Лабораторная работа 3.

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

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

Вывод

```
8 1 6

4 5 8 1 6

```

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, бóльших border, в возрастающем порядке. ***(1 балл)***

Пример кода

```
int main() {
  for (int x : FindGreaterElements(set<int>{1, 5, 7, 8}, 5)) {
    cout << x << " ";
  }
  cout << endl;

  string to_find = "Python";
  cout << FindGreaterElements(set<string>{"C", "C++"}, to_find).size() << endl;
  return 0;
}

```

Вывод

```
7 8
0

```

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

```
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

```
int main() {
  string s = "C Cpp Java Python";

  vector<string> words = SplitIntoWords(s);
  cout << words.size() << " ";
  for (auto it = begin(words); it != end(words); ++it) {
    if (it != begin(words)) {
      cout << "/";
    }
    cout << *it;
  }
  cout << endl;

  return 0;
}

```

Вывод

```
4 C/Cpp/Java/Python

```

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

```
int main() {
  vector<int> v1 = {6, 4, 7, 6, 4, 4, 0, 1};
  RemoveDuplicates(v1);
  for (int x : v1) {
    cout << x << " ";
  }
  cout << endl;

  vector<string> v2 = {"C", "C++", "C++", "C", "C++"};
  RemoveDuplicates(v2);
  for (const string& s : v2) {
    cout << s << " ";
  }
  cout << endl;
  return 0;
}
```

Вывод

```
6 4 7 0 1
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

```
3
```

Вывод

```
3 2 1
3 1 2
2 3 1
2 1 3
1 3 2
1 2 3
```

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:

```
struct Person {
  int age;  // возраст
  Gender gender;  // пол
  bool is_employed;  // имеет ли работу
};

```

Тип Gender определён следующим образом:

```
enum class Gender {
  FEMALE,
  MALE
};

```

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).

```
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:

```
template <typename InputIt>
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

```
Median age = 40
Median age for females = 40
Median age for males = 55
Median age for employed females = 40
Median age for unemployed females = 80
Median age for employed males = 55
Median age for unemployed males = 78

```

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 <.

```
template <typename RandomIt>
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, словарю...), нижележащий тип можно получить с помощью выражения typename RandomIt::value_type. Таким образом, гарантируется, что создать вектор в п. 2 можно следующим образом:

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

Пример кода

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

```

Вывод

```
0 1 4 4 4 6 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

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

```

Вывод

```
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

```
5
Earn 2000-01-02 2000-01-06 20
ComputeIncome 2000-01-01 2001-01-01
ComputeIncome 2000-01-01 2000-01-03
Earn 2000-01-03 2000-01-03 10
ComputeIncome 2000-01-01 2001-01-01
```

Вывод

```
20
8
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

```
8
3
* 3
- 6
/ 1

```

Вывод

```
(((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

```
8
3
* 3
- 6
/ 1

```

Вывод

```
(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.

```
template <typename RandomIt>
pair<RandomIt, RandomIt> FindStartsWith(
    RandomIt range_begin, RandomIt range_end,
    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

```
int main() {
  const vector<string> sorted_strings = {"moscow", "murmansk", "vologda"};

  const auto m_result =
      FindStartsWith(begin(sorted_strings), end(sorted_strings), 'm');
  for (auto it = m_result.first; it != m_result.second; ++it) {
    cout << *it << " ";
  }
  cout << endl;

  const auto p_result =
      FindStartsWith(begin(sorted_strings), end(sorted_strings), 'p');
  cout << (p_result.first - begin(sorted_strings)) << " " <<
      (p_result.second - begin(sorted_strings)) << endl;

  const auto z_result =
      FindStartsWith(begin(sorted_strings), end(sorted_strings), 'z');
  cout << (z_result.first - begin(sorted_strings)) << " " <<
      (z_result.second - begin(sorted_strings)) << endl;

  return 0;
}
```

Вывод

```
moscow murmansk
2 2
3 3
```

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.

```
template <typename RandomIt>
pair<RandomIt, RandomIt> FindStartsWith(
    RandomIt range_begin, RandomIt range_end,
    const string& prefix);
```

Пример кода

```
int main() {
  const vector<string> sorted_strings = {"moscow", "motovilikha", "murmansk"};

  const auto mo_result =
      FindStartsWith(begin(sorted_strings), end(sorted_strings), "mo");
  for (auto it = mo_result.first; it != mo_result.second; ++it) {
    cout << *it << " ";
  }
  cout << endl;

  const auto mt_result =
      FindStartsWith(begin(sorted_strings), end(sorted_strings), "mt");
  cout << (mt_result.first - begin(sorted_strings)) << " " <<
      (mt_result.second - begin(sorted_strings)) << endl;

  const auto na_result =
      FindStartsWith(begin(sorted_strings), end(sorted_strings), "na");
  cout << (na_result.first - begin(sorted_strings)) << " " <<
      (na_result.second - begin(sorted_strings)) << endl;

  return 0;
}

```

Вывод

```
moscow motovilikha
2 2
3 3

```