

Recursion in Computer Programming

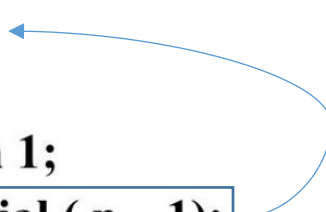
When a function that **calls itself** it is known as a *recursive function*.

This *process* is known as **recursion**.

The principle of **recursion** comes from mathematics.

(Sometimes a mathematical function can be defined as a function of itself).

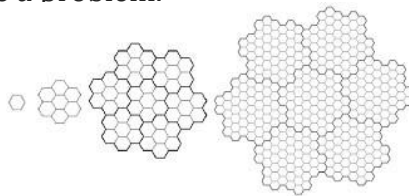
```
int factorial (int n)
{
    if (n == 0) return 1;
    return n * factorial ( n - 1);
}
```



Key points:

- In Programming **Recursion** is a way to create a **loop** without a `while`, or `for`, or other looping statements.
- When you see a function that call's itself this means you have a **loop**

Using recursion can be the quickest and simplest way to solve a problem and is a common method used in programming for repeating steps. **You must be able to recognize and understand recursion in code (it is common)**; However, it is important to realize that recursion can be replaced with the other (looping) techniques you have learned. There is always several ways to solve a problem.



Recursive patterns: the product is made from repeating itself

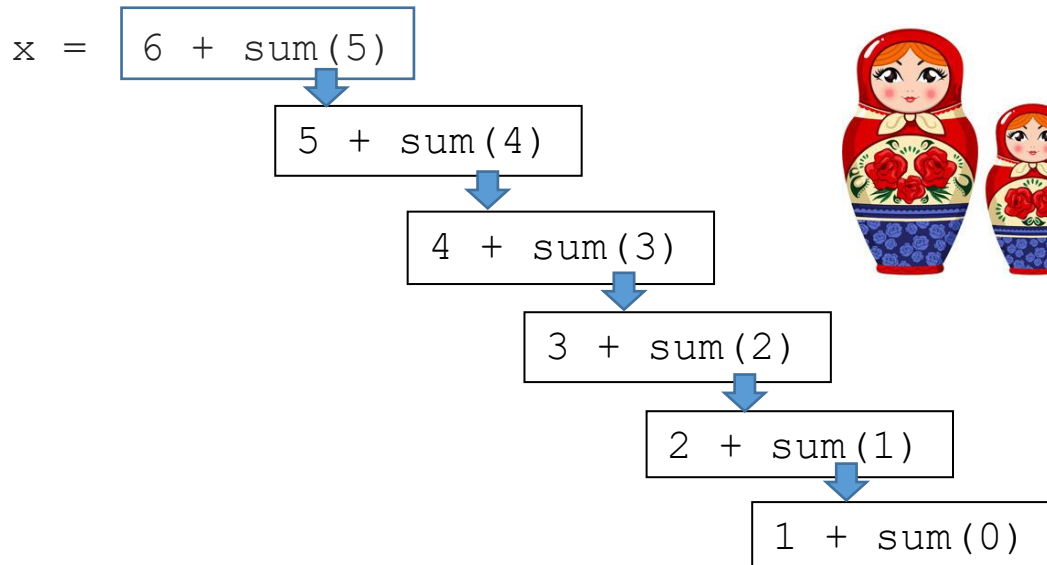
Recursion is a loop!



```
int sum(int x) //Function called "sum" is defined
{
  if (x!=0)
    return x + sum(x-1); //sum() function calls itself!
  else
    return x;
}
```

This line repeats again and again because "sum" calls itself...so every time it is called,...it will be called again and again and again! if (x!=0)

if we input $x = 6$, the result would be:



this will result in $6+5+4+3+2+1 = 21$

Exercise 9.1 (type it in and read the accompanying notes)

The following code will take a positive integer from the user and then find the sum of all positive integers from that number to zero.

Sample input: 4

Sample output: 10

```
#include <stdio.h>

int sum(int x)    //Function called sum is defined
{
    if (x!=0)
        return x + sum(x-1); //sum() function calls itself
    else
        return x;
}

int main()
{
    int number, result;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);

    printf("sum=%d", result);
}
```

This line repeats again and again because "sum" calls itself...so every time it is called,...it will be called again! and again and again! if (x!=0)

Look carefully!

- 1. The main program first takes input from the user (an integer value).**
- 2. then the main program calls the function `sum()`**
- 3. `sum()` will add the number input by the user to the result of `sum()` ... but `sum()` calls itself and therefore keeps adding additional results of `sum()` until `x = 0`;**
- 4. Then the main program and stuffs the result into a variable called `result` and prints it out.**

Exercise 9.2 (try and then check)

Write a programming in C that will do the factorial of a integer.

Reminder: a factorial is $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Example input: 4

Example output: 24

Sample solution:

```
#include <stdio.h>

//function for factorial

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

int main()
{
    int num;
    int fact=0;

    printf("Enter an integer number: ");
    scanf("%d",&num);

    fact=factorial(num);
    printf("Factorial of %d is = %d",num,fact);
    printf("\n");
    return 0;
}
```

Exercise 9.3 (try and then check)

Write a program that will find the value of one number to the power of another number.

Sample input:

Base? 4

Exponent? 2

Sample output:

4 to the power of 2 is 16

Sample solution

```
#include <stdio.h>

int power(int n1, int n2);

int main()
{
    int base, powerRaised, result;

    printf("Enter base number: ");
    scanf("%d",&base);

    printf("Enter power number(positive integer): ");
    scanf("%d",&powerRaised);

    result = power(base, powerRaised);

    printf("%d^%d = %d", base, powerRaised, result);
    return 0;
}

int power(int base, int powerRaised)
{
    if (powerRaised != 0)
        return (base*power(base, powerRaised-1));
    else
        return 1;
}
```

Exercise 9.4 (try and then check)

Write a program that will find the number of digits in an integer

Sample input:

Enter your number: 486584

Sample output:

Number of digits: 6

Before you try:

The standard algorithm for this problem is to:

1. divide the entered number by 10.
2. Then check to see if the result is less than zero.
3. If you keep track of the number of times you divide before getting zero,...then you have found the number of digits.

Example: **675**

$$675/10 = 67.5 \quad 67.1/10=6.75 \quad 6.71/10=0.671$$

Had to divide **3** times before result was less than zero.

Sample Solution

```
/*C program to count digits using recursion.*/

#include <stdio.h>

//function to count digits
int countDigits(int num)
{
    static int count=0;

    if(num>0)
    {
        count++;
        countDigits(num/10);
    }
    else
    {
        return count;
    }
}

int main()
{
    int number;
    int count=0;

    printf("Enter a positive integer number: ");
    scanf("%d",&number);

    count=countDigits(number);

    printf("Total digits in number %d is: %d\n",number,count);

    return 0;
}
```