ReCurSion

Course Code: CSE-121

Course Title: Structural Programming Language

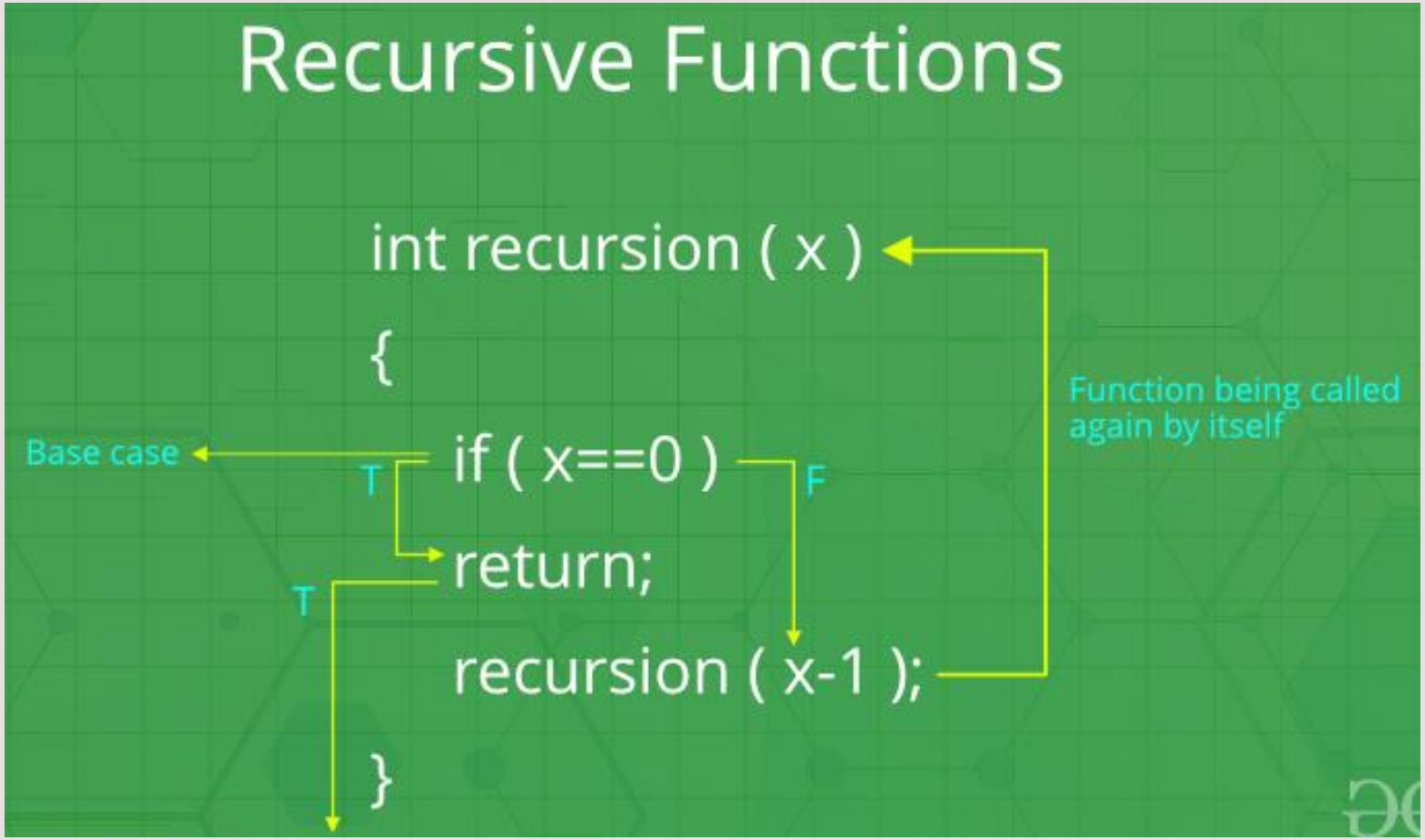
Khandaker Jannatul Ritu, Lecturer, BAIUST

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
    3) Print Even Numbers 2 to N
    4) Print Odd Numbers 1 to N
- 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 Practise problems
- Printing Pyramid Patterns using Recursion

❑ Introduction to Recursion in C

✓ *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.*



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

❑ Fundamental Components of C Recursion

The fundamental of recursion consists of two objects which are essential for any recursive function. These are:
1.Recursion Case
2.Base Condition

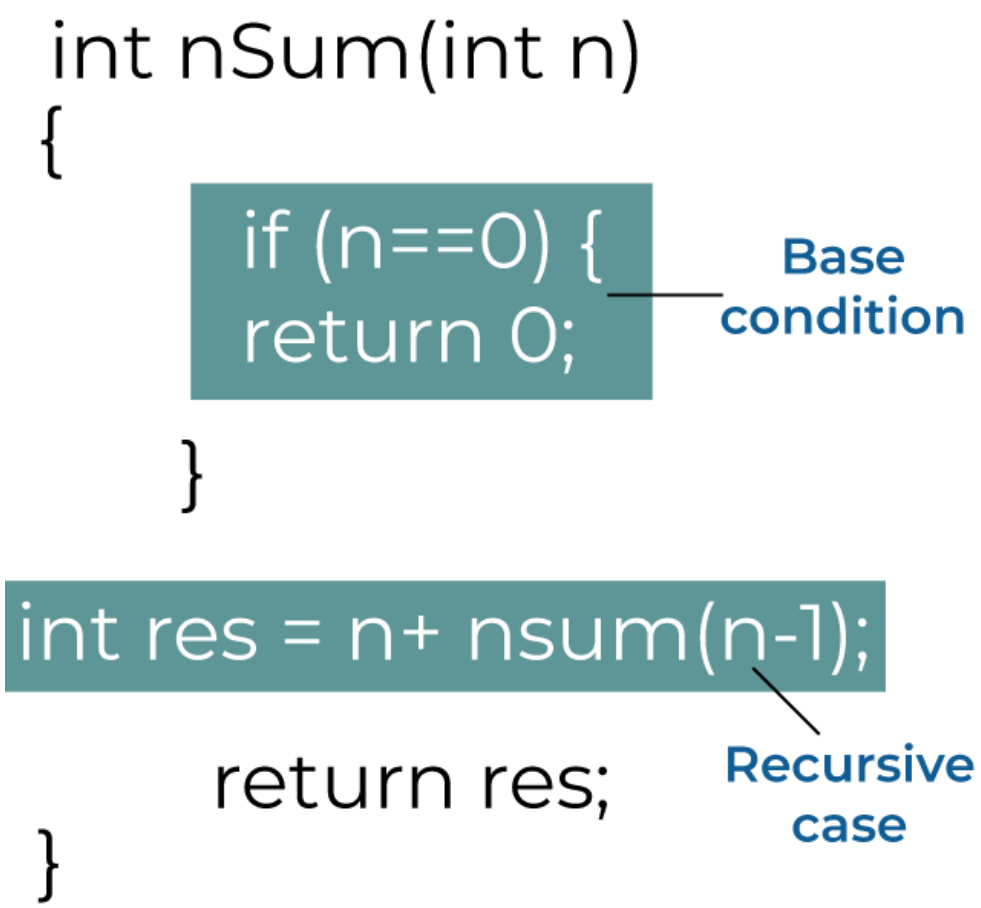
1. Recursion Case

The recursion case refers to the recursive call present in the recursive function. It decides what type of recursion will occur and how the problem will be divided into smaller subproblems.
The recursion case defined in the nSum() function of the above example is:
int res = n + nSum(n - 1);

2. Base Condition

The base condition specifies when the recursion is going to terminate. It is the condition that determines the exit point of the recursion.
Considering the above example again, the base condition defined for the nSum() function:

```
if (n == 0) {
   return 0;
}
```



❑ Need of Recursive Function:

A recursive function is a function that solves a problem by solving smaller instances of the same problem. This technique is often used in programming to solve problems that can be broken down into simpler, similar subproblems.

1. Solving complex tasks:

Recursive functions break complex problems into smaller instances of the same problem, resulting in compact and readable code.

2. Divide and Conquer:

Recursive functions are suitable for divide-and-conquer algorithms such as merge sort and quicksort, breaking problems into smaller subproblems, solving them recursively, and merging the solutions with the original problem.

3. Backtracking:

Recursive backtracking is ideal for exploring and solving problems like N-Queens and Sudoku.

4. Dynamic programming:

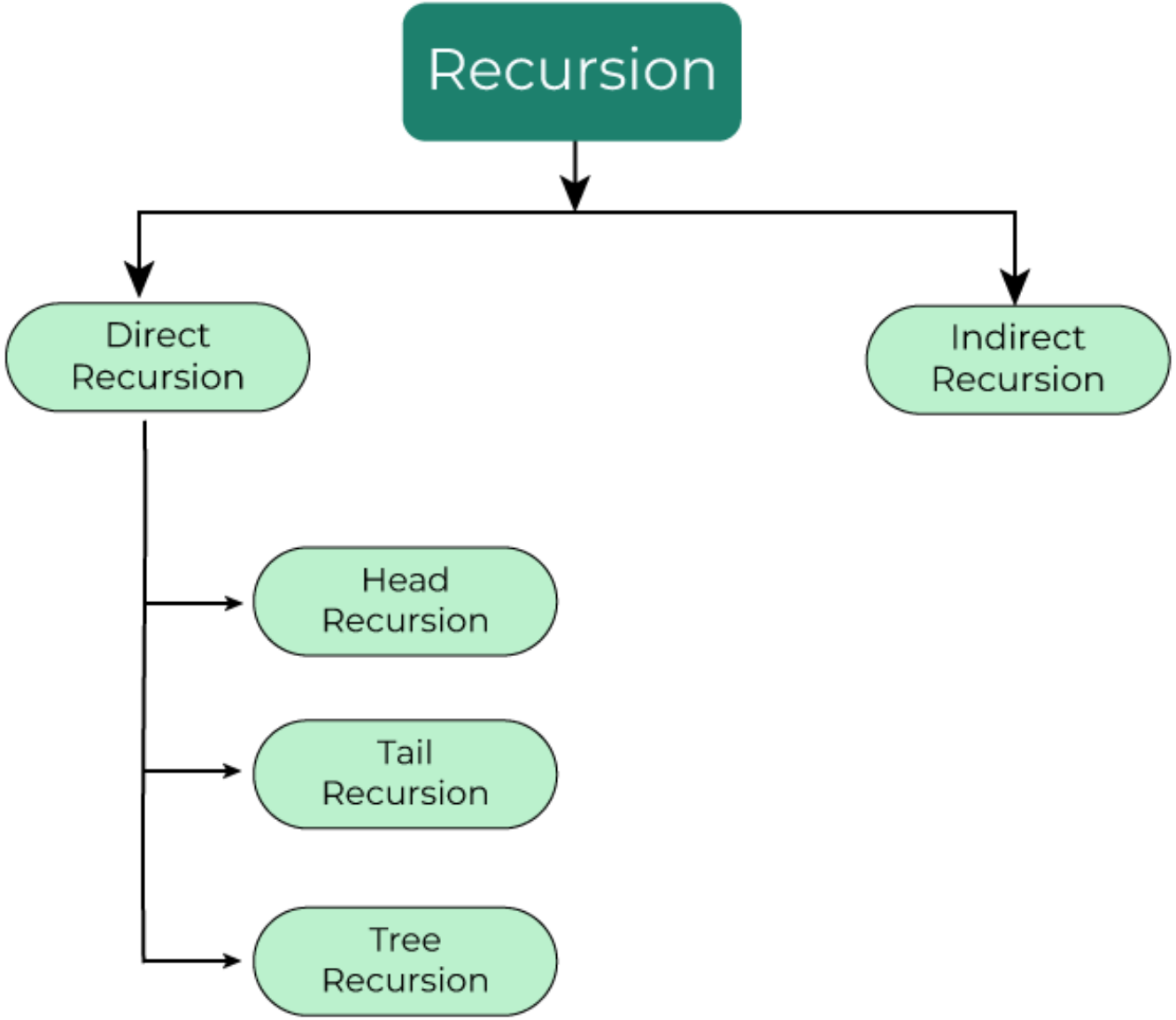
Recursive functions efficiently solve dynamic programming problems by solving subproblems and combining their solutions into a complete solution.

5. Tree and graph structures:

Recursive functions are great for working with tree and graph structures, simplifying traversal and pattern recognition tasks.

❑ Types of C Recursion

In C, recursion can be classified into different types based on what kind of recursive case is present. These types are:



❑ Indirect Recursion:-

```
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.

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

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.

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

      print(n - 1);
}
```

3.What is Tree Recursion

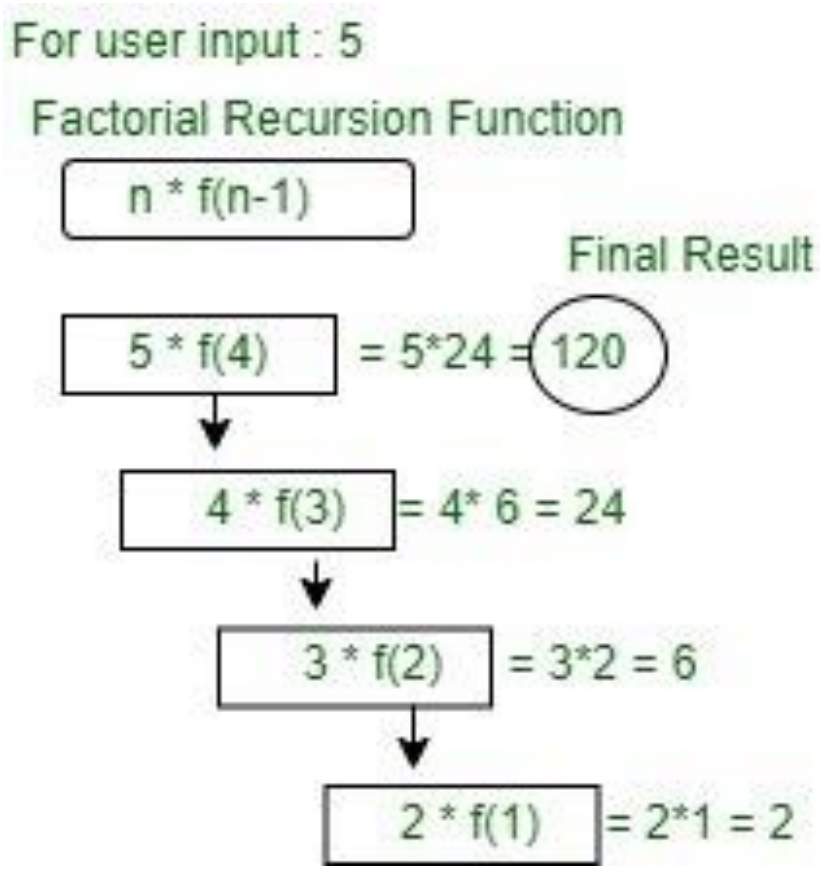
In **tree recursion**, there are multiple recursive calls present in the body of the function. Due to this, while tracing the program flow, it makes a tree-like structure, hence the name Tree Recursion.

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

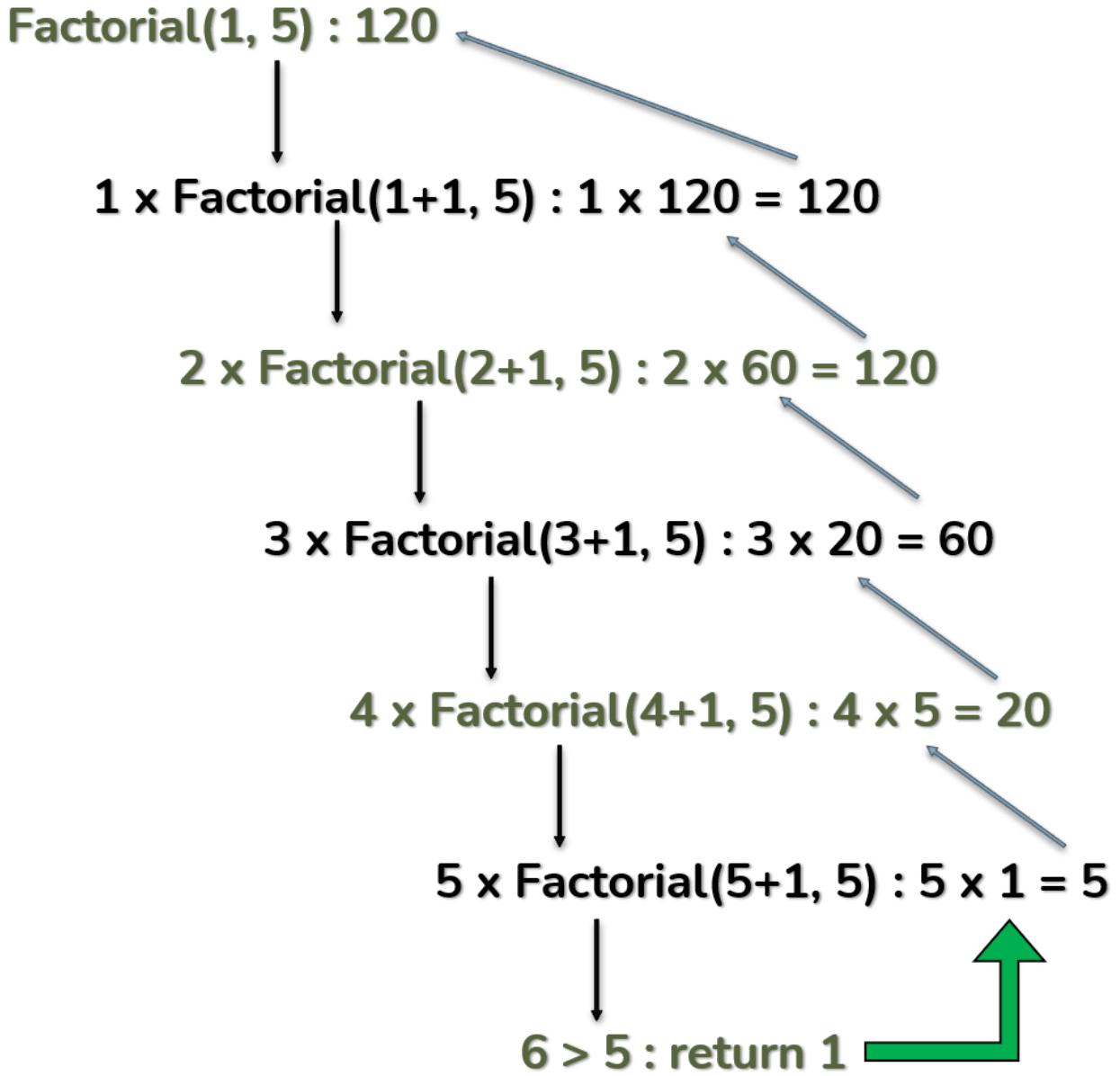
❑Approaches of Recursion

There are two approaches for a recursive function.

a. Top down approach

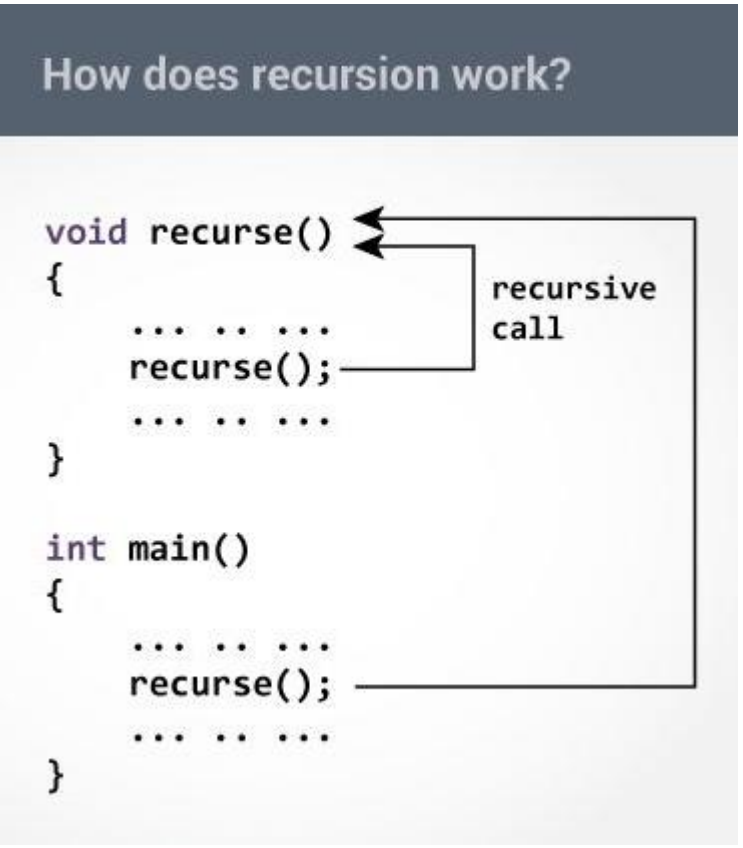


b. Bottom up approach



❑ *Difference between Recursion and Iteration*		
No	**Recursion**	**Iteration**
1)	Terminates when the base case becomes true.	Terminates when the condition becomes false.
2)	Used with functions.	Used with loops.
3)	Every recursive call needs extra space in the stack memory.	Every iteration does not require any extra space.
4)	Smaller code size.	Larger code size.

❑How Recursion works in C?



Print 1 to N
1. manual
2. function
3. recursion
   a. top-down
   b. bottom-up

Print N to 1
1. manual
2. function
3. recursion
   a. top-down
   b. bottom-up

Print Even Numbers 2 to N
1. manual
2. function
3. recursion
   a. top-down
   b. bottom-up

Print Odd Numbers 1 to N
1. manual
2. function
3. recursion
   a. top-down
   b. bottom-up

1.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

```
#include <stdio.h>
int main(){
    int N=10;
    for(int i = 1; i≤ N; i++)
    {
        printf("%d ", i);
    }
}
```

b. Using Normal Function

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

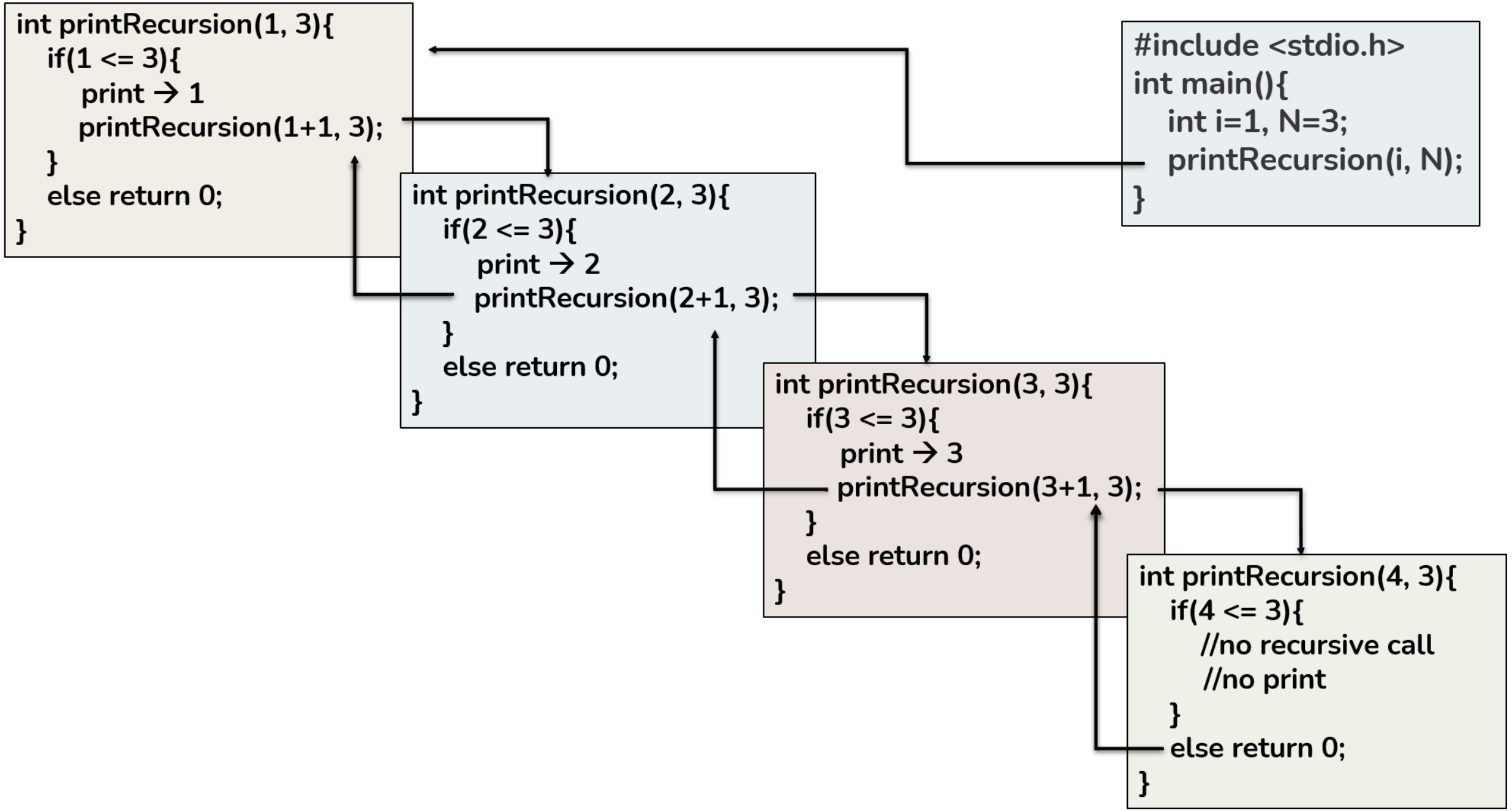
c. Using Recursive Function [Bottom-Up]

```
#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);
}
```

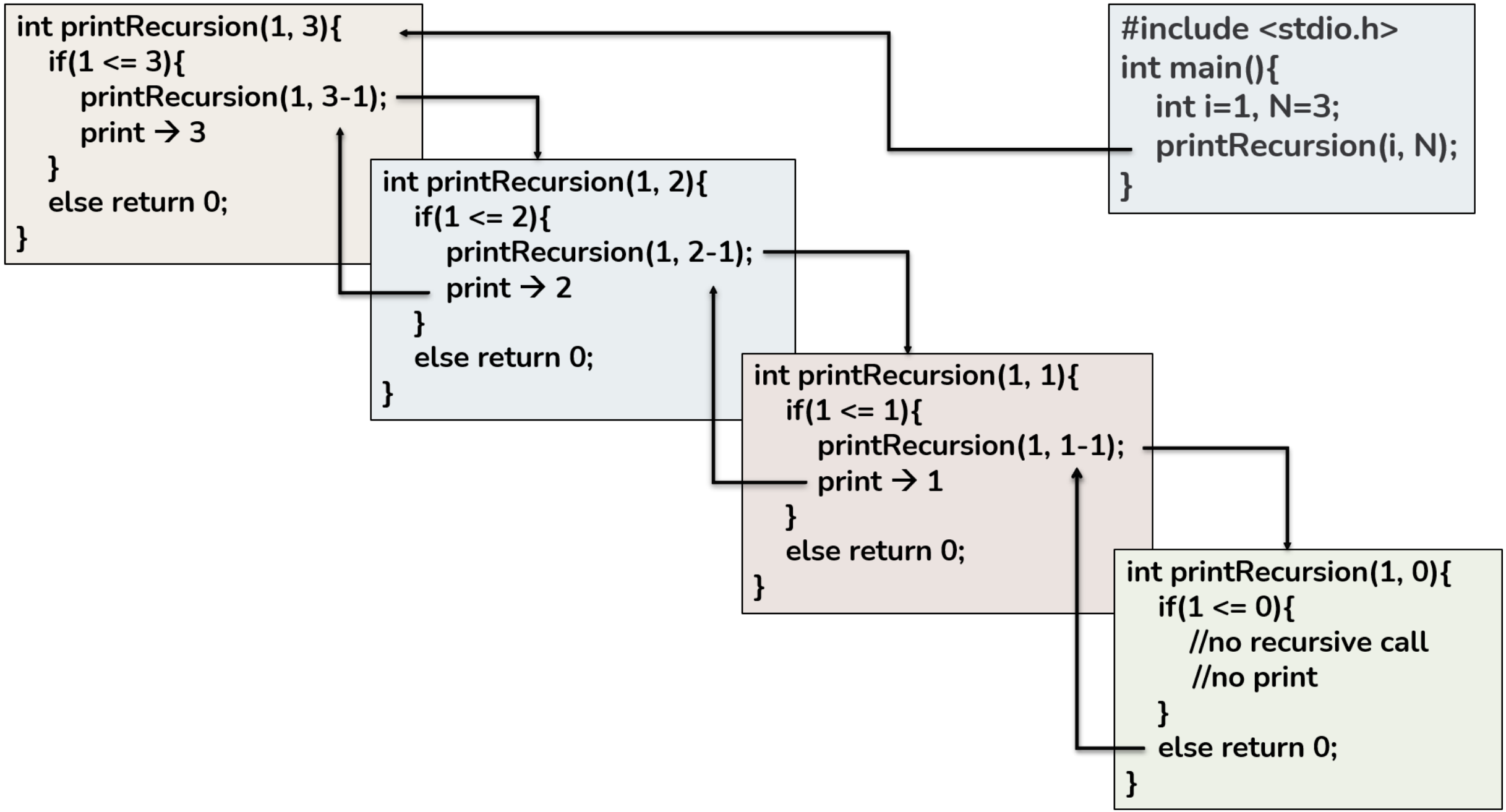
d. Using Recursive Function [Top-Down]

```
#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);
}
```

c. Using Recursive Function [Bottom-Up]



d. Using Recursive Function [Top-Down]



2.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

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

b. Using Normal Function

```
#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]

```
#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);
}
```

d. Using Recursive Function [Top-Down]

```
#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);
}
```

c. Using Recursive Function [Bottom-Up]

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

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

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

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

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

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

d. Using Recursive Function [Top-Down]

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

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

3.Print even numbers 2 to n without using loops

You are given an integer N. Print even numbers from 2 to N without the help of loops.
Input: N = 10
Output: 2 4 6 8 10

a. Manually

```
#include <stdio.h>
int main(){
   int N=10;
   for(int i = 2; i ≤ N; i=i+2)
   {
        printf("%d ", i);
   }
}
```

b. Using Normal Function

```
#include <stdio.h>
int printFunction(int N){
  for(int i = 2; i ≤ N; i=i+2)
  {
        printf("%d ", i);
  }
}
int main(){
    int N=10;
    printFunction(N);
}
```

c. Using Recursive Function [Bottom-Up]

```
#include <stdio.h>
int printRecursion(int i, int N)
{
 if(i ≤ N){
   printf("%d ",i);
   printRecursion(i+2, N);
 }
 else return 0;
}
int main(){
    int i=2, N=10;
    printRecursion(i, N);
}
```

d. Using Recursive Function [Top-Down]

```
#include <stdio.h>
int printRecursion(int i, int N)
{
    if(i ≤ N){
      printRecursion(i, N-2);
      printf("%d ", N);
    }
    else return 0;
}
int main(){
    int i=2, N=10;
    printRecursion(i, N);
}
```

c. Using Recursive Function [Bottom-Up]

```
int printRecursion(2, 6){
    if(2 <= 6){
        print → 2
        printRecursion(2+2, 6);
    }
    else return 0;
}
```

```
int printRecursion(4, 6){
    if(4 <= 6){
        print → 4
        printRecursion(4+2, 6);
    }
    else return 0;
}
```

```
int printRecursion(6, 6){
    if(6 <= 6){
        print → 6
        printRecursion(6+2, 6);
    }
    else return 0;
}
```

```
int printRecursion(8, 6){
    if(2 <= 0){
        //no recursive call
        //no print
    }
    else return 0;
}
```

```
#include <stdio.h>
int main(){
    int i=2, N=6;
    printRecursion(i, N);
}
```

```
if(i <= N){
        printf("%d ",i);
        printRecursion(i+2, N);
    }
    else return 0;
}
```

d. Using Recursive Function [Top-Down]

```
int printRecursion(2, 6){
    if(2 <= 6){
        printRecursion(2, 6-2);
        print → 6
    }
    else return 0;
}
```

```
#include <stdio.h>
int main(){
    int i=2, N=6;
    printRecursion(i, N);
}
```

```
int printRecursion(2, 4){
    if(2 <= 4){
        printRecursion(2, 4-2);
        print → 4
    }
    else return 0;
}
```

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

```
int printRecursion(2, 0){
    if(2 <= 0){
        //no recursive call
        //no print
    }
    else return 0;
}
```

```
int printRecursion(int i, int N){
        if(i <= N){
            printRecursion(i, N-2);
            printf("%d ", N);
        }
else return 0;
}
```

4.Print odd numbers 1 to n without using loops

You are given an integer N. Print even numbers from 2 to N without the help of loops.
Input: N = 10
Output: 1 3 5 7 9

a. Manually

```
#include <stdio.h>
int main(){
    int N=10;
   for(int i = 1; i ≤ N; i=i+2)
    {
        printf("%d ", i);
    }
}
```

b. Using Normal Function

```
#include <stdio.h>
int printFunction(int N){
  for(int i = 1; i ≤ N; i=i+2)
    {
        printf("%d ", i);
    }
}
int main(){
    int N=10;
    printFunction(N);
}
```

c. Using Recursive Function [Bottom-Up]

```
#include <stdio.h>
int printRecursion(int i, int N)
{
    if(i ≤ N){
        printf("%d ",i);
        printRecursion(i+2, N);
    }
    else return 0;
}
int main(){
    int i=1, N=10;
    printRecursion(i, N);
}
```

d. Using Recursive Function [Top-Down]

```
#include <stdio.h>
int printRecursion(int i, int N)
{
    if(i ≤ N){
        printRecursion(i, N-2);
        printf("%d ", N);
    }
else return 0;
}
int main(){
    int i=1, N=10;
    printRecursion(i, N);
}
```

c. Using Recursive Function [Bottom-Up]

```
int printRecursion(1, 5){
    if(1 <= 5){
        print → 1
        printRecursion(1+2, 5);
    }
    else return 0;
}
```

```
int printRecursion(3, 5){
    if(3 <= 5){
        print → 3
        printRecursion(3+2, 5);
    }
    else return 0;
}
```

```
int printRecursion(5, 5){
    if(5 <= 5){
        print → 5
        printRecursion(5+2, 5);
    }
    else return 0;
}
```

```
int printRecursion(7, 5){
    if(2 <= 0){
        //no recursive call
        //no print
    }
    else return 0;
}
```

```
#include <stdio.h>
int main(){
    int i=1, N=5;
    printRecursion(i, N);
}
```

```
int printRecursion(int i, int N){
 if(i <= N){
        printf("%d ",i);
        printRecursion(i+2, N);
    }
    else return 0;
}
```

d. Using Recursive Function [Top-Down]

```
int printRecursion(1, 5){
    if(1 <= 5){
        printRecursion(1, 5-2);
        print → 5
    }
    else return 0;
}
```

```
#include <stdio.h>
int main(){
    int i=1, N=5;
    printRecursion(i, N);
}
```

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

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

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

```
int printRecursion(int i, int N){
        if(i <= N){
            printRecursion(i, N-2);
            printf("%d ", N);
        }
else return 0;
}
```

Assignment

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]

2) **C program to calculate n^{th} Fibonacci numbers**
   1)Using Array[bottom – up]
   2)Using recursive function[top – down]

3) **C program to calculate sum of natural numbers from 1 to N**
   1)Using Array[bottom – up]
   2)Using recursive function[top – down]

1)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

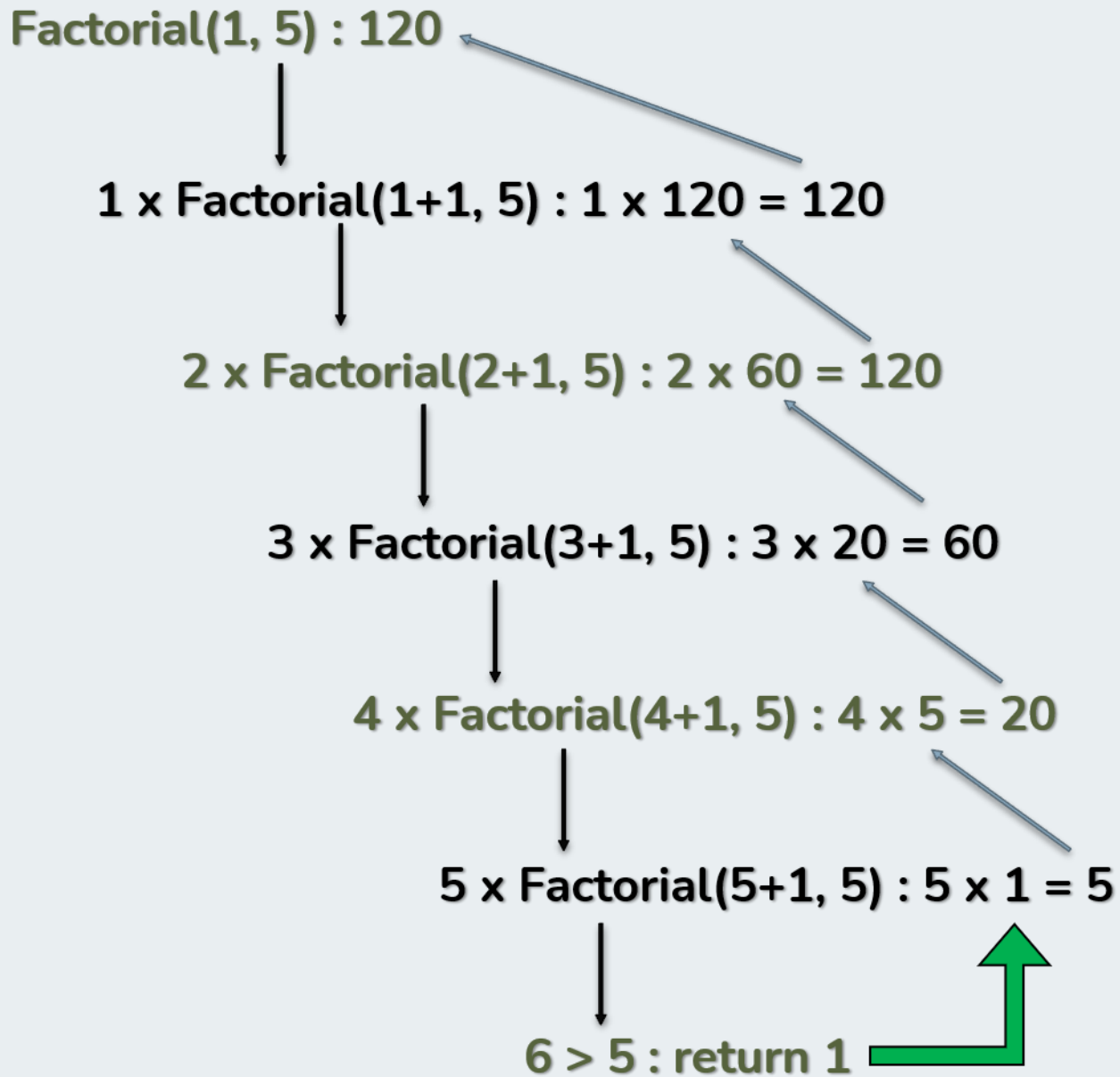
```
#include <stdio.h>
int main(){
    int fact = 1;
    int N=5;
    for(int i=1; i≤N; i++)
    {
        fact = fact * i;
    }
    printf("%d", fact);
}
```

2. Using Function

```
#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]

```
#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));
}
```



Using recursive function [Bottom – up]

```
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;
}
```

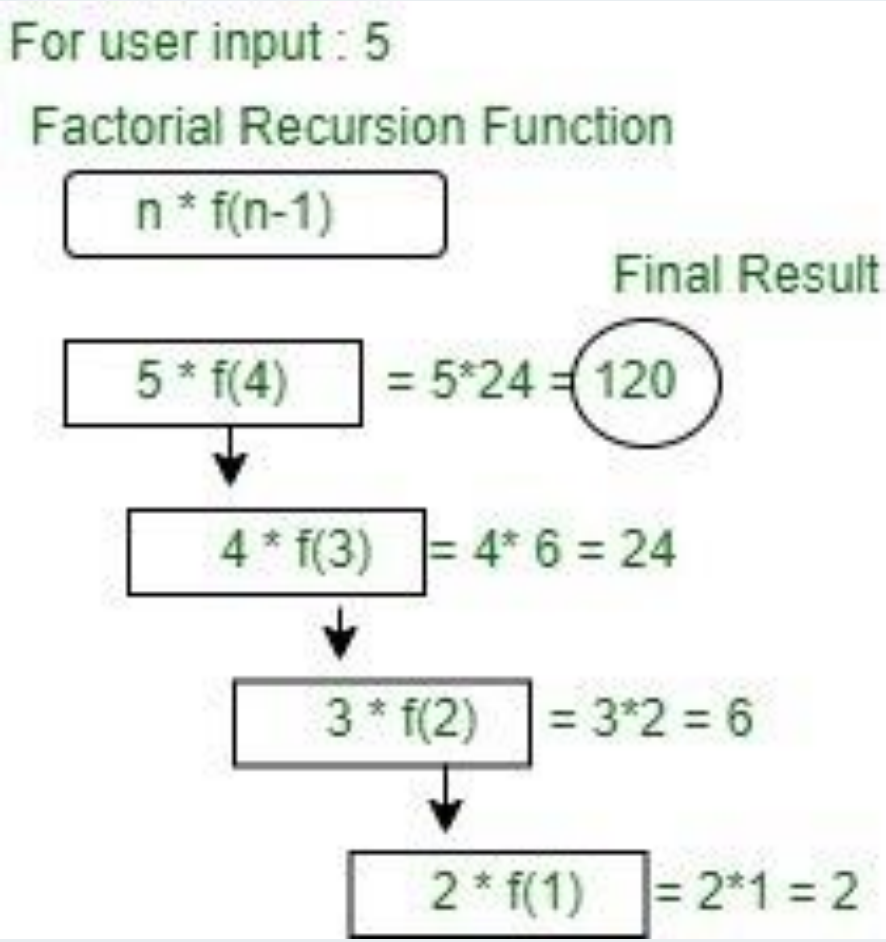
```
int Factorial(3, 5){
    if(3 <= 5){
        return 3 * Factorial(3+1, 5);
    }
    return 1;
}
```

```
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;
}
```

4. Using recursive function [Top – down]

```
#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));
}
```



Using recursive function [Top – down]

```
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);
    }
```

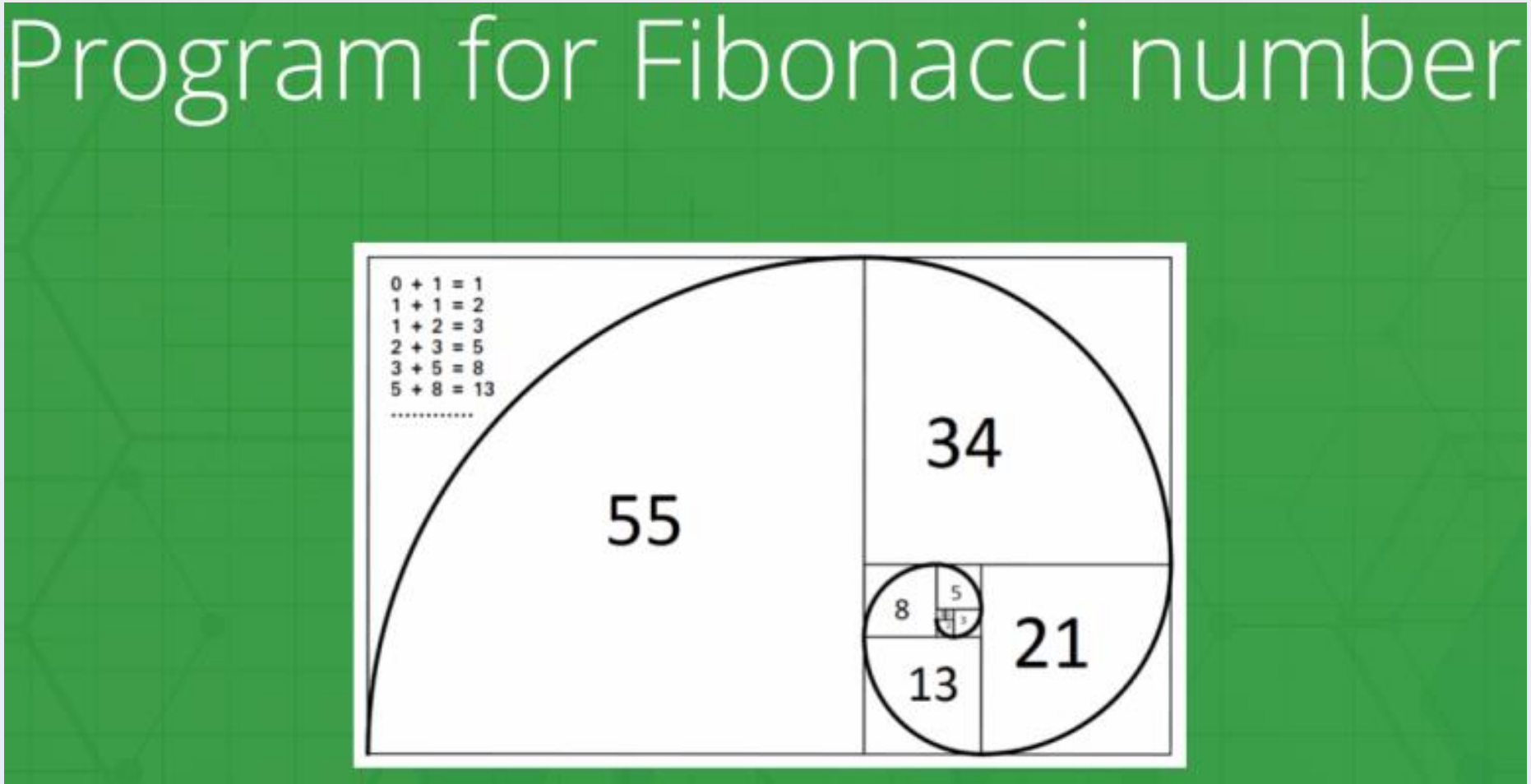
```
int Fact(1){
    if(1 <= 1)  return 1 ;
    else  return 1 * Fact(1-1);
    }
```

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

2. C program to Calculate n^{th} Fibonacci number

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, ……..



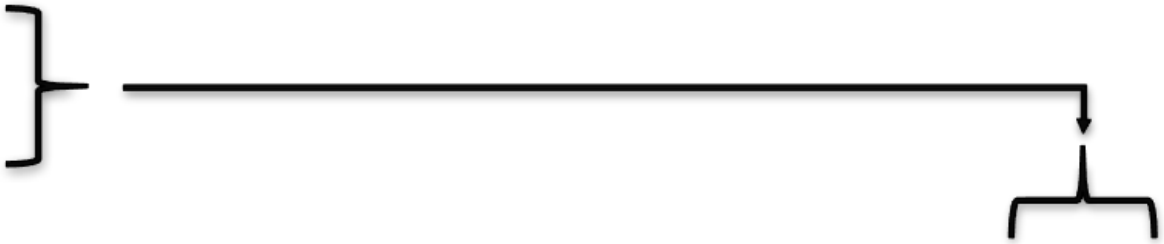
❑ How Fibonacci sequence works?

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……

Using Array [bottom – up]:

0	1	2	3	4	5	6	7	8	9
0	1	0 + 1 = 1	1 + 1 = 2	1 + 2 = 3	2 + 3 = 5	3 + 5 = 8	5 + 8 = 13	8 + 13 = 21	13 + 21 = 34

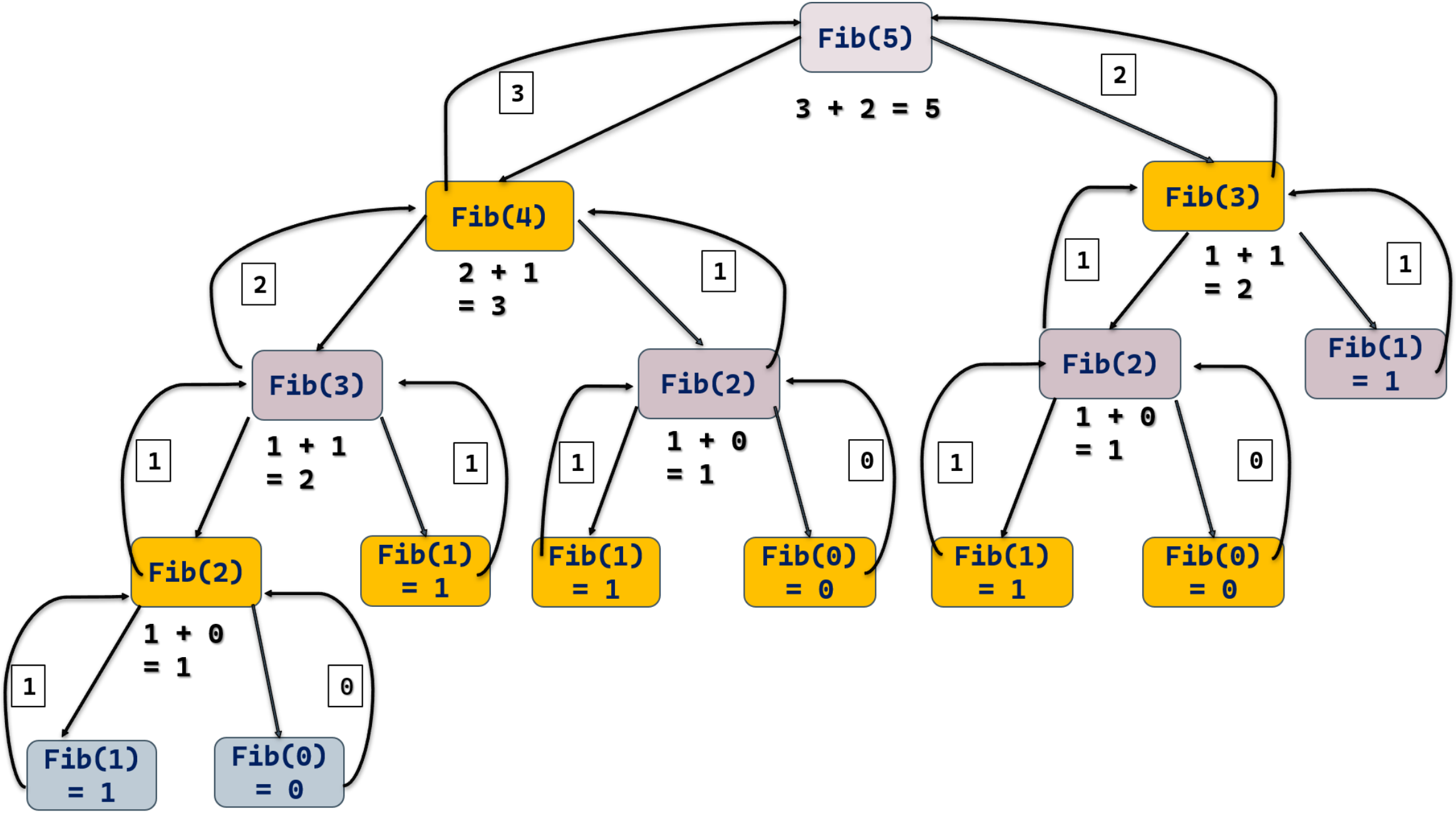
Using Array [bottom – up]:

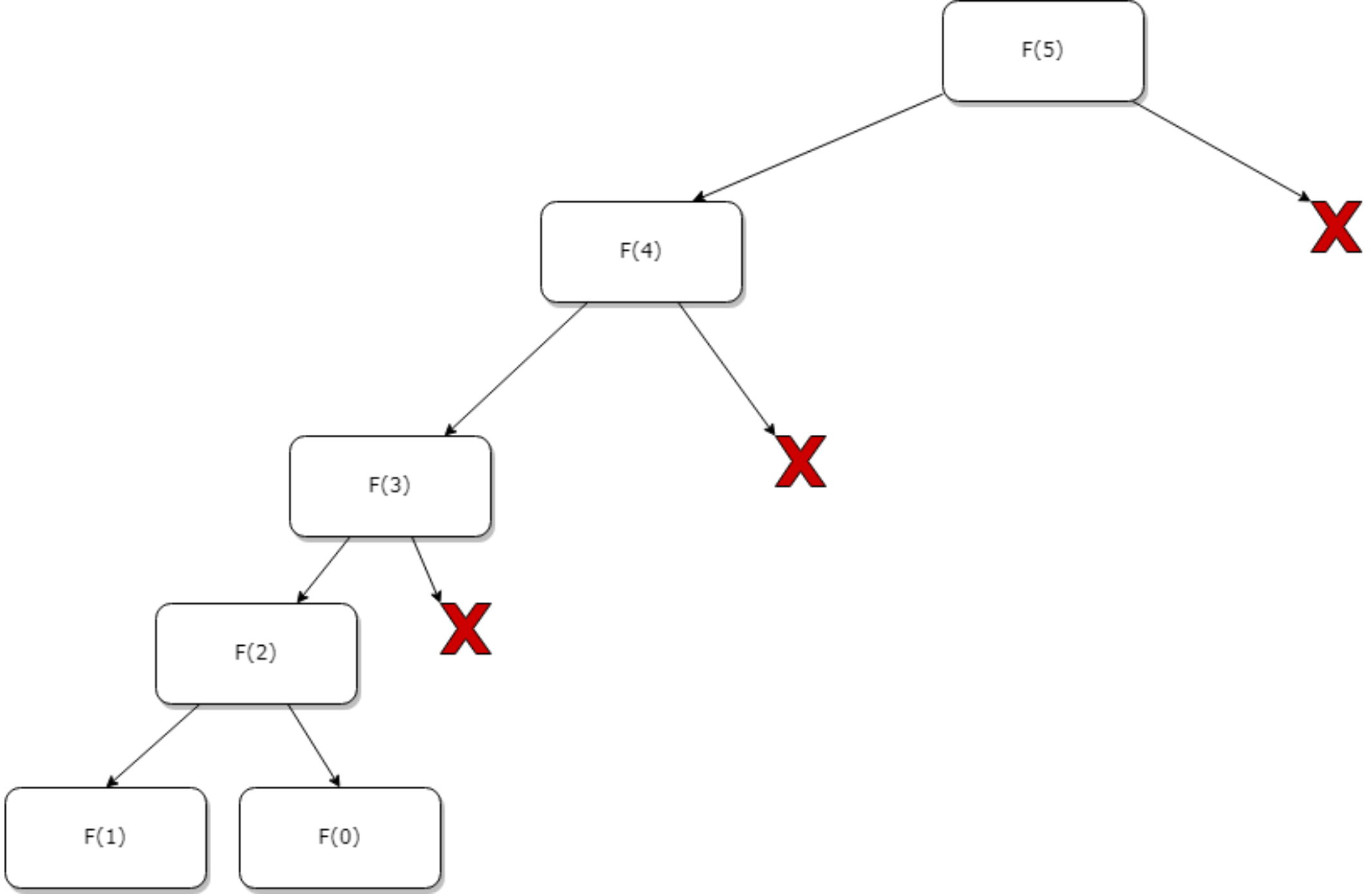
```
#include <stdio.h>
int fib(int n){
    int f[n];
    f[0] = 0;
    f[1] = 1;
    for(int i = 2; i ≤ n; i++) {
        f[i] = f[i – 1] + f[i – 2];
    }
    return f[n];
}
int main(){
    printf("%d", fib(8));
}
```

Using Recursion [Top - Down]:

```
#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);
}
```

Using Recursion[Top – Down]:




F(5)
F(4)
F(3)
F(2)
F(1)
F(0)

3.C program to Calculate Sum of all Natural Number

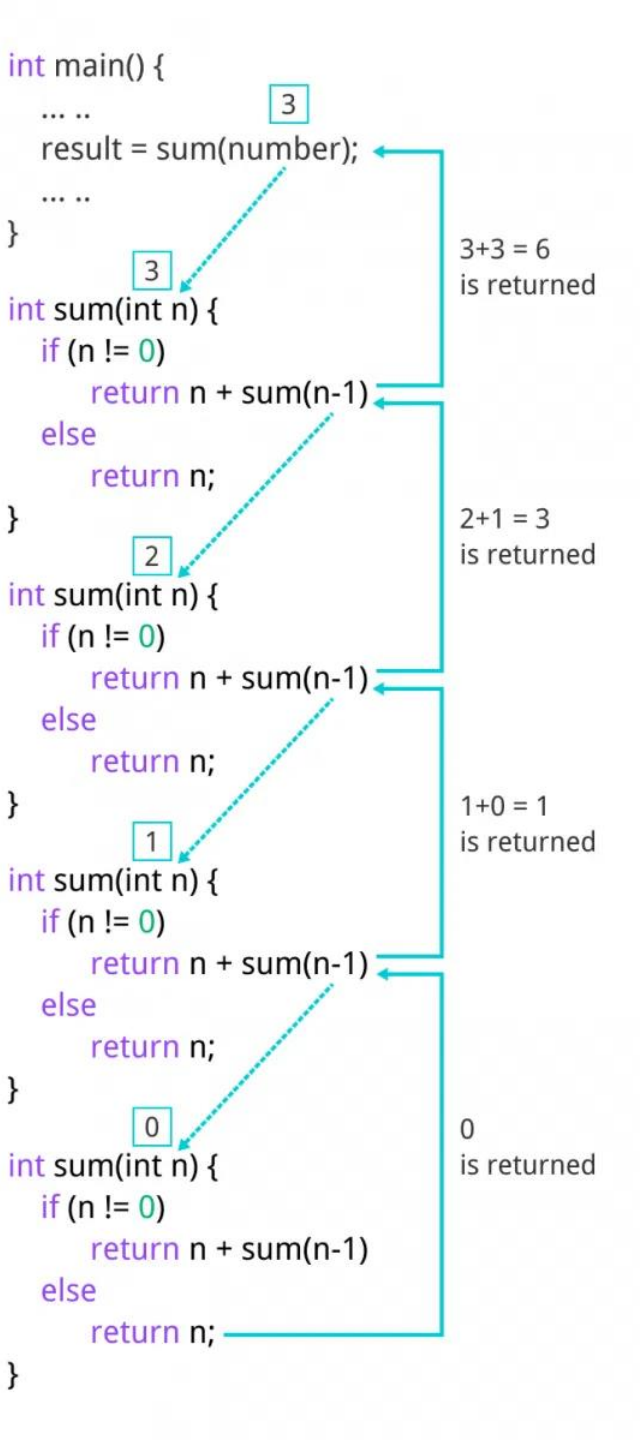
Input the last number of the range starting from 1 : 5
Expected Output: 1 + 2 + 3 + 4 + 5 = 15
The sum of numbers from 1 to 5 : 15

Using Simple Loop:

```
#include <stdio.h>
int main()
{
   int i, sum = 0, n = 10;
   for (i = 0; i ≤ n; i++){
      sum += i;
   }
   printf("Sum = %d", sum);
}
```

Using Recursion:

```
#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));
}
```



❖**Find the output of a pattern without using any loop**

❖ Find the output of a pattern without using any loop - 1

Example-1 : Given a number n, print the following pattern without using any loop.

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

```
#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);
}
```

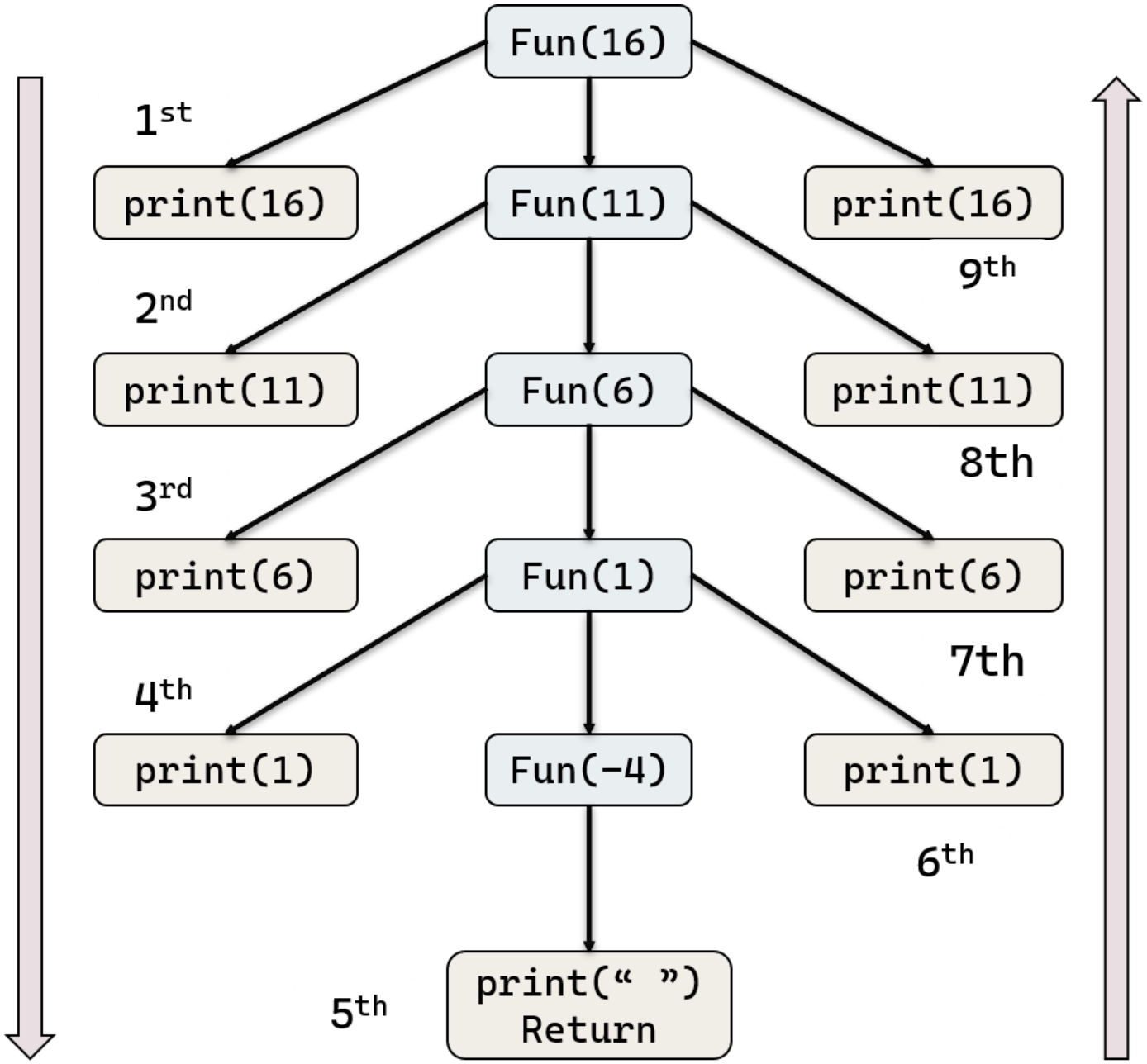
Output

```
16 11 6 1  1 6 11 16
```

```
#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 – 2: Calculate the output

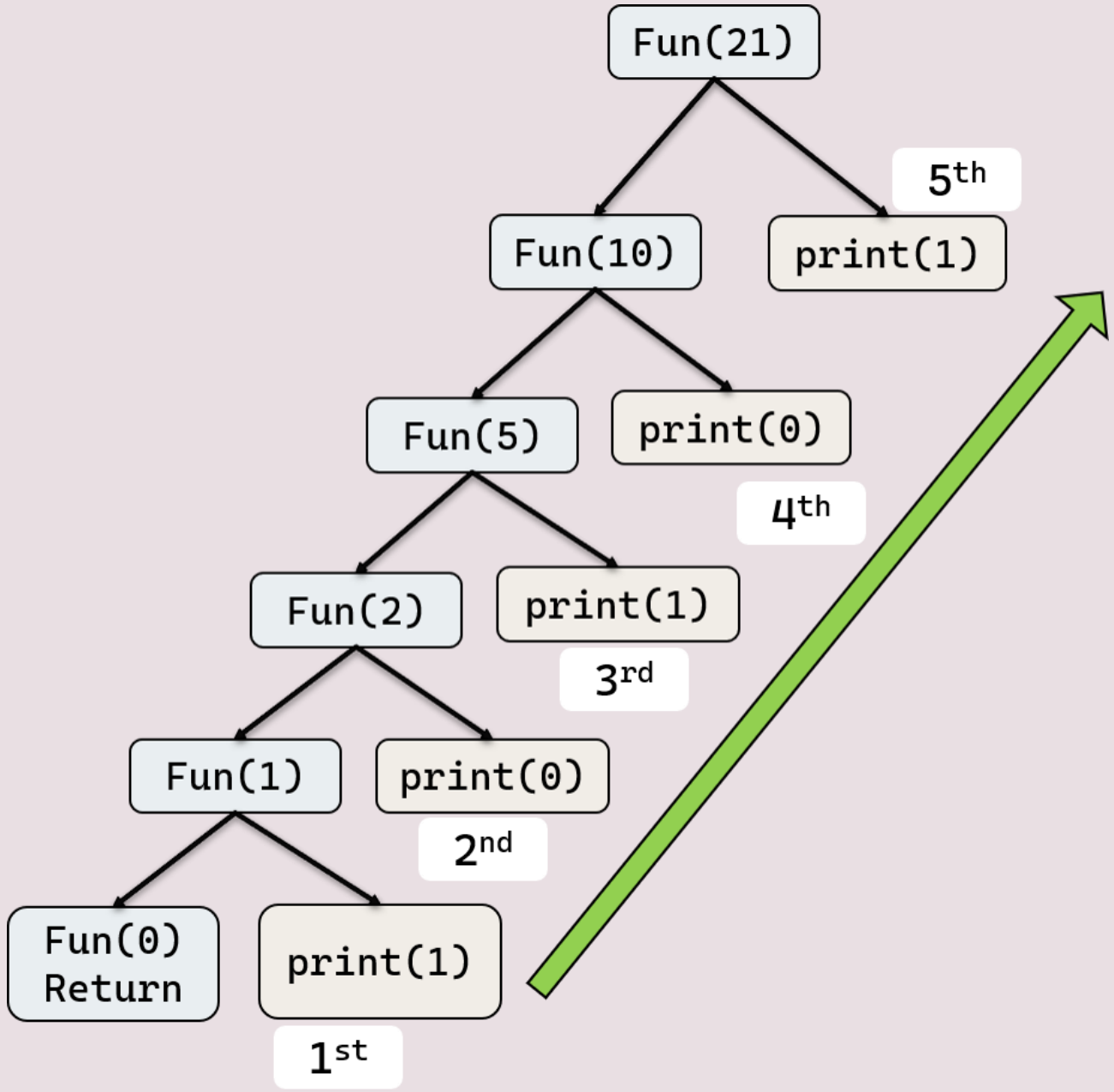
```
#include <stdio.h>
void printFun(int test)
{
   if (test < 1)
      return;
   else {
      printf("%d ",  test);
      printFun(test – 1);
      printf("%d ",  test);
   }
}
int main()
{
   int test = 3;
   printFun(test);
}
```

Assignment!!

Find the Recursion Tree.

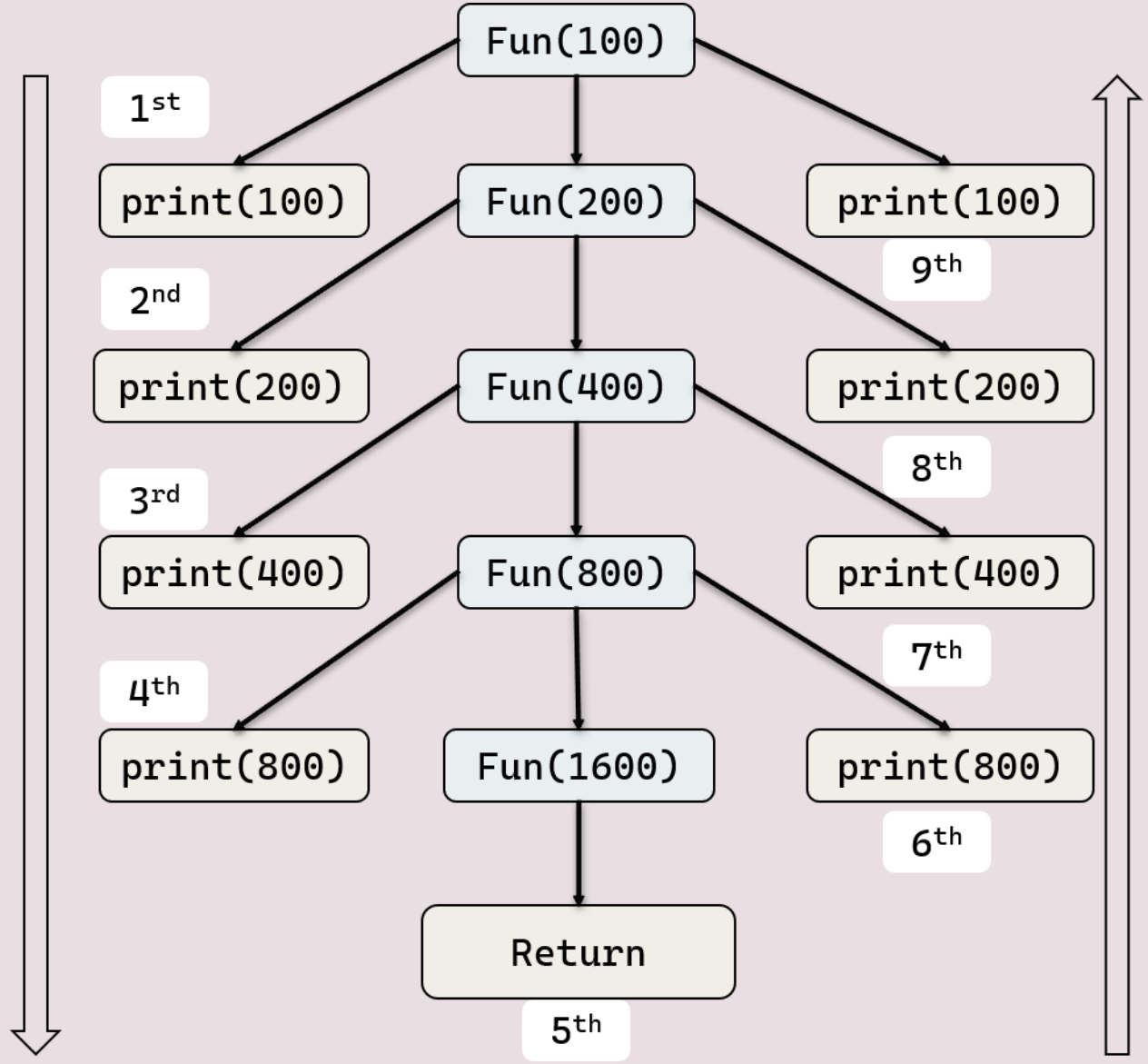
Example – 3: Predict the output pattern

```
#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.
```



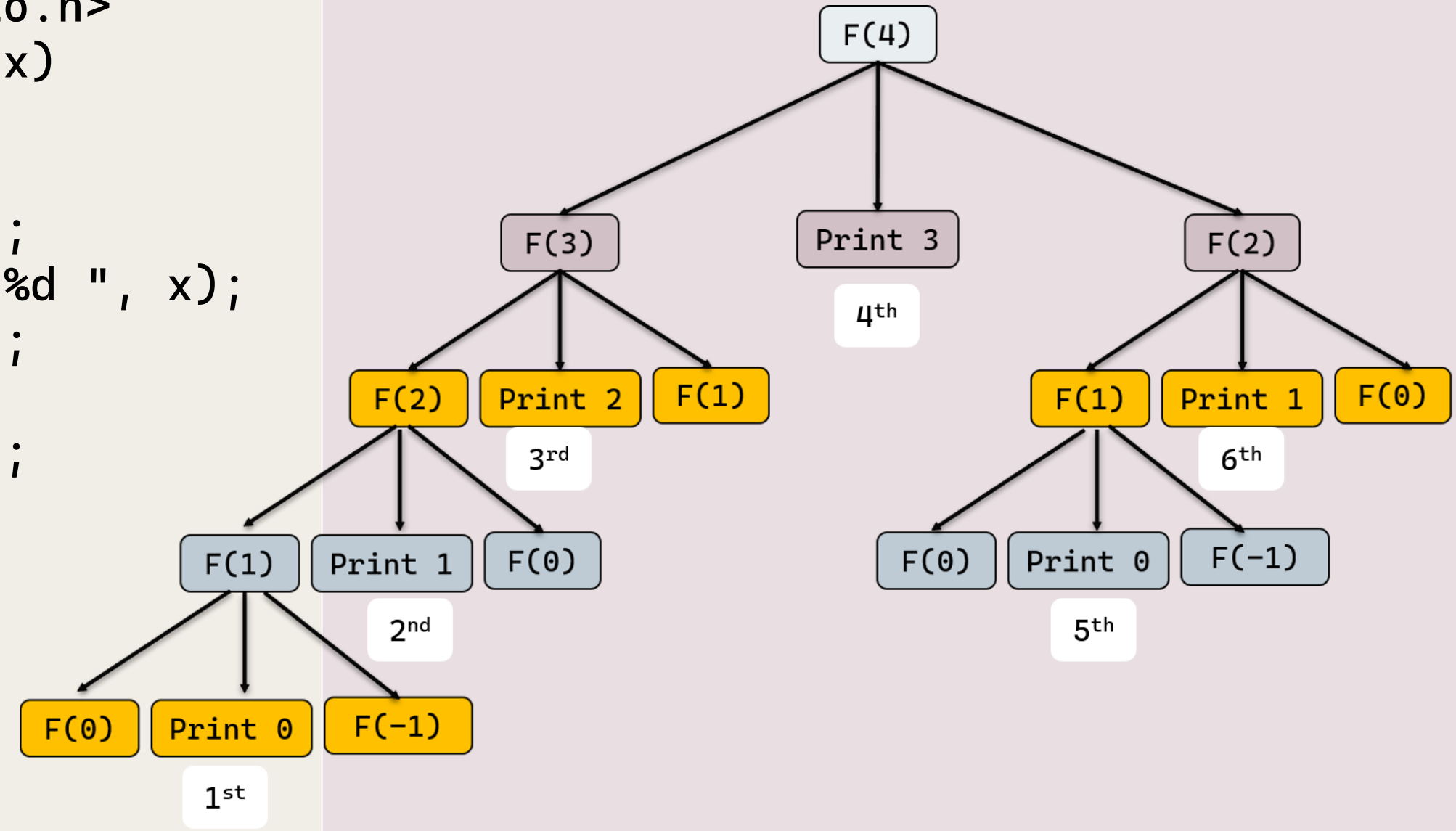
Example – 4: Predict the output pattern

```
#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
```



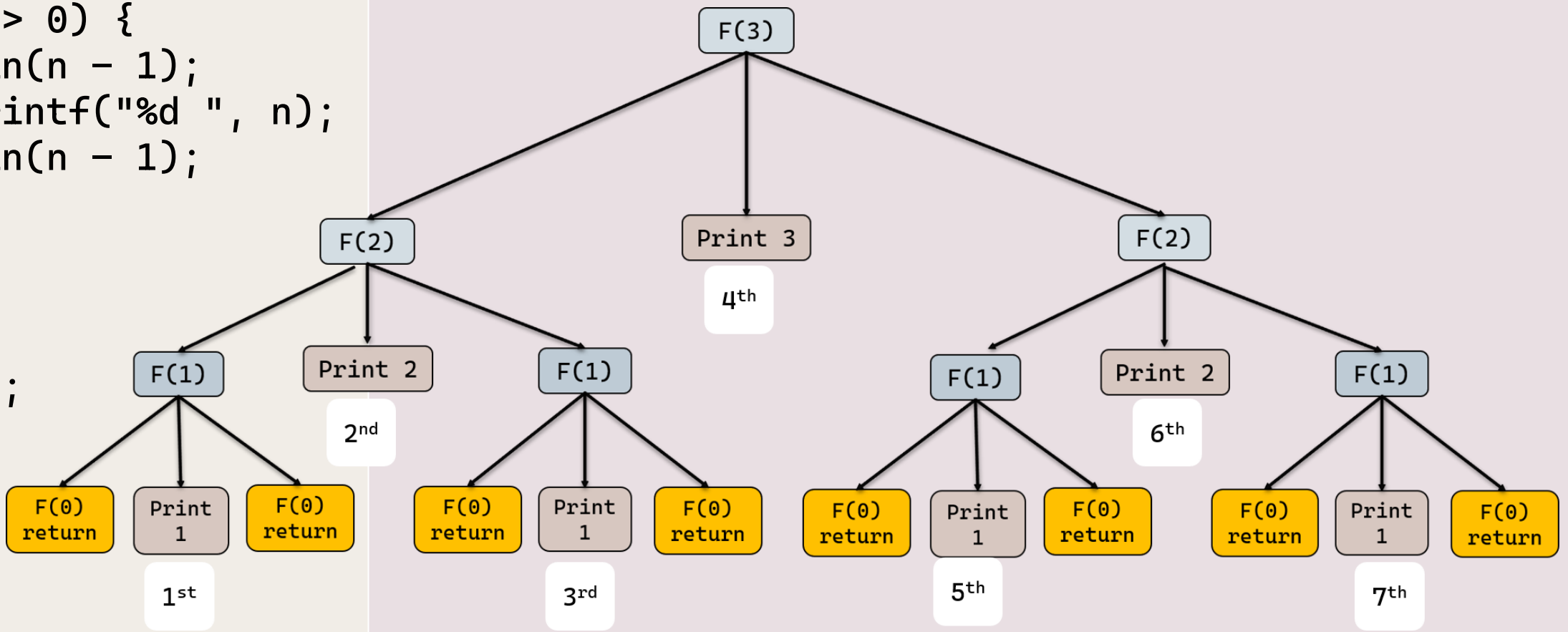
Example – 5: Predict the output pattern

```
#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);
}
```



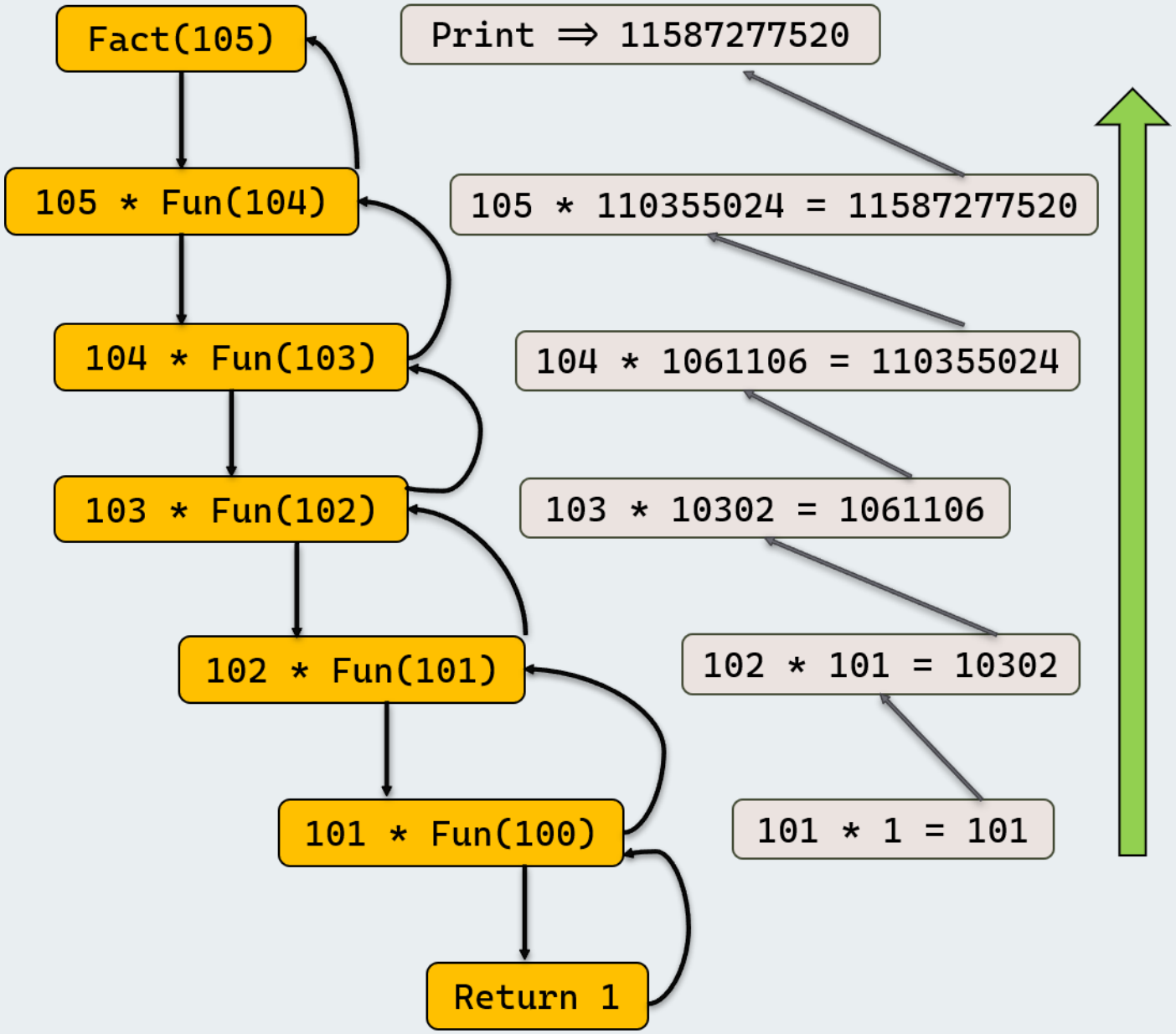
Output

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

- **Practise 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));
}
```



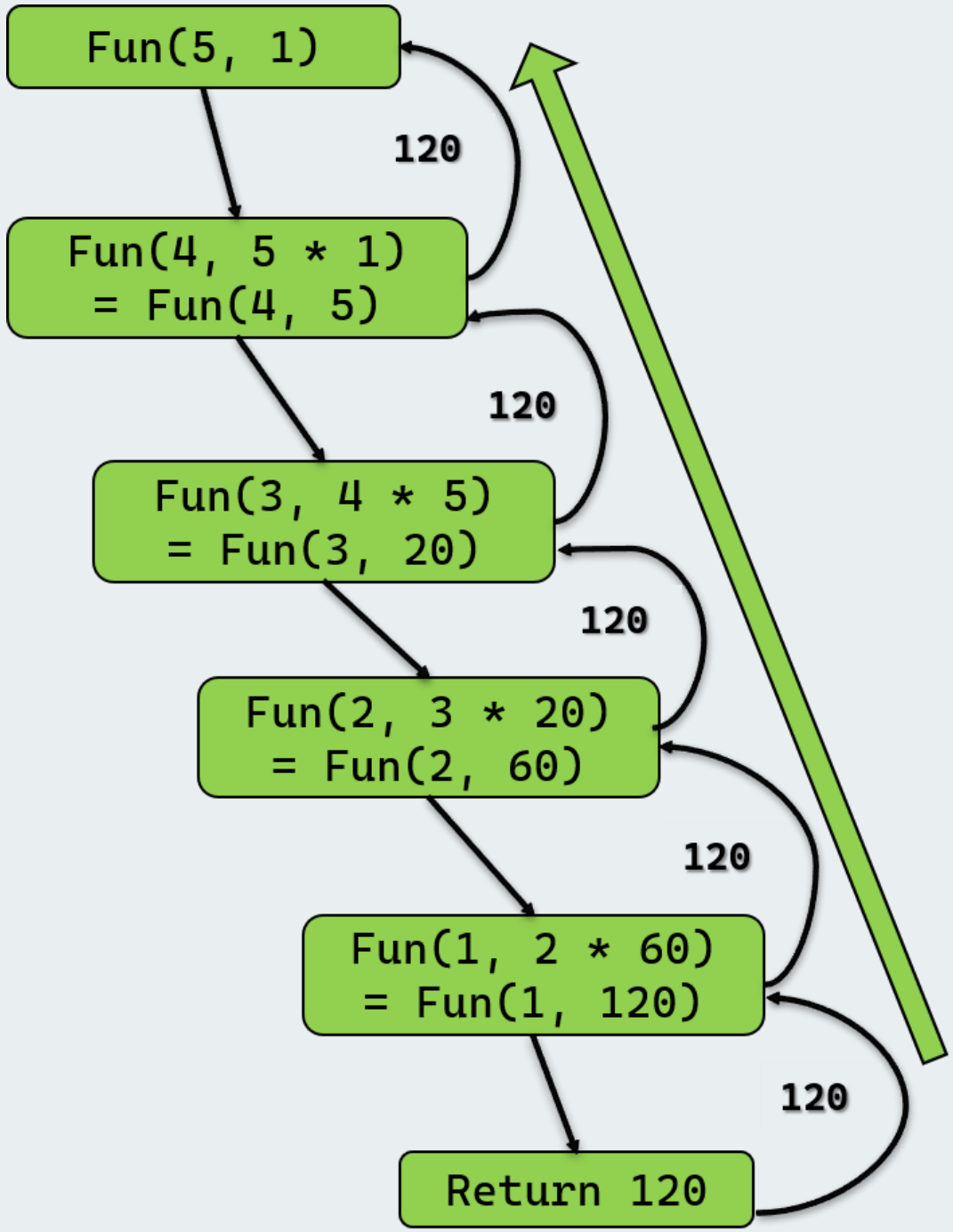
Example – 3: Calculate the output

```
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.
```

Assignment!!
Find the Recursion Tree.

Example – 2: Calculate the output

```
#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()
{
   cout << fact(5);
}
```



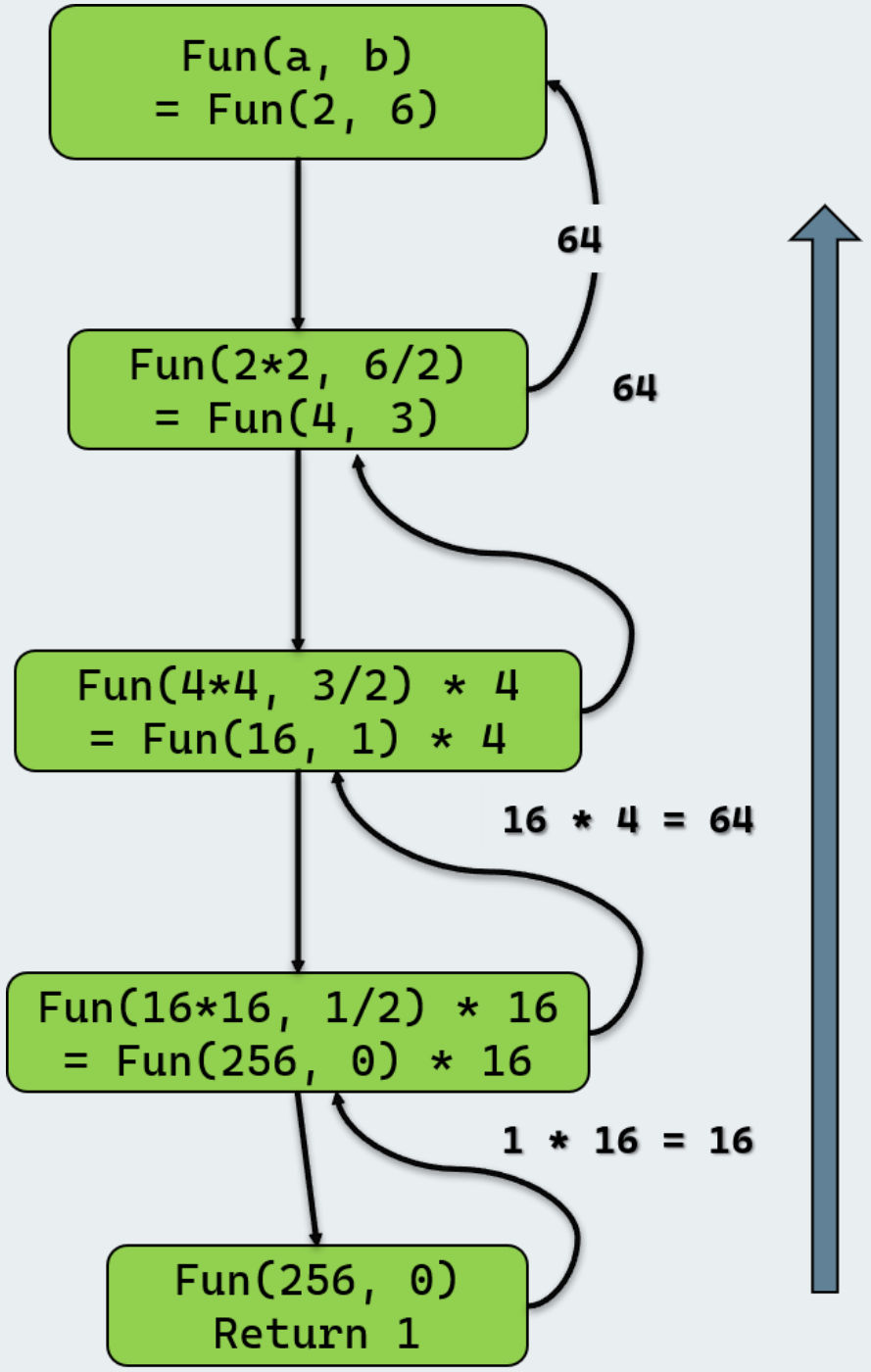
Example – 6: Calculate the output

```
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.
```

Assignment!!
Find the Recursion Tree.

Example – 10: Calculate the output

```
#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
```



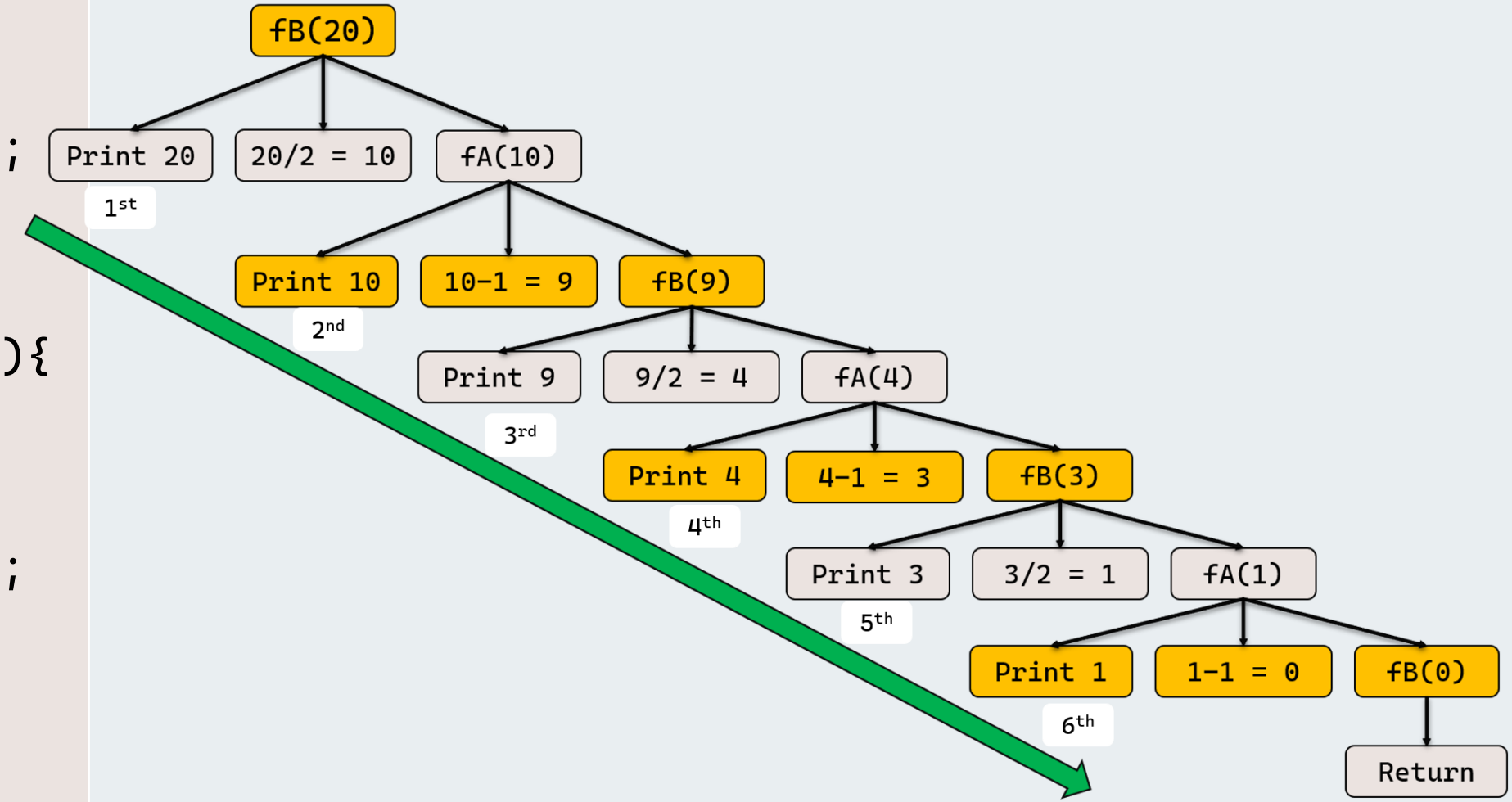
Example – 11: Calculate the output

```
#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));
  getchar();
  return 0;
}
Output:
12
```

Assignment!!
Find the Recursion Tree.

Example – 4: 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(){
    functionB(20);
}
```



Example – 4: Calculate the output

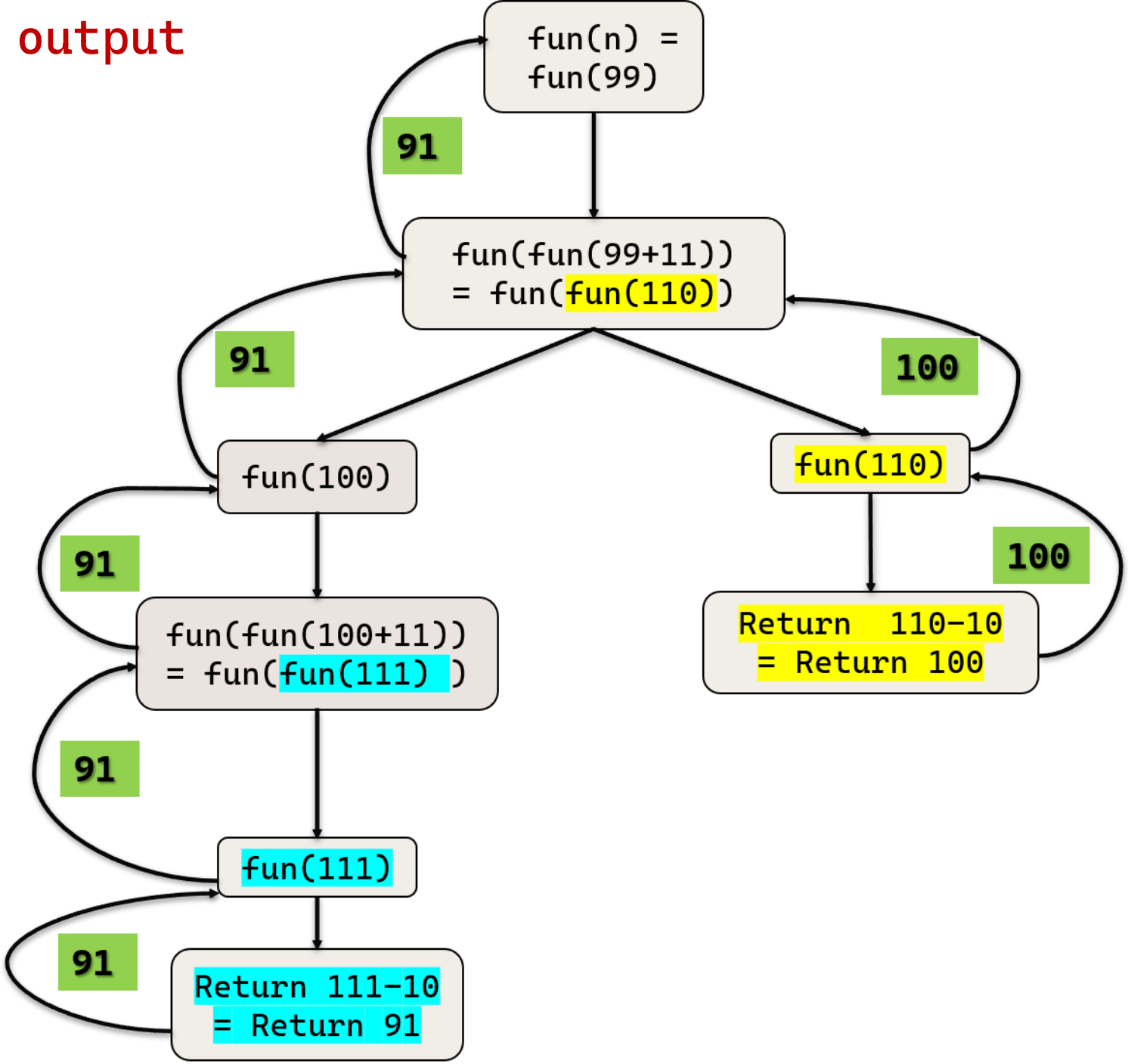
```
void functionB(int n);
void functionA(int n){
    if (n < 1) {
        return;
    }
    printf("%d ", n);
    n = n - 1;
    functionB(n);
}
void functionB(int n){
    if (n < 2) {
        return;
    }
    printf("%d ", n);
    n = n / 2;
    functionA(n);
}
int main(){
    functionA(20);
}
```

Assignment!!

Find the Recursion Tree.

Example – 9: Calculate the output

```
#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
```



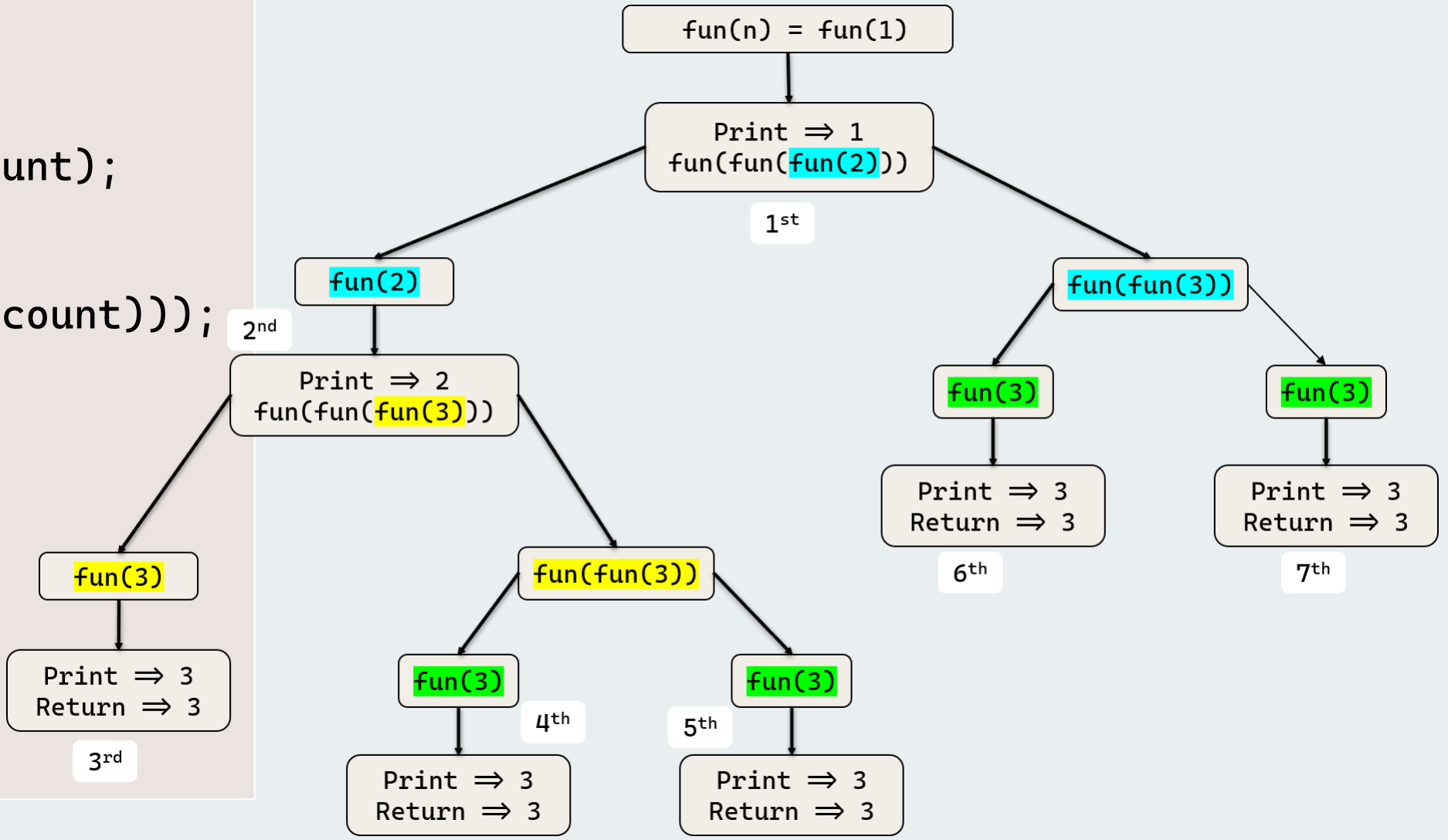
Example – 12: Calculate the output

Guess the output of the following code.

```
#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



❖Printing Pyramid Patterns using Recursion

❖ Printing Pyramid Patterns using Recursion – 1

```
void fun1(int n)
{
   int i = 0;
   if (n > 1)
     fun1(n - 1);
   for (i = 0; i < n; i++)
     printf(“ * ”);
}
Answer:
Total numbers of stars printed is equal to :-
1 + 2 + …. (n-2) + (n-1) + n,
which is n(n+1)/2.
```

Output:

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

Assignment!!

Find the Recursion Tree.

❖ Printing Pyramid Patterns using Recursion – 2

```
#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);
}
```

Assignment!!
Find the Recursion Tree.

Output:

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


THE
END