# Recursion In C

## Recursion In C

### Recursion In C

#### Recursion In C

Course Title :- Structured Programming Language Sessional

Course Code :- CSE-122 [SECTION-B]

Level Term: 1-II-A(G1)  &  1-II-B(G3,G4)

Khandaker Jannatul Ritu, Lecturer(CSE), BAIUST

# *Outlines:*

❑ Introduction to Recursion in C
❑ Fundamental Components of C Recursion
❑ Need of Recursive Function
❑ Types of C Recursion
❑ Approaches of Recursion
❑ Difference between Recursion and Iteration
❑ How Recursion Actually Works in C?
    1) Print 1 to N
    2) Print N to 1
❑ Important Programs Using Recursion
    1) Factorials
    2) Fibonacci
    3) Sum of natural numbers
❑ Examples to Find the output of this pattern
❑ Examples to calculate the output of this Practice problems
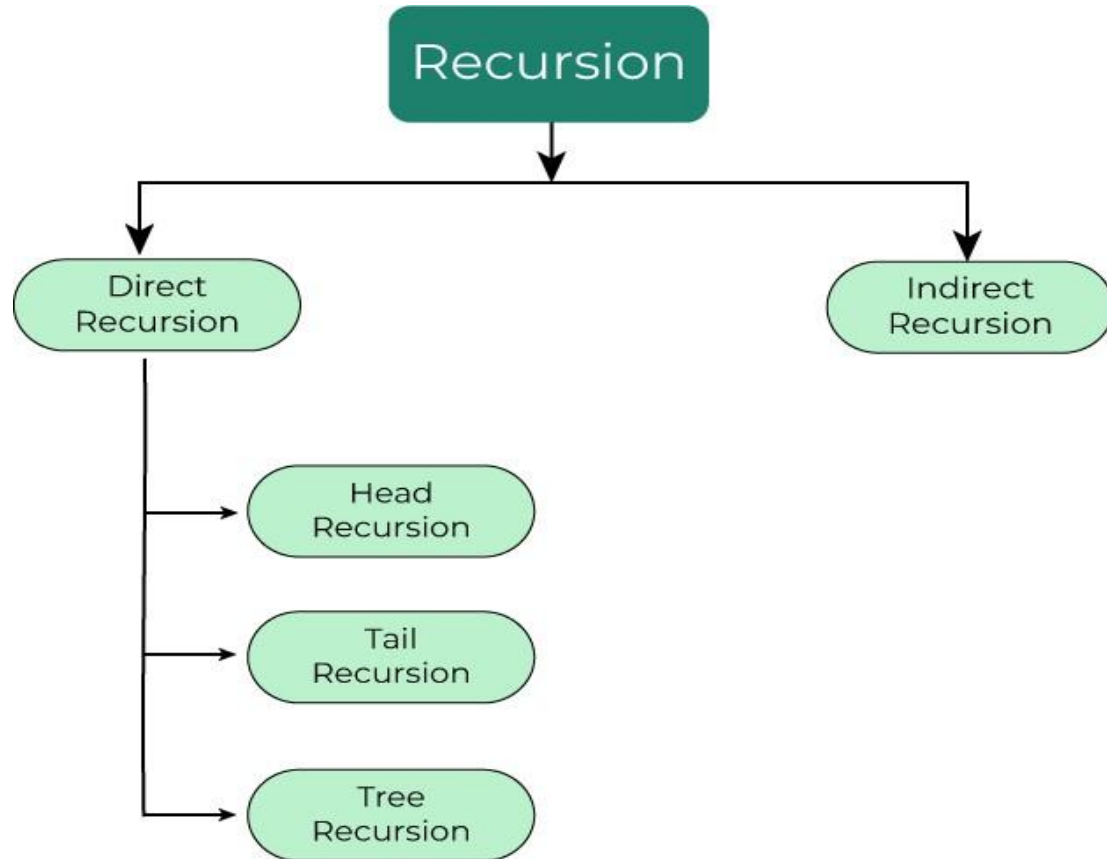❑ Printing Pyramid Patterns using Recursion

✓ A Recursive function can be defined as a function that calls itself directly or indirectly.

❑ Fundamental Components of C Recursion
1. Recursion Case
2. Base Condition

❑ Types of Recursion in C:

```c
int nSum(int n)
{
    if (n==0) {          Base
    return 0;            condition

    }

int res = n+ nsum(n-1);

    return res;          Recursive
                         case
}
```

Recursion

Direct Recursion → Head Recursion, Tail Recursion, Tree Recursion

Indirect Recursion

❑ Indirect Recursion:-
```c
void functionA(int n){
 if (n < 1) { return; }
 printf("%d ", n);
 n = n - 1;
 functionB(n);
}
void functionB(int n){
 if (n < 2) { return; }
 printf("%d ", n);
 n = n / 2;
 functionA(n);
}
```

# 1.What is Head Recursion

The **head recursion** is a linear recursion where the position of its only recursive call is at the start of the function. It is generally the first statement in the function.

# 2.What is Tail Recursion

**Tail recursion** is defined as a recursive function in which the recursive call is the last statement that is executed by the function. So basically, nothing is left to execute after the recursion call.

# 3.What is Tree Recursion

In **tree recursion**, there are multiple recursive calls present in the body of the function. Due to this, while tracing the program flow, it makes a tree-like structure, hence the name Tree Recursion.

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

```
void print(int n)
{
    if (n < 0)
    return;
    printf("%d ", n);
    print(n - 1);
}
```

```
int fib(int n)
{
    if (n ≤ 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```

# ❑ Approaches of Recursion

There are two approaches for a recursive function.

## a. Top down approach

For user input : 5

Factorial Recursion Function

$$n * f(n-1)$$

Final Result

$5 * f(4)$  $= 5*24 = \boxed{120}$

$4 * f(3)$  $= 4* 6 = 24$

$3 * f(2)$  $= 3*2 = 6$

$2 * f(1)$  $= 2*1 = 2$

## b. Bottom up approach

Factorial(1, 5) : 120

$1 \times$ Factorial(1+1, 5) : $1 \times 120 = 120$

$2 \times$ Factorial(2+1, 5) : $2 \times 60 = 120$

$3 \times$ Factorial(3+1, 5) : $3 \times 20 = 60$

$4 \times$ Factorial(4+1, 5) : $4 \times 5 = 20$

$5 \times$ Factorial(5+1, 5) : $5 \times 1 = 5$

$6 > 5$ : return 1

# ❑How Recursion works in C?
1. Print 1 to N using recursion
   a. manual
   b. function
   c. recursion top-down
   d. recursion bottom-up

## Print 1 to n without using loops
You are given an integer N. Print numbers from 1 to N without the help of loops.
Input: N = 5
Output: 1 2 3 4 5

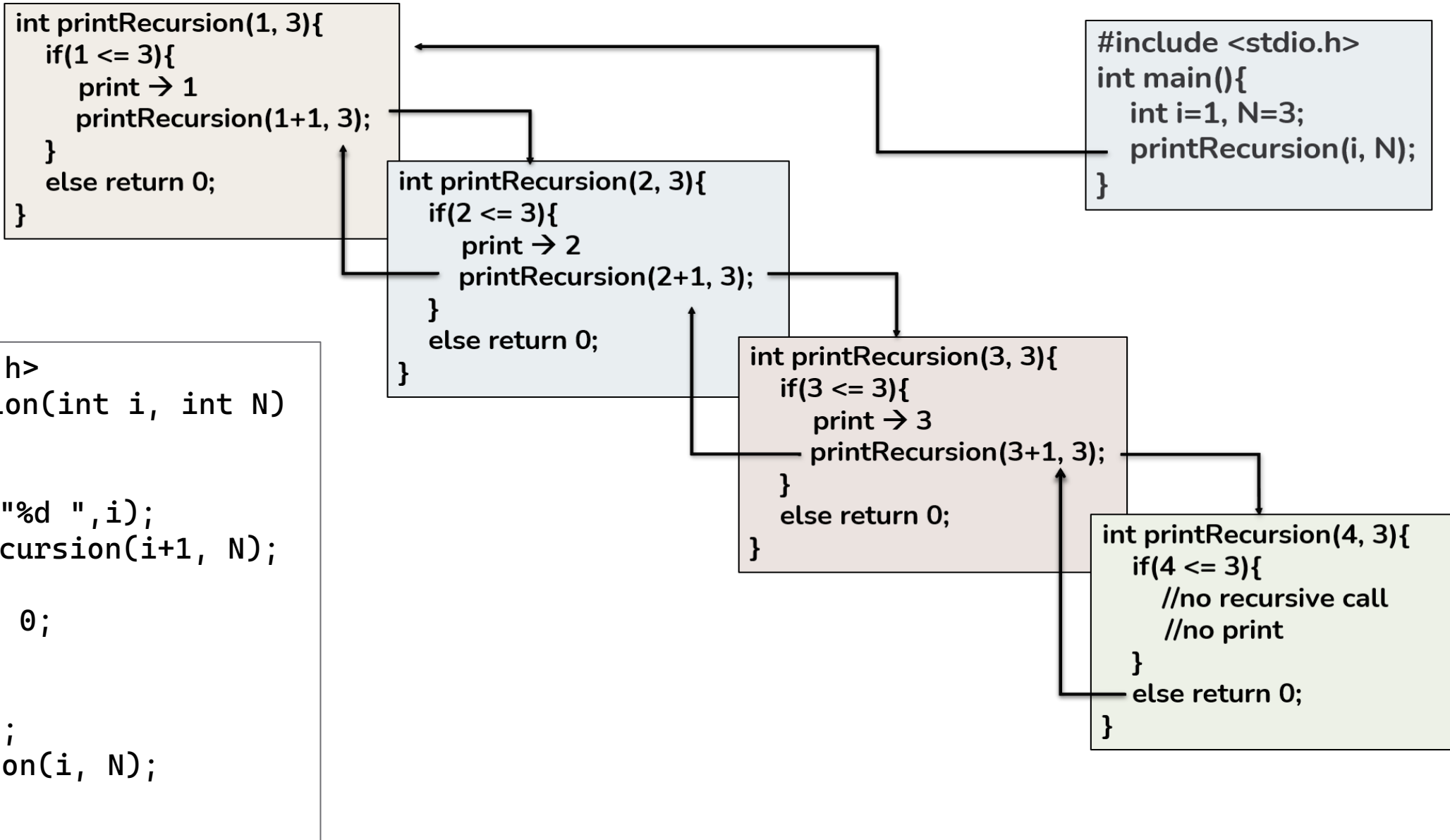### a. Manually
```c
#include <stdio.h>
int main(){
    int N=10;
    for(int i = 1; i≤ N; i++)
    {
        printf("%d ", i);
    }
}
```
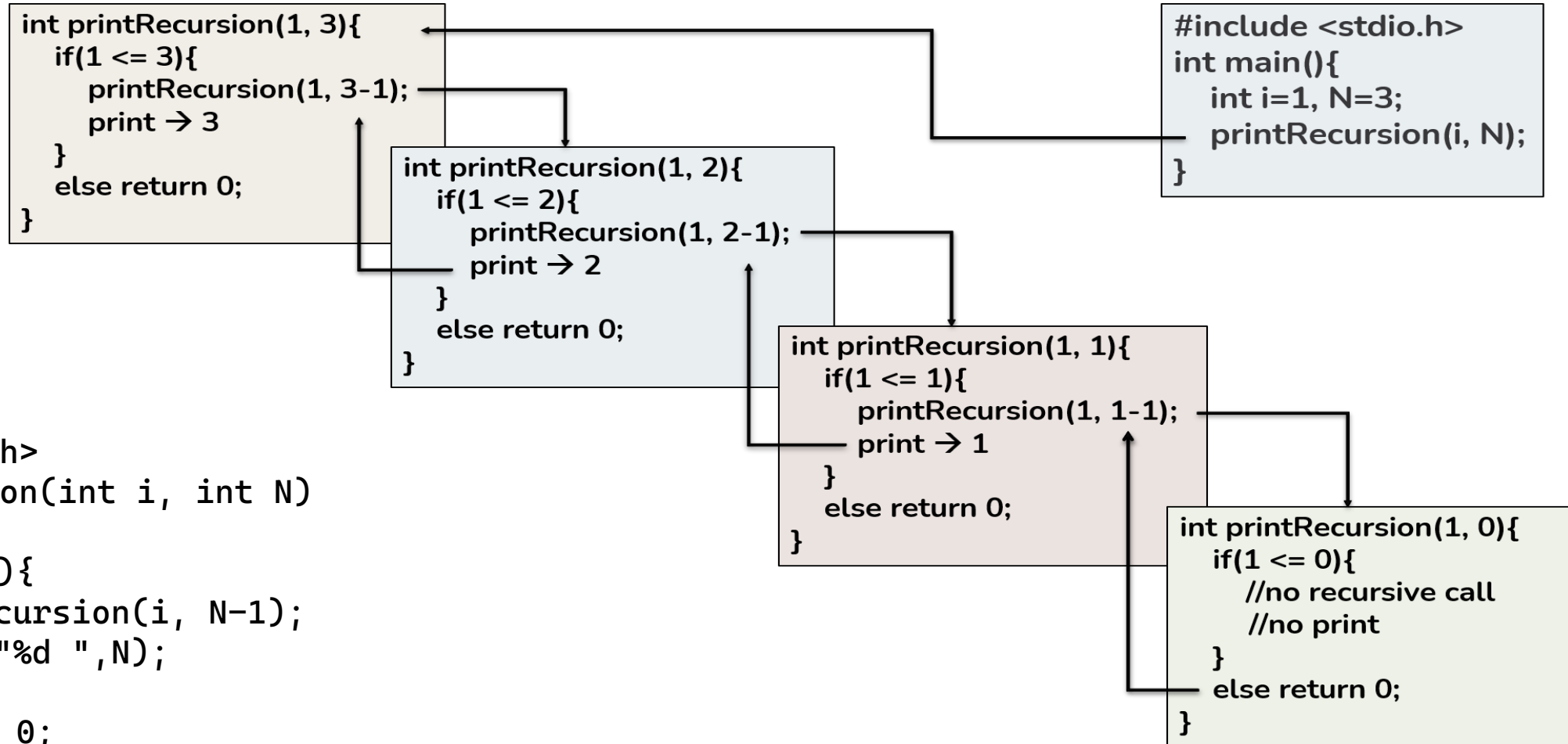
### b. Using Normal Function
```c
#include <stdio.h>
int printFunction(int N){
    for(int i = 1; i≤ N; i++)
    {
        printf("%d ", i);
    }
}
int main(){
    int N=10;
    printFunction(N);
}
```

# c. Using Recursive Function [Bottom-Up]

```
int printRecursion(1, 3){
    if(1 <= 3){
        print → 1
        printRecursion(1+1, 3);
    }
    else return 0;
}
```

```
int printRecursion(2, 3){
    if(2 <= 3){
        print → 2
        printRecursion(2+1, 3);
    }
    else return 0;
}
```

```
int printRecursion(3, 3){
    if(3 <= 3){
        print → 3
        printRecursion(3+1, 3);
    }
    else return 0;
}
```

```
int printRecursion(4, 3){
    if(4 <= 3){
        //no recursive call
        //no print
    }
    else return 0;
}
```

```
#include <stdio.h>
int main(){
    int i=1, N=3;
    printRecursion(i, N);
}
```

```c
#include <stdio.h>
int printRecursion(int i, int N)
{
    if(i ≤ N){
        printf("%d ",i);
        printRecursion(i+1, N);
    }
    else return 0;
}
int main(){
    int i=1, N=5;
    printRecursion(i, N);
}
```

# d. Using Recursive Function [Top-Down]

```
int printRecursion(1, 3){
    if(1 <= 3){
        printRecursion(1, 3-1);
        print → 3
    }
    else return 0;
}
```

```
#include <stdio.h>
int main(){
    int i=1, N=3;
    printRecursion(i, N);
}
```

```
int printRecursion(1, 2){
    if(1 <= 2){
        printRecursion(1, 2-1);
        print → 2
    }
    else return 0;
}
```

```
int printRecursion(1, 1){
    if(1 <= 1){
        printRecursion(1, 1-1);
        print → 1
    }
    else return 0;
}
```

```
#include <stdio.h>
int printRecursion(int i, int N)
{
    if( i ≤ N ){
        printRecursion(i, N-1);
        printf("%d ",N);
    }
    else return 0;
}
int main(){
    int i=1, N=5;
    printRecursion(i, N);
}
```

```
int printRecursion(1, 0){
    if(1 <= 0){
        //no recursive call
        //no print
    }
    else return 0;
}
```

H.W. Recursive C Programs:-
1.Print odd numbers 1 to n using recursion.
Numbers : 1, 3, 5, 7, 9,……….n
2.Print even numbers 2 to n using recursion.
Numbers : 2, 4, 6, 8, 10,……...n

# 2. Print N to 1 using recursion
   a. manual
   b. function
   c. recursion top-down
   d. recursion bottom-up


Print n to 1 without using loops
You are given an integer N. Print numbers from 1 to N without the help of loops.
Input: N = 5
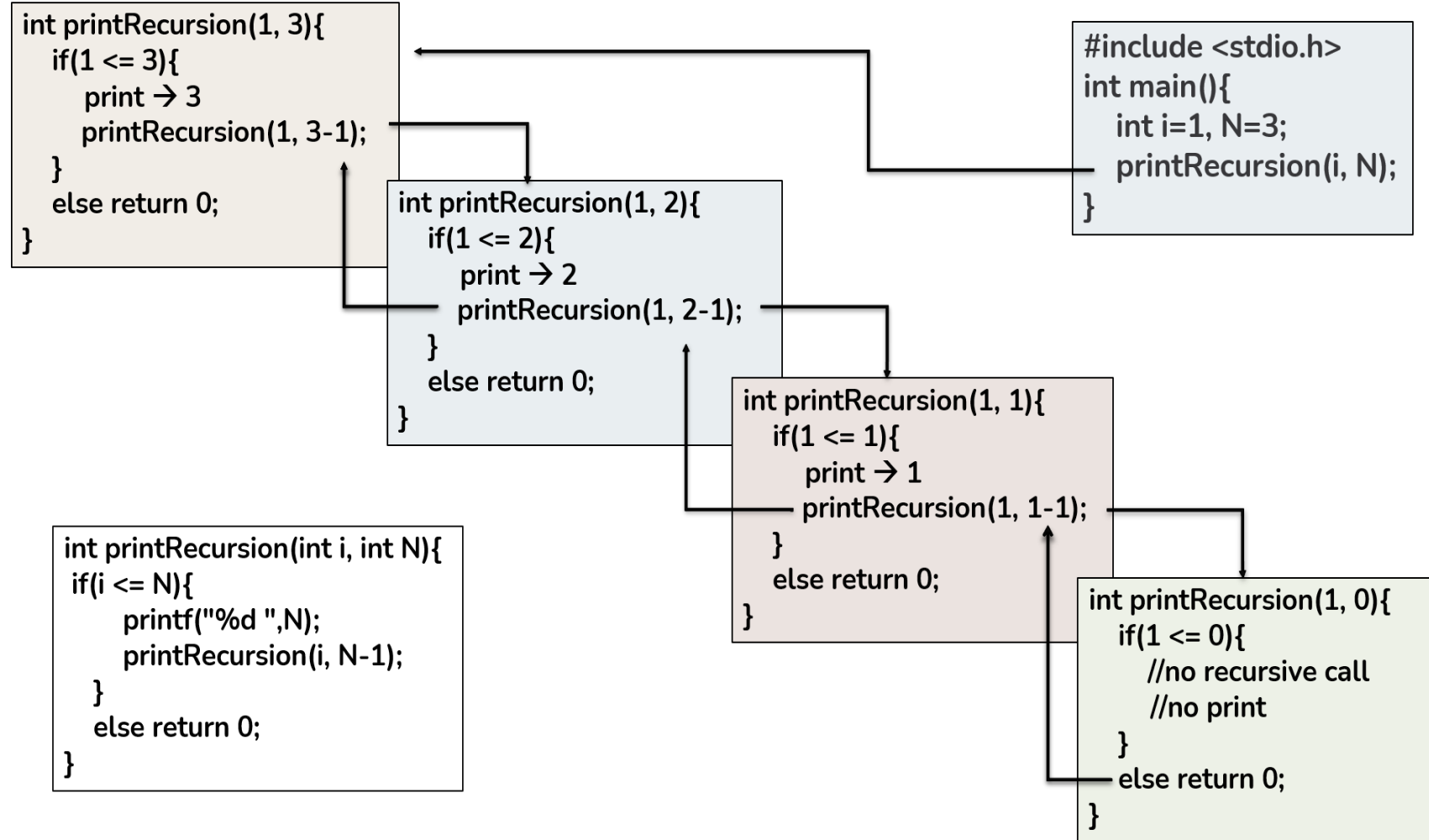Output: 5 4 3 2 1

a. Manually
```c
#include <stdio.h>
int main(){
  int N=10;
  for(int i = N; i ≥ 1; i--) {
     printf("%d ", i);
  }
}
```

b. Using Normal Function
```c
#include <stdio.h>
int printFunction(int N){
     for(int i = N; i ≥ 1; i--) {
        printf("%d ", i);
     }
}
int main(){
     int N=10;
     printFunction(N);
}
```

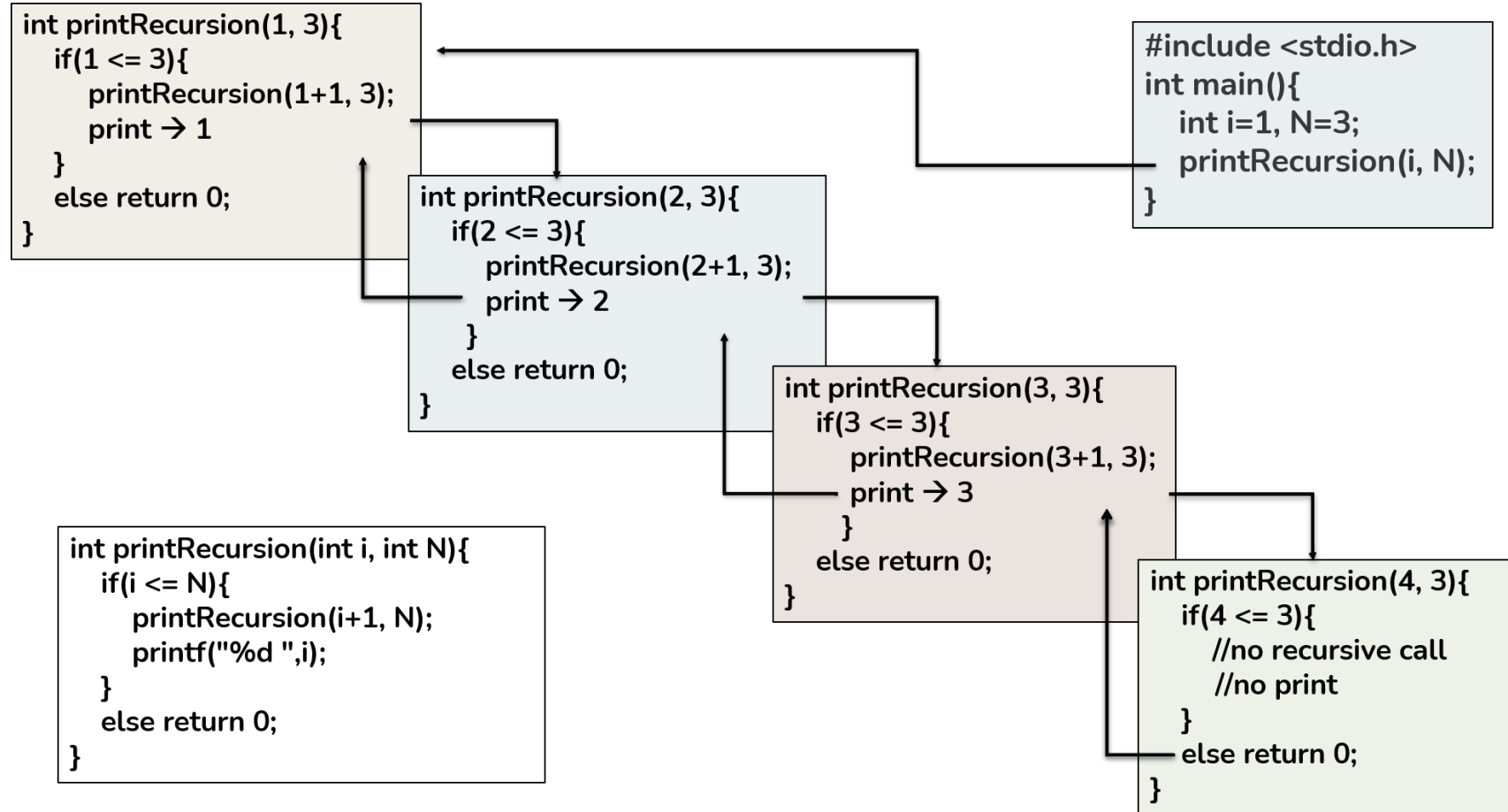# c. Using Recursive Function [Bottom-Up]

```c
#include <stdio.h>
int printRecursion(int i, int N){
    if(i ≤ N){
        printf("%d ",N);
        printRecursion(i, N-1);
    }
    else return 0;
}
int main(){
    int i=1, N=5;
    printRecursion(i, N);
}
```

```c
int printRecursion(1, 3){
    if(1 <= 3){
        print → 3
        printRecursion(1, 3-1);
    }
    else return 0;
}
```

```c
#include <stdio.h>
int main(){
    int i=1, N=3;
    printRecursion(i, N);
}
```

```c
int printRecursion(1, 2){
    if(1 <= 2){
        print → 2
        printRecursion(1, 2-1);
    }
    else return 0;
}
```

```c
int printRecursion(1, 1){
    if(1 <= 1){
        print → 1
        printRecursion(1, 1-1);
    }
    else return 0;
}
```

```c
int printRecursion(int i, int N){
    if(i <= N){
        printf("%d ",N);
        printRecursion(i, N-1);
    }
    else return 0;
}
```

```c
int printRecursion(1, 0){
    if(1 <= 0){
        //no recursive call
        //no print
    }
    else return 0;
}
```

# d. Using Recursive Function [Top-Down]

```c
#include <stdio.h>
int printRecursion(int i, int N){
    if(i ≤ N){
        printRecursion(i+1, N);
        printf("%d ",i);
    }
    else return 0;
}
int main(){
    int i=1, N=5;
    printRecursion(i, N);
}
```

```
int printRecursion(1, 3){
    if(1 <= 3){
        printRecursion(1+1, 3);
        print → 1
    }
    else return 0;
}
```

```
#include <stdio.h>
int main(){
    int i=1, N=3;
    printRecursion(i, N);
}
```

```
int printRecursion(2, 3){
    if(2 <= 3){
        printRecursion(2+1, 3);
        print → 2
    }
    else return 0;
}
```

```
int printRecursion(3, 3){
    if(3 <= 3){
        printRecursion(3+1, 3);
        print → 3
    }
    else return 0;
}
```

```
int printRecursion(int i, int N){
    if(i <= N){
        printRecursion(i+1, N);
        printf("%d ",i);
    }
    else return 0;
}
```

```
int printRecursion(4, 3){
    if(4 <= 3){
        //no recursive call
        //no print
    }
    else return 0;
}
```

H.W. Recursive C Programs:-
1.Print even number from N to 2 using recursion [top down & bottom up]
2.Print odd number from N to 1 using recursion [top down & bottom up]

# ❑Important Programs Using Recursion

1) C program to calculate Factorials of N
  1. Basic
  2. Using function
  3. Using recursive function[bottom – up ]
  4. Using recursive function[top – down ]


C program to calculate Factorials of N. You are given a number N. Find the factorial of N.
Input N = 5
Output : 120
Explanation: 5 x 4 x 3 x 2 x 1 = 120


## 1. Manually

```c
#include <stdio.h>
int main(){
    int fact = 1;
    int N=5;
    for(int i=1; i≤N; i++)
    {
        fact = fact * i;
    }
    printf("%d", fact);
}
```
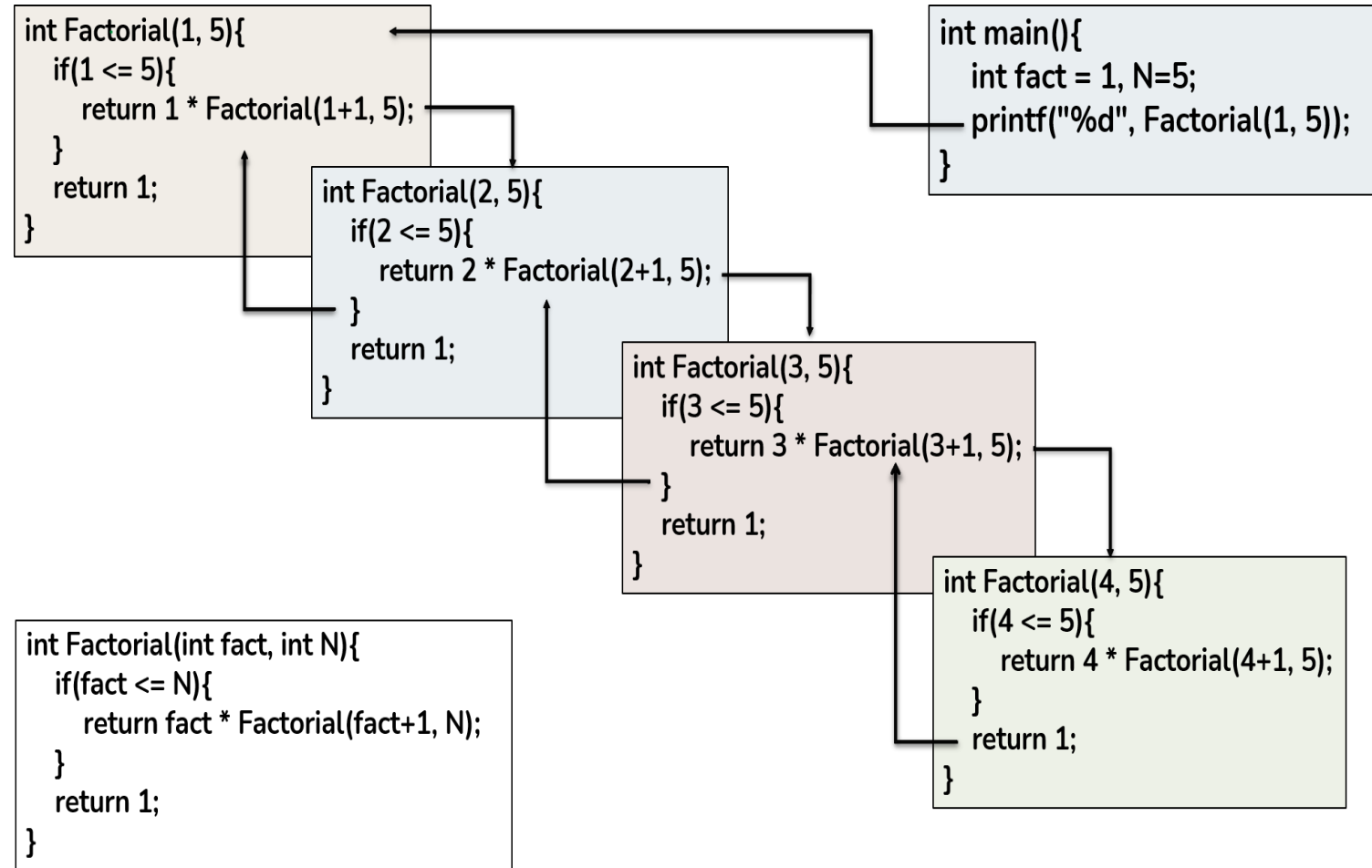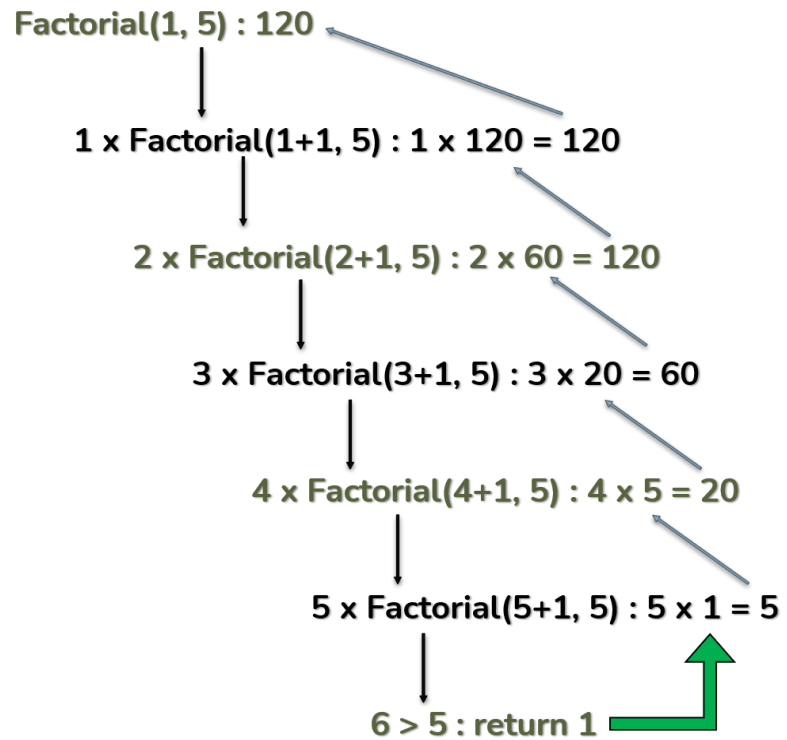
## 2. Using Function

```c
#include <stdio.h>
int Factorial(int fact, int N){
        for(int i=1; i≤N; i++)
        {
            fact = fact * i;
        }
        return fact;
}
int main(){
        int fact = 1, N=5;
        printf("%d", Factorial(1, 5));
}
```
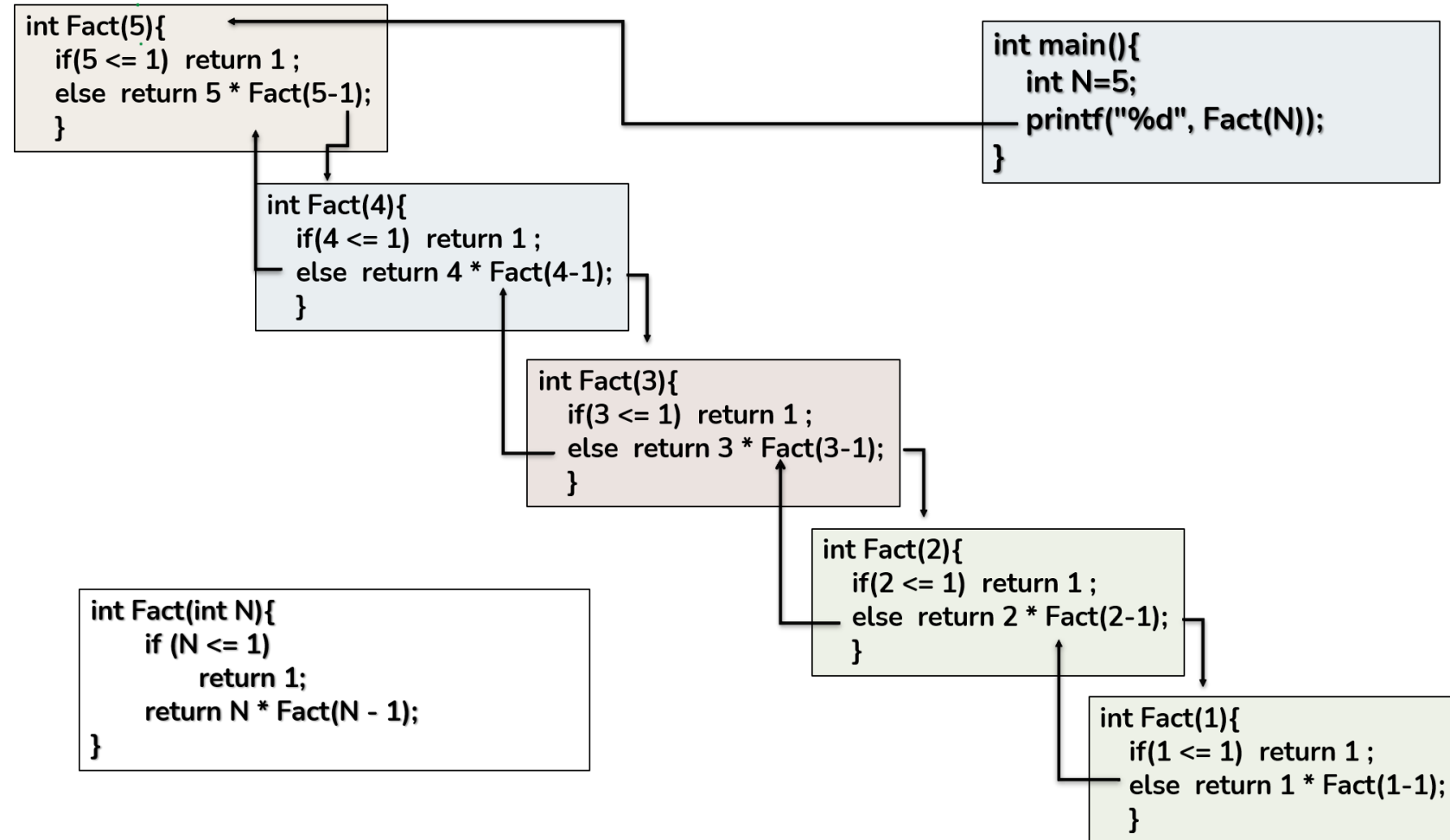
# 3. Using recursive function [Bottom – up]

```c
#include <stdio.h>
int Factorial(int fact, int N){
    if(fact ≤ N){
        return fact * Factorial(fact+1, N);
    }
    return 1;
}
int main(){
    int fact = 1, N=5;
    printf("%d", Factorial(1, 5));
}
```

```
int Factorial(1, 5){
    if(1 <= 5){
        return 1 * Factorial(1+1, 5);
    }
    return 1;
}
```

```
int main(){
    int fact = 1, N=5;
    printf("%d", Factorial(1, 5));
}
```

```
int Factorial(2, 5){
    if(2 <= 5){
        return 2 * Factorial(2+1, 5);
    }
    return 1;
}
```

```
int Factorial(3, 5){
    if(3 <= 5){
        return 3 * Factorial(3+1, 5);
    }
    return 1;
}
```

```
int Factorial(4, 5){
    if(4 <= 5){
        return 4 * Factorial(4+1, 5);
    }
    return 1;
}
```

```
int Factorial(int fact, int N){
    if(fact <= N){
        return fact * Factorial(fact+1, N);
    }
    return 1;
}
```

Factorial(1, 5) : 120

1 x Factorial(1+1, 5) : 1 x 120 = 120

2 x Factorial(2+1, 5) : 2 x 60 = 120

3 x Factorial(3+1, 5) : 3 x 20 = 60

4 x Factorial(4+1, 5) : 4 x 5 = 20

5 x Factorial(5+1, 5) : 5 x 1 = 5

6 > 5 : return 1

# 4. Using recursive function [Top – down]

```c
#include <stdio.h>
int fact(int N){
    if (N ≤ 1)
        return 1;
    return N * fact(N – 1);
}
int main(){
    int N=5;
    printf("%d", fact(N));
}
```

```
int Fact(5){
    if(5 <= 1)  return 1 ;
    else  return 5 * Fact(5-1);
}
```

```
int main(){
    int N=5;
    printf("%d", Fact(N));
}
```

```
int Fact(4){
    if(4 <= 1)  return 1 ;
    else  return 4 * Fact(4-1);
}
```

```
int Fact(3){
    if(3 <= 1)  return 1 ;
    else  return 3 * Fact(3-1);
}
```

```
int Fact(2){
    if(2 <= 1)  return 1 ;
    else  return 2 * Fact(2-1);
}
```

```
int Fact(int N){
    if (N <= 1)
        return 1;
    return N * Fact(N - 1);
}
```

```
int Fact(1){
    if(1 <= 1)  return 1 ;
    else  return 1 * Fact(1-1);
}
```

For user input : 5

Factorial Recursion Function

n * f(n-1)

Final Result

5 * f(4)  = 5*24 = 120

4 * f(3)  = 4* 6 = 24

3 * f(2)  = 3*2 = 6

2 * f(1)  = 2*1 = 2

# 2) C program to calculate nth Fibonacci numbers
    a) Using Array[bottom – up ]
    b) Using recursive function[top – down ]

Given a number n, print n-th Fibonacci Number.  The Fibonacci numbers are the numbers in the following integer sequence:  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……

## a) Using Array [bottom – up]:

```c
#include <stdio.h>
int fib(int n){
    int f[n];
    f[0] = 0;
    f[1] = 1;
    for(int i = 2; i ≤ n; i++)
    {
        f[i] = f[i - 1] + f[i - 2];
    }

    return f[n];
}
int main(){
    printf("%d", fib(8));
}
```

## ❏ How Fibonacci sequence works?

$N^{th}$ Fibonacci is summation of previous 2 Fibonacci number.

Fibonacci [ n ] = Fibonacci [ n - 1 ] + Fibonacci [ n - 2 ]

$0^{th}$ Fibonacci = 0

$1^{th}$ Fibonacci = 1

$2^{nd}$ Fibonacci = $0^{th}$ Fibonacci  + $1^{th}$ Fibonacci = 0 + 1 = 1

$3^{rd}$ Fibonacci = $1^{th}$ Fibonacci  + $2^{nd}$ Fibonacci = 1 + 1 = 2

$4^{th}$ Fibonacci = $2^{nd}$ Fibonacci  + $3^{rd}$ Fibonacci = 1 + 2 = 3

$5^{th}$ Fibonacci = $3^{rd}$ Fibonacci  + $4^{th}$ Fibonacci = 2 + 3 = 5

And so on………………

# b) Using Recursion[Top − Down]:

```c
#include <stdio.h>
int fibo(int n){
    if (n ≤ 1) {
        return n;
    }
    else {
        return fibo(n − 1) + fibo(n − 2);
    }
}
int main(){
    int n = fibo(5);
    printf("%d", n);
}
```
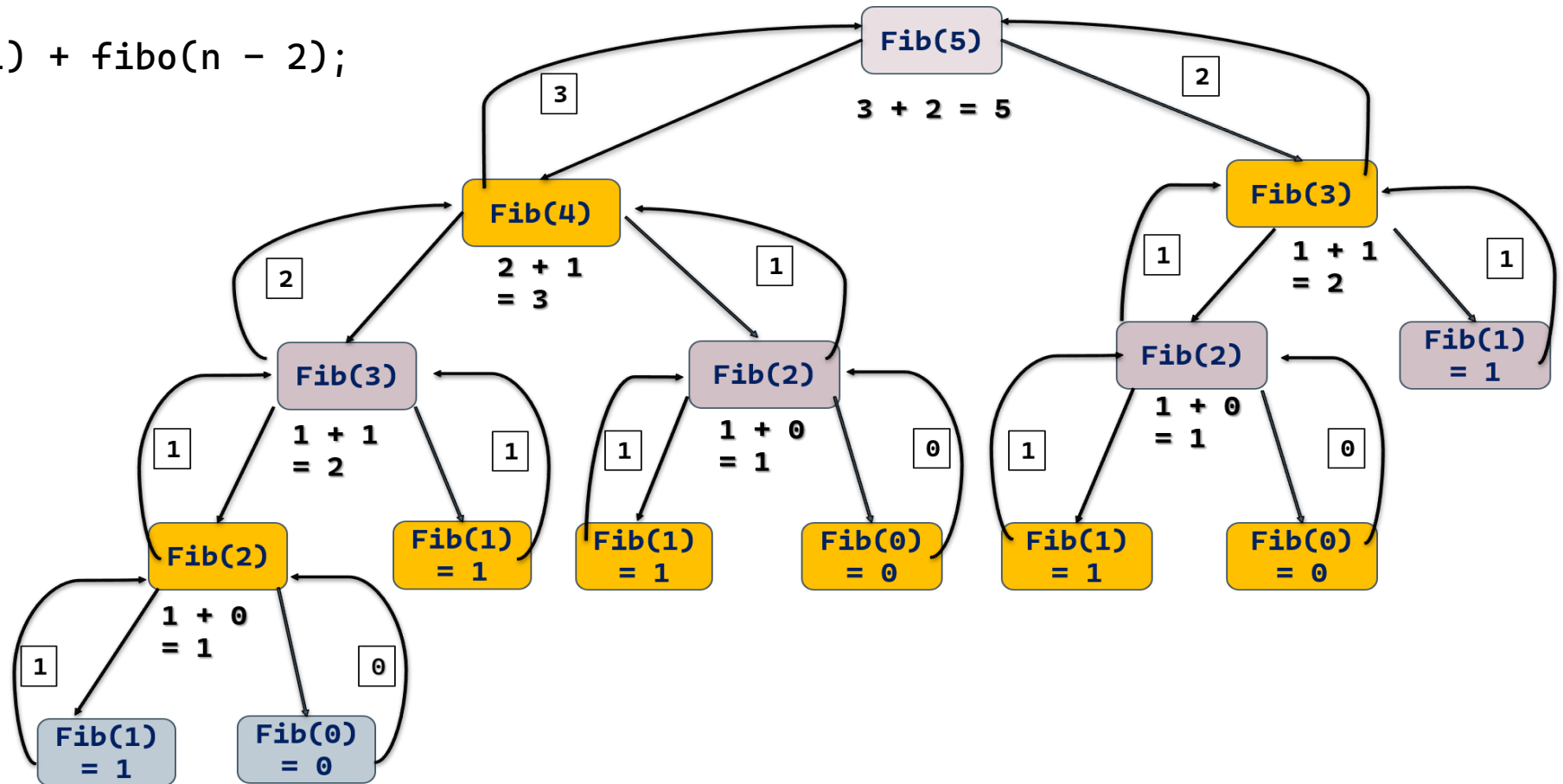
3) C program to calculate sum of natural numbers from 1 to N
    [top - down ]


C program to Calculate Sum of all-Natural Number
Input n = 5
Output: 15

N.B. The sum of numbers from 1 to 5 : 1 + 2 + 3 + 4 + 5 = 15

```c
#include <stdio.h>
int sum(int n){
    if( n ≥ 1)
    {
        return n + sum(n-1);
    }
    else
        return 0;
}
int main()
{
    int n = 10;
    printf("Sum = %d", sum(n));
}
```

```c
int main() {
    ... ..                    3
    result = sum(number);
    ... ..
}
                 3
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
                 2
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
                 1
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
                 0
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
```

3+3 = 6
is returned

2+1 = 3
is returned

1+0 = 1
is returned

0
is returned
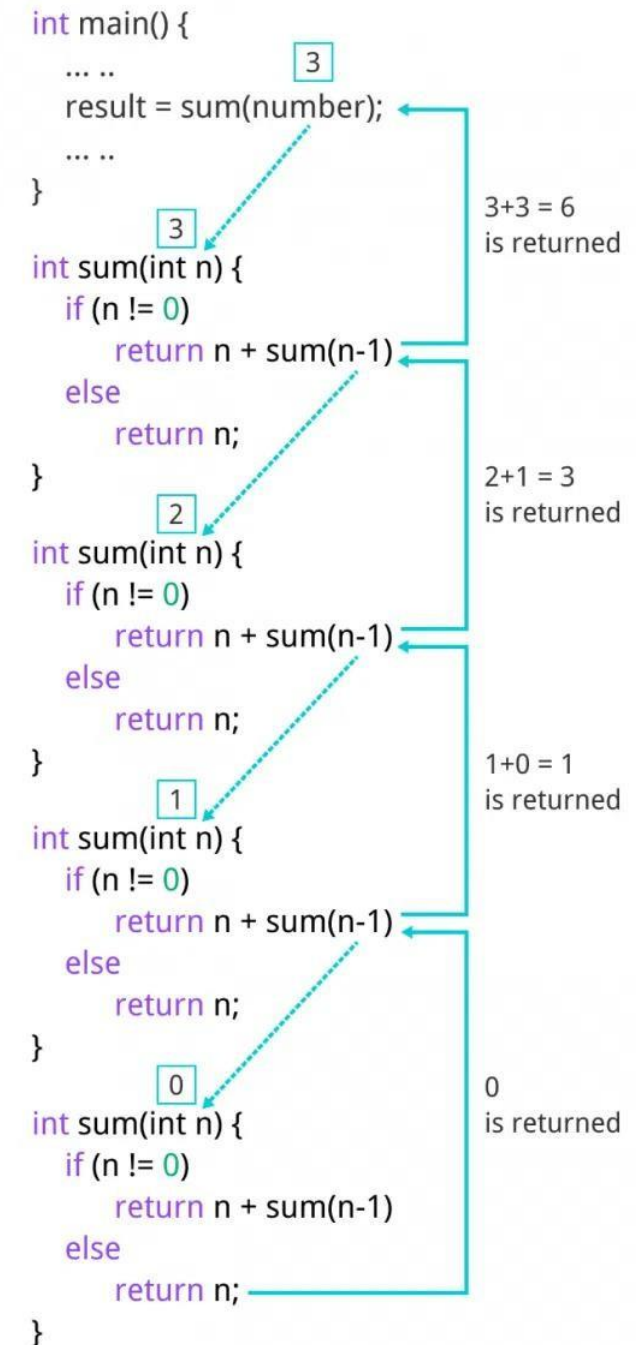
# ❖ Find the output of a pattern without using any loop

Example-1 : Given a number n, print the following pattern without using any loop.
Sequence: n, n-5, n-10, …, 0, 5, 10, …, n-5, n
Examples :
Input: n = 16
Output: 16, 11, 6, 1, -4, 1, 6, 11, 16
Input: n = 10
Output: 10, 5, 0, 5, 10

Example-2:-
Assignment!! Find the Recursion Tree for this program:-

```c
#include <stdio.h>
void printFun(int test){
    if (test < 1)
        return;
    else {
        printf("%d ", test);
        printFun(test - 1);
        printf("%d ", test);
    }
}
int main(){
    int test = 3;
    printFun(test);
}
```

```c
#include <stdio.h>
void printPattern(int n){
    if (n ≤ 0){
        printf(" ");
        return;
    }
    printf("%d ", n);
    printPattern(n - 5);
    printf("%d ", n);
}
int main(){
    int n = 16;
    printPattern(n);
}
```
Output
16 11 6 1 1 6 11 16

**Example – 3:** Predict the output pattern

```c
#include <stdio.h>
void fun(int n){
    if(n == 0)
        return;
    fun(n/2);
    printf("%d", n%2);
}
int main(){
    int n = 21;
    fun(n);
}
```

For example, if n is 21 then fun2() prints 10101.

Fun(21)

Fun(10)     print(1)    5th

Fun(5)     print(0)

Fun(2)     print(1)     4th

Fun(1)     print(0)     3rd

Fun(0)     print(1)     2nd
Return

print(1)

1st
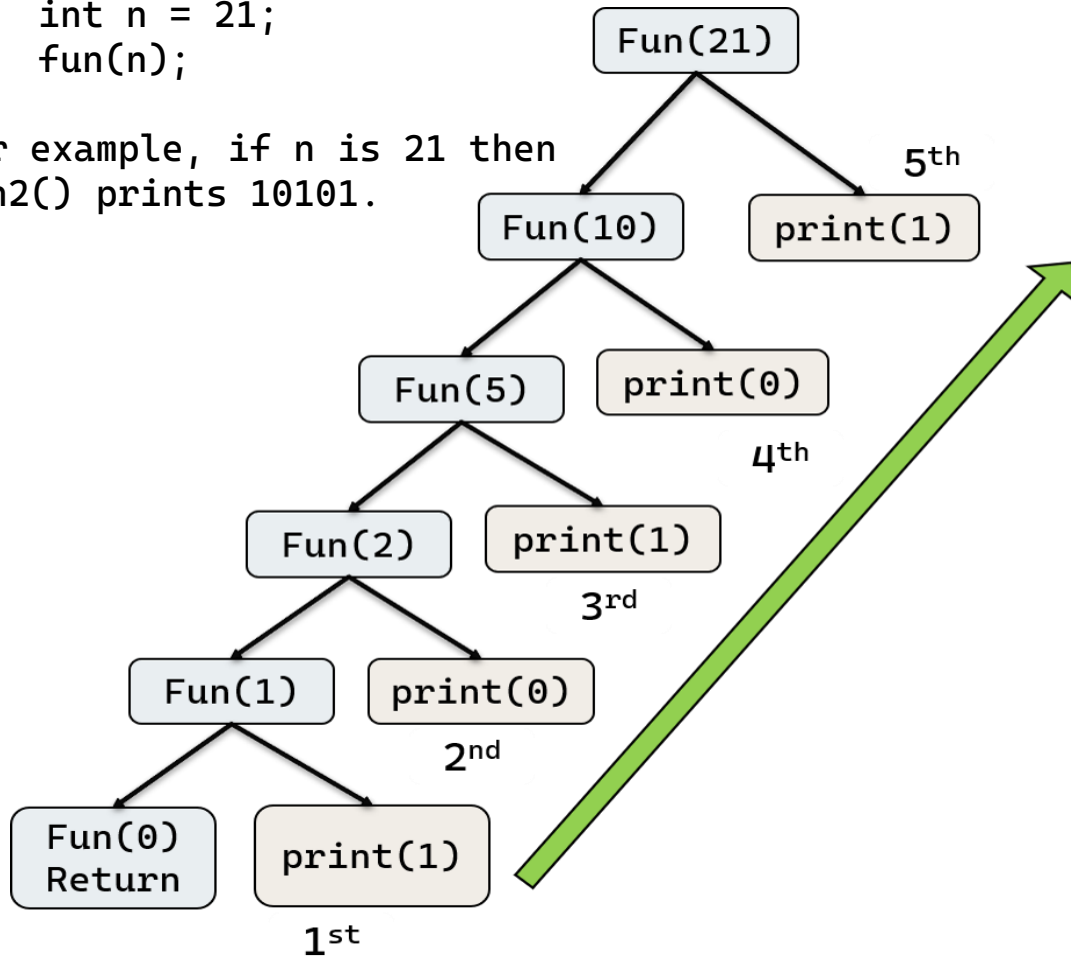
**Example – 4:** Predict the output pattern

```c
#define LIMIT 1000
void fun(int n){
    if (n > LIMIT |= n ≤ 0) return;
    printf("%d ", n);
    fun(2*n);
    printf("%d ", n);
}
int main(){
    int n = 100;
    fun(n);
}
```

For example fun2(100) prints :-
100, 200, 400, 800, 800, 400, 200, 100

Fun(100)

1st    print(100)    Fun(200)    print(100)
                                  9th

2nd    print(200)    Fun(400)    print(200)
                                  8th

3rd    print(400)    Fun(800)    print(400)
                                  7th

4th    print(800)    Fun(1600)    print(800)
                                   6th

5th    Return
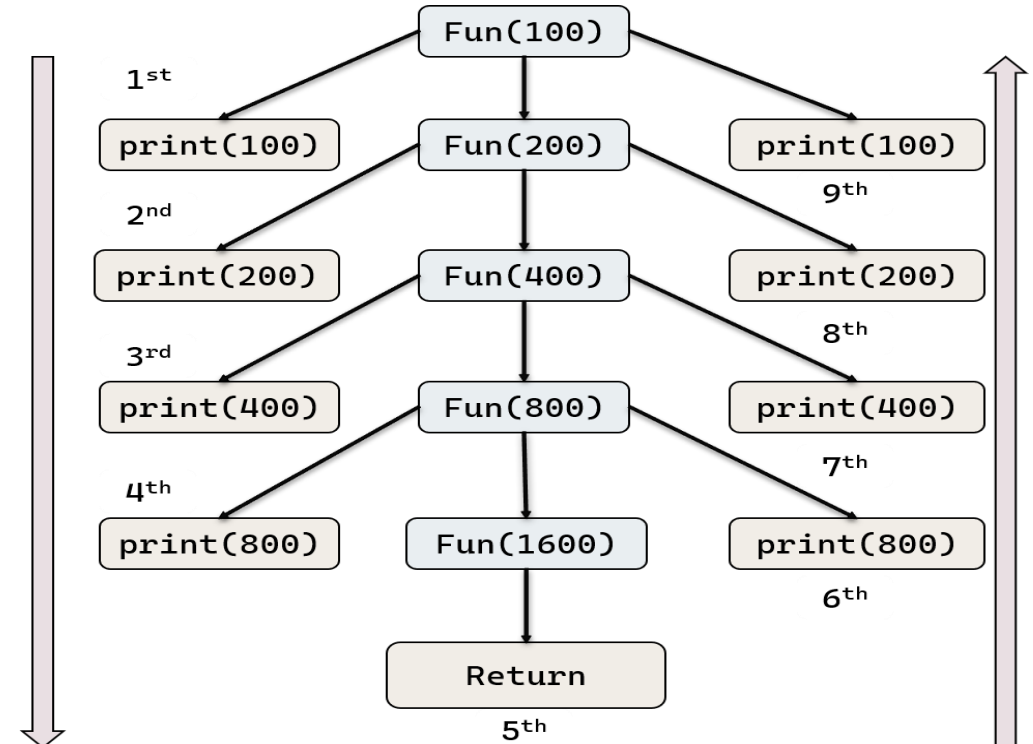
Example – 5: Predict the output pattern

```c
#include<stdio.h>
void fun(int x)
{
    if(x > 0){
        fun(--x);
        printf("%d ", x);
        fun(--x);
    }
    else return;
}
int main()
{
    int a = 4;
    fun(a);
}
```

Output:
0 1 2 0 3 0 1
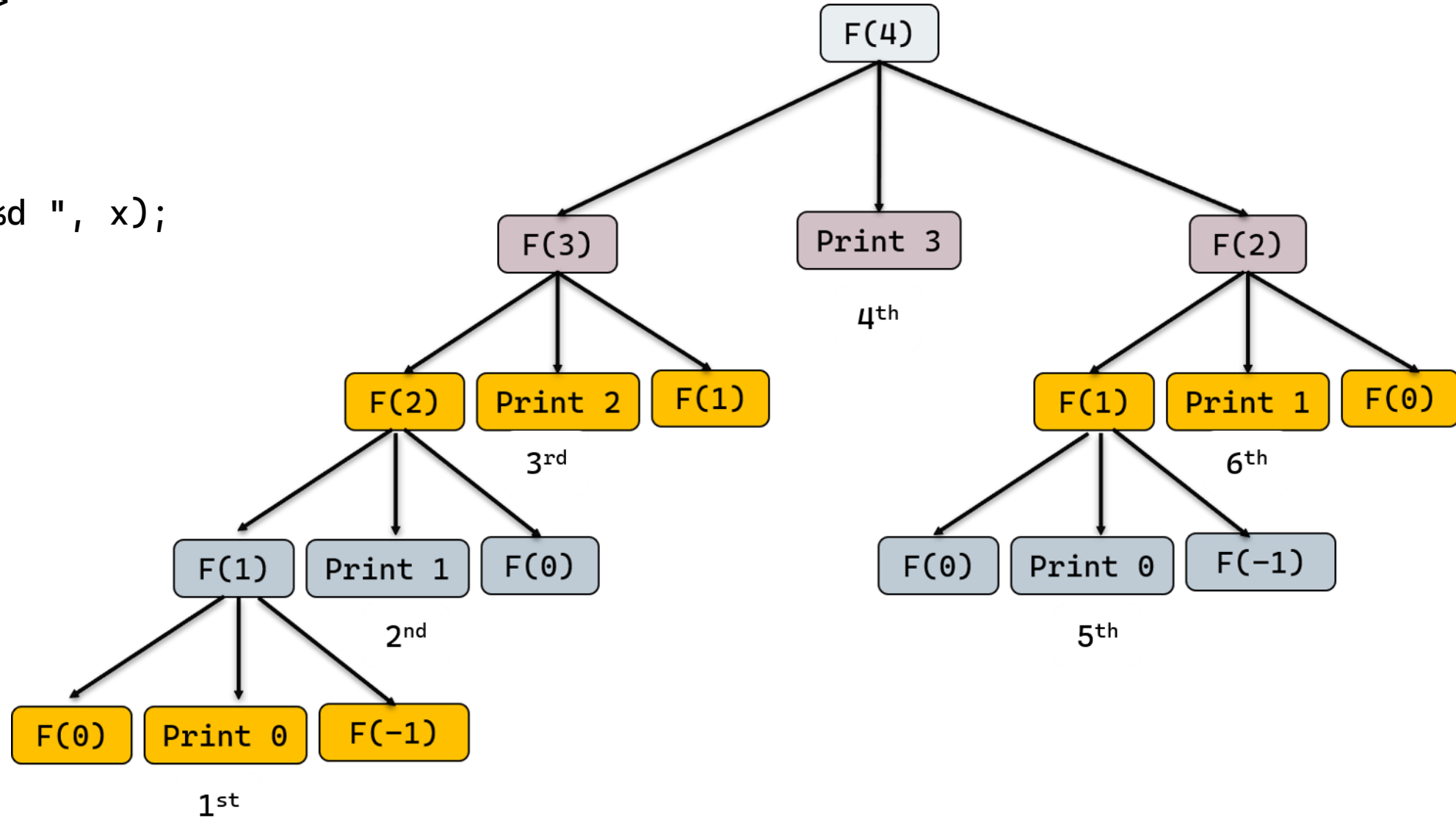
Example – 6: Predict the output pattern
```
#include <stdio.h>
void fun(int n)
{
    if (n > 0) {
        fun(n – 1);
        printf("%d ", n);
        fun(n – 1);
    }
}
int main()
{
    fun(4);
}
```

Output
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
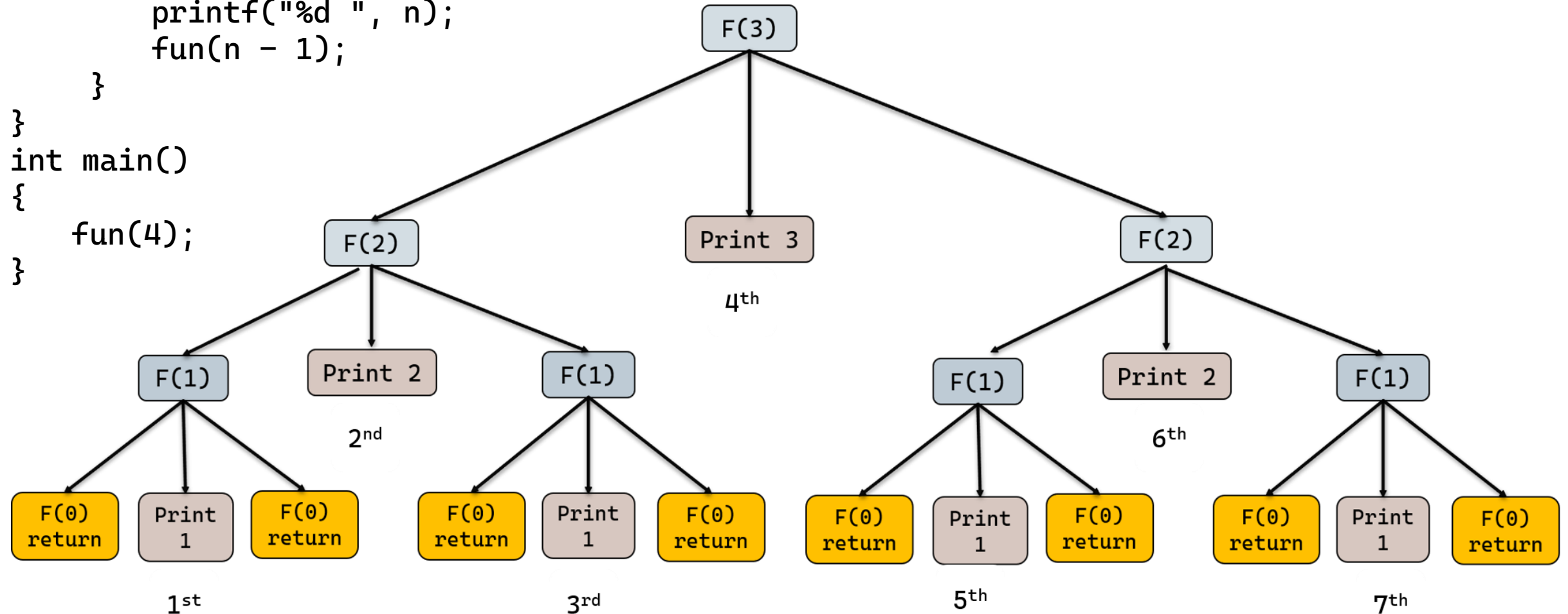
# Practice problems for recursion

**Example – 1:** Calculate the output

```
int fact(int n)
{
    if (n == 100)
        return 1;
    else
        return n*fact(n-1);
}
int main(){
    printf("%d", fact(105));
}
```

**Example – 2:** Assignment!!
Find the Recursion Tree and Calculate the output

```
int fun1(int n)
{
    if (n == 1)
        return 0;
    else
        return 1 + fun1(n / 2);
}
```

For example, if n is between 8 and 15 then fun1() returns 3.
If n is between 16 to 31 then fun1() returns 4.

Fact(105)

Print ⟹ 11587277520

105 * Fun(104)

105 * 110355024 = 11587277520

104 * Fun(103)

104 * 1061106 = 110355024

103 * Fun(102)

103 * 10302 = 1061106

102 * Fun(101)

102 * 101 = 10302

101 * Fun(100)

101 * 1 = 101
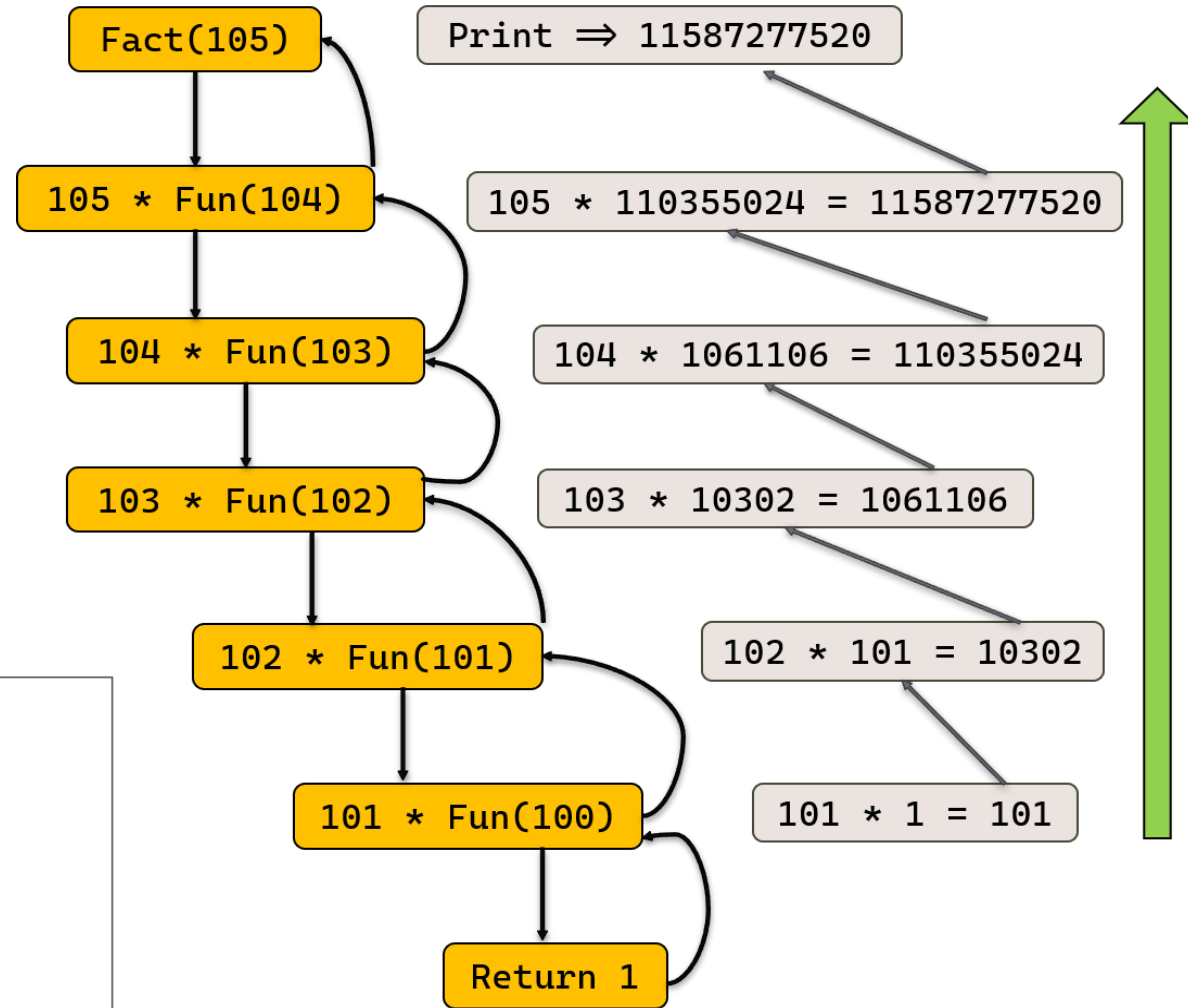
Return 1

Calculate the output

```c
#include <stdio.h>
int factTR(int n, int a){
    if (n ≤ 1)
        return a;
    return factTR(n - 1, n * a);
}
int fact(int n)
{
    return factTR(n, 1);
}
int main(){
    printf("%d", fact(5));
}
```

Example – 4: Assignment!!
Find the Recursion Tree and Calculate the output

```c
int fun1(int x, int y)
{
    if (x == 0)
        return y;
    else
        return fun1(x - 1, x + y);
}
```
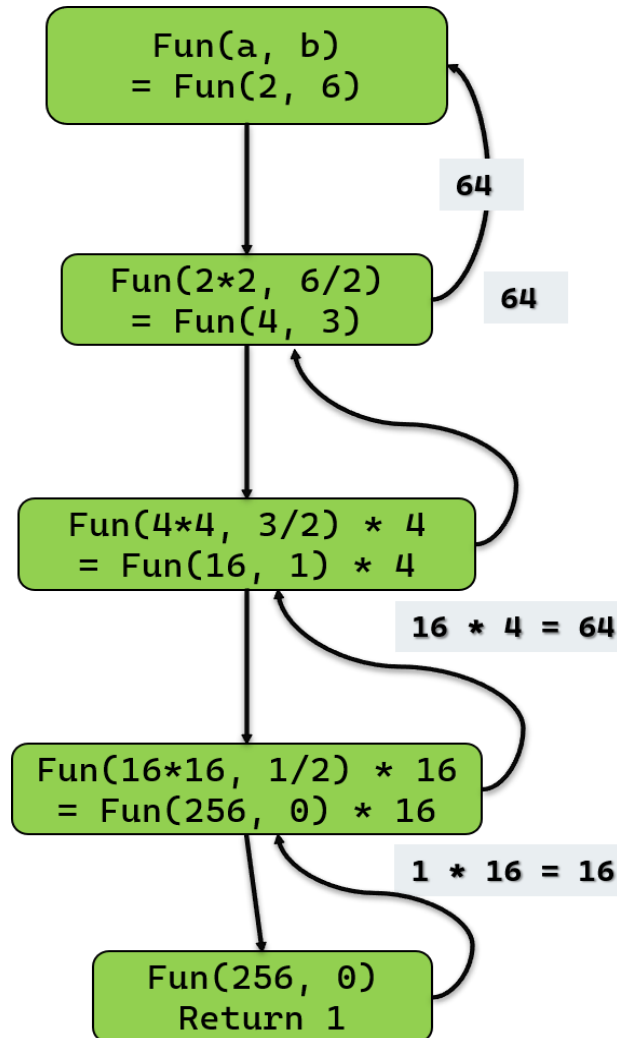
For example, if x is 5 and y is 2, then
fun should return 15 + 2 = 17.
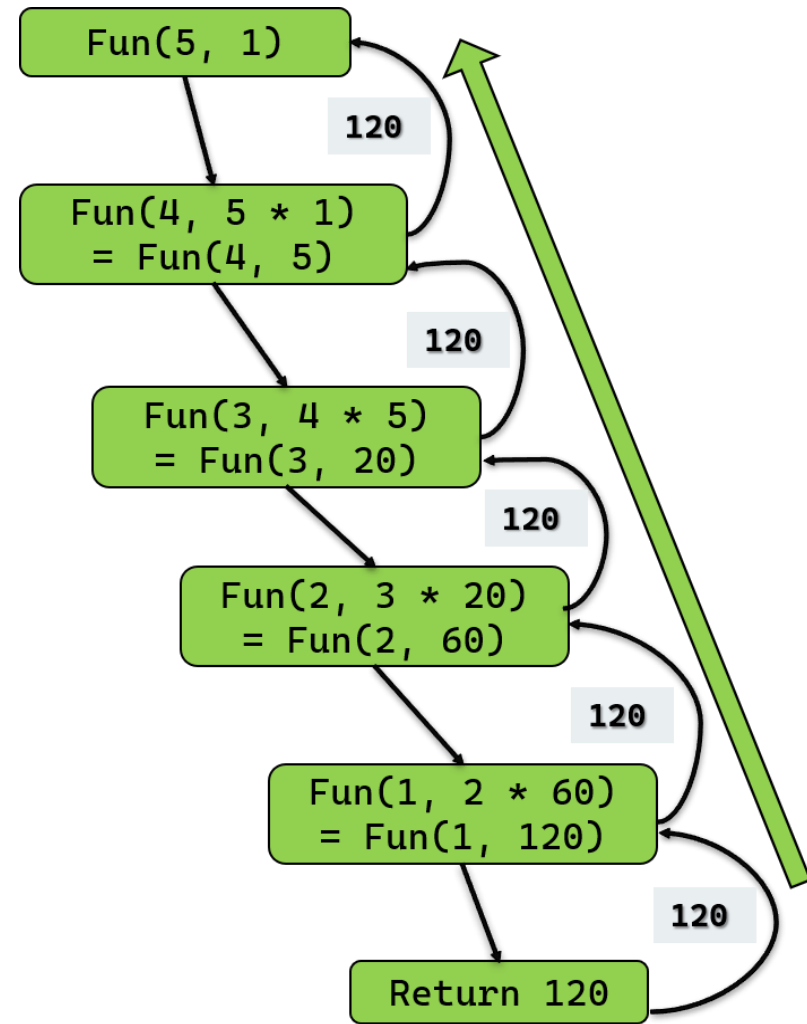
## Example – 5: Calculate the output

```c
#include<stdio.h>
int fun(int a, int b){
    if (b == 0)
        return 1;
    if (b % 2 == 0)
        return fun(a*a, b/2);
        return fun(a*a, b/2)*a;
}
int main(){
 printf("%d", fun(2, 6));
}
```

Output:
64

## Example – 6: Assignment!!
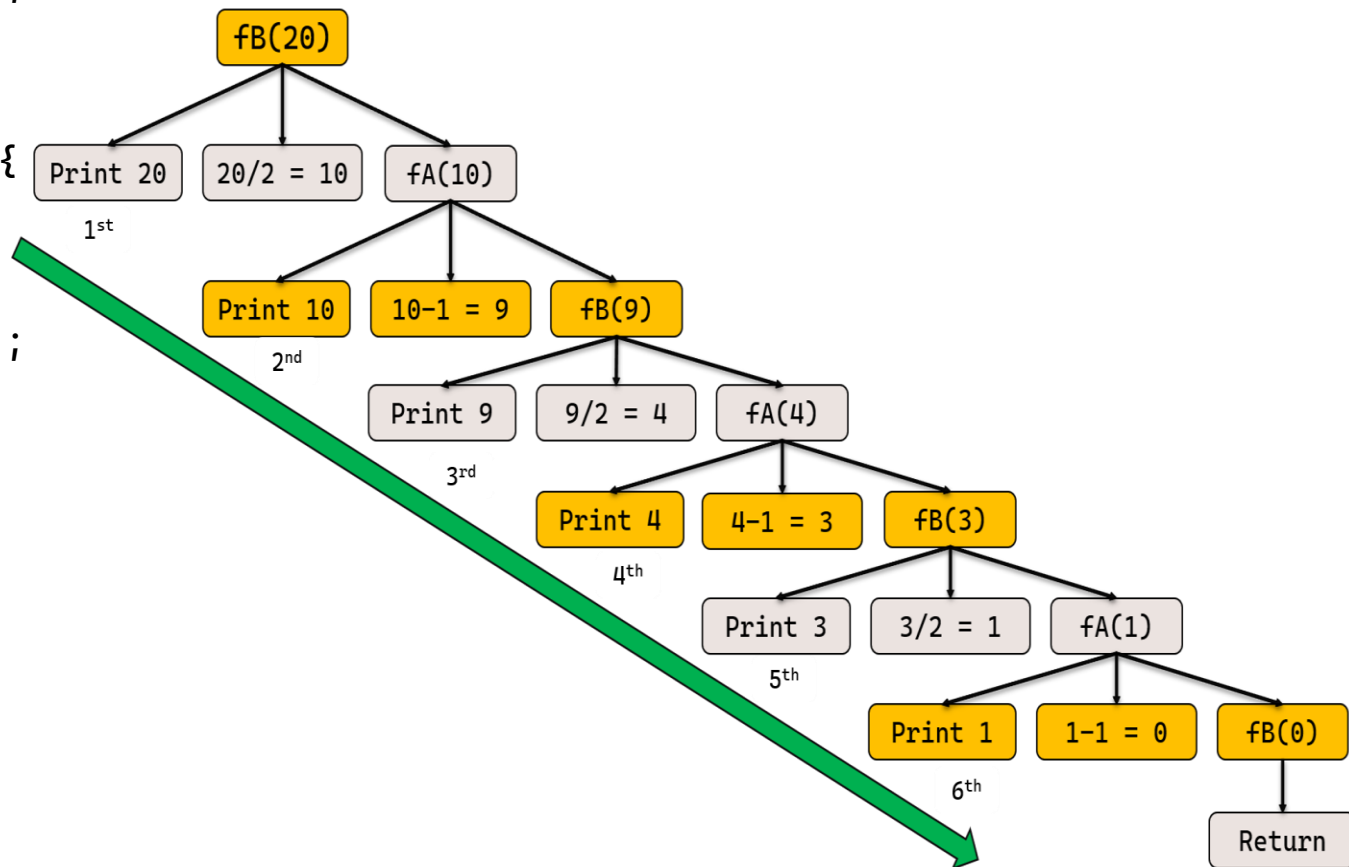Find the Recursion Tree and Calculate the output

```c
#include<stdio.h>
int fun(int a, int b) {
    if (b == 0)
        return 0;
    if (b % 2 == 0)
        return fun(a+a, b/2);
        return fun(a+a, b/2) + a;
}
int main(){
    printf("%d", fun(4, 3));
}
```

Output:
12

Fun(a, b)
= Fun(2, 6)

64

Fun(2*2, 6/2)
= Fun(4, 3)

64

Fun(4*4, 3/2) * 4
= Fun(16, 1) * 4

16 * 4 = 64

Fun(16*16, 1/2) * 16
= Fun(256, 0) * 16

1 * 16 = 16

Fun(256, 0)
Return 1

```
void functionB(int n);
void functionA(int n){
    if (n < 1) {
        return;
    }
    printf("%d ", n);
    n = n - 1;
    functionB(n);
}
void functionB(int n){
    if (n < 2) {
        return;
    }
    printf("%d ", n);
    n = n / 2;
    functionA(n);
}
int main(){
    functionB(20);
}
```

Find the Recursion Tree and Calculate the output

```
void functionB(int n);
void functionA(int n)
{
    if (n < 1) {
        return;
    }
    printf("%d ", n);
    n = n - 1;
    functionB(n);
}
void functionB(int n)
{
    if (n < 2) {
        return;
    }
    printf("%d ", n);
    n = n / 2;
    functionA(n);
}
int main(){
    functionA(20);
}
```

Calculate the output

```c
#include<stdio.h>
int fun(int n)
{

    if (n > 100)
    return n – 10;
    return fun(fun(n+11));
}
int main()
{

    printf(" %d ", fun(99));
}
```
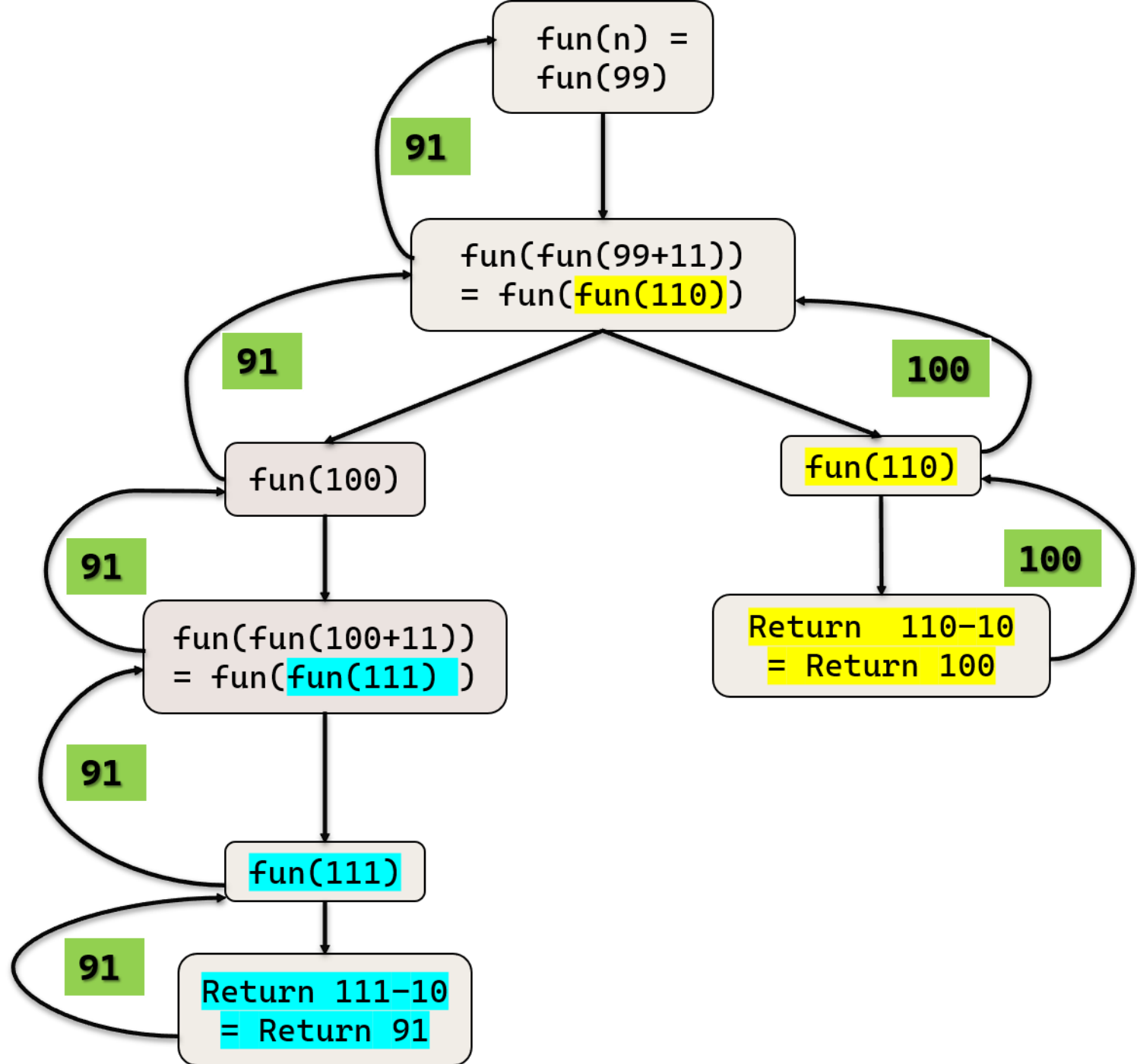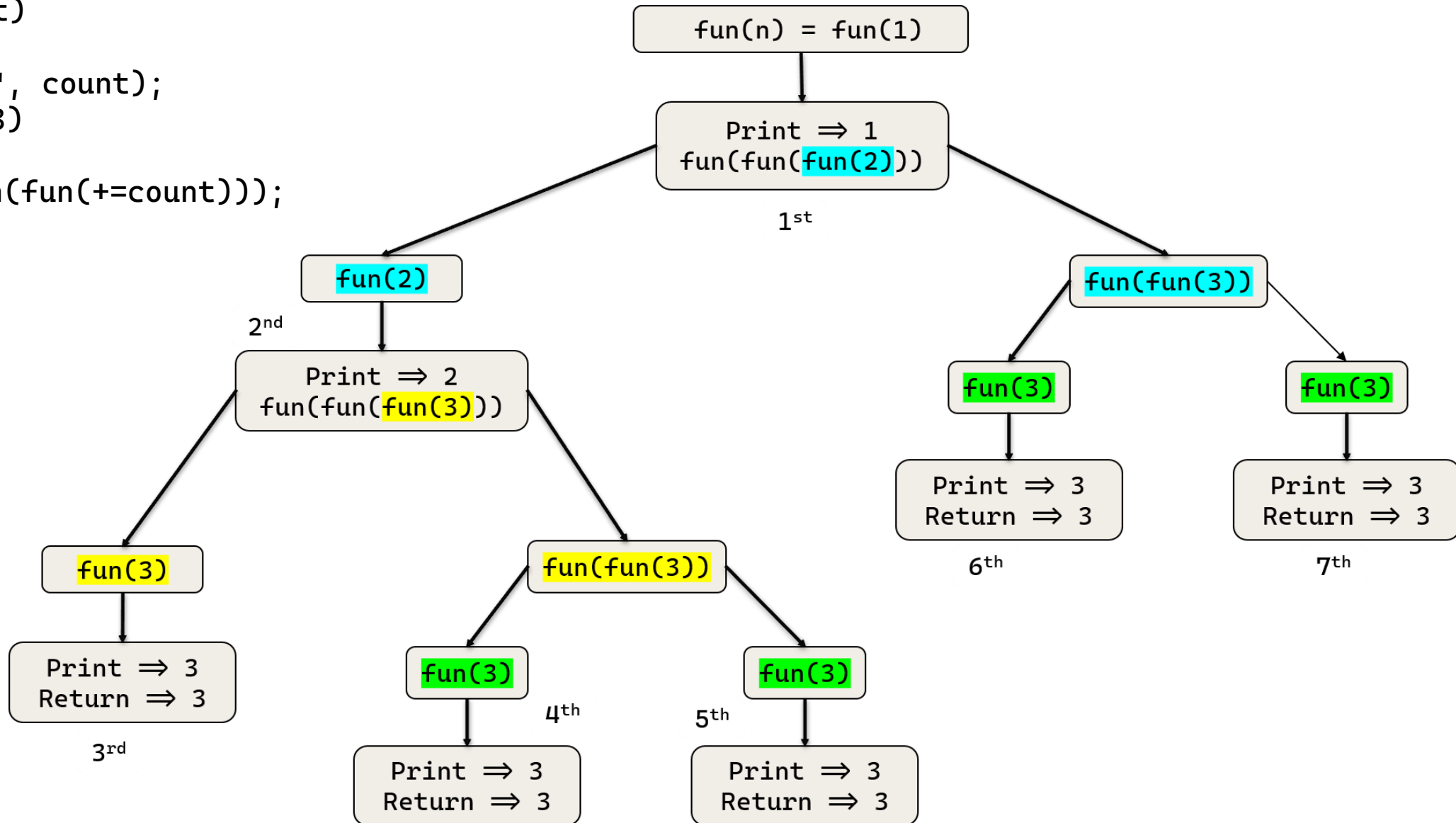
Output:
91

Guess the output of the  following code.

```c
#include<stdio.h>
int fun(int count)
{
    printf("%d ", count);
    if(count < 3)
    {
        fun(fun(fun(+=count)));
    }
    return
        count;
}
int main()
{
    fun(1);
}
```

Output:
1 2 3 3 3 3 3

fun(n) = fun(1)

Print ⇒ 1
fun(fun(fun(2)))

1st

fun(2)

2nd

fun(fun(3))

fun(3)

fun(3)

Print ⇒ 2
fun(fun(fun(3)))

fun(3)

3rd

Print ⇒ 3
Return ⇒ 3

fun(fun(3))

fun(3)

4th

fun(3)

5th

Print ⇒ 3
Return ⇒ 3

Print ⇒ 3
Return ⇒ 3

Print ⇒ 3
Return ⇒ 3

6th

Print ⇒ 3
Return ⇒ 3

7th

# ❖Printing Pyramid Patterns using Recursion

Write a recursion code for this pattern

```
*
* *
* * *
* * * *
* * * * *
```

```c
#include <stdio.h>
void printn(int num) {
    if (num == 0)
        return;
    printf("* ");
    printn(num - 1);
}
void pattern(int n, int i) {
    if (n == 0)
        return;
    printn(i);
    printf("\n");
    pattern(n - 1, i + 1);
}
int main() {
    int n = 5;
    pattern(n, 1);
    return 0;
}
```

Write a recursion code for this pattern

```
* * * * *
* * * *
* * *
* *
*
```

```c
#include <stdio.h>
void printn(int num) {
    if (num == 0)
        return;
    printf("* ");
    printn(num - 1);
}
void pattern(int n) {
    if (n == 0)
        return;
    printn(n);
    printf("\n");
    pattern(n - 1);
}
int main() {
    int n = 5;
    pattern(n);
    return 0;
}
```

# Online Judge problems

- https://codeforces.com/group/MWSDmqGsZm/contest/223339/problem/A

- https://codeforces.com/group/MWSDmqGsZm/contest/223339/problem/B

- https://codeforces.com/group/MWSDmqGsZm/contest/223339/problem/C

- https://codeforces.com/group/MWSDmqGsZm/contest/223339/problem/D

- https://codeforces.com/group/MWSDmqGsZm/contest/223339/problem/F

- https://codeforces.com/group/MWSDmqGsZm/contest/223339/problem/J

- https://codeforces.com/group/MWSDmqGsZm/contest/223339/problem/O