



# Рекурсия

---

Изготвена от Никола Светославов

# Summary



Какво е рекурсия?



Пряка и косвена рекурсия



Дъно на рекурсията



Създаване на рекурсивни методи



Рекурсия vs Цикли

# Какво е рекурсия?

Всеки да напише „рекурсия“ в google търсачката си

Какво се случи?

# Какво е рекурсия?

- Един обект наричаме **рекурсивен**, ако съдържа себе си или е дефиниран чрез себе си.
- **Рекурсия** е програмна техника, при която даден **метод извиква сам себе си** при решаването на определен проблем. Такива методи наричаме **рекурсивни**.
- Рекурсията е програмна техника, чиято правилна употреба води до елегантни решения на определени проблеми. Понякога нейното използване може да опрости значително кода и да подобри четимостта му.

# Защо да учим рекурсия?

- Има проблеми, които са много сложни за решаване чрез цикли
- Може да бъдат разрешени итеративно, но на всяка итерация се налага да се съхранява в паметта стойността на променливите
- Вместо да извършвате цялата тази процедура ръчно, за да решите проблем от такъв тип, може да използвате рекурсия
- В случая рекурсивното решение ще бъде просто няколко реда

Пример



Пример



Пример



```
//Returns the sum of all numbers from 0 to a given integer

int sumRange(int num)

{
    if (num == 0)
    {
        return 0
    }
    return num + sumRange(num - 1)
}
```

# RECURSION

# Примерна програма

- Ако  $n$  е произволно естествено число, следната дефиниция на функцията факториел е рекурсивна
- Условието при  $n = 0$  не съдържа обръщение към функцията факториел и се нарича гранично.

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

Пример



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

# Примерна програма

- В тази програма е описана рекурсивната функция fact, която приложена към естествено число, връща факториела на това число
- Стойността на функцията се определя посредством обръщение към самата функция в оператора `return n * fact(n - 1);`
- В този случай, рекурсивното дефиниране на функцията факториел не е подходящо, тъй като съществува лесно итеративно решение

# Примерна програма

- Да се напише рекурсивна функция, която намира най-големия общ делител на две естествени числа.
- В случая  $a = b$  играе ролята на гранично условие, защото не съдържа обръщение към функцията  $\text{gcd}$ (Greatest common divisor) и чрез него се достига "дъното".

$$\text{gcd}(a, b) = \begin{cases} a, & a = b, \\ \text{gcd}(a - b, b), & a > b, \\ \text{gcd}(a, b - a), & a < b. \end{cases}$$

## Пример



```
int gcd(const int a, const int b)
{
    if (a == b)
        return a;

    if (a > b)
        return gcd(a - b, a);

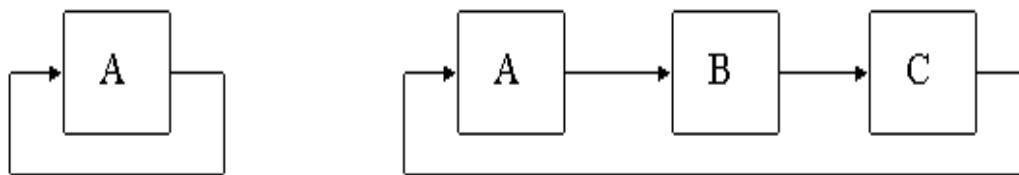
    return gcd(a, a - b);
}
```

# Пряка и косвена рекурсия

- Когато в тялото на метод се извършва извикване на същия метод, казваме, че методът е **пряко рекурсивен**.
- Ако метод A се обръща към метод B, B към C, а C отново към A, казваме, че методът A, както и методите B и C са **непряко (косвено) рекурсивни** или **взаимно-рекурсивни**.
- Веригата от извиквания при косвената рекурсия може да съдържа множество методи, както и разклонения, т.е. при наличие на едно условие да се извика един метод, а при различно условие да се извика друг.



# Пример



# Пример

```
// An example of direct recursion

void directRecFun()

{
    // Some code....
    directRecFun();
    // Some code...
}
```

# Пример

```
// An example of indirect recursion

void indirectRecFun1()

{
    // Some code...
    indirectRecFun2();
}

void indirectRecFun2()

{
    // Some code...
    indirectRecFun3();
}

void indirectRecFun3()

{
    // Some code...
    indirectRecFun1();
}
```



First Skill

Second Skill

## Дъно на рекурсията

- Реализирайки рекурсия, трява да сме сигурни, че след краен брой стъпки ще получим конкретен резултат.
- Трява да имаме един или няколко случаи, чието решение можем да намерим директно, без рекурсивно извикване. Тези случаи наричаме **дъно на рекурсията**.
- Ако даден рекурсивен метод няма дъно на рекурсията, тя ще стане **безкрайна** и резултатът ще е **StackOverflowException**.



Third Skill

Conclusion

PAGE 19

# Пример

```
void stackoverflow(int random)
{
    stackoverflow(random - 1);
}

int main()
{
    int test = INT_MAX;
    stackoverflow(test);
}
```

# Stack vs Heap

## Stack

Стекът е област за временно съхранение на информация.

Той е кратковременна памет.

C++ използва стека основно за реализиране на обръщения към функции.

Когато пък всяка от тези функции завършва, стековите рамки на тези функции автоматично се разрушават. Така стекът се изчиства.

## Heap

**Хийпът** е по-постоянна област за съхранение на данни.

Той е един вид дълготрайна памет. Особеност на тази памет е, че тя не се свързва с имена на променливи.

С разположените в нея обекти се работи косвено чрез указатели.

Обикновено се използва при работа с т.нр. **динамични структури от данни**.

# Създаване на рекурсивни методи

- Когато създаваме рекурсивни методи, трябва да разбием задачата, която решаваме, на подзадачи, за чието решение можем да използваме същия алгоритъм (рекурсивно).
- Комбинирането на решенията на всички подзадачи, трябва да води до решение на изходната задача.
- При всяко рекурсивно извикване, проблемната област трябва да се ограничава така, че в даден момент да бъде достигнато дъното на рекурсията, т.е. разбиването на всяка от подзадачите трябва да води рано или късно до дъното на рекурсията.

# Рекурсия vs цикли(loops)

## ■ Рекурсията:

- Рамка в стековата памет се алокира при всяко едно извикване на функцията.
- Получава се "верига" от много стекови рамки, докато се достигне дъното на рекурсията.
- Рекурсията заема много памет, защото се пазят всички стекови рамки докато не се достигне дъното.

## ■ Циклите:

- Можете да използвате цикъли и без да създавате отделна функция.
- При всяка итерация в цикъла не се създава нова рамка и не се заема непрекъснато още памет.

# Рекурсия vs цикли(loops)

- Рискове при рекурсията:
  - StackOverFlow - няма повече стекова памет, в която да се заделят нови рамки (от там идва и името на форума).
  - OutOfMemory - няма повече Heap памет, в която да се заделя за рекурсивните обекти, които се инциализират вътре в рекурсивната функция.
- Рискове при циклите:
  - Може да се получи в някой частен случай безкраен цикъл.

# Рекурсия vs Цикли(loops)

Property	Recursion	Iteration
<b>Definition</b>	Function calls itself.	A set of instructions repeatedly executed.
<b>Application</b>	For functions.	For loops.
<b>Termination</b>	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
<b>Usage</b>	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
<b>Code Size</b>	Smaller code size	Larger Code Size.
<b>Time Complexity</b>	Very high(generally exponential) time complexity.	Relatively lower time complexity(generally polynomial-logarithmic).

# Препоръки

- **Препоръка:** Ако за решаването на някаква задача е уместно да се използва итеративен алгоритъм, реализирайте го.
- **Не се препоръчва** винаги използването на рекурсия, тъй като това води до големи загуби на памет.

# Demo

