Здесь будет титульник, листай ниже

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].

ООП(Объектно-ориентированное программирование) - это парадигма разработки в программировании, в которой основной концепцией является понятие объекта, который отождествляется с предметной областью. Объект - отдельный представитель класса, имеющий конкретное поведение и состояние, полностью определяемое классом. Объект имеет конкретные значения атрибутов и методы , которые работают с этими значениями на основе заданных в классе правил.

Моделирование систем с использованием объектно-ориентированного программирования (ООП) является важной частью современной разработки программного обеспечения. Применение концепций ООП при моделировании систем позволяет создать более гибкие и модульные решения. это облегчает и ускоряет разработку продукта, а также последующую его поддержку.

В связи с этим, в данной работе была поставлена задача смоделировать работу банкомата с использованием концепций ООП.

1 ПОСТАНОВКА ЗАДАЧИ

Надо моделировать работу банкомата следующей конструкции. Банкомат состоит из следующих элементов:

- пульт управления;
- устройство идентификации банковской карты (номер карты + пин код);
- устройство приема денег;
- устройство выдачи денег.
- устройство отображения состояния банкомата и состояния выполняемой операции.

Кнопки пульта управления:

- цифровая клавиатура;
- кнопки выполнения операции;
- кнопка завершения операции.

Правила работы с банкоматом.

Банкомат готовиться к работе следующим образом:

1. Загружается информация относительно зарегистрированных карт и имеющих на них средств.
2. Загружается заданное количество купюр для выдачи денег.
3. Выводится сообщение о готовности банкомата к работе.

После готовности банкомата выполняются действия:

- приложить карту к устройству идентификации;
- распознавание карты;
- запрос, ввод и подтверждение ПИН-кода;
- выбор команды;
- вносить можно купюры кратные 100 р.;
- при внесении денег осуществляется суммирование;

- если для снятия запрошенного количества денег средств на карте или в банкомате недостаточно, сообщается о недостаточности средств.

Вносить можно купюры с достоинством 5000, 2000, 1000, 500 и 100 рублей в неограниченном количестве.

При снятии денег выдаются купюры с достоинством 5000, 2000, 1000, 500 и 100 рублей. Снятие денег осуществляется в режиме «Выдать крупными».

Идентификация карты, нажатия на кнопки пульта управления, внесение или снятие денег моделируется посредством клавиатурного ввода.

Ввод делится на команды и их восприятие зависит от состояния банкомата последовательности выдачи. Шаблоны команд:

```
Identification «номер карты»
```

Команда выдается при состоянии банкомата «Готов к работе». Номер карты задается четырьмя группами из 4 цифр: xxxx xxxx xxxx xxxx

```
PIN-code «четыре цифры»
```

Команда может быть выдана только после подтверждения корректности номера карты и запроса «Enter the PIN code»

```
Deposit money «значение купюры»
```

Команды могут быть выданы последовательно, тогда их значения суммируются и выдается сообщение:

```
The amount: «введенная сумма»
```

«значение купюры» может равняться 5000, 2000, 1000, 500, 100. Купюры других достоинств не принимаются. После этих команд надо ввести команду подтверждения:

```
Deposit money to the card
```

После выдается сообщение:

```
Card balance «сумма на карте»
```

Если введена другая команда, то банкомат внесенные купюры возвращает.

```
Withdraw money «сумма»
```

По данной команде выдаются деньги в режиме «Выдать крупными». «сумма» должна быть кратна 100. Если денег на карте достаточно и купюр в банкомате тоже, деньги выдаются согласно сообщению:

```
Take the money: 5000 * «количество купюр» rub., 2000 * «количество купюр» rub., 1000 * «количество купюр» rub., 500 * «количество купюр» rub., 100 * «количество купюр» rub.
```

Противном случает выдается сообщение:

```
The amount is not a multiple of 100
```

Если запрошенная сумма не кратна 100.

```
There is not enough money on the card
```

Если на карте недостаточно средств.

```
There is not enough money in the ATM
```

Если в банкомате недостаточно средств или из наличных купюр нельзя собрать запрошенную сумму.

```
End the session
```

Команда завершения сеанса. Последняя не завершенная команда отменяется и банкомат переходит в состоянии готовности. Выдается сообщение Ready to work

```
Turn off the ATM
```

Команда выключения банкомата. Последняя не завершенная команда отменяется и банкомат выключается. Эта команда всегда присутствует.

Отображение текста состояния банкомата и результата операции моделируется посредством вывода на консоли.

Построить систему, которая использует объекты:

1. Объект «система».
2. Объект для чтения команд и данных. Считывает данные для подготовки и

настройки банкомата. После чтения очередной порции данных для настройки или данных команды, объект выдает сигнал с текстом полученных данных. Все данные настройки и данные команды синтаксический корректны.

3. Объект пульта управления, для отработки нажатия кнопок (команд). Объект после нажатия кнопки анализирует возможность его выполнения и выдает соответствующий сигнал. Выдает сигнал если запрошенная для снятия сумма не кратна 100 или на карте нет достаточных средств.

4. Объект, моделирующий устройства идентификации банковской карты. Выдает сигнал, содержащий текст результата идентификации.

5. Объект, моделирующий устройства приема денег. После принятия очередной купюры сохраняет его. При внесении денег подряд производит суммирование. Выдает сигнал, содержащий сумму введенных денег для отображения на экран.

6. Объект, моделирующий устройства выдачи денег. Выдает сигнал, содержащий количество купюр возвращаемой суммы или информацию о невозможности получить необходимый состав купюр.

7. Объект для вывода состояния или результата операции банкомата на консоль.

Написать программу, реализующую следующий алгоритм:

1. Вызов метода объекта «система» build_tree_objects ( ).
    1.1. Построение дерева иерархии объектов.
    1.2. Установка связей сигналов и обработчиков между объектами.
2. Вызов метода объекта «система» exec_app ( ).
    2.1. Приведение всех объектов в состояние готовности.
    2.2. Цикл для обработки вводимых данных для загрузки банкомата.
        2.2.1. Выдача сигнала объекту для ввода данных.

2.2.2. Отработка операции загрузки.

2.3. Цикл для обработки вводимых команд банкомата.

2.3.1. Выдача сигнала объекту для ввода команды.

2.3.2. Отработка команды.

2.4. После ввода команды «Turn off the ATM» завершить работу.

Все приведенные сигналы и соответствующие обработчики должны быть реализованы.

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 программы.

1.1 Описание входных данных

Ввод информации о картах по последовательности первых строк:

```
«номер карты» «ПИН-код» «сумма на карте»
```

Ввод информации о картах завершается строкой:

```
End card loading
```

Следующая строка содержит количество загружаемых купюр разного достоинства в последовательности: 5000, 2000, 1000, 500 и 100 рублевых:

```
«количество купюр» «количество купюр» «количество купюр» «количество купюр» «количество купюр»
```

Далее следуют команды.

Последней командой является:

```
Turn off the ATM
```

Пример ввода:

```
1001 2022 0021 0001 5001 15000
1001 2022 0021 0002 5002 25000
```

```
1001 2022 0021 0003 5003 35000
1001 2022 0021 0004 5004 45000
1001 2022 0021 0005 5005 55000
End card loading
5000 2000 1000 500 100
End money loading
Deposit money 1000
Identification 1001 2022 0021 0004
PIN-code 5004
Deposit money 1000
Deposit money 1000
Deposit money 50
Deposit money to the card
Withdraw money 10000
Deposit money 1000
Deposit money to the card
End the session
Identification 1001 2022 0021 0003
Deposit money 1000
Deposit money to the card
End the session
Deposit money 1000
Identification 1001 2022 0021 0001
PIN-code 5001
Withdraw money 8500
Deposit money 1500
Deposit money 500
Deposit money to the card
End the session
Turn off the ATM
```

1.2 Описание выходных данных

Шаблоны текстов, которые отображаются на консоли:

Готов к работе, отображается в начале работы системы, после завершения загрузки банкомата. Также отображается после завершения сеанса работы. Банкомат готов для обслуживания нового клиента т.е. идентификации карты. Другая команда игнорируется.

```
Ready to work
```

После идентификации карты выдается сообщение:

```
Select the command
```

Сообщение о величине внесенной суммы после ввода очередной купюры.

```
The amount: «сумма денег»
```

Сообщение для получения денег:

```
Take the money: 5000 * «количество купюр» rub., 2000 * «количество купюр» rub., 1000 * «количество купюр» rub., 500 * «количество купюр» rub., 100 * «количество купюр» rub.
```

Сообщение баланса карты:

```
Card balance «сумма на карте»
```

Сообщение завершения работы банкомата:

```
Turn off the ATM
```

Пример вывода:

```
Ready to work
Ready to work
Enter the PIN code
Select the command
The amount: 1000
The amount: 2000
Card balance 47000
Take the money: 5000 * 2 rub., 2000 * 0 rub., 1000 * 0 rub., 500 * 0 rub., 100 * 0 rub.
The amount: 1000
Card balance 38000
Ready to work
Enter the PIN code
Ready to work
Ready to work
Ready to work
Ready to work
Enter the PIN code
Select the command
Take the money: 5000 * 1 rub., 2000 * 1 rub., 1000 * 1 rub., 500 * 1 rub., 100 * 0 rub.
The amount: 500
Card balance 7000
Ready to work
Turn off the ATM
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект ob_cl_application класса cl_application предназначен для ;
- объект класса CardIdentifier предназначен для ;
- объект класса cl_base предназначен для ;
- объект класса ControlPanel предназначен для ;
- объект класса Enter предназначен для ;
- объект класса Issue предназначен для ;
- объект класса Printer предназначен для ;
- объект класса Reader предназначен для ;
- функция getline для считывания строки из входного потока;
- cin - объект стандартного потока ввода;
- cout - объект стандартного потока вывода;
- if, else - условный оператор;
- do...while - оператор цикла с постусловием;
- for - оператор цикла со счетчиком;
- while - оператор цикла с предусловием;
- new - оператор для выделения памяти под объект;
- delete - оператор для удалаения выделенной памяти;
- switch-case - оператор множественного выбора;
- vector - базовый контейнер;
- stack - контейнер, реализующий структуру данных "стек";
- queue - контейнер последовательности;
- typedef - определение новых типов данных;
- struct - структура;
- #define - параметризованное макроопределение препроцессора;

- tYPE_SIGNAL p_signal - поле структуры o_sh, хранящее указатель на метод сигнала;
- cl_base* p_cl_base - поле структуры o_sh, хранящее указатель на целевой объект;
- tYPE_HANDLER p_handler - поле структуры o_sh, хранящее указатель на метод обработчика;
- string number - поле структуры card, хранящее номер карты;
- string PIN - поле структуры card, хранящее пин-код карты;
- int balance - поле структуры card, хранящее баланс карты.

Класс cl_base:

- свойства/поля:
    - поле наименование объекта:
        - наименование — name;
        - тип — string;
        - модификатор доступа — private;
    - поле указатель на головной объект:
        - наименование — p_head_object;
        - тип — cl_base*;
        - модификатор доступа — private;
    - поле вектор подчиненных объектов:
        - наименование — p_sub_objects;
        - тип — vector<cl_base*>;
        - модификатор доступа — private;
    - поле индикатор состояния объекта:
        - наименование — state;
        - тип — int;
        - модификатор доступа — private;

- поле номер класса:
  - наименование — class_num;
  - тип — int;
  - модификатор доступа — private;
- поле список соединений объекта:
  - наименование — connects;
  - тип — vector<o_sh*>;
  - модификатор доступа — private;

- функционал:
  - метод cl_base — параметризованный конструктор;
  - метод ~cl_base — деструктор;
  - метод change_name — метод установки имени;
  - метод get_name — метод получения имени;
  - метод get_head_object — метод получения указателя на родительский класс;
  - метод get_sub_object — метод получения указателя на дочерний объект;
  - метод search_by_name — метод поиска объекта на дереве иерархии по имени;
  - метод show_ready_tree — метод вывода иерархии объектов и отметок их готовности;
  - метод set_connect — установка соединения;
  - метод emit_signal — выдача сигнала;
  - метод get_state — метод для получения состояния объекта;
  - метод enable_all — метод для включения(активации) объекта и всех подчиненных объектов;
  - метод delete_connect — метод удаления(разрыва) связи между

сигналом текущего объекта и обработчиком целевого.

Класс cl_application:

- свойства/поля:
  - поле команда:
    - наименование — command;
    - тип — string;
    - модификатор доступа — private;
- функционал:
  - метод cl_application — параметризованный конструктор;
  - метод build_tree_objects — метод построения дерева иерархии объектов;
  - метод exec_app — запуск основного алгоритма программы;
  - метод command_request_signal — метод-сигнал для запроса команды;
  - метод command_obr — метод-обработчик для получения команды;
  - метод text_print_signal — метод-сигнал для запроса вывода текста.

Класс CardIdentifier:

- свойства/поля:
  - поле определение типа данных для STATUS, возможные значения - READINFO, MONEYLOADING, WAITING, PINWAIT, LOGGED:
    - наименование — STATUS;
    - тип — enum;
    - модификатор доступа — private;
  - поле текущий статус:
    - наименование — status;
    - тип — STATUS;
    - модификатор доступа — private;

- поле номер текущей карты:
  - наименование — curCard;
  - тип — int;
  - модификатор доступа — private;
- поле вектор с картами, объектами структуры card:
  - наименование — cards;
  - тип — vector<card>;
  - модификатор доступа — private;

- функционал:
  - метод CardIdentifier — параметризованный конструктор;
  - метод command_obr — метод-обработчик для получения и обработки поступившей команды;
  - метод set_status_signal — метод-сигнал для установки статуса;
  - метод balance_increasing_obr — метод-обработчик для увеличения баланса;
  - метод set_available_sum_signal — метод-сигнал для установки доступного баланса;
  - метод print_signal — метод-сигнал для запроса вывода текста;
  - метод balance_reduction_obr — метод-обработчик для уменьшения баланса.

Класс ControlPanel:

- свойства/поля:
  - поле определение типа данных для STATUS, возможные значения - WAITING, PIN, LOGGED:
    - наименование — STATUS;
    - тип — enum;
    - модификатор доступа — private;

- поле вставлены деньги или нет:
  - наименование — hasMoneyInserted;
  - тип — bool;
  - модификатор доступа — private;
- поле доступный баланс:
  - наименование — availableSum;
  - тип — int;
  - модификатор доступа — private;
- поле текуший статус:
  - наименование — status;
  - тип — STATUS;
  - модификатор доступа — private;

- функционал:
  - метод ControlPanel — параметризованный конструктор;
  - метод command_obr — метод-обработчик для получения и обработки поступившей команды;
  - метод set_status_obr — метод-обработчик для установки статуса;
  - метод add_bill_signal — метод-сигнал для внесения денег;
  - метод deposit_money_signal — метод-сигнал для окончательного внесения денег;
  - метод return_bills_signal — метод-сигнал для обнуления суммы;
  - метод text_print_signal — метод-сигнал для запроса вывода текста;
  - метод set_available_sum_obr — метод-обработчик для установки доступного баланса карты;
  - метод withdraw_signal — метод-сигнал для снятия денег.

Класс Enter:

- свойства/поля:

- поле сумма:
    - наименование — sum;
    - тип — int;
    - модификатор доступа — private;

- функционал:
    - метод Enter — параметризованный конструктор;
    - метод add_bill_obr — метод-обработчик для внесения денег;
    - метод sum_print_signal — метод-сигнал для вывода баланса карты;
    - метод deposit_money_obr — метод-обработчик для окончательного внесения денег;
    - метод add_money_signal — метод-сигнал для увеличения баланса карты;
    - метод return_money_obr — метод обработчик для установки sum в 0.

Класс Issue:

- свойства/поля:
    - поле определение типа данных для STATUS, возможные значения - WAITING, READING, WORKING:
        - наименование — STATUS;
        - тип — enum;
        - модификатор доступа — private;
    - поле вектор:
        - наименование — bills;
        - тип — vector<int>;
        - модификатор доступа — private;
    - поле текущий статус:
        - наименование — status;
        - тип — STATUS;

        - модификатор доступа — private;
- функционал:
    - метод Issue — параметризованный конструктор;
    - метод command_obr — метод-обработчик для получения и обработки поступившей команды;
    - метод withdraw_obr — метод-обработчик для проверки и вывода денег;
    - метод print_signal — метод-сигнал для вывода текста на экран;
    - метод balance_reduction_signal — метод-сигнал для уменьшения баланса.

Класс Printer:

- свойства/поля:
    - поле переменная для вывода символа новой строки:
        - наименование — first;
        - тип — bool;
        - модификатор доступа — private;
- функционал:
    - метод Printer — параметризованный конструктор;
    - метод print_obr — вывод текста на экран.

Класс Reader:

- функционал:
    - метод Reader — параметризованный конструктор;
    - метод read_obr — метод-сигнал для чтения команды;
    - метод read_signal — метод-обработчик для чтения команды.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base				
		cl_application	public		2
		CardIdentifier	public		3
		ControlPanel	public		4
		Enter	public		5
		Issue	public		6
		Printer	public		7
		Reader	public		8
2	cl_application				
3	CardIdentifier				
4	ControlPanel				
5	Enter				
6	Issue				
7	Printer				
8	Reader				

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 конструктора класса Printer

Функционал: Параметризованный конструктор.

Параметры: cl_base* parent - указатель на головной объект в дереве иерархии , string name - наименование сохдаваемого объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса Printer

№	Предикат	Действия	№ перехода
1		Вызов парметризованного конструктора класса cl_base с параметрами parent и name	∅

3.2 Алгоритм метода print_obr класса Printer

Функционал: вывод текста на экран.

Параметры: string message - сообщение для вывода на экран.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода print_obr класса Printer

№	Предикат	Действия	№ перехода
1	first == true	Вывод на экран значения переменной message	2
		Переход на новую строку и вывод на экран	2

№	Предикат	Действия	№ перехода
		значения переменной message	
2		Присваивание полю first значения false	∅

3.3 Алгоритм конструктора класса Reader

Функционал: параметризованный конструктор.

Параметры: cl_base* parent - указатель на головной объект в дереве иерархии , string name - наименование сохдаваемого объекта.

Алгоритм конструктора представлен в таблице 4.

Таблица 4 – Алгоритм конструктора класса Reader

№	Предикат	Действия	№ перехода
1		Вызов парметризованного конструктора класса cl_base с параметрами parent и name	∅

3.4 Алгоритм метода read_obr класса Reader

Функционал: метод-сигнал для чтения команды.

Параметры: string message - сообщение.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода read_obr класса Reader

№	Предикат	Действия	№ перехода
1		Вызов метода emit_signal(с передачей макроса SIGNAL_D(Reader::read_signal) и значения переменной message в качестве аргументов	∅

3.5 Алгоритм метода read_signal класса Reader

Функционал: метод-обработчик для чтения команды.

Параметры: string &command - ссылка на строку содержащую команду.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода read_signal класса Reader

№	Предикат	Действия	№ перехода
1		Вызов функции getline с передачей cin и command в качестве аргументов	∅

3.6 Алгоритм конструктора класса cl_base

Функционал: параметризованный конструктор.

Параметры: cl_base* p_head_object - указатель на головной объект в дереве иерархии , string name - наименование сохдаваемого объекта.

Алгоритм конструктора представлен в таблице 7.

Таблица 7 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Присвоение полю p_head_object текущего объекта значения аргумента p_head_object	2
2		Присвоение полю name текущего объекта значения аргумента name	3
3	p_head_object	Обращение к вектору p_sub_objects объекта указателя p_head_object и добавление в него указателя на текущий объект	∅
			∅

3.7 Алгоритм деструктора класса cl_base

Функционал: деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 8.

Таблица 8 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной счетчика i = 0	2
2	i < p_sub_objects.size()	Очищение памяти выделенной под объект указателя p_sub_objects[i] с помощью оператора delete Инкремент i	2
			3
3		Инициализация указателя p_root на объект класса cl_base со значением текущего объекта	4
4	p_root->get_head_object() != nullptr	Присвоение указателю p_root результата обращения к методу get_head_object объекта указателя p_root	4
		Объявление очереди q, что принисает указатели на объекты cl_base	5
5		Добавление p_root в очередь q	6
6	!q.empty()	Инициализация указателя p_front на объект класса cl_base значением первого элемента в очереди q	7
			11
7		Удаление первого элемента q	8
8		Инициализация целочисленной переменной счетчика i = 0	9
9	i< p_front->connects.size()	Обращение к методу delete_connect объекта	9

№	Предикат	Действия	№ перехода
		указателя p_front с передачей p_front->connects[i]->p_signal, p_front->connects[i]->p_cl_base, p_front->connects[i]->p_handler в качестве аргументов Инкремент i	
		Инициализация целочисленной переменной-счетчика i = 0	10
10	i<p_front->p_sub_objects.size()	Добавление p_sub_objects[i] объекта указателяр_front в очередь q Инкремент i	10
			6
11	!connects.empty()	Вызов метода delete_connect с передачей p_signal, p_cl_base, p_handler объекта указателя connects[0] в качестве аргументов	11
			12
12	this->p_head_object!= nullptr	Инициализация целочисленной переменной счетчика i = 0	13
			∅
13	i < this->p_head_object->p_sub_objects.size()		14
			∅
14	this->p_head_object->p_sub_objects[i]==this	Удаление элемента p_sub_objects[i] из вектора подчиненного головному объекту текущего объекта p_head_object->p_sub_objects.begin()+i)	15
			13
15		Инкремент i	∅

3.8 Алгоритм метода change_name класса cl_base

Функционал: метод установки имени.

Параметры: string new_name - содержит новое наименование объекта.

Возвращаемое значение: bool - есть или нет дубляж имени подчиненных объектов.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода change_name класса cl_base

№	Предикат	Действия	№ перехода
1	p_head_object != nullptr	Инициализация целочисленной переменной-счетчика i = 0	2
			4
2	i < p_head_object->p_sub_objects.size()		3
			4
3	p_head_object->p_sub_objects[i]->get_name() == new_name	Возврат false	∅
			2
4		Присвоение полю name текущего объектазначения аргумента new_name	5
5		Возврат true	∅

3.9 Алгоритм метода get_name класса cl_base

Функционал: метод получения имени.

Параметры: нет.

Возвращаемое значение: string - наименование объекта.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля name текущего объекта	∅

3.10 Алгоритм метода get_head_object класса cl_base

Функционал: метод получения указателя на родительский класс.

Параметры: нет.

Возвращаемое значение: cl_base* - указатель на головной объект текущего объекта.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода get_head_object класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля p_head_object текущего объекта	∅

3.11 Алгоритм метода get_sub_object класса cl_base

Функционал: метод получения указателя на дочерний объект.

Параметры: string name - наименование объекта.

Возвращаемое значение: cl_base* - указатель на объект.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода get_sub_object класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной-счетчика i = 0	2
2	i <p_sub_objects.size()		3
			4
3	p_sub_objects[i]->get_name()	Возврат значения указателя p_sub_objects[i]	∅

№	Предикат	Действия	№ перехода
	== name		
			2
4		Инициализация целочисленной переменной-счетчика i = 0	5
5	i <p_sub_objects.size()		6
		Возврат нулевого указателя	∅
6	p_sub_objects[i]->get_sub_object(name) != nullptr	Возврат результата обращения к методу get_sub_object объекта указателя с передачей name в качестве аргумента	∅
			5

3.12 Алгоритм метода search_by_name класса cl_base

Функционал: метод поиска объекта на дереве иерархии по имени.

Параметры: string name - наименование объекта.

Возвращаемое значение: cl_base* - указатель на искомы йобъект.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода search_by_name класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i = 0	2
2		Инициализация вектора object, принимающего указатели на объекты класса cl_base , значением одного указателя на текущий объект	3
3		Инициализация указателя p на объект класса cl_base со значением нулевого указателя	4
4	object.size()>0	Инициализация указателя temp на объект класса cl_base значением последнего элемента вектора object	5

№	Предикат	Действия	№ перехода
			10
5		Удаление последнего элемента вектора object	6
6	temp->name == name	Присвоение указателю p значения указателя temp	7
			8
7		Инремент i	8
8		Инициализация целочисленной переменной-счетчика j = 0	9
9	j < temp->p_sub_objects.size()	Добавление в вектор object значения указателя p_sub_objects[j] объекта указателя temp	9
			4
10	i < 2	Возврат значения p	∅
		Возврат пустого указателя	∅

3.13 Алгоритм метода show_ready_tree класса cl_base

Функционал: метод вывода иерархии объектов и отметок их готовности.

Параметры: int number - уровень на дереве иерархии.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода show_ready_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Переход на новую строку	2
2		Инициализация целочисленной переменной-счетчика i = 1	3
3	i <number	Вывод на экран строки " "	3
		Вывод на экран значения name	4
4	state	Вывод на экран строки " is ready"	5
		Вывод на экран строки	5

№	Предикат	Действия	№ перехода
5		Инициализация целочисленной переменной-счетчика i = 0	6
6	i < p_sub_objects.size()	Обращение к методу show_ready_tree объекта указателя p_sub_objects[i] с передачей number+1 в качестве аргумента	6
			∅

3.14 Алгоритм метода set_connect класса cl_base

Функционал: установка соединения.

Параметры: TYPE_SIGNAL p_signal - сигнал, cl_base* p_object - указатель на объект, TYPE_HANDLER p_ob_handler - обработчик.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода set_connect класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление указателя p_value на структуру o_sh	2
2		Инициализация целочисленной переменной-счетчика i= 0	3
3	i <connects.size()		4
			5
4	connects[i] -> p_signal == p_signal && connects[i]->p_cl_base == p_object && connects[i]->p_handler==p_ob_handler		∅
		Инкремент i	3
5		Присвоение указателю p_value объекта структуры	6

№	Предикат	Действия	№ перехода
		o_sh для хранения информации о новой связи	
6		Присвоение параметру p_signal объекту структуры указателя p_value значения p_signal	7
7		Присвоение параметру p_cl_base объекту структуры указателя p_value значения p_object	8
8		Присвоение параметру p_handler объекту структуры указателя p_value значения p_ob_handler	9
9		Добавление p_value в массив connects указателей на соединения текущего объекта	∅

3.15 Алгоритм метода emit_signal класса cl_base

Функционал: выдача сигнала.

Параметры: TYPE_SIGNAL p_signal - сигнал, string& s_command - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода emit_signal класса cl_base

№	Предикат	Действия	№ перехода
1	!get_state()		∅
		Вызов функции на которую указывает указатель-член p_signal структуры o_sh с передачей s_command в качестве аргумента	2
2		Инициализация челочисленной переменной-счетчика i = 0	3
3	i <connects.size()		4
			∅

№	Предикат	Действия	№ перехода
4	connects[i]->p_signal == p_signal	Вызов метода на который указывает указатель-член p_handler объекта, на который указывает указатель-член p_cl_base структуры connects[i] с передачей s_command в качестве аргумента	5
			5
5		Инкремент i	3

3.16 Алгоритм метода get_state класса cl_base

Функционал: метод для получения состояния объекта.

Параметры: нет.

Возвращаемое значение: int - статус состояния объекта.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода get_state класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения state	∅

3.17 Алгоритм метода enable_all класса cl_base

Функционал: метод для включения(активации) объекта и всех подчиненных объектов.

Параметры: нет.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода enable_all класса cl_base

№	Предикат	Действия	№ перехода
1		Присвоение полю state текущего объекта значения 1	2
2		Инициализация целочисленной переменной-счетчика i = 0	3
3	i <p_sub_objects.size()	Обращение к методу enable_all объекта указателя p_sub_objects[i]	3
			∅

3.18 Алгоритм конструктора класса cl_application

Функционал: параметризованный конструктор.

Параметры: cl_base* p_head_object - указатель на головной объект в дереве иерархии , string name - наименование создаваемого объекта.

Алгоритм конструктора представлен в таблице 19.

Таблица 19 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов парметризованного конструктора класса cl_base с параметрами p_head_object и name	∅

3.19 Алгоритм метода build_tree_objects класса cl_application

Функционал: метод построения дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода change_name с передачей строки "System" в качестве аргумента	2
2		Инициализация указателя controlPanel адресом объекта класса cl_base с помощью параметризованного конструктора ControlPanel с передачей ему текущего объекта и строки "ControlPanel" в качестве аргументов и выделение памяти под объект при помощи оператора new	3
3		Инициализация указателя reader адресом объекта класса cl_base с помощью параметризованного конструктора Readerс передачей ему текущего объекта и строки "Reader" в качестве аргументов и выделение памяти под объект при помощи оператора new	4
4		Инициализация указателя cardIdentifier адресом объекта класса cl_base с помощью параметризованного конструктора CardIdentifier с передачей ему текущего объекта и строки "CardIdentifier" в качестве аргументов и выделение памяти под объект при помощи оператора new	5
5		Инициализация указателя enter адресом объекта класса cl_base с помощью параметризованного конструктора Enterс передачей ему текущего объекта и строки "Enter" в качестве аргументов и выделение памяти под объект при помощи оператора new	6
6		Инициализация указателя issue адресом объекта класса cl_base с помощью параметризованного конструктора Issue с передачей ему текущего объекта и строки "Issue" в качестве аргументов и выделение памяти под объект при помощи оператора new	7
7		Инициализация указателя printer адресом объекта класса cl_base с помощью параметризованного конструктора Printer с передачей ему текущего объекта и строки "Printer" в качестве аргументов и выделение памяти под объект при помощи оператора new	8

№	Предикат	Действия	№ перехода
8		Обращение к методу set_connect текущего объекта с передачей SIGNAL_D(cl_application::command_request_signal), reader, HANDLER_D(Reader::read_obr) в качестве аргументов	9
9		Обращение к методу set_connect объекта указателя reader с передачей SIGNAL_D(Reader::read_signal), this, HANDLER_D(cl_application::command_obr) в качестве аргументов	10
10		Обращение к методу set_connect объекта указателя reader с передачей SIGNAL_D(Reader::read_signal), controlPanel, HANDLER_D(ControlPanel::command_obr) в качестве аргументов	11
11		Обращение к методу set_connect объекта указателя reader с передачей SIGNAL_D(Reader::read_signal), cardIdentifier, HANDLER_D(CardIdentifier::command_obr) в качестве аргументов	12
12		Обращение к методу set_connect объекта указателя reader с передачей SIGNAL_D(Reader::read_signal), issue, HANDLER_D(Issue::command_obr) в качестве аргументов	13
13		Обращение к методу set_connect текущего объекта с передачей SIGNAL_D(cl_application::text_print_signal), printer, HANDLER_D(Printer::print_obr) в качестве аргументов	14
14		Обращение к методу set_connect объекта указателя controlPanel с передачей SIGNAL_D(ControlPanel::text_print_signal), printer, HANDLER_D(Printer::print_obr) в качестве аргументов	15
15		Обращение к методу set_connect объекта указателя cardIdentifier с передачей SIGNAL_D(CardIdentifier::print_signal), printer, HANDLER_D(Printer::print_obr) в качестве аргументов	16
16		Обращение к методу set_connect объекта указателя issue с передачей SIGNAL_D(Issue::print_signal), printer, HANDLER_D(Printer::print_obr) в качестве аргументов	17
17		Обращение к методу set_connect объекта указателя enter с передачей SIGNAL_D(Enter::sum_print_signal), printer,	18

№	Предикат	Действия	№ перехода
		HANDLER_D(Printer::print_obr) в качестве аргументов	
18		Обращение к методу set_connect объекта указателя cardIdentifier с передачей SIGNAL_D(CardIdentifier::set_status_signal), controlPanel, HANDLER_D(ControlPanel::set_status_obr) в качестве аргументов	19
19		Обращение к методу set_connect объекта указателя cardIdentifier с передачей SIGNAL_D(CardIdentifier::set_available_sum_signal), controlPanel, HANDLER_D(ControlPanel::set_available_sum_obr) в качестве аргументов	20
20		Обращение к методу set_connect объекта указателя controlPanel с передачей SIGNAL_D(ControlPanel::withdraw_signal), issue, HANDLER_D(Issue::withdraw_obr) в качестве аргументов	21
21		Обращение к методу set_connect объекта указателя issue с передачей SIGNAL_D(Issue::balance_reduction_signal), cardIdentifier, HANDLER_D(CardIdentifier::balance_reduction_obr) в качестве аргументов	22
22		Обращение к методу set_connect объекта указателя controlPanel с передачей SIGNAL_D(ControlPanel::add_bill_signal), enter, HANDLER_D(Enter::add_bill_obr) в качестве аргументов	23
23		Обращение к методу set_connect объекта указателя controlPanel с передачей SIGNAL_D(ControlPanel::deposit_money_signal), enter, HANDLER_D(Enter::deposit_money_obr) в качестве аргументов	24
24		Обращение к методу set_connect объекта указателя controlPanel с передачей SIGNAL_D(ControlPanel::return_bills_signal), enter, HANDLER_D(Enter::return_money_obr) в качестве аргументов	25
25		Обращение к методу set_connect объекта указателя enter с передачей SIGNAL_D(Enter::add_money_signal), cardIdentifier, HANDLER_D(CardIdentifier::balance_increasing_obr) в качестве аргументов	∅

3.20 Алгоритм метода exec_app класса cl_application

Функционал: запуск основного алгоритма программы.

Параметры: нет.

Возвращаемое значение: int - индикатор успешного выполнения алгоритма.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода enable_all()	2
2		Инициализация пременной ready строковго типа со значением "Ready to work"	3
3		Вызов метода emit_signal с передачей SIGNAL_D(cl_application::text_print_signal), ready в качестве аргументов	4
4		Вызов метода emit_signal с передачей SIGNAL_D(cl_application::command_request_signal), command в качестве аргментов	5
5	command != "End card loading"		4
			6
6		Вызов метода emit_signal с передачей SIGNAL_D(cl_application::command_request_signal), command в качестве аргументов	7
7		Вызов метода emit_signal с передачей SIGNAL_D(cl_application::command_request_signal), command в качестве аргументов	8
8	true	Вызов метода emit_signal с передачей SIGNAL_D(cl_application::command_request_signal), command в качестве аргументов	9

№	Предикат	Действия	№ перехода
			∅
9	command == "Turn off the ATM"	Вызов метода emit_signal с передачей SIGNAL_D(cl_application::text_print_signal), command в качестве аргументов	10
			11
10		break	∅
11	command == "SHOWTREE"	Инициализация переменной buf строкового типа со значением ""	12
			8
12		Вызов метода emit_signal с передачей SIGNAL_D(cl_application::text_print_signal), buf в качестве аргументов	13
13		Вызов метода show_ready_tree с передачей 0 в качестве аргумента	14
14		break	∅

3.21 Алгоритм метода command_request_signal класса cl_application

Функционал: метод-сигнал для запроса команды.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода command_request_signal класса cl_application

№	Предикат	Действия	№ перехода
1			∅

3.22 Алгоритм метода command_obr класса cl_application

Функционал: метод-обработчик для получения команды.

Параметры: string _command - строка-команда.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода command_obr класса cl_application

№	Предикат	Действия	№ перехода
1		Присвоение полю command значения аргумента _command	∅

3.23 Алгоритм метода text_print_signal класса cl_application

Функционал: метод-сигнал для запроса вывода текста.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода text_print_signal класса cl_application

№	Предикат	Действия	№ перехода
1			∅

3.24 Алгоритм конструктора класса CardIdentifier

Функционал: параметризованный конструктор.

Параметры: cl_base* parent - указатель на головной объект в дереве иерархии , string name - наименование сохдаваемого объекта.

Алгоритм конструктора представлен в таблице 25.

Таблица 25 – Алгоритм конструктора класса CardIdentifier

№	Предикат	Действия	№ перехода
1		Вызов парметризованного конструктора класса cl_base с параметрами parent и name	∅

3.25 Алгоритм метода command_obr класса CardIdentifier

Функционал: метод-обработчик для получения и обработки поступившей команды.

Параметры: string command - строка-команда.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода command_obr класса CardIdentifier

№	Предикат	Действия	№ перехода
1	status == MONEYLOADING		2
			4
2	command == "End money loading"	Присвоение переменной status значения WAITING	3
			3
3		break	37
4	status == WAITING		5
			17
5	command[0] != 'I' && command[0] != 'E' && command[0] != 'T' && command[0] != 'S'	Инициализация переменной ready строкового типа значением "Ready to work"	6
			7
6		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::print_signal), ready в	7

№	Предикат	Действия	№ перехода
		качестве аргументов	
7	command[0] == 'I'	Инициализация переменной number строкового типа значением переменной command(начиная с 15-ого символа)	8
			16
8		Инициализация целочисленной переменной-счетчика i = 0	9
9	i < cards.size()		10
			16
10	cards[i].number == number	Присвоение переменной status знчения PINWAIT	11
		Инкремент i	9
11		Присвоение полю curCard значения i	12
12		Инициализация переменной msg1 строкового типа со значением = "Enter the PIN code"	13
13		Инициализация переменной msg2 строкового типа со значением = "PIN"	14
14		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::print_signal), msg1 в качестве аргументов	15
15		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::set_status_signal), msg2 в качестве аргументов	∅
16		break	37
17	status == READINFO		18
			19
18	command == "End card loading"	Присвоение status значения MONEYLOADING	∅
		Добавление в вектор cards {command.substr(0, 19), command.substr(20, 4), stoi(command.substr(25))}	∅

№	Предикат	Действия	№ перехода
19	status == PINWAIT		20
			37
20	command[0] == 'P'		21
			30
21	cards[curCard].PIN == command.substr(9)	Присвоение status значения LOGGED	22
			28
22		Инициализация переменной msg1 строкового типа со значением = "Select the command"	23
23		Инициализация переменной msg2 строкового типа со значением = "LOGGED"	24
24		Инициализация переменной string msg3 строкового типа со значением = приведению к строковому типу balance объекта cards[curCard]	25
25		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::print_signal), msg1 в качестве аргументов	26
26		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::set_status_signal), msg2 в качестве аргументов	27
27		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::set_available_sum_signal), msg3 в качестве аргументов	∅
28		Инициализация переменной msg строкового типа со значением = "Enter the PIN code"	29
29		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::print_signal), msg в качестве аргументов	∅
30		Присвоение полю curCard значения -1	31

№	Предикат	Действия	№ перехода
31		Присвоение status значения WAITING	32
32		Инициализация переменной msg1 строкового типа со значением = "READY"	33
33		Инициализация переменной msg2 строкового типа со значением = "Ready to work"	34
34		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::set_status_signal), msg1 в качестве аргументов	35
35		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::print_signal), msg2 в качестве аргументов	36
36		break	37
37	command == "End the session"	Присвоение полю curCard значения -1	38
			∅
38		Присвоение status значения WAITING	39
39		Инициализация переменной msg1 строкового типа со значением = "READY"	40
40		Инициализация переменной msg2 строкового типа со значением = "Ready to work"	41
41		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::set_status_signal), msg1 в качестве аргументов	42
42		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::print_signal), msg2 в качестве аргументов	∅

3.26 Алгоритм метода set_status_signal класса CardIdentifier

Функционал: метод-сигнал для установки статуса.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода set_status_signal класса CardIdentifier

№	Предикат	Действия	№ перехода
1			∅

3.27 Алгоритм метода balance_increasing_obr класса CardIdentifier

Функционал: метод-обработчик для увеличения баланса.

Параметры: string sum - сумма.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода balance_increasing_obr класса CardIdentifier

№	Предикат	Действия	№ перехода
1		Увеличение значения поля balance объекта cards[curCard] на значение sum(привеленной к целочисленному типу)	2
2		Инициализация переменной msg строкового типа со значением = "Card balance " и приведенного к строке поля balance объекта cards[curCard]	3
3		Вызов метода emit_signal с передачей SIGNAL_D(CardIdentifier::print_signal), msg в качестве аргументов	∅

3.28 Алгоритм метода set_available_sum_signal класса CardIdentifier

Функционал: метод-сигнал для установки доступного баланса.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 29.

Таблица 29 – Алгоритм метода set_available_sum_signal класса CardIdentifier

№	Предикат	Действия	№ перехода
1			∅

3.29 Алгоритм метода print_signal класса CardIdentifier

Функционал: метод-сигнал для запроса вывода текста.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 30.

Таблица 30 – Алгоритм метода print_signal класса CardIdentifier

№	Предикат	Действия	№ перехода
1			∅

3.30 Алгоритм метода balance_reduction_obr класса CardIdentifier

Функционал: метод-обработчик для уменьшения баланса.

Параметры: string sum - сумма.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 31.

Таблица 31 – Алгоритм метода balance_reduction_obr класса CardIdentifier

№	Предикат	Действия	№ перехода
1		Уменьшение значения поля balance объекта cards[curCard] на значение sum(привеленной к целочисленному типу)	∅

3.31 Алгоритм конструктора класса ControlPanel

Функционал: параметризованный конструктор.

Параметры: cl_base* parent - указатель на головной объект в дереве иерархии , string name - наименование сохдаваемого объекта.

Алгоритм конструктора представлен в таблице 32.

Таблица 32 – Алгоритм конструктора класса ControlPanel

№	Предикат	Действия	№ перехода
1		Вызов парметризованного конструктора класса cl_base с параметрами parent и name	∅

3.32 Алгоритм метода command_obr класса ControlPanel

Функционал: метод-обработчик для получения и обработки поступившей команды.

Параметры: string command - команда.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 33.

Таблица 33 – Алгоритм метода command_obr класса ControlPanel

№	Предикат	Действия	№ перехода
1	status == LOGGED		2

№	Предикат	Действия	№ перехода
			∅
2	command[0] == 'D'		3
			9
3	command[14] == 't'	Присвоение hasMoneyInserted значения false	4
		Присвоение hasMoneyInserted значения true	7
4		Инициализация переменной msg строкового типа со значением = ""	5
5		Вызов метода emit_signal с передачей SIGNAL_D(ControlPanel::deposit_money_signal), command в качестве аргументов	6
6		Вызов метода emit_signal с передачей SIGNAL_D(ControlPanel::set_status_obr), msg в качестве аргументов	11
7		Инициализация переменной msg1 строкового типа со значением = подстроки строки command(начиная с 14-ого символа)	8
8		Вызов метода emit_signal с передачей SIGNAL_D(ControlPanel::add_bill_signal), msg1 в качестве аргумента	11
9	hasMoneyInserted	Инициализация переменной msg1 строкового типа со значением = подстроки строки command(начиная с 14-ого символа)	10
			11
10		Вызов метода emit_signal с передачей SIGNAL_D(ControlPanel::add_bill_signal), msg1 в качестве аргумента	11
11	command[0] == 'W'	Инициализация целочисленной переменной sum со значением = приведенной к целочисленному виду подстроки command(начиная с 15-ого	12

№	Предикат	Действия	№ перехода
		символа)	
			∅
12	sum % 100 != 0	Инициализация переменной msg строкового типа со значением = "The amount is not a multiple of 100"	13
			14
13		Вызов метода emit_signal с передачей SIGNAL_D(ControlPanel::text_print_signal), msg в качестве аргументов	∅
14	availableSum < sum	Инициализация переменной msg строкового типа со значением = "There is not enough money on the card"	15
		Инициализация переменной msg1 строкового типа со значением = sum ,привеленного к строковому виду	16
15		Вызов метода emit_signal с передачей SIGNAL_D(ControlPanel::text_print_signal), msg в качестве аргументов	∅
16		Вызов метода emit_signal с передачей SIGNAL_D(ControlPanel::withdraw_signal), msg1 в качестве аргументов	∅

3.33 Алгоритм метода set_status_obr класса ControlPanel

Функционал: метод-обработчик для установки статуса.

Параметры: string message - строка-сообщение.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 34.

Таблица 34 – Алгоритм метода set_status_obr класса ControlPanel

№	Предикат	Действия	№ перехода
1	message[0] == 'L'	Присвоение status значения LOGGED break	∅
			2
2	message[0] == 'R'	Присвоение status значения WAITING break	∅
			3
3	message[0] == 'P'	Присвоение status значения PIN break	∅
			∅

3.34 Алгоритм метода add_bill_signal класса ControlPanel

Функционал: метод-сигнал для внесения денег.

Параметры: string& message - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 35.

Таблица 35 – Алгоритм метода add_bill_signal класса ControlPanel

№	Предикат	Действия	№ перехода
1			∅

3.35 Алгоритм метода deposit_money_signal класса ControlPanel

Функционал: метод-сигнал для окончательного внесения денег.

Параметры: string& message - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 36.

Таблица 36 – Алгоритм метода deposit_money_signal класса ControlPanel

№	Предикат	Действия	№ перехода
1			∅

3.36 Алгоритм метода return_bills_signal класса ControlPanel

Функционал: метод-сигнал для обнуления суммы.

Параметры: string& message - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 37.

Таблица 37 – Алгоритм метода return_bills_signal класса ControlPanel

№	Предикат	Действия	№ перехода
1			∅

3.37 Алгоритм метода text_print_signal класса ControlPanel

Функционал: метод-сигнал для запроса вывода текста.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 38.

Таблица 38 – Алгоритм метода text_print_signal класса ControlPanel

№	Предикат	Действия	№ перехода
1			∅

3.38 Алгоритм метода set_available_sum_obr класса ControlPanel

Функционал: метод-обработчик для установки доступного баланса карты.

Параметры: string sum - сумма.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 39.

Таблица 39 – Алгоритм метода set_available_sum_obr класса ControlPanel

№	Предикат	Действия	№ перехода
1		Присвоение полю availableSum значения sum, приведенного к целочисленному виду	∅

3.39 Алгоритм метода withdraw_signal класса ControlPanel

Функционал: метод-сигнал для снятия денег.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 40.

Таблица 40 – Алгоритм метода withdraw_signal класса ControlPanel

№	Предикат	Действия	№ перехода
1			∅

3.40 Алгоритм конструктора класса Enter

Функционал: параметризованный конструктор.

Параметры: cl_base* parent - указатель на головной объект в дереве иерархии , string name - наименование сохдаваемого объекта.

Алгоритм конструктора представлен в таблице 41.

Таблица 41 – Алгоритм конструктора класса Enter

№	Предикат	Действия	№ перехода
1		Вызов парметризованного конструктора класса cl_base с параметрами	∅

№	Предикат	Действия	№ перехода
		parent и name	

3.41 Алгоритм метода add_bill_obr класса Enter

Функционал: метод-обработчик для внесения денег.

Параметры: string bill - полученное значение.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 42.

Таблица 42 – Алгоритм метода add_bill_obr класса Enter

№	Предикат	Действия	№ перехода
1		Инициализация целочиленной переменной value со значением = bill(приведенным к целочисленному виду)	2
2	value == 100 || value == 500 || value == 1000 || value == 2000 || value == 5000	Инициализация переменной msg строкового типа со значением = "The amount: " и приведенного к строковому типу значения(sum+value)	3
			∅
3		Вызов метода emit_signal с передачей SIGNAL_D(Enter::sum_print_signal), msg в качестве аргументов	4
4		Увеличение значения sum на величину value	∅

3.42 Алгоритм метода sum_print_signal класса Enter

Функционал: метод-сигнал для вывода баланса карты.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 43.

Таблица 43 – Алгоритм метода sum_print_signal класса Enter

№	Предикат	Действия	№ перехода
1			∅

3.43 Алгоритм метода deposit_money_obr класса Enter

Функционал: метод-обработчик для окончательного внесения денег.

Параметры: string - строка.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 44.

Таблица 44 – Алгоритм метода deposit_money_obr класса Enter

№	Предикат	Действия	№ перехода
1		Инициализация переменной msg строкового типа со значением = значению sum(приведенного к строковому типу)	2
2		Вызов метода emit_signal с передачей SIGNAL_D(Enter::add_money_signal), msg в качестве аргументов	3
3		Присвоению полю sum значения 0	∅

3.44 Алгоритм метода add_money_signal класса Enter

Функционал: метод-сигнал для увеличения баланса карты.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 45.

Таблица 45 – Алгоритм метода add_money_signal класса Enter

№	Предикат	Действия	№ перехода
1			∅

3.45 Алгоритм метода return_money_obr класса Enter

Функционал: метод обработчик для установки sum в 0.

Параметры: string - строка.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 46.

Таблица 46 – Алгоритм метода return_money_obr класса Enter

№	Предикат	Действия	№ перехода
1		Присвоение полю sum значения 0	∅

3.46 Алгоритм конструктора класса Issue

Функционал: параметризованный конструктор.

Параметры: cl_base* parent - указатель на головной объект в дереве иерархии , string name - наименование сохдаваемого объекта.

Алгоритм конструктора представлен в таблице 47.

Таблица 47 – Алгоритм конструктора класса Issue

№	Предикат	Действия	№ перехода
1		Вызов парметризованного конструктора класса cl_base с параметрами parent и name	∅

3.47 Алгоритм метода command_obr класса Issue

Функционал: метод-обработчик для получения и обработки поступившей команды.

Параметры: string command - получаемая команда.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 48.

Таблица 48 – Алгоритм метода command_obr класса Issue

№	Предикат	Действия	№ перехода
1	command == "End card loading"	Присвоение status значения = READING	∅
			2
2	status == READING	Создание вектора counts типа <int>, содержащий 5 элементов	3
			∅
3		Инициализация переменной str строкового типа со значением = command	4
4		Инициализация целочисленной переменной-счетчика i = 0	5
5	i <5	Присвоение counts[i] значения подстроки str(от 0-ого элемента до вхождения строки ' ')(с приведением к целочисленному типу)	6
			8
6		Удаление подстроки из строки str(от 0-ого элемента до вхождения строки ' '+1)	7
7		Инкремент i	5
8		Присвоение bills значения counts	9
9		Присвоени еполю status значения = WORKING	∅

3.48 Алгоритм метода withdraw_obr класса Issue

Функционал: метод-обработчик для проверки и вывода денег.

Параметры: string _sum - сумма.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 49.

Таблица 49 – Алгоритм метода withdraw_obr класса Issue

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной sum значением = значению _sum(приведенного к целочисленному виду)	2
2		Создание вектора withdraw типа <int>, содержащий 5 элементов	3
3		Инициализация вектора value с элементами типа <int> со значениями = {5000, 2000, 1000, 500, 100}	4
4		Инициализация целочисленной переменной-счетчика i = 0	5
5	i <5		6
			9
6	sum - value[i] >= 0 && withdraw[i] < bills[i]	Декремент bills[i]	7
		Инкремент i	5
7		Инкремент withdraw[i]	8
8		Уменьшение значения sum на величину value[i]	6
9	sum != 0	Инициализация переменной msg строкового типа со значением = "There is not enough money in the ATM"	10
		Инициализация целочисленной переменной счетчика i =0	11
10		Вызов метода emit_signal с передачей SIGNAL_D(Issue::print_signal), msg в качестве аргументов	∅
11	i < 5	Уменьшение значения bills[i] на величину згачения withdraw[i] Инкремент i	11

№	Предикат	Действия	№ перехода
		Инициализация переменной msg строкового типа со значением = "Take the money: 5000 * " + значения withdraw[0], приведенного к строковому типу + " rub., 2000 * " + значения withdraw[1], приведенного к строковому типу + " rub., 1000 * " + значения withdraw[2], приведенного к строковому типу + " rub., 500 * " + значения withdraw[3], приведенного к строковому типу + " rub., 100 * " + значения withdraw[4], приведенного к строковому типу + " rub.";	12
12		Вызов метода emit_signal с передачей SIGNAL_D(Issue::balance_reduction_signal), _sum в качестве аргументов	13
13		Вызов метода emit_signal с передачей SIGNAL_D(Issue::print_signal), msg в качестве аргументов	∅

3.49 Алгоритм метода print_signal класса Issue

Функционал: метод-сигнал для вывода текста на экран.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 50.

Таблица 50 – Алгоритм метода print_signal класса Issue

№	Предикат	Действия	№ перехода
1			∅

3.50 Алгоритм метода balance_reduction_signal класса Issue

Функционал: метод-сигнал для уменьшения баланса.

Параметры: string& - ссылка на строку.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 51.

Таблица 51 – Алгоритм метода balance_reduction_signal класса Issue

№	Предикат	Действия	№ перехода
1			∅

3.51 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: целочисленное - индикатор успешного выполнения программы.

Алгоритм функции представлен в таблице 52.

Таблица 52 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_cl_application класса cl_application с передачей nullptr в качестве аргумента для параметризованного конструктора	2
2		Вызов метода build_tree_objects объекта ob_cl_application	3
3		Возврат результата вызова метода exec_app объекта ob_cl_application	∅

3.52 Алгоритм метода delete_connect класса cl_base

Функционал: метод удаления(разрыва) связи между сигналом текущего объекта и обработчиком целевого.

Параметры: TYPE_SIGNAL p_signal - сигнал, cl_base* p_object - указатель на объект, TYPE_HANDLER p_ob_handler - обработчик.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 53.

Таблица 53 – Алгоритм метода delete_connect класса cl_base

№	**Предикат**	**Действия**	**№ перехода**
1		Объявление указателя p_value на объект структуры o_sh	2
2		Инициализация целочисленной переменной-счетчика i = 0	3
3	i <connects.size()		4
			∅
4	connects[i]->p_signal == p_signal && connects[i]->p_cl_base == p_object && connects[i]->p_handler == p_ob_handler	Удаление i-ого элемента из connects	5
			5
5		Инкремент i	3

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-34.

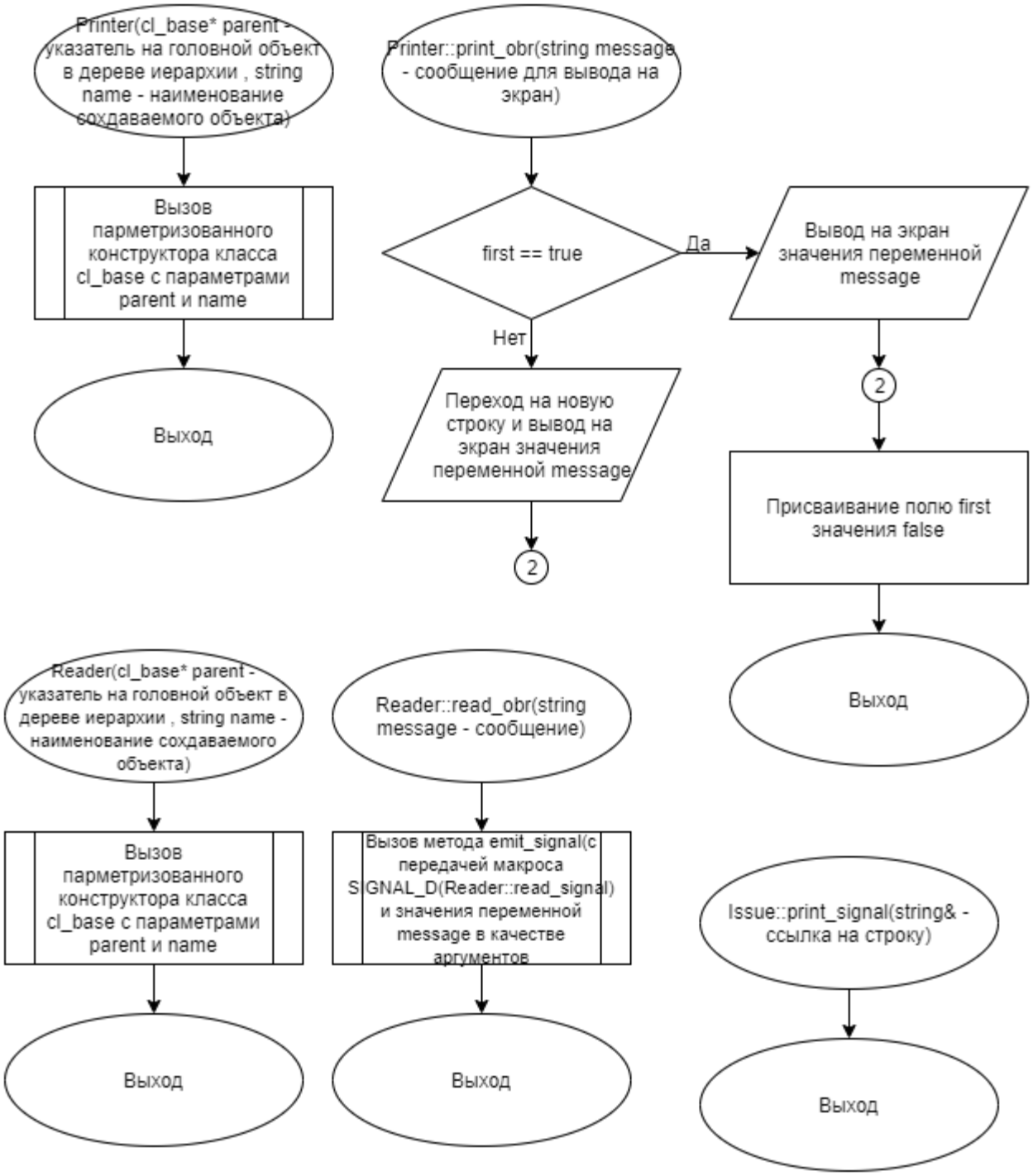

Рисунок 1 – Блок-схема алгоритма

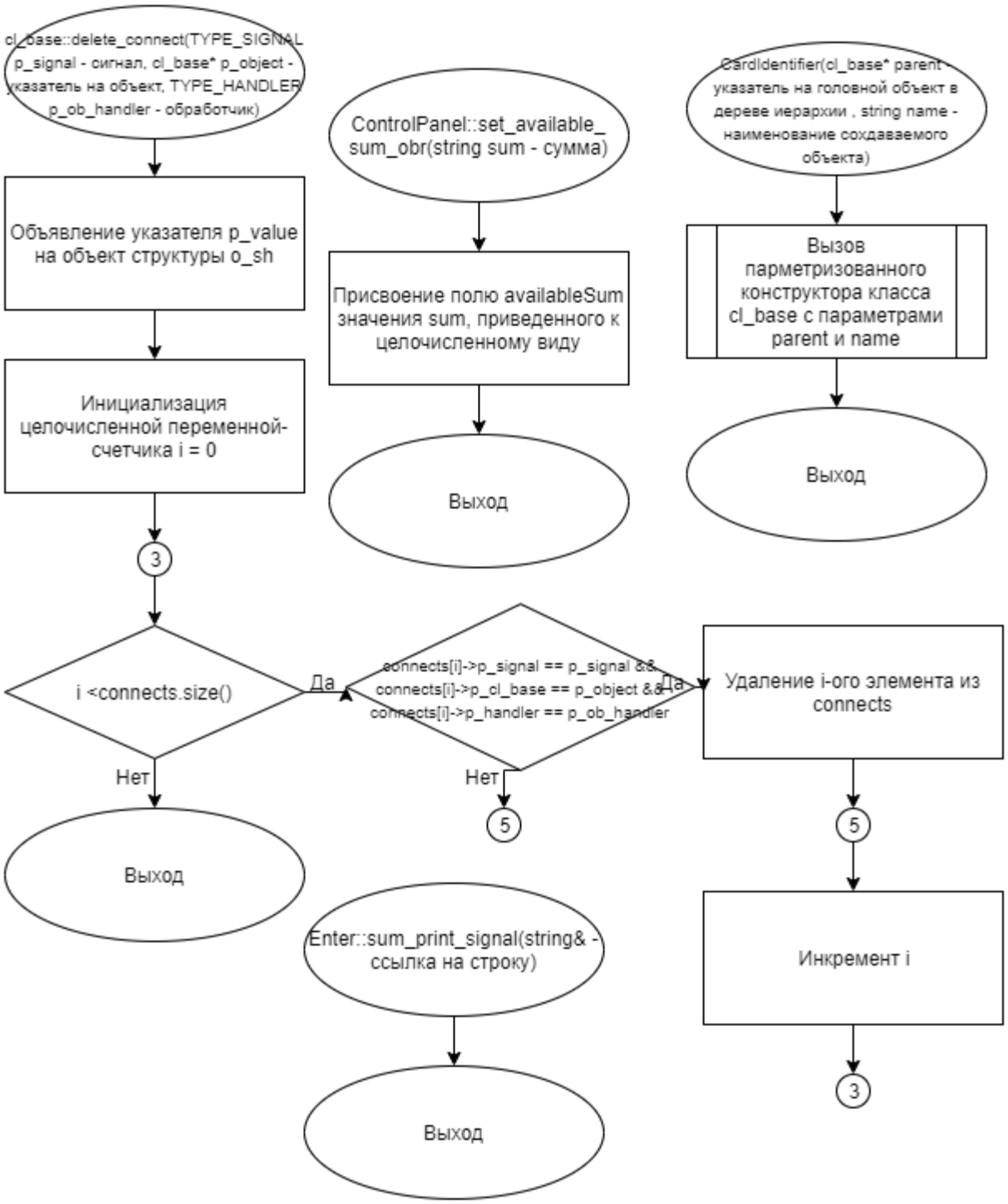

Рисунок 2 – Блок-схема алгоритма

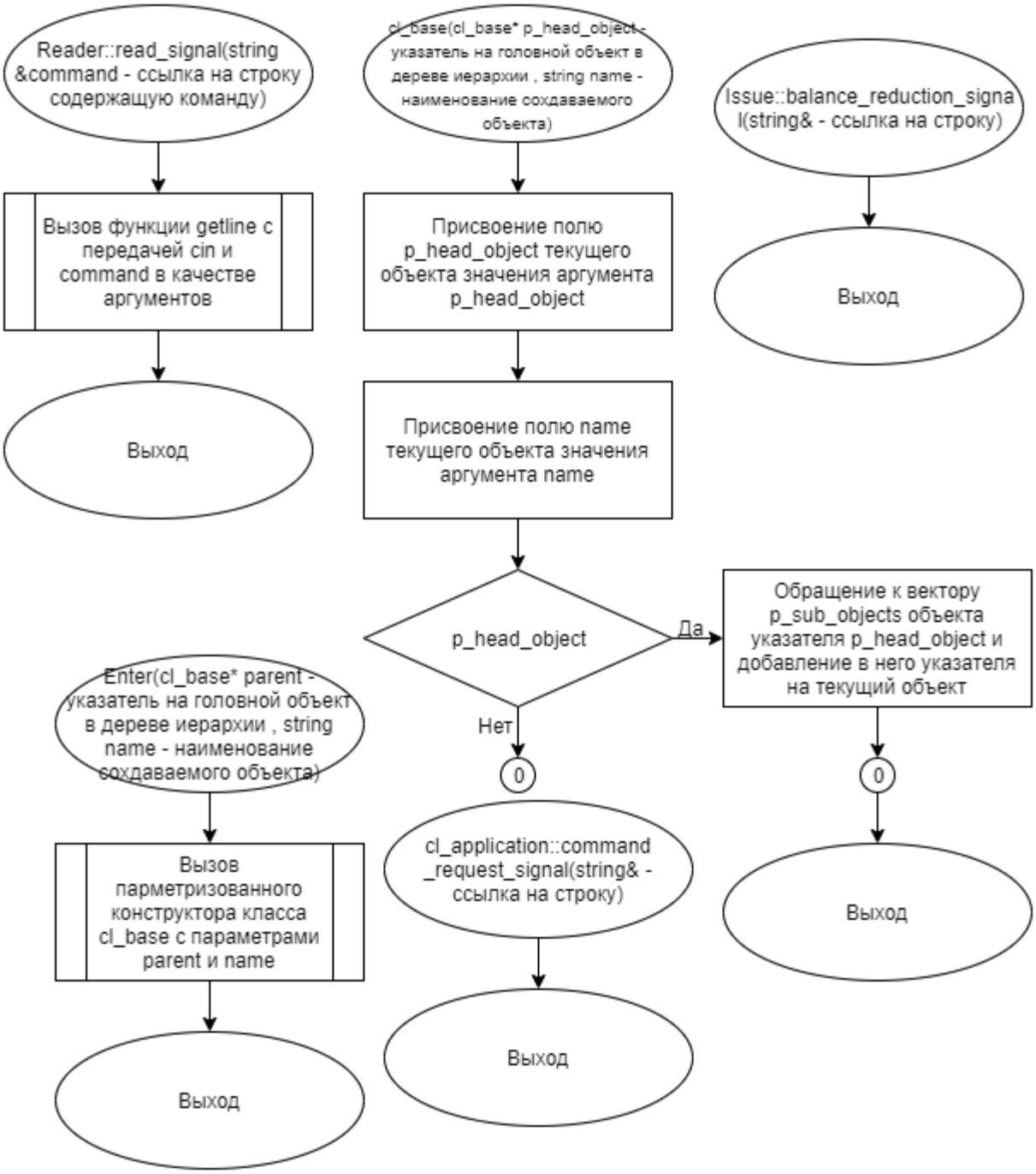

Рисунок 3 – Блок-схема алгоритма

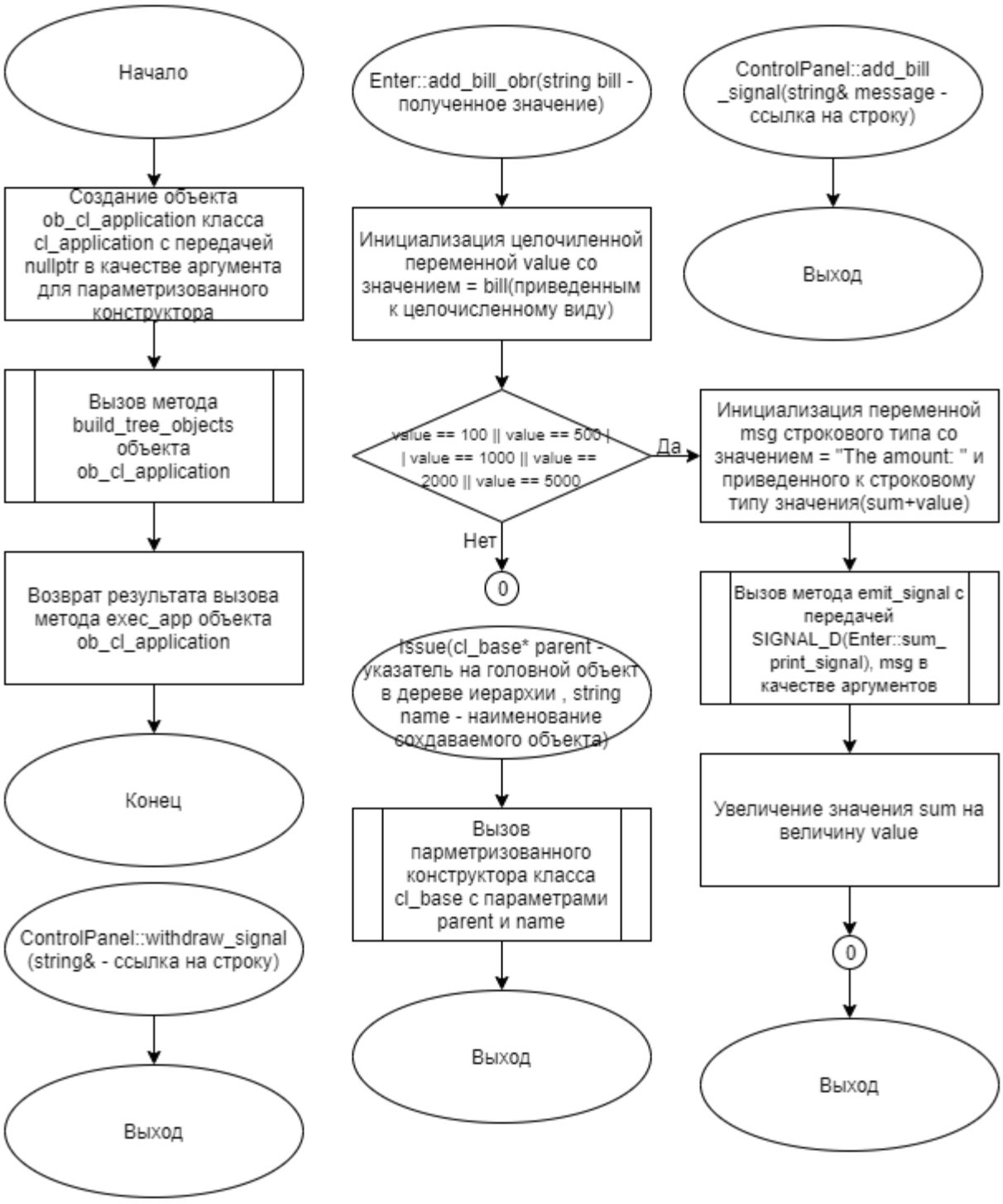

Рисунок 4 – Блок-схема алгоритма

Рисунок 5 – Блок-схема алгоритма

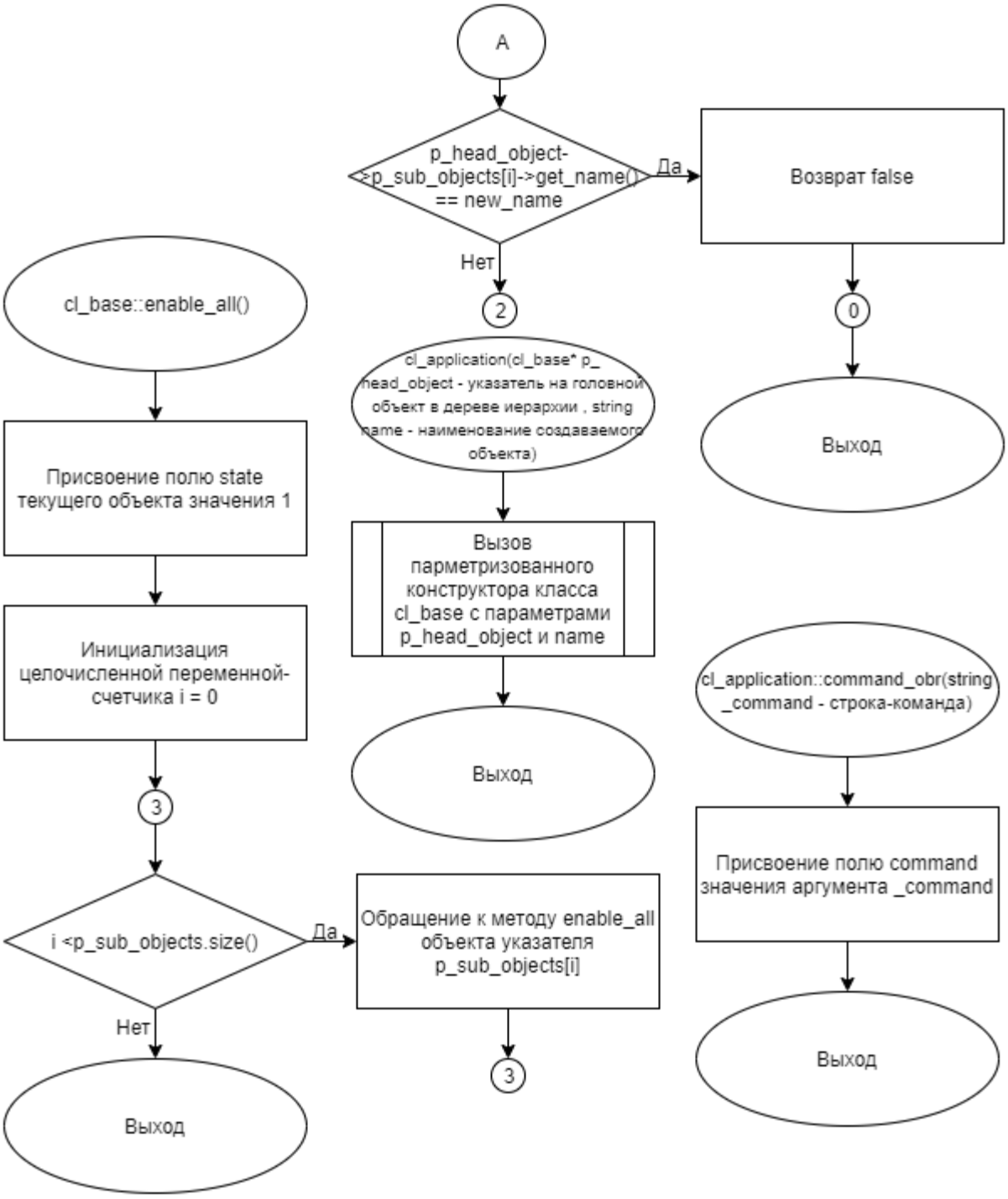

Рисунок 6 – Блок-схема алгоритма

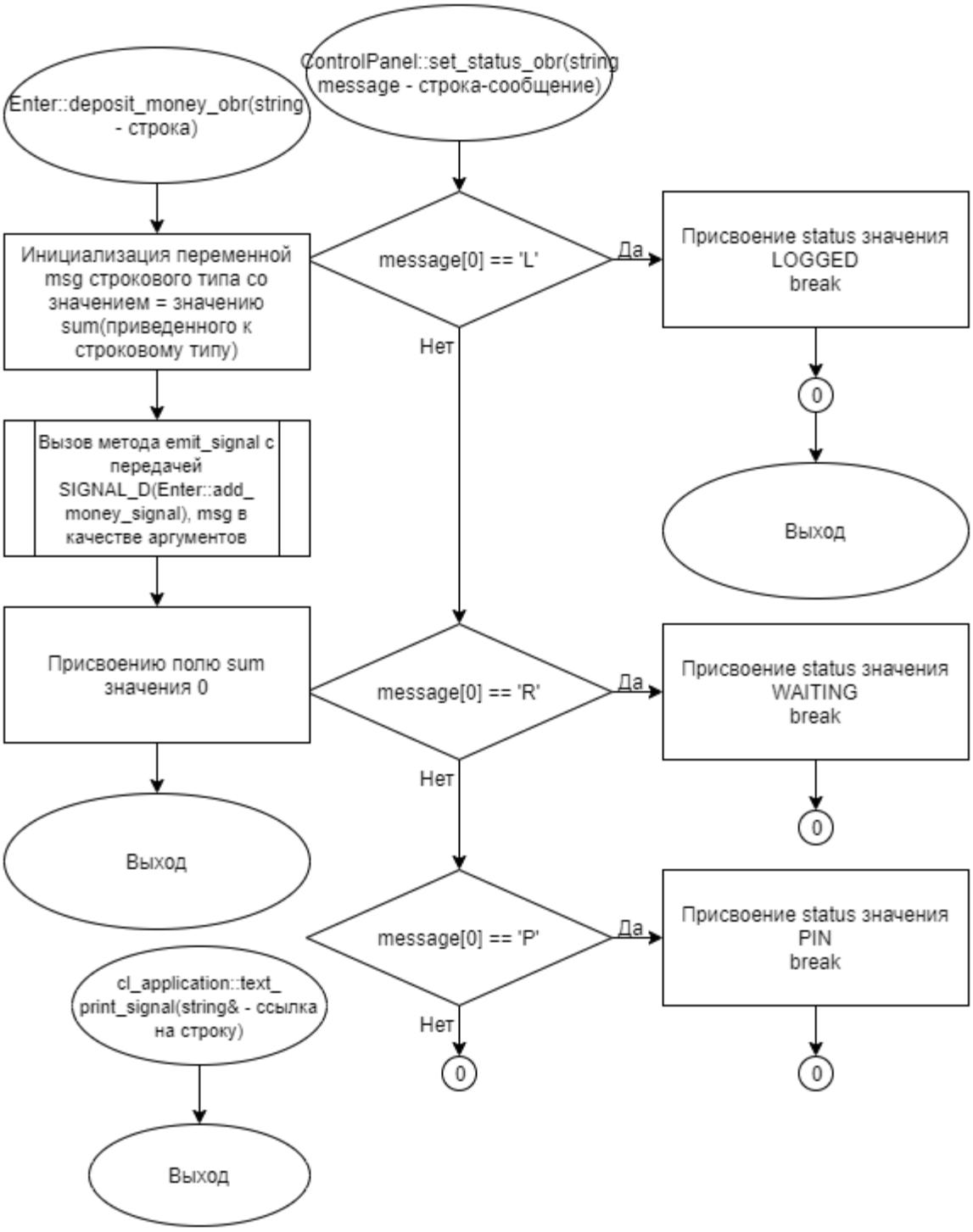

Рисунок 7 – Блок-схема алгоритма

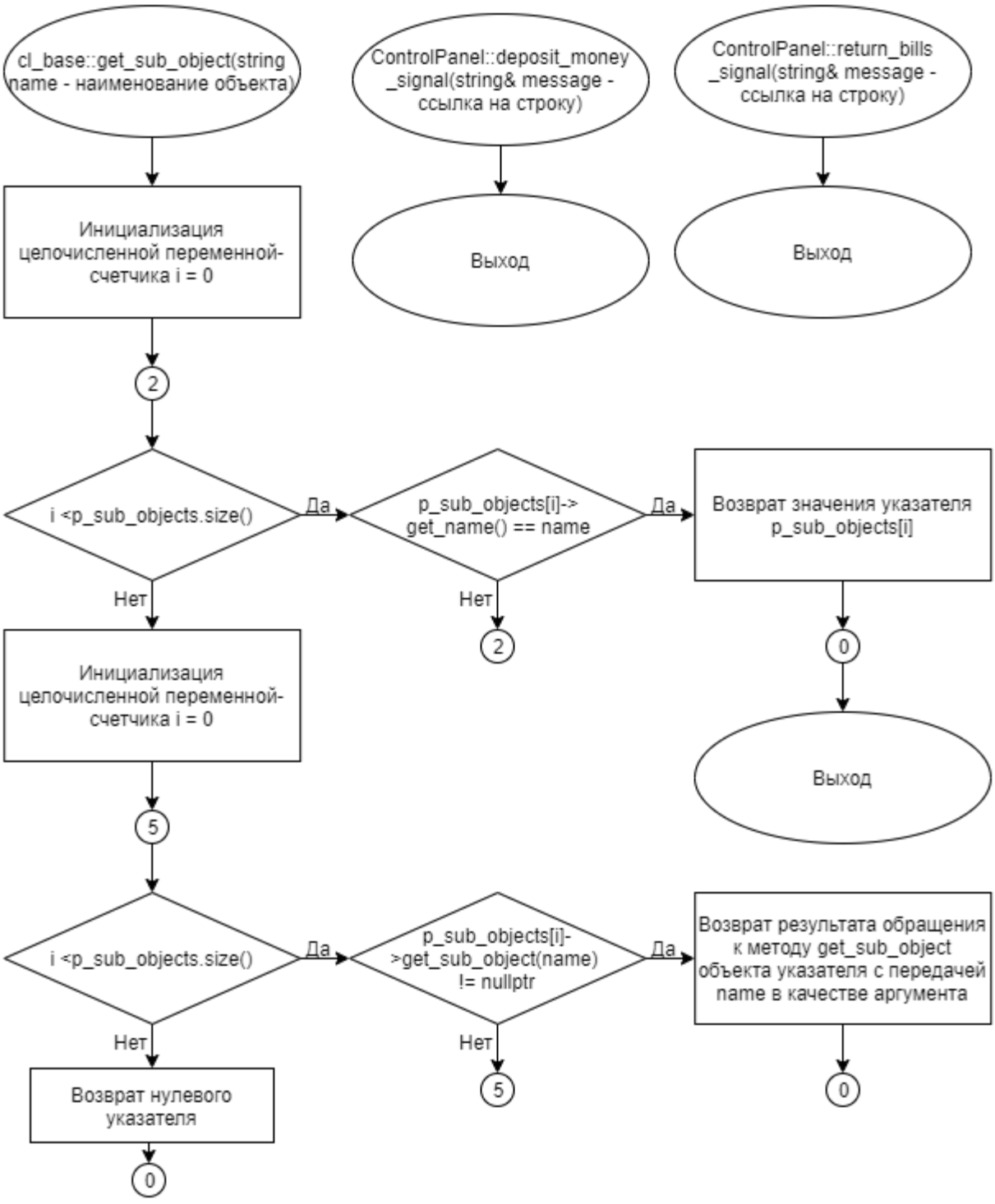

Рисунок 8 – Блок-схема алгоритма

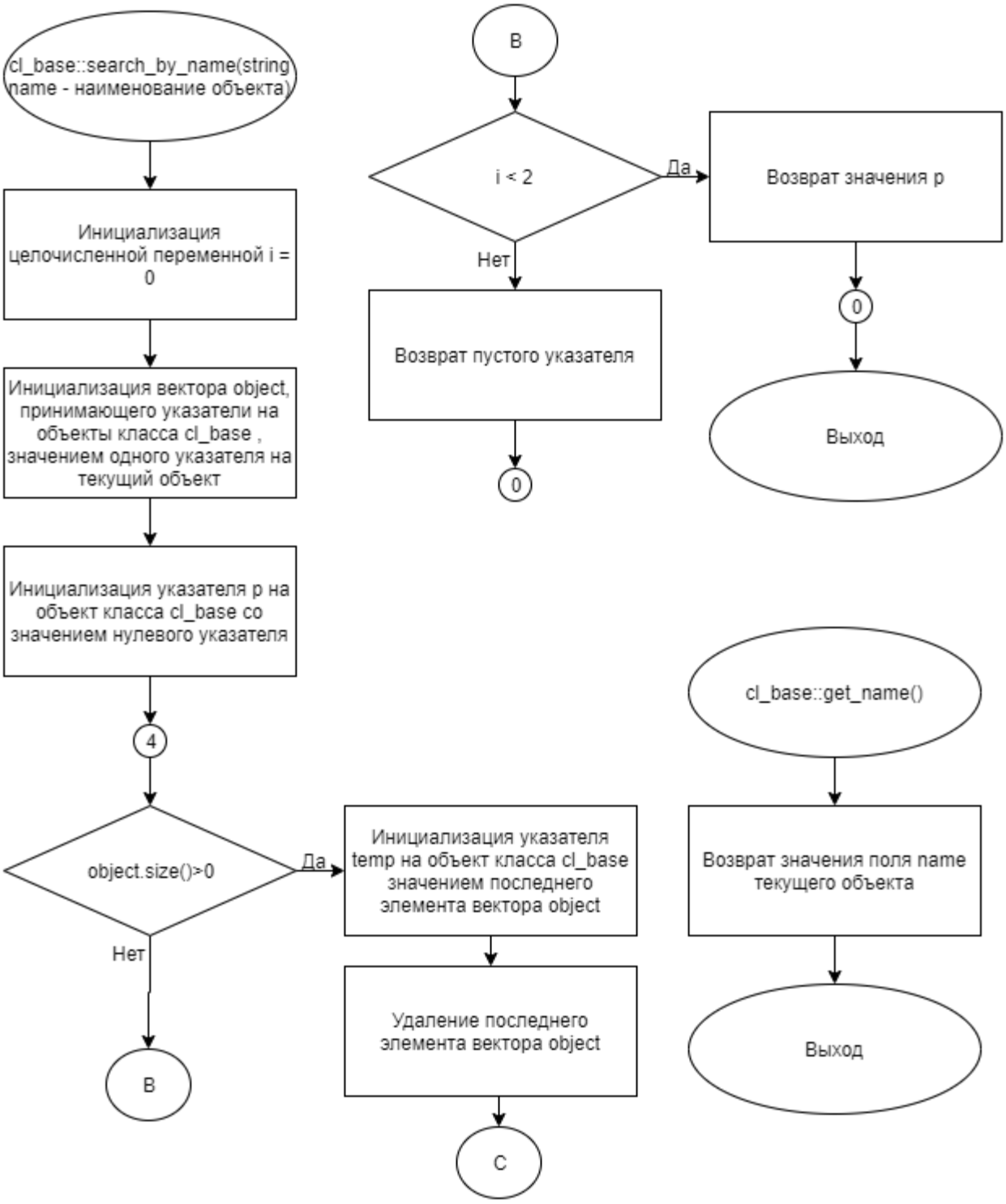

Рисунок 9 – Блок-схема алгоритма

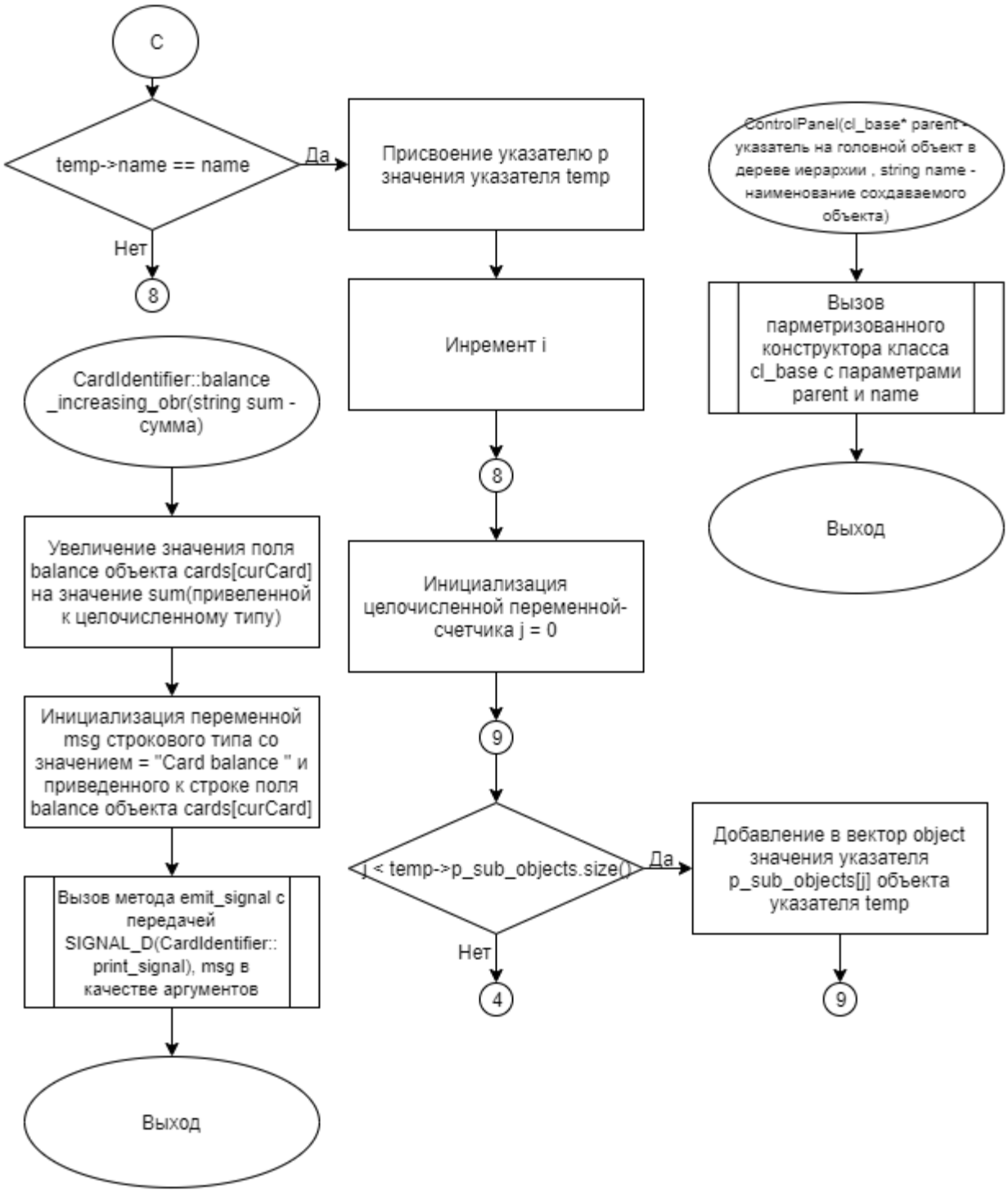

Рисунок 10 – Блок-схема алгоритма

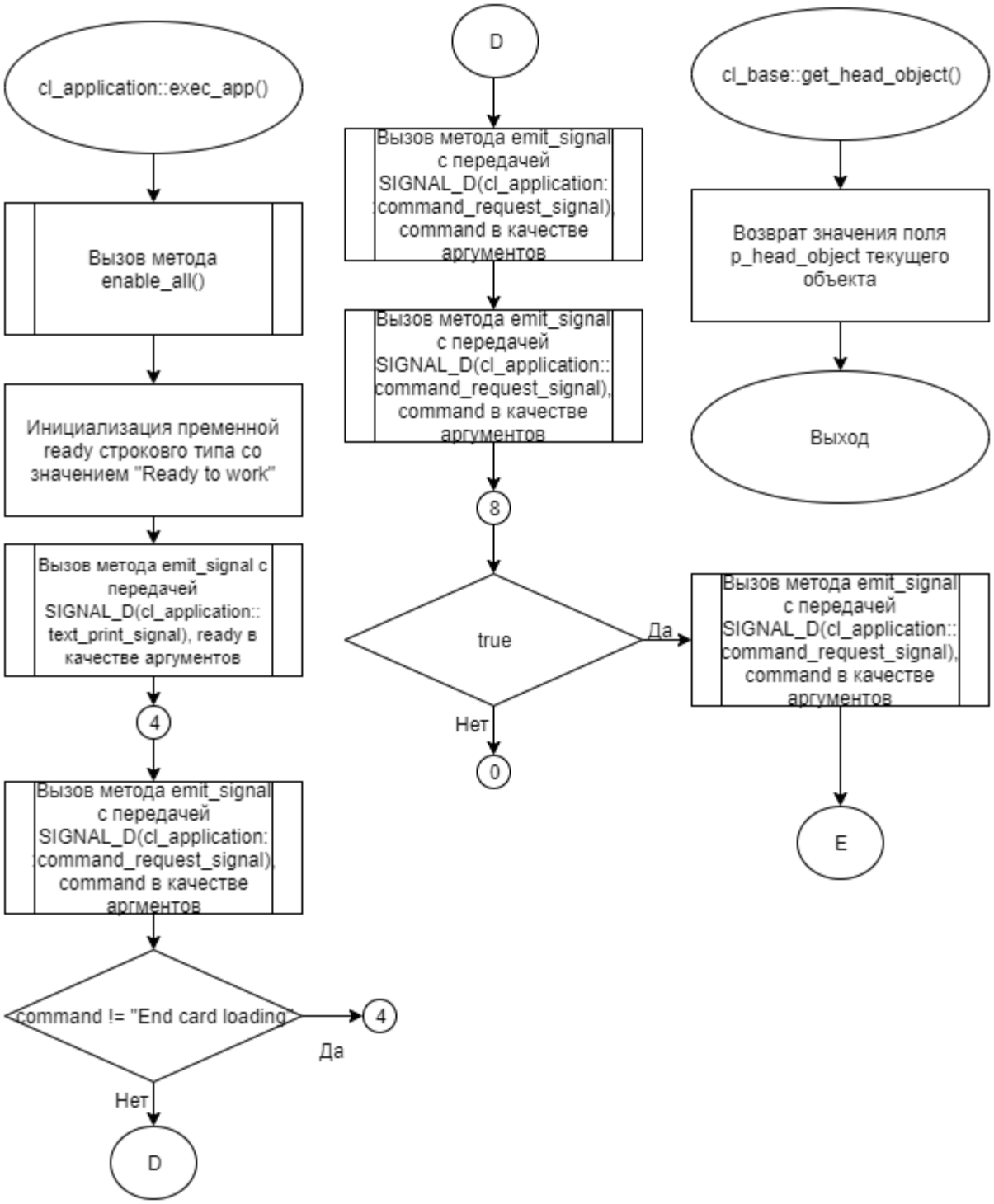

Рисунок 11 – Блок-схема алгоритма

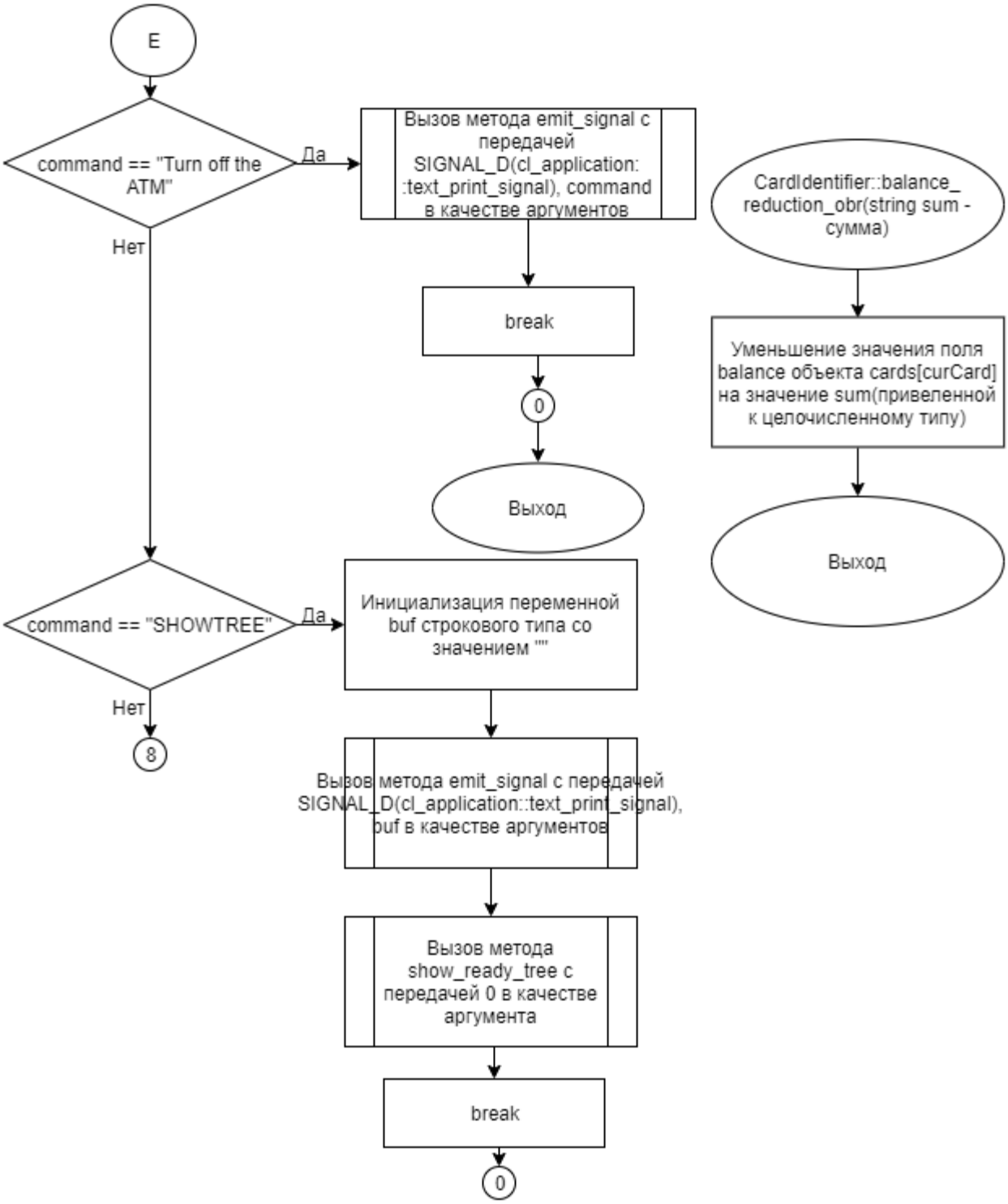

Рисунок 12 – Блок-схема алгоритма

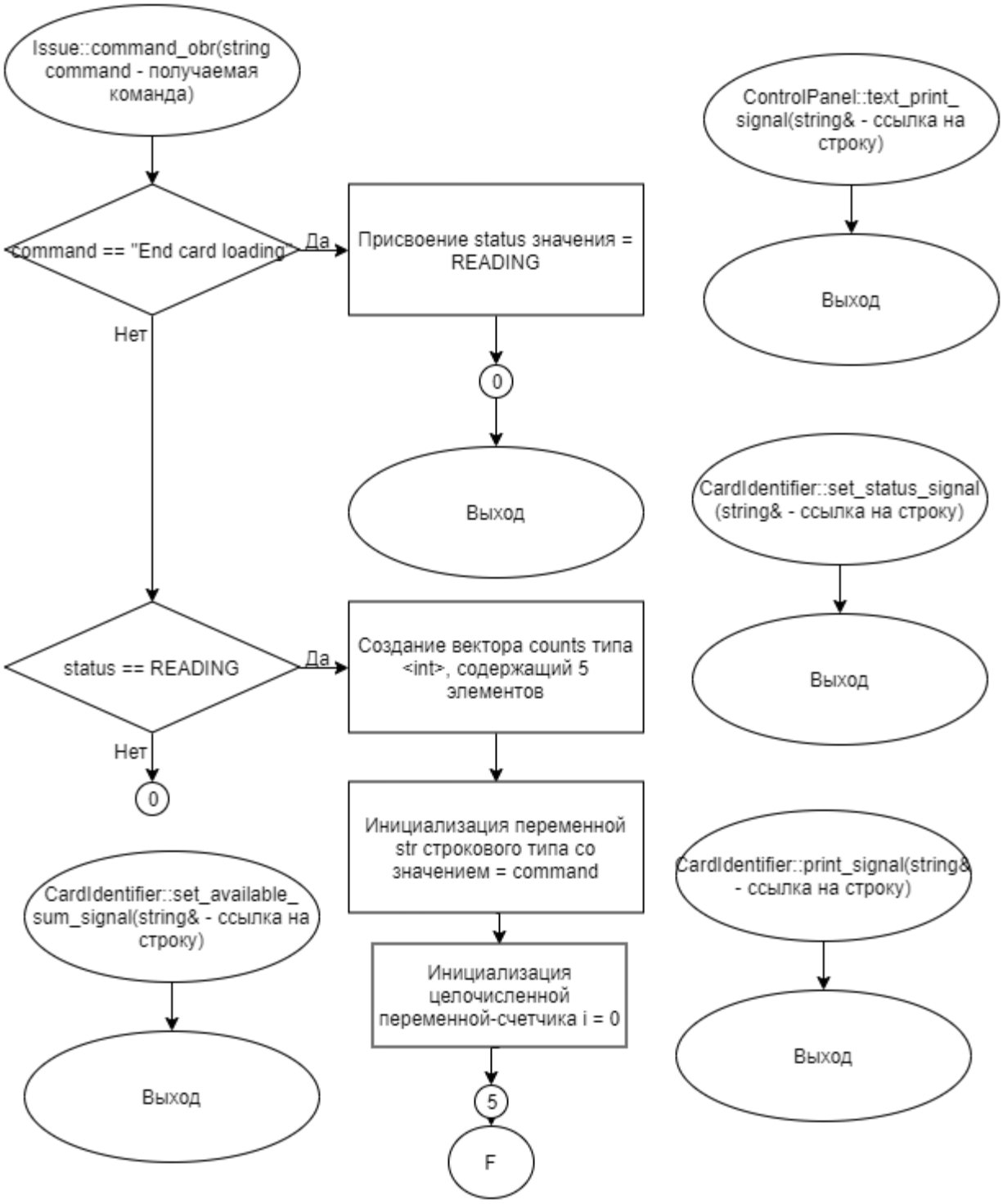

Рисунок 13 – Блок-схема алгоритма

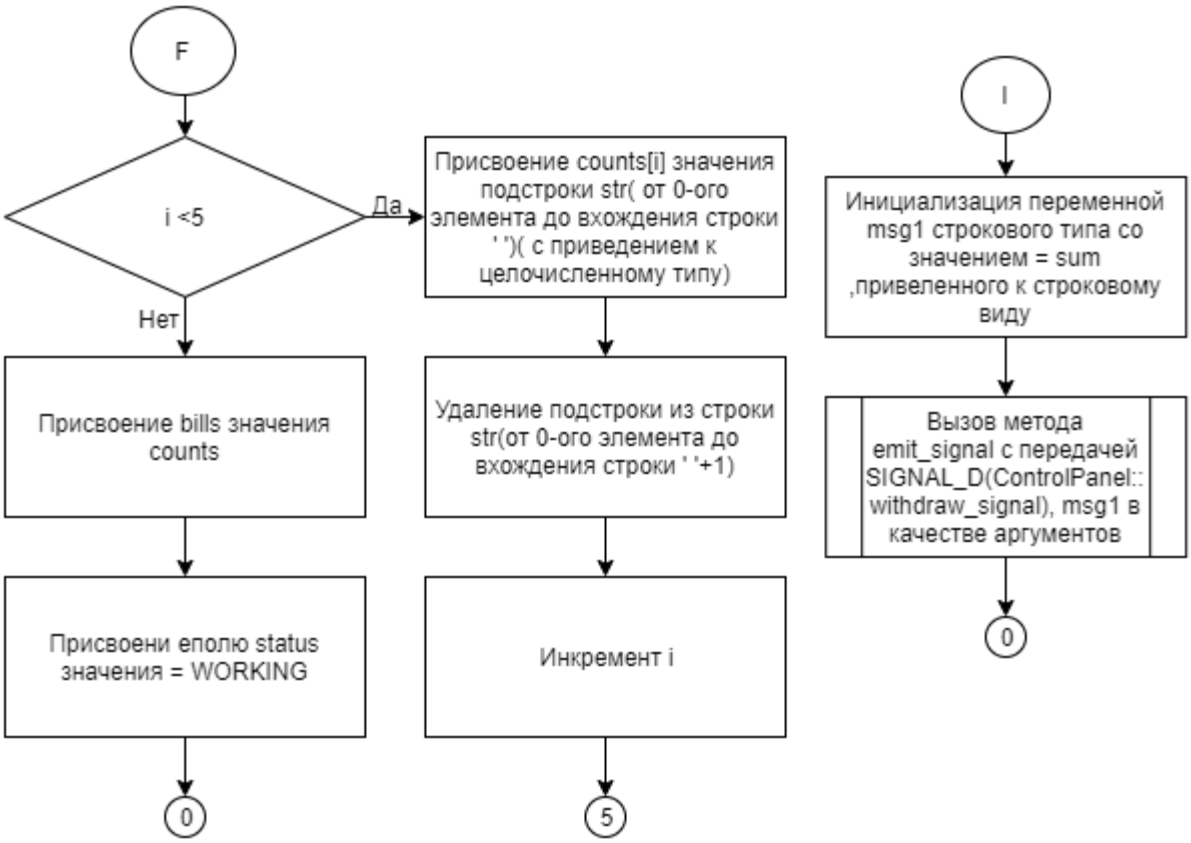

Рисунок 14 – Блок-схема алгоритма

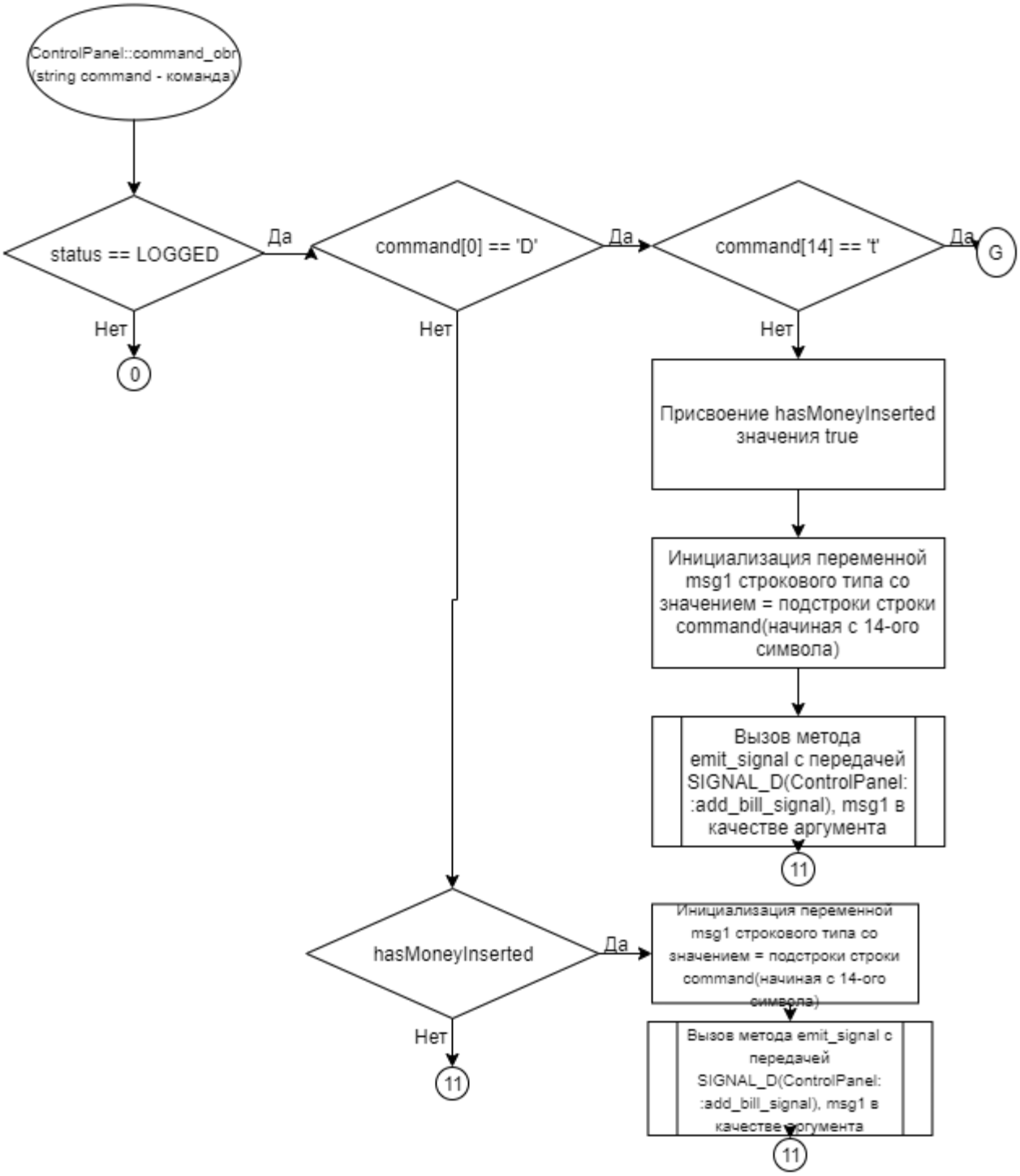

Рисунок 15 – Блок-схема алгоритма

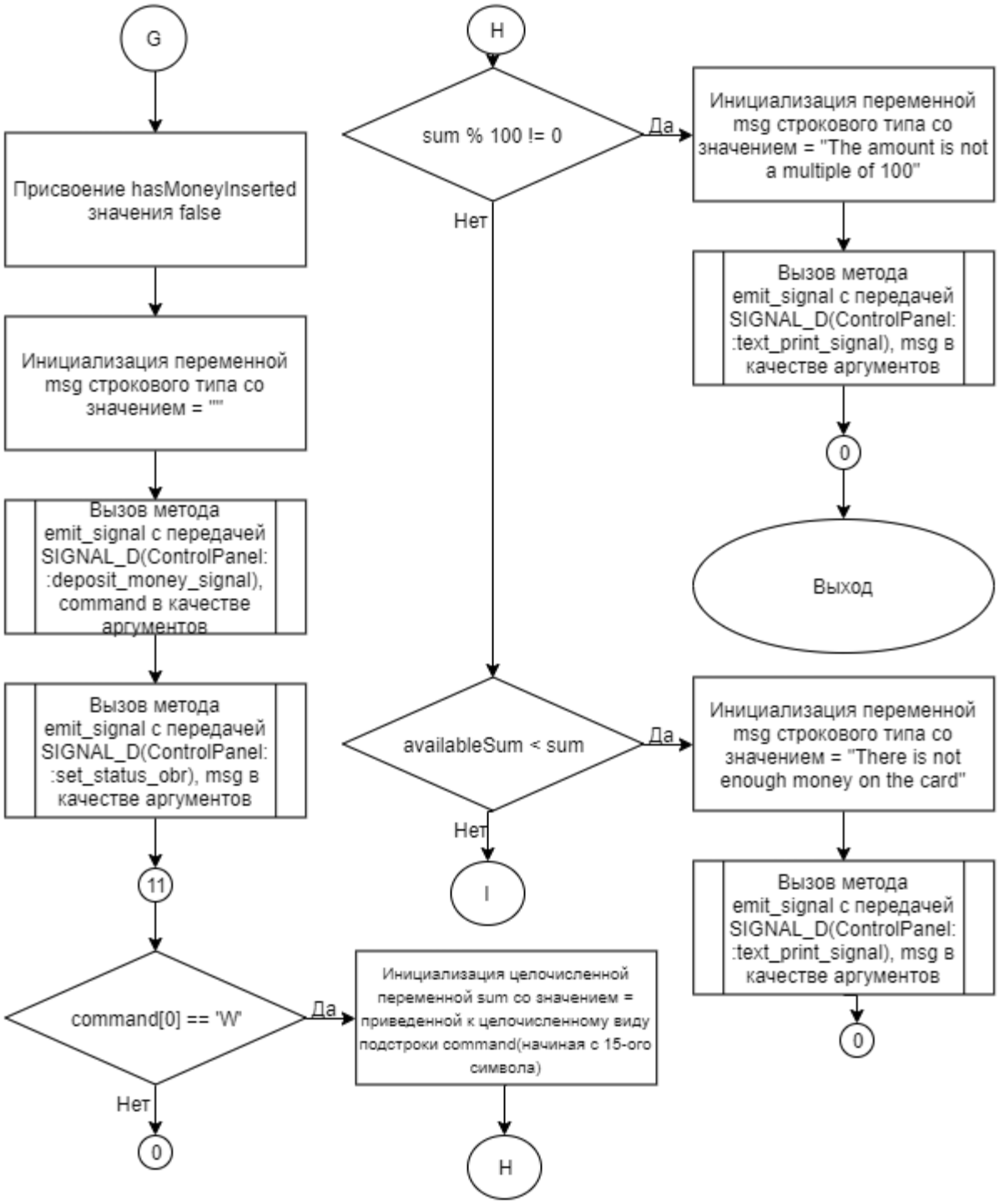

Рисунок 16 – Блок-схема алгоритма

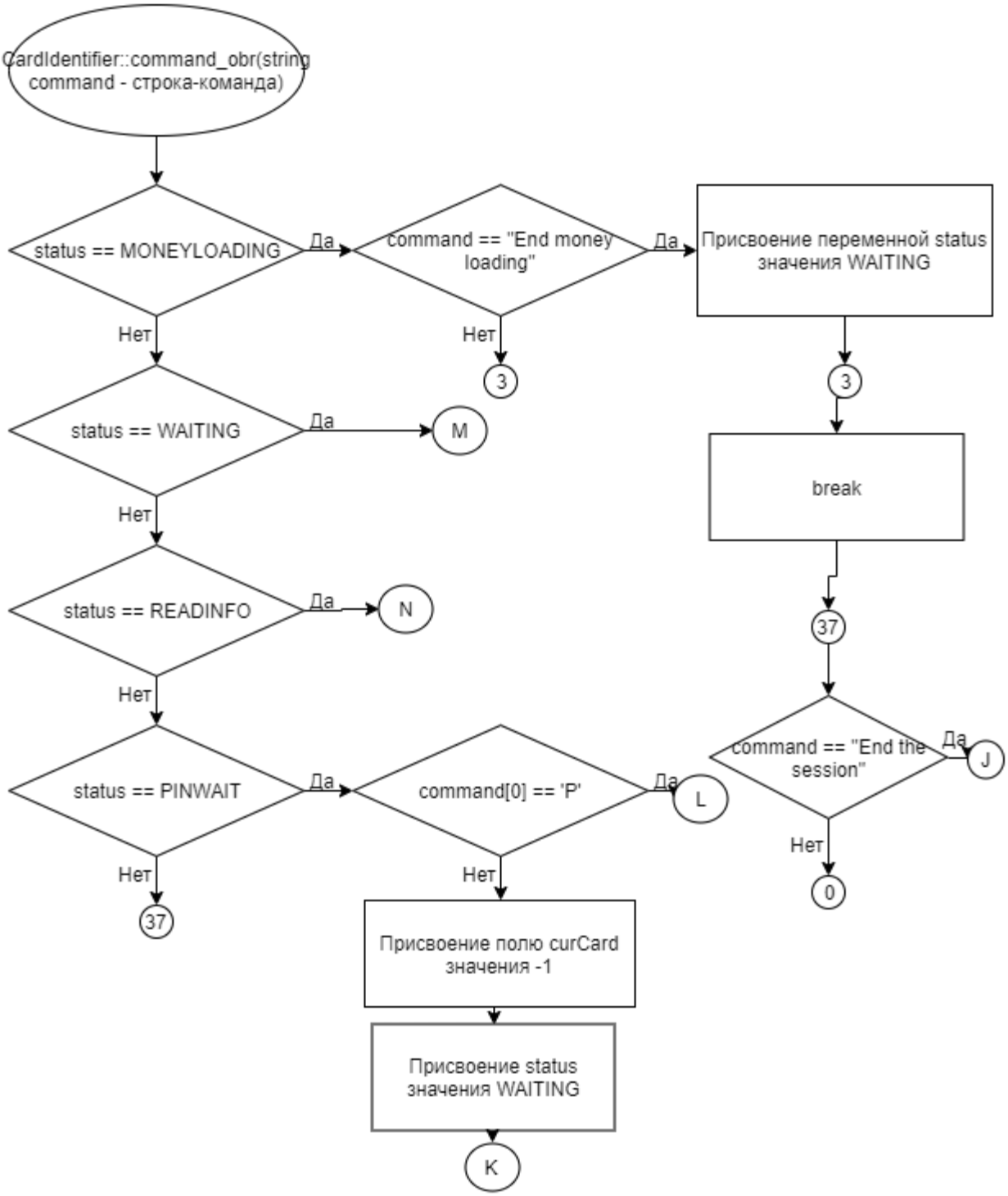

Рисунок 17 – Блок-схема алгоритма

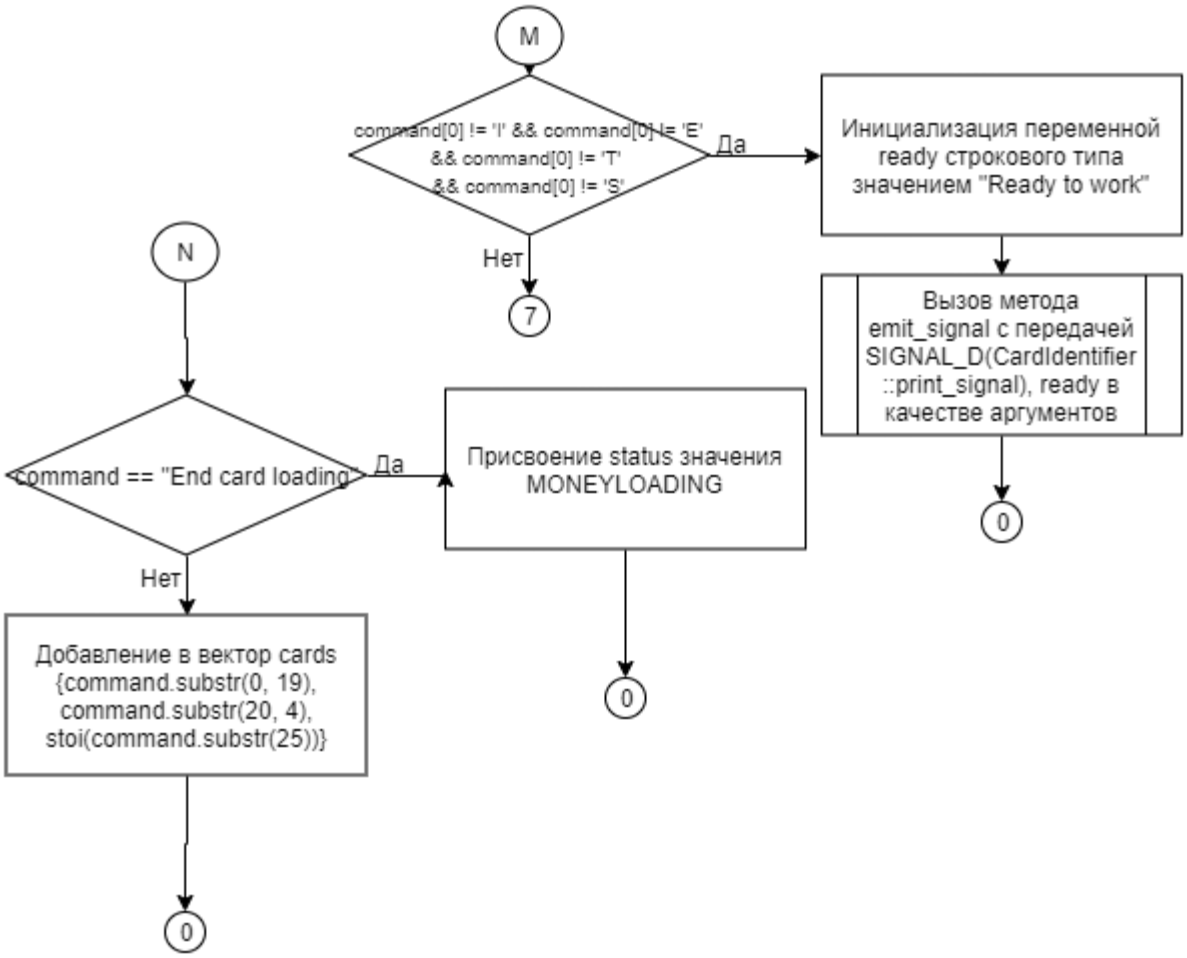

Рисунок 18 – Блок-схема алгоритма

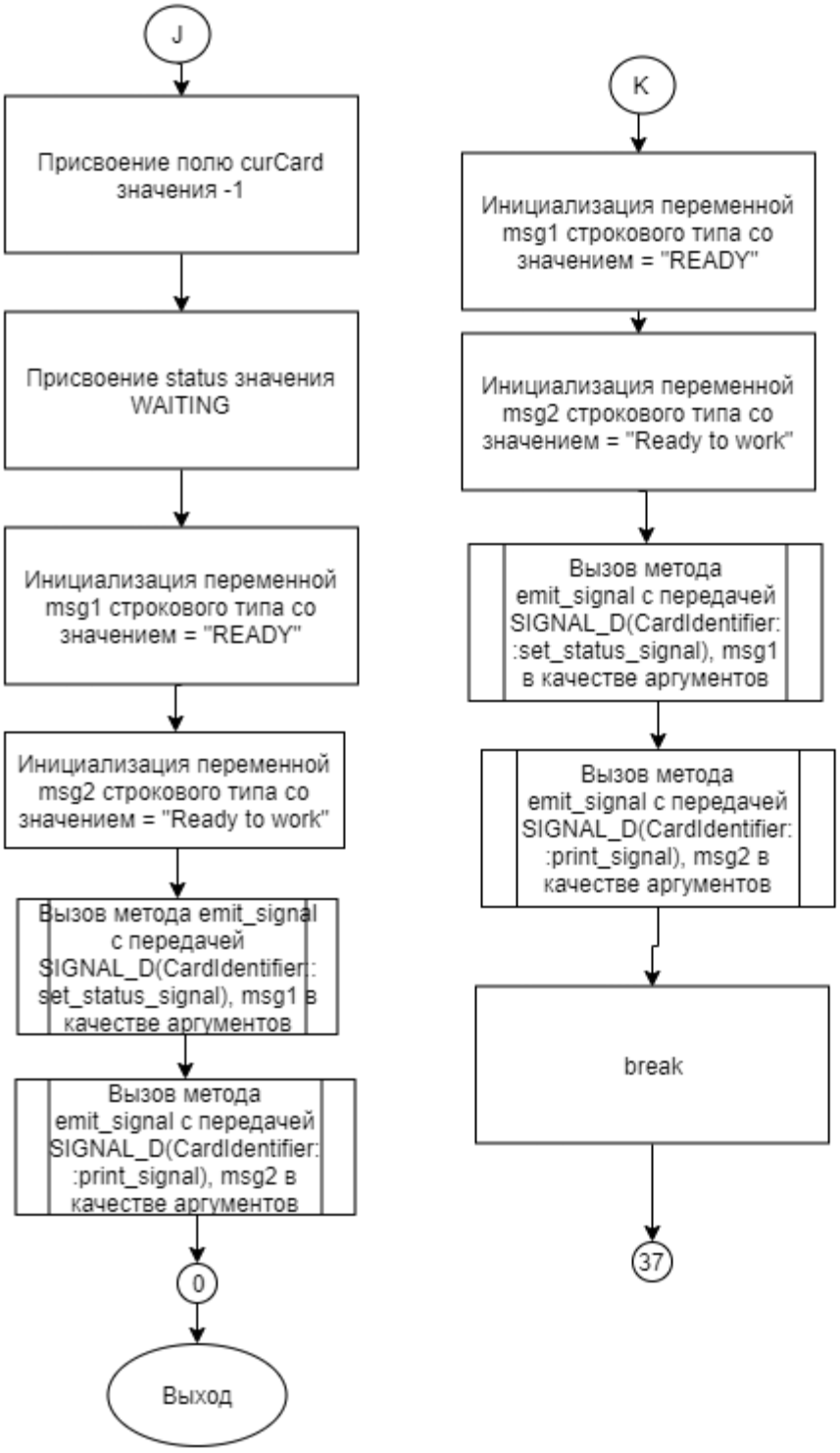

Рисунок 19 – Блок-схема алгоритма

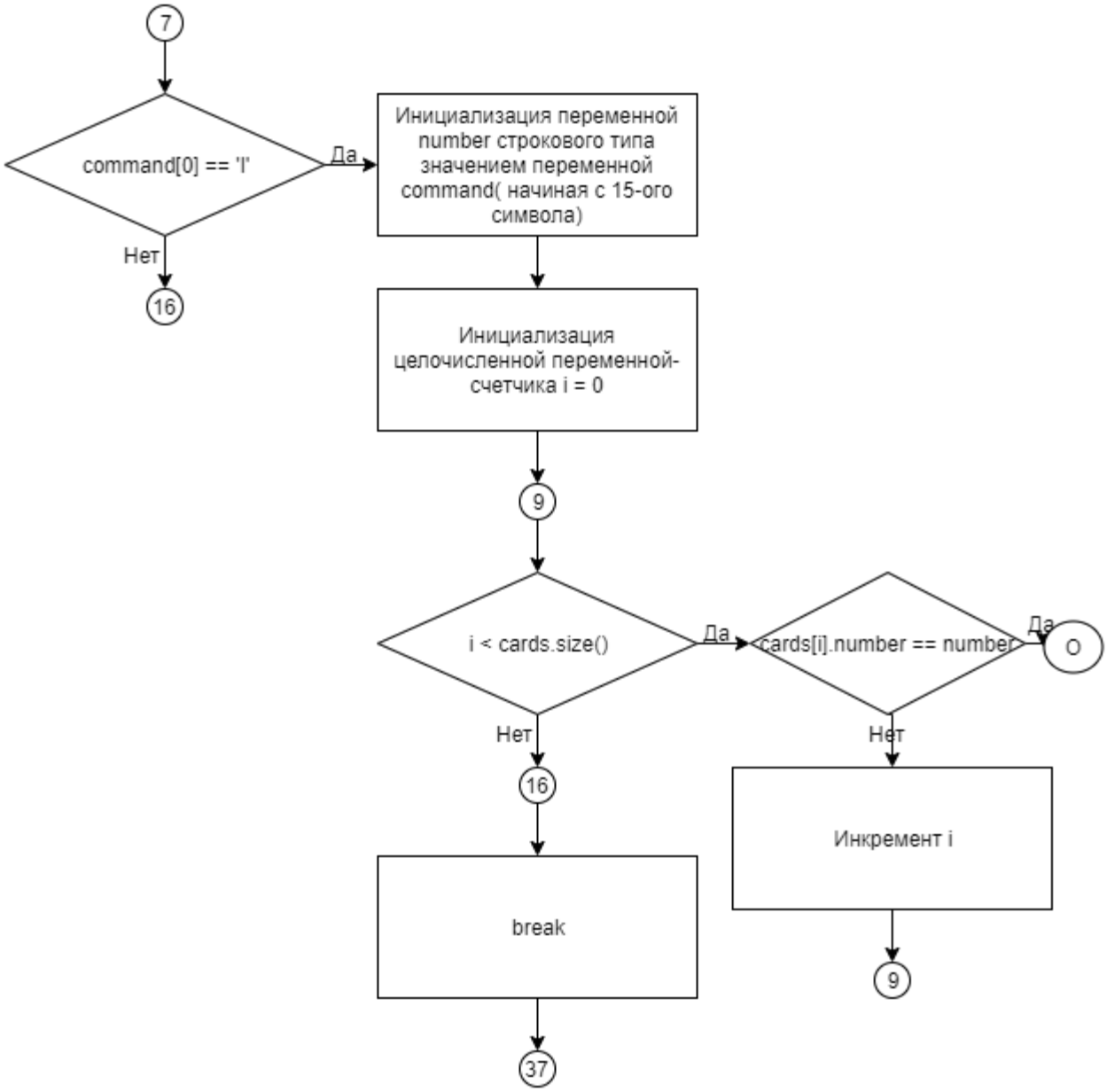

Рисунок 20 – Блок-схема алгоритма

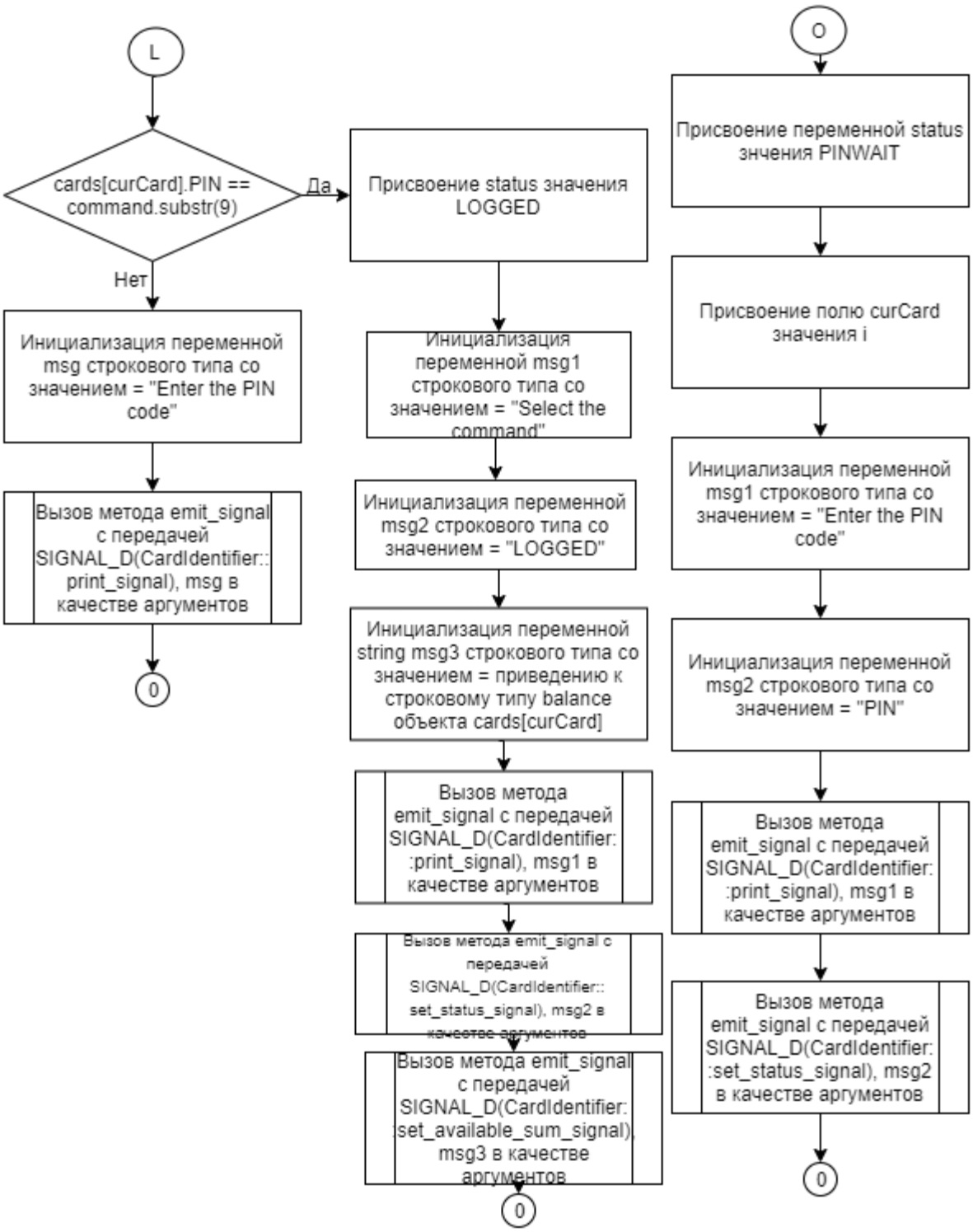

Рисунок 21 – Блок-схема алгоритма

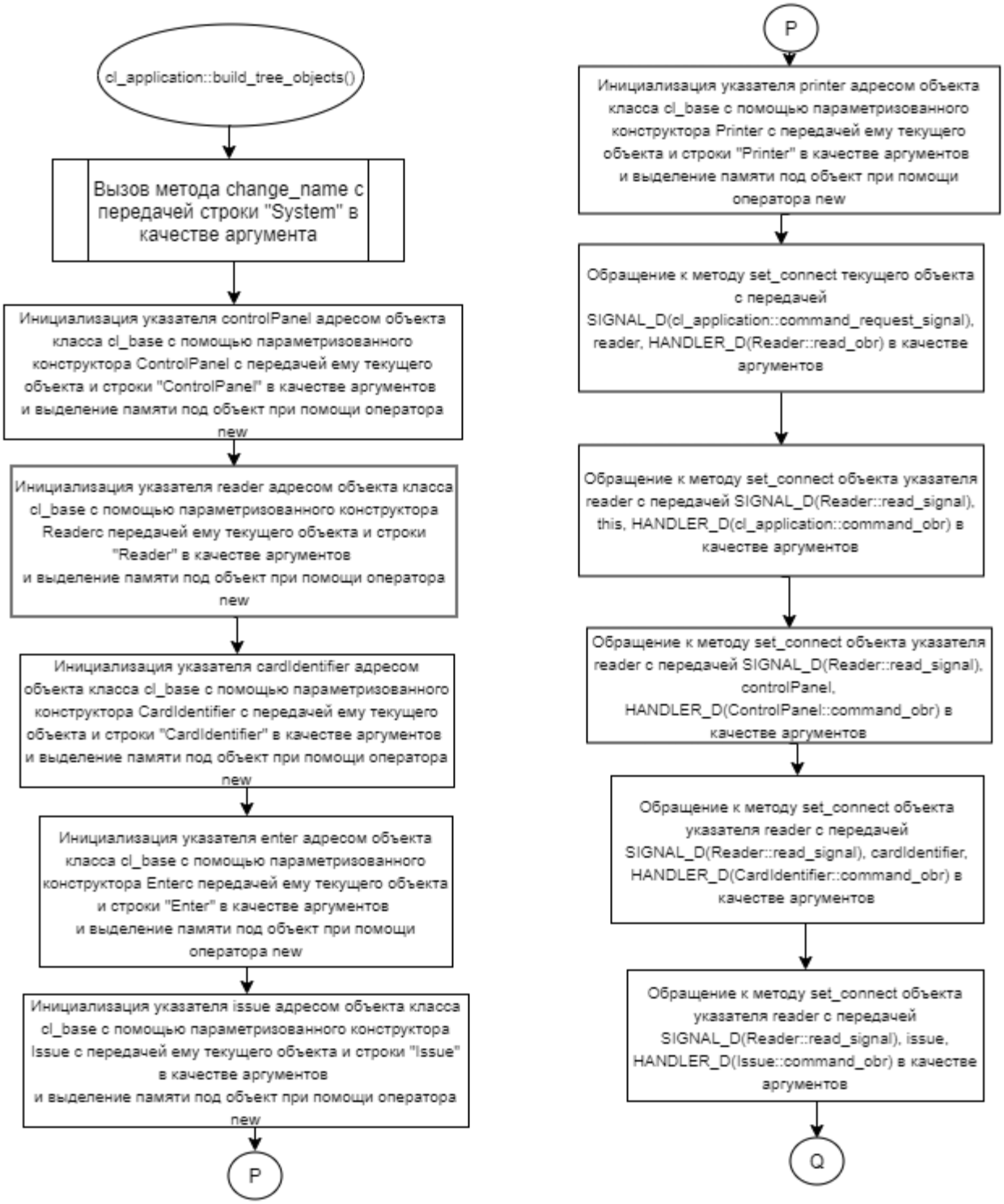

Рисунок 22 – Блок-схема алгоритма

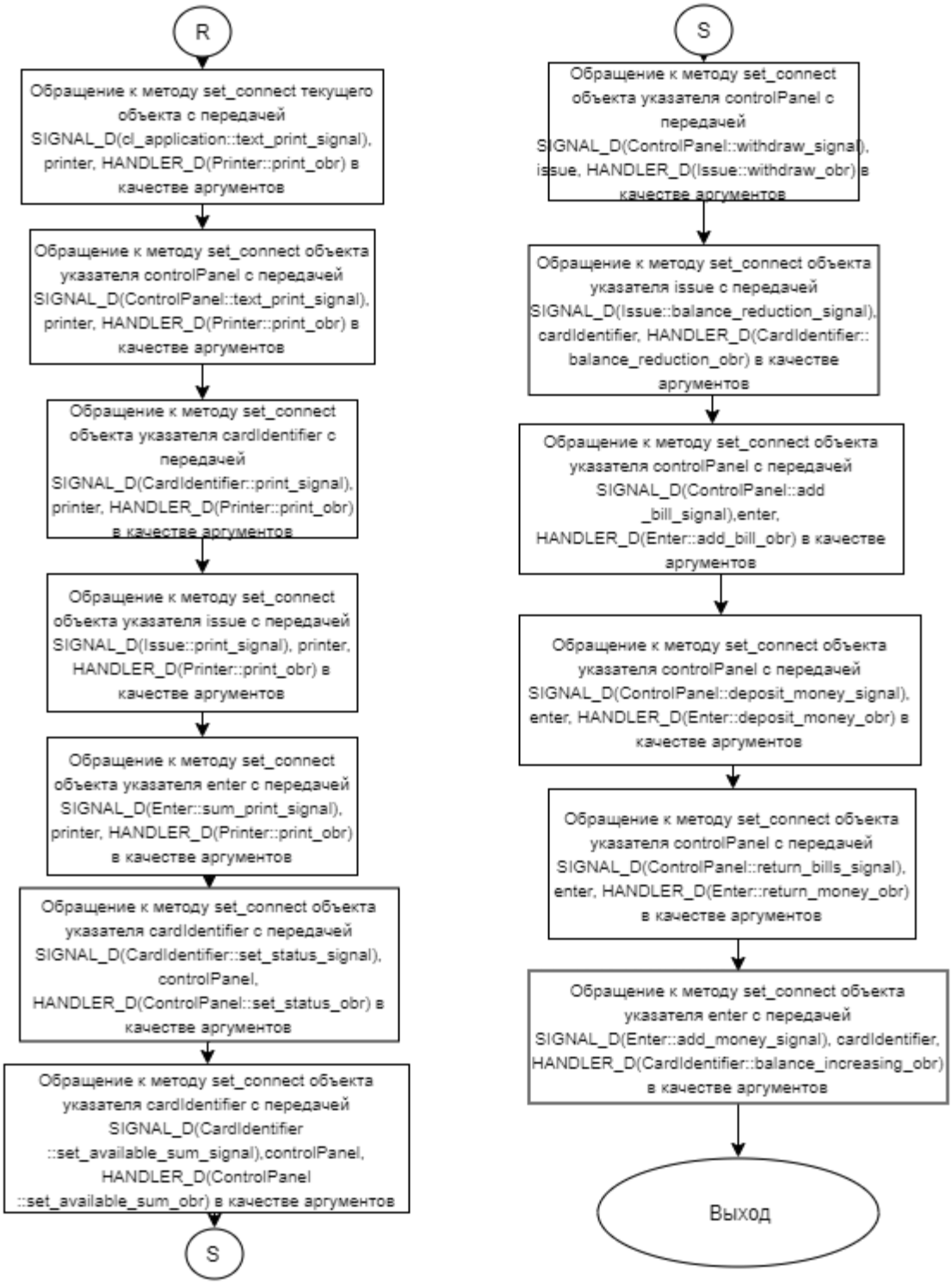

Рисунок 23 – Блок-схема алгоритма

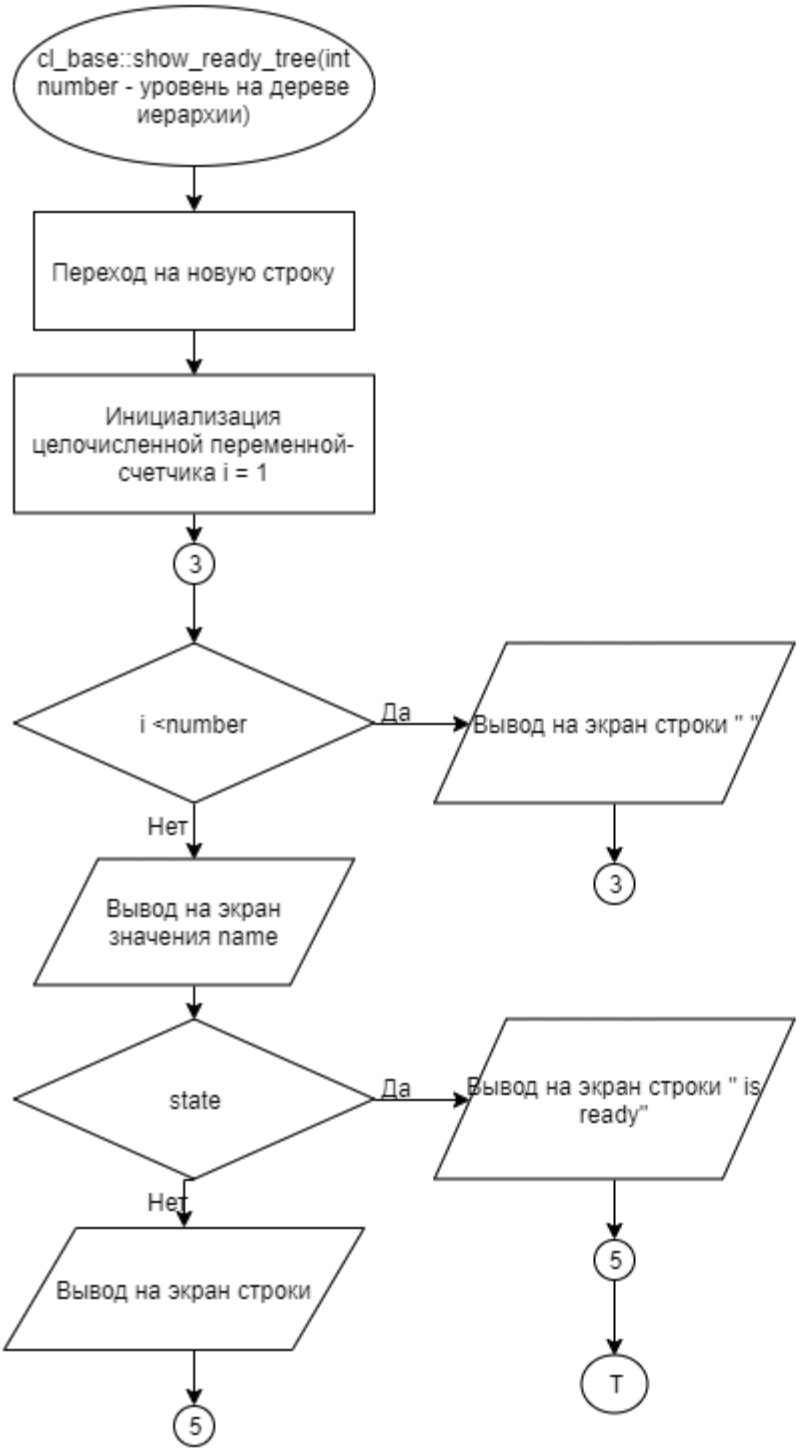

Рисунок 24 – Блок-схема алгоритма

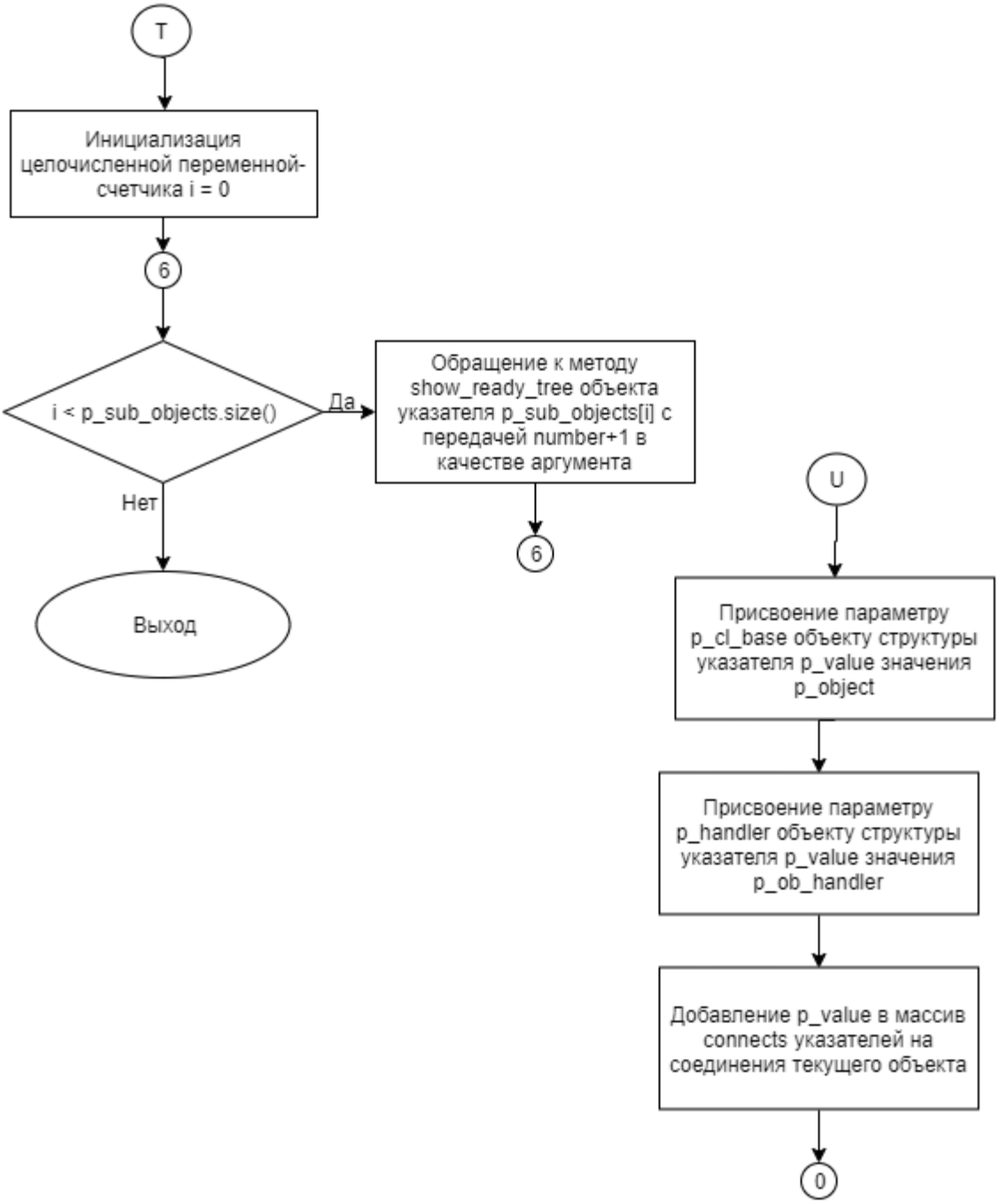

Рисунок 25 – Блок-схема алгоритма

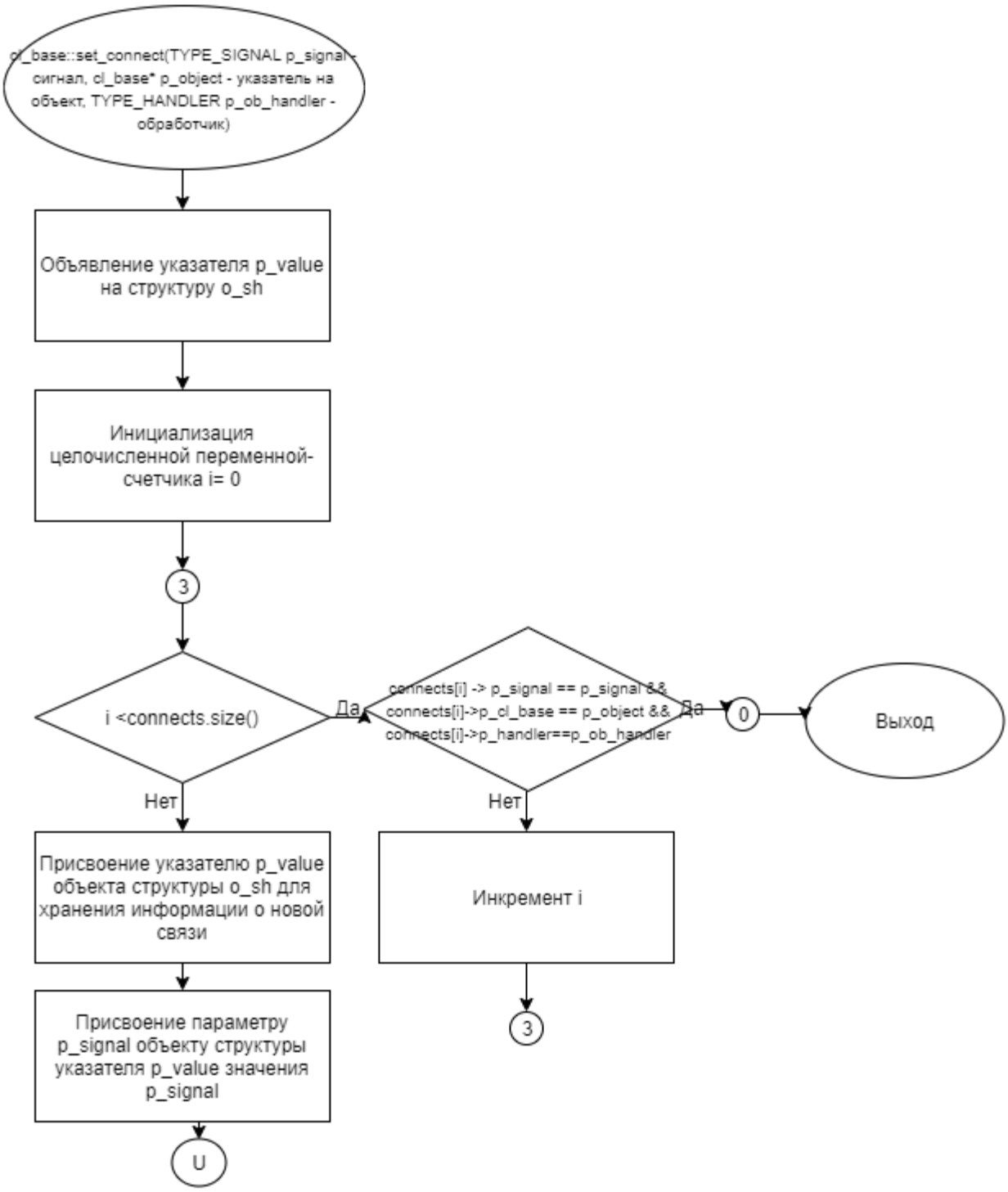

Рисунок 26 – Блок-схема алгоритма

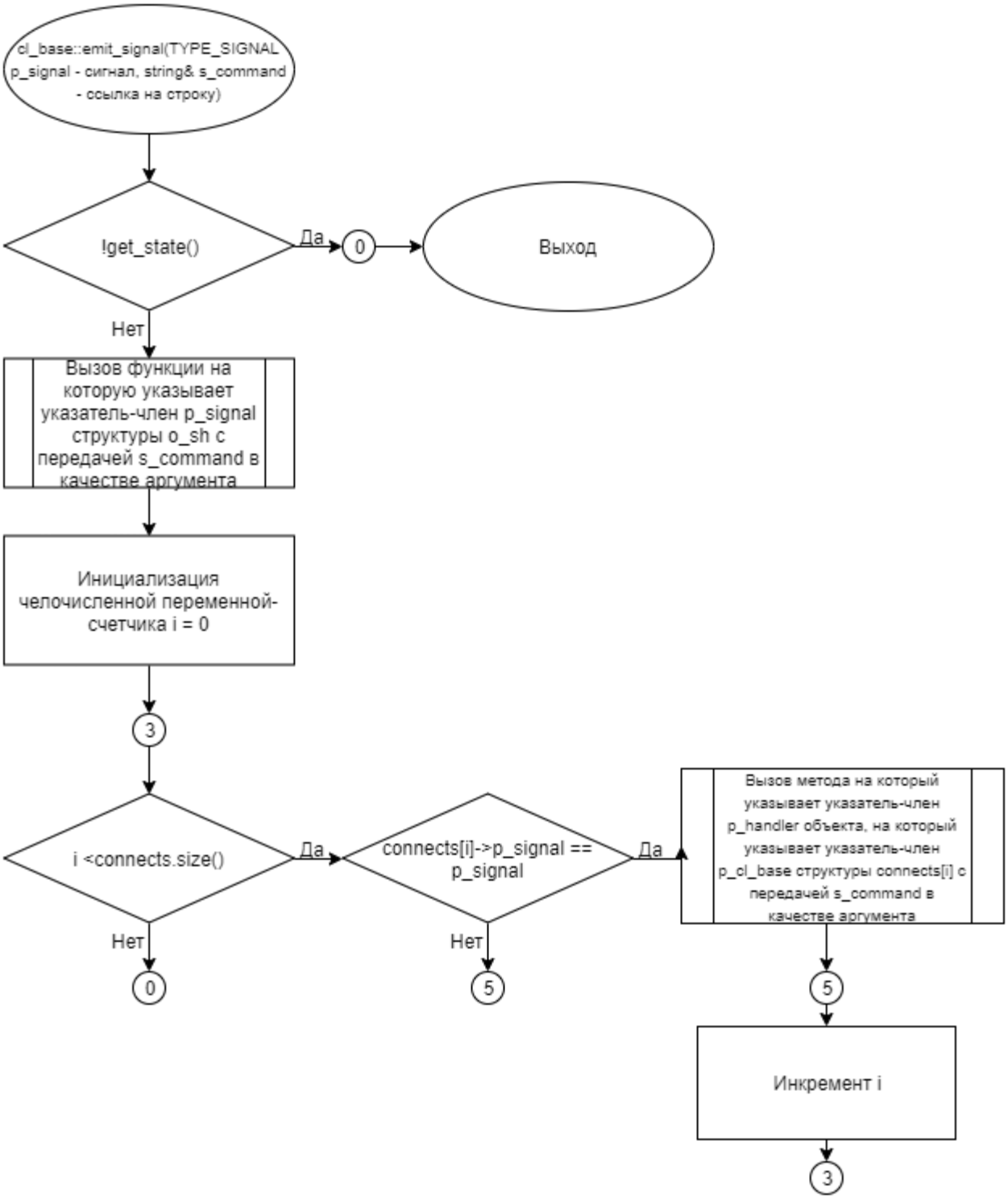

Рисунок 27 – Блок-схема алгоритма

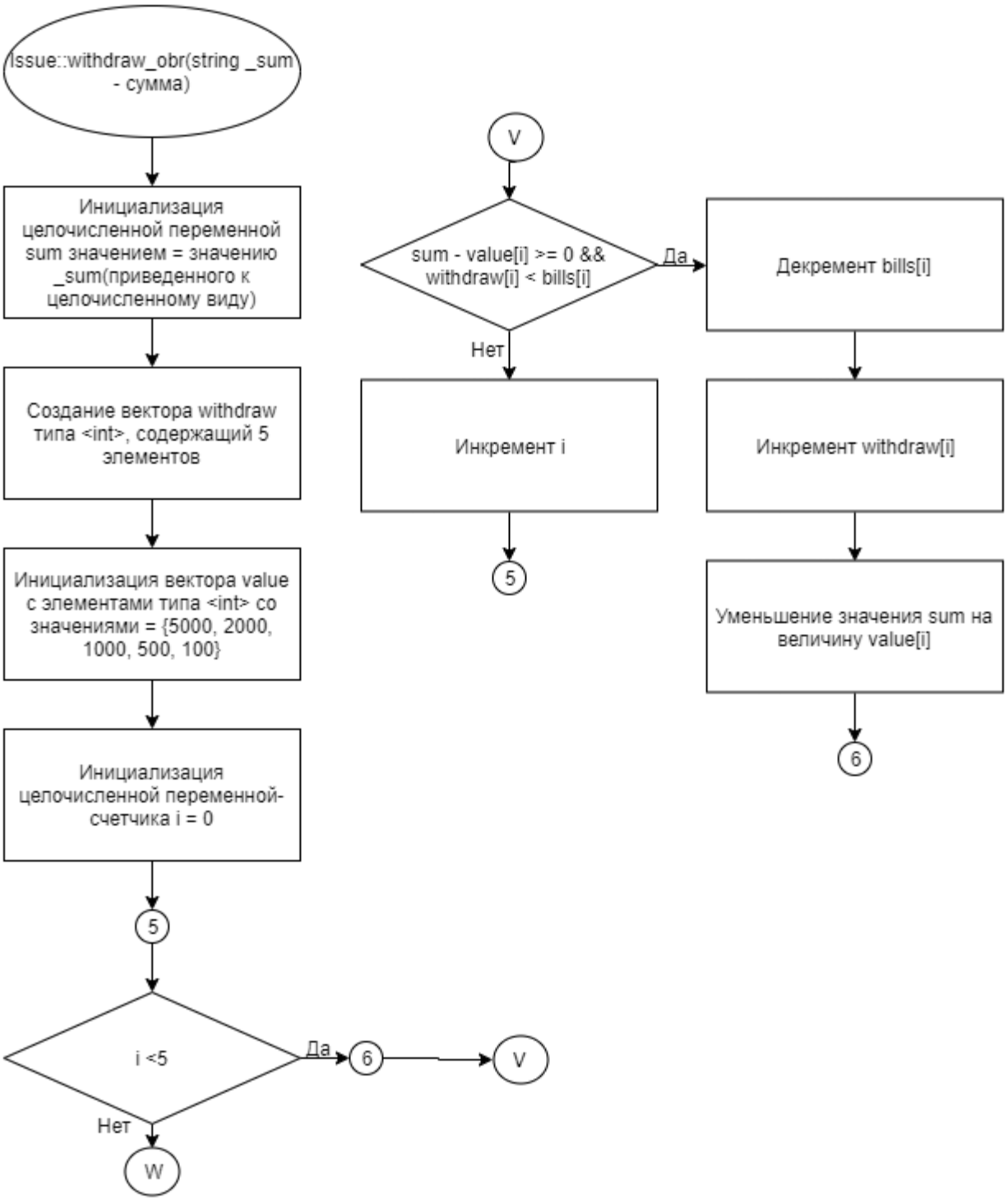

Рисунок 28 – Блок-схема алгоритма

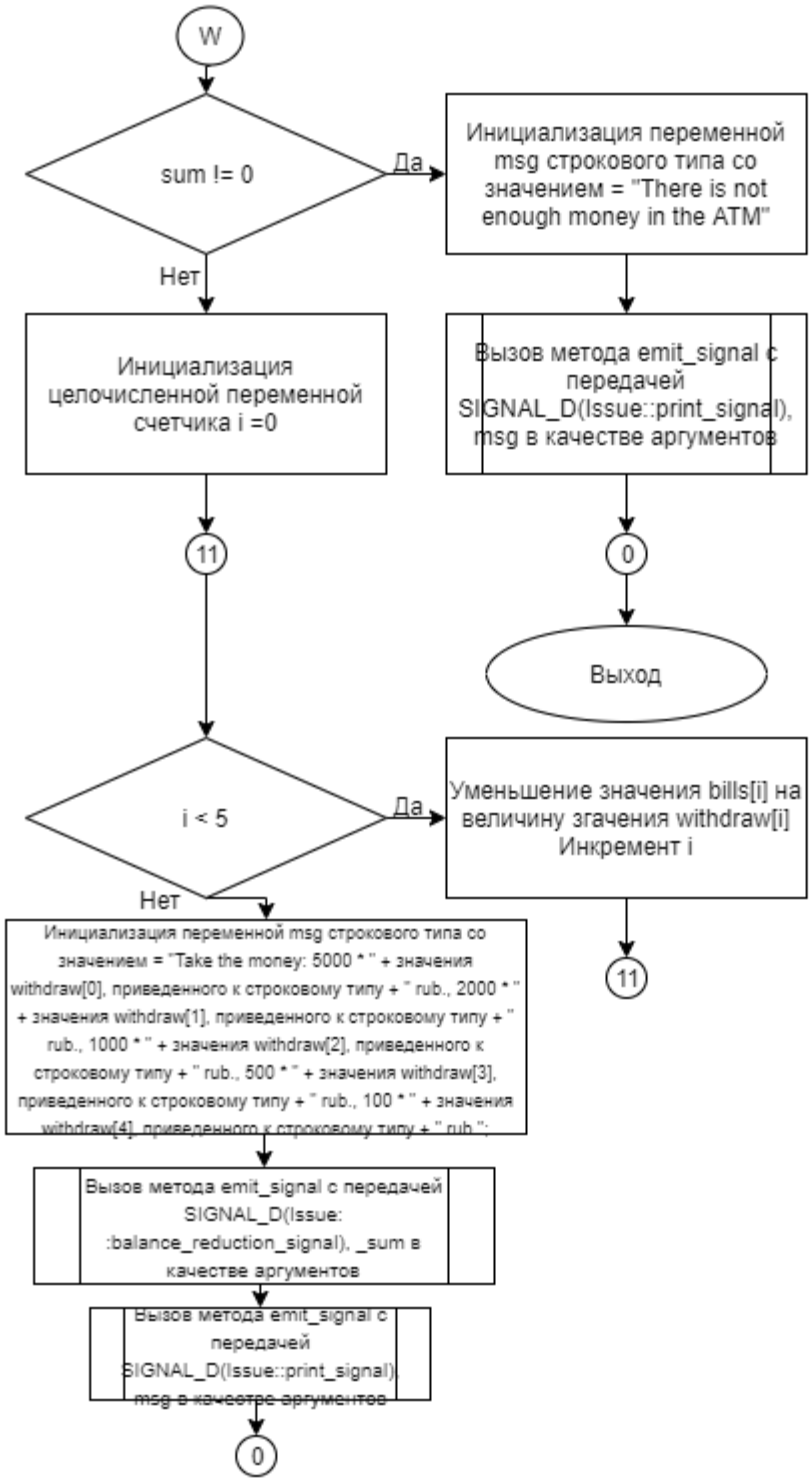

Рисунок 29 – Блок-схема алгоритма

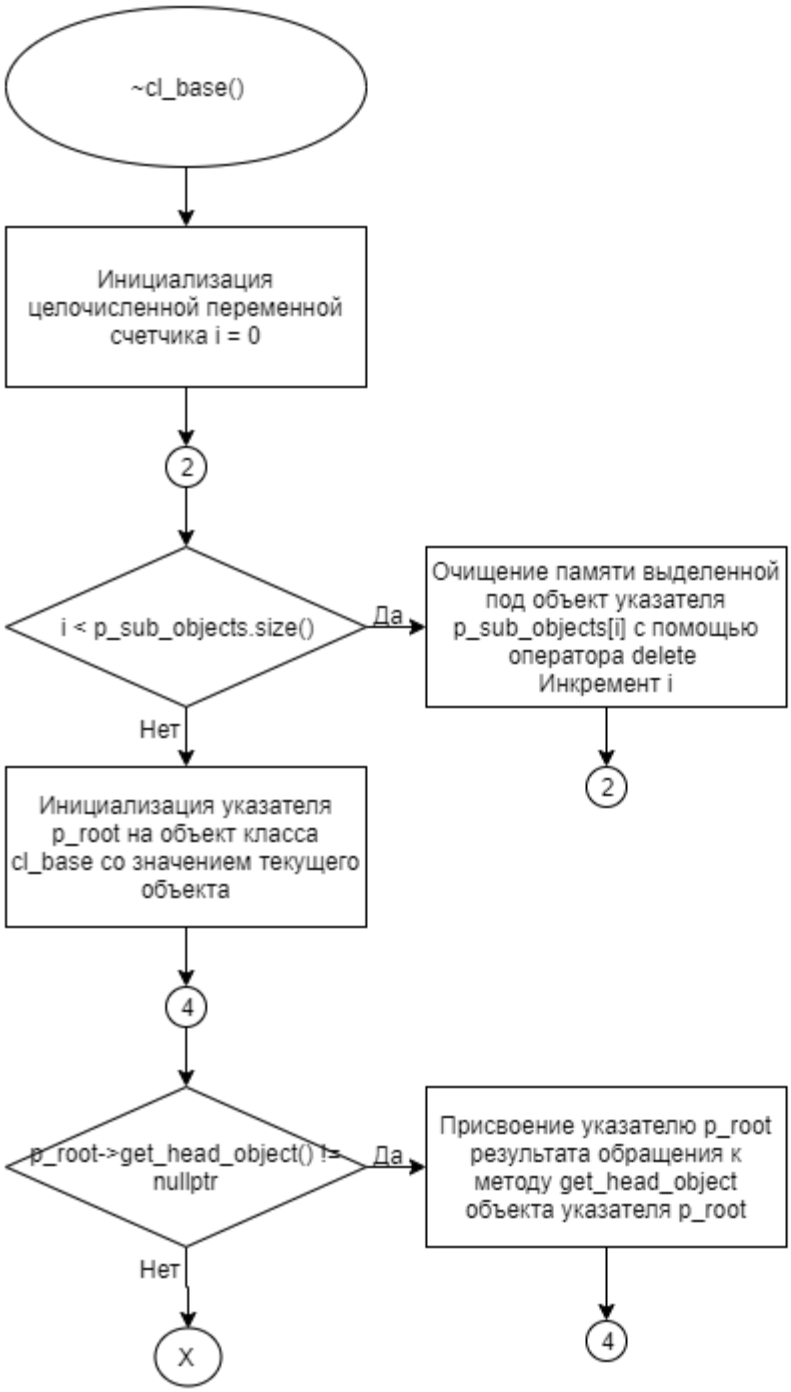

Рисунок 30 – Блок-схема алгоритма

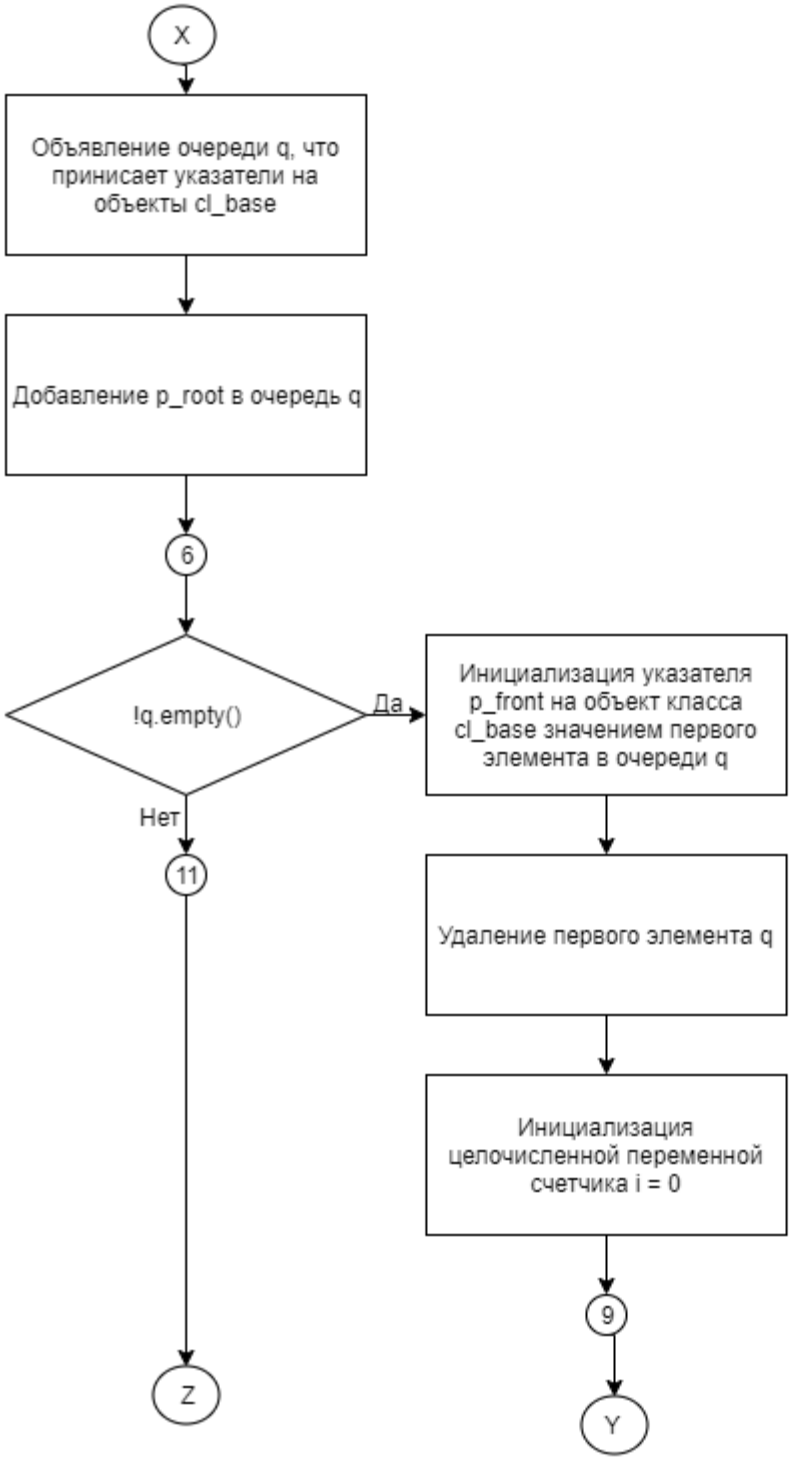

Рисунок 31 – Блок-схема алгоритма

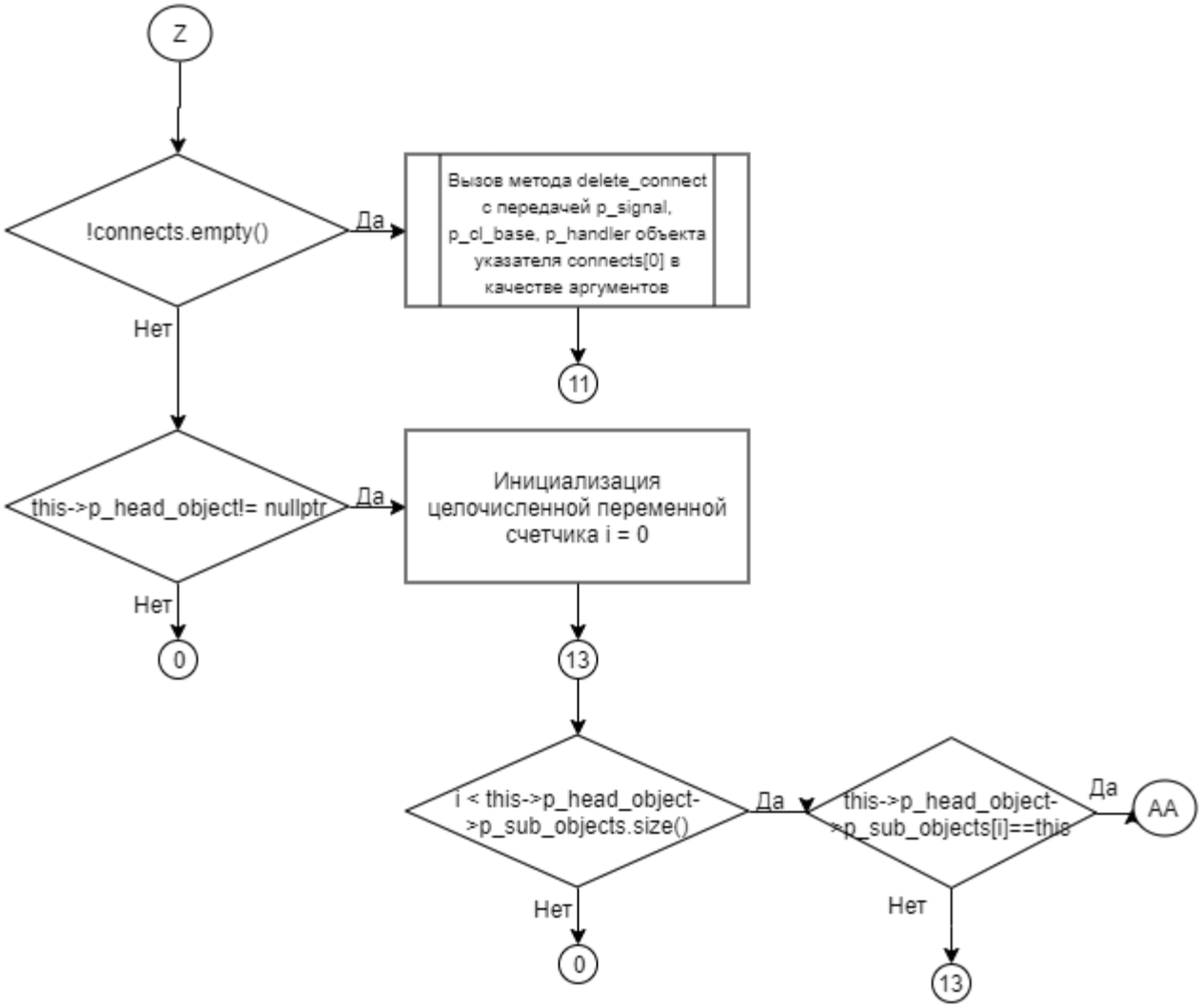

Рисунок 32 – Блок-схема алгоритма

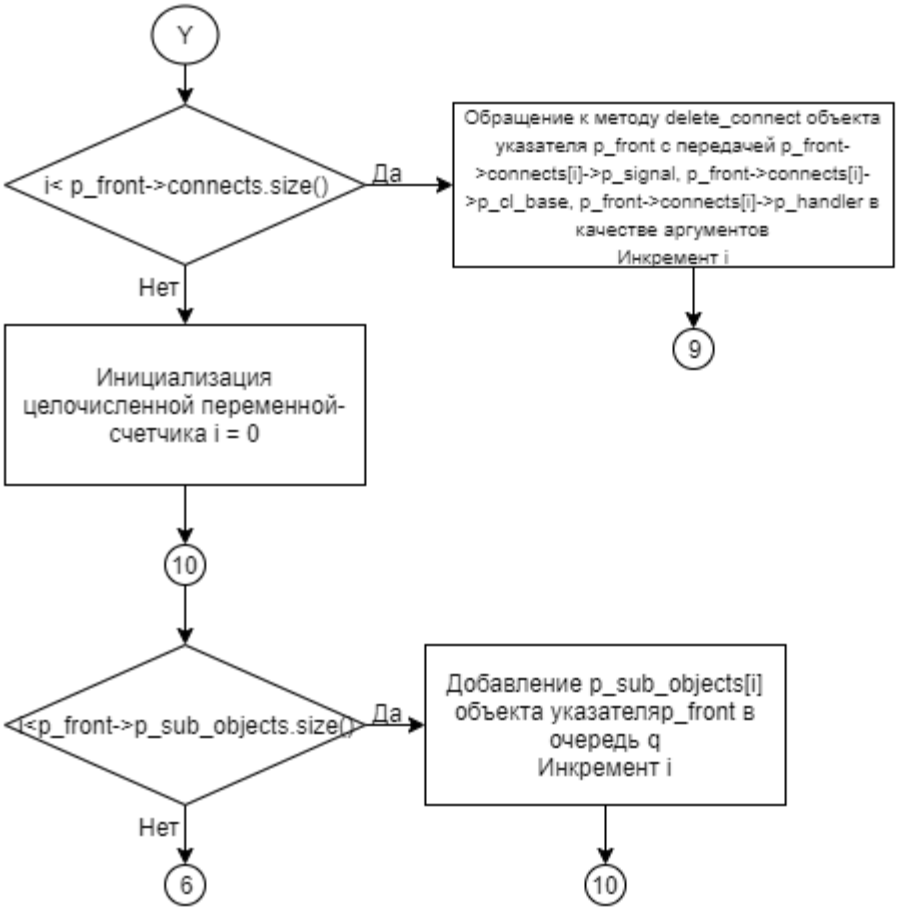

Рисунок 33 – Блок-схема алгоритма

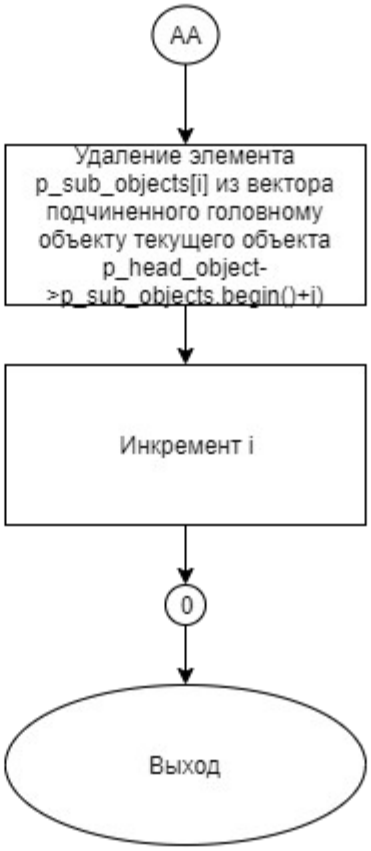

Рисунок 34 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл CardIdentifier.cpp

Листинг 1 – CardIdentifier.cpp

```
#include "CardIdentifier.h"
using namespace std;
// конструктор класса CardIdentifier
CardIdentifier::CardIdentifier(cl_base* parent, string name):cl_base(parent,
name){};
// Метод для обработки комманд
void CardIdentifier::command_obr(string command){
   switch(status){
         // обработка состояния загрузки денег
      case MONEYLOADING:
         if(command == "End money loading")
            status = WAITING;
         break;
      // обработка состояния ожидания
      case WAITING:
         // проверка, что команда не является идентификацией, завершением
сеанса, выключением или отображением дерева
         if(command[0] != 'I' && command[0] != 'E' && command[0] != 'T' &&
command[0] != 'S'){ // identification i end of the session i turn off i
showtree
            string ready = "Ready to work";
            emit_signal(SIGNAL_D(CardIdentifier::print_signal), ready);
         }
         // обработка команды идентификации
         if(command[0] == 'I'){
            string number = command.substr(15); // извлечение номера карты из
команды
            for(int i = 0; i < cards.size(); i++){
               if(cards[i].number == number){
                  status = PINWAIT; // переход в состояние ожидания ввода
ПИН-кода
                  curCard = i;
                  string msg1 = "Enter the PIN code";
                  string msg2 = "PIN";
                  emit_signal(SIGNAL_D(CardIdentifier::print_signal), msg1);
                  emit_signal(SIGNAL_D(CardIdentifier::set_status_signal),
msg2);

                  return;
```

```
                }
            }
        }
        break;

    // обработка состояния загрузки информаци о карте
    case READINFO:
        if(command == "End card loading"){
            status = MONEYLOADING;
            return;
        }
        // добавление новой карты в вектор
        cards.push_back({command.substr(0,   19),   command.substr(20,   4),
stoi(command.substr(25))});
        return;
    // обработка состояния ожидания ввода ПИН-кода
    case PINWAIT:
        if (command[0] == 'P'){
            if(cards[curCard].PIN   ==   command.substr(9)){   //   Проверка
корректности введенного ПИН-кода
                status = LOGGED; // ПЕреход в состояние "вход выполнен"
                string msg1 = "Select the command";
                string msg2 = "LOGGED";
                string msg3 = to_string(cards[curCard].balance);
                emit_signal(SIGNAL_D(CardIdentifier::print_signal), msg1);
                emit_signal(SIGNAL_D(CardIdentifier::set_status_signal),
msg2);


   emit_signal(SIGNAL_D(CardIdentifier::set_available_sum_signal), msg3);
                return;
            }
            string msg = "Enter the PIN code";
            emit_signal(SIGNAL_D(CardIdentifier::print_signal), msg);
            return;
        }
        // сброс текущей карты и переход в состояние ожидания
        curCard = -1;
        status = WAITING;
        string msg1 = "READY";
        string msg2 = "Ready to work";
        emit_signal(SIGNAL_D(CardIdentifier::set_status_signal), msg1);
        emit_signal(SIGNAL_D(CardIdentifier::print_signal), msg2);
        break;
   }
   // обработка команды завершения сеанса
   if(command == "End the session"){
      curCard = -1;
      status = WAITING;
      string msg1 = "READY";
      string msg2 = "Ready to work";
      emit_signal(SIGNAL_D(CardIdentifier::set_status_signal), msg1);
      emit_signal(SIGNAL_D(CardIdentifier::print_signal), msg2);
   }
}
```

```
// метод-сигнал для печати
void CardIdentifier::print_signal(string&){}
// метод для увеличения баланса карты
void CardIdentifier::balance_increasing_obr (string sum){
   cards[curCard].balance += atoi(sum.c_str()); // увеличение баланса текущей
карты
   string msg = "Card balance " + to_string(cards[curCard].balance);
   emit_signal(SIGNAL_D(CardIdentifier::print_signal), msg);
}
// метод для уменьшения баланса карты
void CardIdentifier::balance_reduction_obr(string sum){
   cards[curCard].balance -= atoi(sum.c_str());
}
// меотд-сигнал для установки доступной суммы
void CardIdentifier::set_available_sum_signal(string&){}
// метод-сигнал для установки статуса
void CardIdentifier::set_status_signal(string&){}
```

5.2 Файл CardIdentifier.h

Листинг 2 – CardIdentifier.h

```
#ifndef __CARDIDENTIFIER__H
#define __CARDIDENTIFIER__H
#include "cl_base.h"
using namespace std;
// определение класса CardIdentifier, который наследуется от cl_base
class CardIdentifier : public cl_base{
   private:
   // перечисление для обозначения различных состояний
   enum STATUS{READINFO, MONEYLOADING, WAITING, PINWAIT, LOGGED};
   // структура для хранения информации о карте
   struct card{
      string number; // номер карты
      string PIN; // ПИН-код карты
      int balance; // Баланс карты
   };
   STATUS status = READINFO; // текущее состояние
   int curCard; // индекс текущей карты
   vector<card> cards; // вектор для зранения всех карт
   public:
   // крнструктор класса
   CardIdentifier(cl_base* parent, string name);
   // обработчик команды
   void command_obr(string);
   // сигнал для установки статуса
   void set_status_signal(string&);
   // обработчик для увеличения баланса
   void balance_increasing_obr(string);
   // сигнал для установки доступной суммы
```

```
    void set_available_sum_signal(string&);
    // сигнал для печати
    void print_signal(string&);
    // обработчик для уменьшения баланса
    void balance_reduction_obr(string);
};



#endif
```

5.3 Файл cl_application.cpp

Листинг 3 – cl_application.cpp

```
#include "cl_application.h"
#include "cl_base.h"
#include "CardIdentifier.h"
#include "ControlPanel.h"
#include "Enter.h"
#include "Issue.h"
#include "Reader.h"
#include "Printer.h"
#include <string>
#include <iostream>
#include <map>
using namespace std;

// конструктор класса cl_application, вызывающий конструктор базового класса
cl_base
cl_application::cl_application(cl_base*         p_head_object,         string
name):cl_base(p_head_object, name){
}
// пустая реализация сигнала запроса команды
void cl_application::command_request_signal(string&){}

// обработчик команды, который сохраняет команду в члене класса command
void cl_application::command_obr(string _command) {
   command = _command;
}
// пустая реализация сигнала печати текста
void cl_application::text_print_signal(string&) {}

// метод для построения дерева объектов
void cl_application::build_tree_objects(){
   // изменения имени объекта на "System"
   change_name("System");

   // создание объектов и добавление их к дереву объектов
   cl_base* controlPanel = new ControlPanel(this, "ControlPanel");
```

```
   cl_base* reader = new Reader(this, "Reader");
   cl_base* cardIdentifier = new CardIdentifier(this, "CardIdentifier");
   cl_base* enter = new Enter(this, "Enter");
   cl_base* issue = new Issue(this, "Issue");
   cl_base* printer = new Printer(this, "Printer");

   // установка соединения между сигналом и обработчиком
   // строка кода ниже устанавливает связь между сигналом и обработчиком.
Когда сигнал "command_request_signal" будет отправлен из объекта
"cl_application", он будет напрвлен к обработчику "read_obr" объекта
"reader"
   this->set_connect(SIGNAL_D(cl_application::command_request_signal),
reader, HANDLER_D(Reader::read_obr));
   reader->set_connect(SIGNAL_D(Reader::read_signal),                  this,
HANDLER_D(cl_application::command_obr));
   reader->set_connect(SIGNAL_D(Reader::read_signal),          controlPanel,
HANDLER_D(ControlPanel::command_obr));
   reader->set_connect(SIGNAL_D(Reader::read_signal),        cardIdentifier,
HANDLER_D(CardIdentifier::command_obr));
   reader->set_connect(SIGNAL_D(Reader::read_signal),                 issue,
HANDLER_D(Issue::command_obr));
   this->set_connect(SIGNAL_D(cl_application::text_print_signal),   printer,
HANDLER_D(Printer::print_obr));
   controlPanel->set_connect(SIGNAL_D(ControlPanel::text_print_signal),
printer, HANDLER_D(Printer::print_obr));
   cardIdentifier->set_connect(SIGNAL_D(CardIdentifier::print_signal),
printer, HANDLER_D(Printer::print_obr));
   issue->set_connect(SIGNAL_D(Issue::print_signal),                printer,
HANDLER_D(Printer::print_obr));
   enter->set_connect(SIGNAL_D(Enter::sum_print_signal),            printer,
HANDLER_D(Printer::print_obr));
   cardIdentifier->set_connect(SIGNAL_D(CardIdentifier::set_status_signal),
controlPanel, HANDLER_D(ControlPanel::set_status_obr));
   cardIdentifier-
>set_connect(SIGNAL_D(CardIdentifier::set_available_sum_signal),
controlPanel, HANDLER_D(ControlPanel::set_available_sum_obr));
   controlPanel->set_connect(SIGNAL_D(ControlPanel::withdraw_signal),  issue,
HANDLER_D(Issue::withdraw_obr));
   issue->set_connect(SIGNAL_D(Issue::balance_reduction_signal),
cardIdentifier, HANDLER_D(CardIdentifier::balance_reduction_obr));
   controlPanel->set_connect(SIGNAL_D(ControlPanel::add_bill_signal),  enter,
HANDLER_D(Enter::add_bill_obr));
   controlPanel->set_connect(SIGNAL_D(ControlPanel::deposit_money_signal),
enter, HANDLER_D(Enter::deposit_money_obr));
   controlPanel->set_connect(SIGNAL_D(ControlPanel::return_bills_signal),
enter, HANDLER_D(Enter::return_money_obr));
   enter->set_connect(SIGNAL_D(Enter::add_money_signal),      cardIdentifier,
HANDLER_D(CardIdentifier::balance_increasing_obr));
}

//  метод для выполнения приложения
int cl_application::exec_app(){
   // активация всех объектов
   enable_all();
   // вывод сообщения о готовности к работе
```

```
        string ready = "Ready to work";
        emit_signal(SIGNAL_D(cl_application::text_print_signal), ready);

        // запуск цикла для обработки ввода команд до команды "End card loading"
        do{
           emit_signal(SIGNAL_D(cl_application::command_request_signal), command);
        }while (command != "End card loading");
        // Обрабатываем команду завершения загрузки карт
        emit_signal(SIGNAL_D(cl_application::command_request_signal), command);
        // обрабатываем команду завершения загрузки денег
        emit_signal(SIGNAL_D(cl_application::command_request_signal), command);
        // основной цикл обработки команд
        while(true){
           emit_signal(SIGNAL_D(cl_application::command_request_signal), command);

           // если получена команда "Turn off the ATM", завершаем работу
           if(command == "Turn off the ATM"){
              emit_signal(SIGNAL_D(cl_application::text_print_signal), command);
              break;
           }
           //  если  получена  команда  "SHOWTREE",  выводим  дерево  объектов  и
завершаем работу
           if(command == "SHOWTREE"){
              string buf = "";
              emit_signal(SIGNAL_D(cl_application::text_print_signal), buf);
              show_ready_tree(0);
              break;
           }
        }
        return 0;
     }
```

5.4 Файл cl_application.h

Листинг 4 – cl_application.h

```
     #ifndef __CL_APPLICATION__H
     #define __CL_APPLICATION__H
     #include "cl_base.h"
     #include <string>
     using namespace std;
     class cl_application: public cl_base{
        private:
        string command; // строка для хранения команды
        public:
        // конструктор класса
        cl_application(cl_base*   p_head_object   =   nullptr,   string   name   =
     "Base_object");
        // метод для построения дерева объектов
        void build_tree_objects();
```

```
   // метод для выполнения программы
   int exec_app();
   // сигнал для запроса команды
   void command_request_signal(string&);
   // обработчик для обработки команды
   void command_obr(string);
   // сигнал для печати текста
   void text_print_signal(string&);
};


#endif
```

5.5 Файл cl_base.cpp

Листинг 5 – cl_base.cpp

```
#include "cl_base.h"
#include "cl_application.h"
#include "CardIdentifier.h"
#include "ControlPanel.h"
#include "Enter.h"
#include "Issue.h"
#include "Reader.h"
#include <string>
#include <iostream>
#include <algorithm>
#include <sstream>
using namespace std;

// Конструктор класса cl_base
cl_base::cl_base(cl_base* p_head_object, string name){
   this->p_head_object = p_head_object; // установка головного объекта
   this->name = name; // установка имени объекта
   if(p_head_object){
      p_head_object->p_sub_objects.push_back(this);  //  добавление  текущего
объекта в список подчиненных объектов головного объекта
   }
}

// Деструктор класса cl_base
cl_base::~cl_base(){
   // удаление всех подчиненных объектов
   for(int i = 0; i < p_sub_objects.size(); i++){
      delete p_sub_objects[i];
   }
   // нахождение корневого объекта
   cl_base* p_root = this;
   while(p_root->get_head_object() != nullptr){
      p_root = p_root->get_head_object();}
```

```
   // очередь для обхода всех объектов дерева
   queue<cl_base*> q;
   q.push(p_root);
   while(!q.empty()){
      cl_base* p_front = q.front();
      q.pop();
      // удаление соединений
      for(int i = 0; i <p_front->connects.size(); i++){
         p_front->delete_connect(p_front->connects[i]->p_signal,     p_front-
>connects[i]->p_cl_base, p_front->connects[i]->p_handler);
      }
      // добавление подчиненных объектов в очередь
      for(int i = 0; i < p_front->p_sub_objects.size(); i++){
         q.push(p_front->p_sub_objects[i]);
      }
   }
   // Удаление оставшихся соединений
   while(!connects.empty()){
      delete_connect(connects[0]->p_signal,           connects[0]->p_cl_base,
connects[0]->p_handler);
   }
   // удаление текущего объекта из списка подчиненных объектов головного
объекта
   if(this->p_head_object != nullptr){
      for(int i = 0; i < this->p_head_object->p_sub_objects.size();i++){
         if(this->p_head_object->p_sub_objects[i]==this){
            this->p_head_object->p_sub_objects.erase(this->p_head_object-
>p_sub_objects.begin()+i);
            break;
         }
      }
   }
}
// метод изменения имени объекта
bool cl_base::change_name(string new_name){
   // проверка на уникальность имени среди подчиненных объектов
   if(p_head_object != nullptr){
      for(int i = 0; i < p_head_object->p_sub_objects.size(); i++){
         if(p_head_object->p_sub_objects[i]->get_name() == new_name){
            return false; // имя уже занято
         }
      }
   }
   this->name=new_name; // установка нового имени
   return true;
}

// метод получения имени объекта
string cl_base::get_name(){
   return this->name; // возврат имен объекта
}
// меотд получения головного объекта
cl_base* cl_base::get_head_object(){
   return this->p_head_object; // возврат головного объекта
}
```

```
/*
void cl_base::show_object_tree(){
   if(p_sub_objects.size()>0){
      cout << endl << get_name() << " ";
      for (int i = 0; i < p_sub_objects.size(); i++){
         cout << p_sub_objects[i]->get_name();
         if(p_sub_objects.size() - 1 != i){
            cout << "  ";
         }
      }
      for(int i = 0; i <p_sub_objects.size();i++){
         p_sub_objects[i]->show_object_tree();
      }
   }
}
*/
// метод получения подчиненного объекта по имени
cl_base* cl_base::get_sub_object(string name){
   // поиск среди непосредственных подчиненных объектов
   for(int i = 0; i <p_sub_objects.size(); i++){
      if(p_sub_objects[i]->get_name() == name){
         return  p_sub_objects[i];  //  возвращает  подчиненный  объект  с
указанным именем
      }
   }
   // рекурсивный поиск среди подчиненных объектов подчиненных объектов
   for(int i = 0;i <p_sub_objects.size();i++){
      if(p_sub_objects[i]->get_sub_object(name) != nullptr){
         return p_sub_objects[i]->get_sub_object(name);
      }
   }
   return nullptr; // возврат nullptr, если объект не найден
}

/*cl_base* cl_base::search_from_current(string name){
   if(this->name==name){
      return this;
   }
   for(int i = 0;i <p_sub_objects.size(); i++){
      cl_base* found = p_sub_objects[i]->search_from_current(name);
      if(found != nullptr){
         return found;
      }
   }
   return nullptr;
}*/

// МЕод поиска объекта по имени
cl_base* cl_base::search_by_name(string name){
   int i = 0;
   vector<cl_base*>object(1, this);
   cl_base *p = nullptr;
   // поиск в ширину среди всех объектов дерева
   while(object.size()>0){
      cl_base* temp = object.back();
```

```
            object.pop_back();
            if(temp->name == name){
               p = temp;
               i++;
            }
            for(int j = 0; j < temp->p_sub_objects.size(); j++){
               object.push_back(temp->p_sub_objects[j]);
            }
         }
         if(i < 2){
            return p; // возвращает первый найденный объект или nullptr, если
      найдено больше одного
         }
         return nullptr;
      }

      /*void cl_base::show_from_current_tree(int number){
         cout << endl;
         for(int i = 1; i <number; i++){
            cout << "    ";
         }
         cout << name;
         for(int i = 0; i < p_sub_objects.size(); i++){
            p_sub_objects[i]->show_from_current_tree(number+1);}
      }*/

      // метод отображения дерева готовности объектов
      void cl_base::show_ready_tree(int number){
         //cout << endl;
         // Отступы для отображения иерархии
         for(int i = 1; i <number;i++){
            cout << "    ";
         }
         cout <<name; // вывод имени объектов
         if(state){
            cout << " is ready" << endl;} // вывод состояния готовности объекта
         else{
            cout << " is not ready" << endl;} // вывод состояния неготовности
      объекта
         // рекурсивный вывод подчиненных объектов
         for(int i = 0; i < p_sub_objects.size(); i++){
            p_sub_objects[i]->show_ready_tree(number+1);
         }
      }
      /*void cl_base::set_state(int number){
         if(p_head_object&& ! (p_head_object->state)){
            return;
         }
         if(number){
            state = number;
         }
         else{
            for(int i = 0; i <p_sub_objects.size(); i++){
               p_sub_objects[i]->set_state(0);
            }
```

```
      state = number;
   }
}*/

/*bool cl_base::set_head_object(cl_base* p_head_object){
   if(p_head_object == nullptr){
      return false;
   }
   cl_base* temp = p_head_object;
   while(temp){
      temp = temp->p_head_object;
      if(this==temp){
         return false;
      }
   }
   if(((p_head_object->get_sub_object(get_name())      ==      nullptr      ||
p_head_object->get_sub_object(get_name())->get_head_object()!=p_head_object)
&& p_head_object!= nullptr)){
      p_head_object->p_sub_objects.erase(find(p_head_object-
>p_sub_objects.begin(), p_head_object->p_sub_objects.end(), this));
      p_head_object ->p_sub_objects.push_back(this);
      this->p_head_object = p_head_object;
      return true;
   }
   return false;
}*/

/*void cl_base::delete_sub_object(string name){
   cl_base* delete_obj = get_sub_object(name);
   if(delete_obj && delete_obj->get_head_object() == this){
      p_sub_objects.erase(find(p_sub_objects.begin(),  p_sub_objects.end()-1,
delete_obj));
      delete delete_obj;
   }
}*/

/*cl_base* cl_base::get_object_by_path(string path){
   if(path == ""){
      return nullptr;
   }
   cl_base* head_obj = this;
   while(head_obj->get_head_object()){
      head_obj = head_obj->get_head_object();
   }
   string path_item;
   if(path == "/"){
      return head_obj;
   }
   if(path == "."){
      return this;
   }
   else if(path[0] == '.'){
      path.erase(path.begin());
      return search_by_name(path);
   }
```

```
   else if(path[0] = '/' && path[1] == '/'){
      path.erase(path.begin());
      path.erase(path.begin());
      return head_obj->search_by_name(path);
   }
   else if(path[0] == '/'){
      path.erase(path.begin());
   }
   stringstream temp_path(path);
   while(getline(temp_path, path_item, '/')){
      head_obj = head_obj->get_sub_object(path_item);
      if(!head_obj){
         return nullptr;
      }
   }
      return head_obj;
}*/

// меотд установки соединения
void   cl_base::set_connect(TYPE_SIGNAL   p_signal,   cl_base*   p_object,
TYPE_HANDLER p_ob_handler){
   o_sh* p_value;
   // проверка на существованиие такого же соединения
   for(int i= 0; i <connects.size(); i++){
      if(connects[i] -> p_signal == p_signal && connects[i]->p_cl_base ==
p_object && connects[i]->p_handler==p_ob_handler){
         return; // соединение уже существует
      }
   }
      p_value = new o_sh; // создаем объект структуры соединения
      p_value->p_signal=p_signal; // установка сигнала
      p_value->p_cl_base=p_object; // установка объекта
      p_value->p_handler = p_ob_handler; // установка обработчика
      connects.push_back(p_value);  //  добавление  новой  связи  в  список
соединений
}

// метод удаления соединения
void   cl_base::delete_connect(TYPE_SIGNAL   p_signal,   cl_base*   p_object,
TYPE_HANDLER p_ob_handler){
   // o_sh* p_value;
   // поиск и удаление соединения
   for(int i = 0; i <connects.size(); i++){
      if(connects[i]->p_signal  ==  p_signal  &&  connects[i]->p_cl_base  ==
p_object && connects[i]->p_handler == p_ob_handler){
         connects.erase(connects.begin()+i); // удаление связи
      }
   }
}

// метод отправки сигнала
void cl_base::emit_signal(TYPE_SIGNAL p_signal, string& s_command){
   if(!get_state()){
      return; // если объект не готов, сигнал не отправляется
   }
```

```
   (this->*p_signal)(s_command); // вызов сигнала у текущего объекта
   // вызов обработчиков у связанных объектов
   for(int i = 0; i <connects.size(); i++){
      if(connects[i]->p_signal == p_signal){
         (connects[i]->p_cl_base->*connects[i]->p_handler)(s_command);     //
выщов обработчика у связанного объекта
      }
   }
}

/*string cl_base::get_path(){
   vector<string> v;
   string path = "";
   cl_base* obj = this;
   while(obj->p_head_object){
      v.push_back(obj->name);
      obj = obj->p_head_object;
   }
   for(int i = v.size() - 1;i>= 0;i--){
      path=path+"/";
      path=path+v[i];
   }
   if(path.empty()){
      path="/";
   }
   return path;
}*/

// меотд получения состояния объекта
int cl_base::get_state(){
   return state; // возвращает состояние объекта
}

// метод включения всех подчиненных объектов
void cl_base::enable_all(){
   this->state = 1; // установка состояния текущего объекта в готовность
   // рекурсивное включения всех подчиненных объектов
   for(int i = 0; i <p_sub_objects.size(); i++){
      p_sub_objects[i]->enable_all();
   }
}

/*void cl_base::class_num_set(int num){
   class_num = num;
}*/
```

5.6 Файл cl_base.h

Листинг 6 – cl_base.h

```
#ifndef __CL_BASE__H
```

```
#define __CL_BASE__H

// макросы, преобразующие указатели на функции-члены класса в типы "TYPE_SIGNAL" и "TYPE_HANDLER"
#define SIGNAL_D(signal_f)(TYPE_SIGNAL)(&signal_f) // парматризованное макроопределение для получения указателя на метод сигнала объекта
#define HANDLER_D(handler_f)(TYPE_HANDLER)(&handler_f) // парматризованное макроопределение для получения указателя на метод обработчика объекта

#include <string>
#include <vector>
#include <queue>
using namespace std;
class cl_base;

// объявление типов "TYPE_SIGNAL" и "TYPE_HANDLER" как указателей на функции-члены класса cl_base, которые принимат строку в качестве параметра
typedef void (cl_base::*TYPE_SIGNAL)(std::string&); // определение нового типа данных - указатель на метод сигнала объекта
typedef void(cl_base::*TYPE_HANDLER)(std::string); // определение нового типа данных - указатель на метод обработчика объекта

class cl_base{
   private:
   string name; // имя объекта
   cl_base* p_head_object = nullptr; // указатель на родительский об
   vector<cl_base*> p_sub_objects; // вектор указателей на дочерние объекты
   int state = 0; // состояние объекта (готов/не готов)
   int class_num = 1; // номер класса
   struct o_sh{
      TYPE_SIGNAL p_signal; // сигнал
      cl_base* p_cl_base; // указатель на объек
      TYPE_HANDLER p_handler; // обработчик сигнала
   };
   vector<o_sh*>connects; // вектор связей между сигналами и обработчиками
   public:
   // конструктор класса
   cl_base(cl_base* p_head_object=nullptr, std::string name = "Base object");
   // деструктор класса
   ~cl_base();
   // метод для изменения имени объекта
   bool change_name(std::string new_name);
   // метод для получения имени объекта
   std::string get_name();
   // метод для получения указателя на родительский объект
   cl_base* get_head_object();
   //void show_object_tree();
   // метод для получения указателя на дочерний объект по имени
   cl_base* get_sub_object(std::string name);
   //cl_base* search_from_current(std::string name);
   // метод для поиска объекта по его имени
   cl_base* search_by_name(std::string name);
   //void show_from_current_tree(int number);
   // метод для отображения готовности дерева
```

```
   void show_ready_tree(int number);
   //void set_state(int number);
   //bool set_head_object(cl_base* p_head_object);
   //void delete_sub_object(std::string name);
   //cl_base* get_object_by_path(std::string path);
   // меотд для установки связи
   void  set_connect(TYPE_SIGNAL  p_signal,  cl_base*  p_object,  TYPE_HANDLER
p_ob_handler);
   // метод для удаления связи
   void delete_connect(TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER
p_ob_handler);
   // метод для отправки сигнала объекту
   void emit_signal(TYPE_SIGNAL p_signal, std::string& s_command);
   //std::string get_path();
   // метод для получения состояния объекта
   int get_state();
   // меотд для активации всех дочерних объектов
   void enable_all();
   //void class_num_set(int num);
};

#endif
```

5.7 Файл ControlPanel.cpp

Листинг 7 – ControlPanel.cpp

```
#include "ControlPanel.h"
#include <iostream>
using namespace std;

ControlPanel::ControlPanel(cl_base*  parent,  string  name):  cl_base(parent,
name){}

void ControlPanel::add_bill_signal(string& message) {}
void ControlPanel::deposit_money_signal(string& message){}
void ControlPanel::return_bills_signal(string& message){}
void ControlPanel::text_print_signal(string&){}
void ControlPanel::set_available_sum_obr(string sum){
   availableSum = stoi(sum);
}
void ControlPanel::set_status_obr(string message){
   switch(message[0]){
      case 'L': // если передано LOGGED
         status = LOGGED;
         break;
      case 'R': // если передано READY
         status = WAITING;
         break;
      case 'P': // если передано PIN
```

```
            status = PIN;
            break;
      }
   }

   void ControlPanel::withdraw_signal(string&){}
   void ControlPanel::command_obr(string command){
      // проверяем, авторизован ли пользователь
      if(status == LOGGED){
         // Обработка команд, начинающихся с D(Deposit......)
         if(command[0] == 'D'){
            // Проверяем, завершена ли загрузка денег
            if(command[14] == 't'){
               hasMoneyInserted = false;
               string msg = "";
               emit_signal(SIGNAL_D(ControlPanel::deposit_money_signal),
   command); // отправляем сигнал для внесения денег
               emit_signal(SIGNAL_D(ControlPanel::set_status_obr),    msg);    //
   сбрасываем статус
            }
            else{
               // если загрузка денег продолжается
               hasMoneyInserted = true;
               string msg1 = command.substr(14);
               emit_signal(SIGNAL_D(ControlPanel::add_bill_signal),   msg1);   //
   отправляем сигнал для добавления купюры
            }
         }
         else if(hasMoneyInserted){
            // если деньги были внесены
            string msg1 = command.substr(14);
            emit_signal(SIGNAL_D(ControlPanel::return_bills_signal), msg1);
         }
         if(command[0] == 'W'){
            // обработка команд начинающихся с W
            int sum = atoi((command.substr(15)).c_str());
            // проверяем, кратна ли сумма 100
            if(sum % 100 != 0){
               string msg = "The amount is not a multiple of 100";
               emit_signal(SIGNAL_D(ControlPanel::text_print_signal), msg);
            }
            else if(availableSum < sum){
               // проверяем достаточно ли денег на счету
               string msg = "There is not enough money on the card";
               emit_signal(SIGNAL_D(ControlPanel::text_print_signal), msg);
            }
            else{
               // отправляем сигнал для снятия денег
               string msg1 = to_string(sum);
               emit_signal(SIGNAL_D(ControlPanel::withdraw_signal), msg1);
            }
         }
      }
   }
```

5.8 Файл ControlPanel.h

Листинг 8 – ControlPanel.h

```
#ifndef __CONTROLPANEL__H
#define __CONTROLPANEL__H
#include "cl_base.h"
using namespace std;

// объявление класса ControlPanel, который наследует от cl_base
class ControlPanel: public cl_base{
   private:
   // перечисление возможных состояний панели управления
   enum STATUS{WAITING, PIN, LOGGED};
   bool hasMoneyInserted = false; // флаг, указывающий были ли внесены деньги
   int availableSum; // доступная сумма на счету
   STATUS status = WAITING; // начальное состояние
   public:
   // Конструктор класса ControlPanel, принимающий указатель на родительский
объект и имя
   ControlPanel(cl_base*, string);
   // обработчик комманд , принимающий строку команды
   void command_obr(string);
   // обработчик установки статуса
   void set_status_obr(string);
   // сигнал для добавления купюры
   void add_bill_signal(string&);
   // сигнал для внесения денег
   void deposit_money_signal(string&);
   //
   void return_bills_signal(string&);
   // сигнал для вывода текста
   void text_print_signal(string&);
   // обработчик устапновки доступной суммы
   void set_available_sum_obr(string);
   // сигнал для снятия денег
   void withdraw_signal(string&);
};


#endif
```

5.9 Файл Enter.cpp

Листинг 9 – Enter.cpp

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

```
using namespace std;

// конструктор класса Enter
Enter::Enter(cl_base* parent, string name): cl_base(parent, name){};

// сигнал вывода суммы
void Enter::sum_print_signal(string&){}
// обработчик для добавления купюр
void Enter::add_bill_obr(string bill){
   int value = stoi(bill);
   if(value == 100 || value == 500 || value == 1000 || value == 2000 || value
== 5000){
      string msg = "The amount: " + to_string(sum+value);
      emit_signal(SIGNAL_D(Enter::sum_print_signal), msg);
      sum += value;
   }
}

// обработчик для внесения денег
void Enter::deposit_money_obr(string){
   string msg = to_string(sum); // преобразование суммы в строку
   emit_signal(SIGNAL_D(Enter::add_money_signal), msg); // отправка сигнала
для добавления денег
   sum = 0;
}
// сигнал для добавления денег
void Enter::add_money_signal(string&){}
// обработлчик для возврата денег
void Enter::return_money_obr(string){
   sum= 0;
}
```

5.10 Файл Enter.h

Листинг 10 – Enter.h

```
#ifndef __ENTER__H
#define __ENTER__H
#include "cl_base.h"
using namespace std;

// Определение класса Enter, который наследуется от cl_base
class Enter: public cl_base{
   private:
   int sum = 0; // переменная для хранения суммы
   public:
   // конструктор
   Enter(cl_base* parent, string name);
   // обработчик для добавления купюр
   void add_bill_obr(string);
```

```
   // сигнал для вывода суммы
   void sum_print_signal(string&);
   // обработчик для внесения денег
   void deposit_money_obr(string);
   // сигнал для добавлени яденег
   void add_money_signal(string&);
   // обработчик для возврата денег
   void return_money_obr(string);
};


#endif
```

5.11 Файл Issue.cpp

Листинг 11 – Issue.cpp

```
#include "Issue.h"
#include <iostream>
using namespace std;

// конструктор класса
Issue::Issue(cl_base* parent, string name) : cl_base(parent, name){}

// обработчик команд
void Issue::command_obr(string command){
   // Если команда - завершени езагрузки
   if(command == "End card loading"){
      status = READING; // установка статуса - чтение
   }
   else if(status == READING){ // если статус - чтение
      vector<int> counts(5); // создание вектора для купюр разного номинала
      string str = command;
      for(int i = 0; i <5; i++){ // цикл по каждому номиналу
         counts[i] = stoi(str.substr(0, str.find(' '))); // запись количества
купюр в вектор
         str.erase(0, str.find(' ')+1); // удалеине обработанной части строки
      }
      bills = counts; // присваивание значения вектора купюр
      status = WORKING; // изменение статуса на WORKING
   }
}

// обработчик для вывода денег
void Issue::withdraw_obr(string _sum){
   int sum = stoi(_sum); // преобразование суммы к типу int
   vector<int> withdraw(5, 0); // создание вектора для купюр, которые
необходимо выдать
   vector <int> value = {5000, 2000, 1000, 500, 100}; // создание вектора для
номиналов купюр
```

```
   for(int i = 0; i <5; i++){ // цикл по каждому номиналу
      while(sum - value[i] >= 0 && withdraw[i] < bills[i]){ // пока сумма
больше или равна номиналу и есть доступные купюры этого номинала
         bills[i]--; // уменьшение количества доступных купюр
         withdraw[i]++; // увеличение количества купюр для выдачи
         sum -= value[i]; // уменьшение суммы
      }
   }
   if(sum != 0){ // если остаток суммы не равен 0
      string msg = "There is not enough money in the ATM"; //  в банкомате не
достаточно денег
      emit_signal(SIGNAL_D(Issue::print_signal), msg); // отправква сигнала
для печати сообщения
   }
   else{
      for(int i =0; i < 5; i++){ // цикл по каждому номиналу
         bills[i] -= withdraw[i]; // уменьшение количества купюр данного
номинала на количество выдпнных купюр
      }
      // создание сообщения о выдаче денег
      string msg = "Take the money: 5000 * " + to_string(withdraw[0]) + "
rub., 2000 * " + to_string(withdraw[1]) + " rub., 1000 * " +
to_string(withdraw[2]) + " rub., 500 * " + to_string(withdraw[3]) + " rub.,
100 * " + to_string(withdraw[4]) + " rub.";
      // отправка сигнала об уменьшении баланса
      emit_signal(SIGNAL_D(Issue::balance_reduction_signal), _sum);
      // отправка сигнала для печати сообщения
      emit_signal(SIGNAL_D(Issue::print_signal), msg);
   }
}

void Issue::print_signal(string&) {} // сигнал печати сообщения
void Issue::balance_reduction_signal(string&){} // сигнал уменьшения баланса
```

5.12 Файл Issue.h

Листинг 12 – Issue.h

```
#ifndef __ISSUE__H
#define __ISSUE__H
#include "cl_base.h"
using namespace std;

// определение класса Issue, наследующегося от cl_base
class Issue: public cl_base{
   private:
   enum STATUS{WAITING, READING, WORKING}; // перечисление возможных статусов
   vector<int> bills; // вектор для хранения купюр
   STATUS status = WAITING; // изначальный статус - ожидание
   public:
```

```
   Issue(cl_base* parent, string name); // конструктор класса
   void command_obr(string); // метод-обработчик команд
   void withdraw_obr(string); // обработчик для вывода денег
   void print_signal(string&); // сигнал печати сообщения
   void balance_reduction_signal(string&); // сигнал уменьшения баланса
};


#endif
```

5.13 Файл main.cpp

Листинг 13 – main.cpp

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

int main()
{
   cl_application ob_cl_application(nullptr); // создание объекта приложение
   ob_cl_application.build_tree_objects();   //   конструирование   системы,
построения дерева
   return ob_cl_application.exec_app(); // запуск системы
}
```

5.14 Файл Printer.cpp

Листинг 14 – Printer.cpp

```
#include "Printer.h"
#include <iostream>
using namespace std;

// конструктор
Printer::Printer(cl_base* parent, string name): cl_base(parent, name){}

// обработчик для печати
void Printer::print_obr(string message){
   cout << (first ? "" : "\n") << message; // вывод сообщения на экран с
добавлением перехода на новую строку при необходимости
   first = false; // установка флага первой печати в false после первого
вывода
}
```

5.15 Файл Printer.h

Листинг 15 – Printer.h

```
#ifndef __PRINTER__H
#define __PRINTER__H
#include "cl_base.h"
using namespace std;

class Printer: public cl_base{ // опередение класса Printer, наследуемого
от cl_base
   private:
   bool first = true; // флаг первой печати
   public:
   Printer(cl_base*, string); // параметризованный конструктор Printer
   void print_obr(string); // метод-обработчик для печати
};


#endif
```

5.16 Файл Reader.cpp

Листинг 16 – Reader.cpp

```
#include "Reader.h"
#include <iostream>
using namespace std;

// конструктор
Reader::Reader(cl_base* parent, string name) : cl_base(parent, name){}

// обработчик для чтения
void Reader::read_obr(string message){
   emit_signal(SIGNAL_D(Reader::read_signal),   message);//   вызов   метода-
сигнала для передачи команды на чтение
}
// сигнал для чтения команды
void Reader::read_signal(string &command){
   getline(cin, command); // считывание команды из стандартного ввода
}
```

5.17 Файл Reader.h

Листинг 17 – Reader.h

```
#ifndef __READER__H
#define __READER__H
#include "cl_base.h"
using namespace std;

// определение класса Reader, наследующегося от cl_base
class Reader: public cl_base{
   public:
   Reader(cl_base*, string);// конструктор
   void read_obr(string);// метод-обработчик для чтения
   void read_signal(string&); // сигнал для чтения команды
};

#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 54.

Таблица 54 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
1001 2022 0021 0001 5001 15000 1001 2022 0021 0002 5002 25000 1001 2022 0021 0003 5003 35000 1001 2022 0021 0004 5004 45000 1001 2022 0021 0005 5005 55000 End card loading 5000 2000 1000 500 100 End money loading Deposit money 1000 Identification 1001 2022 0021 0004 PIN-code 5004 Deposit money 1000 Deposit money 1000 Deposit money 50 Deposit money to the card Withdraw money 10000 Deposit money 1000 Deposit money to the card End the session Identification 1001 2022 0021 0003 Deposit money 1000 Deposit money to the card End the session Deposit money 1000 Identification 1001 2022 0021 0001 PIN-code 5001 Withdraw money 8500 Deposit money 1500 Deposit money 500 Deposit money to the card End the session Turn off the ATM	Ready to work Ready to work Enter the PIN code Select the command The amount: 1000 The amount: 2000 Card balance 47000 Take the money: 5000 * 2 rub., 2000 * 0 rub., 1000 * 0 rub., 500 * 0 rub., 100 * 0 rub. The amount: 1000 Card balance 38000 Ready to work Enter the PIN code Ready to work Ready to work Ready to work Ready to work Enter the PIN code Select the command Take the money: 5000 * 1 rub., 2000 * 1 rub., 1000 * 1 rub., 500 * 1 rub., 100 * 0 rub. The amount: 500 Card balance 7000 Ready to work Turn off the ATM	Ready to work Ready to work Enter the PIN code Select the command The amount: 1000 The amount: 2000 Card balance 47000 Take the money: 5000 * 2 rub., 2000 * 0 rub., 1000 * 0 rub., 500 * 0 rub., 100 * 0 rub. The amount: 1000 Card balance 38000 Ready to work Enter the PIN code Ready to work Ready to work Ready to work Ready to work Enter the PIN code Select the command Take the money: 5000 * 1 rub., 2000 * 1 rub., 1000 * 1 rub., 500 * 1 rub., 100 * 0 rub. The amount: 500 Card balance 7000 Ready to work Turn off the ATM

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
1001 2022 0021 0001 5001 15000 1001 2022 0021 0002 5002 25000 1001 2022 0021 0003 5003 35000 1001 2022 0021 0004 5004 45000 1001 2022 0021 0005 5005 55000 End card loading 5000 2000 1000 500 100 End money loading SHOWTREE	Ready to work System is ready ControlPanel is ready Reader is ready CardIdentifier is ready Enter is ready Issue is ready Printer is ready	Ready to work System is ready ControlPanel is ready Reader is ready CardIdentifier is ready Enter is ready Issue is ready Printer is ready
1001 2022 0021 0001 5001 15000 1001 2022 0021 0002 5002 25000 1001 2022 0021 0003 5003 35000 1001 2022 0021 0004 5004 45000 1001 2022 0021 0005 5005 55000 End card loading 5000 2000 1000 500 100 End money loading Deposit money 1000 Identification 1001 2022 0021 0004 PIN-code 5004 Deposit money 1000 Deposit money 1000 Deposit money 50 Deposit money to the card Withdraw money 10000 Deposit money 1000 Deposit money to the card End the session End the session Turn off the ATM	Ready to work Ready to work Enter the PIN code Select the command The amount: 1000 The amount: 2000 Card balance 47000 Take the money: 5000 * 2 rub., 2000 * 0 rub., 1000 * 0 rub., 500 * 0 rub., 100 * 0 rub. The amount: 1000 Card balance 38000 Ready to work Ready to work Turn off the ATM	Ready to work Ready to work Enter the PIN code Select the command The amount: 1000 The amount: 2000 Card balance 47000 Take the money: 5000 * 2 rub., 2000 * 0 rub., 1000 * 0 rub., 500 * 0 rub., 100 * 0 rub. The amount: 1000 Card balance 38000 Ready to work Ready to work Turn off the ATM

ЗАКЛЮЧЕНИЕ

В ходе работы были изучены основные принципы разработки систем с использованием ООП, получены практические навыки работы с C++ и симуляции работы систем.

Был определен метод решения, а также были описаны алгоритмы работы программы. Была построена блок схема в соответствии с алгоритмом, было осуществлено тестирование программы на нескольких наборах входных данных. В рамках работы над поставленной задачей было реализовано дерево объектов, методы, используемые для упраления программой. Также был реализован механизм сигналов и обработчиков.

Была разработана программа, моделирующая работу банкомата. Были получены навыки практической работы с учебным проектом.

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).