Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «ООП. Последовательные контейнеры библиотеки STL»

Выполнил студент ИВТ-22-2б:

Казанцев Антон Васильевич

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

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.

**Программное решение**

#include <iostream>

#include <list>

using namespace std;

void PushArithmeticMiddle(list<double>& m\_list) {

if (m\_list.size() > 0) {

double Sum = 0;

for (auto i = m\_list.begin(); i != m\_list.end(); i++) {

Sum += \*i;

}

m\_list.push\_back(Sum / m\_list.size());

}

else {

cout << "Список пуст!\n";

}

}

void EraseIfInRange(list<double>& m\_list, const double BeginValue, const double EndValue) {

auto i = m\_list.begin();

while (i != m\_list.end()) {

if (\*i >= BeginValue && \*i <= EndValue) {

i = m\_list.erase(i);

}

else {

i++;

}

}

}

void PlusMinAndMax(list<double>& m\_list) {

if (m\_list.size() > 0) {

double MaxValue, MinValue;

MaxValue = MinValue = \*m\_list.begin();

for (auto i = m\_list.begin(); i != m\_list.end(); i++) {

if (\*i > MaxValue) {

MaxValue = \*i;

}

else if (\*i < MinValue) {

MinValue = \*i;

}

}

for (auto i = m\_list.begin(); i != m\_list.end(); i++) {

\*i += MaxValue + MinValue;

}

}

}

void ShowList(const list<double>& m\_list) {

if (m\_list.size() > 0) {

for (auto i = m\_list.begin(); i != m\_list.end(); i++) {

cout << \*i << ' ';

}

}

else {

cout << "Список пуст!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

list<double> a = { 1, 2, 3, 4, 5, 6, 7 };

cout << "Список а:\n";

ShowList(a);

cout << "Добавляем среднее арифмитическое списка а в конец контейнера\n";

PushArithmeticMiddle(a);

cout << "Список а:\n";

ShowList(a);

cout << "Удаляем из списка а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2, 3);

cout << "Список а:\n";

ShowList(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

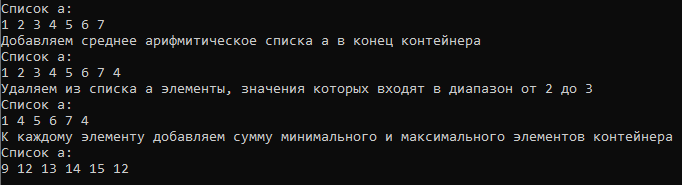
PlusMinAndMax(a);

cout << "Список а:\n";

ShowList(a);

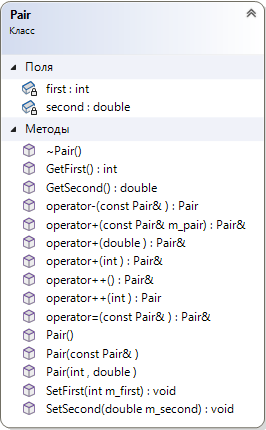
return 0;

}



**Задача 2:**

**UML-Диаграмма**



**Программное решение**

**Pair.h**

#pragma once

#include <iostream>

using namespace std;

class Pair {

int first;

double second;

public:

friend istream& operator>>(istream& in, Pair& p);

friend ostream& operator<<(ostream& out, const Pair& p);

Pair(int, double);

Pair();

Pair(const Pair&);

~Pair();

Pair& operator=(const Pair&);

Pair operator-(const Pair&);

Pair& operator+(int);

Pair& operator+(double);

Pair& operator++();

Pair operator++(int);

int GetFirst() const;

double GetSecond() const;

void SetFirst(int m\_first);

void SetSecond(double m\_second);

friend bool operator<(const Pair&, const Pair&);

Pair& operator + (const Pair& m\_pair);

};

**Pair.cpp**

#include "Pair.h"

Pair::Pair(int first, double second) {//конструктор с параметрами

this->first = first;

this->second = second;

}

istream& operator>>(istream& in, Pair& m\_pair) {//перегрузка оператора ввода

in >> m\_pair.first;

in >> m\_pair.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& m\_pair) {//перегрузка оператора вывода

return (out << m\_pair.first << " : " << m\_pair.second);

}

Pair::Pair() {//конструктор без параметров

first = 0;

second = 0;

}

Pair::~Pair() {//деструктор

}

Pair::Pair(const Pair& m\_pair) {//конструктор копирования

first = m\_pair.first;

second = m\_pair.second;

}

Pair& Pair::operator=(const Pair& m\_pair) {//перегрузка оператора присваивания

if (&m\_pair != this) {

first = m\_pair.first;

second = m\_pair.second;

}

return \*this;

}

Pair Pair::operator-(const Pair& m\_pair) {//перегрузка оператора вычитания

Pair result(first - m\_pair.first, second - m\_pair.second);

return result;

}

Pair& Pair::operator+(int first) {//перегрузка оператора сложения - целый аргумент

this->first += first;

return \*this;

}

Pair& Pair::operator+(double second) {//перегрузка оператора сложения - дробный аргумент

this->second += second;

return \*this;

}

Pair& Pair::operator++() {//перегрузка оператора ++ префикс

++first;

++second;

return \*this;

}

Pair Pair::operator ++(int) {//перегрузка оператора ++ постфикс

Pair temp = \*this;

this->first++;

this->second++;

return temp;//сначала нужно вывести без изменений

}

int Pair::GetFirst() const{

return first;

}

double Pair::GetSecond() const{

return second;

}

void Pair::SetFirst(int m\_first) {

first = m\_first;

}

void Pair::SetSecond(double m\_second) {

second = m\_second;

}

bool operator < (const Pair& pair1, const Pair& pair) {

return (pair1.GetFirst() + pair1.GetSecond() < pair.GetFirst() + pair.GetSecond());

}

Pair& Pair::operator + (const Pair& m\_pair) {

first += m\_pair.first;

second += m\_pair.second;

return \*this;

}

**task2.cpp**

#include <iostream>

#include "Pair.h"

using namespace std;

#include <list>

void PushArithmeticMiddle(list<Pair>& m\_list) {

if (m\_list.size() > 0) {

int SumFirst = 0;

double SumSecond = 0;

for (auto i = m\_list.begin(); i != m\_list.end(); i++) {

SumFirst += (\*i).GetFirst();

SumSecond += (\*i).GetSecond();

}

m\_list.push\_back(Pair(SumFirst / m\_list.size(), SumSecond / m\_list.size()));

}

else {

cout << "Список пуст!\n";

}

}

void EraseIfInRange(list<Pair>& m\_list, const double BeginValue, const double EndValue) {

auto i = m\_list.begin();

while (i != m\_list.end()) {

if (((\*i).GetFirst() >= BeginValue && (\*i).GetFirst() <= EndValue) || ((\*i).GetSecond() >= BeginValue && (\*i).GetSecond() <= EndValue)) {

i = m\_list.erase(i);

}

else {

i++;

}

}

}

void PlusMinAndMax(list<Pair>& m\_list) {

if (m\_list.size() > 0) {

Pair MaxValue, MinValue;

MaxValue = MinValue = \*m\_list.begin();

for (auto i = m\_list.begin(); i != m\_list.end(); i++) {

if (MaxValue < \*i) {

MaxValue = \*i;

}

else if (\*i < MinValue) {

MinValue = \*i;

}

}

for (auto i = m\_list.begin(); i != m\_list.end(); i++) {

\*i + MaxValue + MinValue;

}

}

}

void ShowList(const list<Pair>& m\_list) {

if (m\_list.size() > 0) {

for (auto i = m\_list.begin(); i != m\_list.end(); i++) {

cout << \*i << '\n';

}

}

else {

cout << "Список пуст!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

list<Pair> a = { Pair(6, 4), Pair(7, 9), Pair(2, 3.5) };

cout << "Список а:\n";

ShowList(a);

cout << "Добавляем среднее арифмитическое списка а в конец контейнера\n";

PushArithmeticMiddle(a);

cout << "Список а:\n";

ShowList(a);

cout << "Удаляем из списка а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2, 3);

cout << "Список а:\n";

ShowList(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

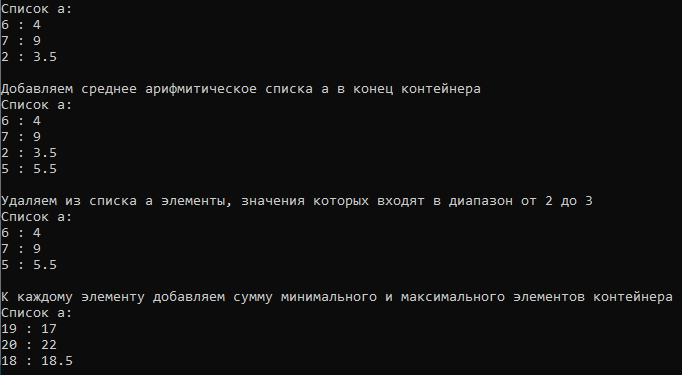
PlusMinAndMax(a);

cout << "Список а:\n";

ShowList(a);

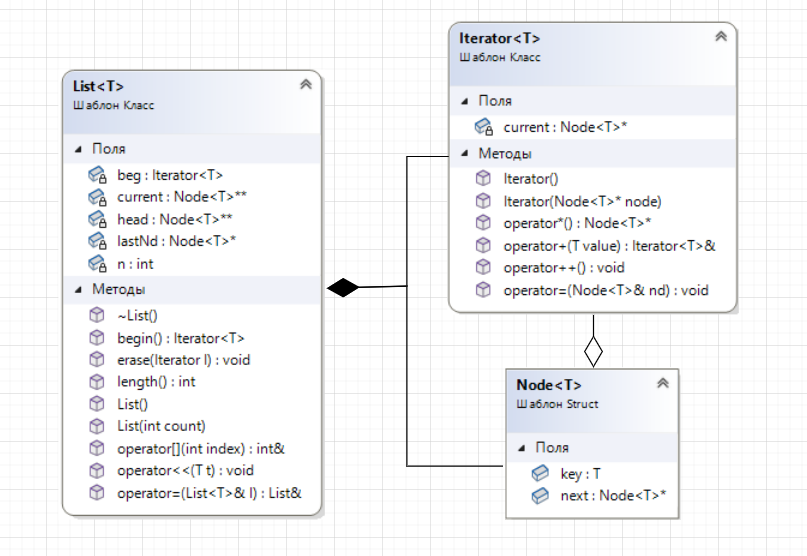
return 0;

}



**Задача 3:**

**UML-Диаграмма**



**Программное решение**

**List.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

template <class T>

struct Node {

T key;

Node<T>\* next;

};

template <class T>

class Iterator {

private:

Node<T>\* current;

template <class T>

friend class List;

public:

Iterator() {

current = NULL;

};

Iterator(Node<T>\* node) {

current = node;

};

void operator ++ () {

current = current->next;

}

Node<T>\* operator \*() const {

return current;

}

Iterator<T>& operator + (T value) {

current->key += value;

return \*this;

}

void operator = (Node<T>& nd) {

\*this = &nd;

}

};

template <class T>

class List {

public:

List() {};

List(int count) {

n = count;

head = new Node<T>;

head->key = 0;

lastNd = head;

for (int i = 1; i < n; i++) {

current = new Node<T>;

current->key = 0;

lastNd->next = current;

lastNd = current;

}

lastNd->next = NULL;

};

~List() {

lastNd = head;

while (lastNd != NULL) {

current = lastNd->next;

delete lastNd;

lastNd = current;

}

n = 0;

};

List& operator = (List<T>& l) {

if (this != &l) {

if (this != 0) {

lastNd = head;

while (lastNd != NULL) {

current = lastNd->next;

delete lastNd;

lastNd = current;

}

n = 0;

}

lastNd = head = new Node<T>;

l.lastNd = l.head->next;/////

lastNd->key = l.head->key;

while (l.lastNd != NULL) {

lastNd->next = new Node<T>;

lastNd->next->key = l.lastNd->key;

l.lastNd = l.lastNd->next;

lastNd = lastNd->next;

}

lastNd->next = NULL;

n = l.n;

}

return \*this;

};

int& operator[] (int index) {

if (index < n) {

lastNd = head;

for (int i = 0; i < index; i++) {

lastNd = lastNd->next;

}

return lastNd->key;

}

else {

cout << "Ошибка: Индекс за пределами диапазона!";

}

};

void operator << (T t) {

if (head == NULL) {

head = new Node<T>;

head->key = t;

head->next = NULL;

}

else {

lastNd = head;

while (lastNd->next != NULL) {

lastNd = lastNd->next;

}

lastNd->next = new Node<T>;

lastNd->next->key = t;

lastNd = lastNd->next;

lastNd->next = NULL;

}

n += 1;

};

Iterator<T> begin() {

beg.current = head;

return beg;

};

int length() {

return n;

}

Node<T>& erase(Iterator<T> I) {

lastNd = head;

if (head == \*I) {

lastNd = head->next;

delete head;

head = lastNd;

}

else {

while (lastNd->next != NULL && lastNd->next != \*I) {

lastNd = lastNd->next;

cout << lastNd->key << ' ';

}

if (lastNd->next != NULL) {

current = lastNd->next->next;

delete lastNd->next;

lastNd->next = current;

lastNd = lastNd->next;

}

}

return \*lastNd;

}

private:

Node<T>\* lastNd, \* current, \* head;

int n = 0;

Iterator<T> beg;

};

**task3.cpp**

#include <iostream>

#include "List.h"

using namespace std;

template <class T>

void PushArithmeticMiddle(List<T>& m\_List) {

if (m\_List.length() > 0) {

T Sum = 0;

for (Iterator<T> i = m\_List.begin(); \*i != NULL; ++i) {

Sum += (\*i)->key;

}

m\_List << (Sum / m\_List.length());

}

else {

cout << "Список пуст!\n";

}

}

template <class T>

void EraseIfInRange(List<T>& m\_List, const T BeginValue, const T EndValue) {

Iterator<T> i = m\_List.begin();

while (\*i != NULL) {

if ((\*i)->key >= BeginValue && (\*i)->key <= EndValue) {

i = m\_List.erase(i);

}

else {

++i;

}

}

}

template <class T>

void PlusMinAndMax(List<T>& m\_List) {

if (m\_List.length() > 0) {

T MaxValue, MinValue;

MaxValue = MinValue = (\*(m\_List.begin()))->key;

for (Iterator<T> i = m\_List.begin(); \*i != NULL; ++i) {

if ((\*i)->key > MaxValue) {

MaxValue = (\*i)->key;

}

else if ((\*i)->key < MinValue) {

MinValue = (\*i)->key;

}

}

for (Iterator<T> i = m\_List.begin(); \*i != NULL; ++i) {

i + MaxValue + MinValue;

}

}

}

template <class T>

void ShowList(List<T>& m\_List) {

if (m\_List.length() > 0) {

for (Iterator<T> i = m\_List.begin(); \*i != NULL; ++i) {

cout << (\*i)->key << ' ';

}

}

else {

cout << "Список пуст!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

List<double> a;

for (double i : { 1, 2, 3, 4, 5, 6, 7 }) {

a << i;

}

cout << "Список а:\n";

ShowList(a);

cout << "Добавляем среднее арифмитическое списка а в конец контейнера\n";

PushArithmeticMiddle(a);

cout << "Список а:\n";

ShowList(a);

cout << "Удаляем из списка а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2.0, 3.0);

cout << "Список а:\n";

ShowList(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

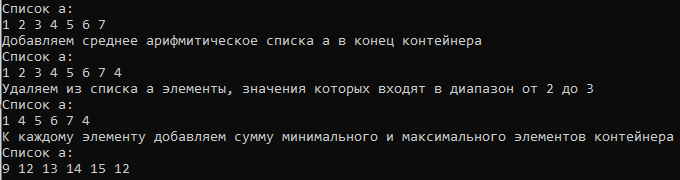
PlusMinAndMax(a);

cout << "Список а:\n";

ShowList(a);

return 0;

}



**Задача 4:**

**Программное решение**

#include <iostream>

#include <queue>

using namespace std;

void PushArithmeticMiddle(priority\_queue<double>& m\_priority\_queue) {

if (m\_priority\_queue.size() > 0) {

priority\_queue<double> tmp\_priority\_queue = m\_priority\_queue;

double Sum = 0;

while (tmp\_priority\_queue.size()) {

Sum += tmp\_priority\_queue.top();

tmp\_priority\_queue.pop();

}

m\_priority\_queue.push(Sum / m\_priority\_queue.size());

}

else {

cout << "Очередь пуста!\n";

}

}

void EraseIfInRange(priority\_queue<double>& m\_priority\_queue, const double BeginValue, const double EndValue) {

priority\_queue<double> tmp\_priority\_queue;

while (m\_priority\_queue.size()) {

if (m\_priority\_queue.top() < BeginValue || m\_priority\_queue.top() > EndValue) {

tmp\_priority\_queue.push(m\_priority\_queue.top());

}

m\_priority\_queue.pop();

}

m\_priority\_queue = tmp\_priority\_queue;

}

void PlusMinAndMax(priority\_queue<double>& m\_priority\_queue) {

if (m\_priority\_queue.size() > 0) {

priority\_queue<double> tmp\_priority\_queue = m\_priority\_queue;

double MaxValue, MinValue;

MaxValue = m\_priority\_queue.top();

while (m\_priority\_queue.size() > 1) {

m\_priority\_queue.pop();

}

MinValue = m\_priority\_queue.top();

m\_priority\_queue.pop();

while (tmp\_priority\_queue.size()) {

m\_priority\_queue.push(tmp\_priority\_queue.top() + MaxValue + MinValue);

tmp\_priority\_queue.pop();

}

}

}

void ShowQueue(priority\_queue<double>& m\_priority\_queue) {

if (m\_priority\_queue.size() > 0) {

priority\_queue<double> tmp\_priority\_queue = m\_priority\_queue;

while (tmp\_priority\_queue.size()) {

cout << tmp\_priority\_queue.top() << ' ';

tmp\_priority\_queue.pop();

}

}

else {

cout << "Очередь пуста!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

priority\_queue<double> a;

for (double i : { 1, 2, 3, 4, 5, 6, 7 }) {

a.push(i);

}

cout << "Очередь с приоритетеми а:\n";

ShowQueue(a);

cout << "Добавляем среднее арифмитическое очереди a\n";

PushArithmeticMiddle(a);

cout << "Очередь а:\n";

ShowQueue(a);

cout << "Удаляем из очереди а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2, 3);

cout << "Очередь а:\n";

ShowQueue(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

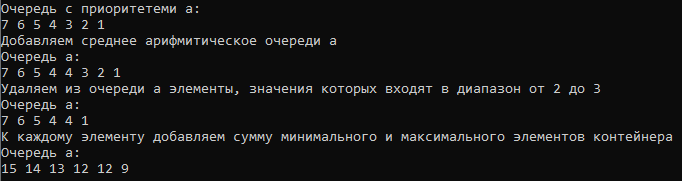
PlusMinAndMax(a);

cout << "Очередь а:\n";

ShowQueue(a);

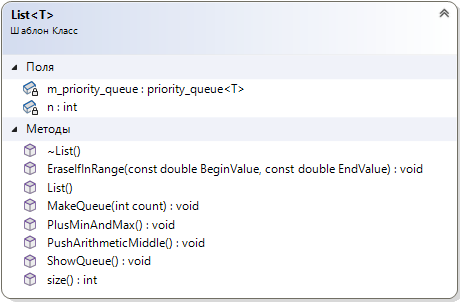
return 0;

}



**Задача 5:**

**UML-Диаграмма**



**Программное решение**

**List.h**

#pragma once

#include <iostream>

#include <queue>

using namespace std;

template <typename T>

class List {

private:

priority\_queue<T> m\_priority\_queue;

int n = 0;

public:

List() {};

~List() {};

void MakeQueue(int count) {

T a;

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

cout << "Введите " << i + 1 << " элемент очереди с приоритетами: ";

cin >> a;

m\_priority\_queue.push(a);

}

n = count;

};

void ShowQueue() {

if (m\_priority\_queue.size() > 0) {

priority\_queue<T> tmp\_priority\_queue = m\_priority\_queue;

while (tmp\_priority\_queue.size()) {

cout << tmp\_priority\_queue.top() << ' ';

tmp\_priority\_queue.pop();

}

}

else {

cout << "Очередь пуста!";

}

cout << '\n';

};

void PushArithmeticMiddle() {

if (m\_priority\_queue.size() > 0) {

priority\_queue<T> tmp\_priority\_queue = m\_priority\_queue;

double Sum = 0;

while (tmp\_priority\_queue.size()) {

Sum += tmp\_priority\_queue.top();

tmp\_priority\_queue.pop();

}

m\_priority\_queue.push(Sum / m\_priority\_queue.size());

}

else {

cout << "Очередь пуста!\n";

}

};

void EraseIfInRange(const double BeginValue, const double EndValue) {

if (m\_priority\_queue.size() > 0) {

priority\_queue<T> tmp\_priority\_queue;

while (m\_priority\_queue.size()) {

if (m\_priority\_queue.top() < BeginValue || m\_priority\_queue.top() > EndValue) {

tmp\_priority\_queue.push(m\_priority\_queue.top());

}

m\_priority\_queue.pop();

}

m\_priority\_queue = tmp\_priority\_queue;

}

};

int size() {

return n;

};

void PlusMinAndMax() {

if (m\_priority\_queue.size() > 0) {

priority\_queue<T> tmp\_priority\_queue = m\_priority\_queue;

double MaxValue, MinValue;

MaxValue = m\_priority\_queue.top();

while (m\_priority\_queue.size() > 1) {

m\_priority\_queue.pop();

}

MinValue = m\_priority\_queue.top();

m\_priority\_queue.pop();

while (tmp\_priority\_queue.size()) {

m\_priority\_queue.push(tmp\_priority\_queue.top() + MaxValue + MinValue);

tmp\_priority\_queue.pop();

}

}

}

};

**task5.cpp**

#include <iostream>

#include "List.h"

using namespace std;

int main() {

system("chcp 1251 > NULL");

List<double> a;

a.MakeQueue(7);

cout << "Очередь а:\n";

a.ShowQueue();

cout << "Добавляем среднее арифмитическое очереди a в контейнер\n";

a.PushArithmeticMiddle();

cout << "Очередь а:\n";

a.ShowQueue();

cout << "Удаляем из очереди а элементы, значения которых входят в диапазон от 2 до 3\n";

a.EraseIfInRange(2, 3);

cout << "Очередь а:\n";

a.ShowQueue();

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

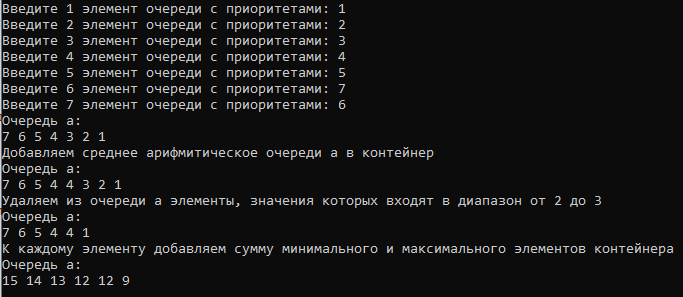
a.PlusMinAndMax();

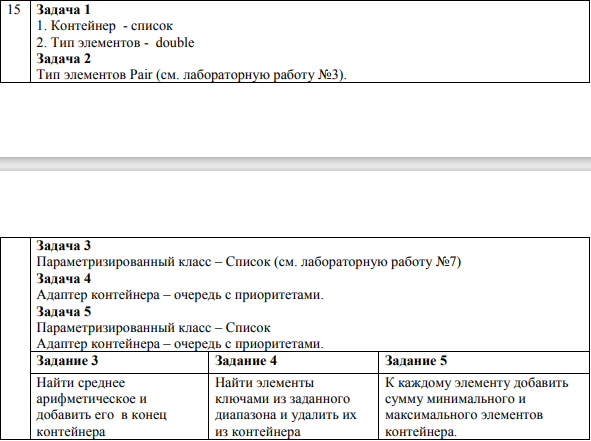
cout << "Очередь а:\n";

a.ShowQueue();

return 0;

}





**Контрольные вопросы**

1. *Из каких частей состоит библиотека STL?*

STL – Standard Template Library, стандартная библиотека шаблонов состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов. Контейнеры – это объекты, содержащие другие однотипные объекты. Обобщенные алгоритмы реализуют большое количество процедур, применимых к контейнерам: поиск, сортировку, слияние и т. п.

1. *Какие типы контейнеров существуют в STL?*

Последовательные контейнеры (векторы (vector), списки (list) и двусторонние очереди (deque)), ассоциативные контейнеры (словари (mар), словари с дубликатами (multimap), множества (set), множества с дубликатами (multiset) и битовые множества (bitset)), есть еще специализированные контейнеры (или адаптеры контейнеров), реализованные на основе базовых: стеки (stack), очереди (queue) и очереди с приоритетами (priority\_queue).

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

Для использования контейнера в программе необходимо включить в нее соответствующий заголовочный файл. Тип объектов, сохраняемых в контейнере, задается с помощью аргумента шаблона, например:

#include <vector>

#include <list>

1. *Что представляет собой итератор?*

Итераторы (iterators) – это объекты, которые по отношению к контейнеру играют роль указателей. Они позволяют получить доступ к содержимому контейнера примерно так же, как указатели используются для доступа к элементам массива.

1. *Какие операции можно выполнять над итераторами?*

С итераторами можно работать так же, как с указателями. К ним можно применить операции \*, инкремента, декремента. Присваивание одного итератора другому. Сравнение итераторов на равенство и неравенство (== и !=).

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

for (iterator it = v.begin(); it != v.end(), ++it) cout << v[it] << endl; //vector

1. *Какие типы итераторов существуют?*

* Итераторы ввода (input iterator) поддерживают операции равенства, разыменования и инкремента. ==, !=, \*i, ++i, i++, \*i++
* Итераторы вывода (output iterator) поддерживают операции разыменования, допустимые только с левой стороны присваивания, и инкремента. ++i, i++, \*i = t, \*i++ = t
* Однонаправленные итераторы (forward iterator) поддерживают все операции итераторов ввода/вывода и, кроме того, позволяют без ограничения применять присваивание. ==, !=, =, \*i, ++i, i++, \*i++
* Двунаправленные итераторы (bidirectional iterator) обладают всеми свойствами forward-итераторов, а также имеют дополнительную операцию декремента (--i, i--, \*i--), что позволяет им проходить контейнер в обоих направлениях.
* Итераторы произвольного доступа (random access iterator) обладают всеми свойствами bidirectional-итераторов, а также поддерживают операции сравнения и адресной арифметики, то есть непосредственный доступ по индексу. i += n, i + n, i -= n, i - n, i1 - i2, i[n], i1 < i2, i1 <= i2, i1 > i2, i1 >= i2

В STL также поддерживаются обратные итераторы (reverse iterators). Обратными итераторами могут быть либо двунаправленные итераторы, либо итераторы произвольного доступа, но проходящие последовательность в обратном направлении.

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

Операции равенства (==), неравенства (!=), операция присваивания (=), clear, insert, erase, size\_type size() const, size\_type max\_size() const, bool empty0 const, iterator begin(), iterator end(), reverse\_iterator begin(), reverse\_iterator end().

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

Контейнер вектор является аналогом обычного массива, за исключением того, что он автоматически выделяет и освобождает память по мере необходимости. Контейнер эффективно обрабатывает произвольную выборку элементов с помощью операции индексации [] или метода at. Происходит это потому, что массив - это последовательно занятая память, так что доступ к любому элементу происходит быстро.

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

Контейнер список организует хранение объектов в виде двусвязного списка. Каждый элемент списка содержит три поля: значение элемента, указатель на предшествующий и указатель на последующий элементы списка. Вставка и удаление работают эффективно для любой позиции элемента в списке, поскольку не требуется заново перевыделять память, достаточно переобозначить связи с предыдущим и последующим элементом.

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

Контейнер двусторонняя очередь во многом аналогичен вектору, элементы хранятся в непрерывной области памяти. Но в отличие от вектора двусторонняя очередь эффективно поддерживает вставку и удаление первого элемента (так же, как и последнего). Дек является сочетанием вектора и списка, поэтому при добавлении элементов в начало создается новый массив, конец которого по принципа списка указывает на изначальный массив. Аналогично с добавлением элементов в конец.

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

push\_back(), pop\_back(), insert, erase, [], at, swap, clear()

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

push\_back(), pop\_back(), push\_front(), pop\_back(), insert(), erase, swap, clear(), splice

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

push\_back(), push\_front(), pop\_back, pop\_front, insert, erase, [] ,at

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

for (int i = 2; i <= 5; ++i) v.erase(v.begin()+i);

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

v.erase(--v.end()); //итератор end() указывает на ячейку памяти после последнего элемента

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

list<int>::iterator beg = l.begin(); advance(beg, 2);

list<int>::iterator end = l.begin(); advance(end, 6); //удаляем включительно

l.erase(beg, end);

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

l.erase(--l.end());

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

auto beg = q.begin(); advance(beg, 2);

auto end = q.begin(); advance(end, 6);

q.erase(beg, end);

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

q.erase(--q.end());

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

void print(deque<int>q) {

for (deque<int>::iterator elem = q.begin(); elem != q.end(); ++elem)

cout << \*elem << " ";

}

1. *Что представляют собой адаптеры контейнеров?*

Специализированные последовательные контейнеры — стек, очередь и очередь с приоритетами — не являются самостоятельными контейнерными классами, а реализованы на основе рассмотренных выше классов, поэтому они называются адаптерами контейнеров.

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

По умолчанию для стека прототипом является класс deque. Объявление stack<int> s создает стек на базе двусторонней очереди (по умолчанию). Если по каким-то причинам нас это не устраивает, и мы хотим создать стек на базе списка, то объявление будет выглядеть следующим образом: stack<int, list<int> > s;

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

push () - добавление в конец; pop () - удаление из конца; top () - получение текущего элемента стека; empty() - проверка пустой стек или нет; size () – получение размера стека.

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

push () – добавление в конец очереди; pop () – удаление из начала очереди; front () – получение первого элемента очереди; back()- получение последнего элемента очереди; empty () – проверка пустая очередь или нет; size() – получение размера очереди.

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

Шаблонный класс priority\_queue (заголовочный файл <queue>) поддерживает такие же операции, как и класс queue, но реализация класса возможна либо на основе вектора (реализация по умолчанию), либо на основе списка. Очередь с приоритетами отличается от обычной очереди тем, что для извлечения выбирается максимальный элемент из хранимых в контейнере. Поэтому после каждого изменения состояния очереди максимальный элемент из оставшихся сдвигается в начало контейнера.

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

stack<int> copy;

int counter = 0;

while (counter != 2) {

copy.push(s.top());

s.pop();

++counter;

}

while (counter != 6) { //удаление невключительно

s.pop();

++counter;}

while (!copy.empty()) {

s.push(copy.top());

copy.pop();}

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

//перемещение всех элементов до удаляемого в конец

for (int i = 1; i < num; i++) {

push(qwe, qwe->head->key);

pop(qwe); }

pop(qwe); //удаление элемента

//возвращение очереди в прежнее состояние

//первый элемент снова ставится в начало

for (int i = 0; i < (qwe->size) - (num + 2 \* k); i++) {

push(qwe, qwe->head->key);

pop(qwe); }

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

void print(stack<int> s) {

int size = s.size();

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

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

s.pop(); }

}

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

void print(queue<int> s) {

int size = s.size();

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

cout << s.front() << " ";

s.pop(); }

}