Início	terça, 7 de junho de 2022 às 15:10
Estado	Prova submetida
Data de submissão:	quinta, 9 de junho de 2022 às 09:45
Tempo gasto	1 dia 18 horas
Nota	**100** do máximo 100

Pergunta 1

Correta Pontuou 20 de 20

Write a C++ function `bool average(const string& input_fname, const string& output_fname)` that reads several series of `double` values, one series per line, stored in input file named `input_fname`, and outputs to file name `output_fname` corresponding lines with the average value of the series rounded to 3 decimal places. In the end, the function outputs the number of lines read.

The function returns `false` if it fails to open the input file and `true` otherwise. Note that `fail()` may be used to test if an error has occurred on the associated stream.

To test your code download the ex1a.zip archive containing the text files used in public tests (`p1a-[1-4].txt`). You may assume that the only blank characters contained in files are the space and newline character.

Por exemplo:

Teste	Resultado

`if (average("p1a-1.txt", "p1a-1_out.txt"))`
`  show_file("p1a-1_out.txt");`
	`==> p1a-1_out.txt <==`
	`1.629`
	`1.222`
	`1.695`
	`lines=3`

`if (average("p1a-2.txt", "p1a-2_out.txt"))`
`  show_file("p1a-2_out.txt");`
	`==> p1a-2_out.txt <==`
	`0.000`
	`26.528`
	`3.141`
	`lines=3`

`if (average("p1a-3.txt", "p1a-3_out.txt"))`
`  show_file("p1a-3_out.txt");`
	`==> p1a-3_out.txt <==`
	`0.000`
	`18.786`
	`lines=2`

`if (average("p1a-4.txt", "p1a-4_out.txt"))`
`  show_file("p1a-4_out.txt");`
	`==> p1a-4_out.txt <==`
	`1.629`
	`1.222`
	`0.848`
	`26.528`
	`3.141`
	`lines=5`

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

Limpar resposta

```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <cfloat>

using namespace std;

//! Show file name and its contents.
void show_file(const string& file) {
  ifstream in(file);
  cout << "==> " << file << " <==\n";
  for (string line; getline(in, line); ) cout << line << '\n';
}

bool average(const string& input_fname, const string& output_fname){
    ifstream reader(input_fname);
    ofstream writter(output_fname);

    int lines = 0;
    for(string line; getline(reader, line);){
        istringstream in_line(line);
```

	Teste	Esperado	Recebido	

	Teste	Esperado	Recebido	

✓	`if (average("p1a-1.txt", "p1a-1_out.txt"))`
	`  show_file("p1a-1_out.txt");`
		`==> p1a-1_out.txt <==`
		1.629
		1.222
		1.695
		lines=3
			`==> p1a-1_out.txt <==`
			1.629
			1.222
			1.695
			lines=3
				✓

✓	`if (average("p1a-2.txt", "p1a-2_out.txt"))`
	`  show_file("p1a-2_out.txt");`
		`==> p1a-2_out.txt <==`
		0.000
		26.528
		3.141
		lines=3
			`==> p1a-2_out.txt <==`
			0.000
			26.528
			3.141
			lines=3
				✓

✓	`if (average("p1a-3.txt", "p1a-3_out.txt"))`
	`  show_file("p1a-3_out.txt");`
		`==> p1a-3_out.txt <==`
		0.000
		18.786
		lines=2
			`==> p1a-3_out.txt <==`
			0.000
			18.786
			lines=2
				✓

✓	`if (average("p1a-4.txt", "p1a-4_out.txt"))`
	`  show_file("p1a-4_out.txt");`
		`==> p1a-4_out.txt <==`
		1.629
		1.222
		0.848
		26.528
		3.141
		lines=5
			`==> p1a-4_out.txt <==`
			1.629
			1.222
			0.848
			26.528
			3.141
			lines=5
				✓

Passou em todos os testes! ✓

Solução do autor da pergunta (C):

```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <cfloat>

using namespace std;

//! Show file name and its contents.
void show_file(const string& file) {
  ifstream in(file);
  cout << "==> " << file << " <==\n";
  for (string line; getline(in, line); ) cout << line << '\n';
}

bool average(const string& input_fname, const string& output_fname) {
  ifstream f_in(input_fname);
  if (f_in.fail())
    return false;
  ofstream f_out(output_fname);
  int lines = 0;
  for (string line; getline(f_in, line); ) {
```

Correta

Nota desta submissão: 20/20

Pergunta 2

Correta Pontuou 20 de 20

Write the C++ code for the `Student` class that represents a student in the Programming course, with the definition given in the `Student.h` header file.

```
class Student {
public:
  // constructor
  Student(const std::string& name, const std::string& id,
          short ac, short p1, short p2);
  // accessors
  std::string get_id() const;
  std::string get_name() const;
  double actual_grade() const; // [0..20]
private:
  std::string name_;
  std::string id_;
  short ac_, p1_, p2_; // [0..20]
};
```

The course has three assessment components: continuous assessment (ac), minitest 1 (p1) and minitest 2 (p2) and the final grade is obtained using the formula: 10% ac + 45% p1 + 45% p2.

Por exemplo:

Teste	Resultado
`Student s("Andre Meira", "up201404877", 20, 18, 10);` `cout << "[" << s.get_id() << "]" << "/";` `cout << fixed << setprecision(2) << s.actual_grade() << endl;`	`[up201404877]/14.60`
`Student s("John Doe", "up19790007", 20, 20, 20);` `cout << "[" << s.get_id() << "]" << "/";` `cout << fixed << setprecision(2) << s.actual_grade() << endl;`	`[up19790007]/20.00`
`Student s("Graham Chapman", "up19790077", 10, 10, 10);` `cout << "[" << s.get_id() << "]" << "/";` `cout << fixed << setprecision(2) << s.actual_grade() << endl;`	`[up19790077]/10.00`
`Student s("John Cleese", "up19790077", 8, 7, 13);` `cout << "[" << s.get_id() << "]" << "/";` `cout << fixed << setprecision(2) << s.actual_grade() << endl;`	`[up19790077]/9.80`

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

Limpar resposta

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

using namespace std;

Student::Student(const std::string& name, const std::string& id, short ac, short p1, short p2){
    name_ = name;
    id_ = id;
    ac_ = ac;
    p1_ = p1;
    p2_ = p2;
}

string Student::get_id() const{
    return id_;
}

string Student::get_name() const{
    return name_;
}

```

	Teste	Esperado	Recebido	

	Teste	Esperado	Recebido	

✓	`Student s("Andre Meira", "up201404877", 20, 18, 10);`
	`cout << "[" << s.get_id() << "]" << "/";`
	`cout << fixed << setprecision(2) << s.actual_grade() << endl;`
		[up201404877]/14.60	[up201404877]/14.60	✓

✓	`Student s("John Doe", "up19790007", 20, 20, 20);`
	`cout << "[" << s.get_id() << "]" << "/";`
	`cout << fixed << setprecision(2) << s.actual_grade() << endl;`
		[up19790007]/20.00	[up19790007]/20.00	✓

✓	`Student s("Graham Chapman", "up19790077", 10, 10, 10);`
	`cout << "[" << s.get_id() << "]" << "/";`
	`cout << fixed << setprecision(2) << s.actual_grade() << endl;`
		[up19790077]/10.00	[up19790077]/10.00	✓

✓	`Student s("John Cleese", "up19790077", 8, 7, 13);`
	`cout << "[" << s.get_id() << "]" << "/";`
	`cout << fixed << setprecision(2) << s.actual_grade() << endl;`
		[up19790077]/9.80	[up19790077]/9.80	✓

Passou em todos os testes! ✓

Solução do autor da pergunta (C):

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

using namespace std;

Student::Student(const string& name, const string& id,
                 short ac, short p1, short p2)
                 : name_(name), id_(id), ac_(ac), p1_(p1), p2_(p2) { }

string Student::get_id() const { return id_; }

string Student::get_name() const { return name_; }

double Student::actual_grade() const {
  return (ac_ * 0.1 + p1_ * 0.45 + p2_ * 0.45);
}

/*
  // private tests (1000 points each)
  {
    Student s("Terry Gilliam", "up19790777", 2, 3, 5);
```

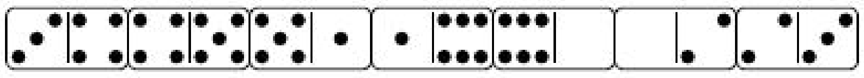
Correta

Nota desta submissão: 20/20

Pergunta 3

Correta Pontuou 20 de 20

Consider designing a class to partially implement a Domino game considering only a deck of pieces and the line of played pieces left-to-right "on the table". A domino piece has two sides, each one with a number in the range from '0' to '6'. The numbers may be equal on both sides of a piece and the complete set of pieces must not have repeated pieces. Nevertheless, when placed in the domino line, the piece (1:5) is different from the piece (5:1).



A piece of the domino game, when placed in the domino line, is represented by a class `Piece` as given in the header `Piece.h`.

```
class Piece {
public:
  // Constructor with parameters
  Piece(int left, int right) : left_(left), right_(right) { }
  // Determine if this piece can be placed on the left of the other piece
  bool can_be_left_to(const Piece& other) const;
  // Determine if this piece can be placed on the right of the other piece
  bool can_be_right_to(const Piece& other) const;
  // Accessors
  int get_left() const { return left_; }
  int get_right() const { return right_; }
  string to_string() const {
    ostringstream os;
    os << left_ << ':' << right_;
    return os.str();
  }
private:
  // The points in the two sides of the piece
  int left_, right_;
};
```

The class `Domino` to represent part of the game is defined and partially implemented as given in the header `Domino.h`:

```
class Domino {
public:
  // Constructor with parameters
  Domino(const list<Piece>& initial) : pieces_(initial) { }
  // Get the left end of the domino line
  const Piece& left() const;
  // Get the right end of the domino line
  const Piece& right() const;
  // Place piece in the left end
  bool place_left(const Piece& p);
  // Place piece in the right end
  bool place_right(const Piece& p);
  // Display the domino line of pieces
  string to_string() const {
    string s("[");
    for(auto p: pieces_)
      s.append(" ").append(p.to_string());
    s.append(" ]");
    return s;
  }
private:
  // The line of played pieces "on the table"
  list<Piece> pieces_;
};
```

Complete the code of classes `Piece` and `Domino` with the implementation of the six member functions not yet implemented.

Por exemplo:

Teste	Resultado

`Piece p (1, 2);`
`cout << boolalpha`
`     << p.can_be_left_to({2, 3}) << ' '`
`     << p.can_be_left_to({3, 2}) << ' '`
`     << p.can_be_right_to({3, 2}) << ' '`
`     << p.can_be_right_to({0, 1}) << '\n';`
	`true false false true`

Teste	Resultado

`Domino d( { {1,2}, {2,0}, {0,6}, {6,6} } );`
`cout << d.left().to_string() << ' '`
`     << d.right().to_string() << ' '`
`     << d.to_string() << '\n';`
	`1:2 6:6 [ 1:2 2:0 0:6 6:6 ]`

`Domino d( { {1,2}, {2,0}, {0,6}, {6,6} } );`
`cout << boolalpha`
`     << d.place_left({6,1})  << ' '`
`     << d.place_right({3,6}) << ' '`
`     << d.to_string() << '\n';`
	`true false [ 6:1 1:2 2:0 0:6 6:6 ]`

`Domino d( { {1,2}, {2,3}, {3,3}, {3,4} } );`
`cout << boolalpha`
`     << d.place_left({4,3})  << ' '`
`     << d.place_right({4,1}) << ' '`
`     << d.to_string() << '\n';`
	`false true [ 1:2 2:3 3:3 3:4 4:1 ]`

`Domino d( { {1,2}, {2,3}, {3,3}, {3,6} } );`
`cout << boolalpha`
`     << d.place_left({1,1})  << ' '`
`     << d.place_right({6,6}) << ' '`
`     << d.to_string() << '\n';`
	`true true [ 1:1 1:2 2:3 3:3 3:6 6:6 ]`

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

Limpar resposta

```cpp
#include <iostream>
#include <iomanip>
#include "Piece.h"
#include "Domino.h"
#include <iterator>

using namespace std;

bool Piece::can_be_left_to(const Piece& other) const{
    return right_ == other.left_;
}

bool Piece::can_be_right_to(const Piece& other) const{
    return left_ == other.right_;
}

const Piece& Domino::left() const{
    return pieces_.front();
}

const Piece& Domino:: right() const{
    return pieces_.back();
```

	Teste	Esperado	Recebido	

✓	`Piece p (1, 2);`
	`cout << boolalpha`
	`     << p.can_be_left_to({2, 3}) << ' '`
	`     << p.can_be_left_to({3, 2}) << ' '`
	`     << p.can_be_right_to({3, 2}) << ' '`
	`     << p.can_be_right_to({0, 1}) << '\n';`
		`true false false true`	`true false false true`	✓

✓	`Domino d( { {1,2}, {2,0}, {0,6}, {6,6} } );`
	`cout << d.left().to_string() << ' '`
	`     << d.right().to_string() << ' '`
	`     << d.to_string() << '\n';`
		`1:2 6:6 [ 1:2 2:0 0:6 6:6 ]`	`1:2 6:6 [ 1:2 2:0 0:6 6:6 ]`	✓

	Teste	Esperado	Recebido	

✓	`Domino d( { {1,2}, {2,0}, {0,6}, {6,6} } );`
	`cout << boolalpha`
	`     << d.place_left({6,1})  << ' '`
	`     << d.place_right({3,6}) << ' '`
	`     << d.to_string() << '\n';`
		`true false [ 6:1 1:2 2:0 0:6 6:6 ]`	`true false [ 6:1 1:2 2:0 0:6 6:6 ]`	✓

✓	`Domino d( { {1,2}, {2,3}, {3,3}, {3,4} } );`
	`cout << boolalpha`
	`     << d.place_left({4,3})  << ' '`
	`     << d.place_right({4,1}) << ' '`
	`     << d.to_string() << '\n';`
		`false true [ 1:2 2:3 3:3 3:4 4:1 ]`	`false true [ 1:2 2:3 3:3 3:4 4:1 ]`	✓

✓	`Domino d( { {1,2}, {2,3}, {3,3}, {3,6} } );`
	`cout << boolalpha`
	`     << d.place_left({1,1})  << ' '`
	`     << d.place_right({6,6}) << ' '`
	`     << d.to_string() << '\n';`
		`true true [ 1:1 1:2 2:3 3:3 3:6 6:6 ]`	`true true [ 1:1 1:2 2:3 3:3 3:6 6:6 ]`	✓

Passou em todos os testes! ✓

Solução do autor da pergunta (C):

```
#include <list>
#include <iostream>
#include <sstream>
#include <iomanip>
#include "Piece.h"
#include "Domino.h"

using namespace std;

//! Determine if this piece can be placed on the left of the other piece
bool Piece::can_be_left_to(const Piece& other) const {
  return right_ == other.left_;
}

//! Determine if this piece can be placed on the right of the other piece
bool Piece::can_be_right_to(const Piece& other) const {
  return other.right_ == left_;
}

//! get the piece on the left side of the domino line
const Piece& Domino::left() const {
  return pieces_.front();
```

Correta

Nota desta submissão: 20/20

Pergunta 4

Correta Pontuou 20 de 20

Write the C++ code for function `cat_keys`, declared as:

```
string cat_keys(list<map<string, unsigned>> lst);
```

The function should iterate the given list of maps to find the map that contains the minimum `unsigned` value, and returns a string that is a concatenation of all the keys in that map. See the public tests below for examples.

You may assume there is a single map with the minimum unsigned value. Note that `UINT_MAX`, defined in header `<climits>`, is the constant for the maximum value of an `unsigned` integer value.

Por exemplo:

Teste	Resultado

`list<map<string, unsigned>> m2 = {`
`  { {"s1", 13}, {"s2", 2} } };`
`cout << cat_keys(m2) << endl;`
	s1s2

`list<map<string, unsigned>> m1 = {`
`  { {"s1", 1} } };`
`cout << cat_keys(m1) << endl;`
	s1

`list<map<string, unsigned>> m3 = {`
`  { {"s1", 13}, {"s2", 2} },`
`  { {"s3", 3}, {"s4", 4} }, { {"s5", 16} } };`
`cout << cat_keys(m3) << endl;`
	s1s2

`list<map<string, unsigned>> m4 = {`
`  { {"s1", 13}, {"s2", 4} }, { {"s3", 3}, {"s4", 4} } };`
`cout << cat_keys(m4) << endl;`
	s3s4

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

Limpar resposta

```
#include <map>
#include <list>
#include <string>
#include <iostream>
#include <climits>
using namespace std;

string cat_keys(list<map<string, unsigned>> lst){
    unsigned max = UINT_MAX;
    int n = 0, i = 0;
    string result;
    for(map<string, unsigned> mzinho : lst){
        map<string, unsigned>::iterator it = mzinho.begin();
        while(it != mzinho.end()){
            if(it->second < max){
                n = i;
                max = it->second;
            }
            it++;
        }
        i++;
    }
```

	Teste	Esperado	Recebido	

✓	`list<map<string, unsigned>> m2 = {`
	`  { {"s1", 13}, {"s2", 2} } };`
	`cout << cat_keys(m2) << endl;`
		s1s2	s1s2	✓

✓	`list<map<string, unsigned>> m1 = {`
	`  { {"s1", 1} } };`
	`cout << cat_keys(m1) << endl;`
		s1	s1	✓

	Teste	Esperado	Recebido	

✓	`list<map<string, unsigned>> m3 = {`
	`  { {"s1", 13}, {"s2", 2} },`
	`  { {"s3", 3}, {"s4", 4} }, { {"s5", 16} } };`
	`cout << cat_keys(m3) << endl;`
		s1s2	s1s2	✓

✓	`list<map<string, unsigned>> m4 = {`
	`  { {"s1", 13}, {"s2", 4} }, { {"s3", 3}, {"s4", 4} } };`
	`cout << cat_keys(m4) << endl;`
		s3s4	s3s4	✓

Passou em todos os testes! ✓

Solução do autor da pergunta (C):

```
// Answer preload -->
#include <map>
#include <list>
#include <string>
#include <iostream>
#include <climits>
// <-- Answer preload

using namespace std;

string cat_keys(list<map<string, unsigned>> lst) {
  unsigned smallest = UINT_MAX;
  string smallest_concat;
  // traverse the list of maps
  for (auto m : lst) {
    string this_concat;
    unsigned this_smallest = UINT_MAX;
    // iterate over the map element of the list
    for (auto kv : m) {
      this_concat += kv.first;
      if (kv.second < this_smallest)
        this_smallest = kv.second;
```

Correta

Nota desta submissão: 20/20

Pergunta 5

Correta Pontuou 20 de 20

Consider the definition of an abstract class `Operation` given in header file Operation.h, that represents operations over two integers:

```
class Operation {
public:
  Operation(int op1, int op2) : op1_(op1), op2_(op2) { };
  int get_op1() const { return op1_; }
  int get_op2() const { return op2_; }
  virtual int operation() const = 0;  // operation
private:
  int op1_, op2_;  // two operands
};
```

Implement the definition of classes `Sum` and `Power` such that they implement, respectively, the addition of the two integers, and raising the first integer to the power of the second. Consider that the exponent argument to Power is always greater or equal than 0.

Por exemplo:

Teste	Resultado

`const Operation& s = Sum(2, 10); cout << s.operation() << ' ';`
`const Operation& p = Power(2, 10); cout << p.operation() << endl;`
	`12 1024`

`Sum s(6, -2); cout << s.operation() << ' ';`
`Power p(-3, 0); cout << p.operation() << endl;`
	`4 1`

`Sum s(-6, 2); cout << s.operation() << ' ';`
`Power p(-2, 5); cout << p.operation() << endl;`
	`-4 -32`

`Sum s(-6, -2); cout << s.operation() << ' ';`
`Power p(2, 7); cout << p.operation() << endl;`
	`-8 128`

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

Limpar resposta

```
#include <iostream>
#include "Operation.h"
#include <math.h>

using namespace std;

class Sum:public Operation{
    public:
        Sum(int op1, int op2) : Operation(op1, op2){}
        int operation() const override {return get_op1() + get_op2();}
};

class Power:public Operation{
    public:
        Power(int op1, int op2) : Operation(op1, op2){}
        int operation() const override {return pow(get_op1(),get_op2());}
};

```

	Teste	Esperado	Recebido	

✓	`const Operation& s = Sum(2, 10); cout << s.operation() << ' ';`
	`const Operation& p = Power(2, 10); cout << p.operation() << endl;`
		`12 1024`	`12 1024`	✓

✓	`Sum s(6, -2); cout << s.operation() << ' ';`
	`Power p(-3, 0); cout << p.operation() << endl;`
		`4 1`	`4 1`	✓

✓	`Sum s(-6, 2); cout << s.operation() << ' ';`
	`Power p(-2, 5); cout << p.operation() << endl;`
		`-4 -32`	`-4 -32`	✓

✓	`Sum s(-6, -2); cout << s.operation() << ' ';`
	`Power p(2, 7); cout << p.operation() << endl;`
		`-8 128`	`-8 128`	✓

Passou em todos os testes! ✓

Solução do autor da pergunta (C):

```
#include <iostream>
#include "Operation.h"

using namespace std;

class Sum : public Operation {
public:
  Sum(int op1, int op2) : Operation(op1, op2) { }
  int operation() const override {
    return get_op1() + get_op2();
  }
};

class Power : public Operation {
public:
  Power(int op1, int op2) : Operation(op1, op2) { }
  int operation() const override {
    int res = 1;
    for (int i = 0; i < get_op2(); i++)
      res *= get_op1();
    return res;
  }
```

Correta

Nota desta submissão: 20/20

◄ T11 31/05

Ir para...

T12 07/06 ►