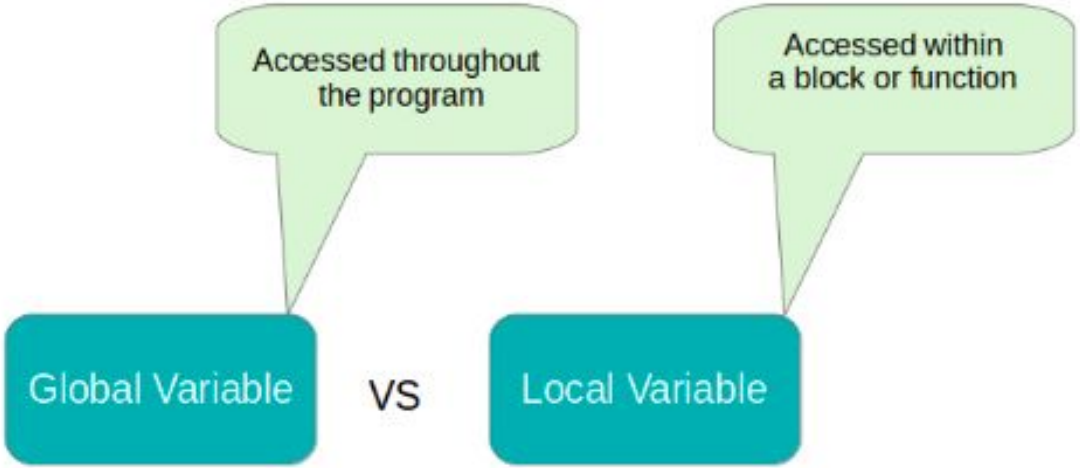
```c
#include <stdio.h>
float array_add(float arr[], int size) ;
int main()
{
   float array[5]={1.5, 2.3,4.5,6.7,1.3};
   array_add(array, 5);

   int i;
   for(i=0; i<5; i++)
      printf("%f", array[i]);
   return 0;
}
float array_add(float arr[], int size)
{
     int i;
     for(i=0; i<size; i++)
        arr[i]=arr[i]+5;
}
```

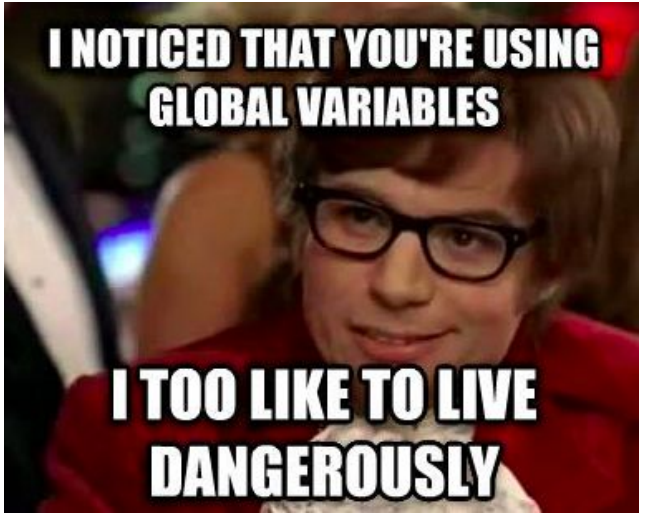
```
6.500000
7.300000
9.500000
11.700000
6.300000
----------------
```

```c
#include <stdio.h>
float array_add(float *p, int size)    ;
int main()
{
   float array[5]={1.5, 2.3,4.5,6.7,1.3};
   array_add(array, 5);

   int i;
   for(i=0; i<5; i++)
      printf("%f", array[i]);
   return 0;
}
float array_add(float *p, int size)
{
     int i;
     for(i=0; i<size; i++)
        p[i]=p[i]+5; // *p=*p+5; p++;
}
```

```
6.500000
7.300000
9.500000
11.700000
6.300000
----------------
```

# Local Vs Global Variables



```c
#include <stdio.h>
int global=20;
void smile();
int main()
{  i
   printf("Local Value is %d\n", local);
   printf("Global Value is %d\n", global);
   smile();
   printf("Global Value is %d\n", global);
   return 0;
}
void smile()
{  global+=20;
   printf("\nSmile, and the world smiles with you...");}
```

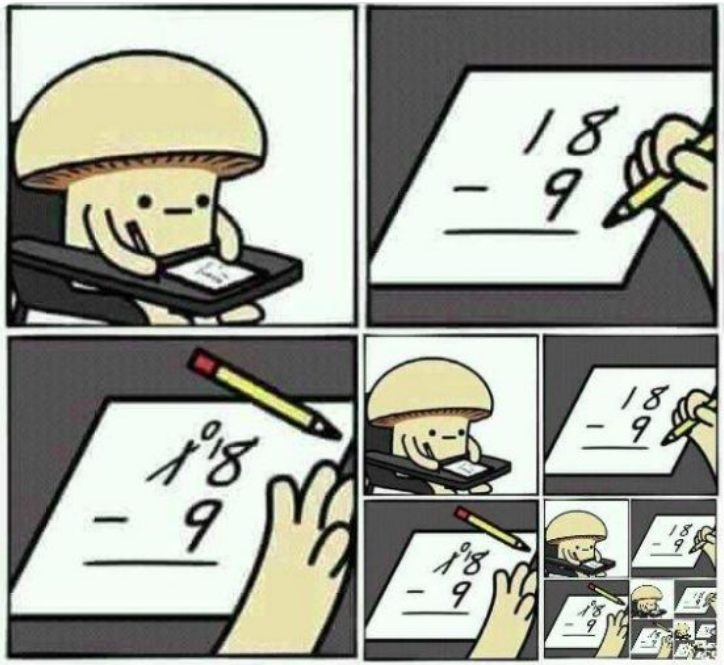
# Nested Functions Calling

We can call another user-defined function inside any user-defined function.

Make a function add which takes two argument and decides whether it’s sum is even or odd.

```c
#include <stdio.h>
int add(int a, int b);
void even_odd(int x);
int main()
{
    int sum;
    sum=add(5,4);
    printf("%d", sum);
    return 0;
}
```

```
int add(int a, int b)
{
    int c=a+b;
    even_odd(c);
    return c;
}
void even_odd(int x)
{
    if (x%2==0)
        printf(“It’s an even number!”);
    else
        printf(“It’s an odd number!”);
}
```




# RECURSION

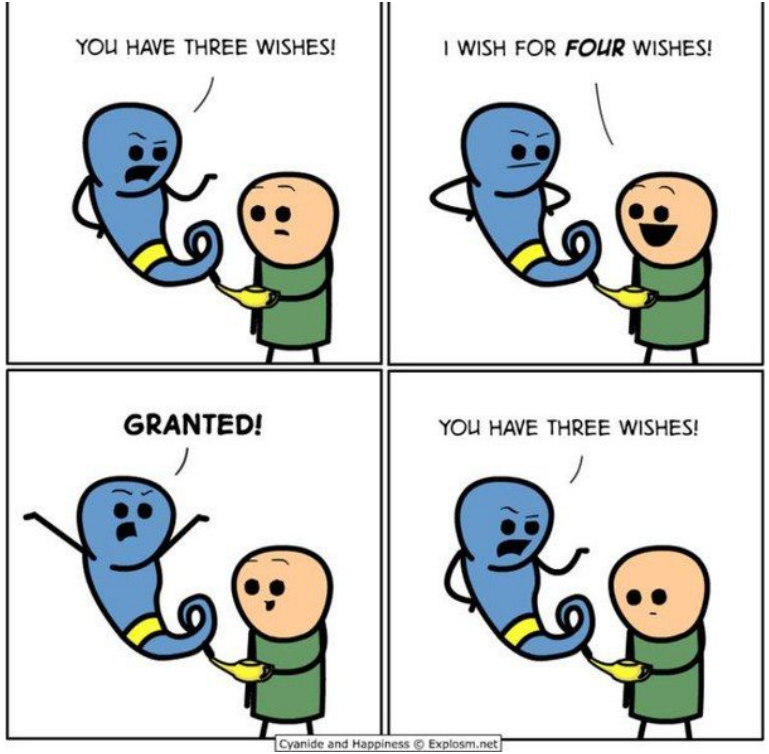
See recursion

# RECURSION

A function calling itself.

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

recursive call



# RECURSION

fact(n) : n!

fact(1)=1

fact(2)=2*1

fact(3)=3*2*1

fact(4)=4*3*2*1

fact(5)=5*4*3*2*1

# RECURSION

fact(n) : n!

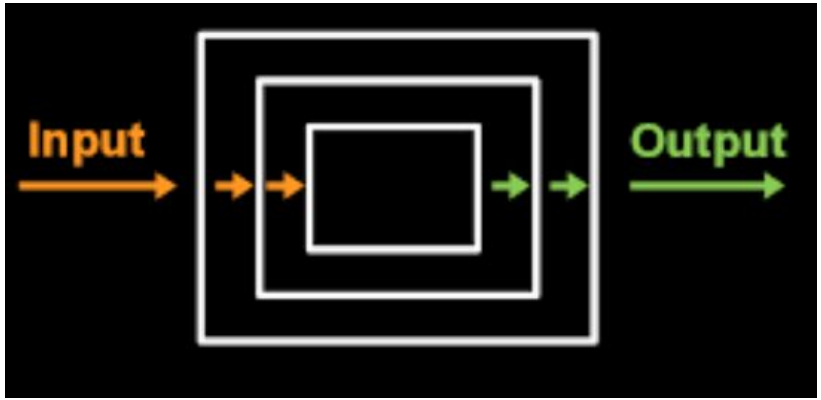
fact(1)=1

fact(2)=2*fact(1)

fact(3)=3*fact(2)

fact(4)=4*fact(3)

fact(5)=5*fact(4)



# RECURSION

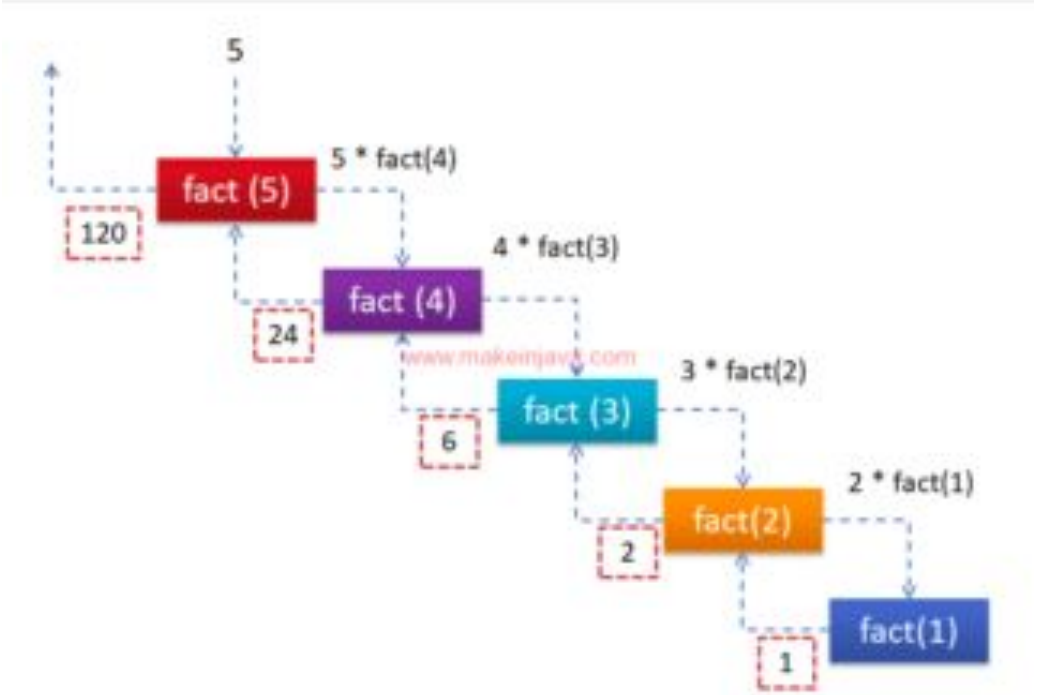
```
fact(n) : n * fact(n-1)
```

Recursive function have 2 cases:

- Base Case
- Recursive Case

# RECURSION

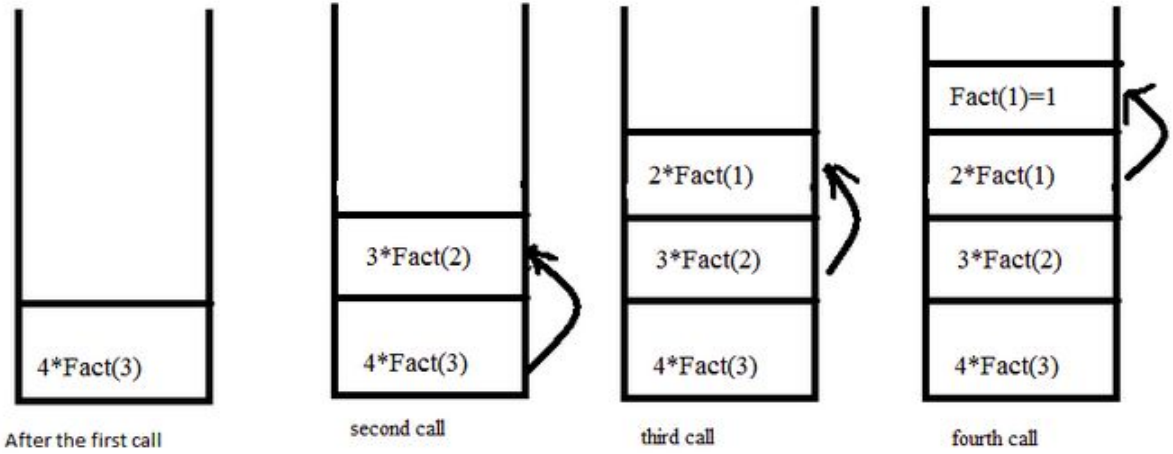
```
fact(n) : n!
fact(1)=1
fact(2)=2*fact(1)
fact(3)=3*fact(2)
fact(4)=4*fact(3)
fact(5)=5*fact(4)
```



# Factorial Function

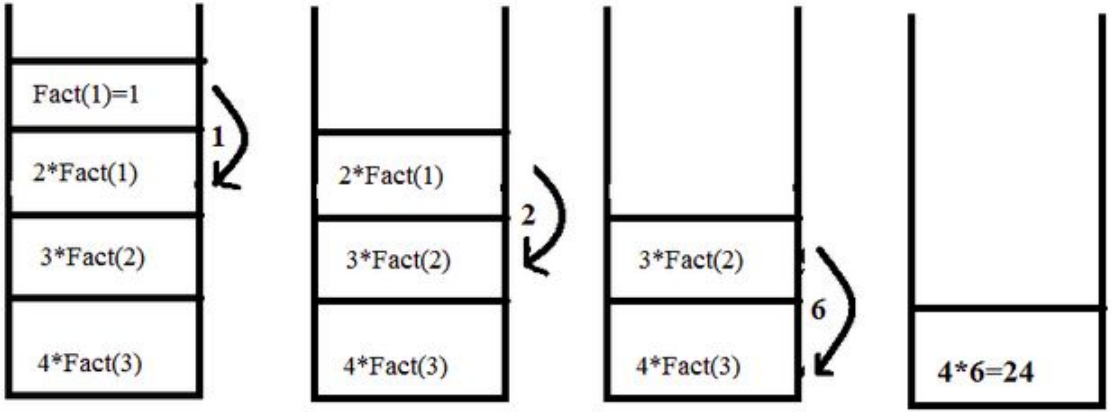
```
int factorial( int n)
{
    if (n==1)          // Base case
        return 1;
    else               // Recursive Case
        return n * factorial(n-1);
}
```

**When function call happens previous variables gets stored in stack**




After the first call | second call | third call | fourth call

**Returning values from base case to caller function**



# Practice Question

Write a program in C to calculate the sum of numbers from 1 to n using recursion.

# Sum Function

```
int sum( int n)
{
   if (n==1)          // Base case
       return 1;
   else               // Recursive Case
       return n + sum(n-1);
}
```

# Practice Question

Write a program in C to print Fibonacci sequence to n terms using recursion.

# Fibonacci Function

```
int fibonacci( int n)
{
   if (n==0)                // 1st Base case
         return 0;
   else if (n==1)           // 2nd Base case
         return 1;
   else                     // Recursive Case
         return fibonacci(n-2) + fibonacci(n-1);
}
```

# Home Assignment

Write a program in C to print first 50 natural numbers using recursion.

Write a program in C to print the array elements using recursion.

Write a program in C to count the digits of a given number using recursion.