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**

**Тема:** Лабораторная работа №11 по ООП

Семестр: 2

Выполнил студент ИВТ-22-2б:

Дияров Тимур Артурович

(дата, подпись)

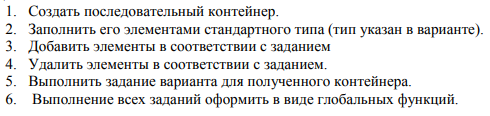
Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2023

**Задача 1:**



**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";

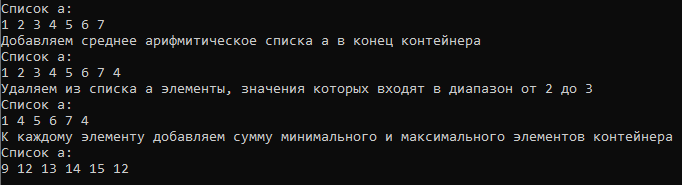
PlusMinAndMax(a);

cout << "Список а:\n";

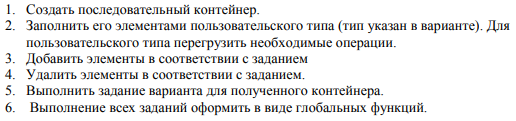
ShowList(a);

return 0;

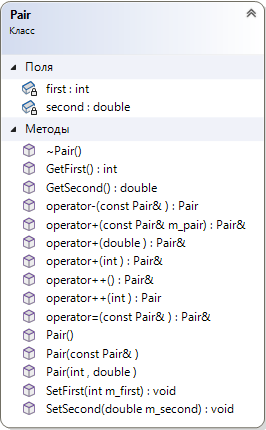
}



**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";

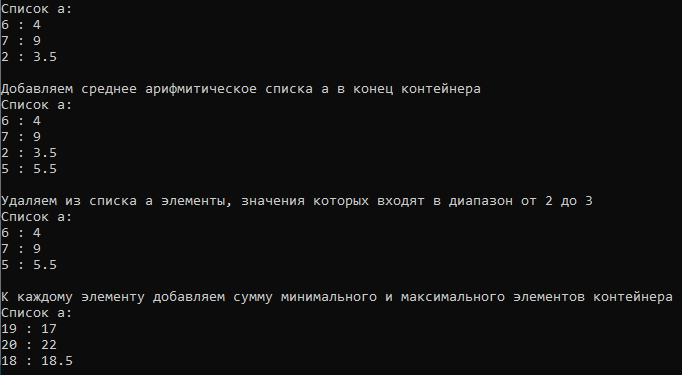
PlusMinAndMax(a);

cout << "Список а:\n";

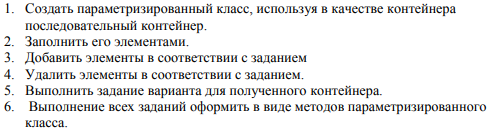
ShowList(a);

return 0;

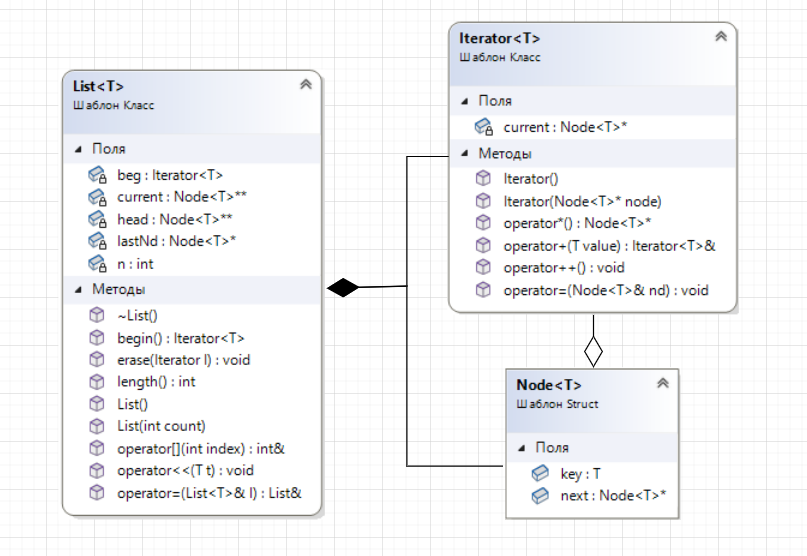
}



**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";

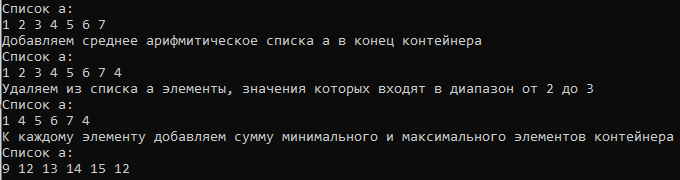
PlusMinAndMax(a);

cout << "Список а:\n";

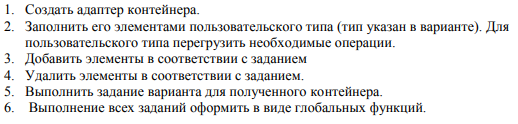
ShowList(a);

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";

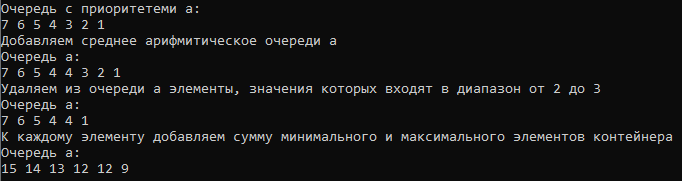
PlusMinAndMax(a);

cout << "Очередь а:\n";

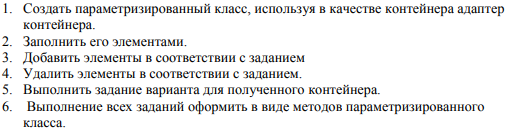
ShowQueue(a);

return 0;

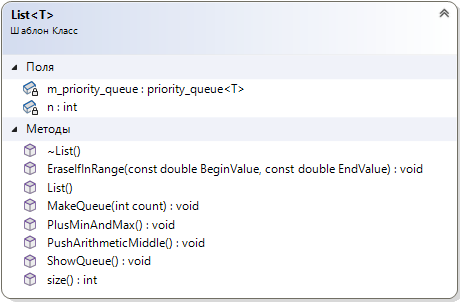
}



**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";

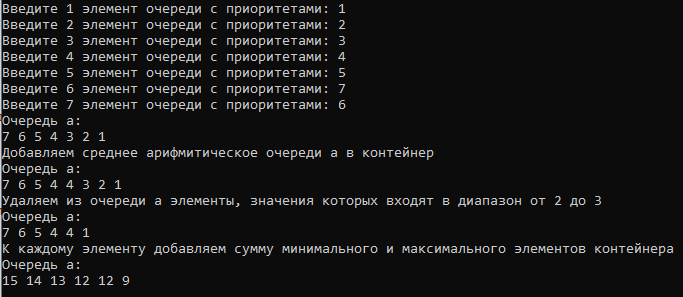
a.PlusMinAndMax();

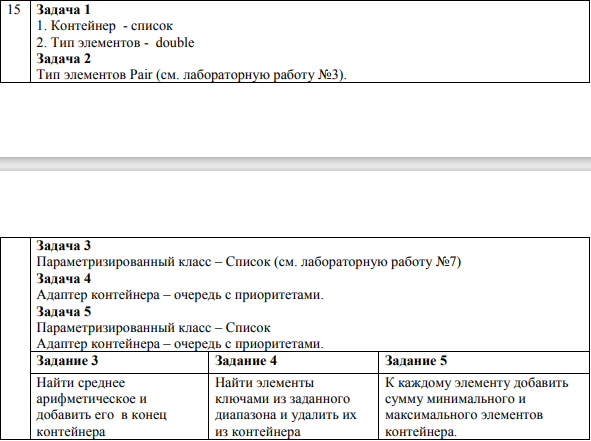
cout << "Очередь а:\n";

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.

1. Какие типы контейнеров существуют в STL?

Контейнеры STL можно разделить на два типа: последовательные и ассоциативные

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>

#include “person.h”

…..

vector<int> v;

list<person> l;

1. Что представляет собой итератор?

Функцию можно использовать для поиска элементов в массиве любого типа, но использовать ее для списка нельзя, поэтому авторы STL ввели понятие итератора. Итератор более общее понятие, чем указатель. Тип iterator определен для всех контейнерных классов STL

1. Какие операции можно выполнять над итераторами?

К основным операциям, выполняемым с любыми итераторами, относятся:

• Разыменование итератора: если р — итератор, то \*р — значение объекта, на который он ссылается.

• Присваивание одного итератора другому.

• Сравнение итераторов на равенство и неравенство (== и !=).

• Перемещение его по всем элементам контейнера с помощью префиксного (++р) или постфиксного (р++) инкремента.

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

Организация циклов просмотра элементов контейнеров тоже имеет некоторую специфику. Так, если i — некоторый итератор, то вместо привычной формы for (i =0; i < n; ++i) используется следующая:

for (i = first; i != last; ++i), где first - значение итератора, указывающее на первый элемент в контейнере, a last — значение итератора, указывающее на воображаемый элемент, который следует за последним элементом контейнера.

Операция сравнения < заменена на операцию ! =, т. к. операции < и > для итераторов в общем случае не поддерживаются.

1. Какие типы итераторов существуют?

В STL существуют следующие типы итераторов:

• входные,

• выходные,

• прямые,

• двунаправленные итераторы,

• итераторы произвольного доступа.

1. Перечислить операции и методы общие для всех контейнеров.

|  |  |
| --- | --- |
| **Операция или метод** | **Пояснение** |
| Операции равенства (==) и  неравенства (!=) | Возвращают значение true или false |
| Операция присваивания (=) | Копирует один контейнер в другой |
| clear | Удаляет все элементы |
| insert | Добавляет один элемент или диапазон элементов |
| erase | Удаляет один элемент или диапазон элементов |
| size\_type size() const | Возвращает число элементов |
| size\_type max\_size() const | Возвращает максимально допустимый размер  контейнера |
| bool empty0 const | Возвращает true, если контейнер пуст |
| iterator begin() | Возвращают итератор на начало контейнера (итерации будут производиться в прямом  направлении) |
| iterator end() | Возвращают итератор на конец контейнера  (итерации в прямом направлении будут закончены) |
| reverse\_iterator begin() | Возвращают реверсивный итератор на конец  контейнера (итерации будут производиться в обратном направлении) |
| reverse\_iterator end() | Возвращают реверсивный итератор на начало  контейнера (итерации в обратном направлении будут закончены |

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. Однако вставка элемента в любую позицию, кроме конца вектора, неэффективна. Для этого потребуется сдвинуть все последующие элементы путем копирования их значений. По этой же причине неэффективным является удаление любого элемента, кроме последнего.

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. Однако список не поддерживает произвольного доступа к своим элементам: например, для выборки n-го элемента нужно последовательно выбрать предыдущие п-1 элементов.

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).

1. Перечислить методы, которые поддерживает последовательный контейнер vector.

|  |  |
| --- | --- |
| Vector | |
| push\_back() | добавление  в конец |
| pop\_back() | удаление из конца |
| insert | Вставка в  произвольное место |
| erase | удаление из произвольного места |
| []  at | доступ к произвольному элементу |
| clear() | очистить  вектор |
| swap | Обмен векторов |

1. Перечислить методы, которые поддерживает последовательный контейнер list.

|  |  |
| --- | --- |
| List | |
| push\_back(T&key) | добавление  в конец |
| pop\_back() | удаление из  конца |
| push\_front(T&key) | добавление в начало |
| pop\_front() | удаление из начала |
| insert | Вставка в произвольное место |
| erase | удаление из произвольного места |
| clear() | Очистить вектор |
| swap | Обмен списков |
| splice | Сцепка списков |

1. Перечислить методы, которые поддерживает последовательный контейнер deque

|  |  |
| --- | --- |
| Deque | |
| push\_back(T&key) | добавление  в конец |
| pop\_back() | удаление из  конца |
| push\_front(T&key) | добавление в начало |
| pop\_front() | удаление из начала |
| insert | Вставка в произвольн ое место |
| erase | удаление из произвольн  ог оместа |
| []  at | доступ к произвольн ому  элементу |

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

vector<int> l;

vector<int>::iterator it\_start = l.begin();

vector<int>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

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

vector<double> l;

l.pop\_back();

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

list<string> l;

list<string>::iterator it\_start = l.begin();

list<string>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

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

list<double> l;

l.pop\_back();

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

deque<int> l;

deque<int>::iterator it\_start = l.begin();

deque<int>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

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

deque<double> l;

l.pop\_back();

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

void print\_list(LinkedList l)

{

cout << "\nСейчас список такой: \n";

copy(l.begin(), l.end(), ostream\_iterator<double>(cout, " "));

}

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?

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;

Смысл такой реализации заключается в том, что специализированный класс просто переопределяет интерфейс класса-прототипа, ограничивая его только теми методами, которые нужны новому классу. Cтек не позволяет выполнить произвольный доступ к своим элементам, а также не дает возможности пошагового перемещения, т. е. итераторы в стеке не поддерживаются

1. Перечислить методы, которые поддерживает контейнер stack.

Методы класса 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.

Методы класса 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?

template <class T>

void del(stack<T>& st, int index)

{

stack<T> temp;

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

{

temp.push(st.top());

st.pop();

}ss

st.pop();

while (temp.size())

{

st.push(temp.top());

temp.pop();

}

}

Но так делать не надо, потому что stack не предназначен для того, чтобы из него удалялись элементы не по порядку. В данном случае будет гораздо логичнее применить структуру данных список или любой контейнер STL.

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

std::deque<int> dq{1, 2, 3, 4, 5};

int index = 2;

auto it = std::next(dq.begin(), index);

dq.erase(it);

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

Это невозможно, так как у контейнера stack даже итераторов нет.

Вот пример как можно посмотреть все, что лежит в stack, но так делать тоже не надо, потому что это ломает весь смысл работы с данным контейнером, гораздо логичнее будет взять любой другой контейнер, например отлично подойдет однонаправленный список.

template <class T>

void printStack(stack<T> s)

{

while (!s.empty()) {

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

s.pop();

}}

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

template <class T>

void printDeque(deque<T> dq)

{

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

cout << \*i << endl;

}