

elevator

0.5

Создано системой Doxygen 1.9.1

1	Симулятор пассажирского лифта	1
2	Алфавитный указатель классов	3
2.1	Классы	3
3	Список файлов	5
3.1	Файлы	5
4	Классы	7
4.1	Класс Control	7
4.1.1	Подробное описание	9
4.1.2	Конструктор(ы)	9
4.1.2.1	Control()	9
4.1.2.2	~Control()	10
4.1.3	Методы	10
4.1.3.1	AddPassengerToQueue()	10
4.1.3.2	getCurrentTime()	10
4.1.3.3	getElevatorButton()	11
4.1.3.4	getElevatorButtons()	12
4.1.3.5	getElevatorDestination()	12
4.1.3.6	getElevatorIndicator()	13
4.1.3.7	getElevatorPosition()	13
4.1.3.8	getFloorDnButton()	14
4.1.3.9	getFloorDnButtons()	14
4.1.3.10	getFloorUpButton()	15
4.1.3.11	getFloorUpButtons()	15
4.1.3.12	isElevatorAccelerating()	16
4.1.3.13	isElevatorAchievedDestination()	16
4.1.3.14	isElevatorBreaking()	17
4.1.3.15	isElevatorDoorsClosing()	18
4.1.3.16	isElevatorDoorsOpened()	18
4.1.3.17	isElevatorDoorsOpening()	18
4.1.3.18	isElevatorEmpty()	20
4.1.3.19	isElevatorEmptyAfterUnloading()	20
4.1.3.20	isElevatorGoingDn()	21
4.1.3.21	isElevatorGoingUniformly()	21
4.1.3.22	isElevatorGoingUp()	22
4.1.3.23	isElevatorStaying()	23
4.1.3.24	isElevatorStayingDoorsClosed()	23
4.1.3.25	MakeStep()	24
4.1.3.26	PrintButtonsState()	29
4.1.3.27	PrintElevatorState()	30
4.1.3.28	PrintPassengerState()	31
4.1.3.29	PrintStatistics()	33

4.1.3.30	ReadTimeTable()	35
4.1.3.31	SetElevatorDestination()	36
4.1.3.32	SetElevatorIndicator()	36
4.1.3.33	unsetDnButton()	37
4.1.3.34	unsetUpButton()	37
4.1.4	Данные класса	38
4.1.4.1	timeAccelerating	38
4.1.4.2	timeBreaking	38
4.1.4.3	timeClosing	38
4.1.4.4	timeEntering	39
4.1.4.5	timeLeaving	39
4.1.4.6	timeOpening	39
4.1.4.7	veloUniform	39
4.1.4.8	waitingTime	39
4.2	Класс Elevator	40
4.2.1	Подробное описание	40
4.2.2	Конструктор(ы)	40
4.2.2.1	Elevator()	40
4.2.3	Документация по друзьям класса и функциям, относящимся к классу	41
4.2.3.1	Control	41
4.3	Класс FloorButtons	41
4.3.1	Подробное описание	41
4.3.2	Конструктор(ы)	41
4.3.2.1	FloorButtons()	41
4.3.3	Документация по друзьям класса и функциям, относящимся к классу	42
4.3.3.1	Control	42
4.4	Структура myParams	42
4.4.1	Подробное описание	42
4.4.2	Данные класса	42
4.4.2.1	arbitraryParam	43
4.4.2.2	started	43
4.5	Класс Passenger	43
4.5.1	Подробное описание	43
4.5.2	Конструктор(ы)	43
4.5.2.1	Passenger()	43
4.5.3	Методы	44
4.5.3.1	operator<()	44
4.5.4	Документация по друзьям класса и функциям, относящимся к классу	44
4.5.4.1	Control	44
4.5.4.2	Elevator	44
4.6	Структура PassengerProperties	45
4.6.1	Подробное описание	45
4.6.2	Данные класса	45

4.6.2.1 criticalWaitTime	45
4.6.2.2 floorDeparture	45
4.6.2.3 floorDestination	46
4.6.2.4 pInverseStartWaiting	46
4.6.2.5 pInverseStopWaiting	46
4.6.2.6 pStartGoing	46
4.6.2.7 timeInit	46
4.7 Класс Queue	47
4.7.1 Подробное описание	47
4.7.2 Конструктор(ы)	47
4.7.2.1 Queue()	47
4.7.3 Методы	47
4.7.3.1 addPassenger()	48
4.7.4 Документация по друзьям класса и функциям, относящимся к классу	48
4.7.4.1 Control	48
5 Файлы	49
5.1 Файл Control.cpp	49
5.1.1 Подробное описание	49
5.2 Файл Control.h	50
5.2.1 Подробное описание	51
5.3 Файл Elevator.cpp	51
5.3.1 Подробное описание	52
5.4 Файл Elevator.h	52
5.4.1 Подробное описание	53
5.4.2 Перечисления	54
5.4.2.1 ElevatorAcceleration	54
5.4.2.2 ElevatorDoorsStatus	54
5.4.2.3 ElevatorIndicator	55
5.4.2.4 ElevatorStatus	55
5.5 Файл FloorButtons.h	56
5.5.1 Подробное описание	56
5.6 Файл main.cpp	57
5.6.1 Подробное описание	58
5.6.2 Функции	58
5.6.2.1 CONTROLSYSTEM()	58
5.6.2.2 main()	64
5.6.3 Переменные	65
5.6.3.1 elevatorCapacity	65
5.6.3.2 maxFloor	66
5.6.3.3 maxTime	66
5.6.3.4 numberOfElevators	66
5.6.3.5 numberOfFloors	66

5.7 Файл Passenger.h	67
5.7.1 Подробное описание	67
5.7.2 Перечисления	68
5.7.2.1 PassengerStatus	68
5.8 Файл Queue.h	68
5.8.1 Подробное описание	69
Предметный указатель	71

Глава 1

Симулятор пассажирского лифта

Данный программный комплекс реализует моделирование работы кабины пассажирского лифта и требует от студентов разработки системы управления лифтом.

Задача состоит в том, чтобы как можно эффективнее (быстрее) перевезти пассажиров, появляющихся на этажах

За каждого перевезенного пассажира начисляется "штраф", равный сумме времени ожидания пассажиром лифта и времени его нахождения внутри кабины. За пассажиров, так и не дождавшихся лифта, или оставшихся в лифте к моменту окончания симуляции (т.е. не доставленных на нужный этаж) начисляются большие штрафы.

Требуется разработать алгоритм, при котором суммарный "штраф" будет как можно меньше!

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

25 мая 2021 г.

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Control	Основной класс — симулятор пассажирского лифта	7
Elevator	Класс — кабина лифта	40
FloorButtons	Класс — кнопки на этажах	41
myParams	Структура, содержащая пользовательские параметры	42
Passenger	Класс — пассажир	43
PassengerProperties	Параметры пассажиров	45
Queue	Класс — очередь пассажиров	47

Глава 3

Список файлов

3.1 Файлы

Полный список файлов.

Control.cpp	Файл кода с описанием класса Control	49
Control.h	Заголовочный файл с описанием основного класса Control	50
Elevator.cpp	Файл кода с описанием класса Elevator	51
Elevator.h	Заголовочный файл с описанием класса Elevator и сопутствующих структур . . .	52
FloorButtons.h	Заголовочный файл с описанием класса FloorButtons	56
main.cpp	Основной файл программы elevator	57
Passenger.h	Заголовочный файл с описанием класса Passenger и сопутствующих структур . .	67
Queue.h	Заголовочный файл с описанием класса Queue	68

Глава 4

Классы

4.1 Класс Control

Основной класс — симулятор пассажирского лифта

```
#include <Control.h>
```

Открытые члены

- `Control` (`size_t numberOfFloors`, `size_t numberOfElevators`, `size_t capacityOfElevator`)
Инициализирующий конструктор
- `~Control` ()
Деструктор
- `void MakeStep` ()
Функция выполнения шага моделирования по времени
- `size_t getCurrentTime` () const
Функция запроса текущего времени
- `void SetElevatorDestination` (`size_t elevatorNumber`, `size_t destination`)
Функция задания назначения лифту
- `void SetElevatorIndicator` (`size_t elevatorNumber`, `ElevatorIndicator indicator`)
Функция задания состояния индикатора лифта (лампочка со стрелочкой, которую видят пассажиры)
- `size_t getElevatorDestination` (`size_t elevatorNumber`) const
Функция запроса текущего назначения
- `ElevatorIndicator getElevatorIndicator` (`size_t elevatorNumber`) const
Функция запроса текущего состояния индикатора
- `bool isElevatorAchievedDestination` (`size_t elevatorNumber`) const
Проверка того, что лифт завершил выполнение текущего назначения
- `bool isElevatorEmptyAfterUnloading` (`size_t elevatorNumber`) const
Проверка того, что лифт пустой после выхода очередного пассажира
- `bool isElevatorEmpty` (`size_t elevatorNumber`) const
Проверка того, что кабина лифта пуста
- `bool isElevatorGoingUp` (`size_t elevatorNumber`) const
Проверка того, что кабина лифта движется вверх
- `bool isElevatorGoingDn` (`size_t elevatorNumber`) const

- Проверка того, что кабина лифта движется вниз
- bool `isElevatorStaying` (size_t elevatorNumber) const
- Проверка того, что кабина лифта стоит (не движется)
- bool `isElevatorGoingUniformly` (size_t elevatorNumber) const
- Проверка того, что кабина лифта движется равномерно
- bool `isElevatorAccelerating` (size_t elevatorNumber) const
- Проверка того, что кабина лифта ускоряется (разгоняется)
- bool `isElevatorBreaking` (size_t elevatorNumber) const
- Проверка того, что кабина лифта замедляется (тормозит)
- bool `isElevatorDoorsOpening` (size_t elevatorNumber) const
- Проверка того, что у кабины лифта в данный момент открываются двери
- bool `isElevatorDoorsClosing` (size_t elevatorNumber) const
- Проверка того, что у кабины лифта в данный момент закрываются двери
- bool `isElevatorDoorsOpened` (size_t elevatorNumber) const
- Проверка того, что у кабины лифта в данный момент открыты двери
- bool `isElevatorStayingDoorsClosed` (size_t elevatorNumber) const
- Проверка того, что у кабина лифта в данный момент стоит на этаже с закрытыми дверьми
- double `getElevatorPosition` (size_t elevatorNumber) const
- Функция запроса текущего положения лифта
- const std::vector< bool > & `getFloorUpButtons` () const
- Функция запроса состояний кнопок "вверх" на этажах
- const std::vector< bool > & `getFloorDnButtons` () const
- Функция запроса состояний кнопок "вниз" на этажах
- bool `getFloorUpButton` (size_t floor) const
- Функция запроса состояния кнопки "вверх" на конкретном этаже
- bool `getFloorDnButton` (size_t floor) const
- Функция запроса состояния кнопки "вниз" на конкретном этаже
- void `unsetUpButton` (size_t floor)
- Функция сброса (выключения) кнопки "вверх" на конкретном этаже
- void `unsetDnButton` (size_t floor)
- Функция сброса (выключения) кнопки "вниз" на конкретном этаже
- const std::vector< bool > & `getElevatorButtons` (size_t elevatorNumber) const
- Функция запроса состояний кнопок в кабине лифта
- bool `getElevatorButton` (size_t elevatorNumber, size_t floor) const
- Функция запроса состояния конкретной кнопки в кабине лифта
- void `AddPassengerToQueue` (const `PassengerProperties` &passProp_)
- Функция добавления пассажира в очередь
- void `ReadTimeTable` (const std::string &fileName_)
- Функция чтения расписания появления пассажиров на этажах
- void `PrintElevatorState` (size_t elevatorNumber, const std::string &fname="") const
- Функция печати в файл или на экран состояния лифта в текущий момент времени
- void `PrintButtonsState` (const std::string &fname="") const
- Функция печати в файл или на экран состояния кнопок в кабинах и на этажах в текущий момент времени
- void `PrintPassengerState` (const std::string &fname="") const
- Функция печати в файл или на экран событий, произошедших с пассажирами за последний шаг (последнюю секунду)
- void `PrintStatistics` (bool passengersDetails, const std::string &fname="") const
- Функция печати в файл или на экран итоговой статистики, включая итоговый "рейтинг" (чем меньше - тем лучше!)

Открытые атрибуты

- `const size_t waitingTime = 5`
Время ожидания до закрытия дверей (если только кто-то не нажмет кнопку "ход" раньше)
- `const size_t timeEntering = 2`
Время между входами двух пассажиров в лифт
- `const size_t timeLeaving = 2`
Время между выходами двух пассажиров в лифт
- `const size_t timeOpening = 4`
Время открывания дверей
- `const size_t timeClosing = 4`
Время закрывания дверей
- `const size_t timeAccelerating = 4`
Время разгона лифта
- `const size_t timeBreaking = 3`
Время торможения лифта
- `const double veloUniform = 0.25`
Скорость равномерно движения лифта (в долях этажа)

4.1.1 Подробное описание

Основной класс — симулятор пассажирского лифта

См. определение в файле Control.h строка 25

4.1.2 Конструктор(ы)

4.1.2.1 Control()

```
Control::Control (
    size_t numberOfFloors,
    size_t numberOfElevators,
    size_t capacityOfElevator )
```

Инициализирующий конструктор

Аргументы

in	numberOfFloors	число этажей (считая подвальный)
in	numberOfElevators	число лифтовых кабин
in	capacityOfElevator	емкость каждой кабины

См. определение в файле Control.cpp строка 22

```
23 : floorButtons(new FloorButtons(numberOfFloors)), queue(new Queue(numberOfFloors)), time(0)
24 {
25     for (size_t id = 0; id < numberOfElevators; ++id)
```

```

26     elevators.emplace_back(new Elevator(numberOfFloors, capacityOfElevator, id));
27 } //Control(...)

```

4.1.2.2 ~Control()

```
Control::~Control ( )
```

Деструктор

См. определение в файле Control.cpp строка 30

```

31 {
32 } //~Control()

```

4.1.3 Методы

4.1.3.1 AddPassengerToQueue()

```
void Control::AddPassengerToQueue (
    const PassengerProperties & passProp_ )
```

Функция добавления пассажира в очередь

Аргументы

in	pass↔ Prop_	константная ссылка на список параметров пассажира
----	----------------	---

См. определение в файле Control.cpp строка 504

```

505 {
506     queue->addPassenger(passProp_);
507 } //AddPassengerToQueue(...)

```

4.1.3.2 getCurrentTime()

```
size_t Control::getCurrentTime ( ) const [inline]
```

Функция запроса текущего времени

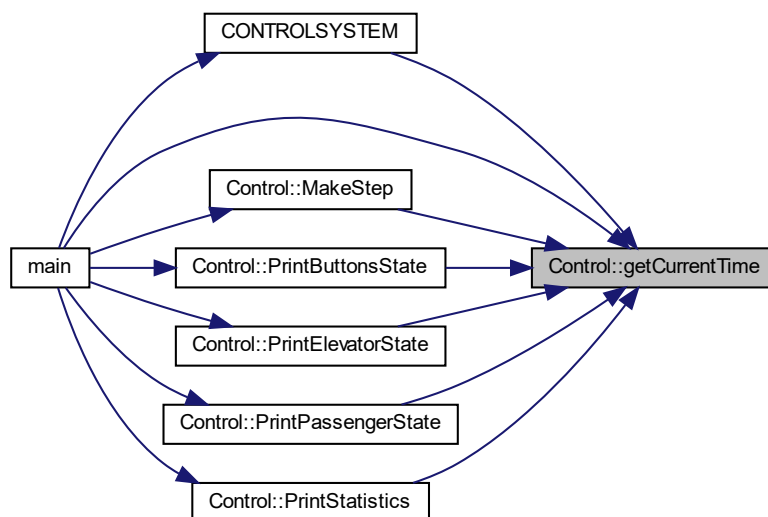
Возвращает

текущее время в секундах от начала моделирования

См. определение в файле Control.h строка 110

```
111 {
112     return time;
113 }
```

Граф вызова функции:



4.1.3.3 getElevatorButton()

```
bool Control::getElevatorButton (
    size_t elevatorNumber,
    size_t floor ) const [inline]
```

Функция запроса состояния конкретной кнопки в кабине лифта

Аргументы

in	elevatorNumber	номер кабины, состояние кнопки в которой запрашивается
in	floor	кнопка, состояние которой запрашивается

Возвращает

признак нажатости конкретной кнопки в соответствующей кабине лифта

См. определение в файле Control.h строка 411

```
412 {
413     return elevators[elevatorNumber]->getButton(floor);
414 }
```

4.1.3.4 getElevatorButtons()

```
const std::vector<bool>& Control::getElevatorButtons (
    size_t elevatorNumber ) const [inline]
```

Функция запроса состояний кнопок в кабине лифта

Аргументы

in	elevatorNumber	номер кабины, состояние кнопок в которой запрашивается
----	----------------	--

Возвращает

константную ссылку на вектор признаков нажатости кнопок в соответствующей кабине лифта

См. определение в файле Control.h строка 400

```
401 {
402     return elevators[elevatorNumber]->getButtons();
403 }
```

4.1.3.5 getElevatorDestination()

```
size_t Control::getElevatorDestination (
    size_t elevatorNumber ) const [inline]
```

Функция запроса текущего назначения

Аргументы

in	elevatorNumber	номер кабины, назначение которой запрашивается
----	----------------	--

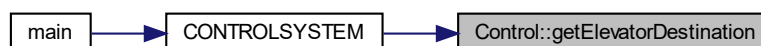
Возвращает

этаж назначения соответствующей кабины лифта

См. определение в файле Control.h строка 140

```
141 {
142     return elevators[elevatorNumber]->getDestination();
143 }
```

Граф вызова функции:



4.1.3.6 getElevatorIndicator()

```
ElevatorIndicator Control::getElevatorIndicator (  
    size_t elevatorNumber ) const [inline]
```

Функция запроса текущего состояния индикатора

Аргументы

in	elevatorNumber	номер кабины, состояние индикатора которой запрашивается
----	----------------	--

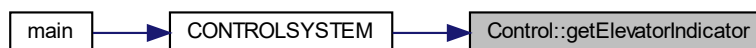
Возвращает

состояние индикатора (вверх, вниз или в обе стороны) соответствующей кабины лифта

См. определение в файле Control.h строка 150

```
151 {  
152     return elevators[elevatorNumber]->getIndicator();  
153 }
```

Граф вызова функции:



4.1.3.7 getElevatorPosition()

```
double Control::getElevatorPosition (  
    size_t elevatorNumber ) const [inline]
```

Функция запроса текущего положения лифта

Аргументы

in	elevatorNumber	номер кабины, положение которой запрашивается
----	----------------	---

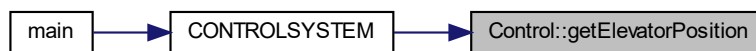
Возвращает

дробное число; если дробная часть нулевая - то лифт на этаже

См. определение в файле Control.h строка 332

```
333 {  
334     return elevators[elevatorNumber]->getPosition();  
335 }
```

Граф вызова функции:



4.1.3.8 getFloorDnButton()

```
bool Control::getFloorDnButton (
    size_t floor ) const [inline]
```

Функция запроса состояния кнопки "вниз" на конкретном этаже

Аргументы

in	floor	номер этажа, на котором опрашивается кнопка
----	-------	---

Возвращает

признак нажатости кнопки "вниз" на соответствующем этаже

См. определение в файле Control.h строка 370

```
371 {
372     return floorButtons->getDnButton(floor);
373 }
```

4.1.3.9 getFloorDnButtons()

```
const std::vector<bool>& Control::getFloorDnButtons ( ) const [inline]
```

Функция запроса состояний кнопок "вниз" на этажах

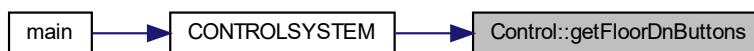
Возвращает

константную ссылку на вектор признаков нажатости кнопок "вниз" на этажах

См. определение в файле Control.h строка 350

```
351 {
352     return floorButtons->getDnButtons();
353 }
```

Граф вызова функции:



4.1.3.10 getFloorUpButton()

```
bool Control::getFloorUpButton (
    size_t floor ) const [inline]
```

Функция запроса состояния кнопки "вверх" на конкретном этаже

Аргументы

in	floor	номер этажа, на котором опрашивается кнопка
----	-------	---

Возвращает

признак нажатости кнопки "вверх" на соответствующем этаже

См. определение в файле Control.h строка 360

```
361 {
362     return floorButtons->getUpButton(floor);
363 }
```

4.1.3.11 getFloorUpButtons()

```
const std::vector<bool>& Control::getFloorUpButtons ( ) const [inline]
```

Функция запроса состояний кнопок "вверх" на этажах

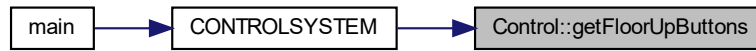
Возвращает

константную ссылку на вектор признаков нажатости кнопок "вверх" на этажах

См. определение в файле Control.h строка 341

```
342 {
343     return floorButtons->getUpButtons();
344 }
```

Граф вызова функции:



4.1.3.12 isElevatorAccelerating()

```
bool Control::isElevatorAccelerating (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что кабина лифта ускоряется (разгоняется)

Может быть истинной только при закрытых дверях, когда лифт движется вверх или вниз с ускорением (разгоном)

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак движения соответствующей кабины лифта с ускорением (при разгоне)

См. определение в файле Control.h строка 262

```
263 {
264     return elevators[elevatorNumber]->isAccelerating();
265 }
```

4.1.3.13 isElevatorAchievedDestination()

```
bool Control::isElevatorAchievedDestination (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что лифт завершил выполнение текущего назначения

Назначение считается выполненным, когда

- лифт приехал на тот этаж, куда его послали, остановился, и
- выполнено одно из трех условий:
 1. в нем есть хотя бы 1 пассажир - тогда открылись двери
 2. он пустой, а на этаже, на который он прибыл, нажата хотя бы одна кнопка - тогда тоже открылись двери
 3. он пустой, а на этаже, на который он прибыл, не нажато ни одной кнопки - тогда двери не открываются

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

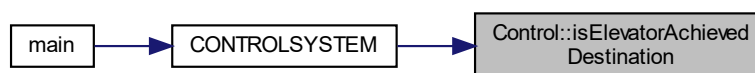
Возвращает

признак выполнения назначения соответствующей кабиной лифта

См. определение в файле Control.h строка 167

```
168 {  
169     return elevators[elevatorNumber]->isDestinationAchieved(floorButtons->getUpButtons(), floorButtons->getDnButtons());  
170 }
```

Граф вызова функции:



4.1.3.14 isElevatorBreaking()

```
bool Control::isElevatorBreaking (  
    size_t elevatorNumber ) const [inline]
```

Проверка того, что кабина лифта замедляется (тормозит)

Может быть истинной только при закрытых дверях, когда лифт движется вверх или вниз с замедлением (тормозит)

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак движения соответствующей кабины лифта с замедлением (при торможении)

См. определение в файле Control.h строка 274

```
275 {  
276     return elevators[elevatorNumber]->isBreaking();  
277 }
```

4.1.3.15 isElevatorDoorsClosing()

```
bool Control::isElevatorDoorsClosing (  
    size_t elevatorNumber ) const    [inline]
```

Проверка того, что у кабины лифта в данный момент закрываются двери

Может быть истинной только при нахождении лифта на этаже, когда он не движется

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак того, что у соответствующей кабины лифта происходит закрывание дверей

См. определение в файле Control.h строка 298

```
299  {  
300      return elevators[elevatorNumber]->isDoorsClosing();  
301  }
```

4.1.3.16 isElevatorDoorsOpened()

```
bool Control::isElevatorDoorsOpened (  
    size_t elevatorNumber ) const    [inline]
```

Проверка того, что у кабины лифта в данный момент открыты двери

Может быть истинной только при нахождении лифта на этаже, когда он не движется

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак того, что у соответствующей кабины лифта открыты двери

См. определение в файле Control.h строка 310

```
311  {  
312      return elevators[elevatorNumber]->isDoorsOpened();  
313  }
```

4.1.3.17 isElevatorDoorsOpening()

```
bool Control::isElevatorDoorsOpening (  
    size_t elevatorNumber ) const    [inline]
```


Проверка того, что у кабины лифта в данный момент открываются двери

Может быть истинной только при нахождении лифта на этаже, когда он не движется

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак того, что у соответствующей кабины лифта происходит открывание дверей

См. определение в файле Control.h строка 286

```
287 {
288     return elevators[elevatorNumber]->isDoorsOpening();
289 }
```

4.1.3.18 isElevatorEmpty()

```
bool Control::isElevatorEmpty (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что кабина лифта пуста

Состояние лифта не проверяется - стоит он или едет, открыты или нет двери

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак пустоты соответствующей кабины лифта

См. определение в файле Control.h строка 200

```
201 {
202     return elevators[elevatorNumber]->isEmpty();
203 }
```

4.1.3.19 isElevatorEmptyAfterUnloading()

```
bool Control::isElevatorEmptyAfterUnloading (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что лифт пустой после выхода очередного пассажира

Возвращает true, если лифт стоит на этаже, и после выхода очередного пассажира лифт оказался пустым — возможно, при этом имеет смысл "включить" индикатор в оба направления, чтобы в любом случае зашел пассажир, стоящий первым в очереди. Но это не обязательно - у Вас может быть своя логика!

Если индикатор лифта "горит" в состоянии both (в обе стороны), при этом он пустой или нет - не важно, и в лифт входит пассажир, то индикатор автоматически переключается в то направление, какую кнопку он нажал, входя в лифт.

Будьте осторожны, "зажигайте" состояние индикатора both (в обе стороны) аккуратно, но и без него обойтись будет трудно!

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак пустоты соответствующей кабины лифта после выхода очередного пассажира

См. определение в файле Control.h строка 188

```
189 {
190     return elevators[elevatorNumber]->isEmptyAfterUnloading();
191 }
```

4.1.3.20 isElevatorGoingDn()

```
bool Control::isElevatorGoingDn (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что кабина лифта движется вниз

Может быть истинной только при закрытых дверях; едет ли лифт равномерно или с ускорением - не важно

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак движения вниз соответствующей кабины лифта

См. определение в файле Control.h строка 224

```
225 {
226     return elevators[elevatorNumber]->isGoingDn();
227 }
```

4.1.3.21 isElevatorGoingUniformly()

```
bool Control::isElevatorGoingUniformly (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что кабина лифта движется равномерно

Может быть истинной только при закрытых дверях, когда лифт движется равномерно (не разгоняется и не тормозит)

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

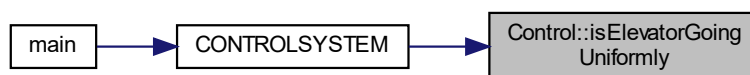
признак равномерного движения соответствующей кабины лифта

См. определение в файле Control.h строка 250

```

251 {
252     return elevators[elevatorNumber]->isGoingUniformly();
253 }
```

Граф вызова функции:



4.1.3.22 isElevatorGoingUp()

```

bool Control::isElevatorGoingUp (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что кабина лифта движется вверх

Может быть истиной только при закрытых дверях; едет ли лифт равномерно или с ускорением - не важно

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак движения вверх соответствующей кабины лифта

См. определение в файле Control.h строка 212

```

213 {
214     return elevators[elevatorNumber]->isGoingUp();
215 }
```

4.1.3.23 isElevatorStaying()

```
bool Control::isElevatorStaying (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что кабина лифта стоит (не движется)

Может быть истиной не только, когда лифт на этаже (при этом состояние дверей не важно), но и между этажами, когда лифт, к примеру ехал вверх, но поступило новое назначение: а этом случае он тормозит, в течение 1 секунды стоит на месте (в этот момент данное условие будет выполненным), а потом разгоняется вниз

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак стояния на месте соответствующей кабины лифта

См. определение в файле Control.h строка 238

```
239 {
240     return elevators[elevatorNumber]->isStaying();
241 }
```

4.1.3.24 isElevatorStayingDoorsClosed()

```
bool Control::isElevatorStayingDoorsClosed (
    size_t elevatorNumber ) const [inline]
```

Проверка того, что у кабина лифта в данный момент стоит на этаже с закрытыми дверьми

Может быть истинной только при нахождении лифта на этаже

Аргументы

in	elevatorNumber	номер кабины, для которой проверяется данное условие
----	----------------	--

Возвращает

признак того, что соответствующая кабина лифта стоит на этаже с закрытыми дверьми

См. определение в файле Control.h строка 322

```
323 {
324     return elevators[elevatorNumber]->isStayingDoorsClosed();
325 }
```

4.1.3.25 MakeStep()

```
void Control::MakeStep ( )
```

Функция выполнения шага моделирования по времени

См. определение в файле Control.cpp строка 102

```
103 {
104     passStatBuffer.resize(0);
105
106     TimeIncrement();
107
108     //Проверка появления пассажиров на этажах и их передача
109     //в соответствующие списки ожидающих на этажах
110     FindAppearingPassengers();
111
112     //Нажатие появившимися пассажирами кнопок на этажах
113     PressingFloorButtons();
114
115     //Пассажиры, ждавшие слишком долго, уходят с этажей, и за это начисляется большой штраф
116     LeavingFloors();
117
118     //Посадка пассажиров в лифты на этажах
119     //цикл по этажам:
120     for (size_t pos = 0; pos < floorButtons->dnButtons.size(); ++pos)
121     {
122         //std::vector<ElevatorIndicator> ind = { ElevatorIndicator::up, ElevatorIndicator::down };
123
124         //for (auto& indValue : ind)
125         //{
126             std::vector<Elevator*> elevOnFloor;
127             for (auto& e : elevators)
128             {
129                 if ((e->position / 100 == pos) && (e->doorsStatus == ElevatorDoorsStatus::openedLoading))
130                     if (e->getNumberOfPassengers() < e->capacity)
131                         elevOnFloor.push_back(e.get());
132                 else
133                 {
134                     if (e->timeToSelfProgramme == 0)
135                     {
136                         e->timeToSelfProgramme = waitingTime;
137                         e->doorsStatus = ElevatorDoorsStatus::waiting;
138                     }
139                 }
140             }
141
142             if (elevOnFloor.size() > 1)
143                 for (auto& pe : elevOnFloor)
144                     pe->timeToSelfProgramme = elevOnFloor[0]->timeToSelfProgramme;
145
146             if (elevOnFloor.size() > 0)
147                 if (elevOnFloor[0]->timeToSelfProgramme == 0)
148                 {
149                     auto& pass = queue->passOnFloor[pos];
150
151                     size_t passToUp = 0, passToDn = 0;
152                     for (auto& p : pass)
153                     {
154                         if (p.getFloorDestination() > pos)
155                             ++passToUp;
156                         else
157                             ++passToDn;
158                     }
159
160                     if (passToUp == 0)
161                         for (auto& e : elevOnFloor)
162                             if (e->getIndicator() == ElevatorIndicator::up)
163                             {
164                                 e->timeToSelfProgramme = waitingTime;
165                                 e->doorsStatus = ElevatorDoorsStatus::waiting;
166                             };
167
168                     if (passToDn == 0)
169                         for (auto& e : elevOnFloor)
170                             if (e->getIndicator() == ElevatorIndicator::down)
171                             {
172                                 e->timeToSelfProgramme = waitingTime;
173                                 e->doorsStatus = ElevatorDoorsStatus::waiting;
174                             };
175
176                     std::vector<Passenger> stillWaiting;
177
178                     for (auto& p : pass)
```

```

180     {
181
182         std::vector<Elevator*> elevAppropriate;
183         bool inverseWay = false;
184
185         bool inv = p.PerformInverseProbability(getCurrentTime());
186
187         for (auto& e : elevOnFloor)
188         {
189             if ((p.getFloorDestination() > pos) && (e->timeToSelfProgramme == 0) && (e->doorsStatus ==
ElevatorDoorsStatus::openedLoading) && (e->indicator == ElevatorIndicator::up || e->indicator ==
ElevatorIndicator::both))
190             {
191                 e->lastChechedPassenger = std::max(p.id, e->lastChechedPassenger);
192
193                 if (e->getNumberOfPassengers() < e->capacity)
194                     elevAppropriate.push_back(e);
195                 else
196                 {
197                     e->timeToSelfProgramme = waitingTime;
198                     e->doorsStatus = ElevatorDoorsStatus::waiting;
199                 }
200             }
201
202             if ((p.getFloorDestination() < pos) && (e->timeToSelfProgramme == 0) && (e->doorsStatus ==
ElevatorDoorsStatus::openedLoading) && (e->indicator == ElevatorIndicator::down || e->indicator ==
ElevatorIndicator::both))
203             {
204                 e->lastChechedPassenger = std::max(p.id, e->lastChechedPassenger);
205
206                 if (e->getNumberOfPassengers() < e->capacity)
207                     elevAppropriate.push_back(e);
208                 else
209                 {
210                     e->timeToSelfProgramme = waitingTime;
211                     e->doorsStatus = ElevatorDoorsStatus::waiting;
212                 }
213             }
214         }
215
216         //Если человек готов сесть не туда
217         if ((elevAppropriate.size() == 0) && (inv))
218         {
219
220             for (auto& e : elevOnFloor)
221             {
222                 if ((e->timeToSelfProgramme == 0) && (e->doorsStatus ==
ElevatorDoorsStatus::openedLoading))
223                 {
224                     e->lastChechedPassenger = std::max(p.id, e->lastChechedPassenger);
225
226                     if (e->lastChechedPassenger <= p.id)
227                     {
228                         if (e->getNumberOfPassengers() < e->capacity)
229                         {
230                             elevAppropriate.push_back(e);
231                             inverseWay = true;
232                         }
233                         else
234                         {
235                             e->timeToSelfProgramme = waitingTime;
236                             e->doorsStatus = ElevatorDoorsStatus::waiting;
237                         }
238                     }
239                 }
240             }
241         }
242
243         if (elevAppropriate.size() > 0)
244         {
245             size_t elevWithSmallestPass = 0;
246             size_t smallestPass = elevAppropriate[0]->getNumberOfPassengers();
247
248             for (size_t numb = 1; numb < elevAppropriate.size(); ++numb)
249                 if (elevAppropriate[numb]->getNumberOfPassengers() < smallestPass)
250                 {
251                     elevWithSmallestPass = numb;
252                     smallestPass = elevAppropriate[numb]->getNumberOfPassengers();
253                 }
254
255             Elevator* e = elevAppropriate[elevWithSmallestPass];
256
257             e->passengers.push_back(p);
258             e->passengers.back().status = PassengerStatus::going;
259             e->passengers.back().timeStart = getCurrentTime();
260
261

```

```

262         passStatBuffer.push_back("time = " + std::to_string(getCurrentTime()) \
263         + "\tPassenger #" + std::to_string(e->passengers.back().id) \
264         + "\tfrom floor #" + std::to_string(e->passengers.back().getFloorDeparture()) \
265         + " to floor #" + std::to_string(e->passengers.back().getFloorDestination()) \
266         + (inverseWay ? "*" : "")) \
267         + "\tentered the elevator #" + std::to_string(e->myid));
268     e->timeToSelfProgramme = timeEntering;
269     e->buttons[e->passengers.back().getFloorDestination()] = true;
270     if (e->indicator == ElevatorIndicator::both)
271     {
272         if (e->passengers.back().properties.floorDestination > pos)
273             e->indicator = ElevatorIndicator::up;
274         else
275             e->indicator = ElevatorIndicator::down;
276     }
277     }
278     else
279     {
280         stillWaiting.push_back(p);
281     }
282 } //for p
283
284 pass.clear();
285 pass = std::move(stillWaiting);
286
287 }
288 } //for indValue
289 } //for pos
290
291
292 //Обработка нажатия кнопки "Ход"
293 for (auto& e : elevators)
294 {
295     if (e->doorsStatus == ElevatorDoorsStatus::waiting)
296     {
297         if (e->isGoingButtonPressed() && (e->timeToSelfProgramme > 1))
298         {
299             e->timeToSelfProgramme = 1;
300         }
301     }
302 }
303
304 //Обработка движения лифта
305 for (auto& e : elevators)
306 {
307     if (e->timeToSelfProgramme == 0)
308     {
309         auto pos = e->position / 100;
310         auto& pass = queue->passOnFloor[pos];
311
312         switch (e->status)
313         {
314             case ElevatorStatus::staying:
315             {
316                 //3.1. Обработываем стоящий лифт
317                 switch (e->doorsStatus)
318                 {
319                     case ElevatorDoorsStatus::openedUnloading:
320                     {
321                         e->timeToSelfProgramme = timeLeaving - 1;
322                         auto it = std::find_if(e->passengers.begin(), e->passengers.end(), \
323                         [=](const Passenger& p) {return p.getFloorDestination() == pos; });
324                         if (it != e->passengers.end())
325                         {
326                             it->status = PassengerStatus::arrived;
327                             it->timeFinish = getCurrentTime();
328                             queue->finished.push_back(*it);
329                             passStatBuffer.push_back("time = " + std::to_string(getCurrentTime()) \
330                             + "\tPassenger #" + std::to_string(it->id) \
331                             + "\tfrom floor #" + std::to_string(it->getFloorDeparture()) \
332                             + " to floor #" + std::to_string(it->getFloorDestination()) \
333                             + "\tgot off the elevator #" + std::to_string(e->myid) \
334                             + " (appeared t = " + std::to_string(it->properties.timeInit) \
335                             + ", in elevator t = " + std::to_string(it->timeStart) + ")");
336                             e->passengers.erase(it);
337                             break;
338                         } // if it!=
339
340                         e->doorsStatus = ElevatorDoorsStatus::openedLoading;
341                         e->lastCheckedPassenger = 0;
342
343                         break;
344                     } //case ElevatorDoorsStatus::openedUnloading:
345
346                     case ElevatorDoorsStatus::waiting:
347                     {
348                         e->doorsStatus = ElevatorDoorsStatus::closing;
349                         e->timeToSelfProgramme = timeClosing - 1;
350                         break;

```



```

349         } //case ElevatorDoorsStatus::waiting:
350
351     case ElevatorDoorsStatus::closing:
352     {
353         //Только что закрывший двери лифт - делаем двери закрытыми
354         e->doorsStatus = ElevatorDoorsStatus::closed;
355         break;
356     } //case ElevatorDoorsStatus::closing:
357
358     case ElevatorDoorsStatus::closed:
359     {
360         //Если есть назначение для стоящего лифта с закрытыми дверьми - разгоняем
361
362         //вверх
363         if ((e->position / 100) < e->destinationFloor)
364         {
365             e->status = ElevatorStatus::movingUp;
366             e->acceleration = ElevatorAcceleration::accelerating;
367             e->timeToSelfProgramme = timeAccelerating - 1;
368         } //if ((e->position...
369
370         //вниз
371         if (((e->position + 99) / 100) > e->destinationFloor)
372         {
373             e->status = ElevatorStatus::movingDn;
374             e->acceleration = ElevatorAcceleration::accelerating;
375             e->timeToSelfProgramme = timeAccelerating - 1;
376         } //if (((e->position...
377
378         //3.1.4. Если лифт прибыл в пункт назначения - открываем двери
379         if (((e->position / 100) == e->destinationFloor) &&
380             ((e->position % 100) == 0))
381         {
382             //но открываем двери только если либо он непустой, либо снаружи нажата кнопка:
383             if ((e->getNumberOfPassengers() > 0) ||
384                 (floorButtons->getDnButton(e->destinationFloor)) ||
385                 (floorButtons->getUpButton(e->destinationFloor)))
386             {
387                 e->doorsStatus = ElevatorDoorsStatus::opening;
388                 e->buttons[pos] = false;
389                 e->timeToSelfProgramme = timeOpening - 1;
390
391                 if ((floorButtons->getDnButton(e->destinationFloor)) &&
392                     (e->indicator == ElevatorIndicator::down || e->indicator == ElevatorIndicator::both))
393                     floorButtons->unsetDnButton(e->destinationFloor);
394
395                 if ((floorButtons->getUpButton(e->destinationFloor)) &&
396                     (e->indicator == ElevatorIndicator::up || e->indicator == ElevatorIndicator::both))
397                     floorButtons->unsetUpButton(e->destinationFloor);
398             } //if ((e->getNumberOfPassengers() > 0) ||...
399         } //if (((e->position...
400         break;
401     } //case ElevatorDoorsStatus::closed:
402
403     case ElevatorDoorsStatus::opening:
404     {
405         //Только что открывший двери лифт - делаем двери открытыми
406         e->doorsStatus = ElevatorDoorsStatus::openedUnloading;
407         break;
408     }
409
410     } //switch (e->doorsStatus)
411
412     break;
413 } // case ElevatorStatus::staying
414
415
416 case ElevatorStatus::movingUp:
417 case ElevatorStatus::movingDn:
418 {
419     switch (e->acceleration)
420     {
421     case ElevatorAcceleration::breaking:
422     {
423         e->acceleration = ElevatorAcceleration::uniform;
424         e->status = ElevatorStatus::staying;
425         break;
426     } //case ElevatorAcceleration::breaking:
427
428     case ElevatorAcceleration::accelerating:
429     {
430         int sign = (e->status == ElevatorStatus::movingUp) ? 1 : -1;
431         e->position += sign * (int)(100*veloUniform);
432         e->acceleration = ElevatorAcceleration::uniform;
433         break;
434     } //case ElevatorAcceleration::accelerating:
435

```

```

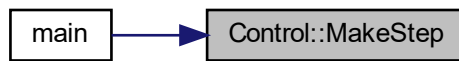
436         case ElevatorAcceleration::uniform:
437         {
438             int sign = (e->status == ElevatorStatus::movingUp) ? 1 : -1;
439             if (abs((int)((e->position - 100 * e->destinationFloor))) != (int)(100 * veloUniform))
440                 e->position += sign * (int)(100 * veloUniform);
441             else
442             {
443                 e->acceleration = ElevatorAcceleration::breaking;
444                 e->position += sign * 12;
445                 e->timeToSelfProgramme = timeBreaking - 1;
446             } //else
447         } //case ElevatorAcceleration::uniform:
448
449     } //switch (e->acceleration)
450
451     break;
452 } //case ElevatorStatus::movingUp:
453 //case ElevatorStatus::movingDn:
454 }
455
456
457 } //if (e->timeToSelfProgramme == 0)
458 else // если продолжается предыдущая операция
459 {
460     if ((e->status == ElevatorStatus::movingUp) ||
461         (e->status == ElevatorStatus::movingDn))
462     {
463         int sign = (e->status == ElevatorStatus::movingUp) ? 1 : -1;
464
465         if (e->acceleration == ElevatorAcceleration::accelerating)
466         {
467             switch (e->timeToSelfProgramme)
468             {
469                 case 3:
470                     e->position += sign * 5;
471                     break;
472
473                 case 2:
474                     e->position += sign * 8;
475                     break;
476
477                 case 1:
478                     e->position += sign * 12;
479                     break;
480             } //switch (e->timeToSelfProgramme)
481         } //if (e->acceleration == ElevatorAcceleration::accelerating)
482
483         if (e->acceleration == ElevatorAcceleration::breaking)
484         {
485             switch (e->timeToSelfProgramme)
486             {
487                 case 2:
488                     e->position += sign * 8;
489                     break;
490
491                 case 1:
492                     e->position += sign * 5;
493                     break;
494             } //switch (e->timeToSelfProgramme)
495         } //if (e->acceleration == ElevatorAcceleration::breaking)
496     } //if ((e->status == ElevatorStatus::movingUp) || (e->status == ElevatorStatus::movingDn))
497
498     --(e->timeToSelfProgramme);
499 } //else
500 } //for e : elevators
501 } //MakeStep()

```

Граф вызовов:



Граф вызова функции:



4.1.3.26 PrintButtonsState()

```
void Control::PrintButtonsState (
    const std::string & fname = "" ) const
```

Функция печати в файл или на экран состояния кнопок в кабинах и на этажах в текущий момент времени

Если вызывается без аргумента - печать на экран, если с аргументом - печать в файл с данным именем.

Если вызывать эту функцию на каждом шаге по времени - получится полный протокол состояния всех кнопок в кабинах и на этажах

Аргументы

in	fname	имя файла, в корорый выводить состояние
----	-------	---

См. определение в файле Control.cpp строка 566

```

567 {
568     std::ofstream fout;
569     if (fname != "")
570     {
571         if (getCurrentTime() <= 1)
572             fout.open(fname);
573         else
574             fout.open(fname, std::ios_base::app);
575     } //if (fname != "")
576
577     std::ostream& str = (fname == "") ? std::cout : fout;
578
579     str << "time = " << getCurrentTime() << ": " << std::endl;
580     for (auto& e : elevators)
581     {
582         str << " in elevator #" << e->myid << ": ";
583         for (size_t i = 0; i < e->buttons.size(); ++i)
584             if (e->getButton(i))
585                 str << i << " ";
586         str << std::endl;
587     } //for e
588     str << " on floors: ";
589     for (size_t i = 0; i < floorButtons->upButtons.size(); ++i)
590     {
591         if (floorButtons->getUpButton(i) || floorButtons->getDnButton(i))
592         {
593             str << "# " << i << "(";
594             if (floorButtons->getUpButton(i))
595                 str << "up ";
596             if (floorButtons->getDnButton(i))
597                 str << "dn";
  
```

```

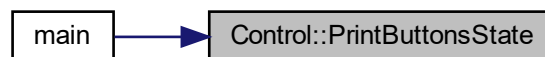
598         str « " ) ";
599     } //if (floorButtons->...
600 } //for i
601 str « std::endl « std::endl;
602
603 if (fname != "")
604     fout.close();
605 } //PrintButtonsState(...)

```

Граф вызовов:



Граф вызова функции:



4.1.3.27 PrintElevatorState()

```

void Control::PrintElevatorState (
    size_t elevatorNumber,
    const std::string & fname = "" ) const

```

Функция печати в файл или на экран состояния лифта в текущий момент времени

Если вызывается без аргумента - печать на экран, если с аргументом - печать в файл с данным именем.

Если вызывать эту функцию на каждом шаге по времени - получится полный протокол работы кабины лифта

Аргументы

in	elevatorNumber	номер кабины, состояние которой печатается
in	fname	имя файла, в который выводить состояние

См. определение в файле Control.cpp строка 545

```

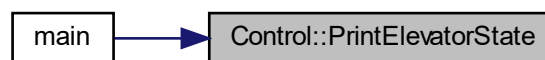
546 {
547     std::ofstream fout;
548     if (fname != "")
549     {
550         if (getCurrentTime() <= 1)
551             fout.open(fname);
552         else
553             fout.open(fname, std::ios_base::app);
554     } //if (fname != "")
555     std::ostream& str = (fname == "") ? std::cout : fout;
556     str << "time = " << getCurrentTime() << ", \telev[" << i << "]: " \
557         << elevators[i]->getStateString() << std::endl;
558     if (fname != "")
559         fout.close();
560 } //PrintElevatorState(...)

```

Граф вызовов:



Граф вызова функции:



4.1.3.28 PrintPassengerState()

```

void Control::PrintPassengerState (
    const std::string & fname = "" ) const

```

Функция печати в файл или на экран событий, произошедших с пассажирами за последний шаг (последнюю секунду)

Если вызывается без аргумента - печать на экран, если с аргументом - печать в файл с данным именем.

Выводит сообщения:

- о появлении пассажира на этаже

- о входе пассажира в кабину лифта
- о выходе пассажира из лифта
- о том, что пассажир ушел, не дождавшись лифта

Предупреждения

Нужна только для отладки. В процессе работы системы управления эта информация недоступна и не может быть использована!

Аргументы

in	fname	имя файла, в корорый выводить состояние
----	-------	---

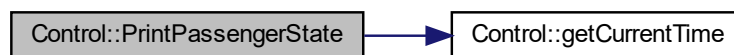
См. определение в файле Control.cpp строка 608

```

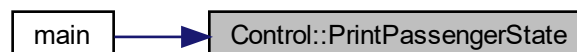
609 {
610     std::ofstream fout;
611     if (fname != "")
612     {
613         if (getCurrentTime() <= 1)
614             fout.open(fname);
615         else
616             fout.open(fname, std::ios_base::app);
617     } //if (fname != "")
618
619     std::ostream& str = (fname == "") ? std::cout : fout;
620
621     if (passStatBuffer.size() > 0)
622         for (auto& st : passStatBuffer)
623             str << st << std::endl;
624
625     if (fname != "")
626         fout.close();
627 } //PrintPassengerState(...)

```

Граф вызовов:



Граф вызова функции:



4.1.3.29 PrintStatistics()

```
void Control::PrintStatistics (
    bool passengersDetails,
    const std::string & fname = "" ) const
```

Функция печати в файл или на экран итоговой статистики, включая итоговый "рейтинг" (чем меньше - тем лучше!)

Аргументы

in	passengersDetails	признак печати статистики по каждому пассажиру
in	fname	имя файла, в который выводить состояние

См. определение в файле Control.cpp строка 639

```
640 {
641     std::ofstream fout;
642     if (fname != "")
643         fout.open(fname);
644
645     std::ostream& str = (fname == "") ? std::cout : fout;
646
647     std::vector<Passenger> allPass(queue->passengers);
648     for (auto& pf : queue->passOnFloor)
649         std::copy(pf.begin(), pf.end(), std::back_inserter(allPass));
650     for (auto& e : elevators)
651         std::copy(e->passengers.begin(), e->passengers.end(), std::back_inserter(allPass));
652     std::copy(queue->finished.begin(), queue->finished.end(), std::back_inserter(allPass));
653
654     std::sort(allPass.begin(), allPass.end());
655
656     size_t numInElevator = 0, numOnFloors = 0, numLeaved = 0;
657     size_t penaltyFinished = 0, penaltyInElevator = 0, penaltyOnFloors = 0, penaltyLeaved = 0;
658
659     if (passengersDetails)
660         str << "Passangers:" << std::endl;
661
662     for (auto& p : allPass)
663     {
664         switch (p.status)
665         {
666             case PassengerStatus::arrived:
667             {
668                 penaltyFinished += p.timeFinish - p.getTimeInit();
669                 if (passengersDetails)
670                     str << "#" << p.id << ", penalty = " << p.timeFinish - p.getTimeInit() \
671                     << " (init = " << p.getTimeInit() << ", started = " << p.timeStart << ", finished = " << p.timeFinish << ")" \
672                     << std::endl;
673                 break;
674             } //case PassengerStatus::arrived:
675
676             case PassengerStatus::going:
677             {
678                 ++numInElevator;
679                 penaltyInElevator += getCurrentTime() - p.getTimeInit();
680                 if (passengersDetails)
681                     str << "#" << p.id << ", penalty = " << getCurrentTime() - p.getTimeInit() \
682                     << " (init = " << p.getTimeInit() << ", started = " << p.timeStart << ", STILL IN ELEVATOR!!!" << ")" \
683                     << std::endl;
684                 break;
685             } //case PassengerStatus::going:
686
687             case PassengerStatus::waiting:
688             {
689                 ++numOnFloors;
690                 penaltyOnFloors += getCurrentTime() - p.getTimeInit();
691                 if (passengersDetails)
692                     str << "#" << p.id << ", penalty = " << getCurrentTime() - p.getTimeInit() \
693                     << " (init = " << p.getTimeInit() << ", STILL WAITING FOR ELEVATOR!!!" << ")" \
694                     << std::endl;
695                 break;
696             } //case PassengerStatus::waiting:
697
698             case PassengerStatus::leaved:
699             {
700                 ++numLeaved;
701                 penaltyLeaved += p.properties.criticalWaitTime * 5;
```

```

702         if (passengersDetails)
703             str << "#" << p.id << ", penalty = " << p.properties.criticalWaitTime * 5 \
704             << " (init = " << p.getTimeInit() << ", LEAVED THE FLOOR!!!" << ")" \
705             << std::endl;
706         break;
707     } //case PassengerStatus::leaved:
708 } //switch (p.status)
709 } //for p
710
711 if (passengersDetails)
712     str << std::endl;
713
714 size_t waitingTime = 0, goingTime = 0, totalTime = 0;
715 for (auto& p : queue->finished)
716 {
717     if (p.status != PassengerStatus::leaved)
718     {
719         waitingTime += p.timeStart - p.getTimeInit();
720         goingTime += p.timeFinish - p.timeStart;
721         totalTime += p.timeFinish - p.getTimeInit();
722     }
723     else
724     {
725         waitingTime += p.properties.criticalWaitTime;
726         totalTime += p.properties.criticalWaitTime;
727     }
728 } //for p
729
730 str << "Number of passengers, that have finished the trip: " << queue->finished.size() << std::endl;
731 str << " average waiting time = " << 1.0 * waitingTime / queue->finished.size() << std::endl;
732 str << " average going time = " << 1.0 * goingTime / queue->finished.size() << std::endl;
733 str << " average total time = " << 1.0 * totalTime / queue->finished.size() << std::endl;
734 str << "Penalty for them = " << penaltyFinished << std::endl;
735 str << std::endl;
736
737 str << "Still waiting on floors = " << numOnFloors << std::endl;
738 str << "Penalty for them = " << penaltyOnFloors << std::endl;
739 str << std::endl;
740
741 str << "Still in elevator = " << numInElevetor << std::endl;
742 str << "Penalty for them = " << penaltyInElevetor << std::endl;
743 str << std::endl;
744
745 str << "Leaved the floors, because of too large waiting time = " << numLeaved << std::endl;
746 str << "Penalty for them = " << penaltyLeaved << std::endl;
747 str << std::endl;
748
749 str << "TOTAL PENALTY = " << penaltyFinished + penaltyInElevetor + penaltyOnFloors + penaltyLeaved << std::endl;
750
751 if (fname != "")
752     fout.close();
753 } //PrintStatistics(...)

```

Граф вызовов:



Граф вызова функции:



4.1.3.30 ReadTimeTable()

```
void Control::ReadTimeTable (
    const std::string & fileName_ )
```

Функция чтения расписания появления пассажиров на этажах

Аргументы

in	file← Name_	константная ссылка на имя файла с расписанием
----	----------------	---

См. определение в файле Control.cpp строка 509

```

510 {
511     std::ifstream fi(fileName_);
512
513     char str[255];
514     fi.getline(str, 100, '\n');
515
516     PassengerProperties passProp;
517     int N;
518
519     while (!fi.eof())
520     {
521         fi » N;
522         fi.get();
523         fi » passProp.timeInit;
524         fi.get();
525         fi » passProp.floorDeparture;
526         fi.get();
527         fi » passProp.floorDestination;
528         fi.get();
529         fi » passProp.criticalWaitTime;
530         fi.get();
531         fi » passProp.pInverseStartWaiting;
532         fi.get();
533         fi » passProp.pInverseStopWaiting;
534         fi.get();
535         fi » passProp.pStartGoing;
536
537         queue->addPassenger(passProp);
538     }
539
540     fi.close();
541     fi.clear();
542 }
```

Граф вызова функции:



4.1.3.31 SetElevatorDestination()

```

void Control::SetElevatorDestination (
    size_t elevatorNumber,
    size_t destination ) [inline]
  
```

Функция задания назначения лифту

Аргументы

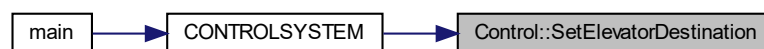
in	elevatorNumber	номер кабины, которой задается назначение
in	destination	этаж назначения

См. определение в файле Control.h строка 120

```

121 {
122     elevators[elevatorNumber]->setDestination(destination);
123 }
  
```

Граф вызова функции:



4.1.3.32 SetElevatorIndicator()

```

void Control::SetElevatorIndicator (
    size_t elevatorNumber,
    ElevatorIndicator indicator ) [inline]
  
```

Функция задания состояния индикатора лифта (лампочка со стрелочкой, которую видят пассажиры)

Аргументы

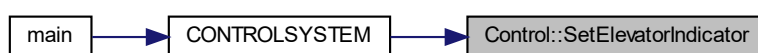
in	elevatorNumber	номер кабины, которой задается назначение
in	indicator	устанавливаемое значение индикатора (вверх, вниз или в обе стороны)

См. определение в файле Control.h строка 130

```

131 {
132     elevators[elevatorNumber]->setIndicator(indicator);
133 }
```

Граф вызова функции:



4.1.3.33 unsetDnButton()

```

void Control::unsetDnButton (
    size_t floor ) [inline]
```

Функция сброса (выключения) кнопки "вниз" на конкретном этаже

Аргументы

in	floor	номер этажа, на котором сбрасывается (гасится) кнопка. Добавлена на всякий случай; если на этаже есть пассажиры, и они увидят, что нужная им кнопка погасла - то они ее нажмут
----	-------	--

См. определение в файле Control.h строка 390

```

391 {
392     floorButtons->unsetDnButton(floor);
393 }
```

4.1.3.34 unsetUpButton()

```

void Control::unsetUpButton (
    size_t floor ) [inline]
```

Функция сброса (выключения) кнопки "вверх" на конкретном этаже

Аргументы

in	floor	номер этажа, на котором сбрасывается (гасится) кнопка Добавлена на всякий случай; если на этаже есть пассажиры, и они увидят, что нужная им кнопка погасла - то они ее нажмут
----	-------	---

См. определение в файле Control.h строка 380

```
381 {  
382     floorButtons->unsetUpButton(floor);  
383 }
```

4.1.4 Данные класса

4.1.4.1 timeAccelerating

```
const size_t Control::timeAccelerating = 4
```

Время разгона лифта

См. определение в файле Control.h строка 80

4.1.4.2 timeBreaking

```
const size_t Control::timeBreaking = 3
```

Время торможения лифта

См. определение в файле Control.h строка 83

4.1.4.3 timeClosing

```
const size_t Control::timeClosing = 4
```

Время закрывания дверей

См. определение в файле Control.h строка 77

4.1.4.4 timeEntering

```
const size_t Control::timeEntering = 2
```

Время между входами двух пассажиров в лифт

См. определение в файле Control.h строка 68

4.1.4.5 timeLeaving

```
const size_t Control::timeLeaving = 2
```

Время между выходами двух пассажиров в лифт

См. определение в файле Control.h строка 71

4.1.4.6 timeOpening

```
const size_t Control::timeOpening = 4
```

Время открывания дверей

См. определение в файле Control.h строка 74

4.1.4.7 veloUniform

```
const double Control::veloUniform = 0.25
```

Скорость равномерно движения лифта (в долях этажа)

См. определение в файле Control.h строка 86

4.1.4.8 waitingTime

```
const size_t Control::waitingTime = 5
```

Время ожидания до закрытия дверей (если только кто-то не нажмет кнопку "ход" раньше)

См. определение в файле Control.h строка 65

Объявления и описания членов классов находятся в файлах:

- [Control.h](#)
- [Control.cpp](#)

4.2 Класс Elevator

Класс — кабина лифта

```
#include <Elevator.h>
```

Открытые члены

- `Elevator` (`size_t numberOfFloors`, `size_t maxCapacity`, `size_t id`)
Инициализирующий конструктор

Друзья

- class `Control`

4.2.1 Подробное описание

Класс — кабина лифта

См. определение в файле `Elevator.h` строка 79

4.2.2 Конструктор(ы)

4.2.2.1 Elevator()

```
Elevator::Elevator (
    size_t numberOfFloors,
    size_t maxCapacity,
    size_t id )
```

Инициализирующий конструктор

Аргументы

in	<code>numberOfFloors</code>	количество этажей (считая подвал)
in	<code>maxCapacity</code>	максимальная вместимость кабины лифта
in	<code>id</code>	порядковый номер кабины лифта

См. определение в файле `Elevator.cpp` строка 17

```
18 : buttons(numberOfFloors, false)
19 , capacity(maxCapacity)
20 , myid(id)
21 , timeToSelfProgramme(0)
22 , position(0)
23 , acceleration(ElevatorAcceleration::uniform)
24 , destinationFloor(0)
25 , doorsStatus(ElevatorDoorsStatus::closed)
```

```
26     , indicator(ElevatorIndicator::both)
27     , status(ElevatorStatus::staying)
28 {
29 }
```

4.2.3 Документация по друзьям класса и функциям, относящимся к классу

4.2.3.1 Control

friend class [Control](#) [friend]

См. определение в файле Elevator.h строка 81

Объявления и описания членов классов находятся в файлах:

- [Elevator.h](#)
- [Elevator.cpp](#)

4.3 Класс FloorButtons

Класс — кнопки на этажах

```
#include <FloorButtons.h>
```

Открытые члены

- [FloorButtons](#) (size_t numberOfFloors)

Друзья

- class [Control](#)

4.3.1 Подробное описание

Класс — кнопки на этажах

См. определение в файле FloorButtons.h строка 19

4.3.2 Конструктор(ы)

4.3.2.1 FloorButtons()

```
FloorButtons::FloorButtons (
    size_t numberOfFloors ) [inline]
```

Инициализирующий конструктор

Аргументы

in	numberOfFloors	количество этажей (включая подвал)
----	----------------	------------------------------------

См. определение в файле FloorButtons.h строка 75

```
76 : upButtons(numberOfFloors, false)
77 , dnButtons(numberOfFloors, false)
78 {}
```

4.3.3 Документация по друзьям класса и функциям, относящимся к классу

4.3.3.1 Control

friend class [Control](#) [friend]

См. определение в файле FloorButtons.h строка 22

Объявления и описания членов класса находятся в файле:

- [FloorButtons.h](#)

4.4 Структура myParams

Структура, содержащая пользовательские параметры

Открытые атрибуты

- size_t [arbitraryParam](#) = 0
Некоторый произвольный параметр, инициализированный значением "0".
- bool [started](#) = false
Признак того, что лифт выполняет работу

4.4.1 Подробное описание

Структура, содержащая пользовательские параметры

Данные параметры сохраняются при переходе от одного шага по времени к следующим, их можно использовать для запоминания необходимых параметров. Набор членов-данных структуры можно модифицировать по собственному усмотрению.

См. определение в файле main.cpp строка 39

4.4.2 Данные класса

4.4.2.1 arbitraryParam

```
size_t myParams::arbitraryParam = 0
```

Некоторый произвольный параметр, инициализированный значением "0".

См. определение в файле main.cpp строка 42

4.4.2.2 started

```
bool myParams::started = false
```

Признак того, что лифт выполняет работу

См. определение в файле main.cpp строка 45

Объявления и описания членов структуры находятся в файле:

- [main.cpp](#)

4.5 Класс Passenger

Класс — пассажир

```
#include <Passenger.h>
```

Открытые члены

- [Passenger](#) (size_t id_, const [PassengerProperties](#) &properties_)
Инициализирующий конструктор
- bool [operator<](#) (const [Passenger](#) &ps) const
Оператор сравнения для возможности сортировки пассажиров по порядковому номеру

Друзья

- class [Control](#)
- class [Elevator](#)

4.5.1 Подробное описание

Класс — пассажир

См. определение в файле Passenger.h строка 54

4.5.2 Конструктор(ы)

4.5.2.1 Passenger()

```
Passenger::Passenger (
    size_t id_,
    const PassengerProperties & properties_ ) [inline]
```

Инициализирующий конструктор

Аргументы

in	id_	порядковый номер пассажира
in	properties↔ —	параметры пассажира

См. определение в файле Passenger.h строка 102

```
103     : id(id_), properties(properties_), timeStart(-1), timeFinish(-1), status(PassengerStatus::waiting)
104     {};
```

4.5.3 Методы

4.5.3.1 operator<()

```
bool Passenger::operator< (
    const Passenger & ps ) const    [inline]
```

Оператор сравнения для возможности сортировки пассажиров по порядковому номеру

См. определение в файле Passenger.h строка 107

```
108     {
109         return id < ps.id;
110     }
```

4.5.4 Документация по друзьям класса и функциям, относящимся к классу

4.5.4.1 Control

```
friend class Control    [friend]
```

См. определение в файле Passenger.h строка 56

4.5.4.2 Elevator

```
friend class Elevator    [friend]
```

См. определение в файле Passenger.h строка 57

Объявления и описания членов класса находятся в файле:

- [Passenger.h](#)

4.6 Структура PassengerProperties

Параметры пассажиров

```
#include <Passenger.h>
```

Открытые атрибуты

- `size_t timeInit`
Время появления пассажира на этаже
- `size_t floorDeparture`
Этаж, с которого пассажир отправляется
- `size_t floorDestination`
Этаж, на который пассажир едет
- `size_t criticalWaitTime`
Время ожидания, после которого пассажир уходит
- `double pInverseStartWaiting`
Вероятность сесть в лифт, едущий в неверном направлении в начале ожидания
- `double pInverseStopWaiting`
Вероятность сесть в лифт, едущий в неверном направлении в конце ожидания
- `double pStartGoing`
Вероятность нажать кнопку "ход", не дожидаясь закрытия дверей

4.6.1 Подробное описание

Параметры пассажиров

См. определение в файле Passenger.h строка 27

4.6.2 Данные класса

4.6.2.1 criticalWaitTime

```
size_t PassengerProperties::criticalWaitTime
```

Время ожидания, после которого пассажир уходит

См. определение в файле Passenger.h строка 39

4.6.2.2 floorDeparture

```
size_t PassengerProperties::floorDeparture
```

Этаж, с которого пассажир отправляется

См. определение в файле Passenger.h строка 33

4.6.2.3 floorDestination

size_t PassengerProperties::floorDestination

Этаж, на который пассажир едет

См. определение в файле Passenger.h строка 36

4.6.2.4 pInverseStartWaiting

double PassengerProperties::pInverseStartWaiting

Вероятность сесть в лифт, едущий в неверном направлении в начале ожидания

См. определение в файле Passenger.h строка 42

4.6.2.5 pInverseStopWaiting

double PassengerProperties::pInverseStopWaiting

Вероятность сесть в лифт, едущий в неверном направлении в конце ожидания

См. определение в файле Passenger.h строка 45

4.6.2.6 pStartGoing

double PassengerProperties::pStartGoing

Вероятность нажать кнопку "ход", не дожидаясь закрытия дверей

См. определение в файле Passenger.h строка 48

4.6.2.7 timeInit

size_t PassengerProperties::timeInit

Время появления пассажира на этаже

См. определение в файле Passenger.h строка 30

Объявления и описания членов структуры находятся в файле:

- [Passenger.h](#)

4.7 Класс Queue

Класс — очередь пассажиров

```
#include <Queue.h>
```

Открытые члены

- [Queue](#) (size_t numberOfFloors_)
Инициализирующий конструктор
- void [addPassenger](#) (const [PassengerProperties](#) &passProp_)
Функция добавки пассажира в очередь

Друзья

- class [Control](#)

4.7.1 Подробное описание

Класс — очередь пассажиров

См. определение в файле Queue.h строка 21

4.7.2 Конструктор(ы)

4.7.2.1 Queue()

```
Queue::Queue (
    size_t numberOfFloors_ ) [inline]
```

Инициализирующий конструктор

Аргументы

in	numberOfFloors_	число этажей, включая подвал
----	-----------------	------------------------------

См. определение в файле Queue.h строка 34

```
35 : passOnFloor(numberOfFloors_)
36 {};
```

4.7.3 Методы

4.7.3.1 addPassenger()

```
void Queue::addPassenger (
    const PassengerProperties & passProp_ ) [inline]
```

Функция добавки пассажира в очередь

Аргументы

in	passProp_	константная ссылка на параметры добавляемого пассажира
----	---------------------------	--

См. определение в файле Queue.h строка 41

```
42 {
43     passengers.emplace_back(passengers.size(), passProp_);
44 }
```

4.7.4 Документация по друзьям класса и функциям, относящимся к классу

4.7.4.1 Control

```
friend class Control [friend]
```

См. определение в файле Queue.h строка 23

Объявления и описания членов класса находятся в файле:

- [Queue.h](#)

Глава 5

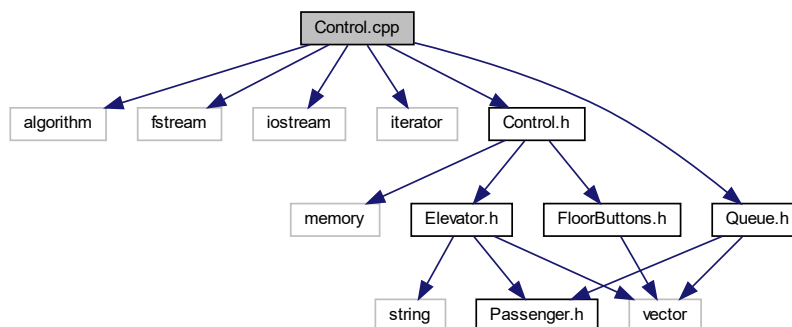
Файлы

5.1 Файл Control.cpp

Файл кода с описанием класса [Control](#).

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <iterator>
#include "Control.h"
#include "Queue.h"
```

Граф включаемых заголовочных файлов для Control.cpp:



5.1.1 Подробное описание

Файл кода с описанием класса [Control](#).

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

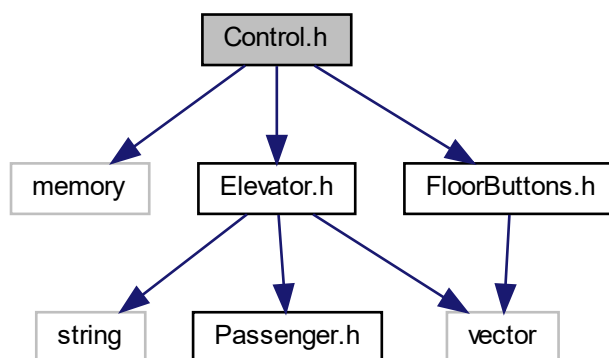
25 мая 2021 г.

5.2 Файл Control.h

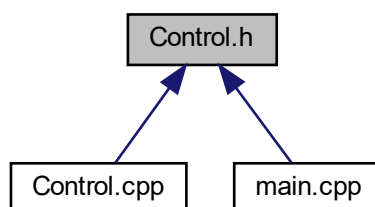
Заголовочный файл с описанием основного класса [Control](#).

```
#include <memory>
#include "Elevator.h"
#include "FloorButtons.h"
```

Граф включаемых заголовочных файлов для Control.h:



Граф файлов, в которые включается этот файл:



Классы

- class [Control](#)

Основной класс — симулятор пассажирского лифта

5.2.1 Подробное описание

Заголовочный файл с описанием основного класса [Control](#).

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

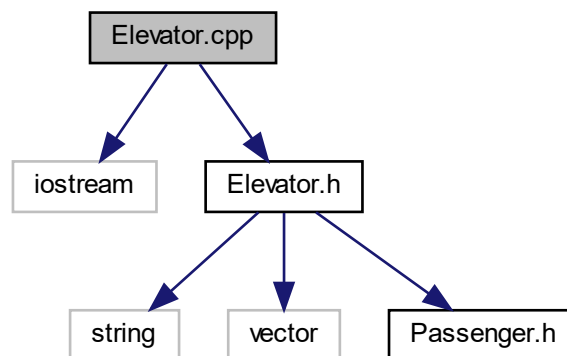
25 мая 2021 г.

5.3 Файл Elevator.cpp

Файл кода с описанием класса [Elevator](#).

```
#include <iostream>
#include "Elevator.h"
```

Граф включаемых заголовочных файлов для Elevator.cpp:



5.3.1 Подробное описание

Файл кода с описанием класса [Elevator](#).

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

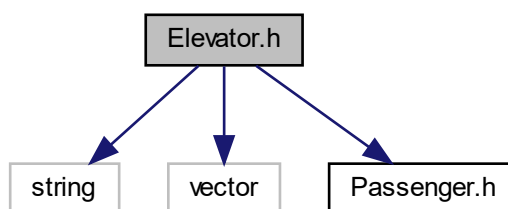
25 мая 2021 г.

5.4 Файл Elevator.h

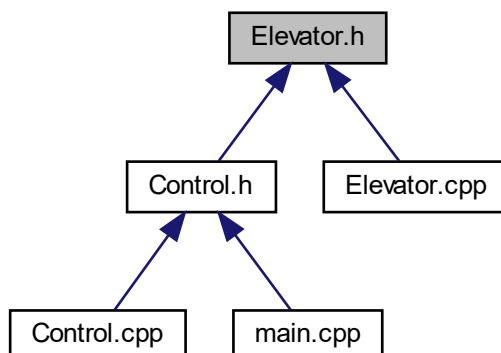
Заголовочный файл с описанием класса [Elevator](#) и сопутствующих структур

```
#include <string>
#include <vector>
#include "Passenger.h"
```

Граф включаемых заголовочных файлов для Elevator.h:



Граф файлов, в которые включается этот файл:



Классы

- class [Elevator](#)
Класс — кабина лифта

Перечисления

- enum class [ElevatorIndicator](#) { [both](#) , [up](#) , [down](#) }
Состояния лампочки (индикатора направления движения) кабины лифта
- enum class [ElevatorStatus](#) { [staying](#) , [movingUp](#) , [movingDn](#) }
- enum class [ElevatorAcceleration](#) { [breaking](#) , [accelerating](#) , [uniform](#) }
Ускорение лифта (не может быть использовано напрямую в системе управления, внутренний параметр)
- enum class [ElevatorDoorsStatus](#) {
[opening](#) , [openedUnloading](#) , [openedLoading](#) , [closing](#) ,
[closed](#) , [waiting](#) }
Состояние дверей кабины лифта (не может быть использовано напрямую в системе управления, внутренний параметр)

5.4.1 Подробное описание

Заголовочный файл с описанием класса [Elevator](#) и сопутствующих структур

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

25 мая 2021 г.

5.4.2 Перечисления

5.4.2.1 ElevatorAcceleration

enum `ElevatorAcceleration` [strong]

Ускорение лифта (не может быть использовано напрямую в системе управления, внутренний параметр)

Элементы перечислений

breaking	замедляется (тормозит)
accelerating	ускоряется (разгоняется)
uniform	движется равномерно или стоит

См. определение в файле `Elevator.h` строка 56

```
56 {
57     breaking,
58     accelerating,
59     uniform,
60 };
```

5.4.2.2 ElevatorDoorsStatus

enum `ElevatorDoorsStatus` [strong]

Состояние дверей кабины лифта (не может быть использовано напрямую в системе управления, внутренний параметр)

Элементы перечислений

opening	открываются
openedUnloading	открыты (идет высадка пассажиров)
openedLoading	открыты (идет посадка пассажиров)
closing	закрываются
closed	закрыты
waiting	открыты (лифт ожидает отправления)

См. определение в файле `Elevator.h` строка 66

```
66 {
67     opening,
68     openedUnloading,
69     openedLoading,
70     closing,
71     closed,
72     waiting,
73 };
```

5.4.2.3 ElevatorIndicator

enum `ElevatorIndicator` [strong]

Состояния лампочки (индикатора направления движения) кабины лифта

Состояние лифта (не может быть использовано напрямую в системе управления, внутренний параметр)

Индикатор (лампочка со стрелочкой) "зажигается" Вами при помощи системы управления. Пассажиры в кабину садятся только те, которые едут в указанном стрелочкой (или двумя стрелочками, если зажечь состояние "both") направлении.

Хотя бывают пассажиры, которые не смотрят на стрелочки и садятся в лифт, едущий не туда (в симуляции тоже такие будут встречаться, причем вероятность их появления повышается с увеличением времени ожидания на этаже!)

Если индикатор лифта "горит" в состоянии both (он пустой или нет - не важно), и в лифт входит пассажир, то индикатор автоматически переключается в то направление, какую кнопку он нажал, входя в лифт.

Если ваша система управления работает "криво", то возможна ситуация, что стрелочка будет гореть вверх, а реально лифт поедет вниз. Едущие вниз при этом в него не садутся.

Элементы перечислений

both	стрелочки в обоих направлениях
up	стрелочка вверх
down	стрелочка вниз

См. определение в файле Elevator.h строка 36

```
36     {  
37         both,  
38         up,  
39         down,  
40     };
```

5.4.2.4 ElevatorStatus

enum `ElevatorStatus` [strong]

Элементы перечислений

staying	стоит (не движется)
movingUp	движется вверх
movingDn	движется вниз

См. определение в файле Elevator.h строка 46

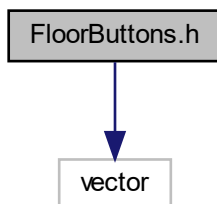
```
46     {  
47         staying,  
48         movingUp,  
49         movingDn,  
50     };
```

5.5 Файл FloorButtons.h

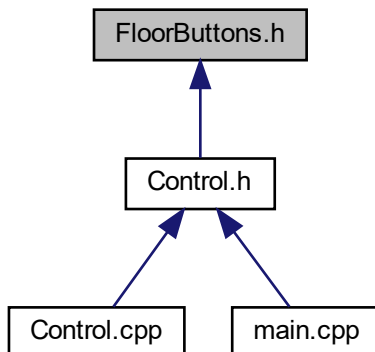
Заголовочный файл с описанием класса [FloorButtons](#).

```
#include <vector>
```

Граф включаемых заголовочных файлов для FloorButtons.h:



Граф файлов, в которые включается этот файл:



Классы

- class [FloorButtons](#)

Класс — кнопки на этажах

5.5.1 Подробное описание

Заголовочный файл с описанием класса [FloorButtons](#).

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

25 мая 2021 г.

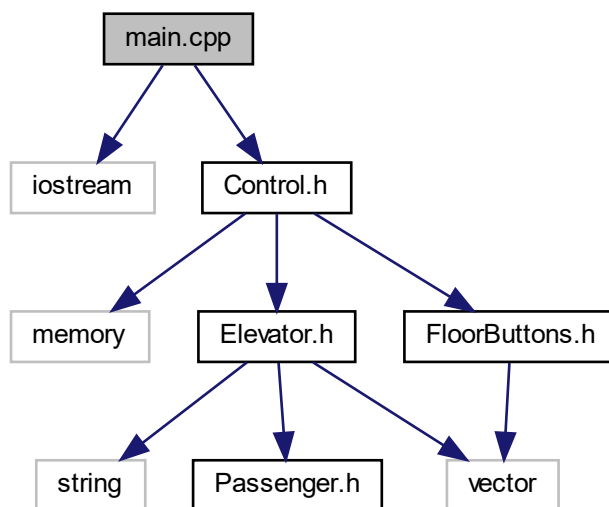
5.6 Файл main.cpp

Основной файл программы elevator.

```
#include <iostream>
```

```
#include "Control.h"
```

Граф включаемых заголовочных файлов для main.cpp:



Классы

- struct [myParams](#)

Структура, содержащая пользовательские параметры

Функции

- void [CONTROLSYSTEM](#) ([Control](#) &control, [myParams](#) ¶ms)
Основная функция системы управления
- int [main](#) (int argc, char **argv)

Переменные

- `const size_t numberOfElevators = 2`
Число лифтов
- `const size_t elevatorCapacity = 4`
- `const size_t maxFloor = 11`
- `const size_t numberOfFloors = maxFloor + 1`
Общее число этажей
- `const size_t maxTime = 6000`

5.6.1 Подробное описание

Основной файл программы `elevator`.

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

25 мая 2021 г.

5.6.2 Функции

5.6.2.1 CONTROLSYSTEM()

```
void CONTROLSYSTEM (  
    Control & control,  
    myParams & params )
```

Основная функция системы управления

Именно эту функцию нужно модифицировав, реализовав здесь оптимизированный алгоритм работы пассажирского лифта.

Другие функции "трогать" запрещается.

Данная функция вызывается на каждом шаге (каждую секунду) работы лифта.

Внутри можно пользоваться структурой `params`, сохраняя в нее при необходимости нужные сведения, которые, соответственно, будут доступны при следующем вызове функции `CONTROLSYSTEM`

Собственно, для активного управления лифтами есть всего две команды:

- `control.setElevatorDestination(elevatorNumber, newDestination);`
- `control.setElevatorIndicator(elevatorNumber, newIndicator)`

Прежде, чем отдавать команду лифту по итогам анализа его текущего состояния - надо решить, имеет ли смысл делать это прямо сейчас

К примеру, если лифт едет куда-то на вызов (скажем, на 10-й этаж), и в этот момент кто-то в подвале (0-й этаж) нажал на кнопку вызова, то если поступить формально и в этот момент изменить назначение лифта путем исполнения команды

- `control.setElevatorDestination(elev, newDestination),`

то он затормозит по пути (возможно, даже между этажами), и потом начнет разгоняться и поедет в обратном направлении (вниз); если лифт прибыл на этаж, начал тормозить, и в этот момент ему установить новое назначение, то он затормозит, двери не откроет, и тут же поедет по новому назначению!

Будьте аккуратны. Наш лифт очень "исполнительный"!

Поэтому нужно тщательно обдумывать, в какой момент отдавать команду на изменение направления и в какой команд отдавать команду на изменение индикатора

- `control.setElevatorIndicator(elev, newIndicator)`

(пассажиры садятся только в тот лифт, который едет в нужную им сторону, судя по индикатору, который они видят, хотя иногда попадают и те, кто садятся не туда, при этом зайдя в лифт, они "жмут" кнопку, куда надо им — это тоже надо как-то обрабатывать!!!)

Аргументы

in,out	control	ссылка на основной класс-симулятор лифта
in,out	params	ссылка на набор пользовательских параметров

См. определение в файле main.cpp строка 192

```

193 {
194     // Прежде, чем отдавать команду лифту по итогам анализа его текущего состояния - надо решить,
195     // имеет ли смысл делать это прямо сейчас
196     //
197     // К примеру, если лифт едет куда-то на вызов (скажем, на 10-й этаж), и в этот момент кто-то
198     // в подвале (0-й этаж) нажал на кнопку вызова,
199     // то если поступить формально и в этот момент изменить назначение лифта путем исполнения команды
200     //
201     // control.setElevatorDestination(elev, newDestination),
202     //
203     // то он затормозит по пути (возможно, даже между этажами), и потом начнет разгоняться
204     // и поедет в обратном направлении (вниз);
205     // если лифт прибыл на этаж, начал тормозить, и в этот момент ему установить новое назначение,
206     // то он затормозит, двери не откроет, и тут же поедет по новому назначению!
207     // Будьте аккуратны. Наш лифт очень "исполнительный"!
208     //
209     // Поэтому нужно тщательно обдумывать, в какой момент отдавать команду на изменение направления
210     // и в какой команд отдавать команду на изменение индикатора
211     //
212     // control.setElevatorIndicator(elev, newIndicator)
213     //
214     // (пассажиры садятся только в тот лифт, который едет в нужную им сторону,
215     // судя по индикатору, который они видят, хотя иногда попадают и те, кто садятся не туда,
216     // при этом зайдя в лифт, они "жмут" кнопку, куда надо им --- это тоже надо как-то обрабатывать!!!)
217     //
218     // ПРИМЕЧАНИЕ: собственно, для активного управления лифтами есть всего две команды:

```

```

219 // control.setElevatorDestination(elev, newDestination);
220 // control.setElevatorIndicator(elev, newIndicator)
221 //
222
223
224 // Для получения текущего времени можно пользоваться командой
225 // control.getCurrentTime()
226
227
228 // Следующие команды носят характер опроса текущего состояния лифта
229 //
230 // - текущее назначение лифта:
231 // control.getElevatorDestination(elev);
232 //
233 // - текущее значение индикатора:
234 // control.getElevatorIndicator(elev);
235 //
236 // - текущее положение лифта (дробное число, когда лифт между этажами;
237 //   даже если оно целое - лифт не обязательно с открытыми дверьми, он может начинать разгоняться
238 //   или тормозить и т.п.)
239 // control.getElevatorPosition(elev);
240 //
241 // - признак того, что лифт движется вверх
242 // control.isElevatorGoingUp(elev)
243 //
244 // - признак того, что лифт движется вниз
245 // control.isElevatorGoingDn(elev)
246 //
247 // - признак того, что лифт стоит на месте
248 // control.isElevatorStaying(elev)
249 //
250 // - признак того, что лифт движется равномерно
251 // control.isElevatorGoingUniformly(elev);
252 //
253 // - признак того, что лифт движется с ускорением (разгоняется)
254 // control.isElevatorAccelerating(elev);
255 //
256 // - признак того, что лифт движется с замедлением (тормозит)
257 // control.isElevatorBreaking(elev);
258 //
259 // - признак того, что лифт стоит на месте (на этаже) с закрытыми дверьми
260 // control.isElevatorStayingDoorsClosed(elev);
261 //
262 // - признак того, что в текущий момент происходит закрывание дверей
263 // control.isElevatorDoorsClosing(elev);
264 //
265 // - признак того, что в текущий момент происходит открывание дверей
266 // control.isElevatorDoorsOpening(elev);
267 //
268 // - признак того, что в текущий момент двери открыты
269 // control.isElevatorDoorsOpened(elev);
270 //
271 // - признак того, что лифт пустой (в нем нет ни одного пассажира)
272 // control.isElevatorEmpty(elev);
273 //
274 // - признак того, что в данный момент завершилась высадка пассажиров, и лифт оказался пустым (см. ниже)
275 // control.isElevatorEmptyAfterUnloading(elev);
276 //
277 // - признак того, что лифт достиг точки назначения
278 // точка назначения считается достигнутой, когда
279 //   1) лифт приехал на тот этаж, куда его послали, остановился, и
280 //   2) выполнено одно из трех условий:
281 //   а) в нем есть хотя бы 1 пассажир - тогда открылись двери
282 //   б) он пустой, а на этаже, на который он прибыл, нажата хотя бы одна
283 //   кнопка - тогда тоже открылись двери
284 //   в) он пустой, а на этаже, на который он прибыл, не нажато ни одной
285 //   кнопки - тогда двери не открываются
286 // control.isElevatorAchievedDestination(elev)
287 //
288 //
289 // Может быть полезной команда
290 //
291 // control.isElevatorEmptyAfterUnloading(elev)
292 //
293 // которая возвращает true, если лифт стоит на этаже, и после выхода очередного пассажира
294 // лифт оказался пустым --- возможно, при этом имеет смысл "включить" индикатор в оба направления,
295 // чтобы в любом случае зашел пассажир, стоящий первым в очереди.
296 // Но это не обязательно - у Вас может быть своя логика!
297 //
298 // Если индикатор лифта "горит" в состоянии both (он пустой или нет - не важно),
299 // и в лифт входит пассажир, то индикатор автоматически переключается в то направление,
300 // какую кнопку он нажал, входя в лифт.
301 // Будьте осторожны, "зажигайте" состояние индикатора both аккуратно, но и без него обойтись будет трудно!
302 //
303
304 // Следующие 4 команды позволяют узнать состояние "нажатости" кнопок на этажах
305 // const std::vector<bool>& getFloorUpButtons() const

```

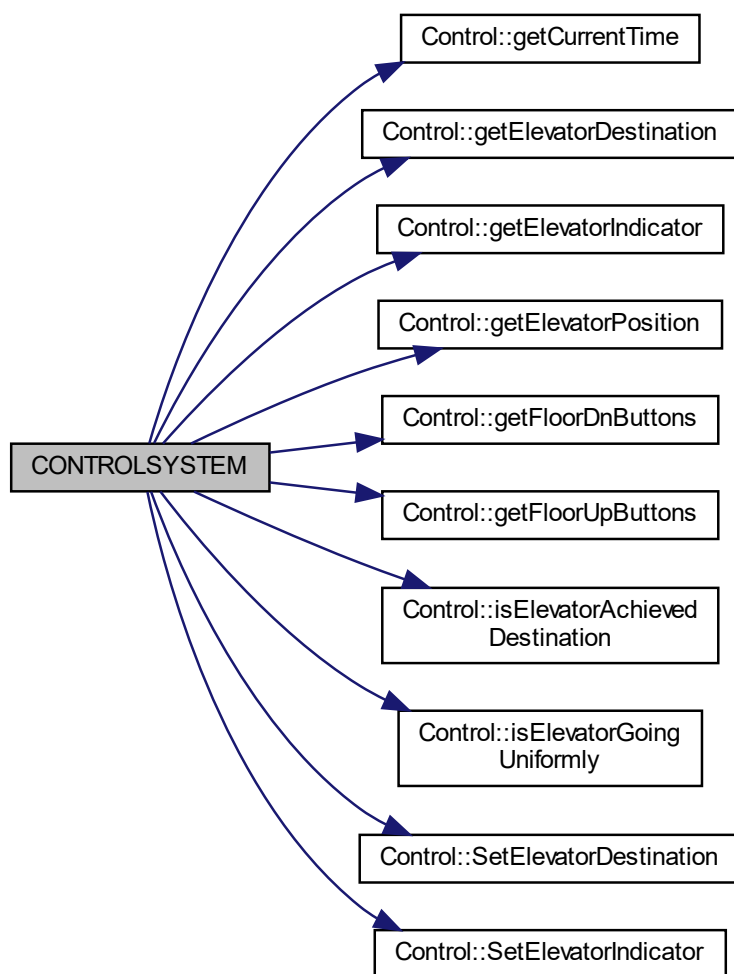
```

306 //
307 // При этом когда лифт приезжает на какой-либо этаж, то в момент открывания дверей на этаже
308 // автоматически гаснет та кнопка, какой индикатор в этот момент установлен у лифта
309 // (если индикатор both - гаснут обе кнопки)
310 // Если пассажиры, оставшиеся на этаже, видят, что нужная им кнопка погасла, они
311 // нажмут ее снова, как только лифт тронется
312 //
313 // - возвращает вектор (массив) состояний нажатия кнопок вверх
314 // control.floorButtons->getUpButtons();
315 //
316 // - возвращает вектор (массив) состояний нажатия кнопок вниз
317 // control.floorButtons->getDownButtons();
318 //
319 // - возвращает состояние нажатия кнопки вверх на i-м этаже
320 // control.floorButtons->getUpButton(i);
321 //
322 // - возвращает состояние нажатия кнопки вниз на i-м этаже
323 // control.floorButtons->getDownButton(i);
324 //
325 // При необходимости можно использовать команды принудительного выключения кнопок на соответствующих
этажах:
326 // control.unsetUpButton(floor);
327 // control.unsetDownButton(floor);
328 //
329 //
330 // Наконец, еще 2 команды позволяют оценить состояние кнопок в кабине лифта
331 // Человек, входящий в лифт, нажимает кнопку этажа назначения
332 // Кнопка, нажатая внутри лифта, гаснет, когда лифт прибывает на этаж и начинает открывать двери
333 //
334 // - возвращает вектор состояния нажатости кнопок в кабине лифта
335 // control.getElevatorButtons(elev)
336 //
337 // - возвращает вектор состояния нажатости кнопки i-го этажа в кабине лифта
338 // control.getElevatorButton(elev, i)
339 //
340 //
342 // ПРИМЕР примитивной системы управления, при которой первоначально лифт #0 стоит
343 // в подвале, а лифту #1 отдается команда уехать на самый верхний этаж.
344 // Потом они оба ждут до момента появления первого пассажира на каком-либо этаже,
345 // после чего начинают кататься вверх-вниз, останавливаясь на каждом этаже
346 // т.е. вообще не реагируя на кнопки!
347 //
348 //
349 if (control.getCurrentTime() == 1)
350 {
351     control.SetElevatorDestination(1, maxFloor);
352     control.SetElevatorIndicator(1, ElevatorIndicator::up);
353 }
354 //
355 if (!params.started)
356 {
357     size_t nUp = std::count(control.getFloorUpButtons().begin(), control.getFloorUpButtons().end(), true);
358     size_t nDn = std::count(control.getFloorDnButtons().begin(), control.getFloorDnButtons().end(), true);
359 //
360 // Если хоть одна кнопка вверх или вниз на этажах нажата - запускаем лифт!
361 if (nUp + nDn > 0)
362 {
363     params.started = true;
364 }
365 }
366 //
367 for (size_t elv = 0; elv < 2; ++elv)
368 {
369 // В данном примере новая команда (назначение) не отдается,
370 // пока не выполнена предыдущая
371 if ((params.started) && (control.isElevatorAchievedDestination(elv)))
372 {
373     // считываем этаж, на который лифт прибыл
374     size_t curDest = control.getElevatorDestination(elv);
375 //
376 // прибывая на этаж назначения лифт открывает двери, если либо он не пустой,
377 // либо на этом этаже нажата кнопка вызова хотя бы в какую-то сторону,
378 // в противном случае прибывает на этаж и стоит, не открывая двери
379 //
380 // считываем текущее положение лифта
381 size_t nextDest = (size_t)(control.getElevatorPosition(elv));
382 //
383 switch (control.getElevatorIndicator(elv))
384 {
385 case ElevatorIndicator::both:
386 case ElevatorIndicator::up:
387     ++nextDest;
388     break;
389 case ElevatorIndicator::down:
390     --nextDest;
391     break;
392 }
393 }

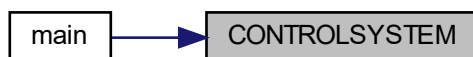
```

```
394
395     control.SetElevatorDestination(elv, nextDest);
396 }
397
398 //Теперь устанавливаем индикатор
399 if (control.isElevatorGoingUniformly(elv))
400 {
401     // считываем текущий индикатор движения (лифт изначально инициализирован в both)
402     ElevatorIndicator curInd = control.getElevatorIndicator(elv);
403
404     // индикатор, который будет установлен дальше, инициализируем его в текущим индикатором
405     ElevatorIndicator nextInd = curInd;
406
407     // поменяем его, если он установлен в both
408     if (curInd == ElevatorIndicator::both)
409         nextInd = ElevatorIndicator::up;
410
411     // при прибытии на максимальный этаж - переключаем индикатор "вниз"
412     if ((control.getElevatorDestination(elv) == maxFloor) && (control.getElevatorPosition(elv) > maxFloor - 1))
413         nextInd = ElevatorIndicator::down;
414
415     // при прибытии на минимальный этаж (в подвал) - переключаем индикатор "вверх"
416     if ((control.getElevatorDestination(elv) == 0) && (control.getElevatorPosition(elv) < 1))
417         nextInd = ElevatorIndicator::up;
418
419     // собственно, установка значения индикатора
420     control.SetElevatorIndicator(elv, nextInd);
421 } //if (control.isElevatorGoingUniformly(elv))
422 }
423 }
```

Граф вызовов:



Граф вызова функции:



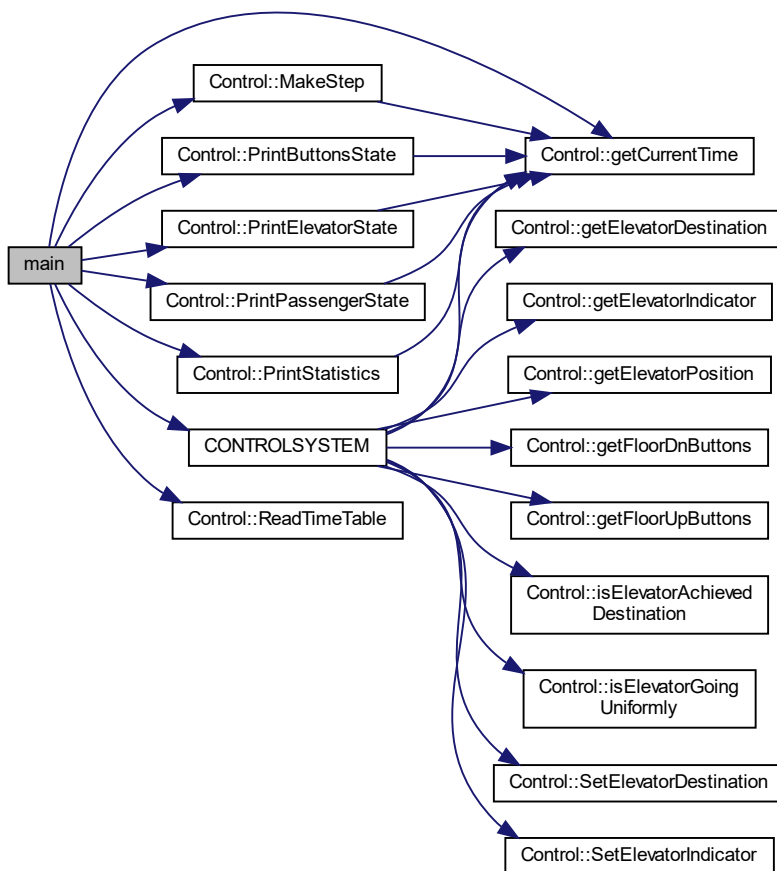
5.6.2.2 main()

```
int main (
    int argc,
    char ** argv )
```

См. определение в файле main.cpp строка 118

```
119 {
120     //Задание конфигурации лифтового хозяйства
121     Control control(numberOfFloors, numberOfElevators, elevatorCapacity);
122
123     //Для загрузки расписания появления пассажиров из файла
124     control.ReadTimeTable("TimeTable/timetable125.csv");
125
126     //Для тестирования вводим появляющихся пассажиров вручную
127     //позже это будет сделано путем чтения файла
128     //параметры в фиг. скобках
129     //{
130     // 1) время появления пассажира (от начала моделирования)
131     // 2) этаж, где появляется пассажир
132     // 3) этаж, куда направляется пассажир
133     // 4) время, которое пассажир ждет и после которого, не выдерживая, уходит (начисляется штраф)
134     // 5) вероятность сесть в лифт, идущий в обратном направлении, в начале ожидания
135     // 6) вероятность сесть в лифт, идущий в обратном направлении, в конце ожидания
136     // 7) вероятность того, что пассажир, войдя в лифт, нажмет "ход" и лифт не будет стоять
137     // }
138     /*
139     control.AddPassengerToQueue({ 5, 5, 3, 300, 0.01, 0.20, 0.50 });
140     control.AddPassengerToQueue({ 6, 5, 10, 300, 0.01, 0.20, 0.50 });
141     control.AddPassengerToQueue({ 7, 5, 2, 300, 0.01, 0.20, 0.50 });
142     control.AddPassengerToQueue({ 8, 5, 8, 300, 0.01, 0.20, 0.50 });
143     control.AddPassengerToQueue({ 9, 5, 10, 300, 0.01, 0.20, 0.50 });
144     control.AddPassengerToQueue({ 10, 5, 6, 300, 0.01, 0.20, 0.50 });
145     control.AddPassengerToQueue({ 11, 5, 9, 300, 0.01, 0.20, 0.50 });
146     control.AddPassengerToQueue({ 12, 5, 8, 300, 0.01, 0.20, 0.50 });
147     control.AddPassengerToQueue({ 13, 5, 11, 300, 0.01, 0.20, 0.50 });
148     control.AddPassengerToQueue({ 14, 5, 10, 300, 0.01, 0.20, 0.50 });
149     */
150
151     myParams params;
152
153     do
154     {
155         //Выполнение одного шага (= 1 секунда) моделирования работы лифта
156         control.MakeStep();
157
158         //Вызов функции системы управления --- в ней можно "отдать команду" лифту,
159         //исходя из его текущего состояния и состояния кнопок в лифте и на этажах
160         CONTROLSYSTEM(control, params);
161
162         //Вывод состояния лифта
163         //control.PrintElevatorState(0); //Вывод состояния лифта #0 на экран
164         //control.PrintElevatorState(1); //Вывод состояния лифта #1 на экран
165
166         control.PrintElevatorState(0, "fileElev0.txt"); //Вывод состояния лифта #0 в файл
167         control.PrintElevatorState(1, "fileElev1.txt"); //Вывод состояния лифта #1 в файл
168
169         //Вывод состояния кнопок в лифте и на этажах
170         //control.PrintButtonsState(); //Вывод состояния кнопок на экран
171         control.PrintButtonsState("fileButtons.txt"); //Вывод состояния кнопок в файл
172
173         //Вывод событий появлений пассажиров, их входа в лифт, выхода из лифта, ухода с этажа
174         //control.PrintPassengerState(); //Вывод статистики пассажиров на экран
175         control.PrintPassengerState("filePassengers.txt"); //Вывод статистики пассажиров в файл
176
177     } while (control.getCurrentTime() <= maxTime);
178
179     //Печать итоговой статистики в конце работы симулятора
180     control.PrintStatistics(true, "Statistics.txt");
181
182     return 0;
183 }
```

Граф вызовов:



5.6.3 Переменные

5.6.3.1 elevatorCapacity

```
const size_t elevatorCapacity = 4
```

Вместимость лифта

Предупреждения

Тренироваться проще с меньшей вместимостью, в реальной задаче будет не менее 6 человек

См. определение в файле `main.cpp` строка 58

5.6.3.2 maxFloor

```
const size_t maxFloor = 11
```

Максимальный номер этажа (не считая подвала, который имеет номер 0). Пассажиры иногда ездят в подвал и из подвала

См. определение в файле main.cpp строка 62

5.6.3.3 maxTime

```
const size_t maxTime = 6000
```

Время моделирования в секундах

Предупреждения

Сейчас для тестирования задано 6000 секунд, в реальной задаче буде 54000 секунд: от 7:00 утра до 22:00 вечера

См. определение в файле main.cpp строка 69

5.6.3.4 numberOfElevators

```
const size_t numberOfElevators = 2
```

Число лифтов

См. определение в файле main.cpp строка 54

5.6.3.5 numberOfFloors

```
const size_t numberOfFloors = maxFloor + 1
```

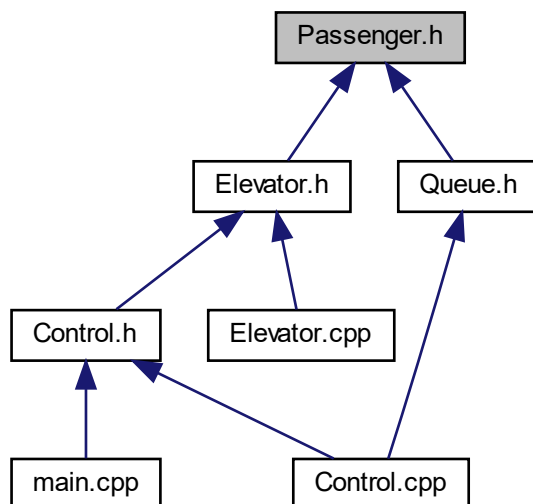
Общее число этажей

См. определение в файле main.cpp строка 65

5.7 Файл Passenger.h

Заголовочный файл с описанием класса [Passenger](#) и сопутствующих структур

Граф файлов, в которые включается этот файл:



Классы

- struct [PassengerProperties](#)
Параметры пассажиров
- class [Passenger](#)
Класс — пассажир

Перечисления

- enum class [PassengerStatus](#) { [waiting](#) , [going](#) , [arrived](#) , [leaved](#) }
Статусы пассажиров (не может быть использовано напрямую в системе управления, внутренний параметр)

5.7.1 Подробное описание

Заголовочный файл с описанием класса [Passenger](#) и сопутствующих структур

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

25 мая 2021 г.

5.7.2 Перечисления

5.7.2.1 PassengerStatus

enum `PassengerStatus` [strong]

Статусы пассажиров (не может быть использовано напрямую в системе управления, внутренний параметр)

Элементы перечислений

waiting	Пассажир ждет лифта на этаже отправления
going	Пассажир находится в лифте
arrived	Пассажир успешно прибыл на нужный этаж
leaved	Пассажир не дождался лифта и ушел

См. определение в файле `Passenger.h` строка 17

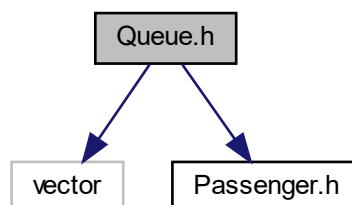
```
17 {  
18     waiting,  
19     going,  
20     arrived,  
21     leaved  
22 };
```

5.8 Файл Queue.h

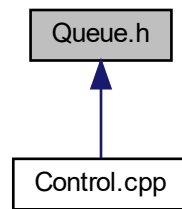
Заголовочный файл с описанием класса `Queue`.

```
#include <vector>  
#include "Passenger.h"
```

Граф включаемых заголовочных файлов для `Queue.h`:



Граф файлов, в которые включается этот файл:



Классы

- class [Queue](#)

Класс — очередь пассажиров

5.8.1 Подробное описание

Заголовочный файл с описанием класса [Queue](#).

Автор

Марчевский Илья Константинович

Версия

0.5

Дата

25 мая 2021 г.

Предметный указатель

- ~Control
 - Control, [10](#)
- accelerating
 - Elevator.h, [54](#)
- addPassenger
 - Queue, [47](#)
- AddPassengerToQueue
 - Control, [10](#)
- arbitraryParam
 - myParams, [42](#)
- arrived
 - Passenger.h, [68](#)
- both
 - Elevator.h, [55](#)
- breaking
 - Elevator.h, [54](#)
- closed
 - Elevator.h, [54](#)
- closing
 - Elevator.h, [54](#)
- Control, [7](#)
 - ~Control, [10](#)
 - AddPassengerToQueue, [10](#)
 - Control, [9](#)
 - Elevator, [41](#)
 - FloorButtons, [42](#)
 - getCurrentTime, [10](#)
 - getElevatorButton, [11](#)
 - getElevatorButtons, [11](#)
 - getElevatorDestination, [12](#)
 - getElevatorIndicator, [12](#)
 - getElevatorPosition, [13](#)
 - getFloorDnButton, [14](#)
 - getFloorDnButtons, [14](#)
 - getFloorUpButton, [15](#)
 - getFloorUpButtons, [15](#)
 - isElevatorAccelerating, [16](#)
 - isElevatorAchievedDestination, [16](#)
 - isElevatorBreaking, [17](#)
 - isElevatorDoorsClosing, [17](#)
 - isElevatorDoorsOpened, [18](#)
 - isElevatorDoorsOpening, [18](#)
 - isElevatorEmpty, [20](#)
 - isElevatorEmptyAfterUnloading, [20](#)
 - isElevatorGoingDn, [21](#)
 - isElevatorGoingUniformly, [21](#)
 - isElevatorGoingUp, [22](#)
 - isElevatorStaying, [22](#)
 - isElevatorStayingDoorsClosed, [23](#)
 - MakeStep, [23](#)
 - Passenger, [44](#)
 - PrintButtonsState, [29](#)
 - PrintElevatorState, [30](#)
 - PrintPassengerState, [31](#)
 - PrintStatistics, [32](#)
 - Queue, [48](#)
 - ReadTimeTable, [35](#)
 - SetElevatorDestination, [36](#)
 - SetElevatorIndicator, [36](#)
 - timeAccelerating, [38](#)
 - timeBreaking, [38](#)
 - timeClosing, [38](#)
 - timeEntering, [38](#)
 - timeLeaving, [39](#)
 - timeOpening, [39](#)
 - unsetDnButton, [37](#)
 - unsetUpButton, [37](#)
 - veloUniform, [39](#)
 - waitingTime, [39](#)
- Control.cpp, [49](#)
- Control.h, [50](#)
- CONTROLSYSTEM
 - main.cpp, [58](#)
- criticalWaitTime
 - PassengerProperties, [45](#)
- down
 - Elevator.h, [55](#)
- Elevator, [40](#)
 - Control, [41](#)
 - Elevator, [40](#)
 - Passenger, [44](#)
- Elevator.cpp, [51](#)
- Elevator.h, [52](#)
 - accelerating, [54](#)
 - both, [55](#)
 - breaking, [54](#)
 - closed, [54](#)
 - closing, [54](#)
 - down, [55](#)
 - ElevatorAcceleration, [54](#)
 - ElevatorDoorsStatus, [54](#)
 - ElevatorIndicator, [54](#)
 - ElevatorStatus, [55](#)
 - movingDn, [55](#)

- movingUp, 55
- openedLoading, 54
- openedUnloading, 54
- opening, 54
- staying, 55
- uniform, 54
- up, 55
- waiting, 54
- ElevatorAcceleration
 - Elevator.h, 54
- elevatorCapacity
 - main.cpp, 65
- ElevatorDoorsStatus
 - Elevator.h, 54
- ElevatorIndicator
 - Elevator.h, 54
- ElevatorStatus
 - Elevator.h, 55
- FloorButtons, 41
 - Control, 42
 - FloorButtons, 41
- FloorButtons.h, 56
- floorDeparture
 - PassengerProperties, 45
- floorDestination
 - PassengerProperties, 45
- getCurrentTime
 - Control, 10
- getElevatorButton
 - Control, 11
- getElevatorButtons
 - Control, 11
- getElevatorDestination
 - Control, 12
- getElevatorIndicator
 - Control, 12
- getElevatorPosition
 - Control, 13
- getFloorDnButton
 - Control, 14
- getFloorDnButtons
 - Control, 14
- getFloorUpButton
 - Control, 15
- getFloorUpButtons
 - Control, 15
- going
 - Passenger.h, 68
- isElevatorAccelerating
 - Control, 16
- isElevatorAchievedDestination
 - Control, 16
- isElevatorBreaking
 - Control, 17
- isElevatorDoorsClosing
 - Control, 17
- isElevatorDoorsOpened
 - Control, 18
- isElevatorDoorsOpening
 - Control, 18
- isElevatorEmpty
 - Control, 20
- isElevatorEmptyAfterUnloading
 - Control, 20
- isElevatorGoingDn
 - Control, 21
- isElevatorGoingUniformly
 - Control, 21
- isElevatorGoingUp
 - Control, 22
- isElevatorStaying
 - Control, 22
- isElevatorStayingDoorsClosed
 - Control, 23
- leaved
 - Passenger.h, 68
- main
 - main.cpp, 63
- main.cpp, 57
 - CONTROLSYSTEM, 58
 - elevatorCapacity, 65
 - main, 63
 - maxFloor, 65
 - maxTime, 66
 - numberOfElevators, 66
 - numberOfFloors, 66
- MakeStep
 - Control, 23
- maxFloor
 - main.cpp, 65
- maxTime
 - main.cpp, 66
- movingDn
 - Elevator.h, 55
- movingUp
 - Elevator.h, 55
- myParams, 42
 - arbitraryParam, 42
 - started, 43
- numberOfElevators
 - main.cpp, 66
- numberOfFloors
 - main.cpp, 66
- openedLoading
 - Elevator.h, 54
- openedUnloading
 - Elevator.h, 54
- opening
 - Elevator.h, 54
- operator<
 - Passenger, 44

- Passenger, 43
 - Control, 44
 - Elevator, 44
 - operator<, 44
 - Passenger, 43
- Passenger.h, 67
 - arrived, 68
 - going, 68
 - leaved, 68
 - PassengerStatus, 68
 - waiting, 68
- PassengerProperties, 45
 - criticalWaitTime, 45
 - floorDeparture, 45
 - floorDestination, 45
 - pInverseStartWaiting, 46
 - pInverseStopWaiting, 46
 - pStartGoing, 46
 - timeInit, 46
- PassengerStatus
 - Passenger.h, 68
- pInverseStartWaiting
 - PassengerProperties, 46
- pInverseStopWaiting
 - PassengerProperties, 46
- PrintButtonsState
 - Control, 29
- PrintElevatorState
 - Control, 30
- PrintPassengerState
 - Control, 31
- PrintStatistics
 - Control, 32
- pStartGoing
 - PassengerProperties, 46
- Queue, 47
 - addPassenger, 47
 - Control, 48
 - Queue, 47
- Queue.h, 68
- ReadTimeTable
 - Control, 35
- SetElevatorDestination
 - Control, 36
- SetElevatorIndicator
 - Control, 36
- started
 - myParams, 43
- staying
 - Elevator.h, 55
- timeAccelerating
 - Control, 38
- timeBreaking
 - Control, 38
- timeClosing
 - Control, 38
- timeEntering
 - Control, 38
- timeInit
 - PassengerProperties, 46
- timeLeaving
 - Control, 39
- timeOpening
 - Control, 39
- uniform
 - Elevator.h, 54
- unsetDnButton
 - Control, 37
- unsetUpButton
 - Control, 37
- up
 - Elevator.h, 55
- veloUniform
 - Control, 39
- waiting
 - Elevator.h, 54
 - Passenger.h, 68
- waitingTime
 - Control, 39