**Курсовая работа**

**Разработать приложение для формирования**

**программы концерта по заявкам**

2018 г

# Задание

(ВСТАВИТЬ БЛАНК ЗАДАНИЯ)

Оглавление

[Задание 2](#_Toc518028558)

[ВВЕДЕНИЕ 4](#_Toc518028559)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc518028560)

[1.1 Язык прогаммирования C++ 5](#_Toc518028561)

[1.2 Стандартная библиотека шаблонов языка C++ 5](#_Toc518028562)

[1.2.1 Общие сведения 5](#_Toc518028563)

[1.2.2 Описание используемых в курсовой контейнеров STL 5](#_Toc518028564)

[1.2.3 Описание используемых в курсовой итераторов STL 5](#_Toc518028565)

[1.2.4 Описание используемых в курсовой алгоритмов STL 5](#_Toc518028566)

[1.3 Краткое описание используемой среды разработки 5](#_Toc518028567)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 6](#_Toc518028568)

[2.1 Описание реализованных классов 6](#_Toc518028569)

[2.2 Блок-схемы основных алгоритмов программы 6](#_Toc518028570)

[2.3 Краткое руководство пользователя 6](#_Toc518028571)

[ЗАКЛЮЧЕНИЕ 7](#_Toc518028572)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 8](#_Toc518028573)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ 9](#_Toc518028574)

# ВВЕДЕНИЕ

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Язык прогаммирования C++

## Стандартная библиотека шаблонов языка C++

### Общие сведения

### Описание используемых в курсовой контейнеров STL

### Описание используемых в курсовой итераторов STL

### Описание используемых в курсовой алгоритмов STL

## Краткое описание используемой среды разработки

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Описание реализованных классов

## Блок-схемы основных алгоритмов программы

## Краткое руководство пользователя

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Герберт Шилдт Искусство программирования на C++ / Герберт Шилдт. - М.: БХВ-Петербург, 2005. - 496 c.

2. Герб Саттер Решение сложных задач на С++ / Герб Саттер. - М.: Вильямс, 2015. - 400 c.

3. Александр Чиртик Программирование на C++. Трюки и эффекты / Александр Чиртик. - М.: "Издательство "Питер", 2010. - 352 c.

4. Роберт Лафоре Объектно-ориентированное программирование в С++ / Роберт Лафоре. - М.: Питер, 2015. - 928 c.

5. Айра Пол Объектно-ориентированное программирование на С++ / Айра Пол. - М.: Не указано, Бином, Невский Диалект, 2001. - 464 c.

6. Николаи М. Джосаттис Стандартная библиотека C++. Справочное руководство / Николаи М. Джосаттис. - М.: Вильямс, 2014. – 123 c.

7. ГОСТ 2.105-95 Общие требования к текстовым документам.

# ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

**main.cpp**

#include <clocale>

#include <sstream>

#include "Concert.h"

// Точка входа приложения

int main(int argc, char\*\* argv) {

// Установка кириллицы в консоли

setlocale(LC\_CTYPE, "Russian");

// Получение таймаута из параметра командной строки

int timeout = 120;

if(argc > 1) {

std::istringstream iss;

iss.str(argv[1]);

int value = 0;

if(iss >> value) {

timeout = value;

}

}

// Основной объект приложения

Concert concert(timeout);

return concert.run();

}

**Song.h**

#ifndef SONG\_H

#define SONG\_H

#include <string>

// Песня

class Song

{

public:

// Конструктор

Song(const std::string& name = "",

const std::string& singer = "", size\_t id = 0);

// Методы доступа к полям класса

inline void setName(const std::string& value) {

name = value;

}

inline std::string getName() const {

return name;

}

inline void setSinger(const std::string& value) {

singer = value;

}

inline std::string getSinger() const {

return singer;

}

inline void setId(size\_t value) {

id = value;

}

inline size\_t getId() const {

return id;

}

inline size\_t getVotes() const {

return votes;

}

// Выбор песни (голосование)

inline size\_t vote() {

return ++votes;

}

// Обнулить результаты голосования

inline void resetVotes() {

votes = 0;

}

private:

// Название

std::string name;

// Исполнитель

std::string singer;

// Идентификатор

size\_t id;

// Количество голосов

size\_t votes;

};

#endif

**Song.cpp**

#include "Song.h"

Song::Song(const std::string& name, const std::string& singer, size\_t id)

: name(name), singer(singer), id(id), votes(0)

{

}

**Concert.h**

#ifndef CONCERT\_H

#define CONCERT\_H

#include <string>

#include <map>

#include <vector>

#include <list>

#include <iostream>

#include <iomanip>

#include <algorithm>

#include <chrono>

#include "Song.h"

// Функциональный тип для фильтра песен по названию

class FindSongByName {

public:

FindSongByName(const std::string& field) : field(field) {

}

bool operator()(const Song\* song);

private:

std::string field;

};

// Функциональный тип для фильтра песен по исполнителю

class FindSongBySinger {

public:

FindSongBySinger(const std::string& field) : field(field) {

}

bool operator()(const Song\* song);

private:

std::string field;

};

// Функция-предикат для сортировки песен по убыванию рейтинга

bool CompareSongsRating(const Song\* a, const Song\* b);

// Концерт

class Concert

{

// Исключение - недопустимый порядковый номер

class InvalidOrdinalError {

public:

InvalidOrdinalError(size\_t value) : value(value) {

}

inline size\_t getValue() const {

return value;

}

private:

size\_t value;

};

public:

// Конструктор

Concert(int timeout);

// Деструктор

~Concert();

// Основная рабочая функция системы приёма заявок

// (общее меню для всех пользователей)

int run();

public:

// Псевдонимы для типов

using Users = std::map< std::string, std::string >;

using Songs = std::vector< Song\* > ;

using UserSongs = std::vector< std::pair< const Song\*, Users::iterator > >;

private:

// Зарегистрированные пользователи

Users users;

// Песни, достпуные для выбора пользователями

Songs songs;

// Время запуска системы приём заявок

std::chrono::steady\_clock::time\_point startTime;

// Время до начала концерта

int timeout;

// Выбранные песни

UserSongs userSongs;

// Авторизованный в данный момент пользователь

Users::iterator currUser;

// Функциональный тип для безусловной распечатки песни

struct PrintSong {

void operator()(const Song\* song);

};

// Функциональный тип для условной распечатки песни

class PrintAvailableSong {

public:

PrintAvailableSong(UserSongs& userSongs, Users::iterator currUser)

: userSongs(userSongs), currUser(currUser) {

}

void operator()(const Song\* song);

private:

UserSongs& userSongs;

Users::iterator currUser;

};

private:

// Авторизация существующего пользователя

bool signIn();

// Регистрация нового пользователя

bool signUp();

// Меню для авторизованного пользователя

bool runAuth();

// Выбор песнипользователем

void selectSong();

// Добавление новой песни

void addSong();

// Редактирование информации о песне

void editSong();

// Удаление песни

void delSong();

// Поиск песни по выбранному полю

void findSong();

// Ввод данных пользователя

Users::value\_type inputUser();

// Ввода данных песни

Song\* inputSong();

// Выбор номера песни

size\_t inputIndex();

// Проверка оставшегося времени до концерта

// и его запуск, в случае истечения времени

bool checkTimeout();

// Параметризированный поиск песни

template<class Finder>

Songs::iterator findSong(const Finder& finder) {

return find\_if(songs.begin(), songs.end(), finder);

}

// Распечатка списка песен

template<class Container, class Print>

void printSongs(Container container, Print print) {

if(container.empty())

std::cout << "Список песен пуст" << std::endl;

else {

std::cout << std::setw(4) << "№" << " Рейтинг Песня" << std::endl;

std::for\_each(container.begin(), container.end(), print);

}

}

// Формирование программы концерта по результатам гоолования

template<size\_t ConcertSongsCount>

Songs createConcertProgram() {

Songs result(songs.size());

std::copy(songs.begin(), songs.end(), result.begin());

std::sort(result.begin(), result.end(), CompareSongsRating);

if(ConcertSongsCount < result.size())

result.resize(ConcertSongsCount);

return result;

}

};

#endif

**Concert.cpp**

#include "Concert.h"

using namespace std;

Concert::Concert(int timeout) : timeout(timeout) {

currUser = users.end();

#ifdef \_\_DEBUG\_\_

users["a"] = "a";

users["b"] = "b";

users["c"] = "c";

users["d"] = "d";

users["e"] = "e";

songs.push\_back(new Song("aa", "aaa", 1));

songs.push\_back(new Song("bb", "bbb", 2));

songs.push\_back(new Song("cc", "ccc", 3));

songs.push\_back(new Song("dd", "ddd", 4));

#endif

startTime = chrono::steady\_clock::now();

checkTimeout();

}

Concert::~Concert() {

for(auto song : songs) {

delete song;

}

}

int Concert::run() {

int counter = 0;

int menu = -1;

while (menu != 0) {

cout << "\n\nДОБРО ПОЖАЛОВАТЬ\n\n";

cout << "[0] Завершение работы\n";

cout << "[1] Войти в систему подачи заявок\n";

cout << "[2] Зарегистрироваться в системе подачи заявок\n";

cout << endl << "Ваш выбор: ";

cin.sync();

cin >> menu;

cout << endl;

switch(menu) {

case 1:

if(signIn() && runAuth())

menu = 0;

break;

case 2:

if(signUp() && runAuth())

menu = 0;

break;

}

if(counter++ == 10) {

cout << "emergency exit" << endl;

break;

}

if(menu != 0 && checkTimeout()) {

break;

}

} // while()

cin.get();

return 0;

}

bool Concert::signIn() {

Concert::Users::value\_type user = inputUser();

currUser = users.find(user.first);

if(currUser == users.end() || currUser->second != user.second) {

cout << "Неверное сочетание имени пользователя и пароля" << endl;

currUser = users.end();

return false;

}

cout << "Авторизация пройдена" << endl;

return true;

}

bool Concert::signUp() {

Concert::Users::value\_type user = inputUser();

Users::const\_iterator it = users.find(user.first);

if(it != users.end()) {

cout << "Такой пользователь уже существует" << endl;

return false;

}

currUser = users.insert(user).first;

cout << "Регистрация завершена" << endl;

return true;

}

bool Concert::runAuth() {

bool exit = false;

bool running = true;

int counter = 0;

int menu = -1;

while (running) {

cout << "\n\nГЛАВНОЕ МЕНЮ\n\n";

cout << "[0] Завершение работы\n";

cout << "[1] Выйти из учётной записи\n";

cout << "[2] Общий cписок песен\n";

cout << "[3] Список песен для выбора\n";

cout << "[4] Проголосовать за песню\n";

cout << "[5] Добавить песню\n";

cout << "[6] Редактировать информацию о песне\n";

cout << "[7] Удалить песню\n";

cout << "[8] Найти песню\n";

cout << endl;

cout << "Ваш выбор: ";

cin.sync();

cin >> menu;

switch(menu) {

case 0:

running = false;

exit = true;

break;

case 1:

running = false;

break;

case 2:

printSongs(songs, PrintSong());

break;

case 3:

printSongs(songs, PrintAvailableSong(userSongs, currUser));

break;

case 4:

selectSong();

break;

case 5:

addSong();

break;

case 6:

editSong();

break;

case 7:

delSong();

break;

case 8:

findSong();

break;

}

if(counter++ == 10) {

cout << "emergency exit" << endl;

break;

}

if(checkTimeout()) {

exit = true;

break;

}

} // while()

currUser = users.end();

return exit;

}

void Concert::selectSong() {

try {

printSongs(songs, PrintAvailableSong(userSongs, currUser));

size\_t index = inputIndex();

Song\* song = songs[index];

song->vote();

userSongs.push\_back(UserSongs::value\_type(song, currUser));

printSongs(songs, PrintAvailableSong(userSongs, currUser));

} catch(InvalidOrdinalError&) {

cout << "Недопустимое значение" << endl;

}

}

void Concert::addSong() {

Song\* song = inputSong();

const auto it = find(songs.begin(), songs.end(), song);

if(it != songs.end()) {

cout << "Такая песня уже есть" << endl;

return;

}

songs.push\_back(song);

for(size\_t i = 0; i < songs.size(); i++)

songs[i]->setId(i + 1);

printSongs(songs, PrintSong());

}

void Concert::editSong() {

try {

size\_t index = inputIndex();

songs[index] = inputSong();

printSongs(songs, PrintSong());

} catch(InvalidOrdinalError&) {

cout << "Недопустимое значение" << endl;

}

}

void Concert::delSong() {

try {

size\_t index = inputIndex();

songs.erase(songs.begin() + index);

for(size\_t i = 0; i < songs.size(); i++)

songs[i]->setId(i + 1);

printSongs(songs, PrintSong());

} catch(InvalidOrdinalError&) {

cout << "Недопустимое значение" << endl;

}

}

void Concert::findSong() {

int menu = -1;

string s;

cout << "По какому параметру искать песню?";

cout << "[0] Отменить поиск\n";

cout << "[1] Название\n";

cout << "[2] Исполнитель" << endl;

cout << "Ваш выбор: ";

cin.sync();

cin >> menu;

cout << "Введите значение для поиска: ";

cin >> s;

Songs::iterator it = songs.end();

switch(menu) {

case 1:

it = findSong(FindSongByName(s));

break;

case 2:

it = findSong(FindSongBySinger(s));

break;

default:

return;

}

if(it == songs.end()) {

cout << "Песня не найдена" << endl;

return;

}

cout << "Песня найдена, её номер " << (\*it)->getId() << endl;

}

Concert::Users::value\_type Concert::inputUser() {

string name, pass;

cout << "Введите имя пользователя: ";

cin >> name;

cout << "Введите пароль: ";

cin >> pass;

return Users::value\_type(name, pass);

}

Song\* Concert::inputSong() {

string name, singer;

cout << "Введите название песни: ";

cin >> name;

cout << "Введите исполнителя: ";

cin >> singer;

return new Song(name, singer);

}

size\_t Concert::inputIndex() {

size\_t index;

cout << "Введите номер песни: ";

cin >> index;

if(0 == index || index > songs.size())

throw InvalidOrdinalError(index);

return index - 1;

}

void Concert::PrintSong::operator()(const Song\* song) {

cout << setw(4) << song->getId() << ". " << setw(7) << song->getVotes() << " \"" << song->getName() << "\" (" << song->getSinger() << ")" << endl;

}

void Concert::PrintAvailableSong::operator()(const Song\* song) {

Concert::UserSongs::iterator it = find(userSongs.begin(), userSongs.end(), Concert::UserSongs::value\_type(song, currUser));

if(it == userSongs.end()) {

cout << setw(4) << song->getId() << ". " << setw(7) << song->getVotes() << " \"" << song->getName() << "\" (" << song->getSinger() << ")" << endl;

}

}

bool Concert::checkTimeout() {

bool result = false;

auto now = chrono::steady\_clock::now();

auto delta = chrono::duration\_cast<chrono::seconds>(now - startTime).count();

auto rest = timeout - delta;

if(rest <= 0) {

cout << "Концерт начинается:" << endl;

Songs program = createConcertProgram<2>();

printSongs(program, PrintSong());

result = true;

} else {

cout << "До начала концерта осталось секунд: " << rest << endl;

}

return result;

}

bool FindSongByName::operator()(const Song\* song) {

return song->getName() == field;

}

bool FindSongBySinger::operator()(const Song\* song) {

return song->getSinger() == field;

}

bool CompareSongsRating(const Song\* a, const Song\* b) {

return a->getVotes() > b->getVotes();

}