c1w2

28 сентября 2020 г.

0.1 task 1

Напишите функцию, которая называется Factorial, возвращает int, принимает int и возвращает факториал своего аргумента. Гарантируется, что аргумент функции по модулю не превышает 10. Для отрицательных аргументов функция должна возвращать 1.

Аргумент функции	Результат
1	1
-2	1
4	24

```
>>> int Factorial(int n) {
...     if (n < 0)
...         return 1;
...
...     int current = 1;
...     for (int i = 1; i <= n; ++i) {
...         current *= i;
...     }
...     return current;
... }
```

0.2 task2

Напишите функцию, которая

- называется IsPalindrom
- возвращает bool
- принимает параметр типа string и возвращает, является ли переданная строка палиндромом

Палиндром - это слово или фраза, которые одинаково читаются слева направо и справа налево. Пустая строка является палиндромом.

```
>>> bool IsPalindrom(string s) {
...     // Замечание: более правильным было бы использовать здесь тип size_t вместо int
...     // О причинах Вы узнаете на Жёлтом поясе
...     for (int i = 0; i < s.size() / 2; ++i) {
...         if (s[i] != s[s.size() - i - 1]) {
...             return false;
...         }
...     }
```

```
...     return true;
... }
```

0.3 task3

Напишите функцию UpdateIfGreater, которая принимает два целочисленных аргумента: first и second. Если first оказался больше second, Ваша функция должна записывать в second значение параметра first. При этом указанная функция не должна ничего возвращать, а изменение параметра second должно быть видно на вызывающей стороне.

```
>>> void UpdateIfGreater(int a, int& b) {
...     if (a > b) {
...         b = a;
...     }
... }
```

0.4 task4

Напишите функцию MoveStrings, которая принимает два вектора строк, source и destination, и дописывает все строки из первого вектора в конец второго. После выполнения функции вектор source должен оказаться пустым.

```
>>> void MoveStrings(vector<string>& source, vector<string>& destination)
... {
...     destination.insert(end(destination), begin(source), end(source));
...     source.clear();
... }
```

0.5 task5

Реализуйте функцию void Reverse(vector& v), которая переставляет элементы вектора в обратном порядке.

```
>>> void Reverse(vector<int>& vInt){
...     for (int i = 0; 2*i < vInt.size(); ++i)
...         swap(vInt[i], vInt[vInt.size() - i - 1]);
... }
```

0.6 task6

Даны значения температуры, наблюдавшиеся в течение N подряд идущих дней. Найдите номера дней (в нумерации с нуля) со значением температуры выше среднего арифметического за все N дней.

Гарантируется, что среднее арифметическое значений температуры является целым числом.

- Формат ввода Вводится число N, затем N неотрицательных целых чисел — значения температуры в 0-й, 1-й, ... $(N-1)$-й день.
- Формат вывода Первое число K — количество дней, значение температуры в которых выше среднего арифметического. Затем K целых чисел — номера этих дней.

```
>>> double MeanArithmetic (vector<int>& temperatures) {
...     double sum = 0;
...     for (const int& temperature : temperatures)
...         sum += temperature;
...     return sum / temperatures.size();
... }
...
...
... vector<int> FilterGreaterThanMean(vector<int>& temperatures)
... {
...     vector<int> filtered;
...     filtered.reserve(temperatures.size());
...     double meanTemperature = MeanArithmetic(temperatures);
...     for (int i = 0; i < temperatures.size(); ++i)
...         if (temperatures[i] > meanTemperature)
...             filtered.push_back(i);
...     return filtered;
... }
```

0.7 task6

Люди стоят в очереди, но никогда не уходят из её начала, зато могут приходить в конец и уходить оттуда. Более того, иногда некоторые люди могут прекращать и начинать беспокоиться из-за того, что очередь не продвигается.

Реализуйте обработку следующих операций над очередью:

- *WORRY i*: пометить i-го человека с начала очереди (в нумерации с 0) как беспокоящегося;
- *QUIET i*: пометить i-го человека как успокоившегося;
- *COME k*: добавить k спокойных человек в конец очереди;
- *COME* $-k$: убрать k человек из конца очереди;
- $WORRY_COUNT$: узнать количество беспокоящихся людей в очереди.

Изначально очередь пуста.

- Формат ввода Количество операций Q, затем описания операций.

Для каждой операции *WORRYi* и *QUIETi* гарантируется, что человек с номером i существует в очереди на момент операции.

Для каждой операции $COME - k$ гарантируется, что k не больше текущего размера очереди.

- Формат вывода Для каждой операции $WORRY_COUNT$ выведите одно целое число -количество беспокоящихся людей в очереди.

```
>>> #include <iostream>
... #include <string>
... #include <algorithm>
... #include <vector>
...
... using namespace std;
...
... void Worry(vector<bool>& worryQueue, size_t idx) {
...     worryQueue[idx] = true;
... }
...
... void Quiet(vector<bool>& worryQueue, size_t idx) {
```

```
...     worryQueue[idx] = false;
... }
...
... void Come(vector<bool>& worryQueue, size_t count) {
...     size_t currentSize = worryQueue.size();
...     worryQueue.resize(currentSize + count);
... }
...
... size_t WorryCount(const vector<bool>& worryQueue) {
...     return count(begin(worryQueue), end(worryQueue), true);
... }
...
...
... int main() {
...     vector<bool> worryQueue;
...     size_t instructionsCount;
...     cin >> instructionsCount;
...
...     string currentInstruction;
...     int currentInstructionArgument;
...
...     for (int i = 0; i < instructionsCount; ++i)
...     {
...         cin >> currentInstruction;
...         if (currentInstruction != "WORRY_COUNT")
...         {
...             cin >> currentInstructionArgument;
...             if (currentInstruction == "WORRY")
...                 Worry(worryQueue, currentInstructionArgument);
...             else if (currentInstruction == "QUIET")
...                 Quiet(worryQueue, currentInstructionArgument);
...             else if (currentInstruction == "COME")
...                 Come(worryQueue, currentInstructionArgument);
...             else
...                 throw invalid_argument("invalid instruction!");
...         } else {
...             cout << WorryCount(worryQueue) << endl;
...         }
...
...     }
...     return 0;
... }
```

0.8 task7

Слова называются анаграммами друг друга, если одно из них можно получить перестановкой букв в другом. Например, слово «eat» можно получить перестановкой букв слова «tea», поэтому эти слова являются анаграммами друг друга. Даны пары слов, проверьте для каждой из них, являются ли слова этой пары анаграммами друг друга.

- Указание Один из способов проверки того, являются ли слова анаграммами друг друга, заключается в следующем. Для каждого слова с помощью словаря подсчитаем, сколько раз в нём встречается каждая буква. Если для обоих слов эти словари равны (а это проверяется с помо-

щью обычного оператора ==), то слова являются анаграммами друг друга, в противном случае не являются.

При этом построение такого словаря по слову удобно вынести в отдельную функцию *BuildCharCounters*.

- Формат ввода Сначала дано число пар слов N, затем в N строках содержатся пары слов, которые необходимо проверить. Гарантируется, что все слова состоят лишь из строчных латинских букв.
- Формат вывода Выведите N строк: для каждой введённой пары слов YES, если эти слова являются анаграммами, и NO в противном случае.

```
>>> #include <iostream>
... #include <string>
... #include <algorithm>
... #include <sstream>
... #include <map>
...
... using namespace std;
...
... map<char, int> buildCharsCounter(const string& word)
... {
...     map<char, int> counter;
...     for (const auto& ch : word)
...         ++counter[ch];
...     return counter;
... }
...
... bool isAnagrams(const string& word1, const string& word2)
... {
...     return buildCharsCounter(word1) == buildCharsCounter(word2);
... }
...
...
... int main() {
...     size_t instCount;
...     cin >> instCount;
...
...     string word1, word2;
...     for (int i = 0; i < instCount; ++i)
...     {
...         cin >> word1 >> word2;
...         bool result = isAnagrams(word1, word2);
...         if (result)
...             cout << "YES" << endl;
...         else
...             cout << "NO" << endl;
...     }
...     return 0;
... }


>>>
```

```
>>> #include <iostream>
... #include <string>
... #include <map>
...
... using namespace std;
...
... int main() {
...   int q;
...   cin >> q;
...
...   map<string, string> country_to_capital;
...
...   for (int i = 0; i < q; ++i) {
...     string operation_code;
...     cin >> operation_code;
...
...     if (operation_code == "CHANGE_CAPITAL") {
...
...       string country, new_capital;
...       cin >> country >> new_capital;
...       if (country_to_capital.count(country) == 0) {
...         cout << "Introduce new country " << country << " with capital " << new_capital << endl;
...       } else {
...         const string& old_capital = country_to_capital[country];
...         if (old_capital == new_capital) {
...           cout << "Country " << country << " hasn't changed its capital" << endl;
...         } else {
...           cout << "Country " << country << " has changed its capital from " << old_capital << " to "
...         }
...       }
...       country_to_capital[country] = new_capital;
...
...     } else if (operation_code == "RENAME") {
...
...       string old_country_name, new_country_name;
...       cin >> old_country_name >> new_country_name;
...       if (old_country_name == new_country_name || country_to_capital.count(old_country_name) == 0 |
...         cout << "Incorrect rename, skip" << endl;
...       } else {
...         cout << "Country " << old_country_name << " with capital " << country_to_capital[old_country
...             " has been renamed to " << new_country_name << endl;
...         country_to_capital[new_country_name] = country_to_capital[old_country_name];
...         country_to_capital.erase(old_country_name);
...       }
...
...     } else if (operation_code == "ABOUT") {
...
...       string country;
...       cin >> country;
...       if (country_to_capital.count(country) == 0) {
...         cout << "Country " << country << " doesn't exist" << endl;
...       } else {
...         cout << "Country " << country << " has capital " << country_to_capital[country] << endl;
...       }
```

```
...
...     } else if (operation_code == "DUMP") {
...
...       if (country_to_capital.empty()) {
...         cout << "There are no countries in the world" << endl;
...       } else {
...         for (const auto& country_item : country_to_capital) {
...           cout << country_item.first << "/" << country_item.second << " ";
...         }
...         cout << endl;
...       }
...
...     }
...
...   }
...
...   return 0;
... }
```