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 №18.6**

Дисциплина: «Информатика»

Тема: “ Объектно-ориентированное программирование.

АДФ. Контейнеры.”

Вариант 19

Выполнил:

Студент группы РИС-20-2Б

Пономарёв Артём Викторович

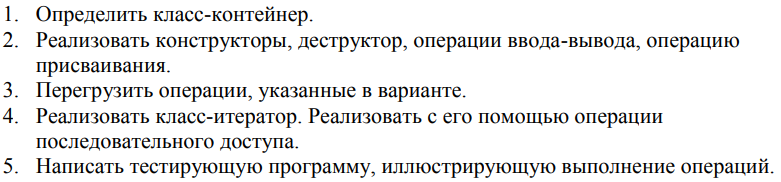
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**



2

**Анализ задачи**

**1.** Для решения задачи необходимо:

**1.1.** Организовать класс Vector с полями типа int – size, указателем head, а также переменные beg, end – объекты класса Iterator.

**1.2.** Организовать класс Iterator, дружественный классу Vector, с полем типа int\*head.

**1.3.** Организовать перегрузку оператора () для класса Vector.

**1.4.** Организовать перегрузку оператора [] для класса Vector.

**1.5.** Организовать перегрузку оператора -- для класса Vector.

**1.6.** Организовать перегрузку оператора -= для класса Vector.

**1.7.** Организовать перегрузку оператора -= для класса Vector.

**1.8.** Организовать перегрузку оператора <<, дружественной классу Vector.

**1.9.** Организовать перегрузку оператора -- для класса Iterator.

**2.0.** Организовать перегрузку оператора ++ для класса Iterator.

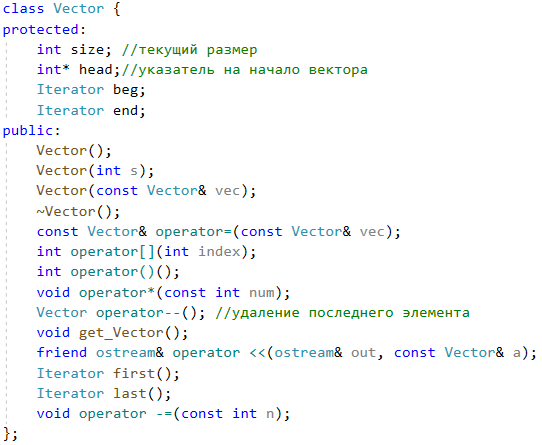
**2.1.** Организовать перегрузку оператора == для класса Iterator.

**2.2.** Организовать перегрузку оператора != для класса Iterator.

**2.3.** Организовать перегрузку оператора \* для класса Iterator.

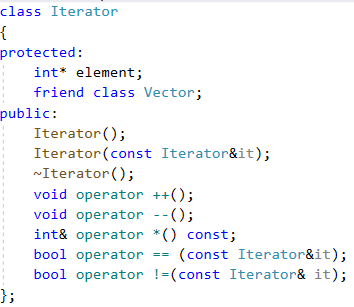
**2.** В ходе работы были использованы следующие типы данных:

**2.1.** Класс Vector с полями типа int – size, указателем head, а также переменные beg, end – объекты класса Iterator.



3

**2.2.** Класс Iterator, дружественный классу Vector, с полем типа int\*head.



**2.3.** Объект класса Vector first – массив типа Vector.

Vector first(size);

**2.4.** Переменная типа int: size, которая отвечает за размер массива, переменная типа int n, которая используется как временная для умножения всех чисел массива на константу n или осуществления сдвига итератора на n.

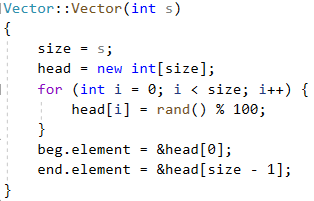
int size;

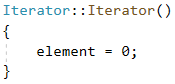
first\* n;

first -= n;

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

**3.1.** Данные инициализируются в конструкторе с параметром в классе Vector, Iterator.

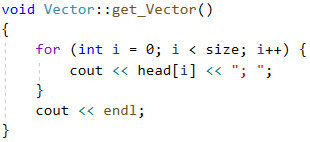




4

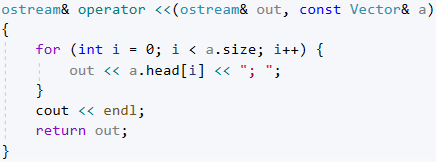
**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Для вывода массива класса Vector на консоль используется метод get\_Vector().

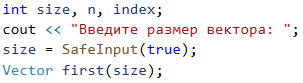


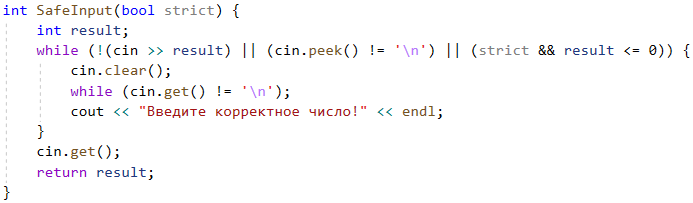
**4.2.** Для вывода массива класса Vector на консоль также используется перегрузка оператора <<.

cout << first;



**4.3.** Для ввода массива класса Vector необходимо сначала ввести размер size, а затем в конструкторе сформировать динамический массив.



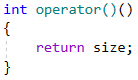


В функции SafeInput() выполняются проверки, исключающие ввод символов, дробных и отрицательных чисел.

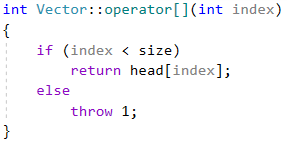
5

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

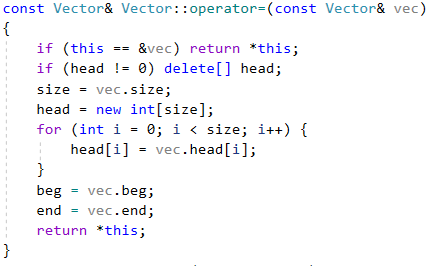
**5.1.** В перегрузке оператора () класса Vector производится вывод размерности объекта класса Vector.



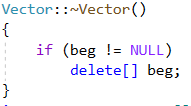
**5.2.** В перегрузке оператора [] класса Vector производится вызов элемента множества по индексу объекта класса Vector.



**5.3.** В перегрузке оператора = класса Vector производится вызов элемента множества по индексу объекта класса Vector.

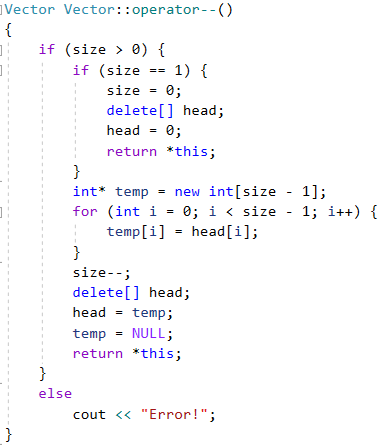


**5.4.** В деструкторе выполнятеся очистка памяти, если указатель не указывает на NULL.

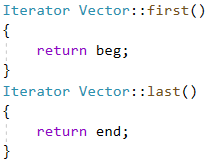


6

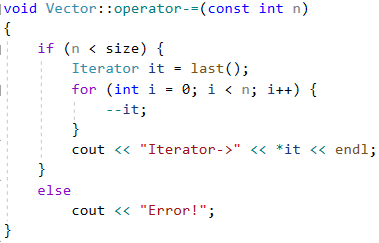
**5.5.** В перегрузке оператора -- класса Vector производится создание динамического массива с размерностью на 1 меньше и производится заполнение до size-1 элемента массива beg[]. Затем освобождается память, на которую указывает \*beg и передаётся память, на которую указывает \*temp.



**5.6.** Методы first() и last() возвращают значения первого и последнего элемента массива соответственно.

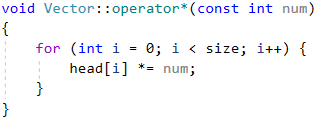


**5.7.** В перегрузке оператора -= выполняется перемещение через итератор: в переменную it класса Iterator присваивается указатель на последний элемент массива и в цикле for происходит сдвиг декрементом в количестве n.



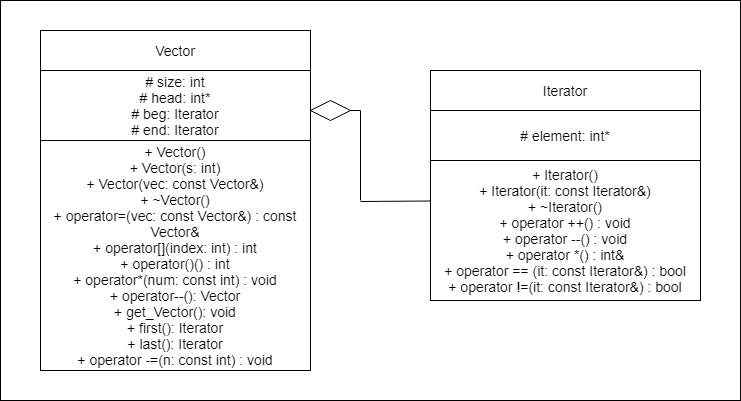
7

**5.8.** В перегрузке оператора \* выполняется умножение на введенное число.



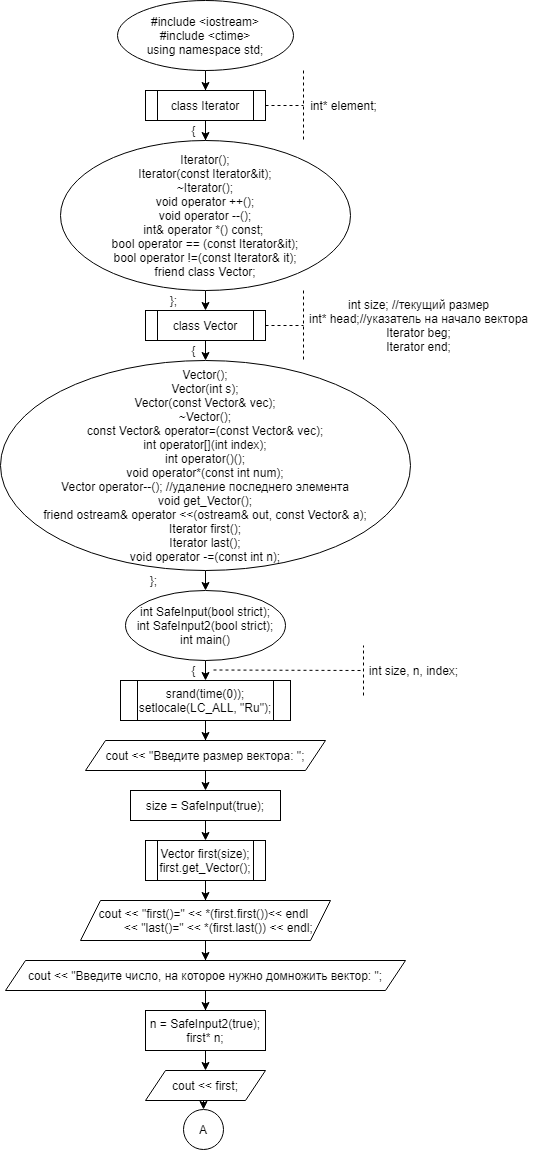
8

**Диаграммы**

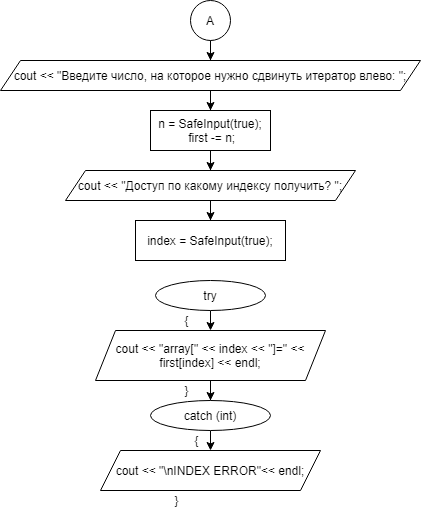


9

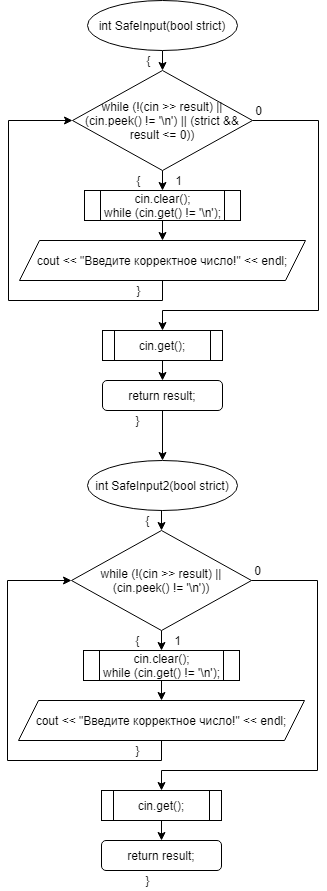
**Блок-схема**



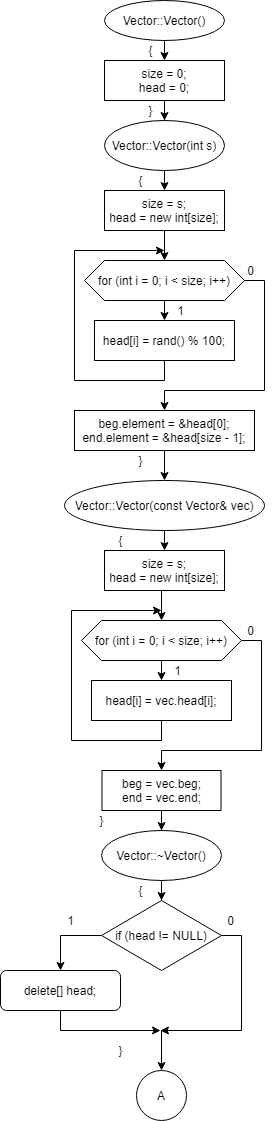
10



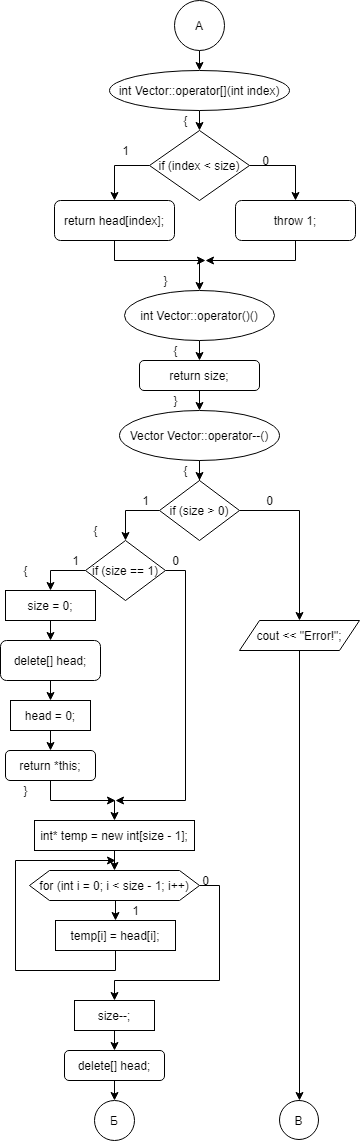
11



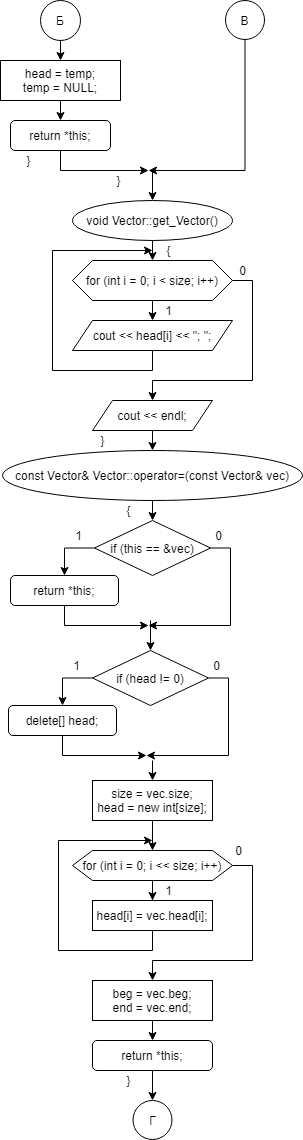
12



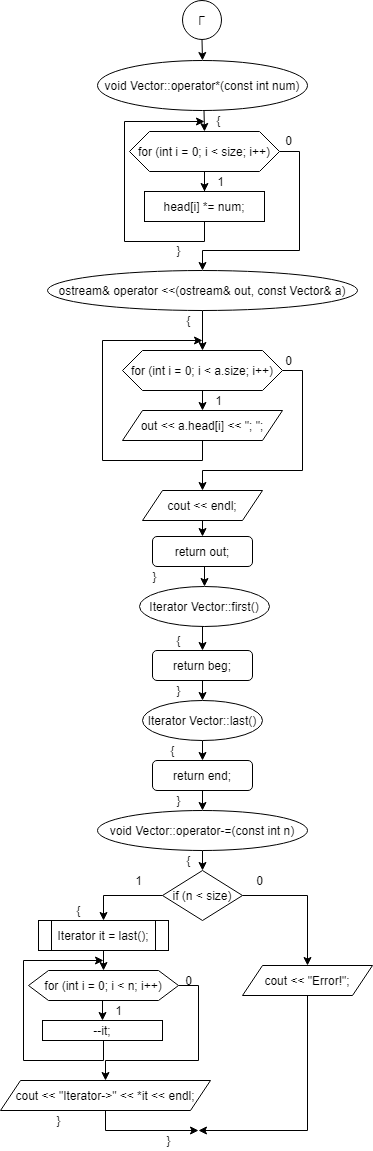
13



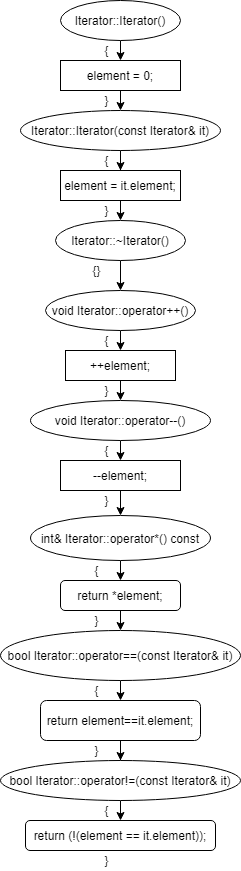
14



15



16



17

**Код**

#include "Vector.h"

int SafeInput(bool strict);

int SafeInput2(bool strict);

int main()

{

srand(time(0));

setlocale(LC\_ALL, "Ru");

int size, n, index;

cout << "Введите размер вектора: ";

size = SafeInput(true);

Vector first(size);

first.get\_Vector();

cout << "first()=" << \*(first.first())<< endl

<< "last()=" << \*(first.last()) << endl;

cout << "Введите число, на которое нужно домножить вектор: ";

n = SafeInput2(true);

first\* n;

cout << first;

cout << "Введите число, на которое нужно сдвинуть итератор влево: ";

n = SafeInput(true);

first -= n;

cout << "Доступ по какому индексу получить? ";

index = SafeInput(true);

try {

cout << "array[" << index << "]=" << first[index] << endl;

}

catch (int) { cout << "\nINDEX ERROR"<< endl; }

}

int SafeInput(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;

}

int SafeInput2(bool strict) {

int result;

while (!(cin >> result) || (cin.peek() != '\n')) {

cin.clear();

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

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

}

cin.get();

return result;

}

18

#pragma once

#include <iostream>

#include <ctime>

#include "Iterator.h"

using namespace std;

class Vector {

protected:

int size; //текущий размер

int\* head;//указатель на начало вектора

Iterator beg;

Iterator end;

public:

Vector();

Vector(int s);

Vector(const Vector& vec);

~Vector();

const Vector& operator=(const Vector& vec);

int operator[](int index);

int operator()();

void operator\*(const int num);

Vector operator--(); //удаление последнего элемента

void get\_Vector();

friend ostream& operator <<(ostream& out, const Vector& a);

Iterator first();

Iterator last();

void operator -=(const int n);

};

19

#include "Vector.h"

Vector::Vector()

{

size = 0;

head = 0;

}

Vector::Vector(int s)

{

size = s;

head = new int[size];

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

head[i] = rand() % 100;

}

beg.element = &head[0];

end.element = &head[size - 1];

}

Vector::Vector(const Vector& vec)

{

size = vec.size;

head = new int[size];

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

head[i] = vec.head[i];

}

beg = vec.beg;

end = vec.end;

}

Vector::~Vector()

{

if (head != NULL)

delete[] head;

}

int Vector::operator[](int index)

{

if (index < size)

return head[index];

else

throw 1;

}

int Vector::operator()()

{

return size;

}

Vector Vector::operator--()

{

if (size > 0) {

if (size == 1) {

size = 0;

delete[] head;

head = 0;

return \*this;

}

int\* temp = new int[size - 1];

for (int i = 0; i < size - 1; i++) {

temp[i] = head[i];

}

size--;

delete[] head;

head = temp;

temp = NULL;

return \*this;

}

else

cout << "Error!";

}

20

void Vector::get\_Vector()

{

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

cout << head[i] << "; ";

}

cout << endl;

}

const Vector& Vector::operator=(const Vector& vec)

{

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

if (head != 0) delete[] head;

size = vec.size;

head = new int[size];

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

head[i] = vec.head[i];

}

beg = vec.beg;

end = vec.end;

return \*this;

}

void Vector::operator\*(const int num)

{

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

head[i] \*= num;

}

}

ostream& operator <<(ostream& out, const Vector& a)

{

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

out << a.head[i] << "; ";

}

cout << endl;

return out;

}

Iterator Vector::first()

{

return beg;

}

Iterator Vector::last()

{

return end;

}

void Vector::operator-=(const int n)

{

if (n < size) {

Iterator it = last();

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

--it;

}

cout << "Iterator->" << \*it << endl;

}

else

cout << "Error!";

}

21

#pragma once

class Iterator

{

protected:

int\* element;

friend class Vector;

public:

Iterator();

Iterator(const Iterator&it);

~Iterator();

void operator ++();

void operator --();

int& operator \*() const;

bool operator == (const Iterator&it);

bool operator !=(const Iterator& it);

};

22

#include "Iterator.h"

Iterator::Iterator()

{

element = 0;

}

Iterator::Iterator(const Iterator& it)

{

element = it.element;

}

Iterator::~Iterator()

{

}

void Iterator::operator++()

{

++element;

}

void Iterator::operator--()

{

--element;

}

int& Iterator::operator\*() const

{

return \*element;

}

bool Iterator::operator==(const Iterator& it)

{

return element==it.element;

}

bool Iterator::operator!=(const Iterator& it)

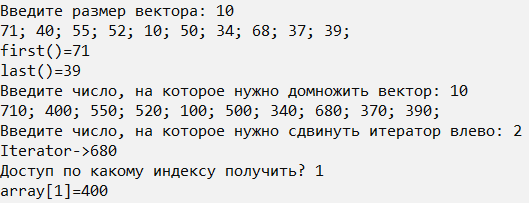
{

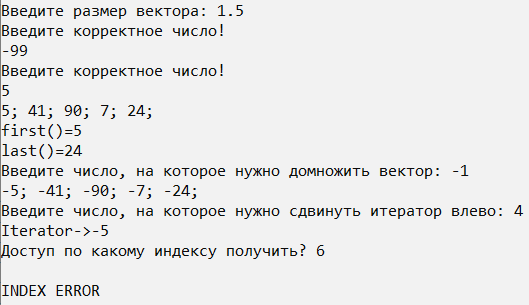
return (!(element == it.element));

}

23

**Скриншоты**





24

**Контрольные вопросы**



1. 





2. 

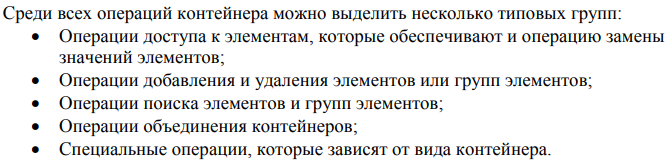


3. 

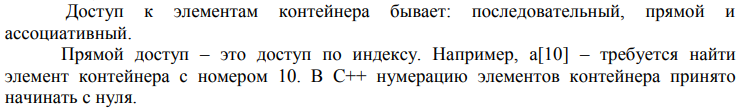


4. 

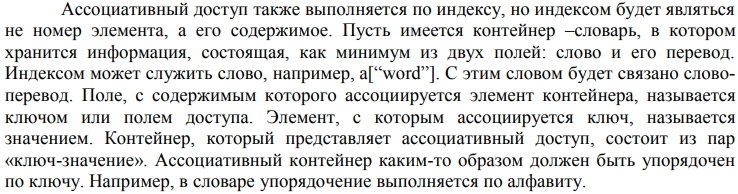


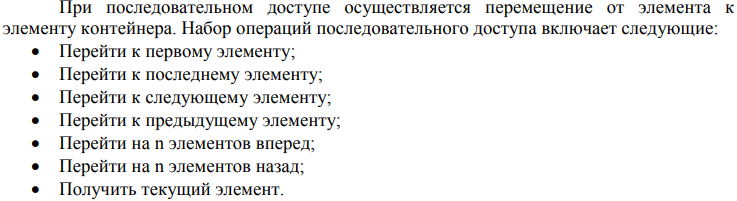
5. 



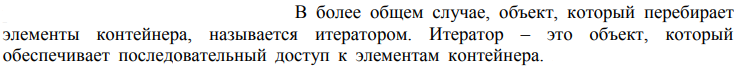
6. 

25



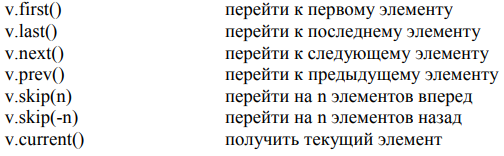




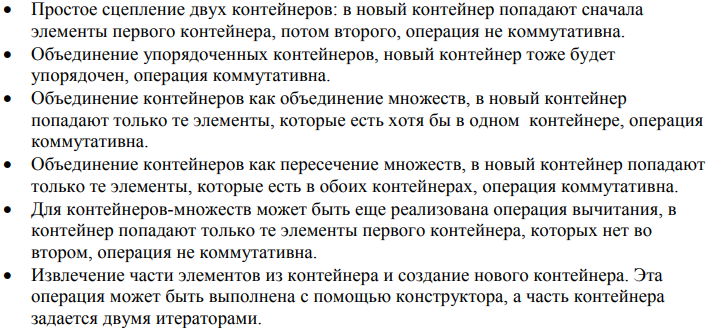
7. 



8. Итератор может быть реализован как часть класса-контейнера в виде набора методов:





9. 

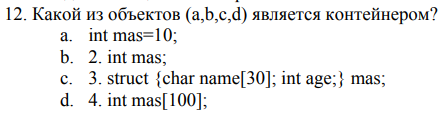
26



10. Ассоциативный доступ.



11. Стек.



12. d.





13. d.



14. Прямой доступ.



15. Последовательный доступ.

27