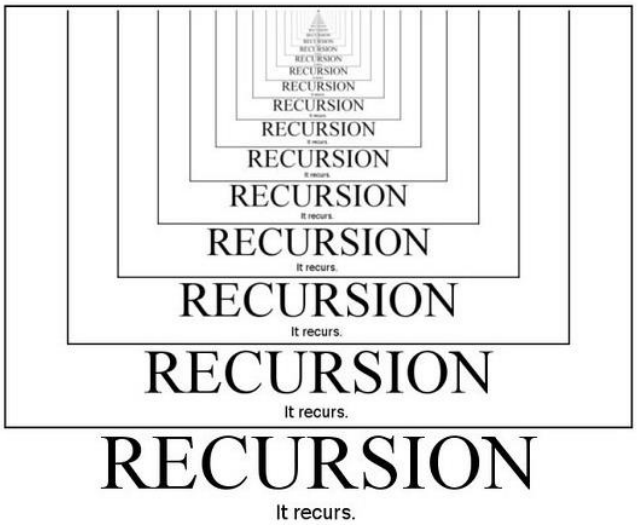
Wykład 11: Rekurencja.

dr inż. Andrzej Stafiniak

Wrocław 2024

Rekurencja

Rekurencją (rekursją, ang. recursion) nazywamy sytuację, kiedy funkcja wywołuje samą siebie.



https://algodaily.com/categories/recursion

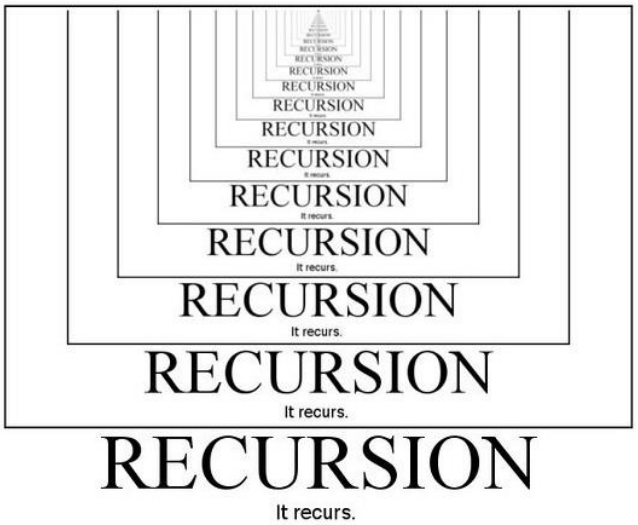
Rekurencja

Mechanizm **rekurencji** jest często wykorzystywany w inżynierii oprogramowania czy algorytmice, umożliwiając uproszczenie rozwiązania niektórych problemów.

Kiedy możemy zastosować **rekurencję** (dwa pytania):

- czy możemy rozwiązać postawiony problem przez jego podział na mniejsze „trywialne” problemy ?

- jeśli tak, to czy jesteśmy w stanie zrobić to regularnie?



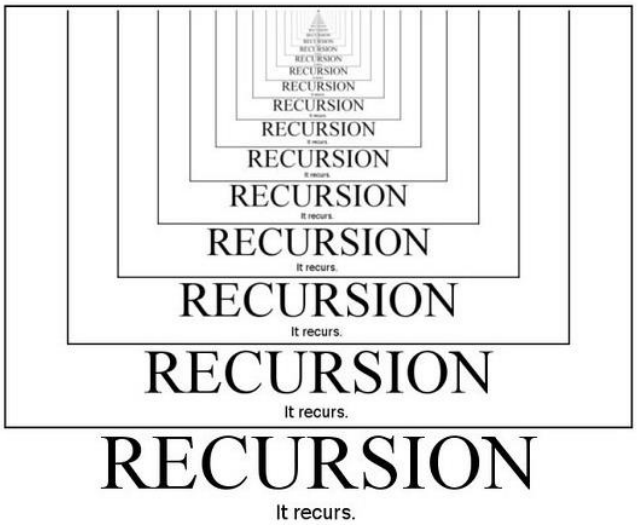
https://algodaily.com/categories/recursion

Rekurencja

Mechanizm **rekurencji** jest często wykorzystywany w inżynierii oprogramowania czy algorytmice, umożliwiając uproszczenie rozwiązania niektórych problemów.

Rekurencja -

- polega na ciągłym i regularnym dzieleniu postawionego problemu na mniejsze problemy, aż do uzyskania warunków „trywialności” czyli jednoznacznej odpowiedzi.

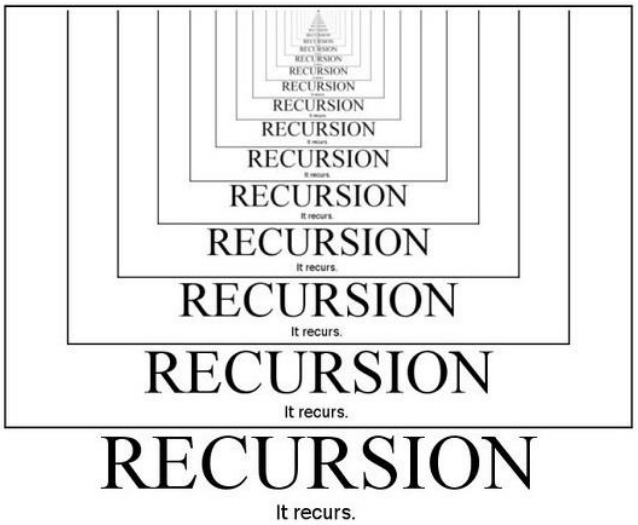


https://algodaily.com/categories/recursion

Rekurencja

Jednak zastosowanie rekurencji nie zawsze jest opłacalne:

- nie sprzyja czytelności kodu,
- wywołanie rekurencyjne funkcji są przeważnie wolniejsze od kolejnych obiegów pętli.



https://algodaily.com/categories/recursion

Rekurencja - prosty przykład

```
#include <stdio.h>

// prototyp funkcji
void odliczaj(int);

int main(){
   //wywołanie funkcji
   odliczaj(3);
   return 0;
}

void odliczaj(int n){
   if(n > 0){
      printf(„Start za %d s\n", n);
      odliczaj(n-1);
   }
}
```

Podejście iteracyjne - odliczanie

```
#include <stdio.h>

// prototyp funkcji
void odliczaj(int);

int main(){
   //wywołanie funkcji
   odliczaj(3);
   return 0;
}

void odliczaj(int n){
   while (n > 0){
      printf(„Start za %d
                        s\n", n);
      --n;
   }
}
```

Rekurencja - prosty przykład

Suma kolejnych elementów

```
#include <stdio.h>

// prototyp funkcji
int sumEl(int);

int main(){
   //wywołanie funkcji
   sumEl(3);
   return 0;
}

int sumEl(int n){
   if(n < 1)
      return 0;
   else
      return n + sumEl(n-1);
}
```

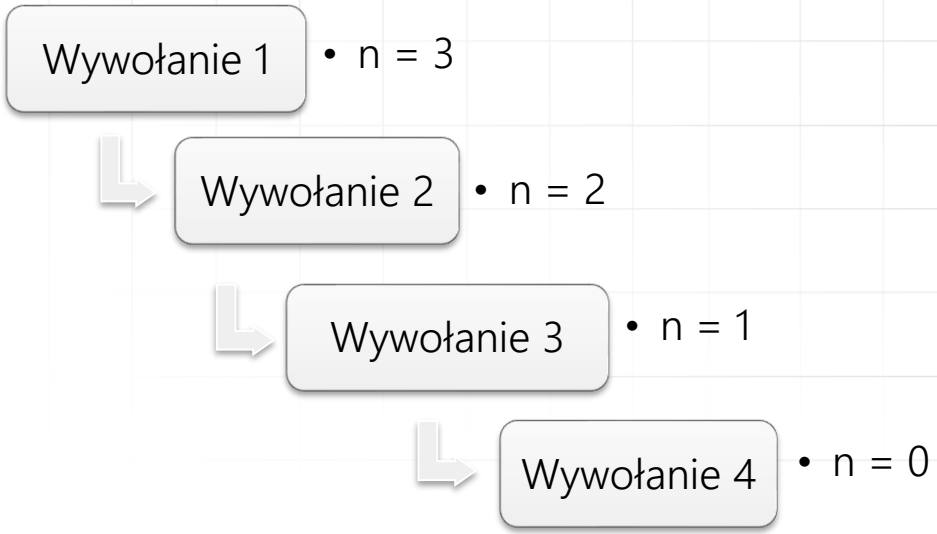
Silnia

```
#include <stdio.h>

// prototyp funkcji
int silnia(int);

int main(){
   //wywołanie funkcji
   silnia(5);
   return 0;
}

int silnia(int n){
   if(n < 1)
      return 1;
   else
      return n * silnia(n-1);
}
```

Rekurencja - prosty przykład

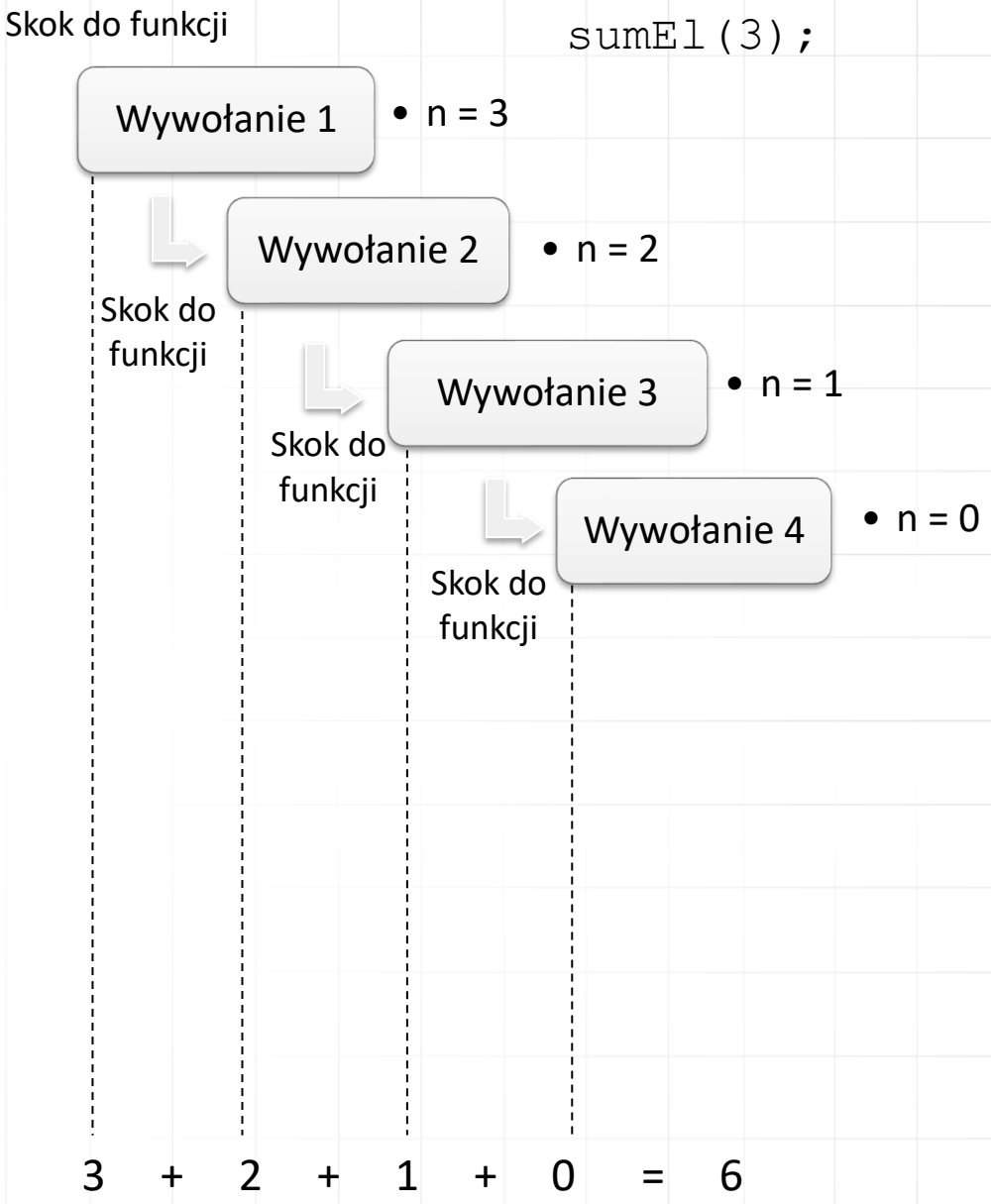
Suma kolejnych elementów

```
#include <stdio.h>

// prototyp funkcji
int sumEl(int);

int main(){
   //wywołanie funkcji
   sumEl(3);
   return 0;
}

int sumEl(int n){
   if(n < 1)
      return 0;
   else
      return n + sumEl(n-1);
}
```



Rekurencja - prosty przykład

- Funkcja `sumEl()` wywołana z arg. 3 wywoła dodatkowo 3 swoje kopie (kolejno z arg. 2, 1 oraz 0)
- Przed skokiem do kodu funkcji, program zapamiętuje adres, pod który ma powrócić sterowanie po zakończeniu wywołania.

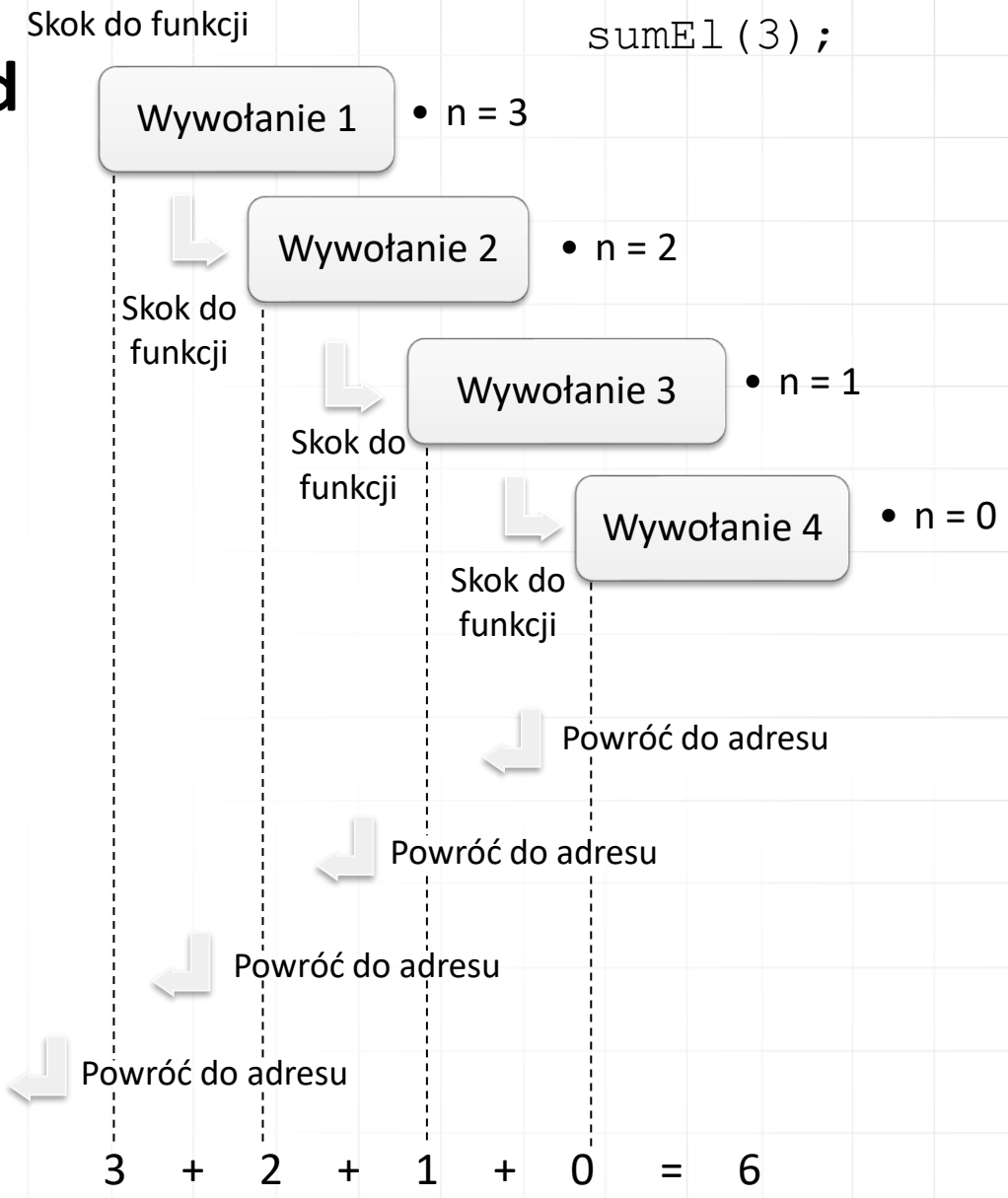



```
int sumEl(int n){
    if(n < 1)
        return 0;
    else
        return n + sumEl(n-1);
}
```

Rekurencja - prosty przykład

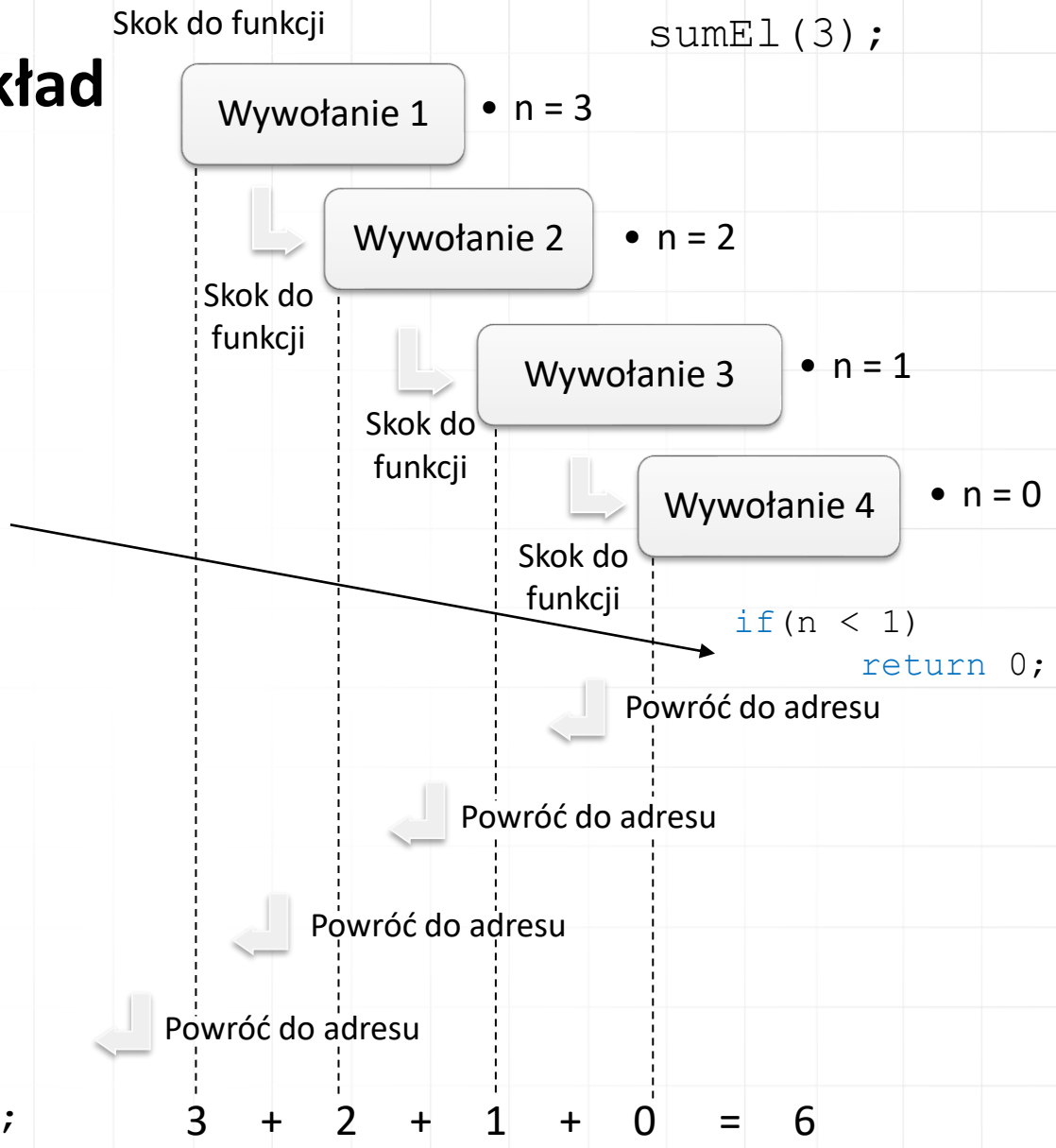
- Funkcja `sumEl()` wywołana z arg. 3 wywoła dodatkowo 3 swoje kopie (kolejno z arg. 2, 1 oraz 0)
- Przed skokiem do kodu funkcji, program zapamiętuje adres, pod który ma powrócić sterowanie po zakończeniu wywołania.
- **Adresy powrotu**, kopie kolejnych **argumentów wywołań funkcji** oraz ich **wartości zwracane** odkładane są na **stosie**.

```
int sumEl(int n){
    if(n < 1)
        return 0;
    else
        return n + sumEl(n-1);
}
```



Rekurencja - prosty przykład

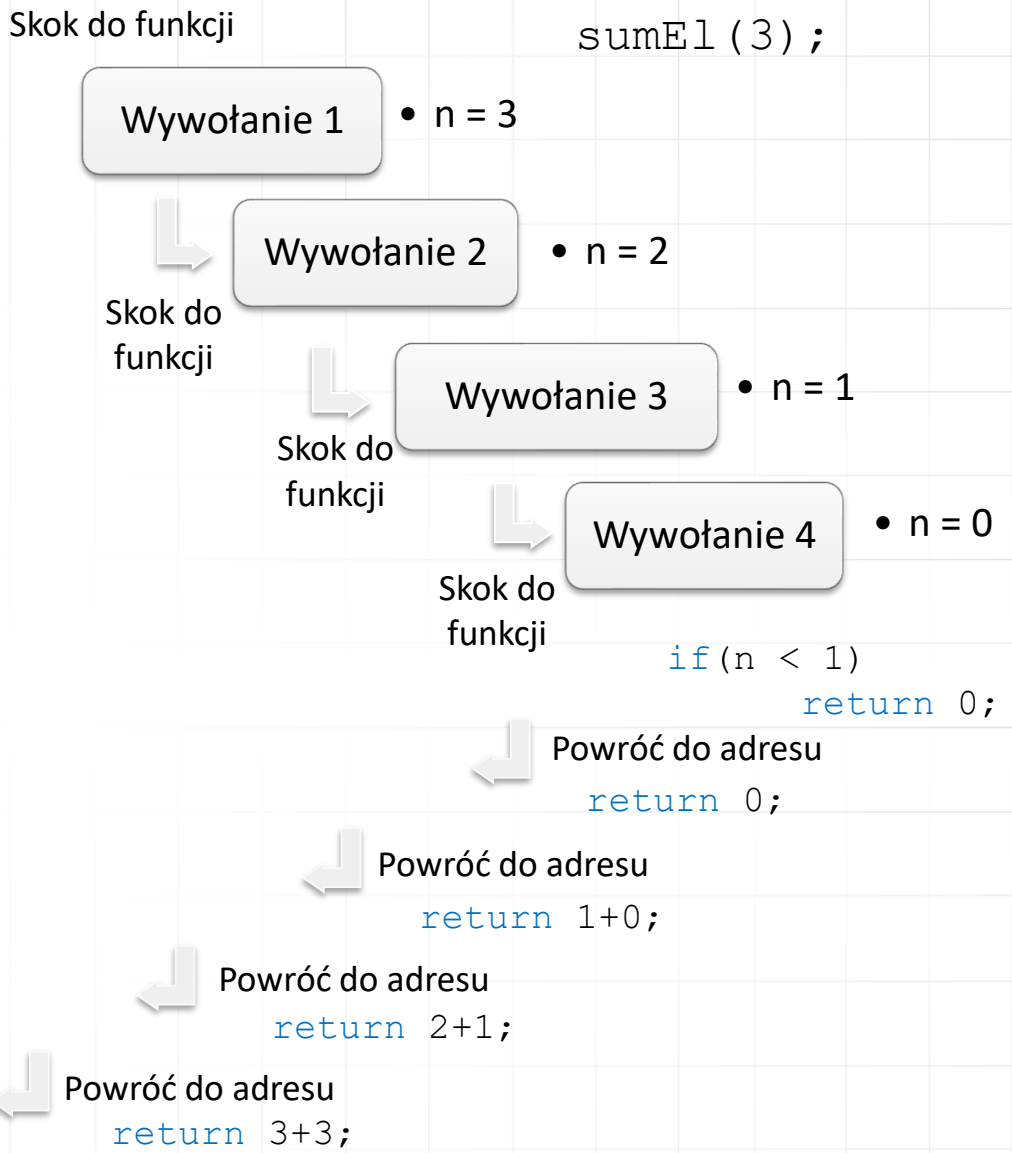
- Górnym ograniczeniem liczby **rekurencyjnych wywołań funkcji** jest rozmiar **stosu**.
- Bardzo ważne jest poprawne zdefiniowanie **warunku zakończenia rekurencji.**




```
int sumEl(int n){
    if(n < 1)
        return 0;
    else
        return n + sumEl(n-1);
}
```

Rekurencja - prosty przykład

- Wartości zliczane będą od końca, czyli od ostatniego wywołania rekurencyjnego zakończonego jednoznacznym warunkiem.
- Ważny jest również powrót sterowania w miejsce zaraz po tym gdzie funkcja była wywołana, ponieważ funkcja wywołująca mogła jeszcze się nie zakończyć.




```
int sumEl(int n){
    if(n < 1)
        return 0;
    else
        return n + sumEl(n-1);
}
```

Rekurencja – powrót z wywołania

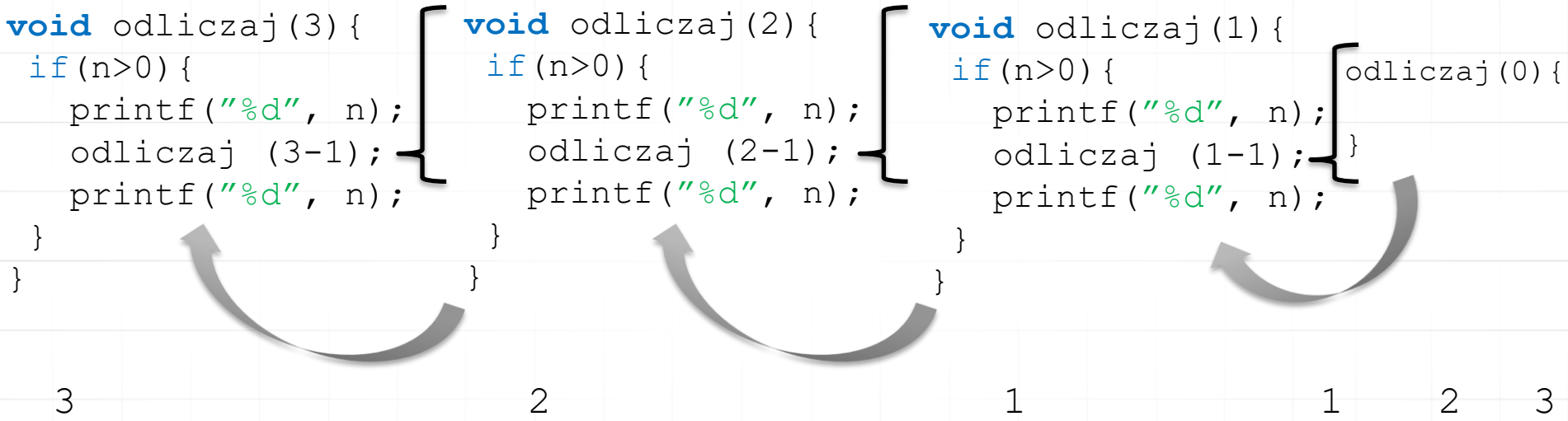
- Ważny jest również powrót sterowania w miejsce zaraz po tym gdzie funkcja była wywołana, ponieważ funkcja wywołująca mogła jeszcze się nie zakończyć.

```c
#include <stdio.h>

void odliczaj(int n)
{
    if (n > 0)
    {
        printf("Pozostalo do startu: %d sek\n", n);
        odliczaj(n-1);
        printf("Powrot nr %d\n", n);
        // ile razy zostanie wyświetlony?
    }
}

int main()
{
    odliczaj(5);
    return 0;
}
```

```
Pozostalo do startu: 5 sek
Pozostalo do startu: 4 sek
Pozostalo do startu: 3 sek
Pozostalo do startu: 2 sek
Pozostalo do startu: 1 sek
Powrot nr 1
Powrot nr 2
Powrot nr 3
Powrot nr 4
Powrot nr 5
```

Rekurencja – powrót z wywołania

- Ważny jest również powrót sterowania w miejsce zaraz po tym gdzie funkcja była wywołana, ponieważ funkcja wywołująca mogła jeszcze się nie zakończyć.



- Funkcja wywołująca zawsze czeka na zakończenie swoich funkcji potomnych. To oczekiwanie wiążę konieczność przechowywania w pamięci (w segmencie stos) danych związanych z ich obsługą.

Jakich danych ?

Rekurencja – powrót z wywołania

```
#include<iostream>
using namespace std;

void toBinary(int);

int main()
{
    int n;
    cout<<"Podaj liczbe calkowita dodatnia: ";
    cin>>n;
    cout<<"Jej postac binarna to: ";

    if(n==0)
        cout<<0;
    else
        toBinary(n);

    return 0;
}

void toBinary(int n){
    if(n==0)
        return;

    //zagnieżdżamy wywołanie rekurencjne
    toBinary(n/2);

    //powroty
    cout<<n%2;
}
```

```
Podaj liczbe naturalna calkowita: 30
Jej postac binarna to: 11110
```

Rekurencja – powrót z wywołania

Pytanie z rozmów rekrutacyjnych:

W jaki sposób wyświetlić liczby od 1 do 100 nie używając pętli?

Rekurencja – powrót z wywołania

Pytanie z rozmów rekrutacyjnych:

W jaki sposób wyświetlić liczby od 1 do 100 nie używając pętli?

```
void odliczaj(int n){
    if(n > 1)
        odliczaj(n-1);
    printf("%d", n);
}
```

Rekurencja - ciąg Fibonacciego

Zapis rekurencyjny

```
int fib(int n){
    if(n < 2)
        return n;
    else
        return fib(n-2) + fib(n-1);
}
```

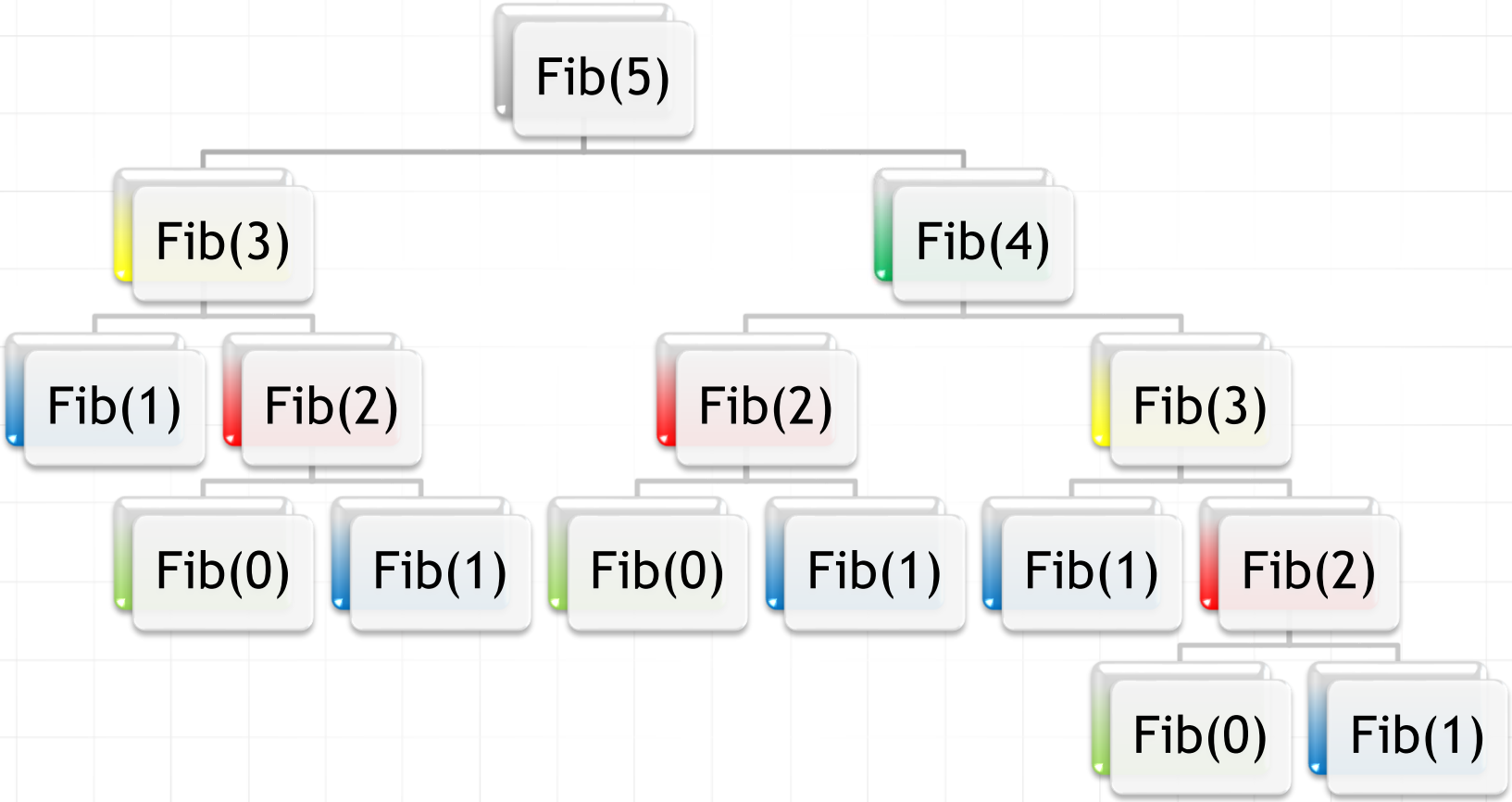
Krótki, szybki zapis, **ale !?**

Zapis iteracyjny

```
int fib(int n){
    if(n < 2)
        return n;
    else{
        int tmp;
        int current = 1;
        int last = 0;
        for(i = 2, i <= n, i++){
            tmp = current;
            current = current + last;
            last = tmp;
        }
    return current;
    }
}
```

Rekurencja - ciąg Fibonacciego

Ciąg Fibonacciego realizowany rekurencyjnie - analiza



Rekurencja - ciąg Fibonacciego

plik `main.cpp`:

```cpp
#include <iostream>
#include <chrono>
#include "fib.h"

int main()
{
    long long nnn=0;
    long long mmm=0;
    long long * mmmPtr = &mmm;
    long long * nnnPtr = &nnn;
    unsigned long long fiBi;

    auto begin = std::chrono::high_resolution_clock::now();
    fiBi = fibonacci(47, nnnPtr);
    auto end = std::chrono::high_resolution_clock::now();

    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count();

    std::cout << "N-ty Fibonacci - rekurencyjnie : " << fiBi << " - liczba wywolan " << nnn
    <<" - czas operacji " << duration << std ::endl << std ::endl;


    begin = std::chrono::high_resolution_clock::now ();
    fiBi = fibonacciI(47, mmmPtr);
    end = std::chrono::high_resolution_clock::now();

    duration = std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count();

    std::cout << "N-ty Fibonacci - iteracyjnie : "<< fiBi <<" - liczba iteracji "<< mmm
    << " - czas operacji "<< duration <<std ::endl<< std ::endl;

    return 0;
}
```

plik `fib.cpp`:

```cpp
#include <iostream>
#include "fib.h"


unsigned long long fibonacci(unsigned int n, long long *nnnPtr) {
    ++(*nnnPtr);
    if (n <= 1){
        return n;
        }
    else{
        return fibonacci(n - 2, nnnPtr) + fibonacci(n - 1, nnnPtr);
        }
}

unsigned long long fibonacciI(unsigned int n, long long * mmmPtr) {

    unsigned long long oldValue=0, nValue=1;

    for(int i=0; i<=n; ++i){
        ++(*mmmPtr);
        if(i==0){
            nValue = 0;
        }
        else if(i==1) {
            nValue=1;
        }
        else {
            nValue = nValue + oldValue;
            oldValue = nValue - oldValue;
        }
    }
    return nValue ;
}
```

```
N-ty Fibonacci - rekurencyjnie : 2971215073 - liczba wywolan 9615053951 - czas operacji 29644021

N-ty Fibonacci - iteracyjnie : 2971215073 - liczba iteracji 48 - czas operacji 0
```

Rekurencja vs iteracja

plik `main.cpp`:

```
#include <iostream>
#include <chrono>
#include "gcd.h"

int main()
{
    unsigned int k=0, l=0;
    unsigned int *ptrK=&k, *ptrL=&l;
    auto duration1=0, duration=0;
    auto begin1= std::chrono::high_resolution_clock::now ();
    auto end1= std::chrono::high_resolution_clock::now ();
    auto begin= std::chrono::high_resolution_clock::now ();
    auto end= std::chrono::high_resolution_clock::now ();

    for(int i=0; i<1000; ++i){
        begin1 = std::chrono::high_resolution_clock::now ();
        iterGcd(111111, 11, ptrL);
        end1 = std::chrono::high_resolution_clock::now ();

        duration1 += std::chrono::duration_cast<std::chrono::microseconds> (end1 - begin1).count();

        begin = std::chrono::high_resolution_clock::now ();
        recurGcd(111111, 11, ptrK);
        end = std::chrono::high_resolution_clock::now ();

        duration += std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count();
    }

    std::cout << " iterGcd(111111, 11): " << iterGcd(111111, 11, ptrL) <<  std::endl;
    std::cout << " Function iterGcd executed in: " << (float)duration1/1000 << " us" <<
    " - Liczba iteracji - " << l/1000 << std::endl << std::endl;

    std::cout << " recurGcd(111111, 11): " << recurGcd(111111, 11, ptrK) <<  std::endl;
    std::cout << " Function recurGcd executed in: " << (float)duration/1000 << " us" <<
    " - Liczba wywolan - " << k/1000 << std::endl << std::endl;

    return 0;
}
```

plik `defsGcd.cpp`:

```
#include "gcd.h"

long long recurGcd(long long x, long long y, unsigned int *k){
    ++(*k);
    if(x==y) return x;
    else if(x>y) return recurGcd(x-y, y, k);
    else return recurGcd(x, y-x, k);
}

long long iterGcd(long long x, long long y, unsigned int *l){
    do {
        ++(*l);
        if(x > y) {
            x = x - y;
        } else if (y > x) {
            y = y - x;
        }
    } while(x != y);
    return y;
}
```

```
iterGcd(111111, 11): 11
Function iterGcd executed in: 18.911 us - Liczba iteracji - 10110

recurGcd(111111, 11): 11
Function recurGcd executed in: 112.032 us - Liczba wywolan - 10111
```

Rekurencja ogonowa

Rekurencja ogonowa zachodzi wtedy gdy ostatnia operacja wykonywana przez funkcje jest operacją wywołania samej siebie lub zwrócenie ostatecznego wyniku.

Rekurencja ogonowa:

```
int funRecurTail(int x, int y=0){
    if(x==0)
        return y;
    else
        return funRecurTail(x-1, x+y);
}
```

Rekurencja ogólna:

```
int funRecur(int x){
    if(x==0)
        return 0;
    else
        return x + funRecur(x-1);
}
```

Rekurencja ogonowa

- Stosowana jest w celu zwiększenia wydajności algorytmów oraz zmniejszenia zajętości stosu przez adresy powrotu z funkcji.

Rekurencja ogonowa:

```
int funRecurTail(int x, int y=0){
    if(x==0)
        return y;
    else
        return funRecurTail(x-1, x+y);
}
```

Rekurencja ogólna:

```
int funRecur(int x){
    if(x==0)
        return 0;
    else
        return x + funRecur(x-1);
}
```

Podczas **rekurencji ogólnej** musimy pamiętać, aby odkładać adresy powrotu z funkcji na **sosie**, aby móc dokończyć wywołanie. W **przypadku rekurencji ogonowej,** podczas ostatniego wywołania rekurencyjnego, zwracamy ostateczny wynik i możemy powrócić do miejsca wywołania pierwotnego, pomijając powroty w łańcuchu wywołań rekurencyjnych.

Rekurencja ogonowa

- Porównanie dwóch algorytmów n-ty element ciągu Fibonacciego:

```
// rekurencja ogólna
unsigned long long fibRecur(unsigned int n, long long *nnnPtr) {
    ++(*nnnPtr);
    if (n <= 1)
        return n;
    else
        return fibRecur(n - 2, nnnPtr) + fibRecur(n - 1, nnnPtr);
}

// rekurencja ogonowa
unsigned long long fibRecurTail(unsigned int n, long long *lllPtr, unsigned long long a, unsigned long long b) {
    ++(*lllPtr);
    if (n == 0)
        return b;
    else
        return fibRecurTail(n - 1, lllPtr, a + b, a);
}
```

```
N-ty Fibonacci - rekurencyjnie : 2971215073 - liczba wywolan 9615053951 - czas operacji 30049790

N-ty Fibonacci - ogonowo rekurencyjnie : 2971215073 - liczba wywolan 48 - czas operacji 0
```