Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.11**

Дисциплина: «Информатика»

Тема: “ Объектно-ориентированное программирование.

Последовательный контейнеры библиотеки STL.”

Вариант 15

Выполнил:

Студент группы ИВТ-20-2Б Чувашев Максим

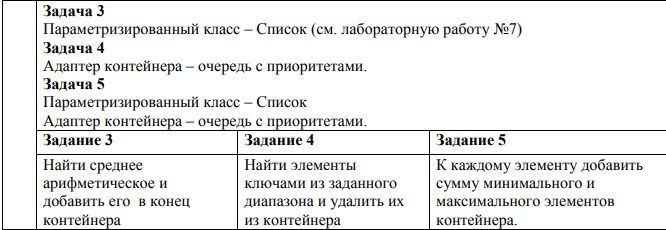
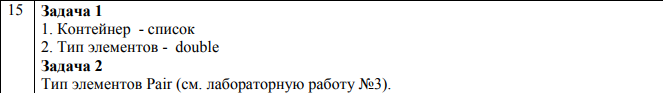
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Использовать класс Pair.
   2. Использовать контейнеры библиотеки STL.
2. В ходе работы были использованы следующие типы данных:
   1. Pair
   2. Double
   3. Int
3. Для решения задачи данные были представлены в следующем виде:
   1. list<double> l;
   2. list<Pair> l;
   3. LIST<double> l(size);
4. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Операторы >> и <<.
   2. Перегруженная функция Print

void Print(list<T> l)

{

cout << endl << "=========================================================================================\n\n";

while (l.empty()!= true)

{

cout << l.front() << " \t";

l.pop\_front();

}

cout << endl << "=========================================================================================\n\n\n";

}

void Print(priority\_queue<double> q)

{

cout << endl << "=========================================================================================\n\n";

while (q.empty() != true)

{

cout << q.top() << " \t";

q.pop();

}

cout << endl << "=========================================================================================\n\n\n";

}

1. Поставленные задачи будут решены следующими действиями:
   1. Функцией List, которая создает список типа дабл, далее совершаются все необходимые действия в соответствии с вариантом

void List()

{

cout << "СПИСОК ИЗ 1 ЗАДАНИЯ\n";

list<double> l;

double size = 0;

SIZE(&size);

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

{

double tmp = rand() % 100 / 2.123;

l.push\_back(tmp);

}

Print(l);

average(&l);

Print(l);

Del(&l);

Print(l);

l = increase(&l);

Print(l);

}

* 1. Функция LISTPAIR, которая создаем список типа Pair, и далее совершаются все необходимые действия согласно 2 заданию варианта

void LISTPAIR()

{

cout << "СПИСОК ИЗ 2 ЗАДАНИЯ\n";

list<Pair> l;

double size = 0;

SIZE(&size);

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

{

Pair tmp(rand() % 100, rand() % 100 / 2.123); // инициализация объекта класса

l.push\_back(tmp);

}

Print(l);

average(&l);

Print(l);

Del(&l);

Print(l);

l = increase(&l);

Print(l);

}

* 1. Функция InitList, которая создает праметризированный список согласно 3 заданию

void InitLIST()

{

cout << "Параметризированный список" << endl;

double size = 0;

SIZE(&size);

LIST<double> l(size);

cout << l;

l.AVG();

cout << l;

l.LR();

cout << l;

l.PLUS\_MIN\_MAX();

cout << l;

}

* 1. Функция удаления по ключу в диапозоне:

void Del(priority\_queue<double> \*q)

{

cout << "Введите диапазон\n";

int left = -1;

while (left < 0 || left > 15)

{

cout << "Левая граница: ";

cin >> left;

}

int right = -1;

while (right < left || right > q->size())

{

cout << "Правая граница граница: ";

cin >> right;

}

vector<double> tmp;

while (q->empty() != true)

{

tmp.push\_back(q->top());

q->pop();

}

tmp.erase(tmp.begin() + left, tmp.begin() + right);

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

{

q->push(tmp[i]);

}

tmp.clear();

}

template <typename T>

void Del(list<T> \*l)

{

cout << "Введите диапазон\n";

int left = -1;

while (left < 0 || left > 15)

{

cout << "Левая граница: ";

cin >> left;

}

int right = -1;

while (right < left || right > l->size())

{

cout << "Правая граница граница: ";

cin >> right;

}

vector<T> tmp;

while (l->empty() != true)

{

tmp.push\_back(l->front());

l->pop\_front();

}

tmp.erase(tmp.begin() + left, tmp.begin() + right);

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

{

l->push\_back(tmp[i]);

}

tmp.clear();

}

* 1. Функция сортировки

priority\_queue<double> increase(priority\_queue<double>\* q)

{

priority\_queue<double> qcopy = \*q;

double max;

max = -1;

double min;

min = 101;

while (qcopy.empty() != true)

{

if (max < qcopy.top())

{

max = qcopy.top();

}

if (min > qcopy.top())

{

min = qcopy.top();

}

qcopy.pop();

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

double res = min + max;

while (q->empty() != true)

{

qcopy.push(q->top() + res);

q->pop();

}

return qcopy;

}

**Код программы**

#include <iostream>

#include <list>

#include "LIST.h"

#include <vector>

#include "Pair.h"

#include <queue>

using namespace std;

template<typename T>

void Print(list<T> l)

{

cout << endl << "=========================================================================================\n\n";

while (l.empty()!= true)

{

cout << l.front() << " \t";

l.pop\_front();

}

cout << endl << "=========================================================================================\n\n\n";

}

void Print(priority\_queue<double> q)

{

cout << endl << "=========================================================================================\n\n";

while (q.empty() != true)

{

cout << q.top() << " \t";

q.pop();

}

cout << endl << "=========================================================================================\n\n\n";

}

int SIZE(double \*size)

{

while (\*size < 1)

{

cout << "Введите размер : ";

cin >> \*size;

}

return \*size;

}

template<typename T>

list<T> increase(list<T> \*l)

{

list<T> lcopy;

copy(l->begin(), l->end(), back\_inserter(lcopy)); // функция копирования списка

T max;

max = -1;

T min;

min = 101;

while (lcopy.empty() != true)

{

if (max < lcopy.front())

{

max = lcopy.front();

}

if (min > lcopy.front())

{

min = lcopy.front();

}

lcopy.pop\_front();

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

T res = min + max;

while (l->empty() != true)

{

lcopy.push\_back(l->front() + res);

l->pop\_front();

}

return lcopy;

}

priority\_queue<double> increase(priority\_queue<double>\* q)

{

priority\_queue<double> qcopy = \*q;

double max;

max = -1;

double min;

min = 101;

while (qcopy.empty() != true)

{

if (max < qcopy.top())

{

max = qcopy.top();

}

if (min > qcopy.top())

{

min = qcopy.top();

}

qcopy.pop();

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

double res = min + max;

while (q->empty() != true)

{

qcopy.push(q->top() + res);

q->pop();

}

return qcopy;

}

template <typename T>

void Del(list<T> \*l)

{

cout << "Введите диапазон\n";

int left = -1;

while (left < 0 || left > 15)

{

cout << "Левая граница: ";

cin >> left;

}

int right = -1;

while (right < left || right > l->size())

{

cout << "Правая граница граница: ";

cin >> right;

}

vector<T> tmp;

while (l->empty() != true)

{

tmp.push\_back(l->front());

l->pop\_front();

}

tmp.erase(tmp.begin() + left, tmp.begin() + right);

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

{

l->push\_back(tmp[i]);

}

tmp.clear();

}

void Del(priority\_queue<double> \*q)

{

cout << "Введите диапазон\n";

int left = -1;

while (left < 0 || left > 15)

{

cout << "Левая граница: ";

cin >> left;

}

int right = -1;

while (right < left || right > q->size())

{

cout << "Правая граница граница: ";

cin >> right;

}

vector<double> tmp;

while (q->empty() != true)

{

tmp.push\_back(q->top());

q->pop();

}

tmp.erase(tmp.begin() + left, tmp.begin() + right);

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

{

q->push(tmp[i]);

}

tmp.clear();

}

void average(list<double>\* l)

{

list<double> lcopy;

copy(l->begin(), l->end(), back\_inserter(lcopy)); // функция копирования списка

double AVG = 0;

while (lcopy.empty() != true) // пока в новом списке есть элементы

{

AVG += lcopy.front();

lcopy.pop\_front();

}

AVG = AVG / l->size();

l->push\_back(AVG);

}

void average(list<Pair>\* l)

{

list<Pair> lcopy;

copy(l->begin(), l->end(), back\_inserter(lcopy)); // функция копирования списка

Pair AVG;

while (lcopy.empty() != true) // пока в новом списке есть элементы

{

AVG = AVG + lcopy.front();

lcopy.pop\_front();

}

AVG.set\_first(AVG.get\_first() / l->size());

AVG.set\_second(AVG.get\_second() / l->size());

l->push\_back(AVG);

}

void average(priority\_queue<double>\* q)

{

priority\_queue<double> qcopy = \*q;

double AVG = 0;

while (qcopy.empty() != true)

{

AVG += qcopy.top();

qcopy.pop();

}

AVG = AVG / q->size();

q->push(AVG);

}

void List()

{

cout << "СПИСОК ИЗ 1 ЗАДАНИЯ\n";

list<double> l;

double size = 0;

SIZE(&size);

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

{

double tmp = rand() % 100 / 2.123;

l.push\_back(tmp);

}

Print(l);

average(&l);

Print(l);

Del(&l);

Print(l);

l = increase(&l);

Print(l);

}

void LISTPAIR()

{

cout << "СПИСОК ИЗ 2 ЗАДАНИЯ\n";

list<Pair> l;

double size = 0;

SIZE(&size);

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

{

Pair tmp(rand() % 100, rand() % 100 / 2.123); // инициализация объекта класса

l.push\_back(tmp);

}

Print(l);

average(&l);

Print(l);

Del(&l);

Print(l);

l = increase(&l);

Print(l);

}

void InitLIST()

{

cout << "Параметризированный список" << endl;

double size = 0;

SIZE(&size);

LIST<double> l(size);

cout << l;

l.AVG();

cout << l;

l.LR();

cout << l;

l.PLUS\_MIN\_MAX();

cout << l;

}

void QUEUE()

{

cout << "ОЧЕРЕДЬ С ПРИОРИТЕТАМИ ИЗ 4 ЗАДАНИЯ\n";

priority\_queue<double> q;

double size = 0;

SIZE(&size);

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

{

q.push(rand() % 100 / 2.123);

}

Print(q);

average(&q);

Print(q);

Del(&q);

Print(q);

q = increase(&q);

Print(q);

}

void QUEUELIST()

{

//priority\_queue<double, LIST<double>> l;

//l.push(10);

//cout << l.top();

}

int main()

{

setlocale(LC\_ALL, "Ru");

List();

LISTPAIR();

InitLIST();

QUEUE();

QUEUELIST();

}

#include "Pair.h"

#include <iostream>

using namespace std;

//перегрузка операции присваивания

Pair& Pair::operator=(const Pair& other)

{

if (&other == this) return \*this; //проверка на самоприсваивание

this->first = other.first;

this->second = other.second;

return \*this;

}

Pair Pair::operator=(int tmp)

{

this->first = tmp;

this->second = tmp;

return \*this;

}

Pair Pair::operator-(const Pair& other)

{

Pair tmp;

tmp.first = this->first - other.first;

tmp.second = this->second - other.second;

return tmp;

}

Pair Pair::operator+(double constant)

{

if (constant == int(constant))

{

this->first = this->first + constant;

}

else

{

this->second = this->second + constant;

}

return \*this;

}

Pair Pair::operator+(Pair other)

{

this->first += other.first;

this->second += other.second;

return \*this;

}

bool Pair::operator>(Pair other)

{

if (this->first > other.second && this->second > other.second)

{

return true;

}

else

{

return false;

}

}

bool Pair::operator <(Pair other)

{

if (this->first < other.second && this->second < other.second)

{

return true;

}

else

{

return false;

}

}

//перегрузка глобальной функции-операции ввода

istream& operator>>(istream& in, Pair& other)

{

cout << "first?";

in >> other.first;

cout << "second?";

in >> other.second;

return in;

}

//перегрузка глобальной функции-операции вывода

ostream& operator<<(ostream& out, const Pair& other)

{

return (out << other.first << " : " << other.second);

}

#pragma once

#include <string>

#include <iostream>

#include "Pair.h"

using namespace std;

template <class T>

class LIST

{

public:

LIST()

{

Size = 0;

data = nullptr;

}

LIST(int Size) // Конструктор с параметрами

{

this->Size = Size;

data = new T[Size];

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

{

data[i] = rand() % 100 / 2.123;

}

}

LIST(const LIST& other) // Конструктор с параметрами

{

Size = other.Size;

if (data != nullptr)

{

delete[] data;

}

data = new T[Size];

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

{

data[i] = other.data[i];

}

}

~LIST() // деструктор

{

if (this->data != nullptr)

{

delete[] this->data;

}

data = nullptr;

}

LIST& operator = (const LIST& a)// оператор присваивания

{

if (this == &a) // Првоверка на самоприсваивание

{

return \*this;

}

Size = a.Size;

if (data != 0) // Если ячейка не пуста, то очищаем ячейку

{

delete[] data;

}

data = new int[Size];

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

{

data[i] = a.Size;

}

return \*this;

}

T& operator [] (int index) // операция доступа к индексу

{

if (index < Size)

{

return data[index];

}

else

{

cout << "\nError! Index>Size";

}

}

LIST<double>& AVG()

{

double\* res = new double[Size];

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

{

res[i] = data[i];

}

delete[] data;

data = new double[Size + 1];

double avg = 0;

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

{

data[i] = res[i];

avg += res[i];

}

avg = avg / this->Size;

data[Size] = avg;

Size++;

return \*this;

}

LIST<double>& PLUS\_MIN\_MAX()

{

double min = 101;

double max = -1;

double\* tmp = new double[Size];

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

{

if (min > data[i])

{

min = data[i];

}

if (max < data[i])

{

max = data[i];

}

tmp[i] = data[i];

}

delete[] data;

data = new double[Size];

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

{

data[i] = tmp[i] + min + max;

}

delete[] tmp;

return \*this;

}

LIST<double>& LR()

{

cout << "Введите диапазон\n";

int left = -1;

while (left < 0 || left > 15)

{

cout << "Левая граница: ";

cin >> left;

}

int right = -1;

while (right < left || right > this->Size)

{

cout << "Правая граница граница: ";

cin >> right;

}

double\* tmp = new double[Size - (right - left + 1)];

int count = 0;

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

{

if (i < left || i > right)

{

tmp[count] = data[i];

count++;

}

}

Size = Size - (right - left + 1);

delete[] data;

data = new double[Size];

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

{

data[i] = tmp[i];

}

delete[] tmp;

return \*this;

}

LIST<T>& operator \* (LIST& other)

{

LIST\* res = new LIST;

if (this->Size > other.Size)

{

int Size1 = this->Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = other.Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = this->Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0; // Заполняем разничу 1 и 2 листа нулями, так как число умножить не на что

}

return \*res;

}

else

{

int Size1 = other.Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = this->Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = other.Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0;

}

return \*res;

}

}

int operator ()()

{

return this->Size;

}

void SetSize(int size) // Сеттер размера вектора

{

this->Size = size;

}

int GetData() // Геттер данных ячейчки

{

return \*this->data;

}

int GetSize() //Геттер размера вектора

{

this->Size = size;

}

int size()

{

return Size;

}

friend ostream& operator<< <>(ostream& out, const LIST<T>& a);

friend istream& operator>> <>(istream& in, LIST<T>& a);

//friend ostream& operator<< (ostream& out, const LIST<Pair>& other);

private:

int Size = 0;

T\* data;

};

template <typename T>

ostream& operator<< <>(ostream& out, const LIST<T>& other)

{

out << endl << "=========================================================================================\n\n";

for (int i = 0; i < other.Size; i++)

{

out << other.data[i] << " \t";

}

out << endl << "=========================================================================================\n\n\n";

return out;

}

template<typename T>

istream& operator>> <>(istream& in, LIST<T>& other)

{

for (int i = 0; i < other.Size; i++)

{

cout << "Введите " << i << " элемент списка: ";

in >> other.data[i];

}

return in;

}

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

// конструктор копирования без параметров

Pair()

{

first = 0; // При создании объекта поле заполняется 0

second = 0; // При создании объекта поле заполняется 0

};

// конструктор копирования с параметрами

Pair(int f, double s)

{

first = f; // При создании объекта поле заполняется переданным в качестве параметра числом

second = s; // При создании объекта поле заполняется переданынм в качестве параметра числом

}

// конструктор копирования

Pair(const Pair& other)

{

this->first = other.first;

this->second = other.second;

}

// деструктор

~Pair()

{

};

int get\_first()

{

return first;

}

int get\_second()

{

return second;

}

void set\_first(int m)

{

first = m;

}

void set\_second(int s)

{

second = s;

}

//перегруженные операции

Pair& operator=(const Pair&);

Pair operator=(int);

Pair operator -(const Pair&); // перегрузка оператора "-", вычитание полей объектов

Pair operator +(double constant); // перегрузка оператора "+" прибавление константы дабл к полям объекта класса

Pair operator +(Pair constant);

bool operator >(Pair other);

bool operator <(Pair other);

//глобальные функции ввода-вывода

friend istream& operator>>(istream& in, Pair& other); // объявление дружественного оператора ввода

friend ostream& operator<<(ostream& out, const Pair& other); // объявление дружественного оператора вывода

};

**Блок-Схема**







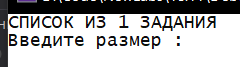
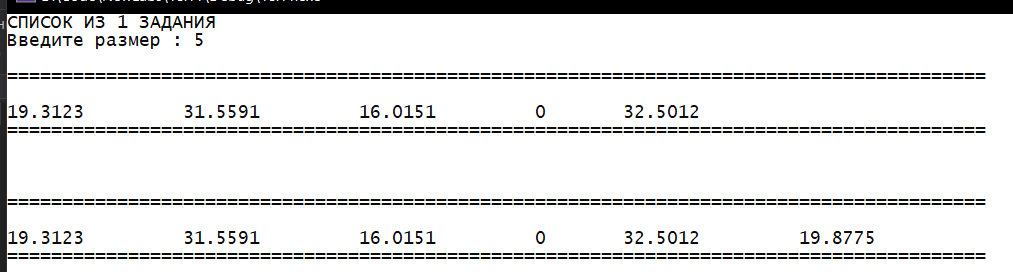
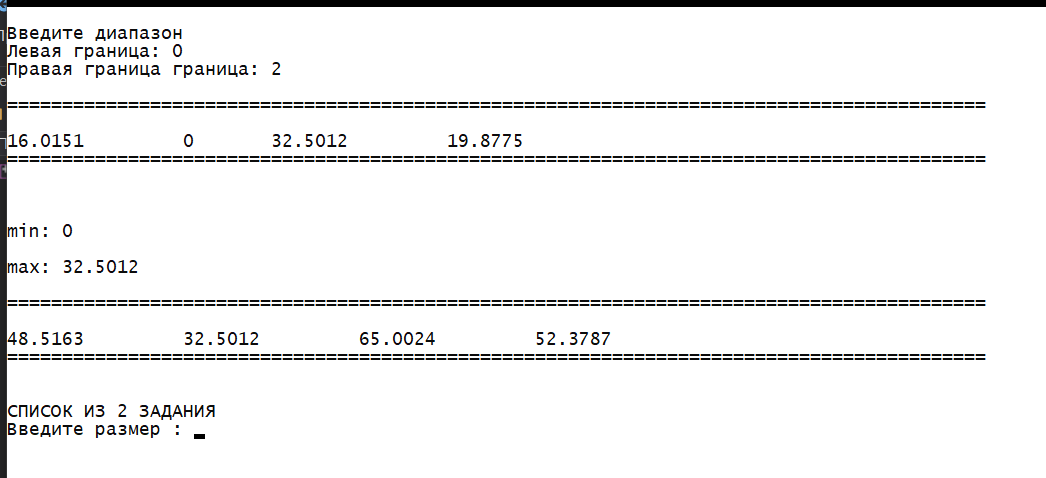
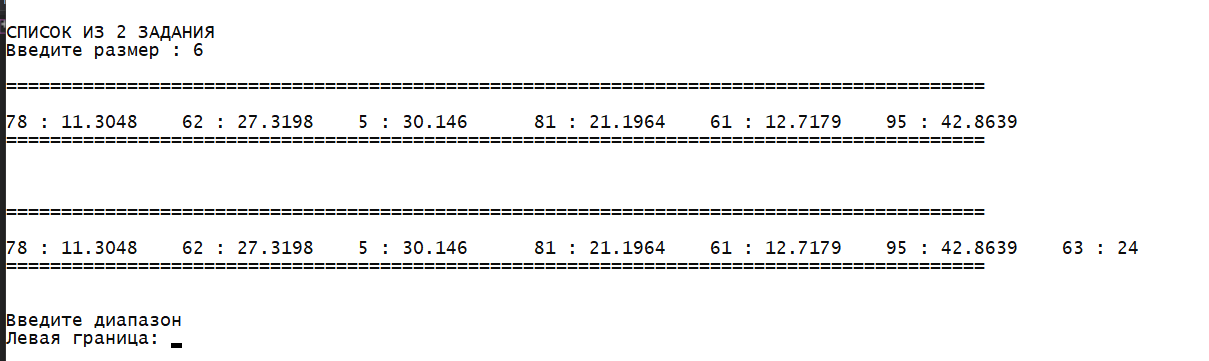
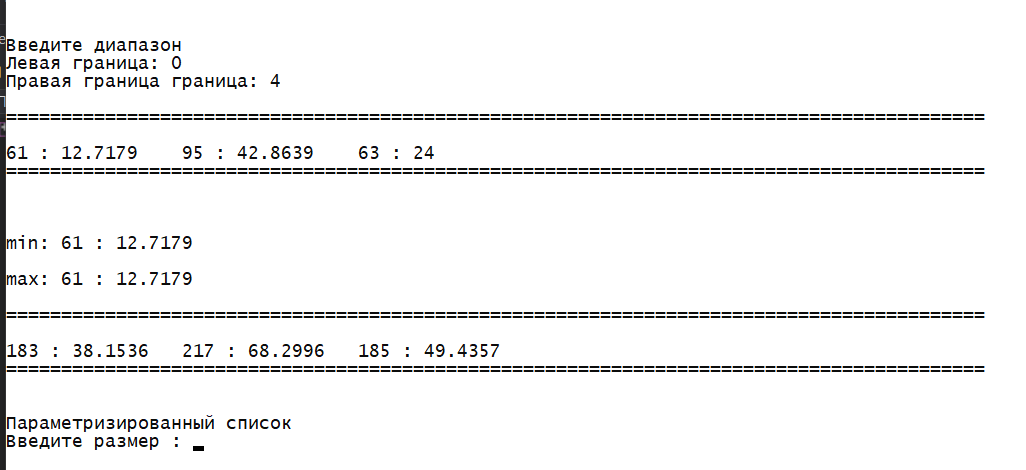
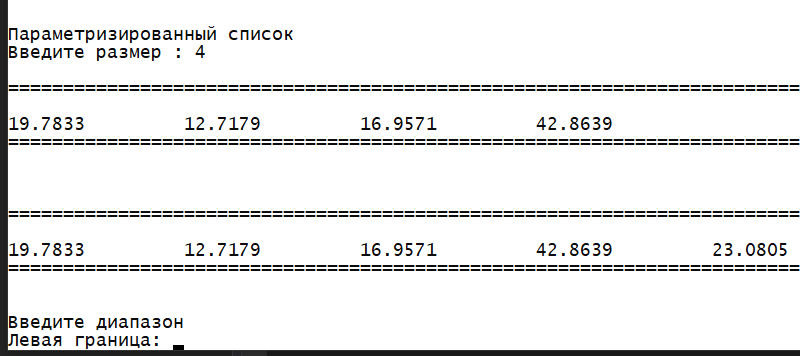
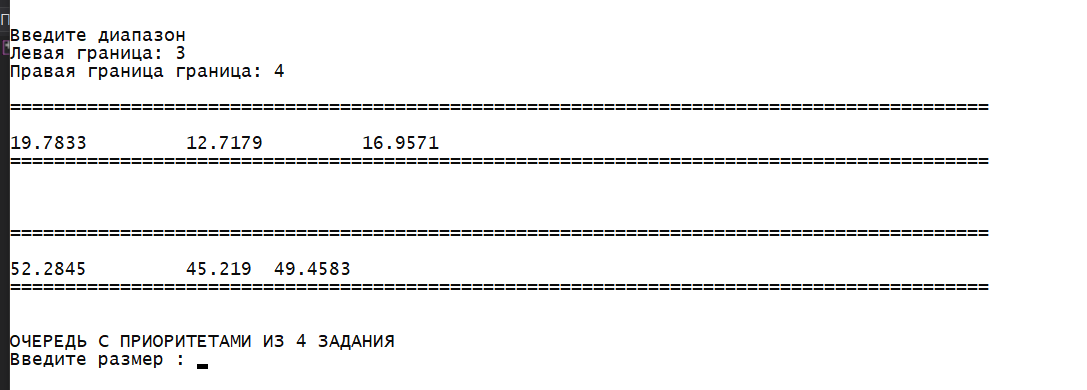
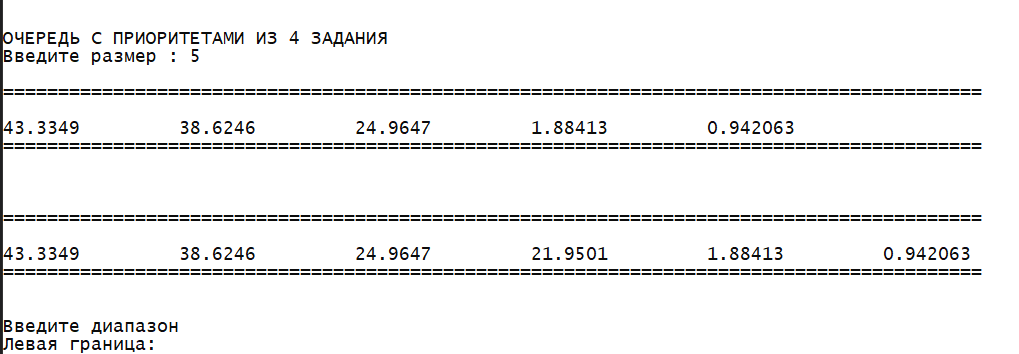
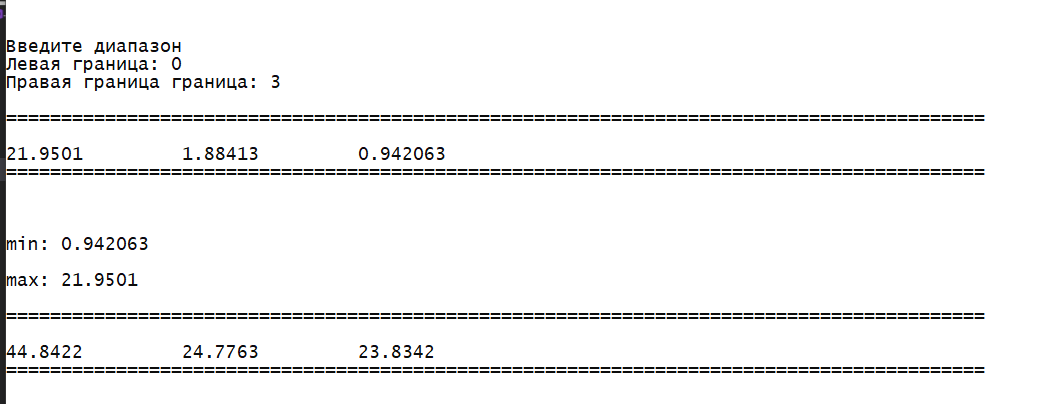








**Скриншоты**

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 

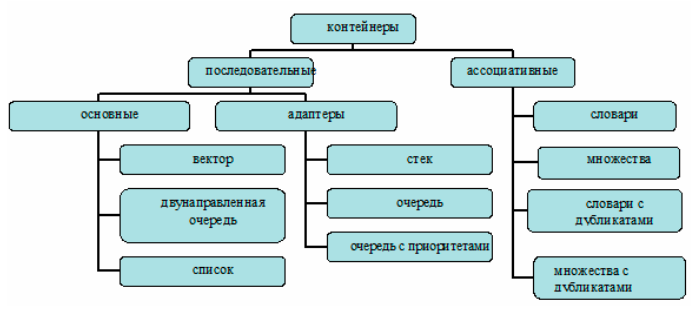
**Контрольные вопросы**



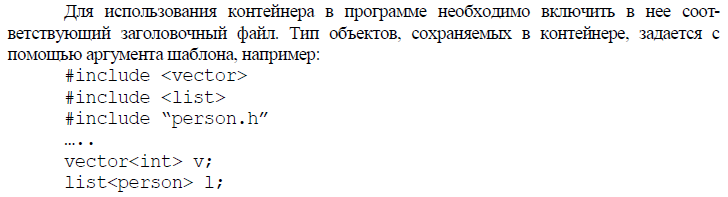
1. Набор контейнерных классов, набор обобщённых алгоритмов.



2. Последовательные, ассоциативные.



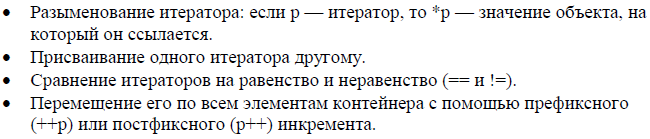


3. 

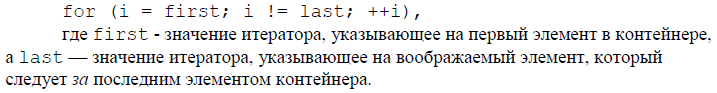


4. Итераторы – это обобщённые концепции указателей: они ссылаются на элементы контейнера.

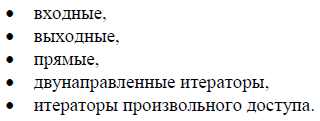


5. 

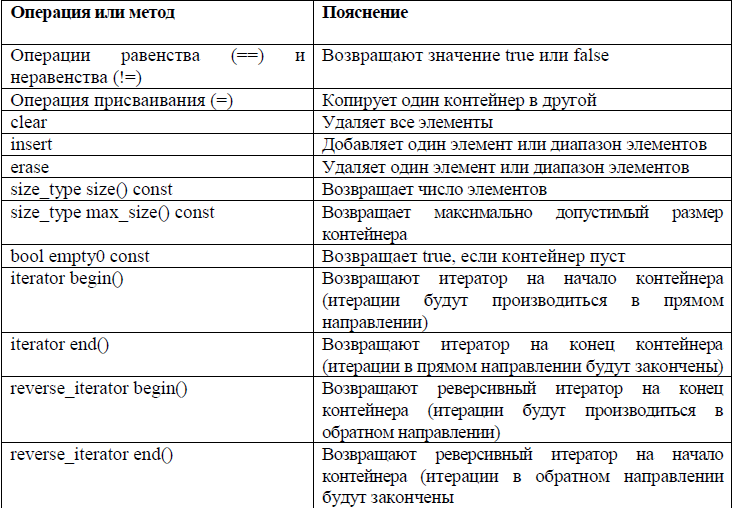


6. 



7. 



8. 



9. Добавление элемента в конец вектора, так как для добавления в середину или в начало вектора приходится сдвигать все следующие элементы путём копирования их значений.

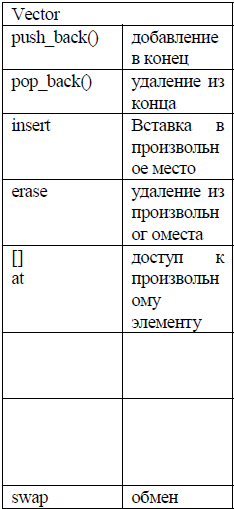


10. Вставка и удаление работают эффективно для любой позиции в списке, однако список не поддерживает произвольного доступа по индексу, так как для выборки n-го элемента нужно последовательно выбрать n-1 элементов.



11. Вставка и удаление первого и последнего элемента, так как двусторонняя очередь хранит элементы в непрерывной области памяти.



12. 



13. 



14. 



15. nameOfVector.erase(1,4);//индексирование от 0



16.nameOfVector.pop\_back();



17. nameOfList.erase(1,4);



18. nameOfList.pop\_back();

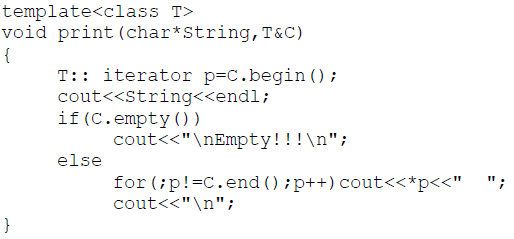


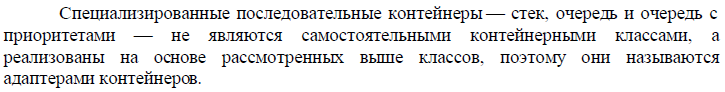
19. nameOfDeque.erase(1,4);



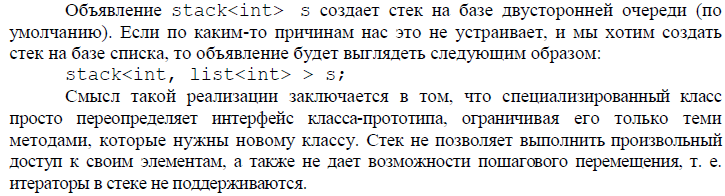
20. nameOfDeque.pop\_back();



21. 

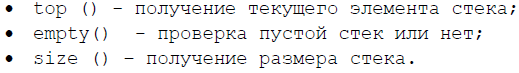
22. 



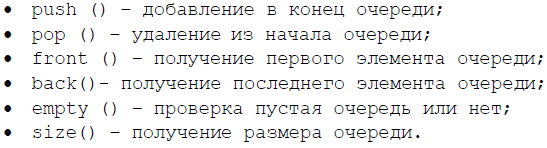
23. 



24.



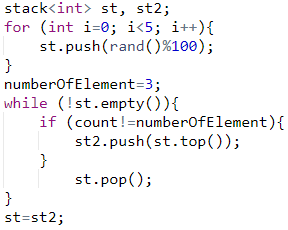


25. 



26. Очередь с приоритетами отличается от обычной очереди тем, что для извлечения выбирается максимальный элемент из хранимых в контейнере.

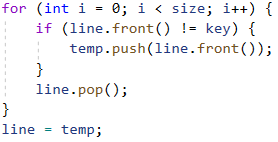


27. 

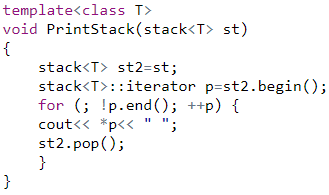


28.







29. 



30. 