

Лабораторная работа 6

Итераторы. Стандартные алгоритмы.

Часть 1.

Выполните следующие задачи по ссылке:

<https://informatics.msk.ru/mod/statements/view.php?id=82912#1>

Часть 2. Работу выполнять исключительно с использованием итераторов!

1. Напишите функцию `PrintVectorPart`, принимающую вектор целых чисел `numbers`, выполняющую поиск первого отрицательного числа в нём и выводящую в стандартный вывод все числа, расположенные левее найденного, в обратном порядке. Если вектор не содержит отрицательных чисел, выведите все числа в обратном порядке. (1 балл)

Пример кода

```
1 int main() {
2     PrintVectorPart({6, 1, 8, -5, 4});
3     cout << endl;
4     PrintVectorPart({-6, 1, 8, -5, 4}); // ничего не выведется
5     cout << endl;
6     PrintVectorPart({6, 1, 8, 5, 4});
7     cout << endl;
8     return 0;
9 }
10
```

Вывод

```
1 8 1 6
2
3 4 5 8 1 6
4
```

2. Напишите шаблонную функцию `FindGreaterElements`, принимающую множество `elements` объектов типа `T` и ещё один объект `border` типа `T` и возвращающую вектор из всех элементов множества, больших `border`, в возрастающем порядке. (1 балл)

Пример кода

```
1 int main() {
2     for (int x : FindGreaterElements(set<int>{1, 5, 7, 8}, 5)) {
3         cout << x << " ";
4     }
5     cout << endl;
6
7     string to_find = "Python";
8     cout << FindGreaterElements(set<string>{"C", "C++"}, to_find).size() << endl;
9     return 0;
10 }
11
```

Вывод

```
1 7 8
2 0
```

3. Напишите функцию SplitIntoWords, разбивающую строку на слова по пробелам. (1 балл)

```
1 vector<string> SplitIntoWords(const string& s);
```

Гарантируется, что:

- строка непуста;
- строка состоит лишь из латинских букв и пробелов;
- первый и последний символы строки не являются пробелами;
- строка не содержит двух пробелов подряд.

Подсказка

Рекомендуется следующий способ решения задачи:

- искать очередной пробел с помощью алгоритма find;
- создавать очередное слово с помощью конструктора строки по двум итераторам.

Пример кода

```
1 int main() {
2     string s = "C Cpp Java Python";
3
4     vector<string> words = SplitIntoWords(s);
5     cout << words.size() << " ";
6     for (auto it = begin(words); it != end(words); ++it) {
7         if (it != begin(words)) {
8             cout << "/";
9         }
10        cout << *it;
11    }
12    cout << endl;
13
14    return 0;
15 }
16
```

Вывод

```
1 4 C/Cpp/Java/Python
2
```

4. Напишите шаблонную функцию RemoveDuplicates, принимающую по ссылке вектор elements объектов типа T и удаляющую из него все дубликаты элементов. Порядок оставшихся элементов может быть любым.

Гарантируется, что объекты типа T можно сравнивать с помощью операторов ==, !=, < и >. (1 балл)

Пример кода

```
1 int main() {
2     vector<int> v1 = {6, 4, 7, 6, 4, 4, 0, 1};
3     RemoveDuplicates(v1);
4     for (int x : v1) {
5         cout << x << " ";
6     }
7     cout << endl;
8
9     vector<string> v2 = {"C", "C++", "C++", "C", "C++"};
10    RemoveDuplicates(v2);
11    for (const string& s : v2) {
12        cout << s << " ";
13    }
14    cout << endl;
15    return 0;
16 }
17
```

Вывод

```
1 6 4 7 0 1
2 C++ C
```

5. Дано целое положительное число N, не превышающее 9. Выведите все перестановки чисел от 1 до N в обратном лексикографическом порядке (см. пример). (1 балл)

Пример

Ввод

```
1 3
```

Вывод

```
1 3 2 1
2 3 1 2
3 2 3 1
4 2 1 3
5 1 3 2
6 1 2 3
7
```

Подсказка

Библиотека <algorithm> содержит готовые функции, позволяющие решить эту задачу.

6. (1 балл) В этой задаче вам необходимо вычислить различные демографические показатели для группы людей. Человек представляется структурой Person:

```

1 struct Person {
2     int age; // возраст
3     Gender gender; // пол
4     bool is_employed; // имеет ли работу
5 };
6

```

Тип Gender определён следующим образом:

```

1 enum class Gender {
2     FEMALE,
3     MALE
4 };
5

```

Вам необходимо написать функцию `PrintStats`, получающую вектор людей, вычисляющую и выводящую медианный возраст для каждой из следующих групп людей:

- все люди;
- все женщины;
- все мужчины;
- все занятые женщины;
- все безработные женщины;
- все занятые мужчины;
- все безработные мужчины.

Все 7 чисел нужно вывести в строгом соответствии с форматом (см. пример).

```

1 void PrintStats(vector<Person> persons);

```

Принимая вектор по значению (а не по константной ссылке), вы получаете возможность модифицировать его копию произвольным образом и тем самым проще произвести вычисления.

Подсказка

Используйте алгоритм `partition`.

Вычисление медианного возраста

Для вычисления медианного возраста группы людей вы должны использовать функцию `ComputeMedianAge`:

```
1 template <typename InputIt>
2 int ComputeMedianAge(InputIt range_begin, InputIt range_end);
```

Пример входных данных.

```
int main() {
    vector<Person> persons = {
        {31, Gender::MALE, false},
        {40, Gender::FEMALE, true},
        {24, Gender::MALE, true},
        {20, Gender::FEMALE, true},
        {80, Gender::FEMALE, false},
        {78, Gender::MALE, false},
        {10, Gender::FEMALE, false},
        {55, Gender::MALE, true},
    };
    PrintStats(persons);
    return 0;
}
```

Вывод

```
1 Median age = 40
2 Median age for females = 40
3 Median age for males = 55
4 Median age for employed females = 40
5 Median age for unemployed females = 80
6 Median age for employed males = 55
7 Median age for unemployed males = 78
8
```

7. (1 балл) Сортировка слиянием.

Напишите шаблонную функцию MergeSort, принимающую два итератора шаблонного типа RandomIt и сортирующую заданный ими диапазон с помощью сортировки слиянием. Гарантируется, что:

- итераторы типа RandomIt аналогичны по функциональности итераторам вектора и строки, то есть их можно сравнивать с помощью операторов <, <=, > и >=, а также вычитать и складывать с числами;
- сортируемые объекты можно сравнивать с помощью оператора <.

```
1 template <typename RandomIt>
2 void MergeSort(RandomIt range_begin, RandomIt range_end);
```

Часть 1. Реализация с разбиением на 2 части

Алгоритм

Классический алгоритм сортировки слиянием выглядит следующим образом:

1. Если диапазон содержит меньше 2 элементов, выйти из функции.
2. Создать вектор, содержащий все элементы текущего диапазона.
3. Разбить вектор на две равные части. *(В этой задаче гарантируется, что длина передаваемого диапазона является степенью двойки, так что вектор всегда можно разбить на две равные части.)*
4. Вызвать функцию MergeSort от каждой половины вектора.
5. С помощью алгоритма std::merge слить отсортированные половины, записав полученный отсортированный диапазон вместо исходного.

Вы должны реализовать **именно этот алгоритм** и никакой другой.

Подсказка

Чтобы создать вектор, содержащий все элементы текущего диапазона (п. 2 алгоритма), необходимо уметь по типу итератора узнавать тип элементов, на которые он указывает. Если итератор RandomIt принадлежит стандартному контейнеру (вектору, строке, множеству, словарию...), нижележащий тип можно получить с помощью выражения `typename RandomIt::value_type`. Таким образом, гарантируется, что создать вектор в п. 2 можно следующим образом:

```
1 vector<typename RandomIt::value_type> elements(range_begin, range_end);
```

Пример кода

```
1 int main() {
2     vector<int> v = {6, 4, 7, 6, 4, 4, 0, 1};
3     MergeSort(begin(v), end(v));
4     for (int x : v) {
5         cout << x << " ";
6     }
7     cout << endl;
8     return 0;
9 }
10
```

Вывод

```
1 0 1 4 4 4 6 7
```

Часть 2. Реализация с разбиением на 3 части

Реализуйте сортировку слиянием, разбивая диапазон на 3 равные части, а не на 2. Гарантируется, что длина исходного диапазона является степенью 3.

Соответственно, пункты 3–5 алгоритма нужно заменить следующими:

- Разбить вектор на 3 равные части.
- Вызвать функцию MergeSort от каждой части вектора.
- Слить первые две трети вектора с помощью алгоритма merge, сохранив результат во временный вектор с помощью back_inserter.
- Слить временный вектор из предыдущего пункта с последней третью вектора из п. 2, записав полученный отсортированный диапазон вместо исходного.

Пример кода

```
1 int main() {
2     vector<int> v = {6, 4, 7, 6, 4, 4, 0, 1, 5};
3     MergeSort(begin(v), end(v));
4     for (int x : v) {
5         cout << x << " ";
6     }
7     cout << endl;
8     return 0;
9 }
10
```

Вывод

```
1 0 1 4 4 4 5 6 6 7
```

8*. (2 балла) Личный бюджет

Реализуйте систему ведения личного бюджета. Вам необходимо обрабатывать запросы следующих типов:

- **ComputeIncome** *from to*: вычислить чистую прибыль за данный диапазон дат.
- **Earn** *from to value*: учесть, что за указанный период (равномерно по дням) была заработана сумма *value*.

Примечания:

- Во всех диапазонах *from to* обе даты *from* и *to* включаются.

Формат ввода

В первой строке вводится количество запросов **Q**, затем в описанном выше формате вводятся сами запросы, по одному на строке.

Формат вывода

Для каждого запроса **ComputeIncome** в отдельной строке выведите вещественное число — прибыль за указанный диапазон дат.

Примечание:

- используйте **std::cout.precision(25)** в вашем коде для единообразия формата вывода вещественных чисел

Ограничения

- Количество запросов Q — натуральное число, не превышающее 50.
- Все даты вводятся в формате YYYY-MM-DD. Даты корректны (с учётом високосных годов) и принадлежат интервалу с 2000 до 2099 гг.
- *value* — положительные целые числа, не превышающие 1000000.
- 1 секунда на обработку всех запросов.

Подсказка Используйте `std::accumulate` для подсчёта сумм.

Пример

Ввод

```
1 5
2 Earn 2000-01-02 2000-01-06 20
3 ComputeIncome 2000-01-01 2001-01-01
4 ComputeIncome 2000-01-01 2000-01-03
5 Earn 2000-01-03 2000-01-03 10
6 ComputeIncome 2000-01-01 2001-01-01
```

Вывод

```
1 20
2 8
3 30
```

9*. (1 балл) Построение арифметического выражения.

Часть 1

Реализуйте построение арифметического выражения согласно следующей схеме:

- изначально есть выражение, состоящее из некоторого целого числа x ;
- на каждом шаге к текущему выражению применяется некоторая операция: прибавление числа, вычитание числа, умножение на число или деление на число; перед применением операции выражение всегда должно быть заключено в скобки.

Пример

Изначально есть число 8, соответствующее выражение:

8

К нему применяется операция умножения на 3, получается выражение

$(8) * 3$

Затем вычитается 6:

$((8) * 3) - 6$

Наконец, происходит деление на 1; итоговое выражение:

$$(((8) * 3) - 6) / 1$$

Формат ввода

В первой строке содержится исходное целое число x . Во второй строке содержится целое неотрицательное число N — количество операций. В каждой из следующих N строк содержится очередная операция:

- прибавление числа a : $+ a$;
- либо вычитание числа b : $- b$;
- либо умножение на число c : $* c$;
- либо деление на число d : $/ d$.

Количество операций может быть нулевым — в этом случае необходимо вывести исходное число.

Формат вывода

Выведите единственную строку — построенное арифметическое выражение.

Обратите внимание на расстановку пробелов вокруг символов:

- каждый символ бинарной операции (+, -, * или /) должен быть окружён ровно одним пробелом с каждой стороны: $(8) * 3$;
- символ унарного минуса (для отрицательных чисел) не нуждается в дополнительном пробеле: -5 ;
- скобки и числа не нуждаются в дополнительных пробелах.

Подсказка

Для преобразования числа к строке используйте функцию `to_string` из библиотеки `<string>`.

Пример

Ввод

```
1 8
2 3
3 * 3
4 - 6
5 / 1
6
```

Вывод

```
1 (((8) * 3) - 6) / 1
```

Часть 2. Без лишних скобок

Модифицируйте решение предыдущей части так, чтобы предыдущее выражение обрамлялось скобками лишь при необходимости, то есть только в том случае, когда очередная операция имеет бóльший приоритет, чем предыдущая.

Ввод

```
1 8
2 3
3 * 3
4 - 6
5 / 1
6
```

Вывод

```
1 (8 * 3 - 6) / 1
```

10*. (2 балла) Группировка строк по префиксу

Часть 1. Группировка по символу

Напишите функцию FindStartsWith:

- принимающую отсортированный набор строк в виде итераторов `range_begin`, `range_end` и один символ `prefix`;
- возвращающую диапазон строк, начинающихся с символа `prefix`, в виде пары итераторов.

```
1 template <typename RandomIt>
2 pair<RandomIt, RandomIt> FindStartsWith(
3     RandomIt range_begin, RandomIt range_end,
4     char prefix);
```

Если итоговый диапазон пуст, его границы должны указывать на то место в контейнере, куда можно без нарушения порядка сортировки вставить любую строку, начинающуюся с символа `prefix` (подобно алгоритму `equal_range`). Гарантируется, что строки состоят лишь из строчных латинских букв и символ `prefix` также является строчной латинской буквой.

Поиск должен осуществляться за **логарифмическую сложность** — например, с помощью двоичного поиска.

Пример кода

```
1 int main() {
2     const vector<string> sorted_strings = {"moscow", "murmansk", "vologda"};
3
4     const auto m_result =
5         FindStartsWith(begin(sorted_strings), end(sorted_strings), 'm');
6     for (auto it = m_result.first; it != m_result.second; ++it) {
7         cout << *it << " ";
8     }
9     cout << endl;
10
11    const auto p_result =
12        FindStartsWith(begin(sorted_strings), end(sorted_strings), 'p');
13    cout << (p_result.first - begin(sorted_strings)) << " " <<
14        (p_result.second - begin(sorted_strings)) << endl;
15
16    const auto z_result =
17        FindStartsWith(begin(sorted_strings), end(sorted_strings), 'z');
18    cout << (z_result.first - begin(sorted_strings)) << " " <<
19        (z_result.second - begin(sorted_strings)) << endl;
20
21    return 0;
22 }
23
```

Вывод

```
1 moscow murmansk
2 2 2
3 3 3
4
```

Подсказка

К символам (char) можно прибавлять или вычитать числа, получая таким образом следующие или предыдущие буквы в алфавитном порядке. Например, для строки s выражение `--s[0]` заменит её первую букву на предыдущую.

Обратите внимание, что выражение `'a' + 1` имеет тип `int` и поэтому может понадобиться привести его к типу `char` с помощью `static_cast`.

Часть 2. Группировка по префиксу

Напишите более универсальный вариант функции `FindStartsWith`, принимающий в качестве префикса произвольную строку, состоящую из строчных латинских букв.

```
1 template <typename RandomIt>
2 pair<RandomIt, RandomIt> FindStartsWith(
3     RandomIt range_begin, RandomIt range_end,
4     const string& prefix);
5
```

Пример кода

```
1  ✓ int main() {  
2    const vector<string> sorted_strings = {"moscow", "motovilikha", "murmansk"};  
3  
4  ✓   const auto mo_result =  
5      | FindStartsWith(begin(sorted_strings), end(sorted_strings), "mo");  
6  ✓   for (auto it = mo_result.first; it != mo_result.second; ++it) {  
7      cout << *it << " ";  
8    }  
9    cout << endl;  
10  
11  ✓   const auto mt_result =  
12      | FindStartsWith(begin(sorted_strings), end(sorted_strings), "mt");  
13  ✓   cout << (mt_result.first - begin(sorted_strings)) << " " <<  
14      | (mt_result.second - begin(sorted_strings)) << endl;  
15  
16  ✓   const auto na_result =  
17      | FindStartsWith(begin(sorted_strings), end(sorted_strings), "na");  
18  ✓   cout << (na_result.first - begin(sorted_strings)) << " " <<  
19      | (na_result.second - begin(sorted_strings)) << endl;  
20  
21   return 0;  
22 }  
23
```

Вывод

```
1  moscow motovilikha  
2  2 2  
3  3 3  
4
```