Iteracja i rekurencja

Pojęcie iteracji *(ang. iteration – powtarzanie)* można utożsamiać z obiegiem pętli. W każdym obiegu algorytm powtarza bowiem te same czynności.

Natomiast rekurencja *(ang. recursion – rekurencja, rekursja)* oznacza zdolność podprogramu (funkcji, procedury) do wywołania samego siebie. Muszą tu jednak być spełnione dwa warunki: jest określony jeden (lub więcej) początkowych wyrazów ciągu oraz kolejne wyrazy ciągu obliczane są za pomocą poprzednich. Na ogół też algorytmów rekurencyjnych – w przeciwieństwie do iteracyjnych – nie przedstawia się za pomocą schematów blokowych. Stosuje się za to opis słowny lub listę kroków.

Istnieje wiele algorytmów, które mogą być realizowane w sposób iteracyjny lub rekurencyjny, chociaż nie zawsze iterację można zastąpić rekurencją. Procedury lub funkcje rekurencyjne, które można zastąpić procedurami lub funkcjami iteracyjnymi, nazywamy pseudorekurencyjnymi.

Klasycznym przykładem porównania iteracji i rekurencji jest definicja silni. Definicja iteracyjna ma postać:

$$n! = \begin{cases} 1 & dla\ n = 0 \\ 1 \cdot 2 \cdot 3 \cdots \cdot n & dla\ n > 0 \end{cases}$$

Program w języku C++, realizujący ten algorytm, może mieć postać:

```
#include <iostream>

using namespace std;

long long int silnia (int n)
{
    long long int silnia = 1;
    for (int i = 1; i <= n; i++)
        silnia = silnia * i;
    return silnia;
}

int main()
{
    int n;
    cout << "Podaj n >= 0: ";
    cin >> n;
    cout << n << "! = " << silnia(n) << endl;
    return 0;
}
```

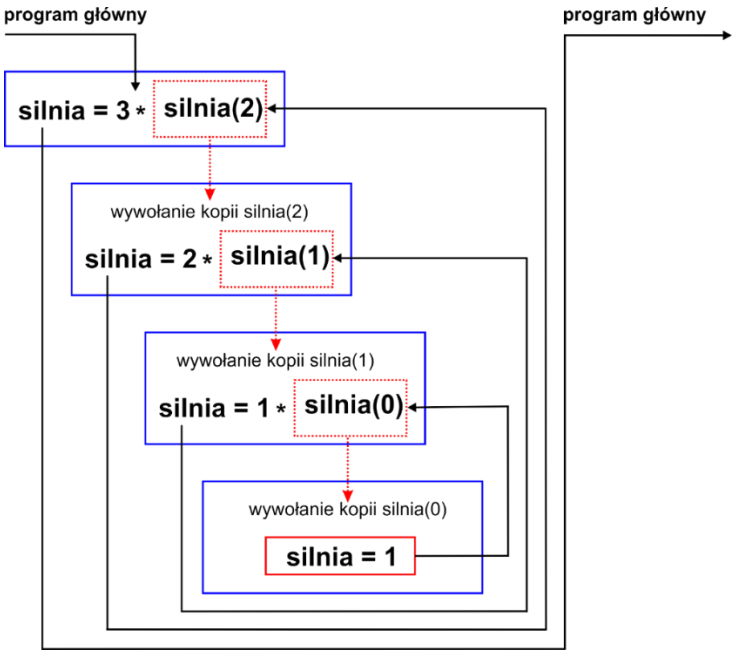
Definicja rekurencyjna silni jest natomiast następująca:

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

A sama funkcja, odpowiadająca tej definicji, może mieć wygląd:

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

Na tym tle łatwo wyjaśnić działanie rekurencji, korzystając z odpowiedniego rysunku do obliczenia $3!$.



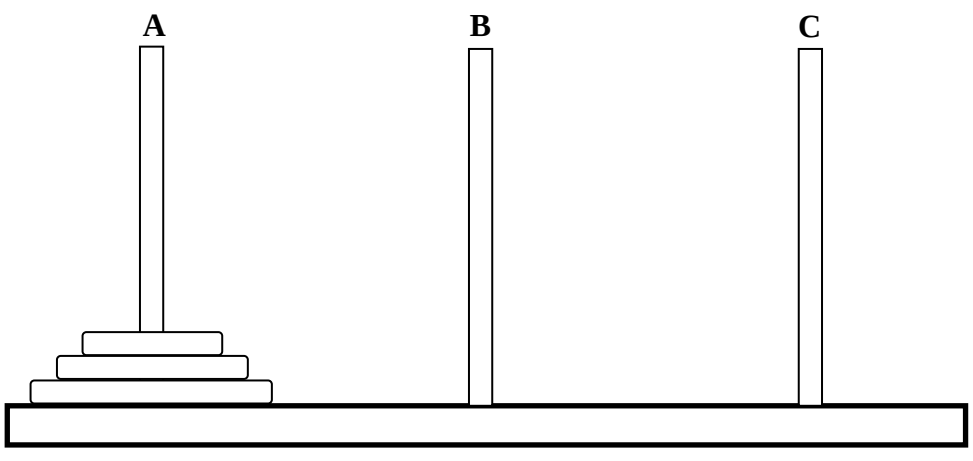
Wywołanie tej funkcji dla $n = 3$ ma następujący przebieg:

1. Program główny wywołuje funkcję $silnia(3)$. Następnie ma miejsce zapamiętanie tzw. adresu powrotnego do programu głównego, a dalej sprawdzenie, czy $n = 0$, ponieważ $0! = 1$ jest jedyną zdefiniowaną na stałe wartością. Czynności te odbywają się na tzw. pierwszym poziomie rekurencji.
2. Ponieważ $n > 0$, następuje chwilowe zawieszenie obliczeń i zapamiętanie adresu powrotnego do funkcji $silnia(3)$, gdyż to właśnie ona próbuje wykonać swoją „kopię” $silnia(2)$. Funkcja $silnia$ zeszła zatem w głąb – na drugi poziom rekurencji.
3. Wywołanie funkcji $silnia(2)$ powoduje te same skutki, co wywołanie funkcji $silnia(3)$. Podobnie rzecz się ma z wywołaniem funkcji $silnia(1)$, gdyż do jej wykonania też potrzebna jest „kopia” $silnia(0)$.
4. Dopiero na czwartym poziomie rekurencji następuje wywołanie wykonywalnej „kopii” funkcji $silnia(0)$, która z definicji ma przypisaną wartość równą 1.
5. Teraz możliwe jest wykonanie funkcji $silnia(1)$ oraz nadanie jej odpowiedniej wartości, a po niej kolejno do funkcji $silnia(2)$ oraz $silnia(3)$. Następuje więc powrót w górę – na coraz wyższe poziomy rekurencji.
6. Powrót na poziom pierwszy i wykonanie funkcji $silnia(3)$ powoduje w następstwie przekazanie sterowania do programu głównego. Funkcja $silnia(3)$ otrzymuje więc wartość 6, która zostanie wykorzystana w programie głównym.

Do wykonania rekurencyjnego podprogramu wykorzystuje się tzw. stos rekursji (ang. *run-time stack*), na którym układa się kolejne „kopie” wraz z adresami powrotu. Zdejmowanie ze stosu następuje w odwrotnej kolejności. Stos stanowi fragment pamięci operacyjnej na ogół rzędu 1MB. W informatyce taką strukturę nazywa się też kolejką typu LIFO *(ang. **L**ast **I**n **F**irst **O**ut – ostatni wszedł, pierwszy wyszedł)*. Poważnym błędem programistycznym jest doprowadzenie do przepełnienia stosu (ang. *stack overflow*), którego powodem może być zbyt duża liczba wywołania funkcji albo użycie zmiennej zbyt dużej objętości. Należy również uzupełnić, że nie wszystkie języki programowania (np. BASIC, Cobol, Fortran) umożliwiają korzystanie z rekurencji.

Ciekawym przykładem porównania iteracji i rekurencji jest też starożytny problem tzw. wież Hanoi *(ang. towers of Hanoi)*. Jest to łamigłówka złożona z trzech pionowych pałeczek *A*, *B*, *C* i różnej wielkości krążków, które nanizano na pierwszą z nich w ten sposób, że średnice krążków rosną ku podstawie.

Zadanie polega na przeniesieniu n krążków (w tym przykładzie trzech) z pałeczki A na pałeczkę C przy ograniczeniu, że w jednym kroku wolno przenieść tylko jeden krążek i nie wolno kłaść krążka większego na mniejszy. Pałeczka B pełni rolę pomocniczą.



Analizę problemu należy rozpocząć od spostrzeżenia, że gdyby $n = 1$, wówczas do rozwiązania wystarczy jeden krok: $A \longrightarrow C$.

Dla $n = 2$ konieczne są trzy kroki:

$$A \longrightarrow B;\ \ A \longrightarrow C;\ \ B \longrightarrow C.$$

Dla $n = 3$ wymaganych jest już siedem kroków:

$$A \longrightarrow C;\ \ A \longrightarrow B;\ \ C \longrightarrow B;\ \ A \longrightarrow C;\ \ B \longrightarrow A;\ \ B \longrightarrow C;\ \ A \longrightarrow C.$$

Łatwo zauważyć, że liczba kroków zmienia się zgodnie z zależnością $2^n - 1$. Legenda głosi, że w pewnym tybetańskim klasztorze mnisi buddyjscy od 3000 lat przenoszą 64 złote krążki w tempie jeden krążek na sekundę. Z chwilą przeniesienia ostatniego krążka nastąpi koniec świata. Nie trzeba się jednak tym zbytnio przejmować, albowiem $(2^{64} - 1)$ sekund odpowiada mniej więcej 585 mld lat (wiek Ziemi szacuje się na ok. 5 mld lat). Równanie rekurencyjne dla tego problemu jest następujące:

$$H(n) = 2H(n-1) + 1$$

przy warunku $H(1) = 1$.

Stosując metodę indukcji zupełnej można udowodnić, że $H(n) = 2^n - 1$.

Przechodząc do konstrukcji rekurencyjnego algorytmu Hanoi (dla $n > 1$) trzeba zauważyć, że zawiera on trzy etapy:

1. Przenieś $n-1$ górnych krążków z pałeczki A na pałeczkę B, używając C.
2. Przenieś największy krążek z pałeczki A na pałeczkę C.
3. Przenieś wszystkie krążki z pałeczki B na pałeczkę C używając A.

Zatem opis słowny algorytmu (n, A, C, B) może być następujący:

1. Jeśli $n = 1$, to $(1, A, C)$ i zakończ algorytm.
2. Jeśli $n > 1$, to zastosuj algorytm dla $(n-1, A, B, C)$.
3. Przenieś pozostały krążek $(A \longrightarrow C)$.
4. Zastosuj ten algorytm dla $(n-1, A, C, B)$.

Istnieje również iteracyjne rozwiązanie problemu wież Hanoi, dokładnie równoważne wersji rekurencyjnej. Wpierw jednak należy sobie wyobrazić, że pałeczki ustawione są na obwodzie koła.

Algorytm iteracyjny dla $n > 1$ działa następująco:

1. Poruszając się w kierunku przeciwnym do ruchu wskazówek zegara przenieś najmniejszy krążek z pałeczki, na której akurat spoczywa, na następny.
2. Wykonaj jedyne możliwe przeniesienie, nie dotyczące najmniejszego krążka.

Algorytm iteracyjny jest niezwykle łatwy do wykonania. Można udowodnić, że daje on dokładnie taką samą sekwencję przeniesień, co odmiana rekurencyjna.