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»

Тема: «Последовательные контейнеры библиотеки STL»

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Желнин Н.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

Постановка задачи

Вариант 14

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.

Указания варианта:

1. Контейнер - вектор; 2. Тип элементов - float

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.

Указание варианта:

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

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.

Указание варианта:

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

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.

Указание варианта:

Адаптер контейнера – очередь.

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 параметризованного класса.

Указания варианта:

1. Параметризированный класс – Список;

2. Адаптер контейнера – очередь.

Дополнение варианта ко всем задачам:

1. Найти минимальный элемент и добавить его на заданную позицию контейнера;

2. Найти меньше среднего арифметического и удалить их из контейнера;

3. Каждый элемент разделить на максимальный элемент контейнера.

Код программы

Задание 1

#include <iostream>

#include <vector>

#include <cstdlib>

using namespace std;

typedef vector<float>Vec;

Vec make\_vector(int n) {

Vec v;

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

float a = (rand() % 100 + 100);

a /= 100;

v.push\_back(a);

}

return v;

}

void print\_vector(Vec v) {

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

cout << v[i] << " ";

cout << endl;

}

float srednee(Vec v){

float s = 0;

for (int i=0;i<v.size();i++) s += v[i];

int n = v.size();

return s/n;

}

void add\_vector(Vec &v, float el, int pos) {

v.insert(v.begin()+pos-1, el);

}

void remove\_elements(Vec &v, float srednee) {

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

if (v[i]<srednee)

v.erase(v.begin()+i);

}

}

float max(Vec v) {

float max = v[0];

int n=0;

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

if (max<v[i]){

max = v[i];

n=i;//pos max

}

return max;

}

float min(Vec v) {

float min = v[0];

int n=0;

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

if (v[i]<min){

min = v[i];

n=i;//pos min

}

return min;

}

void delenie(Vec &v, float max) {

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

v[i] = v[i] / max;

}

int main() {

try {

vector<float> v;

int n, pos;

//float d\_min;

cout << "Size: "; cin >> n;

v = make\_vector(n);

print\_vector(v);

float el = srednee(v);

cout << "Pos: " ; cin >> pos;

add\_vector(v, el, pos);

print\_vector(v);

remove\_elements(v, el);

cout << "Delete numbers < sr\_znach:" << endl;

print\_vector(v);

cout << "Delenie na max:" << endl;

delenie(v, max(v));

print\_vector(v);

}

catch (float) {

cout << "error!";

}

//return 0;

}

Задание 2

//MAIN

#include "pair.h"

#include <iostream>

#include <vector>

#include <cstdlib>

using namespace std;

typedef vector<Pair>Vec;

Vec make\_vector(int n) {

Vec v;

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

Pair a;

cin >> a;

v.push\_back(a);

}

return v;

}

void print\_vector(Vec v) {

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

cout << v[i];

cout << endl;

}

Pair srednee(Vec v){

Pair s;

for (int i=0;i<v.size();i++) s += v[i];

int n = v.size();

s = s / n;

return s;

}

void add\_vector(Vec &v, Pair el, int pos) {

v.insert(v.begin()+pos-1, el);

}

void remove\_elements(Vec &v, Pair srednee) {

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

if (v[i]<srednee)

v.erase(v.begin()+i);

}

}

Pair max(Vec v) {

Pair max(0,0);

int n=0;

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

if (max<v[i]){

max = v[i];

n=i;//pos max

}

return max;

}

Pair min(Vec v) {

Pair min(1110,1110);

int n=0;

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

if (v[i]<min){

min = v[i];

n=i;//pos min

}

return min;

}

void delenie(Vec &v, Pair max) {

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

v[i] /= max;

}

int main() {

try {

vector<Pair> v;

int n, pos;

cout << "Size: "; cin >> n;

v = make\_vector(n);

print\_vector(v);

Pair el = srednee(v);

cout << "Pos: " ; cin >> pos;

add\_vector(v, el, pos);

print\_vector(v);

remove\_elements(v, el);

cout << "Delete numbers < sr\_znach:" << endl;

print\_vector(v);

cout << "Delenie na max:" << endl;

delenie(v, max(v));

print\_vector(v);

}

catch (float) {

cout << "error!";

}

}

//PAIR

#pragma once

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class Pair {

private:

int first;

double second;

public:

Pair() { first = 0; second = 0; };

Pair(int F, double S) { first = F; second = S; };

Pair(const Pair& t) { first = t.first;second = t.second; };

~Pair() {};

Pair operator =(const Pair&);

bool operator <(const Pair&) const;

bool operator >(const Pair&) const;

bool operator <=(const Pair&) const;

bool operator >=(const Pair&) const;

Pair& operator -(const Pair& p);

Pair& operator +(const Pair& p);

Pair& operator +=(const Pair& p);

Pair& operator +(const int X);

Pair& operator +(const double X);

Pair& operator /(const float X);

Pair& operator /=(const Pair& p);

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

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

friend fstream& operator>>(fstream& fin, Pair& p);

friend fstream& operator<<(fstream& fout, const Pair& p);

};

Pair& Pair::operator +(const int X) {

first += X;

return \*this;

}

Pair& Pair::operator +(const double X) {

second += X;

return \*this;

}

Pair& Pair::operator /=(const Pair& p) {

this->first /= p.first;

this->second /= p.second;

return \*this;

}

Pair& Pair::operator /(const float X) {

first = first / X;

second = second / X;

return \*this;

}

Pair& Pair::operator -(const Pair& p) {

this->first -= p.first;

this->second -= p.second;

return \*this;

}

Pair& Pair::operator +(const Pair& p) {

this->first += p.first;

this->second += p.second;

return \*this;

}

Pair& Pair::operator +=(const Pair& p) {

this->first += p.first;

this->second += p.second;

return \*this;

}

istream& operator >>(istream& in, Pair& t) {

cout << "int first = "; in >> t.first;

cout << "double second = "; in >> t.second;

return in;

}

ostream& operator <<(ostream& out, const Pair& t) {

return (out << t.first << " : " << t.second << endl);

}

Pair Pair::operator =(const Pair& p) {

if (&p == this) return \*this;

first = p.first;

second = p.second;

return\*this;

}

fstream& operator>>(fstream& fin, Pair& p) {

fin >> p.first;

fin >> p.second;

return fin;

}

fstream& operator<<(fstream& fout, const Pair& p) {

fout << p.first << "\n" << p.second << "\n";

return fout;

}

bool Pair::operator <(const Pair& p) const {

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis < sother) return true;

return false;

}

bool Pair::operator >(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis > sother) return true;

return false;

}

bool Pair::operator <=(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis <= sother) return true;

return false;

}

bool Pair::operator >=(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis >= sother) return true;

return false;

}

Задание 3

#include "pair.h"

#include "list.h"

#include <iostream>

using namespace std;

int main() {

List<Pair> l(3);

l.Print();

int n;

Pair range\_min, range\_max, min\_el, max\_el;

Pair el = l.srednee();

int pos;

cin >> pos;

l.add\_list(el, pos);

l.Print();

cout << "Remove elements:\n";

l.remove\_elements(l.srednee());

l.Print();

cout << "Increase elements:\n";

l.increase\_elements(l.max());

l.Print();

return 0;

}

#pragma once

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class Pair {

private:

int first;

double second;

public:

Pair() { first = 0; second = 0; };

Pair(int F, double S) { first = F; second = S; };

Pair(const Pair& t) { first = t.first;second = t.second; };

~Pair() {};

Pair operator =(const Pair&);

bool operator <(const Pair&) const;

bool operator >(const Pair&) const;

bool operator <=(const Pair&) const;

bool operator >=(const Pair&) const;

Pair& operator -(const Pair& p);

Pair& operator +(const Pair& p);

Pair& operator +=(const Pair& p);

Pair& operator +(const int X);

Pair& operator +(const double X);

Pair& operator /(const float X);

Pair& operator /=(const Pair& p);

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

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

friend fstream& operator>>(fstream& fin, Pair& p);

friend fstream& operator<<(fstream& fout, const Pair& p);

};

Pair& Pair::operator +(const int X) {

first += X;

return \*this;

}

Pair& Pair::operator +(const double X) {

second += X;

return \*this;

}

Pair& Pair::operator /=(const Pair& p) {

this->first /= p.first;

this->second /= p.second;

return \*this;

}

Pair& Pair::operator /(const float X) {

first = first / X;

second = second / X;

return \*this;

}

Pair& Pair::operator -(const Pair& p) {

this->first -= p.first;

this->second -= p.second;

return \*this;

}

Pair& Pair::operator +(const Pair& p) {

this->first += p.first;

this->second += p.second;

return \*this;

}

Pair& Pair::operator +=(const Pair& p) {

this->first += p.first;

this->second += p.second;

return \*this;

}

istream& operator >>(istream& in, Pair& t) {

cout << "int first = "; in >> t.first;

cout << "double second = "; in >> t.second;

return in;

}

ostream& operator <<(ostream& out, const Pair& t) {

return (out << t.first << " : " << t.second << endl);

}

Pair Pair::operator =(const Pair& p) {

if (&p == this) return \*this;

first = p.first;

second = p.second;

return\*this;

}

fstream& operator>>(fstream& fin, Pair& p) {

fin >> p.first;

fin >> p.second;

return fin;

}

fstream& operator<<(fstream& fout, const Pair& p) {

fout << p.first << "\n" << p.second << "\n";

return fout;

}

bool Pair::operator <(const Pair& p) const {

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis < sother) return true;

return false;

}

bool Pair::operator >(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis > sother) return true;

return false;

}

bool Pair::operator <=(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis <= sother) return true;

return false;

}

bool Pair::operator >=(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis >= sother) return true;

return false;

}

#pragma once

#include <list>

#include <iostream>

using namespace std;

template <class T>

class List {

list <T> l;

int size;

public:

List(void);

List(int n);

void Print();

~List(void);

T srednee();

void add\_list(T el, int pos);

void remove\_elements(T srednee);

T max();

T min();

void increase\_elements(T max);

};

template <class T>

List<T>::List() {

size = 0;

}

template <class T>

List<T>::~List() {}

template <class T>

List<T>::List(int n) {

T a;

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

cin >> a;

l.push\_back(a);

}

size = l.size();

}

template <class T>

void List<T>::Print() {

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

cout << \*it;

cout << endl;

}

template <class T>

T List<T>::srednee() {

T s;

for (auto it = l.begin();it != l.end();++it) s += \*it;

int n = l.size();

s = s / n;

return s;

}

template <class T>

void List<T>::add\_list(T el, int pos) {

l.insert(l.end(), el);

}

template <class T>

void List<T>::remove\_elements(T srednee) {

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

if (\*it < srednee)

it = l.erase(it);

else

++it;

}

}

template <class T>

T List<T>::max() {

T max = \*(l.begin());

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

if (\*it > max)

max = \*it;

return max;

}

template <class T>

T List<T>::min() {

T min = \*(l.begin());

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

if (\*it < min)

min = \*it;

return min;

}

template <class T>

void List<T>::increase\_elements(T max) {

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

\*it /= max;

}

Задание 4

#include "pair.h"

#include <iostream>

#include <queue>

#include <list>

using namespace std;

typedef queue<Pair> Q;

typedef list<Pair> List;

Q make\_q(int n) {

Q s;

Pair t;

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

cin >> t;

s.push(t);

}

return s;

}

List copy\_q\_to\_list(Q q) {

List l;

while (!q.empty()) {

l.push\_back(q.front());

q.pop();

}

return l;

}

Q copy\_list\_to\_q(List l) {

Q q;

for (auto it = l.cbegin();it != l.cend();++it)

q.push(\*it);

return q;

}

void print\_q(Q q) {

List l = copy\_q\_to\_list(q);

for (auto it = l.cbegin();it != l.cend();++it)

cout << \*it;

cout << endl;

}

Pair srednee(Q q) {

Pair s;

List l = copy\_q\_to\_list(q);

for (auto it = l.begin();it != l.end();++it) s += \*it;

int n = l.size();

s = s / n;

return s;

}

void add\_q(Q& q, Pair el) {

q.push(el);

}

void remove\_elements(Q& q, Pair srednee) {

List l = copy\_q\_to\_list(q);

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

if (\*it < srednee)

it = l.erase(it);

else

++it;

}

q = copy\_list\_to\_q(l);

}

Pair max(Q q) {

Pair max(0,0);

List l = copy\_q\_to\_list(q);

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

if (\*it > max)

max = \*it;

return max;

}

Pair min(Q q) {

Pair min(11111,11111);

List l = copy\_q\_to\_list(q);

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

if (\*it < min)

min = \*it;

return min;

}

void delenie\_elements(Q& q, Pair max) {

List l = copy\_q\_to\_list(q);

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

\*it /= max;

q = copy\_list\_to\_q(l);

}

int main() {

Q q;

int n;

cout << "Size: "; cin >> n;

q = make\_q(n);

print\_q(q);

Pair el = srednee(q);

add\_q(q, el);

print\_q(q);

cout << "Remove elements:\n";

remove\_elements(q, el);

print\_q(q);

cout << "Delenie elements:\n";

delenie\_elements(q, max(q));

print\_q(q);

return 0;

}

#pragma once

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class Pair {

private:

int first;

double second;

public:

Pair() { first = 0; second = 0; };

Pair(int F, double S) { first = F; second = S; };

Pair(const Pair& t) { first = t.first;second = t.second; };

~Pair() {};

Pair operator =(const Pair&);

bool operator <(const Pair&) const;

bool operator >(const Pair&) const;

bool operator <=(const Pair&) const;

bool operator >=(const Pair&) const;

Pair& operator -(const Pair& p);

Pair& operator +(const Pair& p);

Pair& operator +=(const Pair& p);

Pair& operator +(const int X);

Pair& operator +(const double X);

Pair& operator /(const float X);

Pair& operator /=(const Pair& p);

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

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

friend fstream& operator>>(fstream& fin, Pair& p);

friend fstream& operator<<(fstream& fout, const Pair& p);

};

Pair& Pair::operator +(const int X) {

first += X;

return \*this;

}

Pair& Pair::operator +(const double X) {

second += X;

return \*this;

}

Pair& Pair::operator /=(const Pair& p) {

this->first /= p.first;

this->second /= p.second;

return \*this;

}

Pair& Pair::operator /(const float X) {

first = first / X;

second = second / X;

return \*this;

}

Pair& Pair::operator -(const Pair& p) {

this->first -= p.first;

this->second -= p.second;

return \*this;

}

Pair& Pair::operator +(const Pair& p) {

this->first += p.first;

this->second += p.second;

return \*this;

}

Pair& Pair::operator +=(const Pair& p) {

this->first += p.first;

this->second += p.second;

return \*this;

}

istream& operator >>(istream& in, Pair& t) {

cout << "int first = "; in >> t.first;

cout << "double second = "; in >> t.second;

return in;

}

ostream& operator <<(ostream& out, const Pair& t) {

return (out << t.first << " : " << t.second << endl);

}

Pair Pair::operator =(const Pair& p) {

if (&p == this) return \*this;

first = p.first;

second = p.second;

return\*this;

}

fstream& operator>>(fstream& fin, Pair& p) {

fin >> p.first;

fin >> p.second;

return fin;

}

fstream& operator<<(fstream& fout, const Pair& p) {

fout << p.first << "\n" << p.second << "\n";

return fout;

}

bool Pair::operator <(const Pair& p) const {

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis < sother) return true;

return false;

}

bool Pair::operator >(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis > sother) return true;

return false;

}

bool Pair::operator <=(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis <= sother) return true;

return false;

}

bool Pair::operator >=(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis >= sother) return true;

return false;

}

Задание 5

#include "pair.h"

#include "list.h"

#include <iostream>

#include <queue>

#include <list>

using namespace std;

int main() {

List<Pair> l(3);

l.Print();

Pair el = l.srednee();

l.add\_list(el);

l.Print();

cout << "Remove elements:\n";

l.remove\_elements(el);

l.Print();

cout << "Delenie elements:\n";

l.delenie\_elements(l.max());

l.Print();

return 0;

}

#pragma once

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class Pair {

private:

int first;

double second;

public:

Pair() { first = 0; second = 0; };

Pair(int F, double S) { first = F; second = S; };

Pair(const Pair& t) { first = t.first;second = t.second; };

~Pair() {};

Pair operator =(const Pair&);

bool operator <(const Pair&) const;

bool operator >(const Pair&) const;

bool operator <=(const Pair&) const;

bool operator >=(const Pair&) const;

Pair& operator -(const Pair& p);

Pair& operator +(const Pair& p);

Pair& operator +=(const Pair& p);

Pair& operator +(const int X);

Pair& operator +(const double X);

Pair& operator /(const float X);

Pair& operator /=(const Pair& p);

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

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

friend fstream& operator>>(fstream& fin, Pair& p);

friend fstream& operator<<(fstream& fout, const Pair& p);

};

Pair& Pair::operator +(const int X) {

first += X;

return \*this;

}

Pair& Pair::operator +(const double X) {

second += X;

return \*this;

}

Pair& Pair::operator /=(const Pair& p) {

this->first /= p.first;

this->second /= p.second;

return \*this;

}

Pair& Pair::operator /(const float X) {

first = first / X;

second = second / X;

return \*this;

}

Pair& Pair::operator -(const Pair& p) {

this->first -= p.first;

this->second -= p.second;

return \*this;

}

Pair& Pair::operator +(const Pair& p) {

this->first += p.first;

this->second += p.second;

return \*this;

}

Pair& Pair::operator +=(const Pair& p) {

this->first += p.first;

this->second += p.second;

return \*this;

}

istream& operator >>(istream& in, Pair& t) {

cout << "int first = "; in >> t.first;

cout << "double second = "; in >> t.second;

return in;

}

ostream& operator <<(ostream& out, const Pair& t) {

return (out << t.first << " : " << t.second << endl);

}

Pair Pair::operator =(const Pair& p) {

if (&p == this) return \*this;

first = p.first;

second = p.second;

return\*this;

}

fstream& operator>>(fstream& fin, Pair& p) {

fin >> p.first;

fin >> p.second;

return fin;

}

fstream& operator<<(fstream& fout, const Pair& p) {

fout << p.first << "\n" << p.second << "\n";

return fout;

}

bool Pair::operator <(const Pair& p) const {

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis < sother) return true;

return false;

}

bool Pair::operator >(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis > sother) return true;

return false;

}

bool Pair::operator <=(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis <= sother) return true;

return false;

}

bool Pair::operator >=(const Pair& p) const{

double sthis = this->first + this->second;

double sother = p.first + p.second;

if (sthis >= sother) return true;

return false;

}

#pragma once

#include <list>

#include <queue>

#include <iostream>

using namespace std;

template <class T>

class List {

queue<T> q;

int size;

public:

List(void);

List(int n);

void Print();

~List(void);

T srednee();

void add\_list(T el);

void remove\_elements(T srednee);

T max();

T min();

void delenie\_elements(T max);

};

template <class T>

list<T> copy\_q\_to\_list(queue<T> q) {

list<T> l;

while (!q.empty()) {

l.push\_back(q.front());

q.pop();

}

return l;

}

template <class T>

queue<T> copy\_list\_to\_q(list<T> l) {

queue<T> q;

for (auto it = l.cbegin();it != l.cend();++it)

q.push(\*it);

return q;

}

template <class T>

List<T>::List() {

size = 0;

}

template <class T>

List<T>::~List() {}

template <class T>

List<T>::List(int n) {

T a;

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

cin >> a;

q.push(a);

}

size = q.size();

}

template <class T>

void List<T>::Print() {

list<T> l = copy\_q\_to\_list(q);

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

cout << \*it;

cout << endl;

}

template <class T>

T List<T>::srednee() {

T s;

list<T> l = copy\_q\_to\_list(q);

for (auto it = l.begin();it != l.end();++it) s += \*it;

int n = l.size();

s = s / n;

return s;

}

template <class T>

void List<T>::add\_list(T el) {

q.push(el);

}

template <class T>

void List<T>::remove\_elements(T srednee) {

list<T> l = copy\_q\_to\_list(q);

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

if (\*it < srednee)

it = l.erase(it);

else

++it;

}

q = copy\_list\_to\_q(l);

}

template <class T>

T List<T>::max() {

list<T> l = copy\_q\_to\_list(q);

T max = \*(l.begin());

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

if (\*it > max)

max = \*it;

return max;

}

template <class T>

T List<T>::min() {

list<T> l = copy\_q\_to\_list(q);

T min = \*(l.begin());

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

if (\*it < min)

min = \*it;

return min;

}

template <class T>

void List<T>::delenie\_elements(T max) {

list<T> l = copy\_q\_to\_list(q);

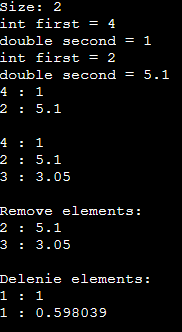
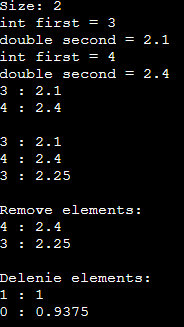
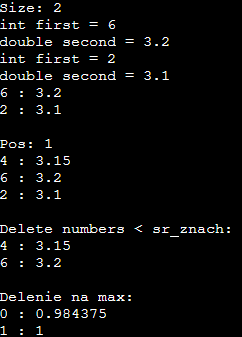
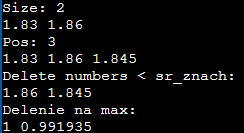
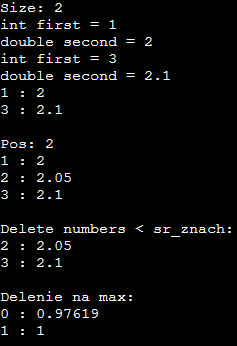
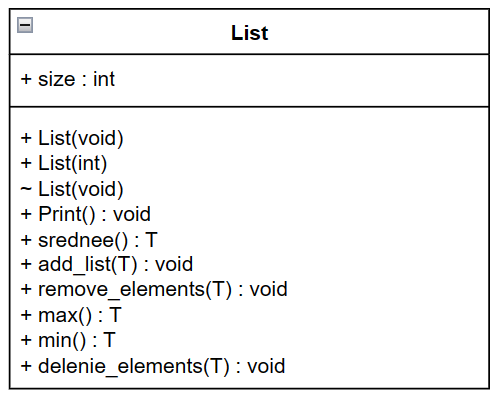
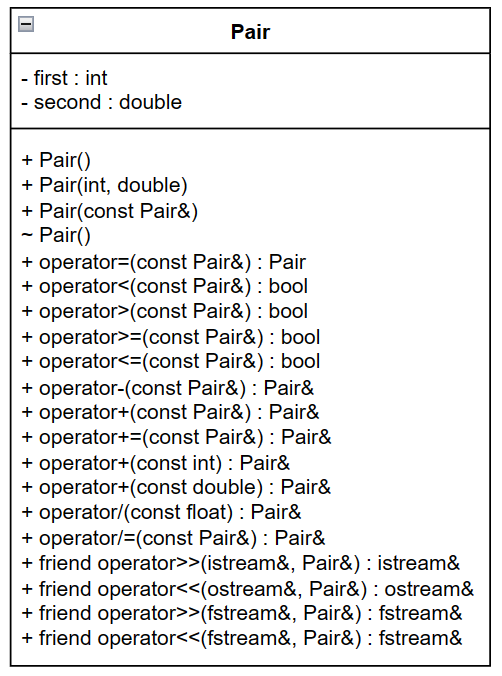
for (auto it = l.begin();it != l.end();++it)

\*it /= max;

q = copy\_list\_to\_q(l);

}

Диаграмма и вывод в консоль



Контрольные вопросы

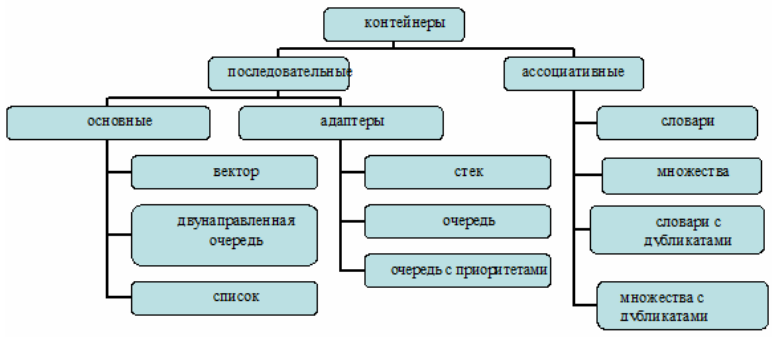
1. Из каких частей состоит библиотека STL?

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.

2. 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)



3. 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>

4. 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.

5. 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 !=).

6. 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

7. Какие типы итераторов существуют?

Существует пять типов итераторов:

* 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.

8. 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 empty() const, iterator begin(), iterator end(), reverse\_iterator begin(), reverse\_iterator end()

9. 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.

10. 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.

11. 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.

12. Перечислить методы, которые поддерживает последовательный контейнер vector.

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

13. Перечислить методы, которые поддерживает последовательный контейнер list.

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

14. Перечислить методы, которые поддерживает последовательный контейнер deque.

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

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

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

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

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

17. 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);

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

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

19. 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);

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

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

21. 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 << " ";

}

22. 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.

23. 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;

24. 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.

25. 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.

26. 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.

27. 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 не включительно

s.pop();

++counter;

}

while (!copy.empty()) {

s.push(copy.top());

copy.pop();

}

28. 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); }

29. 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(); }

}

30. 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(); }

}