МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»
(наименование дисциплины)

Тема курсовой работы К_15 Моделирование работы лифта
(наименование темы)

Студент группы ИКБО-30-23 Павлов Никита Сергеевич
(учебная группа) *(Фамилия Имя Отчество)* *(подпись студента)*

Руководитель курсовой работы доцент Быков А.Ю.
(Должность, звание, ученая степень) *(подпись руководителя)*

Консультант доцент Лозовский В.В.
(Должность, звание, ученая степень) *(подпись консультанта)*

отлично
Быков А.Ю.
31.05.24

Работа представлена к защите «31» мая 2024 г.

Допущен к защите «31» мая 2024 г.

Москва 2024 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой _____
Подпись

Платонова О.В.
ФИО

«21» февраля 2024г.

ЗАДАНИЕ
На выполнение курсовой работы
по дисциплине **«Объектно-ориентированное программирование»**

Студент Павлов Никита Сергеевич Группа ИКБО-30-23

Тема К_15 Моделирование работы лифта

Исходные данные:

1. Описания исходной иерархии дерева объектов.
2. Описание схемы взаимодействия объектов.
3. Множество команд для управления функционированием моделируемой системы.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Построение версий программ.
2. Построение и работа с деревом иерархии объектов.
3. Взаимодействия объектов посредством интерфейса сигналов и обработчиков.
4. Блок-схемы алгоритмов.
5. Управление функционированием моделируемой системы

Срок представления к защите курсовой работы: до «31» мая 2024 г.

Задание на курсовую работу выдал _____ (Быков А.Ю.)
Подпись *ФИО консультанта*
«21» февраля 2024 г.

Задание на курсовую работу получил _____ (Павлов Н.С.)
Подпись *ФИО исполнителя*
«21» февраля 2024 г.

Москва 2024 г.

ОТЗЫВ

на курсовую работу

по дисциплине «Объектно-ориентированное программирование»

Студент **Павлов Никита Сергеевич** группа ИКБО-30-23
(ФИО студента) *(Группа)*

Характеристика курсовой работы

Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме	✓		
2. Соответствие курсовой работы заданию	✓		
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.	✓		
4. Полнота выполнения всех пунктов задания	✓		
5. Логичность и системность содержания курсовой работы	✓		
6. Отсутствие фактических грубых ошибок	✓		

Замечаний: нет

Рекомендуемая оценка: отлично

доцент Быков А.Ю.
(Подпись руководителя) *(ФИО руководителя)*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

Актуальность языка

Язык программирования C++ является одним из самых актуальных и практически-значимых языков в наше время. Этот язык программирования обладает высокой производительностью, предоставляет разработчику контроль над ресурсами программы и также поддерживает множество мощных библиотек и фреймворков.

Цели курсовой работы

Объектно-ориентированные языки программирования, в частности C++, могут быть использованы для моделирования сложных нелинейных процессов. Целями курсовой работы являются:

- Получение практических навыков построения иерархии объектов и моделирования сложных технических систем
- Ознакомление с механизмом сигналов и обработчиков как одним из способов организации взаимодействия объектов вне системы

Задачи курсовой работы

Для достижения поставленных целей в курсовой работе были сформулированы и решены следующие задачи:

1. Организовать иерархическое построение объектов нескольких классов

2. Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков

3. Смоделировать работу лифта с использованием языка программирования C++

1 ПОСТАНОВКА ЗАДАЧИ

Дана система, состоящая из следующих элементов:

- многоэтажного офисного здания;
- на каждом этаже расположены кнопки вызова лифта;
- пульта управления лифтом;
- кабины лифта заданной вместимости, с множеством кнопок в кабине лифта;
- множества пассажиров;
- экрана отображения информации о функционировании системы.

Этажность офисного здания задана. Лифт обслуживает все этажи. На первом этаже расположена кнопка вызова «Вверх». На этажах со второго до предпоследнего кнопки «Вверх» и «Вниз». На последнем этаже расположена кнопка «Вниз».

Система функционирует по тактам.

Правила функционирования системы:

1. Этажность офисного здания задается параметром n. Этажи нумеруются от 1 до n.
2. Вместимость лифта задается параметром m.
3. Каждый пассажир имеет уникальное ФИО.
4. Первоначально лифт пуст и располагается на первом этаже.
5. Пассажир может подойти к лифту на любом этаже. От пассажира подается команда вызова лифта посредством нажатия на кнопку, согласно предполагаемого направления движения.
6. Если лифт пуст (свободен) начинает движение безостановочно согласно первому вызову пассажира до этажа его местонахождения.
7. При движении в определенном направлении с пассажирами лифт

останавливается на этаже, если есть запрос по ходу движения на определенном этаже или достигнут целевой этаж для пассажира в лифте.

8. При остановке лифта, первоначально его покидают пассажиры, которые хотели добраться до данного этажа. При наличии пассажиров на этаже, которым подходит направление движения и есть место в лифте, заходят в лифт.

9. Пассажир при заходе в лифт нажимает на кнопку с номером требуемого (целевого) этажа.

10. Перед началом очередного такта может быть подана команда. Команда отрабатывает до отработки действий такта.

Надо моделировать работу данной системы.

Команды системы:

1. Команда вызова лифта. Моделируется подход пассажира к лифту на любом этаже. Команда содержит уникальное ФИО пассажира, номер этажа расположения пассажира, номер целевого этажа.
2. Команда отображения состояния лифта.
3. Команда отображения состояния на этаже.
4. Команда отображения состояния пассажира.
5. Пустая команда (строка ничего не содержит). Элементы системы выполняют действия согласно такту.
6. Команда завершения работы системы.

Построить программу-систему, которая использует объекты:

1. Объект «система».
2. Объект для чтения исходных данных и команд. Считывает данные для первоначальной подготовки и настройки системы. Считывает команды. Объект моделирует нажатия на кнопки пассажирами. После чтения очередной порции данных для настройки или данных команды, объект

выдает сигнал с текстом полученных данных. Все данные настройки и данные команд по структуре корректны. Каждая строка команд соответствует одному такту (отрабатывается в начале такта). Если строка пустая, то система отрабатывает один такт (все элементы системы отрабатывают положенные действия или находятся в состоянии ожидания).

3. Объект этаж. Содержит номер этажа, кнопки для вызова лифта, список ожидающих лифт пассажиров. Данному объекту по иерархии подчинены объекты пассажиров ожидающих лифт на этаже.

4. Объект пульта управления лифтом, моделирует работу автомата управления лифтом. Анализирует состояние, управляет отработкой действий, предусмотренных в рамках одного такта. Объект выдает соответствующие сигналы для других элементов системы.

5. Объект, моделирующий кабину лифта заданной вместимости. Содержит множество кнопок для запроса этажа остановки лифта. Данному объекту по иерархии подчинены объекты пассажиров, расположенных в лифте.

6. Объект, моделирующий пассажира. Содержит: ФИО пассажира, номер исходного этажа, номер целевого этажа, нажатую кнопку на этаже, номер такта подхода на этаж.

7. Объект для вывода информации. Текст для вывода объект получает по сигналу от других объектов системы. Каждое присланное сообщение выводиться с новой строки.

Архитектура иерархии объектов:

- объект системы моделирования работы лифта;
    - объект ввода;
    - объект пульта управления лифтом;
        - объект лифт;
            - объект пассажир 1 в лифте;

        - объект пассажир 2 в лифте;
        - . . . . .
        - объект пассажир k в лифте;
    - объект этаж «j». Где j = 1:n;
        - объект пассажир 1 на этаже;
        - объект пассажир 2 на этаже;
        - . . . . .
        - объект пассажир h на этаже;
    - объект вывода.

Сконструировать программу-систему, реализующую следующий алгоритм:

1. Вызов от объекта «система» метода build_tree_objects ( ), построения иерархии объектов.

    1.1. Построение исходного дерева иерархии объектов. После построения все объекты перевести в состояние готовности.

    1.2. Цикл для обработки вводимых данных.

        1.2.1. Выдача сигнала объекту чтения для ввода очередной строки данных (количество этаже, вместимость лифта).

        1.2.2. Отработка операции чтения очередной строки входных данных.

        1.2.3. Исходя из значения количества этажей, создание соответствующего количества объектов этажей. Установка связей сигналов и обработчиков с новым объектом.

    1.3. Установка связей сигналов и обработчиков между объектами.

2. Вызов от объекта «система» метода exec_app ( ).

    2.1. Цикл для обработки вводимых команд.

        2.1.1. Определение номера очередного такта.

        2.1.2. Выдача сигнала объекту ввода для чтения очередной

команды.

2.1.3. Отработка команды.

2.1.4. Отработка действий согласно такту.

2.2. После ввода команды «Turn off the system» завершить работу.

Все приведенные сигналы и соответствующие обработчики должны быть реализованы. Запрос от объекта означает выдачу сигнала. Все сообщения на консоль выводятся с новой строки.

В набор поддерживаемых команд добавить команду «SHOWTREE» и по этой команде вывести дерево иерархии объектов системы с отметкой о готовности и завершить работу системы (программы). Реализовать два отладочных теста такой командой. Первый после завершения построения дерева иерархии объектов. Второй перед завершением работы системы. Во втором тесте обязательно отработать не менее одной команды запроса пассажира.

1.1 Описание входных данных

Первая строка, количество этажей офисного здания n и вместимость лифта m:

```
«целое число» «целое число»
```

Вторая строка:

```
End of settings
```

Далее построчно вводятся команды. Они могут следовать в произвольном порядке.

Команда подхода пассажира к лифту на этаже и запроса.

```
Passenger «номер этажа» «номер целевого этажа» «ФИО»
```

После подхода пассажира на этаж, объект пассажира на дереве иерархии объектов подчиняется объекту этажа.

Команда отображения состояния лифта:

```
Elevator condition
```

Команда отображения состояния пассажира:

```
Passenger condition «ФИО»
```

Команда отображения состояния на этаже:

```
Condition on the floor «номер этажа»
```

Команда отображения состояния системы:

```
System status
```

Пустая команда (строка ничего не содержит). Элементы системы выполняют действия согласно такту.

Команда завершения работы системы.

```
Turn off the system
```

Пример ввода:

```
10 4
End of settings
Passenger 1 10 Ivanov I.I.
Passenger 7 2 Petrov A.V.
Passenger 3 9 Cidorov K.V.
System status
System status
System status
Elevator condition
System status
System status
System status
System status
System status
System status
System status
System status
System status
Passenger condition Petrov A.V.
System status
System status
System status
System status
Turn off the system
```

1.2 Описание выходных данных

После завершения ввода исходных данных выводиться текст:

```
Ready to work
```

После команды отображения состояния лифта вывести.

Если лифт движется:

```
Elevator condition: «номер этажа» «количество пассажиров» direction «up|down»
```

Если лифт стоит:

```
Elevator condition: «номер этажа» «количество пассажиров» stop
```

После команды отображения состояния пассажира.

Если пассажир ожидает лифт на этаже:

```
Passenger condition: «ФИО» on the floor «номер этажа»
```

Если пассажир в лифте:

```
Passenger condition: «ФИО» in the elevator
```

Если пассажир не найден:

```
Passenger «ФИО» not found
```

После команды отображения состояния этажа:

```
Condition on the floor «номер этажа» -up-  «ФИО»  «ФИО» . . .
Condition on the floor «номер этажа» -down-  «ФИО»  «ФИО» . . .
```

Где после «-up-» ФИО пассажиров направленных вверх, согласно очередности подхода на этаж.

После «-down-» ФИО пассажиров направленных вниз, согласно очередности подхода на этаж.

После команды отображения состояния системы:

```
«номер такта»: Elevator: «номер этажа расположения лифта» «количество пассажиров в лифте»  Floors: («номер этажа»: «количество пассажиров с запросом вверх»-«количество пассажиров с запросом вниз») («номер этажа»: «количество пассажиров с запросом вверх»-«количество пассажиров с запросом вниз») . . . .
```

Отображается информация об этажах, на которых находится хотя бы один пассажир. Этажи выводятся по возрастанию номеров.

Команда завершения работы системы вывести.

```
Turn off the system
```

Пример вывода:

```
Ready to work
4: Elevator: 3 1  Floors: (3: 1-0) (7: 0-1)
5: Elevator: 3 2  Floors: (7: 0-1)
6: Elevator: 4 2  Floors: (7: 0-1)
Elevator condition: 5 2 direction up
8: Elevator: 6 2  Floors: (7: 0-1)
9: Elevator: 7 2  Floors: (7: 0-1)
10: Elevator: 8 2  Floors: (7: 0-1)
11: Elevator: 9 2  Floors: (7: 0-1)
12: Elevator: 9 1  Floors: (7: 0-1)
13: Elevator: 10 1  Floors: (7: 0-1)
14: Elevator: 10 0  Floors: (7: 0-1)
15: Elevator: 9 0  Floors: (7: 0-1)
16: Elevator: 8 0  Floors: (7: 0-1)
Passenger condition: Petrov A.V. on the floor 7
18: Elevator: 7 1  Floors:
19: Elevator: 6 1  Floors:
20: Elevator: 5 1  Floors:
21: Elevator: 4 1  Floors:
Turn off the system
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект ob_cl_application класса Cl_application предназначен для конструирования и запуска системы;
- объект стандартоного потока ввода с клавиатуры cin;
- объект стандартного потока вывода на экран cout;
- условный оператор if..else;
- оператор цикла for;
- оператор цикла с предусловием while;
- структура connection с полями: p_signal - указатель на метод сигнала, p_target - указатель на объект класса Cl_base, p_handler - указатель на метод обработчика;
- указатель на метод сигнала TYPE_SIGNAL;
- указатель на метод обработчика TYPE_HANDLER.

Класс Cl_base:

- свойства/поля:
    - поле наименование объекта:
        - наименование — s_object_name;
        - тип — string;
        - модификатор доступа — private;
    - поле указатель на головной объект для текущего объекта:
        - наименование — p_head_object;
        - тип — Cl_base*;
        - модификатор доступа — private;
    - поле класс объекта:
        - наименование — object_class;

        - тип — int;
        - модификатор доступа — public;
    - поле состояние объекта:
        - наименование — object_state;
        - тип — int;
        - модификатор доступа — private;
    - поле динамический массив указателей на объекты, подчиненные текущему:
        - наименование — subordinate_objects;
        - тип — vector<Cl_base*>;
        - модификатор доступа — private;
    - поле динамический массив связей сигнал-обработчик:
        - наименование — connects;
        - тип — vector<connection*>;
        - модификатор доступа — private;
- функционал:
    - метод activate — приведение объектов в состояние готовности;
    - метод get_subordinate_objects — получение вектора указателей на подчиненные объекты.

Класс Cl_application:

- свойства/поля:
    - поле флаг на выполнение программы:
        - наименование — run;
        - тип — bool;
        - модификатор доступа — private;
    - поле номер такта:
        - наименование — tik;

        - тип — int;
        - модификатор доступа — private;
- функционал:
    - метод Cl_application — параметризированный конструктор;
    - метод buidl_tree_objects — метод конструирования системы;
    - метод exec_app — метод запуска системы;
    - метод handler_add_floor — метод обработчика сигнала добавления этажа;
    - метод handler_turn_off — метод обработчика сигнала завершения работы;
    - метод handler_syscon — метод обработчика сигнала вывода состояния системы;
    - метод handler_show_tree — метод обработчика сигнала вывода иерархии объектов;
    - метод signal_input — метод сигнала ввода;
    - метод signal_output — метод сигнала вывода;
    - метод signal_upd — метод сигнала обновления состояния системы.

Класс Elevator:

- свойства/поля:
    - поле наименование объекта лифта:
        - наименование — elevator_name;
        - тип — const string;
        - модификатор доступа — public;
    - поле вместимость лифта:
        - наименование — mass;
        - тип — int;
        - модификатор доступа — public;

- поле номер текущего этажа:
  - наименование — floor;
  - тип — int;
  - модификатор доступа — public;
- поле направление движения:
  - наименование — dir;
  - тип — int;
  - модификатор доступа — public;
- поле предыдущее направление движения:
  - наименование — prev_dir;
  - тип — int;
  - модификатор доступа — public;

- функционал:
  - метод Elevator — параметризированный конструктор;
  - метод signal_output — метод сигнала вывода;
  - метод handler_condition — метод обработчика сигнала вывода состояния лифта;
  - метод handler_receive_mass — метод обработчика сигнала изменения вместимости лифта;
  - метод handler_syscon — метод обработчика сигнала вывода состояния системы;
  - метод upd — метод обновления состояния.

Класс Floor:

- свойства/поля:
  - поле наименование объекта этажа:
    - наименование — floor_name;
    - тип — const string;

        - модификатор доступа — public;
    - поле номер этажа:
        - наименование — number;
        - тип — int;
        - модификатор доступа — public;
- функционал:
    - метод Floor — параметризированный конструктор;
    - метод signal_button_up — метод сигнала нажатия кнопки "вверх";
    - метод signal_button_down — метод сигнала нажатия кнопки "вниз";
    - метод signal_output — метод сигнала вывода;
    - метод handler_add_passenger — метод обработчика сигнала добавления пассажира;
    - метод handler_condition — метод обработчика сигнала вывода состояния этажа;
    - метод handler_syscon — метод обработчика сигнала вывода состояния системы.

Класс Input:

- функционал:
    - метод Input — параметризированный конструктор;
    - метод signal_input — метод сигнала ввода;
    - метод handler_input — метод обработчика сигнала ввода.

Класс Output:

- функционал:
    - метод Output — параметризированный конструктор;
    - метод handler_output — метод обработчика сигнала вывода.

Класс Passenger:

- свойства/поля:

- поле номер желаемого этажа:
    - наименование — wish_floor;
    - тип — int;
    - модификатор доступа — public;

- функционал:
    - метод Passenger — параметризированный конструктор.

Класс RemoteController:

- свойства/поля:
    - поле наименование объекта управления системой:
        - наименование — remote_controller_name;
        - тип — const string;
        - модификатор доступа — public;
    - поле очередь пассажиров:
        - наименование — req;
        - тип — int;
        - модификатор доступа — private;
- функционал:
    - метод RemoteController — параметризированный конструктор;
    - метод handler_upd — метод обработчика сигнала обновления состояния;
    - метод handler_input — метод обработчика сигнала ввода;
    - метод handler_button — метод обработчика сигнала нажатия на кнопку вызова;
    - метод signal_add_floor — метод сигнала добавления этажа;
    - метод signal_set_mass — метод сигнала изменения вместимости лифта;
    - метод signal_turn_off — метод сигнала завершения работы системы;

- метод signal_add_passenger — метод сигнала добавления пассажира;
- метод signal_output — метод сигнала вывода;
- метод signal_floor_condition — метод сигнала вывода состояния этажа;
- метод signal_elevator_condition — метод сигнала вывода состояния лифта;
- метод signal_syscon — метод сигнала вывода состояния системы;
- метод signal_show_tree — метод сигнала вывода иерархии объектов.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	Cl_base			Базовый класс	
		Cl_application	public		2
		Elevator	public		3
		Floor	public		4
		Input	public		5
		Output	public		6
		Passenger	public		7
		RemoteController	public		8
2	Cl_application			Класс-приложение	
3	Elevator			Класс лифта	
4	Floor			Класс этажа	
5	Input			Класс ввода данных	
6	Output			Класс вывода данных	
7	Passenger			Класс пассажира	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
8	RemoteController			Класс управления системой	

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм метода activate класса Cl_base

Функционал: приведение объектов в состояние готовности.

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода activate класса Cl_base

№	Предикат	Действия	№ перехода
1		Вызов метода change_object_state с параметром 1	2
2	Перебираем элементы вектора subordinate_objects, subordinate_object - указатель на текущий элемент	Вызов метода activate объекта по адресу subordinate_object	2
			∅

3.2 Алгоритм метода get_subordinate_objects класса Cl_base

Функционал: получение вектора указателей на подчиненные объекты.

Параметры: нет.

Возвращаемое значение: Вектор указателей на объекта класса Cl_base.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода get_subordinate_objects класса Cl_base

№	Предикат	Действия	№ перехода
1		Вернуть subordinate_objects	∅

3.3 Алгоритм конструктора класса Cl_application

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на объект класса Cl_base, строка s_name_object.

Алгоритм конструктора представлен в таблице 4.

Таблица 4 – Алгоритм конструктора класса Cl_application

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Cl_base с параметрами p_head_object и s_name_object	2
2		Вызов метода change_object_name с параметром "object_system"	3
3		Свойству tik присвоить значение 1	∅

3.4 Алгоритм метода buidl_tree_objects класса Cl_application

Функционал: Метод конструирования системы.

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода buidl_tree_objects класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление указателя remote_control на объект класса RemoteController и инициализация новым объектом класса RemoteController с параметром указателя на текущий объект	2

№	Предикат	Действия	№ перехода
2		Объявление указателя elevator на объект класса Elevator и инициализация новым объектом класса Elevator с параметром remote_control	3
3		Объявление указателя input на объект класса Input и инициализация новым объектом класса Input с параметром указателя на текущий объект	4
4		Объявление указателя output на объект класса Output и инициализация новым объектом класса Output с параметром указателя на текущий объект	5
5		Вызов метода activate	6
6		Вызов метода set_connect с параметрами: указатель на метод сигнала signal_input класса Cl_application, input, указатель на метод обработчика handler_input класса Input	7
7		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_set_mass класса RemoteController, elevator, указатель на метод обработчика handler_receive_mass класса Elevator	8
8		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_add_floor класса RemoteController, указатель на текущий объект, указатель на метод обработчика handler_add_floor класса Cl_application	9
9		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_syscon класса RemoteController, указатель на текущий объект, указатель на метод обработчика handler_syscon класса Cl_application	10
10		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_syscon класса RemoteController, elevator, указатель на метод обработчика handler_syscon класса Elevator	11

№	Предикат	Действия	№ перехода
11		Объявление строки message	12
12		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_input, message	13
13		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_input, message	14
14		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_turn_off класса RemoteController, указатель на текущий объект, указатель на метод обработчика handler_turn_off класса Cl_application	15
15		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_output класса RemoteController, output, указатель на метод обработчика handler_output класса Output	16
16		Вызов метода set_connect с параметрами: указатель на метод сигнала signal_output класса Cl_application, output, указатель на метод обработчика handler_output класса Output	17
17		Вызов метода set_connect с параметрами: указатель на метод сигнала signal_upd класса Cl_application, remote_control, указатель на метод обработчика handler_upd класса RemoteController	18
18		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_elevator_condition класса RemoteController, elevator, указатель на метод обработчика handler_condition класса Elevator	19
19		Вызов метода set_connect объекта по адресу elevator с параметрами: указатель на метод сигнала signal_output класса Elevator, output, указатель на метод обработчика handler_output класса Output	20
20		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_show_tree класса RemoteController, указатель на текущий объект, указатель на метод	∅

№	Предикат	Действия	№ перехода
		обработчика handler_show_tree класса Cl_application	

3.5 Алгоритм метода exec_app класса Cl_application

Функционал: Метод запуска системы.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода exec_app класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление строки message и инициализация значением "Ready to work"	2
2		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_output, message	3
3		Свойству run присвоить значение true	4
4		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_input, message	5
5	run равно true?	Объявление строки text и инициализация пустой строкой	6
			8
6		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_upd, text	7
7		Увеличение tik на 1	4
8		Возврат 0	∅

3.6 Алгоритм метода handler_add_floor класса Cl_application

Функционал: Метод обработчика сигнала добавления этажа.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода handler_add_floor класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление целочисленной переменной num и инициализация результатом функции stoi с параметром message	2
2		Объявления указателя remote_control на объект класса Cl_base и инициализация результатом метода get_sub_object_by_name с параметром remote_control_name	3
3		Объявления указателя output на объект класса Cl_base и инициализация результатом метода get_sub_object_by_name с параметром "object_display_screen"	4
4		Объявление целочисленной переменной-счетчика i и инициализация 0	5
5	i < num?	Объявление указателя f на объект класса Floor и инициализация новым объектом класса Floor с параметрами this и i+1	6
			∅
6		Вызов метода activate объекта по адресу f	7
7	i > 0	Вызов метода set_connect объекта по адресу f с параметрами: указатель на метод сигнала signal_button_down класса Floor, remote_control, указатель на метод обработчика handler_button класса RemoteController	8
			8
8	i < n - 1?	Вызов метода set_connect объекта по адресу f с	9

№	Предикат	Действия	№ перехода
		параметрами: указатель на метод сигнала signal_button_up класса Floor, remote_control, указатель на метод обработчика handler_button класса RemoteController	
			9
9		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_add_passenger класса RemoteController, f, указатель на метод обработчика handler_add_passenger класса Floor	10
10		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_floor_condition класса RemoteController, f, указатель на метод обработчика handler_condition класса Floor	11
11		Вызов метода set_connect объекта по адресу remote_control с параметрами: указатель на метод сигнала signal_syscon класса RemoteController, f, указатель на метод обработчика handler_syscon класса Floor	12
12		Вызов метода set_connect объекта по адресу f с параметрами: указатель на метод сигнала signal_output класса Floor, output, указатель на метод обработчика handler_output класса Output	13
13		Увеличение i на 1	5

3.7 Алгоритм метода handler_turn_off класса Cl_application

Функционал: Метод обработчика сигнала завершения работы.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода handler_turn_off класса Cl_application

№	Предикат	Действия	№ перехода
1		Свойству run присвоить значение false	2
2		message присвоить перенос на новую строку + message	3
3		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_output, message	∅

3.8 Алгоритм метода handler_syscon класса Cl_application

Функционал: Метод обработчика сигнала вывода состояния системы.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода handler_syscon класса Cl_application

№	Предикат	Действия	№ перехода
1		message присвоить "\n" + результат функции to_string с параметром tik + ": "	2
2		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_output, message	∅

3.9 Алгоритм метода handler_show_tree класса Cl_application

Функционал: Метод обработчика сигнала вывода иерархии объектов.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода handler_show_tree класса Cl_application

№	Предикат	Действия	№ перехода
1		Свойству run присвоить значение false	2
2		message присвоить перенос на новую строку	3
3		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_output, message	4
4		Вызов метода show_object_tree_full	∅

3.10 Алгоритм метода signal_input класса Cl_application

Функционал: Метод сигнала ввода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода signal_input класса Cl_application

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.11 Алгоритм метода signal_output класса Cl_application

Функционал: Метод сигнала вывода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода signal_output класса Cl_application

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.12 Алгоритм метода signal_upd класса Cl_application

Функционал: Метод сигнала обновления состояния системы.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода signal_upd класса Cl_application

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.13 Алгоритм конструктора класса Elevator

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на объект класса Cl_base.

Алгоритм конструктора представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса Elevator

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Cl_base с параметрами p_head_object и elevator_name	2
2		Присвоить свойствам dir, floor, prev_dir значения stop, 1, up соответсвенно	∅

3.14 Алгоритм метода signal_output класса Elevator

Функционал: Метод сигнала вывода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода signal_output класса Elevator

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.15 Алгоритм метода handler_condition класса Elevator

Функционал: Метод обработчика сигнала вывода состояния лифта.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода handler_condition класса Elevator

№	Предикат	Действия	№ перехода
1		Объявление строки output и инициализация значением: "Elevator condition: " + результат функции to_string с параметром floor + результат функции to_string с параметром размера результата метода get_subordinate_objects	2
2	dir не равно stop?	output присвоить значение output + "direction "	3
		output присвоить значение output + "stop"	4
3	dir равно up?	output присвоить значение output + "up"	4
		output присвоить значение output + "down"	4
4		Вызов метода emit_signal с параметрами:	∅

№	Предикат	Действия	№ перехода
		указатель на метод сигнала signal_input, output	

3.16 Алгоритм метода handler_receive_mass класса Elevator

Функционал: Метод обработчика сигнала изменения вместимости лифта.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода handler_receive_mass класса Elevator

№	Предикат	Действия	№ перехода
1		mass присвоить результат функции stoi с параметром message	∅

3.17 Алгоритм метода handler_syscon класса Elevator

Функционал: Метод обработчика сигнала вывода состояния системы.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода handler_syscon класса Elevator

№	Предикат	Действия	№ перехода
1		message присвоить значение: "Elevator: " + результат функции to_string с параметром floor + результат функции to_string с параметром размера результата метода get_subordinate_objects + " Floor:"	2
2		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_input, message	∅

3.18 Алгоритм метода upd класса Elevator

Функционал: Метод обновления состояния.

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода upd класса Elevator

№	Предикат	Действия	№ перехода
1		Увеличение floor на dir	∅

3.19 Алгоритм конструктора класса Floor

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на объект класса Cl_base, целочисленная переменная number.

Алгоритм конструктора представлен в таблице 20.

Таблица 20 – Алгоритм конструктора класса Floor

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Cl_base с параметрами p_head_object и floor_name + результат функции to_string с параметром number	2
2		Своийству number присвоить значение параметра number	∅

3.20 Алгоритм метода signal_button_up класса Floor

Функционал: Метод сигнала нажатия кнопки "вверх".

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода signal_button_up класса Floor

№	Предикат	Действия	№ перехода
1		mesage присвоить результат функции to_string с параметром number	∅

3.21 Алгоритм метода signal_button_down класса Floor

Функционал: Метод сигнала нажатия кнопки "вниз".

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода signal_button_down класса Floor

№	Предикат	Действия	№ перехода
1		mesage присвоить результат функции to_string с параметром: number * (-1)	∅

3.22 Алгоритм метода signal_output класса Floor

Функционал: Метод сигнала вывода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода signal_output класса Floor

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.23 Алгоритм метода handler_add_passenger класса Floor

Функционал: Метод обработчика сигнала добавления пассажира.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода handler_add_passenger класса Floor

№	Предикат	Действия	№ перехода
1		Объявление целочисленных переменных num, i и инициализация 0	2
2	i-й элемент message не пробел?	Увеличение i на 1	2
		num присвоить результат функции stoi с параметром подстроки message до i-ого элемента	3
3	num не равно number?		∅
		Увеличение i на 1	4
4		Объявление целочисленных переменных wish_floor, j и инициализация значением i	5
5	j-й элемент message не пробел?	Увеличение j на 1	5
		wish_floor присвоить результат функции stoi с параметром подстроки message с i-ого элемента до (j - 1)-ого элемента	6
6		Объявление строки name и инициализация подстрокой message с (j + 1)-ого элемента	7
7		Объявление указателя p на объект класса Passenger и инициализация новым объектом класса Passenger с параметрами: указатель на текущий объект, name, wish_floor	8
8		Вызов метода activate объекта по адресу p	9
9	wish_floor > number?	Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_button_up, message	∅

№	Предикат	Действия	№ перехода
		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_button_down, message	∅

3.24 Алгоритм метода handler_condition класса Floor

Функционал: Метод обработчика сигнала вывода состояния этажа.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода handler_condition класса Floor

№	Предикат	Действия	№ перехода
1	Результат функции stoi с параметром message не равен number?		∅
		Объявление строк up, down и инициализация пустыми строками	2
2	Перебираем элементы вектора результата метода get_subordinate_objects, subordinate_object - указатель на текущий элемент		3
			4
3	Свойство wish_floor объекта по адресу subordinate_object > number?	up присвоить значение up + результат метода get_object_name объекта по адресу subordinate_object	2
		down присвоить значение down + результат метода get_object_name объекта по адресу	2

№	Предикат	Действия	№ перехода
		subordinate_object	
4		message присвоить "\nCondition on the floor " + значение message + " -up-" + значение up + "\nCondition on the floor " + значение message + " -down-" + значение down	5
5		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_output, message	∅

3.25 Алгоритм метода handler_syscon класса Floor

Функционал: Метод обработчика сигнала вывода состояния системы.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода handler_syscon класса Floor

№	Предикат	Действия	№ перехода
1		Объявление целочисленных переменных up, down и инициализация 0	2
2	Перебираем элементы вектора результата метода get_subordinate_objects, subordinate_object - указатель на текущий элемент		3
			4
3	Свойство wish_floor объекта по адресу subordinate_object > number?	Увеличение up на 1	2
		Увеличение down на 1	2

№	Предикат	Действия	№ перехода
4		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_output, message	5
5	up + down > 0?	message присвоить " (" + результат функции to_string с параметром number + ": " + результат функции to_string с параметром up + "-" + результат функции to_string с параметром down + ")"	6
			∅
6		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_output, message	∅

3.26 Алгоритм конструктора класса Input

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на объект класса Cl_base.

Алгоритм конструктора представлен в таблице 27.

Таблица 27 – Алгоритм конструктора класса Input

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Cl_base с параметрами p_head_object и "object_read_command"	∅

3.27 Алгоритм метода signal_input класса Input

Функционал: Метод сигнала ввода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода signal_input класса Input

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.28 Алгоритм метода handler_input класса Input

Функционал: Метод обработчика сигнала ввода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 29.

Таблица 29 – Алгоритм метода handler_input класса Input

№	Предикат	Действия	№ перехода
1		Ввод строки message	2
2		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_input, message	∅

3.29 Алгоритм конструктора класса Output

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на объект класса Cl_base.

Алгоритм конструктора представлен в таблице 30.

Таблица 30 – Алгоритм конструктора класса Output

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Cl_base с параметрами p_head_object и "object_display_screen"	∅

3.30 Алгоритм метода handler_output класса Output

Функционал: Метод обработчика сигнала вывода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 31.

Таблица 31 – Алгоритм метода handler_output класса Output

№	Предикат	Действия	№ перехода
1		Вывод на экран "(message)"	∅

3.31 Алгоритм конструктора класса Passenger

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на объект класса Cl_base, строка name, целочисленная переменная wish_floor.

Алгоритм конструктора представлен в таблице 32.

Таблица 32 – Алгоритм конструктора класса Passenger

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Cl_base с параметрами p_head_object и "object_FIO_" + name	2
2		Свойству wish_floor присвоить значение параметра wish_floor	∅

3.32 Алгоритм конструктора класса RemoteController

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на объект класса Cl_base.

Алгоритм конструктора представлен в таблице 33.

Таблица 33 – Алгоритм конструктора класса RemoteController

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса Cl_base с параметрами p_head_object и remote_controller_name	∅

3.33 Алгоритм метода handler_upd класса RemoteController

Функционал: Метод обработчика сигнала обновления состояния.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 34.

Таблица 34 – Алгоритм метода handler_upd класса RemoteController

№	Предикат	Действия	№ перехода
1		Объявление указателя elevator на объект класса Elevator и инициализация результатом метода get_sub_object_by_name с параметром elevator_name	2
2	elevator равен nullptr?		∅
		Объявление указателя floor на объект класса Floor и инициализация результатом метода get_object_by_coordinate с параметром: "/" + значение floor_name + результат функции to_string с параметром (свойство floor объекта по адресу elevator)	3
3	floor равен nullptr?		∅
			4
4	Значение свойства dir объекта по адресу elevator не равно stop?	Свойству prev_dir объекта по адресу elevator присвоить значение свойства dir объекта по адресу elevator	5

№	Предикат	Действия	№ перехода
			10
5	Перебираем элементы вектора результата метода get_subordinate_objects объекта по адресу elevator, subordinate_object - указатель на текущий элемент		6
			7
6	Значение свойства wish_floor объекта по адресу subordinate_object равно значению свойства floor объекта по адресу elevator?	Свойству dir объекта по адресу elevator присвоить значение stop	7
			5
7	Разность значения свойства mass объекта по адресу elevator и размера результата метода get_subordinate_objects объекта по адресу elevator > 0?		8
			10
8	Перебираем элементы вектора результата метода get_subordinate_objects объекта по адресу floor, subordinate_object - указатель на текущий элемент		9
			10
9	Разность значения свойства	Свойству dir объекта по адресу elevator присвоить	10

№	Предикат	Действия	№ перехода
	wish_floor объекта по адресу subordinate_object и значения свойства floor объекта по адресу elevator, помноженное на значение свойства dir объекта по адресу elevator > 0 или результат метода get_subordinate_objects объекта по адресу elevator пустая строка?	значение stop	
			8
10	Значение свойства dir объекта по адресу elevator равно stop?	Объявление логической переменной should_move и инициализация значением true	11
			29
11	Перебираем элементы вектора результата метода get_subordinate_objects объекта по адресу elevator, subordinate_object - указатель на текущий элемент		12
			14
12	Значение свойства wish_floor объекта по адресу subordinate_object равно значению свойства number объекта по адресу floor?	should_move присвоить значение false	13
			11

№	Предикат	Действия	№ перехода
13		Вызов метода delete_sub_object_by_name объекта по адресу elevator с параметром результата метода get_object_name объекта по адресу subordinate_object	11
14	Результат метода get_subordinate_objects объекта по адресу elevator пустая строка?		15
			20
15	Размер req > 0?		16
		Свойству prev_dir объекта по адресу elevator присвоить значение stop	20
16	Модуль 1-ого элеманта вектора req равен значению свойства number объекта по адресу floor?		17
			18
17	1-й элемент вектора req > 0?	Свойству prev_dir объекта по адресу elevator присвоить значение up	20
		Свойству prev_dir объекта по адресу elevator присвоить значение down	20
18	Разность модуля 1-ого элеманта вектора req и значения свойства number объекта по адресу floor > 0?	Свойству prev_dir объекта по адресу elevator присвоить значение up	20
			19
19		Свойству prev_dir объекта по адресу elevator присвоить значение down	20
20		Объявление целочисленной переменной entered и	21

№	Предикат	Действия	№ перехода
		инициализация 0	
21	Перебираем элементы вектора результата метода get_subordinate_objects объекта по адресу floor, subordinate_object - указатель на текущий элемент		22
			24
22	Значение свойства mass объекта по адресу elevator больше размера результата метода get_subordinate_objects объекта по адресу elevator и разность значения свойства wish_floor объекта по адресу subordinate_object и значения свойства number объекта по адресу floor, умноженная на значение свойства prev_dir объекта по адресу elevator > 0?	Вызов метода change_head_object объекта по адресу subordinate_object с параметром elevator	23
			21
23		should_move присвоить значение false Увеличение entered на 1	21
24		Объявление целочисленной переменной shift и инициализация 0	25
25	Перебираем элементы вектора req, i - номер		26

№	Предикат	Действия	№ перехода
	текущего элемента		
		Вызов метода resize вектора req с параметром: размер req - shift	28
26	entered не равен 0 и i-й элемент req умноженый на значение свойства prev_dir объекта по адресу elevator равен значению свойства number объекта по адресу floor?	Увеличение shift на 1 Уменьшение entered на 1	27
			27
27	Значение i больше либо равно значения shjft?	(i - shift)-ому элементу req присвоить значение i-ого элемента req	25
			25
28	Значение should_move равно true?	Свойству dir объекта по адресу elevator присвоить значение свойства prev_dir объекта по адресу elevator	29
			29
29		Вызов метода upd объекта по адресу elevator	∅

3.34 Алгоритм метода handler_input класса RemoteController

Функционал: Метод обработчика сигнала ввода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 35.

Таблица 35 – Алгоритм метода handler_input класса RemoteController

№	Предикат	Действия	№ перехода
1	message равно пустой строке?		∅
			2
2	message равно "End of settings"?		∅
			3
3	message равно "Turn off the system"?	Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_turn_off, message	∅
			4
4	message равно "SHOWTREE"?	Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_show_tree, message	∅
			5
5		Объявление вектора строк command	6
6	Перебираем элементы строки message, i - текущий символ		7
			8
7	i равен пробелу?	Добавить в конец вектора command пустую строку	6
		Добавить к последнему элементу вектора command символ i	6
8	1-й элемент command равен "Passenger"?		9
			16
9	2-й элемент command равен "condition"?	Объявление строки output	10
		message присвоить значение подстроки message с	15

№	Предикат	Действия	№ перехода
		10-ого элемента	
10		message присвоить значение подстроки message с 20-ого элемента	11
11		Объявление указателя passenger на объект класса Cl_base и инициализация результатом метода get_object_by_name с параметром: "object_FIO_" + message	12
12	passenger не равен nullptr?		13
		output присвоить "\nPassenger " + значение message + " not found"	14
13	Результат метода get_object_name объекта по адресу результата метода get_head_object объекта по адресу passenger равен elevator_name?	output присвоить "\nPassenger condition: " + значение message + " in the elevator"	14
		output присвоить "\nPassenger condition: " + значение message + " on the floor " + результат функции to_string с параметром number объекта по адресу результата метода get_head_object объекта по адресу passenger	14
14		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_output, output	∅
15		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_add_passenger, message	∅
16	1-й элемент command равен "Condition"?	Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_floor_condition, message	∅

№	Предикат	Действия	№ перехода
			17
17	1-й элемент command равен "Elevator"?	Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_elevator_condition, message	∅
			18
18	1-й элемент command равен "System"?	Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_syscon, message	∅
			19
19		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_set_mass, command[1]	20
20		Вызов метода emit_signal с параметрами: указатель на метод сигнала signal_add_floor, command[0]	∅

3.35 Алгоритм метода handler_button класса RemoteController

Функционал: Метод обработчика сигнала нажатия на кнопку вызова.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 36.

Таблица 36 – Алгоритм метода handler_button класса RemoteController

№	Предикат	Действия	№ перехода
1		Добавить в конец вектора req результат функции stoi с параметром message	∅

3.36 Алгоритм метода signal_add_floor класса RemoteController

Функционал: Метод сигнала добавления этажа.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 37.

Таблица 37 – Алгоритм метода signal_add_floor класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.37 Алгоритм метода signal_set_mass класса RemoteController

Функционал: Метод сигнала изменения вместимости лифта.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 38.

Таблица 38 – Алгоритм метода signal_set_mass класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.38 Алгоритм метода signal_turn_off класса RemoteController

Функционал: Метод сигнала завершения работы системы.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 39.

Таблица 39 – Алгоритм метода signal_turn_off класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.39 Алгоритм метода signal_add_passenger класса RemoteController

Функционал: Метод сигнала добавления пассажира.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 40.

Таблица 40 – Алгоритм метода signal_add_passenger класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.40 Алгоритм метода signal_output класса RemoteController

Функционал: Метод сигнала вывода.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 41.

Таблица 41 – Алгоритм метода signal_output класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.41 Алгоритм метода signal_floor_condition класса RemoteController

Функционал: Метод сигнала вывода состояния этажа.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 42.

Таблица 42 – Алгоритм метода signal_floor_condition класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.42 Алгоритм метода signal_elevator_condition класса RemoteController

Функционал: Метод сигнала вывода состояния лифта.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 43.

Таблица 43 – Алгоритм метода signal_elevator_condition класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.43 Алгоритм метода signal_syscon класса RemoteController

Функционал: Метод сигнала вывода состояния системы.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 44.

Таблица 44 – Алгоритм метода signal_syscon класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.44 Алгоритм метода signal_show_tree класса RemoteController

Функционал: Метод сигнала вывода иерархии объектов.

Параметры: Строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 45.

Таблица 45 – Алгоритм метода signal_show_tree класса RemoteController

№	Предикат	Действия	№ перехода
1		Подготовка данных сигнала	∅

3.45 Алгоритм функции main

Функционал: Конструирование и запуск системы.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм функции представлен в таблице 46.

Таблица 46 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Объявление объекта ob_cl_application класса Cl_application с параметром nullptr	2
2		Вызов метода build_tree_object объекта ob_cl_application	3
3		Возврат результата метода exec_app объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-11.

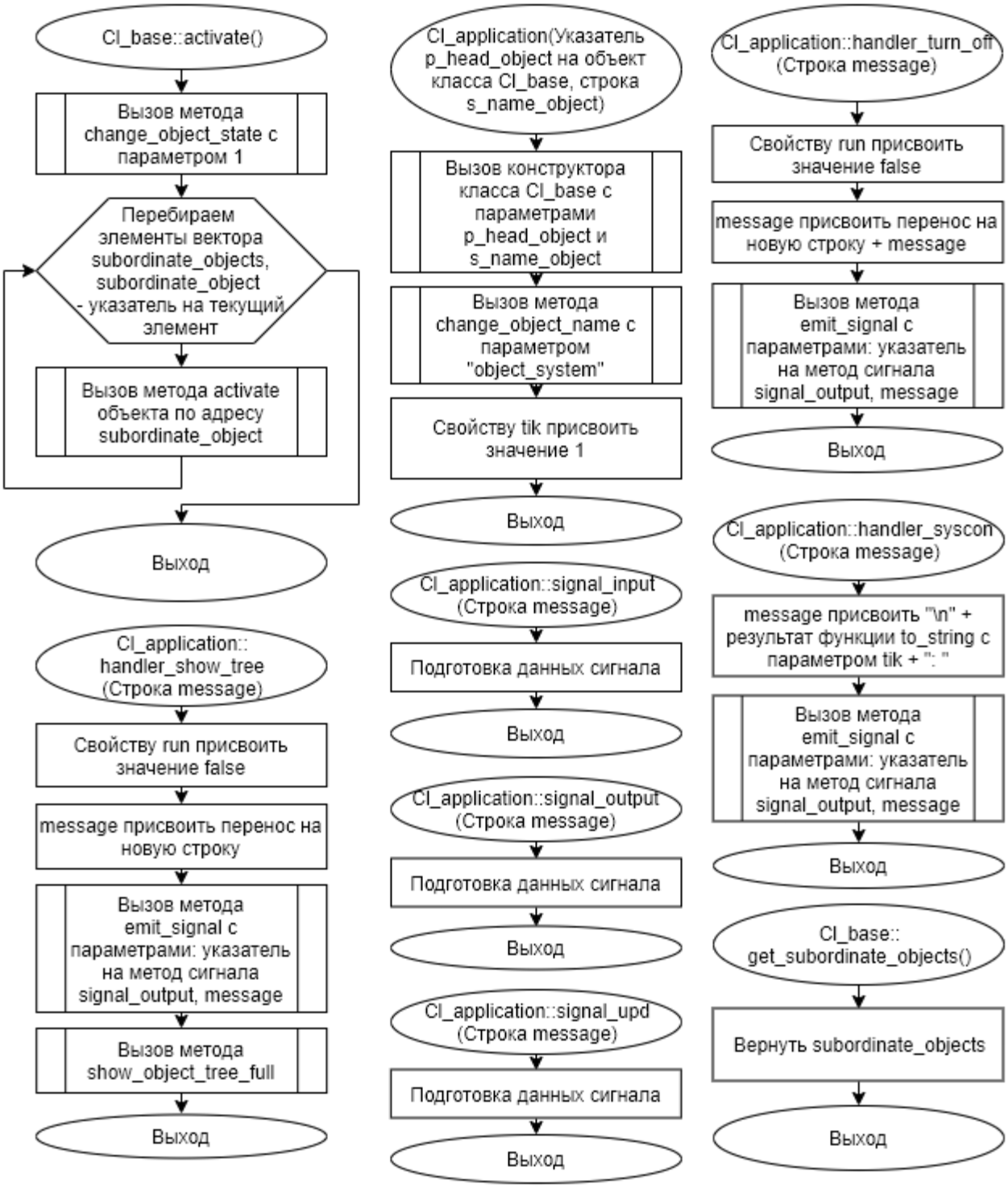

Рисунок 1 – Блок-схема алгоритма

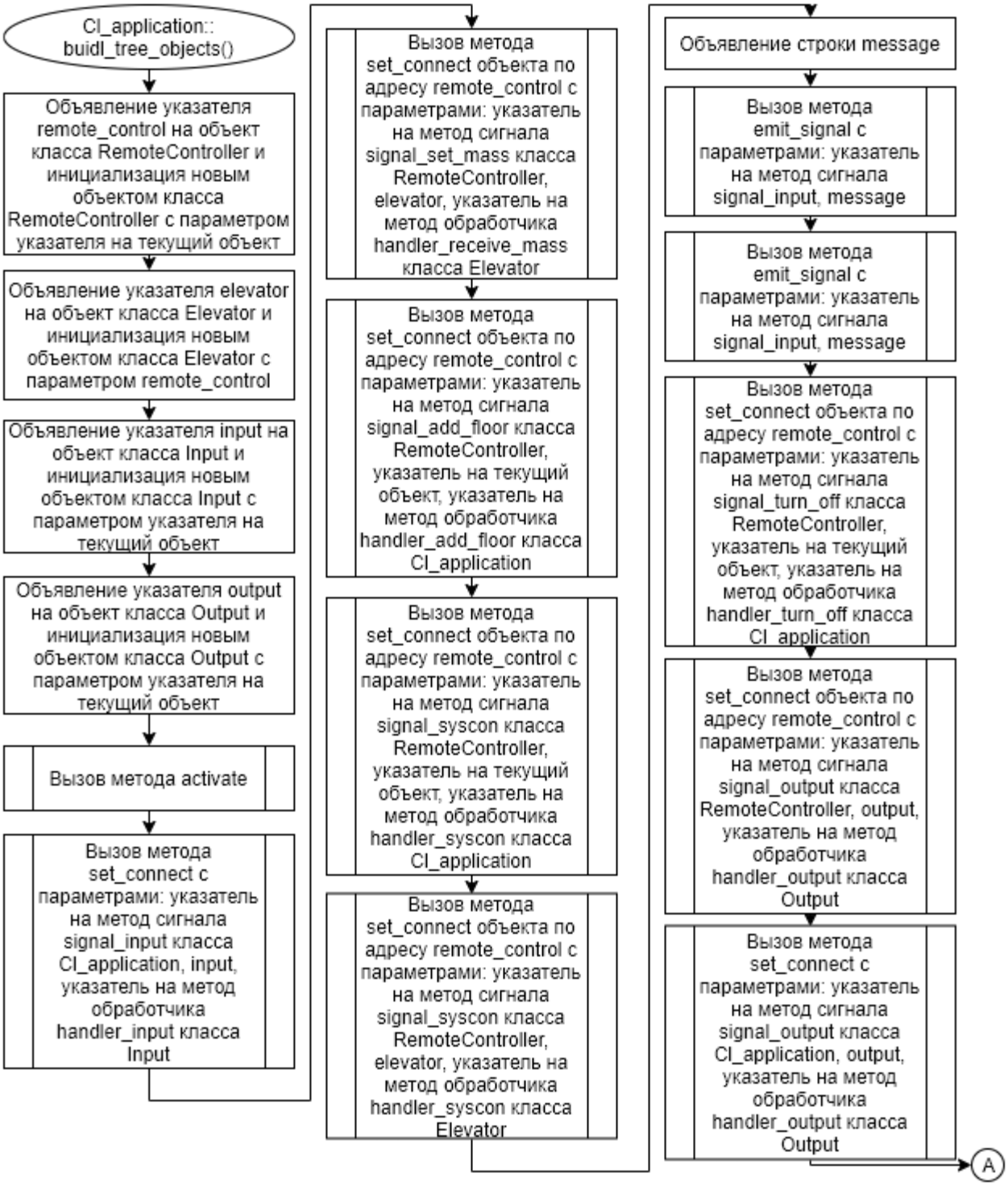

Рисунок 2 – Блок-схема алгоритма

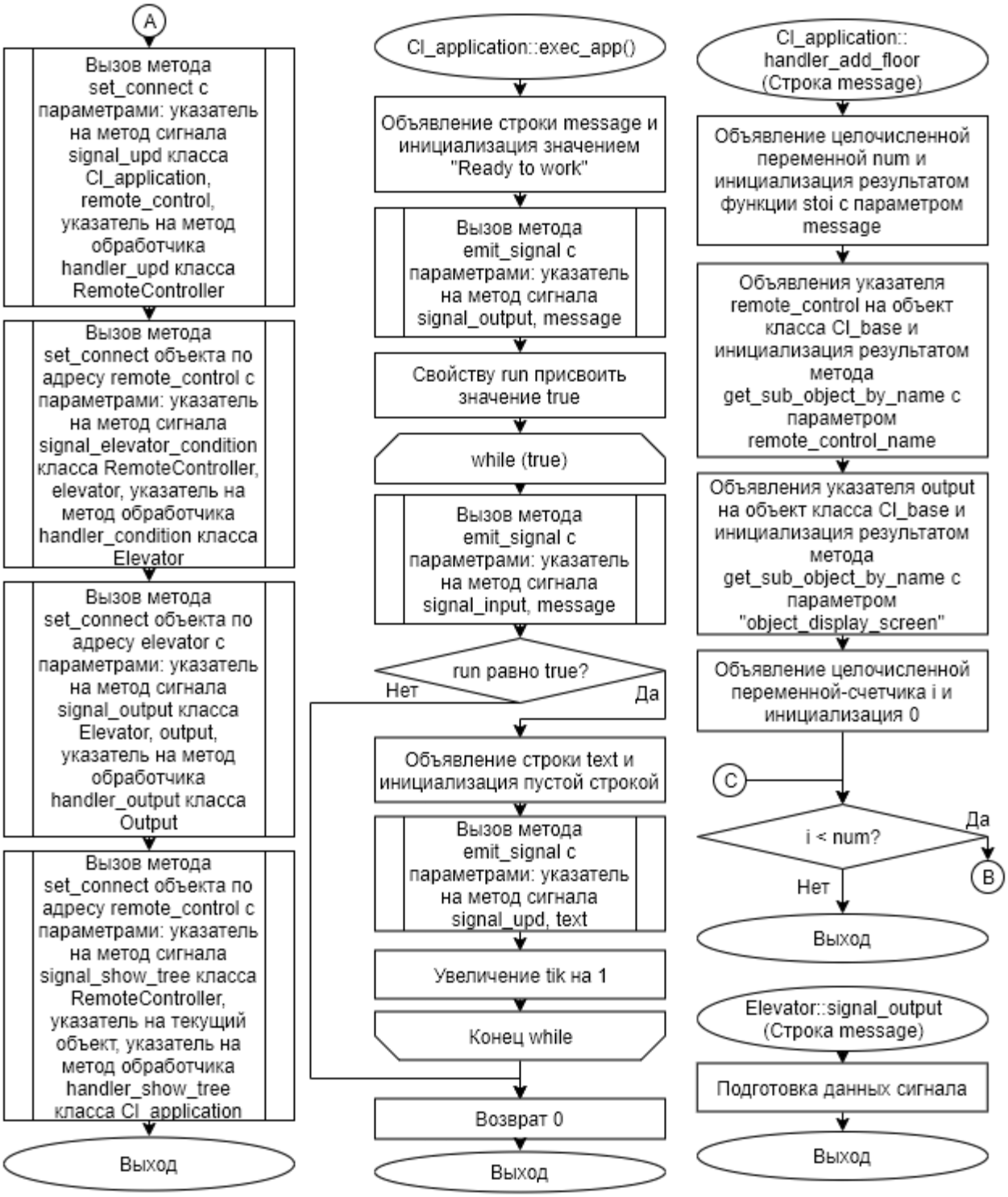

Рисунок 3 – Блок-схема алгоритма

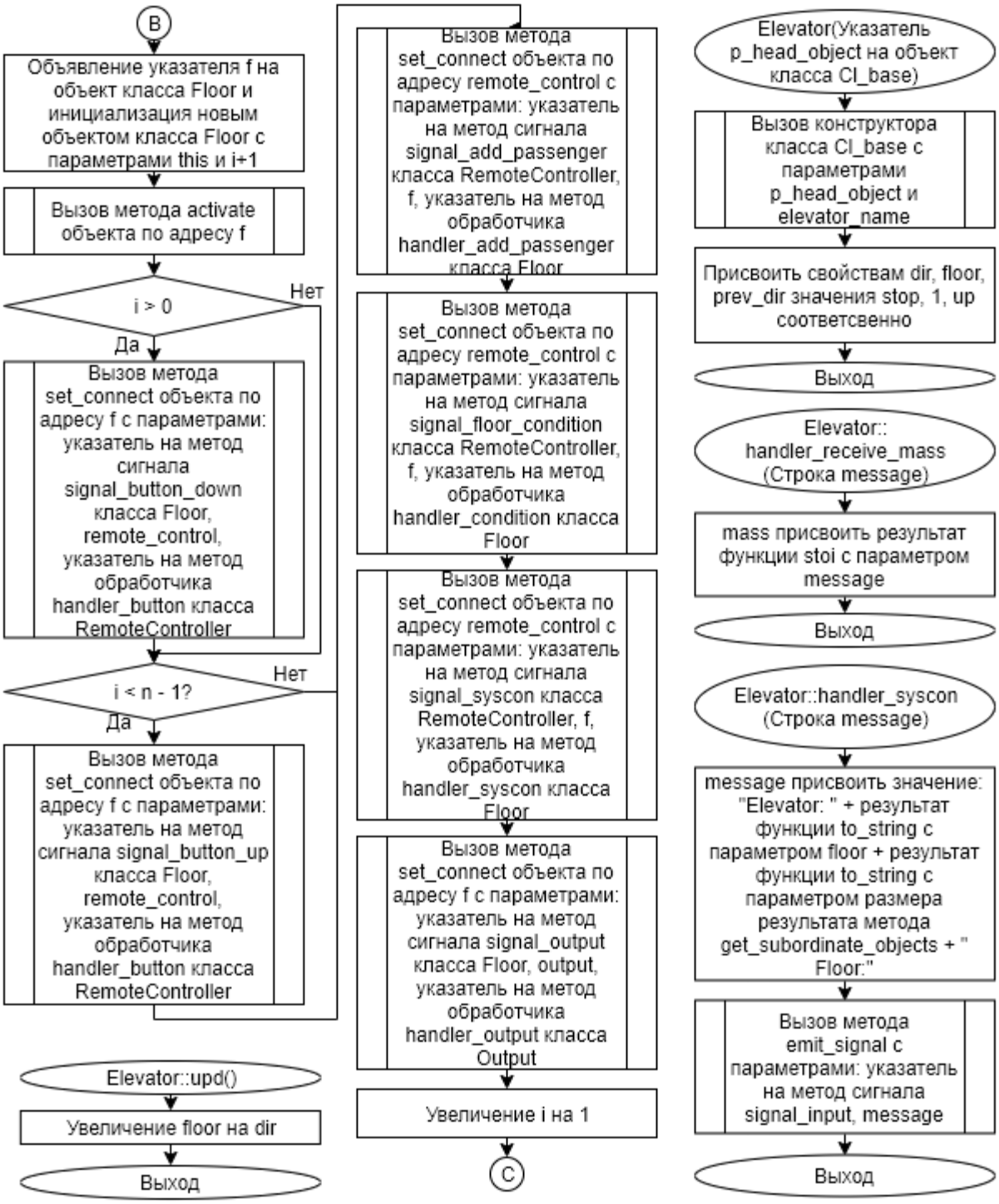

Рисунок 4 – Блок-схема алгоритма

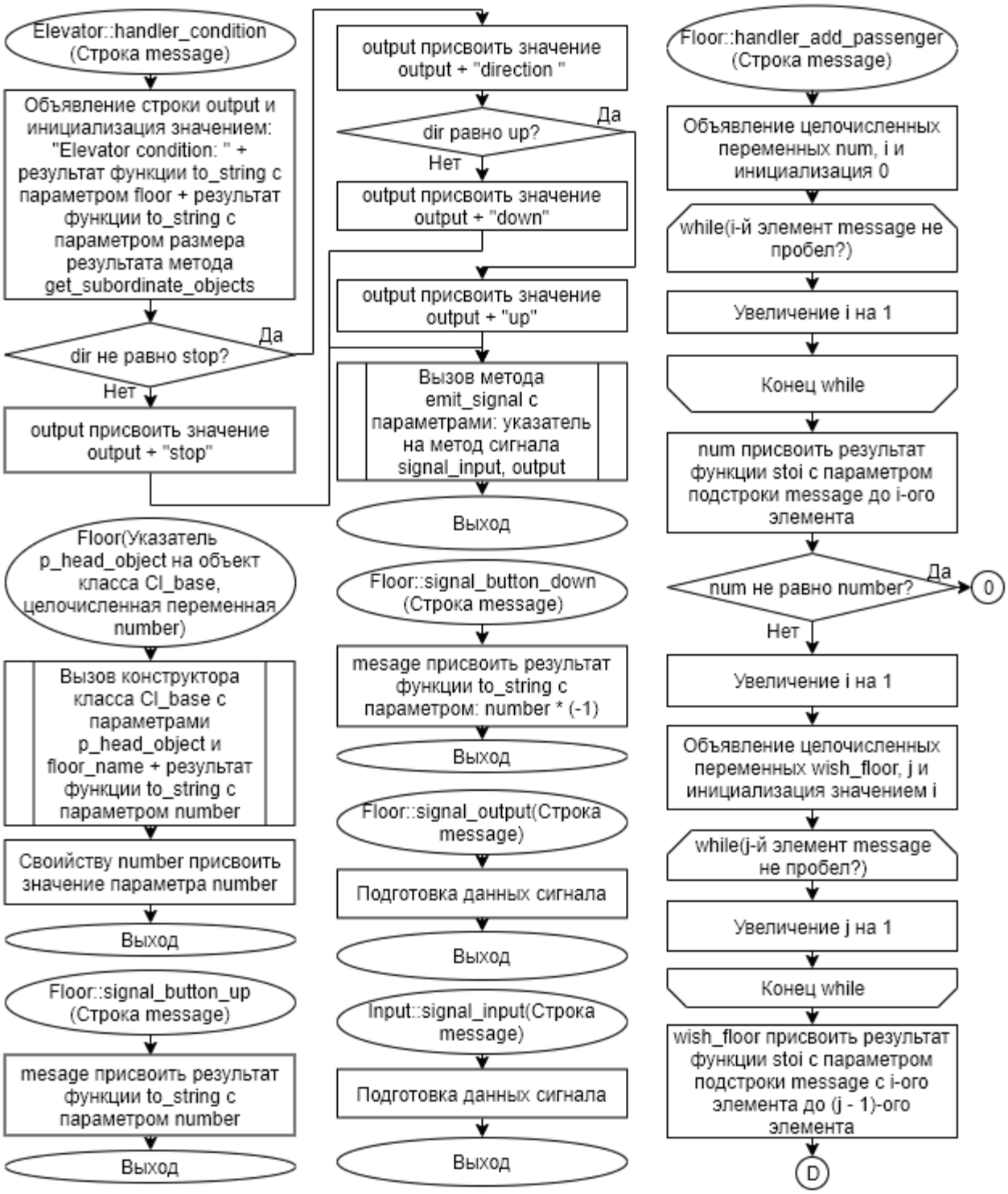

Рисунок 5 – Блок-схема алгоритма

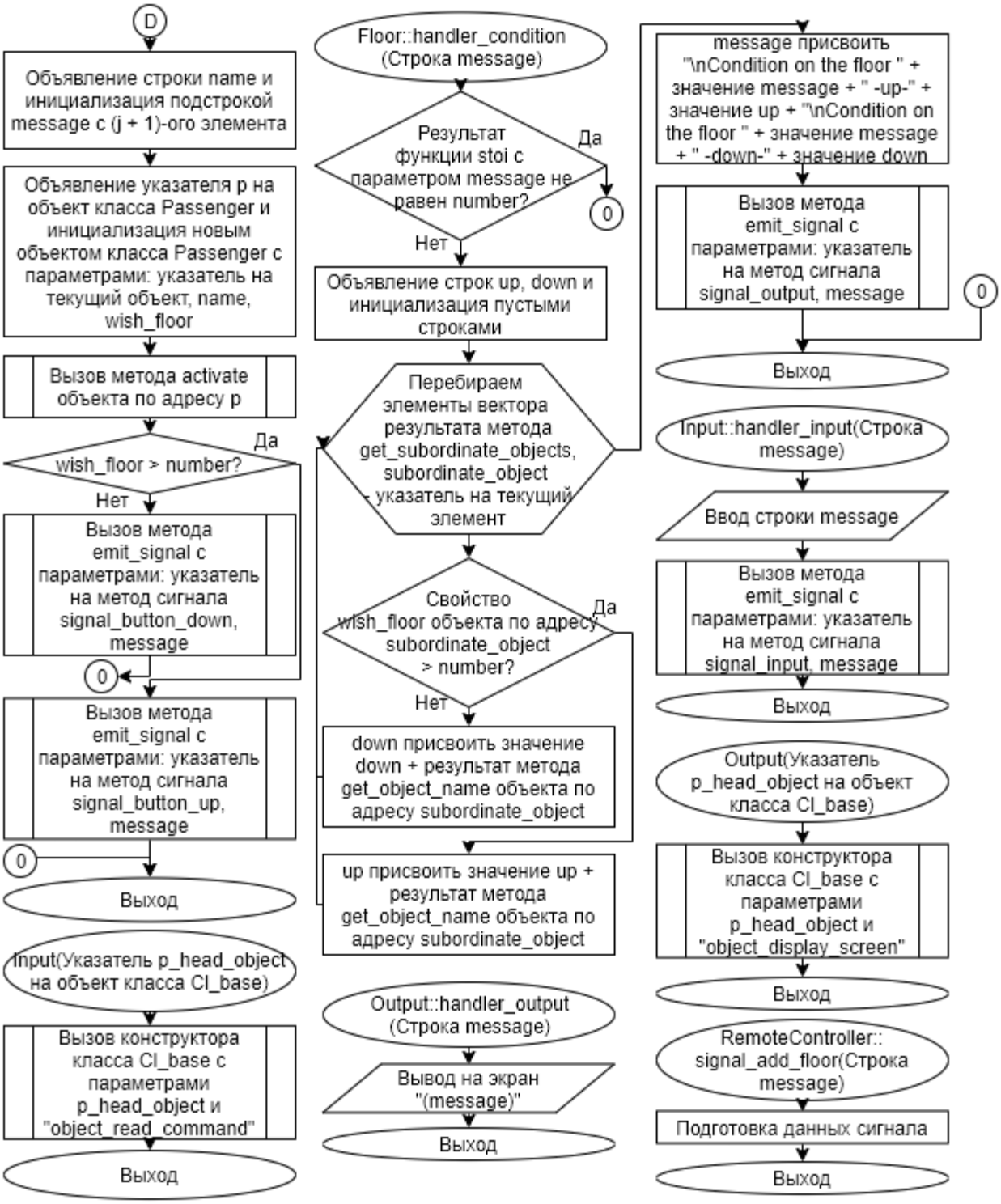

Рисунок 6 – Блок-схема алгоритма

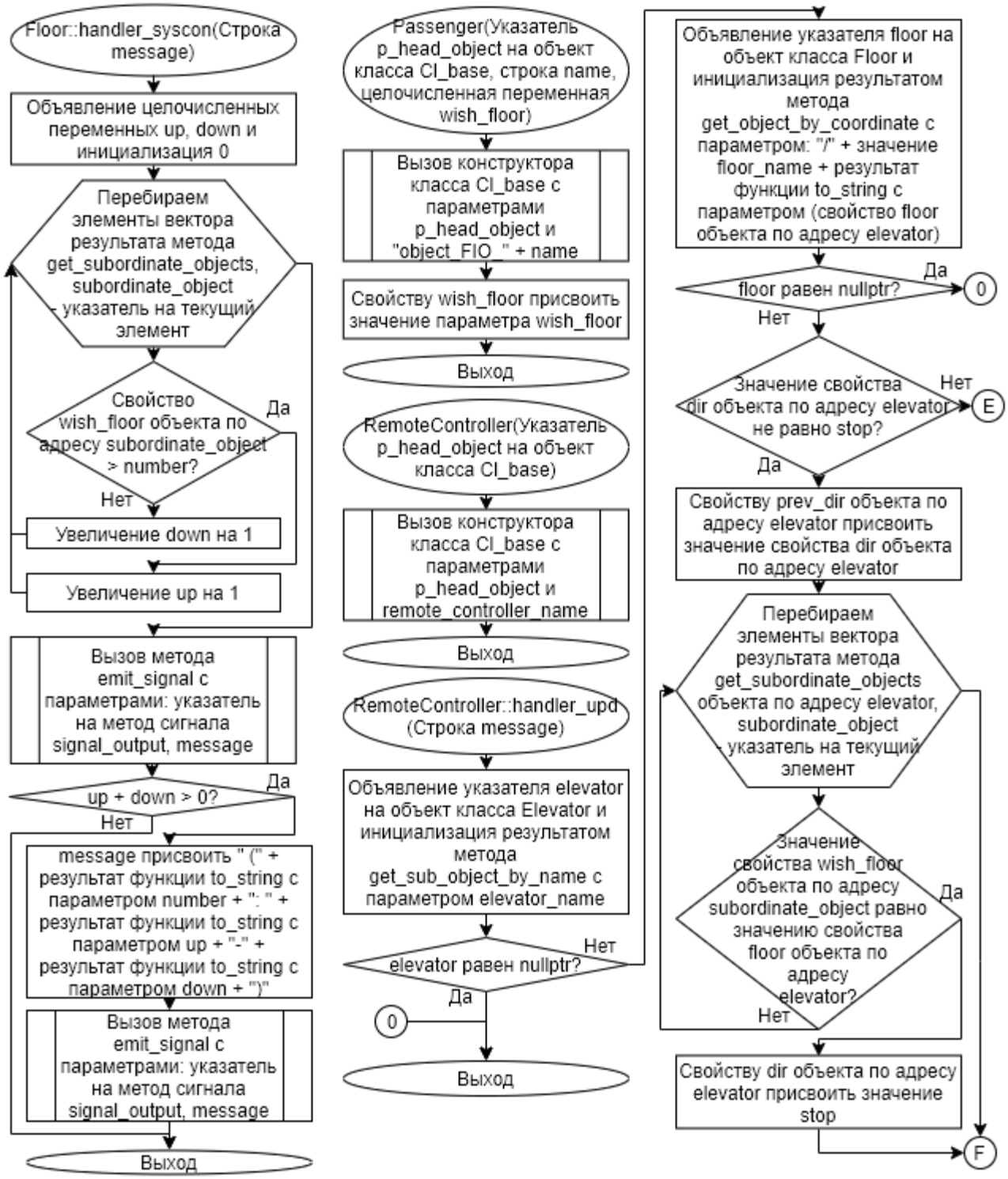

Рисунок 7 – Блок-схема алгоритма

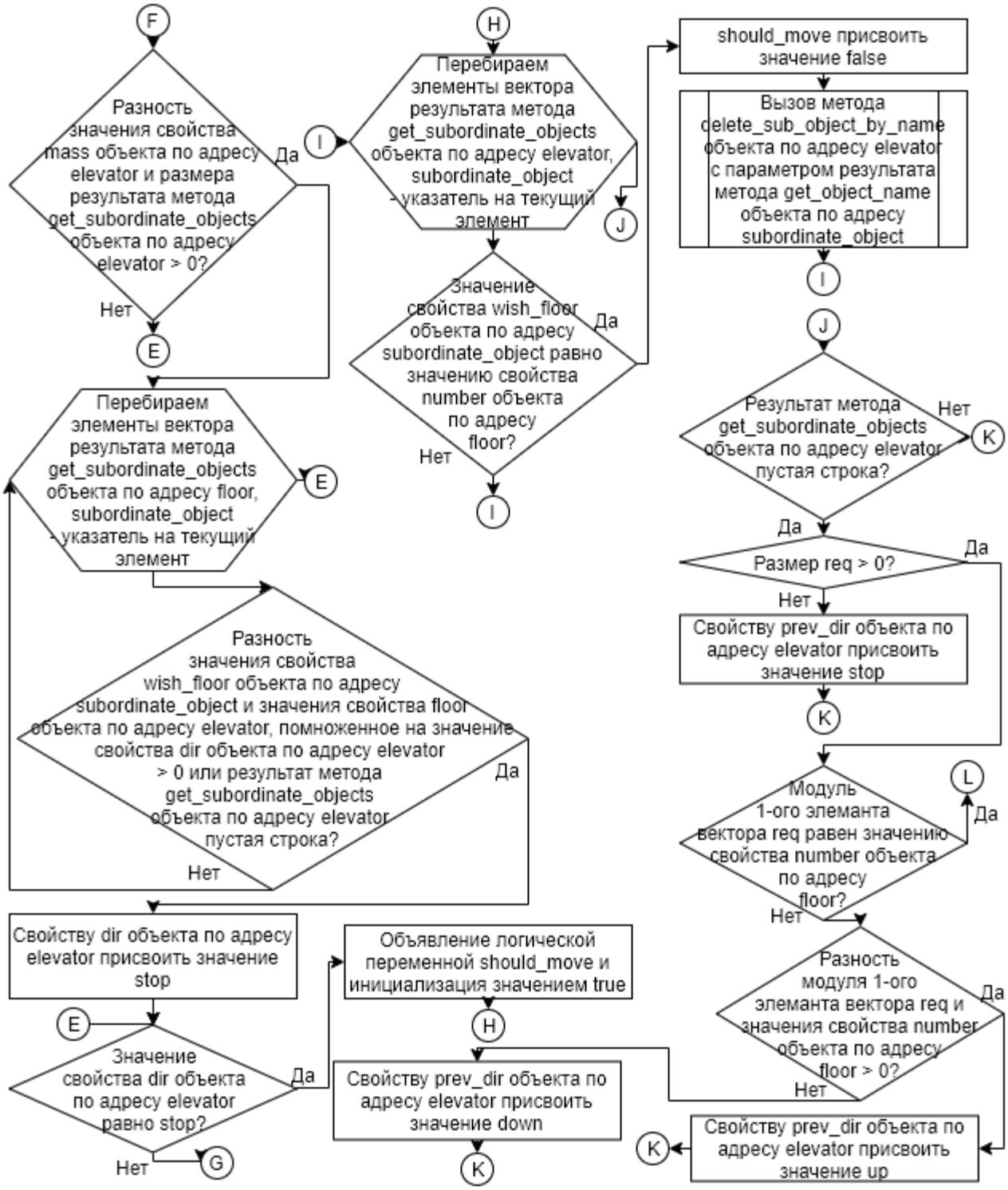

Рисунок 8 – Блок-схема алгоритма

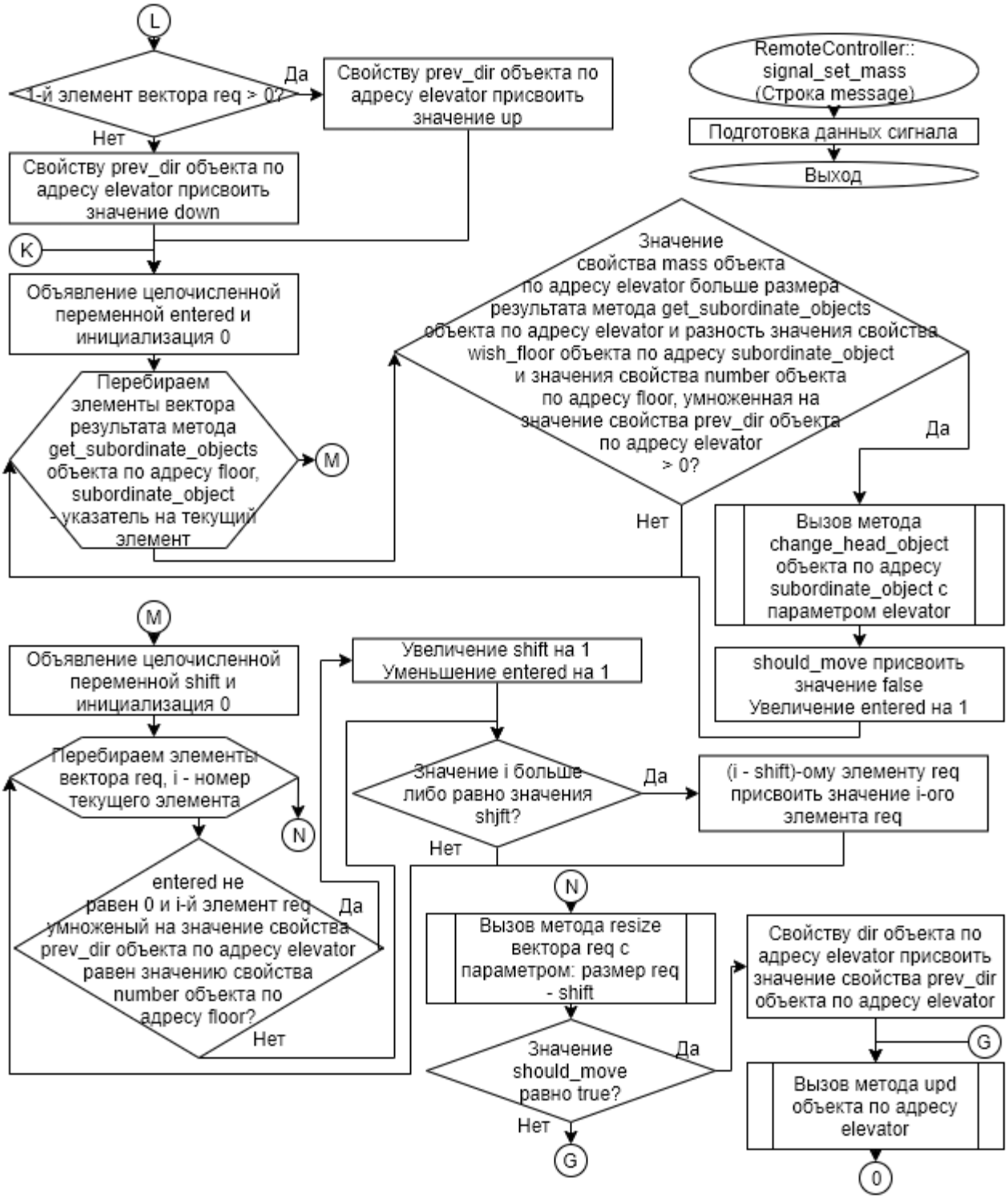

Рисунок 9 – Блок-схема алгоритма

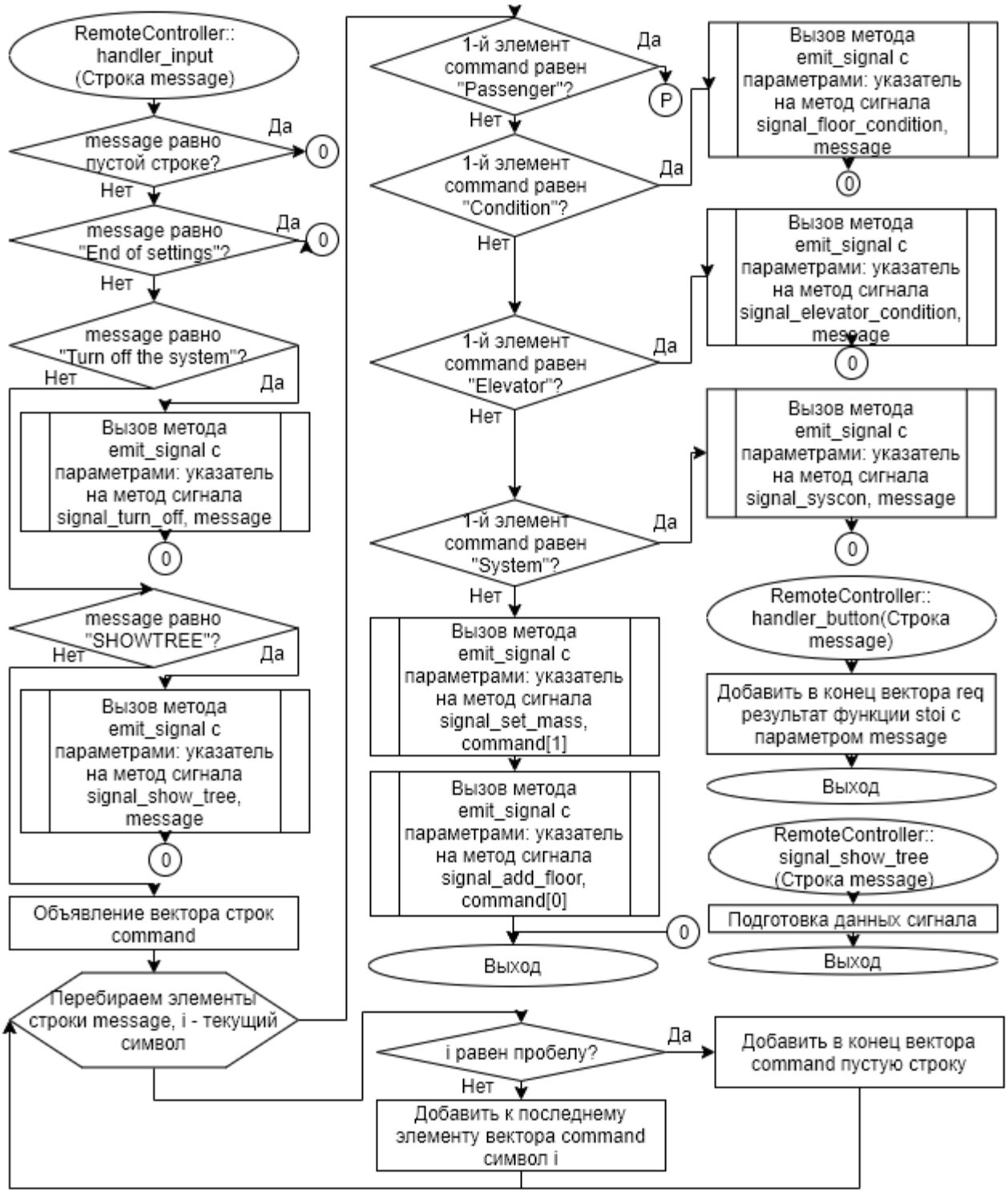

Рисунок 10 – Блок-схема алгоритма

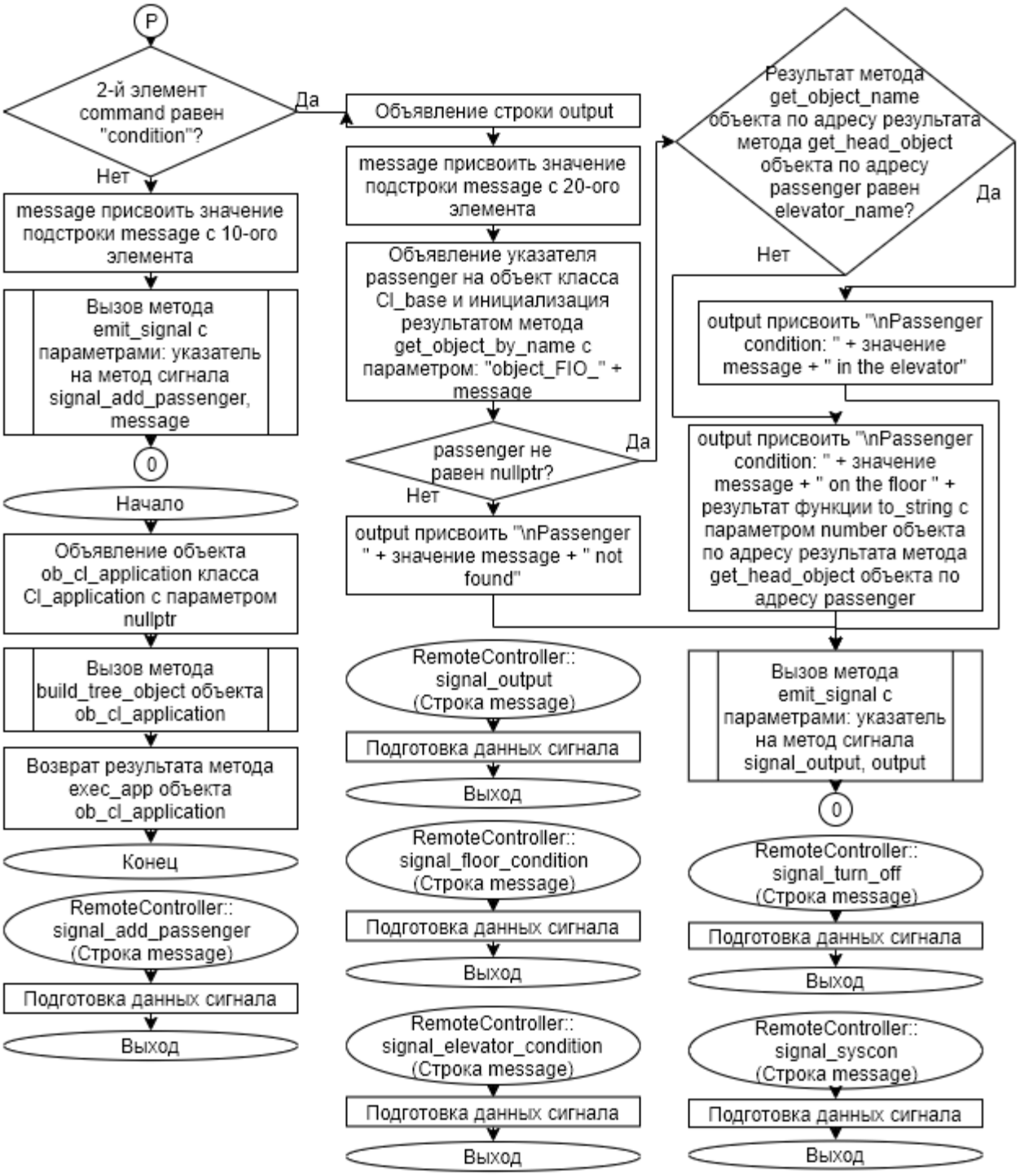

Рисунок 11 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл Cl_application.cpp

Листинг 1 – Cl_application.cpp

```
#include "Cl_application.h"
#include "Elevator.h"
#include "Floor.h"
#include "Input.h"
#include "Output.h"
#include "Passenger.h"
#include "RemoteController.h"


Cl_application::Cl_application(Cl_base*         p_head_object,         string
s_name_object):Cl_base(p_head_object, s_name_object)
{
   change_object_name("object_system");//изменение имени объекта
   tik = 1;//утановка первого такта
}

void Cl_application::build_tree_objects()
{
   RemoteController*  remote_control  =  new  RemoteController(this);//объект
пульта управления
   Elevator* elevator = new Elevator(remote_control);//объект лифта
   Input* input = new Input(this);//объект ввода
   Output* output = new Output(this);//объект вывода
   activate();//активация объектов
   set_connect(SIGNAL_D(Cl_application::signal_input),                 input,
HANDLER_D(Input::handler_input));//установка соединения
   input    ->    set_connect(SIGNAL_D(Input::signal_input),    remote_control,
HANDLER_D(RemoteController::handler_input));//установка соединения
   remote_control -> set_connect(SIGNAL_D(RemoteController::signal_set_mass),
elevator, HANDLER_D(Elevator::handler_receive_mass));//установка соединения
   remote_control                                                          ->
set_connect(SIGNAL_D(RemoteController::signal_add_floor),               this,
HANDLER_D(Cl_application::handler_add_floor));//установка соединения
   remote_control  ->  set_connect(SIGNAL_D(RemoteController::signal_syscon),
this, HANDLER_D(Cl_application::handler_syscon));//установка соединения
   remote_control  ->  set_connect(SIGNAL_D(RemoteController::signal_syscon),
elevator, HANDLER_D(Elevator::handler_syscon));//установка соединения
   string message;//строка сообщения
   emit_signal(SIGNAL_D(Cl_application::signal_input),      message);//выдача
```

```
сигнала ввода
   emit_signal(SIGNAL_D(Cl_application::signal_input),      message);//выдача
сигнала ввода
   remote_control -> set_connect(SIGNAL_D(RemoteController::signal_turn_off),
this, HANDLER_D(Cl_application::handler_turn_off));//установка соединения
   remote_control  ->  set_connect(SIGNAL_D(RemoteController::signal_output),
output, HANDLER_D(Output::handler_output));//установка соединения
   set_connect(SIGNAL_D(Cl_application::signal_output),               output,
HANDLER_D(Output::handler_output));//установка соединения
   set_connect(SIGNAL_D(Cl_application::signal_upd),          remote_control,
HANDLER_D(RemoteController::handler_upd));//установка соединения
   remote_control                                                          ->
set_connect(SIGNAL_D(RemoteController::signal_elevator_condition),  elevator,
HANDLER_D(Elevator::handler_condition));//установка соединения
   elevator    ->    set_connect(SIGNAL_D(Elevator::signal_output),    output,
HANDLER_D(Output::handler_output));//установка соединения
   remote_control                                                          ->
set_connect(SIGNAL_D(RemoteController::signal_show_tree),               this,
HANDLER_D(Cl_application::handler_show_tree));//установка соединения
}

int Cl_application::exec_app()
{
   string message = "Ready to work";//строка сообщения
   emit_signal(SIGNAL_D(Cl_application::signal_output),     message);//выдача
сигнала вывода
   run = true;//разрешение на запуск системы
   while(true) {
      emit_signal(SIGNAL_D(Cl_application::signal_input),   message);//выдача
сигнала ввода
      if (run) {//система запущена?
         string text = "";//пустая строка
         emit_signal(SIGNAL_D(Cl_application::signal_upd),     text);//выдача
сигнала обновления состояния
      }
      else {
         break;
      }
      tik++;//следующий такт
   }
   return 0;
}

void Cl_application::handler_add_floor(string message) {
   int num = stoi(message);//колличество этажей
   Cl_base*                          remote_control                        =
get_sub_object_by_name(remote_controller_name);//объект пульта управления
   Cl_base* output = get_sub_object_by_name("object_display_screen");//объект
вывода
   for (int i = 0; i < num; i++) {//перебор до num
      Floor* f = new Floor(this, i + 1);//добавление нового этажа
      f -> activate();//активация объектов
      if (i) {//этаж не первый?
         f         ->         set_connect(SIGNAL_D(Floor::signal_button_down),
remote_control,      HANDLER_D(RemoteController::handler_button));//установка
```

```
соединения
      }
      if (i < num - 1) {//этаж не последний?
         f -> set_connect(SIGNAL_D(Floor::signal_button_up), remote_control,
HANDLER_D(RemoteController::handler_button));//установка соединения
      }
      remote_control                                                    ->
set_connect(SIGNAL_D(RemoteController::signal_add_passenger),             f,
HANDLER_D(Floor::handler_add_passenger));//установка соединения
      remote_control                                                    ->
set_connect(SIGNAL_D(RemoteController::signal_floor_condition),           f,
HANDLER_D(Floor::handler_condition));//установка соединения
      remote_control                                                    ->
set_connect(SIGNAL_D(RemoteController::signal_syscon),                    f,
HANDLER_D(Floor::handler_syscon));//установка соединения
      f      ->      set_connect(SIGNAL_D(Floor::signal_output),      output,
HANDLER_D(Output::handler_output));//установка соединения
   }
}

void Cl_application::handler_turn_off(string message) {
   run = false;//остановка работы программы
   message = "\n" + message;//перенос строки
   emit_signal(SIGNAL_D(Cl_application::signal_output),     message);//выдача
сигнала вывода
   return;
}

void Cl_application::handler_syscon(string message) {
   message = "\n" + to_string(tik) + ": ";//строка вывода состояния системы
   emit_signal(SIGNAL_D(Cl_application::signal_output),     message);//выдача
сигнала вывода
}

void Cl_application::handler_show_tree(string message) {
   run = false;//остановка работы программы
   message = "\n";//перенос строки
   emit_signal(SIGNAL_D(Cl_application::signal_output),     message);//выдача
сигнала на вывод
   show_object_tree_full();//вывод иерархии объектов
}

void Cl_application::signal_input(string& message){}

void Cl_application::signal_output(string& message){}

void Cl_application::signal_upd(string& message){}
```

5.2 Файл Cl_application.h

Листинг 2 – Cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "Cl_base.h"

class Cl_application: public Cl_base//наследование класса
{
public:
   Cl_application(Cl_base*          p_head_object,          string          =
"Base_object");//параметризированный конструктор
   void build_tree_objects();//метод построения дерева
   int exec_app();//метод запуска системы

   void  handler_add_floor(string);//Метод  обработчика  сигнала  добавления
этажа
   void  handler_turn_off(string);//Метод  обработчика  сигнала  завершения
работы
   void handler_syscon(string);//Метод обработчика сигнала вывода состояния
системы
   void handler_show_tree(string);//Метод обработчика сигнала вывода иерархии
объектов
   void signal_input(string&);//Метод сигнала ввода
   void signal_output(string&);//Метод сигнала вывода
   void signal_upd(string&);//Метод сигнала обновления состояния системы

private:
   bool run;//флаг на выполнение программы
   int tik;//номер такта
};

#endif
```

5.3 Файл Cl_base.cpp

Листинг 3 – Cl_base.cpp

```
#include "Cl_base.h"

Cl_base::Cl_base(Cl_base* p_head_object, string s_object_name)
{
   this   ->   p_head_object   =   p_head_object;//присвоение   указателя   на
родительский объект
   this -> s_object_name = s_object_name;//присвоение имени объекта
   if (p_head_object) {//есть родительский объект?
      p_head_object   ->   subordinate_objects.push_back(this);//добавить   в
```

```
производные объекты
   }
}

bool Cl_base::change_object_name(string s_object_name)
{
   if (s_object_name.empty()) {//пустая строка?
      return false;
   }
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
      if (subordinate_object -> get_object_name() == s_object_name) {//если
имя равно искомому
         return false;
      }
   }
   this -> s_object_name = s_object_name;//сменить имя
   return true;
}

string Cl_base::get_object_name()
{
   return s_object_name;//вернуть имя объекта
}

Cl_base* Cl_base::get_head_object()
{
   return p_head_object;//вернуть указатель на родительский объект
}

void Cl_base::show_object_tree()
{
   cout << endl;
   Cl_base* head_object = p_head_object;//указатель на головной объект
   while (head_object != nullptr) {//существует головной объект?
      cout << "    ";//отступ
      head_object = head_object -> p_head_object;//обновление головного
объекта
   }
   cout << s_object_name;//вывод имени объекта
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
      subordinate_object -> show_object_tree();//уход в рекурсию
   }
}

Cl_base* Cl_base::get_sub_object_by_name(string s_object_name)
{
   if (!s_object_name.empty()) {//строка не пустая?
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
         if (subordinate_object -> get_object_name() == s_object_name) {//имя
равно искомому?
            return subordinate_object;//вернуть указатель на подчиненный
объект
```

```
            }
         }
      }
      return nullptr;
   }

   Cl_base* Cl_base::get_branch_object_by_name(string s_object_name)
   {
      if (this -> s_object_name == s_object_name) {//строка совпадает с именем
   объекта?
         return this;//вернуть объект
      }
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
   подчиненных объеков
         if (subordinate_object -> get_object_name() == s_object_name) {//строка
   совпадает с именем объекта?
            return subordinate_object;//вернуть объект
         }
      }
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
   подчиненных объектов
         if (subordinate_object -> get_branch_object_by_name(s_object_name))
   {//есть в ветви такое имя?
            return                    subordinate_object                    ->
   get_branch_object_by_name(s_object_name);//вернуть объект, если есть
         }
      }
      return nullptr;
   }

   Cl_base* Cl_base::get_object_by_name(string s_object_name)
   {
      Cl_base* base = this;//указатель на текущий объект
      while (true) {
         if (base -> get_head_object()) {//существует головной объект?
            base = base -> get_head_object();//обновить текущий объект
         }
         else {
            break;
         }
      }
      if (base -> get_branch_object_by_name(s_object_name)) {//есть в дереве
   такое имя?
         return   base   ->   get_branch_object_by_name(s_object_name);//вернуть
   объект, если есть
      }
      return nullptr;
   }

   void Cl_base::show_object_tree_full()
   {
      Cl_base* head_object = p_head_object;//указатель на головной объект
      while (head_object != nullptr) {//головной объект существует?
         cout << "    ";//отступ
         head_object  =  head_object  ->  p_head_object;//обновление  головного
```

```
объекта
   }
   cout << s_object_name;//вывод имени объекта
   if (object_state != 0) {//вывод состояния
      cout << " is ready" << endl;
   }
   else {
      cout << " is not ready" << endl;
   }
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
      subordinate_object -> show_object_tree_full();//уход в рекурсию
   }
}

void Cl_base::change_object_state(int object_state)
{
   if (object_state != 0) {//состояние отлично от 0?
      Cl_base* head_object = p_head_object;//указатель на головной объект
      bool f = true;//объявление флага
      while (head_object != nullptr) {//головной объект существует?
         if (head_object -> object_state == 0) {//состояние головного объекта
0?
            f = false;
            break;
         }
         head_object = head_object -> p_head_object;//обновление головного
объекта
      }
      if (f) {
         this -> object_state = object_state;//обновление состояния объекта
      }
   }
   else {
      this -> object_state = 0;//обнуление состояния объекта
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
         subordinate_object -> change_object_state(0);//обнуление
      }
   }
}

bool Cl_base::change_head_object(Cl_base* p_head_object)
{
   if (!p_head_object || !get_head_object()) {//Создается новый корневой
объект или переопределяется корневой объект?
      return false;
   }
   if (p_head_object -> get_sub_object_by_name(this -> get_object_name()) !=
nullptr) {//При переопределеии появляется дубль?
      return false;
   }
   if (this -> get_branch_object_by_name(p_head_object -> s_object_name) ==
p_head_object) {//Новый головной объектв ветви текущего объекта?
      return false;
```

```
   }
   Cl_base* head_object = get_head_object();//указатель на головной объект
   int k = 0;//переменная счетчик
   for (Cl_base* subordinate_object : head_object -> subordinate_objects)
{//для всех подчиненных объектов
      if (subordinate_object == this) {//имя объекта равняетя искомому?
         //head_object                                                    ->
subordinate_objects.erase(subordinate_objects.begin()     +     k);//удаление
текущего объекта из списка подчиненных
         swap(head_object    ->    subordinate_objects[k],    head_object    ->
subordinate_objects.back());//перемещение текущего элемента в конец списка
         head_object  ->  subordinate_objects.pop_back();//удаление  текущего
объекта из списка подчиненных
         break;
      }
      else {
         k++;//увеличение счетчика
      }
   }
   this  ->  p_head_object  =  p_head_object;//Переопределение  указателя  на
головной объект
   this -> p_head_object -> subordinate_objects.push_back(this);//Добавление
объекта в список подчиненных
   return true;
}

void Cl_base::delete_sub_object_by_name(string s_object_name)
{
   if (s_object_name.empty()) {//Строка пустая?
      return;
   }
   int k = 0;//переменная счетчик
   for  (Cl_base*  subordinate_object  :  subordinate_objects)  {//для  всех
подчиненных объектов
      if  (subordinate_object  ->  get_object_name()  ==  s_object_name)  {//имя
объекта равняетя искомому?
         subordinate_objects.erase(subordinate_objects.begin()              +
k);//удаление текущего объекта из списка подчиненных
         delete subordinate_object;//очистка памяти
         break;
      }
      else {
         k++;//увеличение счетчика
      }
   }
}

Cl_base* Cl_base::get_object_by_coordinate(string coordinate)
{
   if (!coordinate.empty()) {//координата не пустая?
      Cl_base* base = this;//Указатель на текущий объект
      if (coordinate[0] == '.') {//Кооринатa начинается с точки?
         if (coordinate.length() == 1) {//Длина равна 1?
            return this;//Вернуь текущий объект
         }
         return     get_branch_object_by_name(coordinate.substr(1));//Вернуть
```

```
объект найденный по имени на ветке
      }
      if (coordinate[0] == '/') {//Координата начинается со слеша?
         while (true) {
            if (base -> get_head_object()) {//существует головной объект?
               base = base -> get_head_object();//обновить текущий объект
            }
            else {
               break;
            }
         }
         if (coordinate.length() == 1) {//Длина равна 1?
            return base;//Вернуть корневой объект
         }
         if (coordinate[1] == '/') {//Второй элемент координаты равен слеш?
            return                              base                              ->
get_sub_object_by_name(coordinate.substr(2));//Вернуть    объект    найденный
среди подчиненных по имени
         }
         coordinate    =    coordinate.substr(1);//Отрезать    первый    элемент
координаты
      }
      vector<string> names;//массив для записи имен объектов в координате
      while (true) {
         if (coordinate.find('/') != string::npos) {//В координате есть слеш?
            names.push_back(coordinate.substr(0,
coordinate.find('/')));//добавить в массив имя объекта до слеша
            coordinate     =     coordinate.substr(coordinate.find('/')     +
1);//отрезать первое имя и слеш
         }
         else {
            names.push_back(coordinate);//добавить в массив имя объекта
            break;
         }
      }
      for (string name : names) {//для всех элементов массива
         base  =  base  ->  get_sub_object_by_name(name);//обновить  текущий
объект
         if (!base) {//Нет подчиненного с таким именем?
            break;
         }
      }
      return base;//вернуть текущий объект
   }
   return nullptr;
}

void    Cl_base::set_connect(TYPE_SIGNAL    p_signal,    Cl_base*    p_target,
TYPE_HANDLER p_handler)
{
   for (connection* value : connects) {//для всех элементов массива
      if (value -> p_signal == p_signal && value -> p_target == p_target &&
value -> p_handler == p_handler) {//существует уже такая связь?
         return;
      }
```

```
        }
        connection* value = new connection();//новый элемент
        value -> p_signal = p_signal;//присвоение значений свойств элемента
        value -> p_target = p_target;//присвоение значений свойств элемента
        value -> p_handler = p_handler;//присвоение значений свойств элемента
        connects.push_back(value);//добавление в массив связей

    }

    void   Cl_base::delete_connect(TYPE_SIGNAL   p_signal,   Cl_base*   p_target,
    TYPE_HANDLER p_handler)
    {
        bool f = false;//флаг
        int index = 0;//индекс
        connection* del;//указатель на удаляемый элемент
        for (connection* value : connects) {//для всех элементов массива
            if (value -> p_signal == p_signal && value -> p_target == p_target &&
    value -> p_handler == p_handler) {//существует такая связь?
                f = true;//смена флага
                del = value;//запоминаем элемент
                break;
            }
            else {
                index++;//увеличение индекса
            }
        }
        if (f) {//true?
            connects.erase(connects.begin() + index);//удаление из списка
            delete del;//очистка памяти
        }
    }

    void Cl_base::emit_signal(TYPE_SIGNAL p_signal, string& message)
    {
        if (object_state == 0) {//объект не активирован?
            return;
        }
        TYPE_HANDLER p_handler;//указатель на метод обраюотчика
        Cl_base* p_target;//указатель на целевой объект
        (this ->* p_signal)(message);//вызов метода сигнала
        for (connection* value : connects) {//для всех элементов массива
            if (value -> p_signal == p_signal) {//нужный нам сигнал?
                p_target = value -> p_target;//извлечение указател на целевой объект
                p_handler  =  value  ->  p_handler;//извлечение  указателя  на  метод
    обработчика
                if (p_target -> object_state != 0) {//объект активирован?
                    (p_target ->* p_handler)(message);//вызов метода обработчика
                }
            }
        }
    }

    string Cl_base::get_absolute_coordinate()
    {
        Cl_base* object = this;//указаеь на текущий объект
```

```
   string absolute = "";//строка координаты
   while (object -> get_head_object())//пока есть головной объект
   {
      absolute = "/" + object -> get_object_name() + absolute;//добавлене
имени объекта в координату
      object = object -> get_head_object();//обновление объекта
   }
   if (absolute.empty()) {//строка пустая?
      return "/";
   }
   return absolute;
}

void Cl_base::activate()
{
   change_object_state(1);//смена состояния
   for (Cl_base* subordinate_object : subordinate_objects){//для всех
подчиненных
      subordinate_object -> activate();//уход в рекурсию
   }
}

vector <Cl_base*> Cl_base::get_subordinate_objects()
{
   return subordinate_objects;//вернуть вектор подчиненных объектов
}
```

5.4 Файл Cl_base.h

Листинг 4 – Cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <vector>
#include <string>
#define SIGNAL_D(signal_f) (TYPE_SIGNAL)(&signal_f)
#define HANDLER_D(handler_f) (TYPE_HANDLER)(&handler_f)

using namespace std;

class Cl_base;
typedef void(Cl_base::*TYPE_SIGNAL)(string&);//тип сигнала
typedef void(Cl_base::*TYPE_HANDLER)(string);//тип обработчика
struct connection//структура связи
{
   TYPE_SIGNAL p_signal;//указатель на метод сигнала
   Cl_base* p_target;//указатель на целевой объект
   TYPE_HANDLER p_handler;//указатель на метод обработчика
};
```

```
class Cl_base//наименование класса
{
public:
   Cl_base     (Cl_base*     p_head_object,     string     s_object_name     =
"Base_object");//параметризированный конструктор
   bool change_object_name(string);//метод изменения имени
   string get_object_name();//метод получения имени
   Cl_base*  get_head_object();//метод  получения  указателя  на  родительский
объект
   Cl_base*   get_sub_object_by_name(string);//метод   поиска   подчиненного
объекта по имени

   Cl_base* get_branch_object_by_name(string);//метод поиска объекта на ветке
по имени
   Cl_base* get_object_by_name(string);//метод поиска объекта по имени
   void show_object_tree();//метод вывода дерева объектов
   void show_object_tree_full();//метод вывода дерева объектов и состояния
   void change_object_state(int);//метод установки состояния

   bool   change_head_object(Cl_base*);//метод   переопределения   головного
объекта
   void   delete_sub_object_by_name(string);//метод   удаления   подчиненного
объекта по наименованию
   Cl_base*  get_object_by_coordinate(string);//метод  получения  указателя  на
объект по его координате

   void  set_connect(TYPE_SIGNAL,  Cl_base*,  TYPE_HANDLER);//метод  установки
связи
   void delete_connect(TYPE_SIGNAL, Cl_base*, TYPE_HANDLER);//метод удаления
связи
   void emit_signal(TYPE_SIGNAL, string&);//метод выдачи синала
   string get_absolute_coordinate();//метод получения абсолютной координаты
   int object_class = 1;//класс объекта

   void activate();//метод приведения объектов в состояние готовности
   vector  <Cl_base*>  get_subordinate_objects();//метод  получения  вектора
указателей на подчиненные объекты

private:
   int object_state = 0;//состояние объекта
   string s_object_name;//имя объекта
   Cl_base* p_head_object;//указатель на родительский объект
   vector <Cl_base*> subordinate_objects;//подчиненные объекты
   vector <connection*> connects;//установленные связи
};


#endif
```

5.5 Файл Elevator.cpp

Листинг 5 – Elevator.cpp

```
#include "Elevator.h"
#include "Passenger.h"

Elevator::Elevator(Cl_base*    p_head_object)    :    Cl_base(p_head_object,
elevator_name)
{
   dir = stop;//остановка движения
   floor = 1;//первый этаж
   prev_dir = up;//предыдущее перемещение вверх
}

void Elevator::signal_output(string &message){}

void Elevator::handler_condition(string message)
{
   string  output  =  "\nElevator  condition:  "  +  to_string(floor)  +  "  "  +
to_string(get_subordinate_objects().size())   +   "   ";//строка   для   вывода
состояния лифта
   if (dir != Elevator::stop) {//лифт не стоит?
      output += "direction ";//добавить в строку вывода direction
      if (dir == Elevator::up) {//лифт движется вверх?
         output += "up";//добавить в строку вывода up
      }
      else {
         output += "down";//добавить в строку вывода down
      }
   }
   else output += "stop";//добавить в строку вывода stop
   emit_signal(SIGNAL_D(Elevator::signal_output),  output);//выдать  сигнал  на
вывод
}

void Elevator::handler_receive_mass(string message)
{
   mass = stoi(message);//обновление вместимости лифта
}

void Elevator::handler_syscon(string message)
{
   message    =    "Elevator:    "    +    to_string(floor)    +    "    "    +
to_string(get_subordinate_objects().size())  +  "    Floors:";//строка  для
вывода состояния лифта в состоянии системы
   emit_signal(SIGNAL_D(Elevator::signal_output), message);//выдать сигнал на
вывод
}

void Elevator::upd()
{
   floor += dir;//смена этажа по ходу движения
}
```

5.6 Файл Elevator.h

Листинг 6 – Elevator.h

```
#ifndef __ELEVATOR__H
#define __ELEVATOR__H
#include "Cl_base.h"

const string elevator_name = "object_elevator";

class Elevator : public Cl_base
{
public:
   enum directions{up = 1, down = -1, stop = 0} dir, prev_dir;//переменные
отслеживания движения лифта
   int mass, floor;//вместимость лифта и текущий этаж
   Elevator (Cl_base* p_head_object);//Параметризированный конструктор
   void signal_output(string&);//Метод сигнала вывода
   void   handler_condition(string);//Метод   обработчика   сигнала   вывода
состояния лифта
   void  handler_receive_mass(string);//Метод  обработчика  сигнала  изменения
вместимости лифта
   void handler_syscon(string);//Метод обработчика сигнала вывода состояния
системы
   void upd();//Метод обновления состояния
};

#endif
```

5.7 Файл Floor.cpp

Листинг 7 – Floor.cpp

```
#include "Floor.h"
#include "Passenger.h"

Floor::Floor(Cl_base*  p_head_object,  int  number)  :  Cl_base(p_head_object,
floor_name + to_string(number))
{
   this -> number = number;//запонминание номера этажа
}

void Floor::signal_button_up(string &message)
{
   message = to_string(number);//обновление message
```

```
}

void Floor::signal_button_down(string &message)
{
   message = to_string(number * (-1));//обновление message
}

void Floor::signal_output(string &message){}

void Floor::handler_add_passenger(string message)
{
   int i = 0, num;
   while (message[i] != ' ') {//поиск пробела в строке
      i++;
   }
   num = stoi(message.substr(0, i));//извлечение номера этажа
   if (num != number) {//извлеченный этаж отличается от текущего?
      return;
   }
   i++;
   int j = i, wish_floor;
   while (message[j] != ' ') {//поиск пробела в строке
      j++;
   }
   wish_floor = stoi(message.substr(i, j - i));//извлечение целевого этажа
   string name = message.substr(j + 1);//извлечение имени пассажира
   Passenger* p = new Passenger(this, name, wish_floor);//создание нового
пассажтра
   p -> activate();//активация объектов
   if (wish_floor > number) {//целевой этаж больше текущего?
      emit_signal(SIGNAL_D(Floor::signal_button_up),        message);//выдача
сигнала кнопки вверх
   }
   else {
      emit_signal(SIGNAL_D(Floor::signal_button_down),      message);//выдача
сигнала кнопки вниз
   }
}

void Floor::handler_condition(string message)
{
   if (stoi(message) != number){//не текущий этаж?
      return;
   }
   string up = "", down = "";//строки имен пассажиров на этаже
   for (auto subordinate_object : get_subordinate_objects()) {//для всех
подчиненных объектав
      if (((Passenger*) subordinate_object) -> wish_floor > number)
{//целевой этаж больше текущего?
         up      +=      "      "      +      subordinate_object      ->
get_object_name().substr(11);//добавить имя в up
      }
      else {
         down     +=     "          "     +     subordinate_object     ->
get_object_name().substr(11);//добавить имя в down
```

```
      }
   }
   message = "\nCondition on the floor " + message + " -up-" + up + "\
nCondition on the floor " + message + " -down-" + down;//строка вывода
   emit_signal(SIGNAL_D(Floor::signal_output),   message);//выдача   сигнала
вывода
}

void Floor::handler_syscon(string message)
{
   int up = 0, down = 0;//колличество пассажиров, желающих поехавть вверх и
вниз
   for (auto subordinate_object : get_subordinate_objects()) {//для всех
подчиненных объектов
      if   (((Passenger*)   subordinate_object)   ->   wish_floor   >   number)
{//целевой этаж больше текущего?
         up++;//увеличение up
      }
      else {
         down++;//увеличение down
      }

   }
   if (up + down > 0) {//есть пассажиры на этаже?
      message = " (" + to_string(number) + ": " + to_string(up) + "-" +
to_string(down) + ")";//строка вывода
      emit_signal(SIGNAL_D(Floor::signal_output),  message);//выдача  сигнала
вывода
   }
}
```

5.8 Файл Floor.h

Листинг 8 – Floor.h

```
#ifndef __FLOOR__H
#define __FLOOR__H
#include "Cl_base.h"

const string floor_name = "object_floor_";

class Floor : public Cl_base
{
public:
   int number;//номер этажа
   Floor(Cl_base*     p_head_object,     int     number);//Параметризированный
конструктор
   void signal_button_up(string&);//Метод сигнала нажатия кнопки "вверх"
   void signal_button_down(string&);//Метод сигнала нажатия кнопки "вниз"
   void signal_output(string&);//Метод сигнала вывода
   void handler_add_passenger(string);//Метод обработчика сигнала добавления
```

```
пассажира
   void   handler_condition(string);//Метод   обработчика   сигнала   вывода
состояния этажа
   void handler_syscon(string);//Метод обработчика сигнала вывода состояния
системы
};

#endif
```

5.9 Файл Input.cpp

Листинг 9 – Input.cpp

```
#include "Input.h"

Input::Input(Cl_base*      p_head_object)      :      Cl_base(p_head_object,
"object_read_command"){}

void Input::signal_input(string& message){}

void Input::handler_input(string message)
{
   getline(cin, message);//ввод строки
   emit_signal(SIGNAL_D(Input::signal_input), message);//выдача сигнала
}
```

5.10 Файл Input.h

Листинг 10 – Input.h

```
#ifndef __INPUT__H
#define __INPUT__H
#include "Cl_base.h"

class Input : public Cl_base
{
public:
   Input(Cl_base* p_head_object);//Параметризированный конструктор
   void signal_input(string&);//Метод сигнала ввода
   void handler_input(string);//Метод обработчика сигнала ввода
};

#endif
```

5.11 Файл main.cpp

Листинг 11 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "Cl_application.h"

int main()
{
   Cl_application ob_cl_application(nullptr);//создание объекта приложения
   ob_cl_application.build_tree_objects();//конструирование системы
   return ob_cl_application.exec_app();//запуск системы
}
```

5.12 Файл Output.cpp

Листинг 12 – Output.cpp

```
#include "Output.h"
#include "Cl_application.h"
#include "Elevator.h"
#include "Floor.h"
#include "Passenger.h"
#include "RemoteController.h"

Output::Output(Cl_base*      p_head_object)      :      Cl_base(p_head_object,
"object_display_screen"){}

void Output::handler_output(string message)
{
   cout << message;//вывод сообщения
}
```

5.13 Файл Output.h

Листинг 13 – Output.h

```
#ifndef __OUTPUT__H
#define __OUTPUT__H
#include "Cl_base.h"

class Output : public Cl_base
```

```
{
public:
   Output(Cl_base* p_head_object);//Параметризированный конструктор
   void handler_output(string);//Метод обработчика сигнала вывода
};

#endif
```

5.14 Файл Passenger.cpp

Листинг 14 – Passenger.cpp

```
#include "Passenger.h"

Passenger::Passenger(Cl_base* p_head_object, string name, int wish_floor) :
Cl_base(p_head_object, "object_FIO_" + name)
{
   this -> wish_floor  = wish_floor;//установка желаемого этажа
}
```

5.15 Файл Passenger.h

Листинг 15 – Passenger.h

```
#ifndef __PASSENGER__H
#define __PASSENGER__H
#include "Cl_base.h"

class Passenger : public Cl_base
{
public:
   int wish_floor;//целевой этаж
   Passenger(Cl_base*       p_head_object,       string       name,       int
wish_floor);//Параметризированный конструктор
};

#endif
```

5.16 Файл RemoteController.cpp

Листинг 16 – RemoteController.cpp

```
#include "RemoteController.h"
#include "Elevator.h"
#include "Passenger.h"
#include "Floor.h"

RemoteController::RemoteController(Cl_base*         p_head_object)         :
Cl_base(p_head_object, remote_controller_name){}

void RemoteController::handler_upd(string message)
{
   Elevator*                              elevator                              =
(Elevator*)get_sub_object_by_name(elevator_name);//объект лифта
   if (!elevator) {
      return;
   }
   Floor*  floor  =  (Floor*)get_object_by_coordinate("/"  +  floor_name  +
to_string(elevator -> floor));//объект этажа
   if (!floor) {
      return;
   }
   if (elevator -> dir != Elevator::stop) {//лифт движется?
      elevator  ->  prev_dir  =  elevator  ->  dir;//обновление  предыдущего
состояния
      for (auto subordinate_object : elevator -> get_subordinate_objects())
{//для всех подчиненных объектов
         if (((Passenger*)subordinate_object) -> wish_floor == elevator ->
floor) {//пассажир едет на этот этаж?
            elevator -> dir = Elevator::stop;//остановка лифта
            break;
         }
      }
      if (elevator -> mass - elevator -> get_subordinate_objects().size() >
0) {//в лифте есть свободное место?
         for (auto subordinate_object : floor -> get_subordinate_objects())
{//для всех подчиненных объектов
            if ((((Passenger*)subordinate_object) -> wish_floor - elevator ->
floor)     *     elevator     ->     dir     >     0     ||     elevator     ->
get_subordinate_objects().empty()) {//пассажир едет в тнаправлении движения
лифта или лифт пустой?
               elevator -> dir = Elevator::stop;//остановка лифта
               break;
            }
         }
      }
   }
   if (elevator -> dir == Elevator::stop) {//лифт стоит?
      bool should_move = true;//разрешение на движение
      for (auto subordinate_object : elevator -> get_subordinate_objects())
{//для всех подчиненных объектов
         if  (((Passenger*)subordinate_object)  ->  wish_floor  ==  floor  ->
```

```
number) {//целевой этаж это текущий?
            should_move = false;//не двигается
            elevator   ->   delete_sub_object_by_name(subordinate_object   ->
get_object_name());//пассажир выходит из лифта
         }
      }
      if (elevator -> get_subordinate_objects().empty()) {//лифт пустой?
         if (req.size() > 0) {//очерель не пуста?
            if (abs(req[0]) == floor -> number) {//лифт на этаже первого в
очереди?
               if (req[0] > 0) {//едет вверх?
                  elevator  ->  prev_dir  =  Elevator::up;//предыдущее  движение
up
               }
               else {
                  elevator -> prev_dir = Elevator::down;//предыдущее движение
down
               }
            }
            else if (abs(req[0]) - floor -> number > 0) {//лифт ниже первого
в очереди?
               elevator -> prev_dir = Elevator::up;//предыдущее движение up
            }
            else {
               elevator  ->  prev_dir  =  Elevator::down;//предыдущее  движение
down
            }
         }
         else {
            elevator -> prev_dir = Elevator::stop;//предыдущее движение stop
         }
      }
      int entered = 0;//число вошедших в лифт
      for  (auto  subordinate_object  :  floor  ->  get_subordinate_objects())
{//для всех подчиненных объектов
         if (elevator -> mass > elevator -> get_subordinate_objects().size()
&&  (((Passenger*)subordinate_object)  ->  wish_floor  -  floor  ->  number)  *
elevator -> prev_dir > 0) {//есть свободное место в лифте и пассажир едет по
напрвлению движения лифта?
            subordinate_object   ->   change_head_object(elevator);//пассажир
заходит в лифт
            should_move = false;//не двигается
            entered++;//увеличение числа вошедших
         }
      }
      int shift = 0;
      for (int i = 0; i < req.size(); i++) {//для всех элементов очереди
         if (entered && req[i] * elevator -> prev_dir == floor -> number)
{//есть вошедшие и текущий элемент очереди зашел?
            shift++;
            entered--;
         }
         if (i >= shift) {
            req[i - shift] = req[i];
         }
```

```
      }
      req.resize(req.size() - shift);//смена размера очереди
      if (should_move) {//должен двигаться?
         elevator -> dir = elevator -> prev_dir;//переопределение направления
движения
      }
   }
   elevator -> upd();//обновление состояния лифта
}

void RemoteController::handler_input(string message)
{
   if (message == "") {//пустая строка?
      return;
   }
   if (message == "End of settings") {//конец настроек?
      return;
   }
   if (message == "Turn off the system") {//завершение работы системы?
      emit_signal(SIGNAL_D(RemoteController::signal_turn_off),
message);//выдача сигнала завершения работы
      return;
   }
   if (message == "SHOWTREE") {//вывод иерархии?
      emit_signal(SIGNAL_D(RemoteController::signal_show_tree),
message);//выдача сигнала вывода иерархии объектов
      return;
   }
   vector<string> command(1, "");//вектор команды
   for (auto i : message) {//перебираем символы строки
      if (i == ' ') {//пробел?
         command.push_back("");//добавить пустую строку
      }
      else {
         command.back().push_back(i);//приписать символ в конец
      }
   }
   if (command[0] == "Passenger") {//команда Pasenger?
      if (command[1] == "condition") {//команда cjndition?
         string output;//строка вывода
         message = message.substr(20);//извлечение имени
         Cl_base*    passenger    =    get_object_by_name("object_FIO_"    +
message);//указатель на объект пассажира
         if (passenger) {//пассажир найден?
            if  (passenger  ->  get_head_object()  ->  get_object_name()  ==
elevator_name) {//пассажир в лифте?
               output  =  "\nPassenger  condition:  "  +  message  +  "  in  the
elevator";//строка состояния пассажира в лифте
            }
            else {
               output = "\nPassenger condition: " + message + " on the floor
" + to_string(((Floor*)passenger -> get_head_object()) -> number);//строка
состояния пассажира на этаже
            }
         }
```

```
         else {
            output = "\nPassenger " + message + " not found";//сообщение:
пассажир не найден
         }
         emit_signal(SIGNAL_D(RemoteController::signal_output),
output);//выдача сигнала вывода
      }
      else {
         message = message.substr(10);//извлечение информации о пассажире
         emit_signal(SIGNAL_D(RemoteController::signal_add_passenger),
message);//выдача сигнала добавления пассажира
      }
   }
   else if (command[0] == "Condition") {//команда Condition?
      emit_signal(SIGNAL_D(RemoteController::signal_floor_condition),
command[4]);//выдача сигнала ввода состояния этажа
   }
   else if (command[0] == "Elevator") {//команда Elevator?
      emit_signal(SIGNAL_D(RemoteController::signal_elevator_condition),
message);//выдача сигнала вывода состояния лифта
   }
   else if (command[0] == "System") {//Команда System?
      emit_signal(SIGNAL_D(RemoteController::signal_syscon),
message);//выдача сигнала вывода состояния системы
   }
   else {
      emit_signal(SIGNAL_D(RemoteController::signal_set_mass),
command[1]);//выдача сигнала установки вместимости лифта
      emit_signal(SIGNAL_D(RemoteController::signal_add_floor),
command[0]);//выдача сигнала создания этажей
   }
}

void RemoteController::handler_button(string message)
{
   req.push_back(stoi(message));//добаление элемента в очередь
}

void RemoteController::signal_add_floor(string& message){}

void RemoteController::signal_set_mass(string& message){}

void RemoteController::signal_turn_off(string& message){}

void RemoteController::signal_add_passenger(string& message){}

void RemoteController::signal_output(string& message){}

void RemoteController::signal_floor_condition(string& message){}

void RemoteController::signal_elevator_condition(string& message){}

void RemoteController::signal_syscon(string& message){}

void RemoteController::signal_show_tree(string& message){}
```

5.17 Файл RemoteController.h

Листинг 17 – RemoteController.h

```
#ifndef __REMOTECONTROLLER__H
#define __REMOTECONTROLLER__H
#include "Cl_base.h"

const string remote_controller_name = "object_remote_control";

class RemoteController : public Cl_base
{
public:
   RemoteController(Cl_base* p_head_object);//Параметризированный конструктор
   void handler_upd(string);//Метод обработчика сигнала обновления состояния
   void handler_input(string);//Метод обработчика сигнала ввода
   void handler_button(string);//Метод обработчика сигнала нажатия на кнопку
вызова
   void signal_add_floor(string&);//Метод сигнала добавления этажа
   void signal_set_mass(string&);//Метод сигнала изменения вместимости лифта
   void signal_turn_off(string&);//Метод сигнала завершения работы системы
   void signal_add_passenger(string&);//Метод сигнала добавления пассажира
   void signal_output(string&);//Метод сигнала вывода
   void  signal_floor_condition(string&);//Метод  сигнала  вывода  состояния
этажа
   void  signal_elevator_condition(string&);//Метод  сигнала  вывода  состояния
лифта
   void signal_syscon(string&);//Метод сигнала вывода состояния системы
   void signal_show_tree(string&);//Метод сигнала вывода иерархии объектов

private:
   vector<int> req;//очередь
};

#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 47.

Таблица 47 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
10 4 End of settings Passenger 1 10 Ivanov I.I. Passenger 7 2 Petrov A.V. Passenger 3 9 Cidorov K.V. System status System status System status Elevator condition System status System status System status System status System status System status System status Passenger condition Petrov A.V. System status System status System status System status Turn off the system	Ready to work 4: Elevator: 3 1 Floors: (3: 1-0) (7: 0-1) 5: Elevator: 3 2 Floors: (7: 0-1) 6: Elevator: 4 2 Floors: (7: 0-1) Elevator condition: 5 2 direction up 8: Elevator: 6 2 Floors: (7: 0-1) 9: Elevator: 7 2 Floors: (7: 0-1) 10: Elevator: 8 2 Floors: (7: 0-1) 11: Elevator: 9 2 Floors: (7: 0-1) 12: Elevator: 9 1 Floors: (7: 0-1) 13: Elevator: 10 1 Floors: (7: 0-1) 14: Elevator: 10 0 Floors: (7: 0-1) 15: Elevator: 9 0 Floors: (7: 0-1) 16: Elevator: 8 0 Floors: (7: 0-1) Passenger condition: Petrov A.V. on the floor 7 18: Elevator: 7 1 Floors: 19: Elevator: 6 1 Floors: 20: Elevator: 5 1 Floors: 21: Elevator: 4 1 Floors: Turn off the system	Ready to work 4: Elevator: 3 1 Floors: (3: 1-0) (7: 0-1) 5: Elevator: 3 2 Floors: (7: 0-1) 6: Elevator: 4 2 Floors: (7: 0-1) Elevator condition: 5 2 direction up 8: Elevator: 6 2 Floors: (7: 0-1) 9: Elevator: 7 2 Floors: (7: 0-1) 10: Elevator: 8 2 Floors: (7: 0-1) 11: Elevator: 9 2 Floors: (7: 0-1) 12: Elevator: 9 1 Floors: (7: 0-1) 13: Elevator: 10 1 Floors: (7: 0-1) 14: Elevator: 10 0 Floors: (7: 0-1) 15: Elevator: 9 0 Floors: (7: 0-1) 16: Elevator: 8 0 Floors: (7: 0-1) Passenger condition: Petrov A.V. on the floor 7 18: Elevator: 7 1 Floors: 19: Elevator: 6 1 Floors: 20: Elevator: 5 1 Floors: 21: Elevator: 4 1 Floors: Turn off the system
10 4 End of settings Passenger 1 10 Ivanov I.I.	Ready to work object_system is ready object_remote_co	Ready to work object_system is ready object_remote_co

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
Passenger 7 2 Petrov A.V. Passenger 3 9 Cidorov K.V. SHOWTREE System status System status System status Elevator condition System status System status System status System status System status System status Passenger condition Petrov A.V. System status System status System status System status Turn off the system	ntrol is ready object_elevator is ready object_FIO_Ivanov I.I. is ready object_read_command is ready object_display_screen is ready object_floor_1 is ready object_floor_2 is ready object_floor_3 is ready object_FIO_Cidorov K.V. is ready object_floor_4 is ready object_floor_5 is ready object_floor_6 is ready object_floor_7 is ready object_FIO_Petrov A.V. is ready object_floor_8 is ready object_floor_9 is ready object_floor_10 is ready	ntrol is ready object_elevator is ready object_FIO_Ivanov I.I. is ready object_read_command is ready object_display_screen is ready object_floor_1 is ready object_floor_2 is ready object_floor_3 is ready object_FIO_Cidorov K.V. is ready object_floor_4 is ready object_floor_5 is ready object_floor_6 is ready object_floor_7 is ready object_FIO_Petrov A.V. is ready object_floor_8 is ready object_floor_9 is ready object_floor_10 is ready
10 4 End of settings SHOWTREE Passenger 1 10 Ivanov I.I. Passenger 7 2 Petrov A.V. Passenger 3 9 Cidorov K.V. System status System status System status Elevator condition System status System status System status System status	Ready to work object_system is ready object_remote_control is ready object_elevator is ready object_read_command is ready object_display_screen is ready object_floor_1 is ready object_floor_2 is ready object_floor_3 is ready object_floor_4	Ready to work object_system is ready object_remote_control is ready object_elevator is ready object_read_command is ready object_display_screen is ready object_floor_1 is ready object_floor_2 is ready object_floor_3 is ready object_floor_4

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
System status System status System status System status System status Passenger condition Petrov A.V. System status System status System status System status Turn off the system	is ready object_floor_5 is ready object_floor_6 is ready object_floor_7 is ready object_floor_8 is ready object_floor_9 is ready object_floor_10 is ready	is ready object_floor_5 is ready object_floor_6 is ready object_floor_7 is ready object_floor_8 is ready object_floor_9 is ready object_floor_10 is ready
10 4 End of settings Passenger 1 10 Ivanov I.I. Passenger 7 2 Petrov A.V. Passenger 3 9 Cidorov K.V. System status System status System status Elevator condition System status Condition on the floor 7 System status Passenger condition Pavlov N.S. System status Passenger condition Petrov A.V. System status System status System status System status SHOWTREE Turn off the system	Ready to work 4: Elevator: 3 1 Floors: (3: 1-0) (7: 0-1) 5: Elevator: 3 2 Floors: (7: 0-1) 6: Elevator: 4 2 Floors: (7: 0-1) Elevator condition: 5 2 direction up 8: Elevator: 6 2 Floors: (7: 0-1) Condition on the floor 7 -up- Condition on the floor 7 -down- Petrov A.V. 10: Elevator: 8 2 Floors: (7: 0-1) Passenger Pavlov N.S. not found 12: Elevator: 9 1 Floors: (7: 0-1) Passenger condition: Petrov A.V. on the floor 7 14: Elevator: 10 0 Floors: (7: 0-1) 15: Elevator: 9 0 Floors: (7: 0-1) 16: Elevator: 8 0 Floors: (7: 0-1) 17: Elevator: 7 0 Floors: (7: 0-1) object_system is ready object_remote_control is ready object_eleva	Ready to work 4: Elevator: 3 1 Floors: (3: 1-0) (7: 0-1) 5: Elevator: 3 2 Floors: (7: 0-1) 6: Elevator: 4 2 Floors: (7: 0-1) Elevator condition: 5 2 direction up 8: Elevator: 6 2 Floors: (7: 0-1) Condition on the floor 7 -up- Condition on the floor 7 -down- Petrov A.V. 10: Elevator: 8 2 Floors: (7: 0-1) Passenger Pavlov N.S. not found 12: Elevator: 9 1 Floors: (7: 0-1) Passenger condition: Petrov A.V. on the floor 7 14: Elevator: 10 0 Floors: (7: 0-1) 15: Elevator: 9 0 Floors: (7: 0-1) 16: Elevator: 8 0 Floors: (7: 0-1) 17: Elevator: 7 0 Floors: (7: 0-1) object_system is ready object_remote_control is ready object_eleva

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	tor is ready object_FIO_Petrov A.V. is ready object_read_command is ready object_display_screen is ready object_floor_1 is ready object_floor_2 is ready object_floor_3 is ready object_floor_4 is ready object_floor_5 is ready object_floor_6 is ready object_floor_7 is ready object_floor_8 is ready object_floor_9 is ready object_floor_10 is ready	tor is ready object_FIO_Petrov A.V. is ready object_read_command is ready object_display_screen is ready object_floor_1 is ready object_floor_2 is ready object_floor_3 is ready object_floor_4 is ready object_floor_5 is ready object_floor_6 is ready object_floor_7 is ready object_floor_8 is ready object_floor_9 is ready object_floor_10 is ready

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были достигнуты все поставленные цели, а именно получены практические навыки построения иерархии объектов и моделирования сложных технических систем, проведено ознакомление с механизмом сигналов и обработчиков как одним из способов организации взаимодействия объектов вне системы.

Для достижения поставленных целей в полном объеме были реализованы задачи, сформулированные в процессе подготовки к выполнению курсовой работы:

- Организовано иерархическое построение объектов нескольких классов
- Реализован механизм взаимодействия объектов с использованием сигналов и обработчиков
- Смоделирована работа лифта с использованием языка программирования C++

Было проведено тестирование, которое показало корректное функционирование разработанной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).