Recursive Functions:

A problem solving technique that reduces large problems to smaller problems of the same form
It calls itself.

General Structure of a Recursive Func.

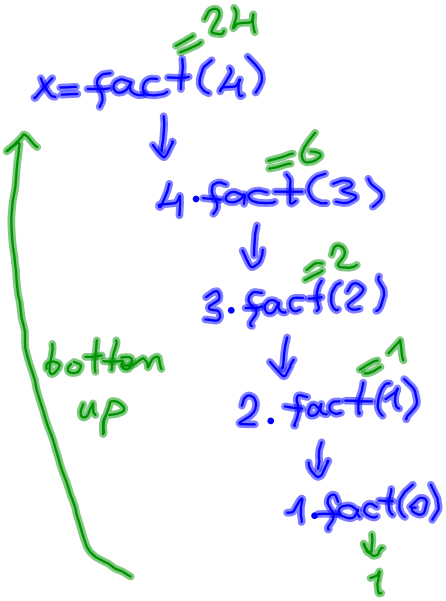
```
if ( base (exit) case condition)
    Calculate base case without recursion
else // recursive case
    break the problem into smaller problem
    solve smaller problems recursively
```

Ex: Factorial Problem

$5! = 5 \cdot 4!$
$4! = 4 \cdot 3!$
$3! = 3 \cdot 2!$
$2! = 2 \cdot 1!$
$1! = 1 \cdot 0!$
$0! = 1$

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n \cdot (n-1)! & \text{if } n>0 \end{cases}$$

```
int facto(int n){
    if (n==0)
        return 1;
    else
        return n*facto(n-1);
}
```

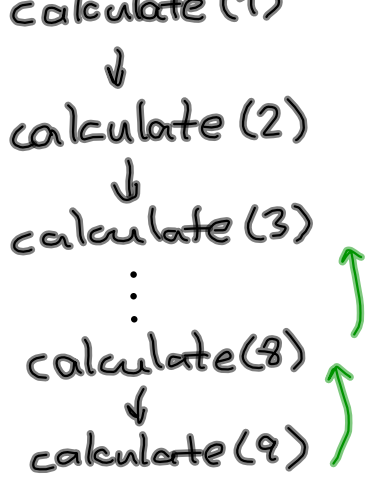


Question:

```
int calculate(int x){
    printf("%d", x);
    if (x<9)
        calculate(x+1);
    printf("%d", x);
}
```

What is the output for calculate(1): 123...89987...1

output: 1 2 3 ... 8998 ... 1



Structures in C: A structure is a collection of variables under a single name.

Let x be a complex number $x = a + ib$ (a: real, b: imaginary)

```
struct complex{
    int real;
    int im;
};
```

Creating objects

```
struct complex s1,s2;
struct complex *s3;
s3=&s2;
```

Accessing members of structures

- dot
- → arrow

```
s1.real=6;
s1.im =7;
s3->real=8;
```

Alternative Struct declaration

```
typedef struct{
    int id;
    char *name;
    float mot;
} student;
```

```
student s1,s2;
```

Ex: Write a function to add two complex numbers

```
int sum(int a,int b){
    return a+b;
}
```

```
struct complex topla(struct complex a, struct complex b)
{
    struct complex toplam;
    toplam.real= a.real+b.real;
    toplam.im = a.im+b.im;
    return toplam;
}
```