

## Lekcja D1

**Temat: Rekurencja - obliczanie silni liczby naturalnej, ciąg Fibonacciego, ćwiczenia.**

### Cele lekcji:

1. Uczeń zna pojęcie rekurencji
2. Uczeń wie kiedy ciąg jest zdefiniowany rekurencyjnie
3. Uczeń wymienia klasyczne przykłady zastosowania rekurencji
4. Uczeń stosuje rekurencję do obliczania silni liczby naturalnej
5. Uczeń zna zasady dotyczące ciągu Fibonacciego.
6. Podaje definicję ciągu Fibonacciego.
7. Rozwiązuje zadania z zastosowaniem rekurencji.

### Przebieg lekcji:

1. Zapoznanie z celami lekcji
2. Ciąg liczb zdefiniowany rekurencyjnie

### Definicja

**Ciąg jest zdefiniowany rekurencyjnie**, jeśli spełnione są następujące warunki:

- a) określony jest pewien skończony zbiór początkowych wyrazów ciągu (na przykład kilka pierwszych lub pierwszy wyraz);
- b) kolejne wyrazy ciągu definiowane są za pomocą poprzednich elementów ciągu.

Warunek (a) definiuje wyrazy początkowe ciągu. Pozostałe wyrazy ciągu są określane za pomocą wzoru (b), czyli rekurencyjnie. Wzór definiujący taki ciąg nazywamy wzorem, równaniem lub zależnością rekurencyjną.

W algorytmice i programowaniu rekurencja występuje wówczas, gdy **procedura lub funkcja wywołuje samą siebie**, czyli uruchomienie procedury lub funkcji następuje wewnątrz niej samej.

Algorytmów rekurencyjnych raczej nie przedstawia się za pomocą schematów blokowych. Wymagałoby to oddzielnego konstruowania procedur czy funkcji rekurencyjnych w postaci schematów. Można jednak zastosować listę kroków, definiując funkcję i wskazując jej parametry.

### 3. Silnia liczby naturalnej – podstawy matematyczne

#### Definicja

**Silnia liczby naturalnej**  $n$  to iloczyn kolejnych liczb naturalnych  $1, 2, 3, \dots, n$ , co oznaczamy  $n!$ . Na przykład  $4! = 1 \cdot 2 \cdot 3 \cdot 4$ . Należy jednak zwrócić uwagę na to, że najmniejszą liczbą, dla której definiuje się silnię, jest  $0$ .  $0!$  równe jest  $1$ .

Podstawą wyznaczenia  $n!$  jest formalna definicja silni, która ma następującą postać:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ (n-1)! \cdot n & \text{dla } n > 0 \end{cases} \quad (1.12)$$

Powyższą definicję ilustruje przykład:

$$4! = 3! \cdot 4 = 2! \cdot 3 \cdot 4 = 1! \cdot 2 \cdot 3 \cdot 4 = 0! \cdot 1 \cdot 2 \cdot 3 \cdot 4 = 1 \cdot 1 \cdot 2 \cdot 3 \cdot 4$$

Skonstruujmy rekurencyjny **algorytm wyznaczający silnię liczby naturalnej**.

#### Specyfikacja:

**Dane:** Liczba naturalna:  $n \geq 0$ .

**Wynik:** Silnia liczby naturalnej  $n$ :  $n!$ .

Definicję rekurencyjną, na podstawie której tworzymy funkcję rekurencyjną wyznaczającą  $n!$ , można przedstawić następująco:

$$\begin{cases} a_0 = 1 \\ a_n = a_{n-1} \cdot n & \text{dla } n > 0 \end{cases} \quad (1.13)$$

### 4. Programowanie rekurencyjne – obliczanie silni.

#### Zadanie do wykonania:

- wyznaczanie wyrazów ciągu Fibonacciego – zapoznaj się z definicją (str. 39)

### 5. Definicja rekurencyjna ciągu Fibonacciego

**Ciąg Fibonacciego** – ciąg liczb naturalnych określony rekurencyjnie w sposób następujący:

Pierwszy wyraz jest równy  $0$ , drugi jest równy  $1$ , każdy następny jest sumą dwóch poprzednich.

Formalnie:

$$F_n := \begin{cases} 0 & \text{dla } n = 0, \\ 1 & \text{dla } n = 1, \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

Kolejne wyrazy tego ciągu nazywane są **liczbami** Fibonacciego. Zaliczanie zera do elementów ciągu Fibonacciego zależy od umowy – część autorów definiuje ciąg od  $F_1=F_2=1$

Pierwsze dwadzieścia wyrazów ciągu Fibonacciego to:

$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$	$F_{11}$	$F_{12}$	$F_{13}$	$F_{14}$	$F_{15}$	$F_{16}$	$F_{17}$	$F_{18}$	$F_{19}$
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181

Ciąg został omówiony w roku 1202 przez Leonarda z Pizy, zwanego Fibonaccim, w dziele *Liber abaci* jako rozwiązanie zadania o rozmnażaniu się królików. Nazwę „ciąg Fibonacciego” spopularyzował w XIX w. Édouard Lucas.

6. Programowanie:  
a) program bez rekurencji

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

void fibonaccii(int n)
{
    long long a = 0, b = 1;

    for(int i=0;i<n;i++)
    {
        cout<<b<<" ";
        b += a; //pod zmienną b przypisujemy wyraz następny czyli a+b
        a = b-a; //pod zmienną a przypisujemy wartość zmiennej b
    }
}

int main()
{
    int n;

    cout<<"Podaj ile chcesz wypisać wyrazów ciągu fibonacciego: ";
    cin>>n;

    fibonaccii(n);

    system("pause");
    return 0;
}
```

## b) Program rekurencyjny

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

int fib(int n)
{
    if(n<3)
        return 1;

    return fib(n-2)+fib(n-1);
}

int main()
{
    int n;

    cout<<"Podaj nr wyrazu ciagu: ";
    cin>>n;

    cout<<n<<" wyraz ciagu ma wartość "<<fib(n)<<endl;

    system("pause");
    return 0;
}
```

Przeanalizujemy powyższy algorytm dla

$$n = 5$$

:

$$\begin{aligned} \text{wynik} = fib(5) &= \underbrace{fib(4)} + \underbrace{fib(3)} = 5 \\ &= \underbrace{\underbrace{fib(3)} + fib(2)} + \underbrace{fib(1) + fib(2)} \\ &= \underbrace{\underbrace{fib(1) + fib(2)} + \underbrace{fib(2)}_1} + \underbrace{\underbrace{fib(1)}_1 + \underbrace{fib(2)}_1} \end{aligned}$$



7. Zadania do ćwiczenia – podręcznik str. 42.

**Zadanie 1.34.** Podaj specyfikację zadania i skonstruuj algorytmy rekurencyjne w postaci programów wyznaczające  $n$ -ty wyraz podanego ciągu liczbowego. Podaj definicje rekurencyjne ciągów:

- a) (4, 7, 10, 13, 16, 19, 22, 25, 28...),
- b) (2, 4, 8, 16, 32, 64, 128, 256, 512...),
- c) (0,2, -0,6, 1,8, -5,4, 16,2, -48,6, 145,8...),
- d) (-10, 5, -2,5, 1,25, -0,625, 0,3125...),
- e) (3, 5, 4, 6, 5, 7, 6, 8, 7...),
- f) (1,5, 1, 0,5, -0,5, -2, -4,5, -8,5, -15...),
- g) (-3, 1, -4, -5, 19, -96, -1825, 175199...),
- h) (-2, 2,5, 3, -5, 7,5, -4,5, -0,5, 8, -12,5...),
- i) (-1, 0, 0,5, 1,5, 1,5, 1, -0,5, -2, -3...),
- j) (0, 1, -1, 2, -2, 3, -3, 4, -4...).

**Zadanie 1.35.** Wygeneruj ciągi liczbowe (podając wartości co najmniej siedmiu kolejnych wyrazów) na podstawie podanej definicji rekurencyjnej. Podaj specyfikacje i skonstruuj rekurencyjne algorytmy w postaci programów, wyznaczające  $n$ -ty wyraz zdefiniowanego ciągu.

- a) 
$$\begin{cases} a_1 = 2 \\ a_n = 3a_{n-1} + \frac{1}{2} \quad \text{dla } n > 1 \end{cases}$$
- b) 
$$\begin{cases} a_1 = 0,5 \\ a_2 = 1 \\ a_n = 3a_{n-2} + a_{n-1} \quad \text{dla } n > 2 \end{cases}$$
- c) 
$$\begin{cases} a_1 = 2 \\ a_2 = -4 \\ a_n = a_{n-2} + 2a_{n-1} + 0,5 \quad \text{dla } n > 2 \end{cases}$$
- d) 
$$\begin{cases} a_1 = -1,5 \\ a_2 = 0 \\ a_3 = 1,5 \\ a_n = 2a_{n-3} + a_{n-2} - a_{n-1} \quad \text{dla } n > 3 \end{cases}$$