Wykład: 13

Rekurencja

Rekurencja - pojęcie

Rekurencja - pojęcie

Rekurencja (rekursja) – wywołanie funkcji przez nią samą wewnątrz ciała funkcji. .

Rozróżniamy dwa typy rekurencji:

- ✓ pośrednia – funkcja jest wywoływana przez inną funkcję, wywołaną (pośrednio lub bezpośrednio) przez samą funkcję
- ✓ bezpośrednia – funkcja wywołuje samą siebie bezpośrednio wewnątrz ciała funkcji (rozwiązanie spotykane częściej).

Rekurencja - pojęcie

Rekurencja pośrednia

```
void a(int i)
{
    if (i==0)
        return;
    else
        return b(i);
}

void b(int i)
{
    i--;
    return a(i)
}
```

Rekurencja - pojęcie

Rekurencja bezpośrednia

```
void a(int i)
{
    if (i==0)
        return;
    else
        return a(--i);
}
```

Rekurencja - pojęcie

Każda funkcja rekurencyjna musi posiadać warunek zatrzymania (warunek stopu) – czyli stan danych, dla którego nie dojdzie do ponownego wywołania funkcji.

W przeciwnym razie będzie się wywoływała do momentu przepełnienia stosu (stos jest obszarem pamięci służącym m.in. do przechowywania zmiennych lokalnych funkcji, parametrów wywołania i wartości zwracanej przez funkcję).

Rekurencja - pojęcie

Zastosowania rekurencji

Rekurencję możemy stosować w celu rozwiązania problemów, które wymagają przetworzenia danych, a następnie takiego samego przetworzenia wyników.

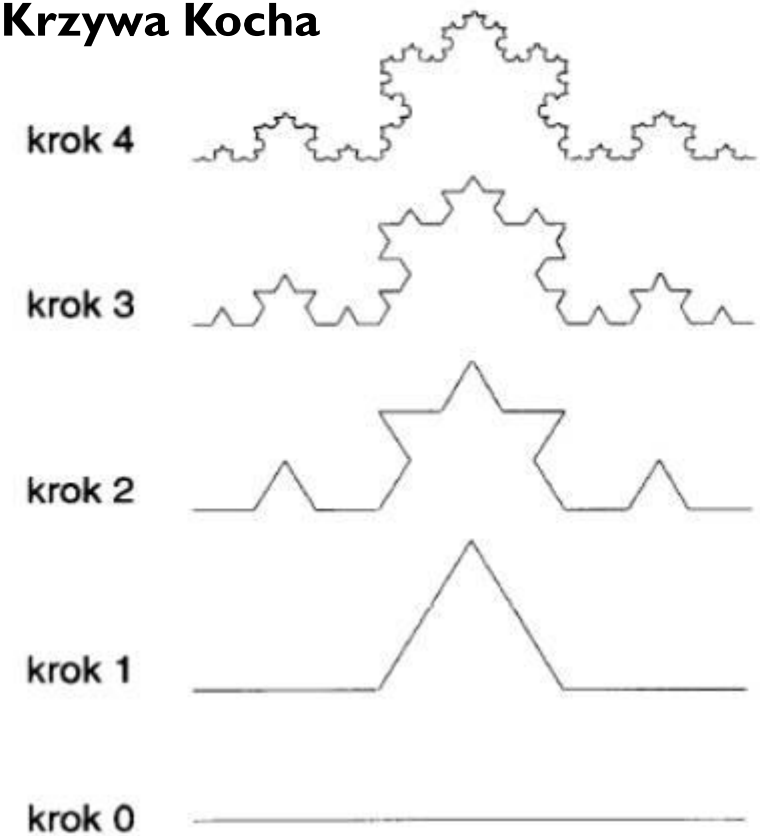
Zwykle problemy rekurencyjne możemy rozwiązać przy użyciu iteracji (takie rozwiązania są zwykle szybsze od rekurencyjnych odpowiedników).

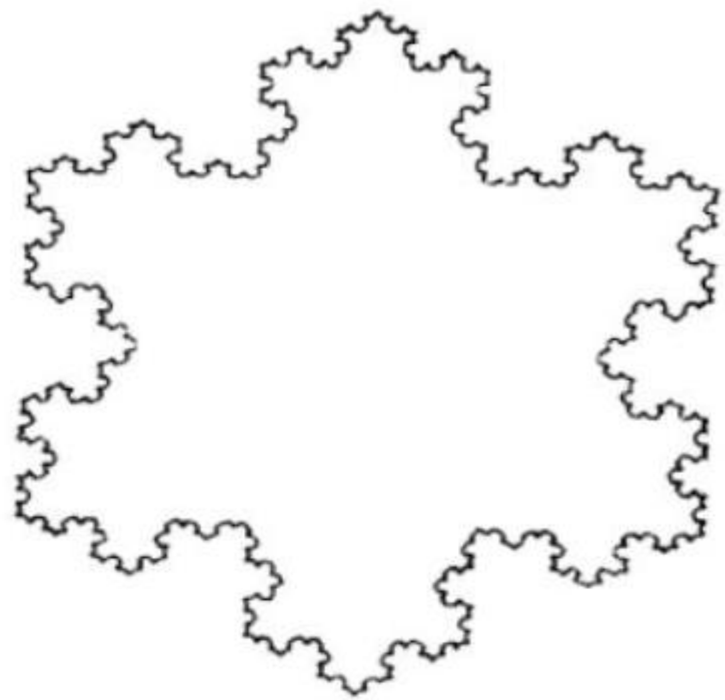
Zaletą rozwiązań rekurencyjnych jest prostota – przejrzystość kodu – niektóre problemy posiadają proste rozwiązania rekurencyjne – i bardzo złożone rozwiązania iteracyjne.

Rekurencja - pojęcie

Rekurencja – przykłady graficznej reprezentacji rekurencji

Krzywa Kocha





Rekurencja - pojęcie

Algorytm rekurencyjny

– to taki, który rozwiązuje problem, przez rozwiązanie pewnej liczby prostszych przypadków tego samego problemu.

- ✓ Algorytmy rekurencyjne implementujemy w C++ przy użyciu funkcji rekurencyjnych.
- ✓ Funkcje rekurencyjne odpowiadają definicjom rekurencyjnym funkcji matematycznych.

Rekurencja - pojęcie

Rekurencyjne obliczanie silni

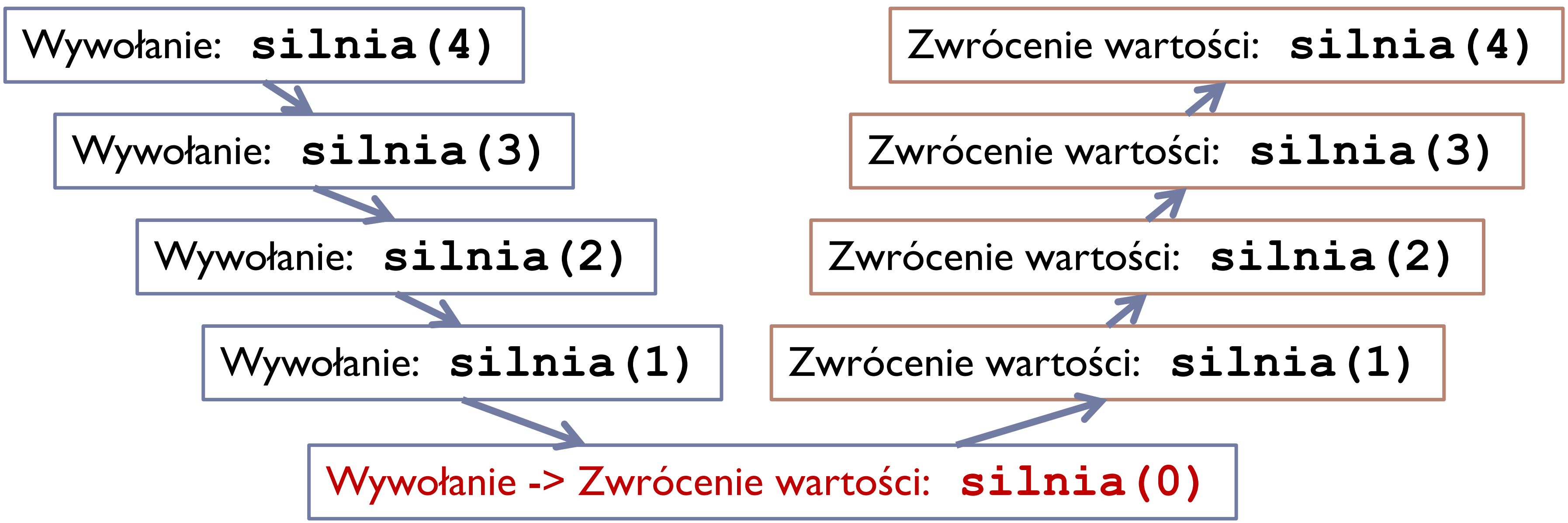
Definicja rekurencyjna silni ma postać:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n \cdot (n-1)! & \text{dla } n \geqslant 1 \end{cases}$$

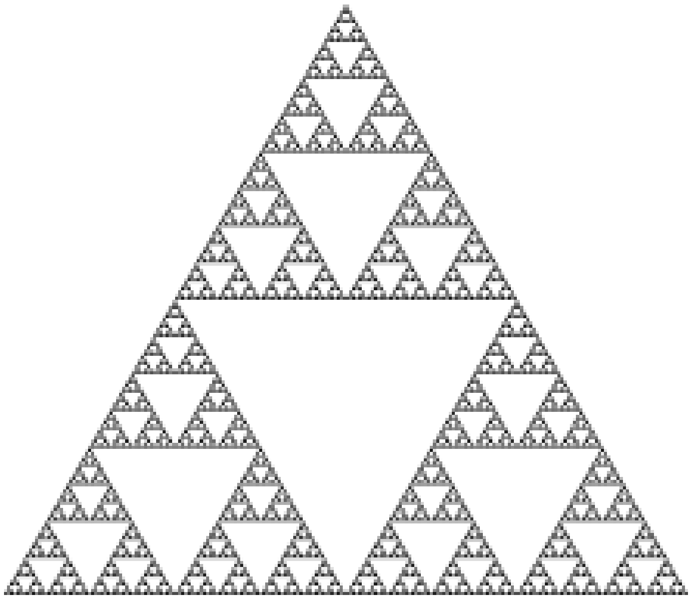
```
int silnia(int n)
{
    if(n == 0)
        return 1;
    else
        return n*silnia(n-1);
}
```

Rekurencja - pojęcie

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



Rekurencja - przykłady



Rekurencja - przykłady

Zadanie: funkcja która oblicza N-tą potęgę liczby X, gdzie N jest nieujemną liczbą całkowitą.

$X^N = X * X^{N-1}$ - to definicja rekurencyjna problemu – przedstawienie go jako zależności między rozwiązaniem problemu dla danego kroku przetwarzania danych i wynikiem z kroku poprzedniego Aby móc obliczyć X^{N-1}, musimy znaleźć X^{N-2}, X^{N-3} ... – czyli rozwiązanie problemu „mniejszego o jeden krok".

W rekurencji problem jest redukowany do prostszej postaci - aż do momentu, gdy możemy znaleźć rozwiązanie bez odwoływania się do postaci rekurencyjnej (w naszym przypadku [$X^N = X * X^{N-1}$]).

Taka postać problemu stanowi warunek zatrzymania – w naszym przypadku dane, dla których nie jest wymagane odwołanie się do definicji rekurencyjnej problemu to $N = 0$

Rekurencja - przykłady

W zapisie matematycznym:

$$X^N = \begin{cases} X * X^{N-1} & dla \quad N > 0 \\ 1 \quad dla \quad N = 0 \end{cases}$$

Rozwiązując problemy rekurencyjne stosujemy 3 kroki:

1. analiza problemu
2. identyfikacja warunku zatrzymania
3. zapisanie zależności rekurencyjnej

Rekurencja - przykłady

Funkcja rekurencyjna:

```
long double potega(long double X,int N)
{
      if (N==0)
            return 1;
      else
            return X*potega(X,N-1);
}
```

Wywołując funkcję potęga powodujemy, że będzie ona wywoływała samą siebie, do momentu, gdy parametr N wywołania nie będzie równy 0.

Funkcja nie może zwrócić wartości, do momentu, gdy wszystkie wywołania wewnętrzne nie zwrócą wartości.

Rekurencja – przykład 2

Napisz funkcję, która wypisze na ekranie n znaków „* "

Rozwiązanie rekurencyjne uzyskujemy w następujący sposób:

funkcja nie zwraca wartości, natomiast wypisuje znak *

1. dla n=0 nie wypisuje znaku i kończy działanie
2. dla n>0 wypisuje znak i rozwiązuje problem wypisania n-1 znaków.

```
void stars(int n)
{
    if (n==0)
        return;
    else
    {
        cout<<'*';
        return stars(n-1);
    }
}
```

Rekurencja – przykład 3

Napisz funkcję, która wyświetli liczby całkowite od n do 1

funkcja nie zwraca wartości, natomiast wypisuje liczbę całkowitą

1. dla n=1 wypisuje 1 i kończy działanie
2. dla n>1 wypisuje n i rozwiązuje problem wypisania liczb od n-1 do 1

```
void reverse(int n)
{
    if (n==1)
        cout<<n;
    else
    {
        cout<<n;
        return reverse(n-1);
    }
}
```

Rekurencja – przykład 4

Napisz rekurencyjną funkcję, znajdującą największy wspólny dzielnik 2 liczb całkowitych dodatnich.

Algorytm Euklidesa opiera się na spostrzeżeniu, że NWD dwóch liczb całkowitych m i n, gdzie m>n, jest równy NWD liczb y i x % y (reszta z dzielenia x przez y).

Liczba x dzieli zarówno liczbę m jak i n <=> x dzieli n i resztę z dzielenia m przez n, ponieważ m jest równe sumie m % n i wielokrotności n

```
int NWD(int m, int n)
{
    if (n==0)
        return m;
    else
        return NWD(n,m%n);
}
```

Rekurencja – przykład 5

Napisz funkcję rekurencyjną, która obliczy, wykorzystując algorytm Hornera wartość wielomianu postaci $a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n$ dla podanej jako argument wartości x i danego wektora współczynników $t[i]= a_i$

Ponieważ wielomian stopnia n możemy zapisać w następującej postaci:

$$a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n = (\dots((a_0 x+ a_1)*x+ a_2)*x+\dots + a_{n-1})x + a_n$$

Korzystając ze schematu Hornera, możemy zauważyć, że:

1. funkcja zwraca t[0] gdy i==0
2. funkcja zwraca sumę: t[i]+x*(wynik funkcji dla i-1) gdy i!=0

Rekurencja – przykład 5

```
long double horner(long double t[], long double x,int i)
{
    if (i==0)
        return t[0];
    else
        return x*horner(t,x,i-1)+t[i];
}
```

Rekurencja – przykład 5

```
long double horner(long double t[], long double x,int i)
{
    if (i==0)
        return t[0];
    else
        return x*horner(t,x,i-1)+t[i];
}
```

Wady rekurencji

Wady rekurencji

Jak obliczać ciąg Fibonacciego?

$F(n) = n$ jeśli $n < 2$

$F(n) = F(n\text{-}2)+F(n\text{-}1)$ jeśli $n \geq 2$

Rekurencja:

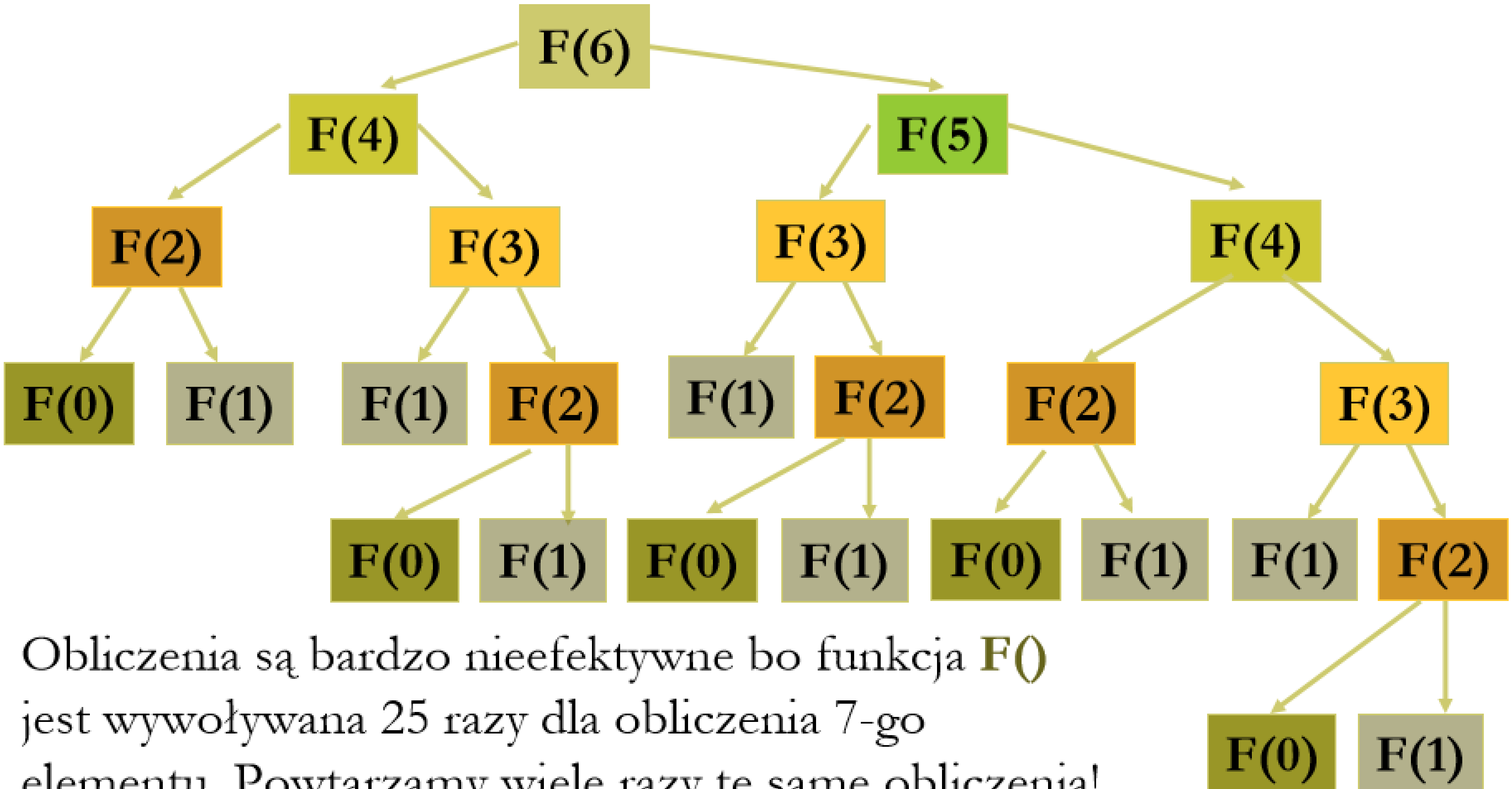
```
Fib (int n)
{
    if (n < 2)
        return n;
    else
        return  Fib(n-2) + Fib(n-1);
}
```

Iteracja:

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

http://th-www.if.uj.edu.pl/~erichter/dydaktyka/Dydaktyka2010/TPI-2010/TPI-wyklad-4-2010.pdf

Wady rekurencji



Obliczenia są bardzo nieefektywne bo funkcja **F()** jest wywoływana 25 razy dla obliczenia 7-go elementu. Powtarzamy wiele razy te same obliczenia!

http://th-www.if.uj.edu.pl/~erichter/dydaktyka/Dydaktyka2010/TPI-2010/TPI-wyklad-4-2010.pdf

Wady rekurencji

Porównacie złożoności obliczeniowe rekurencji i iteracji

n	liczba dodawań	Przypisania	
		Algorytm iteracyjny	Algorytm rekurencyjny
6	5	15	25
10	9	27	177
15	14	42	1973
20	19	57	21891
25	24	72	242785
30	29	87	2692537

http://th-www.if.uj.edu.pl/~erichter/dydaktyka/Dydaktyka2010/TPI-2010/TPI-wyklad-4-2010.pdf

Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Grębosz J. : ***Symfonia C++, Programowanie w języku C++ orientowane obiektowo***, Wydawnictwo Edition 2000.
- Jakubczyk K.: *Turbo Pascal i Borland C++ Przykłady*, Helion.

Warto zajrzeć także do:

- Sokół R. : ***Microsoft Visual Studio 2012 Programowanie w Ci C++,*** Helion.
- Kerninghan B. W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

- Grębosz J. : ***Pasja C++,*** Wydawnictwo Edition 2000.
- Meyers S.: ***język C++ bardziej efektywnie***, Wydawnictwo Naukowo Techniczne