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.

АТД. Контейнеры.”

Вариант 4

Выполнил:

Студент группы РИС-20-1Б

Еске Вячеслав Сергеевич

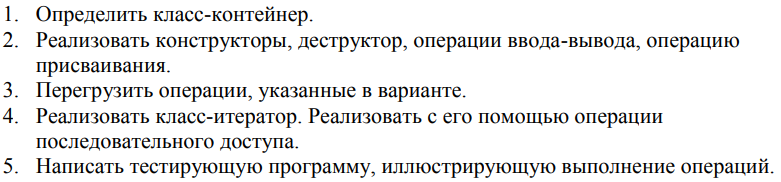
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**



Класс- контейнер ВЕКТОР с элементами типа int.  
 Реализовать операции:  
 [] – доступа по индексу;

() – определение размера вектора;   
\* число – умножает все элементы вектора на число;

- n – переход влево к элементу с номером n ( с помощью класса-итератора).

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.

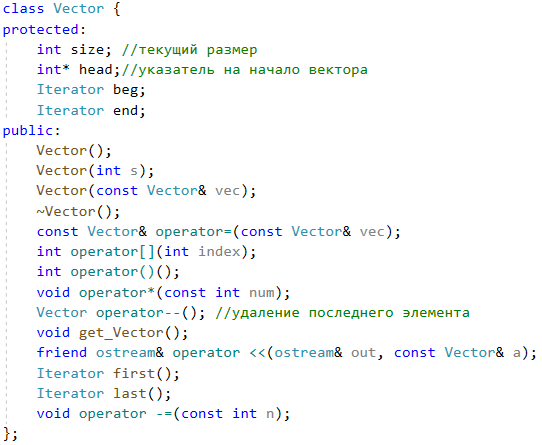
**2.1.** Организовать перегрузку оператора == для класса Iterator.

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

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

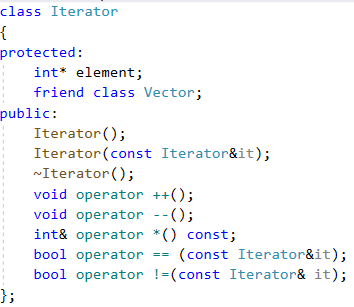
**2.** В ходе работы были использованы следующие типы данных:

**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.

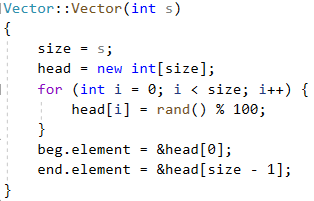
int size;

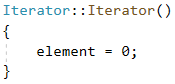
first\* n;

first -= n;

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

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

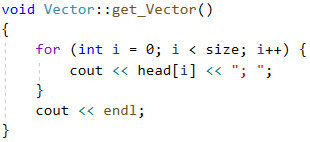




4

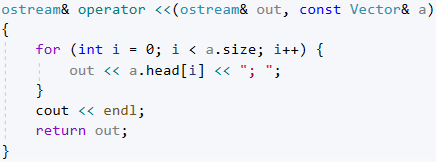
**4.** Для операций ввода и вывода использовались следующие операторы и функции:

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

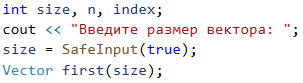


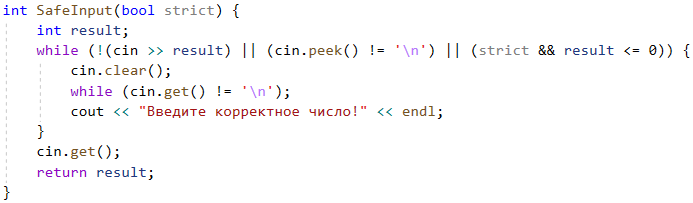
**4.2.** Для вывода массива класса Vector на консоль также используется перегрузка оператора <<.

cout << first;



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



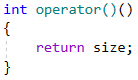


В функции SafeInput() выполняются проверки, исключающие ввод символов, дробных и отрицательных чисел.

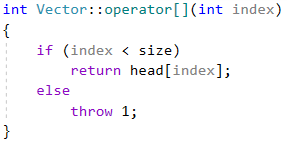
5

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

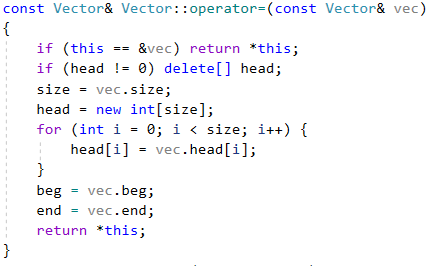
**5.1.** В перегрузке оператора () класса Vector производится вывод размерности объекта класса Vector.



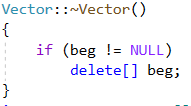
**5.2.** В перегрузке оператора [] класса Vector производится вызов элемента множества по индексу объекта класса Vector.



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

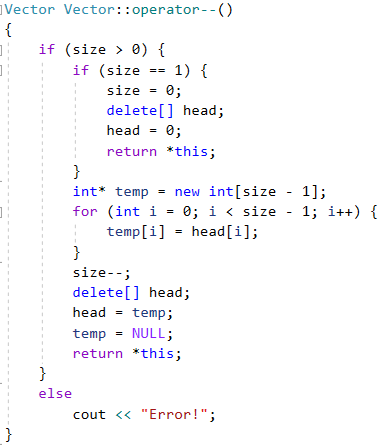


**5.4.** В деструкторе выполнятеся очистка памяти, если указатель не указывает на NULL.

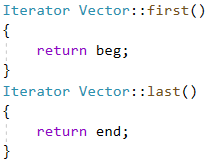


6

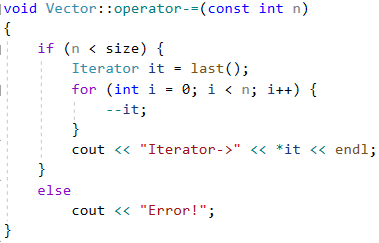
**5.5.** В перегрузке оператора -- класса Vector производится создание динамического массива с размерностью на 1 меньше и производится заполнение до size-1 элемента массива beg[]. Затем освобождается память, на которую указывает \*beg и передаётся память, на которую указывает \*temp.



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

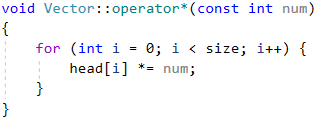


**5.7.** В перегрузке оператора -= выполняется перемещение через итератор: в переменную it класса Iterator присваивается указатель на последний элемент массива и в цикле for происходит сдвиг декрементом в количестве n.



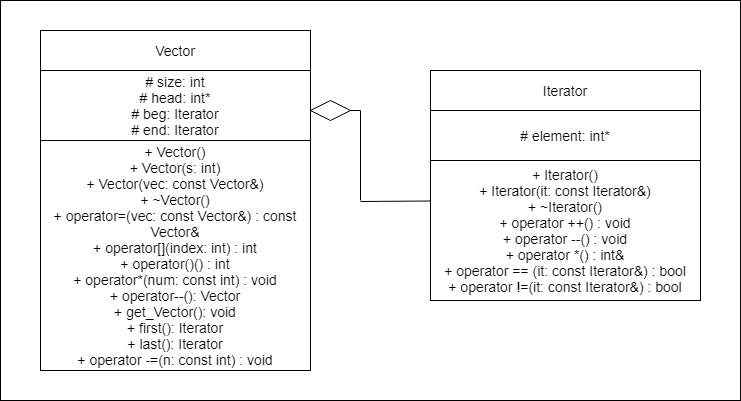
7

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



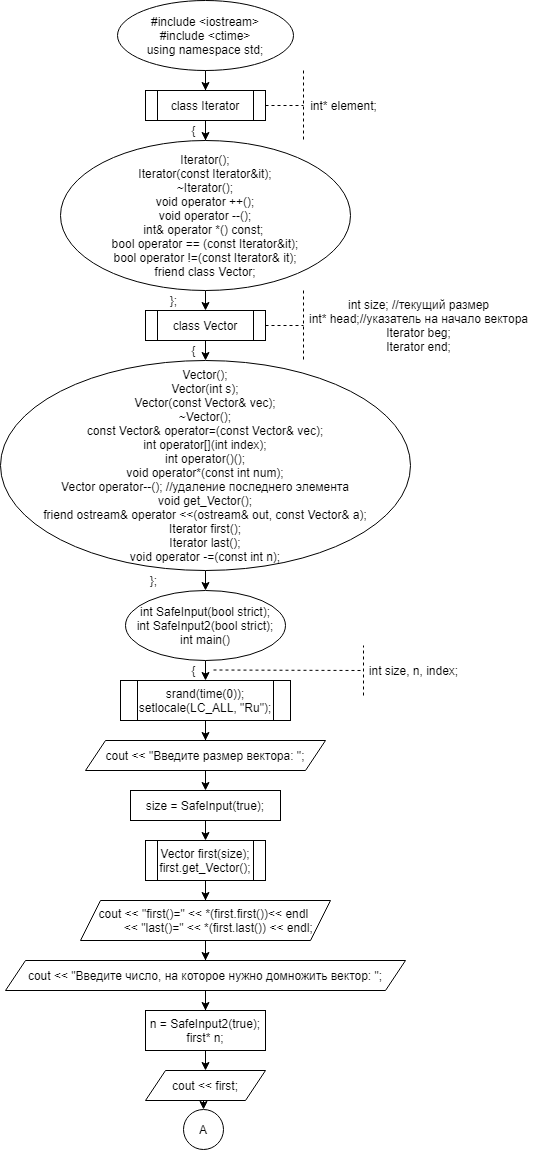
8

**Диаграммы**

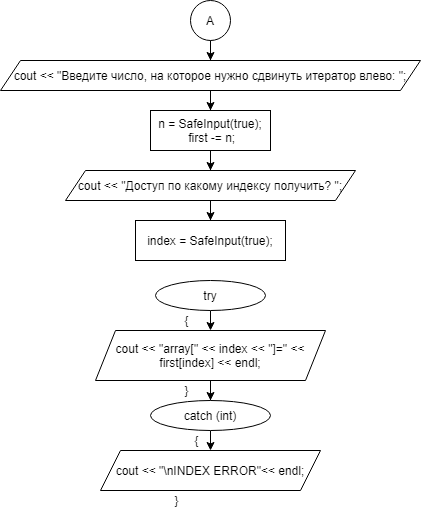


9

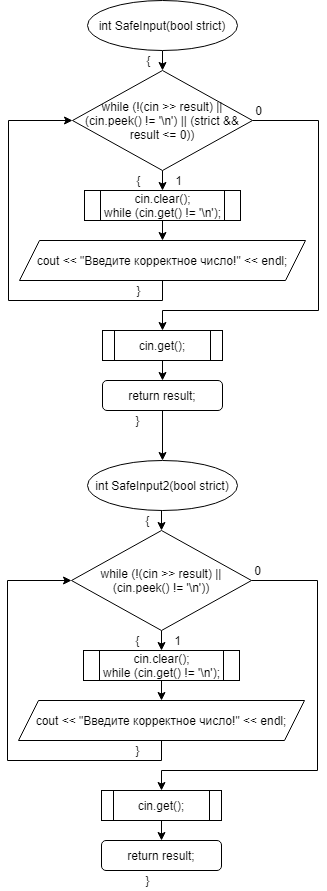
**Блок-схема**



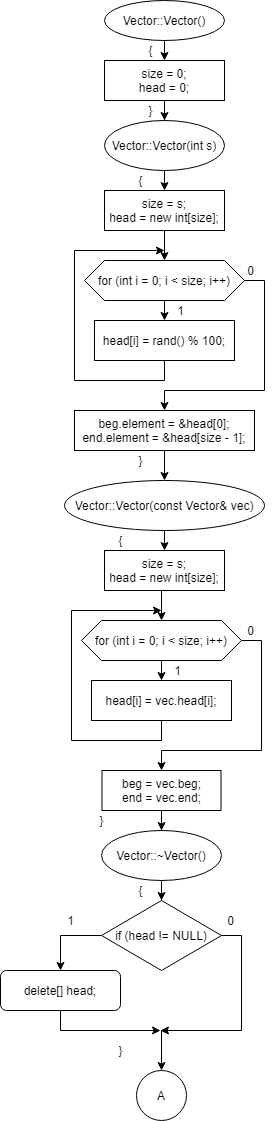
10



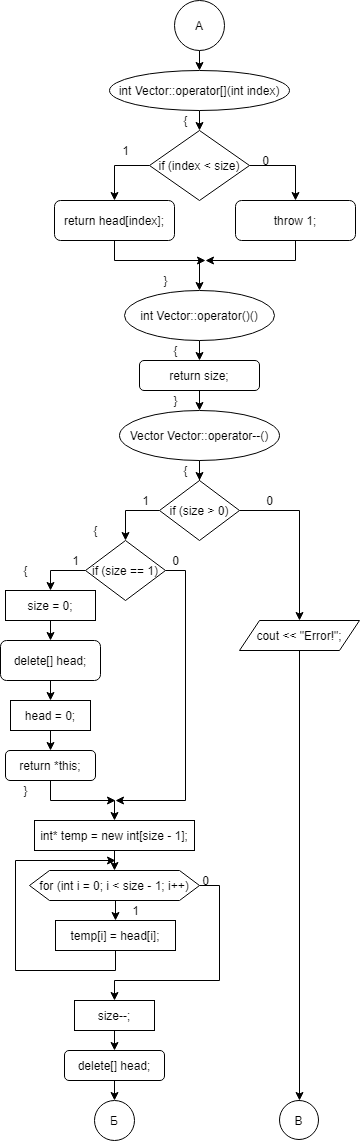
11



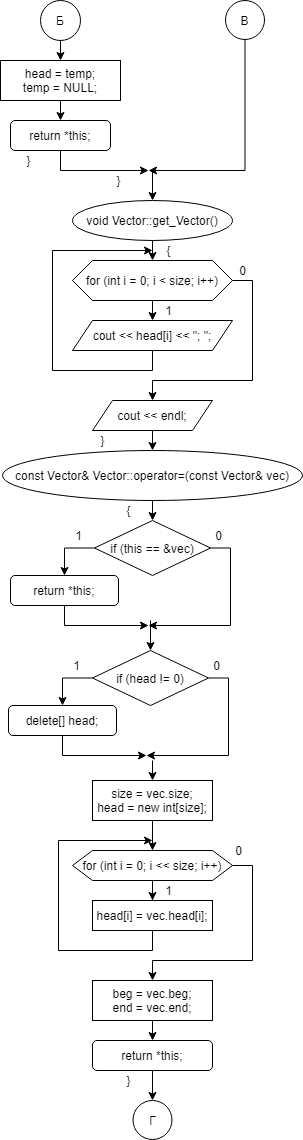
12



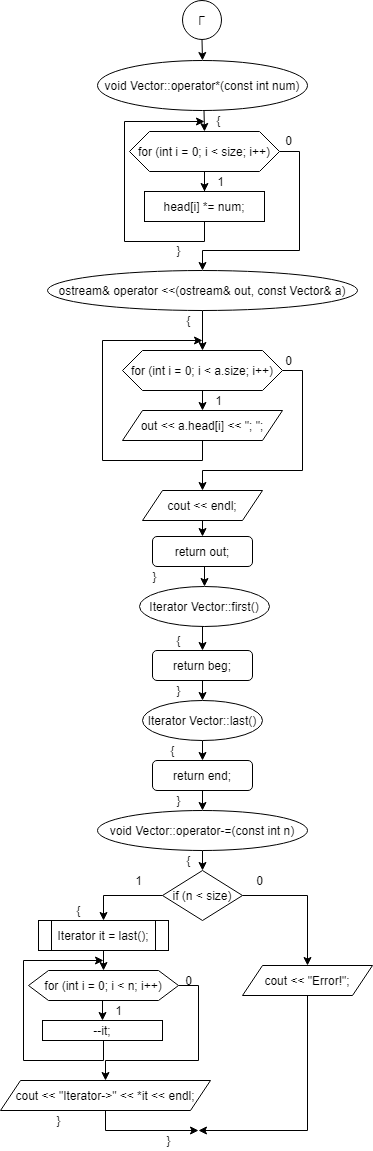
13



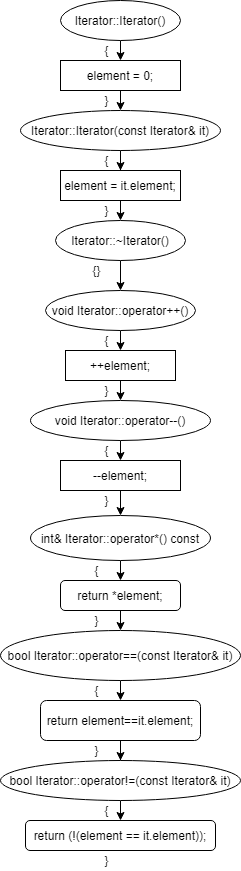
14



15



16



17

**Код**

#include <iostream>

#include "Vector.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

Vector a(5);//создали вектор из 5 элементов, заполненный нулями

cout<<"Вектор а: "<<a<<"\n";//вывели значения элементов вектора

cout << "Введите значения вектора а:" << endl;

cin>>a;//ввели с клавиатуры значения элементов вектора

cout<<"Вектор а: "<<a<<"\n";//вывели значения элементов вектора

cout << "Изменяем 2 элемент вектора а на 100" << endl;

a[2]=100;//используя операцию [] присвоили новое значение элементу

cout<< "Вектор а: " << a<<"\n";//вывели значения элементов вектора

Vector b(10);//создали вектор b из 10 элементов, заполненный нулями

cout<<"Вектор b: "<<b<<"\n";//вывели значения элементов вектора

b=a;//присвоили вектору b значения вектора a

cout << "Присвоили вектору b значения вектора a" << endl;

cout<< "Вектор b: " << b <<"\n";//вывели значения элементов вектора

Vector c(10); //создали вектор c из 10 элементов, заполненный нулями

c=b\*100;//Увеличили значения вектора b в 100 и присвоили вектору c

cout << "Увеличили значения вектора b в 100 и присвоили вектору c" << endl;

cout<<"Вектор с: "<<c<<"\n";//вывели значения элементов вектора c

cout << "\nДлина a=" << a() << endl;//вывели длину вектора a

cout << "Разыменовываем значение, которое возвращает a.first() и выводим его"<<endl;

cout << \*(a.first()) << endl;

//переменную типа Iterator устанавливаем на первый элемент вектора a с помощью метода first

Iterator i = a.first();

i++;

i++;

i++;

i++;

cout << "Разыменовываем итератор и выводим его значение" << endl;

cout << \*i << endl;

cout << "Операция перехода на 3 элемента влево" << endl;

i = i - 3;

cout << \*i << endl;

//выводим значения элеменов вектора с помощью итератора

cout << "Выводим значения элеменов вектора с помощью итератора: ";

for (i = a.first(); i != a.last(); i++)

cout << \*i<<" ";

}

#pragma once

#include <iostream>

using namespace std;

class Iterator {

friend class Vector;//дружественный класс

public:

Iterator() { elem = 0; }//конструктор без параметров

Iterator(const Iterator&it){elem=it.elem;}//конструктор копирования

//перегруженные операции сравнения

bool operator==(const Iterator& it) { return elem == it.elem; };

bool operator!=(const Iterator& it) { return elem != it.elem; };

//перегруженная операция

Iterator operator -(const int n) { elem -= n; return \*this; };

//перегруженная операция инкремент

Iterator operator++(int)

{

elem++;

return \*this;

};

Iterator operator--(int)

{

elem--;

return \*this;

};

//перегруженная операция разыменования

int& operator \*() const{ return\*elem;}

private:

int\* elem;//указатель на элемент типа int

};

class Vector {

public:

//конструктор с параметрами: выделяет память под s элементов и заполняет их значением k

Vector(int s, int k = 0);

//конструктор с параметрами

Vector(const Vector& a);

//деструктор

~Vector();

//оператор присваивания

Vector& operator=(const Vector& a);

//операция доступа по индексу

int& operator[](int index);

//операция для добавление константы

Vector operator\*(const int k);

//операция, возвращающая длину вектора

int operator()();

//перегруженные операции ввода-вывода

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

friend istream& operator>>(istream& in, Vector& a);

Iterator first() { return beg; }//возвращает указатель на первый элемент

Iterator last(){return end;}//возвращает указатель на элемент следующий за последним

private:

int size;//размер вектора

int\*data;//укзатель на динамический массив значений вектора

Iterator beg;//указатель на первый элемент вектора

Iterator end;//указатель на элемент следующий за последним

};

#include "Vector.h"

#include <iostream>

using namespace std;

//конструктор с параметрами

Vector::Vector(int s,int k)

{

size = s; data = new int[size];

for (int i = 0; i < size; i++) data[i] = k;

beg.elem = &data[0];

end.elem = &data[size];

}

//конструктор копирования

Vector::Vector(const Vector&a)

{

size = a.size;

data = new int[size];

for (int i = 0; i < size; i++) data[i] = a.data[i];

beg = a.beg;

end = a.end;

}

//деструктор

Vector::~Vector()

{

delete[]data;

data = 0;

}

//операция присваивания

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

{

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

size = a.size;

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

data = new int[size];

for (int i = 0; i < size; i++) data[i] = a.data[i];

beg = a.beg;

end = a.end;

return \*this;

}

//операция доступа по индексу

int&Vector::operator[](int index)

{

if (index < size) return data[index];

else cout << "\nError! Index>size";

}

//операция для добавления константы

Vector Vector::operator\*(const int k)//\*k

{

Vector temp(size);

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

temp.data[i] += data[i] \* k;

return temp;

}

//операция для получения длины вектора

int Vector::operator ()()

{

return size;

}

//операции для ввода-выода

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

{

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

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

return out;

}

istream& operator>>(istream& in, Vector& a)

{

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

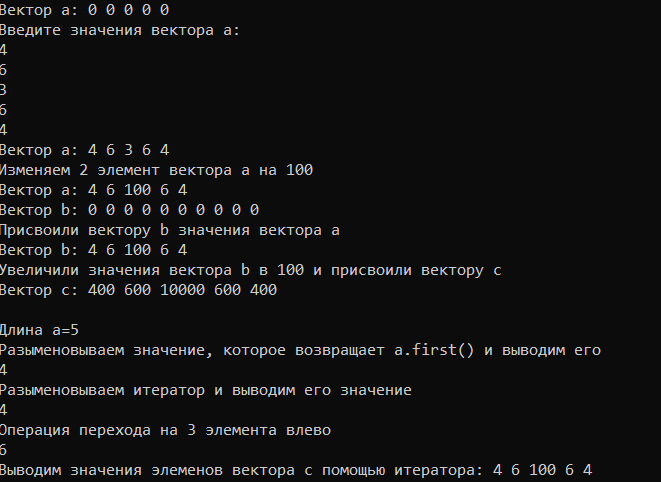
in >> a.data[i];

return in;

}

23

**Скриншоты**



24

**Контрольные вопросы**



1. 





2. 

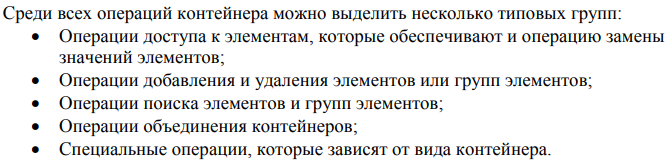


3. 

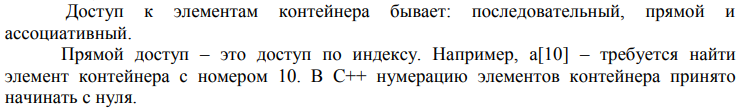


4. 

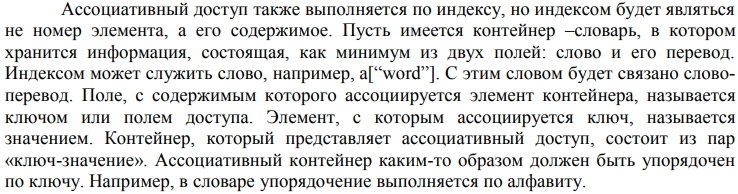


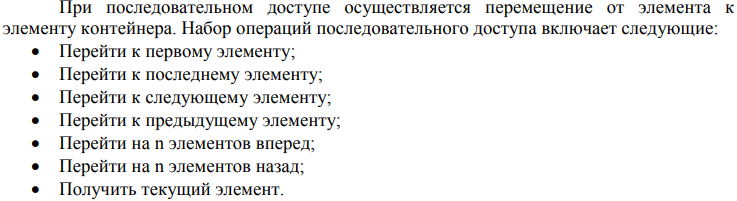
5. 



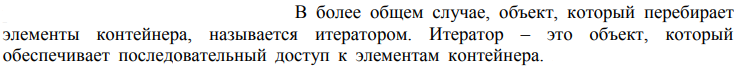
6. 

25



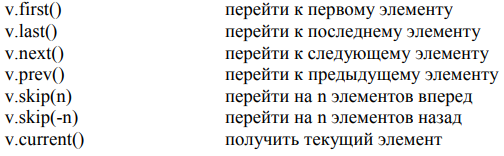




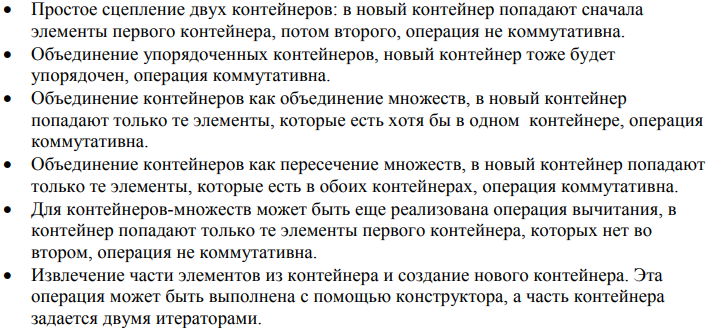
7. 



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





9. 

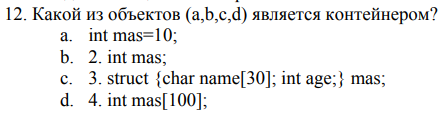
26



10. Ассоциативный доступ.



11. Стек.



12. d.





13. d.



14. Прямой доступ.



15. Последовательный доступ.

27