| | |
|---|---|
| **Início** | quarta, 27 de abril de 2022 às 08:51 |
| **Estado** | Prova submetida |
| **Data de submissão:** | terça, 3 de maio de 2022 às 08:00 |
| **Tempo gasto** | 5 dias 23 horas |
| **Nota** | **80** do máximo 100 |

## Pergunta 1 Correta Pontuou 20 de 20

Write a C++ function `int count(const string& fname, const string& word)` such that `count(fname, word)` returns the number of occurrences of the word in the file named `fname`, where `word` is a string that does not contain blank characters (e.g., spaces, tabs, line breaks). The word count should be case insensitive (e.g. `"string"` and `"STRING"` should be considered equivalent).

To test your code download the ex1.zip archive containing 2 text files used in public tests (`p1_test_a.txt` and `p2_test_b.txt`).

Hint: you need to use an `ifstream` object in conjunction with the `>>` operator (the extraction operator) to read strings from the file ( `>>` skips over blank characters automatically). You may use `toupper` or `tolower` to convert characters to uppercase or lowercase respectively.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `cout << count("p1_test_a.txt", "THE") << '\n';` | 4 |
| `cout << count("p1_test_a.txt", "0") << '\n';` | 1 |
| `cout << count("p1_test_a.txt", "code") << '\n';` | 0 |
| `cout << count("p1_test_b.txt", "Collider") << '\n';` | 6 |
| `cout << count("p1_test_b.txt", "you") << '\n';` | 21 |

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

Limpar resposta

```cpp
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int count(const string& fname, const string& word){
    int count = 0;
    ifstream in(fname);
    string w;
    string up(word);

    for(char& c: up)
        c = toupper(c);

    while(in >> w){
        for(char& c : w)
            c = toupper(c);
        if(w == up) count++;
    }
    return count;
}
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✓ | `cout << count("p1_test_a.txt", "THE") << '\n';` | 4 | 4 | ✓ |
| ✓ | `cout << count("p1_test_a.txt", "0") << '\n';` | 1 | 1 | ✓ |
| ✓ | `cout << count("p1_test_a.txt", "code") << '\n';` | 0 | 0 | ✓ |
| ✓ | `cout << count("p1_test_b.txt", "Collider") << '\n';` | 6 | 6 | ✓ |
| ✓ | `cout << count("p1_test_b.txt", "you") << '\n';` | 21 | 21 | ✓ |

Passou em todos os testes! ✓

## Solução do autor da pergunta (C):

```cpp
#include <iostream>
#include <fstream>
#include <string>

using namespace std;
```

```

//! Compute the number of occurrences of the word in the file
int count(const string& filename, const string& word) {
  ifstream in(filename);
  string word_uc = word;
  for (char& c : word_uc) c = toupper(c);
  int n = 0;
  for (string s; in >> s;) {
    for (char& c : s) c = toupper(c);
    if (s == word_uc) n++;
  }
  return n;
}

/*
  // private (1000 points each)
  cout << count("p1_test_a.txt", "exerCISE") << '\n';  // -> 1
```

Correta

Nota desta submissão: 20/20

## Pergunta 2

Correta Pontuou 20 de 20

Write a C++ function `wcresult wc(const string& filename)` that reads a file and computes statistics that are similar to those computed by the "wc" Linux command line utility: the total number of lines, total number of words, and number of bytes. Any sequence of non-blank characters is considered a word, and `wcresult` is defined as:

```
struct wcresult {
  unsigned int lines;
  unsigned int words;
  unsigned int bytes;
};
```

To test your code download the ex2.zip archive containing the text files used in public tests (`p2_test[1-5].txt`). You may assume that the only blank characters contained in files are the space and newline character.

Hints: You should use an `ifstream` object to read a file and `getline` to read entire lines onto string objects. Employing `istringstream` may also be useful to break a line into words.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `wcresult r = wc("p2_test1.txt");`<br>`cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 1 0 1 |
| `wcresult r = wc("p2_test2.txt");`<br>`cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 6 13 92 |
| `wcresult r = wc("p2_test3.txt");`<br>`cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 1 9 44 |
| `wcresult r = wc("p2_test4.txt");`<br>`cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 4 9 51 |
| `wcresult r = wc("p2_test5.txt");`<br>`cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 39 188 1051 |

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

Limpar resposta

```
#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

struct wcresult {
  unsigned int lines;
  unsigned int words;
  unsigned int bytes;
};

wcresult wc(const string& filename){
    wcresult res = {0,0,0};
    ifstream in(filename);
    string line, wrd;
    while (getline(in, line)){
        istringstream stream(line);
        res.bytes += line.length() +1;
        while(stream >> wrd){
            res.words++;
        }
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✓ | `wcresult r = wc("p2_test1.txt");`<br>`cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 1 0 1 | 1 0 1 | ✓ |
| ✓ | `wcresult r = wc("p2_test2.txt");`<br>`cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 6 13 92 | 6 13 92 | ✓ |

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✓ | `wcresult r = wc("p2_test3.txt");` `cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 1 9 44 | 1 9 44 | ✓ |
| ✓ | `wcresult r = wc("p2_test4.txt");` `cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 4 9 51 | 4 9 51 | ✓ |
| ✓ | `wcresult r = wc("p2_test5.txt");` `cout << r.lines << ' ' << r.words << ' ' << r.bytes << '\n';` | 39 188 1051 | 39 188 1051 | ✓ |

Passou em todos os testes! ✓

## Solução do autor da pergunta (C):

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

using namespace std;

struct wcresult {
  unsigned int lines;
  unsigned int words;
  unsigned int bytes;
};

//! Computes statistics similar to those computed by the "wc".
wcresult wc(const string& filename) {
  ifstream in(filename);
  wcresult r { 0, 0, 0 };
  for (string line; getline(in, line);) {
    r.lines++;
    r.bytes += line.length() + 1;
    istringstream iss(line);
    for (string word; iss >> word;)
      r.words++;
```

Correta

Nota desta submissão: 20/20

## Pergunta 3

Correta

Pontuou 20 de 20

Write a C++ function `void normalise(const string& input_fname, const string& output_fname)` that normalises the contents of input file named `input_fname` and writes them onto an output file named `output_fname`. The input file contents should be converted to the output file as follows:

- Lines that contain only space characters should not be written to the output file;
- Leading and trailing spaces in a line should be erased;
- All characters should be uppercased.

To test your code download the ex3.zip archive containing the text files used in public tests (`p3_test[1-5].txt`). You may assume that the only blank characters contained in files are the space and newline character.

Hints: You should use an `ofstream` object to write the output file. You may use `toupper` to convert characters to uppercase.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `normalise("p3_test1.txt", "p3_test1_out.txt");`<br>`print("p3_test1_out.txt");` | 1\|WORD ONE\|<br>2\|WORD TWO\|<br>3\|WORD THREE SPACES FOLLOW\|<br>4\|AND ONE FINAL EMPTY LINE\| |
| `normalise("p3_test2.txt", "p3_test2_out.txt");`<br>`print("p3_test2_out.txt");` | 1\|C++\|<br>2\|LEIC UP\|<br>3\|PYTHON 3\|<br>4\|PYTHON 2\| |
| `normalise("p3_test3.txt", "p3_test3_out.txt");`<br>`print("p3_test3_out.txt");` | 1\|THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG\| |
| `normalise("p3_test4.txt", "p3_test4_out.txt");`<br>`print("p3_test4_out.txt");` | 1\|T H E\|<br>2\|Q U I C K  B R O W N\|<br>3\|F O X  J U M P S  O V E R  T H E\|<br>4\|L A Z Y\|<br>5\|D\|<br>6\|O\|<br>7\|G\| |
| `normalise("p3_test5.txt", "p3_test5_out.txt");`<br>`print("p3_test5_out.txt");` | 1\|ABCDEFGHIIJKLMOPQRSTUVWXYZ\|<br>2\|0123456789 0123456789 0123456789\|<br>3\|ABCDEFGHIIJKLMOPQRSTUVWXYZ\|<br>4\|0123456789 0123456789 0123456789\| |

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

Limpar resposta

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
using namespace std;

//! Show file contents with line number information and
//! the '|' character indicating the begining and end of lines.
void print(const std::string& file) {
  ifstream in(file);
  size_t line_nr = 1;
  for (string line; getline(in, line);) {
     cout << line_nr << "|" << line << "|\n";
     line_nr++;
  }
}

void strUpper(string& in){
  for(char& c : in) c = toupper(c);
  return;
}


```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✓ | `normalise("p3_test1.txt",`<br>`"p3_test1_out.txt");`<br>`print("p3_test1_out.txt");` | `1\|WORD ONE\|`<br>`2\|WORD TWO\|`<br>`3\|WORD THREE SPACES FOLLOW\|`<br>`4\|AND ONE FINAL EMPTY LINE\|` | `1\|WORD ONE\|`<br>`2\|WORD TWO\|`<br>`3\|WORD THREE SPACES FOLLOW\|`<br>`4\|AND ONE FINAL EMPTY LINE\|` | ✓ |
| ✓ | `normalise("p3_test2.txt",`<br>`"p3_test2_out.txt");`<br>`print("p3_test2_out.txt");` | `1\|C++\|`<br>`2\|LEIC UP\|`<br>`3\|PYTHON 3\|`<br>`4\|PYTHON 2\|` | `1\|C++\|`<br>`2\|LEIC UP\|`<br>`3\|PYTHON 3\|`<br>`4\|PYTHON 2\|` | ✓ |
| ✓ | `normalise("p3_test3.txt",`<br>`"p3_test3_out.txt");`<br>`print("p3_test3_out.txt");` | `1\|THE QUICK BROWN FOX JUMPS OVER`<br>`THE LAZY DOG\|` | `1\|THE QUICK BROWN FOX JUMPS OVER`<br>`THE LAZY DOG\|` | ✓ |
| ✓ | `normalise("p3_test4.txt",`<br>`"p3_test4_out.txt");`<br>`print("p3_test4_out.txt");` | `1\|T H E\|`<br>`2\|Q U I C K  B R O W N\|`<br>`3\|F O X  J U M P S  O V E R  T H`<br>`E\|`<br>`4\|L A Z Y\|`<br>`5\|D\|`<br>`6\|O\|`<br>`7\|G\|` | `1\|T H E\|`<br>`2\|Q U I C K  B R O W N\|`<br>`3\|F O X  J U M P S  O V E R  T H`<br>`E\|`<br>`4\|L A Z Y\|`<br>`5\|D\|`<br>`6\|O\|`<br>`7\|G\|` | ✓ |
| ✓ | `normalise("p3_test5.txt",`<br>`"p3_test5_out.txt");`<br>`print("p3_test5_out.txt");` | `1\|ABCDEFGHIIJKLMOPQRSTUVWXYZ\|`<br>`2\|0123456789 0123456789`<br>`0123456789\|`<br>`3\|ABCDEFGHIIJKLMOPQRSTUVWXYZ\|`<br>`4\|0123456789 0123456789`<br>`0123456789\|` | `1\|ABCDEFGHIIJKLMOPQRSTUVWXYZ\|`<br>`2\|0123456789 0123456789`<br>`0123456789\|`<br>`3\|ABCDEFGHIIJKLMOPQRSTUVWXYZ\|`<br>`4\|0123456789 0123456789`<br>`0123456789\|` | ✓ |

Passou em todos os testes! ✓

## Solução do autor da pergunta (C):

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
using namespace std;

//! Show file contents with line number information and
//! the '|' character indicating the begining and end of lines.
void print(const std::string& file) {
  ifstream in(file);
  size_t line_nr = 1;
  for (string line; getline(in, line);) {
     cout << line_nr << "|" << line << "|\n";
     line_nr++;
  }
}

//! Normalises the contents of input the file into the output file.
void normalise(const std::string& input_file, const std::string& output_file) {
  ifstream in(input_file);
  ofstream out(output_file);
  for (string line; getline(in, line);) {
    line.erase(0, line.find_first_not_of(' '));
```

Correta

Nota desta submissão: 20/20

## Pergunta 4

Correta Pontuou 20 de 20

Write a C++ function `double average(const string& fname)` such that `average(fname)` returns the average of the `double` values stored in a text file named `fname`. The file can contain characters that lead to errors when reading `double` values, and the function should recover from this by skipping one character at a time from the file input stream.

For example, if the file contents are

```
3.0 1.5 invalid_input3.0
```

then the average value computed should be `2.5` (the sum of the three numbers that occur in the file is 7.5).

To test your code download the ex4.zip archive containing the text files used in public tests (`p4_test[1-5].txt`).

Hints: You should use an `ifstream` object to read a file. In addition to the use of the extraction operator (`>>`) for reading `double` values, the following associated member functions in `ifstream` should be useful:

- `eof()` to test if the end of file has been reached;
- `clear()` to clear error flags in the `ifstream` object; and
- `ignore()` to skip a desired amount of characters.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `double m = average("p4_test1.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 2.50 |
| `double m = average("p4_test2.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 2.50 |
| `double m = average("p4_test3.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 0.75 |
| `double m = average("p4_test4.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 1.00 |
| `double m = average("p4_test5.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 3.50 |

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

Limpar resposta

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

using namespace std;

double average(const string& fname){
    int count = 0;
    double total = 0;
    ifstream file(fname);
    while(!file.eof()){
        double val;
        if(file >> val){
            total += val;
            count++;
        }
        else{
            file.clear();
            file.ignore(1);
        }
    }
    return total/count;
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✓ | `double m = average("p4_test1.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 2.50 | 2.50 | ✓ |
| ✓ | `double m = average("p4_test2.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 2.50 | 2.50 | ✓ |

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✓ | `double m = average("p4_test3.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 0.75 | 0.75 | ✓ |
| ✓ | `double m = average("p4_test4.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 1.00 | 1.00 | ✓ |
| ✓ | `double m = average("p4_test5.txt");` `cout << fixed << setprecision(2) << m << '\n';` | 3.50 | 3.50 | ✓ |

Passou em todos os testes! ✓

## Solução do autor da pergunta (C):

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

using namespace std;

//! Compute the average of values stored in a text file.
double average(const string& filename) {
  ifstream in(filename);
  double sum = 0;
  int n = 0;
  while (! in.eof()) {
    double x;
    if (in >> x) {  // got a double
      sum += x;
      n++;
    } else {          // skip invalid char
      in.clear();
      in.ignore(1);
    }
  }
  return sum / n;
```

Correta

Nota desta submissão: 20/20

## Pergunta 5

Incorreta Pontuou 0 de 20

Write a C++ function `void calc_medians(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 median value of the series.

Input files may have comment lines that begin with the `#` character that should be ignored, and each series of values begins with a string identifier followed by the values to consider in the same line separated by one or more space characters.

The output file should have one line per series with the corresponding identifier and median value with one decimal place of precision (you may use `fixed` and `setprecision` defined in the `iomanip` header to guarantee this).

For instance, given the input file with contents:

```
# a 1.2 1.0 this is a comment line
b 1.2 1.0
# another comment line
c 1.3
d -1.0 2.0 1.0
```

then the generated output file should have contents:

```
b 1.1
c 1.3
d 1.0
```

The median of a sequence of `n` sorted values `v[0], . . . , v[n-1]` is:

- `v[n/2]`, if `n` is odd; or
- `0.5 * (v[ n / 2 - 1] + v[ n / 2])`, if `n` is even.

Note that the values in each series in input files are not guaranteed to be sorted. To handle this detail, you may store read values in a vector `v` and then sort the values using `sort`, i.e. call `sort(v.begin(), v.end())` before calculating the median value.

To test your code download the ex5.zip archive containing the text files used in public tests (`p5_test[1-5].txt`).

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `calc_medians("p5_test1.txt", "p5_test1_out.txt");` `show_file("p5_test1_out.txt");` | `==> p5_test1_out.txt <==` `a 3.0` |
| `calc_medians("p5_test2.txt", "p5_test2_out.txt");` `show_file("p5_test2_out.txt");` | `==> p5_test2_out.txt <==` `b 1.1` `c 1.3` `d 1.0` |
| `calc_medians("p5_test3.txt", "p5_test3_out.txt");` `show_file("p5_test3_out.txt");` | `==> p5_test3_out.txt <==` `u1 1.5` `u2 2.0` `u3 2.0` `u4 -0.5` |
| `calc_medians("p5_test4.txt", "p5_test4_out.txt");` `show_file("p5_test4_out.txt");` | `==> p5_test4_out.txt <==` `___a 1.0` `___b 1.0` `___c 0.5` `___d 1.0` |
| `calc_medians("p5_test5.txt", "p5_test5_out.txt");` `show_file("p5_test5_out.txt");` | `==> p5_test5_out.txt <==` `___a 1.0` `___b 1.0` `___c 0.5` `___d 1.0` |

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

Limpar resposta

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
```

```
using namespace std;

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

```

Erro(s) de sintaxe

```
__tester__.cpp: In function ‘int main()’:
__tester__.cpp:48:4: error: ‘calc_medians’ was not declared in this scope
   48 |    calc_medians("p5_test1.txt", "p5_test1_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:53:4: error: ‘calc_medians’ was not declared in this scope
   53 |    calc_medians("p5_test2.txt", "p5_test2_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:58:4: error: ‘calc_medians’ was not declared in this scope
   58 |    calc_medians("p5_test3.txt", "p5_test3_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:63:4: error: ‘calc_medians’ was not declared in this scope
   63 |    calc_medians("p5_test4.txt", "p5_test4_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:68:4: error: ‘calc_medians’ was not declared in this scope
   68 |    calc_medians("p5_test5.txt", "p5_test5_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:73:4: error: ‘calc_medians’ was not declared in this scope
   73 |    calc_medians("p5_test6.txt", "p5_test6_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:78:4: error: ‘calc_medians’ was not declared in this scope
   78 |    calc_medians("p5_test7.txt", "p5_test7_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:83:4: error: ‘calc_medians’ was not declared in this scope
   83 |    calc_medians("p5_test8.txt", "p5_test8_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:88:4: error: ‘calc_medians’ was not declared in this scope
   88 |    calc_medians("p5_test9.txt", "p5_test9_out.txt");
      |    ^~~~~~~~~~~~
__tester__.cpp:93:4: error: ‘calc_medians’ was not declared in this scope
   93 |    calc_medians("p5_test10.txt", "p5_test10_out.txt");
      |    ^~~~~~~~~~~~
```

## Solução do autor da pergunta (C):

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>

using namespace std;

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

//! Read lines with a series of double values and write corresponding
//! lines with the median value
void calc_medians(const string& input_fname, const string& output_fname) {
  ifstream in(input_fname);
  ofstream out(output_fname);
  for (string line; getline(in, line); ) {
```

Incorreta

Nota desta submissão: 0/20

◄ T06 19/04

Ir para...

T07 26/04 ►