| | |
|---|---|
| **Início** | quarta, 11 de maio de 2022 às 08:41 |
| **Estado** | Prova submetida |
| **Data de submissão:** | sábado, 14 de maio de 2022 às 23:16 |
| **Tempo gasto** | 3 dias 14 horas |
| **Nota** | **100** do máximo 100 |

## Pergunta 1

Correta Pontuou 20 de 20

Consider the code given in Point2d.h containing the definition of class `Point2d` to represent 2D-coordinates of type `double`:

```
class Point2d {
  public:
    Point2d();                  // default constructor (builds origin)
    Point2d(const Point2d& p);  // copy constructor
    Point2d(double x, double y); // constructor using supplied coordinates
    double get_x() const;  // accessor for X coordinate
    double get_y() const;  // accessor for Y coordinate
    void set_x(double x);  // mutator for X coordinate
    void set_y(double y);  // mutator for Y coordinate
    void translate(const Point2d& t);           // translate coordinates
    double distance_to(const Point2d& p) const; // get distance to given point
  private:
    double x;  // X coordinate
    double y;  // Y coordinate
};
```

Write the C++ code for the `translate()` and `distance_to()` member functions (not yet implemented), such that:

- `a.translate(t)` changes `a` with a translation by coordinates `t`, i.e., if `a` has coordinates `(x,y)` on entry, then `a` should have coordinates `(x + t.x, y + t.y)` on exit; and
- `a.distance_to(b)` returns the Euclidean distance between `a` and `b`, i.e., $\sqrt{(a.x - b.x)^2 + (a.y - b.y)^2}$

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `Point2d a { 1.0, 2.1 };`<br>`a.translate({ -0.3, 0.5 });`<br>`cout << fixed << setprecision(2)`<br>`  << a.get_x() << ' ' << a.get_y() << '\n';` | `0.70 2.60` |
| `Point2d a { 1.0, 2.1 }, b { 2.0, 3.1 };`<br>`cout << fixed << setprecision(2)`<br>`  << a.distance_to(b) << '\n';` | `1.41` |
| `Point2d a { 0, 0 }; Point2d b = a;`<br>`a.translate({ 0, 0 });`<br>`cout << fixed << setprecision(2)`<br>`  << a.get_x() << ' ' << a.get_y() << ' '`<br>`  << a.distance_to(b) << '\n';` | `0.00 0.00 0.00` |
| `Point2d a { 0, 1 }; Point2d b = a;`<br>`a.translate({ 1.5, 0 });`<br>`cout << fixed << setprecision(2)`<br>`  << a.get_x() << ' ' << a.get_y() << ' '`<br>`  << a.distance_to(b) << '\n';` | `1.50 1.00 1.50` |
| `Point2d a { 1, 0 }; Point2d b = a;`<br>`a.translate({ -1, 2.5 });`<br>`cout << fixed << setprecision(2)`<br>`  << a.get_x() << ' ' << a.get_y() << ' '`<br>`  << a.distance_to(b) << '\n';` | `0.00 2.50 2.69` |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

```
1  #include <iostream>
2  #include <iomanip>
3  #include <cmath>
4  using namespace std;
5
6  #include "Point2d.h"
7
8  void Point2d::translate(const Point2d& t){
9      this->x += t.x;
10     this->y += t.y;
11 }
12
13 double Point2d::distance_to(const Point2d& p) const{
14     return sqrt(pow(this->x - p.x, 2) + pow(this->y - p.y, 2));
```

```
15 }
16
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `Point2d a { 1.0, 2.1 };`<br>`a.translate({ -0.3, 0.5 });`<br>`cout << fixed << setprecision(2)`<br>`  << a.get_x() << ' ' << a.get_y() << '\n';` | 0.70 2.60 | 0.70 2.60 | ✔ |
| ✔ | `Point2d a { 1.0, 2.1 }, b { 2.0, 3.1 };`<br>`cout << fixed << setprecision(2)`<br>`  << a.distance_to(b) << '\n';` | 1.41 | 1.41 | ✔ |
| ✔ | `Point2d a { 0, 0 }; Point2d b = a;`<br>`a.translate({ 0, 0 });`<br>`cout << fixed << setprecision(2)`<br>`  << a.get_x() << ' ' << a.get_y() << ' '`<br>`  << a.distance_to(b) << '\n';` | 0.00 0.00 0.00 | 0.00 0.00 0.00 | ✔ |
| ✔ | `Point2d a { 0, 1 }; Point2d b = a;`<br>`a.translate({ 1.5, 0 });`<br>`cout << fixed << setprecision(2)`<br>`  << a.get_x() << ' ' << a.get_y() << ' '`<br>`  << a.distance_to(b) << '\n';` | 1.50 1.00 1.50 | 1.50 1.00 1.50 | ✔ |
| ✔ | `Point2d a { 1, 0 }; Point2d b = a;`<br>`a.translate({ -1, 2.5 });`<br>`cout << fixed << setprecision(2)`<br>`  << a.get_x() << ' ' << a.get_y() << ' '`<br>`  << a.distance_to(b) << '\n';` | 0.00 2.50 2.69 | 0.00 2.50 2.69 | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
1  #include <iostream>
2  #include <iomanip>
3  #include <cmath>
4  using namespace std;
5
6  #include "Point2d.h"
7
8  void Point2d::translate(const Point2d& t) {
9    x += t.x;
10   y += t.y;
11 }
12
13 double Point2d::distance_to(const Point2d& p) const {
14   double dx = x - p.x;
15   double dy = y - p.y;
16   return std::sqrt(dx * dx + dy * dy);
17 }
18 /*
19   // private (1000 points each)
20   {
21     Point2d a { 5.0, -2.1 };
22     a.translate({ -0.3, 0.5 });
```

Correta

Nota desta submissão: 20/20

## Pergunta 2

Correta Pontuou 20 de 20

Consider the code given in Date1.h containing the definition of class Date:

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

//! A simple class Date definition
class Date {
  public:
    Date();                             // Default constructor
    Date(int year, int month, int day); // Constructor with parameters
    int get_year() const;               // Year accessor
    int get_month() const;              // Month accessor
    int get_day() const;                // Day accessor
    void write() const;                 // Writes the date as "yyyy/mm/dd"
  private:
    int year;   // Private attribute year
    int month;  // Private attribute month
    int day;    // Private attribute day
};
```

Write the C++ code for all member functions not yet implemented:

- The default constructor must build the date 1/1/1;
- For the constructor with parameters, assume that the supplied arguments always define a valid date and that the year value is between 1 and 9999;
- Write also the code of the accessors that return the private attributes of the class.

Finally, write a function (not part of class Date), bool is_before(const Date& date1, const Date& date2) that returns true if date1 is before date2, and false otherwise.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| Date d1; d1.write(); cout << '\n'; | 0001/01/01 |
| Date d2(2020, 2, 29); d2.write(); cout << '\n'; | 2020/02/29 |
| Date d3(1925, 4, 30); d3.write(); cout << '\n'; | 1925/04/30 |
| Date d4, d5(2022,4,29); d4.write(); cout << '-' << boolalpha << is_before(d4, d5) << '\n'; | 0001/01/01-true |
| Date d6(2023,10,31), d7(2022, 4, 29); d6.write(); cout << '-' << boolalpha << is_before(d6, d7) << '\n'; | 2023/10/31-false |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

```
#include <iostream>
#include <iomanip>
#include "Date1.h"

using namespace std;

Date::Date(){
    year = 1;
    month = 1;
    day = 1;
}

Date::Date(int year, int month, int day) {
    this->year = year;
    this->month = month;
    this->day = day;
}

int Date::get_day() const{
    return this->day;
}

```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `Date d1; d1.write(); cout << '\n';` | `0001/01/01` | `0001/01/01` | ✔ |
| ✔ | `Date d2(2020, 2, 29); d2.write(); cout << '\n';` | `2020/02/29` | `2020/02/29` | ✔ |
| ✔ | `Date d3(1925, 4, 30); d3.write(); cout << '\n';` | `1925/04/30` | `1925/04/30` | ✔ |
| ✔ | `Date d4, d5(2022,4,29); d4.write(); cout << '-' << boolalpha << is_before(d4, d5) << '\n';` | `0001/01/01-true` | `0001/01/01-true` | ✔ |
| ✔ | `Date d6(2023,10,31), d7(2022, 4, 29); d6.write(); cout << '-' << boolalpha << is_before(d6, d7) << '\n';` | `2023/10/31-false` | `2023/10/31-false` | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
#include <iostream>
#include <iomanip>
#include "Date1.h"

using namespace std;

Date::Date() : year(1), month(1), day(1) { }

Date::Date(int year, int month, int day)
  :  year(year), month(month), day(day) {
  // internal use only
  assert (year >= 1 && year <= 9999 &&
          month >= 1 && month <= 12 &&
          day >= 1 && day <= days_in_month(year, month), "Invalid date-");
}

int Date::get_year() const {
  return year;
}

int Date::get_month() const {
  return month;
```

Correta

Nota desta submissão: 20/20

## Pergunta 3

Correta Pontuou 20 de 20

Consider the code given in Date2.h containing the definition of class `Date` :

```
#include <iostream>
#include <iomanip>
#include <string>

class Date {
  public:
    // constructors
    Date();
    Date(int year, int month, int day);
    Date(const std::string& year_month_day);
    // accessors
    int get_year() const;
    int get_month() const;
    int get_day() const;
    // other methods
    bool is_valid() const; // tests date validity
    void write() const;    // writes the date as "yyyy/mm/dd"
  private:
    // attributes
    int year;
    int month;
    int day;
    // compute the number of days of month
    static int num_days(int year, int month);
};
```

Write the C++ code for the member functions of the class not yet implemented, observing the following requirements:

- The default constructor must build the date `1/1/1`;
- When the parameters of the other constructors constitute an invalid date, the values of the attributes must take the value zero (consider that a date is valid for years between 1 and 9999);
- The parameter of the constructor `Date(const std::string& year_month_day)` must be in the format: `y/m/d`, where `y`, `m` and `d` can have a variable number of digits (examples of valid dates: "2022/04/01", "2022/4/13", "899/1/1"); the separator must always be the character `'/'`;
- The member function `is_valid()` returns false if one of the attributes is equal to zero.

Hint: use a `istringstream` object to decompose the string year_month_day.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `Date d1; d1.write(); cout << (d1.is_valid() ? "" : "-invalid") << endl;` | `0001/01/01` |
| `Date d2(2022, 4, 31); d2.write(); cout << (d2.is_valid() ? "" : "-invalid") << endl;` | `0000/00/00-invalid` |
| `Date d3("2ooo/2/28"); d3.write(); cout << (d3.is_valid() ? "" : "-invalid") << endl;` | `0000/00/00-invalid` |
| `Date d4("1900/1/1"); d4.write(); cout << (d4.is_valid() ? "" : "-invalid") << endl;` | `1900/01/01` |
| `Date d5("2022#12#31"); d5.write(); cout << (d5.is_valid() ? "" : "-invalid") << endl;` | `0000/00/00-invalid` |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

```
#include <iostream>
#include <iomanip>
#include <sstream>

#include "Date2.h"

using namespace std;

int days_in_month(int year, int month) {
    int feb = (month == 2 && year % 4 == 0 && (year % 100 != 0 || year % 400 == 0))? 29 : 28;
    int n_days[12] = {31, feb, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    return n_days[month - 1];
}

Date::Date(){
```

```
Date::Date(){
    year = 1;
    month = 1;
    day = 1;
}

Date::Date(int year, int month, int day) {
    if(day <= days_in_month(year, month) && year >= 1 &&year <= 9999){
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `Date d1; d1.write(); cout << (d1.is_valid() ? "" : "-invalid") << endl;` | 0001/01/01 | 0001/01/01 | ✔ |
| ✔ | `Date d2(2022, 4, 31); d2.write(); cout << (d2.is_valid() ? "" : "-invalid") << endl;` | 0000/00/00-invalid | 0000/00/00-invalid | ✔ |
| ✔ | `Date d3("2ooo/2/28"); d3.write(); cout << (d3.is_valid() ? "" : "-invalid") << endl;` | 0000/00/00-invalid | 0000/00/00-invalid | ✔ |
| ✔ | `Date d4("1900/1/1"); d4.write(); cout << (d4.is_valid() ? "" : "-invalid") << endl;` | 1900/01/01 | 1900/01/01 | ✔ |
| ✔ | `Date d5("2022#12#31"); d5.write(); cout << (d5.is_valid() ? "" : "-invalid") << endl;` | 0000/00/00-invalid | 0000/00/00-invalid | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
#include <iostream>
#include <iomanip>
#include <sstream>

#include "Date2.h"

using namespace std;

Date::Date() : year(1), month(1), day(1) { }

Date::Date(int year, int month, int day) {
  if (year >= 1 && year <= 9999 &&
    month >= 1 && month <= 12 &&
    day >= 1 && day <= num_days(year, month))
  {
    this->year = year;
    this->month = month;
    this->day = day;
  }
  else {
    this->year = 0;
    this->month = 0;
```

Correta

Nota desta submissão: 20/20

## Pergunta 4

Correta Pontuou 20 de 20

Consider a new definition of class Date, given in Date3.h, where the date is represented in a single data member (attribute) of `string` type, in the format `"yyyymmdd"` (ex: `20220401`, for the *1st of April of 2022*):

```
#include <iostream>
#include <string>

class Date {
  public:
    // constructors
    Date();
    Date(int year, int month, int day);
    Date(const std::string& year_month_day);
    // accessors
    int get_year() const;
    int get_month() const;
    int get_day() const;
    // other methods
    bool is_valid() const;
    void write() const;

  private:
    // compute the number of days of month
    static int num_days(int year, int month);
    // attribute
    std::string yyyymmdd;
};
```

Rewrite the member functions of the class `Date` of problem 3, keeping their signature.

Note: the class implementation is changed without requiring changes in the code that uses it (encapsulation).

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `Date d1; d1.write(); cout << (d1.is_valid() ? "" : "-invalid") << endl;` | `0001/01/01` |
| `Date d2(2022, 4, 31); d2.write(); cout << (d2.is_valid() ? "" : "-invalid") << endl;` | `0000/00/00-invalid` |
| `Date d3("2ooo/2/28"); d3.write(); cout << (d3.is_valid() ? "" : "-invalid") << endl;` | `0000/00/00-invalid` |
| `Date d4("1900/1/1"); d4.write(); cout << (d4.is_valid() ? "" : "-invalid") << endl;` | `1900/01/01` |
| `Date d5("2022#12#31"); d5.write(); cout << (d5.is_valid() ? "" : "-invalid") << endl;` | `0000/00/00-invalid` |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include "Date3.h"

using namespace std;

int days_in_month(int year, int month) {
    int feb = (month == 2 && year % 4 == 0 && (year % 100 != 0 || year % 400 == 0))? 29 : 28;
    int n_days[12] = {31, feb, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    return n_days[month - 1];
}

Date::Date(){
    yyyymmdd = "00010101";
}

Date::Date(int year, int month, int day){
    if(day <= days_in_month(year, month) && year >= 1 && year <= 9999){
        ostringstream oss;
        oss << setfill('0') << setw(4) << year << setw(2) << month << setw(2) << day;
        yyyymmdd = oss.str();
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `Date d1; d1.write(); cout << (d1.is_valid() ? "" : "-invalid") << endl;` | 0001/01/01 | 0001/01/01 | ✔ |
| ✔ | `Date d2(2022, 4, 31); d2.write(); cout << (d2.is_valid() ? "" : "-invalid") << endl;` | 0000/00/00-invalid | 0000/00/00-invalid | ✔ |
| ✔ | `Date d3("2ooo/2/28"); d3.write(); cout << (d3.is_valid() ? "" : "-invalid") << endl;` | 0000/00/00-invalid | 0000/00/00-invalid | ✔ |
| ✔ | `Date d4("1900/1/1"); d4.write(); cout << (d4.is_valid() ? "" : "-invalid") << endl;` | 1900/01/01 | 1900/01/01 | ✔ |
| ✔ | `Date d5("2022#12#31"); d5.write(); cout << (d5.is_valid() ? "" : "-invalid") << endl;` | 0000/00/00-invalid | 0000/00/00-invalid | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include "Date3.h"

using namespace std;

Date::Date() : yyyymmdd("00010101") { }

Date::Date(int year, int month, int day) {
  if (year >= 1 && year <= 9999 &&
    month >= 1 && month <= 12 &&
    day >= 1 && day <= num_days(year, month))
  {
    std::ostringstream oss;
    oss << setfill('0') << setw(4) << year
      << setw(2) << month
      << setw(2) << day;
    yyyymmdd = oss.str();
  }
  else {
    yyyymmdd = "00000000";
```

Correta

Nota desta submissão: 20/20

## Pergunta 5

Correta Pontuou 20 de 20

Create a class `Fraction` for performing arithmetic operations with fractions according to the specification below. You are free to name the attributes of the class, but the names and signatures of the member functions must be as stated.

- Use private `int` member fields to represent the numerator and the denominator.
- Provide two constructors for the class: a default constructor that initialises the numerator with `0` and denominator with `1`, and a constructor that takes two `int` arguments, corresponding to the numerator and denominator values. You may assume that the denominator is always different from `0`.
- The constructors must ensure that the fraction is stored in normalised form, that is, it must be irreducible and the denominator must always be positive. For example, the fractions 3/15, -3/-4 and 2/-4 must be stored, respectively, as 1/5, 3/4 and -1/2.
- Define public `numerator()` and `denominator()` accessors for the numerator and denominator fields.
- Provide public member functions to perform the basic arithmetic operations with fractions: sum, subtraction, multiplication and division, with the following corresponding signatures:

```
Fraction sum(const Fraction& right)
Fraction sub(const Fraction& right)
Fraction mul(const Fraction& right)
Fraction div(const Fraction& right)
```

The code of the following functions is given (see the preload code in Moodle):

- `Fraction::gcd`, to calculate the greatest common divisor of 2 numbers;
- `Fraction::normalise`, to normalise a fraction;
- `Fraction::write`, to write a fraction on the screen.

Note that you need to declare these functions in the `Fraction` class.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `Fraction().sum({2, 4}).write(); cout << ' ';`<br>`Fraction(1,1).sum({2, 4}).write(); cout << ' ';`<br>`Fraction(2,4).sum({3, 9}).write(); cout << ' ';`<br>`Fraction(-2,4).sum({1, 2}).write(); cout << ' ';`<br>`Fraction(3,27).sum({-27, 81}).write(); cout << '\n';` | `1/2 3/2 5/6 0/1 -2/9` |
| `Fraction().sub({2, 4}).write(); cout << ' ';`<br>`Fraction(1,1).sub({2, 4}).write(); cout << ' ';`<br>`Fraction(2,4).sub({3, 9}).write(); cout << ' ';`<br>`Fraction(-2,4).sub({1, 2}).write(); cout << ' ';`<br>`Fraction(3,27).sub({-27, 81}).write(); cout << '\n';` | `-1/2 1/2 1/6 -1/1 4/9` |
| `Fraction().mul({2, 4}).write(); cout << ' ';`<br>`Fraction(1,1).mul({2, 4}).write(); cout << ' ';`<br>`Fraction(2,4).mul({3, 9}).write(); cout << ' ';`<br>`Fraction(-2,4).mul({1, 2}).write(); cout << ' ';`<br>`Fraction(3,27).mul({-27, 81}).write(); cout << '\n';` | `0/1 1/2 1/6 -1/4 -1/27` |
| `Fraction().div({2, 4}).write(); cout << ' ';`<br>`Fraction(1,1).div({2, 4}).write(); cout << ' ';`<br>`Fraction(2,4).div({3, 9}).write(); cout << ' ';`<br>`Fraction(-2,4).div({1, 2}).write(); cout << ' ';`<br>`Fraction(3,27).div({-27, 81}).write(); cout << '\n';` | `0/1 2/1 3/2 -1/1 -1/3` |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

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

using namespace std;

//! define the Fraction class.
class Fraction {
  public:
    // constructors
    Fraction();
    Fraction(int numerator, int denominator);
    // accessors
    int get_numerator() const;
    int get_denominator() const;
```

```
    int get_denominator() const;
    // operations
    Fraction sum(const Fraction& right);
    Fraction sub(const Fraction& right);
    Fraction mul(const Fraction& right);
    Fraction div(const Fraction& right);
    // other  member functions
    void normalise();
    int gcd(int a, int b);
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `Fraction().sum({2, 4}).write(); cout << ' ';`<br>`Fraction(1,1).sum({2, 4}).write(); cout << ' ';`<br>`Fraction(2,4).sum({3, 9}).write(); cout << ' ';`<br>`Fraction(-2,4).sum({1, 2}).write(); cout << ' ';`<br>`Fraction(3,27).sum({-27, 81}).write(); cout << '\n';` | `1/2 3/2 5/6 0/1 -2/9` | `1/2 3/2 5/6 0/1 -2/9` | ✔ |
| ✔ | `Fraction().sub({2, 4}).write(); cout << ' ';`<br>`Fraction(1,1).sub({2, 4}).write(); cout << ' ';`<br>`Fraction(2,4).sub({3, 9}).write(); cout << ' ';`<br>`Fraction(-2,4).sub({1, 2}).write(); cout << ' ';`<br>`Fraction(3,27).sub({-27, 81}).write(); cout << '\n';` | `-1/2 1/2 1/6 -1/1 4/9` | `-1/2 1/2 1/6 -1/1 4/9` | ✔ |
| ✔ | `Fraction().mul({2, 4}).write(); cout << ' ';`<br>`Fraction(1,1).mul({2, 4}).write(); cout << ' ';`<br>`Fraction(2,4).mul({3, 9}).write(); cout << ' ';`<br>`Fraction(-2,4).mul({1, 2}).write(); cout << ' ';`<br>`Fraction(3,27).mul({-27, 81}).write(); cout << '\n';` | `0/1 1/2 1/6 -1/4 -1/27` | `0/1 1/2 1/6 -1/4 -1/27` | ✔ |
| ✔ | `Fraction().div({2, 4}).write(); cout << ' ';`<br>`Fraction(1,1).div({2, 4}).write(); cout << ' ';`<br>`Fraction(2,4).div({3, 9}).write(); cout << ' ';`<br>`Fraction(-2,4).div({1, 2}).write(); cout << ' ';`<br>`Fraction(3,27).div({-27, 81}).write(); cout << '\n';` | `0/1 2/1 3/2 -1/1 -1/3` | `0/1 2/1 3/2 -1/1 -1/3` | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
class Fraction {
public:
  Fraction();                           // default constructor
  Fraction(int num, int den);           // constructor from numerator and denominator values
  int numerator() const;                // return numerator value
  int denominator() const;              // return denominator value
  Fraction sum(const Fraction& right);  // sum this fraction with 'right'
  Fraction sub(const Fraction& right);  // subtract 'right' to this fraction
  Fraction mul(const Fraction& right);  // multiply this fraction with 'right'
  Fraction div(const Fraction& right);  // divide this fraction with 'right'
  void normalise();                     // normalise fraction (irreducible & negative sign in n
  void write() const;                   // shows fraction f on the screen
private:
  static int gcd(int a, int b);  // computes greatest common denominator of two integers
  // attributes
  int numerator_;    // fraction numerator
  int denominator_;  // fraction denominator
};

#include <iostream>
#include <cmath>

```

Correta

Nota desta submissão: 20/20

◄ T07 26/04

Ir para...

T08 10/05 ►