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 № 15.11**

Дисциплина: «Информатика»

Тема Объектно-ориентированное программирование Последовательные контейнеры библиотеки STL

Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь 2023**

**Цель работы**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Использование последовательных контейнеров библиотеки STL в ОО программе.

**Постановка задачи**

Задача 1.

1. Создать последовательный контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте).
3. Добавить элементы в соответствии с заданием.
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 2.

1. Создать последовательный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 3

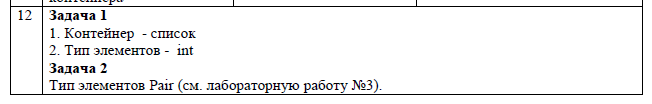
1. Создать параметризированный класс, используя в качестве контейнера последовательный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

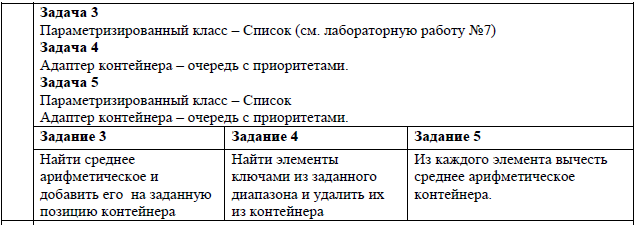
Задача 4

1. Создать адаптер контейнера.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 5

1. Создать параметризированный класс, используя в качестве контейнера адаптер контейнера.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

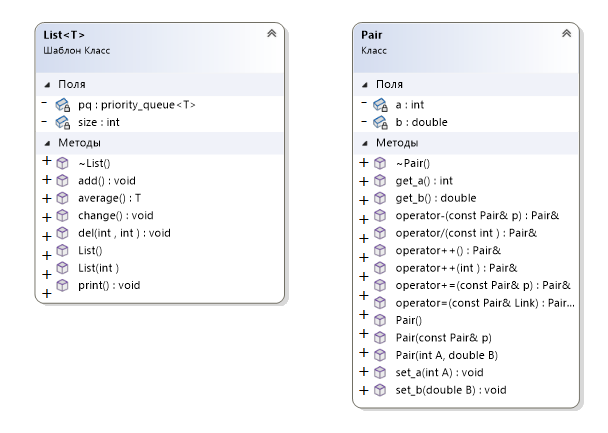




**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение функций для работы со списком типа int: создание списка, вычисление среднего арифметического, вычитание из каждого элемента списка значение соответствующее среднему арифметическому значению, добавление среднего арифметического на заданную позицию, удаление элементов списка в заданном диапазоне вывод списка.
   2. Реализовать определение функций для работы списка с типом Pair: создание списка, вычисление среднего арифметического, вычитание из каждого элемента списка значение соответствующее среднему арифметическому значению, добавление среднего арифметического на заданную позицию, удаление элементов списка в заданном диапазоне вывод списка.
   3. Реализовать определение параметризированного класса List и методов данного класса.
   4. Реализовать определение функций для работы с адаптером контейнера priority\_queue.
   5. Реализовать определение параметризированного класса List и его методов для работы с адаптером шаблонного контейнера priority\_queue.
   6. Реализовать применение этих функций в главной функции.
2. Поставленные задачи решены следующими действиями:
   1. Для каждой из задач (1 – 5) были реализованы функции создания списка, вычисления среднего арифметического, вычитания среднего арифметического значения, добавления среднего арифметического на заданную позицию, удаления элементов из заданного диапазона и вывода списка.
   2. В классе Pair находится реализация ввода-вывода и работы с числами

**UML – диаграмма**

****

**Реализация задачи на языке С++**

**Заголовочный файл Pair.h**

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int a;

double b;

public:

Pair();

Pair(int A, double B);

Pair(const Pair& p);

~Pair();

int get\_a();

void set\_a(int A);

double get\_b();

void set\_b(double B);

Pair& operator ++(); // Оператор префиксного инкремента

Pair& operator ++(int); // Оператор постфиксного инкремента

Pair& operator = (const Pair& Link); // Оператор присваивания

friend bool operator < (const Pair& p, const Pair& q); // Оператор сравнения

friend bool operator > (const Pair& p, const Pair& q); // Оператор сравнения

friend bool operator ==(const Pair& p, const Pair& q);

friend istream& operator >> (istream& in, Pair& Pointer); // Оператор ввода

friend ostream& operator << (ostream& out, const Pair& Pointer); // Оператор вывода

Pair& operator /(const int);

Pair& operator +=(const Pair& p);

Pair& operator -(const Pair& p);

};

**Файл с описанием методов класса Pair.cpp**

#include "Pair.h"

Pair::Pair()

{

a = 0;

b = 0.0;

}

Pair::Pair(int A, double B)

{

a = A;

b = B;

}

Pair::Pair(const Pair& p)

{

a = p.a;

b = p.b;

}

Pair::~Pair()

{

}

int Pair::get\_a()

{

return a;

}

void Pair::set\_a(int A)

{

a = A;

}

double Pair::get\_b()

{

return b;

}

void Pair::set\_b(double B)

{

b = B;

}

Pair& Pair::operator ++() // Оператор префиксного инкремента

{

++a;

return \*this;

}

Pair& Pair::operator ++(int) // Оператор постфиксного инкремента

{

b++;

return \*this;

}

Pair& Pair::operator = (const Pair& Link) // Оператор присваивания

{

if (&Link != this)

{

a = Link.a;

b = Link.b;

}

return \*this;

}

bool operator < (const Pair& p, const Pair& q) // Оператор сравнения

{

if (p.a < q.a)

{

return true;

}

else

{

if (p.a > q.a)

{

return false;

}

else

{

if (p.b < q.b)

{

return true;

}

else

{

return false;

}

}

}

}

bool operator > (const Pair& p, const Pair& q) // Оператор сравнения

{

if (p.a > q.a)

{

return true;

}

else

{

if (p.a < q.a)

{

return false;

}

else

{

if (p.b > q.b)

{

return true;

}

else

{

return false;

}

}

}

}

bool operator == (const Pair& p, const Pair& q) // Оператор сравнения

{

return (p.a == q.a and p.b == q.b);

}

istream& operator >> (istream& in, Pair& Pointer) // Оператор ввода

{

cout << "input a: ";

in >> Pointer.a;

cout << "input b: ";

in >> Pointer.b;

return in;

}

ostream& operator << (ostream& out, const Pair& Pointer) // Оператор вывода

{

cout << "a:b ";

return (out << Pointer.a << " : " << Pointer.b << "\n");

}

Pair& Pair::operator /(const int x)

{

a = a / x;

b = b / x;

return \*this;

}

Pair& Pair::operator +=(const Pair& p)

{

this->a += p.a;

this->b += p.b;

return \*this;

}

Pair& Pair::operator -(const Pair& p) {

this->a -= p.a;

this->b -= p.b;

return \*this;

}

**(1)**

**Файл с главной программой main.cpp**

#include <iostream>

#include <list>

#include <ctime>

using namespace std;

typedef list<int> lst;

lst make\_list(int n)

{

lst l;

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

l.push\_back(rand() % 100 - 65);

return l;

}

void print\_list(lst &l)

{

for (auto it = l.begin(); it != l.end(); it++)

cout << \*it << " ";

cout << endl;

}

int average(lst &l)

{

int sum = 0;

for (auto it = l.begin(); it != l.end(); it++)

sum += \*it;

int size = l.size();

return sum / size;

}

lst add(lst& l, int pos)

{

int elem = average(l);

auto it = l.begin();

advance(it, pos - 1); // -1, т.к учитывается begin

l.insert(it, elem);

return l;

}

lst del(lst& l, int start, int finish)

{

auto it = l.begin();

advance(it, start - 1);

auto start\_it = it;

it = l.begin();

advance(it, finish);

auto finish\_it = it;

l.erase(start\_it, finish\_it);

return l;

}

lst change(lst& l)

{

int elem = average(l);

for (auto it = l.begin(); it != l.end(); it++)

\*it = \*it - elem;

return l;

}

void main()

{

srand(time(NULL));

try

{

list <int> tmp;

int n;

cout << "enter n: ";

cin >> n;

tmp = make\_list(n);

cout << "make: ";

print\_list(tmp);

int pos;

cout << "enter pos: ";

cin >> pos;

tmp = add(tmp, pos);

cout << "add: ";

print\_list(tmp);

int l, r;

cout << "enter range: ";

cin >> l >> r;

tmp = del(tmp, l, r);

cout << "del: ";

print\_list(tmp);

tmp = change(tmp);

cout << "change: ";

print\_list(tmp);

}

catch(int)

{

cout << "\nError!\n";

}

**(2)**

**Файл с главной программой main.cpp**

#include <iostream>

#include <list>

#include <ctime>

#include "Pair.h"

using namespace std;

typedef list<Pair> lst;

lst make\_list(int n)

{

lst l;

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

{

Pair a;

cin >> a;

l.push\_back(a);

}

return l;

}

void print\_list(lst& l)

{

for (auto it = l.begin(); it != l.end(); it++)

cout << \*it;

cout << endl;

}

Pair average(lst& l)

{

Pair sum ;

for (auto it = l.begin(); it != l.end(); it++)

sum += \*it;

int size = l.size();

return sum / size;

}

lst add(lst& l, int pos)

{

Pair elem = average(l);

auto it = l.begin();

advance(it, pos - 1); // -1, т.к учитывается begin

l.insert(it, elem);

return l;

}

lst del(lst& l, int start, int finish)

{

auto it = l.begin();

advance(it, start - 1);

auto start\_it = it;

it = l.begin();

advance(it, finish);

auto finish\_it = it;

l.erase(start\_it, finish\_it);

return l;

}

lst change(lst& l)

{

Pair elem = average(l);

for (auto it = l.begin(); it != l.end(); it++)

\*it = \*it - elem;

return l;

}

void main()

{

srand(time(NULL));

try

{

list <Pair> tmp;

int n;

cout << "enter n: ";

cin >> n;

tmp = make\_list(n);

cout << "make: ";

print\_list(tmp);

int pos;

cout << "enter pos: ";

cin >> pos;

tmp = add(tmp, pos);

cout << "add: ";

print\_list(tmp);

int l, r;

cout << "enter range: ";

cin >> l >> r;

tmp = del(tmp, l, r);

cout << "del: ";

print\_list(tmp);

tmp = change(tmp);

cout << "change: ";

print\_list(tmp);

}

catch (int)

{

cout << "\nError!\n";

}

}

**(3)**

**Заголовочный файл List.h**

#pragma once

#include <iostream>

#include <list>

using namespace std;

template <class T>

class List

{

private:

list <T> lst;

int size;

public:

List();

List(int); // Конструктор с параметрами: выделение под s элементов и заполнение их значением k

~List(); // Деструктор

T average();

void add(int);

void del(int, int);

void change();

void print();

};

template <class T>

List<T>::List() { size = 0;}

template <class T>

List<T>::List(int n)

{

T el;

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

{

cin >> el;

lst.push\_back(el);

}

size = lst.size();

}

template <class T>

List<T>::~List() {}

template <class T>

void List<T>::print()

{

for (auto it = lst.begin(); it != lst.end(); it++)

cout << \*it;

cout << endl;

}

template <class T>

T List<T>::average()

{

T sum;

for (auto it = lst.begin(); it != lst.end(); it++)

sum += \*it;

size = lst.size();

return sum / size;

}

template <class T>

void List<T>::add(int pos)

{

T elem = average();

auto it = lst.begin();

advance(it, pos - 1); // -1, т.к учитывается begin

lst.insert(it, elem);

}

template <class T>

void List<T>::del(int start, int finish)

{

auto it = lst.begin();

advance(it, start - 1);

auto start\_it = it;

it = lst.begin();

advance(it, finish);

auto finish\_it = it;

lst.erase(start\_it, finish\_it);

}

template <class T>

void List<T>::change()

{

T elem = average();

for (auto it = lst.begin(); it != lst.end(); it++)

\*it = \*it - elem;

}

**Файл с главной программой main.cpp**

#include <iostream>

#include <ctime>

#include "Pair.h"

#include "List.h"

using namespace std;

int main()

{

srand(time(NULL));

try

{

int n;

cout << "enter n: ";

cin >> n;

List <Pair> tmp(n);

cout << "\nmake: " << endl;

tmp.print();

int pos;

cout << "enter pos: ";

cin >> pos;

tmp.add(pos);

cout << "\nadd: " << endl;

tmp.print();

int l, r;

cout << "enter range: ";

cin >> l >> r;

tmp.del(l, r);

cout << "\ndel: " << endl;

tmp.print();

tmp.change();

cout << "\nchange: " << endl;

tmp.print();

}

catch (int)

{

cout << "\nError!\n";

}

}

**(4)**

**Файл с главной программой main.cpp**

#include <iostream>

#include "Pair.h"

#include <queue>

#include<list>

using namespace std;

typedef priority\_queue <Pair> pq;

pq make(int n)

{

pq tmp;

Pair el;

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

{

cin >> el;

tmp.push(el);

}

return tmp;

}

void print(pq t)

{

while(!t.empty())

{

cout << t.top();

t.pop();

}

cout << endl;

}

Pair average(pq t)

{

Pair sum;

int size = t.size();

while (!t.empty())

{

sum += t.top();

t.pop();

}

return sum / size;

}

void add(pq& t) //в данном случае, не получиться добавить в указанную позицию. Добавление по приоритету

{

Pair el = average(t);

t.push(el);

}

void del(pq& t, int l, int r)

{

pq tmp;

int i = 1;

while (!t.empty())

{

Pair el = t.top();

if (i < l or i > r) tmp.push(el);

t.pop();

i++;

}

swap(tmp, t);

}

void change(pq& t)

{

Pair el = average(t);

pq tmp;

while (!t.empty())

{

Pair elem = t.top();

Pair em = elem - el;

tmp.push(em);

t.pop();

}

swap(tmp, t);

}

int main()

{

try

{

int n;

cout << "enter n: ";

cin >> n;

pq tmp;

tmp = make(n);

cout << "\nmake: " << endl;

print(tmp);

add(tmp);

cout << "\nadd: " << endl;

print(tmp);

int l, r;

cout << "enter range: ";

cin >> l >> r;

del(tmp, l, r);

cout << "\ndel: " << endl;

print(tmp);

change(tmp);

cout << "\nchange: " << endl;

print(tmp);

}

catch (int)

{

cout << "\nError!\n";

}

}

**(5)**

**Заголовочный файл List.h**

#pragma once

#include <iostream>

#include <list>

#include <queue>

using namespace std;

template <class T>

class List

{

private:

priority\_queue <T> pq;

int size;

public:

List();

List(int); // Конструктор с параметрами: выделение под s элементов и заполнение их значением k

~List(); // Деструктор

T average();

void add();

void del(int, int);

void change();

void print();

};

template <class T>

List<T>::List() { size = 0; }

template <class T>

List<T>::List(int n)

{

T el;

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

{

cin >> el;

pq.push(el);

}

size = pq.size();

}

template <class T>

List<T>::~List() {}

template <class T>

void List<T>::print()

{

priority\_queue<T> tmp;

while (!pq.empty())

{

T el = pq.top();

cout << el << "\n";

tmp.push(el);

pq.pop();

}

swap(tmp, pq);

cout << endl;

}

template <class T>

T List<T>::average()

{

T sum;

size = pq.size();

priority\_queue<T> tmp = pq;

while (!tmp.empty())

{

sum += tmp.top();

tmp.pop();

}

return sum / size;

}

template <class T>

void List<T>::add()

{

T el = average();

pq.push(el);

}

template <class T>

void List<T>::del(int l, int r)

{

priority\_queue <T> tmp;

int i = 1;

while (!pq.empty())

{

T el = pq.top();

if (i < l or i > r) tmp.push(el);

pq.pop();

i++;

}

swap(tmp, pq);

}

template <class T>

void List<T>::change()

{

T el = average();

priority\_queue <T> tmp;

while (!pq.empty())

{

T elem = pq.top();

T em = elem - el;

tmp.push(em);

pq.pop();

}

swap(tmp, pq);

}

**Файл с главной программой main.cpp**

#include <iostream>

#include <ctime>

#include "Pair.h"

#include "List.h"

using namespace std;

int main()

{

try

{

int n;

cout << "enter n: ";

cin >> n;

List <Pair> tmp(n);

cout << "\nmake: " << endl;

tmp.print();

tmp.add();

cout << "\nadd: " << endl;

tmp.print();

int l, r;

cout << "enter range: ";

cin >> l >> r;

tmp.del(l, r);

cout << "\ndel: " << endl;

tmp.print();

tmp.change();

cout << "\nchange: " << endl;

tmp.print();

}

catch (int)

{

cout << "\nError!\n";

}

}

**Ответы на вопросы**

Ответы на контрольные вопросы

1. Из каких частей состоит библиотека STL?

STL – Standard Template Library, стандартная библиотека шаблонов

состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов. Контейнеры — это объекты, содержащие другие однотипные объекты. Обобщенные алгоритмы реализуют большое количество процедур, применимых к контейнерам: поиск, сортировку, слияние и т. п.

2. Какие типы контейнеров существуют в STL?

Последовательные контейнеры (векторы (vector), списки (list) и двусторонние очереди (deque)), ассоциативные контейнеры (словари (mар), словари с дубликатами (multimap), множества (set), множества с дубликатами (multiset) и битовые множества (bitset)), есть еще специализированные контейнеры (или адаптеры контейнеров), реализованные на основе базовых — стеки (stack), очереди (queue) и очереди с приоритетами (priority\_queue)

3. Что нужно сделать для использования контейнера STL в своей программе?

Для использования контейнера в программе необходимо включить в нее соответствующий заголовочный файл. Тип объектов, сохраняемых в контейнере, задается с помощью аргумента шаблона, например:

#include <vector>

#include <list>

4. Что представляет собой итератор?

Итераторы (iterators) - это объекты, которые по отношению к контейнеру играют роль указателей. Они позволяют получить доступ к содержимому контейнера примерно так же, как указатели используются для доступа к элементам массива.

5. Какие операции можно выполнять над итераторами?

С итераторами можно работать так же, как с указателями. К ним можно применить операции \*, инкремента, декремента. Присваивание одного итератора другому. Сравнение итераторов на равенство и неравенство (== и !=).

6. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?

for (iterator it = v.begin(); it != v.end(), ++it) cout << v[it] << endl; //vector

7. Какие типы итераторов существуют?

Существует пять типов итераторов:

- Итераторы ввода (input iterator) поддерживают операции равенства, разыменования и инкремента. ==, !=, \*i, ++i, i++, \*i++

- Итераторы вывода (output iterator) поддерживают операции разыменования, допустимые только с левой стороны присваивания, и инкремента. ++i, i++, \*i = t, \*i++ = t

- Однонаправленные итераторы (forward iterator) поддерживают все операции итераторов ввода/вывода и, кроме того, позволяют без ограничения применять присваивание. ==, !=, =, \*i, ++i, i++, \*i++

- Двунаправленные итераторы (bidirectional iterator) обладают всеми свойствами forward-итераторов, а также имеют дополнительную операцию декремента (--i, i--, \*i--), что позволяет им проходить контейнер в обоих направлениях.

- Итераторы произвольного доступа (random access iterator) обладают всеми свойствами bidirectional-итераторов, а также поддерживают операции сравнения и адресной арифметики, то есть непосредственный доступ по индексу. i += n, i + n, i -= n, i - n, i1 - i2, i[n], i1 < i2, i1 <= i2, i1 > i2, i1 >= i2

В STL также поддерживаются обратные итераторы (reverse iterators). Обратными итераторами могут быть либо двунаправленные итераторы, либо итераторы произвольного доступа, но проходящие последовательность в обратном направлении.

8. Перечислить операции и методы общие для всех контейнеров.

Операции равенства (==), неравенства (!=), операция присваивания (=), clear, insert, erase, size\_type size() const, size\_type max\_size() const, bool empty() const, iterator begin(), iterator end(), reverse\_iterator begin(), reverse\_iterator end()

9. Какие операции являются эффективными для контейнера vector? Почему?

Контейнер вектор является аналогом обычного массива, за исключением того, что он автоматически выделяет и освобождает память по мере необходимости. Контейнер эффективно обрабатывает произвольную выборку элементов с помощью операции индексации [] или метода at. Происходит это потому, что массив - это последовательно занятая память, так что доступ к любому элементу происходит быстро.

10. Какие операции являются эффективными для контейнера list? Почему?

Контейнер список организует хранение объектов в виде двусвязного списка. Каждый элемент списка содержит три поля: значение элемента, указатель на предшествующий и указатель на последующий элементы списка. Вставка и удаление работают эффективно для любой позиции элемента в списке, поскольку не требуется заново перевыделять память, достаточно переобозначить связи с предыдущим и последующим элементом.

11. Какие операции являются эффективными для контейнера deque? Почему?

Контейнер двусторонняя очередь во многом аналогичен вектору, элементы хранятся в непрерывной области памяти. Но в отличие от вектора двусторонняя очередь эффективно поддерживает вставку и удаление первого элемента (так же, как и последнего). Дек является сочетанием вектора и списка, поэтому при добавлении элементов в начало создается новый массив, конец которого по принципа списка указывает на изначальный массив. Аналогично с добавлением элементов в конец.

12. Перечислить методы, которые поддерживает последовательный контейнер vector.

push\_back(), pop\_back(), insert, erase, [], at, swap, clear()

13. Перечислить методы, которые поддерживает последовательный контейнер list.

push\_back(), pop\_back(), push\_front(), pop\_back(), insert(), erase, swap, clear(), splice

14. Перечислить методы, которые поддерживает последовательный контейнер deque.

push\_back(), push\_front(), pop\_back, pop\_front, insert, erase, [] ,at

15. Задан контейнер vector. Как удалить из него элементы со 2 по 5?

for (int i = 2; i <= 5; ++i) v.erase(v.begin()+i);

16. Задан контейнер vector. Как удалить из него последний элемент?

v.erase(--v.end()); //итератор end() указывает на ячейку памяти после последнего элемента

17. Задан контейнер list. Как удалить из него элементы со 2 по 5?

list<int>::iterator beg = l.begin(); advance(beg, 2);

list<int>::iterator end = l.begin(); advance(end, 6); //удаляем включительно

l.erase(beg, end);

18. Задан контейнер list. Как удалить из него последний элемент?

l.erase(--l.end());

19. Задан контейнер deque. Как удалить из него элементы со 2 по 5?

auto beg = q.begin(); advance(beg, 2);

auto end = q.begin(); advance(end, 6);

q.erase(beg, end);

20. Задан контейнер deque. Как удалить из него последний элемент?

q.erase(--q.end());

21. Написать функцию для печати последовательного контейнера с использованием итератора.

void print(deque<int>q) {

for (deque<int>::iterator elem = q.begin(); elem != q.end(); ++elem)

cout << \*elem << " ";

}

22. Что представляют собой адаптеры контейнеров?

Специализированные последовательные контейнеры — стек, очередь и очередь с приоритетами — не являются самостоятельными контейнерными классами, а реализованы на основе рассмотренных выше классов, поэтому они называются адаптерами контейнеров.

23. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?

По умолчанию для стека прототипом является класс deque.

Объявление stack<int> s создает стек на базе двусторонней очереди (по

умолчанию). Если по каким-то причинам нас это не устраивает, и мы хотим создать стек на базе списка, то объявление будет выглядеть следующим образом: stack<int, list<int> > s;

24. Перечислить методы, которые поддерживает контейнер stack.

push () - добавление в конец; pop () - удаление из конца; top () - получение текущего элемента стека; empty() - проверка пустой стек или нет; size () – получение размера стека.

25. Перечислить методы, которые поддерживает контейнер queue.

push () – добавление в конец очереди; pop () – удаление из начала очереди; front () – получение первого элемента очереди; back()- получение последнего элемента очереди; empty () – проверка пустая очередь или нет; size() – получение размера очереди.

26. Чем отличаются друг от друга контейнеры queue и priority\_queue?

Шаблонный класс priority\_queue (заголовочный файл <queue>)

поддерживает такие же операции, как и класс queue, но реализация класса возможна либо на основе вектора (реализация по умолчанию), либо на основе списка. Очередь с приоритетами отличается от обычной очереди тем, что для извлечения выбирается максимальный элемент из хранимых в контейнере. Поэтому после каждого изменения состояния очереди максимальный элемент из оставшихся сдвигается в начало контейнера.

27. Задан контейнер stack. Как удалить из него элемент с заданным номером?

stack<int> copy;

int counter = 0;

while (counter != 2) {

copy.push(s.top());

s.pop();

++counter;

}

while (counter != 6) { //удаление не включительно

s.pop();

++counter;

}

while (!copy.empty()) {

s.push(copy.top());

copy.pop();

}

28. Задан контейнер queue. Как удалить из него элемент с заданным номером?

//перемещение всех элементов до удаляемого в конец

for (int i = 1; i < num; i++) {

push(qwe, qwe->head->key);

pop(qwe); }

pop(qwe); //удаление элемента

//возвращение очереди в прежнее состояние

//первый элемент снова ставится в начало

for (int i = 0; i < (qwe->size) - (num + 2 \* k); i++) {

push(qwe, qwe->head->key);

pop(qwe); }

29. Написать функцию для печати контейнера stack с использованием итератора.

void print(stack<int> s) {

int size = s.size();

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

cout << s.top() << " ";

s.pop(); }

}

30. Написать функцию для печати контейнера queue с использованием итератора.

void print(queue<int> s) {

int size = s.size();

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

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

s.pop(); }

}