Metody Numeryczne, WFiIS AGH: Lab01
Krzysztof Szyszkiewicz

Ćwiczenie 1.
Dany jest ciąg liczb rzeczywistych $(x_n)_{n=0}^{\infty}$ określony następująco:

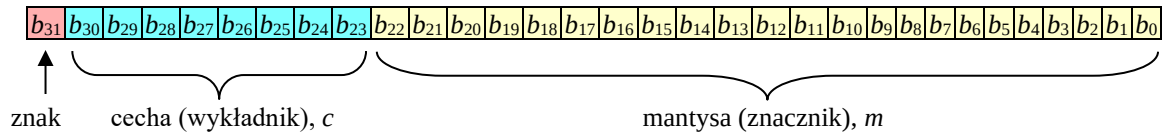
$$\begin{cases} x_0 = 1, \;\; x_1 = \dfrac{1}{3}, \\ x_{n+1} = \dfrac{13}{3} x_n - \dfrac{4}{3} x_{n-1} \quad \text{dla} \;\; n \geq 1. \end{cases}$$

Można pokazać, że tak zdefiniowany ciąg jest równy $x_n = 1/3^n$.
Napisać program, który będzie wypisywał sto pierwszych wyrazów tego ciągu w oparciu o powyższą definicję. Czy tak obliczone wyrazy x'_n przybliżają dobrze wartości dokładne ciagu $x_n = 1/3^n$?

Przypomnieć sobie z wykładu jak wygląda reprezentacja liczb zmiennopozycyjnych (standard IEEE 754), dla którego punktem wyjścia jest następujący rozkład dowolnej liczby rzeczywistej $x \in \mathbb{R}$ (tu przyjęto jako podstawę reprezentacji liczbę 2):

$$x = \pm m \cdot 2^c, \tag{1}$$

gdzie $\frac{1}{2} \leq m < 1$ to tzw. *mantysa*, a c to *cecha*, która jest liczbą całkowitą, $c \in \mathbb{Z}$. Spotyka się też inny wariant, w którym mantysa spełnia nierównosci $1 \leq m < 2$. W standardzie IEEE 754 dla liczby *pojedynczej precyzji* (w normie nazywa się on *binary32*) długość reprezentacji (liczba bitów) wynosi 32. Bity dzielimy na **znak**, **cechę** i **mantysę** jak poniżej, gdzie pokazano bity:



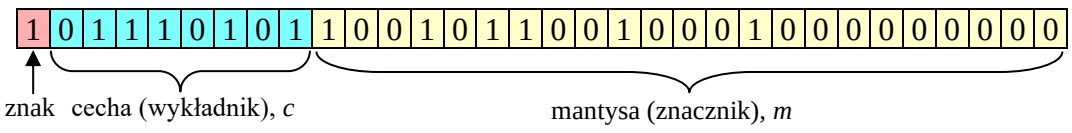
Teraz w oparciu o rozkład zawarty we wzorze (1) przypisujemy do powyższego ciągu bitów wartość liczby rzeczywistej (a tak na prawdę – liczby wymiernej) wg ściśle określnych reguł. Trzeba tylko jeszcze ustalić kilka szczegółów. Poniższy przykład to wyjaśnia.

Ćwiczenie 2. Złóżmy, że w pamięci jest zapamiętany ciąg bitów

10111010110010110010001000000000

a obszar pamięci ma być interpretowany jako zmienna typu **`float`** (zakładamy, że ten typ implementuje standard IEEE 754 dla liczb zmiennopozycyjnej pojedynczej precyzji). Jaką wartość liczbową w zapisie dziesiętnym powyższy ciąg bitów przedstawia?

Ciąg dzielimy trzy fragmenty:



gdzie pierwszy bit z lewej strony (na czerwonym tle) oznacza znak: „+" gdy $b_{31} = 0$, „–" gdy $b_{31} = 1$. Należy jeszcze dokładnie ustalić jaką liczbę wymierną reprezentuje powyższy ciąg bitów. Przede wszystkim gdybyśmy cechę interpretowali jako zapis dwójkowy liczby całkowitej bez znaku, to dla 8 bitów mielibyśmy zbiór wartości {0,1,…, 255}. Dla cechy z przykładu byłaby to wartość

$$c = 2^0 + 2^2 + 2^4 + 2^5 + 2^6 = 1 + 4 + 32 + 64 = 117. \tag{2}$$

Problem z tym sposobem obliczania cechy jest taki, że wtedy uzyskamy z wzoru (1) wartości (pomijając znak całej liczby x) z zakresu: $[m \cdot 2^0; m \cdot 2^{255}]$, ale $\frac{1}{2} \le m < 1$ (lub $1 \le m < 2$), więc nie uzyskalibyśmy liczb z przedziału $0 \le x \le 1/2$ (lub $0 \le x \le 1$). Dlatego wprowadza się w wykładniku tzw. stałe *odchylenie* (ang. *bias*) B, które po prostu odejmujemy od wartości wykładnika c. W przypadku formatu pojedynczej precyzji $B = 127$. Najwygodniej jest tę wartość wprowadzić do wzoru (1), a cechę c obliczać jak dla zwykłej liczby całkowitej nieujemnej. Możemy więc dla liczb pojedynczej precyzji wg normy IEEE 754 zapisać:

$$x = \pm m \cdot 2^{c-127}. \tag{3}$$

Jeżeli chodzi o mantysę, to przypisujemy kolejnym bitom (licząc od b_{22}) wagi 1/2, $1/2^2$, $1/2^3$, …, $1/2^{23}$. (Dla wariantu mantysy $1 \le m < 2$ dodajemy jeszcze 1). Tak więc dla podanego przykładu mamy

$$m = 1 + \left(\frac{1}{2} + \frac{1}{2^4} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^{10}} + \frac{1}{2^{14}} \right) = \frac{26001}{16384}.$$

Ostatecznie wartość liczbowa przypisana do ciągu bitów z przykładu wynosi

$$x = (-1) \cdot \left(\frac{1}{2} + \frac{1}{2^4} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^{10}} + \frac{1}{2^{14}} \right) \cdot 2^{117-127} = -\frac{26001}{16384} \cdot 2^{-10}$$

$$= -\frac{26001}{16777216} = -0.00154978036880493164062 5.$$

Ćwiczenie 3.

Napisać i przetestować program, który będzie wyświetlał bitową reprezentację zmiennych typu **`float`**. Poniżej jest przykładowa realizacja – mogą być inne. Nie mam pewności czy jest do końca poprawny – proszę pomyśleć ewentualnie nad własną wersją.

```
#include <stdio.h>

void showbits(unsigned long x, int n) {
  if (--n) showbits(x >> 1, n);
     putchar("01"[x&1]);
}

int main(void) {
     long unsigned lu;
     float x = 37.0681;

     lu = *(long*)&x;
     printf("%f -> ", x);
     showbits(lu, 32);

     printf("\n");
     return 0;
}
```

Ćwiczenie 4.
Napisać program w języku C/C++, który wczyta dwie macierze A i B rozmiaru 5 x 5 z plików tekstowych *A.txt* oraz *B.txt*, a następnie wykona mnożenie tych macierzy $C = A \cdot B$, a wynik zapisze w macierzy C, wypisze na ekranie i do pliku *C.txt*. Do implementacji macierzy w języku C/C++ wybrać zwykłe tablice statyczne dwuwymiarowe. Format zapisu macierzy w pliku txt: wiersze macierzy w kolejnych wierszach tekstu, liczby oddzielane spacjami.

Ćwiczenie 5.
Napisać i przetestować poniższy program, który testuje bibliotekę GSL:

```
#include <stdio.h>
#include "/usr/include/gsl/gsl_math.h"
#include "/usr/include/gsl/gsl_linalg.h"

int main() {
    int n, m;
    int i, j;

    n = 3; m = 3;
    gsl_matrix *A = gsl_matrix_calloc(n,m);
    gsl_vector *b = gsl_vector_calloc(n);


    for (i=0; i < n; i++) {
        for (j=0; j < m; j++)
            gsl_matrix_set(A,i,j,1.0/(1.0+i+j));
        gsl_vector_set(b,i,2.0*i);
    }

    printf("\n***** Matrix A *****\n");
    for (i=0; i < n; i++) {
        for (j=0; j < m; j++) printf("%f, ", gsl_matrix_get(A,i,j));
        printf("\n");
    }

    printf("\n***** Vector b *****\n");
    for (i=0; i < n; i++) printf("%f,\n", gsl_vector_get(b,i));
}
```