Файл algorithm.h

```
#ifndef ALGORITHM_H
#define ALGORITHM_H
#include "iterator.h"

class Algorithm {
public:
    //Функция, обменивающая значения объектов,
    //на которые указывают два итератора
    template <class T>
    static void swapIterators(Iterator<T>& itr1, Iterator<T>& itr2);

    //Функция сортировки контейнера пузырьком
    template <class T>
    static void sort(Iterator<T> first, Iterator<T> last);
};

#endif
```

Файл algorithm.cpp

```
#include "algorithm.h"
#include "iterator.cpp"
#include "constiterator.cpp"

//Функция, обменивающая значения объектов,
//на которые указывают два итератора
template <class T>
void Algorithm::swapIterators(Iterator<T>& itr1, Iterator<T>& itr2)
{
    T temp = (*itr1);
    (*itr1) = (*itr2);
    (*itr2) = temp;
}

//Функция сортировки контейнера пузырьком
template <class T>
void Algorithm::sort(Iterator<T> first, Iterator<T> last)
{
    Iterator<T> i, j, k;

    for (i = first; i != last; i++)
        for (j = first, k = j; j != last; j++) {
            ++k;
            //Если значение по j > значения по k
            if (*j > *k)
                //Вызов функции обмена значений
                Algorithm::swapIterators(j, k);
        }
}
```

Файл calendarwindow.h

```
#ifndef CALENDARWINDOW_H
#define CALENDARWINDOW_H

#include <QDialog>
#include "film.h"
```

```
namespace Ui {
class CalendarWindow;
}

class CalendarWindow : public QDialog
{
    Q_OBJECT

public:
    //Конструктор класса CalendarWindow
    explicit CalendarWindow(QWidget *parent = nullptr);
    //Деструктор класса CalendarWindow
    ~CalendarWindow();

private slots:
    //Обработчик изменения выбранной даты, выводит
    //изменения на экран с помощью QLabel
    void on_calendarWidget_selectionChanged();
    //Обработчик нажатия на кнопку chooseFilmButton,
    //осуществляет переход на экран выбора фильмов
    void on_chooseFilmButton_clicked();
    //Обработчик нажатия на кнопку listButton,
    //осуществляет переход к окну с ведомостью проданных
    //билетов на определенный день
    void on_listButton_clicked();
    //Обработчик нажатия на кнопку reportButton,
    //осуществляет переход к окну с отчетом о расписании
    //сеансов со стоимостью билетов
    void on_reportButton_clicked();

private:
    //Указатель на графическую составляющую класса
    Ui::CalendarWindow *ui;
};
#endif
```

Файл calendarwindow.cpp

```
#include "calendarwindow.h"
#include "ui_calendarwindow.h"
#include "filmswindow.h"
#include <QString>
#include "informationwindow.h"

//Конструктор класса CalendarWindow
CalendarWindow::CalendarWindow(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::CalendarWindow)
{
    //Инициализация графического окружения
    ui->setupUi(this);
    //Установка формата текста для dateLabel
    QString format = tr("<p align='center'><span style=' "
                        "font-size:36pt;'>%1</span></p>");
    //Изменение текста в dateLabel на выбранную дату
    ui->dateLabel->setText(format.arg(ui->calendarWidget->
                            selectedDate().toString("dd.MM.yyyy")));
}

//Деструктор класса CalendarWindow
```

```
CalendarWindow::~CalendarWindow()
{
    delete ui;  //очистка дизайна
}

//Обработчик изменения выбранной даты
void CalendarWindow::on_calendarWidget_selectionChanged()
{
    //Установка формата текста для dateLabel
    QString format = tr("<p align='center'><span style=' "
                        "font-size:36pt;'>%1</span></p>");
    //Изменение текста в dateLabel на выбранную дату
    ui->dateLabel->setText(format.arg(ui->calendarWidget->
                           selectedDate().toString("dd.MM.yyyy")));
}

//Обработчик нажатия на кнопку chooseFilmButton,
void CalendarWindow::on_chooseFilmButton_clicked()
{
    //Создание объекта класса FilmsWindow с выбранной датой
    FilmsWindow filmsWindow(ui->calendarWidget->
                            selectedDate().toString("dd.MM.yyyy"));
    this->close();          //Закрытие текущего окна
    filmsWindow.exec();     //Исполнение окна filmsWindow
}

//Обработчик нажатия на кнопку listButton
void CalendarWindow::on_listButton_clicked()
{
    //Создание объекта класса InformationWindow с выбранной датой
    InformationWindow informationWindow(ui->calendarWidget->

selectedDate().toString("dd.MM.yyyy"));
    this->close();              //Закрытие текущего окна
    informationWindow.exec();   //Исполнение окна filmsWindow
}

//Обработчик нажатия на кнопку reportButton
void CalendarWindow::on_reportButton_clicked()
{
    //Создание объекта класса InformationWindow с выбранной датой
    //и передачей флага "REPORT"
    InformationWindow informationWindow(ui->calendarWidget->

selectedDate().toString("dd.MM.yyyy"),
                                        "REPORT");
    this->close();              //Закрытие текущего окна
    informationWindow.exec();   //Исполнение окна filmsWindow
}
```

Файл constiterator.h

```
#ifndef CONSTITERATOR_H
#define CONSTITERATOR_H
#include "Node.h"

template <class T>
class Queue;

template <class T>
class ConstIterator {
public:
```

```
    //Конструктор класса ConstIterator
    ConstIterator();
    //Оператор * возвращает ссылку на элемент
    const T& operator*() const;
    //Перегрузка операторов сравнения
    bool operator==(const ConstIterator& itr) const;
    bool operator!=(const ConstIterator& itr) const;
    //Перегрузка префиксного инкремента
    ConstIterator& operator++();
    //Перегрузка постфиксного инкремента
    ConstIterator operator++(int);
protected:
    //Указатель на текущий узел
    Node<T>* current;
    //Защищенный конструктор
    ConstIterator(Node<T>* ptr);
    //Дружественный класс Queue<T>
    friend class Queue<T>;
};
#endif
```

Файл constiterator.cpp

```
#include "constiterator.h"
#include "exception.h"

//Конструктор класса ConstIterator
template <class T>
ConstIterator<T>::ConstIterator()
{
    current = nullptr;
}

//Защищенный конструктор
template <class T>
ConstIterator<T>::ConstIterator(Node<T>* ptr)
{
    current = ptr;
}

//Оператор * возвращает ссылку на элемент
template <class T>
const T& ConstIterator<T>::operator*() const
{
    if (!current) throw Exception();
    return current->data;
}

//Перегрузка оператора сравнения
template <class T>
bool ConstIterator<T>::operator==(const ConstIterator& itr) const
{
    return (current == itr.current);
}

//Перегрузка оператора сравнения
template <class T>
bool ConstIterator<T>::operator!=(const ConstIterator& itr) const
{
    return (current != itr.current);
}
```

```
//Перегрузка префиксного инкремента
template <class T>
ConstIterator<T>& ConstIterator<T>::operator++()
{
    current = current->next;
    return *this;
}

//Перегрузка постфиксного инкремента
template <class T>
ConstIterator<T> ConstIterator<T>::operator++(int)
{
    ConstIterator<T> copy = *this;
    current = current->next;
    return copy;
}
```

Файл editwindow.h

```
#ifndef EDITWINDOW_H
#define EDITWINDOW_H

#include <QDialog>
#include "seance.h"

namespace Ui {
class EditWindow;
}

class EditWindow : public QDialog
{
    Q_OBJECT
    //Очередь сеансов на определенный день
    Queue<Seance> arrayOfSeances;
    //Выбранная дата
    QString selectedDate;
    //Флаг редактирования/добавления объекта
    QString buttonName;
    //Индекс редактируемого объекта
    int index;

public:
    //Конструктор класса EditWindow
    explicit EditWindow(QString _buttonName, QString _selectedDate,
                        Queue<Seance> _arrayOfSeances = {},
                        int _index = 0, *parent = nullptr);
    //Деструктор класса EditWindow
    ~EditWindow();

private slots:
    //Обработчик нажатия на кнопку addButton, добавляет
    //новый сеанс или изменяет старый в зависимости от флага
    void on_addButton_clicked();

private:
    //Указатель на графическую составляющую класса
    Ui::EditWindow *ui;
};

#endif
```

Файл editwindow.cpp

```
#include "editwindow.h"
#include "ui_editwindow.h"
#include "exception.h"
#include "filmswindow.h"
#include "algorithm.cpp"

//Конструктор класса EditWindow
EditWindow::EditWindow(QString _buttonName, QString _selectedDate,
                       Queue<Seance> _arrayOfSeances, int _index,
                       QWidget *parent) :
    QDialog(parent),
    ui(new Ui::EditWindow)
{
    buttonName = _buttonName;
    selectedDate = _selectedDate;
    index = _index;
    arrayOfSeances = _arrayOfSeances;
    //Инициализация графического окружения
    ui->setupUi(this);
    //Установка текста для объектов класса QLabel
    //в зависимости от флага
    if (buttonName != ui->addButton->text())
    {
        ui->addButton->setText(buttonName);
        ui->nameEdit->setText(arrayOfSeances[index].getName());
        ui->genreEdit->setText(arrayOfSeances[index].getGenre());
        ui->timeEdit->setText(arrayOfSeances[index].getTime());
    }
}

//Деструктор класса EditWindow
EditWindow::~EditWindow()
{
    delete ui; //очистка дизайна
}

//Обработчик нажатия на кнопку addButton
void EditWindow::on_addButton_clicked()
{
    if (Exception::isString(this, ui->nameEdit->text()) &&
            Exception::isString(this, ui->genreEdit->text()) &&
            Exception::isValidTime(this, ui->timeEdit->
                                   text().toStdString()))
    {
        //Если флаг равен "Edit", редактируется объект
        if (ui->addButton->text() == "Edit")
        {
            arrayOfSeances[index].setName(ui->nameEdit->text());
            arrayOfSeances[index].setGenre(ui->genreEdit->text());
            arrayOfSeances[index].setTime(ui->timeEdit->text());
        }
        else    //Если флаг равен "Add", добавляется новый объект
        {
            Seance newSeance;
            newSeance.setName(ui->nameEdit->text());
            newSeance.setGenre(ui->genreEdit->text());
            newSeance.setDate(selectedDate);
            newSeance.setTime(ui->timeEdit->text());
```

```
            arrayOfSeances.push(newSeance);
        }

        //Сортировка очереди сеансов
        Algorithm::sort(arrayOfSeances.begin(), arrayOfSeances.end());
        //Перезапись сеансов в файл
        FilmsWindow::rewriteFile(this, arrayOfSeances, selectedDate);

        //Создание объекта класса FilmsWindow с выбранной датой
        FilmsWindow filmsWindow(selectedDate);
        this->close();          //Закрытие текущего окна
        filmsWindow.exec();     //Исполнение окна filmsWindow
    }
}
```

Файл exception.h

```
#ifndef EXCEPTION_H
#define EXCEPTION_H
#include <QException>
#include <QString>
#include "filmswindow.h"
#include <string>
#include <iostream>
#include <QTableWidget>

class Exception: public QException
{
    //Сообщение об ошибке
    QString message;
public:
    //Конструктор класса Exception
    Exception(QString _message = "");
    //Деструктор класса Exception
    ~Exception() noexcept {}
    //Функция получения сообщения об ошибке
    virtual QString Message();
    //Функция проверки, открыт ли файл для чтения
    static bool isFileOpenForReading(QWidget *parent, ifstream& ifile,
                                     const string& fileName);
    //Функция проверки, открыт ли файл для записи
    static bool isFileOpenForWriting(QWidget *parent, ofstream& ofile,
                                     const string& fileName);
    //Функция проверки на ошибки таблиц
    static bool tableProblems(QWidget *parent, QTableWidget& tableWidget);
    //Функция проверки ввода строки
    static bool isString(QWidget *parent, const QString& str,
                         string language = "ENG");
    //Функция проверки ввода времени
    static bool isValidTime(QWidget *parent, const string& str);
};
#endif
```

Файл exception.cpp

```
#include "exception.h"
#include <QFile>
#include <QErrorMessage>
#include <QMessageBox>
#include <regex>

//Конструктор класса Exception
Exception::Exception(QString _message)
{
    message = _message;
}

//Функция получения сообщения об ошибке
QString Exception::Message()
{
    return message;
}

//Проверка, открыт ли файл для чтения
bool Exception::isFileOpenForReading(QWidget *parent,
                                     ifstream& ifile,
                                     const string& fileName)
{
    bool fileProblem = false;   //Создание флага ошибки
    try     //Проверка, открыт ли файл
    {
        ifile.open(fileName);
        if (!ifile.is_open())
            //Генерация исключения
           throw Exception("File cannot be opened!");
    }
    catch (Exception exception) //Обработчик исключений
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             exception.Message());
        fileProblem = true;
    }
    catch (...)     //Абсолютный обработчик
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             "Some other exception.");
        fileProblem = true;
    }
    return fileProblem; //Возврат флага ошибки
}

//Проверка, открыт ли файл для записи
bool Exception::isFileOpenForWriting(QWidget *parent,
                                     ofstream& ofile,
                                     const string& fileName)
{
    bool fileProblem = false;   //Создание флага ошибки
    try     //Проверка, открыт ли файл
    {
        ofile.open(fileName);
        if (!ofile.is_open())
            //Генерация исключения
           throw Exception("File cannot be opened!");
    }
    catch (Exception exception) //Обработчик исключений
```

```
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             exception.Message());
        fileProblem = true;
    }
    catch (...)     //Абсолютный обработчик
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             "Some other exception.");
        fileProblem = true;
    }
    return fileProblem; //Возврат флага ошибки
}

//Проверка на ошибки с таблицей
bool Exception::tableProblems(QWidget *parent,
                              QTableWidget& tableWidget)
{
    bool tableProblem = false;  //Создание флага ошибки
    try
    {
        //Проверка, пустая ли таблица
        if (tableWidget.rowCount() == 0)
            //Генерация исключения
            throw Exception("Table is empty!\n"
                            "Please, add something "
                            "or select another date.");
        //Проверка, выбран ли элемент таблицы
        else if (tableWidget.selectedItems().size() == 0)
            //Генерация исключения
            throw Exception("You didn't select any film!");
    }
    catch (Exception exception) //Обработчик исключений
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             exception.Message());
        tableProblem = true;
    }
    catch (...)     //Абсолютный обработчик
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             "Some other exception.");
        tableProblem = true;
    }
    return tableProblem;    //Возврат флага ошибки
}

//Проверка на ввод строки
bool Exception::isString(QWidget *parent,
                         const QString& str, string language)
{
    bool isString = true;   //Создание флага ошибки
    try
    {
        int leftBorder = 0, rightBorder = 0;
        //Установка крайних значений
        if (language == "ENG") {
            leftBorder = 'A';
            rightBorder = 'z';
        }
```

```
        //Проверка, пуста ли строка
        if (str.length() == 0)
            //Генерация исключения
            throw Exception("String is empty!");
        //Проверка на правильно введенную строку
        for (int i = 0; i < str.length(); i++)
        {
            if ((str[i] < leftBorder || str[i] > rightBorder) &&
                    str[i] != ' ' && str[i] != ':' &&
                    (str[i] < '0' || str[i] > '9'))
                //Генерация исключения
                throw Exception("Invalid symbols!");
        }

        //Проверка на строку без букв
        for(int i = 0; i < str.length(); i++)
        {
            if (str[i] != ' ' && str[i] != ':' &&
                    (str[i] < '0' || str[i] > '9'))
                break;
            else if (i + 1 == str.length())
                //Генерация исключения
                throw Exception("Invalid symbols!");
        }
    }
    catch (Exception exception) //Обработчик исключений
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             exception.Message());
        isString = false;
    }
    catch (...)     //Абсолютный обработчик
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             "Some other exception.");
        isString = false;
    }
    return isString;    //Возврат флага ошибки
}

//Проверка на ввод времени
bool Exception::isValidTime(QWidget *parent, const string& str)
{
    bool isValid = true;    //Создание флага ошибки
    //Задание стиля ввода времени
    std::regex validTime("([0-9]|0[0-9]|1[0-9]|"
                         "2[0-3]):([0-5][0-9])");
    try     //Проверка, праивльно ли введено время
    {
        if (!std::regex_match(str, validTime))
            throw Exception("Not a valid time!");
    }
    catch (Exception exception) //Обработчик исключений
    {
        //Вывод предупреждения на экран
        QMessageBox::warning(parent,"Attention!",
                             exception.Message());
        isValid = false;
    }
    catch (...)     //Абсолютный обработчик
    {
        //Вывод предупреждения на экран
```

```
        QMessageBox::warning(parent,"Attention!",
                             "Some other exception.");
        isValid = false;
    }
    return isValid; //Возврат флага ошибки
}
```

Файл film.h

```
#ifndef FILM_H
#define FILM_H
#include <fstream>
#include <QString>
using namespace std;

class Film
{
    //Название фильма
    QString name;
    //Жанр фильма
    QString genre;
    //Статическое поле с именем файла с фильмами
    static QString fileName;
public:
    //Конструктор класса Film
    Film(QString _name = "", QString _genre = "");
    //Деструктор класса Film
    virtual ~Film();
    //Перегрузка считывания из файла
    friend ifstream & operator >> (ifstream &ifile, Film &film);
    //Перегрузка записи в файл
    friend ofstream & operator << (ofstream &ofile, Film &film);
    //Функции получения и установки значений полей класса
    QString getName() const;
    void setName(const QString &value);
    QString getGenre() const;
    void setGenre(const QString &value);
    static QString getFileName();
    static void setFileName(const QString &value);
    //Перегрузка сравнения объектов
    friend bool operator>(Film obj1, Film obj2);
};
#endif
```

Файл film.cpp

```
#include "film.h"
QString Film::fileName;
//Максимальная длина строки
#define MAX_STRING_LENGHT 50

//Конструктор класса Film
Film::Film(QString _name, QString _genre)
{
    name = _name;
    genre = _genre;
}
```

```
//Деструктор класса Film
Film::~Film() {}

//Перегрузка считывания из файла
ifstream &operator >> (ifstream &ifile, Film &film)
{
    char* temp;   //создание промежуточной строки
    temp = new char[MAX_STRING_LENGHT];
    //Считывание имени
    ifile.getline(temp, MAX_STRING_LENGHT, ',');
    film.name = temp;
    //Считывание жанра
    ifile.getline(temp, MAX_STRING_LENGHT, ',');
    film.genre = temp;
    //Удаление промежуточной строки
    delete[] temp;
    return ifile;
}

//Перегрузка записи в файл
ofstream &operator << (ofstream &ofile, Film &film)
{
    //Запись имени
    ofile.write(film.name.toUtf8(), film.name.size());
    ofile.write(",", 1);
    //Запись жанра
    ofile.write(film.genre.toUtf8(), film.genre.size());
    ofile.write(",", 1);
    return ofile;
}

//Перегрузка сравнения по имени
bool operator>(Film obj1, Film obj2)
{
    if (obj1.getName().compare(obj2.getName()) > 0)
        return true;
    else return false;
}
//Получение значения поля genre
QString Film::getGenre() const
{
    return genre;
}
//Установка значения поля genre
void Film::setGenre(const QString &value)
{
    genre = value;
}
//Получение значения поля name
QString Film::getName() const
{
    return name;
}
//Установка значения поля name
void Film::setName(const QString &value)
{
    name = value;
}
//Получение значения поля fileName
QString Film::getFileName()
{
    return fileName;
}
```

```
//Получение значения поля fileName
void Film::setFileName(const QString &value)
{
    fileName = value;
}
```

Файл filmswindow.h

```
#ifndef FILMSWINDOW_H
#define FILMSWINDOW_H

#include "ticket.h"
#include "film.h"
#include "queue.h"
#include "queue.cpp"
#include <QStack>

namespace Ui {
class FilmsWindow;
}

class FilmsWindow : public QDialog
{
    Q_OBJECT
    //Очередь сеансов на определенный день
    Queue<Seance> arrayOfSeances;
    //Выбранная дата
    QString selectedDate;
public:
    //Конструктор класса FilmsWindow
    explicit FilmsWindow(QString _selectedDate = "",
                         QWidget *parent = nullptr);
    //Деструктор класса FilmsWindow
    ~FilmsWindow();
    //Функция создания таблицы с сеансами
    //путем считывания информации из файла
    void configure();
    //Функция перезаписи информации в файл
    static void rewriteFile(QWidget *parent,
                            Queue<Seance> arrayOfSeances,
                            QString selectedDate);
    //Функции получения и установки значений полей
    Queue<Seance> getArrayOfSeances() const;
    void setArrayOfSeances(const Queue<Seance> &value);
private slots:
    //Обработчик изменения выбранного сеанса, выводит
    //изменения на экран с помощью QLabel
    void on_filmsTableWidget_itemSelectionChanged();
    //Обработчик нажатия на кнопку choosePlacesButton,
    //осуществляет переход на экран выбора мест
    void on_choosePlacesButton_clicked();
    //Обработчик нажатия на кнопку backButton,
    //осуществляет возврат на предыдущий экран
    void on_backButton_clicked();
    //Обработчик нажатия на кнопку addButton, осуществляет
    //переход на экран добавления нового сеанса
    void on_addButton_clicked();
    //Обработчик нажатия на кнопку deleteButton, осуществляет
    //удаляет выбранный сеанс
    void on_deleteButton_clicked();
    //Обработчик нажатия на кнопку editButton, осуществляет
    //переход на экран редактирования выбранного фильма
```

```
    void on_editButton_clicked();
    //Обработчик нажатия на кнопку addButton,
    //осуществляет отмену последнего действия
    void on_undoButton_clicked();
private:
    //Указатель на графическую составляющую класса
    Ui::FilmsWindow *ui;
};

#endif
```

Файл filmswindow.cpp

```
#include "filmswindow.h"
#include "ui_filmswindow.h"
#include "placeswindow.h"
#include "exception.h"
#include <QMessageBox>
#include "calendarwindow.h"
#include "editwindow.h"
#include "algorithm.cpp"
//Объявление стека для отмены последнего действия
QStack< Queue<Seance> > Seance::undoStack;

//Конструктор класса FilmsWindow
FilmsWindow::FilmsWindow(QString _selectedDate, QWidget *parent):
    QDialog(parent),
    ui(new Ui::FilmsWindow)
{
    selectedDate = _selectedDate;
    //Инициализация графического окружения
    ui->setupUi(this);
    //Установка параметров выбора элемента для таблицы
    ui->filmsTableWidget->
            setEditTriggers(QAbstractItemView::NoEditTriggers);
    ui->filmsTableWidget->horizontalHeader()->
            setSectionResizeMode(QHeaderView::Stretch);
    ui->filmsTableWidget->
            setSelectionBehavior(QAbstractItemView::SelectRows);
    ui->filmsTableWidget->
            setSelectionMode(QAbstractItemView::SingleSelection);
    //Функция создания таблицы с сеансами
    configure();
}

//Деструктор класса FilmsWindow
FilmsWindow::~FilmsWindow()
{
    delete ui;  //очистка дизайна
}

//Получение значения поля arrayOfSeances
Queue<Seance> FilmsWindow::getArrayOfSeances() const
{
    return arrayOfSeances;
}
//Установка значения поля arrayOfSeances
void FilmsWindow::setArrayOfSeances(const Queue<Seance> &value)
{
    arrayOfSeances = value;
}
```

```
//Функция создания таблицы с сеансами
void FilmsWindow::configure()
{
    ifstream ifile; //Создание файла для считывания
    //Проверка, открыт ли файл для чтения
    if (!Exception::isFileOpenForReading(this, ifile,
                                         Film::getFileName().toStdString()))
    {   //Пока не конец файла, считывание сеансов в контейнер
        while (!ifile.eof())
        {
            Seance newSeance;
            ifile >> newSeance;
            if (!newSeance.getDate().compare(selectedDate))
                arrayOfSeances.push(newSeance);
        }
        ifile.close();  //Закрытие файла
        //Сортировка контейнера
        Algorithm::sort(arrayOfSeances.begin(), arrayOfSeances.end());
        //Создание заголовка таблицы
        ui->filmsTableWidget->setColumnCount(3);
        QStringList list;
        list << "Film's name" << "Genre" << "Time";
        ui->filmsTableWidget->setHorizontalHeaderLabels(list);
        list.clear();
        //Добавление сеансов в таблицу из контейнера
        for(int i = 0; i < arrayOfSeances.getSize(); i++)
        {
            ui->filmsTableWidget->insertRow(ui->filmsTableWidget-
>rowCount());
            int numRow = ui->filmsTableWidget->rowCount() - 1;
            ui->filmsTableWidget->setItem(numRow, 0, new QTableWidgetItem(
                                              arrayOfSeances[i].getName()));
            ui->filmsTableWidget->setItem(numRow, 1, new QTableWidgetItem(
                                              arrayOfSeances[i].getGenre()));
            ui->filmsTableWidget->setItem(numRow, 2, new QTableWidgetItem(
                                              arrayOfSeances[i].getTime()));
        }
    }
}

//Перезапись сеансов в файл
void FilmsWindow::rewriteFile(QWidget *parent, Queue<Seance> arrayOfSeances,
                              QString selectedDate)
{
    ifstream ifile; //Создание файла для чтения
    //Создание массива всех сеансов в файле
    Queue<Seance> arrayOfAllSeances;
    //Проверка, открыт ли файл для считвывания
    if (!Exception::isFileOpenForReading(parent, ifile,
                                         Film::getFileName().toStdString()))
    {   //Пока не конец файла, считывание сеансов в контейнер
        while (!ifile.eof())
        {
            Seance newSeance;
            ifile >> newSeance;
            if (newSeance.getDate().compare(selectedDate) &&
                    newSeance.getDate() != "")
                arrayOfAllSeances.push(newSeance);
        }
        ifile.close();  //Закрытие файла
    }
    ofstream ofile; //Создание файла для записи
    //Проверка, открыт ли файл для записи
    if (!Exception::isFileOpenForWriting(parent, ofile,
```

```
                                                Film::getFileName().toStdString()))
    {   //Запись сеансов в файл из двух контейнеров
        for (int i = 0; i < arrayOfAllSeances.getSize(); i++)
        {
            ofile << arrayOfAllSeances[i];
        }
        for (int i = 0; i < arrayOfSeances.getSize(); i++)
        {
            ofile << arrayOfSeances[i];
        }
        ofile.close();  //Закрытие файла
    }
}

//Обработчик изменения выбранного сеанса
void FilmsWindow::on_filmsTableWidget_itemSelectionChanged()
{   //Если сеанс выбран
    if (ui->filmsTableWidget->selectedItems().size() > 0)
    {   //Вывод информации о выбранном сеансе на экран
        QItemSelectionModel *select = ui->filmsTableWidget->selectionModel();
        QString nameFormat = tr("<p align='center'><span style=' "
                                "font-size:24pt;'>%1</span></p>");
        QString otherFormat = tr("<p align='center'><span style=' "
                                 "font-size:18pt;'>%1</span></p>");
        ui->nameLabel->setText(nameFormat.arg(select->selectedRows(0).
                                              first().data().toString()));
        ui->genreLabel->setText(otherFormat.arg(select->selectedRows(1).
                                                first().data().toString()));
        ui->timeLabel->setText(otherFormat.arg(select->selectedRows(2).
                                               first().data().toString()));
    }
}

//Обработчик нажатия на кнопку choosePlacesButton
void FilmsWindow::on_choosePlacesButton_clicked()
{   //Проверка таблицы на проблемы
    if (!Exception::tableProblems(this, *ui->filmsTableWidget))
    {   //Установка индекса выбранного сеанса
        int index = ui->filmsTableWidget->selectionModel()->
                selectedIndexes().first().row();
        //Создание элемента класса placesWindow
        PlacesWindow placesWindow(arrayOfSeances, index);
        this->close();          //Закрытие текущего окна
        placesWindow.exec();    //Исполнение окна placesWindow
    }
}

//Обработчик нажатия на кнопку backButton
void FilmsWindow::on_backButton_clicked()
{   //Перезапись файла
    rewriteFile(this, arrayOfSeances, selectedDate);
    //Создание элемента класса CalendarWindow
    CalendarWindow calendarWindow;
    this->close();          //Закрытие текущего окна
    calendarWindow.exec();  //Исполнение окна calendarWindow
}

//Обработчик нажатия на кнопку addButton
void FilmsWindow::on_addButton_clicked()
{
    //Запись текущего массива сеансов
    //в стек отмены последнего действия
    Seance::undoStack.push(arrayOfSeances);
    //Создание элемента класса EditWindow
```

```
    EditWindow editWindow(ui->addButton->text(), selectedDate,
                          arrayOfSeances);
    this->close();      //Закрытие текущего окна
    editWindow.exec();  //Исполнение окна editWindow
}

//Обработчик нажатия на кнопку deleteButton
void FilmsWindow::on_deleteButton_clicked()
{   //Проверка таблицы на проблемы
    if (!Exception::tableProblems(this, *ui->filmsTableWidget))
    {   //Запись текущего массива сеансов
        //в стек отмены последнего действия
        Seance::undoStack.push(arrayOfSeances);
        //Установка индекса выбранного сеанса
        int index = ui->filmsTableWidget->selectionModel()->
                selectedIndexes().first().row();
        //Удаление сеанса из контейнера
        arrayOfSeances.eraseByIndex(index);
        //Удаление сеанса из таблицы
        ui->filmsTableWidget->removeRow(index);
        //Перезапись сеансов в файл
        rewriteFile(this, arrayOfSeances, selectedDate);
    }
}

//Обработчик нажатия на кнопку editButton
void FilmsWindow::on_editButton_clicked()
{   //Проверка таблицы на проблемы
    if (!Exception::tableProblems(this, *ui->filmsTableWidget))
    {   //Запись текущего массива сеансов
        //в стек отмены последнего действия
        Seance::undoStack.push(arrayOfSeances);
        //Создание элемента класса EditWindow
        EditWindow editWindow(ui->editButton->text(), selectedDate,
                              arrayOfSeances,
                              ui->filmsTableWidget->selectionModel()->
                              selectedIndexes().first().row());
        this->close();      //Закрытие текущего окна
        editWindow.exec();  //Исполнение окна editWindow
    }
}

//Обработчик нажатия на кнопку undoButton
void FilmsWindow::on_undoButton_clicked()
{   //Если стек пустой, вывод предупреждения на экран
    if (Seance::undoStack.empty())
        QMessageBox::warning(this,"Attention!",
                             "You haven't done anything yet!");
    else
    {   //Создание контейнера сеансов из последнего
        //объекта стека
        arrayOfSeances = Seance::undoStack.top();
        //Удаление последнего объекта стека
        Seance::undoStack.pop();
        //Перезапись сеансов в файл
        rewriteFile(this, arrayOfSeances, selectedDate);
        //Перезапись сеансов в таблицу
        ui->filmsTableWidget->setRowCount(0);
        for(int i = 0; i < arrayOfSeances.getSize(); i++)
        {
            ui->filmsTableWidget->insertRow(ui->
                                            filmsTableWidget->rowCount());
            int numRow = ui->filmsTableWidget->rowCount() - 1;
            ui->filmsTableWidget->setItem(numRow, 0, new QTableWidgetItem(
```

```
                                                arrayOfSeances[i].getName()));
            ui->filmsTableWidget->setItem(numRow, 1, new QTableWidgetItem(
                                                arrayOfSeances[i].getGenre()));
            ui->filmsTableWidget->setItem(numRow, 2, new QTableWidgetItem(
                                                arrayOfSeances[i].getTime()));
        }
    }
}
```

Файл informationwindow.h

```
#ifndef INFORMATIONWINDOW_H
#define INFORMATIONWINDOW_H

#include <QDialog>
#include "seance.h"

namespace Ui {
class InformationWindow;
}

class InformationWindow : public QDialog
{
    Q_OBJECT
    //Выбранная дата
    QString selectedDate;
    //Очередь сеансов на определенный день
    Queue<Seance> arrayOfSeances;
public:
    //Конструктор класса InformationWindow
    explicit InformationWindow(QString _selectedDate,
                               QString config = "LIST",
                               QWidget *parent = nullptr);
    //Деструктор класса InformationWindow
    ~InformationWindow();
    //Функция создания таблицы с сеансами
    //путем считывания информации из файла
    void configure(QString config);
private slots:
    //Обработчик нажатия на кнопку backButton,
    //осуществляет возврат на предыдущий экран
    void on_backButton_clicked();
private:
    //Указатель на графическую составляющую класса
    Ui::InformationWindow *ui;
};

#endif
```

Файл informationwindow.cpp

```
#include "informationwindow.h"
#include "ui_informationwindow.h"
#include "calendarwindow.h"
#include "exception.h"
#include "algorithm.cpp"

//Конструктор класса InformationWindow
InformationWindow::InformationWindow(QString _selectedDate,
                                     QString config, QWidget *parent) :
```

```
    QDialog(parent),
    ui(new Ui::InformationWindow)
{
    selectedDate = _selectedDate;
    //Инициализация графического окружения
    ui->setupUi(this);
    //Установка стиля заголовка таблицы
    ui->tableWidget->horizontalHeader()->
            setSectionResizeMode(QHeaderView::Stretch);
    //Функция создания таблицы с сеансами
    configure(config);
}

//Деструктор класса InformationWindow
InformationWindow::~InformationWindow()
{
    delete ui;  //очистка дизайна
}

//Функция создания таблицы с сеансами
void InformationWindow::configure(QString config)
{
    ifstream ifile; //Создание файла для считывания
    //  Проверка, открыт ли файл для чтения
    if (!Exception::isFileOpenForReading(this, ifile,
                                         Film::getFileName().toStdString()))
    {   //Пока не конец файла, считывание сеансов
        while (!ifile.eof())
        {
            Seance newSeance;
            ifile >> newSeance;
            if (!newSeance.getDate().compare(selectedDate))
                arrayOfSeances.push(newSeance);
        }
        ifile.close();  //Закрытие файла
        //Сортировка контейнера сеансов
        Algorithm::sort(arrayOfSeances.begin(), arrayOfSeances.end());
        //Создание заголовка таблицы
        ui->tableWidget->setColumnCount(4);
        QStringList list;
        list << "Film's name" << "Genre" << "Time" << "Number of sold
tickets";
        ui->tableWidget->setHorizontalHeaderLabels(list);
        list.clear();
        //Добавление сеансов в таблицу
        for(int i = 0; i < arrayOfSeances.getSize(); i++)
        {
            ui->tableWidget->insertRow(ui->tableWidget->rowCount());
            int numRow = ui->tableWidget->rowCount() - 1;
            ui->tableWidget->setItem(numRow, 0, new QTableWidgetItem(
                                         arrayOfSeances[i].getName()));
            ui->tableWidget->setItem(numRow, 1, new QTableWidgetItem(
                                         arrayOfSeances[i].getGenre()));
            ui->tableWidget->setItem(numRow, 2, new QTableWidgetItem(
                                         arrayOfSeances[i].getTime()));
            if (!config.compare("LIST"))    //Если нужна ведомость
            {   //Добавление в таблицу элемента ведомости
                ui->tableWidget->setItem(numRow, 3,
                                         new QTableWidgetItem(
                                             QString::number(

arrayOfSeances[i].getSoldPlaces().
                                                     getSize())));
            }
```

```
            else    //Иначе добавление в таблицу элемента отчета
                ui->tableWidget->setItem(numRow, 3, new QTableWidgetItem(
                                                "10 - 20 BYN"));
        }
    }
}

//Обработчик нажатия на кнопку backButton
void InformationWindow::on_backButton_clicked()
{   //Создание элемента класса calendarWindow
    CalendarWindow calendarWindow;
    this->close();          //Закрытие текущего окна
    calendarWindow.exec();  //Исполнение окна calendarWindow
}
```

Файл iterator.h

```
#ifndef ITERATOR_H
#define ITERATOR_H
#include "constiterator.h"
#include "Node.h"

template <class T>
class Queue;

template <class T>
class Iterator: public ConstIterator<T> {
public:
    //Конструктор класса Iterator
    Iterator();
    //Оператор * возвращает изменяемую ссылку на
    //данные в текущем узле
    T& operator*();
    //Оператор * возвращает ссылку на элемент
    const T& operator*() const;
    //Перегрузка префиксного инкремента
    Iterator<T>& operator++();
    //Перегрузка постфиксного инкремента
    Iterator<T> operator++(int);
    //Перегрузки операторов сравнения инераторов
    bool operator>(const Iterator<T> itr);
    bool operator<(const Iterator<T> itr);
protected:
    //Защищенный конструктор класса Iterator
    Iterator( Node<T>* ptr);
    //Дружественный класс Queue<T>
    friend class Queue<T>;
};
#endif
```

Файл iterator.cpp

```
#include "iterator.h"
#include "exception.h"

//Конструктор класса Iterator
template <class T>
Iterator<T>::Iterator() {}
```

```
//Защищенный конструктор
template <class T>
Iterator<T>::Iterator(Node<T>* ptr): ConstIterator<T>(ptr) {}

//Перегрузка префиксного инкремента
template <class T>
Iterator<T>& Iterator<T>::operator++()
{
    this->current = this->current->next;
    if (!this->current) throw Exception();
    return *this;
}

//Перегрузка постфиксного инкремента
template <class T>
Iterator<T> Iterator<T>::operator++(int)
{
    Iterator<T> copy = *this;
    this->current = this->current->next;
    if (!this->current) throw Exception();
    return copy;
}

//Перегрузки оператора сравнения инераторов
template <class T>
bool Iterator<T>::operator>(const Iterator<T> itr)
{
    Node<T>* ptr = itr.current;
    while (ptr != this->current && ptr->next)
        ptr = ptr->next;
    if (ptr->next)
        return false;
    else
        return true;
}

//Перегрузки оператора сравнения инераторов
template <class T>
bool Iterator<T>::operator<(const Iterator<T> itr)
{   //Использование предыдущей перегрузки
    if ((*this) > itr)
        return false;
    else
        return true;
}

//Оператор * возвращает изменяемую ссылку на
//данные в текущем узле
template <class T>
T& Iterator<T>::operator*()
{
    if (!this->current) throw Exception();
    return this->current->data;
}

//Оператор * возвращает ссылку на элемент
template <class T>
const T& Iterator<T>::operator*() const
{
    if (!this->current) throw Exception();
    return this->current->data;
}
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "calendarwindow.h"
#include "film.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    //Конструктор класса MainWindow
    MainWindow(QWidget *parent = nullptr);
    //Деструктор класса MainWindow
    ~MainWindow();

private slots:
    //Обработчик нажатия на кнопку buyTicketsButton,
    //осуществляет переход на следующий экран
    void on_buyTicketsButton_clicked();

private:
    //Указатель на графическую составляющую класса
    Ui::MainWindow *ui;
};
#endif
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "calendarwindow.h"
#include <QFileDialog>

//Конструктор класса MainWindow
MainWindow::MainWindow(QWidget *parent): QMainWindow(parent),
    ui(new Ui::MainWindow)
{   //Инициализация графического окружения
    ui->setupUi(this);
    //Установка картинки на задний фон окна
    QPixmap background("/Users/margo/Documents/"
                       "QTProjects/MovieTickets/Background.jpg");
    background = background.scaled(this->size(),
                                   Qt::IgnoreAspectRatio);
    QPalette palette;
    palette.setBrush(QPalette::Background, background);
    this->setPalette(palette);
}

//Деструктор класса MainWindow
MainWindow::~MainWindow()
```

```
{
    delete ui;  //очистка дизайна
}

//Обработчик нажатия на кнопку buyTicketsButton
void MainWindow::on_buyTicketsButton_clicked()
{   //Открытие окна для выбора рабочего файла
    Film::setFileName(QFileDialog::
                      getOpenFileName(this,
                      tr("Open File with films"), "",
                      tr("Films (*.csv);;All Files (*)")));
    //Создание объекта класса CalendarWindow
    CalendarWindow calendarWindow;
    this->close();          //Закрытие текущего окна
    calendarWindow.exec();  //Исполнение окна calendarWindow
}
```

Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{   //Создание приложения
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    //Запуск первого окна
    return a.exec();
}
```

Файл Node.h

```
#ifndef NODE_H
#define NODE_H

#include <iostream>

template <class T>
struct Node {
    //Информация об объекте
    T data;
    //Указатель на следующий элемент очереди
    Node* next;
    //Конструктор структуры Node
    Node(const T& _data = T{}, Node* _next = nullptr) :
        data{ _data }, next{ _next } {}
};

#endif
```

Файл placeswindow.h

```
#ifndef PLACESWINDOW_H
#define PLACESWINDOW_H
```

```
#include <QDialog>
#include <QButtonGroup>
#include "ticket.h"
#include "film.h"

namespace Ui {
class PlacesWindow;
}

class PlacesWindow : public QDialog
{
    Q_OBJECT
public:
    //Очередь сеансов на определенный день
    Queue<Seance> arrayOfSeances;
    //Индекс выбранного сеанса
    int index;
    //Конструктор класса PlacesWindow
    explicit PlacesWindow(Queue<Seance> _arrayOfSeances, int _index,
                          QWidget *parent = nullptr);
    //Деструктор класса PlacesWindow
    ~PlacesWindow();
    //Функция создания группы кнопок (зрительный зал)
    void createPlacesGroup();
private slots:
    //Обработчик нажатия на место в зале,
    //изменяет цвет выбранного места
    void placeClicked(int i);
    //Обработчик нажатия на кнопку backButton,
    //осуществляет возврат на предыдущий экран
    void on_backButton_clicked();
    //Обработчик нажатия на кнопку choosePlacesButton,
    //осуществляет переход на экран
    //с информацией о купленном билете
    void on_choosePlacesButton_clicked();
private:
    //Указатель на графическую составляющую класса
    Ui::PlacesWindow *ui;
    //Указатель на группу кнопок (зрительный зал)
    QButtonGroup *placesGroup;
};

#endif
```

Файл placeswindow.cpp

```
#include "placeswindow.h"
#include "ui_placeswindow.h"
#include "seance.h"
#include "filmswindow.h"
#include <QMessageBox>
#include "ticketwindow.h"

//Конструктор класса PlacesWindow
PlacesWindow::PlacesWindow(Queue<Seance> _arrayOfSeances,
                           int _index, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::PlacesWindow)
{
    arrayOfSeances = _arrayOfSeances;
    index = _index;
```

```
    //Инициализация графического окружения
    ui->setupUi(this);
    //Создание группы кнопок (зрительного зала)
    createPlacesGroup();
}

//Деструктор класса PlacesWindow
PlacesWindow::~PlacesWindow()
{
    delete ui;  //очистка дизайна
}

//Создание группы кнопок (зрительного зала)
void PlacesWindow::createPlacesGroup()
{
    this->placesGroup = new QButtonGroup(this);
    placesGroup->addButton(ui->place_1, 1);
    placesGroup->addButton(ui->place_2, 2);
    placesGroup->addButton(ui->place_3, 3);
    placesGroup->addButton(ui->place_4, 4);
    placesGroup->addButton(ui->place_5, 5);
    placesGroup->addButton(ui->place_6, 6);
    placesGroup->addButton(ui->place_7, 7);
    placesGroup->addButton(ui->place_8, 8);
    placesGroup->addButton(ui->place_9, 9);
    placesGroup->addButton(ui->place_10, 10);
    placesGroup->addButton(ui->place_11, 11);
    placesGroup->addButton(ui->place_12, 12);
    placesGroup->addButton(ui->place_13, 13);
    placesGroup->addButton(ui->place_14, 14);
    placesGroup->addButton(ui->place_15, 15);
    placesGroup->addButton(ui->place_16, 16);
    placesGroup->addButton(ui->place_17, 17);
    placesGroup->addButton(ui->place_18, 18);
    placesGroup->addButton(ui->place_19, 19);
    placesGroup->addButton(ui->place_20, 20);
    placesGroup->addButton(ui->place_21, 21);
    placesGroup->addButton(ui->place_22, 22);
    placesGroup->addButton(ui->place_23, 23);
    placesGroup->addButton(ui->place_24, 24);
    placesGroup->addButton(ui->place_25, 25);
    placesGroup->addButton(ui->place_26, 26);
    placesGroup->addButton(ui->place_27, 27);
    placesGroup->addButton(ui->place_28, 28);
    placesGroup->addButton(ui->place_29, 29);
    placesGroup->addButton(ui->place_30, 30);
    placesGroup->addButton(ui->place_31, 31);
    placesGroup->addButton(ui->place_32, 32);
    placesGroup->addButton(ui->place_33, 33);
    placesGroup->addButton(ui->place_34, 34);
    placesGroup->addButton(ui->place_35, 35);
    placesGroup->addButton(ui->place_36, 36);
    placesGroup->addButton(ui->place_37, 37);
    placesGroup->addButton(ui->place_38, 38);
    placesGroup->addButton(ui->place_39, 39);
    placesGroup->addButton(ui->place_40, 40);
    placesGroup->addButton(ui->place_41, 41);
    placesGroup->addButton(ui->place_42, 42);
    placesGroup->addButton(ui->place_43, 43);
    placesGroup->addButton(ui->place_44, 44);
    placesGroup->addButton(ui->place_45, 45);
    placesGroup->addButton(ui->place_46, 46);
    placesGroup->addButton(ui->place_47, 47);
    placesGroup->addButton(ui->place_48, 48);
```

```
placesGroup->addButton(ui->place_49, 49);
placesGroup->addButton(ui->place_50, 50);
placesGroup->addButton(ui->place_51, 51);
placesGroup->addButton(ui->place_52, 52);
placesGroup->addButton(ui->place_53, 53);
placesGroup->addButton(ui->place_54, 54);
placesGroup->addButton(ui->place_55, 55);
placesGroup->addButton(ui->place_56, 56);
placesGroup->addButton(ui->place_57, 57);
placesGroup->addButton(ui->place_58, 58);
placesGroup->addButton(ui->place_59, 59);
placesGroup->addButton(ui->place_60, 60);
placesGroup->addButton(ui->place_61, 61);
placesGroup->addButton(ui->place_62, 62);
placesGroup->addButton(ui->place_63, 63);
placesGroup->addButton(ui->place_64, 64);
placesGroup->addButton(ui->place_65, 65);
placesGroup->addButton(ui->place_66, 66);
placesGroup->addButton(ui->place_67, 67);
placesGroup->addButton(ui->place_68, 68);
placesGroup->addButton(ui->place_69, 69);
placesGroup->addButton(ui->place_70, 70);
placesGroup->addButton(ui->place_71, 71);
placesGroup->addButton(ui->place_72, 72);
placesGroup->addButton(ui->place_73, 73);
placesGroup->addButton(ui->place_74, 74);
placesGroup->addButton(ui->place_75, 75);
placesGroup->addButton(ui->place_76, 76);
placesGroup->addButton(ui->place_77, 77);
placesGroup->addButton(ui->place_78, 78);
placesGroup->addButton(ui->place_79, 79);
placesGroup->addButton(ui->place_80, 80);
placesGroup->addButton(ui->place_81, 81);
placesGroup->addButton(ui->place_82, 82);
placesGroup->addButton(ui->place_83, 83);
placesGroup->addButton(ui->place_84, 84);
placesGroup->addButton(ui->place_85, 85);
placesGroup->addButton(ui->place_86, 86);
placesGroup->addButton(ui->place_87, 87);
placesGroup->addButton(ui->place_88, 88);
placesGroup->addButton(ui->place_89, 89);
placesGroup->addButton(ui->place_90, 90);
placesGroup->addButton(ui->place_91, 91);
placesGroup->addButton(ui->place_92, 92);
placesGroup->addButton(ui->place_93, 93);
placesGroup->addButton(ui->place_94, 94);
placesGroup->addButton(ui->place_95, 95);
placesGroup->addButton(ui->place_96, 96);
placesGroup->addButton(ui->place_97, 97);
placesGroup->addButton(ui->place_98, 98);
placesGroup->addButton(ui->place_99, 99);
placesGroup->addButton(ui->place_100, 100);
placesGroup->addButton(ui->place_101, 101);
placesGroup->addButton(ui->place_102, 102);
placesGroup->addButton(ui->place_103, 103);
placesGroup->addButton(ui->place_104, 104);
placesGroup->addButton(ui->place_105, 105);
placesGroup->addButton(ui->place_106, 106);
placesGroup->addButton(ui->place_107, 107);
placesGroup->addButton(ui->place_108, 108);
placesGroup->addButton(ui->place_109, 109);
placesGroup->addButton(ui->place_110, 110);
placesGroup->addButton(ui->place_111, 111);
placesGroup->addButton(ui->place_112, 112);
```

```
    placesGroup->addButton(ui->place_113, 113);
    placesGroup->addButton(ui->place_114, 114);
    placesGroup->addButton(ui->place_115, 115);
    placesGroup->addButton(ui->place_116, 116);
    placesGroup->addButton(ui->place_117, 117);
    placesGroup->addButton(ui->place_118, 118);
    placesGroup->addButton(ui->place_119, 119);
    placesGroup->addButton(ui->place_120, 120);
    placesGroup->addButton(ui->place_121, 121);
    placesGroup->addButton(ui->place_122, 122);
    //Установка стиля кнопок
    for(int i = 1; i <= placesGroup->buttons().size(); i++)
        placesGroup->button(i)->
        setStyleSheet("background-color: qlineargradient("
                      "spread:pad, x1:0.505208,"
                      "y1:0.0797727, x2:0.510417, y2:1, "
                      "stop:0 rgba(0, 0, 0, 204),"
                      "stop:1 rgba(59, 59, 59, 255));");
    //Задание красного цвета купленным местам
    for (int i = 0; i < arrayOfSeances[index].
         getSoldPlaces().getSize(); i++)
    {
        int j = arrayOfSeances[index].getSoldPlaces()[i];
        placesGroup->button(j)->setStyleSheet(
                    "background-color: red;");
    }
    //Подключение группы кнопок к сигналам и слотам
    connect(placesGroup, SIGNAL(buttonClicked(int)),
            this, SLOT(placeClicked(int)));
}

//Обработчик нажатия на место в зале
void PlacesWindow::placeClicked(int i)
{   //Если место выбрано, то установка стандартного стиля
    if (placesGroup->button(i)->
            styleSheet() == "background-color: gray;")
        placesGroup->button(i)->
            setStyleSheet("background-color: qlineargradient("
                          "spread:pad, x1:0.505208,"
                          "y1:0.0797727, x2:0.510417, y2:1, "
                          "stop:0 rgba(0, 0, 0, 204),"
                          "stop:1 rgba(59, 59, 59, 255));");
    //Если место не было выбрано и не куплено,
    //то окрашивание в серый цвет
    else if (placesGroup->button(i)->
             styleSheet() != "background-color: red;")
        placesGroup->button(i)->
                setStyleSheet("background-color: gray;");
}

//Обработчик нажатия на кнопку backButton
void PlacesWindow::on_backButton_clicked()
{   //Создание элемента класса filmsWindow
    FilmsWindow filmsWindow(arrayOfSeances[index].getDate());
    this->close();      //Закрытие текущего окна
    filmsWindow.exec(); //Исполнение окна filmsWindow
}

//Обработчик нажатия на кнопку choosePlacesButton
void PlacesWindow::on_choosePlacesButton_clicked()
{
    Queue<int> soldPlaces = arrayOfSeances[index].getSoldPlaces();
    Queue<int> selectedPlaces;
    //Запись выбранных мест в контейнер
```

```
    for (int i = 1; i <= placesGroup->buttons().size(); i++)
        if (placesGroup->button(i)->styleSheet() ==
                "background-color: gray;")
        {
            soldPlaces.push(i);
            arrayOfSeances[index].setSoldPlaces(soldPlaces);
            selectedPlaces.push(i);
        }
    //Если не были выбраны места, вывод предупреждения
    if (selectedPlaces.isEmpty())
        QMessageBox::warning(this,"Attention!",
                             "You didn't choose places!");
    else
    {   //Перезапись сеансов в файл
        FilmsWindow::rewriteFile(this, arrayOfSeances,
                                 arrayOfSeances[index].getDate());
        //Создание элемента класса filmsWindow
        TicketWindow ticketWindow(arrayOfSeances[index],
                                  selectedPlaces);
        this->close();          //Закрытие текущего окна
        ticketWindow.exec();    //Открытие окна ticketWindow
    }
}
```

Файл queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

#include "Node.h"
#include "iterator.h"
#include "constiterator.h"

template <class T>
class Queue {
    int size;       //Количество элементов очереди
    Node<T>* first; //Указатель на первый элемент очереди
    Node<T>* last;  //Указатель на последний элемент очереди
public:
    //Конструктор по умолчанию класса Queue
    Queue();
    //Конструктор с параметром класса Queue
    Queue(const Queue& obj);
    //Деструктор класса Queue
    ~Queue();
    //Функции получения значений полей класса
    int getSize();
    T getFirst();
    T getLast();
    //Добавление элемента в конец очереди
    void push(const T& value);
    //Удаление элемента с начала очереди
    void pop();
    //Проверка, пуста ли очередь
    bool isEmpty();
    //Очистка очереди
    void eraseAll();
    //Удаление элемента по индексу
    void eraseByIndex(int index);
    //Функция, обменивающая значения объектов
    void swap(Queue<T>& obj);
```

```
    //Итератор на начало очереди
    Iterator<T> begin();
    ConstIterator<T> begin() const;
    //Итератор на конец очереди
    Iterator<T> end();
    ConstIterator<T> end() const;
    //Перегрузка оператора присваивания
    const Queue<T>& operator=(const Queue<T>& obj);
    //Получение элемента по индексу
    T& operator[](int index);
    //Перегрузка операторов сравнения
    friend bool operator==(const Queue<T>& obj1,
                           const Queue<T>& obj2);
    inline friend bool operator!=(const Queue<T>& obj1,
                           const Queue<T>& obj2);
};

#endif
```

Файл queue.cpp

```
#include "queue.h"
#include <QMessageBox>

//Конструктор по умолчанию класса Queue
template<class T>
Queue<T>::Queue()
{
    size = 0;
    first = new Node<T>();
    last = new Node<T>();
    first = last = nullptr;
}

//Конструктор с параметром класса Queue
template<class T>
Queue<T>::Queue(const Queue& obj)
{
    size = 0;
    first = new Node<T>();
    last = first;
    first = last = nullptr;
    if (obj.size == 0) return;

    Node<T>* temp = obj.first;
    while (temp != nullptr)
    {
        push(temp->data);
        temp = temp->next;
    }
}

//Деструктор класса Queue
template<class T>
Queue<T>::~Queue()
{
    eraseAll();
}

//Получение размера
template<class T>
```

```
int Queue<T>::getSize()
{
    return size;
}

//Получение значения первого элемента
template<class T>
T Queue<T>::getFirst()
{
    return first->data;
}

//Получение значения последнего элемента
template<class T>
T Queue<T>::getLast()
{
    return last->data;
}

//Добавление элемента в конец очереди
template<class T>
void Queue<T>::push(const T& value)
{
    Node<T>* temp = new Node<T>;
    temp->data = value;

    if (size == 0)
    {
        first = temp;
        last = first;
    }
    else
    {
        last->next = temp;
        last = temp;
    }
    size++;
}

//Удаление элемента с начала очереди
template<class T>
void Queue<T>::pop()
{
    if (first == nullptr)
    {
        return;
    }

    Node<T>* temp = first;
    first = first->next;
    delete temp;
    size--;
}

//Проверка, пуста ли очередь
template<class T>
bool Queue<T>::isEmpty()
{
    return (size == 0);
}

//Удаление всех элементов очереди
template<class T>
void Queue<T>::eraseAll()
```

```
{

    while (first != nullptr)
        pop();
}

//Перегрузка операции присваивания
template<class T>
const Queue<T>& Queue<T>::operator=(const Queue<T>& obj)
{
    Queue<T> copy(obj);
    this->swap(copy);
    return *this;
}

//Получение элемента по индексу
template <class T>
T& Queue<T>::operator[](int index)
{

        Node<T>* temp = first;

        for (int i = 0; i < index; ++i)
            temp = temp->next;

        return temp->data;
}

//Удаление элемента по индексу
template <class T>
void Queue<T>::eraseByIndex(int index)
{
    Node<T>* temp = first;
    //Удаление нулевого элемента
    if (index == 0)
    {
        temp = first;
        first = first->next;
        delete temp;
    }
    else
    {   //Удаление ненулевого элемента
        for (int i = 0; i < index - 1; ++i)
            temp = temp->next;
        Node<T>* deleteTemp = temp->next;
        temp->next = temp->next->next;
        delete deleteTemp;
    }
    size--;
}

//Функция, обменивающая значения объектов
template <class T>
void Queue<T>::swap(Queue<T>& obj)
{
    std::swap(first, obj.first);
    std::swap(last, obj.last);
    std::swap(size, obj.size);
}

//Итератор на начало очереди
template<class T>
Iterator<T> Queue<T>::begin()
{
```

```
    Iterator<T> itr{ first };
    return itr;
}

//Константный итератор на начало очереди
template<class T>
ConstIterator<T> Queue<T>::begin() const
{
    if (!isEmpty())
    {
        ConstIterator<T> constItr{ first };
        return constItr;
    }
}

//Итератор на конец очереди
template<class T>
Iterator<T> Queue<T>::end()
{
    Iterator<T> itr{ last };
    return itr;
}

//Константный итератор на конец очереди
template<class T>
ConstIterator<T> Queue<T>::end() const
{
    ConstIterator<T> constItr{ last };
    return constItr;
}
```

Файл seance.h

```
#ifndef SEANCE_H
#define SEANCE_H
#include "film.h"
#include "queue.h"

class Seance: public Film
{

    QString date;           //Дата сеанса
    QString time;           //Время сеанса
    Queue<int> soldPlaces;  //Проданные места
public:
    //Конструктор класса Seance
    Seance(QString _name = "", QString _genre = "",
           QString _date = "", QString _time = "",
           Queue<int> _soldPlaces = {});
    //Деструктор класса Seance
    virtual ~Seance();
    //Перегрузка считывания из файла
    friend ifstream & operator >> (ifstream &ifile, Seance &seance);
    //Перегрузка записи в файл
    friend ofstream & operator << (ofstream &ofile, Seance &seance);
    //Функции получения и установки значений полей класса
    QString getTime() const;
    void setTime(const QString &value);
    Queue<int> getSoldPlaces() const;
    void setSoldPlaces(const Queue<int> &value);
    QString getDate() const;
```

```
    void setDate(const QString &value);
    //Статическое поле со стеком для отмены последнего действия
    static QStack< Queue<Seance> > undoStack;
    //Перегрузка сравнения объектов
    friend bool operator>(Seance obj1, Seance obj2);
};

#endif
```

Файл seance.cpp

```
#include "seance.h"
#include <string>
#include <string.h>
#include "queue.cpp"
//Максимальная длина строки
#define MAX_STRING_LENGHT 50

//Конструктор класса Seance
Seance::Seance(QString _name, QString _genre,
               QString _date, QString _time,
               Queue<int> _soldPlaces):
    Film(_name, _genre)
{
    date = _date;
    time = _time;
    soldPlaces = _soldPlaces;
}

//Деструктор класса Seance
Seance::~Seance() {}

//Приведение строки в очередь
Queue<int> castStringToArray(char* str)
{
    Queue<int> array;
    if (str[strlen(str) - 1] != '\r')
        str[strlen(str)] = '\r';
    for (int i = 0; str[i]; i++)
    {
        char temp[5] = "";
        for (int j = 0; str[i] != ' ' &&
             str[i] != '\r' && str[i]; j++, i++)
            temp[j] = str[i];
        array.push(atoi(temp));
        if (str[i] == '\r') return array;
    }
    return array;
}

//Приведение очереди в строку
char* castArrayToString(Queue<int> array)
{
    if (array.getSize() == 0)
        return nullptr;
    string str = to_string(array[0]);
    for (int i = 1; i < array.getSize(); i++)
    {
        str.append(" ");
        str.append(to_string(array[i]));
    }
```

```
    char* cstr;
    cstr = new char[str.length() + 1];
    strcpy(cstr, str.c_str());
    return cstr;
}

//Перегрузка считывания из файла
ifstream & operator >> (ifstream &ifile, Seance &seance)
{   //Считывание полей класса Film
    ifile >> static_cast<Film&>(seance);
    char* temp;     //Создание промежуточной строки
    temp = new char[MAX_STRING_LENGHT];
    //Считывание даты
    ifile.getline(temp, MAX_STRING_LENGHT, ',');
    seance.date = temp;
    //Считывание времени
    ifile.getline(temp, MAX_STRING_LENGHT, ',');
    seance.time = temp;
    //Считывание купленных мест
    ifile.getline(temp, MAX_STRING_LENGHT, '\n');
    if (strlen(temp) > 0)
        seance.soldPlaces = castStringToArray(temp);
    //Удаление промежуточной строки
    delete[] temp;
    return ifile;
}

//Перегрузка записи в файл
ofstream & operator << (ofstream &ofile, Seance &seance)
{   //Запись полей класса Film
    ofile << static_cast<Film&>(seance);
    //Запись даты
    ofile.write(seance.date.toUtf8(), seance.date.size());
    ofile.write(",", 1);
    //Запись времени
    ofile.write(seance.time.toUtf8(), seance.time.size());
    ofile.write(",", 1);
    //Запись купленных мест
    if (seance.soldPlaces.getSize() > 0)
        ofile.write(castArrayToString(seance.soldPlaces),
             strlen(castArrayToString(seance.soldPlaces)));
    ofile.write("\n", 1);
    return ofile;
}

//Перегрузка сравнения объектов
bool operator>(Seance obj1, Seance obj2)
{   //Сравнение по имени
    if (obj1.getName().compare(obj2.getName()) > 0)
        return true;
    //Сравнение по времени
    else if (obj1.getName().compare(obj2.getName()) == 0
             && obj1.getTime().compare(obj2.getTime()) > 0)
        return true;
    else return false;
}

//Получение значения поля date
QString Seance::getDate() const
{
    return date;
}
//Установка значения поля date
void Seance::setDate(const QString &value)
```

```
{
    date = value;
}
//Получение значения поля time
QString Seance::getTime() const
{
    return time;
}
//Установка значения поля time
void Seance::setTime(const QString &value)
{
    time = value;
}
//Получение значения поля soldPlaces
Queue<int> Seance::getSoldPlaces() const
{
    return soldPlaces;
}
//Установка значения поля soldPlaces
void Seance::setSoldPlaces(const Queue<int> &value)
{
    soldPlaces = value;
}
```

Файл ticket.h

```
#ifndef TICKET_H
#define TICKET_H
#include "seance.h"

class Ticket: public Seance
{
    //Купленные места
    Queue<int> places;
    //Цена билета
    int price;
public:
    //Конструктор класса Ticket
    Ticket(QString _name = "", QString _genre = "",
           QString _date = "", QString _time = "",
           Queue<int> _soldPlaces = {},
           Queue<int> _places = {}, int _price = 0);
    //Деструктор класса Ticket
    ~Ticket();
    //Функции получения и установки значений полей класса
    Queue<int> getPlaces() const;
    void setPlaces(const Queue<int> &value);
    int getPrice() const;
    void setPrice(int value);
};
#endif
```

Файл ticket.cpp

```
#include "ticket.h"
#include "queue.cpp"

//Конструктор класса Ticket
Ticket::Ticket(QString _name, QString _genre,
```

```
                QString _date, QString _time,
                Queue<int> _soldPlaces,
                Queue<int> _places, int _price):
    Seance(_name, _genre, _date, _time, _soldPlaces)
{
    places = _places;
    price = _price;
}

//Деструктор класса Ticket
Ticket::~Ticket() {}

//Получение значения поля places
Queue<int> Ticket::getPlaces() const
{
    return places;
}
//Установка значения поля places
void Ticket::setPlaces(const Queue<int> &value)
{
    places = value;
}
//Получение значения поля price
int Ticket::getPrice() const
{
    return price;
}
//Установка значения поля price
void Ticket::setPrice(int value)
{
    price = value;
}
```

Файл ticketwindow.h

```
#ifndef TICKETWINDOW_H
#define TICKETWINDOW_H

#include <QDialog>
#include "ticket.h"

namespace Ui {
class TicketWindow;
}

class TicketWindow : public QDialog
{
    Q_OBJECT
    //Билет
    Ticket ticket;
public:
    //Конструктор класса TicketWindow
    explicit TicketWindow(Seance selectedSeance, Queue<int> selectedPlaces,
                          QWidget *parent = nullptr);
    //Деструктор класса TicketWindow
    ~TicketWindow();
private slots:
    //Обработчик нажатия на кнопку exitButton,
    //осуществляет выход из программы
    void on_exitButton_clicked();
    //Обработчик нажатия на кнопку buyMoreButton,
    //осуществляет переход на экран выбора даты
```

```
    void on_buyMoreButton_clicked();
private:
    //Указатель на графическую составляющую класса
    Ui::TicketWindow *ui;
};

#endif
```

Файл ticketwindow.h

```
#include "ticketwindow.h"
#include "ui_ticketwindow.h"
#include <QString>
#include "seance.cpp"
#include "calendarwindow.h"

//Кончтруктор класса TicketWindow
TicketWindow::TicketWindow(Seance selectedSeance,
                           Queue<int> selectedPlaces,
                           QWidget *parent) :
    QDialog(parent),
    ui(new Ui::TicketWindow)
{
    ticket.setName(selectedSeance.getName());
    ticket.setGenre(selectedSeance.getGenre());
    ticket.setDate(selectedSeance.getDate());
    ticket.setTime(selectedSeance.getTime());
    ticket.setPlaces(selectedPlaces);
    int _price = 0;
    //Установка цены в зависимости от выбранных мест
    for (int i = 0; i < selectedPlaces.getSize(); i++)
    {
        if(selectedPlaces[i] >= 118)
            _price += 20;
        else
            _price += 10;
    }
    ticket.setPrice(_price);
    //Инициализация графического окружения
    ui->setupUi(this);
    //Вывод информации о билете на экран
    QString format = tr("<p align='center'><span style='"
                        "font-size:24pt;'>%1</span></p>");
    ui->filmLabel_2->setText(format.arg(ticket.getName()));
    ui->genreLabel_2->setText(format.arg(ticket.getGenre()));
    ui->dateLabel_2->setText(format.arg(ticket.getDate()));
    ui->timeLabel_2->setText(format.arg(ticket.getTime()));
    ui->placesLabel_2->setText(format.arg(castArrayToString(
                                          ticket.getPlaces())));
    ui->priceLabel_2->setText(format.arg(ticket.getPrice()));
}

//Деструктор класса TicketWindow
TicketWindow::~TicketWindow()
{
    delete ui;  //очистка дизайна
}

//Обработчик нажатия на кнопку exitButton
void TicketWindow::on_exitButton_clicked()
{
    this->close();  //Закрытие текущего окна
}
```

```
//Обработчик нажатия на кнопку buyMoreButton
void TicketWindow::on_buyMoreButton_clicked()
{   //Создание объекта класса CalendarWindow
    CalendarWindow calendarWindow;
    this->close();          //Закрытие текущего окна
    calendarWindow.exec();  //Исполнение окна calendarWindow
}
```