Celem ćwiczenia jest zapoznanie się z bibliotekami, kompilacją warunkową oraz instrukcją `switch`.

5.1 Biblioteka do obliczeń na liczbach wymiernych

Celem ćwiczenia jest stworzenie linkowalnej biblioteki wraz z nagłówkami, pozwalającą na wykonywanie operacji na liczbach wymiernych. Wybór struktury danych do przechowywania liczb wymiernych zostawiam do indywidualnej decyzji. Przykłady to

i) struktura zawierająca dwa pola: licznik i mianownik, typu `int`;

ii) struktura zawierająca trzy pola: licznik i mianownik, typu `unsigned int` oraz znak ($\pm$);

iii) wektor 2-elementowy;

iv*) jako liczby całkowite Gaussa, gdzie część rzeczywista i urojona (`<complex.h>`) będą przechowywać licznik i mianownik;

v*) jako ułamek łańcuchowy;

vi**) jako drzewo Sterna-Brocota

vi†) jako n-ty ułamek $\frac{f(n+1)}{f(n)}$, gdzie $f(n)$ jest funkcją `fusc` Dijkstry

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f\left(\frac{n}{2}\right) & \text{if } n \text{ is even} \\ f\left(\frac{n-1}{2}\right) + f\left(\frac{n+1}{2}\right) & \text{if } n \text{ is odd.} \end{cases}$$

Następnie napisać funkcje wykonujące dodawanie `add` , odejmowanie `sub`, mnożenie `mul`, dzielenie `dvd`, odwrotność `inv`, zmianę znaku `neg` podnoszenie do kwadratu `sqr` oraz zwracające licznik `num` i mianownik `den`. Funkcje powinny skracać ułamek, czyli nie $\frac{2}{4}$ tylko $\frac{1}{2}$ itd., chyba że użyta reprezentacja tego nie wymaga. Liczby całkowite traktujemy jako ułamek $\frac{n}{1}$. W przypadku skracania przy pomocy funkcji `gcd`, dodać flagi preprocesora pozwalające warunkowo wybrać wariant funkcji podczas kompilacji, powiedzmy

```
#define GCD_RECURSIVE 1
```

Następnie utworzyć bibliotekę `librational.a` oraz `librational.so`, która wraz z nagłówkiem `rational.h` pozwoli na użycie powyższych funkcji i struktur w kodzie `rat.c` (przykład używa wariantu i) w następujący sposób:

```
#include "rational.h"

int main()
{
    rational r,p,q;

    r.licznik   = 1;
    r.mianownik = 137;

    p.licznik   = 2;
    p.mianownik = 9;

    q = mul(r,p);
    q = sqr(q);

    return 0;
}
```

Kompilacja powyższego programu powinna działać w poniższy sposób:
`gcc rat.c -lrational -o rat`

5.2 Interpreter działań na liczbach wymiernych

Korzystając z powyższego kodu, napisać program wykonujący obliczenia z jego użyciem, w stylu kalkulatora RPN. Przykładowy plik wejściowy wykonujący mnożenie $\frac{2}{3} \times \frac{3}{4}$

```
2/3
3/4
MUL
```

zwraca 1/2, a działanie

$$\left[\left(\frac{1}{137}\right)^2 - \frac{2}{9}\right]^{-1}$$

```
1/137
SQR
2/9
SUB
INV
```

-168921/37529.

Nazwy operacji powinny być zgodne z biblioteką z Zad. 1. Program powinien sprawdzać poprawność kodu wejściowego, np: próba wywołania

```
3/4
MUL
```

czy

```
0/1
INV
```

powinna dać odpowiedni komunikat o błędzie[1].

Wskazówka Ze względu na dużą liczbę wariantów wygodne jest użycie instrukcji `switch`.

5.3 Przybliżenia $\sqrt{2}$, π, e oraz innych liczb niewymiernych liczbami wymiernymi

Wygenerować listę 1024 coraz dokładniejszych przybliżeń wymiernych liczby niewymiernej q.

Wskazówka:

W przypadku użycia typowej reprezentacji liczb wymiernych do przeskanowania wszystkich możliwych par licznik/mianownik można wykorzystać pętle z programów dotyczących *Zad. 1/2 oraz zadania 1.3 Krzywa wypełniająca kwadrat (pkt. 3,4)*. Zastanowić się czy i jak wygenerowana lista zależy od użytej reprezentacji.

[1] Alternatywą w ostatnim przypadku jest wprowadzenie symbolu ∞ i wyrzucenie z kodu operacji nieoznaczonych `DVD` (∞/∞) oraz `SUB` ($\infty - \infty$).