Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Рекурзија

Структурно програмирање

ФИНКИ 2024

Што ако функцијата се повика самата себеси?

```
void Funk(int i) {
        cout << i;
        Funk(--i);
        }
```

Рекурзија

Постојат 2 начина за составување ПОВТОРУВАЧКИ алгоритми:

- Итерација (Iteration)
  - Се повторуваат само параметрите на алгоритмот, а не и самиот алгоритам. (со употреба на циклуси се повторува дел од кодот)
- Рекурзија (Recursion)
  - Повторувачки процес во кој алгоритмот СЕ ПОВИКУВА СЕБЕСИ. Алгоритмот се појавува во сопствената дефиниција. (функцијата се повикува себе си за да се повтори кодот)

Преку пример: Факториел на бројот n

- Производ на целите вредности од 1 до бројот n.
- Итеративен алгоритам
  - n!=n* (n-1)*(n-2)* … 2*1
  - Пр. 4!=4*3*2*1 = 24
- Рекурзивен алгоритам
  - n!=n*(n-1)!

Начин на пресметување

5! = 5 * 4!

4! = 4 * 3! …

По дефиниција 1! = 1

Тогаш важи

2! = 2 * 1! = 2 * 1 = 2;

3! = 3 * 2! = 3 * 2 = 6;

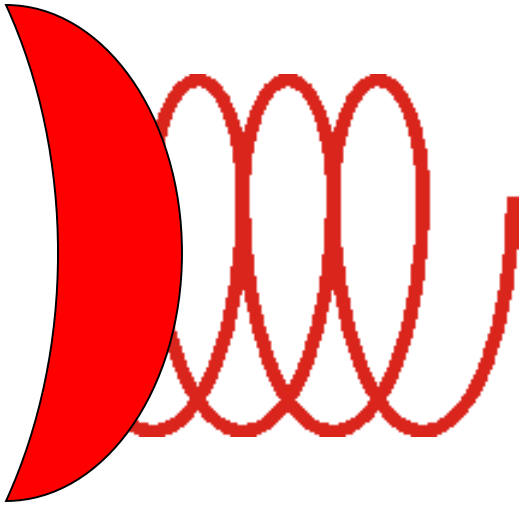
(поедноставување на проблемот)

(основен случај)

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

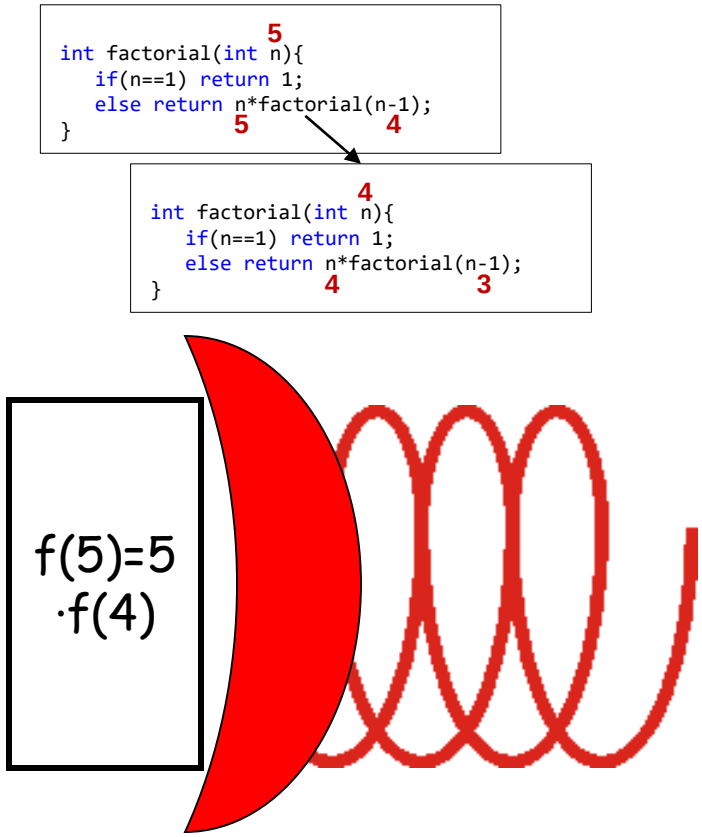
Патување во две насоки

- Прво – се разложува од врвот кон дното (од n до 1)
- Второ – се решава одејќи од дното кон врвот (од 1 до n)



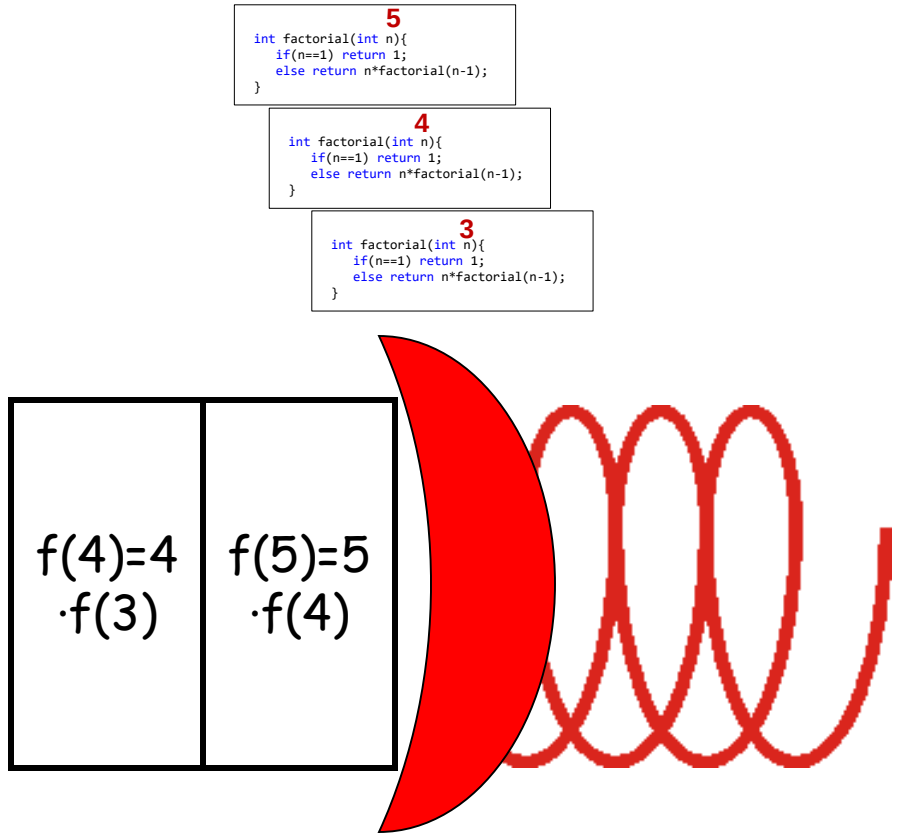
```
int factorial(int n){
    if(n==1) return 1;
    else return n*factorial(n-1);
}
```

Патување во две насоки



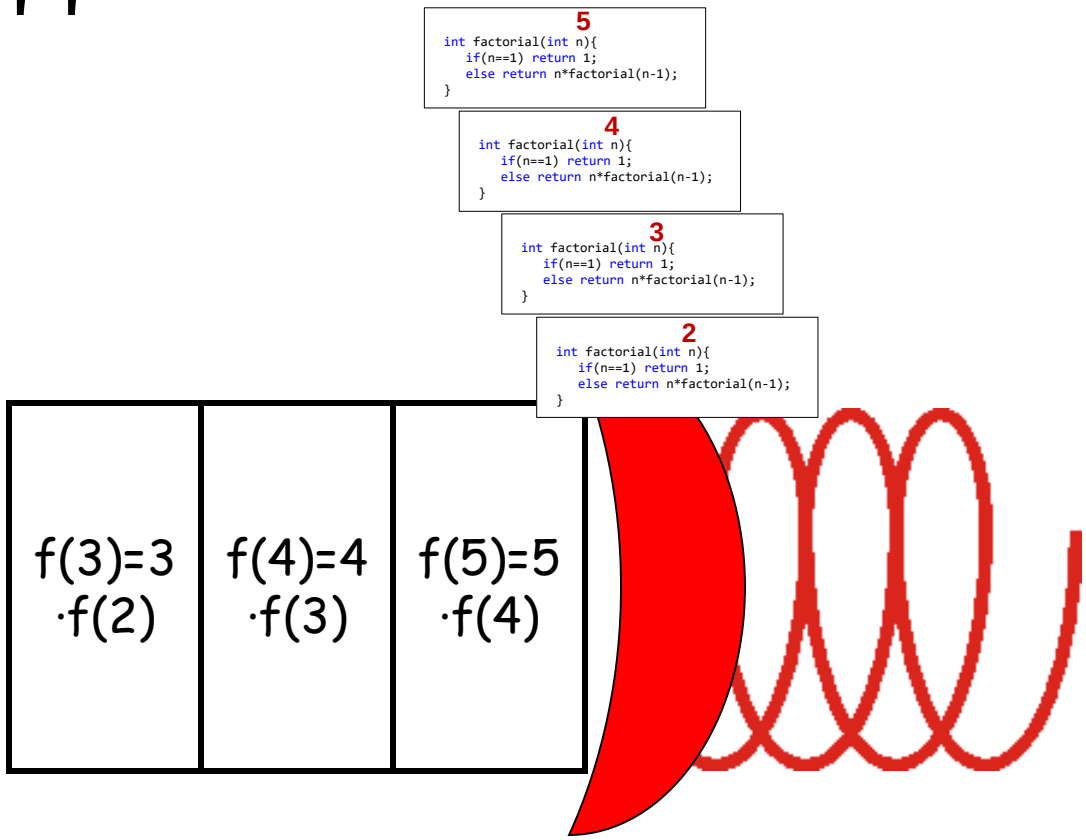

```
int factorial(int n){
    if(n==1) return 1;
    else return n*factorial(n-1);
}
```

Патување во две насоки



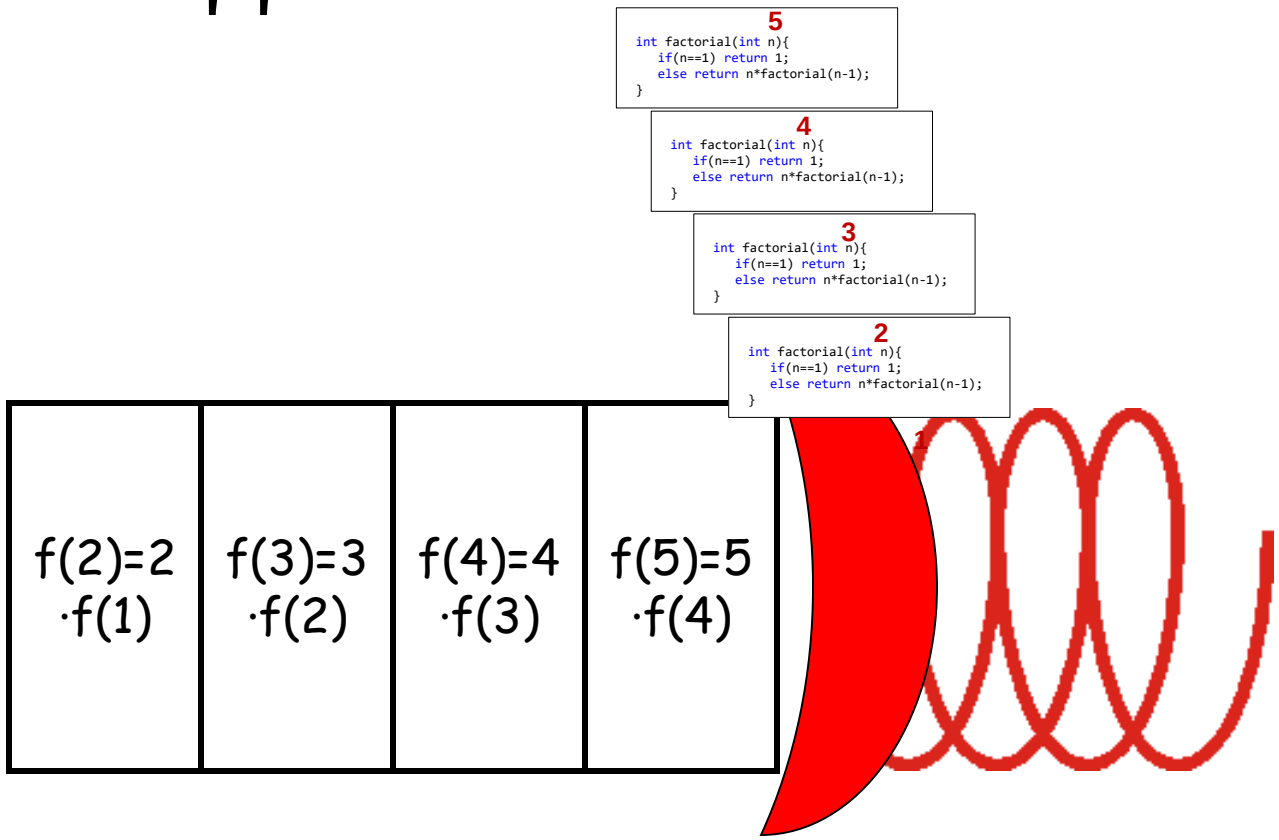

```
int factorial(int n){
    if(n==1) return 1;
    else return n*factorial(n-1);
}
```

Патување во две насоки



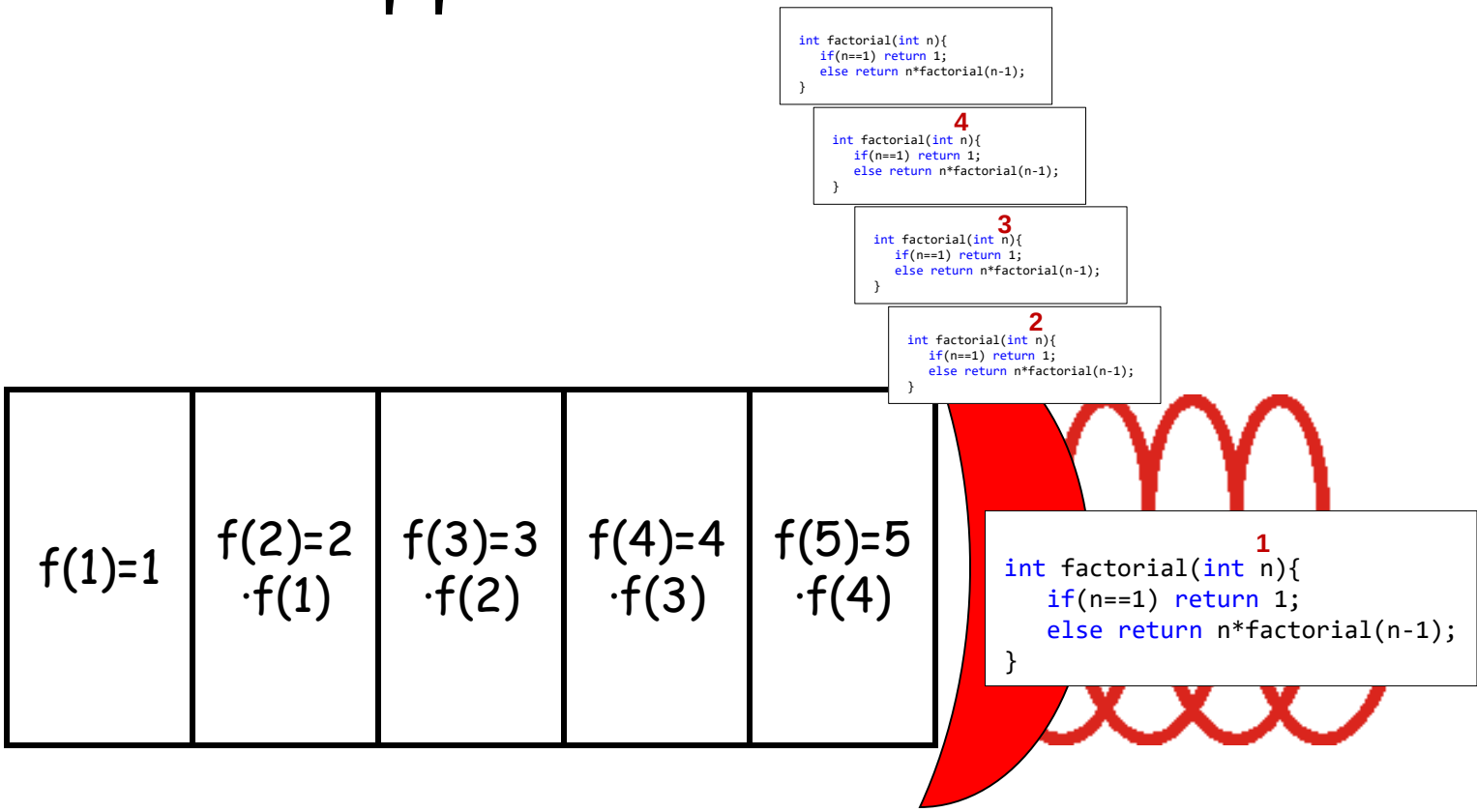

```
int factorial(int n){
    if(n==1) return 1;
    else return n*factorial(n-1);
}
```

Патување во две насоки




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

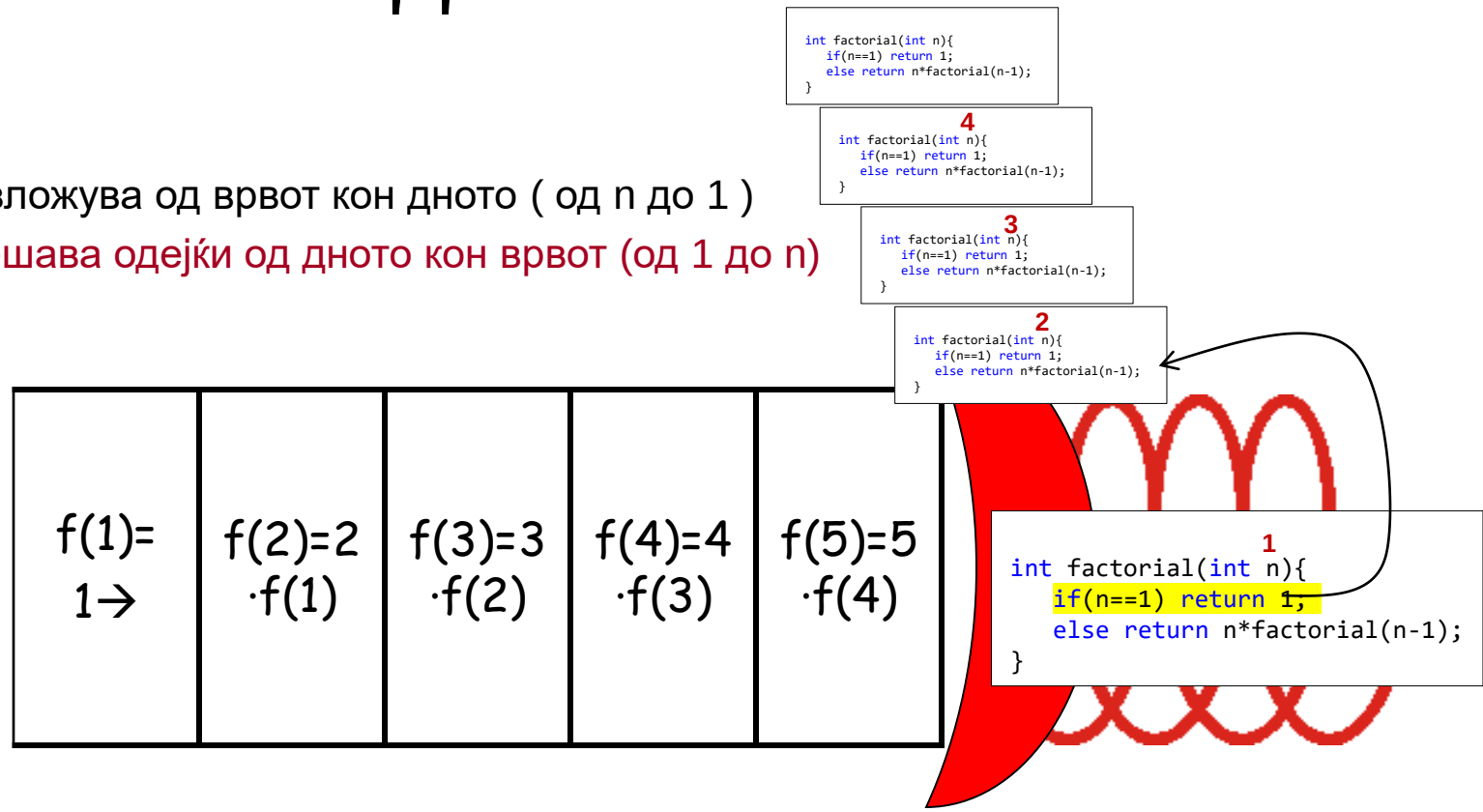
Патување во две насоки




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

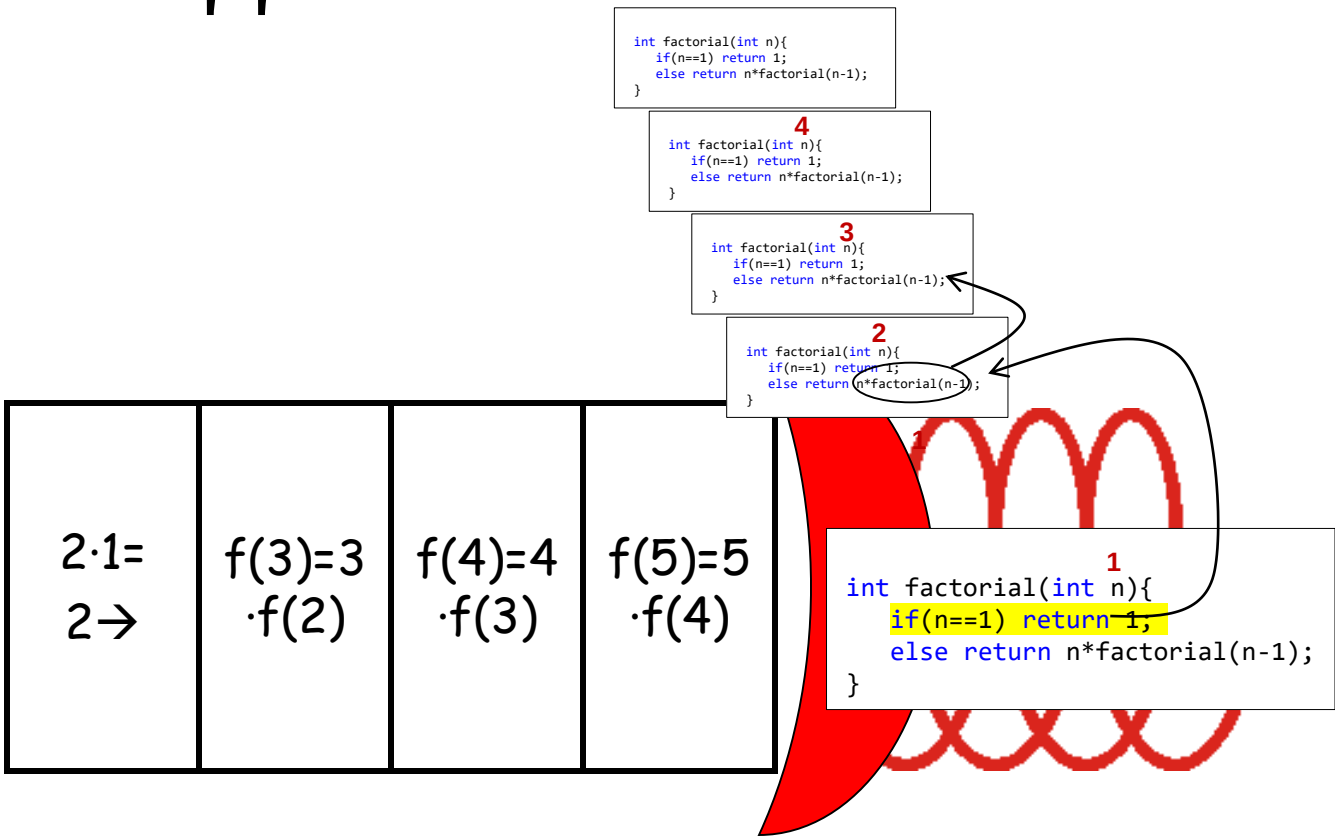
Патување во две насоки

- Прво – се разложува од врвот кон дното (од n до 1)
- Второ – се решава одејќи од дното кон врвот (од 1 до n)



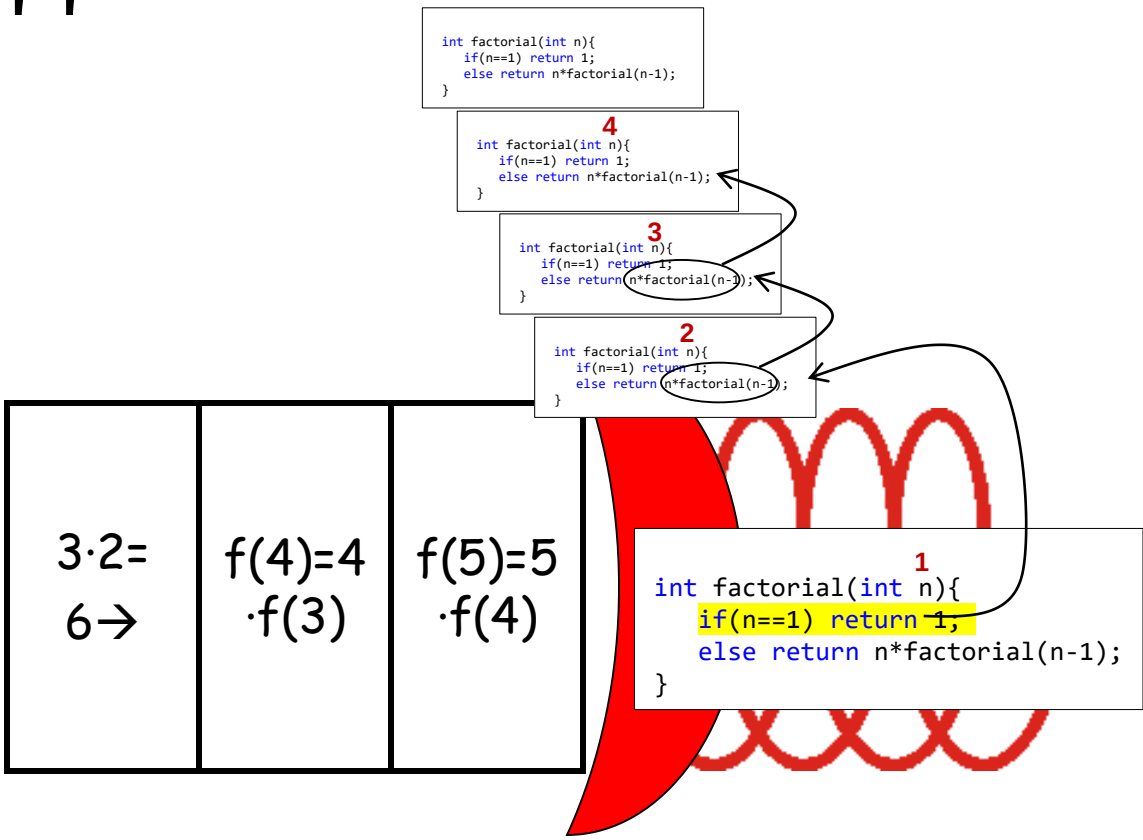

```
int factorial(int n){
   if(n==1) return 1;
   else return n*factorial(n-1);
}
```

Патување во две насоки



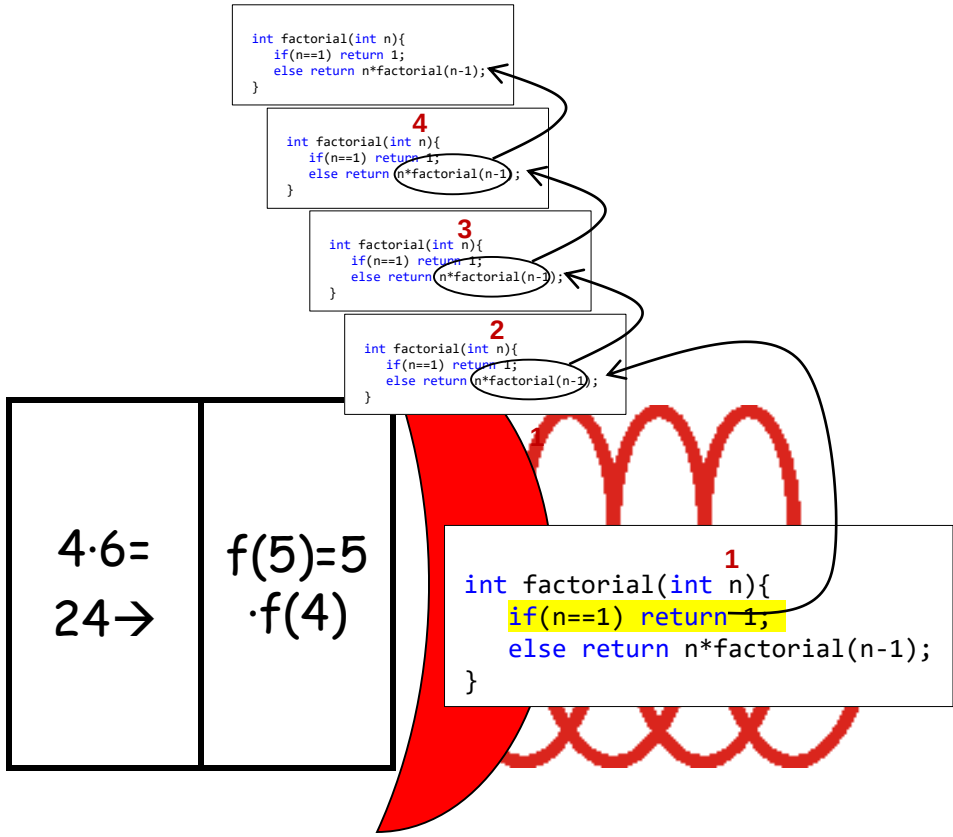

```
int factorial(int n){
    if(n==1) return 1;
    else return n*factorial(n-1);
}
```

Патување во две насоки



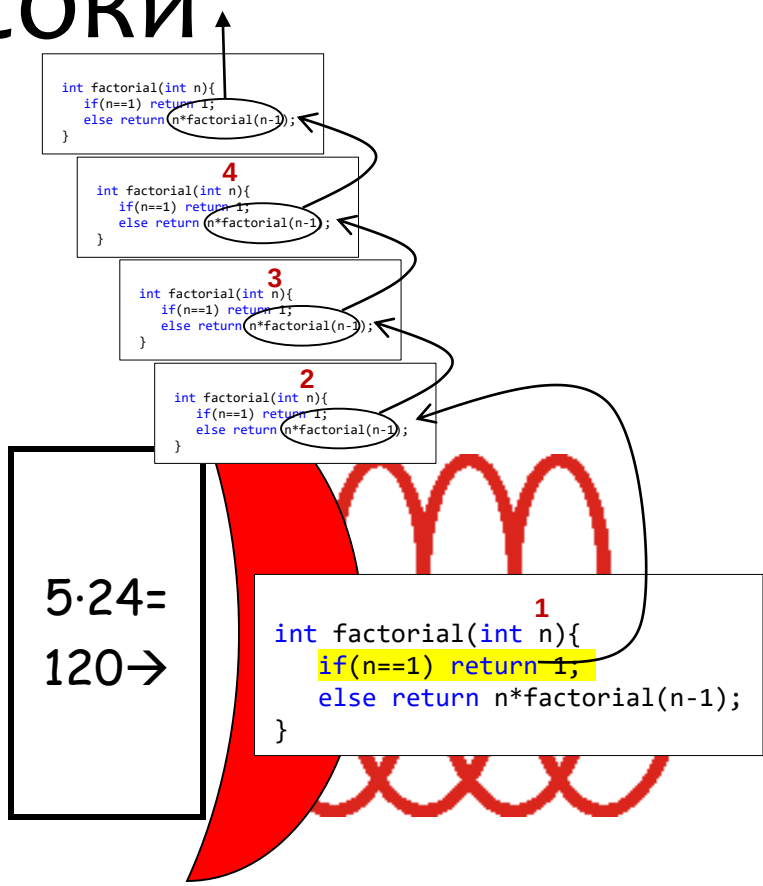

```
int factorial(int n){
    if(n==1) return 1;
    else return n*factorial(n-1);
}
```

Патување во две насоки



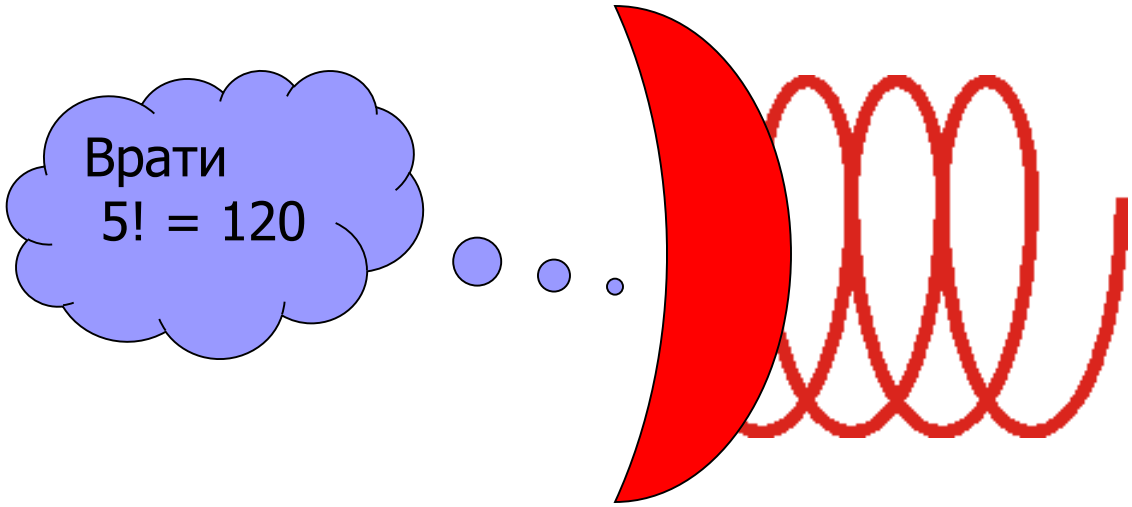

```
int factorial(int n){
    if(n==1) return 1;
    else return n*factorial(n-1);
}
```

Патување во две насоки




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

Патување во две насоки




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

Креирање рекурентни алгоритми

- Секоја рекурзивен алгоритам МОРА да има основен (граничен случај) :
  - Изразот што го решава проблемот директно (без влегување во рекурзија)
  - пр. factoriel (0)

- Остатокот од алгоритмот се нарекува општ случај (рекурентна врска)
  - n·factorial(n-1)

Пример

- За да се напише рекурзивен алгоритам:
  - Одредете го основниот случај factorial (0)
  - Одредете го општиот случај (рекурентната врска)
    n * factorial(n-1)
- СЕКОЈ рекурзивен повик мора да реши
  - Дел од проблемот

  ИЛИ
  - Да ја редуцира големината на проблемот

Пример

- Да се пресмета рекурзивно m^n, $n \in \mathbb{N}$
- m=2, n=3, $m^n = 2^3 = 8$
- $m^n \equiv$ power(m, n)
- Решение:
  - основен случај n=0, power(x,0)=1
  - Одредување на општиот случај
    n=1, power(2,1)=2
    n=2, power(2,2)=4
    n=3, power(2,3)=2·power(2,2)

Пример

- Да се пресмета рекурзивно m^n, $n \in \mathbb{N}$
- m=2, n=3, $m^n = 2^3 = 8$
- $m^n \equiv$ power(m, n)
- Решение:
  - основен случај n=0, power(x,0)=1
  - Одредување на општиот случај
    n=1, power(2,1)=2
    n=2, power(2,2)=4
    n=3, power(2,3)=2·power(2,2)

  → **power(m,n)=m·power(m,n-1)**

```
int power(int m, int n) {
        if(n) return m*power(m,n-1);
        else return 1;
}
```

Примери

Збир на целите броеви не поголеми од n

Примери

Збир на целите броеви не поголеми од n

```
int zbirN(int n) {
    if (n==0) return 0;
    else return n+zbirN(n-1);
}
```

```
void dzvezdi(int n) {
    if (n>0) {
      cout << '*';
      dzvezdi(n-1);
    }
    else cout << endl;
}
```

Збир на цифри на број

```
int zbirCifri(int n) {
        if (n<10) return n;
        else return n%10 + zbirCifri(n/10);
}
```

Опаѓачка низа броеви

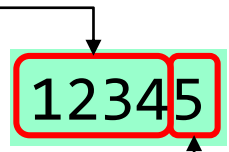
- За даден цел број n (n>0) да се отпечати опаѓачката низа броеви, почнувајќи од n, завршувајќи со 1

```
void niza(int n) {
    if (n > 0) {
        cout << n;
        niza(n - 1);
    }
}
```

Растечка низа броеви

- За даден цел број n (n>0) да се отпечати растечката низа броеви, почнувајќи од 1, завршувајќи со n

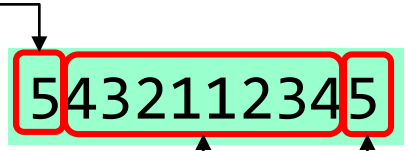
```
void niza(int n) {
    if (n > 0) {
        niza(n - 1);
        cout << n;
    }
}
```



Опаѓачка-растечка низа броеви

- За даден цел број n (n>0) да се отпечати прво опаѓачката низа броеви, почнувајќи од n, завршувајќи со 1 а потоа во во продолжение и растечката од 1 до n

```
void niza(int n) {
    if (n > 0) {
        cout << n;
        niza(n - 1);
        cout << n;
    }
}
```



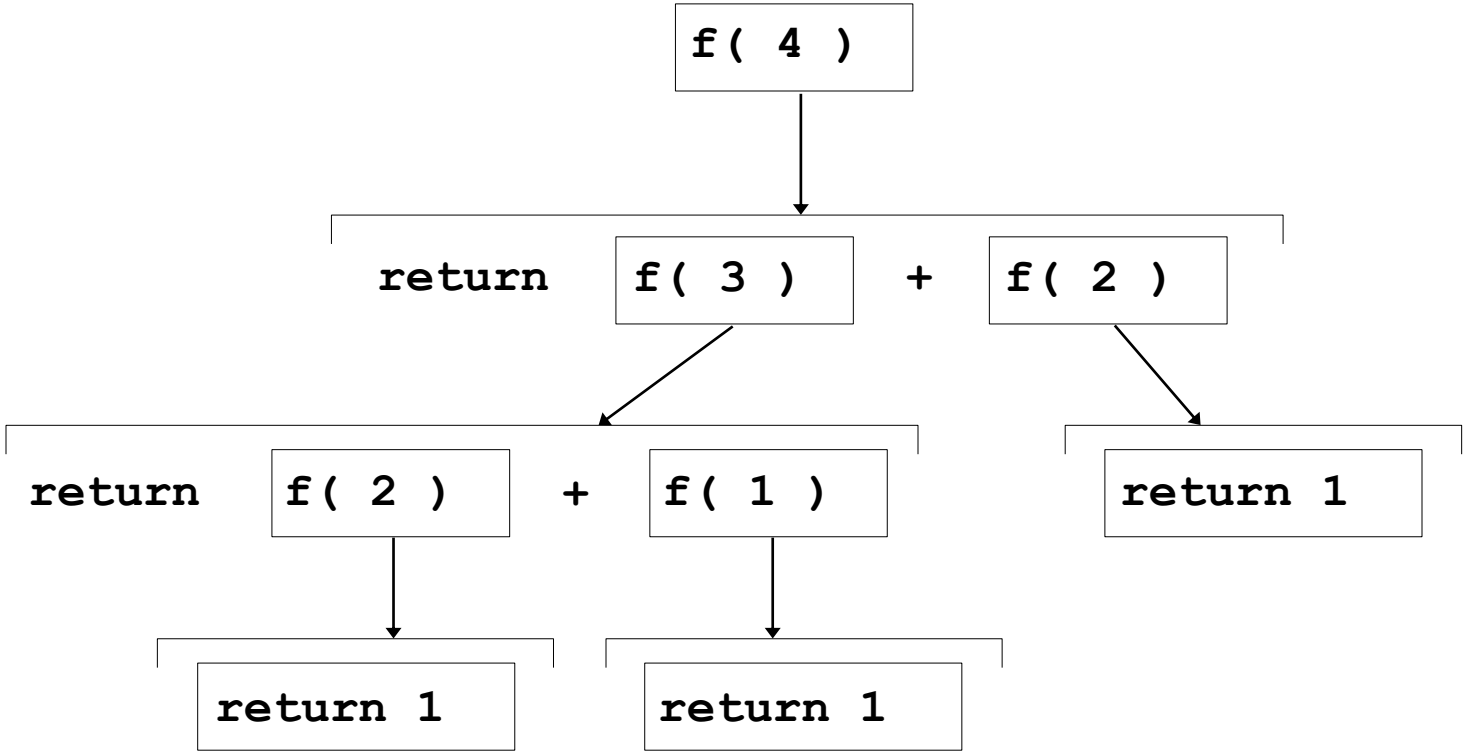
Фибоначиева низа броеви со рекурзија

- Фибоначиева низа: 1, 1, 2, 3, 5, 8...
  секој број се добива како збир на претходните два

  f(n) = f(n-1) + f(n-2) – рекурзивна формула

```
long fibonachi(long n) {
   if (n==1 || n==2) return 1; /*osnoven sluchaj*/
   else return fibonachi(n-1) + fibonachi(n-2);
}
```

Начин на извршување на функцијата за пресметка на Фибоначиева низа



Рекурзија vs Итерација

- Повторување
  - Итерација: експлицитни циклуси
  - Рекурзија: функциски повици
- Прекин на повторувањето
  - Итерација: условот за повторување повеќе не важи
  - Рекурзија: се препознава основниот случај
- И во обата случаи можни се појави на бесконечно повторување

Кога да НЕ се користи рекурзија

- Ако се одговори со НЕ на било кое од следните прашања:
  - Дали алгоритмот или податочните структури природно се зададени со рекурзивна формула?
  - Дали рекурзивното решение е пократко и поразбирливо?
  - Дали рекурзивното решение се одвива во прифатливи временски и просторни граници?
- Рекурзивните алгоритми генерално се малку побавни од итеративните алгоритми
  - Функциските повици земаат повеќе време отколку инструкција во циклус

Прашања?