Podstawy programowania w języku C

```c
#include <math.h>
#include <stdio.h>
#include <conio.h>


double X(double t) {
    return (1.0 + sin(2.0 * M_PI * 10000.0 * t) + cos(2.0 * M_PI * 10000.0 * t));

    struct complexx {
        double Re;
        double Im;
    };

    struct complexx cmplx(double A, souble B) {
        struct complexx result;
        result.Re = A;
        result.Im = B;
        return (result);
    }


    struct complex cMult(struct complexx wA, struct complexx B) {

        struct complexx result;

        result.Re = A.Re * B.Im + A.Im * B.Re;
```

Rekurencja

Jest to odwoływanie się np. funkcji lub definicji do samej siebie. Rekurencja jest podstawową techniką wykorzystywaną w funkcyjnych językach programowania. Należy zachować ostrożność przy używaniu. Ryzyko istnieje szczególnie przy przetwarzaniu dużej ilości głęboko zagnieżdżonych danych.

Lab

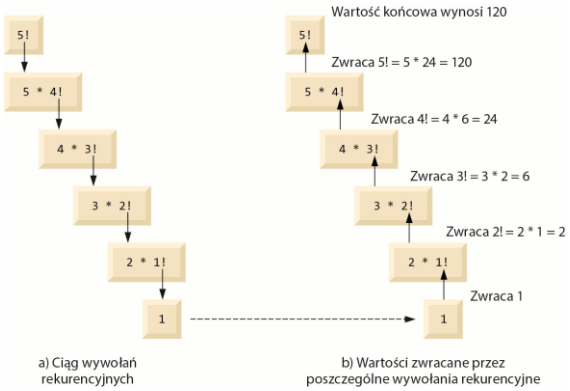
Uniwersytet Rzeszowski
ur.edu.pl

Wykonał: Dawid Kosior
kosiordawid@gmail.com

Funkcje rekurencyjne

Język C ma możliwość tworzenia tzw. funkcji rekurencyjnych. Jest to funkcja, która w swojej własnej definicji (ciele) wywołuje samą siebie. Klasycznym przykładem może tu być silnia. Funkcja rekurencyjna, która oblicza silnię może mieć postać:

```c
int silnia(int liczba) {
    int sil;
    if (liczba < 0) return 0; /* wywołanie
jest bezsensowne, zwracamy 0 */
    if (liczba == 0 || liczba == 1) return 1;
    sil = liczba * silnia(liczba - 1);
    return sil;
}
```

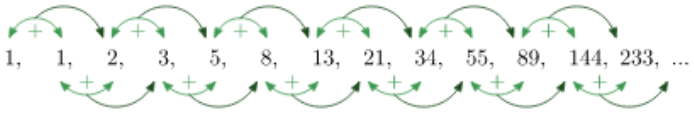


Musimy być ostrożni przy funkcjach rekurencyjnych, gdyż łatwo za ich pomocą utworzyć funkcję, która będzie sama siebie wywoływała w nieskończoność, a co za tym idzie będzie zawieszała program. Tutaj pierwszymi instrukcjami ***if*** ustalamy "warunki stopu", gdzie kończy się wywoływanie funkcji przez samą siebie, a następnie określamy, jak funkcja będzie wywoływać samą siebie (odjęcie jedynki od argumentu, co do którego wiemy, że jest dodatni, gwarantuje, że dojdziemy do warunku stopu w skończonej liczbie kroków).
Warto też zauważyć, że funkcje rekurencyjne czasami mogą być znacznie wolniejsze niż podejście nierekurencyjne (iteracyjne, przy użyciu pętli). Flagowym przykładem może tu być funkcja obliczająca wyrazy ciągu Fibonacciego opisany poniżej.

Wzór rekurencyjny ciągu Fibonacciego:

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_{n+2} = F_{n+1} + F_n \end{cases}$$

Kolejne liczby Fibonacciego tworzone są przez dodanie do siebie dwóch poprzednich wyrazów ciągu.




```c
#include <stdio.h>

unsigned count;

unsigned fib_rec(unsigned n) {
    ++count;
    return n < 2 ? n : (fib_rec(n - 2) + fib_rec(n - 1));
}

unsigned fib_it(unsigned n) {
    unsigned a = 0, b = 0, c = 1;
    ++count;
    if (!n) return 0;
    while (--n) {
        ++count;
        a = b;
        b = c;
        c = a + b;
    }
    return c;
}

int main(void) {
    unsigned n, result;
    printf("Ktory element ciagu Fibonacciego obliczyc? ");
    while (scanf("%d", &n) == 1) {
        count = 0;
        result = fib_rec(n);
        printf("fib_rec(%3u) = %6u (wywolan: %5u)\n", n, result,
        count);
        count = 0;
        result = fib_it(n);
        printf("fib_it (%3u) = %6u (wywolan: %5u)\n", n, result,
        count);
    }
    return 0;
}
```

W tym przypadku funkcja rekurencyjna jest bardzo nieefektywna.

Zadania do wykonania

1. Przetestuj zamieszczone wyżej fragmenty kodu i sprawdź ich działanie.
2. Napisz program wyznaczający n-ty wyraz ciągu zdefiniowanego przez następujący wzór rekurencyjny:
$$a_n = \begin{cases} -1 & dla\ n = 1 \\ -a_{n-1} \cdot n - 2 & dla\ n > 1 \end{cases}$$
3. Napisz program, który wyznaczy sumę cyfr liczby naturalnej podanej przez użytkownika. Rozwiąż zadanie metodą rekurencyjną.

   Podpowiedź:

   Aby wyłuskać cyfrę jedności danej liczby należy wykonać operację: ***cyfra=n mod 10***

   gdzie ***mod*** oznacza resztę z dzielenia. Następnym krokiem jest skrócenie liczby o jedną cyfrę wykonując operację: ***n=n div 10***

   gdzie ***div*** oznacza dzielenie całkowite. Powtarzamy te operacje do momentu otrzymania liczby 0.

   Rekurencyjnie wygląda to następująco. Przeanalizujmy przykład dla liczby 123.

$$sumacyfr(123) = \overbrace{123\ mod\ 10}^{3} + \underbrace{sumacyfr(\overbrace{123\ div\ 10}^{12})}_{2} = \overbrace{12\ mod\ 10}^{2} + \underbrace{sumacyfr(\overbrace{12\ div\ 10}^{1})}_{1} = \overbrace{1\ mod\ 10}^{1} + \underbrace{sumacyfr(\overbrace{1\ div\ 10}^{0})}_{0} = 3 + 2 + 1 + 0 = 6$$

4. Napisz program, który zapisze podaną liczbę dziesiętną naturalną w systemie binarnym. Rozwiąż zadany problem rekurencyjnie.
5. Jaś i Małgosia zabawiali się w następującą grę. Jaś wymyślał liczbę z zakresu od 0 do 10000. Zadaniem Małgosi było odgadnąć tę liczbę zadając Jasiowi pytania, na które mógł odpowiadać jedynie TAK lub NIE. Napisz program symulujący grę Jasia i Małgosi. Wyznacz najmniejszą liczbę pytań, po której Małgosia odgadnie szukaną liczbę. Rozwiąż zadany problem rekurencyjnie.

   Podpowiedź:

   Załóżmy, że Małgosia ma pewność, że szukana liczba x znajduje się w przedziale [a,b]. Po uzyskaniu odpowiedzi na pytanie, czy x jest większa niż $\lfloor(a+b+1)/2\rfloor$, (zapis liczby w takich nawiasach $\lfloor 2,8 \rfloor$, oznacza zaokrąglenie wartości w dół w wyniku czego otrzymamy 2) Małgosia zawęzi zbiór poszukiwań o połowę, szukając później liczby x w przedziale $[a, \lfloor(a+b-1)/2\rfloor]$, albo w przedziale $[\lfloor(a+b+1)/2\rfloor, b]$.