Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Информатика»

Тема: АТД. Контейнеры.

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Протасов Н.С.

Проверил

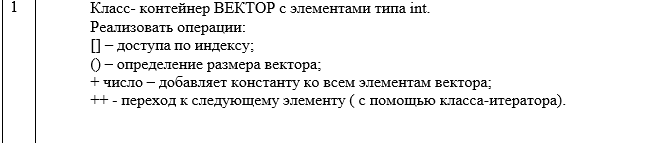
Доцент кафедры ИТАС

Полякова О.А.

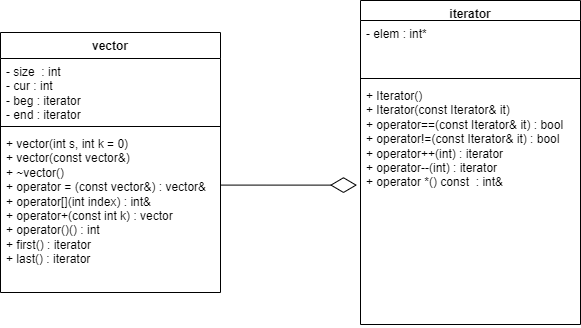
Г. Пермь-2023

**Постановка задачи (общая и конкретного варианта).**

1. Определитькласс-контейнер.
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операциюприсваивания.
3. Перегрузитьоперации,указанныевварианте.
4. Реализовать класс-итератор. Реализовать с его помощью операциипоследовательногодоступа.
5. Написатьтестирующуюпрограмму,иллюстрирующуювыполнениеопераций.



**Описание класса-контейнера.**

****

**Vector.h**

#pragmaonce

#include<iostream>

usingnamespacestd;

classIterator

{

friendclassVector;

public:

Iterator() { elem = 0; }

Iterator(constIterator&it) { elem = it.elem; }

booloperator==(constIterator&it) { returnelem == it.elem; }

booloperator!=(constIterator&it) { returnelem != it.elem; }

voidoperator++() { ++elem; }

voidoperator--() { --elem; }

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

private:

int\* elem;

};

classVector

{

public:

Vector(ints, intk = 0);

Vector(constVector&a);

~Vector();

Iteratorfirst() { return beg; }

Iteratorlast() { return end; }

Vector&operator=(constVector&a);

int&operator[](intindex);

Vectoroperator+(constintk);

intoperator()();

friendostream&operator<<(ostream&out, Vector&a);

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

private:

int size;

int\* data;

Iteratorbeg;

Iteratorend;

};

**Vector.cpp**

#include"Vector.h"

Vector::Vector(ints, intk)

{

size = s;

data = newint[size];

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

{

data[i] = k;

}

beg.elem = &data[0];

end.elem = &data[size];

}

Vector::Vector(constVector&a)

{

size = a.size;

data = newint[size];

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

{

data[i] = a.data[i];

}

beg.elem = &data[0];

end.elem = &data[size];

}

Vector::~Vector()

{

delete[] data;

data = 0;

}

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

{

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

size = a.size;

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

data = newint[size];

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

{

data[i] = a.data[i];

}

beg.elem = &data[0];

end.elem = &data[size];

return \*this;

}

int&Vector::operator[](intindex)

{

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

elsecout<<"Error! Index > size\n";

}

VectorVector::operator+(constintk)

{

Vector temp(\*this);

for (inti = 0; i< size; ++i) temp.data[i] += k;

return temp;

}

intVector::operator()()

{

returnsize;

}

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

{

for (inti = 0; i<a(); ++i) out<<a.data[i] <<" ";

out<<"\n";

returnout;

}

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

{

for (inti = 0; i<a(); ++i) in>>a.data[i];

returnin;

}

**main.cpp**

#include<iostream>

#include"Vector.h"

// Lab 6. Containers

usingnamespacestd;

intmain()

{

Vectora(5);

cout<< a <<"\n";

cin>> a;

cout<< a <<"\n";

a[2] = 100;

cout<< a <<"\n";

Vectorb(10);

cout<< b <<"\n";

b = a;

cout<< b <<"\n";

Vectorc(10);

c = b + 100;

cout<< c <<"\n";

cout<<"\nThe length of a = "<<a()<<endl;

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

Iteratori = a.first();

++i;

cout<<\*i<<endl;

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

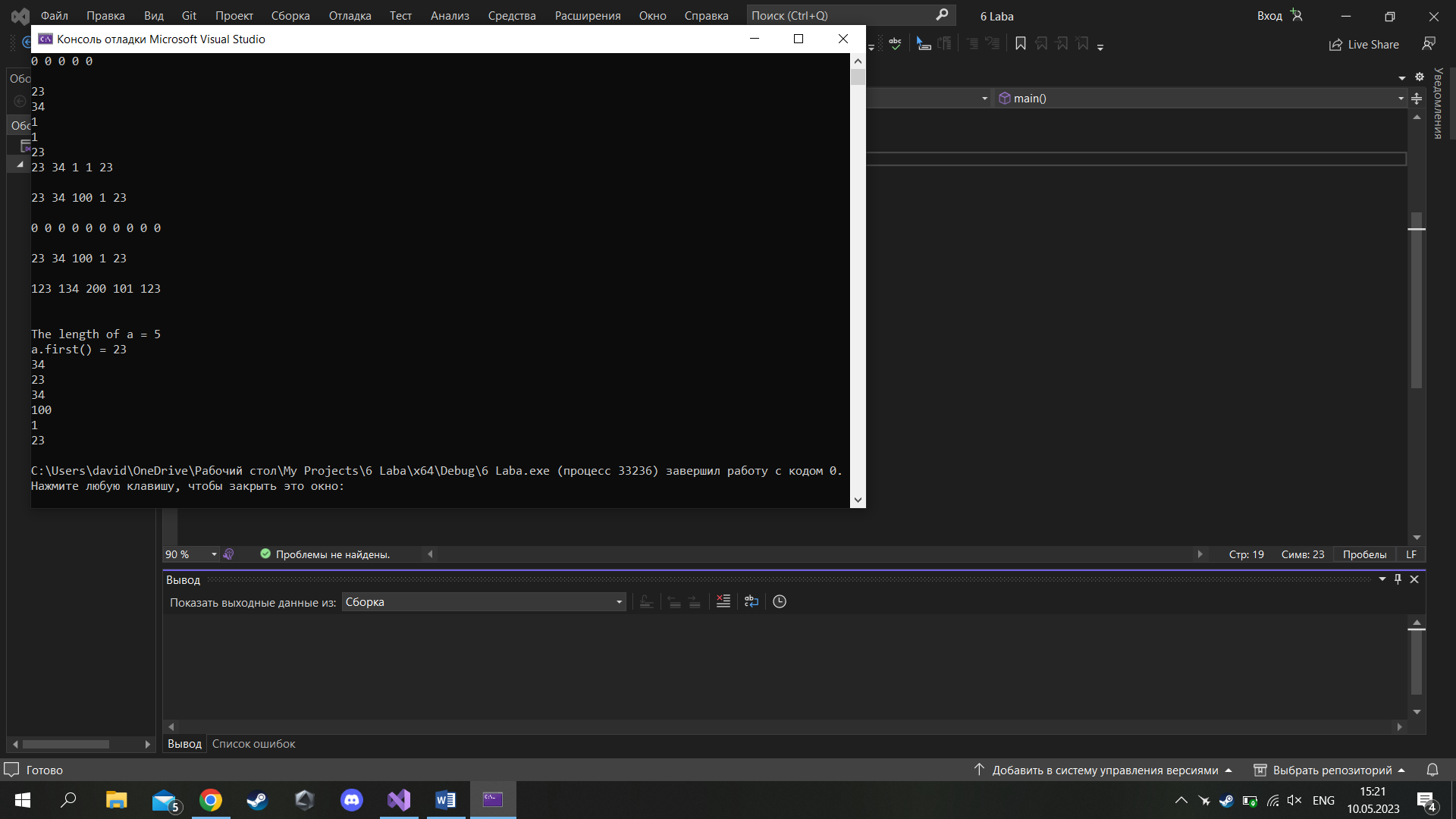
{

cout<<\*i <<endl;

}

}

**Результаты работы программы.**



**Ответы на контрольные вопросы.**

1. Что такое абстрактный тип данных? Привести примеры АТД

АТД - тип данных, определяемый только через операции, которые могут

выполняться над соответствующими объектами безотносительно к способу представления этих объектов.

template<typename T>

T &Stack<T>::push(){ //операция добавления элементов в АТД типа стек

returnhead->data;

}

2. Привести примеры абстракции через параметризацию.

template <typename T>

class node {

T data;

node\* next, \*prev;

}

3. Привести примеры абстракции через спецификацию.

class complex {

double re, im;

friend complex operator+(complex, complex);

public:

complex() { re = im = 0;}

complex(double r) { re = r; im = 0; }

complex(double r, double i) { re = r; im = i; }

};

complex operator+(complex a1, complex a2) {

return complex(a1.re + a2.re, a1.im + a2.im);

}

Для поддержки смешанной арифметики комплексных и действительных чисел (Re x, In y) + double A требуется специфицировать перегруженную функцию.

complex operator+(complex,complex);

complex operator+(complex,double);

complexoperator+(double,complex);

Абстракция через спецификацию достигается за счет того, что операции

представляются как часть типа (абстракция вычислений, перегрузка операторов).

4. Что такое контейнер? Привести примеры.

Контейнер – это объект. Имя контейнера – это имя переменной. Контейнер, так же как и другие объекты, обладает временем жизни. Время жизни контейнера в общем случае не зависит от времени жизни его элементов. Элементами контейнера могут любые объекты, в том числе, и другие контейнеры.

class queue() {

int size;

node \*head, \*tail;

…

};

//в основной функции

queue\* q; //контейнер-очередь

float a = 7.5;

q->head = a; //головой очереди q является float переменная

queue \*c;

c->tail = q; //хвостом очереди c является контейнер queue

5. Какие группы операций выделяют в контейнерах?

- Операции доступа к элементам

template<typename T>

class queue() {

int size;

node \*head, \*tail;

node &getHead() { return head->data; }

…

};

- Операциизаменызначенийэлементов;

void queue::setHead(node head){ this->head = head; }

- Операции добавления и удаления элементов или групп элементов;

T queue::pop(){

T data = tail->data;

size–;

tail = tail->prev;

tail->next = nullptr;

returndata;

}

- Операции поиска элементов и групп элементов;

intqueue::find\_index(T data) {

int counter = 0;

node \* curr = this->head;

while (counter < size) {

if (curr->data = data) return counter;

curr = curr->next;

counter++;

}

}

- Операции объединения контейнеров;

queue queue::merge(queue q1, queue q2) {

q1->tail->next = q2->head;

q1->tail = q2->tail;

delete q2;

return q1;

}

- Специальные операции, которые зависят от вида контейнера.

bool queue::isCircular(queue q) {

return (q->tail->next == q->head); }

6. Какие виды доступа к элементам контейнера существуют? Привести примеры.

Доступ к элементам контейнера бывает: последовательный, прямой и

ассоциативный.

Прямой доступ – это доступ по индексу. Например, a[10] – требуется найти

элемент контейнера с номером 10 (с учетом индексации с нуля)

Ассоциативный доступ также выполняется по индексу, но индексом будет являться не номер элемента, а его содержимое. Пусть имеется контейнер–словарь, в котором хранится информация, состоящая, как минимум из двух полей: слово и его перевод.

Индексом будут данные поля, например, q[T data]. Поле, с содержимым которого ассоциируется элемент контейнера, называется ключом или полем доступа. Элемент, с которым ассоциируется ключ, называется

значением. Контейнер, который представляет ассоциативный доступ, состоит из пар «ключ-значение» (ключ - T data, значение - nodename).

При последовательном доступе осуществляется перемещение от элемента к элементу контейнера.

Например, дана очередь, и для того, чтобы найти индекс элемента, требуется поэлементно пройти по каждому узлу.

intqueue::find\_index(T data) {

int counter = 0;

node \* curr = this->head;

while (counter < size) {

if (curr->data = data) return counter;

curr = curr->next;

counter++;

}

}

7. Что такое итератор?

Итератор (перечислитель) — интерфейс, предоставляющий доступ к элементам контейнера и навигацию по ним. Главное предназначение

итераторов заключается в предоставлении возможности пользователю обращаться к любому элементу контейнера при сокрытии внутренней структуры контейнера от пользователя. Это позволяет контейнеру хранить элементы любым способом при допустимости работы пользователя с ним как с простой последовательностью или списком. Итератор можно реализовать как класс, представляющий такой же набор операций.

В С++ итератор реализуется как класс, который имеет такой же интерфейс, как и указатель для совместимости с массивами.

8. Каким образом может быть реализован итератор?

classiterator() {

friendclassqueue; //поддержка работы с классом очередей

public:

iterator() { elem = nullptr; }

…

iterator&operator++() { elem = elem->next; //реализация части кода с переходом на следующий элемент

}

…

};

9. Каким образом можно организовать объединение контейнеров?

queue queue::merge(queue q1, queue q2) {

q1->tail->next = q2->head;

q1->tail = q2->tail;

delete q2;

return q1;

}

10. Какой доступ к элементам предоставляет контейнер, состоящий из элементов «ключ-значение»?

Ассоциативный доступ (индексом будут данные поля, например, q[T data]. Поле, с содержимым которого ассоциируется элемент контейнера, называется ключом или полем доступа. Элемент, с которым ассоциируется ключ, называется значением)

11. Как называется контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера?

Стек

12. Какой из объектов (a,b,c,d) является контейнером?

a. intmas=10; //переменная типа int

b. 2. intmas; //переменная типа int

c. 3. struct {char name[30]; int age;} mas; //структура

d. 4. intmas[100]; //массив переменных int

Ответ: d

13. Какой из объектов (a,b,c,d) не является контейнером?

a. inta[]={1,2,3,4,5}; //заполненный массив переменных int

b. 2. intmas[30]; //пустой массив переменных int

c. 3. struct {char name[30]; int age;} mas[30]; //массивструктур

d. 4. intmas; //переменная типа int

Ответ: d

14. Контейнер реализован как динамический массив, в нем определена операция доступ по индексу. Каким будет доступ к элементам контейнера?

В С++ массивы поддерживают доступ по индексу. Доступ по индексу - это прямой доступ.

15. Контейнер реализован как линейный список. Каким будет доступ к элементам контейнера?

В С++ в линейной списке нет доступа по индексу, и требуется пройти по каждому элементу. Это последовательный доступ