Министерство науки и высшего образования РФ

Федеральное бюджетное государственное образовательное учреждение высшего профессионального образования

Пермский национальный исследовательский политехнический университет (ПНИПУ)

Факультет: Электротехнический (ЭТФ)

Направление: 09.03.01 «Информатика и вычислительная техника (Автоматизированные системы обработки информации и управления)»

Кафедра: «Информационные технологии и автоматизированных систем» (ИТАС)

Основы алгоритмизации

Лабораторная работа №11

Последовательные контейнеры библиотеки STL

Студент: Балтаев Э. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Группа: АСУ-20-1бзу

Работу проверил:

доцент кафедры ИТАС

Полякова О. А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_

Пермь 2021г.

Постановка задачи:

Часть 1)

1. Создать последовательный контейнер

2. Заполнить его элементами стандартного типа

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

Часть 2)

1. Создать последовательный контейнер

2. Заполнить его элементами пользовательского типа. 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

Часть 3)

1. Создать параметризованный 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 параметризованного класса

Часть 4)

1. Создать адаптер контейнера

2. Заполнить его элементами пользовательского типа. 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

Часть 5)

1. Создать параметризованный 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 глобальных функций

Задание варианта:

Задача 1

1. Контейнер – вектор

2. Тип элементов – float

Задача 2

Тип элементов Money

Задача 3

Параметризованный класс – Вектор

Задача 4

Адаптер контейнера – очередь

Задача 5

Параметризованный класс – Вектор

Адаптер контейнера – очередь

Задание 3: Найти минимальный элемент и добавить его в конец контейнера

Задание 4: Найти элемент с заданным ключом и удалить его из контейнера

Задание 5: К каждому элементу добавить сумму минимального и максимального элементов контейнера

Определение классов

1) Сумма

#pragma once

#include <iostream>

using namespace std;

class money

{

private:

long rub;

int cop;

public:

money();

money(long r, int c);

money(const money& m);

~money() {}

long get\_rub() { return rub; }

long get\_cop() { return cop; }

void set\_rub(long r);

void set\_cop(int c);

money& operator= (const money& m);

money operator+ (const money& m);

bool operator== (const money& m);

bool operator!= (const money& m);

bool operator< (const money& m);

friend istream& operator>> (istream& in, money& m);

friend ostream& operator<< (ostream& out, const money& m);

};

Определение компонентных функций

1) Сумма

#include "money.h"

money::money()

{

rub = 0;

cop = 0;

}

money::money(long r, int c)

{

set\_rub(r);

set\_cop(c);

}

money::money(const money& m)

{

rub = m.rub;

cop = m.cop;

}

void money::set\_rub(long r)

{

if (r < 0)

rub = 0;

else

rub = r;

}

void money::set\_cop(int c)

{

if (c < 0)

cop = 0;

else {

rub += c / 100;

cop = c % 100;

}

}

money& money::operator=(const money& m)

{

if (&m == this)

return \*this;

rub = m.rub;

cop = m.cop;

return \*this;

}

money money::operator+(const money& m)

{

money temp(rub + m.rub, cop + m.cop);

return temp;

}

bool money::operator==(const money& m)

{

if (rub == m.rub && cop == m.cop)

return true;

return false;

}

bool money::operator!=(const money& m)

{

if (rub != m.rub || cop != m.cop)

return true;

return false;

}

bool money::operator<(const money& m)

{

if (rub < m.rub)

return true;

if (rub == m.rub && cop < m.cop)

return true;

return false;

}

istream& operator>>(istream& in, money& m)

{

long r;

int c;

cout << "Введите сумму через пробел: ";

in >> r; m.set\_rub(r);

in >> c; m.set\_cop(c);

return in;

}

ostream& operator<<(ostream& out, const money& m)

{

out << m.rub << "," << m.cop;

return out;

}

Часть 1)

#pragma once

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<float> first\_make() {

int n;

cout << "Количество элементов? "; cin >> n;

if (n < 1)

throw exception("неверное количество");

cout << "Введите через пробел " << n << " элементов: ";

vector<float> temp;

float val;

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

cin >> val;

temp.push\_back(val);

}

cout << "Успешно" << endl;

return temp;

}

void first\_show(string text, vector<float>& v) {

cout << text;

if (v.size() == 0)

cout << "Пусто" << endl;

for (auto iter = v.begin(); iter != v.end(); iter++)

cout << \*iter << " ";

cout << endl;

}

void first\_insert(vector<float>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

auto minIter = min\_element(v.begin(), v.end());

v.push\_back(\*minIter);

cout << "Успешно" << endl;

}

void first\_delete(vector<float>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

float key;

cout << "Key? "; cin >> key;

auto findIter = find(v.begin(), v.end(), key);

if(findIter == v.end())

cout << "Нет данного значения" << endl;

else {

v.erase(findIter);

cout << "Успешно" << endl;

}

}

void first\_update(vector<float>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

auto minIter = min\_element(v.begin(), v.end());

cout << " Минимальный = " << \*minIter;

auto maxIter = max\_element(v.begin(), v.end());

cout << " Максимальный = " << \*maxIter;

float dif = \*minIter + \*maxIter;

for (auto iter = v.begin(); iter != v.end(); iter++)

\*iter = (\*iter) + dif;

cout << " Успешно" << endl;

}

void first\_exec() {

vector<float> v = first\_make();

first\_show("Исходный: ", v);

cout << "Добавить минимальный в конец: ";

first\_insert(v);

first\_show("После добавления: ", v);

cout << "Удалить с заданным ключом: ";

first\_delete(v);

first\_show("После удаления: ", v);

cout << "Ко всем элементам добавить сумму минимального и максимального: ";

first\_update(v);

first\_show("После изменения: ", v);

}

Часть 2)

#pragma once

#include <iostream>

#include <vector>

#include <algorithm>

#include "money.h"

using namespace std;

vector<money> second\_make() {

int n;

cout << "Количество сумм? "; cin >> n;

if (n < 1)

throw exception("неверное количество");

vector<money> temp;

money val;

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

cin >> val;

temp.push\_back(val);

}

cout << "Успешно" << endl;

return temp;

}

void second\_show(string text, vector<money>& v) {

cout << text;

if (v.size() == 0)

cout << "Пусто" << endl;

for (auto iter = v.begin(); iter != v.end(); iter++)

cout << \*iter << " ";

cout << endl;

}

void second\_insert(vector<money>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

auto minIter = min\_element(v.begin(), v.end());

v.push\_back(\*minIter);

cout << "Успешно" << endl;

}

void second\_delete(vector<money>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

money key;

cout << "Key? "; cin >> key;

auto findIter = find(v.begin(), v.end(), key);

if (findIter == v.end())

cout << "Нет данного значения" << endl;

else {

v.erase(findIter);

cout << "Успешно" << endl;

}

}

void second\_update(vector<money>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

auto minIter = min\_element(v.begin(), v.end());

cout << " Минимальный = " << \*minIter;

auto maxIter = max\_element(v.begin(), v.end());

cout << " Максимальный = " << \*maxIter;

money dif = \*minIter + \*maxIter;

for (auto iter = v.begin(); iter != v.end(); iter++)

\*iter = (\*iter) + dif;

cout << " Успешно" << endl;

}

void second\_exec() {

vector<money> v = second\_make();

second\_show("Исходный: ", v);

cout << "Добавить минимальный в конец: ";

second\_insert(v);

second\_show("После добавления: ", v);

cout << "Удалить с заданным ключом: ";

second\_delete(v);

second\_show("После удаления: ", v);

cout << "Ко всем элементам добавить сумму минимального и максимального: ";

second\_update(v);

second\_show("После изменения: ", v);

}

Часть 3)

#pragma once

template <class T>

class VectorThird

{

private:

vector<T> v;

public:

VectorThird();

~VectorThird();

void make();

void show(string text);

void insert();

void del();

void upd();

};

template<class T>

inline VectorThird<T>::VectorThird()

{

}

template<class T>

inline VectorThird<T>::~VectorThird()

{

v.clear();

}

template<class T>

inline void VectorThird<T>::make()

{

int n;

cout << "Количество элементов? "; cin >> n;

if (n < 1)

throw exception("неверное количество");

T val;

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

cin >> val;

v.push\_back(val);

}

cout << "Успешно" << endl;

}

template<class T>

inline void VectorThird<T>::show(string text)

{

cout << text;

if (v.size() == 0)

cout << "Пусто" << endl;

for (auto iter = v.begin(); iter != v.end(); iter++)

cout << \*iter << " ";

cout << endl;

}

template<class T>

inline void VectorThird<T>::insert()

{

if (v.size() == 0)

throw exception("пустой вектор");

auto minIter = min\_element(v.begin(), v.end());

v.push\_back(\*minIter);

cout << "Успешно" << endl;

}

template<class T>

inline void VectorThird<T>::del()

{

if (v.size() == 0)

throw exception("пустой вектор");

T key;

cout << "Key? "; cin >> key;

auto findIter = find(v.begin(), v.end(), key);

if (findIter == v.end())

cout << "Нет данного значения" << endl;

else {

v.erase(findIter);

cout << "Успешно" << endl;

}

}

template<class T>

inline void VectorThird<T>::upd()

{

if (v.size() == 0)

throw exception("пустой вектор");

auto minIter = min\_element(v.begin(), v.end());

cout << " Минимальный = " << \*minIter;

auto maxIter = max\_element(v.begin(), v.end());

cout << " Максимальный = " << \*maxIter;

T dif = \*minIter + \*maxIter;

for (auto iter = v.begin(); iter != v.end(); iter++)

\*iter = (\*iter) + dif;

cout << " Успешно" << endl;

}

void third\_exec() {

VectorThird<int> v;

v.make();

v.show("Исходный: ");

cout << "Добавить минимальный в конец: ";

v.insert();

v.show("После добавления: ");

cout << "Удалить с заданным ключом: ";

v.del();

v.show("После удаления: ");

cout << "Ко всем элементам добавить сумму минимального и максимального: ";

v.upd();

v.show("После изменения: ");

}

Часть 4)

#pragma once

#include <iostream>

#include <queue>

#include "money.h"

#include <algorithm>

using namespace std;

queue<money> four\_make() {

int n;

cout << "Количество сумм? "; cin >> n;

if (n < 1)

throw exception("неверное количество");

queue<money> temp;

money val;

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

cin >> val;

temp.push(val);

}

cout << "Успешно" << endl;

return temp;

}

void four\_show(string text, queue<money>& v) {

cout << text;

if (v.size() == 0)

cout << "Пусто" << endl;

queue<money> temp;

money m;

while (!v.empty()) {

m = v.front();

cout << m << " ";

temp.push(m);

v.pop();

}

while (!temp.empty()) {

m = temp.front();

v.push(m);

temp.pop();

}

cout << endl;

}

void four\_insert(queue<money>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

queue<money> temp;

money m = v.front(), min = m;

while (!v.empty()) {

m = v.front();

if (m < min)

min = m;

temp.push(m);

v.pop();

}

while (!temp.empty()) {

m = temp.front();

v.push(m);

temp.pop();

}

v.push(min);

cout << "Успешно" << endl;

}

void four\_delete(queue<money>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

money m, key;

cout << "Key? "; cin >> key;

queue<money> temp;

bool ok = false;

while (!v.empty()) {

m = v.front();

if (m != key)

temp.push(m);

else

ok = true;

v.pop();

}

while (!temp.empty()) {

m = temp.front();

v.push(m);

temp.pop();

}

if (ok == true)

cout << "Успешно" << endl;

else

cout << "Нет заданных значений" << endl;

}

void four\_update(queue<money>& v) {

if (v.size() == 0)

throw exception("пустой вектор");

queue<money> temp;

money m = v.front(), min = m, max = m;

while (!v.empty()) {

m = v.front();

if (m < min)

min = m;

if (max < m)

max = m;

temp.push(m);

v.pop();

}

cout << " Минимальный = " << min;

cout << " Максимальный = " << max;

while (!temp.empty()) {

m = temp.front();

m = m + min + max;

v.push(m);

temp.pop();

}

cout << " Успешно" << endl;

}

void four\_exec() {

queue<money> v = four\_make();

four\_show("Исходный: ", v);

cout << "Добавить минимальный в конец: ";

four\_insert(v);

four\_show("После добавления: ", v);

cout << "Удалить с заданным ключом: ";

four\_delete(v);

four\_show("После удаления: ", v);

cout << "Ко всем элементам добавить сумму минимального и максимального: ";

four\_update(v);

four\_show("После изменения: ", v);

}

Часть 5)

#pragma once

#include <iostream>

#include <queue>

#include "money.h"

#include <algorithm>

using namespace std;

template <class T>

class VectorFifth

{

private:

queue<T> v;

public:

VectorFifth();

~VectorFifth();

void make();

void show(string text);

void insert();

void del();

void upd();

};

template<class T>

inline VectorFifth<T>::VectorFifth()

{

}

template<class T>

inline VectorFifth<T>::~VectorFifth()

{

while(!v.empty())

v.pop();

}

template<class T>

inline void VectorFifth<T>::make()

{

int n;

cout << "Количество элементов? "; cin >> n;

if (n < 1)

throw exception("неверное количество");

T val;

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

cin >> val;

v.push(val);

}

cout << "Успешно" << endl;

}

template<class T>

inline void VectorFifth<T>::show(string text)

{

cout << text;

if (v.size() == 0)

cout << "Пусто" << endl;

queue<T> temp;

T m;

while (!v.empty()) {

m = v.front();

cout << m << " ";

temp.push(m);

v.pop();

}

while (!temp.empty()) {

m = temp.front();

v.push(m);

temp.pop();

}

cout << endl;

}

template<class T>

inline void VectorFifth<T>::insert()

{

if (v.size() == 0)

throw exception("пустой вектор");

queue<T> temp;

T m = v.front(), min = m;

while (!v.empty()) {

m = v.front();

if (m < min)

min = m;

temp.push(m);

v.pop();

}

while (!temp.empty()) {

m = temp.front();

v.push(m);

temp.pop();

}

v.push(min);

cout << "Успешно" << endl;

}

template<class T>

inline void VectorFifth<T>::del()

{

if (v.size() == 0)

throw exception("пустой вектор");

T m, key;

cout << "Key? "; cin >> key;

queue<T> temp;

bool ok = false;

while (!v.empty()) {

m = v.front();

if (m != key)

temp.push(m);

else

ok = true;

v.pop();

}

while (!temp.empty()) {

m = temp.front();

v.push(m);

temp.pop();

}

if (ok == true)

cout << "Успешно" << endl;

else

cout << "Нет заданных значений" << endl;

}

template<class T>

inline void VectorFifth<T>::upd()

{

if (v.size() == 0)

throw exception("пустой вектор");

queue<T> temp;

T m = v.front(), min = m, max = m;

while (!v.empty()) {

m = v.front();

if (m < min)

min = m;

if (max < m)

max = m;

temp.push(m);

v.pop();

}

cout << " Минимальный = " << min;

cout << " Максимальный = " << max;

while (!temp.empty()) {

m = temp.front();

m = m + min + max;

v.push(m);

temp.pop();

}

cout << " Успешно" << endl;

}

void fifth\_exec() {

VectorFifth<int> v;

v.make();

v.show("Исходный: ");

cout << "Добавить минимальный в конец: ";

v.insert();

v.show("После добавления: ");

cout << "Удалить с заданным ключом: ";

v.del();

v.show("После удаления: ");

cout << "Ко всем элементам добавить сумму минимального и максимального: ";

v.upd();

v.show("После изменения: ");

}

Определение главного файла

#include <iostream>

#include "first.h"

#include "second.h"

#include "third.h"

#include "four.h"

#include "fifth.h"

int main()

{

system("chcp 1251");

cout << "\n1 часть\n"; first\_exec();

cout << "\n2 часть\n"; second\_exec();

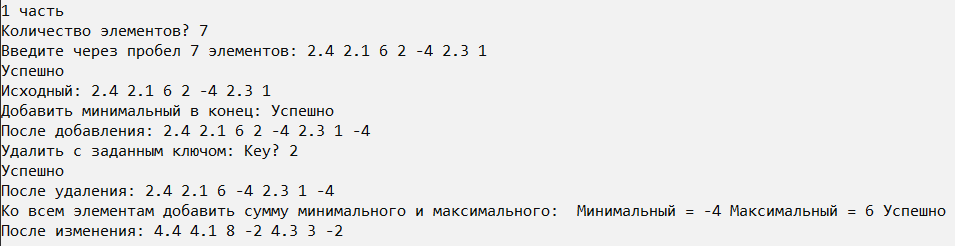
cout << "\n3 часть\n"; third\_exec();

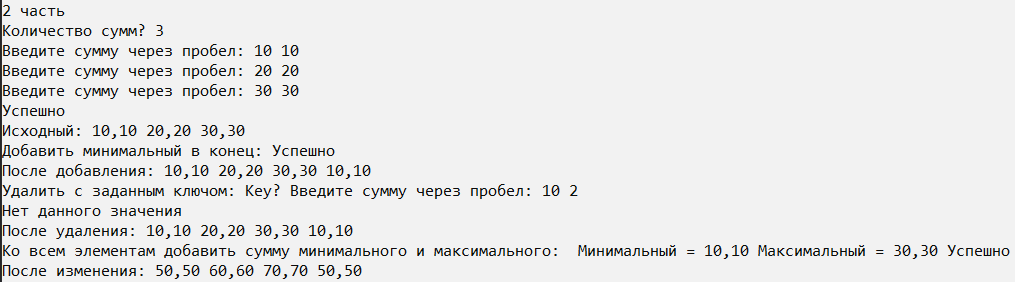
cout << "\n4 часть\n"; four\_exec();

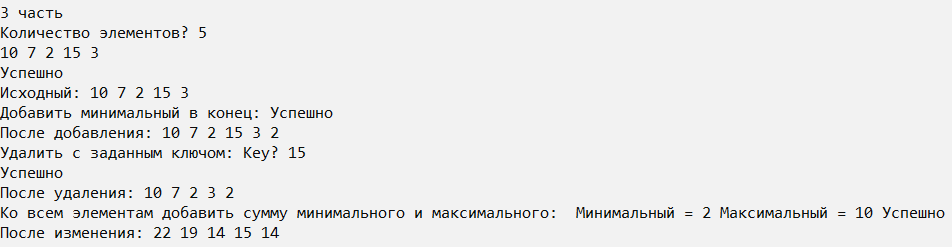
cout << "\n5 часть\n"; fifth\_exec();

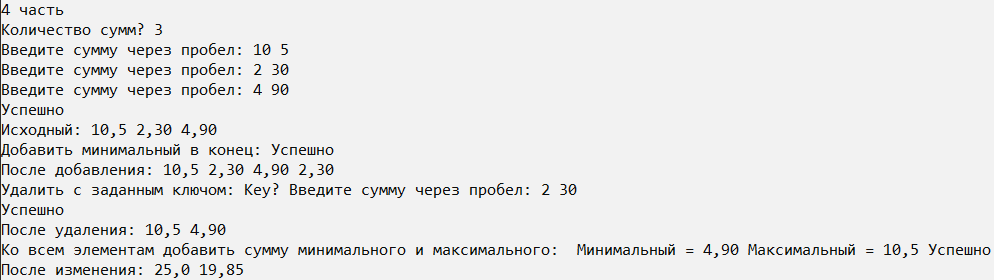
}

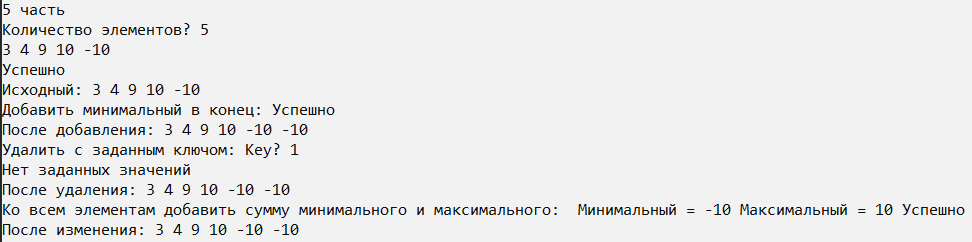
Результаты работы программы:











Ответы на контрольные вопросы:

1) Набор контейнерных классов и набор обобщенных алгоритмов

2) Последовательные и ассоциативные

3) Включить заголовочный файл

4) Обобщение концепции указателей: они ссылаются на элементы контейнера.

5) =, ==/!= , ++/--, +/-, \* (разыменование)

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

7) По типу контейнеров: последовательные и ассоциативные

8) ==/!=, =, clear(), insert(), erase(), empty(), begin(), end(), size(), max\_size()

9) [], вставка в конец – так как не нужно ничего передвигать

10) Вставка и удаление на любую позицию, так как итератор содержит ссылку на предш. и следующую и можно быстро менять ссылки

11) [], вставка и удаление первого/последнего – так как известно начало и конец очереди

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

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

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

15) auto it1 = v.begin()+1; auto it2 = v.begin()+4; v.erase(it1, it2);

16) v.pop\_back();

17) auto it1 = v.begin()+1; auto it2 = v.begin()+4; v.erase(it1, it2);

18) v.pop\_back();

19) auto it1 = v.begin()+1; auto it2 = v.begin()+4; v.erase(it1, it2);

20) v.pop\_back();

21) void print(cont) { for(auto it = cont.begin(); it != cont.end(); it++) cout<<\*it<<endl; }

22) Контейнер построенный на основе последовательных контейнеров, но имеющий специфичные функции: стек и очередь

23) 1 – на основе deque, 2 – на основе list

24) push, pop, top, empty, size

25) push, pop, front, back, empty, size

26) Извлечение происходит максимального элемента а не первого

27) Создать вспомогательный контейнер, записывать туда элементы, которые не равны номеру текущего. В конце переписать из вспомогательного в стек

28) Создать вспомогательный контейнер, записывать туда элементы, которые не равны номеру текущего. В конце переписать из вспомогательного в стек

29-30)

void print(cont) {

list l = cont\_to\_list(cont);

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

cout<<\*it<<endl;

}