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 19

Выполнил:

Студент группы РИС-20-2Б

Пономарёв Артём Викторович

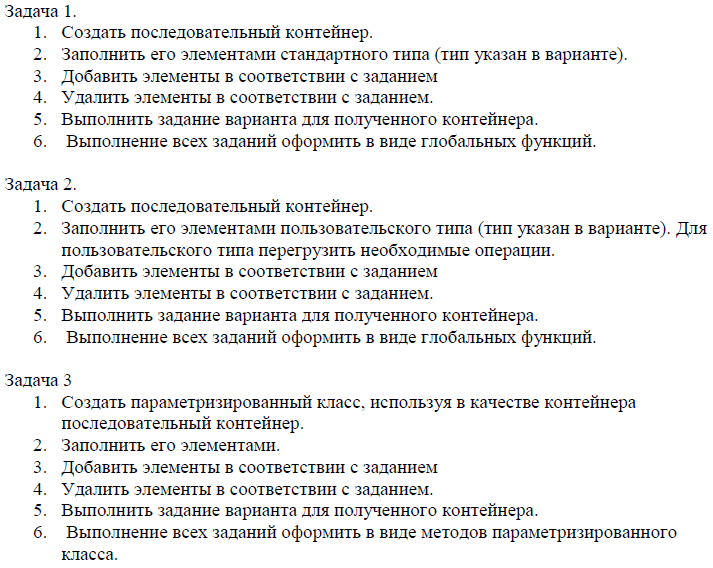
Проверила:

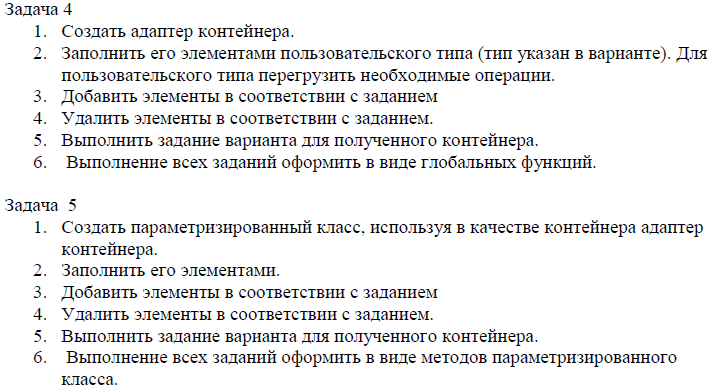
Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**





2

3

**Анализ задачи**

**1.** Для решения задач необходимо:

**1.1.** Организовать функцию Problem1(), в которой будет использоваться библиотека deque для создания экземпляра двунаправленной очереди, для решения задачи №1.

**1.2.** Организовать функцию Problem2(), в которой будет использоваться библиотека deque для создания экземпляра двунаправленной очереди типа Time, для решения задачи №2.

**1.3.** Организовать класс Time для хранения в полях для минут и секунд (int: mins, secs) для экземпляров данного класса.

**1.4.** Организовать функцию Problem3() 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 AddingMaxValue() для поиска максимального элемента экземпляра класса Vector и добавления этого элемента в экземпляр.

**1.7.** Организовать функцию Problem4(), в которой будет использоваться библиотека queue для создания экземпляра очереди, для решения задачи №4.

**1.8.** Организовать функцию Problem5() для решения задачи №5.

**1.9.** Организовать перегруженный класс Vector5 с полями под размер очереди и очереди перегруженного типа Type.

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

**2.1.** В функции main() используется пременная vvod типа int для меню, реализованного через ветвление switch().

int vvod = 6;

while (vvod != 0) {

cout << "\nКакое задание?" << endl

<< "1) " << endl

<< "2) " << endl

<< "3) " << endl

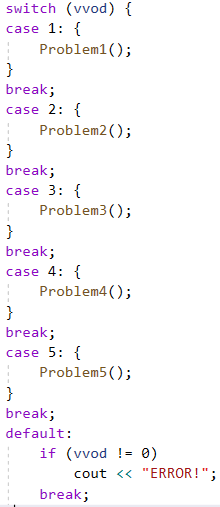
<< "4) " << "\n"

<< "5)" << endl

<< "0) Выход из меню" << "\n\n" << ">";

cin >> vvod;

4



**2.2.** Функция Problem1(), в которой будет использоваться библиотека deque для создания экземпляра двунаправленной очереди - dq, для решения задачи №1. Переменная size устанавливает размер очереди.

deque<int> dq;

int size = SafeInput1(true);

**2.3.** Функция Problem2(), в которой будет использоваться библиотека deque для создания экземпляра типа Time двунаправленной очереди – dq, и в которой будет использоваться класс Time для создания экземпляра класса Time, для решения задачи №2. Переменная size устанавливает размер очереди, переменная tmp необходима для хранения временных данных.

Time obj;

deque<Time>dq;

int size = SafeInput1(true);

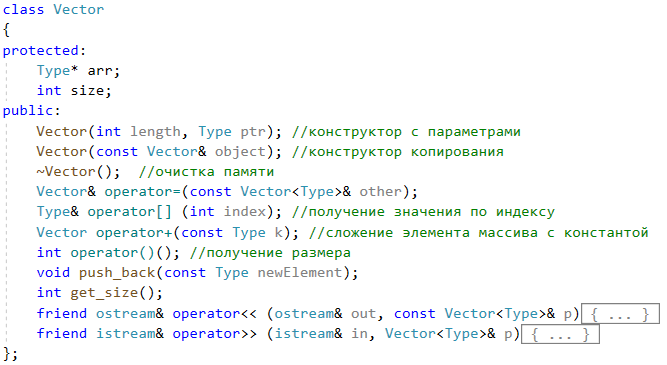
int tmp = obj.GetMins();

**2.4.** Функция Problem3(), в которой будет использоваться, в которой будет использоваться класс Vector для создания экземпляра класса Vector, для решения задачи №3. Переменная size устанавливает размер массива, переменная tmp необходима для хранения временных данных.

int size = SafeInput1(true);

Vector<int> intVec(size, 0);

5



**2.5.** Функция Problem4(), в которой будет использоваться библиотека queue для создания экземпляров очереди – line, temp, для решения задачи №4. Переменная size устанавливает размер очереди.

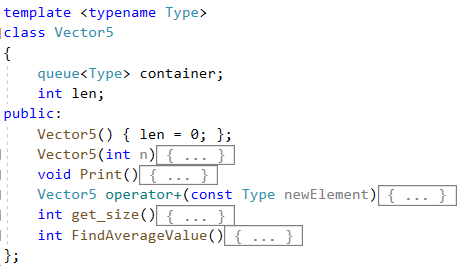
int size = SafeInput1(true);

queue<int> line, temp;

**2.6.** Функция Problem5(), в которой будет использоваться перегруженный класс Vector5 с полями под размер очереди и очереди перегруженного типа Type, для решения задачи №5. Переменная size устанавливает размер очереди.

int size = SafeInput1(true);

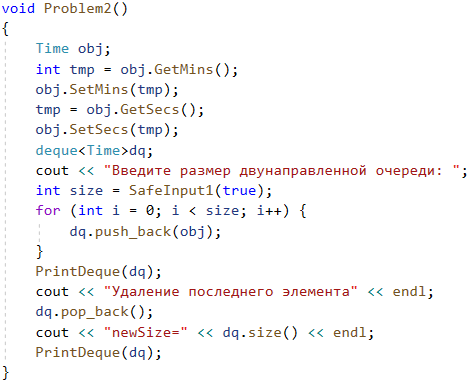
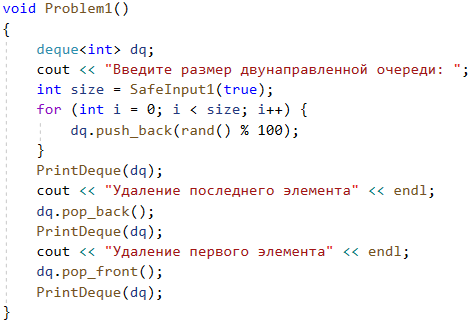
Vector5<int> intVec(size);

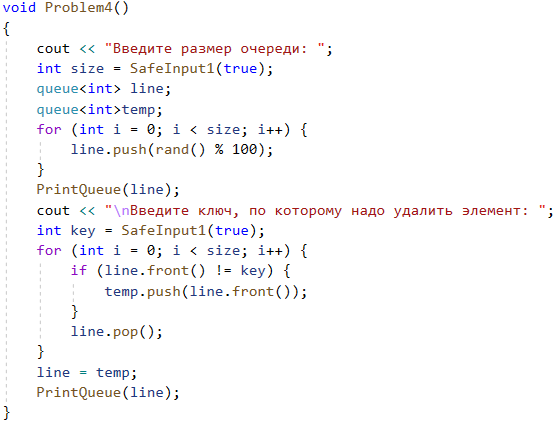
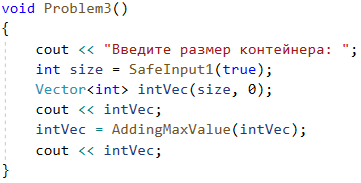


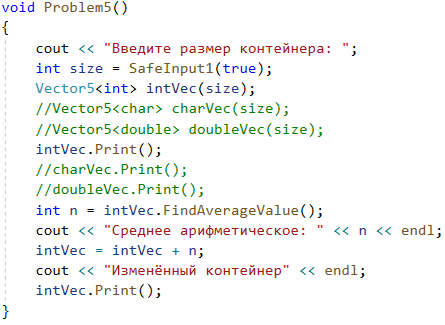
**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Решения задач осуществляются в функциях Problem1(), Problem2(), Problem3(), Problem4(), Problem5().

6



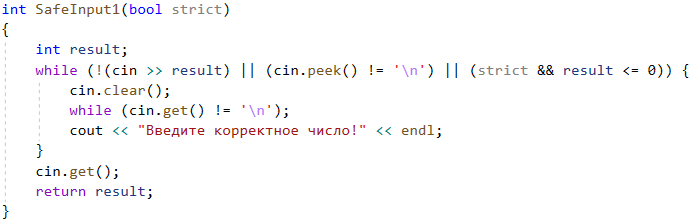




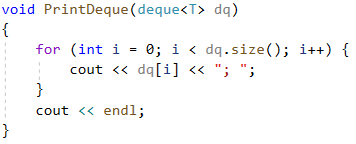
**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Для ввода целочисленных значений используется функция SafeInput1(), которая проверяет введённое данное. Таким образом будут игнорироваться данные типа char, double, float.

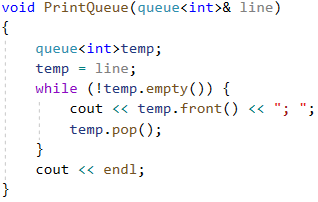
7



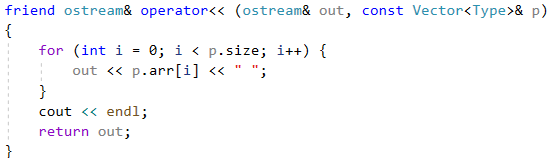
**4.2.** Для вывода на консоль двунаправленной очереди используется функция PrintDeque().



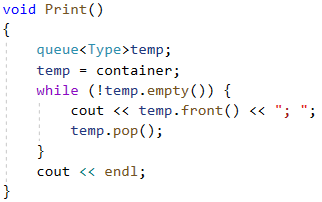
**4.3.** Для вывода на консоль очереди используется функция PrintQueue().



**4.4.** Для вывода экземпляра класса Vector используется перегрузка оператора <<.



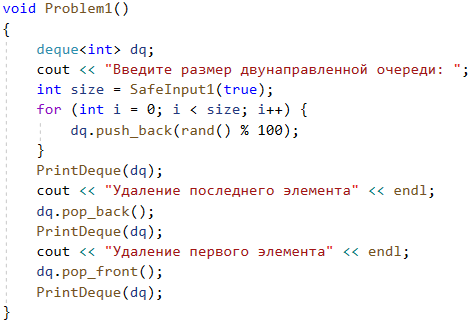
**4.4.** Для вывода экземпляра класса Vector5 используется метод Print().

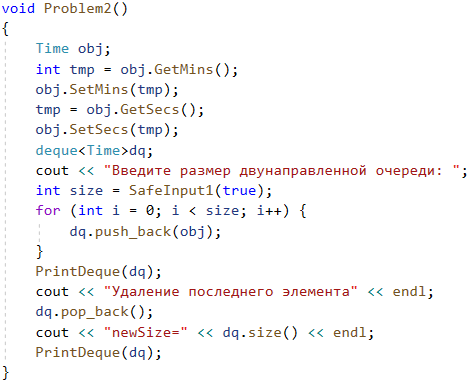


8

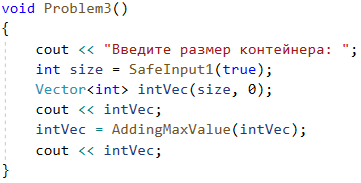
**5.** Поставленные задачи будут решены следующими действиями:

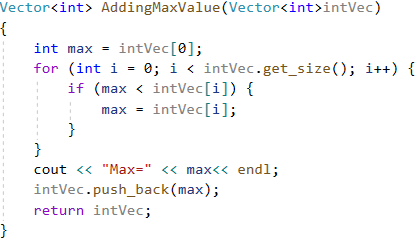
**5.1.** В функциях Problem1() и Problem2() используются методы из библиотеки deque - push\_back() и pop\_back() для внесения в конец множества данных и удаления последнего элемента множества.





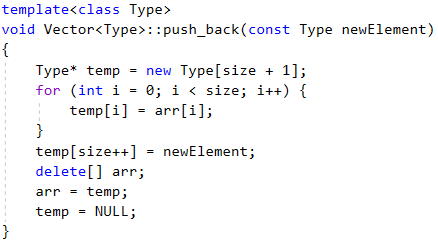
**5.2.** В функциях Problem3() используются функция AddingMaxValue(), которая находит максимальный элемент множества класса Vector и добавляет его в конец.



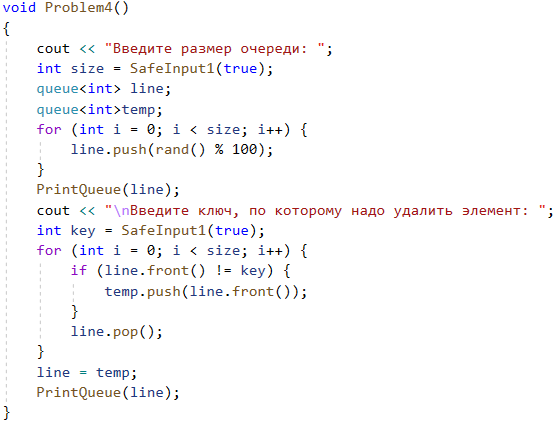


9

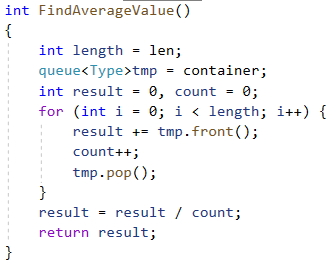
Для внесения максимального элемента множества используется метод push\_back() класса Vector.



**5.3.** В функции Problem4() используются методы библиотеки queue push(), pop(), front() для внесения элемента в очередь, удаления элемента из очереди и для вызова первого внесённого элемента в множество.



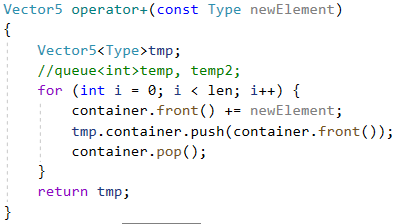
**5.4.** В функции Problem5() используются перегруженные методы перегруженного класса Vector5. Метод FindAverageValue() необходим для поиска среднего арифметического значения.

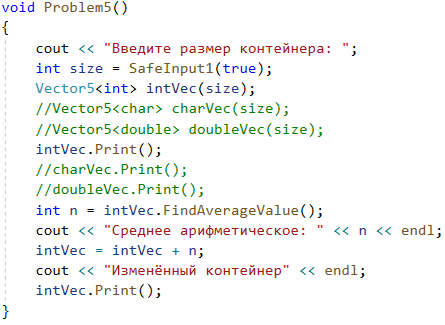


Создаётся очередь типа Type – tmp, в которую записывается введённая очередь (поле container). В цикле for происходит суммирование всех элементов в переменную result и увеличение переменной count – счётчика на 1. Возвращается значение result, делённое на count.

10

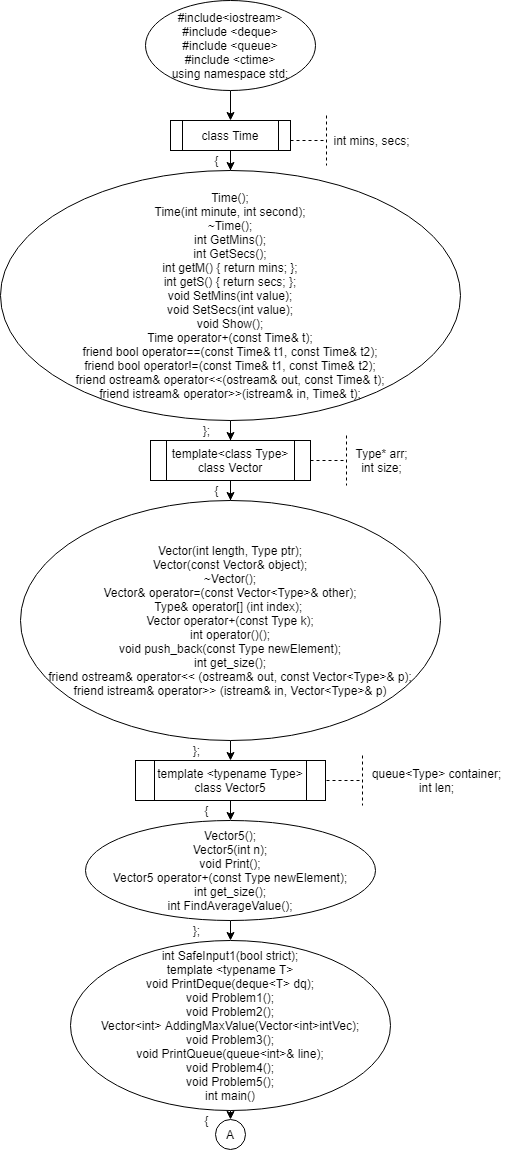
В перегрузке оператора + происходит прибавление среднего арифметического значения ко всем элементам множества.



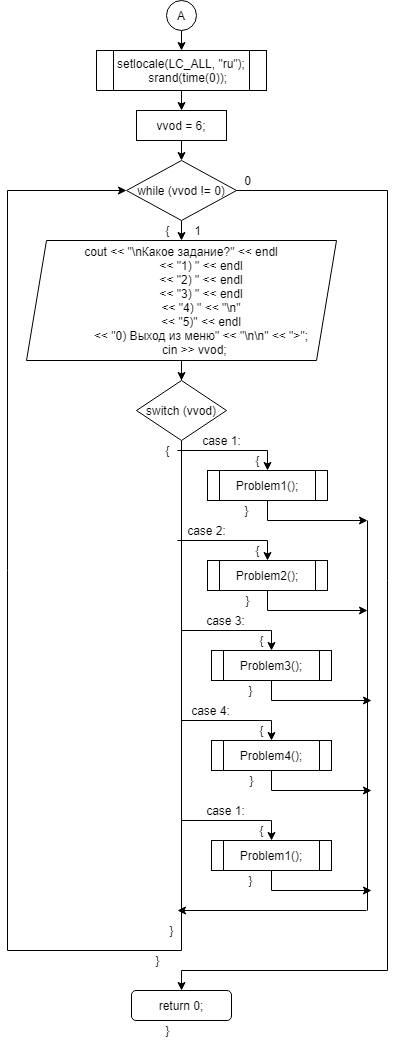


11

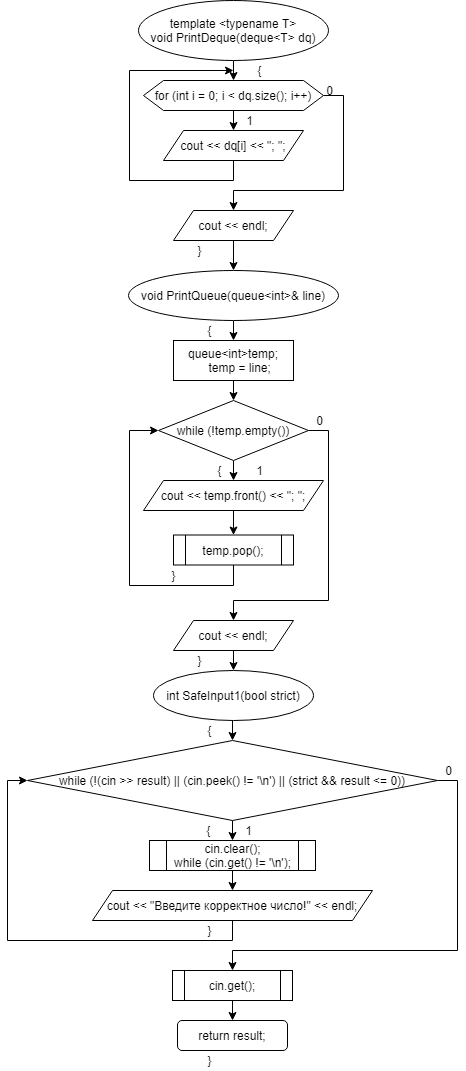
**Блок-схема**



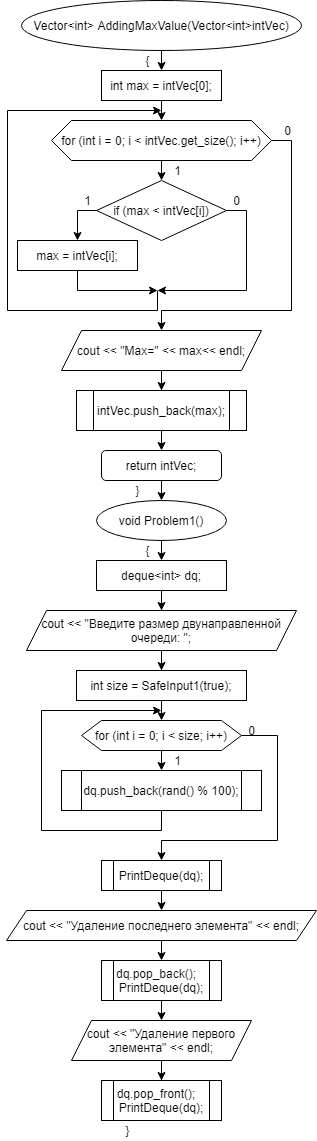
12



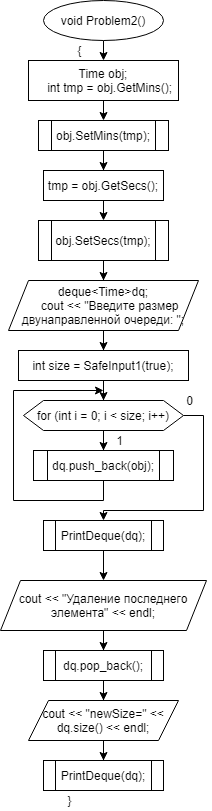
13



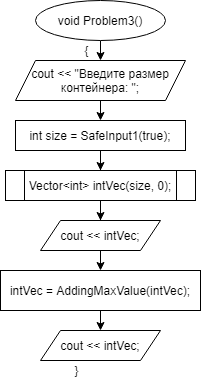
14



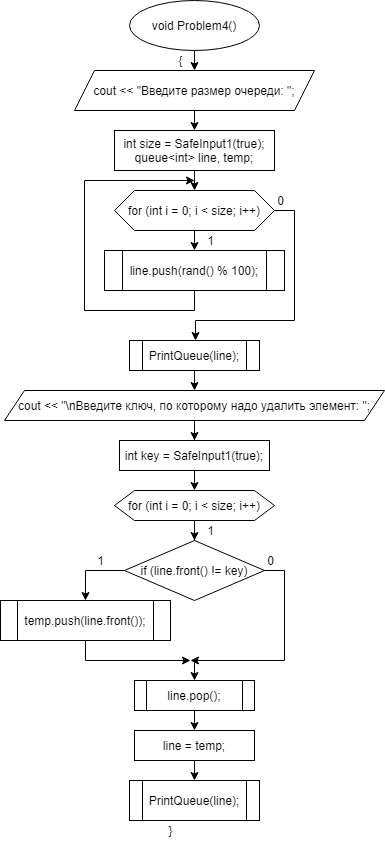
15



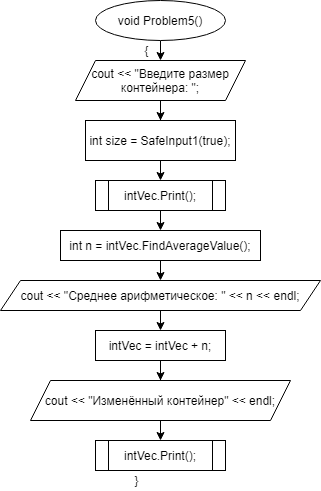
16



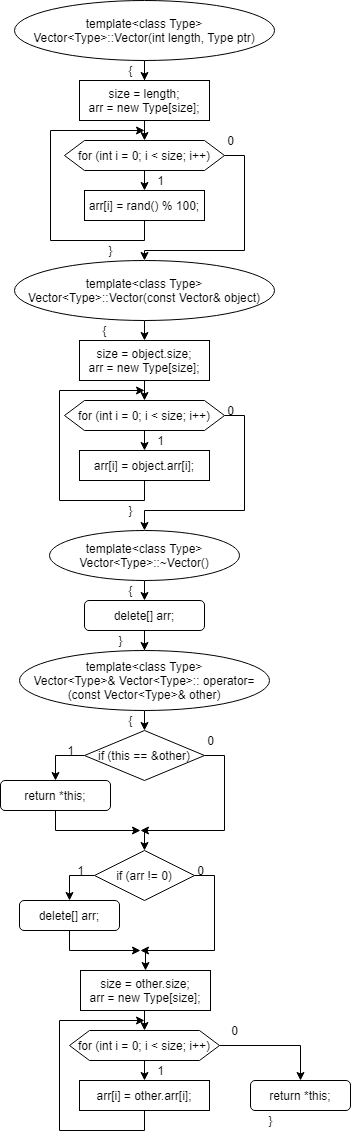
17



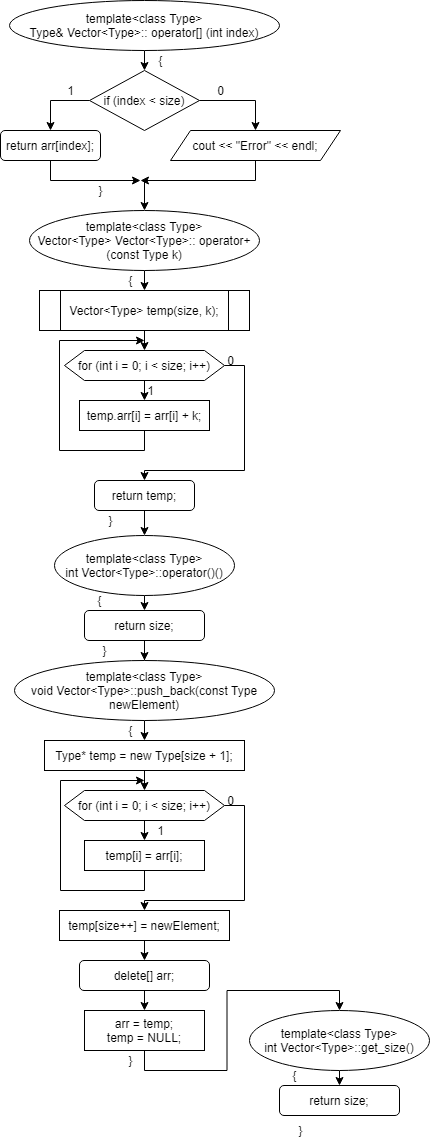
18



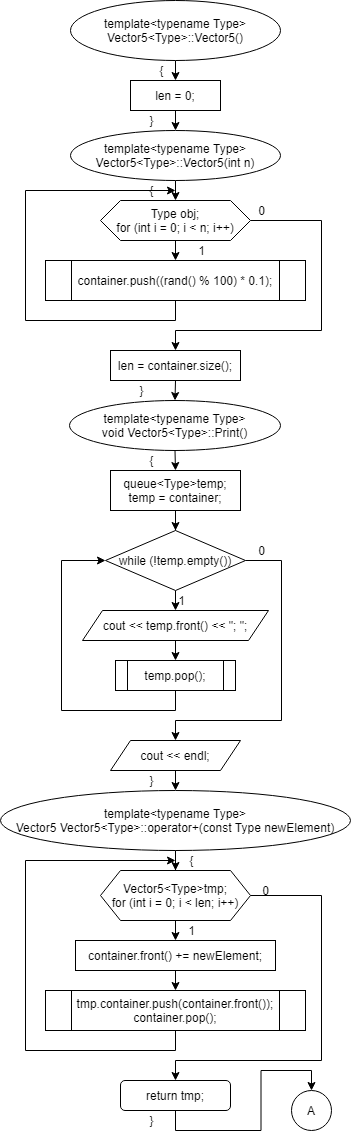
19



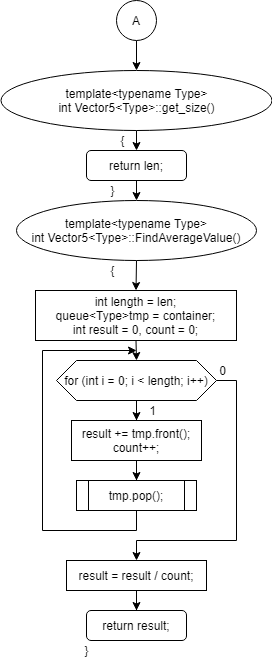
20



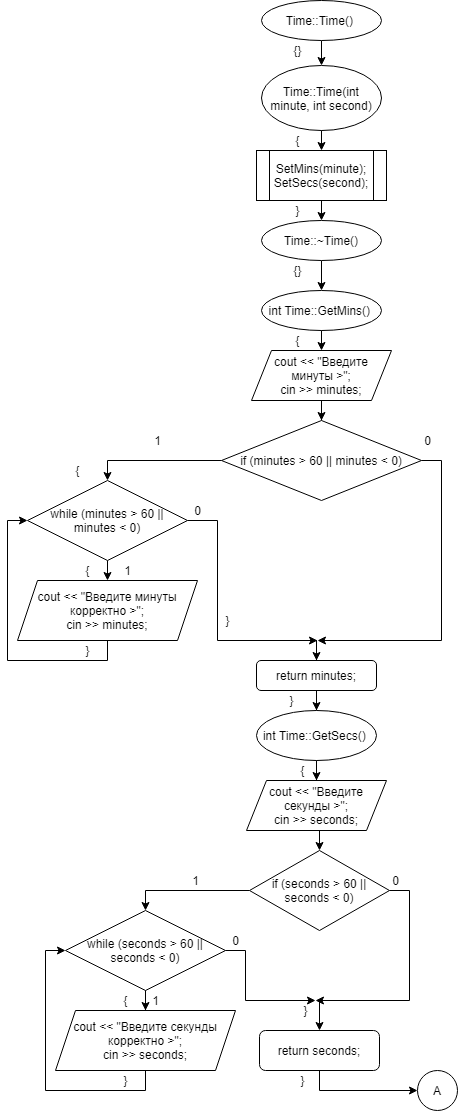
21



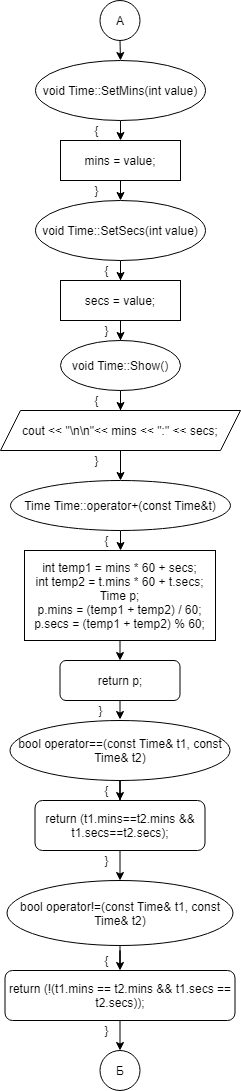
22



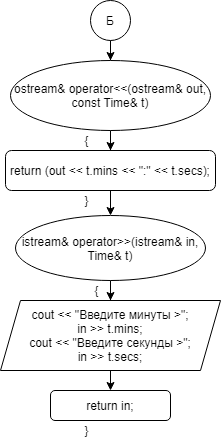
23



24



25



26

**Код**

#include"Time.h"

#include"Vector.h"

#include <deque>

#include <queue>

#include <ctime>

#include "Vector5.h"

int SafeInput1(bool strict);

template <typename T>

void PrintDeque(deque<T> dq);

void Problem1()

{

deque<int> dq;

cout << "Введите размер двунаправленной очереди: ";

int size = SafeInput1(true);

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

dq.push\_back(rand() % 100);

}

PrintDeque(dq);

cout << "Удаление последнего элемента" << endl;

dq.pop\_back();

PrintDeque(dq);

cout << "Удаление первого элемента" << endl;

dq.pop\_front();

PrintDeque(dq);

}

void Problem2()

{

Time obj;

int tmp = obj.GetMins();

obj.SetMins(tmp);

tmp = obj.GetSecs();

obj.SetSecs(tmp);

deque<Time>dq;

cout << "Введите размер двунаправленной очереди: ";

int size = SafeInput1(true);

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

dq.push\_back(obj);

}

PrintDeque(dq);

cout << "Удаление последнего элемента" << endl;

dq.pop\_back();

cout << "newSize=" << dq.size() << endl;

PrintDeque(dq);

}

Vector<int> AddingMaxValue(Vector<int>intVec)

{

int max = intVec[0];

for (int i = 0; i < intVec.get\_size(); i++) {

if (max < intVec[i]) {

max = intVec[i];

}

}

cout << "Max=" << max<< endl;

intVec.push\_back(max);

return intVec;

}

void Problem3()

{

cout << "Введите размер контейнера: ";

int size = SafeInput1(true);

Vector<int> intVec(size, 0);

27

cout << intVec;

intVec = AddingMaxValue(intVec);

cout << intVec;

}

void PrintQueue(queue<int>& line);

void Problem4()

{

cout << "Введите размер очереди: ";

int size = SafeInput1(true);

queue<int> line;

queue<int>temp;

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

line.push(rand() % 100);

}

PrintQueue(line);

cout << "\nВведите ключ, по которому надо удалить элемент: ";

int key = SafeInput1(true);

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

if (line.front() != key) {

temp.push(line.front());

}

line.pop();

}

line = temp;

PrintQueue(line);

}

void Problem5()

{

cout << "Введите размер контейнера: ";

int size = SafeInput1(true);

Vector5<int> intVec(size);

//Vector5<char> charVec(size);

//Vector5<double> doubleVec(size);

intVec.Print();

//charVec.Print();

//doubleVec.Print();

int n = intVec.FindAverageValue();

cout << "Среднее арифметическое: " << n << endl;

intVec = intVec + n;

cout << "Изменённый контейнер" << endl;

intVec.Print();

}

int main()

{

setlocale(LC\_ALL, "ru");

srand(time(0));

int vvod = 6;

while (vvod != 0) {

cout << "\nКакое задание?" << endl

<< "1) " << endl

<< "2) " << endl

<< "3) " << endl

<< "4) " << "\n"

<< "5)" << endl

<< "0) Выход из меню" << "\n\n" << ">";

cin >> vvod;

switch (vvod)

{

case 1:

{

Problem1();

}

break;

case 2:

28

{

Problem2();

}

break;

case 3:

{

Problem3();

}

break;

case 4:

{

Problem4();

}

break;

case 5:

{

Problem5();

}

break;

default:

if (vvod != 0)

cout << "ERROR!";

break;

}

}

return 0;

}

int SafeInput1(bool strict)

{

int result;

while (!(cin >> result) || (cin.peek() != '\n') || (strict && result <= 0)) {

cin.clear();

while (cin.get() != '\n');

cout << "Введите корректное число!" << endl;

}

cin.get();

return result;

}

template <typename T>

void PrintDeque(deque<T> dq)

{

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

cout << dq[i] << "; ";

}

cout << endl;

}

void PrintQueue(queue<int>& line)

{

queue<int>temp;

temp = line;

while (!temp.empty()) {

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

temp.pop();

}

cout << endl;

}

29

#pragma once

#include<iostream>

using namespace std;

class Time {

protected:

int mins, secs;

public:

Time();

Time(int minute, int second);

~Time();

int GetMins();

int GetSecs();

int getM() { return mins; };

int getS() { return secs; };

void SetMins(int value);

void SetSecs(int value);

void Show();

Time operator+(const Time& t);

friend bool operator==(const Time& t1, const Time& t2);

friend bool operator!=(const Time& t1, const Time& t2);

friend ostream& operator<<(ostream& out, const Time& t);

friend istream& operator>>(istream& in, Time& t);

};

30

#include "Time.h"

using namespace std;

int SafeInput(bool strict)

{

int result;

while (!(cin >> result) || (cin.peek() != '\n') || (strict && result <= 0) || (result > 60)) {

cin.clear();

while (cin.get() != '\n');

cout << "Введите корректное число!" << endl;

}

cin.get();

return result;

}

Time::Time() {

}

Time::Time(int minute, int second)

{

SetMins(minute);

SetSecs(second);

}

Time::~Time() {

}

int Time::GetMins()

{

int minutes;

cout << "Введите минуты >";

minutes = SafeInput(true);

return minutes;

}

int Time::GetSecs()

{

int seconds;

cout << "Введите секунды >";

seconds = SafeInput(true);

return seconds;

}

void Time::SetMins(int value)

{

mins = value;

}

void Time::SetSecs(int value)

{

secs = value;

}

void Time::Show()

{

cout << "\n\n" << mins << ":" << secs;

}

Time Time::operator+(const Time& t)

{

int temp1 = mins \* 60 + secs;

int temp2 = t.mins \* 60 + t.secs;

Time p;

p.mins = (temp1 + temp2) / 60;

p.secs = (temp1 + temp2) % 60;

return p;

}

bool operator==(const Time& t1, const Time& t2)

{

return (t1.mins == t2.mins && t1.secs == t2.secs);

}

bool operator!=(const Time& t1, const Time& t2)

{

31

return (!(t1.mins == t2.mins && t1.secs == t2.secs));

}

ostream& operator<<(ostream& out, const Time& t)

{

return (out << t.mins << ":" << t.secs);

}

istream& operator>>(istream& in, Time& t)

{

cout << "Введите минуты >";

in >> t.mins;

cout << "Введите секунды >";

in >> t.secs;

return in;

}

32

#pragma once

#include<iostream>

using namespace std;

template<class Type>

class Vector

{

protected:

Type\* arr;

int size;

public:

Vector(int length, Type ptr); //конструктор с параметрами

Vector(const Vector& object); //конструктор копирования

~Vector(); //очистка памяти

Vector& operator=(const Vector<Type>& other);

Type& operator[] (int index); //получение значения по индексу

Vector operator+(const Type k); //сложение элемента массива с константой

int operator()(); //получение размера

void push\_back(const Type newElement);

int get\_size();

friend ostream& operator<< (ostream& out, const Vector<Type>& p)

{

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

out << p.arr[i] << " ";

}

cout << endl;

return out;

}

friend istream& operator>> (istream& in, Vector<Type>& p)

{

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

in >> p.arr[i];

}

return in;

}

};

#include "Vector.cpp"

33

#ifndef \_VECTOR\_CPP\_

#define \_VECTOR\_CPP\_

#include "Vector.h"

#include <assert.h>

using namespace std;

template<class Type>

Vector<Type>::Vector(int length, Type ptr)

{

size = length;

arr = new Type[size];

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

arr[i] = rand() % 100;

}

}

template<class Type>

Vector<Type>::Vector(const Vector& object)

{

size = object.size;

arr = new Type[size];

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

arr[i] = object.arr[i];

}

}

template<class Type>

Vector<Type>::~Vector()

{

delete[] arr;

}

template<class Type>

Vector<Type>& Vector<Type>:: operator=(const Vector<Type>& other)

{

if (this == &other)

return \*this;

if (arr != 0)

delete[] arr;

size = other.size;

arr = new Type[size];

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

arr[i] = other.arr[i];

}

return \*this;

}

template<class Type>

Type& Vector<Type>:: operator[] (int index)

{

if (index < size)

return arr[index];

else

cout << "Error" << endl;

}

template<class Type>

Vector<Type> Vector<Type>:: operator+(const Type k)

{

Vector<Type> temp(size, k);

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

temp.arr[i] = arr[i] + k;

}

return temp;

}

template<class Type>

int Vector<Type>::operator()()

{

return size;

}

34

template<class Type>

void Vector<Type>::push\_back(const Type newElement)

{

Type\* temp = new Type[size + 1];

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

temp[i] = arr[i];

}

temp[size++] = newElement;

delete[] arr;

arr = temp;

temp = NULL;

}

template<class Type>

int Vector<Type>::get\_size()

{

return size;

}

#endif

35

#pragma once

#include<iostream>

#include <queue>

using namespace std;

template <typename Type>

class Vector5

{

queue<Type> container;

int len;

public:

Vector5() { len = 0; };

Vector5(int n)

{

Type obj;

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

obj = ((rand() % 100) \* 0.1);

container.push(obj);

}

len = container.size();

}

void Print()

{

queue<Type>temp;

temp = container;

while (!temp.empty()) {

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

temp.pop();

}

cout << endl;

}

Vector5 operator+(const Type newElement)

{

Vector5<Type>tmp;

//queue<int>temp, temp2;

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

container.front() += newElement;

tmp.container.push(container.front());

container.pop();

}

return tmp;

}

int get\_size()

{

return len;

}

int FindAverageValue()

{

int length = len;

queue<Type>tmp = container;

int result = 0, count = 0;

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

result += tmp.front();

count++;

tmp.pop();

}

result = result / count;

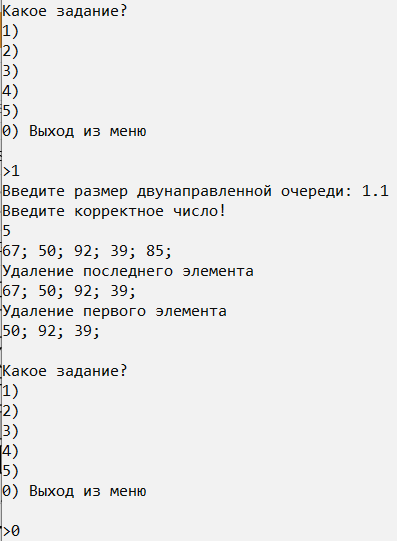
return result;

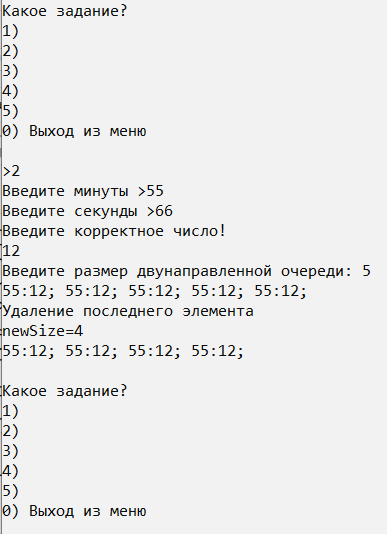
}

};

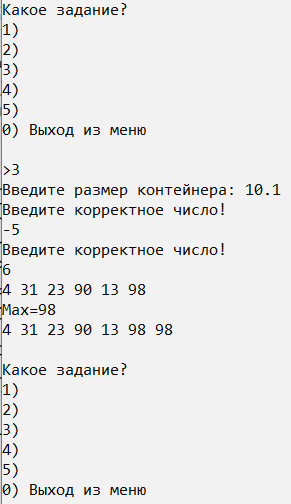
36

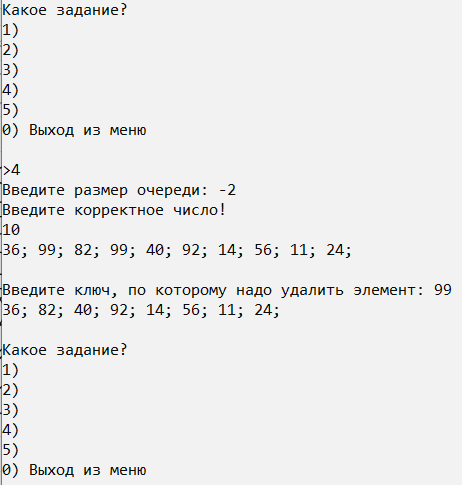
**Скриншоты**



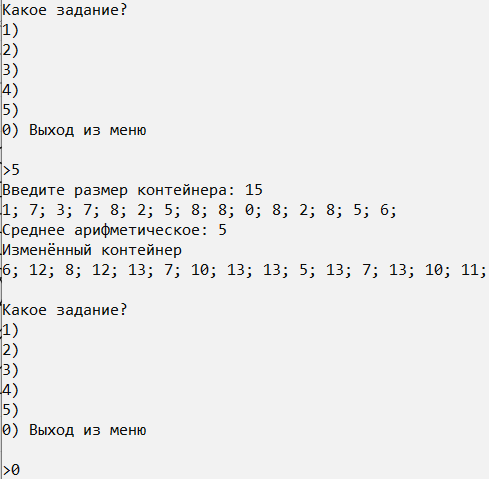


37





38



39

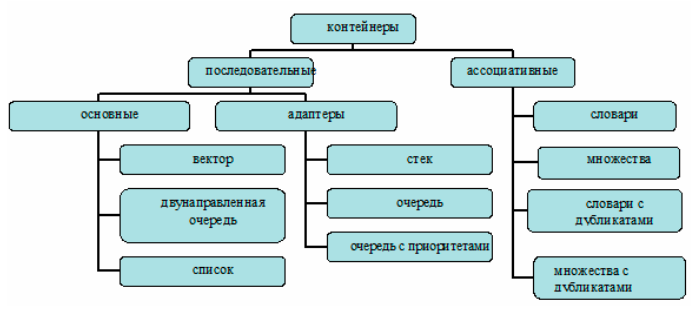
**Контрольные вопросы**



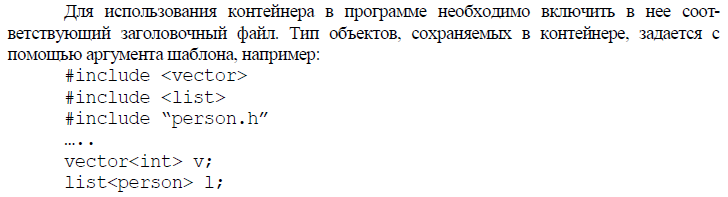
1. Набор контейнерных классов, набор обобщённых алгоритмов.



2. Последовательные, ассоциативные.



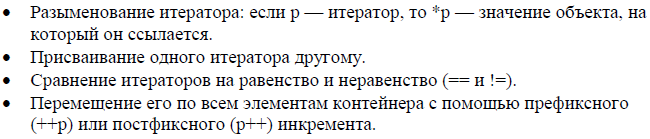


3. 



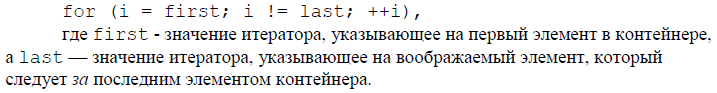
4. Итераторы – это обобщённые концепции указателей: они ссылаются на элементы контейнера.



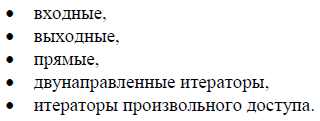
5. 

40

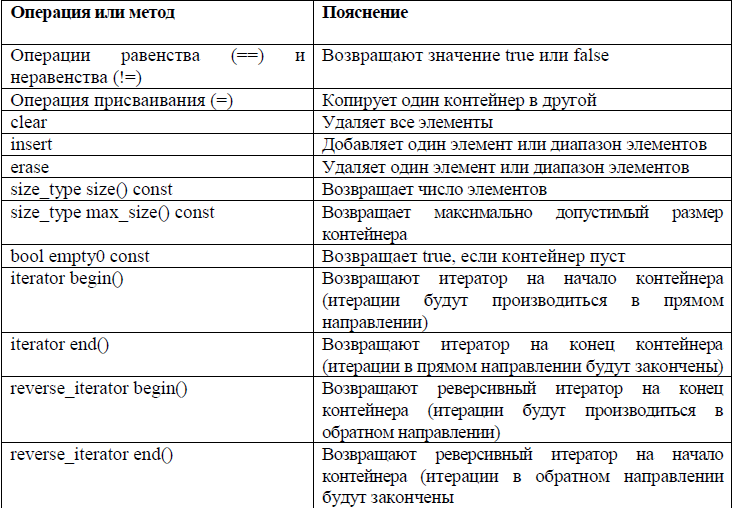


6. 



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.



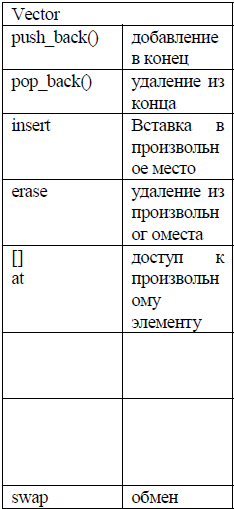
41

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. 

42



14. 



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



16.nameOfVector.pop\_back();



17. nameOfList.erase(1,4);



18. nameOfList.pop\_back();



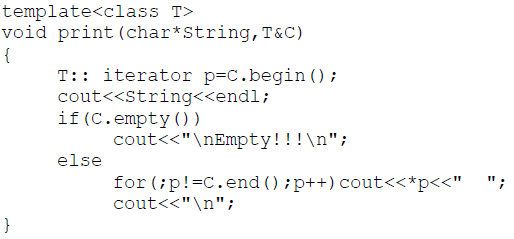
19. nameOfDeque.erase(1,4);

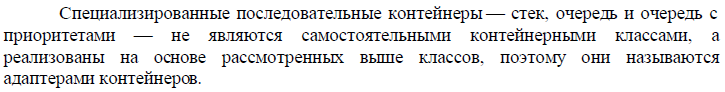


20. nameOfDeque.pop\_back();

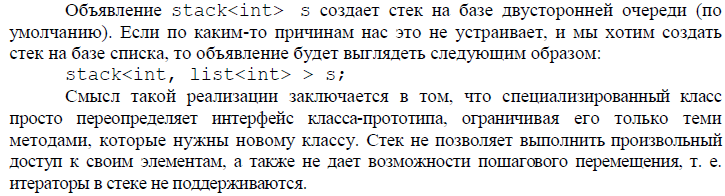


43

21. 

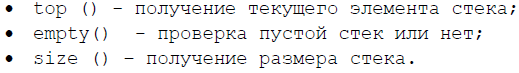
22. 



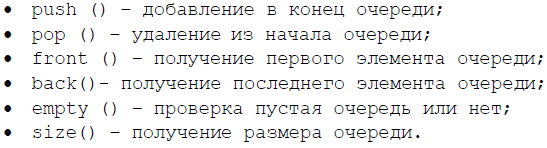
23. 



24.





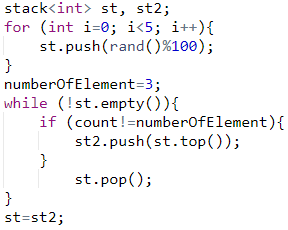
25. 

44



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

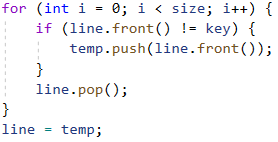


27. 

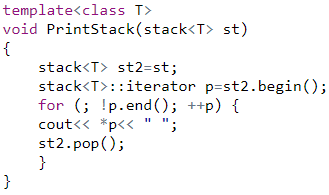


28.

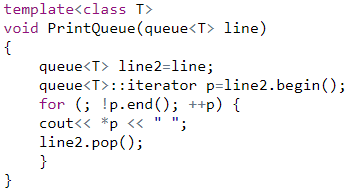






29. 



30. 

45