Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ИНФОРМАЦИОННАЯ СИСТЕМА

АЭРОПОРТА

БГУИР КП 1-40 02 01 127 ПЗ

Студент: гр. 250501 Щербаков Д. И.

Руководитель: старший преподаватель

каф. ЭВМ Ковальчук А. М.

МИНСК 2023

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 3

1 ПОСТАНОВКА ЗАДАЧИ 4

2 ОБЗОР ЛИТЕРАТУРЫ 5

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВНИЕ 6

3.1 Структура входных и выходных данных 6

3.2 Разработка диаграммы класов 7

3.3 Описание классов 7

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 24

4.1 Разработка схем алгоритмов 24

4.2 Разработка алгоритмов 24

5 РЕЗУЛЬТАТ РАБОТЫ 28

ЗАКЛЮЧЕНИЕ 31

СПИСОК ЛИТЕРАТУРЫ 32

ПРИЛОЖЕНИЕ А 33

ПРИЛОЖЕНИЕ Б 34

ПРИЛОЖЕНИЕ В 35

ПРИЛОЖЕНИЕ Г 36

ПРИЛОЖЕНИЕ Д 37

# 

# ВВЕДЕНИЕ

Современные аэропорты являются сложными организациями, обрабатывающими огромные объемы информации и операций ежедневно. Для эффективного управления всеми аспектами аэропортовской деятельности, начиная от бронирования билетов и заканчивая обслуживанием самолетов на взлетно-посадочной полосе, широко используются информационные системы. В данном курсовом проекте рассматривается разработка информационной системы для аэропорта с использованием языка программирования C++.

C++ является языком программирования общего назначения, который обладает высокой производительностью и эффективностью. Современные аэропорты обрабатывают огромные объемы данных и операций в режиме реального времени. Этот язык программирования позволяет оптимизировать производительность системы, обеспечивая быстрый доступ к данным и эффективную обработку информации.

C++ имеет богатую стандартную библиотеку, включающую в себя множество полезных инструментов и структур данных. Это упрощает разработку и уменьшает время, затрачиваемое на написание кода, так как многие стандартные задачи уже имеют готовые решения.

Наличие объектно-ориентированного программирования позволяет организовать код вокруг объектов, инкапсулировать данные и методы, упростить сопровождение кода, а также обеспечить гибкость и расширяемость системы. Возможности наследования позволяют создавать новые классы, основанные на существующих, что сокращает дублирование кода и способствует легкости разработки специализированных компонентов. Полиморфизм, в свою очередь, позволяет обращаться к разным элементам системы через общий интерфейс, что делает код более гибким и способствует упрощению обработки разнообразных объектов, таких как разные виды билетов или рейсов, обеспечивая легкость внесения изменений в систему.

# 1 ПОСТАНОВКА ЗАДАЧИ

Программа “Информационная система аэропорта” должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню. Вся информация должна находиться в различных файлах, связанных определенным образом. Реализовать функции выдачи информации о нужном рейсе, добавления новой информации, резервирования билетов. Другие методы, в зависимости от специфики задачи.

При реализации операции редактирования, добавления, удаления информации необходимо предусмотреть операцию отмены последних действий. Разработать иерархию классов с использованием наследования (не меньше 3-х уровней наследования). Разработать и использовать в программе классы контейнеров, итераторов и алгоритмов (свои и STL). Производить обработку исключительных ситуаций.

# 2 ОБЗОР ЛИТЕРАТУРЫ

Для решения задачи был выбран язык программирования С++ который поддерживает такую парадигму программирования как объектно-ориентированное программирование (ООП), которая включает в себя три основных принципа: инкапсуляция, наследование и полиморфизм.

Инкапсуляция позволяет скрыть детали реализации объектов, представляя только необходимый интерфейс для взаимодействия с ними.

Наследование позволяет создавать новые классы, основанные на существующих, строить иерархии классов, где базовый класс содержит общие атрибуты и методы, а производные классы могут расширять или изменять их.

Полиморфизм позволяет обрабатывать объекты разных классов с использованием общего интерфейса.

Для создания удобного пользовательского интерфейса был использован фреймворк Qt, который содержит множество классов для работы с графическим интерфейсом.

Язык программирования C++ содержит в себе стандартную библиотеку шаблонов (STL), которая предоставляет множество шаблонных классов и функций, которые упрощают разработку программного обеспечения и обеспечивают общие структуры данных и алгоритмы. Для реализации алгоритма отмены последних действий был использован один из классов STL

– std::stack.

Для хранения объектов был реализован свой шаблонный класс контейнера – двунаправленная очередь. В программе реализован класс итератора и алгоритмов.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВНИЕ

**3.1 Структура входных и выходных данных**

Таблица 3.1 – файл прилетов пассажирских самолетов passengerarrivals.txt

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Модель самолета | Авиа- компания | Всего мест | Занято мест | Время прибытия | Прибывает из | Задержка | Время выдачи багажа |
| Boeing 737 | Belavia | 138/20/16 | 98/11/10 | 01.11.2023 16:00 | Хургада | 1 | 15 |

Таблица 3.2 - файл вылетов пассажирских самолетов passengerdepartures.txt

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Модель самолета | Авиа- компания | Всего мест | Занято мест | Время вылета | Направление | Стоимость билетов | Время полета |
| A330-3 | Air China | 240/28/12 | 101/11/8 | 23.11.2023 5:00 | Хургада | 566/750/920 | 8 |

Таблица 3.3 - файл прилетов грузовых самолетов cargoarrivals.txt

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Модель самолета | Авиа- компания | Грузо-подъемность | Вес груза | Кол-во грузов | Время прибытия | Прибывает из | Задержка | Время выдачи багажа |
| Boeing 747 | Atlas Air | 113400 | 1085 | 15 | 11.11.2023 16:00 | Дубай | 0 | 15 |

Таблица 3.4 - файл вылетов грузовых самолетов cargodepartures.txt

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Модель самолета | Авиа- компания | Грузо-подъемность | Вес груза | Кол-во грузов | Время вылета | Направ-ление | Стоимость | Время полёта |
| Airbus Beluga | Cargolux | 47000 | 890 | 13 | 01.11.2023 10:00 | Люксем-бург | 65 | 12 |

Таблица 3.5 - файл пассажиров passengers.txt

|  |  |  |  |
| --- | --- | --- | --- |
| Фамилия | Имя | Контактные данные | Место |
| Иванов | Иван | +375294562255, ivan@gmail.com | 2 |

Таблица 3.6 - файл грузов cargo.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Фамилия | Имя | Контактные данные | Название груза | Вес груза |
| Иванов | Иван | +375294562255, ivan@gmail.com | Автомобильные детали | 138 |

**3.2 Разработка диаграммы классов**

Диаграмма классов показана в приложении А.

**3.3 Описание классов**

Класс **Plane –** класс самолет.

Описание полей класса:

Qtring model – модель самолета.

Qtring company – название авиакомпании.

Описание методов класса:

Plane(const QString& model, const Qstring& company) – конструктор.

Plane(const Plane& obj) – конструктор копирования.

~Plane() – деструктор.

void setModel(const QString&) – установить модель самолета.

void setCompany(const QString&) – установить название авиакомпании.

QString getModel() – получить модель самолета.

QString getCompany() – получить название авиакомпании.

Plane& operator=(const Plane&) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const Plane&) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, Plane&) – оператор чтения с файла.

Класс **PassengerPlane –** класс пассажирский самолет.

Описание полей класса:

spaceType maxSpace – вместимость самолета.

spaceType space – занятые места.

int passengersCount – количество пассажиров.

Описание методов класса:

PassengerPlane(int econom, int business, int first) – конструктор.

PassengerPlane(const PassengerPlane& obj) – конструктор копирования.

~PassengerPlane() – деструктор.

void setMaxEconom(int) – установить количество мест эконом класса.

void setMaxBusiness(int) – установить количество мест бизнес класса.

void setMaxFirst(int) – установить количество мест первого класса.

void setPassengersCount() –установить количество пассажиров

int getMaxEconom() – получить количество мест эконом класса.

int getMaxBusiness() – получить количество мест бизнес класса.

int getMaxFirst() – получить количество мест первого класса.

int getPassengersCount() – получить количество пассажиров

void setEconom(int) – установить количество свободных мест эконом класса.

void setBusiness(int) – установить количество свободных мест бизнес класса.

void setFirst(int) – установить количество свободных мест первого класса.

int getEconom() – получить количество свободных мест эконом класса.

int getBusiness() – получить количество свободных мест бизнес класса.

int getFirst() – получить количество свободных мест первого класса.

bool isFool() – проверка на заполненность всех мест.

PassengerPlane & operator=(const PassengerPlane &) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const PassengerPlane &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, PassengerPlane &) – оператор чтения с файла.

Структура **spaceType** – структура классов мест самолета.

Описание полей структуры:

int economClass – количество мест эконом класса.

int businessClass – количество мест бизнес класса.

int firstClass – количество мест первого класса.

Класс **PassengerFlight –** класс пассажирский рейс.

Описание полей класса:

QDateTime date – дата и время полета.

QString location – местоположение

Описание методов класса:

PassengerFlight(QDateTime, QString) – конструктор.

PassengerFlight(const PassengerFlight&) – конструктор копирования.

~PassengerFlight() – деструктор.

void setDate(const QDateTime&) – установить дату и время полета.

void setLocation(const QString&) – установить местоположение.

QDateTime getDate() – получить дату и время полета.

QString getLocation() – получить местоположение.

PassengerFlight & operator=(const PassengerFlight &) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const PassengerFlight &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, PassengerFlight &) – оператор чтения с файла.

Класс **PassengerDepartures –** класс пассажирских вылетов.

Описание полей класса:

int flyingTime – время полета

int costEconom – стоимость билета эконом-класса

int costBusiness – стоимость билета бизнес-класса

int costFirst – стоимость билета первого класса

std::list<Passenger> passengerList – список пассажиров

Описание методов класса:

PassengerDepartures(int, int, int, int) – конструктор.

PassengerDepartures (const PassengerDepartures&) – конструктор копирования.

~ PassengerDepartures () – деструктор.

void setFlyingTime(int) – установить время полета

void setCostEconom(int) – установить стоимость билета эконом-класса

void setCostBusiness(int) – установить стоимость билета бизнес-класса

void setCostFirst(int) – установить стоимость билета первого класса

int getFlyingTime() – получить время полета

int getCostEconom() – получить стоимость билета эконом-класса

int getCostBusiness() – получить стоимость билета бизнес-класса

int getCostFirst() – получить стоимость билета первого класса

PassengerDepartures& operator=(const PassengerDepartures&) – оператор присваивания.

std::list<Passenger>& getPassengerList() – получить список пассажиров

friend QTextStream& operator<<(QTextStream&, const PassengerDepartures &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, PassengerDepartures &) – оператор чтения с файла.

Класс **PassengerArrivals –** класс пассажирских прилетов.

Описание полей класса:

int baggageTime – время выдачи багажа.

bool delayed – задержка прилета.

Описание методов класса:

PassengerArrivals(int, bool) – конструктор.

PassengerArrivals (const PassengerArrivals &) – конструктор копирования.

~ PassengerArrivals () – деструктор.

void setBaggageTime(int) – установить время выдачи багажа.

void setDelayed(bool) – установить задержку прилета.

int getBaggageTime() – получить время выдачи багажа.

bool isDelayed () – проверка на задержку прилета.

PassengerArrivals & operator=(const PassengerArrivals &) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const PassengerArrivals &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, PassengerArrivals &) – оператор чтения с файла.

Класс **Passenger –** класс пассажир.

Описание полей класса:

QString name – имя.

QString surname – фамилия.

QString number – контактные данные.

int space – тип места.

Описание методов класса:

Passenger (const QString&, const QString&, const QString&, int) – конструктор.

Passenger (const Passenger &) – конструктор копирования.

~ Passenger () – деструктор.

void setName(const QString&) – установить имя.

void setSurname(const QString&) – установить фамилию.

void setNumber(const QString&) – установить контактные данные.

void setSpace(int) – установить тип места.

QString getName() – получить имя.

QString getSurname() – получить фамилию.

QString getNumber()– получить контактные данные.

int getSpace() – получить тип места.

Passenger & operator=(const Passenger &) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const Passenger &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, Passenger &) – оператор чтения с файла.

Класс **CargoPlane –** класс грузовой самолет.

Описание полей класса:

float cargoCapacity – грузоподъемность самолета.

float currentWeight – вес текущего груза.

float freeWeight – свободный вес.

int cargoCount – количество грузов.

Описание методов класса:

CargoPlane(float, float, int) – конструктор.

CargoPlane(const CargoPlane& obj) – конструктор копирования.

~CargoPlane() – деструктор.

void setCargoCapacity(float) – установить грузоподъемность.

void setWeight(float) – установить текущий вес.

void setCargoCount(int) – установить количество грузов.

void setFreeWeight() – установить свободный вес.

float getCargoCapacity() – получить грузоподъемность.

float getWeight() – получить текущий вес.

float getFreeWeight() – получить свободный вес.

int getCargoCount() – получить количетсво грузов.

CargoPlane & operator=(const CargoPlane &) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const CargoPlane &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, CargoPlane &) – оператор чтения с файла.

Класс **CargoFlight –** класс грузовой рейс.

Описание полей класса:

QDateTime date – дата и время полета.

QString location – местоположение

Описание методов класса:

CargoFlight(QDateTime, QString) – конструктор.

CargoFlight (const PassengerFlight&) – конструктор копирования.

~ CargoFlight () – деструктор.

void setDate(const QDateTime&) – установить дату и время полета.

void setLocation(const QString&) – установить местоположение.

QDateTime getDate() – получить дату и время полета.

QString getLocation() – получить местоположение.

CargoFlight & operator=(const CargoFlight &) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const CargoFlight &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, CargoFlight &) – оператор чтения с файла.

Класс **CargoDepartures –** класс грузовых вылетов.

Описание полей класса:

int flyingTime – время полета

int cost – стоимость транспортировки груза.

std::list<Cargo> cargoList – список пассажиров

Описание методов класса:

CargoDepartures(int, int) – конструктор.

CargoDepartures (const CargoDepartures &) – конструктор копирования.

~ CargoDepartures () – деструктор.

void setFlyingTime(int) – установить время полета.

void setCost (int) – установить стоимость транспортировки груза.

int getFlyingTime() – получить время полета

int getCost() – получить стоимость транспортировки груза.

CargoDepartures & operator=(const CargoDepartures &) – оператор присваивания.

std::list<Cargo>& getCargoList() – получить список грузов.

friend QTextStream& operator<<(QTextStream&, const CargoDepartures &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, CargoDepartures &) – оператор чтения с файла

Класс **CargoArrivals –** класс грузовых прилетов.

Описание полей класса:

int baggageTime – время выдачи багажа.

bool delayed – задержка прилета.

Описание методов класса:

CargoArrivals(int, bool) – конструктор.

CargoArrivals (const CargoArrivals &) – конструктор копирования.

~ CargoArrivals () – деструктор.

void setBaggageTime(int) – установить время выдачи багажа.

void setDelayed(bool) – установить задержку прилета.

int getBaggageTime() – получить время выдачи багажа.

bool isDelayed () – проверка на задержку прилета.

CargoArrivals & operator=(const CargoArrivals &) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const CargoArrivals &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, CargoArrivals &) – оператор чтения с файла.

Класс **Cargo –** класс груз.

Описание полей класса:

QString name – имя.

QString surname – фамилия.

QString number – контактные данные.

QString cargoName – название груза.

float cargoWeight – вес груза.

Описание методов класса:

Cargo (const QString&, const QString&, const QString&, const QString&, int) – конструктор.

Cargo (const Cargo &) – конструктор копирования.

~ Cargo () – деструктор.

void setName(const QString&) – установить имя.

void setSurname(const QString&) – установить фамилию.

void setNumber(const QString&) – установить контактные данные.

void setCargoWeight(float) – установить тип места.

QString getName() – получить имя.

QString getSurname() – получить фамилию.

QString getNumber()– получить контактные данные.

QString getCargoName() – получить название груза.

float getCargoWeight() – получить вес груза.

int getSpace() – получить тип места.

Cargo & operator=(const Cargo &) – оператор присваивания.

friend QTextStream& operator<<(QTextStream&, const Cargo &) – оператор записи в файл.

friend QTextStream& operator>>(QTextStream&, Cargo &) – оператор чтения с файла.

Класс **Deque<T> -** шаблонный класс контейнера двунаправленной очереди.

Описание полей класса:

Node<T>\* head – указатель на первый элемент очереди.

Node<T>\* tail – указатель на последний элемент очереди.

int size – количество элементов кольца.

class Iterator – вложенный класс итератора.

Описание методов класса:

Deque() – конструктор.

Deque(const Deque<T>& obj) – конструктор копирования.

~Deque() – деструктор.

bool empty() – проверка на пустоту очереди.

int getSize() – получить размер очереди.

void pushFront(const T&) – добавить элемент в начало очереди.

void pushBack(const T&) – добавить элемент в конец очереди.

void popFront() – удалить элемент из начала очереди.

void popBack() – удалить элемент с конца очереди.

void clear() – удалить все элементы очереди.

void removeAt(int) – удалить элемент с заданной позиции.

Iterator begin() – установить итератор на начало контейнера.

Iterator end() – установить итератор на конец контейнера.

Iterator last() – установить итератор на последний элемент контейнера.

Deque<T>& operator=(const Deque<T>&) – оператор присваивания.

Структура **Node<T>** – шаблонная структура элемента контейнера

Описание полей структуры:

Node\* next – указатель на следующий элемент контейнера.

Node\* prev – указатель на предыдущий элемент контейнера.

T data – информация, хранящаяся в элементе.

Описание методов структуры:

Node(const T&) – конструктор.

Класс **Iterator –** класс итератор вложенный в Deque.

Описание полей класса:

Node<T>\* curr – указатель на текущий элемент.

Описание методов класса:

Iterator (Node<T>\*) – конструктор.

~ Iterator () – деструктор.

Iterator& operator++(int) – перемещение итератора на 1 позицию вперед.

Iterator& operator--(int) – перемещение итератора на 1 позицию назад.

Iterator& operator+=(int) – перемещение итератора на несколько позиций вперед.

T& operator\*() – получить данные по указателю.

bool operator==(const Iterator&) – оператор равенства.

bool operator!=(const Iterator&) – оператор неравенства.

Класс **Algorithm** – класс для манипуляций с контейнером

Описание методов класса:

static void sort(Deque<T>\*) – сортировка элементов контейнера.

static Deque<T>\* search(Deque<T>\*, T&) – поиск элементов равных T.

static int saveDeque(Deque<T>\*, const QString&) – сохранение контейнера в файл.

static int loadDeque(Deque<T>\*, const QString&) – чтение данных из файла в контейнер.

Класс **File** – класс для работы с файлами

Описание методов класса:

static int savePassengers(Deque<PassengerDepartures>\*, const QString&) – запись информации о пассажирах в файл.

static int saveCargo(Deque<CargoDepartures>\*, const QString&) – запись информации о грузах в файл.

static int loadPassengers(Deque<PassengerDepartures>\*, const QString&) – чтение информации о пассажирах из файла.

static int loadCargo(Deque<CargoDepartures>\*, const QString&) – чтение информации о грузах из файла.

Класс **Exception –** класс исключений.

Описание полей класса:

int code – код ошибки.

QString msg – текст ошибки.

Описание методов класса:

Exception (int code, const Qstring& msg) – конструктор.

void show() – вывод ошибки.

Класс **InputException –** класс исключений ввода.

Описание методов класса:

InputException (const Qstring& msg) – конструктор.

Класс **FileException –** класс исключений файлов.

Описание методов класса:

FileException (const Qstring& msg) – конструктор.

Класс **ContException –** класс исключений контейнера.

Описание методов класса:

ContException (const Qstring& msg) – конструктор.

Класс **OtherException –** класс остальных исключений.

Описание методов класса:

OtherException (const Qstring& msg) – конструктор.

Класс **Userinfo –** класс окна расписания рейсов.

Описание полей класса:

Ui::Userinfo \*ui – указатель на интерфейс.

AddFlightWindow\* addflightwindow – указатель на окно добавления рейса.

EditFlightWindow\* editflightwindow – указатель на окно изменения рейса.

Deque<PassengerDepartures>\* passdep – указатель на очередь пассажирских вылетов.

Deque<PassengerArrivals>\* passarr – указатель на очередь пассажирских прилетов.

Deque<CargoDepartures>\* cargodep – указатель на очередь грузовых вылетов.

Deque<CargoArrivals>\* cargoarr – указатель на очередь грузовых прилетов.

std::stack<Deque<PassengerDepartures>>\* passdepStack – указатель на стек очередей пассажирских вылетов.

std::stack<Deque<PassengerArrivals>>\* passarrStack – указатель на стек очередей пассажирских прилетов.

std::stack<Deque<CargoDepartures>>\* cargodepStack – указатель на стек очередей грузовых вылетов.

std::stack<Deque<CargoArrivals>>\* cargoarrStack – указатель на стек очередей грузовых прилетов.

std::stack<int>\* typeStack – указатель на стек для хранения типа рейсов.

int type – тип рейса.

Описание методов класса:

Userinfo (Qwidget \*parent) – конструктор.

~Userinfo() – деструктор.

void save() – сохранение всех данных в файлы.

void load() – чтение всех данных из файлов.

void checkTime() – проверка расписания с реальным временем.

void on\_passDepButton\_clicked() – просмотр информации о пассажирских вылетах.

void on\_passArrButton\_clicked() – просмотр информации о пассажирских прилетах.

void on\_cargoDepButton\_clicked() – просмотр информации о грузовых вылетах.

void on\_cargoArrButton\_clicked() – просмотр информации о грузовых прилетах.

void on\_addFlightButton\_clicked() – открытие окна добавления нового рейса.

void on\_deleteFlightButton\_clicked() – удаление рейса.

void on\_editButton\_clicked() – открытие окна изменения рейса.

void on\_undoButton\_clicked() – отмена последнего действия.

Класс **AddFlightWindow –** класс окна добавления рейсов.

Описание полей класса:

Ui::AddFlightWindow \*ui – указатель на интерфейс.

Deque<PassengerDepartures>\* passdep – указатель на очередь пассажирских вылетов.

Deque<PassengerArrivals>\* passarr – указатель на очередь пассажирских прилетов.

Deque<CargoDepartures>\* cargodep – указатель на очередь грузовых вылетов.

Deque<CargoArrivals>\* cargoarr – указатель на очередь грузовых прилетов.

std::stack<Deque<PassengerDepartures>>\* passdepStack – указатель на стек очередей пассажирских вылетов.

std::stack<Deque<PassengerArrivals>>\* passarrStack – указатель на стек очередей пассажирских прилетов.

std::stack<Deque<CargoDepartures>>\* cargodepStack – указатель на стек очередей грузовых вылетов.

std::stack<Deque<CargoArrivals>>\* cargoarrStack – указатель на стек очередей грузовых прилетов.

std::stack<int>\* typeStack – указатель на стек для хранения типа рейсов.

int type – тип рейса.

Описание методов класса:

AddFlightWindow (Deque<PassengerDepartures>\*, Deque<PassengerArrivals>\*, Deque<CargoDepartures>\*,Deque<CargoArrivals>\*, std::stack<Deque<PassengerDepartures>>\*,

std::stack<Deque<PassengerArrivals>>\*, std::stack< Deque<CargoDepartures>>\*, std::stack< Deque<CargoArrivals>>\*, std::stack<int>\*, QWidget \*) – конструктор.

~ AddFlightWindow () – деструктор.

void setType(int) – установить тип рейса.

void setTable() – выбор таблицы по типу рейса.

void setPassDepTable() – отображение таблицы пассажирских вылетов.

void setPassArrTable() – отображение таблицы пассажирских прилетов.

void setCargoDepTable() – отображение таблицы грузовых вылетов.

void setCargoArrTable() – отображение таблицы грузовых прилетов.

void addPassDep() – добавление пассажирского вылета.

void addPassArr() – добавление пассажирского прилета.

void addCargoDep() – добавление грузового вылета.

void addCargoArr() – добавление грузового прилета.

void on\_addButton\_clicked() – добавление рейса.

void on\_cancelButton\_clicked() – отмена добавления рейса.

Класс **EditFlightWindow –** класс окна изменения рейсов.

Описание полей класса:

Ui::EditFlightWindow \*ui – указатель на интерфейс.

Deque<PassengerDepartures>\* passdep – указатель на очередь пассажирских вылетов.

Deque<PassengerArrivals>\* passarr – указатель на очередь пассажирских прилетов.

Deque<CargoDepartures>\* cargodep – указатель на очередь грузовых вылетов.

Deque<CargoArrivals>\* cargoarr – указатель на очередь грузовых прилетов.

std::stack<Deque<PassengerDepartures>>\* passdepStack – указатель на стек очередей пассажирских вылетов.

std::stack<Deque<PassengerArrivals>>\* passarrStack – указатель на стек очередей пассажирских прилетов.

std::stack<Deque<CargoDepartures>>\* cargodepStack – указатель на стек очередей грузовых вылетов.

std::stack<Deque<CargoArrivals>>\* cargoarrStack – указатель на стек очередей грузовых прилетов.

std::stack<int>\* typeStack – указатель на стек для хранения типа рейсов.

int type – тип рейса.

int index – индекс рейса.

PassengerWindow \* passWind – указатель на окно бронирования билетов.

Описание методов класса:

EditFlightWindow (Deque<PassengerDepartures>\*, Deque<PassengerArrivals>\*, Deque<CargoDepartures>\*,Deque<CargoArrivals>\*, std::stack<Deque<PassengerDepartures>>\*,

std::stack<Deque<PassengerArrivals>>\*, std::stack< Deque<CargoDepartures>>\*, std::stack< Deque<CargoArrivals>>\*, std::stack<int>\*, QWidget \*) – конструктор.

~ EditFlightWindow () – деструктор.

void setValue(int, int) – установить тип и индекс рейса.

void setTable() – выбор таблицы по типу рейса.

void setPassDepTable() – отображение таблицы пассажирских вылетов.

void setPassArrTable() – отображение таблицы пассажирских прилетов.

void setCargoDepTable() – отображение таблицы грузовых вылетов.

void setCargoArrTable() – отображение таблицы грузовых прилетов.

void editPassDep() – изменение пассажирского вылета.

void editPassArr() –изменение пассажирского прилета.

void editCargoDep() –изменение грузового вылета.

void editCargoArr() –изменение грузового прилета.

void on\_editButton\_clicked() – изменение рейса.

void on\_cancelButton\_clicked() – отмена изменения рейса.

void on\_passengerButton\_clicked() – открытие окна бронирования билетов.

Класс **PassengerWindow –** класс окна бронирования билетов.

Описание полей класса:

Ui::PassengerWindow \*ui – указатель на интерфейс.

Deque<PassengerDepartures>\* passdep – указатель на очередь пассажирских вылетов.

Deque<CargoDepartures>\* cargodep – указатель на очередь грузовых вылетов.

std::stack<Deque<PassengerDepartures>>\* passdepStack – указатель на стек очередей пассажирских вылетов.

std::stack<Deque<CargoDepartures>>\* cargodepStack – указатель на стек очередей грузовых вылетов.

std::stack<int>\* typeStack – указатель на стек для хранения типа рейсов.

int type – тип рейса.

int index – индекс рейса.

Описание методов класса:

PassengerWindow (Deque<PassengerDepartures>\*, Deque<CargoDepartures>\*, std::stack<Deque<PassengerDepartures>>\*, std::stack<Deque<CargoDepartures>>\*, std::stack<int>\*, QWidget\*) – конструктор.

~ PassengerWindow () – деструктор.

void setValue(int, int) – установить тип и индекс рейса.

void setTable() – выбор таблицы по типу рейса.

void showPassenger() – отображение таблицы пассажиров.

void showCargo() – отображение таблицы грузов.

void editPassenger() – изменение информации о пассажирах.

void editCargo() –изменение информации о грузах.

void on\_addButton\_clicked() – добавление пассажира или груза.

void on\_dellButton\_clicked() – удаление пассажира или груза.

void on\_editButton\_clicked() – сохранение изменений.

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

# 4.1 Разработка схем алгоритмов

Метод search() класса Algorithm для поиска элемента в очереди. Схема алгоритма метода search() показана в приложении Б.

Метод on\_addButton\_clicked() класса PassengerWindow добавляет строку в таблицу для ввода данных о пассажире или грузе. Схема алгоритма метода on\_addButton\_clicked() показана в приложении В.

# 4.2 Разработка алгоритмов

Алгоритм метода checkTime() класса Userinfo

(метод проверки расписания с реальным временем)

Шаг 1. Начало

Шаг 2. Входные данные:

Deque<PassengerDepartures>\* passdep – указатель на очередь пассажирских вылетов.

Deque<PassengerArrivals>\* passarr – указатель на очередь пассажирских прилетов.

Deque<CargoDepartures>\* cargodep – указатель на очередь грузовых вылетов.

Deque<CargoArrivals>\* cargoarr – указатель на очередь грузовых прилетов.

Объявление переменных:

QDateTime current – текущее время

QDateTime schedule – время в расписании

Шаг 3. Вызов функции current = QDateTime::currentDateTime() для получения текущего времени

Шаг 4. Цикл по элементам очереди passdep

Шаг 5. Вызов метода schedule = getDate() элемента очереди для получения времени в расписании

Шаг 6. Если schedule > current, переход на шаг 8

Шаг 7. Вызов функции удаления элемента removeAt()

Шаг 8. Конец цикла по элементам очереди passdep

Шаг 9. Цикл по элементам очереди passarr

Шаг 10. Вызов метода schedule = getDate() элемента очереди для получения времени в расписании

Шаг 11. Если schedule > current, переход на шаг 13

Шаг 12. Вызов функции setDelay() элемента для установления задержки

Шаг 13. Конец цикла по элементам очереди passarr

Шаг 14. Цикл по элементам очереди cargodep

Шаг 15. Вызов метода schedule = getDate() элемента очереди для получения времени в расписании

Шаг 16. Если schedule > current, переход на шаг 18

Шаг 17. Вызов функции удаления элемента removeAt()

Шаг 18. Конец цикла по элементам очереди cargodep

Шаг 19. Цикл по элементам очереди cargoarr

Шаг 20. Вызов метода schedule = getDate() элемента очереди для получения времени в расписании

Шаг 21. Если schedule > current, переход на шаг 23

Шаг 22. Вызов функции setDelay() элемента для установления задержки

Шаг 23. Конец цикла по элементам очереди cargoarr

Шаг 24. Конец

Алгоритм метода sort() шаблонного класса Algorithm

(метод сортировки элементов контейнера)

Шаг 1. Начало

Шаг 2. Входные данные:

Deque<T>& deque – контейнер для сортировки

Объявление переменных:

int n – количество элементов контейнера

T key – элемент для сравнения  
Deque<T>::Iterator current, previous – итераторы контейнера

Шаг 3. Сохранение в переменную n размер контейнера (n = deque -> size())

Шаг 4. Если n <= 1 переход на шаг 19, иначе переход на шаг 5

Шаг 5. Установить итератор current на начало контейнера (current = deque.begin())

Шаг 6. Перемещение итератора current на 1 позицию вперед (current++)

Шаг 7. Если итератор в конце контейнера, переход на шаг КОНЕЦ, иначе переход на шаг 8

Шаг 8. Записать в key данные итератора (key = \*current)

Шаг 9. Приравнять итераторы previous и current

Шаг 10. Перемещение итератора previous на 1 позицию назад

(previous--)

Шаг 11. Если итератор previous не в начале контейнера и его данные меньше данных key (previous != deque.begin() && \*previous > key), переход на шаг 12, иначе переход на шаг 16

Шаг 12. Приравнять данные итераторов current и previous (\*current = \*previous)

Шаг 13. Перемещение итератора current на 1 позицию назад (current--)

Шаг 14. Перемещение итератора previous на 1 позицию назад

(previous--)

Шаг 15. Переход на шаг 11

Шаг 16. Перемещение итератора previous на 1 позицию назад

(previous--)

Шаг 17. Приравнять данные итератора current к key (\*current = key)

Шаг 18. Перемещение итератора current на 1 позицию вперед (current++)

Шаг 19. Конец

# 5 РЕЗУЛЬТАТ РАБОТЫ

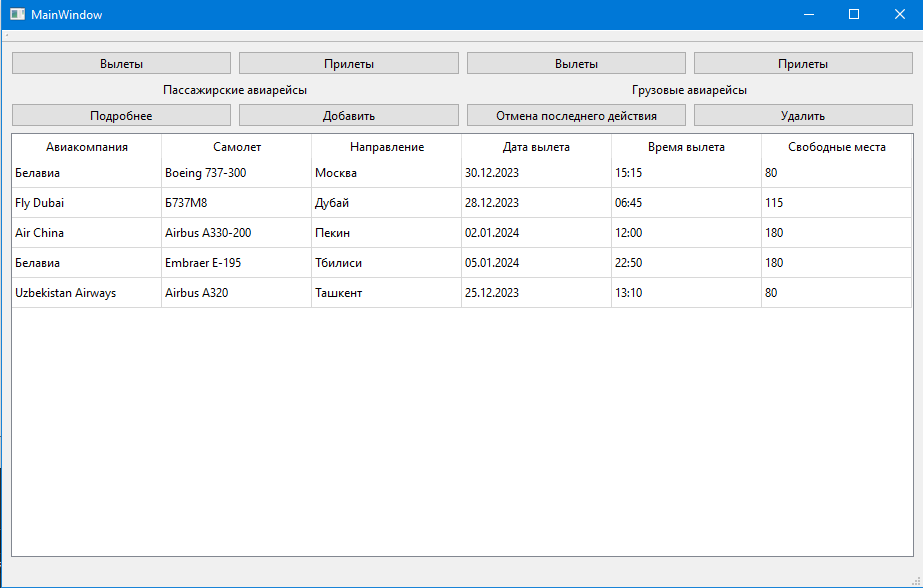


Рисунок 1 – Информация о пассажирских вылетах

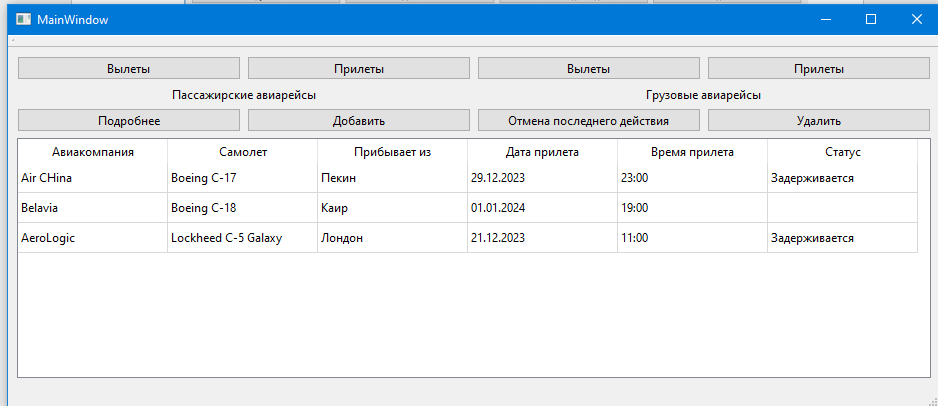


Рисунок 2 – Информация о грузовых прилетах

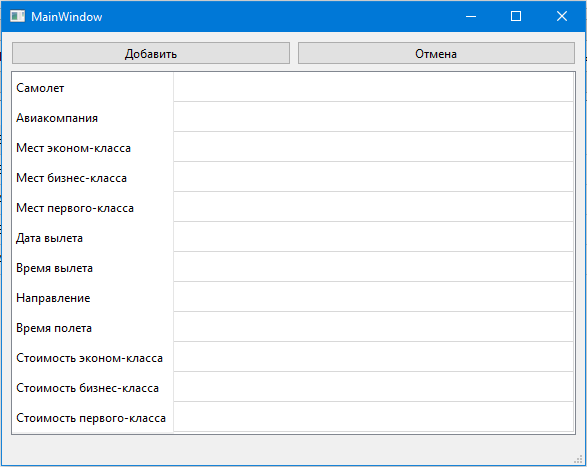


Рисунок 3 – Окно добавления рейсов

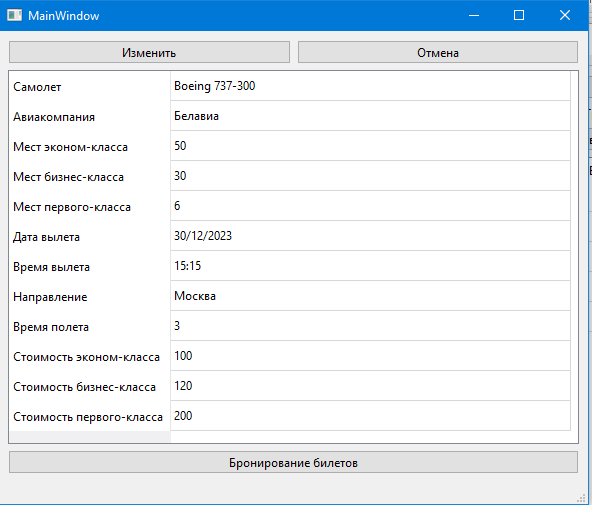


Рисунок 4 – Окно изменения рейсов

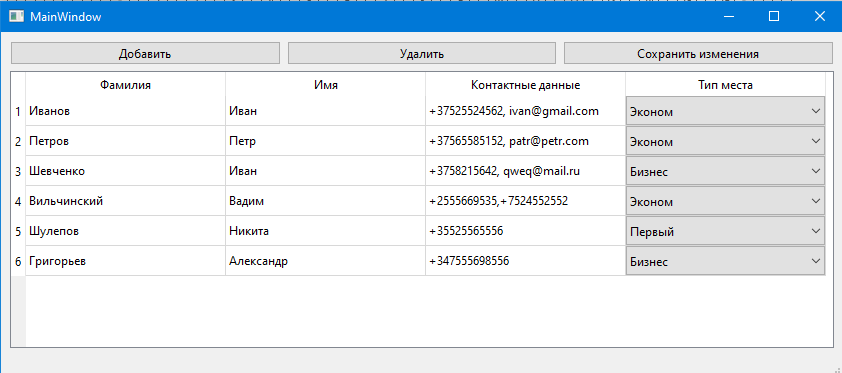


Рисунок 5 – Окно добавления пассажиров

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта реализована программа информационной системы аэропорта для обработки данных в аэропорте с удобным графическим пользовательским интерфейсом. Проект охватывает широкий спектр функциональности, связанный с пассажирскими и грузовыми перевозками, а также операционной деятельностью аэропорта. Были реализованы функции выдачи информации о нужном рейсе, добавления и изменения информации, бронирования билетов и грузоперевозок, отмены последнего действия. Так же были реализованы свои классы контейнера, итератора, алгоритма.

Системные требования:

– ОС – Windows 10

– Среда разработки – Qt Creator 11.0.3

# СПИСОК ЛИТЕРАТУРЫ

1. Объектно-ориентированное программирование на языке С++: учеб. пособие / Ю. А. Луцик, В. Н. Комличенко. – Минск : БГУИР, 2008.

2. Конструирование программ и языки программирования: метод. указания по курсовому проектированию для студ. спец. I-40 02 01 “Вычислительные машины, системы и сети” для всех форм обуч. / сост. А. В. Бушкевич, А. М. Ковальчук, И. В. Лукьянова. – Минск : БГУИР, 2009.

# ПРИЛОЖЕНИЕ А

*(обязательное)*

Диаграмма классов

# ПРИЛОЖЕНИЕ Б

*(обязательное)*

Схема метода search()

# ПРИЛОЖЕНИЕ В

*(обязательное)*

Схема метода on\_addButton\_clicked()

# ПРИЛОЖЕНИЕ Г

*(обязательное)*

Код программы

# ПРИЛОЖЕНИЕ Д

*(обязательное)*

Ведомость документов