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 10

Выполнил:

Студент группы ИВТ-20-2Б Галинов О.Ю.

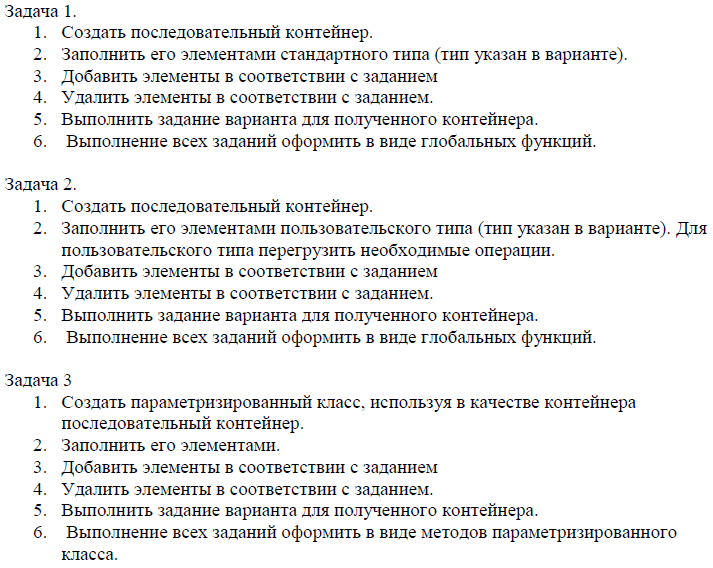
Проверила:

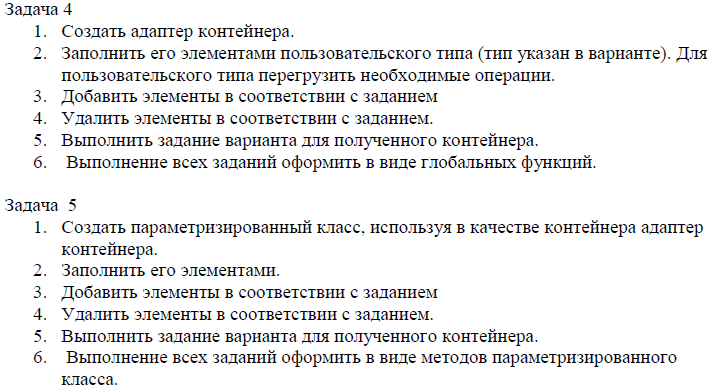
Доцент кафедры ИТАС

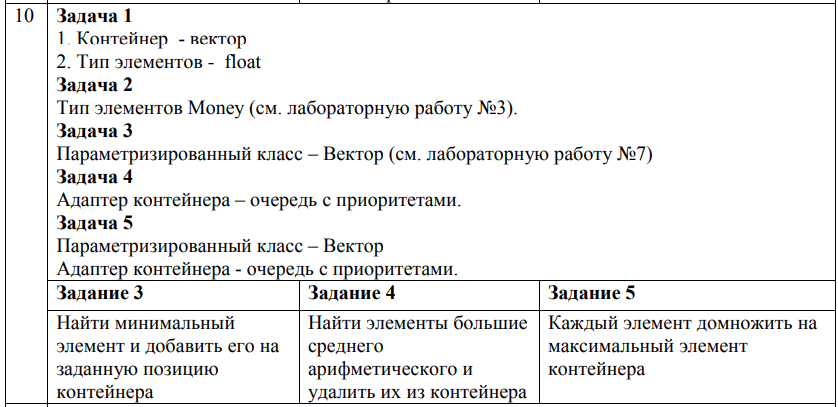
Полякова О.А.

Пермь, 2021

**Постановка задачи**







**Анализ задачи**

**1.** Для решения задач необходимо:

**1.1.** Организовать функцию Task1(), в которой будет использоваться библиотека vector для создания экземпляра вектора, для решения задачи №1.

**1.2.** Организовать функцию Task2(), в которой будет использоваться библиотека vector для создания экземпляра двунаправленной очереди типа Money, для решения задачи №2.

**1.3.** Организовать класс Money для хранения в полях для рублей и копеек(int: rub, kop) для экземпляров данного класса.

**1.4.** Организовать функцию Task3() для решения задачи №3.

**1.5.** Организовать перегруженный класс Vector с полями под размер массива и указатель собственного типа (например, Type).

**1.6.** Организовать функцию max() для поиска максимального элемента экземпляра класса queue.

**1.7.** Организовать функцию Task4(), в которой будет использоваться библиотека queue для создания экземпляра очереди, для решения задачи №4.

**1.8.** Организовать функцию Task5() для решения задачи №5.

**1.9.** Организовать шаблон класса Vector для использование его для типов Money.

**2.** В ходе работы были использованы следующие типы данных:

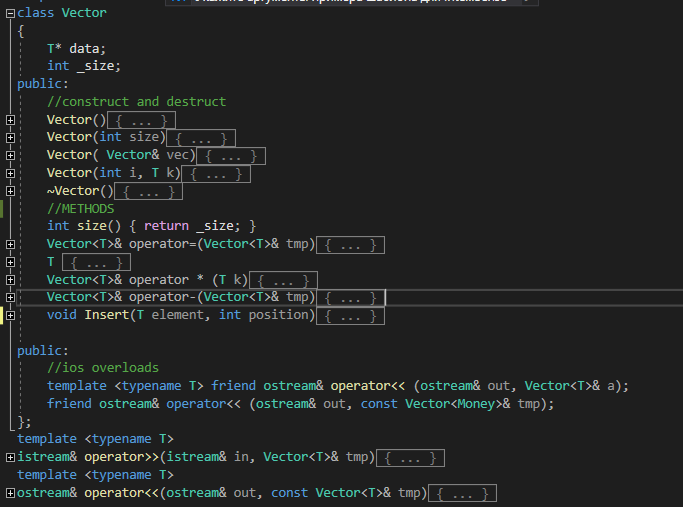
**2.1.** Функция Task1(), в которой будет использоваться библиотека vector для создания экземпляра вектора – l1, для решения задачи №1. Размер вектора изначально равен 10.

vector<float> l1;

**2.2.** Функция Task2(), в которой будет использоваться библиотека vector для создания экземпляра типа Money вектора – L, и в котором будет использоваться класс Money для создания экземпляра класса Money, для решения задачи №2.

vector<Money> L;

**2.3.** Функция Task3(), в которой будет использоваться, в которой будет использоваться класс Vector для создания экземпляра класса Vector, для решения задачи №3.

****

**2.4.** Функция Task4(), в которой будет использоваться библиотека priority\_queue для создания экземпляров приоритетной очереди – qu для решения задачи №4.

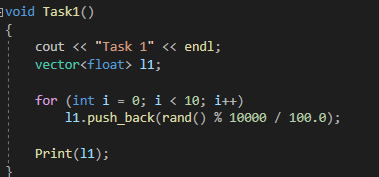
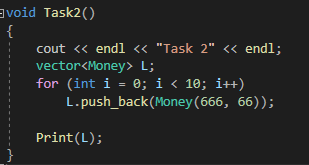


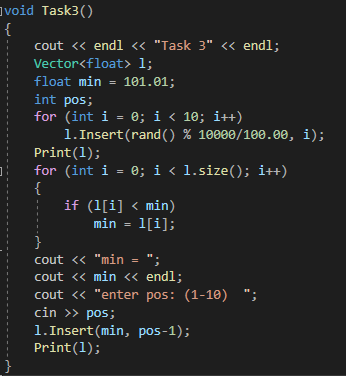
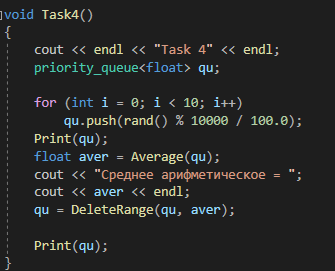
**2.5.** Функция Task5(), в которой будет использоваться библиотека priority\_queue для создания экземпляров приоритетной очереди – qu для решения задачи №5.

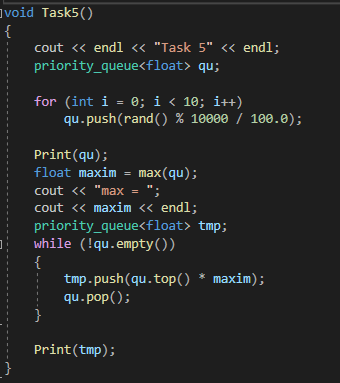


**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Решения задач осуществляются в функциях Task1(), Task2(), Task3(), Task4(), Task5().

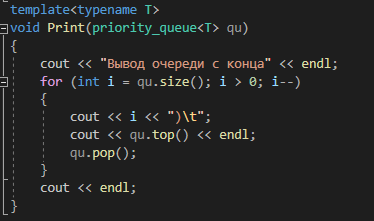
 



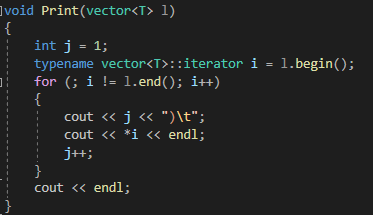
**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Для вывода на консоль приоритетной очереди используется функция Print().

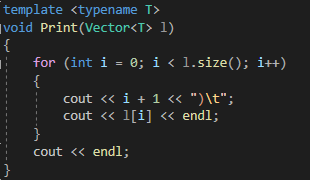


**4.2.** Для вывода на консоль объекта класса vector используется функция Print().



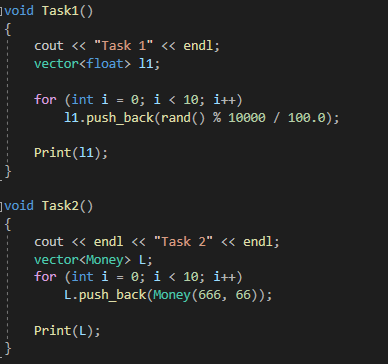


**4.3.** Для вывода экземпляра класса Vector используется функция Print().

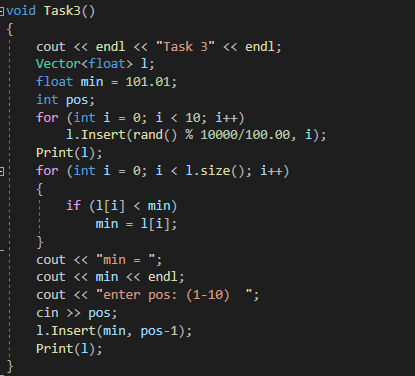


**5.** Поставленные задачи будут решены следующими действиями:

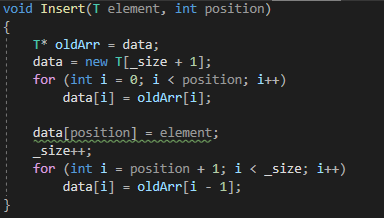
**5.1.** В функциях Task1() и Task2() используются методы из библиотеки priority\_queue и метод описанный самостоятельно в классе Vector - push\_back() для внесения в конец множества данных.



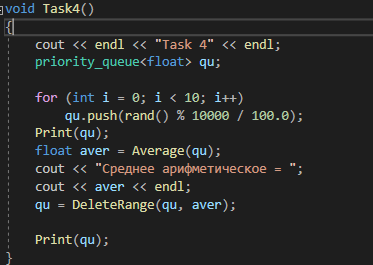
**5.2.** В функциях Task3() используются цикл, с помощью которого находится минимальный элемент и добавляется в вектор по определенному индексу.

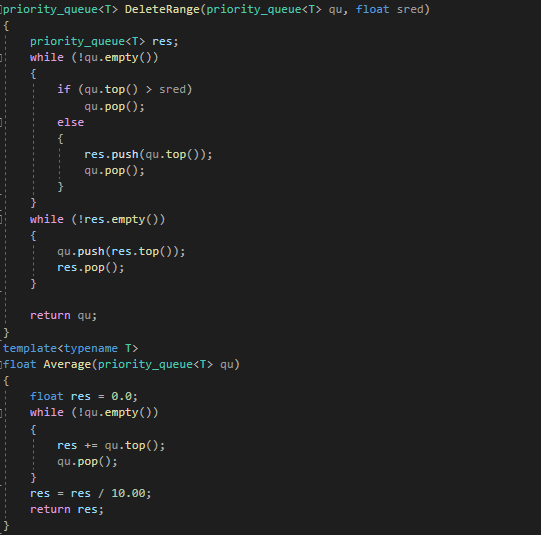


Для внесения максимального элемента множества используется метод Insert() класса Vector.

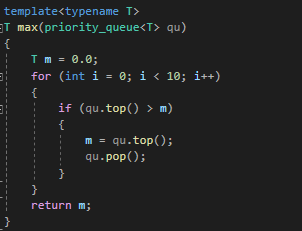


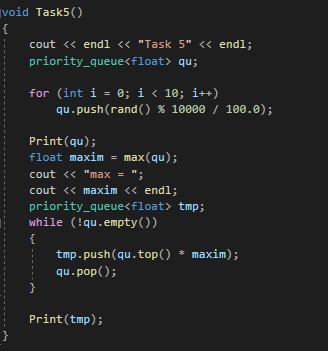
**5.3.** В функции Task4() описаны две функции для нахождения среднего арифметического и удаления из приоритетной очереди всех элементов больше среднего арифметического.



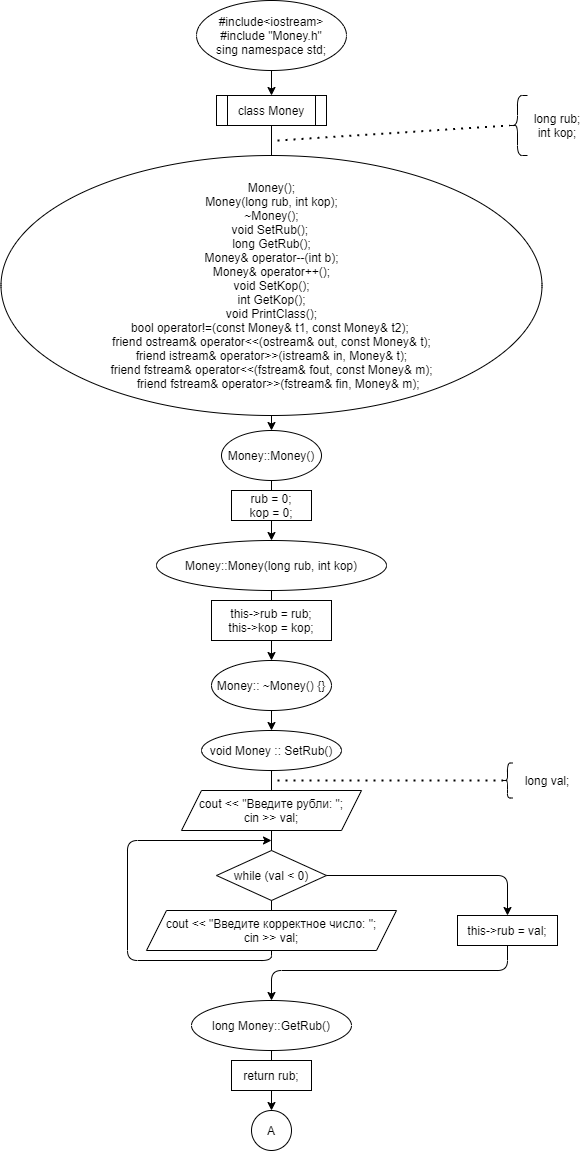


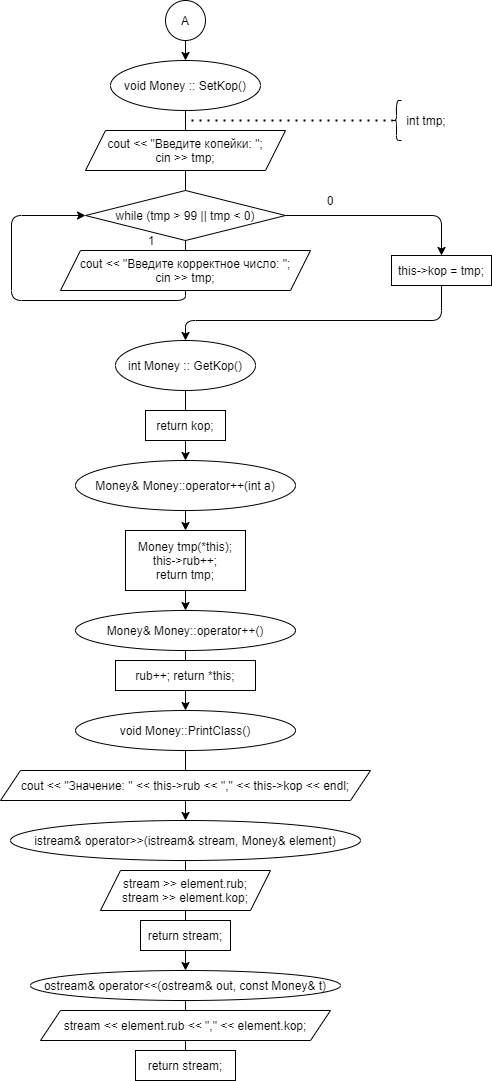
**5.4.** В функции Task5() создается приоритетная очередь, в которой находится максимальный элемент, а далее все элементы очереди умножаются на максимальный.

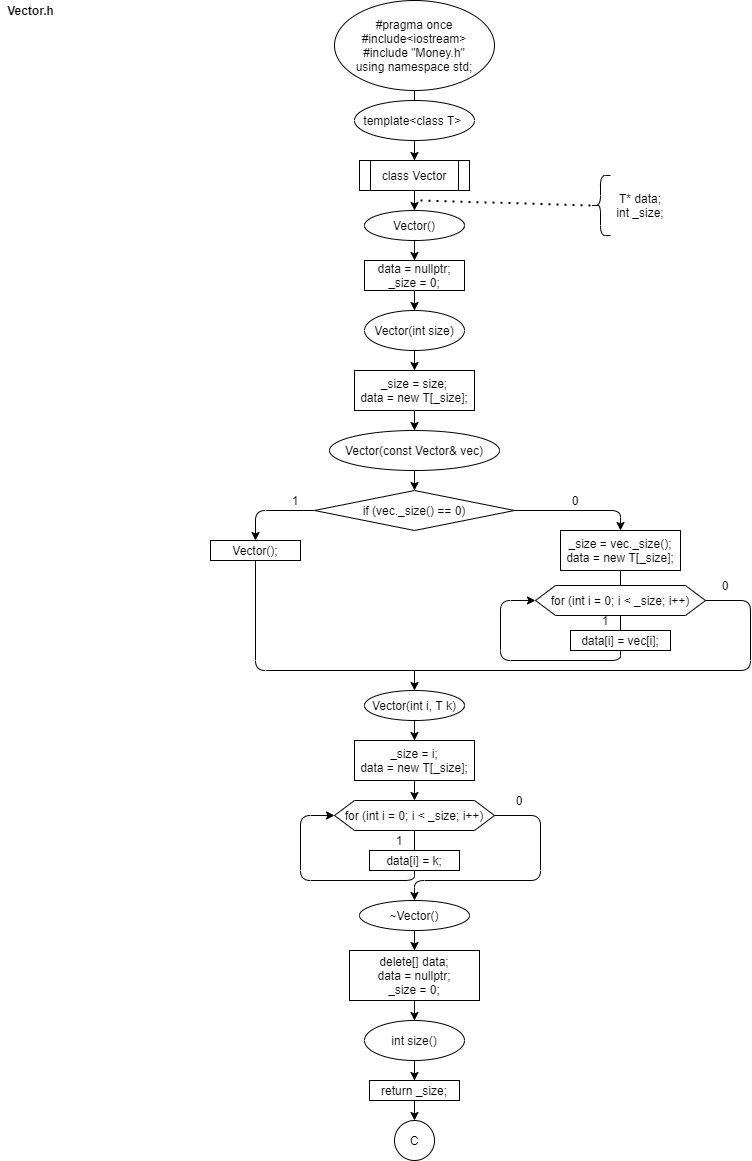


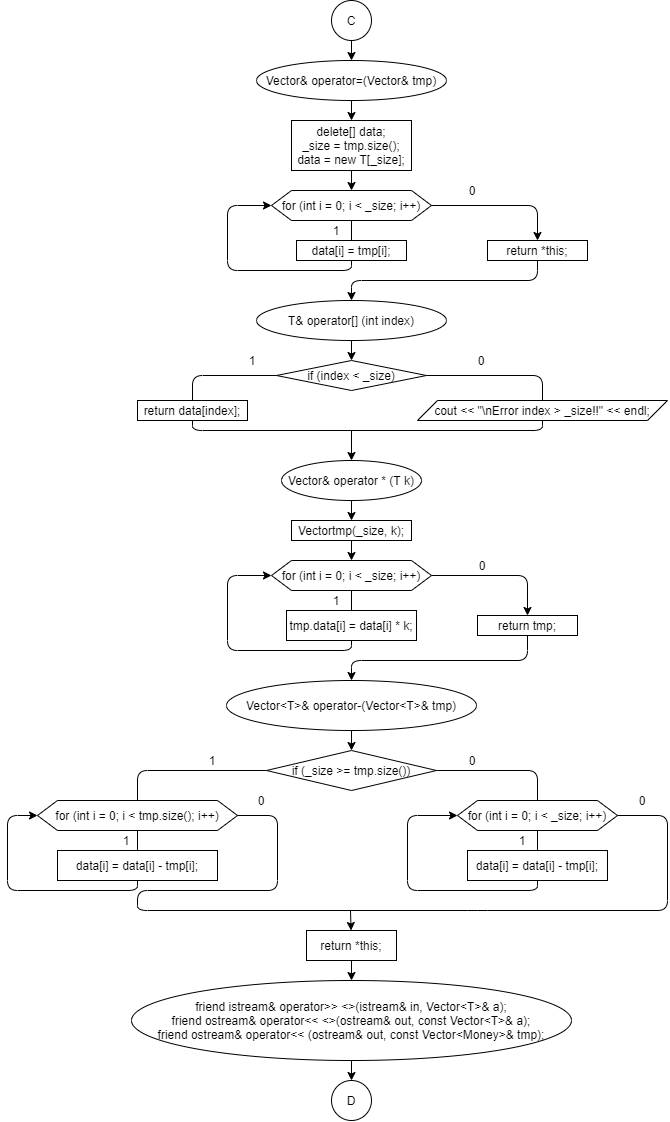


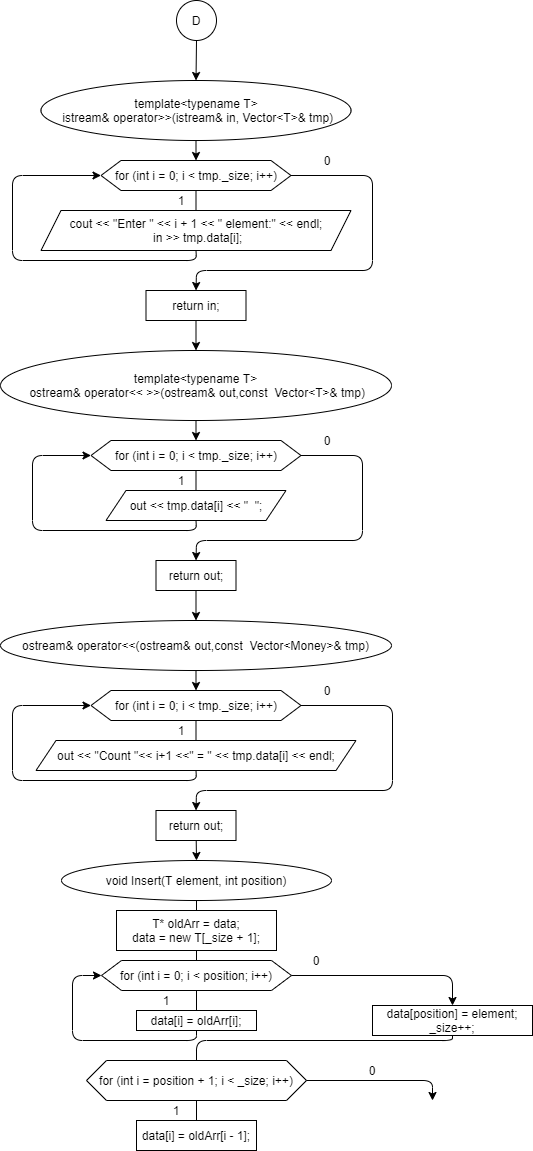
**Блок-схема**

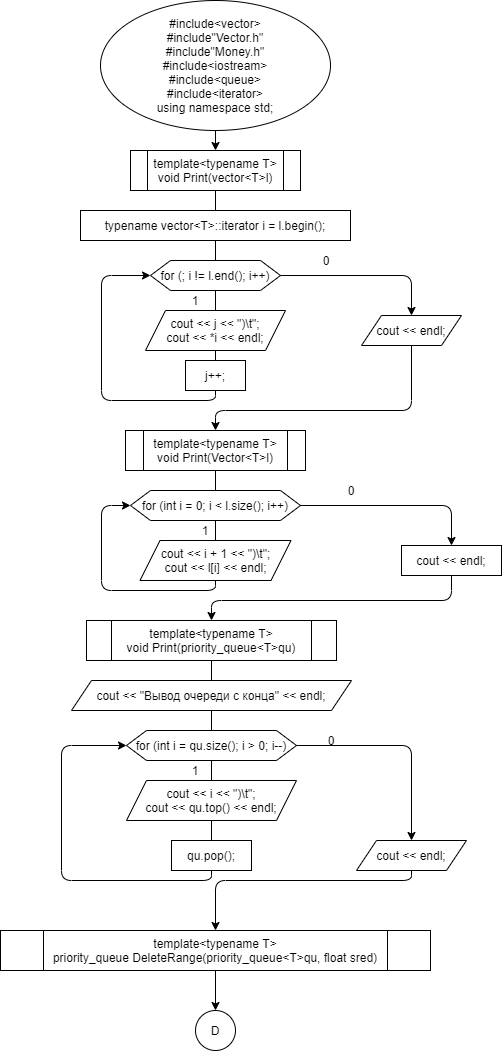


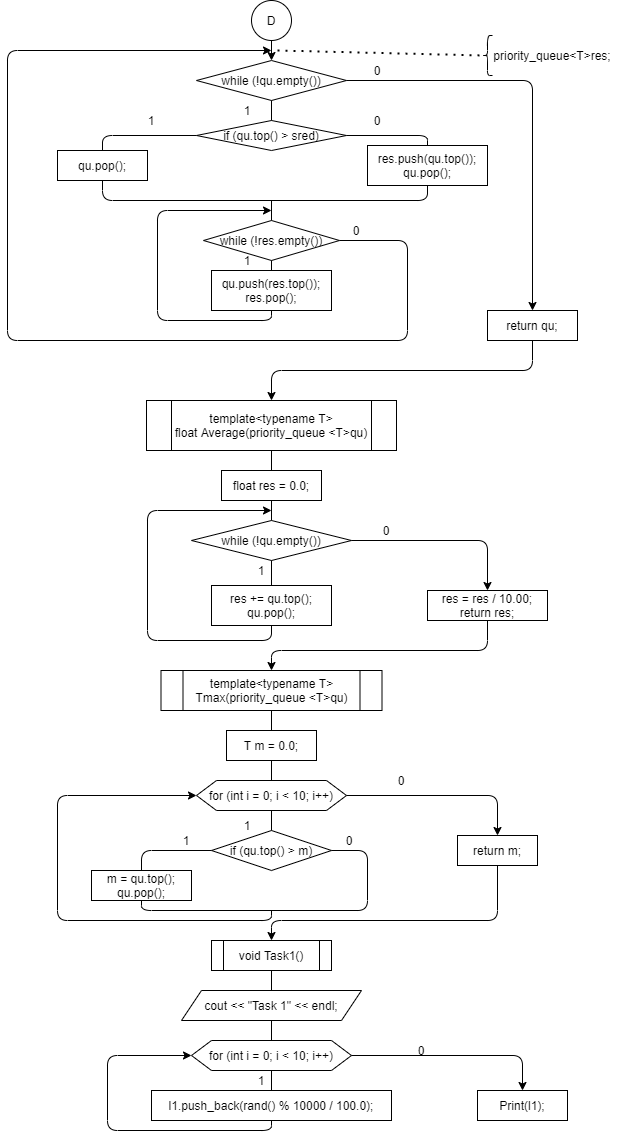


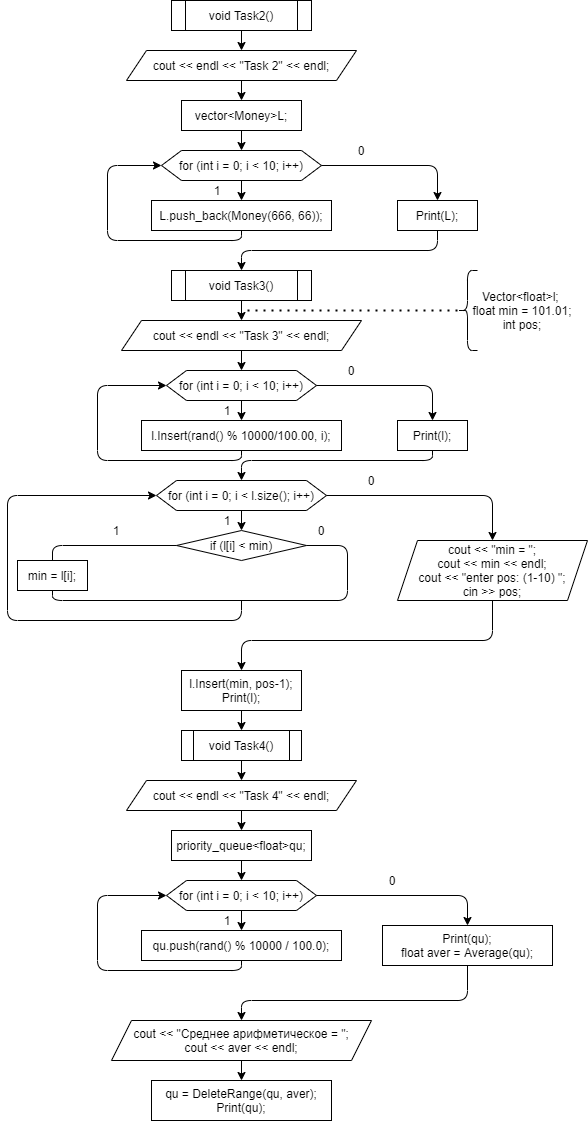


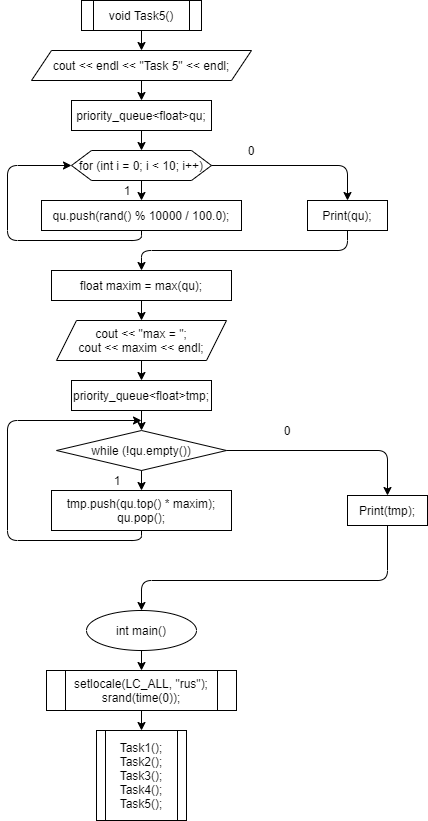












**Код**

#include <iostream>

#include"Vector.h"

#include"Money.h"

#include<vector>

#include<queue>

#include<iterator>

using namespace std;

template<typename T>

void Print(vector<T> l)

{

int j = 1;

typename vector<T>::iterator i = l.begin();

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

{

cout << j << ")\t";

cout << \*i << endl;

j++;

}

cout << endl;

}

template <typename T>

void Print(Vector<T> l)

{

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

{

cout << i + 1 << ")\t";

cout << l[i] << endl;

}

cout << endl;

}

template<typename T>

void Print(priority\_queue<T> qu)

{

cout << "Вывод очереди с конца" << endl;

for (int i = qu.size(); i > 0; i--)

{

cout << i << ")\t";

cout << qu.top() << endl;

qu.pop();

}

cout << endl;

}

template<typename T>

priority\_queue<T> DeleteRange(priority\_queue<T> qu, float sred)

{

priority\_queue<T> res;

while (!qu.empty())

{

if (qu.top() > sred)

qu.pop();

else

{

res.push(qu.top());

qu.pop();

}

}

while (!res.empty())

{

qu.push(res.top());

res.pop();

}

return qu;

}

template<typename T>

float Average(priority\_queue<T> qu)

{

float res = 0.0;

while (!qu.empty())

{

res += qu.top();

qu.pop();

}

res = res / 10.00;

return res;

}

template<typename T>

T max(priority\_queue<T> qu)

{

T m = 0.0;

for (int i = 0; i < 10; i++)

{

if (qu.top() > m)

{

m = qu.top();

qu.pop();

}

}

return m;

}

void Task1()

{

cout << "Task 1" << endl;

vector<float> l1;

for (int i = 0; i < 10; i++)

l1.push\_back(rand() % 10000 / 100.0);

Print(l1);

}

void Task2()

{

cout << endl << "Task 2" << endl;

vector<Money> L;

for (int i = 0; i < 10; i++)

L.push\_back(Money(666, 66));

Print(L);

}

void Task3()

{

cout << endl << "Task 3" << endl;

Vector<float> l;

float min = 101.01;

int pos;

for (int i = 0; i < 10; i++)

l.Insert(rand() % 10000/100.00, i);

Print(l);

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

{

if (l[i] < min)

min = l[i];

}

cout << "min = ";

cout << min << endl;

cout << "enter pos: (1-10) ";

cin >> pos;

l.Insert(min, pos-1);

Print(l);

}

void Task4()

{

cout << endl << "Task 4" << endl;

priority\_queue<float> qu;

for (int i = 0; i < 10; i++)

qu.push(rand() % 10000 / 100.0);

Print(qu);

float aver = Average(qu);

cout << "Среднее арифмeтическое = ";

cout << aver << endl;

qu = DeleteRange(qu, aver);

Print(qu);

}

void Task5()

{

cout << endl << "Task 5" << endl;

priority\_queue<float> qu;

for (int i = 0; i < 10; i++)

qu.push(rand() % 10000 / 100.0);

Print(qu);

float maxim = max(qu);

cout << "max = ";

cout << maxim << endl;

priority\_queue<float> tmp;

while (!qu.empty())

{

tmp.push(qu.top() \* maxim);

qu.pop();

}

Print(tmp);

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(0));

Task1();

Task2();

Task3();

Task4();

Task5();

}

#pragma once

#include<iostream>

using namespace std;

class Money

{

long rub;

int kop;

public:

Money();

Money(long rub, int kop);

~Money();

void SetRub();

long GetRub();

Money& operator++();

Money& operator++(int a);

void SetKop();

int GetKop();

void PrintClass();

friend istream& operator>>(istream& stream, Money& element);

friend ostream& operator<<(ostream& stream, Money element);

};

#include <iostream>

#include "Money.h"

using namespace std;

Money::Money()

{

rub = 0;

kop = 0;

}

Money::Money(long rub, int kop)

{

this->rub = rub;

this->kop = kop;

}

Money:: ~Money() {}

void Money::SetRub()

{

long val;

cout << "Введите рубли: ";

cin >> val;

while (val < 0)

{

cout << "Введите корректное число: ";

cin >> val;

}

this->rub = val;

}

long Money::GetRub() { return rub; }

void Money::SetKop()

{

int tmp;

cout << "Введите копейки: ";

cin >> tmp;

while (tmp > 99 || tmp < 0)

{

cout << "Введите корректное число: ";

cin >> tmp;

}

this->kop = tmp;

}

int Money::GetKop() { return kop; }

void Money::PrintClass()

{

cout << "Значение: " << this->rub << "," << this->kop << endl;

}

Money& Money::operator++()

{

rub++;

return \*this;

}

Money& Money::operator++(int a)

{

Money tmp(\*this);

this->rub++;

return tmp;

}

istream& operator>>(istream& stream, Money& element)

{

stream >> element.rub;

stream >> element.kop;

return stream;

}

ostream& operator<<(ostream& stream, Money element)

{

stream << element.rub << "," << element.kop;

return stream;

}

#pragma once

#include<iostream>

#include "Money.h"

#include <typeinfo>

using namespace std;

template <class T>

class Vector

{

T\* data;

int \_size;

public:

//construct and destruct

Vector()

{

data = nullptr;

\_size = 0;

}

Vector(int size)

{

\_size = size;

data = new T[\_size];

}

Vector( Vector& vec)

{

\_size = vec.\_size;

data = new T[vec.\_size];

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

data[i] = vec.data[i];

}

Vector(int i, T k)

{

\_size = i;

data = new T[\_size];

for (int i = 0; i < \_size; i++)

data[i] = k;

}

~Vector()

{

delete[] data;

data = nullptr;

\_size = 0;

}

//METHODS

int size() { return \_size; }

Vector<T>& operator=(Vector<T>& tmp)

{

delete[] data;

\_size = tmp.size();

data = new T[\_size];

for (int i = 0; i < \_size; i++)

{

data[i] = tmp[i];

}

return \*this;

}

T operator[] (int index)

{

if (index < \_size)

return data[index];

else

cout << "\nError! index > \_size!!" << endl;

}

Vector<T>& operator \* (T k)

{

Vector <T> tmp(\_size, k);

for (int i = 0; i < \_size; i++)

tmp.data[i] = data[i] \* k;

return tmp;

}

Vector<T>& operator-(Vector<T>& tmp)

{

if (\_size >= tmp.size())

{

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

{

data[i] = data[i] - tmp[i];

}

}

else

{

for (int i = 0; i < \_size; i++)

{

data[i] = data[i] - tmp[i];

}

}

return \*this;

}

void Insert(T element, int position)

{

T\* oldArr = data;

data = new T[\_size + 1];

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

data[i] = oldArr[i];

data[position] = element;

\_size++;

for (int i = position + 1; i < \_size; i++)

data[i] = oldArr[i - 1];

}

public:

//ios overloads

template <typename T> friend ostream& operator<< (ostream& out, Vector<T>& a);

friend ostream& operator<< (ostream& out, const Vector<Money>& tmp);

};

template <typename T>

istream& operator>>(istream& in, Vector<T>& tmp)

{

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

{

cout << "Ââåäèòå " << i + 1 << " ýëåìåíò âåêòîðà: ";

in >> tmp.data[i];

}

return in;

}

template <typename T>

ostream& operator<<(ostream& out, const Vector<T>& tmp)

{

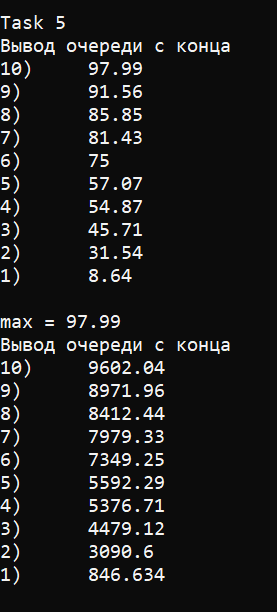
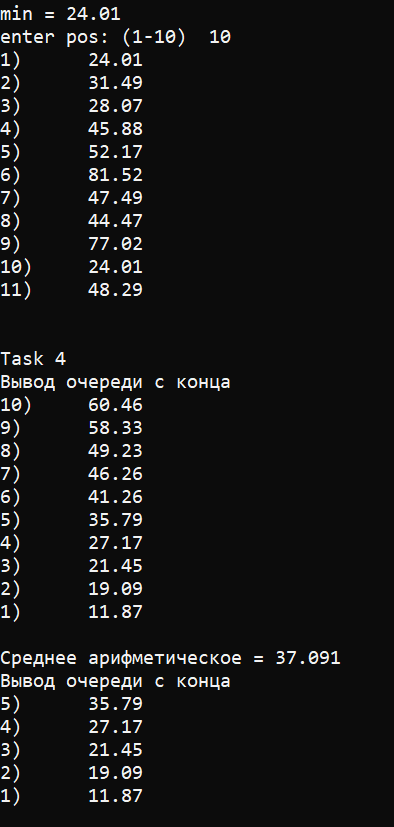
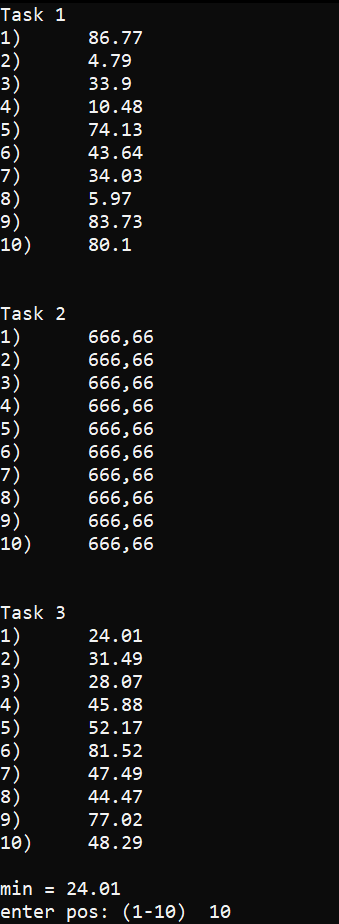
for (int i = 0; i < tmp.\_size; i++)

out << tmp.data[i] << " ";

return out;

}

**Скриншоты**

****

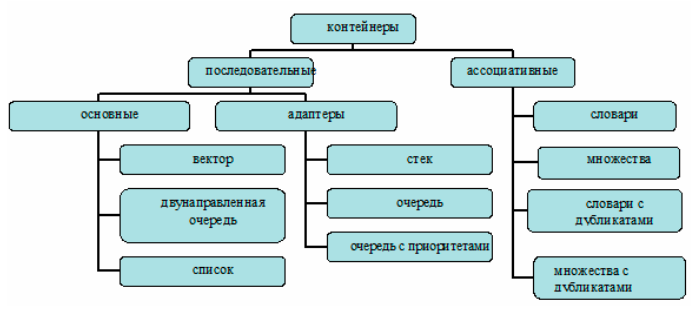
**Контрольные вопросы**



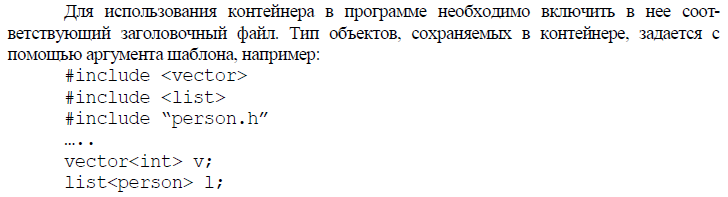
1. Набор контейнерных классов, набор обобщённых алгоритмов.



2. Последовательные, ассоциативные.



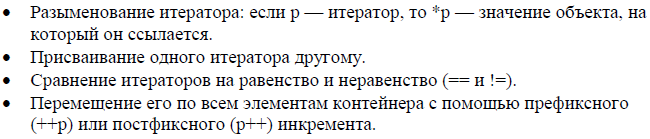


3. 

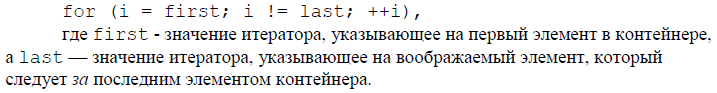


4. Итераторы – это обобщённые концепции указателей: они ссылаются на элементы контейнера.

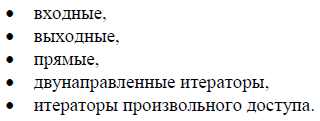


5. 

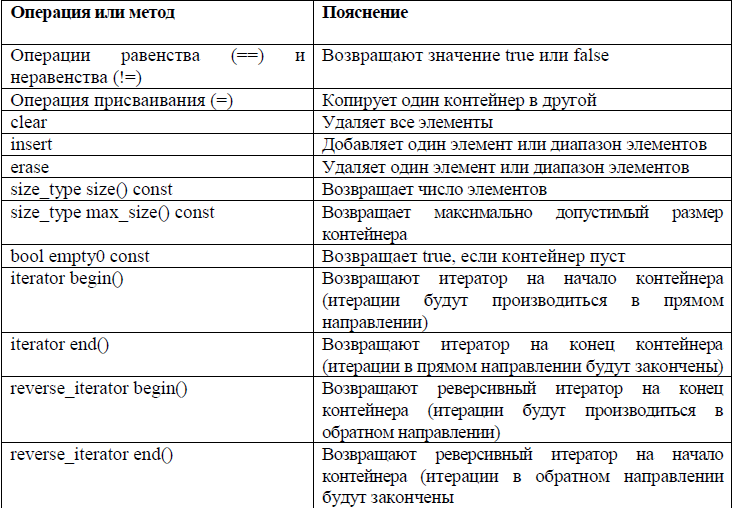


6. 



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.

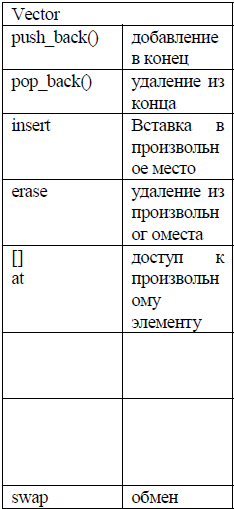


10. Вставка и удаление работают эффективно для любой позиции в списке, однако список не поддерживает произвольного доступа по индексу, так как для выборки n-го элемента нужно последовательно выбрать n-1 элементов.



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();

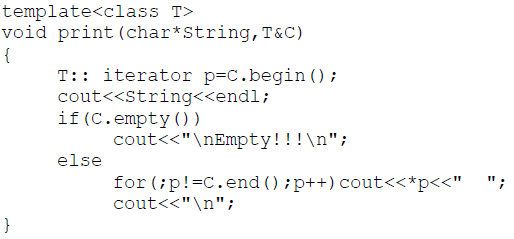


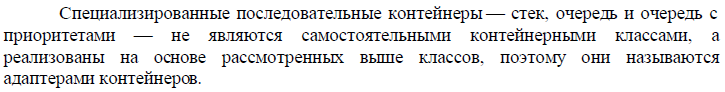
19. nameOfDeque.erase(1,4);



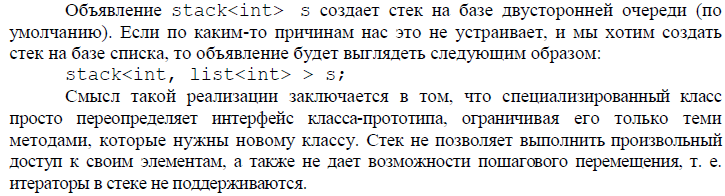
20. nameOfDeque.pop\_back();



21. 

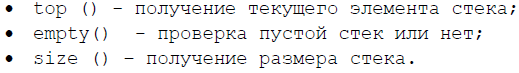
22. 



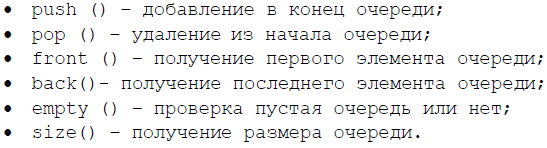
23. 



24.



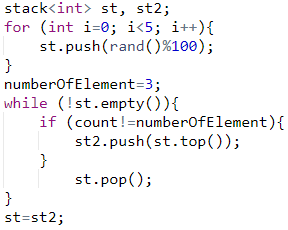


25. 



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

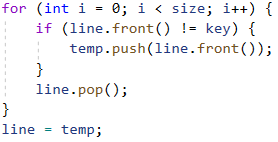


27. 

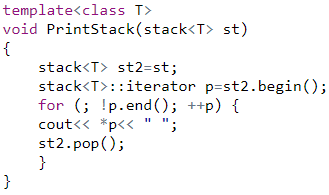


28.







29. 



30. 