Języki i paradygmaty programowania 1 – studia stacjonarne 2025/26
Lab 13. Rekurencja jako technika programowania - przykłady, wady i zalety.

1. Rekurencja to technika programowania, która polega na tym, że funkcja wywołuje samą siebie. Jest to ważne narzędzie, które może uprościć rozwiązanie wielu problemów, zwłaszcza tych, które mają strukturę hierarchiczną lub rekurencyjną, takich jak przeszukiwanie drzew, sortowanie, czy obliczanie wartości ciągów liczbowych.
   W niektórych językach programowania (Lisp, Scheme) pętle mogą być definiowane tylko poprzez wywołania rekurencyjne.
   Każda funkcja w programie C może być wywoływana rekurencyjnie. Liczba wywołań rekurencyjnych jest ograniczona do rozmiaru stosu. Za każdym razem, gdy funkcja jest wywoływana, nowa pamięć jest przydzielana dla parametrów oraz dla zmiennych **auto** i **register**, tak aby ich wartości w poprzednich, niedokończonych wywołaniach nie zostały nadpisane. Parametry są bezpośrednio dostępne tylko dla wystąpienia funkcji, w której są tworzone. Poprzednie parametry nie są bezpośrednio dostępne dla kolejnych wystąpień funkcji.

   Rekurencja jest prostą koncepcją zastosowania procedury w procedurze.
   Rekurencyjne procedury powinny mieć co najmniej dwa podstawowe elementy:
   - przypadek bazowy - kończy rekurencję dając procedurze wartość
   - krok rekurencyjny - daje wartość (pod względem wartości zastosowanej procedury) do innego argumentu

```c
#include <stdio.h>

//wersja rekurencyjna funkcji
int silnia(int n) {
    if (n == 0)
    {
        return 1;                   // przypadek bazowy
    }
    else
    {
        return n * silnia(n - 1);   // krok rekurencyjny
    }
}

int main() {
    int n;
    printf("Podaj liczbe: ");
    scanf_s("%d", &n);
    printf("Silnia %d wynosi %d\n", n, silnia(n));
    return 0;
}


//wersja iteracyjna funkcji
int silnia(int n) {
    int wynik = 1;
    for (int i = 1; i <= n; i++) {
        wynik *= i;
    }
    return wynik;
}
```

2. Zalety rekurencji*:
    a) Czytelność i zwięzłość: Rekurencyjne rozwiązania często są bardziej zwięzłe i czytelne niż ich iteracyjne odpowiedniki. Dzięki temu kod jest łatwiejszy do zrozumienia i utrzymania.
    b) Naturalne odwzorowanie problemów: Rekurencja jest idealna do rozwiązywania problemów, które mają naturalną strukturę rekurencyjną, takich jak przeszukiwanie drzew, sortowanie, czy obliczanie wartości ciągów liczbowych.
    c) Elegancja: Rekurencyjne rozwiązania mogą być bardziej eleganckie i intuicyjne, co ułatwia ich implementację i debugowanie.
    d) Modularność: Rekurencja pozwala na podział problemu na mniejsze, bardziej zarządzalne części, co ułatwia ich rozwiązanie.
    e) Łatwość implementacji: W wielu przypadkach rekurencyjne rozwiązania są łatwiejsze do zaimplementowania niż ich iteracyjne odpowiedniki, zwłaszcza gdy problem jest złożony.

3. Wady rekurencji
    a) Wydajność: Rekurencja może być mniej wydajna niż iteracja, zwłaszcza w przypadku problemów, które wymagają wielu wywołań rekurencyjnych. Każde wywołanie funkcji wiąże się z narzutem pamięciowym i czasowym.
    b) Przepełnienie stosu: Zbyt głębokie wywołania rekurencyjne mogą prowadzić do przepełnienia stosu, co skutkuje błędem StackOverflow. Jest to szczególnie problematyczne w językach programowania, które nie optymalizują rekurencji ogonowej.
    c) Trudności w debugowaniu: Rekurencyjne funkcje mogą być trudniejsze do debugowania niż ich iteracyjne odpowiedniki, zwłaszcza gdy problem leży głęboko w strukturze wywołań.
    d) Złożoność pamięciowa: Rekurencja może prowadzić do większego zużycia pamięci, ponieważ każde wywołanie funkcji zajmuje miejsce na stosie wywołań.
    e) Brak optymalizacji: W niektórych językach programowania rekurencja nie jest optymalizowana tak dobrze jak iteracja, co może prowadzić do mniej efektywnego kodu.

4. Rekurencja ogonowa.
   Rekurencja ogonowa to specjalny przypadek rekurencji, w którym wywołanie rekurencyjne jest ostatnią operacją wykonywaną przez funkcję. Dzięki temu kompilator może zoptymalizować kod, zamieniając rekurencję na iterację, co pozwala uniknąć przepełnienia stosu i zwiększa wydajność.

   Jeśli **proces iteracyjny może się wykonywać w stałej pamięci i opisany jest za pomocą procedury rekurencyjnej** (w Lispie jest to możliwe w odróżnieniu np. od Pascala, Ady czy C) to jeśli implementacja danego języka programowania ma tę cechę **to mówimy wówczas o rekursji ogonowej** (ang. tail recursion).

```c
#include <stdio.h>

// Funkcja pomocnicza do obliczania silni z rekurencją ogonową
int silnia_pomocnicza(int n, int wynik) {
    if (n == 0) {
        return wynik;
    }
    else {
        return silnia_pomocnicza(n - 1, n * wynik);
    }
}
// Funkcja główna do obliczania silni
int silnia(int n) {
    return silnia_pomocnicza(n, 1);
}

int main() {
    int n;
    printf("Podaj liczbe: ");
    scanf_s("%d", &n);
    printf("Silnia %d wynosi: %d\n", n, silnia(n));
    return 0;
}
```

5. Rekurencja i rekurencja ogonowa na przykładzie sumy n początkowych liczb naturalnych.

Proces rekurencyjny	Proces iteracyjny

`int suma(int n) {`
`  if (n ==0) {`
`  return 0;`
`  }`
`  else {`
`  return n + suma(n - 1);`
`  }`
`}`
	`int suma_pom(int n, int wynik) {`
	`  if (n == 0) {`
	`  return wynik;`
	`  }`
	`  else {`
	`  return suma_pom(n - 1, n + wynik);`
	`  }`
	`}`
	`int suma(int n) {`
	`  return suma_pom(n, 0);`
	`}`

suma(5) 5 + suma(4) 5 + (4 + suma(3)) 5 + (4 + (3 + suma(2))) 5 + (4 + (3 + (2 + suma(1)))) 5 + (4 + (3 + (2 + (1 + suma(0))))) 5 + (4 + (3 + (2 + (1 + 0)))) 5 + (4 + (3 + (2 + 1))) 5 + (4 + (3 + 3)) 5 + (4 + 6) 5 + 10 = 15	suma(5) suma_pom(5,0) suma_pom(4,5) (5-1, 5+0) suma_pom(3,9) (4-1, 4+5) suma_pom(2,12) (3-1, 3+4+5) suma_pom(1,14) (2-1, 2+3+4+5) suma_pom(0,15) (1-1, 1+2+3+4+5) = 15
• w odróżnieniu od pierwszego, „kształt” drugiego procesu się nie rozszerza ani nie zwęża. Jedyne co musimy śledzić w każdym kroku (dla każdego n) to: licznik (n) i suma (wynik); • proces iteracyjny to proces którego stan może być opisany przez ustaloną liczbę zmiennych stanu (ang. state variables), którego zmiany stanów możemy opisać ustaloną regułą mówiącą, jak zmieniają się wartości zmiennych stanu, oraz którego zakończenie (opcjonalnie) możemy opisać za pomocą warunku końcowego; • warunek rekurencji ogonowej: ostatnia wykonywana operacja funkcji musi być albo wywołaniem samej siebie, albo zwróceniem wyniku; • zastosowanie rekurencji ogonowej zapobiega przepełnieniu pamięci stosu;	

6. Napisz funkcję liczącą **rekurencyjnie** podany wyraz ciągu Fibonacciego.
   https://pl.wikipedia.org/wiki/Ciąg_Fibonacciego

7. Napisz funkcję liczącą **iteracyjnie** (rekurencja ogonowa) podany wyraz ciągu Fibonacciego.

8. Napisz funkcję liczącą dowolną naturalną potęgę dowolnej liczby całkowitej w wersji **rekurencyjnej** oraz **iteracyjnej**.

9. Napisz funkcję sprawdzającą czy podana liczba jest pierwsza w wersji **rekurencyjnej** oraz **iteracyjnej**.
   https://pl.wikipedia.org/wiki/Liczby_pierwsze

10. Napisz funkcję F przyjmującą jako argument dwie liczby a i b oraz funkcję f. Funkcja F powinna liczyć sumę:

$$F(a, b, f) = \sum_{i=a}^{b} f(i)$$

    Zaprezentuj działanie funkcji F w wersji rekurencyjnej i iteracyjnej.
    Przykładowo, dla pewnej funkcji f zwracającej liczbę 1, wywołanie F(2,5,f) powinno wywołać f 4 razy (z argumentem 2, 3, 4 i 5) i zwrócić liczbę 4 (suma czterech jedynek).

**Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.*