Podstawy Programowania
Zajęcia laboratoryjne 8
I rok Bioinformatyki Politechniki Poznańskiej

1 Rekurencja

Rekurencja jest techniką programowania wykorzystującą funkcje, które wywołują same siebie (ze zmienionym argumentem). Jest to alternatywny dla pętli sposób powtarzania pewnych czynności, gdzie kolejny etap jest podzadaniem poprzedniego. Rekurencja może być zamieniona na iteracje. Przy redukcji problemu wymagany jest wyraźnie określony warunek zakończenia (przypadek podstawowy). Rekurencja daje wrażenie istnienia wielu kopii tego samego algorytmu (aktywacji), jednak tylko jedna aktywacja jest aktywna w danej chwili. Typowym przykładem rekurencji jest definicja silni:

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

$0! = 1$ (warunek początkowy)
$1! = 1 = 1 \cdot 0!$
$2! = 2 \cdot 1 = 2 \cdot 1!$
$3! = 3 \cdot 2 \cdot 1 = 3 \cdot 2!$
$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 4 \cdot 3!$

Przykład rekurencji (kod 1 dostępny w materiałach na stronie). Zwróć uwagę, że funkcja wywołuje samą siebie ze zmienionym argumentem. Aby funkcja była zaimplementowana poprawnie należy bezwzględnie pamiętać o zaprojektowaniu warunku wyjścia z rekurencji.

```c
#include <stdio.h>

int silnia(int n);

int main()
{
    int n;

    printf("Podaj wartosc liczby n: ");
    scanf("%d",&n);
    printf("n! = %d", silnia(n));

    return 0;
}

int silnia(int n)
{
    if(n==0)
        return 1;
    else //wywolanie funkcji przez sama siebie ze zmienionym argumentem
        return n*silnia(n-1);
}
```

Wykorzystanie funkcji rekurencyjnych często ułatwia zaprojektowanie rozwiązania, przede wszystkim sprawia, że jest ono bardziej czytelne. Jednak takie rozwiązanie nie zawsze jest optymalne. Rekurencja bywa kosztowana tzn. im więcej argumentów ma funkcja i im większy jest ich rozmiar, tym mniej wywołań rekurencyjnych jest możliwych.

2 Zadania do zrealizowania na zajęciach

Zad 1. Napisz program, który obliczy n-tą liczbę ciągu Fibonacciego (zaimplementuj dwie funkcje: iteracyjną i rekurencyjną). Program prosi użytkownika o podanie wartości liczby n, gdzie n może być dowolną liczbą naturalną.

Zad 2. Napisz program, który obliczy potęgę liczby naturalnej, zarówno podstawa jak i wykładnik powinny być określone przez użytkownika (zaimplementuj dwie funkcje: iteracyjną i rekurencyjną).

Zad 3. Napisz program, który obliczy silnię liczby naturalnej n. Program prosi użytkownika o podanie wartości liczby n (zaimplementuj dwie funkcje: iteracyjną i rekurencyjną).

Zad 4. Napisz program, który obliczy największy wspólny dzielnik (NWD) dwóch liczb naturalnych. Program prosi użytkownika o podanie wartości tych liczb (zaimplementuj dwie funkcje: iteracyjną i rekurencyjną).

Zad 5. Napisz program, który obliczy najmniejszą wspólną wielokrotność (NWW) dwóch liczb naturalnych. Program prosi użytkownika o podanie wartości tych liczb (zaimplementuj dwie funkcje: iteracyjną i rekurencyjną).

Zad 6. Wykorzystaj poniższy kod (kod 2 dostępny w materiałach na stronie), aby sprawdzić czas działania dla funkcji iteracyjnej i rekurencyjnej z zadań 1-3. Przetestuj odpowiednio duże wartości aby zauważyć różnicę w czasie, która z funkcji działa szybciej?

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    double czas;
    clock_t start, koniec;

    start = clock(); //rozpoczecie odliczania czasu

    for (int i = 0; i < 10000; i++)
    {
        printf("%d ", i);
    }

    koniec = clock();  //zakonczenie odliczania czasu
    czas = (double)(koniec - start) / CLOCKS_PER_SEC; //zwrocenie czasu dzialania
    printf("\n\nCzas dzialania = %lf \n", czas);

    return 0;
}
```