Compte Rendu TP3 CPOO

Théo CHAPON, Hassan EL OMARI ALAOUI

INSA de Rennes
4INFO, groupe 1.1

12 octobre 2014

TP3 : Gestion des exceptions

1 Etat du TP

Le projet est terminé, fonctionel et testé.

2 Objectif

Ce TP avait pour but d'apprendre à lever et à gérer des exceptions.

3 Réalisation

Pour apprendre à gérer les exceptions, nous avons dû créer une classe fraction. Nous devions donc créer des exceptions pour les divisions par zéro, pour les overflows. Nous devions aussi surcharger des opérateurs de calcule.

Nous devions aussi gérer les exceptions dans une seconde partie. Nous devions cette fois-ci gérer les suites de nucléotides de l'ADN et de l'ARN. Les exceptions à gérer étaient de savoir si les bonnes lettres étaient utilisés, si les codons START et STOP étaient bien placés et si les suites étaient uniquement composés de codons.

Aucune difficulté n'a été rencontrée lors de ce TP.

Listing 1 – Définition de la class fraction en C++

```
#ifndef FRACTION_H
#define FRACTION_H
#include <algorithm>
#include <exception>
#include <iostream>
using namespace std;
class Fraction
{
  int _num;/*<!Numerator>*/
  int _den;/*<!Denominator>*/
  /**
   * \fn ostream& print(ostream& os)
   * \brief Display Fraction object
   * \param[in,out] os output stream
   * \return ostream& reference of output stream
   */
  ostream& _print(ostream& os);
public:
  /**
  * \fn Fraction()
  * \brief Fraction constructor without parameters
  */
  Fraction();
  /**
  * \fn Fraction(const int num, const int den)
  * \brief Fraction constructor with parameters
  * \param[in] num : fraction's numerator
  * \param[in] den : fraction's denominator
  */
  Fraction(const int num, const int den);
  /**
  * \fn Fraction(const Fraction& f)
  * \brief Fraction copy constructor
  * \param[in] f : fraction
  */
  Fraction(const Fraction &f);
  /**
  * \fn int getNum() const
  * \brief Getter of _num
  * \return int
  */
  int getNum() const;
  /**
  * \fn int getNum() const
  * \brief Getter of _den
  * \return int
  */
  int getDen() const;
```

```
49  /**
50  * \fn double eval() const;
51  * \brief Give the real value in double of the fraction
52  * \return double
53  */
54  double eval() const;
55  /**
56  * \fn Fraction& simplifier()
57  * \brief Simplify a fraction
58  * \return void
59  */
60  Fraction& simplifier();
61  /**
62  * \fn friend Fraction& operator+=(Fraction& f1, const Fraction f2)
63  * \brief Overloads += operator
64  * Return the addition between two fractions
65  * \param[in,out] f1 Fraction&
66  * \param[in] f2 Fraction&
67  * \return Fraction&
68  */
69  friend Fraction& operator+=(Fraction& f1, const Fraction f2);
70  /**
71  * \fn friend Fraction& operator+(const Fraction& f1, const Fraction f2)
72  * \brief Overloads + operator
73  * Return the addition between two fractions
74  * \param[in] f1 Fraction&
75  * \param[in] f2 Fraction&
76  * \return Fraction*
77  */
78  friend Fraction* operator+(const Fraction& f1, const Fraction f2);
79  /**
80  * \fn friend Fraction& operator-=(Fraction& f1, const Fraction f2)
81  * \brief Overloads -= operator
82  * Return the substraction between two fractions
83  * \param[in,out] f1 Fraction&
84  * \param[in] f2 Fraction&
85  * \return Fraction&
86  */
87  friend Fraction& operator-=(Fraction& f1, const Fraction f2);
88  /**
89  * \fn friend Fraction& operator-(const Fraction& f1, const Fraction f2)
90  * \brief Overloads - operator
91  * Return the substraction between two fractions
92  * \param[in] f1 Fraction&
93  * \param[in] f2 Fraction&
94  * \return Fraction*
95  */
96  friend Fraction* operator-(const Fraction& f1, const Fraction f2);
97  /**
```

```
98    * \fn friend Fraction& operator*=(Fraction& f1, const Fraction f2)
99    * \brief Overloads *= operator
100   * Return the multiplication between two fractions
101   * \param[in,out] f1 Fraction&
102   * \param[in] f2 Fraction&
103   * \return Fraction&
104   */
105   friend Fraction& operator*=(Fraction& f1, const Fraction f2);
106   /**
107   * \fn friend Fraction& operator*(const Fraction& f1, const Fraction f2)
108   * \brief Overloads * operator
109   * Return the multiplication between two fractions
110   * \param[in] f1 Fraction&
111   * \param[in] f2 Fraction&
112   * \return Fraction*
113   */
114   friend Fraction* operator*(const Fraction& f1, const Fraction f2);
115   /**
116   * \fn friend Fraction& operator/=(Fraction& f1, const Fraction f2)
117   * \brief Overloads /= operator
118   * Return the division between two fractions
119   * \param[in,out] f1 Fraction&
120   * \param[in] f2 Fraction&
121   * \return Fraction&
122   */
123   friend Fraction& operator/=(Fraction& f1, const Fraction f2);
124   /**
125   * \fn friend Fraction& operator/(const Fraction& f1, const Fraction f2)
126   * \brief Overloads / operator
127   * Return the division between two fractions
128   * \param[in] f1 Fraction&
129   * \param[in] f2 Fraction&
130   * \return Fraction*
131   */
132   friend Fraction* operator/(const Fraction& f1, const Fraction f2);
133   /**
134   * \fn friend bool operator<<(ostream& os, Fraction& f)
135   * \brief Overloads << operator
136   * Return output stream to display Fraction object
137   * \param[in,out] os ostream&
138   * \param[in] f Fraction&
139   * \return ostream&
140   */
141   friend ostream& operator<<(ostream& os, Fraction& f);
142 };
143 enum Over{PLUS, MULT};
144 #endif
```

Listing 2 – Class fraction en C++

```
#include "stdafx.h"
#include "Fraction.h"

Fraction::Fraction() :_num(0), _den(1){}
Fraction::Fraction(const int num, const int den) : _num(num), _den(den){
  if (_den == 0)
  {
    throw exception("The denominator must be not equal to 0");
  }
}
Fraction::Fraction(const Fraction& f) {
  _num = f.getNum();
  _den = f.getDen();
}
int Fraction::getNum() const {
  return _num;
}
int Fraction::getDen() const {
  return _den;
}
double Fraction::eval() const {
  return (double) _num / _den;
}
int pgcd(int a, int b) {
  int d = abs(a - b);
  if (d == 0) {
    return a;
  }
  else {
    return pgcd(min(a, b), d);
  }
}
Fraction& Fraction::simplifier() {
  int a;
  while ((a = pgcd(_num, _den)) != 1) {
    _num /= a;
    _den /= a;
  }
  return *this;
}
void testOverflow(int a, int b, int operation) {
  char sign = a > 0 && b > 0 ? 1 : a < 0 && b < 0 ? -1 : 0;
  int sum;
  switch (operation) {
  case PLUS:
    sum = a + b;
    break;
  case MULT:
```

```
49      sum = a * b;
50      break;
51    }
52    if (sign && sign == 1 && sum < 0 || sign == -1 && sum > 0){
53      throw exception("Overflow occured");
54    }
55  }
56  Fraction& operator+=(Fraction& f1, const Fraction f2) {
57    testOverflow(f1._num, f2._den, MULT);
58    int a = f1._num * f2._den;
59    testOverflow(f2._num, f1._den, MULT);
60    int b = f2._num * f1._den;
61    testOverflow(a, b, PLUS);
62    f1._num = a + b;
63    testOverflow(f1._den, f2._den, MULT);
64    f1._den *= f2._den;
65    f1.simplifier();
66    return f1;
67  }
68  Fraction* operator+(const Fraction& f1, const Fraction f2) {
69    Fraction* f3 = new Fraction(f1);
70    *f3 += f2;
71    return f3;
72  }
73  Fraction& operator-=(Fraction& f1, const Fraction f2) {
74    Fraction f3 = Fraction(-f2._num, f2._den);
75    f1 += f3;
76    return f1;
77  }
78  Fraction* operator-(const Fraction& f1, const Fraction f2) {
79    Fraction* f3 = new Fraction(f1);
80    *f3 -= f2;
81    return f3;
82  }
83  Fraction& operator*=(Fraction& f1, const Fraction f2) {
84    testOverflow(f1._num, f2._num, MULT);
85    f1._num *= f2._num;
86    testOverflow(f1._den, f2._den, MULT);
87    f1._den *= f2._den;
88    f1.simplifier();
89    return f1;
90  }
91  Fraction* operator*(const Fraction& f1, const Fraction f2) {
92    Fraction* f3 = new Fraction(f1);
93    *f3 *= f2;
94    return f3;
95  }
96  Fraction& operator/=(Fraction& f1, const Fraction f2) {
97    Fraction f3 = Fraction(f2._den, f2._num);
```

```
98   f1 *= f3;
99   return f1;
100 }
101 Fraction* operator/(const Fraction& f1, const Fraction f2) {
102   Fraction* f3 = new Fraction(f1);
103   *f3 /= f2;
104   return f3;
105 }
106 ostream& Fraction::_print(ostream& os) {
107   return os << _num << "/" << _den;
108 }
109 ostream& operator<<(ostream& os, Fraction& f) {
110   return f._print(os);
111 }
```

Listing 3 – Class sequences en C++