# 4. feladat
# Racionális szám osztály

Erdősi Lili
DMVR5F

Korszerű számítástechnikai módszerek beadandó feladat

# Bevezetés

A feladat egy *Rational* osztály létrehozása volt, mely két egész szám hányadosaként (törtszámként) tárol el egy racionális számot. Az osztályon belül elvégezhetőek az alapműveletek (összeadás, kivonás, szorzás, osztás), illetve a hatványozás és a gyökvonás is. A gyökvonás Newton iteráció segítségével lett beépítve.

# A program

A kód a különböző könyvtárak meghívásával kezdődik.

```
#include <iostream>
#include <numeric>
#include <cmath>
```

A *numeric*-re a legnagyobb közös osztó megtalálásához szükséges függvény miatt van szükség.

Ezután létrehoztam a *Rational* osztályt, és deklaráltam a különböző függvényeket, amik részei lesznek az osztálynak.

```
class Rational{

    private:

            int szamlalo;
            int nevezo;

            void egysz();

    public:

            Rational(int A, int B);

            void print() const;

            Rational operator+(const Rational &other) const;
            Rational operator-(const Rational &other) const;
            Rational operator*(const Rational &other) const;
            Rational operator/(const Rational &other) const;
            Rational operator^(int C) const;

            Rational gyok(int P, int N) const;

};
```

Látható, hogy az alapműveleteknél és a hatványozásnál a már létező operátorokat fogom újradefiniálni, ahogy azok törtekre alkalmazhatóak.

A beadott számokat a legnagyobb közös osztó módszerével, a *gcd* beépített függvényt használva egyszerűsítettem:

```
void Rational::egysz() {

    int LNKO = std::gcd(szamlalo, nevezo);

    szamlalo /= LNKO;
    nevezo /= LNKO;

       if(nevezo < 0){

              szamlalo *= -1;
              nevezo *= -1;
       }
}
```

Ebben a függvényben végeztem el azt is, hogy negatív számok esetén a számlálóban legyen a mínusz jel. Ez egyrészt esztétikailag is jobb, másrészt megkönnyítette a kódolást a későbbiekben, a műveleteknél.

A következő függvény hozza létre magukat a *Rational* értékeket, egy $A$ számláló és egy $B$ nevező beadásával. Tudja azt, hogy ha a nevező nulla, akkor hibát dob, illetve itt történik meg az előbb megírt egyszerűsítés is.

```
Rational::Rational(int A, int B) {

       szamlalo = A;
       nevezo = B;

    if (B == 0) {
        throw std::invalid_argument("Nullával osztás.");
    }

    egysz();
}
```

Az alapműveletek megírásánál újradefiniáltam az operátorokat. A racionális számaink esetén az eredmény (összeg/különbség/szorzat/hányados) számlálója és nevezője külön-külön kerül kiszámításra a beadott számok számlálójából és nevezőjéből, és azonos formátumban (*Rational* értékként) kerül eltárolásra is.

```
Rational Rational::operator+(const Rational& other) const {

    int ujSzamlalo = (szamlalo * other.nevezo) + (other.szamlalo * nevezo);
       int ujNevezo = nevezo * other.nevezo;

       Rational uj(ujSzamlalo, ujNevezo);

       return uj;
}

Rational Rational::operator-(const Rational& other) const {

    int ujSzamlalo = (szamlalo * other.nevezo) - (other.szamlalo * nevezo);
       int ujNevezo = nevezo * other.nevezo;

       Rational uj(ujSzamlalo, ujNevezo);

       return uj;
}

Rational Rational::operator*(const Rational& other) const {

    int ujSzamlalo = szamlalo * other.szamlalo;
       int ujNevezo = nevezo * other.nevezo;


       Rational uj(ujSzamlalo, ujNevezo);

       return uj;
}

Rational Rational::operator/(const Rational& other) const {

    int ujSzamlalo = szamlalo * other.nevezo;
       int ujNevezo = nevezo * other.szamlalo;


       Rational uj(ujSzamlalo, ujNevezo);

       return uj;
}
```

Újdonság leghamarabb a hatványozásnál van. Ez is az alapműveletekhez hasonló elven működik, viszont negatív kitevő esetén felcseréli a számlálót és a nevezőt, és a kitevő abszolútértékére emeli őket.

```
Rational Rational::operator^(int C) const {

        int ujSzamlalo;
        int ujNevezo;

        if (C > 0) {
            ujSzamlalo = std::pow(szamlalo, C);
            ujNevezo = std::pow(nevezo, C);
        } else {
            ujSzamlalo = std::pow(nevezo, -C);
            ujNevezo = std::pow(szamlalo, -C);
        }

        Rational uj(ujSzamlalo, ujNevezo);

        return uj;
}
```

Gyökvonáshoz a Newton iterációt alkalmaztam. A függvényben alapértelmezetten második gyök és 10 iteráció van beállítva, de a függvény meghívásakor ezeket lehet változtatni.

Először is hibaüzenetet dobatunk negatív szám és nulladik gyök esetén, ezek a racionális számokon belül nem értelmezettek.

```
Rational Rational::gyok(int P = 2, int N = 10) const {

        int kitev;

    if (szamlalo < 0) {
        throw std::invalid_argument("Negatív a gyök alatt.");
    }

    if (P == 0) {
        throw std::invalid_argument("Nulladik gyök nincs.");
    }
```

Ezután double-lé konvertálom a kis törtünket, illetve negatív kitevő esetén a reciprokát veszem és a kitevő abszolútértékével dolgozok.

```
double Q;

if (P>0) {
    Q = (double)szamlalo / (double)nevezo;
    kitev = P;

} else {
    Q =  (double)nevezo / (double)szamlalo;
kitev = -P;
}
```

A Newton iteráció gyökvonásra:

$$x_{n+1} = x_n - \frac{x_n^p - q}{p \cdot x_n^{p-1}},$$

ahol $x$ a keresett értékünk, $p$ a gyök kitevője, $q$ pedig a racionális szám, aminek a gyökére kíváncsiak vagyunk. Az iteráció első lépéseként magát a számot vettem, azaz $x_1 = q$. Ezt implementáltam a kódba is.

```
double GyokQ = Q;

for (int c = 0; c < N; c++) {
    GyokQ = GyokQ - (std::pow(GyokQ, kitev) - Q) /
    (kitev * std::pow(GyokQ, kitev - 1));
}
```

Ahhoz, hogy a végeredmény is megfelelő formátumú racionális szám legyen, a kapott számot ($x$-et) $\frac{x \cdot 10^n}{10^n}$ alakra hoztam. Ehhez a szám 10-es alapú logaritmusát vettem $n$ kiszámolásához, $n$-hez pedig kis kísérletezés után hozzáadtam még 5-öt, hogy a többi értékes jegy se vesszen el átkonvertáláskor. Ez sajnos még így sem mindig működik, de ez volt a legjobb megoldás, amit találtam.

```
int ujNevezo = (int)(std::pow(10, (int)(std::log10(GyokQ)*(-1)+5)));

int ujSzamlalo = (int)(GyokQ*ujNevezo);

Rational uj(ujSzamlalo, ujNevezo);

   return uj;
}
```

A végére még megírtam a kiíratót függvényt is:

```
void Rational::print() const {

    std::cout << szamlalo << "/" << nevezo << std::endl;
}
```

## Tesztelés

Legvégül teszteltem, hogy minden jól működik-e.

```
int main(){

 Rational Q1(9, 5);
 Rational Q2(27, 343);

 Rational QA = Q1 + Q2;
 Rational QB = Q1 - Q2;
 Rational QC = Q1 * Q2;
 Rational QD = Q1 / Q2;

 Rational QE = Q1^2;
 Rational QF = Q1^(-2);
 Rational QG = Q1.gyok();
 Rational QH = Q2.gyok(-3, 100);

 QA.print();
 QB.print();
 QC.print();
 QD.print();
 QE.print();
 QF.print();
 QG.print();
 QH.print();

}
```

A kapott output alább látható, korrektnek tűnik.

```
3222/1715
2952/1715
243/1715
343/15
81/25
25/81
1677/1250
23333/10000
```