Ludwig-Maximilians-Universität München
Mathematisches Institut
Gregor Kleen

Programmieren II für Studierende der Mathematik

Blatt 3 – Lösungsvorschlag

Aufgabe 3 Erstellen Sie eine Klasse `rational` deren Objekte rationale Zahlen modellieren. Die interne Darstellung soll Zähler und Nenner jeweils als private Datenkomponenten vom Typ `long int` speichern. Die interne Darstellung jeder Zahl soll eindeutig sein, insb. der gespeicherte Bruch also jederzeit vollständig gekürzt sein.

Hinweis (Euklidischer Algorithmus). Gegeben zwei Zahlen $a \in \mathbb{N} \setminus \{0\}$ und $b \in \mathbb{N}$ mit obdA. $a > b$ ist ihr größter gemeinsamer Teiler $\operatorname{ggT}(a, b) \in \mathbb{N}$ die größte natürliche Zahl, sodass a und b beide ohne Rest durch $\operatorname{ggT}(a, b)$ teilbar sind.

Es gilt $\operatorname{ggT}(a, b) = \operatorname{ggT}(b, a \bmod b)$ und $\operatorname{ggT}(a, 0) = a$.

Ein Bruch kann vollständig gekürzt werden indem Zähler $p \in \mathbb{N}$ und Nenner $q \in \mathbb{N} \setminus \{0\}$ jeweils geteilt werden durch $\operatorname{ggT}(p, q)$.

Klasse

```
class rational {
  private:
    long int p, q;

    void normalisieren() {
      long int teiler = ggt(p, q);
      p /= teiler;
      q /= teiler;

      if (q < 0) {
        p = -p;
        q = -q;
      }
      if (p == 0)
        q = 1;
    }
    static long int ggt(long int a, long int b) {
      if (a == 0) return abs(b);
      if (b == 0) return abs(a);

      long int h;
      do {
        h = a % b;
        a = b;
        b = h;
      } while (h != 0);

      return abs(a);
    }

  public:
    rational(long int p_ = 0, long int q_ = 1): p(p_), q(q_) {
```

```
      normalisieren();
    }

    Typkonvertierung

    Arithmetik

    Einausgabe
};
```

Überladen Sie die arithmetischen Grundoperationen, die arithmetischen Zuweisungsoperatoren und die Operatoren zur Ein- und Ausgabe auf Strömen. Für $p \in \mathbb{N}$ und $q \in \mathbb{N} \setminus \{0\}$ sollen rationale Zahlen in der Form p/q ein- und ausgegeben werden, optional mit einem positiven oder negativen Vorzeichen davor.

Arithmetik

```
rational& operator+=(const rational& r) {
  p *= r.q;
  p += r.p * q;
  q *= r.q;
  normalisieren();
  return *this;
}
friend rational operator-(const rational& r) {
  return rational{-r.p, r.q};
}
rational& operator-=(const rational& r) {
  return *this += -r;
}
rational& operator*=(const rational& r) {
  p *= r.p;
  q *= r.q;
  normalisieren();
  return *this;
}
rational& operator/=(const rational& r) {
  return *this *= rational{r.q, r.p};
}

friend rational operator+(rational r, const rational& s) {
  return r += s;
}
friend rational operator-(rational r, const rational& s) {
  return r -= s;
}
friend rational operator*(rational r, const rational& s) {
  return r *= s;
}
friend rational operator/(rational r, const rational& s) {
  return r /= s;
}
```

Einausgabe

```
friend ostream& operator<<(ostream& stream, const rational& r) {
  return stream << r.p << "/" << r.q;
}
friend istream& operator>>(istream& stream, rational& r) {
  long int p_, q_;
  char c;
  stream >> p_ >> c >> q_;
  if (c != '/' || q_ == 0)
    stream.setstate(ios::failbit);
  else
    r = rational{p_, q_};
  return stream;
}
```

Implementieren Sie eine Typkonvertierung von `rational` nach `long double`.

Typkonvertierung

```
operator long double() {
  return static_cast<long double>(p) / static_cast<long double>(q);
}
```

Erstellen Sie eine Potenzfunktion `pow` zur Berechnung von r^n für rationales r und $n \in \mathbb{Z}$.

pow

```
rational pow(const rational& r, long int z) {
  rational res{1};

  for (long int i = 0; i < abs(z); i++) {
    if (z < 0)
      res /= r;
    else
      res *= r;
  }

  return res;
}
```

Erstellen Sie eine Funktion zur Berechnung von $\sum_{k=1}^{n} kr^k$.

Potreihe

```
rational pot_reihe(const rational& r, long int n) {
  rational res{0};
  for (long int k = 1; k <= n; k++) {
    res += static_cast<rational>(k) * pow(r, k);
  }
  return res;
}
```

Erstellen Sie eine Funktion zur Berechnung des Kettenbruchs

$$b_0 + \cfrac{a_0}{b_1 + \cfrac{a_1}{b_2 + \cfrac{a_2}{\ddots \quad b_{n-1} + \cfrac{a_{n-1}}{b_n}}}}$$

mit $a_0, \ldots, a_{n-1}, b_0, \ldots, b_n \in \mathbb{N}$.

Kettenbruch

```
rational kettenbruch(const vector<long int>& a, const vector<long int>& b) {
  vector<long int>::size_type n = a.size();
  rational res{b[n]};
  for (vector<long int>::size_type i = n; i > 0; i--)
    res = rational(b[i - 1]) + rational(a[i - 1]) / res;
  return res;
}
```

Erstellen Sie ein Hauptprogramm und berechnen Sie die folgenden Beispiele sowohl als rationale Zahl wie auch eine dezimale Näherung:

a)

$$\sum_{k=1}^{n} kr^k$$

mit $r = \frac{2}{3}, -\frac{10}{7}$ und $n = 8$

b)

$$\frac{nr^{n+2} - (n+1)r^{n+1} + r}{(r-1)^2}$$

mit $r = \frac{2}{3}, -\frac{10}{7}$ und $n = 8$

c)

$$3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292}}}}$$

d) Kettenbrüche mit $a_0 = 4$, $b_0 = 0$, $a_i = i^2$, und $b_i = 2i - 1$ für $i \geq 1$ und $n = 1, 2, \ldots, 10$

rational.cpp

```
#include <iostream>
#include <iomanip>
#include <vector>

using namespace std;
```

Klasse

pow

Potreihe

Kettenbruch

```
int main() {
  cout << setprecision(18);

  long int n = 8;
  vector<rational> rs{rational{2, 3}, -rational{10, 7}};

  // a)
  for (rational r: rs)
    cout << "pot_reihe(" << r << ", " << n << ") = "
         << pot_reihe(r, n) << " =~ "
         << static_cast<long double>(pot_reihe(r, n)) << endl;
  cout << endl;

  // b)
  for (rational r: rs) {
    rational p = (rational{n} * pow(r, n + 2) - rational{n + 1} * pow(r, n + 1) + r) / pow(r -
    ↪  rational{1}, 2);
    cout << "p = " << p << " =~ " << static_cast<long double>(p) << endl;
  }
  cout << endl;

  // c)
  rational r = kettenbruch({1, 1, 1, 1}, {3,7, 15, 1, 292});
  cout << "kettenbruch(...) = "
       << r << " =~ "
       << static_cast<long double>(r) << endl;
  cout << endl;

  // d)
  for (long int n = 1; n <= 10; n++) {
   vector<long int> a(n), b(n + 1);
   a[0] = 4;
   b[0] = 0;
   for (long int i = 1; i <= n; i++) {
     if (i < n)
       a[i] = i * i;
     b[i] = 2 * i - 1;
   }

   rational k = kettenbruch(a, b);
   cout << "k[" << setw(2) << n << "] = "
        << k << " =~ "
        << static_cast<long double>(k) << endl;
  }
}
```

```
pot_reihe(2/3, 8) = 33734/6561 =~ 5.14159426916628563
pot_reihe(-10/7, 8) = 493413370/5764801 =~ 85.5907029574828342

p = 33734/6561 =~ 5.14159426916628563
p = 493413370/5764801 =~ 85.5907029574828342

kettenbruch(...) = 103993/33102 =~ 3.1415926530119026

k[ 1] = 4/1 =~ 4
k[ 2] = 3/1 =~ 3
k[ 3] = 19/6 =~ 3.16666666666666667
k[ 4] = 160/51 =~ 3.13725490196078431
k[ 5] = 1744/555 =~ 3.14234234234234234
k[ 6] = 644/205 =~ 3.14146341463414634
k[ 7] = 2529/805 =~ 3.14161490683229814
k[ 8] = 183296/58345 =~ 3.14158882509212443
k[ 9] = 3763456/1197945 =~ 3.14159331187992771
k[10] = 4317632/1374345 =~ 3.14159254044653999
```