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: Последовательные контейнеры библиотеки STL

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).**

Задача1.

1. Создать последовательный контейнер.
2. Заполнить его элементами стандартного типа(тип указан в варианте).
3. Добавитьэлементывсоответствиис заданием
4. Удалитьэлементывсоответствиисзаданием.
5. Выполнитьзаданиевариантадляполученного контейнера.
6. Выполнениевсех заданийоформитьввидеглобальных функций.

Задача2.

1. Создатьпоследовательныйконтейнер.
2. Заполнитьегоэлементамипользовательскоготипа(типуказанвварианте).Для пользовательского типа перегрузить необходимые операции.
3. Добавитьэлементывсоответствиис заданием
4. Удалитьэлементывсоответствиисзаданием.
5. Выполнитьзаданиевариантадляполученного контейнера.
6. Выполнениевсех заданийоформитьввидеглобальных функций.

Задача3

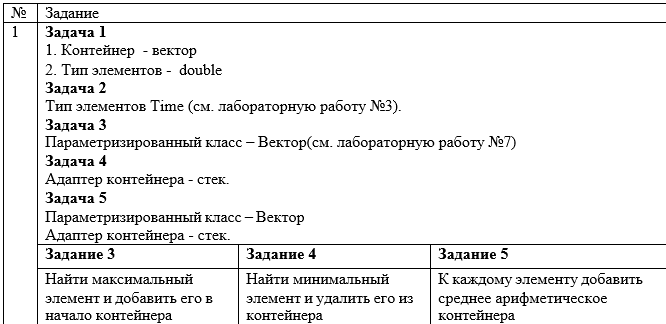
1. Создатьпараметризированныйкласс,используявкачествеконтейнера последовательный контейнер.
2. Заполнить его элементами.
3. Добавитьэлементывсоответствиис заданием
4. Удалитьэлементывсоответствиисзаданием.
5. Выполнитьзаданиевариантадляполученного контейнера.
6. Выполнениевсехзаданийоформитьввидеметодовпараметризированного класса.

Задача4

1. Создать адаптер контейнера.
2. Заполнитьегоэлементамипользовательскоготипа(типуказанвварианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавитьэлементывсоответствиис заданием
4. Удалитьэлементывсоответствиисзаданием.
5. Выполнитьзаданиевариантадляполученного контейнера.
6. Выполнениевсех заданийоформитьввидеглобальных функций.

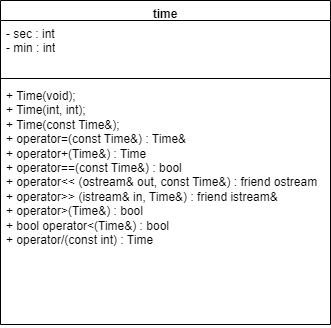
Задача5

1. Создатьпараметризированныйкласс,используявкачествеконтейнераадаптер контейнера.
2. Заполнить его элементами.
3. Добавитьэлементывсоответствиис заданием
4. Удалитьэлементывсоответствиис заданием.
5. Выполнитьзаданиевариантадляполученного контейнера.
6. Выполнениевсехзаданийоформитьввидеметодовпараметризированного класса.

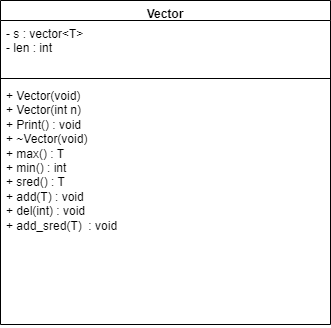


**UML**

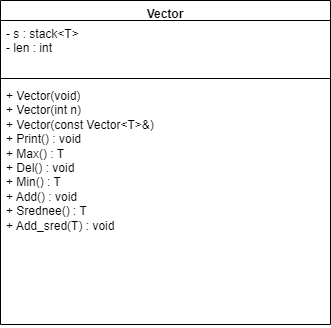
**2**

****

**3**

****

**5**

****

**Функции для решения задачи 1**

vector<double> MakeVector(int n)

{

vector<double> v(n);

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

{

v[i] = rand() % 100 - 50;

}

return v;

}

double GetMax(vector<double>\* v)

{

double max = (\*v)[0];

for (int i = 1; i < v->size(); i++)

{

if ((\*v)[i] > max)

{

max = (\*v)[i];

}

}

return max;

}

double GetMin(vector<double>\* v, int& pos)

{

double min = (\*v)[0];

pos = 0;

for (int i = 1; i < v->size(); i++)

{

if ((\*v)[i] < min)

{

min = (\*v)[i];

pos = i;

}

}

return min;

}

double GetAverage(vector<double>\* v)

{

double sum = 0;

for (int i = 0; i < v->size(); i++)

{

sum += (\*v)[i];

}

return sum / v->size();

}

void AddForEach(vector<double>\* v, double b)

{

for (int i = 0; i < v->size(); i++)

{

(\*v)[i] += b;

}

}

**Основная функция для решения задачи 1**

void Execute1()

{

auto vec = MakeVector(10);

vec.insert(vec.begin(), GetMax(&vec));

int pos = -1;

GetMin(&vec, pos);

vec.erase(vec.begin() + pos);

AddForEach(&vec, GetAverage(&vec));

}

**Описание пользовательского класса для решения задачи 2.**

#pragma once

#include <iostream>

using namespace std;

class Time

{

private:

int min, sec;

public:

Time(int m = 0, int s = 0) { min = m; sec = s; }

Time(const Time& t) { min = t.min;sec = t.sec; }

~Time() {};

int get\_min() { return min; }

int get\_sec() { return sec; }

void set\_min(int m) { min = m; }

void set\_sec(int s) { sec = s; }

int toNumber() { return min \* 60 + sec; }

Time operator+(const Time& t)

{

int temp1 = toNumber();

int temp2 = t.min \* 60 + t.sec;

Time p;

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

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

return p;

}

bool operator>(const Time& t)

{

if (min > t.min || (min == t.min && sec > t.sec)) return true;

return false;

}

bool operator==(const Time& t)

{

return t.min == min && t.sec == sec;

}

bool operator<(const Time& t)

{

if (\*this > t || \*this == t) return false;

return true;

}

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

{

cout << "minutes = ";

in >> t.min;

cout << "seconds = ";

in >> t.sec;

return in;

}

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

{

return (out << t.min << ":" << t.sec);

}

};

**Функции для решения задачи 2**

#pragma once

#include <iostream>

#include <vector>

#include "Time.h"

using namespace std;

vector<Time> MakeVector2(int n)

{

vector<Time> v(n);

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

{

v[i] = Time(rand() % 60, rand() % 60);

}

return v;

}

Time GetMax(vector<Time>\* v)

{

Time max = (\*v)[0];

for (int i = 1; i < v->size(); i++)

{

if ((\*v)[i] > max)

{

max = (\*v)[i];

}

}

return max;

}

Time GetMin(vector<Time>\* v, int& pos)

{

Time min = (\*v)[0];

pos = 0;

for (int i = 1; i < v->size(); i++)

{

if ((\*v)[i] < min)

{

min = (\*v)[i];

pos = i;

}

}

return min;

}

Time GetAverage(vector<Time>\* v)

{

int sum = 0;

for (int i = 0; i < v->size(); i++)

{

sum += (\*v)[i].toNumber();

}

int a = sum / v->size();

return Time(a / 60, a % 60);

}

void AddForEach(vector<Time>\* v, Time b)

{

for (int i = 0; i < v->size(); i++)

{

(\*v)[i] = (\*v)[i] + b;

}

}

void Execute2()

{

auto vec = MakeVector2(10);

vec.insert(vec.begin(), GetMax(&vec));

int pos = -1;

GetMin(&vec, pos);

vec.erase(vec.begin() + pos);

AddForEach(&vec, GetAverage(&vec));

}

**Описание параметризированного класса для решения задачи 3**

#pragma once

#include <iostream>

using namespace std;

template <class T>

class Vector

{

public:

Vector(int s);

Vector(int s, T k);

Vector(const Vector<T>& a);

int Size()

{

return size;

}

void erase(int pos)

{

auto temp = new T[size - 1];

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

{

temp[i] = data[i];

}

for (int i = pos + 1; i < size; i++)

{

temp[i - 1] = data[i];

}

data = temp;

}

void insert(int pos, T elem)

{

auto temp = new T[size + 1];

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

{

temp[i] = data[i];

}

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

{

temp[i + 1] = data[i];

}

temp[pos] = elem;

data = temp;

}

Vector& operator=(const Vector<T>& a);

T& operator[](int index);

Vector operator+(const T k);

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

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

private:

int size;

T\* data;

};

template <class T>

Vector<T>::Vector(int s)

{

size = s;

data = new T[size];

}

template <class T>

Vector<T>::Vector(int s, T k)

{

size = s;

data = new T[size];

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

{

data[i] = k;

}

}

template <class T>

Vector<T>::Vector(const Vector& a)

{

size = a.size;

data = new T[size];

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

{

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

}

}

template <class T>

Vector<T>& Vector<T>::operator=(const Vector<T>& a)

{

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

size = a.size;

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

data = new T[size];

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

{

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

}

return \*this;

}

template <class T>

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

{

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

else cout << "Error! Index > size\n";

}

template <class T>

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

{

Vector<T> temp(\*this);

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

return temp;

}

template <class T>

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

{

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

out << "\n";

return out;

}

template <class T>

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

{

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

return in;

}

**Функции для решения задачи 3**

#pragma once

#include <iostream>

#include "Vector.h"

#include "Time.h"

using namespace std;

Vector<Time> MakeVector3(int n)

{

Vector<Time> v(n);

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

{

v[i] = Time(rand() % 60, rand() % 60);

}

return v;

}

Time GetMax(Vector<Time>\* v)

{

Time max = (\*v)[0];

for (int i = 1; i < v->Size(); i++)

{

if ((\*v)[i] > max)

{

max = (\*v)[i];

}

}

return max;

}

Time GetMin(Vector<Time>\* v, int& pos)

{

Time min = (\*v)[0];

pos = 0;

for (int i = 1; i < v->Size(); i++)

{

if ((\*v)[i] < min)

{

min = (\*v)[i];

pos = i;

}

}

return min;

}

Time GetAverage(Vector<Time>\* v)

{

int sum = 0;

for (int i = 0; i < v->Size(); i++)

{

sum += (\*v)[i].toNumber();

}

int a = sum / v->Size();

return Time(a / 60, a % 60);

}

void AddForEach(Vector<Time>\* v, Time b)

{

for (int i = 0; i < v->Size(); i++)

{

(\*v)[i] = (\*v)[i] + b;

}

}

void Execute3()

{

auto vec = MakeVector3(10);

vec.insert(0, GetMax(&vec));

int pos = -1;

GetMin(&vec, pos);

vec.erase(pos);

AddForEach(&vec, GetAverage(&vec));

}

**Функции для решения задачи 4**

#pragma once

#include <iostream>

#include <stack>

#include "Time.h"

using namespace std;

stack<Time> MakeStack(int n)

{

stack<Time> s;

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

{

s.push(Time(rand() % 60, rand() % 60));

}

return s;

}

Time GetMax(stack<Time> s)

{

Time max = s.top();

s.pop();

while (!s.empty())

{

if (s.top() > max)

{

max = s.top();

}

s.pop();

}

return max;

}

Time GetMin(stack<Time> s, int& pos)

{

Time min = s.top();

s.pop();

pos = 0;

int i = 1;

while(!s.empty())

{

if (s.top() < min)

{

min = s.top();

pos = i;

}

s.pop();

i++;

}

return min;

}

Time GetAverage(stack<Time> s)

{

int sum = 0;

int size = s.size();

while(!s.empty())

{

sum += s.top().toNumber();

s.pop();

}

int a = sum / size;

return Time(a / 60, a % 60);

}

void AddForEach(stack<Time>& s, Time b)

{

stack<Time> tmp;

int size = s.size();

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

{

tmp.push(s.top() + b);

s.pop();

}

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

{

s.push(tmp.top());

tmp.pop();

}

}

/\*

void Insert(stack<Time>& s, int pos, Time elem)

{

stack<Time> tmp;

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

{

tmp.push(s.top());

s.pop();

}

s.push(elem);

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

{

s.push(tmp.top());

tmp.pop();

}

}

\*/

void Erase(stack<Time>& s, int pos)

{

stack<Time> tmp;

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

{

tmp.push(s.top());

s.pop();

}

s.pop();

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

{

s.push(tmp.top());

tmp.pop();

}

}

void Execute4()

{

auto st = MakeStack(10);

st.push(GetMax(st));

int pos = -1;

GetMin(st, pos);

Erase(st, pos);

AddForEach(st, GetAverage(st));

}

**Описание параметризированного класса для решения задачи 5**

#pragma once

#include <iostream>

#include <stack>

using namespace std;

template <class T>

class VectorS

{

public:

VectorS(int s);

VectorS(int s, T k);

VectorS(const VectorS<T>& a);

int Size()

{

return size;

}

void erase(int pos)

{

stack<Time> tmp;

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

{

tmp.push(st.top());

st.pop();

}

st.pop();

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

{

st.push(tmp.top());

tmp.pop();

}

data = copyToVec(st);

}

void insert(int pos, T elem)

{

stack<T> tmp;

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

{

tmp.push(st.top());

st.pop();

}

st.push(elem);

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

{

st.push(tmp.top());

tmp.pop();

}

data = copyToVec(st);

}

VectorS& operator=(const VectorS<T>& a);

T& operator[](int index);

VectorS operator+(const T k);

friend ostream& operator<< <>(ostream& out, VectorS<T>& a);

friend istream& operator>> <>(istream& in, VectorS<T>& a);

stack<T> st;

stack<T> copyToStack()

{

stack<T> s;

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

{

s.push(data[i]);

}

return s;

}

stack<T> copyToStack(T\* a, int n)

{

stack<T> s;

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

{

s.push(a[i]);

}

return s;

}

T\* copyToVec(stack<T> s)

{

T\* a = new T[s.size()];

int i = 0;

while (!s.empty())

{

a[i] = s.top();

s.pop();

i++;

}

return a;

}

private:

int size;

T\* data;

};

template <class T>

VectorS<T>::VectorS(int s)

{

size = s;

data = new T[size];

st = copyToStack(data, size);

}

template <class T>

VectorS<T>::VectorS(int s, T k)

{

size = s;

data = new T[size];

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

{

data[i] = k;

}

st = copyToStack(data, size);

}

template <class T>

VectorS<T>::VectorS(const VectorS& a)

{

size = a.size;

data = new T[size];

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

{

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

}

st = copyToStack(data, size);

}

template <class T>

VectorS<T>& VectorS<T>::operator=(const VectorS<T>& a)

{

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

size = a.size;

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

data = new T[size];

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

{

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

}

st = copyToStack(data, size);

return \*this;

}

template <class T>

T& VectorS<T>::operator[](int index)

{

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

else cout << "Error! Index > size\n";

}

template <class T>

VectorS<T> VectorS<T>::operator+(const T b)

{

VectorS<T> temp(\*this);

stack<Time> tmp;

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

{

tmp.push(temp.st.top() + b);

temp.st.pop();

}

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

{

temp.st.push(tmp.top());

tmp.pop();

}

temp.data = copyToVec(temp.st);

return temp;

}

template <class T>

ostream& operator<<(ostream& out, VectorS<T>& a)

{

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

out << "\n";

return out;

}

template <class T>

istream& operator>>(istream& in, VectorS<T>& a)

{

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

return in;

}

**Функции для решения задачи 5**

#pragma once

#include <iostream>

#include "VectorSt.h"

#include "Time.h"

using namespace std;

VectorS<Time> MakeVectorS5(int n)

{

VectorS<Time> v(n);

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

{

v[i] = Time(rand() % 60, rand() % 60);

}

v.st = v.copyToStack();

return v;

}

Time GetMax2(stack<Time> s)

{

Time max = s.top();

s.pop();

while (!s.empty())

{

if (s.top() > max)

{

max = s.top();

}

s.pop();

}

return max;

}

Time GetMin2(stack<Time> s, int& pos)

{

Time min = s.top();

s.pop();

pos = 0;

int i = 1;

while (!s.empty())

{

if (s.top() < min)

{

min = s.top();

pos = i;

}

s.pop();

i++;

}

return min;

}

Time GetAverage2(stack<Time> s)

{

int sum = 0;

int size = s.size();

while (!s.empty())

{

sum += s.top().toNumber();

s.pop();

}

int a = sum / size;

return Time(a / 60, a % 60);

}

void AddForEach(VectorS<Time>\* v, Time b)

{

for (int i = 0; i < v->Size(); i++)

{

(\*v)[i] = (\*v)[i] + b;

}

}

void Execute5()

{

auto vec = MakeVectorS5(10);

vec.insert(0, GetMax2(vec.st));

int pos = -1;

GetMin2(vec.st, pos);

vec.erase(pos);

vec = vec + GetAverage2(vec.st);

}

**Ответы на контрольные вопросы**

1. Из каких частей состоит библиотека STL?

STL – Standard Template Library, стандартнаябиблиотекашаблонов

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.

Операцииравенства (==), неравенства (!=), операцияприсваивания (=), 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); //удаляемвключительно

l.erase(beg, end);

18. Заданконтейнер 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. Заданконтейнер 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. Заданконтейнер 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(); }

}