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**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №11**

Дисциплина: «Основы алгоритмизации и программирования»

Семестр 2

Тема: Последовательные контейнеры библиотеки STL

Вариант 6.

Выполнил работу

Студент группы РИС-22-1Б

Поважный В. Е.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**Введение**

Для выполнения лабораторной работы требуется сделать отчет программы. Создание программ – отличный способ практики программирования.

**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.

Указания варианта:

1. Контейнер - список; 2. Тип элементов - double

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.

Указание варианта:

Тип элементов Pair (см. лабораторную работу №3).

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.

Указание варианта:

Параметризированный класс – Список (см. лабораторную работу №7).

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.

Указание варианта:

Адаптер контейнера – очередь с приоритетами.

Задача 5

1. Создать параметризированный класс, используя в качестве контейнера адаптер контейнера.

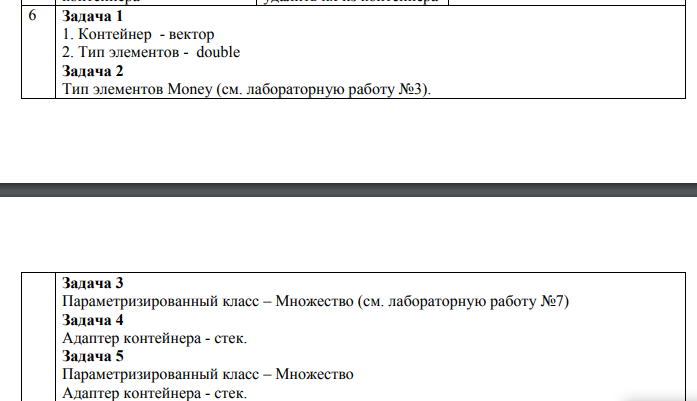
2. Заполнить его элементами.

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 параметризованного класса.



**Код программы**

**Задача 1**

#include <iostream>

#include <vector>

#include <cstdlib>

using namespace std;

typedef vector<float> Vec;

Vec make\_vector(int n)

{

Vec v;

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

v.push\_back(rand() % 100 - 50);

return v;

}

void print\_vector(Vec v)

{

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

cout << v[i] << " ";

cout << endl;

}

int srednee(Vec v)

{

int s = 0;

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

s += v[i];

return s / v.size();

}

void add\_vector(Vec& v, int el, int pos)

{

v.insert(v.begin() + pos, el);

}

int max(Vec v)

{

int m = v[0];

int n = 0;

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

if (m < v[i])

{

m = v[i];

n = i;

}

return n;

}

void del\_vector(Vec& v, int pos)

{

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

}

int min(Vec v)

{

int m = v[0];

int n = 0;

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

if (m > v[i])

{

m = v[i];

n = i;

}

return n;

}

void delenie(Vec& v)

{

int m = min(v);

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

v[i] = v[i] / v[m];

}

void main()

{

try

{

vector<float> v;

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

int n;

cout << "Size: ";

cin >> n;

v = make\_vector(n);

print\_vector(v);

int el = srednee(v);

cout << "Pos: ";

int pos;

cin >> pos;

if (pos > v.size())

throw 1;

add\_vector(v, el, pos);

int n\_del;

cout << "Delete: " << endl;

cin >> n\_del;

del\_vector(v, n\_del);

print\_vector(v);

}

catch (int)

{

cout << "error!";

}

}

**Задача 2**

**#include** <vector>

#include "Header.h"

#include <iostream>

using namespace std;

typedef vector<Money> Vec;

Vec make\_vector(int n)

{

Vec v;

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

{

Money t;

cin >> t;

v.push\_back(t);

}

return v;

}

void print\_vector(Vec v)

{

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

cout << v[i] << " ";

cout << endl;

}

Money srednee(Vec v)

{

int m = 0;

int s = 0;

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

{

m += v[i].get\_rub();

s += v[i].get\_kop();

}

int n = v.size();

Money p;

p.set\_rub(m / n);

p.set\_kop(s / n);

return p;

}

void add\_vector(Vec& v, Money el, int pos)

{

v.insert(v.begin() + pos, el);

}

int max(Vec v)

{

Money m = v[0];

int n = 0;

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

if (m < v[i])

{

m = v[i];

n = i;

}

return n;

}

void del\_vector(Vec& v, int pos)

{

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

}

int min(Vec v)

{

Money m = v[0];

int n = 0;

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

if (m > v[i])

{

m = v[i];

n = i;

}

return n;

}

void delenie(Vec& v)

{

int m = min(v);

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

v[i] = v[i] / v[m];

}

void main()

{

try

{

vector<Money> v;

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

int n;

cout << "Kolvo elementov: ";

cin >> n;

v = make\_vector(n);

print\_vector(v);

Money el = srednee(v);

cout << "middle number pos: ";

int pos;

cin >> pos;

if (pos > v.size())

throw 1;

add\_vector(v, el, pos);

print\_vector(v);

cout << "Max Element: " << v[max(v)] << "\n";

del\_vector(v, max(v));

delenie(v);

print\_vector(v);

}

catch (int)

{

cout << "error!";

}

}

**Задача 3**

**#include "Vector.h"**

#include <iostream>

#include "Money.h"

using namespace std;

void main()

{

Vector<Money>vec(2);

vec.Print();

Money s = vec.Srednee();

cout << "Middle = " << s << endl;

cout << "Pos: ";

int p;

cin >> p;

p = vec.Min();

vec.Add(s, p);

vec.Print();

p = vec.Max();

vec.Del(p);

vec.Print();

vec.Delenie();

vec.Print();

}

**Задача 4**

**#include <iostream>**

#include <stack>

#include <vector>

#include "Money.h"

using namespace std;

typedef stack<Money> St;

typedef vector<Money> Vec;

St make\_stack(int n)

{

St s;

Money t;

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

{

cin >> t;

s.push(t);

}

return s;

}

void print\_stack(St s)

{

Vec v;

while (!s.empty())

{

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

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

s.pop();

}

}

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

{

Vec v;

while (!s.empty())

{

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

s.pop();

}

return v;

}

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

{

St s;

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

s.push(v[i]);

return s;

}

Money Max(St s)

{

Money m = s.top();

Vec 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;

}

void Delete\_from\_stack(St& s)

{

Vec v;

Money t;

while (!s.empty())

{

t = s.top();

s.pop();

}

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

}

Money Min(St s)

{

Money m = s.top();

Vec 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;

}

void Delenie(St& s)

{

Money m = Min(s);

Vec v;

Money t;

while (!s.empty())

{

t = s.top();

v.push\_back(t / m);

s.pop();

}

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

}

void Add\_to\_stack(St& s, Money el)

{

s.push(el);

}

void del\_to\_queue(St& s, int pos)

{

int i = 1;

Vec v;

Money t;

while (!s.empty())

{

t = s.top();

if (i != pos)

v.push\_back(t);

s.pop();

i++;

}

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

}

int main()

{

Money t;

St s;

int n;

cin >> n;

s = make\_stack(n);

print\_stack(s);

cout << "Max: " << Max(s) << endl;

cout << "Remove Pos: ";

int pos;

cin >> pos;

del\_to\_queue(s, pos);

print\_stack(s);

**}**

**Задача 5**

**#include "Money.h"**

#include <iostream>

#include <stack>

#include <vector>

#include "Vector.h"

using namespace std;

int main()

{

Vector<Money> v(3);

v.Print();

cout << "Min: " << v.Min() << endl;

v.Add();

v.Print();

cout << "Remove pos:";

int pos;

cin >> pos;

v.Del(pos);

v.Print();

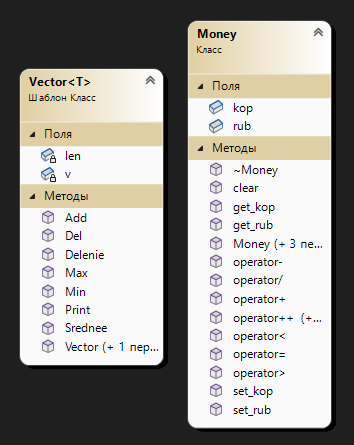
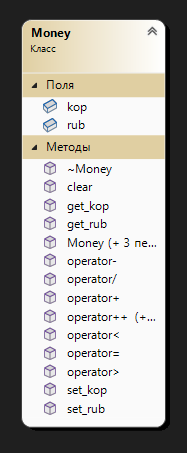
cout << "Summ of min, max and element: " << endl;

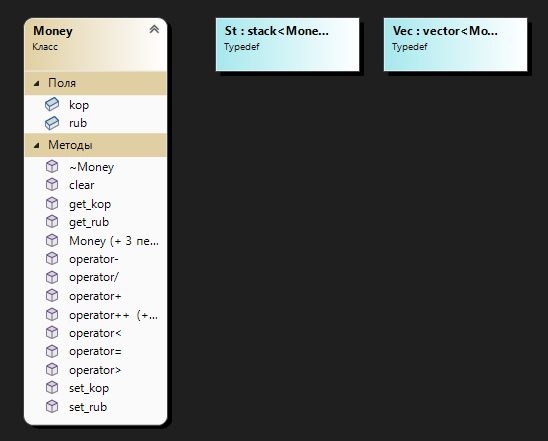
v.Summa();

v.Print();

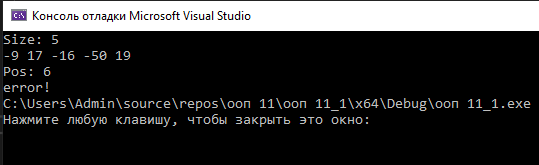
**}**

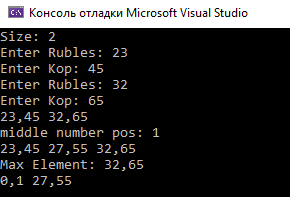
**UML диаграммы**

****

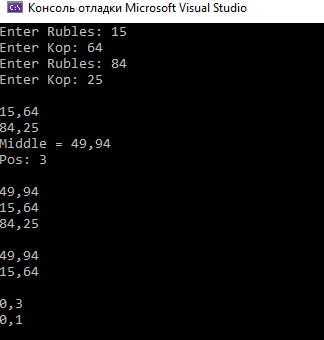
******

**Вывод программы**

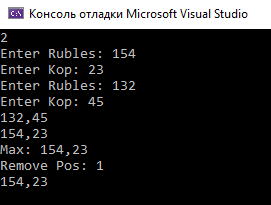
вывод программы в задании под номером 1.

****

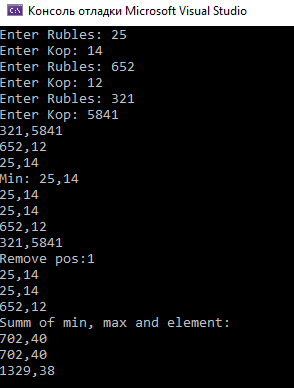
вывод программы в задании под номером 2.



вывод программы в задании под номером 3.

****

вывод программы в задании под номером 4.



вывод программы в задании под номером 5.

**Заключение**

Для решения задачи потребовались знания языка программирования, а конкретнее C++. Программа выполняет те условия, что были указаны в постановке задачи и работает без проблем.

# Ответы на контрольные вопросы

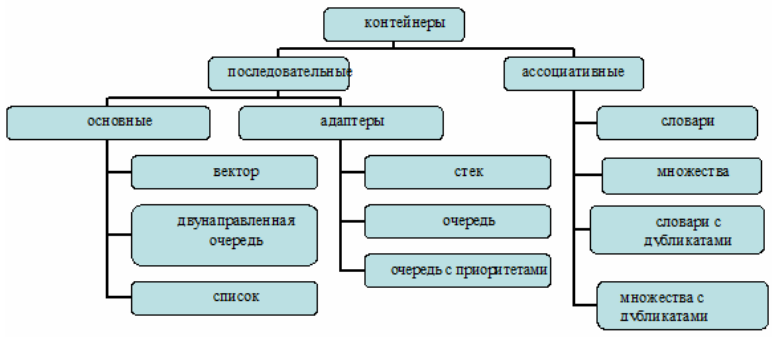
1. Из каких частей состоит библиотека STL?

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(); }

}