ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3
ОЧЕРЕДЬ

Оглавление

Описание задачи.

В рамках лабораторной работы необходимо изучить и реализовать одну из трёх структур (двухсвязный список, стек, очередь), в соответствии со своим вариантом, при этом, все структуры должны:

- Использовать шаблонный подход, обеспечивая работу контейнера с произвольными данными.
- Реализовывать свой итератор, предоставляющий стандартный для языка механизм работы с ним (для C++ это операции ++ и операция! =, для python это)
- Обеспечивать работу стандартных библиотек и конструкции for each если она есть в языке, если их нет, то реализовать собственную функцию использующую итератор.
- Проверку на пустоту и подсчет количества элементов.
- Операцию сортировки с использованием стандартной библиотеки.

Очередь должна реализовывать операции:

- добавление в конец
- взятие с начала

Для демонстрации работы структуры необходимо создать набор тестов (под тестом понимается функция, которая создаёт структуру, проводит операцию или операции над структурой и удаляет структуру):

- заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и максимального.
- Провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.
- заполнение контейнера 100 структур, содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры, содержащие результат фильтрации, проверить выполнение на правильность подсчётом кол-ва элементов, не подходящих под условие в новых структурах.
- Заполнить структуру 1000 элементов и отсортировать ее, проверить правильность использую структуру из стандартной библиотеки и сравнив результат.
- Инверсировать содержимое контейнера, заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

Описание метода/модели.

Очередь, как и стек является частным случаем списка, т.е. так же, как и стек как правило просто надстройкой над базовой реализацией списка, которая просто запрещает доступ к определенным операциям, и добавляя новые.

Очередь реализует следующую идею: “Первый пришел – Первый ушел”

Каждый элемент очереди состоит из поля хранящего значение элемента очереди, и поле указателя на следующий элемент очереди. Это соответствует однонаправленному списку, во всех отношениях, за исключением того, что извлекать значения из очереди можно только с одной стороны, причем не важно с какой, но при этом добавлять можно будет только в противоположную от места извлечения сторону.

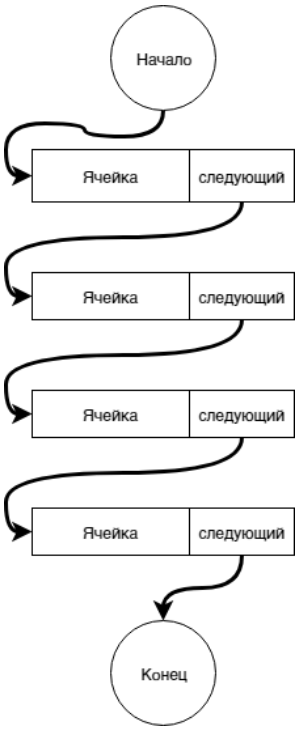

Рис.1. Очередь

Алгоритм Вставки

Алгоритм вставки выглядит следующим образом:

- Получаем новое значение
- Создаем новый элемент очереди с полями значения и указателя на последующий элемент
- Присваиваем значение к полю значения.
- Присваиваем у текущего последнего элемента в очереди к указателю на последующий элемент указатель на новосозданный элемент
- Присваиваем к указателю на текущий конечный элемент, указатель на новосозданный элемент

Алгоритм Удаления

Алгоритм удаления выглядит следующим образом:

- Берем первый(верхний) элемент очереди
- Сохраняем значение взятого элемента
- Записываем значение поля указателя на следующий элемент у взятого элемента в переменную указателя на текущий первый элемент списка
- Удаляем взятый элемент очереди
- Возвращаем сохраненное ранее значение

Использование

В каких ситуациях можно использовать очередь?

- Очереди подходят под описания множества жизненных ситуаций в которых требуется соблюдения честного порядка обработки каких-либо агентов, данных или ситуаций, таких как моделирования поведения покупателей, банковские и биржевые операции, моделирование трафика чего-либо.
- В компьютерной технике очереди используются для описания порядка предоставления процессорного времени различным программам, которые мы запускаем.
- Во многих языках программирования и Фреймворках используется модель событий, которые так же могут иметь очередность выполнения.
- Один из часто используемых паттернов проектирования это конвейер обработки данных, который можно представить, как очередь использования функций на какой-либо набор данных

Выполнение задачи.

Программа для получения тестовых данных была написана на C++.

Код:

```
#include <iostream>
#include <algorithm>
#include <queue>
#include <cstdlib>
#include <ctime>
#include <string>
#include <vector>
#include <numeric>
#include <fstream>
using namespace std;
//Очередь
template<typename T>
class Queue {

private:

    // Узел очереди
    class Node {
    public:
        T* value = nullptr;  // Значение
        Node* next = nullptr;  // Указатель на след. элемент
        Node(T value) {
            this->value = new T(value);
        }
    };
    Node* first = nullptr;  // Первый элемент
    Node* last = nullptr;  // Последний элемент
    int elems = 0;
public:
    // Итератор очереди
    class Iterator {
    public:
        using difference_type = std::ptrdiff_t;
        using value_type = T;
        using pointer = T*;
        using reference = T&;
        using iterator_category = std::random_access_iterator_tag;


    Node* current_node;  // Указатель на узел

    public:
        Iterator(Node* pointer_node) : current_node(pointer_node) {}

        /// <summary>
        /// Перегрузка постфиксного инкремента
        /// </summary>
        /// <param name="">int</param>
        void operator++(int) {
            this->current_node = this->current_node->next;

        }

        /// <summary>
        /// Перегрузка префиксного инкремента
        /// </summary>
        /// <returns>Iterator&</returns>
        Iterator& operator++() {
            this->current_node = this->current_node->next;
            return (*this);
        }
        /// <summary>
```

```
        /// Перегрузка оператора разыменования
        /// </summary>
        /// <returns>разыменованное значение</returns>
        T& operator*() {
            return *(this->current_node->value);

        }

        /// <summary>
        /// Перегрузка оператора равенства
        /// </summary>
        /// <param name="b">Итератор на сравниваемый элемент</param>
        /// <returns>bool</returns>
        bool operator==(const Iterator& b) const {
            return this->current_node == b.current_node;
        }

        /// <summary>
        /// Перегрузка оператора неравенства
        /// </summary>
        /// <param name="b">Итератор на сравниваемый элемент</param>
        /// <returns>bool</returns>
        bool operator!=(const Iterator& b) const {
            return this->current_node != b.current_node;
        }


    };

    /// <summary>
    /// Добавление элемента в конец
    /// </summary>
    /// <param name="value">значение</param>
    void push(T value) {
        Node* new_node = new Node(value);//Создаем новый элемент очереди
        if (this->last == nullptr) {//Проверяем не является очередь пустой
            this->first = new_node;
            this->last = new_node;
        }
        else {//в остальных случаях
            this->last->next = new_node;//Ставим у текущего последнего элемента в очереди к
указателю на последующий элемент указатель на новосозданный элемент
            this->last = new_node;//Присваиваем к указателю на текущий конечный элемент,
указатель на новосозданный элемент
        }

        this->elems++;
    }

    /// <summary>
    /// Изъетие элемента из конца
    /// </summary>
    void pop() {
        if (this->last == nullptr) {//Проверяем не является очередь пустой
            throw std::out_of_range("Не осталось элементов для удаления");
        }
        else {
            Node* temp = this->first;//Берем первый(верхний) элемент очереди
            this->first = this->first->next;//Записываем значение поля указателя на следующий
элемент у взятого элемента в переменную указателя на текущий первый элемент списка
            if (this->first == nullptr) {
                this->last = nullptr;
            };
            delete(temp);//Удаляем взятый элемент очереди
            this->elems--;
        }

    }
    /// <summary>
```

```
    /// Метод возврата итератора на начало
    /// </summary>
    /// <returns>Iterator</returns>
    Iterator begin() {
        return Iterator(this->first);
    }

    /// <summary>
    /// Метод возврата итератора на конец
    /// </summary>
    /// <returns>Iterator</returns>
    Iterator end() {
        return nullptr;
    }
    /// <summary>
    /// Получение первого элемента
    /// </summary>
    /// <returns>первый элемент</returns>
    T front() {
        return *(this->first->value);
    }
    /// <summary>
    /// Размер очереди
    /// </summary>
    /// <returns>значение размера очередиа</returns>
    int size() {
        return this->elems;
    }
    /// <summary>
    /// Проверка на пустоту
    /// </summary>
    /// <returns>bool</returns>
    bool empty() {
        return this->first ? false : true;
    }
    /// <summary>
    /// Сортировка очереди
    /// </summary>
    void sort() {
        vector<T> temp_vector;
        for (auto el : *this) {//Заполняем вектор элементами очереди
            temp_vector.push_back(el);
        }
        std::sort(temp_vector.begin(), temp_vector.end());//Сортируем вектор
        int temp_elems = elems;
        for (int i = 0; i < temp_elems; i++) {//Очищаем очередь
            this->pop();
            this->elems--;
        }
        for (int i = 0; i < temp_elems; i++) {//Заполняем очередь элементами отсортированного
вектора
            this->push(temp_vector[i]);
        }
    }
  };
/// <summary>
/// Тест1
/// </summary>
void test_1() {
    std::ofstream out;
    out.open("Test1.txt");
    if (out.is_open())
    {
        Queue<int> queue;
        const int AMOUNT = 1000;

        const int MIN = -1000;
        const int MAX = 1000;
```

```
        for (int i = 0; i < AMOUNT; i++) {
            queue.push(MIN + rand() % (MAX - MIN + 1));
        }
        int sum = std::accumulate(queue.begin(), queue.end(), 0, std::plus<int>());
        out << "Сумма: " << sum << endl;
        int max = *max_element(queue.begin(), queue.end());
        int min = *min_element(queue.begin(), queue.end());
        out << "Минимальный элемент: " << min << "\nМаксимальный элемент: " << max <<
"\nСредний элемент: " << sum / queue.size() << endl;

    }
    out.close();
}

/// <summary>
/// Тест2
/// </summary>
void test_2() {
    std::ofstream out;
    out.open("Test2.txt");

    if (out.is_open())
    {
        Queue<string> queue;
        const int AMOUNT = 10;
        const string AMOUNTSTRING[AMOUNT] = { "Midnight calling", "Mist of resolving", "Crown
me with the", "Pure green leaf", "Praise to my father", "Blessed by the water" , "Black night,
dark sky", "The devil's cry", "Bless me with the", "Leaf of the world tree" };
        out << "--------------Изначальный массив--------------" << endl;
        for (int i = 0; i < AMOUNT; i++) {
            queue.push(AMOUNTSTRING[i]);
        }
        for (auto el : queue) {
            out << el << endl;
        }
        out << "--------------С добавленным элементом--------------" << endl;
        queue.push("On it I see");
        for (auto el : queue) {
            out << el << endl;
        }
        out << "--------------С изъятым элементом--------------" << endl;
        queue.pop();
        queue.pop();
        for (auto el : queue) {
            out << el << endl;
        }
    }
    out.close();
}
//Структура даты
struct Date {
    int day;
    int month;
    int year;
};

//Структура человека
struct Person {
    string name;
    string surname;
    string patronymic;
    Date birth;
};
/// <summary>
/// Тест3
/// </summary>
void test_3() {
    std::ofstream out;
    out.open("Test3.txt");
```

```
    if (out.is_open())
    {
        const int DAY = 86400;
        const int AMOUNT = 100;
        const int NAMES_POOL = 7;
        // Данные для генерации
        const string NAMES[NAMES_POOL] = { "Peter", "Betty", "Miguel", "Rendy", "Hobby",
"Paul", "J.M." };
        const string SURNAMES[NAMES_POOL] = { "O`Harra", "Parker", "Brant", "Robertson",
"Brown", "Jenkins", "Dematees" };
        const string PATRONYMICS[NAMES_POOL] = { "Grigirovich", "Ivanovna", "Petrovich",
"Fedorovna", "Pavlova", "Vladislavovich", };


        // Данные для генерации даты:
        const int DAY_MAX = 31;
        const int MONTH_MAX = 12;
        const int YEAR_MIN = 1980;
        const int YEAR_MAX = 2020;

        Queue<Person> queue;

        // Генерация
        for (int i = 0; i < AMOUNT; i++) {

            queue.push(Person{
                NAMES[rand() % NAMES_POOL],
                SURNAMES[rand() % NAMES_POOL],
                PATRONYMICS[rand() % NAMES_POOL],
                Date{
                    (1 + rand() % DAY_MAX),
                    (1 + rand() % MONTH_MAX),
                    (YEAR_MIN + rand() % (YEAR_MAX - YEAR_MIN + 1))
                }
                });

        }
        for (auto el : queue) {
            out << el.name << " " << el.surname << " " << el.patronymic << " ";
            out << el.birth.day << ":" << el.birth.month << ":" << el.birth.year << endl;
        }
        Queue<Person> older_30;
        Queue<Person> younger_20;


        time_t time_now = time(0);

        // Фильтрация
        for (int i = queue.size(); i > 0; i--) {
            queue.pop();
            Person temp_q = queue.front();


            tm dateBirth = { 0, 0, 0, temp_q.birth.day, temp_q.birth.month - 1,
temp_q.birth.year - 1900, 0, 0, 0 };

            time_t t_birth = mktime(&dateBirth);

            time_t daysDifference = ((t_birth > time_now) ? t_birth - time_now : time_now -
t_birth) / DAY;
            daysDifference /= 365;


            if (daysDifference < 20) {
                younger_20.push(temp_q);
            }
            else if (daysDifference > 30) {
                older_30.push(temp_q);
```

```
            }

            else {
                queue.push(temp_q);
            }
        }
        out << "\n---------------Test3---------------" << endl;
        out << "Младше 20: " << younger_20.size() << "\nСтарше 30: " << older_30.size() <<
"\nОстальные: " << queue.size() << endl;
        out << "\nПроверка выполнения: " << younger_20.size() << " + " << older_30.size() << "
+ " << queue.size() << " = " << AMOUNT << "? - ";
        (((younger_20.size() + older_30.size() + queue.size()) == AMOUNT) ? (out <<
"Правильно") : (out << "Неправильно"));
    }
    out.close();
}

/// <summary>
/// Тест4
/// </summary>
void test_4() {
    std::ofstream out;
    out.open("Test4.txt");

    if (out.is_open())
    {
        Queue<int> queue1;
        deque<int> queue_test;

        const int AMOUNT = 1000;

        const int MIN = -1000;
        const int MAX = 1000;
        //Заполнение обоих очередей
        for (int i = 0; i < AMOUNT; i++) {
            int r = MIN + rand() % (MAX - MIN + 1);
            queue_test.push_back(r);
            queue1.push(r);

        }
        out << "------------------Заполнение до Сортировки кастомного контейнера-------------
-----------" << endl;
        for (auto el : queue1) {
            out << el << " ";
        }
        out << endl;
        out << "------------------Заполнение до Сортировки stl контейнера---------------------
----" << endl;
        for (auto el : queue_test) {
            out << el << " ";
        }
        out << endl;
        queue1.sort();
        sort(queue_test.begin(), queue_test.end());
        out << "------------------Заполнение после Сортировки кастомного контейнера-----------
--------------" << endl;
        for (auto el : queue1) {
            out << el << " ";
        }
        out << endl;
        out << "------------------Заполнение после Сортировки stl контейнера-----------------
-------" << endl;
        for (auto el : queue_test) {
            out << el << " ";
        }
        // Проверка поэлементная
        bool is_correct = true;

        for (int i = 0; i < queue1.size(); i++) {
```

```
            if (queue1.front() != queue_test.front()) {
                is_correct = false;
                break;
            }
            queue_test.pop_front();
        }
        out << endl << "Результат:" << endl;
        if (is_correct) {
            out << "Очереди отсортированы одинаково" << endl;
        }
        else {
            out << "Очереди отсортированы по-разному" << endl;
        }
    }
    out.close();
}
/// <summary>
/// Тест5
/// </summary>
void test_5() {
    std::ofstream out;
    out.open("Test5.txt");

    if (out.is_open())
    {
        Queue<int> queue1;
        const int AMOUNT = 1000;

        const int MIN = -1000;
        const int MAX = 1000;

        // Заполняем
        for (int i = 0; i < AMOUNT; i++) {
            int r = MIN + rand() % (MAX - MIN + 1);
            queue1.push(r);

        }
        //Сортировка
        queue1.sort();
        out << "--------------------До Риверсии---------------------" << endl;
        for (auto el : queue1) {
            out << el << " ";
        }
        int temp_array[AMOUNT];//Создание массива для хранения элементов
        for (int i = 0; i < AMOUNT; i++) {//Заполнение массива
            temp_array[i] = queue1.front();
            queue1.pop();
        }
        for (int i = AMOUNT - 1; i >= 0; i--) {//Обратный обход этого массива
            queue1.push(temp_array[i]);//Заполнение очереди в обратном порядке
        }
        out << endl << endl;
        out << "--------------------После Риверсии--------------------" << endl;
        for (auto el : queue1) {
            out << el << " ";
        }

    }
    out.close();
}
/// <summary>
/// Программа, реализующая работу и тестирование класса Очереди
/// </summary>
/// <returns></returns>
int main() {
    srand(time(0));
    setlocale(LC_ALL, "Russian");
    test_1();
    test_2();
```

```
  test_3();
  test_4();
  test_5();
}
```

Реализация:

1)Создаю класс Очереди и внутри него класс Node, который будет в данной реализации отвечать за роль узла. Он содержит поле со значением и указателем на следующий элемент. Сам класс очереди содержит три главных поля elems(количество элементов), first(указатель на первый элемент) и last(указатель на последний элемент).

2) Создаю метод push (отвечает за добавление новых элементов в конец очереди), где после проверки не является ли очередь пустой, создаю новый элемент для очереди, ставлю у текущего последнего элемента в очереди указатель на новый следующий и присваиваю к указателю на текущий конечный элемент новосозданный конечный элемент.

3)Создаю метод pop(отвечает за удаление элемента с начала очереди), где после проверки на пустоту, в случае которой выбрасываю ошибку, я беру первый элемент очереди и записываю в поле указателя на текущий первый элемент указатель на следующий за взятым нами. Удаляю взятый нами элемент.

4)Создаю методы front, size, empty, где возвращаю первый элемент, размер и отвечаю пустая ли очередь соответственно.

5)Создаю метод sort, который сортирует очередь. Сортирую я её используя вектор, так как функция std требует наличия итераторов произвольного доступа, которых не может быть у очереди. Записываю в вектор значения из очереди, сортирую их, очищаю вектор и возвращаю в него уже отсортированные значения.

6)Создаю итератор для моей очереди. Сначала пишу методы возврата итератора на начало(begin) и на конец(end). Внутри Очереди создаю класс итератора, где по шаблону прописываю его тип, категорию и тд. Создаю поле указателя на узел, к которому буду обращаться для получения данного экземпляра. А после последовательно перезагружаю все основные итераторы, необходимые для работы for each и стандартных функций библиотеки:

- ++ пост/префиксный, в которых просто нынешнему узлу передаю указатель на следующий
- == где возвращаю равен ли данный элемент аргументу
- != где возвращаю не равен ли данный элемент аргументу

7)Пишу тесты для проверки контейнера:

- заполняю контейнер 1000 целыми числами в диапазоне от -1000 до 1000 и подсчитываю их сумму, среднего, минимального и максимального.
- провожу проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.
- заполняю контейнера 100 структур содержащих фамилию, имя, отчество и дату рождения(от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение нахожу всех людей младше 20 лет и старше 30 и создаю новые структуры, содержащие результат фильтрации, проверяю на правильность подсчётом кол-ва элементов, не подходящих под условие в новых структурах.
- заполняю структуру 1000 элементов и сортирую ее, проверяю правильность используя структуру из стандартной библиотеки и сравнивая результат.
- инверсирую содержимое контейнера, заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

Результаты:

1)Заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и мак Сумма: -16064:

Минимальный элемент: -999
Максимальный элемент: 998
Средний элемент: -16

2) Провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.

- Изначальный массив:

Midnight calling
Mist of resolving
Crown me with the
Pure green leaf
Praise to my father
Blessed by the water
Black night, dark sky
The devil's cry
Bless me with the
Leaf of the world tree

- С добавленным элементом:

Midnight calling
Mist of resolving
Crown me with the
Pure green leaf
Praise to my father
Blessed by the water
Black night, dark sky
The devil's cry
Bless me with the
Leaf of the world tree
On it I see

- С изъятым элементом

Crown me with the
Pure green leaf
Praise to my father
Blessed by the water
Black night, dark sky
The devil's cry
Bless me with the
Leaf of the world tree
On it I see

3) заполнение контейнера 100 структур, содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры, содержащие результат фильтрации, проверить выполнение на правильность подсчётом кол-ва элементов не подходящих под условие в новых структурах.

- Часть данных:

Betty O`Harra Fedorovna 7:3:2007
Paul Brant Grigirovich 1:4:1986
J.M. Brown 25:12:2011
Rendy Dematees Grigirovich 17:1:2007
Paul Jenkins Fedorovna 22:11:1989
Hobby O`Harra Vladislavovich 15:10:2002
J.M. Jenkins Vladislavovich 21:4:1994

J.M. Dematees 10:1:2014
Betty O`Harra Grigirovich 15:6:2012
J.M. Jenkins 5:12:2002
Paul Robertson Vladislavovich 15:1:1980
J.M. Robertson Petrovich 24:12:2018
Betty Brown Pavlova 29:10:2013
J.M. Brown Grigirovich 14:5:2004
Miguel Robertson Grigirovich 9:3:2016
Betty Jenkins Petrovich 11:5:2019
J.M. Parker Petrovich 15:3:2003
Peter O`Harra Ivanovna 26:3:2009
J.M. Dematees 26:12:1980
Paul O`Harra 10:6:2005
Miguel Brown Pavlova 25:11:1992
Paul Jenkins Ivanovna 2:2:2010
Hobby Parker Petrovich 1:4:2012
Miguel Dematees Vladislavovich 13:9:1994
Rendy Dematees Fedorovna 28:10:1984
Betty O`Harra Ivanovna 11:10:1999
Peter O`Harra 4:4:2011
Miguel Parker Ivanovna 27:2:1986
J.M. Robertson 7:1:1987
J.M. Parker Grigirovich 16:11:2002
Peter Parker Petrovich 15:2:2019
Betty Parker Ivanovna 11:10:1986
Hobby Parker 13:1:1991
J.M. Brant Fedorovna 16:1:2006
Hobby O`Harra Fedorovna 8:4:2008
Paul Dematees 17:5:2019
Rendy Parker 10:7:2010

- Полученные результаты:
  Младше 20: 50
  Старше 30: 26
  Остальные: 24
- Результат проверки выполнения:
  Проверка выполнения: 50 + 26 + 24 = 100? – Правильно

4) Заполнить структуру 1000 элементов и отсортировать ее, проверить правильность использую структуру из стандартной библиотеки и сравнив результат.

- Заполнение до Сортировки кастомного контейнера:
  161 -38 211 -372 916 -17 461 700 719 … 127 509 279 351 206 -127 -284 -820 626
- Заполнение до Сортировки stl контейнера:
  161 -38 211 -372 916 -17 461 700 719 … 127 509 279 351 206 -127 -284 -820 626
- Заполнение после Сортировки кастомного контейнера:
  -998 -997 -994 -990 -987 -985 -980 -977 -976 … 972 975 985 987 988 995 995 998 999
- Заполнение после Сортировки stl контейнера:
  -998 -997 -994 -990 -987 -985 -980 -977 -976 -976 … 972 975 985 987 988 995 995 998 999
- Результат:
  Очереди отсортированы одинаково.

5) Инверсировать содержимое контейнера, заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

- До Риверсии:
  -1000 -999 -998 -995 -995 -994 -994 -990 -988 … 991 992 992 994 996 997 997 1000 1000

- После Риверсии:
  1000 1000 997 997 996 994 992 992 991 …-988 -990 -994 -994 -995 -995 -998 -999 -1000

Заключение.

Реализация самой очереди через узлы оказалась не очень сложной, но прописывание подходящего итератора совместимого с std оказалось задачей потруднее. В ходе тестов было заметно, что очереди удобны для описания описания жизненных ситуаций, как например структура с людьми из третьего теста.