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

Тарасов C.В.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

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

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

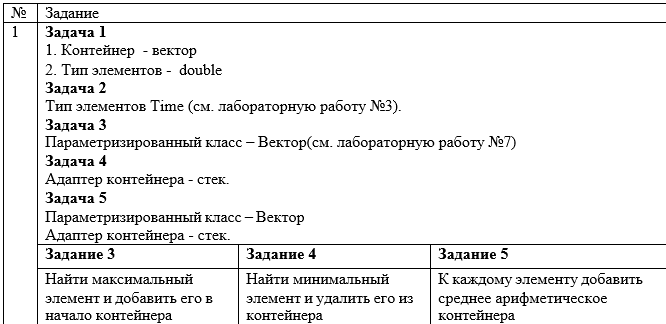
1. Создать параметризированный класс, используя в качестве контейнера последовательный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий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

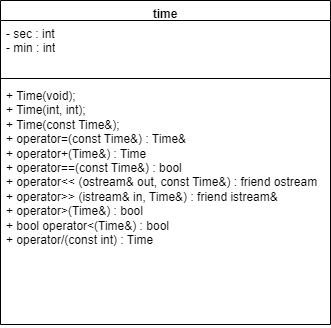
Задача 5

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.

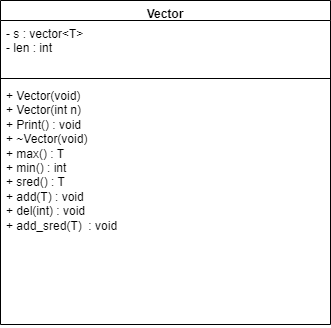


**UML**

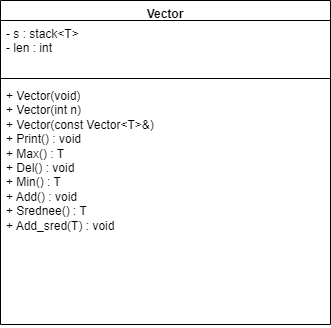
**2**

****

**3**

****

**5**

****

**Функции для решения задачи 1.**

Vec make\_vector(int n)

{

Vec v;//пустой вектор

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

{

double a = rand() % 100 - 50;

v.push\_back(a);//добавляем а в конец вектора

}

return v;//возвращаем вектор как результа работы функции

}

//функция для печати вектора

void print\_vector(Vec v)

{

for (int i = 0;i < v.size();i++) cout << v[i] << " ";

cout << endl;

}

int max(Vec v)

{

int m = v[0];

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

{

if (v[i] > m)

{

m = v[i];

}

}

return m;

}

Vec add(Vec v, int m)

{

v.insert(v.begin(), m);

return v;

}

int min(Vec v)

{

int m = v[0];

int n = 0;

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

{

if (v[i] < m)

{

m = v[i]; n = i;

}

}

return n;

}

Vec del(Vec v, int n)

{

v.erase(v.begin() + n);

return v;

}

double srednee(Vec v)

{

double sum = 0;

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

{

sum += v[i];

}

sum /= v.size();

return sum;

}

Vec add\_srednee(Vec v, double sred)

{

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

{

v[i] += sred;

}

return v;

}

**Основная программа для решения задачи 1**

void main()

{

vector<double> v;

vector<double>::iterator vi = v.begin();

//формирование вектора

int n;

cout << "N?";

cin >> n;

v = make\_vector(n);

print\_vector(v);

//3

int maxi = max(v);

cout << maxi << endl;

v = add(v, maxi);

print\_vector(v);

//4

int mini = min(v);

cout << mini << endl;

v = del(v, mini);

print\_vector(v);

//5

double sredn = srednee(v);

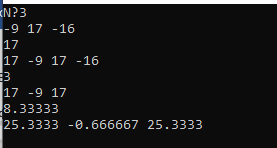
cout << sredn << endl;

v = add\_srednee(v, sredn);

print\_vector(v);

}

**Объяснение результатов работы программы.**



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

class Time

{

private:

int min;

int sec;

public:

Time(void);

Time(int, int);

Time(const Time&);

Time& operator=(const Time&);

//перегруженные операции ввода-вывода

Time operator+(Time&);

bool operator==(const Time&);

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

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

bool operator>(Time&);

bool operator<(Time&);

Time operator/(const int);

};

**Определение перегруженных операций для пользовательского класса.**

Time::Time(void)

{

min = sec = 0;

}

Time::Time(int M, int S)

{

if (S >= 60)

{

min = M + (S / 60);

sec = S % 60;

}

else

{

min = M;sec = S;

}

}

Time::Time(const Time& t)

{

min = t.min; sec = t.sec;

}

Time& Time::operator =(const Time& t)

{

min = t.min; sec = t.sec; return\*this;

}

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

{

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

}

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

{

cout << "\nmin? ";

in >> t.min;

cout << "\nsec? ";

in >> t.sec;

while (t.sec >= 60) {

cout << "Ошибка: количество секунд не может быть 60 или больше. Введите значение еще раз: " << endl;

in >> t.sec;

}

return in;

}

bool Time::operator==(const Time& t)

{

int temp = min \* 60 + sec;

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

return temp == temp2;

}

Time Time::operator+(Time& t)

{

Time temp;

temp.min = ((min + t.min) \* 60 + (sec + t.sec)) / 60;

temp.sec = ((min + t.min) \* 60 + (sec + t.sec)) % 60;

return temp;

}

bool Time::operator>(Time& t)

{

int temp = min \* 60 + sec;

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

return temp > temp2;

}

bool Time::operator<(Time& t)

{

int temp = min \* 60 + sec;

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

return temp < temp2;

}

Time Time::operator/(const int a)

{

int temp = min \* 60 + sec;

int res = temp / a;

Time time;

time.min = res / 60;

time.sec = res % 60;

return time;

}

**Функции для решения задачи 2 .**

Vec make\_vector(int n)

{

Vec v;//пустой вектор

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

{

int a = rand() % 50;

int b = rand() % 60;

Time temp(a, b);

v.push\_back(temp);//добавляем temp в конец вектора

}

return v;//возвращаем вектор как результа работы функции

}

//функция для печати вектора

void print\_vector(Vec v)

{

for (int i = 0;i < v.size();i++) cout << v[i] << " ";

cout << endl;

}

Time max(Vec v)

{

Time max = v[0];

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

{

if (max < v[i]) max = v[i];

}

return max;

}

Vec add(Vec v, Time max)

{

v.insert(v.begin(), max);

return v;

}

int min(Vec v)

{

Time min = v[0];

int n = 0;

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

{

if (min > v[i]) { min = v[i]; n = i; }

}

return n;

}

Vec del(Vec v, int n)

{

v.erase(v.begin() + n);

return v;

}

Time sred(Vec v)

{

Time temp;

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

{

temp = temp + v[i];

}

temp = temp / v.size();

return temp;

}

Vec add\_sred(Vec v, Time temp)

{

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

{

v[i] = v[i] + temp;

}

return v;

}

**Основная программа для решения задачи 2**

void main()

{

srand(time(NULL));

vector<Time> v;

//формирование вектора

int n;

cout << "N?";

cin >> n;

v = make\_vector(n);

print\_vector(v);

//3

Time temp = max(v);

cout << temp << endl;

v = add(v, temp);

print\_vector(v);

//4

int mini = min(v);

cout << mini << endl;

v = del(v, mini);

print\_vector(v);

//5

Time temp1 = sred(v);

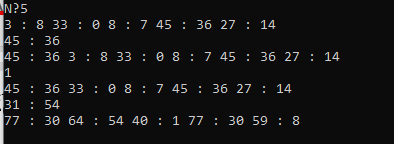
cout << temp1 << endl;

v = add\_sred(v, temp1);

print\_vector(v);

}

**Объяснение результатов работы программы.**



**Описание параметризированного класса для решения задачи 3.**

template<class T> class Vector

{

vector <T> v;//последовательный контейнер для хранения элементов вектора

int len;

public:

Vector(void);//конструктор без параметра

Vector(int n);//конструктор с параметром

void Print();//печать

~Vector(void);//деструктор

T max();

int min();

T sred();

void add(T);

void del(int);

void add\_sred(T);

};

**Определение методов и операций для решения задачи 3.**

template <class T> Vector<T>::Vector()

{

len = 0;

}

//деструктор

template <class T>

Vector<T>::~Vector(void)

{

}

//конструктор с параметром

template <class T> Vector<T>::Vector(int n)

{

T a;

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

{

cin >> a; v.push\_back(a);

}

len = v.size();

}

//печать

template <class T> void Vector<T>::Print()

{

for (int i = 0;i < v.size();i++) cout << v[i] << " ";

cout << endl;

}

template <class T>

T Vector<T>::max()

{

T temp = v[0];

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

{

if (v[i] > temp) temp = v[i];

}

return temp;

}

template <class T>

int Vector<T>::min()

{

T temp = v[0];

int n=0;

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

{

if (v[i] < temp)

{

temp = v[i]; n = i;

}

}

return n;

}

template <class T>

T Vector<T>::sred()

{

T temp = v[0];

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

{

temp = temp + v[i];

}

int n = v.size();

return temp / n;

}

template <class T>

void Vector<T>::add(T a)

{

v.insert(v.begin(), a);

}

template <class T>

void Vector<T>::del(int a)

{

v.erase(v.begin() + a);

}

template <class T>

void Vector<T>::add\_sred(T a)

{

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

{

v[i] = v[i] + a;

}

}

**Основная программа для решения задачи 3**

void main()

{

Vector<Time>vec(5);//создать вектор из 5 элементов

vec.Print();//печать вектора

Time temp;

temp = vec.max();

cout << temp << endl;

vec.add(temp);

vec.Print();

int mini;

mini = vec.min();

cout << mini << endl;

vec.del(mini);

vec.Print();

vec.sred();

temp = vec.sred();

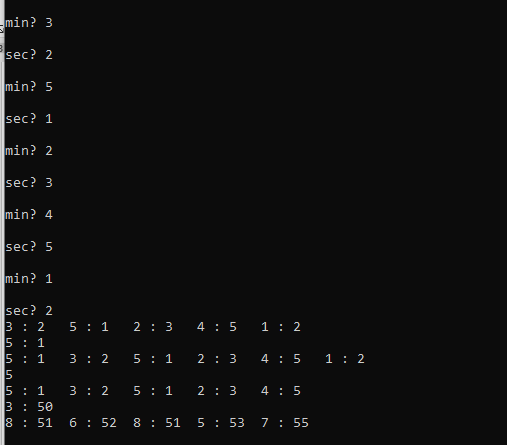
cout << temp << endl;

vec.add\_sred(temp);

vec.Print();

}

**Объяснение результатов работы программы.**



**Функции для решения задачи 4.**

St make\_stack(int n)

{

St s;

Time t;

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

{

cin >> t;//ввод переменной

s.push(t);//добавление ее в стек

}

return s;//вернуть стек как результат функции

}

//копирует стек в вектор

Vec copy\_stack\_to\_vector(St s)

{

Vec v;

while (!s.empty())//пока стек не пустой

{

//добавить в вектор элемент из вершиы стека

v.push\_back(s.top());

s.pop();

}

return v; //вернуть вектор как результат функции

}

//копирует вектор в стек

St copy\_vector\_to\_stack(Vec v)

{

St s;

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

{

s.push(v[v.size()-1-i]);//добавить в стек элемент вектора

}

return s; //вернуть стек как результат функции

}

void print\_stack(St s)

{

v = copy\_stack\_to\_vector(s);

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

{

Time temp = v[v.size() - 1 - i];

cout << temp <<" ";

}

cout << endl;

}

Time max(St s)

{

v = copy\_stack\_to\_vector(s);

int n = s.size();

Time m = s.top();

s.pop();

while (!s.empty())

{

if (m < s.top()) m = s.top();

s.pop();

}

s = copy\_vector\_to\_stack(v);

return m;

}

Time min(St s)

{

v = copy\_stack\_to\_vector(s);

int n = s.size();

int a = 0;

Time m = s.top();

s.pop();

while (!s.empty())

{

if (m > s.top())

m = s.top();

s.pop();

}

s = copy\_vector\_to\_stack(v);

return m;

}

Time sred(St s)

{

v = copy\_stack\_to\_vector(s);

int n = s.size();

Time m = s.top();

s.pop();

while (!s.empty())

{

m = m + s.top();

s.pop();

}

m = m / n;

s = copy\_vector\_to\_stack(v);

return m;

}

void add(St& s,Time time)

{

Vec vv;

Time t;

while (!s.empty())//пока стек не пустой

{

t = s.top();

vv.push\_back(t);

s.pop();

}

vv.push\_back(time);

s = copy\_vector\_to\_stack(vv);

}

void del(St& s, Time time)

{

Time t;

Vec vv;

while (!s.empty())//пока стек не пустой

{

t = s.top();//получаем элемент из вершины стека

//если он не равен минимальному, заносим элемент в вектор

if(!(t==time)) vv.push\_back(t);

s.pop();//удаляем элемент из стека

}

s = copy\_vector\_to\_stack(vv);

}

void add\_sred(St& s, Time time)

{

Vec vv;

Time temp;

while (!s.empty())

{

temp = s.top() + time;

vv.push\_back(temp);

s.pop();

}

s = copy\_vector\_to\_stack(vv);

}

**Основная программа для решения задачи 4**

int main()

{

int n;

cout << "n?";

cin >> n;

s = make\_stack(n);//создать стек

print\_stack(s);//печать стека

Time temp;

temp = max(s);

cout << temp << endl;

add(s, temp);

print\_stack(s);

temp = min(s);

cout << temp << endl;

del(s, temp);

print\_stack(s);

temp = sred(s);

cout << temp << endl;

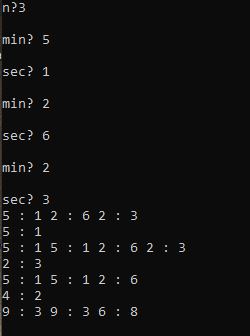
add\_sred(s, temp);

print\_stack(s);

return 0;

}

**Объяснение результатов работы программы.**



**Описание параметризированного класса для решения задачи 5.**

class Vector

{

stack <T> s;//контейнер

int len;//размер контейнера

public:

Vector();//конструктор без параметров

Vector(int n);//конструктор с параметрами

Vector(const Vector<T>&);//конструктор копирования

void Print();

T Max();

void Del();

T Min();

void Add();

T Srednee();

void Add\_sred(T);

};

**Определение методов и операций для решения задачи 5.**

//копирование стека в вектор

template <class T>

vector<T> copy\_stack\_to\_vector(stack<T> s)

{

vector<T> v;

while (!s.empty())

{

v.push\_back(s.top()); s.pop();

}

return v;

}

//копирование вектора в стек

template <class T>

stack<T> copy\_vector\_to\_stack(vector<T> v)

{

stack<T> s;

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

{

s.push(v[i]);

}

return s;

}

//конструктор без параметров

template <class T>

Vector<T>::Vector()

{

len = 0;

}

//конструктор с параметром

template <class T>

Vector<T>::Vector(int n)

{

T a;

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

{

cin >> a;

s.push(a);//добавить в стек элемент а

}

len = s.size();

}

//конструктор копирования

template <class T>

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

{

len = Vec.len;

//копируем значения стека Vec.s в вектор v

vector<T> v=copy\_stack\_to\_vector(Vec.s);

//копируем вектор v в стек s

s=copy\_vector\_to\_stack(v);

}

//печать

template <class T>

void Vector<T>::Print()

{

//копируем стек в вектор

vector<T> v = copy\_stack\_to\_vector(s);

while (!s.empty())//пока стек не пустой

{

cout << s.top() << endl;//вывод элемента в вершине стека

s.pop();//удаляем элемент из вершины

}

//копируем вектор в стек

s=copy\_vector\_to\_stack(v);

}

template <class T>

T Vector<T>::Max()

{

T m = s.top();//m присвоить значение из вершины стека

//в вектор скопировать элементы стека

vector<T> v=copy\_stack\_to\_vector(s);

while(!s.empty())//пока стек не пустой

{

//сравниваем m и элемент в вершине стека

if(s.top()>m) m=s.top();

s.pop();//удаляем элемент из вершины стека

}

s = copy\_vector\_to\_stack(v);//копируем вектор в стек

return m;

}

template <class T>

void Vector<T>::Del()

{

T m = Min();//поиск минимального

vector<T> v;

T t;

while (!s.empty())//пока стек не пустой

{

t = s.top();//получить элемент из вершины стека

//если t не равен минимальному, то добавить его в вектор

if(!(t==m)) v.push\_back(t);

s.pop();//удалить элемент из стека

}

//копируем вектор в стек

s = copy\_vector\_to\_stack(v);

}

template <class T>

T Vector<T>::Min()

{

T m = s.top();

vector<T> v = copy\_stack\_to\_vector(s);

while (!s.empty())

{

if (s.top() < m)m = s.top();

s.pop();

}

s = copy\_vector\_to\_stack(v);

return m;

}

template <class T>

void Vector<T>::Add()

{

vector <T>v;//вспомогательный вектор

T t;

T el = Max();

int i = 1;

v.push\_back(el);

while (!s.empty())//пока стек не пустой

{

t = s.top();//получить элемент из вершины стека

v.push\_back(t);//добавляем t в вектор

s.pop();//удаляем элемент из вершины стека

i++;

}

s = copy\_vector\_to\_stack(v);//копируем вектор в стек

}

template <class T>

T Vector<T>::Srednee()

{

//копируем стек в вектор

vector<T> v = copy\_stack\_to\_vector(s);

int n = 1;//количество элементов в стеке

T sum = s.top();//начальное значение для суммы

s.pop();//удаляем элемент из вершины стека

while (!s.empty())//пока стек не пустой

{

sum = sum + s.top();//добавляем в сумму элемент из вершины стека

s.pop();//удаляем элемент из вершины стека

n++;//увеличиваем количество элементов

}

//копируем вектор в стек

s=copy\_vector\_to\_stack(v);

return sum/n;

}

template <class T>

void Vector<T>::Add\_sred(T el)

{

vector <T>v;//вспомогательный вектор

T t;

T e;

int i = 1;

while (!s.empty())//пока стек не пустой

{

t = s.top();//получить элемент из вершины стека

e = t + el;

v.push\_back(e);//добавляем t в вектор

s.pop();//удаляем элемент из вершины стека

i++;

}

s = copy\_vector\_to\_stack(v);//копируем вектор в стек

}

**Основная программа для решения задачи 5**

void main()

{

Vector<Time>v(3);

v.Print();

cout << endl;

v.Add();

v.Print();

cout << endl;

v.Del();

v.Print();

cout << endl;

Time e;

e = v.Srednee();

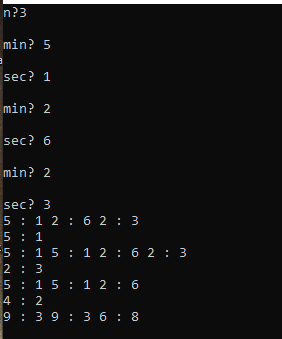
v.Add\_sred(e);

v.Print();

cout << endl;

}

**Объяснение результатов работы программы.**



**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(); }

}