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_18 Моделирование работы телефона
(наименование темы)

Студент группы ИКБО-35-22 Румянцев Дмитрий Дмитриевич
(учебная группа) *(Фамилия Имя Отчество)* *(подпись студента)*

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

Консультант ст.преп. Грач Е.П.
(Должность, звание, ученая степень) *(подпись консультанта)*

Работа представлена к защите «20» мая 2023 г.

Допущен к защите «20» мая 2023 г.

Москва 2023 г.

ОТЗЫВ

на курсовую работу

по дисциплине «Объектно-ориентированное программирование»

Студент **Румянцев Дмитрий Дмитриевич** группа ИКБО-35-22

(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, рисушков и пр.			
4. Полнота выполнения всех пунктов задания			
5. Логичность и системность содержания курсовой работы			
6. Отсутствие фактических грубых ошибок			

Замечаний: _____

Рекомендуемая оценка: _____

_____ доцент Унгер А.Ю.

(Подпись руководителя) *(ФИО руководителя)*

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 2023г.

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

Студент Румянцев Дмитрий Дмитриевич Группа ИКБО-35-22

Тема К_18 Моделирование работы телефона

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 «20» мая 2023 г.

Задание на курсовую работу выдал _____ (Унгер А.Ю.)
Подпись ФИО консультанта
«21» февраля 2023 г.

Задание на курсовую работу получил _____ (Румянцев Д.Д.)
Подпись ФИО исполнителя
«21» февраля 2023 г.

Москва 2023 г.

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].

В рамках изучения курса "Объектно-ориентированное программирование" на языке C++ требовалось выполнение курсовой работы. Курсовая работа состоит из 5 задач: КВ_1; КВ_2; КВ_3; КВ_4; КВ_18. Первые 4 задачи являются общими. Они подготавливают основные классы и методы для последующего решения пятой, заключительной задачи. В качестве пятой задачи была выбрана 18 тема - "Моделирование работы телефона".

Суть пятой задачи 18 темы сводится к разработке систему имитирующей работу телефона, каждый из которых содержит: кнопки для набора телефона; кнопка управления «вызов»; кнопка управления «отказ»; микрофон, динами; экран отображения; телефон имеет состояние «свободен» или «занят». Реализована данная система с помощью 6 классов, один из которых является базовым. Остальные 5 классов имитируют различные элементы такие как: автоматизированной телефонной станции (АТС);пульта управления АТС;множества стационарных телефонов;множества пользователей;экрана отображения информации о функционировании системы. С помощью сигналов, мы имеем возможность взаимодействовать между элементами телефона, что позволяет имитировать его работу.

1 ПОСТАНОВКА ЗАДАЧИ

Дана система связи телефонов, состоящая из следующих элементов:

- автоматизированной телефонной станции (АТС);
- пульта управления АТС;
- множества стационарных телефонов;
- множества пользователей;
- экрана отображения информации о функционировании системы.

Телефон имеет следующую конструкцию:

- кнопки для набора номера;
- кнопка управления «вызов», после набора номера для установки соединения надо нажать на эту кнопку. Телефон автономно проверяет полноту набора, больше или меньше количество цифр номера и соответствующую информацию отображает на экране;
- кнопка управления «отказ»;
- микрофон, динамик;
- экран отображения;
- телефон имеет состояние «свободен» или «занят» (когда идет набор номера, устанавливается соединение или идет разговор).

Правила функционирования системы:

1. Каждый телефон имеет уникальный номер.
2. Пользователь телефона набирает номер абонента и нажимает на кнопку «вызов».
3. Далее запрос поступает на АТС, которая определяет корректность набора. Если номер абонента корректен, то запрос на соединение отрабатывает, иначе пользователю поступает сообщение об ошибочном наборе.
4. АТС определяет занятость абонента.

5. Если абонент свободен, устанавливается связь, иначе сообщает пользователю о занятости абонента.

Надо моделировать работу данной системы. Допускаем, что система функционирует по тактам. Такт соответствует 30 секундам. В рамках одного такта отрабатывает одна команда и связанные пользователи могут проговорить 30 секунд.

Команды системы.

Команда запроса установки связи.

```
Call request «номер отправителя» «номер абонента» «продолжительность разговора»
```

Данная команда моделирует набор номера телефона к абоненту и нажатье на кнопку «вызов». После этого автономно проверяется корректность набора номер абонента (больше или меньше допустимого количества цифр). Далее посылается запрос на установку связи к АТС. АТС проверят возможность установки связи (телефон абонента свободен) и устанавливает связь в начале текущего такта.

Команда вывода списка отправленных вызовов с телефона и полученных звонков.

```
Display phone information «номер телефона»
```

По данной команде выдается построчно список вызовов и звонков, по упорядоченным номерам тактов, их успешность и продолжительность. Команда выдачи состояния системы.

```
Display the system status
```

По данной команде выводиться состояние системы. Информация содержит перечень телефонов, упорядоченных по возрастанию.

Относительно каждого номера телефона: сколько вызовов было отправлено, сколько установлено связей, сколько звонков было принято.

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

действия согласно такту.

Команда завершения работы системы.

```
Turn off the system
```

Построить программу-систему, которая использует объекты:

1. Объект «система». Соответствует АТС.
2. Объект для чтения данных и команд. Объект моделирует работу пользователей телефонов. Считывает данные для первоначальной подготовки и настройки системы. Считывает команды. 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 пульта управления моделирует оперативную работу АТС, для отработки поступивших команд (запросов на соединение). Объект выдает соответствующий сигнал или сигналы. Содержит список номеров телефонов. Определяет корректность номера абонента в тексте запроса. Если номер набран некорректно, то выдает сигнал соответствующим сообщением. Готовит данные для запроса о состоянии системы.
4. Объект, моделирующий телефон. Выдает сигнал, содержащий команду запроса на связь. Входящие и исходящие звонки сохраняет в журнале (в памяти телефона). По команде вывода списка звонков формирует содержание строк, выдает сигналы к устройству вывода.
5. Объект для вывода состояния или результата команды системы на консоль. Текст для вывода объект получает по сигналу от других объектов системы. Каждое присланное сообщение выводиться с новой строки.

Архитектура иерархи объектов.

```
Система АТС (приложение)
    Объект ввода.
    Пульт управления АТС.
        Телефон 1
        Телефон 2
        . . . . .
        Телефон n
    Объект вывода.
```

Написать программу-систему, реализующую следующий алгоритм:

1. Вызов от объекта «система» метода build_tree_objects ( ).
    1.1. Построение исходного дерева иерархии объектов. После создания любой объект перевести в состоянии готовности.
    1.2. Цикл для обработки вводимых данных.
        1.2.1. Выдача сигнала объекту чтения для ввода очередной строки данных (номер телефона). Допускаем, что номера телефонов различны и заданы корректно, состоят из 7 цифр.
        1.2.2. Отработка операции чтения очередной строки входных данных.
        1.2.3. Создание нового объекта. 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 отладочный тест такой командой.

1.1 Описание входных данных

Начиная с первой строки, построчно вводятся номера телефонов. Номер телефона семизначное целое число без знака.

```
«цело число»
```

Ввод номеров телефонов заканчивается вводом текста:

```
End of phones
```

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

Команда запроса вызова.

```
Call request «номер отправителя» «номер абонента» «продолжительность разговора в секундах»
```

Команда вывода информации о телефоне.

```
Display phone information «номер телефона»
```

Команда вывода информации о системе АТС.

```
Display system status information
```

Команда завершения работы системы.

```
Turn off the system
```

Пример ввода:

```
1111111
1111112
1111113
1111114
1111115
1111116
1111117
1111118
End of phones
Display the system status
Call request 1111115 1111117 55
Call request 1111116 1111117 15
Display the system status
Display phone information 1111116
Display phone information 1111115
Display phone information 1111117
Call request 1111115 1111117 55
Display the system status
Turn off the system
```

1.2 Описание выходных данных

После завершения ввода исходных данных выводиться текст:

```
Ready to work
```

После команды запроса вызова могут быть выданы разные сообщения. Если номер абонента набран не из семи цифр:

```
The subscriber's number was dialed incorrectly: «набранный номер»
```

Если поступила команда на вызов, а телефон занят (невозможность звонка, если пользователь разговаривает):

```
The phone is busy, a new call is not possible: «номер телефона»
```

Если абонент не найден:

```
Subscriber «номер абонента» not found
```

Если абонент занят:

```
Subscriber «номер абонента» is busy
```

Если связь установлен:

```
Call from «номер телефона от которого вызов» to «номер абонента», talk: «продолжительность разговора в секундах»
```

После команды вывода информации о телефоне:

```
Phone log: «номер телефона»
```

Далее построчно по возрастанию номеров тактов. Если вызов был удачен и разговор состоялся:

```
«номер такта вызова» Call «номер абонента» «продолжительность разговора в секундах»
```

Если абонент был занят:

```
«номер такта вызова» Call «номер абонента» 0
```

Если был получен звонок:

```
«номер такта звонка» Bell «номер телефона откуда вызов» «продолжительность разговора в секундах»
```

Команда вывода информации о системе АТС. В очередной строке:

```
ATC
```

Далее построчна упорядоченный список телефонов:

```
telephone «номер телефона» «количество вызовов» «количество разговоров» «количество звонков»
```

Пример вывода:

```
Ready to work
ATC
telephone 1111111 0 0 0
telephone 1111112 0 0 0
telephone 1111113 0 0 0
telephone 1111114 0 0 0
telephone 1111115 0 0 0
telephone 1111116 0 0 0
telephone 1111117 0 0 0
telephone 1111118 0 0 0
Call from 1111115 to 1111117, talk: 55
Subscriber 1111117 is busy
ATC
telephone 1111111 0 0 0
telephone 1111112 0 0 0
telephone 1111113 0 0 0
telephone 1111114 0 0 0
telephone 1111115 1 1 0
telephone 1111116 1 0 0
telephone 1111117 0 0 1
telephone 1111118 0 0 0
Phone log: 1111116
```

```
3 Call 1111117 0
Phone log: 1111115
2 Call 1111117 55
Phone log: 1111117
2 Bell 1111115 55
Call from 1111115 to 1111117, talk: 55
ATC
telephone 1111111 0 0 0
telephone 1111112 0 0 0
telephone 1111113 0 0 0
telephone 1111114 0 0 0
telephone 1111115 2 2 0
telephone 1111116 1 0 0
telephone 1111117 0 0 2
telephone 1111118 0 0 0
Turn off the ATM
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи потребуется:

- Объект потокового ввода и вывода cin/cout
- Условные операторы if/else
- Оператор цикла с предуесловием while
- Библиотека <vector>
- Библиотека algorithm
- Библиотека cctype
- Библиотека sstream
- Методы и поля cl_base из предыдущей задачи
- Структура journal - структура данных для представления записей в журнале:
    - takt - целочисленное поле, инициализированное значением 0:
        - Представляет номер такта для записи в журнале;
    - type - строковое поле:
        - Представляет информацию о типа звонка входящий/исходящий;
    - number - строковое поле:
        - Хранит информацию о номере телефона абонента;
    - time_talk - целочисленное поле:
        - Представляет информацию о длительности времени разговора
- Изменение в функциональности классов:
    - Класс cl_base:
        - Поля:
            - Поле, отвечающее за журнал звонков для каждого номера телефона:

        - Наименование: journal_list;
        - Тип: вектор, содержащий элементы типа journal;
        - Модификатор доступа - protected;
    - Поле, отвечающее за количество вызовов:
    - Поле, отвечающее за количество разговоров:
    - Поле, отвечающее за количество звонков:
    - Поле, отвечающее за занятость номера телефона:
    - Поле, отвечающее за номера такта:
    - Статическое поле, отвечающее за номера такта:
    - Поле, отвечающее за вводимую команду:
- Методы:
    - set_count_call:
        - Функционал - используется для установки количества вызовов;
        - Параметры:
            - Целочисленный параметр - count_call;
        - Тип возвращаемого значения: нет;
        - Модификатор доступа - public;
    - set_sum_calls:
        - Функционал - используется для установки количества разговоров;
        - Параметры:
            - Целочисленный параметр - sum_calls;
        - Тип возвращаемого значения: нет;
        - Модификатор доступа - public;
    - set_count_talk:
        - Функционал - используется для установки

количества звонков;

- Параметры:
  - Целочисленный параметр - count_talk;
- Тип возвращаемого значения: нет;
- Модификатор доступа - public;

- set_is_free:
  - Функционал - используется для установки занятости номера телефона;
  - Параметры:
    - Целочисленный параметр - is_free;
  - Тип возвращаемого значения: нет;
  - Модификатор доступа - public;
- get_count_call:
  - Функционал - используется для получения количества вызовов;
  - Параметры: нет;
  - Тип возвращаемого значения: целочисленное;
  - Модификатор доступа - public;
- get_sum_calls:
  - Функционал - используется для получения количества разговоров;
  - Параметры: нет;
  - Тип возвращаемого значения: целочисленное;
  - Модификатор доступа - public;
- get_count_talk:
  - Функционал - используется для получения количества звонков;

        - Параметры: нет;
        - Тип возвращаемого значения: целочисленное ;
        - Модификатор доступа - public;
    - get_is_free:
        - Функционал - используется для получения занятости номера телефона;
        - Параметры: нет;
        - Тип возвращаемого значения: булеове;
        - Модификатор доступа - public;

- Новые классы для решения данной задачи:
- signal_reader_to_all:
    - Функционал - метод сигнала, считывающий строку;
    - Параметры:
        - Ссылка на строковую переменную msg;
    - Тип возвращаемого значения: нет;
    - Модификатор доступа - public;
- handler_input_data:
    - Функционал - метод обработчика, создающий новый объект;
    - Параметры:
        - Строковая переменная msg;
    - Тип возвращаемого значения: нет;
    - Модификатор доступа - public;

наследуемый от класса cl_base:

- Поля: наследуемые от класса cl_base;
- Методы:
    - Параметризированный конструктор reader_commands :
        - Функционал: параметризированный констркукторкоторый в

качестве параметров принимает указатель наголовной объект в дереве иерархии и возможное наименование текущего объекта;

- Параметры:
  - Указатель head_object на объект класса cl_base - отвечает за доступа к головному объекту текущего объекта;
  - Строковый параметр s_name - возможное наименование текущего объекта;
- Тип возвращаемого значения: объект класса cl_base;
- Модификатор доступа: public;

- signal_tact:
  - Функционал - метод сигнала, считывающий команду и обрабатывающий такт;
  - Параметры:
    - Ссылка на строковую переменную msg;
  - Тип возвращаемого значения: нет;
  - Модификатор доступа - public;
- handler_input_commands:
  - Функционал - метод обработчика, принимает строку с командой и выдает сигнал соответствующему объекту;
  - Параметры:
    - Строковая переменная msg;
  - Тип возвращаемого значения: нет;
  - Модификатор доступа - public;

Класс control_panel

наследуемый от класса cl_base:

- Поля: наследуемые от класса cl_base;

- Методы:
    - Параметризированный конструктор control_panel :
        - Функционал: параметризированный констркукторкоторый в качестве параметров принимает указатель наголовной объект в дереве иерархии и возможное наименование текущего объекта;
        - Параметры:
            - Указатель head_object на объект класса cl_base - отвечает за доступа к головному объекту текущегообъекта;
            - Строковый параметр s_name - возможное наименование текущего объекта;
        - Тип возвращаемого значения: объект класса cl_base;
        - Модификатор доступа: public;
    - signal_commands:
        - Функционал - метод сигнала, принимающий команду;
        - Параметры:
            - Ссылка на строковую переменную msg;
        - Тип возвращаемого значения: нет;
        - Модификатор доступа - public;
    - handler_commands:
        - Функционал - метод обработчика, обрабатывает поступившую команду;
        - Параметры:
            - Строковая переменная msg;
        - Тип возвращаемого значения: нет;
        - Модификатор доступа - public;

Класс prints

наследуемый от класса cl_base:

- Поля: наследуемые от класса cl_base;
- Методы:
    - Параметризированный конструктор prints :
        - Функционал: параметризированный констркуктор который в качестве параметров принимает указатель наголовной объект в дереве иерархии и возможное наименование текущего объекта;
        - Параметры:
            - Указатель head_object на объект класса cl_base - отвечает за доступа к головному объекту текущегообъекта;
            - Строковый параметр s_name - возможное наименование текущего объекта;
        - Тип возвращаемого значения: объект класса cl_base;
        - Модификатор доступа: public;
    - signal_prints:
        - Функционал - метод сигнала, принимающий команду;
        - Параметры:
            - Ссылка на строковую переменную msg;
        - Тип возвращаемого значения: нет;
        - Модификатор доступа - public;
    - handler_prints:
        - Функционал - метод обрабочтика, выводит информацию о системе АТС;
        - Параметры:
            - Строковая переменная msg;
        - Тип возвращаемого значения: нет;
        - Модификатор доступа - public;

Класс phones

наследуемый от класса cl_base:

- Поля: наследуемые от класса cl_base;
- Методы:
  - Параметризированный конструктор phones :
    - Функционал: параметризированный констркуктор который в качестве параметров принимает указатель наголовной объект в дереве иерархии и возможное наименование текущего объекта;
    - Параметры:
      - Указатель head_object на объект класса cl_base - отвечает за доступа к головному объекту текущего объекта;
      - Строковый параметр s_name - возможное наименование текущего объекта;
    - Тип возвращаемого значения: объект класса cl_base;
    - Модификатор доступа: public;
  - signal_phone:
    - Функционал - метод сигнала, принимающий команду;
    - Параметры:
      - Ссылка на строковую переменную msg;
    - Тип возвращаемого значения: нет;
    - Модификатор доступа - public;
  - handler_phone:
    - Функционал - метод обрабочтика, устанавливает связь между телефонами;
    - Параметры:
      - Строковая переменная msg;
    - Тип возвращаемого значения: нет;

- Модификатор доступа - public;

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	cl_base	ATC	public	Базовый класс	2	
		reader_commands	public		3	
		control_panel	public		4	
		prtins	public		5	
		phones	public		6	
2	ATC			Класс объекта системы		
3	reader_commands			Класс объекта для чтения команд		
4	control_panel			Класс объекта пульта управления		
5	prtins			Класс объекта вывода информации о системе		
6	phones			Класс объекта модилирующий работу телефона		

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 set_count_call класса cl_base

Функционал: Установка количества вызовов.

Параметры: Целочисленная переменная count_call.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_count_call класса cl_base

№	Предикат	Действия	№ перехода
1		Присваиваем новое значение count_talk текущему объекту	∅

3.2 Алгоритм метода get_count_call класса cl_base

Функционал: Получения количества вызовов.

Параметры: Нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода get_count_call класса cl_base

№	Предикат	Действия	№ перехода
1		Вернуть значение поля count_call у текущего объекта	∅

3.3 Алгоритм метода set_sum_calls класса cl_base

Функционал: Установки количества разговоров.

Параметры: Целочисленная переменная sum_calls.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_sum_calls класса cl_base

№	Предикат	Действия	№ перехода
1		Присваиваем новое значение sum_calls текущему объекту	∅

3.4 Алгоритм метода get_sum_calls класса cl_base

Функционал: Получения количества разговоров.

Параметры: Нет.

Возвращаемое значение: Целочисленное значение sum_calls.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_sum_calls класса cl_base

№	Предикат	Действия	№ перехода
1		Вернуть значение поля sum_calls у текущего объекта	∅

3.5 Алгоритм метода set_count_talk класса cl_base

Функционал: Установки количества звонков.

Параметры: Целочисленное значение count_talk.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода set_count_talk класса cl_base

№	Предикат	Действия	№ перехода
1		Присваиваем новое значение count_talk текущему объекту	∅

3.6 Алгоритм метода get_count_talk класса cl_base

Функционал: Получения количества звонков.

Параметры: Нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_count_talk класса cl_base

№	Предикат	Действия	№ перехода
1		Вернуть значение поля count_talk у текущего объекта	∅

3.7 Алгоритм метода set_is_free класса cl_base

Функционал: Установка занятости номера телефона.

Параметры: Булевое значение if_free.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода set_is_free класса cl_base

№	Предикат	Действия	№ перехода
1		Присваиваем новое значение is_free текущему объекту	∅

3.8 Алгоритм метода get_is_free класса cl_base

Функционал: Получение занятости номера телефона.

Параметры: Нет.

Возвращаемое значение: Булвое значение.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_is_free класса cl_base

№	Предикат	Действия	№ перехода
1		Вернуть значение поля is_free у текущего объекта	∅

3.9 Алгоритм конструктора класса ATC

Функционал: Параметризированный конструктор, который в качестве параметров принимает указатель на головной объект в дереве иерахии и возможное наименование текущего объекта.

Параметры: Указатель head_object на объект класса cl_base, строковый параметр s_name .

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса ATC

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base и передача в него в качествепараметра значения параметра head_object	∅

3.10 Алгоритм метода build_tree_objects класса ATC

Функционал: Используется для построения дерева иерархии объектов.

Параметры: Нет.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода build_tree_objects класса АТС

№	Предикат	Действия	№ перехода
1		Вызов метода set_name с параметром "АТС"	2
2		Создает указатель p_sub на объект текущего класса АТС	3
3		Создает новый объект класса reader_commands с передачей указателя this и строкового значения read	4
4		Создает новый объект класса control_panel с передачей указателя this и строкового значения pult	5
5		Создает новый объект класса prints с передачей указателя this и строкового значения prints	6
6		Создает новый объект класса phones с передачей указателя this и строкового значения phones	7
7		Вызов метода setFullReadiness для перевода все объектов в состояние готовности	8
8		Инициализация строковой переменной s_msg	9
9		Вызов метода set_connection который устанавливает соединение между сигнало signal_reader_to_all и обработчиком handler_input_data	10
10	c_cmd не равен "End of phones"		11
			12
11		Вызов метода emit_signal который генерирует сигнал signal_reader_to_all со значением c_cmd	10
12		Установка связей между сигналами и обработчиками для различных классов reader_commands, control_panel, prints, phones	∅

3.11 Алгоритм метода exec_app класса АТС

Функционал: Запуск системы.

Параметры: Нет.

Возвращаемое значение: Индикатор корректности завершения программы.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода exec_app класса АТС

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной command	2
2		Вывод "Ready to work" + перенос на новую строку	3
3	c_cmd != "SHOWTREE" и c_cmd != "Turn off the system"		4
			5
4		Вызов метода emit_signal который генерирует сигнал signal_tact со значением c_cmd	3
5		Вывод "Turn off the ATM" + перенос на новую строку	6
6		Вернуть 0	∅

3.12 Алгоритм конструктора класса reader_commands

Функционал: Параметризированный конструктор, который в качестве параметров принимает указатель на головной объект в дереве иерахии и возможное наименование текущего объекта.

Параметры: Указатель head_object на объект класса cl_base, строковый параметр s_name .

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса reader_commands

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base и передача в него в качествепараметра значения параметра head_object	∅

3.13 Алгоритм метода signal_reader_to_all класса АТС

Функционал: Метод сигнала, считывающий строку.

Параметры: Ссылка на строковую переменную msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода signal_reader_to_all класса АТС

№	Предикат	Действия	№ перехода
1		Ввод строки в переменную msg	∅

3.14 Алгоритм метода handler_input_data класса АТС

Функционал: Метод обработчика, создающий новый объект.

Параметры: Строковая переменная msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода handler_input_data класса АТС

№	Предикат	Действия	№ перехода
1	Размер строки msg = 7 и объект с данным номером еще не создан	Создает новй объект obj класса control_panel	2
			∅
2		Вызов метода emit_signal, который устанавливает	∅

№	Предикат	Действия	№ перехода
		соединение между сигнало signal_prints и обработчиком handler_prints объекта obj	

3.15 Алгоритм метода signal_tact класса reader_commands

Функционал: Метод сигнала, считывающий команду и обрабатывающий такт.

Параметры: Ссылка на строковую переменную msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода signal_tact класса reader_commands

№	Предикат	Действия	№ перехода
1		Считывает строку и записывает ее в msg	2
2	msg равен "Turn off the system"	Присвоение переменной c_cmd = "Turn off the system" и закончить цикл	∅
			3
3	msg равен "SHOWTREE"	Присвоение переменной c_cmd = "SHOWTREE"	4
			5
4		Присвоение переменной msg = "Display the system status" и закончить цикл	∅
5	Перебраны не все подобъекты объекта pult		6
			∅
6	journal_list текущего подобъекта не пустой и текущий звонок завершился	Вызов метода set_is_free с параметром true	5
			5

3.16 Алгоритм метода handler_input_commands класса reader_commands

Функционал: Метод обработчика, принимает строку с командой и выдает сигнал соответствующему объекту.

Параметры: Строковая переменная msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода handler_input_commands класса reader_commands

№	Предикат	Действия	№ перехода
1		Постфиксный инкремент переменной tact	2
2	msg.substr(0, 12) == "Call request"	Вызов метода emit_signal сигнала signal_commands класса control_panel с передачей msg в качестве аргумента	∅
			3
3	msg.substr(0,25) "Display phone "	Вызов метода emit_signal сигнала signal_prints класса prints с передачей msg в качестве аргумента	∅
			4
4	msg == "Display the system status"	Вызов метода emit_signal сигнала signal_prints класса prints с передачей msg в качестве аргумента	∅
			∅

3.17 Алгоритм конструктора класса control_panel

Функционал: Параметризированный конструктор, который в качестве параметров принимает указатель на головной объект в дереве иерахии и возможное наименование текущего объекта.

Параметры: Указатель head_object на объект класса cl_base, строковый параметр s_name .

Алгоритм конструктора представлен в таблице 18.

Таблица 18 – Алгоритм конструктора класса control_panel

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base и передача в него в качествепараметра значения параметра head_object	2
2		Вызов от текущего объекта метода set_sum_call с параметром 0	3
3		Вызов от текущего объекта метода set_count_talk с параметром 0	4
4		Вызов от текущего объекта метода set_count_call с параметром 0	5
5		Вызов от текущего объекта метода set_is_free с параметром true	∅

3.18 Алгоритм метода signal_commands класса control_panel

Функционал: Метод сигнала, принимающий команду.

Параметры: Ссылка на строковую переменную msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода signal_commands класса control_panel

№	Предикат	Действия	№ перехода
1			∅

3.19 Алгоритм метода handler_commands класса control_panel

Функционал: Метод обработчика, обрабатывает поступившую команду.

Параметры: Строковая переменная msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода handler_commands класса control_panel

№	Предикат	Действия	№ перехода
1		Инициализирует переменную command и присваивает ей значение msg	2
2	command.substr(0, 12) == "Call request"		3
			∅
3		находит позицию первого пробела в строке command и сохраняет ее в переменную firstSpacePos	4
4		находит позицию второго пробела в строке command, начиная с позиции firstSpacePos + 1, и сохраняет ее в переменную secondSpace	5
5		создает подстроку subString из строки command, начиная с позиции	6
6		создает объект istringstream с инициализацией строки subString	7
7		объявляет строки first и second	8
8		объявляет целочисленную переменную time	9
9		считывает значения из потока iss и присваивает их переменным first, second и time	10
10		объявляет и инициализирует переменную i со значением 0	11
11	размер строки second не равен 7	Присваивает переменной msg строковое сообщение об ошибке, содержащее некорректный номер абонента и выводит ее на экран	∅
			12
12	i != 7		13
			14
13	second[i] < '1' или second[i]> '9'	Присваивает переменной msg строковое сообщение об ошибке, содержащее некорректный	∅

№	Предикат	Действия	№ перехода
		номер абонента и выводит ее на экран	
		i++	12
14		Вызов метода find_object_from_root который находит объект типа cl_base с именем first в иерархии объектов	15
15		Вызов метода find_object_from_root который находит объект типа cl_base с именем second в иерархии объектов	16
16	obj_1 свободен		17
		Вызов метода emit_signal сигнала signal_prints класса prints с передачей msg в качестве аргумента и вывод "The phone is busy, a new call is not possible: " + first + перенос строки	∅
17		Вызов метода get_count_call()+1 и присвоение его переменной count_call_1	18
18		Вызов метода set_count_call параметром count_call_1	19
19	obj_2 != nullptr		20
		Вызов метода emit_signal сигнала signal_prints класса prints с передачей msg в качестве аргумента и вывод "Subscriber " + second + " not found" + перенос строки	∅
20	obj_2 свободен		21
		Вызов метода emit_signal сигнала signal_prints класса prints с передачей msg в качестве аргумента и вывод "The phone is busy, a new call is not possible: " + first	∅
21		Вызов метода для объекта obj_1 get_sum_calls()+1 и присвоение его переменной sum_call_1	22

№	Предикат	Действия	№ перехода
22		Вызов метода для объекта obj_1 set_sum_calls()с параметром sum_call_1	23
23		Вызов метода для объекта obj_2 get_count_talk()+1 и присвоение его переменной count_talk_2	24
24		Вызов метода для объекта obj_2 set_count_talk с параметром count_talk_2	25
25		Вызов метода set_is_free для объектов obj_1 и obj_2 с параметром false	26
26		Вывод "Call from " + first + " to " + second + ", talk: " + time + перенос строки	27
27		Вызов метода emit_signal сигнала signal_prints класса prints с передачей msg в качестве аргумента	∅

3.20 Алгоритм конструктора класса prints

Функционал: Параметризированный конструктор, который в качестве параметров принимает указатель на головной объект в дереве иерахии и возможное наименование текущего объекта.

Параметры: Указатель head_object на объект класса cl_base, строковый параметр s_name .

Алгоритм конструктора представлен в таблице 21.

Таблица 21 – Алгоритм конструктора класса prints

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base и передача в него в качествепараметра значения параметра head_object	∅

3.21 Алгоритм метода signal_prints класса prints

Функционал: Метод сигнала, принимающий команду.

Параметры: Ссылка на строковую переменную msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода signal_prints класса prints

№	Предикат	Действия	№ перехода
1	msg.substr(0, 25) == "Display phone "	Вывод "Phone log: " + msg.substr(25) + перенос строки	∅
			2
2	msg.substr(0, 25) == "Display the system status"	Вывод "АТС" + перенос строки	∅
			∅

3.22 Алгоритм метода handler_prints класса prints

Функционал: Метод обрабочтика, выводит информацию о системе АТС.

Параметры: Строковая переменная msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода handler_prints класса prints

№	Предикат	Действия	№ перехода
1	msg.substr(0, 12) == "Call request"	Вызове метода emit_signal сигнала signal_phone класса phones с параметром msg	∅
			2
2	msg.substr(0, 25) == "Display phone "	Извлекает подстроку msg начиная с позиции 26 и присваивает ее переменной number	3

№	Предикат	Действия	№ перехода
			5
3		получает подобъект с именем number от подобъекта с именем "pult" у корневого объекта и присваивает его указатель переменной obj	4
4	Перебраны не все элементы вектора journal_list	Вывод stat.takt + " " + stat.type + " " + stat.number + " " + stat.time_talk + перенос строки	4
			∅
5	msg.substr(0, 25) == "Display the system status"	Получает подобъект с именем "pult" у корневого объекта и присваивает его указатель переменной obg	6
			∅
6	Перебраны не все подобъекты	Вывод "telephone " + вызов метода от текущего объекта get_name + " " + вызов метода от текущего объекта get_count_call + " " + вызов метода от текущего объекта get_sum_calls + " " вызов метода от текущего объекта get_count_talk + перенос строки	6
			∅

3.23 Алгоритм конструктора класса phones

Функционал: Параметризированный конструктор, который в качестве параметров принимает указатель на головной объект в дереве иерахии и возможное наименование текущего объекта.

Параметры: Указатель head_object на объект класса cl_base, строковый параметр s_name .

Алгоритм конструктора представлен в таблице 24.

Таблица 24 – Алгоритм конструктора класса phones

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base и передача в него в качествепараметра значения параметра head_object	∅

3.24 Алгоритм метода signal_phones класса phones

Функционал: Метод сигнала, принимающий команду.

Параметры: Ссылка на строковую переменную msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода signal_phones класса phones

№	Предикат	Действия	№ перехода
1			∅

3.25 Алгоритм метода handler_phone класса phone

Функционал: Метод обрабочтика, устанавливает связь между телефонами.

Параметры: Строковая переменная msg.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода handler_phone класса phone

№	Предикат	Действия	№ перехода
1	command.substr(0, 12) == "Call request"	находит позицию первого пробела в строке command и сохраняет ее в переменную firstSpacePos	2
			∅
2		находит позицию второго пробела в строке	3

№	Предикат	Действия	№ перехода
		command, начиная с позиции firstSpacePos + 1, и сохраняет ее в переменную secondSpace	
3		создает подстроку subString из строки command, начиная с позиции	4
4		создает объект istringstream с инициализацией строки subString	5
5		объявляет строки first и second	6
6		объявляет целочисленную переменную time	7
7		считывает значения из потока iss и присваивает их переменным first, second и time	8
8		создает объект entry структуры journal	9
9	Вызов метода get_is_free для первого номера равен false и вызов метода get_is_free для второго номера равен false	присваивает значения полям takt, type, number и time_talk объекта entry для первого номера и добавляет объект entry в конец списка journal_list для первого номера	10
			11
10		присваивает значения полям takt, type, number и time_talk объекта entry для второго номера добавляет объект entry в конец списка journal_list для второго номера	11
11	Вызов метода get_is_free для первого номера равен false и вызов метода get_is_free для второго номера равен true	присваивает значения полям takt, type, number и time_talk объекта entry для второго номера добавляет объект entry в конец списка journal_list для второго номера	∅
			12
12	Вызов метода get_is_free для первого номера равен true и вызов метода get_is_free для второго номера равен fakse	присваивает значения полям takt, type, number и time_talk объекта entry для первого номера и добавляет объект entry в конец списка journal_list для первого номера	∅

№	Предикат	Действия	№ перехода
			∅

3.26 Алгоритм функции main

Функционал: Главная функция программы.

Параметры: Нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм функции представлен в таблице 27.

Таблица 27 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта object класса АТС	2
2		Вызов метода build_tree_objects() объекта object	3
3		Вызов метода exec_app объекта object	4
4		Возврат значения метода exec_app	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-21.

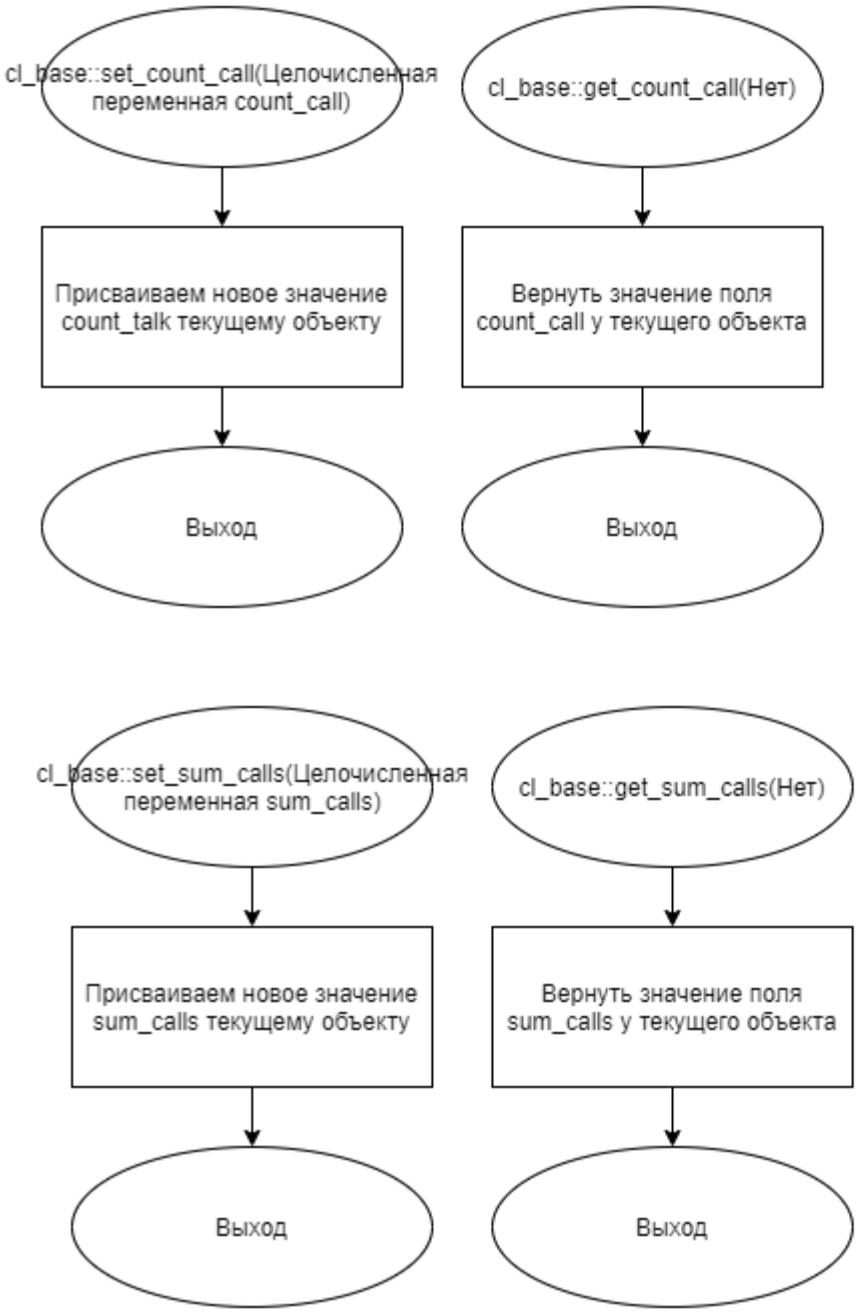

Рисунок 1 – Блок-схема алгоритма

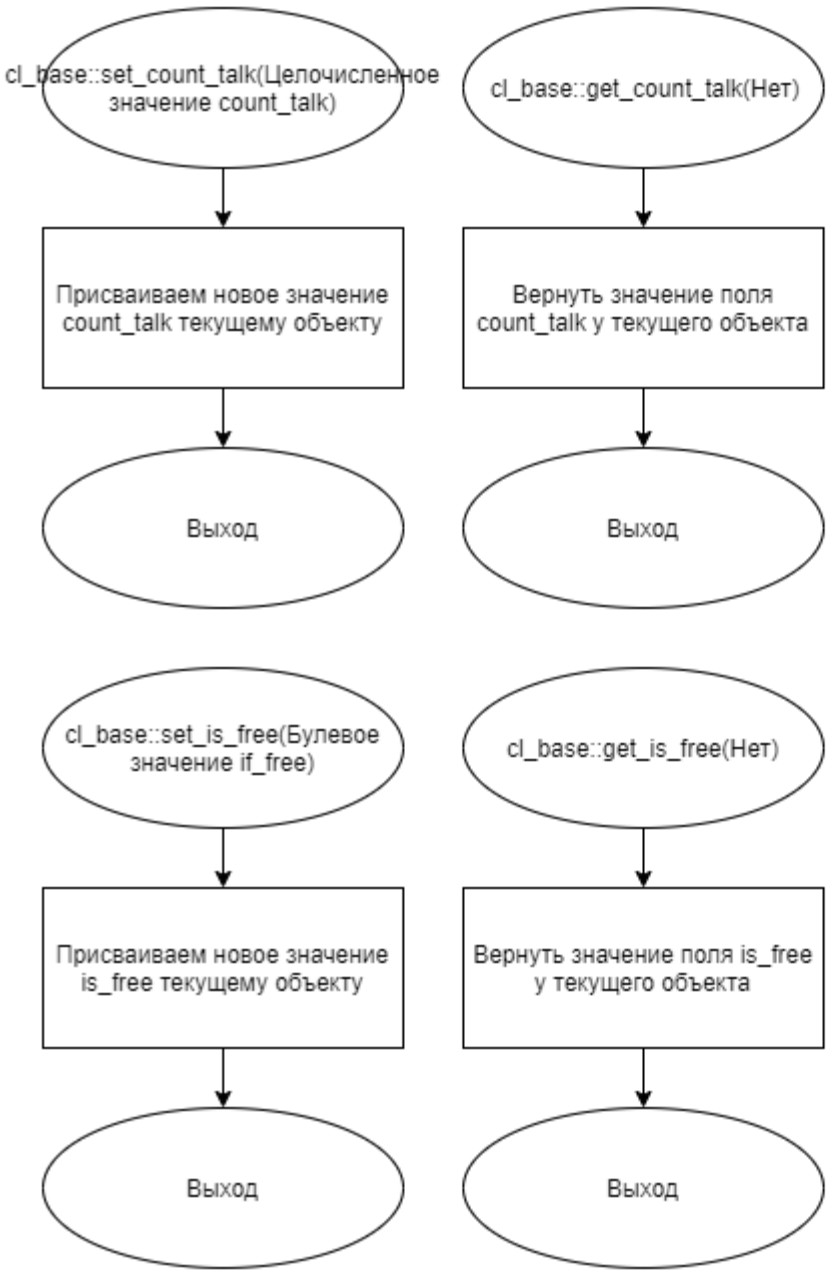

Рисунок 2 – Блок-схема алгоритма

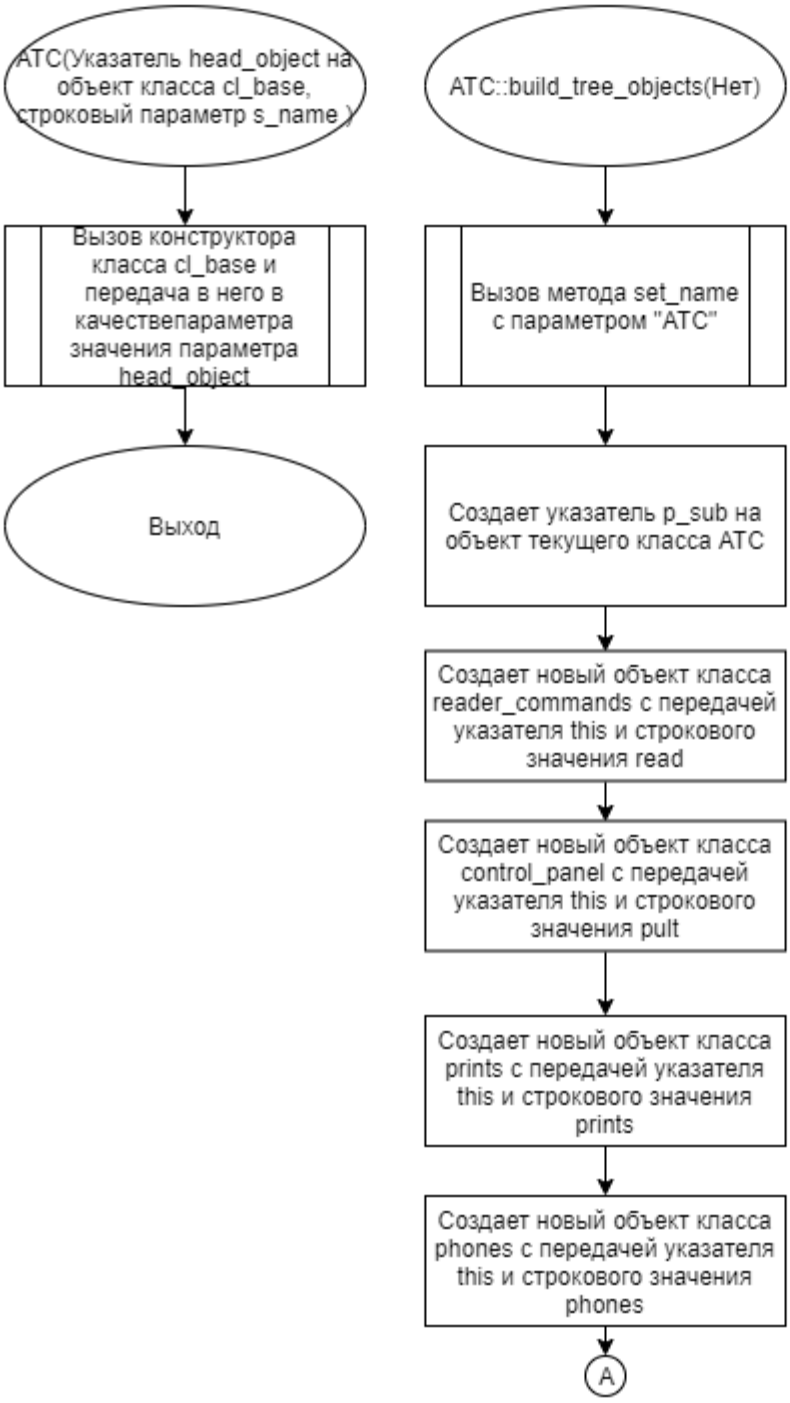

Рисунок 3 – Блок-схема алгоритма

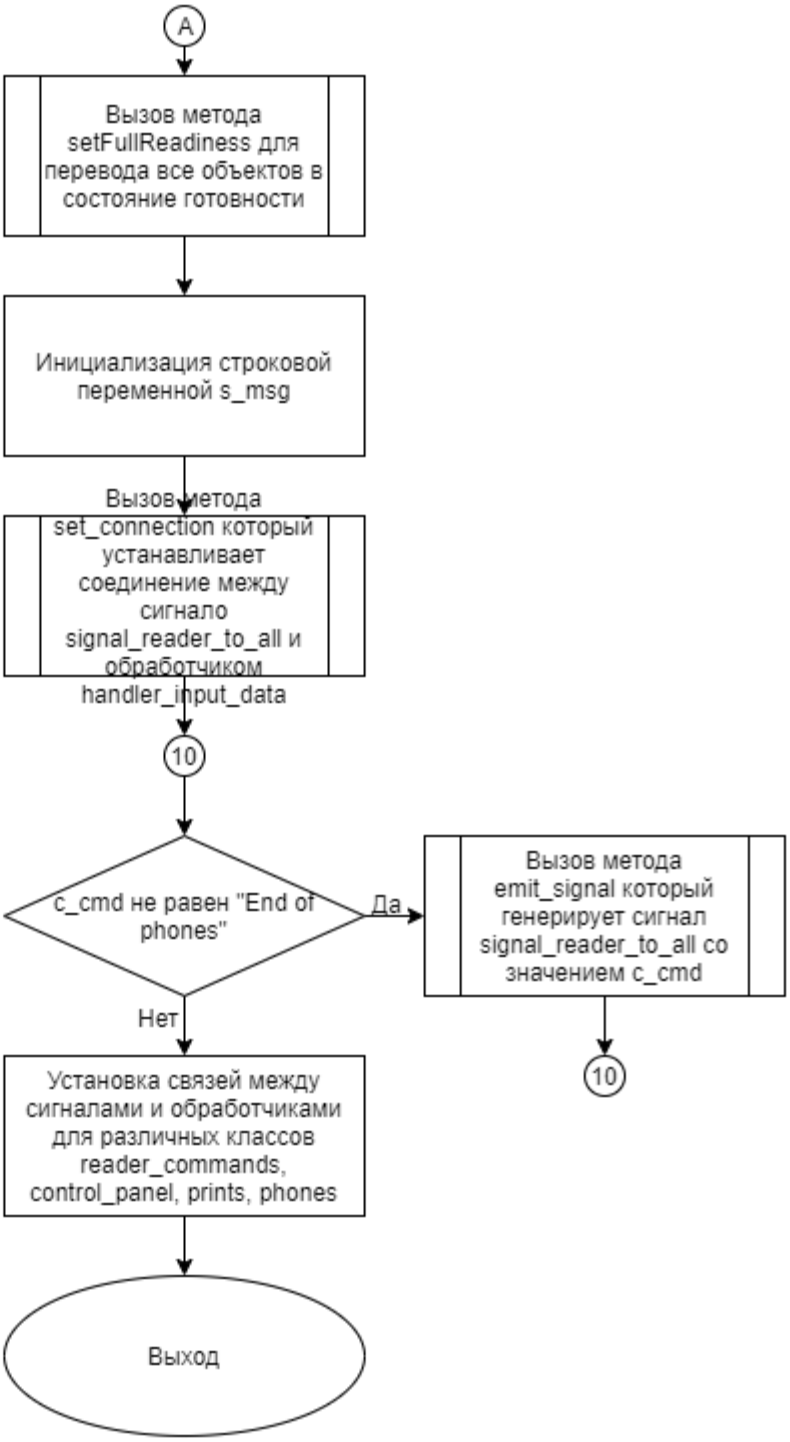

Рисунок 4 – Блок-схема алгоритма

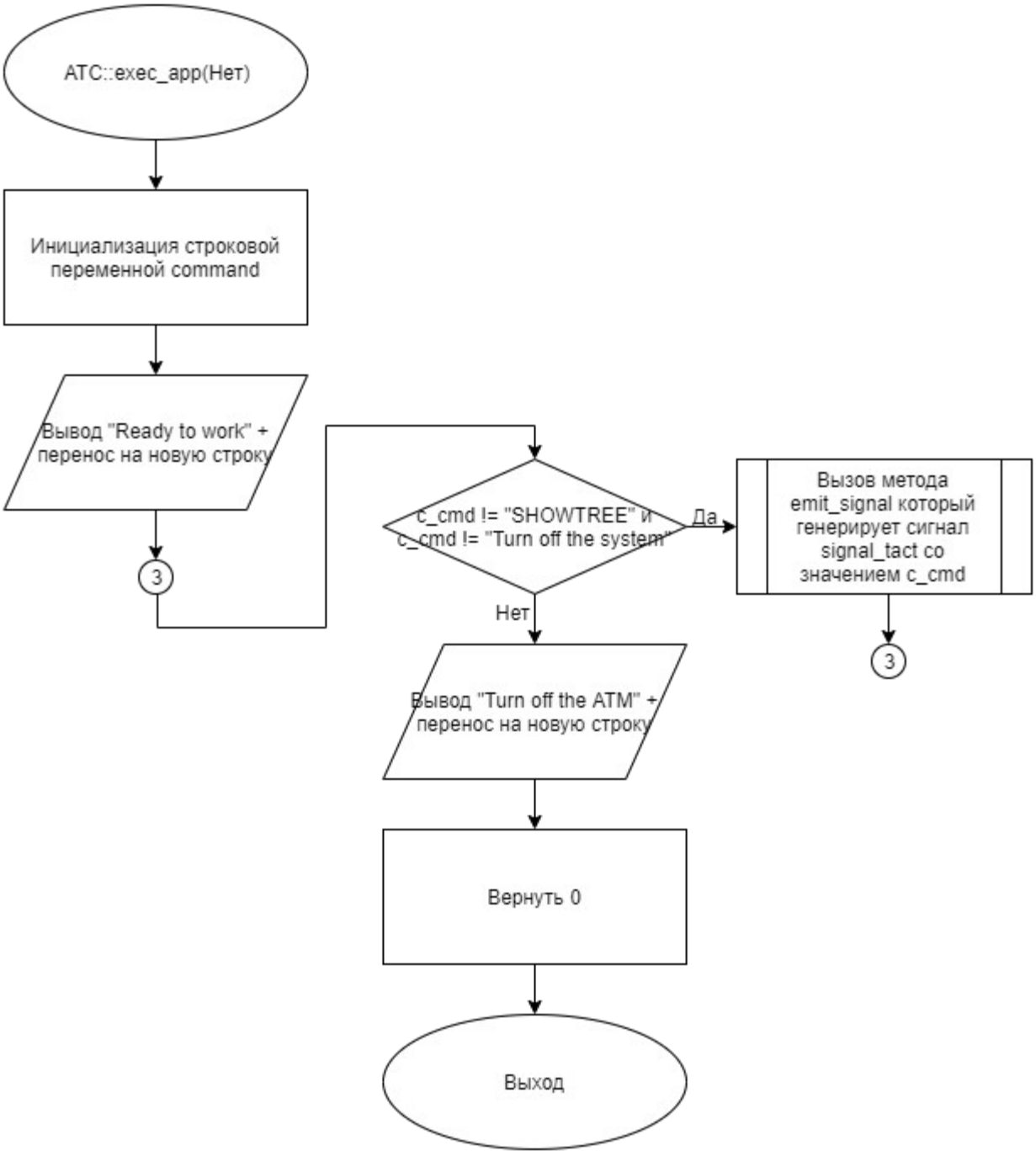

Рисунок 5 – Блок-схема алгоритма

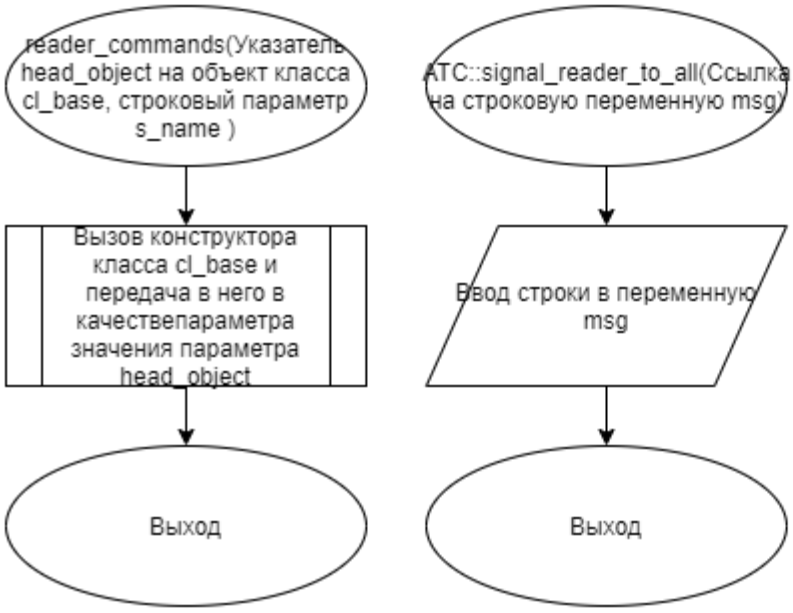

Рисунок 6 – Блок-схема алгоритма

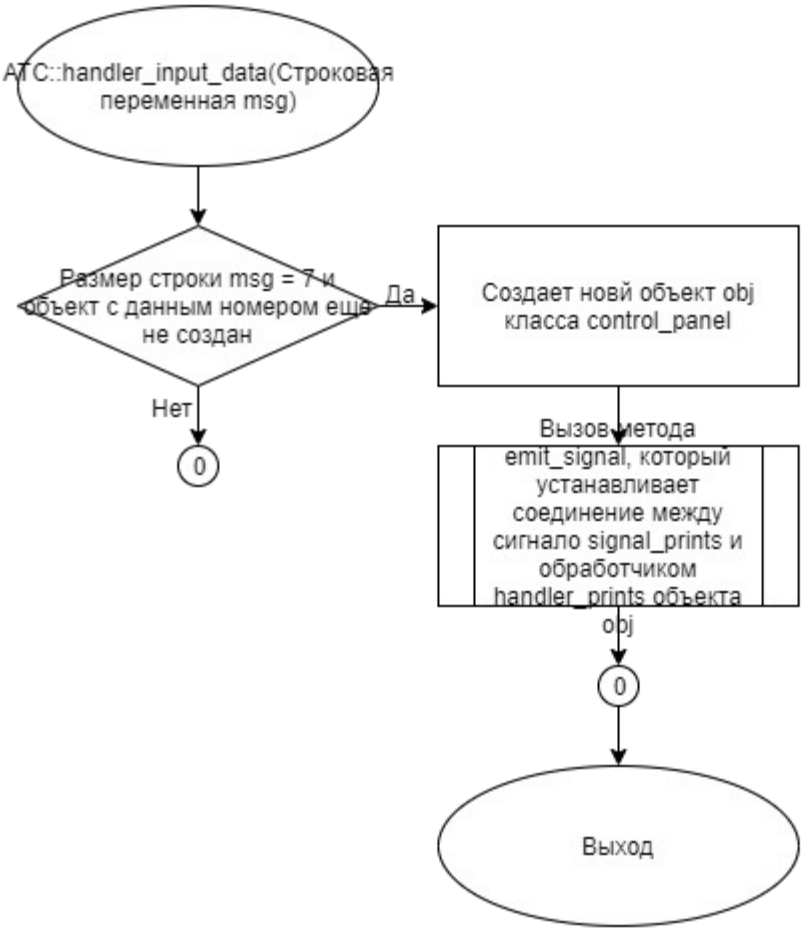

Рисунок 7 – Блок-схема алгоритма

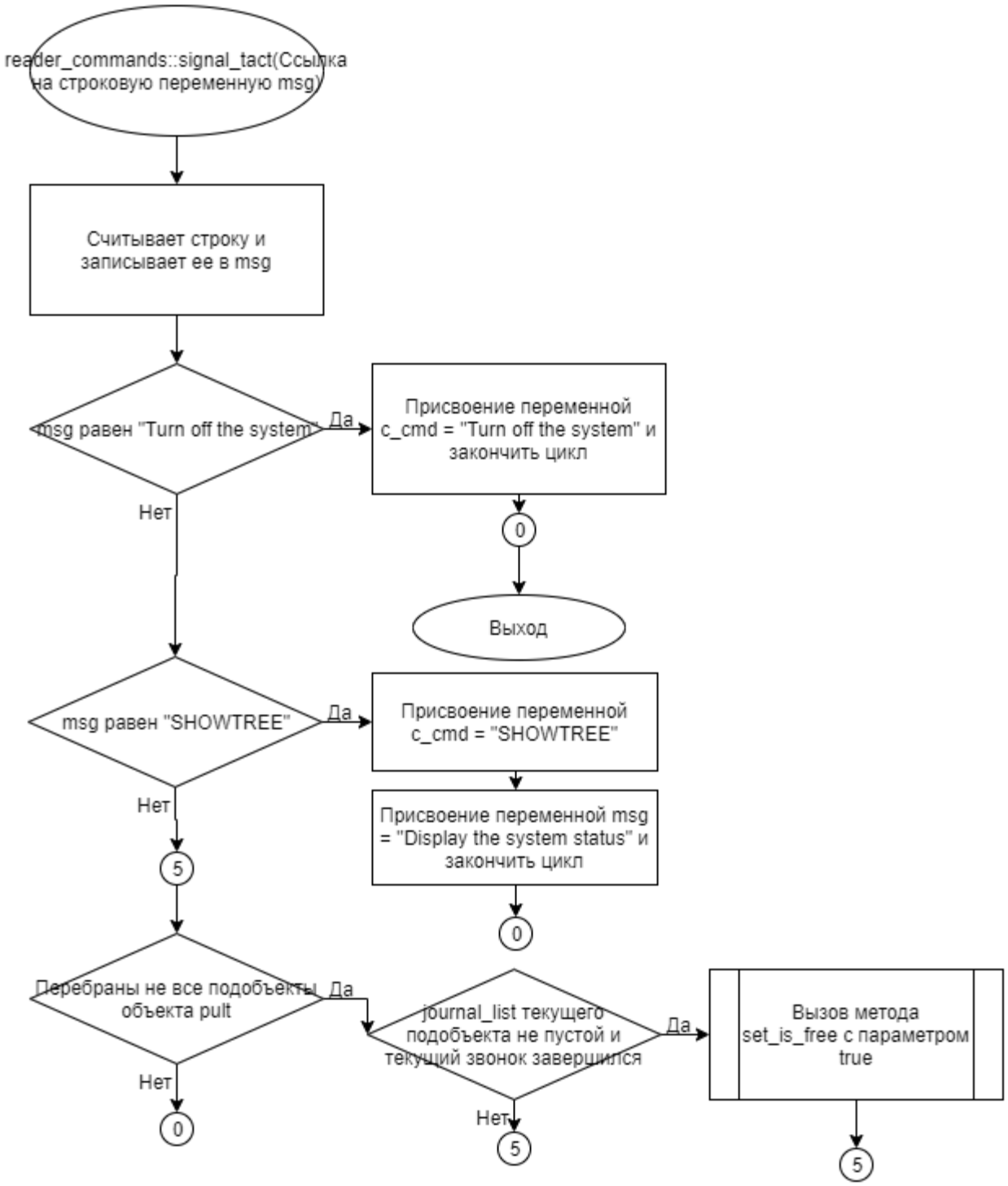

Рисунок 8 – Блок-схема алгоритма

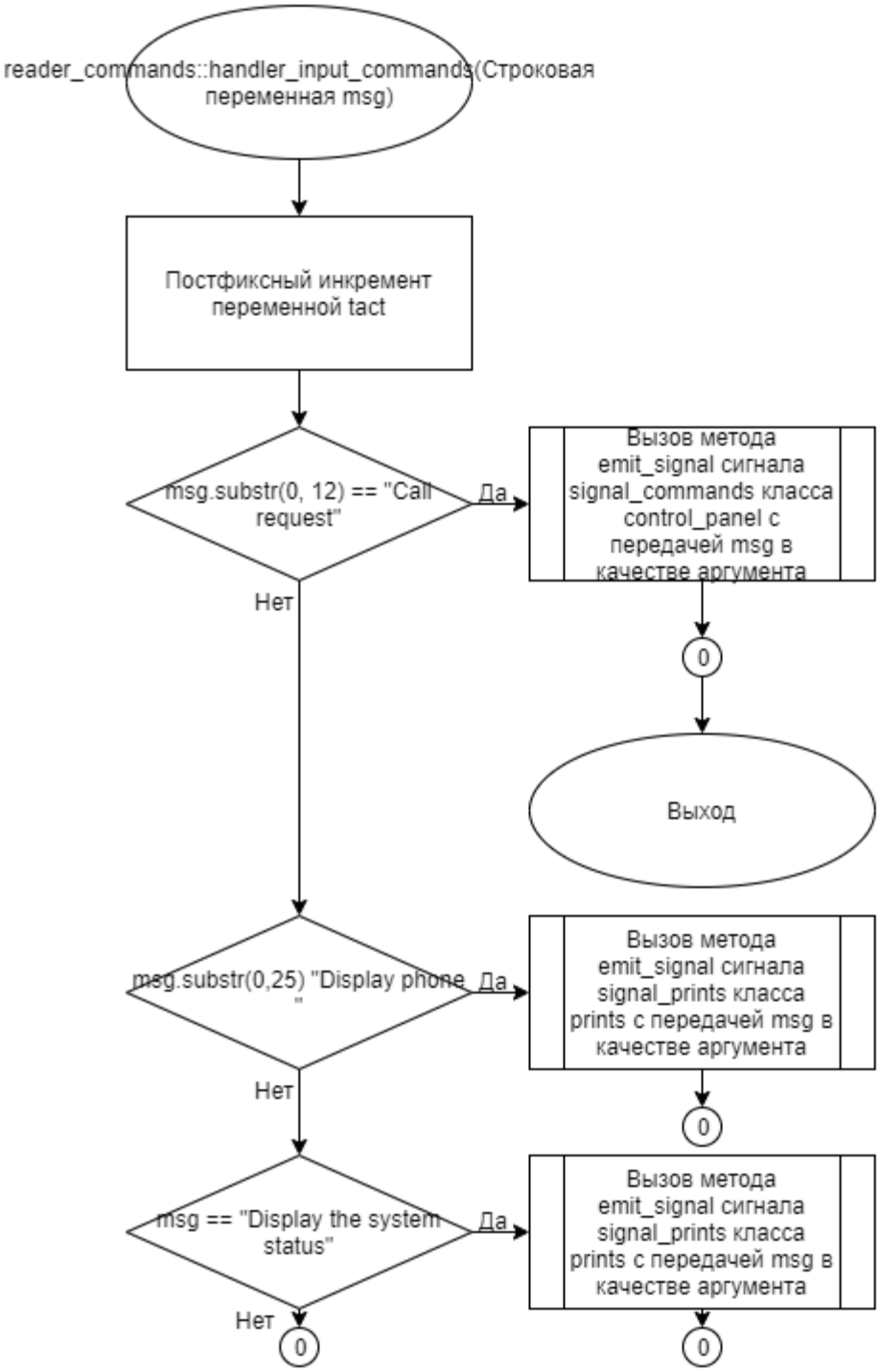

Рисунок 9 – Блок-схема алгоритма

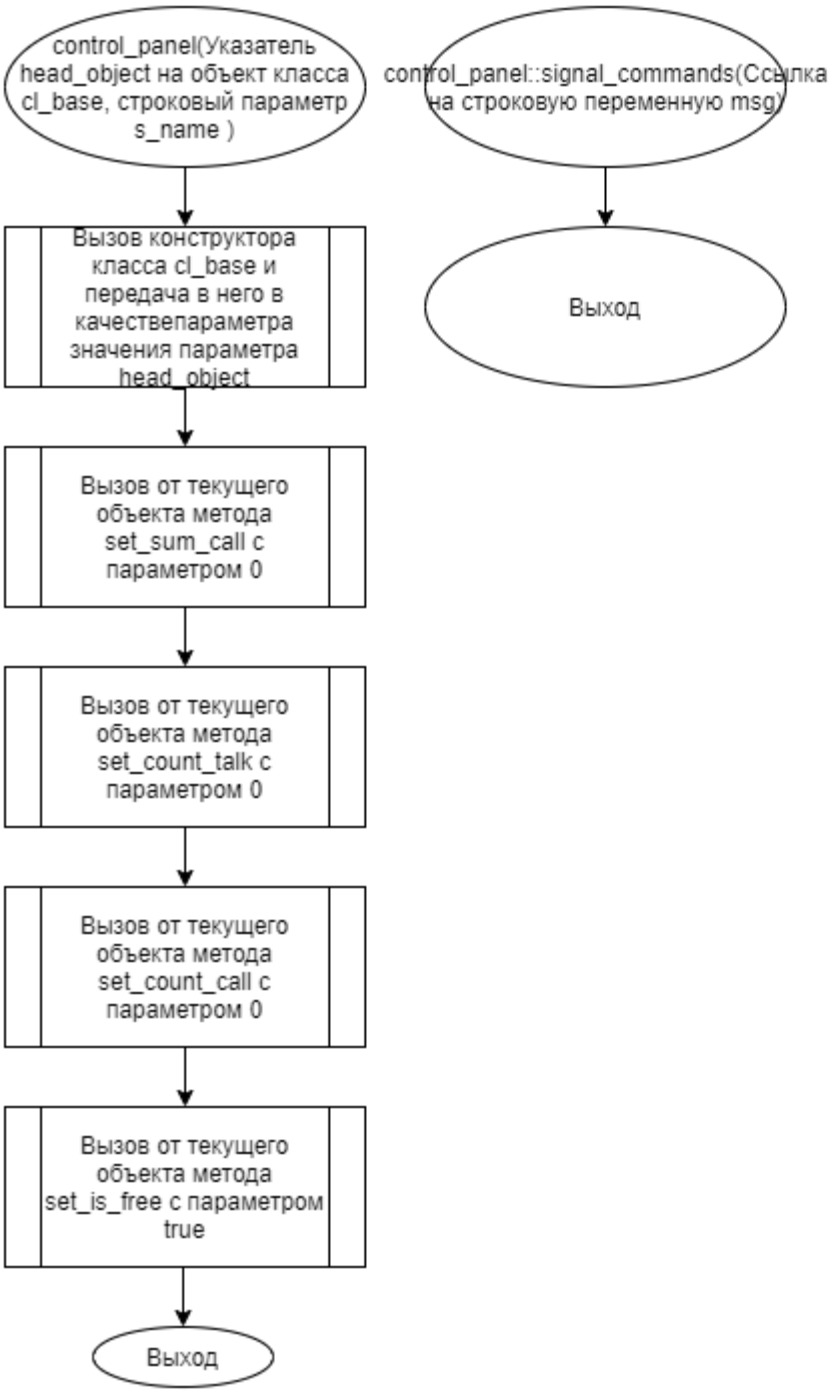

Рисунок 10 – Блок-схема алгоритма

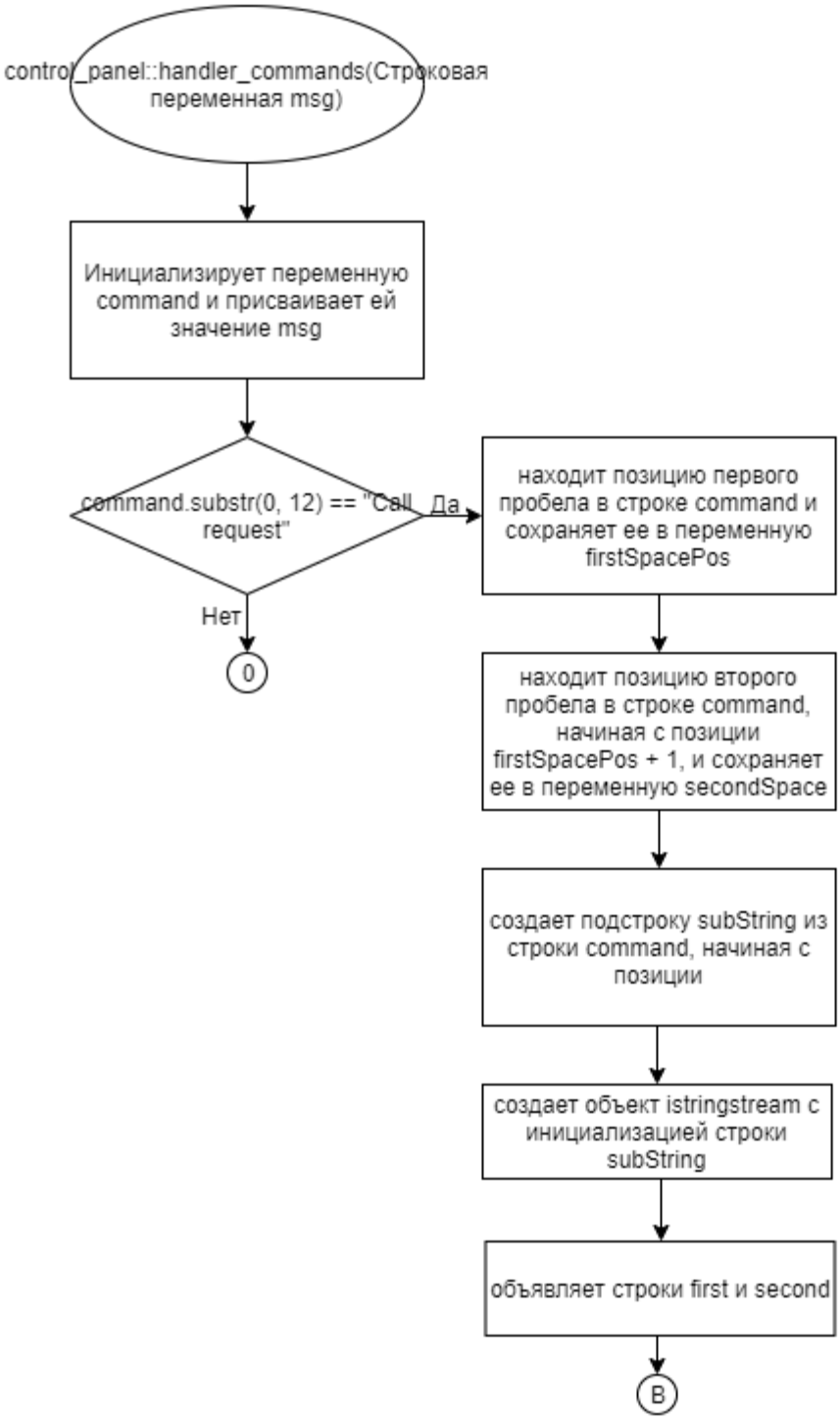

Рисунок 11 – Блок-схема алгоритма

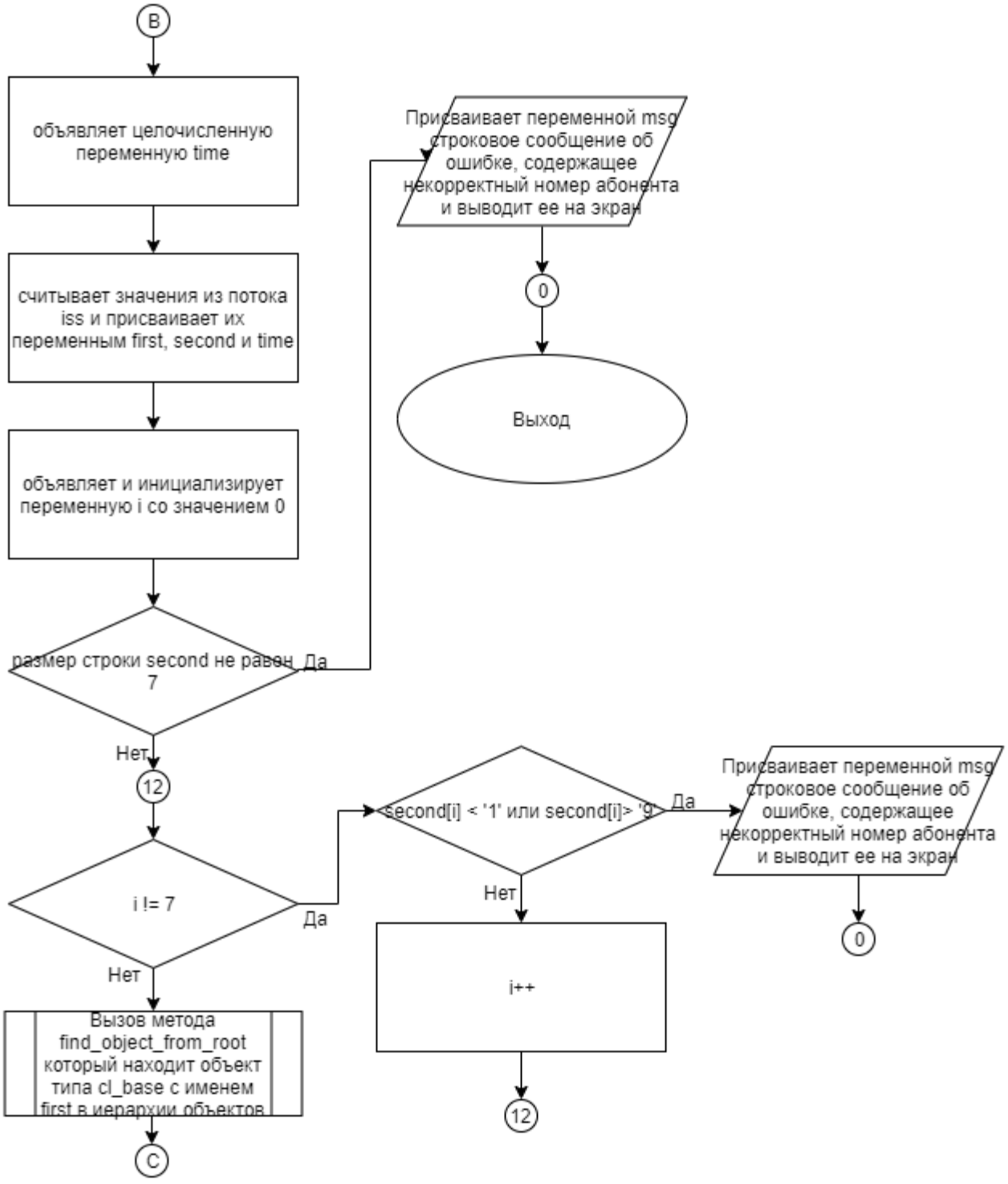

Рисунок 12 – Блок-схема алгоритма

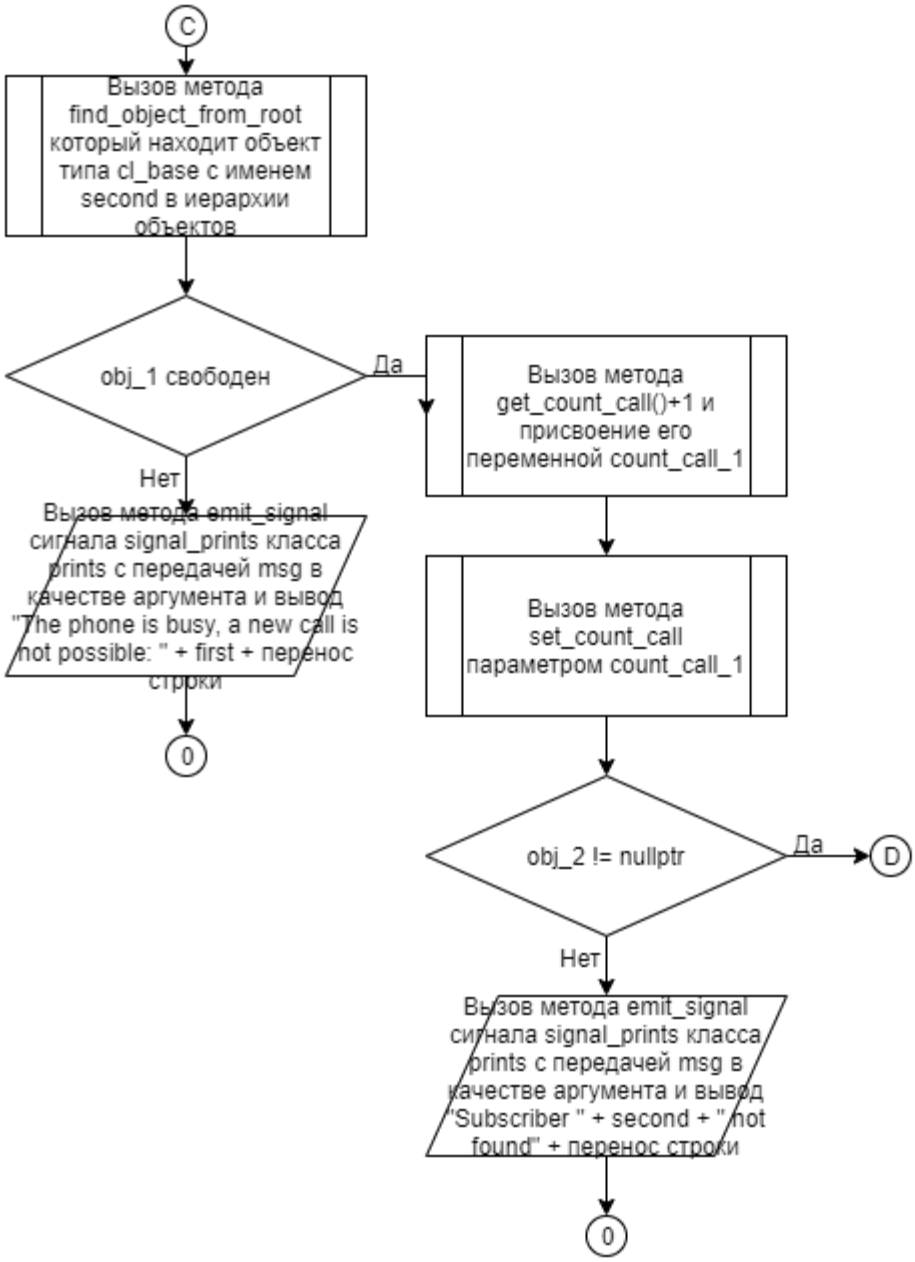

Рисунок 13 – Блок-схема алгоритма

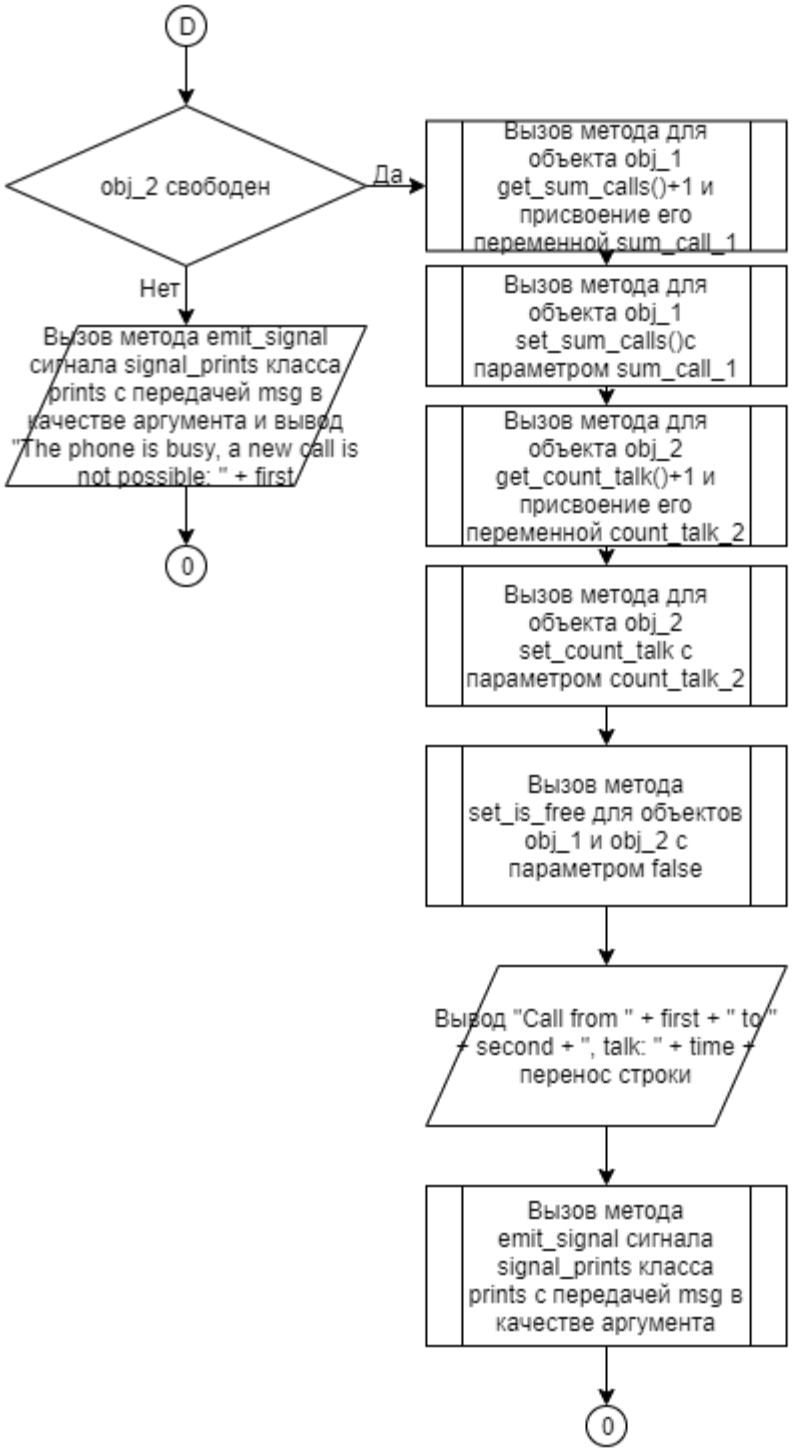

Рисунок 14 – Блок-схема алгоритма

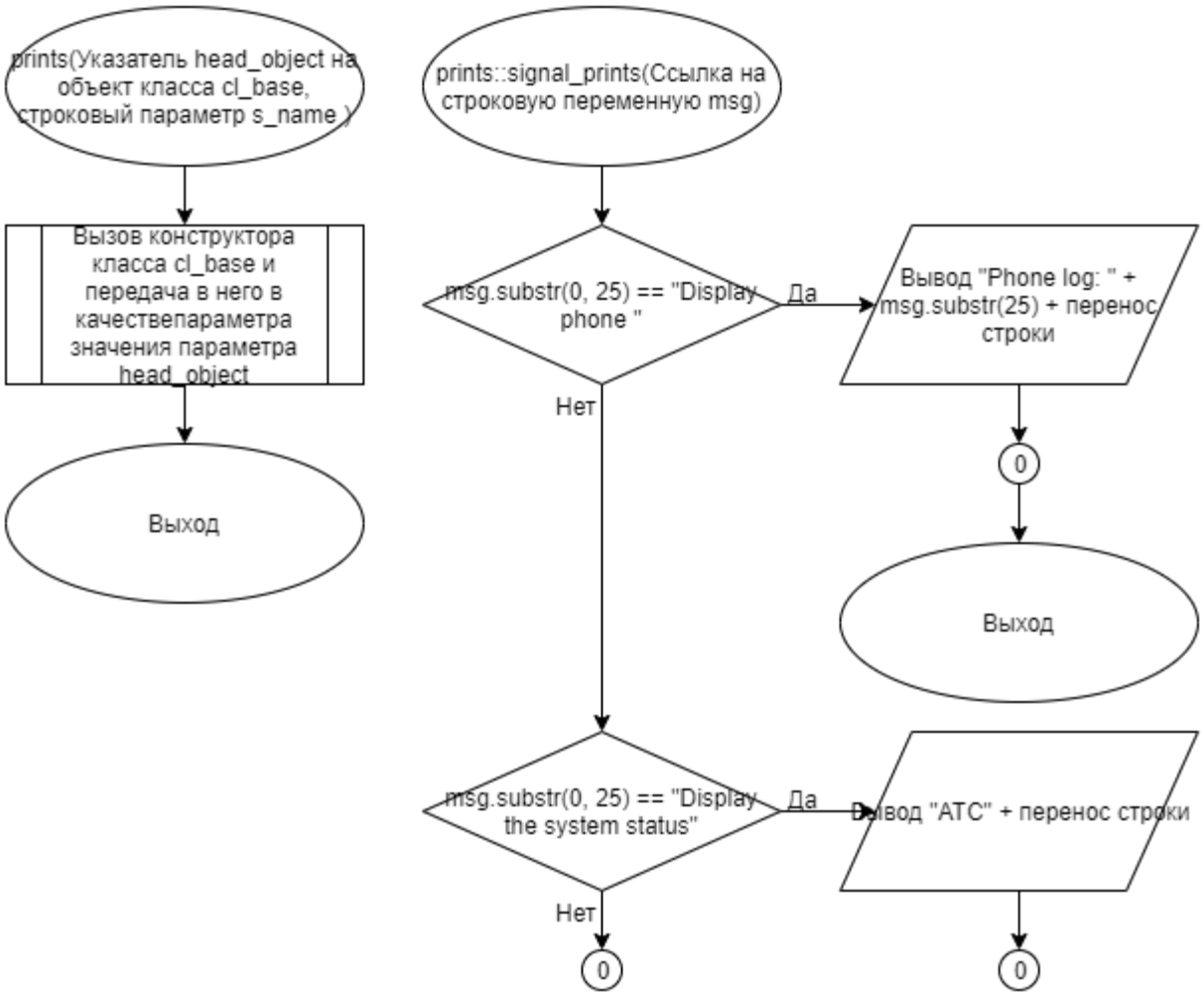

Рисунок 15 – Блок-схема алгоритма

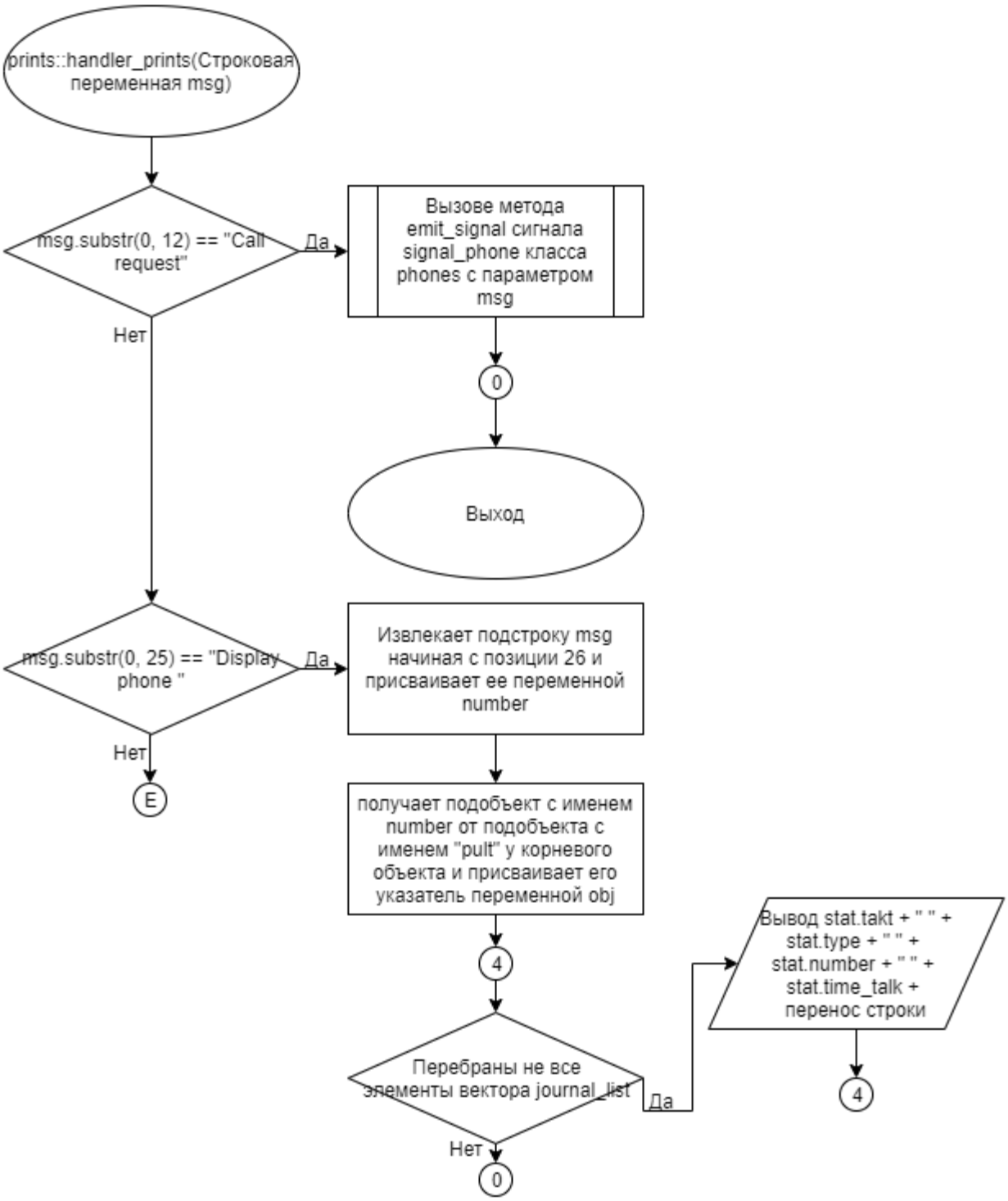

Рисунок 16 – Блок-схема алгоритма

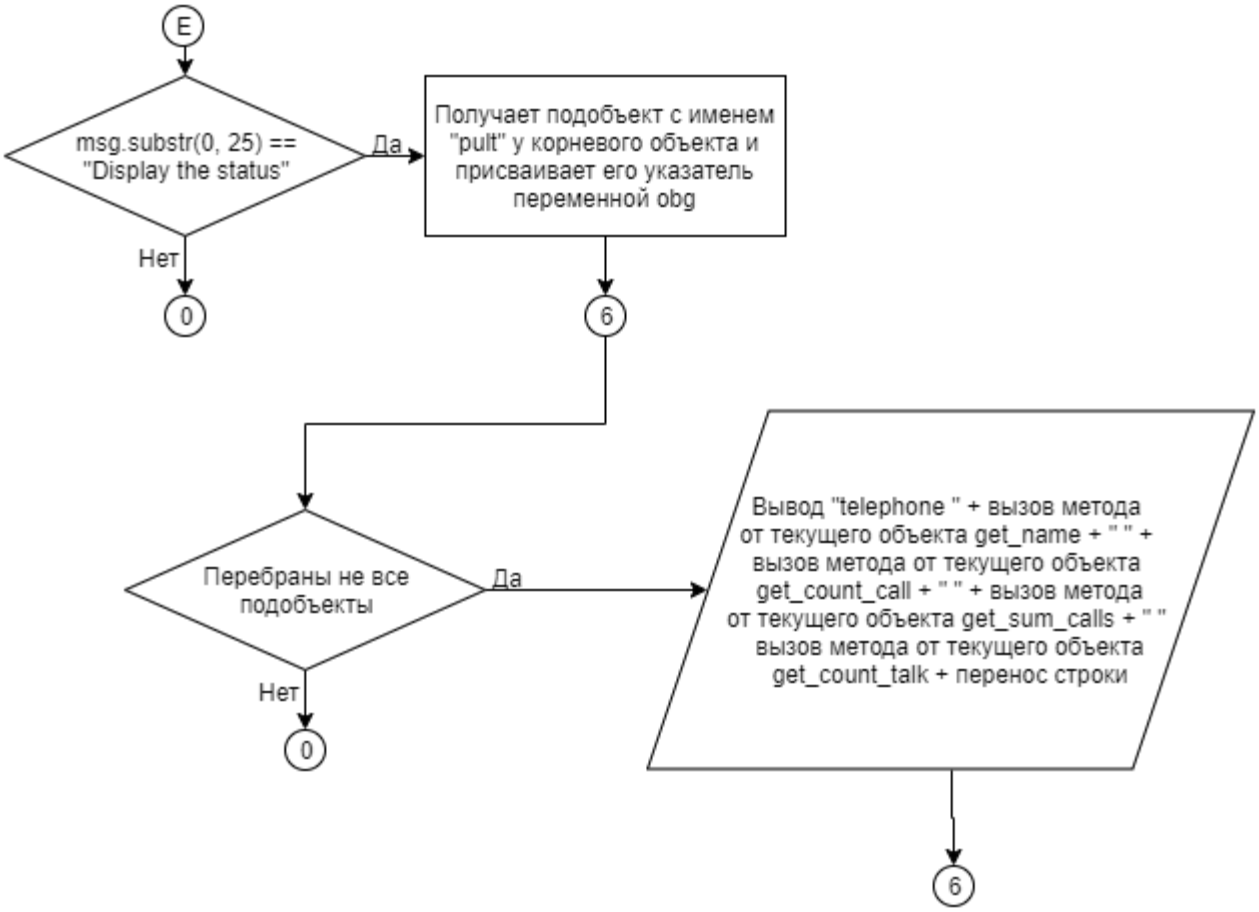

Рисунок 17 – Блок-схема алгоритма

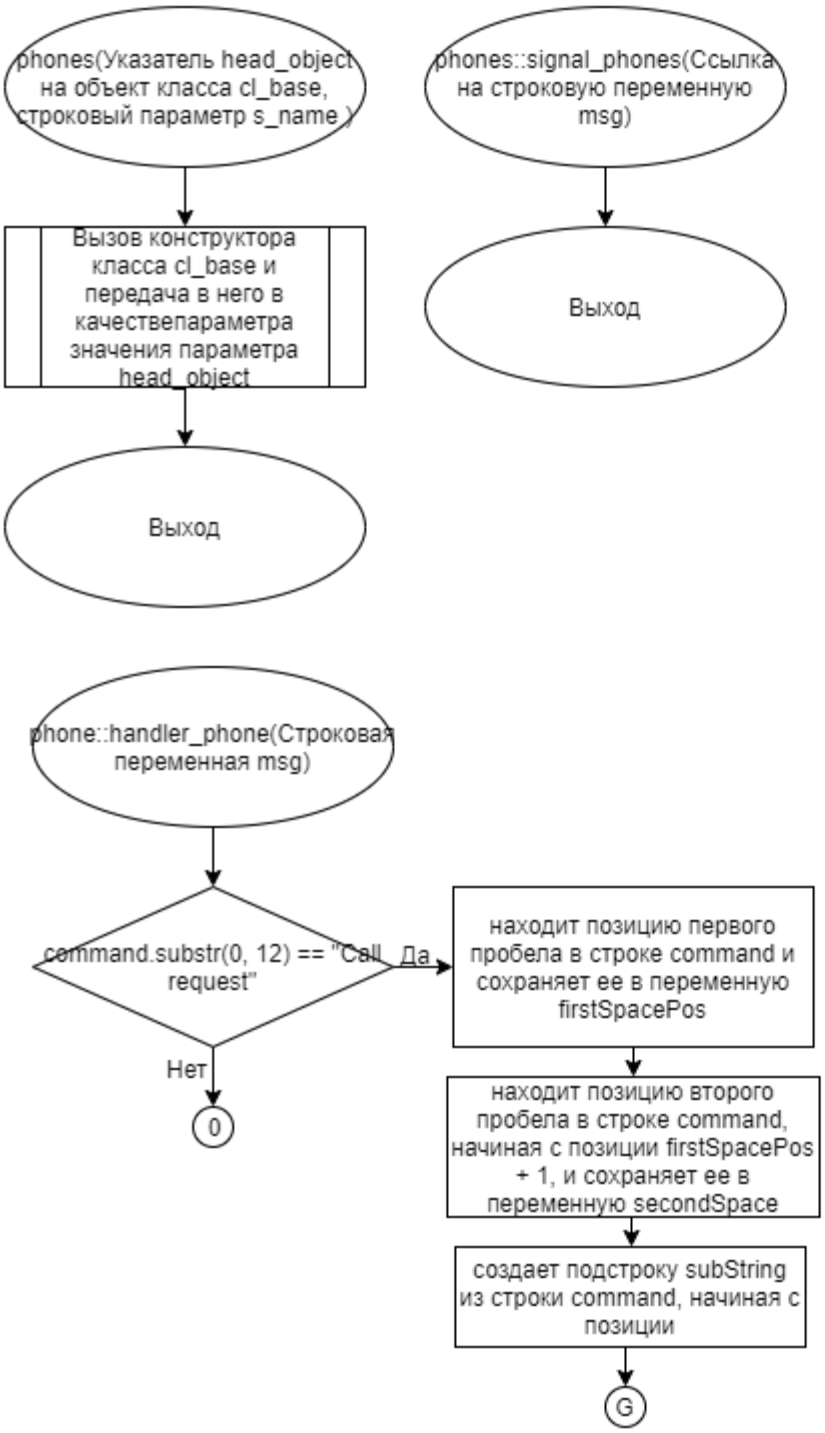

Рисунок 18 – Блок-схема алгоритма

Рисунок 19 – Блок-схема алгоритма

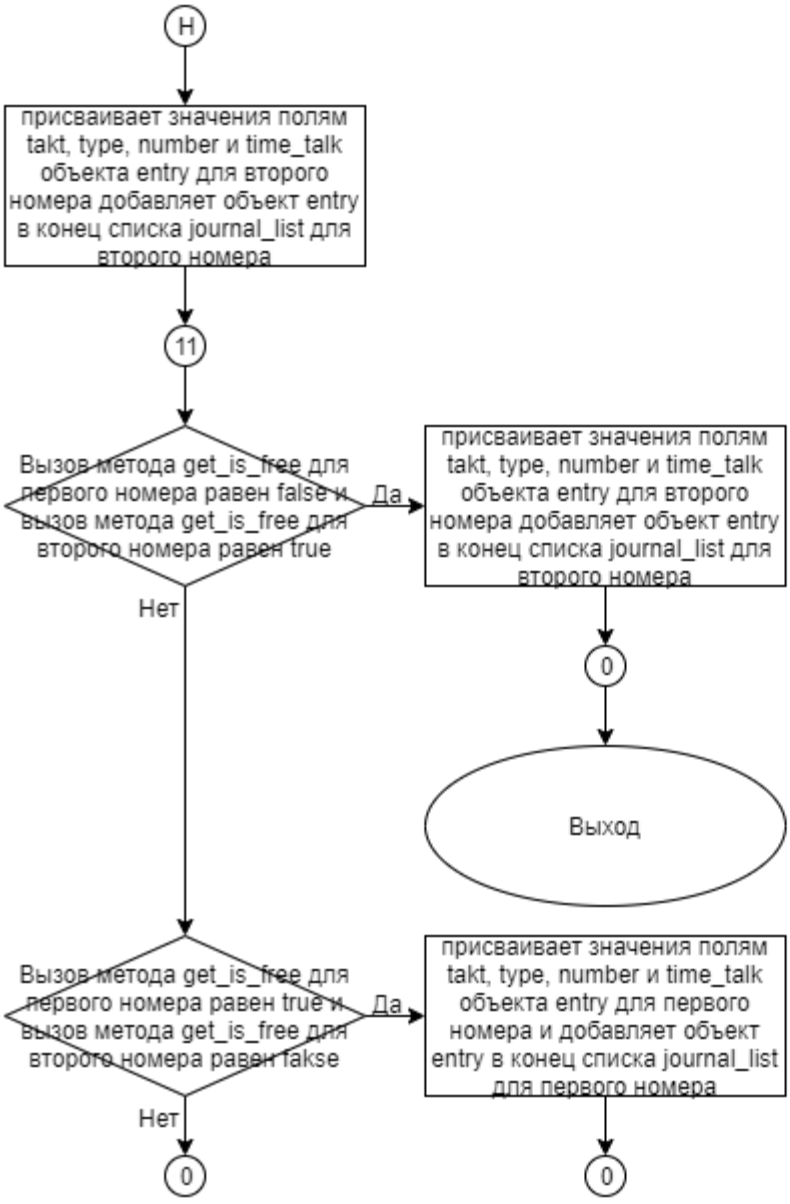

Рисунок 20 – Блок-схема алгоритма

Рисунок 21 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл ATC.cpp

Листинг 1 – ATC.cpp

```
#include "ATC.h"
#include "reader_commands.h"
#include "control_panel.h"
#include "prints.h"
#include "phones.h"
#include "cctype"
#include "algorithm"


ATC::ATC(cl_base* head_object) : cl_base(head_object, "ATC") {}

void ATC::build_tree_objects() {
   this->set_name("ATC");
   cl_base* p_sub = this;
   p_sub = new reader_commands(this, "read");
   p_sub = new control_panel(this, "pult");
   p_sub = new prints(this, "prints");
   p_sub = new phones(this, "phones");
   setFullReadiness();
   string s_msg;
   c_cmd = "";
   this->set_connection(SIGNAL_D(ATC::signal_reader_to_all),            this,
HANDLER_D(ATC::handler_input_data));
   while (c_cmd != "End of phones") {
      emit_signal(SIGNAL_D(ATC::signal_reader_to_all), c_cmd);
   }
   this->set_connection(SIGNAL_D(reader_commands::signal_tact),
get_sub_object("read"), HANDLER_D(reader_commands::handler_input_commands));
   get_sub_object("read")-
>set_connection(SIGNAL_D(reader_commands::signal_tact),
get_sub_object("read"), HANDLER_D(reader_commands::handler_input_commands));
   get_sub_object("read")-
>set_connection(SIGNAL_D(reader_commands::signal_tact),
get_sub_object("pult"), HANDLER_D(control_panel::handler_commands));
   get_sub_object("read")-
>set_connection(SIGNAL_D(reader_commands::signal_tact),
get_sub_object("prints"), HANDLER_D(prints::handler_prints));
   get_sub_object("pult")-
>set_connection(SIGNAL_D(control_panel::signal_commands),
```

```
get_sub_object("pult"), HANDLER_D(control_panel::handler_commands));
   get_sub_object("prints")->set_connection(SIGNAL_D(prints::signal_prints),
get_sub_object("prints"), HANDLER_D(prints::handler_prints));
   get_sub_object("phones")->set_connection(SIGNAL_D(phones::signal_phone),
get_sub_object("phones"), HANDLER_D(phones::handler_phone));
}

int ATC::exec_app() {
   string command = "";
   cout << "Ready to work\n";
   while (c_cmd != "SHOWTREE" && c_cmd != "Turn off the system")
   {
      emit_signal(SIGNAL_D(reader_commands::signal_tact), command);
   }
   cout << "Turn off the ATM" << endl;
   return 0;
}


void ATC::signal_reader_to_all(string& msg) {
   getline(cin, msg);
}

void ATC::handler_input_data(string msg)
{
   if (msg.size() == 7  && !this->get_sub_object("pult")->search_object(msg))
{
      control_panel*  obj  =  new  control_panel(this->get_sub_object("pult"),
msg);
      obj->set_connection(SIGNAL_D(prints::signal_prints),               obj,
HANDLER_D(prints::handler_prints));
   }
}
```

5.2 Файл ATC.h

Листинг 2 – ATC.h

```
#ifndef __ATC__H
#define __ATC__H

#include "cl_base.h"

class ATC : public cl_base {
public:
   ATC(cl_base* head_object);
   void build_tree_objects();
   int exec_app();

   void handler_input_data(string msg);
```

```
   void signal_reader_to_all(string& msg);
};


#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"
#include "ATC.h"
#include "phones.h"
#include "reader_commands.h"
#include "control_panel.h"
#include "prints.h"
cl_base::cl_base(cl_base* head_object, string s_name) {
   this->s_name = s_name;
   this->p_head_object = head_object;
   if (p_head_object != nullptr)
      p_head_object->sub_objects.push_back(this);
}

int cl_base::tact = 0;

cl_base::~cl_base() {
   for (int i = 0; i < this->sub_objects.size(); i++) {
      delete sub_objects[i];
   }
}
bool cl_base::set_name(string s_new_name) {
   if (p_head_object != nullptr) {
      for (int i = 0; i < p_head_object->sub_objects.size(); i++) {
         if (p_head_object->sub_objects[i]->get_name() == get_name())
            return false;
      }
   }
   this->s_name = s_new_name;
   return true;
}
string cl_base::get_name() {
   return s_name;
}
bool cl_base::can_be_ready() {
   cl_base* root = this;
   while (root->p_head_object) {
      if (root->p_head_object->state) {
         root = root->p_head_object;
      }
      else {
```

```
            return false;
         }
      }
      return true;
   }
   cl_base* cl_base::get_head_object() {
      return this->p_head_object;
   }
   cl_base* cl_base::get_sub_object(string s_name) {
      for (int i = 0; i < sub_objects.size(); i++) {
         if (sub_objects[i]->get_name() == s_name)
            return this->sub_objects[i];
      }
      for (int i = 0; i < sub_objects.size(); i++)
      {
         if (sub_objects[i]->get_sub_object(s_name)->get_name() == s_name)
            return sub_objects[i]->get_sub_object(s_name);
      }
      return this;
   }
   int cl_base::count(string s_name)
   {
      int counter = 0;
      if (this->get_name() == s_name)
         counter++;
      for (auto p_sub_object : sub_objects)
         counter += p_sub_object->count(s_name);
      return counter;
   }
   cl_base* cl_base::find_object_from_current(string s_name) {
      if (this->count(s_name) != 1) {
         return nullptr;
      }
      return search_object(s_name);
   }
   cl_base* cl_base::search_object(string s_name)
   {
      if (this->count(s_name) != 1)
         return nullptr;
      if (this->get_name() == s_name)
         return this;
      for (auto p_sub_object : sub_objects)
      {
         cl_base* p_found = p_sub_object->search_object(s_name);
         if (p_found != nullptr)
         {
            return p_found;
         }
      }
      return nullptr;
   }
   cl_base* cl_base::find_object_from_root(string name) {
      if (true) {
         if (name == s_name)
            return this;
```

```
            for (auto& el : sub_objects) {
               if (el->get_name() == s_name)
                  return el;
               cl_base* ans = el->find_object_from_root(name);
               if (ans)
                  return ans;
            }
         }
         else {
            for (auto& el : sub_objects) {
               if (el->get_name() == name) return el;
            }
         }
         return nullptr;
      }
      void cl_base::set_state(int state) {
         if (state == 0) {
            this->state = 0;
            for (int i = 0; i < sub_objects.size(); i++) {
               sub_objects[i]->set_state(0);
            }
            return;
         }
         if (this->p_head_object == nullptr || this->p_head_object->state != 0) {
            this->state = state;
         }
      }
      void cl_base::print_tree() {
         static int level = 0;
         for (int i = 0; i < level; i++) {
            cout << " ";
         }
         cout << s_name << endl;
         level++;
         for (auto sub_object : sub_objects) {
            sub_object->print_tree();
         }
         level--;
      }
      void cl_base::print_status() {
         static int level = 0;
         for (int i = 0; i < level; i++) {
            cout << " ";
         }
         if (state) {
            cout << s_name << " is ready" << endl;
         }
         else {
            cout << s_name << " is not ready" << endl;
         }
         level++;
         for (auto sub_object : sub_objects) {
            sub_object->print_status();
         }
         level--;
```

```
}
bool cl_base::delete_sub_object(string s_name) {
   if (this->get_name() == s_name)
      return false;
   for (int i = 0; i < this->sub_objects.size(); i++) {
      if (this->sub_objects[i]->get_name() == s_name) {
         delete this->sub_objects[i];
         this->sub_objects.erase(this->sub_objects.begin() + i);
         return true;
      }
   }
   return false;
}

cl_base* cl_base::get_root() {
   cl_base* p_root = this;
   while (p_root->get_head_object() != nullptr) {
      p_root = p_root->get_head_object();
   }
   return p_root;
}
cl_base* cl_base::find_obj_by_coord(string s_coord) {
   string s_nam;
   int i_slash_2;
   cl_base* p_obj = nullptr;
   if (s_coord == "") {
      return nullptr;
   }
   if (s_coord == "/") {
      return this->get_root();
   }
   if (s_coord == ".") {
      return this;
   }
   if (s_coord[0] == '/' && s_coord[1] == '/') {
      s_nam = s_coord.substr(2);
      return this->find_object_from_root(s_nam);
   }
   if (s_coord[0] == '.') {
      s_nam = s_coord.substr(1);
      return this->find_object_from_current(s_nam);
   }
   i_slash_2 = s_coord.find("/", 1);
   if (s_coord[0] == '/') {
      if (i_slash_2 != -1) {
         s_nam = s_coord.substr(1, i_slash_2 - 1);
         p_obj = this->get_root()->get_sub_object(s_nam);
         if (p_obj != nullptr) {
            return p_obj->find_obj_by_coord(s_coord.substr(i_slash_2 + 1));
         }
         else {
            return p_obj;
         }
      }
      else {
```

```
            s_nam = s_coord.substr(1);
            return this->get_root()->get_sub_object(s_nam);
         }
      }
      else {
         if (i_slash_2 != -1) {
            s_nam = s_coord.substr(0, i_slash_2);
            p_obj = this->get_sub_object(s_nam);
            if (p_obj != nullptr) {
               return p_obj->find_obj_by_coord(s_coord.substr(i_slash_2 + 1));
            }
            else {
               return p_obj;
            }
         }
         else {
            s_nam = s_coord;
            return this->get_sub_object(s_nam);
         }
      }
   }

   void   cl_base::set_connection(TYPE_SIGNAL   p_signal,   cl_base*   p_target,
   TYPE_HANDLER
      p_handler) {
      o_sh* p_value;
      for (int i = 0; i < connects.size(); i++) {
         if (connects[i]->p_signal == p_signal &&
            connects[i]->p_target == p_target &&
            connects[i]->p_handler == p_handler)
            return;
      }
      p_value = new o_sh();
      p_value->p_signal = p_signal;
      p_value->p_target = p_target;
      p_value->p_handler = p_handler;
      connects.push_back(p_value);
   }
   void  cl_base::delete_connection(TYPE_SIGNAL  p_signal,  cl_base*  p_target,
   TYPE_HANDLER p_handler) {
      vector<o_sh*>::iterator p_it;
      for( p_it = connects.begin(); p_it != connects.end(); p_it++){
         if( (*p_it)->p_signal == p_signal &&
            (*p_it)->p_target == p_target &&
            (*p_it)->p_handler == p_handler)
         {
            delete *p_it;
            p_it = connects.erase(p_it);
            p_it--;
         }
      }
   }
   string cl_base::get_absolute_way() {
      vector<string> v;
      string way = "";
```

```
        cl_base* obj = this;
        while (obj->p_head_object) {
           v.push_back(obj->s_name);
           obj = obj->p_head_object;
        }
        for (int i = v.size() - 1; i >= 0; i--) {
           way = way + "/";
           way = way + v[i];
        }
        if (way.empty()) way = "/";
        return way;
     }
     void cl_base::emit_signal(TYPE_SIGNAL p_signal, string& msg) {

        if (this->state == 0)
           return;
        TYPE_HANDLER p_handler;
        cl_base* obj;
        (this->*p_signal)(msg);
        if (this->flag) {
           for (int i = 0; i < connects.size(); i++) {
              if (connects[i]->p_signal == p_signal) {
                 p_handler = connects[i]->p_handler;
                 obj = connects[i]->p_target;
                 if (obj->state != 0) {
                    (obj->*p_handler)(msg);
                 }
              }
           }
        }
     }


     int cl_base::get_ready() {
        return state;
     }
     void cl_base::setFullReadiness() {
        this->set_state(1);
        for (auto sub : sub_objects) {
           sub->setFullReadiness();
        }
     }


     void cl_base::set_count_talk(int count_talk) {
        this->count_talk = count_talk;
     }

     void cl_base::set_sum_calls(int sum_calls) {
        this->sum_calls = sum_calls;
     }

     void cl_base::set_count_call(int count_call) {
        this->count_call = count_call;
     }
```

```
void cl_base::set_is_free(bool is_free) {
   this->is_free = is_free;
}

int cl_base::get_count_talk() {
   return count_talk;
}

int cl_base::get_sum_calls() {
   return sum_calls;
}

int cl_base::get_count_call() {
   return count_call;
}

bool cl_base::get_is_free() {
   return is_free;
}
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
#ifndef __cl_base_H
#define __cl_base_H
#include <vector>
#include <string>
#include <algorithm>
#include <iostream>
#include <queue>
using namespace std;
#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HANDLER_D(hendler_f) (TYPE_HANDLER) (&hendler_f)
class cl_base;
typedef void (cl_base::* TYPE_SIGNAL)(string& msg);
typedef void (cl_base::* TYPE_HANDLER)(string msg);

struct journal
{
   int takt = 0;
   string type;
   string number;
   int time_talk = 0;
};
struct o_sh {
   TYPE_SIGNAL p_signal;
   cl_base* p_target;
   TYPE_HANDLER p_handler;
```

```
};

class cl_base {
protected:
   string s_name;
   int state = 1;
   cl_base* p_head_object;
   vector <cl_base*> sub_objects;
   vector <o_sh*> connects;
   bool flag = true;
   int count_call,sum_calls;
   int count_talk;
   bool is_free = true;
   string c_cmd;
   vector<journal> journal_list;
   static int tact;
public:
   cl_base();
   cl_base(cl_base* head_object, string s_name = "Base object");
   ~cl_base();
   bool set_name(string new_name);
   string get_name();
   cl_base* get_head_object();
   cl_base* get_sub_object(string s_name);
   void print_tree();
   int count(string s_name);
   cl_base* search_object(string s_name);
   cl_base* find_object_from_root(string s_name);
   bool can_be_ready();
   int get_ready();
   void set_state(int state);
   void print_status();
   bool set_head_object(cl_base* now);
   bool delete_sub_object(string find_name);
   cl_base* find_object_from_current(string s_name);
   cl_base * search_object_from_path(string s_path);
   string get_absolute_way();
   cl_base* find_obj_by_coord(string s_coord);
   cl_base* get_root();
   void set_connection(TYPE_SIGNAL p_signal, cl_base* p_target, TYPE_HANDLER
p_handler);
   void    delete_connection(TYPE_SIGNAL    p_signal,    cl_base*    p_target,
TYPE_HANDLER p_handler);
   void emit_signal(TYPE_SIGNAL p_signal, string& msg);
   TYPE_SIGNAL get_signal(cl_base* object);
   TYPE_HANDLER get_handler(cl_base* object);
   int number = 1;
   void setFullReadiness();


   friend class reader_commands;
   friend class control_panel;
   friend class prints;
   friend class phones;
```

```
        void set_count_call(int count_call);
        void set_sum_calls(int sum_calls);
        void set_count_talk(int count_talk);
        void set_is_free(bool is_free);

        bool get_is_free();
        int get_count_call();
        int get_sum_calls();
        int get_count_talk();
    };
    #endif //UNTITLED9_CL_BASE_H
```

5.5 Файл control_panel.cpp

Листинг 5 – control_panel.cpp

```
    #include "control_panel.h"
    #include "prints.h"
    #include <sstream>

    control_panel::control_panel(cl_base*           head_object,           string
    s_name) :cl_base(head_object, s_name) {
        this->set_sum_calls(0);
        this->set_count_talk(0);
        this->set_count_call(0);
        this->set_is_free(true);
    }

    void control_panel::signal_commands(string& msg) {

    }

    void control_panel::handler_commands(string msg) {
        string command = msg;
        if (command.substr(0, 12) == "Call request") {
            size_t firstSpacePos = command.find(' ');
            size_t secondSpace = command.find(' ', firstSpacePos + 1);
            string subString = command.substr(secondSpace + 1);
            istringstream iss(subString);
            string first, second;
            int time;
            iss >> first >> second >> time;
            int i = 0;
            if(second.size() != 7){
                msg = "The subscriber's number was dialed incorrectly: " + second;
                cout << msg << endl;
                return;
            }
            else{
                while(i != 7){
```

```
            if(second[i] < '1' || second[i] > '9'){
               msg = "The subscriber's number was dialed incorrectly: " +
second;
               cout << msg << endl;
               return;
            }
         ++i;
         }
      }
      cl_base* obj_1 = find_object_from_root(first);
      cl_base* obj_2 = find_object_from_root(second);
      journal ready;
      if (obj_1->get_is_free()) {
         int count_call_1 = obj_1->get_count_call() + 1;
         obj_1->set_count_call(count_call_1);
         if (obj_2 != nullptr) {
            if (obj_2->get_is_free()) {
               int sum_call_1 = obj_1->get_sum_calls() + 1;
               obj_1->set_sum_calls(sum_call_1);
               int count_talk_2 = obj_2->get_count_talk() + 1;
               obj_2->set_count_talk(count_talk_2);
               obj_1->set_is_free(false);
               obj_2->set_is_free(false);
               cout << "Call from " << first << " to " << second << ", talk:
" << time << endl;
               p_head_object->get_sub_object("prints")-
>emit_signal(SIGNAL_D(prints::signal_prints), msg);
            }
            else {
               p_head_object->get_sub_object("prints")-
>emit_signal(SIGNAL_D(prints::signal_prints), msg);
               msg = "Subscriber " + second + " is busy";
               cout << msg << endl;
               return;
            }
         }
         else {
            p_head_object->emit_signal(SIGNAL_D(prints::signal_prints), msg);
            msg = "Subscriber " + second + " not found";
            cout << msg << endl;
            return;
         }
      }
      else {
         p_head_object->emit_signal(SIGNAL_D(prints::signal_prints), msg);
         msg = "The phone is busy, a new call is not possible: " + first;
         cout << msg << endl;
         return;
      }
   }
}
```

5.6 Файл control_panel.h

Листинг 6 – control_panel.h

```
#ifndef __CONTROL_PANEL__H
#define __CONTROL_PANEL__H

#include "cl_base.h"
#include "set"

class control_panel : public cl_base {
public:
   control_panel(cl_base* p_head_object, string s_name);
   void signal_commands(string& msg);
   void handler_commands(string msg);
};

#endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "ATC.h"
int main() {
   ATC object(nullptr); // создание объекта приложение
   object.build_tree_objects(); // конструирование системы,
   return object.exec_app(); // запуск системы
}
```

5.8 Файл phones.cpp

Листинг 8 – phones.cpp

```
#include "phones.h"
#include <sstream>
phones::phones(cl_base*  head_object,  string  s_name)  :cl_base(head_object,
s_name) {}


void phones::signal_phone(string& msg) {
```

```
}
void phones::handler_phone(string msg) {
   if (msg.substr(0, 12) == "Call request") {
      size_t firstSpacePos = msg.find(' ');
      size_t secondSpace = msg.find(' ', firstSpacePos + 1);
      string subString = msg.substr(secondSpace + 1);
      istringstream iss(subString);
      string first, second;
      int time;
      iss >> first >> second >> time;

      journal entry;
      if (p_head_object->find_object_from_root(first)->get_is_free() == false
&& p_head_object->find_object_from_root(second)->get_is_free() == false) {
         entry.takt = this->tact;
         entry.type = "Call";
         entry.number = second;
         entry.time_talk = time;
         p_head_object->find_object_from_root(first)-
>journal_list.push_back(entry);
         entry.takt = this->tact;
         entry.type = "Bell";
         entry.number = first;
         entry.time_talk = time;
         p_head_object->find_object_from_root(second)-
>journal_list.push_back(entry);
      }
      else if (p_head_object->find_object_from_root(first)->get_is_free() ==
false   &&   p_head_object->find_object_from_root(second)->get_is_free()   ==
true)
      {
         entry.takt = this->tact;
         entry.type = "Bell";
         entry.number = first;
         entry.time_talk = 0;
         p_head_object->find_object_from_root(second)-
>journal_list.push_back(entry);
      }
      else if (p_head_object->find_object_from_root(first)->get_is_free() ==
true   &&   p_head_object->find_object_from_root(second)->get_is_free()   ==
false)
      {
         entry.takt = this->tact;
         entry.type = "Call";
         entry.number = second;
         entry.time_talk = 0;
         p_head_object->find_object_from_root(first)-
>journal_list.push_back(entry);
      }
   }
}
```

5.9 Файл phones.h

Листинг 9 – phones.h

```
#ifndef __PHONES__H
#define __PHONES__H

#include "cl_base.h"

class phones : public cl_base {
public:
   phones(cl_base* p_head_object, string s_name);
   void signal_phone(string& msg);
   void handler_phone(string msg);
};

#endif
```

5.10 Файл prints.cpp

Листинг 10 – prints.cpp

```
#include "prints.h"
#include "phones.h"
#include <sstream>
prints::prints(cl_base*  head_object,  string  s_name)  :cl_base(head_object,
s_name) {}

void prints::signal_prints(string& msg) {
   if (msg.substr(0, 25) == "Display phone information") {
      cout << "Phone log:" << msg.substr(25) << endl;
   }
   else if (msg.substr(0, 25) == "Display the system status") {
      cout << "ATC" << endl;
   }
}

void prints::handler_prints(string msg) {
   if (msg.substr(0, 12) == "Call request") {
      p_head_object->get_sub_object("phones")-
>emit_signal(SIGNAL_D(phones::signal_phone), msg);
   }
   else if (msg.substr(0, 25) == "Display phone information") {
      string number = msg.substr(26);
      cl_base*       obj       =       p_head_object->get_sub_object("pult")-
>get_sub_object(number);
      for (auto stat : obj->journal_list)
      {
         cout << stat.takt << " " << stat.type << " " << stat.number << " "
```

```
<< stat.time_talk << endl;
      }
   }
   else if (msg.substr(0, 25) == "Display the system status") {
      cl_base* obg = p_head_object->get_sub_object("pult");
      for (auto subs : obg->sub_objects)
      {
         cout  <<  "telephone  "  <<  subs->get_name()  <<  "  "  <<  subs-
>get_count_call()  <<  "  "  <<  subs->get_sum_calls()  <<  "  "  <<  subs-
>get_count_talk() << endl;
      }
   }
}
```

5.11 Файл prints.h

Листинг 11 – prints.h

```
#ifndef __PRINTS__H
#define __PRINTS__H

#include "cl_base.h"

class prints : public cl_base {
public:
   prints(cl_base* p_head_object, string s_name);
   void signal_prints(string& msg);
   void handler_prints(string msg);
};

#endif
```

5.12 Файл reader_commands.cpp

Листинг 12 – reader_commands.cpp

```
#include "reader_commands.h"
#include "control_panel.h"
#include "prints.h"

reader_commands::reader_commands(cl_base*         head_object,         string
s_name) :cl_base(head_object, s_name) {}


void reader_commands::signal_tact(string& msg) {
```

```
   getline(cin, msg);
   if (msg == "Turn off the system")
   {
      c_cmd = "Turn off the system";
      return;
   }
   if (msg == "SHOWTREE") {
      c_cmd = "SHOWTREE";
      msg = "Display the system status";
      return;
   }
   for (auto sub : get_sub_object("pult")->sub_objects)
   {
      if            (!sub->journal_list.empty()            &&            sub-
>journal_list[journal_list.size()].takt       *       30       +       sub-
>journal_list[journal_list.size()].time_talk - 30 * tact < 0)
      {
         sub->set_is_free(true);
      }
   }
}

void reader_commands::handler_input_commands(string msg) {
   tact++;
   if (msg.substr(0, 12) == "Call request") {
      p_head_object->get_sub_object("pult")-
>emit_signal(SIGNAL_D(control_panel::signal_commands), msg);
   }
   else if (msg.substr(0, 25) == "Display phone information") {
      p_head_object->get_sub_object("prints")-
>emit_signal(SIGNAL_D(prints::signal_prints), msg);
   }
   else if (msg == "Display the system status") {
      p_head_object->get_sub_object("prints")-
>emit_signal(SIGNAL_D(prints::signal_prints), msg);
   }
}
```

5.13 Файл reader_commands.h

Листинг 13 – reader_commands.h

```
#ifndef __READER_COMMANDS__H
#define __READER_COMMANDS__H

#include "cl_base.h"

class reader_commands : public cl_base {
public:
   reader_commands(cl_base* p_head_object, string s_name);
```

```
        void signal_tact(string&);
        void handler_input_commands(string msg);
    };

    #endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 28.

Таблица 28 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
1111111 1111112 1111113 1111114 1111115 1111116 1111117 1111118 End of phones Display the system status Call request 1111115 1111117 55 Call request 1111116 1111117 15 Display the system status Display phone information 1111116 Display phone information 1111115 Display phone information 1111117 Call request 1111115 1111117 55 Display the system status Turn off the system	Ready to work ATC telephone 1111111 0 0 0 telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 0 0 0 telephone 1111116 0 0 0 telephone 1111117 0 0 0 telephone 1111118 0 0 0 Call from 1111115 to 1111117, talk: 55 Subscriber 1111117 is busy ATC telephone 1111111 0 0 0 telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 1 1 0 telephone 1111116 1 0 0 telephone 1111117 0 0 1 telephone 1111118 0 0 0 Phone log: 1111116 3 Call 1111117 0 Phone log: 1111115 2 Call 1111117 55 Phone log: 1111117 2 Bell 1111115 55 Call from 1111115 to	Ready to work ATC telephone 1111111 0 0 0 telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 0 0 0 telephone 1111116 0 0 0 telephone 1111117 0 0 0 telephone 1111118 0 0 0 Call from 1111115 to 1111117, talk: 55 Subscriber 1111117 is busy ATC telephone 1111111 0 0 0 telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 1 1 0 telephone 1111116 1 0 0 telephone 1111117 0 0 1 telephone 1111118 0 0 0 Phone log: 1111116 3 Call 1111117 0 Phone log: 1111115 2 Call 1111117 55 Phone log: 1111117 2 Bell 1111115 55 Call from 1111115 to

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	1111117, talk: 55 ATC telephone 1111111 0 0 0 telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 2 2 0 telephone 1111116 1 0 0 telephone 1111117 0 0 2 telephone 1111118 0 0 0 Turn off the ATM	1111117, talk: 55 ATC telephone 1111111 0 0 0 telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 2 2 0 telephone 1111116 1 0 0 telephone 1111117 0 0 2 telephone 1111118 0 0 0 Turn off the ATM
1111112 1111113 1111114 1111115 1111116 1111117 End of phones Display the system status Call request 1111115 1111117 55 Display the system status Display phone information 1111116 Display phone information 1111115 Display phone information 1111117 Call request 1111115 1111117 55 Display the system status Turn off the system	Ready to work ATC telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 0 0 0 telephone 1111116 0 0 0 telephone 1111117 0 0 0 Call from 1111115 to 1111117, talk: 55 ATC telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 1 1 0 telephone 1111116 0 0 0 telephone 1111117 0 0 1 Phone log: 1111116 Phone log: 1111115 2 Call 1111117 55	Ready to work ATC telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 0 0 0 telephone 1111116 0 0 0 telephone 1111117 0 0 0 Call from 1111115 to 1111117, talk: 55 ATC telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 1 1 0 telephone 1111116 0 0 0 telephone 1111117 0 0 1 Phone log: 1111116 Phone log: 1111115 2 Call 1111117 55

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	Phone log: 1111117 2 Bell 1111115 55 Call from 1111115 to 1111117, talk: 55 ATC telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 2 2 0 telephone 1111116 0 0 0 telephone 1111117 0 0 2 Turn off the ATM	Phone log: 1111117 2 Bell 1111115 55 Call from 1111115 to 1111117, talk: 55 ATC telephone 1111112 0 0 0 telephone 1111113 0 0 0 telephone 1111114 0 0 0 telephone 1111115 2 2 0 telephone 1111116 0 0 0 telephone 1111117 0 0 2 Turn off the ATM
1111111 End of phones Call request 1111111 s123456 SHOWTREE	Ready to work The subscriber's number was dialed incorrectly: s123456 ATC telephone 1111111 0 0 0 Turn off the ATM	Ready to work The subscriber's number was dialed incorrectly: s123456 ATC telephone 1111111 0 0 0 Turn off the ATM

ЗАКЛЮЧЕНИЕ

Курсовая работу на тему "Моделирование работы телефона" выполнена. Отчет составлен и готов быть продемонстрирован руководителю курсовой работы.

Была осуществлена разработка систему, реализующая работу телефона. В процессе реализации задачи было использовано 6 классов: cl_base; ATC; reader_commands; control_panel; prints; phones, каждая из которых имитирует определенную часть систему телефона. Все необходимые методы, сигналы и обработчики были реализованы. Функционал программы протестирован созданными нами тестами.

В ходе выполнения работы были усвоены навыки объектно-ориентированного программирования и разработки на языке C++. Получен опыт применения сигналов и обработчиков, составления документации, описание блок-схем и алгоритмов.

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).