

Podstawy Programowania

Zajęcia laboratoryjne 8

I rok Bioinformatyki Politechniki Poznańskiej

Rekurencja

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.

```
1 #include <stdio.h>
2
3 int silnia(int n);
4
5 int main()
6 {
7     int n;
8
9     printf("Podaj wartosc liczby n: ");
10    scanf("%d",&n);
11    printf("n! = %d", silnia(n));
12
13    return 0;
14 }
15
16 int silnia(int n)
17 {
18     if(n==0)
19         return 1;
20     else //wywołanie funkcji przez sama siebie ze zmienionym argumentem
21         return n*silnia(n-1);
22 }
```

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 zadanej liczby, zarówno podstawa jak i wykładnik powinny być liczbami naturalnymi określonymi 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 całkowitych dodatnich (> 0). Program prosi użytkownika o podanie wartości tych liczb (zaimplementuj dwie funkcje: iteracyjną i rekurencyjną).

Zad 5. Wykorzystaj kod z zadania 4 i przerób go tak aby obliczał najmniejszą wspólną wielokrotność (NWW) dwóch liczb całkowitych dodatnich (> 0). 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?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main()
6 {
7     double czas;
8     clock_t start, koniec;
9
10    start = clock(); //rozpoczenie odliczania czasu
11
12    for (int i = 0; i < 10000; i++)
13    {
14        printf("%d ", i);
15    }
16
17    koniec = clock(); //zakonczenie odliczania czasu
18    czas = (double)(koniec - start) / CLOCKS_PER_SEC; //zwrocenie czasu dzialania
19    printf("\n\nCzas dzialania = %lf \n", czas);
20
21    return 0;
22 }
```

Aby sprawdzić jakie są zakresy dla poszczególnych typów wejdź w link i znajdź tabelę "Main types", przy użyciu typów zwróć uwagę na ich identyfikatory. Wykroczenie poza zakres używanego typu jest częstym błędem.