Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Лабораторная работа №11

«Последовательные контейнеры библиотеки STL»

# Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Юхновец В.Г.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

**Задача 1.**

1. Создать последовательный контейнер.

2. Заполнить его элементами стандартного типа (тип указан в варианте).

3. Добавить элементы в соответствии с заданием

4. Удалить элементы в соответствии с заданием.

5. Выполнить задание варианта для полученного контейнера.

6. Выполнение всех заданий оформить в виде глобальных функций.

**Задача 2.**

1. Создать последовательный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в

варианте). Для пользовательского типа перегрузить необходимые операции.

3. Добавить элементы в соответствии с заданием

4. Удалить элементы в соответствии с заданием.

5. Выполнить задание варианта для полученного контейнера.

6. Выполнение всех заданий оформить в виде глобальных функций.

**Задача 3.**

1. Создать параметризированный класс, используя в качестве контейнера последовательный контейнер.

2. Заполнить его элементами.

3. Добавить элементы в соответствии с заданием

4. Удалить элементы в соответствии с заданием.

5. Выполнить задание варианта для полученного контейнера.

6. Выполнение всех заданий оформить в виде методов параметризированного класса.

**Задача 4.**

1. Создать адаптер контейнера.

2. Заполнить его элементами пользовательского типа (тип указан в

варианте). Для пользовательского типа перегрузить необходимые операции.

3. Добавить элементы в соответствии с заданием

4. Удалить элементы в соответствии с заданием.

5. Выполнить задание варианта для полученного контейнера.

6. Выполнение всех заданий оформить в виде глобальных функций.

**Задача 5.**

1. Создать параметризированный класс, используя в качестве контейнера адаптер контейнера.

2. Заполнить его элементами.

3. Добавить элементы в соответствии с заданием

4. Удалить элементы в соответствии с заданием.

5. Выполнить задание варианта для полученного контейнера.

6. Выполнение всех заданий оформить в виде методов параметризированного класса.

**Вариант 15**

**Задача 1.**

1. Контейнер – список  
2. Тип элементов - double

**Задача 2**

Тип элементов Pair (см. лабораторную работу №3).

**Задача 3**

Параметризированный класс – Список (см. лабораторную работу №7)

**Задача 4**

Адаптер контейнера – очередь с приоритетами.

**Задача 5**

Параметризированный класс – Список   
Адаптер контейнера – очередь с приоритетами.

**Задание 3**

Найти среднее арифметическое и добавить его в конец контейнера

**Задание 4**

Найти элементы ключами из заданного диапазона и удалить их из контейнера

**Задание 5**

К каждому элементу добавить сумму минимального и максимального элементов контейнера.

**Ответы на контрольные вопросы**

1. **Из каких частей состоит библиотека STL?**

Библиотека STL состоит из трех основных частей: контейнеров (containers), алгоритмов (algorithms) и итераторов (iterators).

1. **Какие типы контейнеров существуют в STL?**

В STL существует несколько типов контейнеров, включая vector, list, deque, set, map, unordered\_set, unordered\_map и многие другие.

1. **Что нужно сделать для использования контейнера STL в своей программе?**

Для использования контейнера STL в программе необходимо подключить соответствующий заголовочный файл, например, <vector> или <list>, и затем можно создавать экземпляры контейнеров и использовать их методы и операции.

1. **Что представляет собой итератор?**

Итератор в STL представляет собой объект, который указывает на элемент контейнера и позволяет осуществлять доступ к этому элементу и производить операции со смещением, такие как перемещение к следующему или предыдущему элементу.

1. **Какие операции можно выполнять над итераторами?**

С помощью итераторов можно выполнять различные операции, включая доступ к элементу (чтение или запись), смещение на следующий или предыдущий элемент, сравнение итераторов на равенство или неравенство, а также выполнение арифметических операций для перемещения итератора на определенное количество позиций.

1. **Каким образом можно организовать цикл для перебора контейнера с использованием итератора?**

Для перебора контейнера с использованием итератора можно организовать цикл, например, с помощью цикла for. Начальное значение итератора устанавливается на начало контейнера, условием продолжения цикла является проверка, что итератор не достиг конца контейнера, и на каждой итерации итератор смещается к следующему элементу контейнера.

1. **Какие типы итераторов существуют?**

В STL существуют различные типы итераторов, включая итераторы ввода (input iterators), итераторы вывода (output iterators), итераторы прямого доступа (forward iterators), итераторы двунаправленного доступа (bidirectional iterators) и итераторы случайного доступа (random access iterators). Каждый тип итератора обладает определенными возможностями и ограничениями.

1. **Перечислить операции и методы общие для всех контейнеров:**

begin() и end(): методы, возвращающие итераторы на начало и конец контейнера соответственно.

empty(): метод, проверяющий, является ли контейнер пустым.

size(): метод, возвращающий количество элементов в контейнере.

clear(): метод, удаляющий все элементы из контейнера.

insert(): метод, позволяющий вставить элемент в контейнер на определенную позицию.

erase(): метод, удаляющий элемент из контейнера по указанной позиции или диапазону.

find(): метод, выполняющий поиск элемента в контейнере и возвращающий итератор на его позицию.

swap(): метод, обменивающий содержимое двух контейнеров.

1. **Какие операции являются эффективными для контейнера vector? Почему?**

Контейнер vector обеспечивает эффективный доступ к элементам по индексу, так как элементы хранятся последовательно в памяти. Поэтому операции, такие как чтение и запись элемента по индексу, а также изменение размера контейнера, выполняются за константное время O(1). Также vector поддерживает быструю вставку и удаление элементов в конец контейнера с помощью методов push\_back() и pop\_back().

1. **Какие операции являются эффективными для контейнера list? Почему?**

Контейнер list является двусвязным списком, и его основное преимущество заключается в эффективности вставки и удаления элементов в любой позиции контейнера. Операции, такие как вставка и удаление элемента в начало, конец или внутреннюю часть списка, выполняются за постоянное время O(1). Однако доступ к элементам списка по индексу не является эффективным, и для этого лучше использовать контейнеры с поддержкой случайного доступа, например, vector.

1. **Какие операции являются эффективными для контейнера deque? Почему?**

Контейнер deque (double-ended queue) поддерживает эффективные операции вставки и удаления элементов как в начале, так и в конце контейнера. Это достигается благодаря его внутренней структуре данных, которая представляет собой блоки памяти, содержащие элементы. Таким образом, операции вставки и удаления элементов выполняются за постоянное время O(1), независимо от размера контейнера.

1. **Перечислить методы, которые поддерживает последовательный контейнер vector:**

Некоторые из методов, поддерживаемых последовательным контейнером vector, включают: push\_back(), pop\_back(), size(), empty(), clear(), begin(), end(), front(), back(), insert(), erase(), swap(), resize() и другие.

1. **Перечислить методы, которые поддерживает последовательный контейнер list:**

Некоторые из методов, поддерживаемых последовательным контейнером list, включают: push\_back(), push\_front(), pop\_back(), pop\_front(), size(), empty(), clear(), begin(), end(), front(), back(), insert(), erase() и другие.

1. **Перечислить методы, которые поддерживает последовательный контейнер deque:**

Некоторые из методов, поддерживаемых последовательным контейнером deque, включают: push\_back(), push\_front(), pop\_back(), pop\_front(), size(), empty(), clear(), begin(), end(), front(), back(), insert(), erase(), swap() и другие.

1. **Задан контейнер vector. Как удалить из него элементы со 2 по 5?**

Чтобы удалить элементы со 2 по 5 из контейнера vector, можно использовать метод erase() с передачей двух итераторов - на начало и конец диапазона элементов, которые нужно удалить. В данном случае, можно использовать следующую команду: myVector.erase(myVector.begin() + 1, myVector.begin() + 5);.

1. **Задан контейнер vector. Как удалить из него последний элемент?**

Чтобы удалить последний элемент из контейнера vector, можно использовать метод pop\_back(). Он удаляет последний элемент контейнера. Пример использования: myVector.pop\_back();

1. **Задан контейнер list. Как удалить из него элементы со 2 по 5?**

Для удаления элементов со 2 по 5 из контейнера list можно использовать метод erase() с передачей двух итераторов, указывающих на начало и конец диапазона элементов для удаления. Пример использования: myList.erase(std::next(myList.begin(), 1), std::next(myList.begin(), 5));

1. **Задан контейнер list. Как удалить из него последний элемент?**

Чтобы удалить последний элемент из контейнера list, можно использовать метод pop\_back(). Он удаляет последний элемент из списка. Пример использования: myList.pop\_back();

1. **Задан контейнер deque. Как удалить из него элементы со 2 по 5?**

Для удаления элементов со 2 по 5 из контейнера deque можно использовать метод erase() с передачей двух итераторов, указывающих на начало и конец диапазона элементов для удаления. Пример использования: myDeque.erase(myDeque.begin() + 1, myDeque.begin() + 5);

1. **Задан контейнер deque. Как удалить из него последний элемент?**

Чтобы удалить последний элемент из контейнера deque, можно использовать метод pop\_back(). Он удаляет последний элемент из дека. Пример использования: myDeque.pop\_back();

1. **Написать функцию для печати последовательного контейнера с использованием итератора.**

template <typename T>

void printContainer(const T& container) {

for (auto it = container.begin(); it != container.end(); ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

}

1. **Что представляют собой адаптеры контейнеров?**

Адаптеры контейнеров (container adapters) представляют собой специальные классы, которые предоставляют интерфейс для работы с базовыми контейнерами, изменяя или ограничивая их функциональность. Примеры адаптеров контейнеров в STL включают stack (стек), queue (очередь) и priority\_queue (очередь с приоритетом).

1. **Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?**

Объявление stack<int> s использует контейнер по умолчанию (обычно deque<int>) для реализации стека, в то время как объявление stack<int, list<int>> s явно указывает использование списка (list<int>) в качестве контейнера для реализации стека. Таким образом, основное отличие состоит в типе контейнера, который будет использоваться для хранения элементов стека.

1. **Перечислить методы, которые поддерживает контейнер stack:**

Некоторые из методов, поддерживаемых контейнером stack, включают: push(), pop(), top(), empty() и size(). Метод push() используется для добавления элемента на вершину стека, pop() - для удаления элемента с вершины стека, top() - для доступа к верхнему элементу стека, empty() - для проверки, является ли стек пустым, и size() - для получения размера стека.

1. **Перечислить методы, которые поддерживает контейнер queue:**

Некоторые из методов, поддерживаемых контейнером queue, включают: push(), pop(), front(), back(), empty() и size(). Метод push() используется для добавления элемента в конец очереди, pop() - для удаления элемента из начала очереди, front() - для доступа к элементу в начале очереди, back() - для доступа к элементу в конце очереди, empty() - для проверки, является ли очередь пустой, и size() - для получения размера очереди.

1. **Чем отличаются друг от друга контейнеры queue и priority\_queue?**

Контейнер queue представляет собой обычную очередь с принципом "первым пришел, первым вышел" (FIFO), где элементы добавляются в конец и удаляются из начала очереди. С другой стороны, контейнер priority\_queue представляет собой очередь с приоритетом, где элементы добавляются с учетом их приоритета и извлекаются в порядке убывания приоритета. Это позволяет использовать priority\_queue для реализации очереди с наибольшим приоритетом.

1. **Задан контейнер stack. Как удалить из него элемент с заданным номером?**

Контейнер stack не предоставляет прямого доступа к элементам по индексу или номеру. Он реализует структуру данных стек, где доступ возможен только к верхнему элементу. Чтобы удалить элемент с заданным номером, необходимо последовательно извлечь верхние элементы с помощью операции pop(), до достижения нужного элемента. Если нужно удалить элемент с определенным номером, возможно, стоит использовать другую структуру данных, такую как vector или list, которые позволяют доступ к элементам по индексу.

1. **Задан контейнер queue. Как удалить из него элемент с заданным номером?**

Контейнер queue не предоставляет прямого доступа к элементам по индексу или номеру. Он реализует структуру данных очередь, где доступ возможен только к элементу в начале очереди. Чтобы удалить элемент с заданным номером, необходимо последовательно извлечь элементы из начала очереди с помощью операции pop(), сохраняя только нужные элементы, пока не будет достигнут нужный номер. Если требуется удаление элементов по номеру, возможно, стоит использовать другую структуру данных, такую как vector или list.

1. **Написать функцию для печати контейнера stack с использованием итератора.**

template <typename T>

void printStack(const T& stack) {

T tempStack = stack; // Создаем временную копию стека

while (!tempStack.empty()) {

std::cout << tempStack.top() << " "; // Выводим верхний элемент

tempStack.pop(); // Удаляем верхний элемент

}

std::cout << std::endl;

}

1. **Написать функцию для печати контейнера queue с использованием итератора.**

template <typename T>

void printQueue(const T& queue) {

T tempQueue = queue; // Создаем временную копию очереди

while (!tempQueue.empty()) {

std::cout << tempQueue.front() << " "; // Выводим элемент в начале очереди

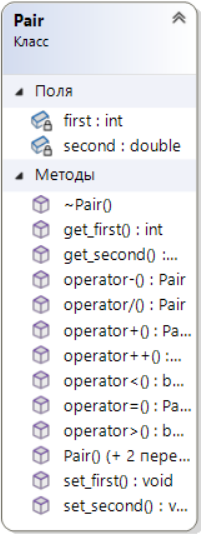
tempQueue.pop(); // Удаляем элемент из начала очереди

}

std::cout << std::endl;

}

**Диаграммы классов**



%

Рисунок 1 – диаграмма класса Pair

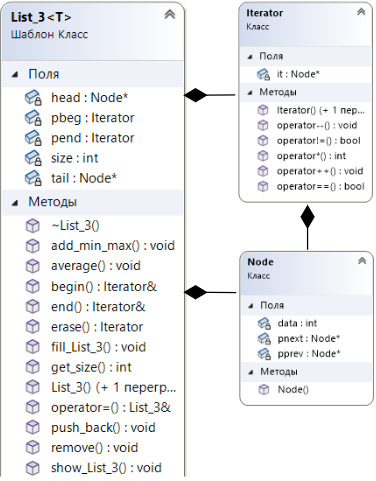


Рисунок 2 – диаграмма класса Лист для 3 задачи

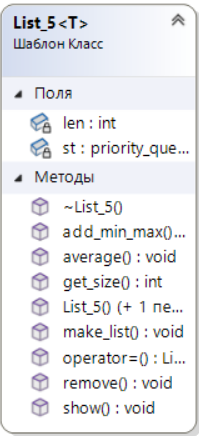


Рисунок 3 – диаграмма класса Лист для 5 задачи

**Описание задач**

**Задача 1**

#pragma once

typedef list<double> lst;

void fill\_list(lst& l)

{

int n = 0;

double el = 0;

cout << "Input size: "; cin >> n; cout << endl;

for (size\_t i = 0; i < n; i++)

l.push\_back((double)(rand() % 500 - 50) / 10);

}

void show\_list(lst& l)

{

if (l.size() != 0)

{

cout << "List: ";

for (auto& i : l)

cout << i << " ";

cout << endl;

}

}

void average(lst& l)

{

if (l.size() != 0)

{

double av = 0;

for (auto& i : l)

av += i;

l.push\_back(av / l.size());

}

}

void remove(lst& l)

{

if (l.size() != 0)

{

double start, end;

cout << "\nStart position: "; cin >> start;

cout << "End position: "; cin >> end;

for (auto i = l.begin(); i != l.end();)

{

if (start <= \*i && \*i <= end)

i = l.erase(i);

else

++i;

}

}

}

void add\_min\_max(lst& l)

{

if (l.size() != 0)

{

double min = \*l.begin(), max = \*l.begin();

for (auto& i : l)

{

if (i > max)

max = i;

if (i < min)

min = i;

}

for (auto& i : l)

i += (max + min);

}

}

void foo\_1()

{

cout << "\t\t\tEXERCISE 1\n\n";

srand(time(0));

list<double> lst1;

fill\_list(lst1);

cout << "---------------------- [initial list] ----------------------\n";

show\_list(lst1);

average(lst1);

cout << "\n--------------------- [list + average] ---------------------\n";

show\_list(lst1);

remove(lst1);

cout << "\n------------------- [list after deletion] ------------------\n";

show\_list(lst1);

add\_min\_max(lst1);

cout << "\n------------------- [list + (min + max)] -------------------\n";

show\_list(lst1);

cout << endl;

system("pause"); }

**Задача 2**

#pragma once

typedef list<Pair> lst2;

void fill\_list(lst2& l)

{

int n = 0;

Pair p;

cout << "Input size: "; cin >> n; cout << endl;

for (size\_t i = 0; i < n; i++)

{

p.set\_first((int)(rand() % 500 - 50) / 10);

p.set\_second((double)(rand() % 500 - 50) / 10);

l.push\_back(p);

}

}

void show\_list(lst2& l)

{

if (l.size() != 0)

{

cout << "List: ";

for (auto& i : l)

cout << i << " ";

cout << endl;

}

}

void average(lst2& l)

{

if (l.size() != 0)

{

int av\_first = 0;

double av\_second = 0;

for (auto& i : l)

{

av\_first += i.get\_first();

av\_second += i.get\_second();

}

av\_first /= l.size();

av\_second /= l.size();

Pair p(av\_first, av\_second);

l.push\_back(p);

}

}

void remove(lst2& l)

{

if (l.size() != 0)

{

Pair start, end;

cout << "\n\t[Start position]\n"; cin >> start;

cout << "\t[End position]\n"; cin >> end;

for (auto i = l.begin(); i != l.end();)

{

if ((start.get\_first() <= (\*i).get\_first() && start.get\_second() <= (\*i).get\_second()) && ((\*i).get\_first() <= end.get\_first() && (\*i).get\_second() <= end.get\_second()))

i = l.erase(i);

else

++i;

}

}

}

void add\_min\_max(lst2& l)

{

if (l.size() != 0)

{

Pair p\_min, p\_max;

auto i = l.begin();

p\_max.set\_first((\*i).get\_first());

p\_max.set\_second((\*i).get\_second());

p\_min.set\_first((\*i).get\_first());

p\_min.set\_second((\*i).get\_second());

for (auto& i : l)

{

if ((p\_max.get\_first() + p\_max.get\_second()) < (i.get\_first() + i.get\_second()))

{

p\_max.set\_first(i.get\_first());

p\_max.set\_second(i.get\_second());

}

else if ((p\_min.get\_first() + p\_min.get\_second()) > (i.get\_first() + i.get\_second()))

{

p\_min.set\_first(i.get\_first());

p\_min.set\_second(i.get\_second());

}

}

for (auto& i : l)

{

i = i + (p\_max.get\_first() + p\_min.get\_first());

i = i + (p\_max.get\_second() + p\_min.get\_second());

}

}

}

void foo\_2()

{

system("cls");

cout << "\t\t\tEXERCISE 2\n\n";

srand(time(0));

list<Pair> lst2;

fill\_list(lst2);

cout << "---------------------- [initial list] ----------------------\n";

show\_list(lst2);

average(lst2);

cout << "\n--------------------- [list + average] ---------------------\n";

show\_list(lst2);

remove(lst2);

cout << "\n------------------- [list after deletion] ------------------\n";

show\_list(lst2);

add\_min\_max(lst2);

cout << "\n------------------- [list + (min + max)] -------------------\n";

show\_list(lst2);

cout << endl;

system("pause");

}

**Задача 3**

#pragma once

#include <iostream>

#include <string>

using namespace std;

template <typename T>

class List\_3

{

private:

class Node

{

friend class List\_3;

public:

Node(T data);

T data;

Node\* pnext = nullptr;

Node\* pprev = nullptr;

};

int size = 0;

Node\* head = nullptr;

Node\* tail = nullptr;

public:

class Iterator

{

friend class List\_3;

private:

Node\* it = nullptr;

public:

Iterator();

Iterator(const Iterator& it);

bool operator==(const Iterator& it);

bool operator!=(const Iterator& it);

void operator++();

void operator--();

Iterator& operator=(const Iterator& it);

T& operator\*();

};

List\_3();

List\_3(List\_3& object);

~List\_3();

void push\_back(T data);

Iterator erase(Iterator& it);

int get\_size() { return size; }

void fill\_List\_3();

void show\_List\_3();

void average();

void remove();

void add\_min\_max();

List\_3& operator=(const List\_3& object);

Iterator& begin();

Iterator& end();

private:

Iterator pbeg;

Iterator pend;

};

template <typename T>

List\_3<T>::List\_3()

{

size = 0;

head = nullptr;

tail = nullptr;

}

template <typename T>

List\_3<T>::List\_3(List\_3& object)

{

for (int i = 0; i < object.size; i++)

(\*this).push\_back(object[i]);

}

template <typename T>

List\_3<T>::~List\_3()

{

if (head != nullptr)

{

Node\* current = head;

while (current != nullptr)

{

Node\* next = current->pnext;

delete current;

current = next;

}

head = nullptr;

tail = nullptr;

size = 0;

}

}

template <typename T>

List\_3<T>::Node::Node(T data)

{

this->data = data;

}

template <typename T>

void List\_3<T>::push\_back(T data)

{

if (head == nullptr)

{

head = new Node(data);

tail = head;

}

else

{

Node\* current = tail;

current->pnext = new Node(data);

tail = current->pnext;

tail->pprev = current;

tail->pnext = 0;

}

++size;

}

template<typename T>

typename List\_3<T>::Iterator List\_3<T>::erase(Iterator& it)

{

Node\* t = it.it;

if (t == head)

{

t = head->pnext;

delete head;

head = t;

it.it = head;

if (head != nullptr)

head->pprev = nullptr;

else

tail = nullptr;

}

else if (t == tail)

{

t = tail->pprev;

delete tail;

tail = t;

it.it = head;

if (tail != nullptr)

tail->pnext = nullptr;

else

head = nullptr;

}

else

{

Iterator temp = it;

t = t->pprev;

++temp;

delete it.it;

t->pnext = temp.it;

temp.it->pprev = t;

it = temp;

}

--size;

return it;

}

template<typename T>

void List\_3<T>::fill\_List\_3()

{

int n = 0;

Pair p;

cout << "Input size: "; cin >> n; cout << endl;

for (size\_t i = 0; i < n; i++)

{

p.set\_first((int)(rand() % 500 - 50) / 10);

p.set\_second((double)(rand() % 500 - 50) / 10);

(\*this).push\_back(p);

}

}

template<typename T>

void List\_3<T>::show\_List\_3()

{

if ((\*this).get\_size() != 0)

{

cout << "List\_3: ";

for (auto& i : (\*this))

cout << i << " ";

cout << endl;

}

}

template<typename T>

void List\_3<T>::average()

{

if ((\*this).get\_size() != 0)

{

int av\_first = 0;

double av\_second = 0;

for (auto i = (\*this).begin(); i != (\*this).end(); ++i)

{

av\_first += (\*i).get\_first();

av\_second += (\*i).get\_second();

}

av\_first /= (\*this).get\_size();

av\_second /= (\*this).get\_size();

Pair p(av\_first, av\_second);

(\*this).push\_back(p);

}

}

template<typename T>

void List\_3<T>::remove()

{

if ((\*this).get\_size() != 0)

{

Pair start, end;

cout << "\n\t[ Start position ]\n"; cin >> start;

cout << "\t[ End position ]\n"; cin >> end;

for (auto& i = (\*this).begin(); i != (\*this).end();)

{

if ((start.get\_first() <= (\*i).get\_first() && start.get\_second() <= (\*i).get\_second()) && ((\*i).get\_first() <= end.get\_first() && (\*i).get\_second() <= end.get\_second()))

i = (\*this).erase(i);

else

++i;

}

}

}

template<typename T>

void List\_3<T>::add\_min\_max()

{

if ((\*this).get\_size() != 0)

{

Pair p\_min, p\_max;

auto& i = (\*this).begin();

p\_max.set\_first((\*i).get\_first());

p\_max.set\_second((\*i).get\_second());

p\_min.set\_first((\*i).get\_first());

p\_min.set\_second((\*i).get\_second());

for (auto& i : (\*this))

{

if (p\_max < i)

{

p\_max.set\_first(i.get\_first());

p\_max.set\_second(i.get\_second());

}

else if (i < p\_min)

{

p\_min.set\_first(i.get\_first());

p\_min.set\_second(i.get\_second());

}

}

for (auto& i : (\*this))

{

i = i + (p\_max.get\_first() + p\_min.get\_first());

i = i + (p\_max.get\_second() + p\_min.get\_second());

}

}

}

template <typename T>

List\_3<T>& List\_3<T>::operator=(const List\_3& object)

{

if (this != &object)

if (this->head != nullptr)

{

this->~List\_3();

for (int i = 0; i < object.size; i++)

this->push\_back(object[i]);

}

else

for (int i = 0; i < object.size; i++)

this->push\_back(object[i]);

return \*this;

}

template <typename T>

List\_3<T>::Iterator::Iterator()

{

it = nullptr;

}

template <typename T>

List\_3<T>::Iterator::Iterator(const Iterator& it)

{

this->it = it.it;

}

template <typename T>

bool List\_3<T>::Iterator::operator==(const Iterator& it)

{

return this->it = it.it;

}

template <typename T>

bool List\_3<T>::Iterator::operator!=(const Iterator& it)

{

return this->it != it.it;

}

template <typename T>

void List\_3<T>::Iterator::operator++()

{

it = it->pnext;

}

template <typename T>

void List\_3<T>::Iterator::operator--()

{

it = it->pprev;

}

template<typename T>

typename List\_3<T>::Iterator& List\_3<T>::Iterator::operator=(const Iterator& it)

{

this->it = it.it;

return \*this;

}

template <typename T>

T& List\_3<T>::Iterator::operator\*()

{

return this->it->data;

}

template<typename T>

typename List\_3<T>::Iterator& List\_3<T>::begin()

{

if (head != nullptr)

pbeg.it = head;

return pbeg;

}

template<typename T>

typename List\_3<T>::Iterator& List\_3<T>::end()

{

if (tail != nullptr)

pend.it = tail->pnext;

return pend;

}

typedef List\_3<Pair> lst3;

void fill\_List\_3(lst3& l)

{

int n = 0;

Pair p;

cout << "Input size: "; cin >> n;

for (size\_t i = 0; i < n; i++)

{

p.set\_first((int)(rand() % 500 - 50) / 10);

p.set\_second((double)(rand() % 500 - 50) / 10);

l.push\_back(p);

}

cout << endl;

}

void show\_List\_3(lst3& l)

{

if (l.get\_size() != 0)

{

cout << "List\_3: ";

for (auto& i : l)

cout << i << " ";

cout << endl;

}

}

void average(lst3& l)

{

if (l.get\_size() != 0)

{

int av\_first = 0;

double av\_second = 0;

for (auto i = l.begin(); i != l.end(); ++i)

{

av\_first += (\*i).get\_first();

av\_second += (\*i).get\_second();

}

av\_first /= l.get\_size();

av\_second /= l.get\_size();

Pair p(av\_first, av\_second);

l.push\_back(p);

}

}

void remove(lst3& l)

{

if (l.get\_size() != 0)

{

Pair start, end;

cout << "\n\t[Start position]\n"; cin >> start;

cout << "\t[End position]\n"; cin >> end;

for (auto& i = l.begin(); i != l.end();)

{

if ((start.get\_first() <= (\*i).get\_first() && start.get\_second() <= (\*i).get\_second()) && ((\*i).get\_first() <= end.get\_first() && (\*i).get\_second() <= end.get\_second()))

i = l.erase(i);

else

++i;

}

}

}

void add\_min\_max(lst3& l)

{

if (l.get\_size() != 0)

{

Pair p\_min, p\_max;

auto& i = l.begin();

p\_max.set\_first((\*i).get\_first());

p\_max.set\_second((\*i).get\_second());

p\_min.set\_first((\*i).get\_first());

p\_min.set\_second((\*i).get\_second());

for (auto& i : l)

{

if ((p\_max.get\_first() + p\_max.get\_second()) < (i.get\_first() + i.get\_second()))

{

p\_max.set\_first(i.get\_first());

p\_max.set\_second(i.get\_second());

}

else if ((p\_min.get\_first() + p\_min.get\_second()) > (i.get\_first() + i.get\_second()))

{

p\_min.set\_first(i.get\_first());

p\_min.set\_second(i.get\_second());

}

}

for (auto& i : l)

{

i = i + (p\_max.get\_first() + p\_min.get\_first());

i = i + (p\_max.get\_second() + p\_min.get\_second());

}

}

}

void foo\_3()

{

system("cls");

cout << "\t\t\tEXERCISE 3\n\n";

srand(time(0));

List\_3<Pair> lst3;

lst3.fill\_List\_3();

cout << "---------------------- [initial List\_3] ----------------------\n";

lst3.show\_List\_3();

average(lst3);

cout << "\n--------------------- [List\_3 + average] ---------------------\n";

lst3.show\_List\_3();

lst3.remove();

cout << "\n------------------- [List\_3 after deletion] ------------------\n";

lst3.show\_List\_3();

lst3.add\_min\_max();

cout << "\n------------------- [List\_3 + (min + max)] -------------------\n";

lst3.show\_List\_3();

cout << endl;

system("pause");

}

**Задача 4**

#pragma once

typedef priority\_queue<Pair> p\_q;

void fill\_list(p\_q& l)

{

int n = 0;

Pair p;

cout << "Input size: "; cin >> n;

for (size\_t i = 0; i < n; i++)

{

p.set\_first((int)(rand() % 500 - 50) / 10);

p.set\_second((double)(rand() % 500 - 50) / 10);

l.emplace(p);

}

cout << endl;

}

void show\_list(p\_q l)

{

if (l.size() != 0)

{

cout << "List: ";

while (!l.empty())

{

cout << l.top() << " ";

l.pop();

}

cout << endl;

}

}

void average(p\_q& l)

{

p\_q temp;

Pair p;

if (l.size() != 0)

{

int av\_first = 0;

double av\_second = 0;

while (!l.empty())

{

p = l.top();

temp.push(p);

av\_first += p.get\_first();

av\_second += p.get\_second();

l.pop();

}

av\_first /= temp.size();

av\_second /= temp.size();

p.set\_first(av\_first);

p.set\_second(av\_second);

l = temp;

l.push(p);

}

}

void remove(p\_q& l)

{

if (l.size() != 0)

{

Pair start, end;

cout << "\n\t[Start position]\n"; cin >> start;

cout << "\t[End position]\n"; cin >> end;

p\_q temp;

Pair p;

while (!l.empty())

{

p = l.top();

if (!((start.get\_first() <= p.get\_first() && start.get\_second() <= p.get\_second()) && (p.get\_first() <= end.get\_first() && p.get\_second() <= end.get\_second())))

temp.push(p);

l.pop();

}

l = temp;

}

}

void add\_min\_max(p\_q& l)

{

if (l.size() != 0)

{

Pair p\_min, p\_max;

Pair p = l.top();

p\_max.set\_first(p.get\_first());

p\_max.set\_second(p.get\_second());

p\_min.set\_first(p.get\_first());

p\_min.set\_second(p.get\_second());

p\_q temp;

while (!l.empty())

{

p = l.top();

temp.push(p);

if ((p\_max.get\_first() + p\_max.get\_second()) < (p.get\_first() + p.get\_second()))

{

p\_max.set\_first(p.get\_first());

p\_max.set\_second(p.get\_second());

}

else if ((p\_min.get\_first() + p\_min.get\_second()) > (p.get\_first() + p.get\_second()))

{

p\_min.set\_first(p.get\_first());

p\_min.set\_second(p.get\_second());

}

l.pop();

}

while (!temp.empty())

{

p = temp.top();

p = p + p\_max.get\_first() + p\_min.get\_first();

p = p + p\_max.get\_second() + p\_min.get\_second();

l.push(p);

temp.pop();

}

}

}

void foo\_4()

{

system("cls");

cout << "\t\t\tEXERCISE 4\n\n";

srand(time(0));

priority\_queue<Pair> queue;

fill\_list(queue);

cout << "---------------------- [initial priority\_queue] ----------------------\n";

show\_list(queue);

average(queue);

cout << "\n--------------------- [priority\_queue + average] ---------------------\n";

show\_list(queue);

remove(queue);

cout << "\n------------------- [priority\_queue after deletion] ------------------\n";

show\_list(queue);

add\_min\_max(queue);

cout << "\n------------------- [priority\_queue + (min + max)] -------------------\n";

show\_list(queue);

cout << endl;

system("pause");

}

**Задача 5**

#pragma once

#include <iostream>

#include <queue>

#include <string>

using namespace std;

template <typename T>

class List\_5

{

private:

priority\_queue<T> st;

int len;

public:

List\_5();

List\_5(int n);

~List\_5();

void make\_list();

void show();

void average();

void remove();

void add\_min\_max();

int get\_size() { return size; }

List\_5& operator=(const List\_5& object);

};

template <typename T>

List\_5<T>::List\_5()

{

}

template<typename T>

List\_5<T>::List\_5(int n)

{

T a;

for (size\_t i = 0; i < n; i++)

{

cin >> a;

st.push();

}

}

template <typename T>

List\_5<T>::~List\_5()

{

}

template<typename T>

void List\_5<T>::make\_list()

{

int n;

cout << "Input size: "; cin >> n; cout << endl;

T a;

for (size\_t i = 0; i < n; i++)

{

a.set\_first((int)(rand() % 500 - 50) / 10);

a.set\_second((double)(rand() % 500 - 50) / 10);

st.push(a);

}

len = st.size();

}

template<typename T>

void List\_5<T>::show()

{

if (len != 0)

{

priority\_queue<T> temp;

for (size\_t i = 0; i < len; i++)

{

cout << st.top() << " ";

temp.push(st.top());

st.pop();

}

st = temp;

cout << endl;

}

}

template<typename T>

void List\_5<T>::average()

{

if (len != 0)

{

T p;

priority\_queue<T> temp;

for (size\_t i = 0; i < len; ++i)

{

p = p + st.top();

temp.push(st.top());

st.pop();

}

p = p / len;

temp.push(p);

len = temp.size();

st = temp;

}

}

template<typename T>

void List\_5<T>::remove()

{

if (len != 0)

{

T start, end;

cout << "\n\t[Start position]\n"; cin >> start;

cout << "\t[End position]\n"; cin >> end;

priority\_queue<T> temp;

T p;

while (!st.empty())

{

p = st.top();

if (start > p || p > end )

temp.push(p);

st.pop();

}

st = temp;

len = st.size();

}

}

template<typename T>

void List\_5<T>::add\_min\_max()

{

if (len != 0)

{

T p\_min, p\_max;

T p = st.top();

p\_max.set\_first(p.get\_first());

p\_max.set\_second(p.get\_second());

p\_min.set\_first(p.get\_first());

p\_min.set\_second(p.get\_second());

priority\_queue<T> temp;

while (!st.empty())

{

p = st.top();

temp.push(p);

if (p\_max < p)

{

p\_max.set\_first(p.get\_first());

p\_max.set\_second(p.get\_second());

}

else if (p < p\_min)

{

p\_min.set\_first(p.get\_first());

p\_min.set\_second(p.get\_second());

}

st.pop();

}

while (!temp.empty())

{

p = temp.top();

p = p + p\_max.get\_first() + p\_min.get\_first();

p = p + p\_max.get\_second() + p\_min.get\_second();

st.push(p);

temp.pop();

}

len = st.size();

}

}

template <typename T>

List\_5<T>& List\_5<T>::operator=(const List\_5& object)

{

if (this != &object)

if (this->head != nullptr)

{

this->~List\_5();

for (int i = 0; i < object.size; i++)

this->push\_back(object[i]);

}

else

for (int i = 0; i < object.size; i++)

this->push\_back(object[i]);

return \*this;

}

typedef List\_5<Pair> l\_p;

void foo\_5()

{

system("cls");

cout << "\t\t\tEXERCISE 5\n\n";

srand(time(0));

l\_p lst5;

lst5.make\_list();

cout << "---------------------- [initial List] ----------------------\n";

lst5.show();

lst5.average();

cout << "\n--------------------- [List\_5 + average] ---------------------\n";

lst5.show();

lst5.remove();

cout << "\n------------------- [List\_5 after deletion] ------------------\n";

lst5.show();

lst5.add\_min\_max();

cout << "\n------------------- [List\_5 + (min + max)] -------------------\n";

lst5.show();

cout << endl;

system("pause");

}

**Результаты работы программы**

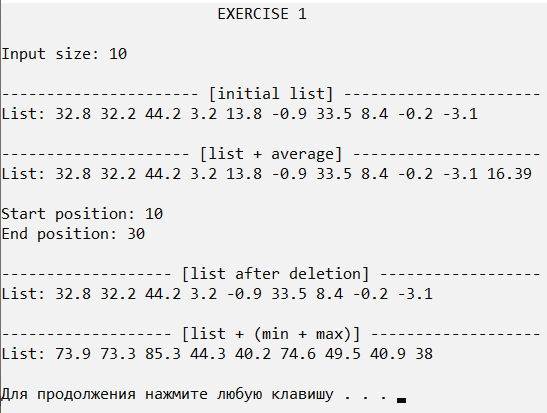


Рисунок 4 – задача 1

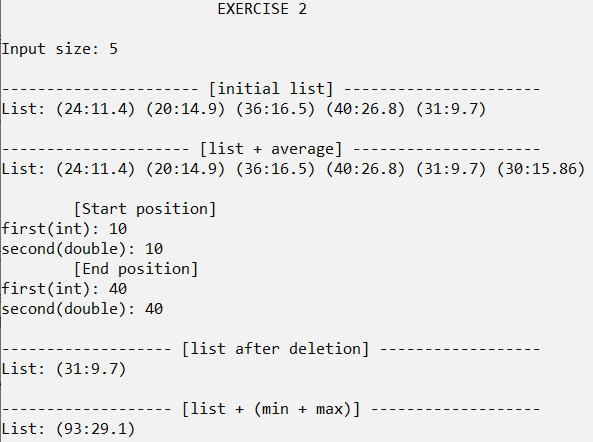


Рисунок 5 – задача 2

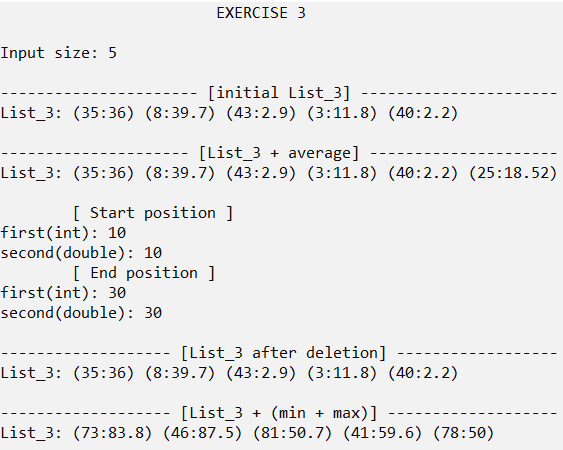


Рисунок 6 – задача 3

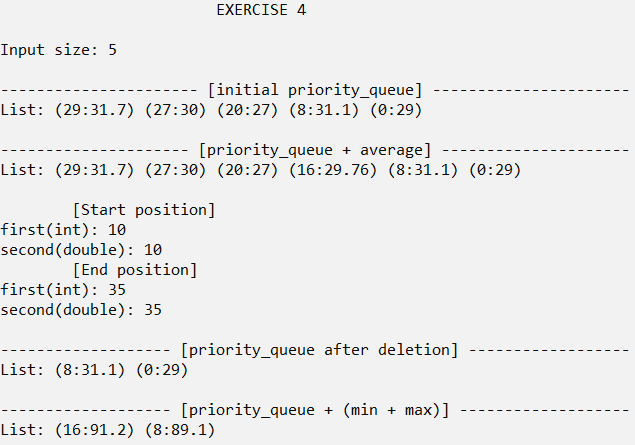


Рисунок 7 – задача 4

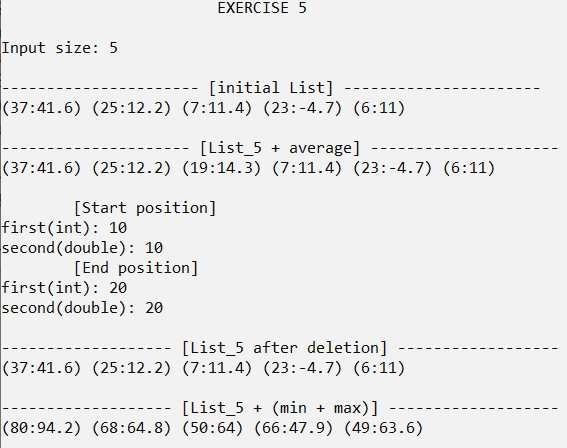


Рисунок 8 - задача 5