RECURSION

What is Recursion?

- A function is said to be RECURSIVE if it calls itself.

Example:

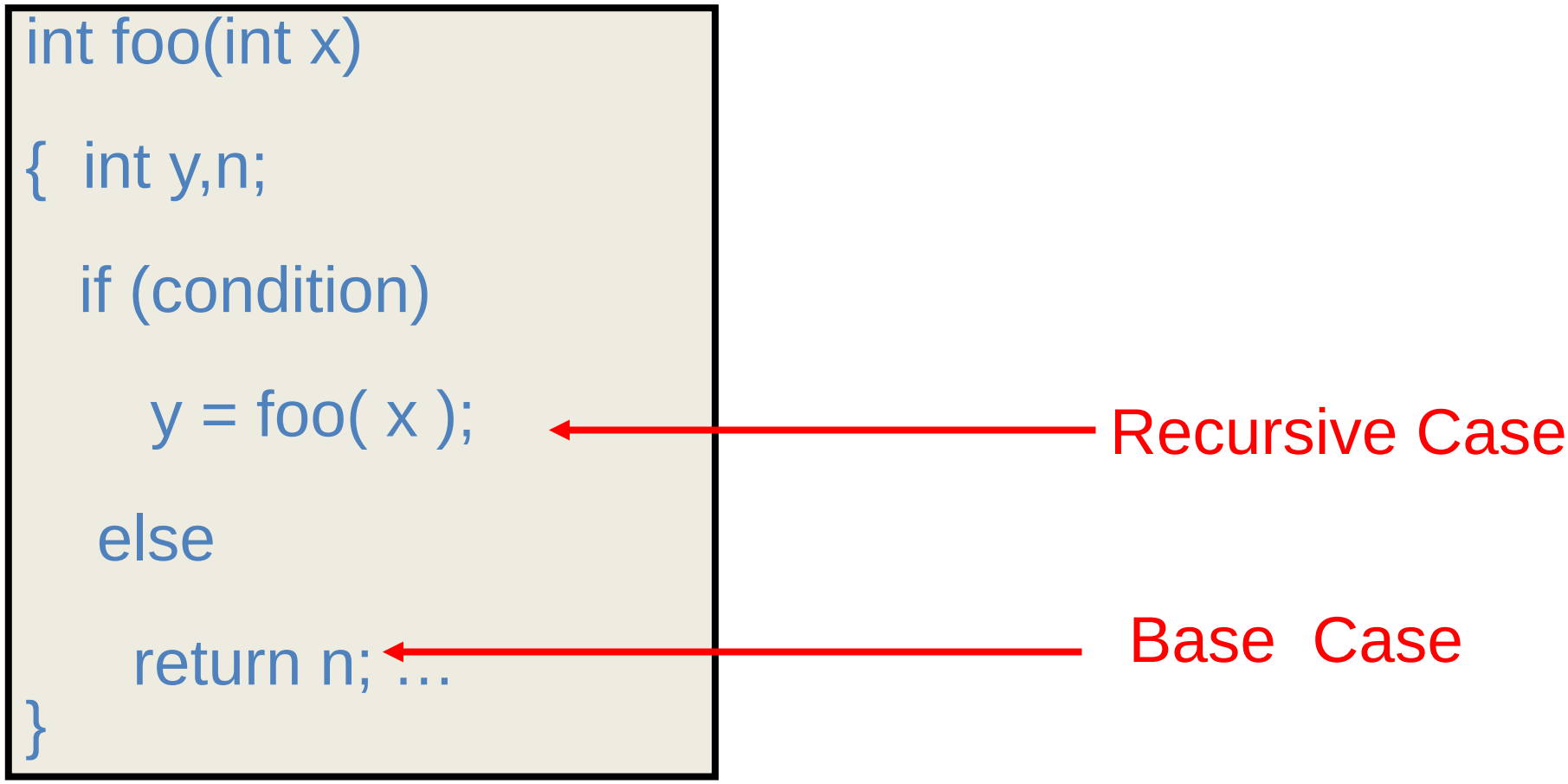
```
        void function_1(int x)
{   int y;
    ……….
        if ( condition )
            function_1(y);
}
```

Function calls it self repeatedly until some specified condition is satisfied.

Two cases

- A recursive definition has two parts:
  - One or more recursive cases where the function calls itself
  - One or more base cases that returns a result without a recursive call
- There must be at least one base case
- Every recursive case must make progress towards a base case

Recursive and Base case



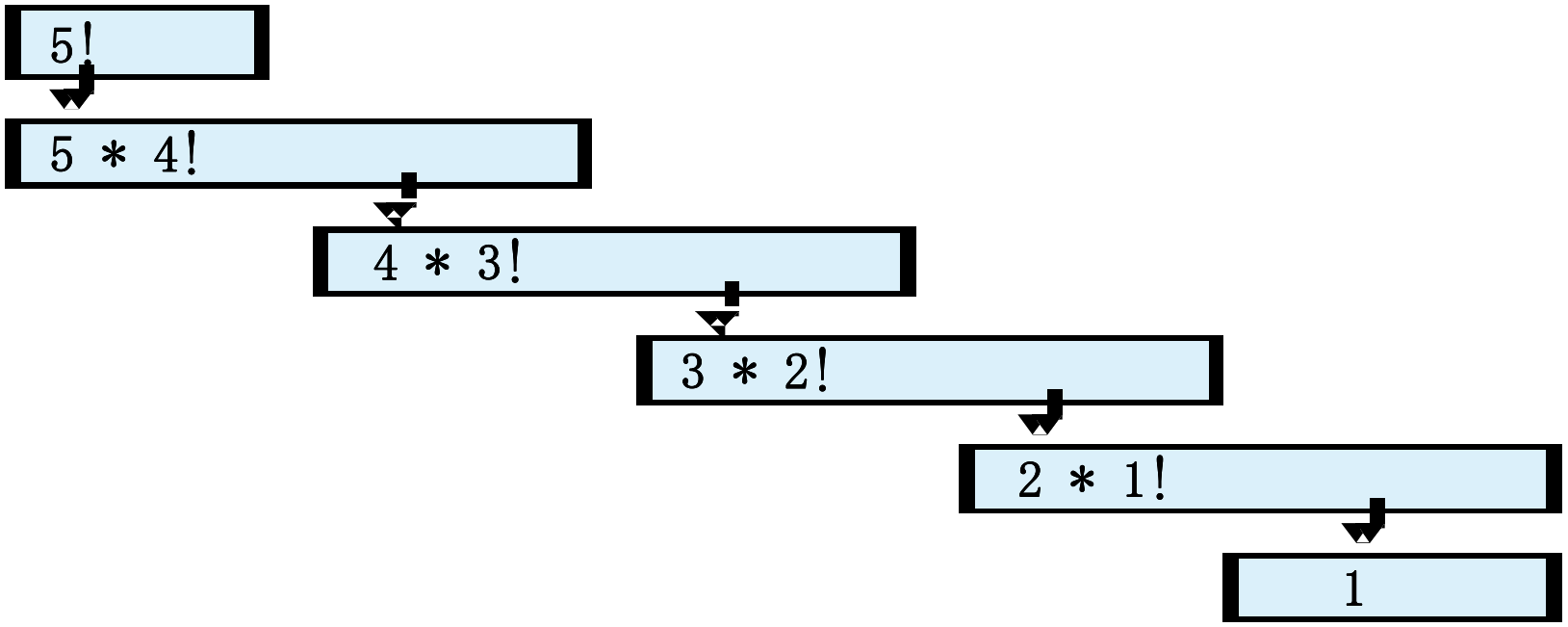
Example 1

Factorial of a number n! is defined mathematically as :

$$n! = \begin{cases} 1 & n = 0 \\ n * (n - 1)! & \text{otherwise} \end{cases}$$

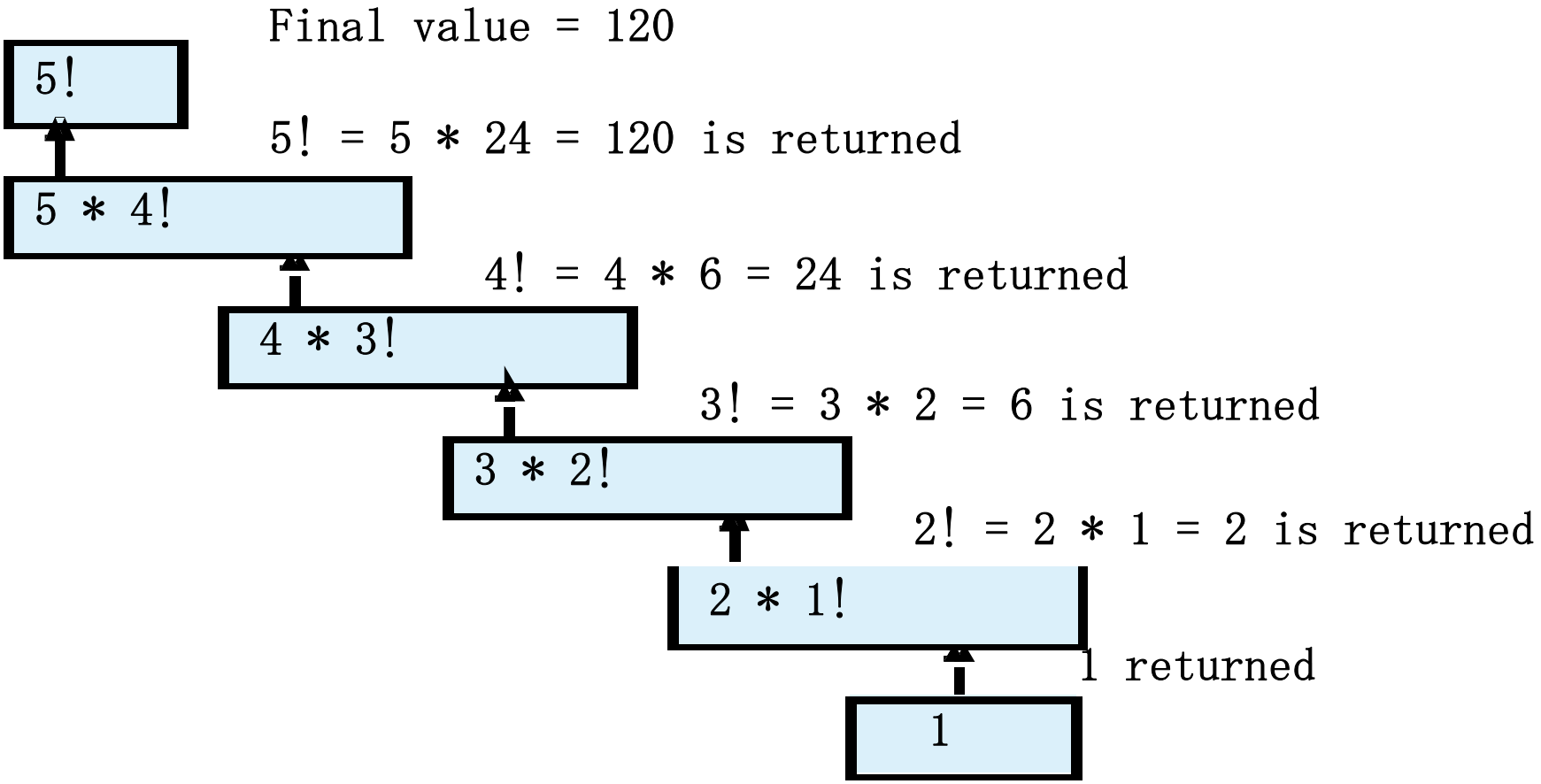
For calculating n! first (n-1)! should be calculated and for that (n-2)! should be computed and so on …

- Factorial of 5 (5 !) ->



Sequence of recursive calls.

Values returned from each recursive call.



Iterative version of factorial program

```
int n, m;
int fact=1;
printf(“Enter the number:”);
scanf(“ %d”,&n);
for( m = n ; m>=1; m- -)
    fact= fact x m;
printf(“The factorial is %d”, fact);
```

Recursive version of factorial program

```
int factorial(int n)
{
   int result;

   if (n == 0||n==1))            /* if (n <= 1)*/
         return 1;
   else     result = n * factorial(n - 1);
   return result;
}
```

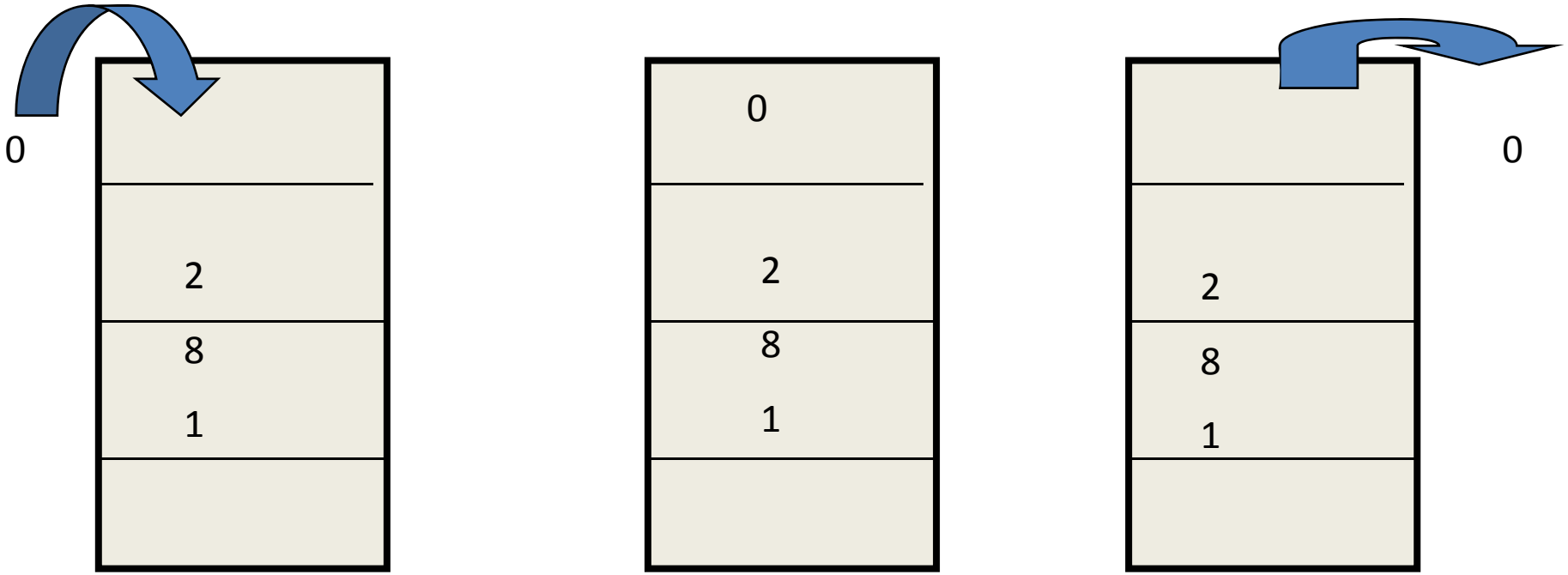
LOGIC

- When a recursive function is executed, the recursive calls are not executed immediately. Rather, they are placed on a stack until the condition that terminates the recursion is encountered.

- The function calls are then executed in reverse order, as they are popped off the stack.

STACK

- A STACK is a LAST-IN, FIRST-OUT data structure in which successive data items are pushed down upon the preceding data items. The data items are removed (popped off) from the stack in REVERSE order.



Factorial (animation 1)

- x = factorial(3)

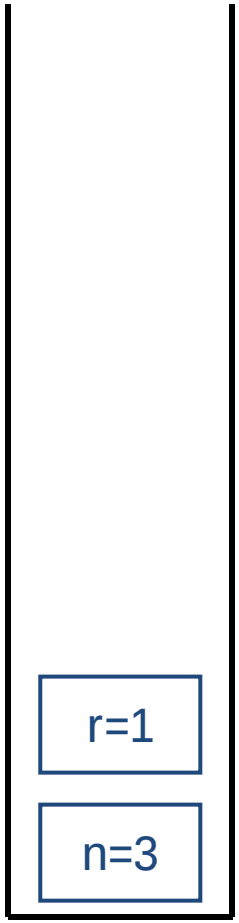
3 is put on stack as n

- int factorial(int n) { //n=3
  int r = 1; r is put on stack with value 1
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

All references to r use this r

All references to n use this n

Now we recur with 2...



Factorial (animation 2)

- r = n * factorial(n - 1);

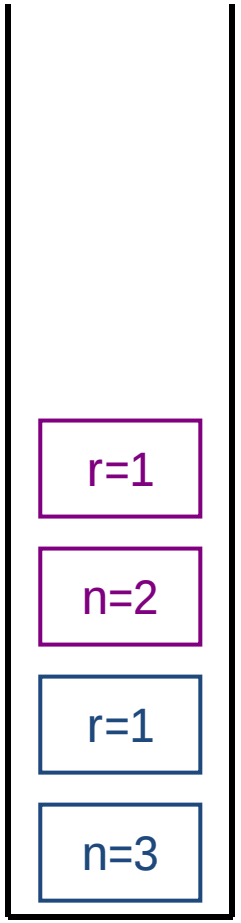
2 is put on stack as n

- int factorial(int n) {//n=2
  int r = 1; r is put on stack with value 1
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

Now using this r

And this n

Now we recur with 1...



Factorial (animation 3)

- r = n * factorial(n - 1);

1 is put on stack as n

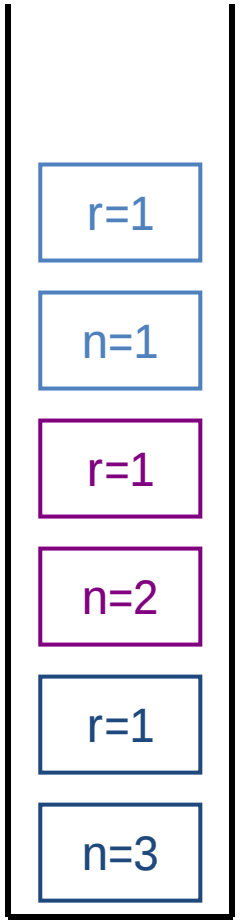
- int factorial(int n) { // n=1
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

r is put on stack with value 1

Now using this r

And this n

Now we pop r and n off the stack and return 1 as factorial(1)



Factorial (animation 4)

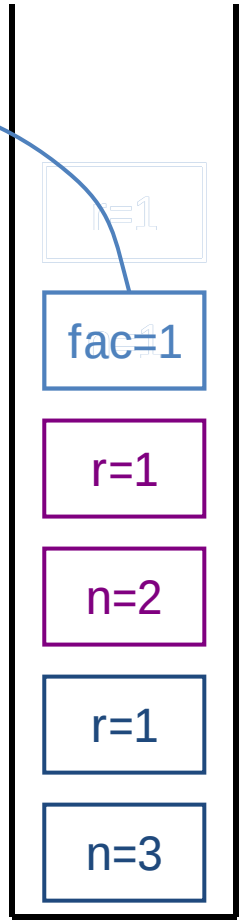
- r = n * factorial(n - 1);

- int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

Now using this r

And this n

Now we pop r and n off the stack and return 1 as factorial(1)



Factorial (animation 5)

- r = n * factorial(n - 1);

- ```
  int factorial(int n) {
      int r = 1;
      if (n <= 1) return r;
      else {
          r = n * factorial(n - 1);
          return r;
      }
  }
  ```

Now using this r

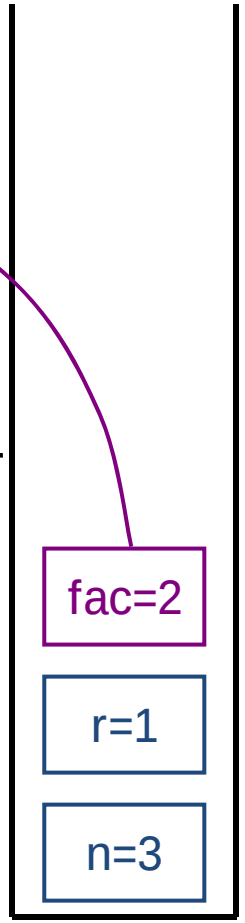
And this n

1

2 * 1 is 2;

Pop r and n;

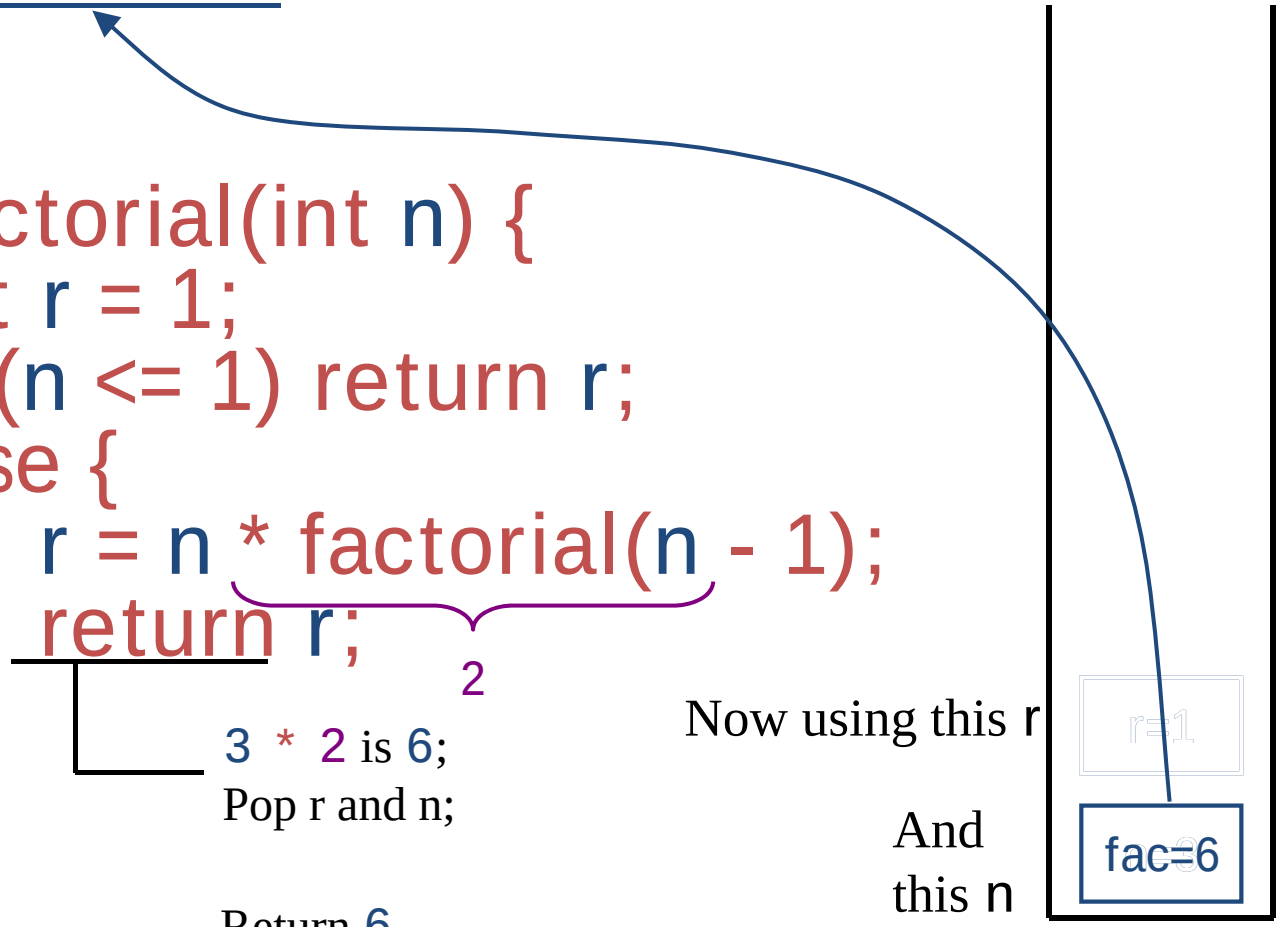
Return 2


  ```

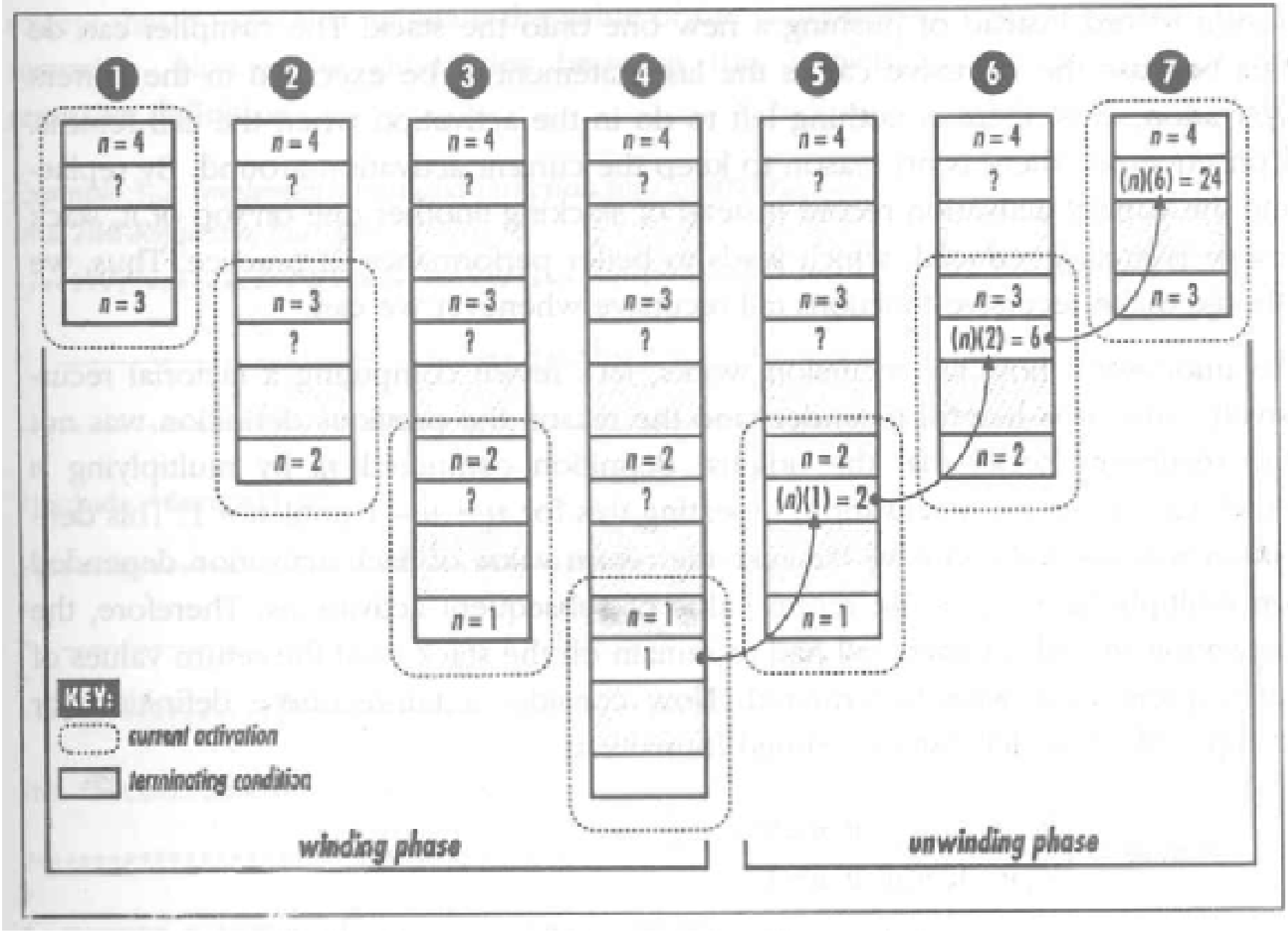
Factorial (animation 6)

- x = factorial(3)

- int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }



The stack of C program while computing 4! recursively



Example 2

- Fibonacci series: 0, 1, 1, 2, 3, 5, 8...
  - **Each term is the sum of the previous two terms.**
- Recursive definition:
  - `F(0) = 0;`
  - `F(1) = 1;`
  - `F( n ) = F( n - 1 ) + F( n – 2 );`

if n = 6 , 6 th term

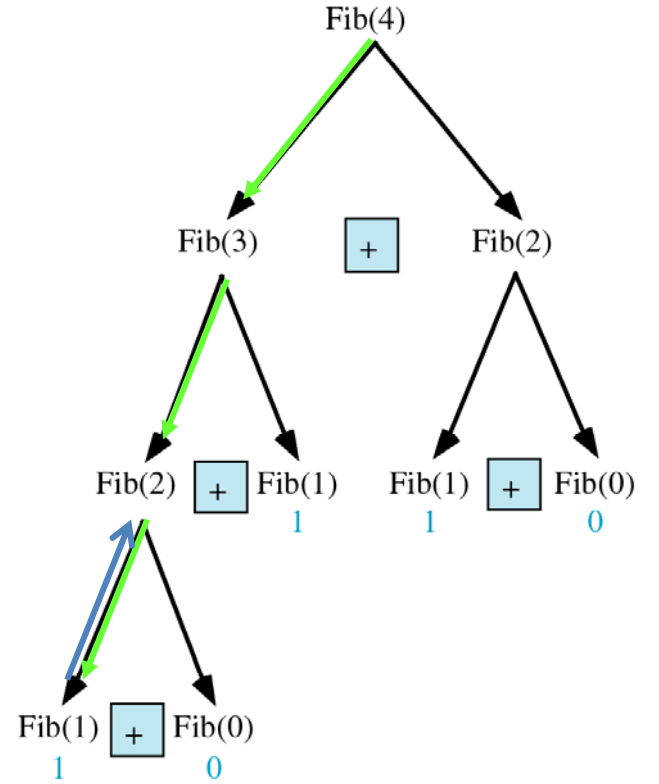
fib(6) = fib (5) + fib (4)

Trace a Fibonacci Number

- Assume the input number is 4, that is, num=4:

```
fib(4):
    4 == 0 ? No;   4 == 1? No.
    fib(4) = fib(3) + fib(2)
    fib(3):
        3 == 0 ? No; 3 == 1? No.
        fib(3) = fib(2) + fib(1)
        fib(2):
            2 == 0? No; 2==1? No.
            fib(2) = fib(1)+fib(0)
            fib(1):
                1== 0 ? No; 1 == 1? Yes.
                fib(1) = 1;
                return fib(1);
```

```
int fib(int num)
{
    if (num == 0) return 0;
    if (num == 1) return 1;
    return
      (fib(num-1)+fib(num-2));
}
```



Trace a Fibonacci Number

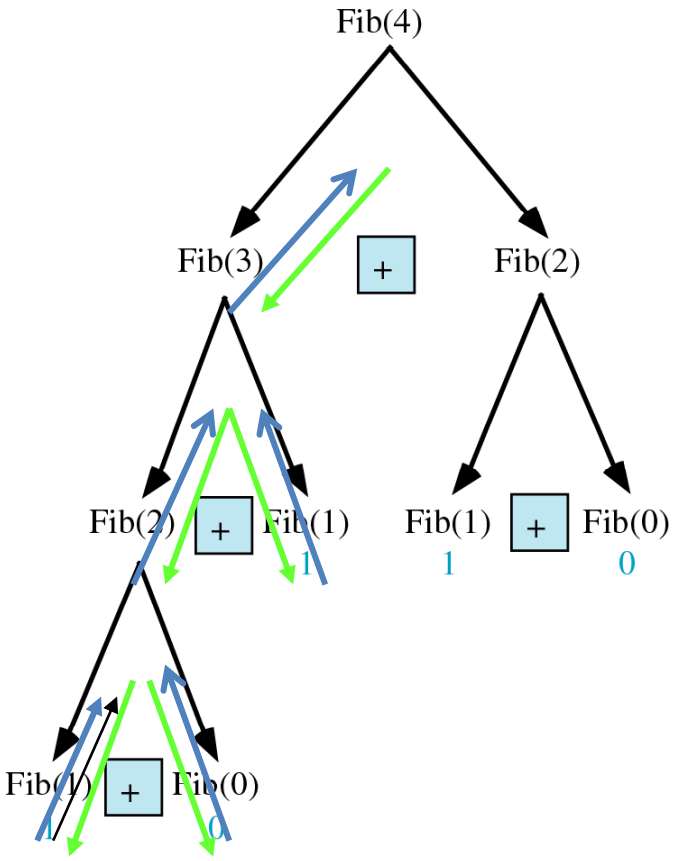
```
fib(0):
0 == 0 ?  Yes.
fib(0) = 0;
return fib(0);
```

```
fib(2) = 1 + 0 = 1;
return fib(2);
```

```
fib(3) = 1 + fib(1)
```

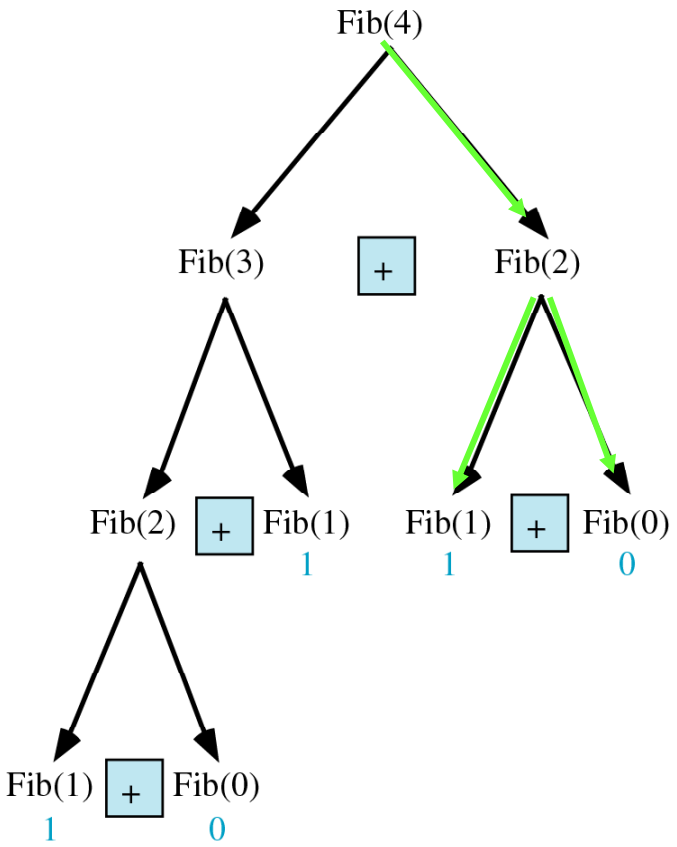
```
  fib(1):
 1 == 0 ? No; 1 == 1? Yes
 fib(1) = 1;
return fib(1);
```

```
fib(3) = 1 + 1 = 2;
 return fib(3)
```



Trace a Fibonacci Number

```
fib(2):
    2 == 0 ? No; 2 == 1?    No.
    fib(2) = fib(1) + fib(0)
    fib(1):
       1== 0 ? No; 1 == 1?  Yes.
        fib(1) = 1;
        return fib(1);
    fib(0):
        0 == 0 ?   Yes.
        fib(0) = 0;
        return fib(0);
    fib(2) = 1 + 0 = 1;
    return fib(2);
fib(4) = fib(3) + fib(2)
       = 2 + 1 = 3;
return fib(4);
```



Recursion General Form

- How to write recursively?

```
int recur_fn(parameters){
  if(stopping condition)
     return stopping value;
  // other stopping conditions if needed
  return function of recur_fn(revised parameters)

}
```

Exercise

- **C Program to calculate sum of numbers 1 to N using recursion**

```
int calculateSum(int num);
void main() {
  int i, num;
  int result;
  printf("Input a number : ");
  scanf("%d", &num);
  result = calculateSum(num);
  printf("\nSum of Number From 1 to %d : %d", num, result);
}
int calculateSum(int num) {
  int res;
  if (num == 1) {
    return (1);
  } else {
    res = num + calculateSum(num - 1);
  }
  return (res); }
```

Exercise 2

- **Find Sum of Digits of the Number using Recursive Function**

```
#include<stdio.h>
 int calsum(int num) {
   int rem, sum;
   if (num != 0) {
      rem = num % 10;
      sum = sum + rem;
      calsum(num / 10);
   }
   return sum;
}

int main() {
   int num, val;
   printf("\nEnter a number: ");
   scanf("%d", &num);

   val = calsum(num);
   printf("\nSum of the digits of %d is : %d", num, val);

   return 0;
}
```

Exercise

- **WAP to compute GCD of Two Numbers using Recursion**

```
#include <stdio.h>
int hcf(int n1, int n2);
int main()
{
  int n1, n2;
  printf("Enter two positive integers: ");
  scanf("%d %d", &n1, &n2);

  printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1,n2));
  return 0;
}

int hcf(int n1, int n2)
{
   if (n2 != 0)
     return hcf(n2, n1%n2);
   else
     return n1;
}
```

Homework

Write a recursive function for the Ackermann's Function

$$A(x,y) \equiv \begin{cases} y+1 & \text{if } x = 0 \\ A(x-1,1) & \text{if } y = 0 \\ A(x-1,A(x,y-1)) & \text{otherwise.} \end{cases}$$

Solution

```
int ackermann(int m, int n)
{
    if (m == 0) {
            return n + 1; }
    else if ((m > 0) && (n == 0)) {
            return ackermann(m-1, n); }
    else if ((m > 0) && (n > 0)) {
            return ackermann(m-1, ackermann(m,n-1)); }
    else {
            return 0; }
}
```