Compte Rendu TP3 CPOO

Loick Bonniot, Quentin Dufour

INSA Rennes
4INFO, groupe 1.2

13 octobre 2015

Nous n'avons pas eu le temps de terminer le TP, nous nous sommes par contre concentré sur le premier exercice.

Listing 1 – Le fichier principal

```
#include <assert.h>
#include <iostream>
#include "Fraction.h"
#include "sequences.h"

void testFraction() {
  // Test constructors

  Fraction a = Fraction(2);
  Fraction b = Fraction(5, 2);
  Fraction c = Fraction(-1, 5);

  assert(a.eval() == 2);
  assert(b.eval() == 2.5);
  assert(c.eval() == -0.2);

  // Test operators

  Fraction sum = a + b;
  Fraction mul = a * b;

  assert(sum.eval() == 4.5);
  assert(mul.eval() == 5);

  // Test exceptions

  bool passed;
```

```
28    passed = false;
29    try {
30      Fraction zero = Fraction(1, 0);
31    }
32    catch (Fraction::DivideByZeroError) {
33      passed = true;
34    }
35    assert(passed);
36
37    passed = false;
38    try {
39      Fraction huge = Fraction(INT_MAX - 1) + a;
40    }
41    catch (Fraction::OverflowError) {
42      passed = true;
43    }
44    assert(passed);
45  }
46
47  void testSeq() {
48
49    try {
50      std::string s1, s2, s3;
51
52      std::cout << "Entrez une sequence proteique: ";
53      std::cin >> s1;
54      std::cout << "Entrez son nom: ";
55      std::cin >> s2;
56
57      seqprot P1(s1, s2);
58      std::cout << P1 << std::endl;
59
60      std::cout << "Entrez une sequence d'adn: ";
61      std::cin >> s1;
62      std::cout << "Entrez son nom: ";
63      std::cin >> s2;
64
65      seqadn A1(s1, s2);
66      std::cout << A1 << std::endl;
67
68      std::cout << "Entrez une sequence d'arn: ";
69      std::cin >> s1;
70      std::cout << "Entrez son nom: ";
71      std::cin >> s2;
72
73      seqarn R1(s1, s2);
74      std::cout << R1 << std::endl;
75    }
76    catch (seqmac::UnknownCharError e) {
```

```
77      std::cout << "ERROR: illegal character '" << e.getChar() << "' in sequence."
            << std::endl;
78    }
79  }
80
81  int main(int argv, char** argc) {
82
83    std::cout << "Testing Fraction..." << std::endl;
84    testFraction();
85    std::cout << "Testing Sequences..." << std::endl;
86    testSeq();
87    std::cout << "All tests passed!" << std::endl;
88
89    system("pause");
90
91    return 0;
92  }
```

Listing 2 – Les en-têtes de Fraction

```
1  #ifndef FRACTION_H
2  #define FRACTION_H
3
4  #include <exception>
5
6  class Fraction {
7
8  private:
9    int _n;
10   int _d;
11
12   static int safeAdd(int a, int b);
13   static int safeMul(int a, int b);
14
15 public:
16   Fraction(int n, int d = 1);
17
18   Fraction operator+(const Fraction& f);
19   Fraction operator*(const Fraction& f);
20
21   double eval();
22
23   // Exceptions
24
25   class Error : public std::exception {
26   public:
27     virtual const char* what(void);
28   };
29
30   class DivideByZeroError : public Fraction::Error {
```

```
31    public:
32      virtual const char* what(void);
33    };
34
35    class OverflowError : public Fraction::Error {
36    public:
37      virtual const char* what(void);
38    };
39
40  };
41
42  #endif
```

Listing 3 – La logique de Fraction

```
1  #include "Fraction.h"
2
3  Fraction::Fraction(int n, int d) : _n(n), _d(d) {
4    if (d == 0) {
5      throw Fraction::DivideByZeroError();
6    }
7  }
8
9  int Fraction::safeAdd(int a, int b) {
10   if (a > INT_MAX - b) {
11     throw Fraction::OverflowError();
12   }
13   return a + b;
14 }
15
16 int Fraction::safeMul(int a, int b) {
17   if (a == 0 || b == 0) {
18     return 0;
19   }
20   if (b > 0 && (a > INT_MAX / b || a < INT_MIN / b)) {
21     throw Fraction::OverflowError();
22   }
23   if (b < 0 && (a < INT_MAX / b || a > INT_MIN / b)) {
24     throw Fraction::OverflowError();
25   }
26   return a * b;
27 }
28
29 Fraction Fraction::operator+(const Fraction & f) {
30   int n = Fraction::safeAdd(
31     Fraction::safeMul(_n, f._d),
32     Fraction::safeMul(_d, f._n)
33   );
34   return Fraction(n, Fraction::safeMul(_d, f._d));
35 }
```

```

Fraction Fraction::operator*(const Fraction & f) {
  return Fraction(Fraction::safeMul(_n, f._n), Fraction::safeMul(_d, f._d));
}

double Fraction::eval() {
  return (double) _n / (double) _d;
}

const char * Fraction::Error::what(void) { return "Generic error in Fraction"; }
const char * Fraction::DivideByZeroError::what(void) { return "Division by zero in Fraction"; }
const char * Fraction::OverflowError::what(void) { return "Integer overflow in Fraction"; }
```

Listing 4 – Les en-têtes de séquence

```
/*!
 * \file  sequences.h
 * \brief  Sequences reading
 * Read new nucleotide or protein sequences
*/

#ifndef SEQ_H
#define SEQ_H

#include <set>
#include <string>
#include <iostream>

/*! \brief Alphabet class. Used to validate a character against an alphabet. */
class alpha
{

    public:

  /*! \brief Constructor. Builds an alphabet from an input string.
  * \param s String used as alphabet */
  alpha(const std::string & s)
  {
      for(std::string::const_iterator c = s.begin();c != s.end(); c++ )
      _cs.insert(*c);
  }

  /*! \brief Checks if the given character is in the alphabet.
  * \param c Character to validate against the alphabet.
  * \return True if the character is in the alphabet. False otherwise. */
  bool is_in_alpha(char c) const { return (_cs.find(c)!=_cs.end()); }

    private:
```

```
34
35   /*! The alphabet */
36   std::set<char> _cs;
37
38 };
39
40
41 /*! \brief Generic sequence with alphabet. */
42 class seqmac
43 {
44     public:
45
46   /*! \brief Constructor. Builds the sequence from an input string and an
         alphabet.
47   * \param seq Sequence of characters
48   * \param name Name of the sequence
49   * \param alphabet alphabet to be used to verify the input sequence */
50   seqmac (const std::string & seq, const std::string & name, const std::string &
         alphabet);
51
52   /*! \brief Output the sequence to an output stream.
53   * \param os Output stream
54   * \param seq Sequence
55   * \return The output stream */
56   friend std::ostream & operator<< (std::ostream & os, const seqmac & seq);
57
58   /*! \brief An exception used when encountering wrong character in the sequence
         */
59   class UnknownCharError : public std::exception {
60     private:
61       char _c;
62     public:
63       UnknownCharError(char c) : _c(c) {}
64       char getChar() { return _c; }
65   };
66
67     protected:
68
69   /*! Sequence */
70   std::string _seq;
71
72   /*! Name */
73   std::string _name;
74
75     private:
76
77   /*! Alphabet */
78   const alpha _alph;
79
```

```
80   /*! \brief Formatted output of the sequence.
81   * \param os Output stream */
82   void writeseq(std::ostream & os) const
83   {
84       os << "SEQUENCE" << std::endl << "--------------\n";
85       os << "Nom : " << _name << std::endl;
86       os << "Seq : " << _seq << std::endl;
87       os << "aa  : " << _seq.size() << std::endl;
88   }
89 };
90
91
92 /*! \brief Specialized sequence for proteins. */
93 class seqprot: public seqmac
94 {
95
96     public:
97
98   /*! \brief Constructor. Builds the proteine sequence from an input string with
        alphabet check. The alphabet is hardcoded for protein characters.
99   * \param seq Sequence of characters
100  * \param name Name of the sequence */
101  seqprot(const std::string & seq="", const std::string & name="") :
        seqmac(seq,name,"ACDEFGHIKLMNPQRSTV") {}
102
103 };
104
105
106 /*! \brief Specialized sequence for ADN. */
107 class seqadn: public seqmac
108 {
109
110     public:
111
112  /*! \brief Constructor. Builds the ADN sequence from an input string with
        alphabet check. The alphabet is hardcoded for ADN characters.
113  * \param seq Sequence of characters
114  * \param name Name of the sequence */
115  seqadn(const std::string & seq="", const std::string & name="");
116
117 };
118
119
120 /*! \brief Specialized sequence for ARN. */
121 class seqarn: public seqmac
122 {
123
124     public:
125
```

```
126   /*! \brief Constructor. Builds the ARN sequence from an input string with
          alphabet check. The alphabet is hardcoded for ARN characters.
127   * \param seq Sequence of characters
128   * \param name Name of the sequence */
129   seqarn(const std::string & seq="", const std::string & name="");
130
131 };
132
133 #endif // SEQ_H
```

Listing 5 – La logique de séquence

```
1  /*!
2   * \file  sequences.cpp
3   * \brief  Sequences reading
4   * \date to be updated
5   * \author to be updated
6  */
7
8  #include "sequences.h"
9
10 using namespace std;
11
12 ostream & operator<< (ostream & os, const seqmac & s)
13 {
14     s.writeseq(os);
15     return os;
16 }
17
18 seqmac::seqmac(const string & seq, const string & name, const string & alphabet) :
19   _alph(alphabet), _name(name)
20 {
21     string s="";
22     for(string::const_iterator c = seq.begin();c != seq.end(); c++ )
23     {
24     if (_alph.is_in_alpha(*c)) { s += *c; }
25     else { throw seqmac::UnknownCharError(*c); }
26     }
27     _seq = s;
28 }
29
30 seqadn::seqadn(const string & seq, const string & name) : seqmac(seq,name,"CGAT")
31 {
32 }
33
34 seqarn::seqarn(const string & seq, const string & name) : seqmac(seq,name,"ACGU")
35 {
36 }
```