ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Лекція 18.
Рекурсія

Що таке рекурсія ?

Рекурсія — не те саме, що нескінченний цикл.

Рекурсія навколо нас

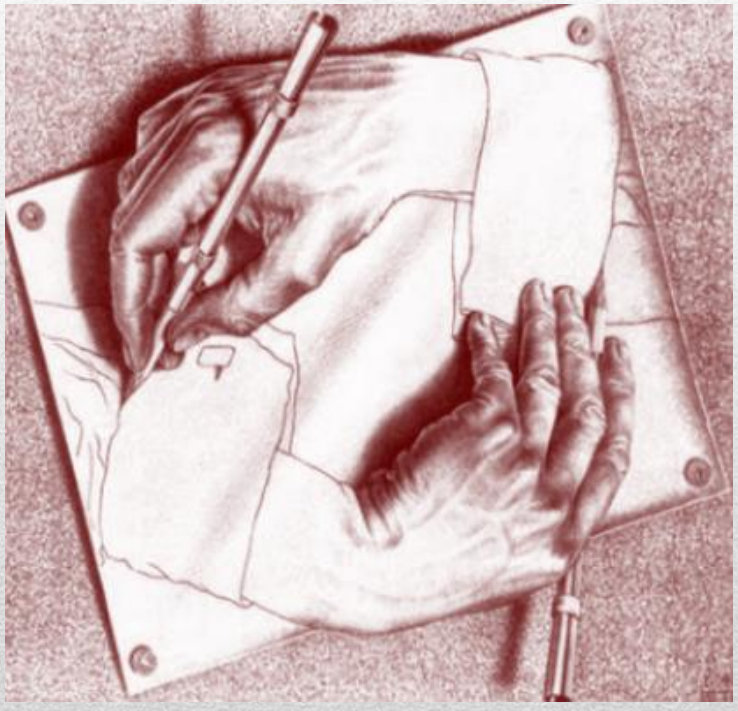
Ішов я лісом, бачу міст, під мостом ворона мокне. Взяв її за хвіст, поклав на міст, хай ворона сохне. Ішов я лісом, бачу міст, на мосту ворона сохне. Взяв її за хвіст, поклав під міст, хай ворона мокне...

Макс Фрай,
«Ворона на мосту»

Оригінальна реклама какао Droste

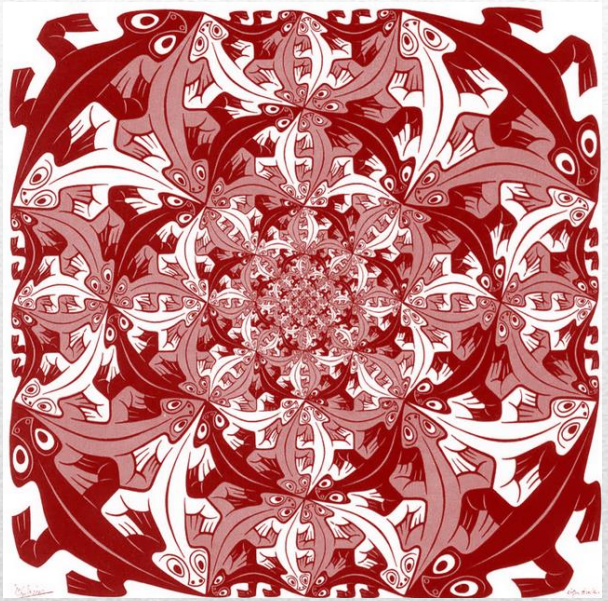
Можемо бачити трикутник Серпінського на зображенні Римської мозаїки 3 — 4 століття в античному музеї міста Арль, Франція

Рекурсія навколо нас



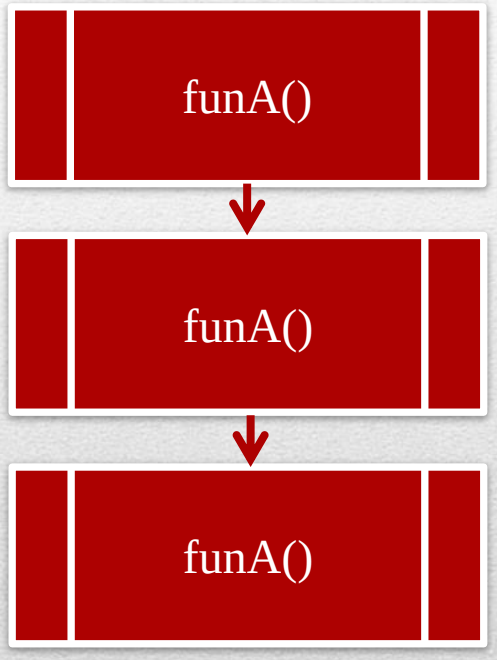
Поняття рекурсії

Рекурсія – це спосіб організації обчислювального процесу, при якому функція в ході виконання операторів звертається сама до себе.

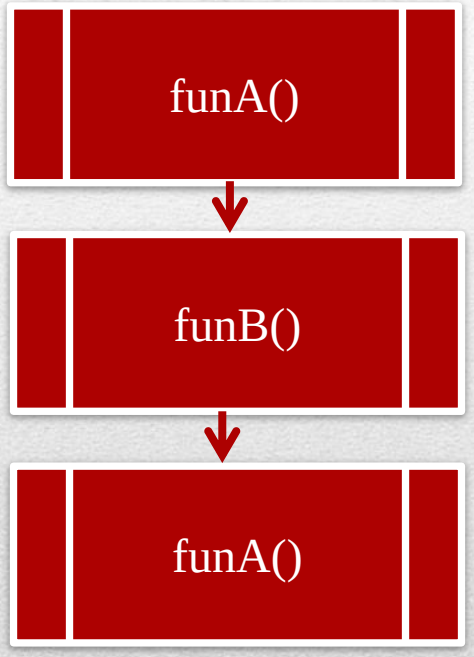


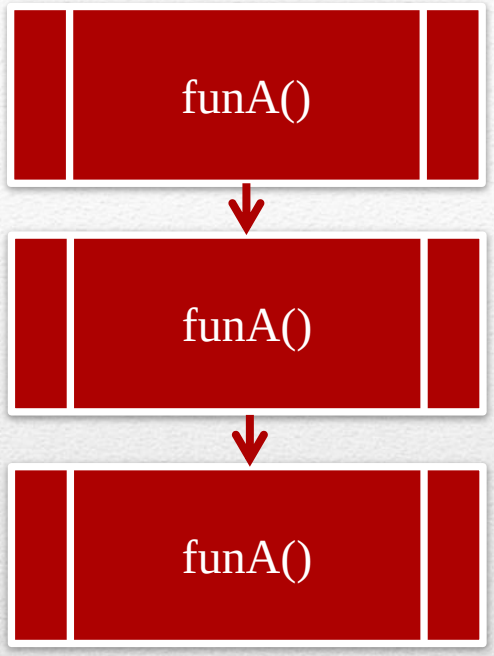
Функція називається рекурсивною, якщо під час її роботи можливий повторний її виклик безпосередньо (прямий виклик) або шляхом виклику іншої функції, в якій міститься звернення до неї (непрямий виклик).

Прямою (безпосередньою) рекурсією називається рекурсія, при якій в середині тіла деякої функції міститься виклик тієї ж функції.

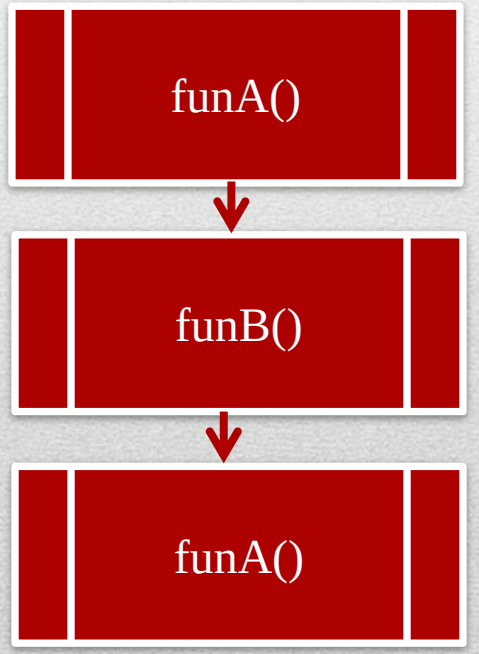


Непрямою рекурсією називається рекурсія, що здійснює рекурсивний виклик функції шляхом ланцюга викликів інших функцій. При цьому всі функції ланцюга, що здійснюють рекурсію, вважаються рекурсивними.

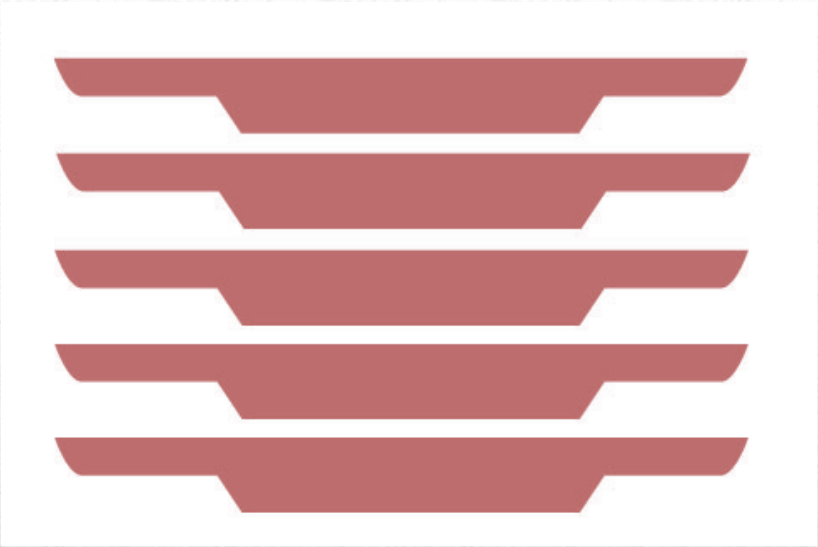


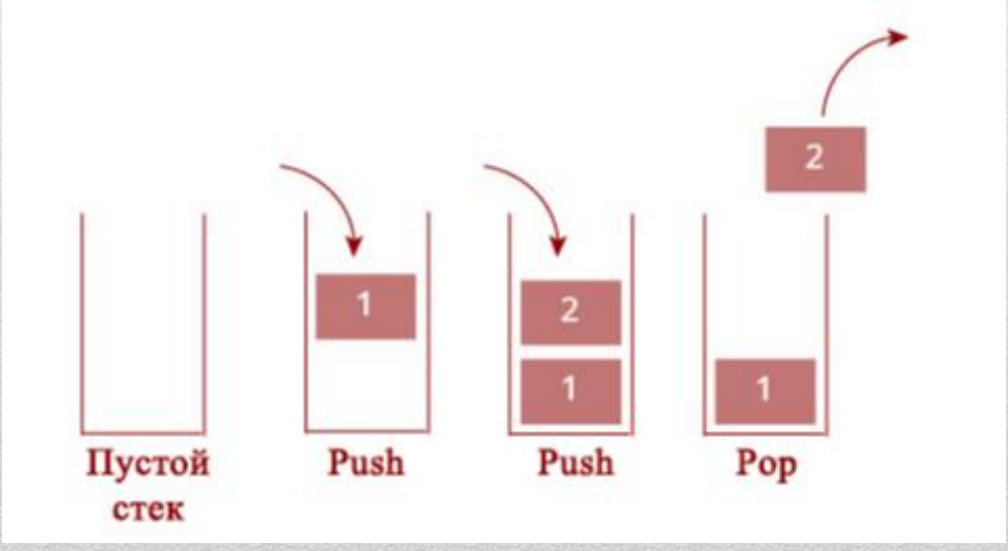



```
void funA() {
    Оператори;
    funA();
    Оператори;
}
```




```
void funA() {
    Оператори;
    funB();
    Оператори;
}
void funB() {
    Оператори;
    funA();
    Оператори;
}
```



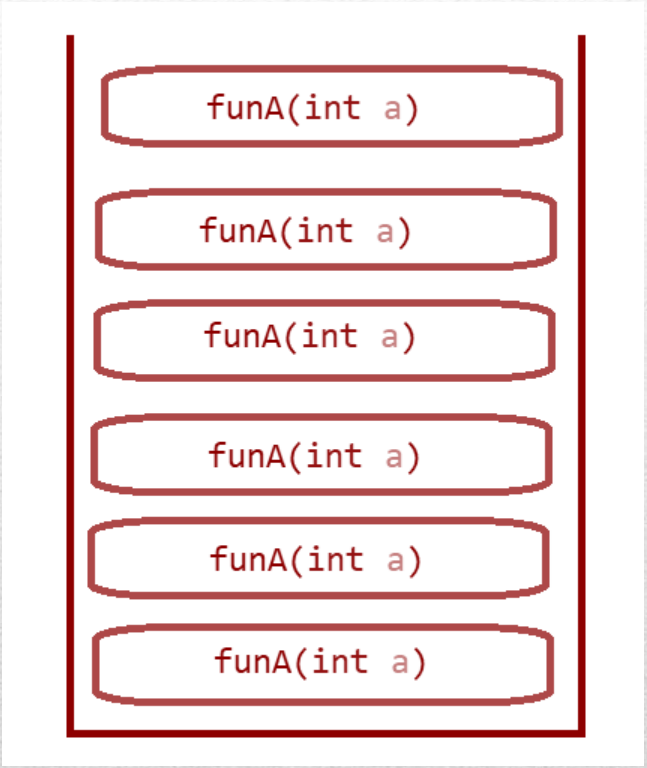

1
2
1
2
1
Пустой стек
Push
Push
Pop

```c
#include<stdio.h>


int funA(int);

int main() {
    funA(5);
    return 0;
}

int funA(int a) {
    if (a < 1) return 0;
    printf("%d\n", a);
    a--;
    return funA(a);
}
```



ПРИКЛАД №1

Класичний приклад рекурсивної функції - обчислення ***факторіалу,*** тобто добутку натуральних чисел від 1 до N.

$$N! = 1* 2* 3* \dots *N$$

Факторіал визначається рівнянням

$$n! = n \cdot (n - 1) \cdot (n - 2) \dots 3 \cdot 2 \cdot 1.$$

n! для будь - якого позитивного цілого числа n дорівнює n, помноженому на (n −1)!:

$$n! = n \cdot [(n - 1) \cdot (n - 2) \dots 3 \cdot 2 \cdot 1] = n \cdot (n - 1)!$$

$$0!=1$$

$$5!=5*4!=5*4*3!=5*4*3*2!=5*4*3*2*1!=5*4*3*2*1$$

Результат виконання програми

```
n=5
factorial 5 = 120
Для продолжения нажмите любую клавишу
```

```c
#include <stdio.h>

void factorial(int n);

int main() {
    int n;
    printf("n="); scanf_s("%d", &n);
    factorial(n);
return 0;
}

void factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++)
        result *= i;
    printf("factorial %d = %d\n", n, result);
}
```

Результат виконання програми

```
n=5
factorial 5 = 120
Для продолжения нажмите любую клавишу
```

```c
#include <stdio.h>

int factorial(int n);

int main() {
    int n;
    printf("n=");
    scanf_s("%d", &n);
    printf("factorial %d = %d\n", n, factorial(n));
return 0;
}

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

n! визначається наступним чином:
якщо n = 0, то n! = 1;
якщо n> 0, то n! = n * (n-1)!

$$n! = \begin{cases} 1, & n = 0, \\ (n-1)! \times n, & n > 0. \end{cases}$$

Результат виконання програми

```
n=5
factorial 5 = 120
Для продолжения нажмите любую клавишу
```

```c
#include <stdio.h>

int factorial(int n);

int main() {
    int n;
    printf("n="); scanf_s("%d", &n);
    printf("factorial %d = %d\n", n, factorial(n));
return 0;
}

int factorial(int n) {
return ((n == 1) ? 1 : n * factorial(n - 1));
}
```

Очікування множення

Нічого не множиться, поки не спустимося до базового випадку factorial (1). Потім починаємо згортку, по одному кроку.

```
factorial(5);
5 * factorial(4);
5* 4 * factorial(3);
5* 4* 3 * factorial(2);
5* 4* 3 * 2 * factorial(1);
5*4*3*2*1;
5*4*3*2;
5*4*6;
5*24;
120;
```

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>

int fibonacci(int); // Виведемо перші 13 чисел Фібоначчі

int main() {
    for (int i = 1; i < 14; i++)
    printf("%d ", fibonacci(i));
    printf("\n");
return 0;
}

int fibonacci(int number){
    if (number == 0)
    return 0; // базовий випадок (умова завершення)
    if (number == 1)
    return 1; // базовий випадок (умова завершення)
    return fibonacci(number - 1) + fibonacci(number - 2);
}
```

ПРИКЛАД №2

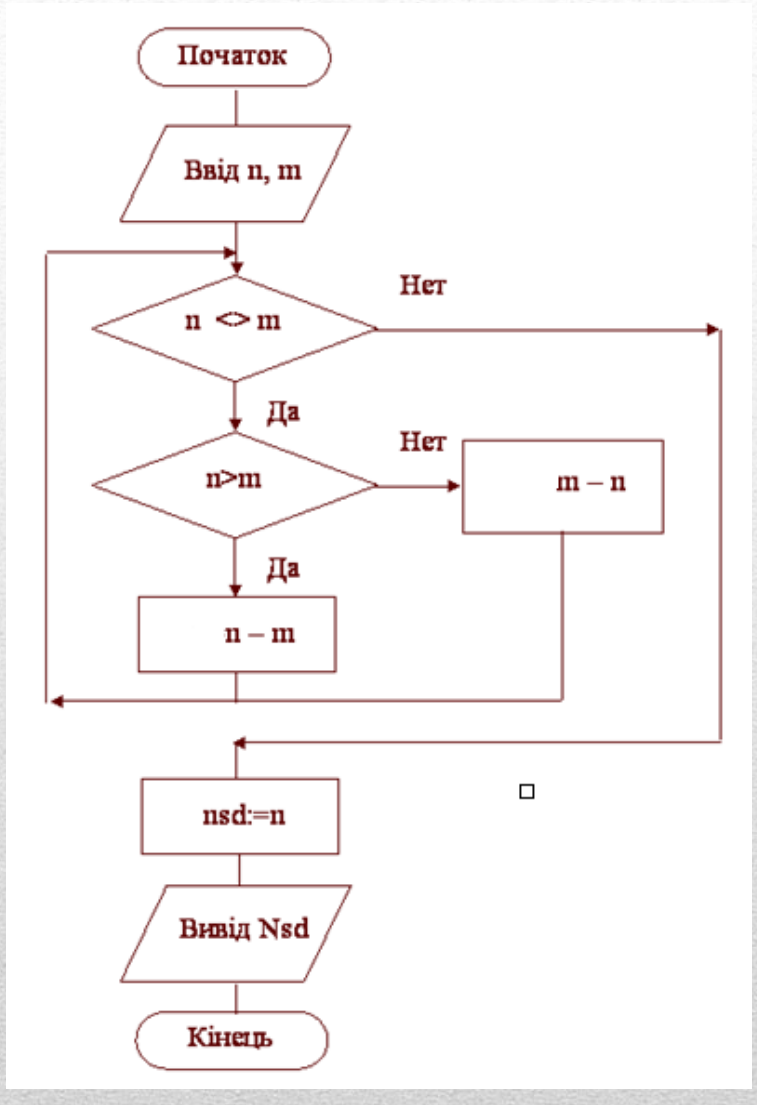
Алгоритм Евкліда

(також називається **евклідів алгоритм**) — ефективний метод обчислення найбільшого спільного дільника (НСД). Названий на честь грецького математика Евкліда.

Найбільший спільний дільник за алгоритмом Евкліда

Число ***n*** є ***дільником*** числа ***m***, якщо
число ***m*** ділиться на число ***n*** без остачі.
Дільники числа 18: 1, 2, 3, **6**, 9, 18.
Дільники числа 24: 1, 2, 3, 4, **6**, 12, 24.
Найбільший спільний дільник чисел 18 та 24 - 6.

Скорочено: НСД (18, 24) = 6.



		n	m	nsd
	scanf_s("%d %d", &n, &m);	18	24	

1	if (n == m) return n;
	if (n > m) n -= m;
	else m - =n;
		18	6	

2	if (n == m) return n;
	if (n > m) n -= m;
	else m - =n;
		12	6	

3	if (n == m) return n;
	if (n > m) n -= m;
	else m - =n;
		6	6	

4	if (n == m) return n;			6

```c
#include <stdio.h>
#include <windows.h>

int MaxDiv(int, int);

int main() {
    SetConsoleCP(1251); SetConsoleOutputCP(1251);
    int a, b;
    printf_s("введіть два числа для пошуку найбільшого
    спільного дільника\n");
    scanf_s("%d %d", &a, &b);
    printf_s("Найбільший спільний дільник %d\n", MaxDiv(a, b));
return 0;
}

int MaxDiv(int a, int b) {
    if (a == b) return a;
    else if (a > b) return MaxDiv(a - b, b);
        else return MaxDiv(a, b - a);
}
```

```
введіть два для пошуку найбільшого спільного дільника
18 24
Найбільший спільний дільник 6
Для продолжения нажмите любую клавишу . . .
```

Трикутник Паскаля

Трикутник Паскаля - безкінечна таблиця трикутної форми біноміальних коефіцієнтів.

ПРИКЛАД №3

Трикутник Паскаля

Трикутник Паскаля складається з числових рядків. На першій сходинці одне число, на другій — два, на третій … Перше і останнє число кожного рядка дорівнює 1. Кожне з інших чисел дорівнює сумі двох розташованих над ним чисел попереднього рядка.

0	1
1	1 1
2	1 2 1
3	1 3 3 1
4	1 4 6 4 1
5	1 5 10 10 5 1
6	1 6 15 20 15 6 1
7	1 7 21 35 35 21 7 1
8	1 8 28 56 70 56 28 8 1

n \ k	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	56	70	56	28	8	1

n \ k	0	1	2	3	4	5	6	7	8
0	C_0^0								
1	C_1^0	C_1^1							
2	C_2^0	C_2^1	C_2^2						
3	C_3^0	C_3^1	C_3^2	C_3^3					
4	C_4^0	C_4^1	C_4^2	C_4^3	C_4^4				
5	C_5^0	C_5^1	C_5^2	C_5^3	C_5^4	C_5^5			
6	C_6^0	C_6^1	C_6^2	C_6^3	C_6^4	C_6^5	C_6^6		
7	C_7^0	C_7^1	C_7^2	C_7^3	C_7^4	C_7^5	C_7^6	C_7^7	
8	C_8^0	C_8^1	C_8^2	C_8^3	C_8^4	C_8^5	C_8^6	C_8^7	C_8^8

Для будь-якого допустимого значення *n* діє

$$C_n^0 = 1$$

$$C_n^n = 1$$

$$C_n^k = \frac{n!}{k!\,(n-k)!}$$

$$\mathrm{v}(i,j) = \mathrm{v}(i-1,j-1) + \mathrm{v}(i-1,j)$$

```c
#include <stdio.h>
#include <windows.h>

// Знаходження елемента трикутника Паскаля

int PascalTriangle(int, int);

int main() {
    SetConsoleOutputCP(1251);
    int a, b;
    printf_s("Введіть рівень та індекс елемента(починаючи з нуля)\n");
    scanf_s("%d %d", &a, &b);

    printf_s("%d\n", PascalTriangle(a, b));
    return 0;
}

        // n - рівень, к - елемент рівня

int PascalTriangle(int n, int k) {
  if (n == k || k == 0) return 1;
  return PascalTriangle(n - 1, k - 1) + PascalTriangle(n - 1, k);
}
```

n \ k	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	56	70	56	28	8	1

Результат виконання програми

```
Введіть рівень та індекс елемента(починаючи з нуля)
6 3
20
Для продолжения нажмите любую клавишу . . .
```