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].

Язык программирования C++ имеет большой набор возможностей. Он используется для создания программного обеспечения разного рода: от игр до операционных систем, также широко применяется в обработке данных и научных расчётах. Одной из возможностей C++ является объектно-ориентированное программирование. Именно с помощью ее реализации будет выполнена данная курсовая работа. ООП (Объектно-ориентированное программирование) - методология или стиль программирования на основе описания типов/моделей предметной области и их взаимодействия, представленных порождением из прототипов или как экземпляры классов, которые образуют иерархию наследования. Цели курсовой работы:

1. Познакомиться с древовидным представлением задач в виде графа с иерархией объектов;
2. Научиться работать с готовностью объектов;
3. Научиться добавлять различные методы для работы с объектами на дереве;
4. Научиться работать с сигналами и обработчиками сигналов в отношении объектов;
5. Разработать алгоритм, реализующий моделирование работы 3D-принтера.

В рамках освоения объектно-ориентированной парадигмы очень важно ознакомиться и приобрести навык работы с такими механизмами и инструментами как реализация древовидной иерархии объектов, описание базового класса, реализация сигналов и их обработчиков.

1 ПОСТАНОВКА ЗАДАЧИ

Постановку задачи сформулировал и выполнил реализацию студент первого курса Егор Гулякин.

Описание системы:

Дана система, состоящая из следующих элементов:

- 3D принтер;
- панель управления 3D принтером;
- множество катушек с филаментом (расходного материала, используемого для печати на 3D-принтере);
- компьютер;
- множество изделий для печати;
- экрана отображения информации о функционировании системы.

3D принтер имеет следующие параметры:

- скорость печати, измеряемая в метрах в такт (метр/такт);
- скорость нагрева/охлаждения (℃/такт);
- температура нагрева сопла экструдера (печатающей головки 3D принтера) в градусах Цельсия (℃);
- катушки филамента.

Катушка филамента имеет следующие параметры:

- температура печати пластика в градусах Цельсия (℃);
- длина нитки филамента (объём катушки) в метрах;
- идентификатор (одно слово).

Изделие имеет следующие параметры:

- наименование;
- идентификатор катушки (одно слово);
- объём изделия (длина нити филамента, необходимая для печати изделия)

в метрах.

3D принтер обслуживает ПК.

У 3D принтера множество состояний:

- выключен = 0;
- ожидание команды = 1;
- печать изделия = 2;
- нагрев/охлаждение сопла = 3;
- ожидание катушки филамента = 4;
- настройка 3D принтера = 5.

Система функционирует по тактам. Такты нумеруются с 1.

Модель системы.

Правила функционирования системы:

1. В сети один персональный компьютер.
2. Количество катушек филамента задаётся параметром n.
3. Объём и температура нагрева для катушки филамента задаётся параметрами k и t соответственно.
4. Скорость печати для принтера – количество метров филамента в такт задаётся параметром v.
5. Скорость охлаждения и нагрева принтера, а также его начальная температура задаются параметрами r и j соответственно.
6. ПК организует очередь изделий для печати. Изделие характеризуется именем, названием филамента и объёмом (количеством метров филамента, необходимого для печати изделия).
7. После поступления изделия в очередь для распечатки на ПК, посылается сигнал 3D принтеру. Постановка изделия на ПК происходит вначале такта. (То есть, принтер полностью отрабатывает последующие действия.) Принтер организует очередь запросов. По мере готовности (температуры

принтера) делает запрос к ПК, копирует модель изделия (получает от ПК) и организует печать. После передачи на 3D принтер изделие из очереди ПК удаляется.

8. Копирование нового изделия происходит с началом нового такта. То есть, если в этот такт до печаталось изделие, то копирование нового произойдёт в следующий такт.

9. Копирование изделия на печать занимает весь такт. (Действия после копирования не отрабатываются).

10. Перед началом печати 3D принтер нагревается/охлаждается до температуры, необходимой, чтобы данный филамент начал плавиться. Нагрев/охлаждение происходит со скоростью r. В данный момент 3D принтер переводится в состояния нагрева/охлаждения. Скорость r – максимальная для 1 такта, однако не обязательно достигать скорости r для нагрева до определённой температуры (пример currentTemp = 110, r = 100, то для нагрева до 120 всё равно необходим 1 такт).

11. Если в катушке закончился филамент, то принтер переходит в состоянии ожидания загрузки катушки. Катушка загружается в течение трёх тактов в независимости от объёма (на конец 3 такта считается восполненной). Первый такт восполнения катушки проходит в момент её окончания после печати. Катушка не восполняется, если в этом нет необходимости. (То есть, в такт, когда 3D принтер напечатал некоторый объём изделия и закончилась катушка, происходит её первый такт восполнения, за ним следует ещё два, а потом она начинает работать).

12. Если очередь запросов 3D принтера пуста, то он переходит в состоянии ожидания.

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

Необходимо моделировать работу данной системы.

Команды системы:

1. Команда постановки изделия в очередь для печати на ПК.
2. Команда отображения состояния ПК.
3. Команда отображения состояния системы.
4. Пустая команда (строка ничего не содержит). Элементы системы выполняют действия согласно такту.
5. Команда завершения работы системы.

Построить программу-систему, которая использует объекты:

1. Объект «система».
2. Панель управления 3D принтером - объект для чтения исходных данных и команд. Считывает данные для первоначальной подготовки и настройки системы. Считывает команды. Объект моделирует нажатия на кнопки 3D принтера или действия пользователя за ПК. После чтения очередной порции данных для настройки или данных команды, объект выдает сигнал с текстом полученных данных. Все данные настройки и данные команд по структуре корректны. Каждая строка команд соответствует одному такту (отрабатывается в начале такта). Если строка пустая, то система отрабатывает один такт (все элементы системы отрабатывают положенные действия или находятся в состоянии ожидания).
3. Объект ПК. Содержит очередь изделий для печати. Данному объекту по иерархии подчинены объекты изделий, ожидающих печать. Сигналами общается с 3D принтером.
4. Объект изделие. Содержит наименование, объём (длина нити филамента, необходимая для печати изделия) в метрах, идентификатор катушки.
5. Объект катушка филамента. Содержит идентификатор (одно слово), объём (длина нити филамента в катушке) в метрах, температура печати.

6. Объект 3D принтера, моделирует работу контролера управления 3D принтером. Анализирует состояние, управляет отработкой действий, предусмотренных в рамках одного такта. Объект выдает соответствующие сигналы для других основных элементов системы.

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

Архитектура иерархи объектов:

Объект системы моделирования работы 3д принтера.

```
Объект ввода (панель управления).
Объект вывода (информационное окно).
Объект 3D принтера
    Объект катушка филамента 1.
    ….
    Объект катушка филамента n.
    Объект изделие, которое печатается.
Объект ПК
    Объект изделие 1 в очереди на печать.
    Объект изделие 2 в очереди на печать.
    ….
    Объект изделие m в очереди на печать.
```

Сконструировать программу-систему, реализующую следующий алгоритм:

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

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

1.2. Цикл для обработки вводимых данных.

1.2.1. Выдача сигнала объекту чтения для ввода очередной строки данных (количество катушек филамента, начальная температура сопла, скорость нагрева/охлаждения сопла, скорость печати).

1.2.2. Отработка операции чтения очередной строки

входных данных.

1.2.3. Исходя из значения количества катушек филамента, создание соответствующего количества объектов катушек филамента с параметрами (идентификатор, температура печати, объём). Установка связей сигналов и обработчиков с новым объектом.

1.3. Установка связей сигналов и обработчиков между объектами.

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

2.1. Цикл для обработки вводимых команд.

2.1.1. Определение номера очередного такта.

2.1.2. Выдача сигнала объекту ввода для чтения очередной команды.

2.1.3. Отработка команды.

2.1.4. Отработка действий согласно такту.

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

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

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

При решении задачи необходимо руководствоваться методическим пособием и приложением к методическому пособию.

1.1 Описание входных данных

Первая строка, количество катушек филамента n, скорость 3D принтера v, скорость нагрева/охлаждения r, начальная температура сопла 3D принтера j:

```
«целое число»_«целое число»_«целое число»_«целое число»
```

Далее n строк вводятся параметры катушки филамента (идентификатор, температура печати, длина нитки филамента):

```
«одно слово»_«целое число»_«целое число»
```

Далее строка:

```
End of settings
```

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

Команда постановки изделия в очередь для печати на ПК.

```
PC_«объём изделия»_«идентификатор катушки»_«название изделия»
```

По этой команде, создается объект изделие и на дереве иерархии объектов подчиняется объекту ПК. Изделие ставится в очередь для печати на ПК и сигнал запроса на печать посылается 3D принтеру. Если 3D принтер не печатает изделие в текущий момент, то объект переназначается на 3D принтер.

Гарантируется, что катушка по параметру идентификатора существует в системе.

Команда отображения состояния ПК:

```
PC condition
```

Команда отображения состояния катушки филамента:

```
Filament coil condition «идентификатор»
```

Команда отображения состояния системы:

```
System status
```

Пустая команда (строка ничего не содержит).

Элементы системы выполняют действия согласно такту.

Команда завершения работы системы.

```
Turn off the system
```

Пример ввода:

```
3 5 100 0
abs 200 50
pla 160 100
petg 190 5
End of settings
System status
PC 5 pla Product 1
PC 10 abs Product 2
PC condition
Filament coil condition pla
PC 10 petg Product 3
System status
PC condition




Filament coil condition petg
PC condition
System status
Turn off the system
```

1.2 Описание выходных данных

После завершения ввода исходных данных выводиться текст:

```
Ready to work
```

После команды ввода параметров катушки филамента, если существует объект с таким названием во всём дереве.

```
Failed to create filament coil
```

После команды отображения состояния ПК вывести.

Если у ПК есть очередь изделий на печать, не считая текущего:

```
PC condition turned on «наименование изделия 1»; «наименование изделия  2»; .
```

. .

Наименования документов упорядочены согласно очереди на печать на ПК.

Если у пк нет изделий на печать:

```
PC condition turned off
```

После команды создания изделия.

Если в дереве есть объект с именем изделия:

```
Failed to create product
```

После команды отображения состояния катушки филамента:

```
Filament coil condition: «идентификатор» «оставшееся количество филамента»
```

После команды отображения состояния системы:

```
3D printer tact: «номер такта»; temp: «температура сопла»; status: «статус принтера»; print product: «количество оставшегося объёма печатаемого изделия»; queue products: «количество изделий в очереди на печать»; PC: («название изделия 1»:«объём изделия») («название изделия 2»:«объём изделия») («название изделия 3»:«объём изделия») . . .
```

После команды завершения работы системы вывести:

```
Turn off the system
```

Пример вывода:

```
Ready to work
3D printer tact: 1; temp: 0; status: 1; print product: 0; queue products: 0; PC:
PC condition turned on Product 1; Product 2;
Filament coil condition: pla 100
3D printer tact: 7; temp: 200; status: 3; print product: 0; queue products: 2; PC: Product 2:10 Product 3:10
PC condition turned on Product 3;
Filament coil condition: petg 5
PC condition turned off
3D printer tact: 17; temp: 190; status: 1; print product: 0; queue products: 0; PC:
Turn off the system
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект printer класса printer предназначен для Модель 3D-принтера;
- объект Command_panel класса Command_panel предназначен для Модель нажатия кнопок принтера и ввода команд с компьютера;
- объект Display класса Display предназначен для Модель вывода информации на экран;
- объект Details класса Details предназначен для Модель деталей для изготовки;
- объект Coils класса coils предназначен для Модель катушек с филаментом;
- объект PC класса PC предназначен для Модель компьютера;
- объект ob_cl_application класса cl_application предназначен для Модель приложения;
- cin - стандартный объект потока ввода;
- cout - стандартный объект потока ввода;
- stack - Динамический список данных с ограниченным выходом по правилу стэка;
- iterator - интерфейс, предоставляющий доступ к элементам массива;
- vector - Контейнер типа Vector;
- if - Условный оператор;
- while - Оператор цикла с условием;
- break - Оператор выхода из цикла;
- string - Строковый тип данных;
- getline - Оператор ввода строк;
- typedef - Определение типа данных;

- struct - Структура данных;
- for - Оператор цикла ;
- stoi - Преобразование строки в целочисленный тип.

Класс cl_base:

- свойства/поля:
    - поле Имя объекта:
        - наименование — s_object_name;
        - тип — string;
        - модификатор доступа — private;
    - поле Статус готовности объекта:
        - наименование — status;
        - тип — int;
        - модификатор доступа — private;
    - поле Указатель на головной объект:
        - наименование — p_head_object;
        - тип — cl_base*;
        - модификатор доступа — private;
    - поле Массив указателей на подчиненные объекты:
        - наименование — subordinate_objects;
        - тип — vector <cl_base*>;
        - модификатор доступа — private;
    - поле Массив указателей на объекты с проложенными связями:
        - наименование — connects;
        - тип — vector <o_sh*>;
        - модификатор доступа — private;
- функционал:
    - метод cl_base — Конструктор;

- метод Change_name — Изменение имени объекта;
- метод Get_name — Получение имени объекта;
- метод Get_baseptr — Получение указателя на головной объект;
- метод Out — Вывод дерева на экран с состояниями готовности;
- метод Out_fc — Вывод дерева на экран;
- метод can_off — Изменение состояния готовности объекта;
- метод get_root — Получение указателя на корневой объект;
- метод Move_head — Изменение головного объекта;
- метод Del_obj — Удаление объекта;
- метод Absolute — Получение абсолютной координаты объекта;
- метод Find_current — Поиск объекта на дереве, начиная от текущего;
- метод Find_global — Поиск объекта на дереве, начиная от корня;
- метод Get_ptr — Поиск объекта среди подчиненных текущему;
- метод find_obj_bc — Поиск объекта на дереве по координате;
- метод Set_all_ready — Приведение всех объектов в положительное состояние готовности;
- метод set_connection — Создание связи между объектами;
- метод delete_connection — Удаление связи между объектами;
- метод emit_signal — Отправка сигнала от объекта к объекту;
- метод delete_links — Удаление всех связей объекта;
- метод ~cl_base — Деструктор.

Класс cl_application:

- свойства/поля:
  - поле Такт:
    - наименование — tact;
    - тип — int;

        - модификатор доступа — private;
- функционал:
    - метод cl_application — Конструктор;
    - метод Tree — Создание дерева объектов;
    - метод Start — Начало работы приложения;
    - метод signal_app_to_panel — Сигнал для командной панели;
    - метод handler_app_from_printer — Получение сигнала от принтера.

Класс printer:

- свойства/поля:
    - поле Скорость печати:
        - наименование — speed_of_print;
        - тип — int;
        - модификатор доступа — public;
    - поле Скорость нагрева/охлаждения:
        - наименование — speed_of_temp;
        - тип — int;
        - модификатор доступа — public;
    - поле Текущая температура:
        - наименование — Celsium;
        - тип — int;
        - модификатор доступа — public;
    - поле Состояние:
        - наименование — state;
        - тип — int;
        - модификатор доступа — public;
    - поле Объем напечатанного изделия:
        - наименование — len;

        - тип — int;
        - модификатор доступа — public;
    - поле Массив с катушками:
        - наименование — All_coils;
        - тип — vector <coils*>;
        - модификатор доступа — public;
    - поле Имя детали на печати:
        - наименование — name_of_details;
        - тип — string;
        - модификатор доступа — public;
- функционал:
    - метод Set_state — Установить состояние;
    - метод printer — Конструктор;
    - метод signal_printer_to_PC — Сигнал компьютеру;
    - метод signal_printer_to_app — Сигнал приложению;
    - метод signal_printer_to_panel — Сигнал командной панели;
    - метод signal_printer_to_display — Сигнал дисплею;
    - метод handler_printer_from_panel — Сигнал от командной панели;
    - метод handler_printer_from_PC — Сигнал от компьютера.

Класс PC:

- свойства/поля:
    - поле Массив с указателями на детали в очереди на печать:
        - наименование — Q_Details;
        - тип — vector <Details*>;
        - модификатор доступа — public;
- функционал:
    - метод PC — Конструктор;

    - метод signal_PC_to_printer — Сигнал принтеру;
    - метод handler_PC_from_panel — Сигнал от командной панели;
    - метод handler_PC_from_printer — Сигнал от принтера.

Класс Display:

- функционал:
    - метод Display — Конструктор;
    - метод handler_display_from_panel — Сигнал от командной панели.

Класс Details:

- свойства/поля:
    - поле Имя детали:
        - наименование — name;
        - тип — string;
        - модификатор доступа — public;
    - поле Идентификатор катушки:
        - наименование — id;
        - тип — string;
        - модификатор доступа — public;
    - поле Объем изделия:
        - наименование — Volume;
        - тип — int;
        - модификатор доступа — public;
- функционал:
    - метод Details — Конструктор.

Класс coils:

- свойства/поля:
    - поле Температура плавления:
        - наименование — p_temp;

        - тип — int;
        - модификатор доступа — public;
    - поле Длина нити филамента:
        - наименование — lenght;
        - тип — int;
        - модификатор доступа — public;
    - поле Оставшаяся длина нити филамента:
        - наименование — still_l;
        - тип — int;
        - модификатор доступа — public;
    - поле Идентификатор:
        - наименование — id;
        - тип — string;
        - модификатор доступа — public;
- функционал:
    - метод coils — Конструктор.

Класс Command_panel:

- функционал:
    - метод Command_panel — Конструктор;
    - метод signal_panel_to_PC — Сигнал компьютеру;
    - метод signal_panel_to_printer — Сигнал принтеру;
    - метод signal_panel_to_display — Сигнал дисплею;
    - метод handler_panel_from_app — Сигнал от приложения;
    - метод handler_panel_from_printer — Сигнал от принтера.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс дерева	
		cl_application	public		2
		printer	public		3
		PC	public		4
		Display	public		5
		Details	public		6
		coils	public		7
		Command_panel	public		8
2	cl_application			Модель приложения	
3	printer			Модель принтера	
4	PC			Модель компьютера	
5	Display			Модель дисплея	
6	Details			Модель деталей	
7	coils			Модель катушек с филаментом	
8	Command_panel			Модель командной панели	

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 cl_base

Функционал: Конструктор.

Параметры: cl_base* - p_head_object - указатель на головной объект; string - s_object_name - имя объекта .

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Присвоение полям текущего объекта значений соответствующих параметров	2
2	Существует головной объект	Добавить головному объекту текущий в качестве подчиненного	∅
			∅

3.2 Алгоритм метода Change_name класса cl_base

Функционал: Изменение имени объекта.

Параметры: string name - новое имя.

Возвращаемое значение: bool - индикатор изменения имени.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода Change_name класса cl_base

№	Предикат	Действия	№

			перехода
1	Среди объектов подчиненных головному текущего есть объект с именем name	вернуть false	∅
		Присвоить имени текущего объекта значения name	2
2		вернуть true	∅

3.3 Алгоритм метода Get_name класса cl_base

Функционал: Получение имени объекта.

Параметры: нет.

Возвращаемое значение: string - имя объекта.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода Get_name класса cl_base

№	Предикат	Действия	№ перехода
1		Вернуть имя текущего объекта	∅

3.4 Алгоритм метода Get_baseptr класса cl_base

Функционал: Получение указателя на головной объект.

Параметры: нет.

Возвращаемое значение: cl_base* - указатель на головной объект.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода Get_baseptr класса cl_base

№	Предикат	Действия	№ перехода
1		Вернуть указатель на головной объект текущего	∅

3.5 Алгоритм метода Out класса cl_base

Функционал: Вывод дерева на экран с состояниями готовности.

Параметры: int - probel - количество отступов.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода Out класса cl_base

№	Предикат	Действия	№ перехода
1		вывод на экран имени текущего объекта	2
2	статус готовности объекта равен нулю	вывод на экран " is not ready"	3
		вывод на экран " is ready"	3
3	Существуют подчиненные текущему объекты	Увеличить probel на 4; Рекурсивный вызов функции Out для каждого из подчиненных объектов	3
			∅

3.6 Алгоритм метода Out_fc класса cl_base

Функционал: Вывод дерева на экран.

Параметры: int - probel - количество отступов.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода Out_fc класса cl_base

№	Предикат	Действия	№ перехода
1		вывод на экран имени текущего объекта	2
2	Существуют подчиненные текущему объекты	Увеличить probel на 4; Рекурсивный вызов функции Out_fc для каждого	2

№	Предикат	Действия	№ перехода
		из подчиненных объектов	
			∅

3.7 Алгоритм метода can_off класса cl_base

Функционал: Изменение состояния готовности объекта.

Параметры: int - status1 - новое состояние объекта.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода can_off класса cl_base

№	Предикат	Действия	№ перехода
1	головной объект выключен		∅
			2
2	новый статус равен нулю	Выключить текущий объект и все подчиненные ему с помощью рекурсивного вызова метода can_off	∅
		Включить текущий объект	∅

3.8 Алгоритм метода get_root класса cl_base

Функционал: Получение указателя на корневой объект.

Параметры: нет.

Возвращаемое значение: cl_base* - указатель на корневой объект.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_root класса cl_base

№	Предикат	Действия	№ перехода
1	Существует головной объект	Заменить текущий объект на его головной	1

№	Предикат	Действия	№ перехода
	у текущего		
		вернуть текущий объект	∅

3.9 Алгоритм метода Move_head класса cl_base

Функционал: Изменение головного объекта.

Параметры: cl_base* - h - указатель на новый головной объект.

Возвращаемое значение: bool - индикатор выполнения метода.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода Move_head класса cl_base

№	Предикат	Действия	№ перехода
1	Существует головной объект у текущего	удалить указатель на текущий объект из списка подчиненных головному	2
		вернуть false	∅
2		присвоить указателю на головной объект текущего h	3
3		добавить текущий объект в список подчиненных нового головного	4
4		вывести на экран имя нового головного объекта; вернуть true	∅

3.10 Алгоритм метода Del_obj класса cl_base

Функционал: Удаление объекта.

Параметры: string - name - имя удаляемого объекта.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода Del_obj класса cl_base

№	Предикат	Действия	№ перехода
1	Существует подчиненный текущему объект с именем name	Удалить искомый объект из списка подчиненных; удалить объект	∅
			∅

3.11 Алгоритм метода Absolute класса cl_base

Функционал: Получение абсолютной координаты объекта.

Параметры: нет.

Возвращаемое значение: string - координата объекта.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода Absolute класса cl_base

№	Предикат	Действия	№ перехода
1	Текущий объект - корневой	вернуть "/"	∅
		объявление текстовой переменной координаты - text1	2
2	существует головной объект текущего и он не является корневым	добавить к координате имя головного объекта спереди через "/"	2
		вернуть координату	∅

3.12 Алгоритм метода Find_current класса cl_base

Функционал: Поиск объекта на дереве, начиная от текущего.

Параметры: string - name - имя искомого объекта.

Возвращаемое значение: cl_base* - указатель на объект.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода Find_current класса cl_base

№	Предикат	Действия	№ перехода
1	имя текущего объекта равно name	Вернуть указатель на текущий объект	∅
			2
2	Существуют подчиненные текущему объекты	рекурсивный вызов метода Find_current для каждого из подчиненных объектов	3
			3
3	Существует объект с именем name	вернуть данный объект	∅
		вернуть nullptr	∅

3.13 Алгоритм метода Find_global класса cl_base

Функционал: Поиск объекта на дереве, начиная от корня.

Параметры: string - name - имя искомого объекта.

Возвращаемое значение: cl_base* - указатель на объект.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода Find_global класса cl_base

№	Предикат	Действия	№ перехода
1		Проверка от корня на наличие нескольких объектов с именем name	2
2	Количество объектов с именем name не равно 1	вернуть nullptr	∅
		вызов метода Find_current от корня с параметром name	∅

3.14 Алгоритм метода Get_ptr класса cl_base

Функционал: Поиск объекта среди подчиненных текущему.

Параметры: string - s_object_name - имя искомого объекта.

Возвращаемое значение: cl_base* - указатель на объект.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода Get_ptr класса cl_base

№	Предикат	Действия	№ перехода
1	Среди подчиненных текущему существует объект с именем s_object_name	вернуть указатель на этот объект	∅
		вернуть nullptr	∅

3.15 Алгоритм метода find_obj_bc класса cl_base

Функционал: Поиск объекта на дереве по координате.

Параметры: string coord - координата.

Возвращаемое значение: cl_base* - указатель на объект.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода find_obj_bc класса cl_base

№	Предикат	Действия	№ перехода
1	координата имеет вид "/"	вернуть корневой объект	∅
			2
2	координата имеет вид "."	Вернуть текущий объект	∅
			3
3	координата имеет вид "//name"	Вернуть метод Find_global с параметром name	∅
			4

№	Предикат	Действия	№ перехода
4	координата имеет вид ".name"	Вернуть метод Find_current для текущего объекта с параметром name	∅
			5
5	координата имеет вид "/name1/name2/....."		8
			6
6	координата имеет вид "name1/name2....."		10
			7
7		вернуть nullptr	∅
8	Существует объект с именем name1	рекурсивный вызов для этого объекта find_object_bc с параметром name2....	9
		вернуть nullptr	∅
9	Координата кончилась и объект существует	вернуть объект	∅
		вернуть nullptr	∅
10	Существует объект с именем name1 среди подчиненных текущему	рекурсивный вызов для этого объекта find_object_bc с параметром name2....	9
		вернуть nullptr	∅

3.16 Алгоритм метода Set_all_ready класса cl_base

Функционал: Приведение всех объектов в положительное состояние готовности.

Параметры: нет.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода Set_all_ready класса cl_base

№	Предикат	Действия	№ перехода
1		вызов метода can_off для текущего объекта с параметром 1	2
2		рекурсивный вызов метода set_all_ready для каждого подчиненного текущему объекта	∅

3.17 Алгоритм метода set_connection класса cl_base

Функционал: Создание связи между объектами.

Параметры: TYPE_SIGNAL - signal - функция void с сигналом отправки; cl_base* - target - указатель на получателя сигнала; TYPE_HANDLER - handler - функция void получения сигнала.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода set_connection класса cl_base

№	Предикат	Действия	№ перехода
1	связь с такими параметрами существует		∅
		создание объекта структуры o_sh с полями равными параметрам	2
2		добавление объекта в массив connects - список всех связей текущего объекта	∅

3.18 Алгоритм метода delete_connection класса cl_base

Функционал: Удаление связи между объектами.

Параметры: TYPE_SIGNAL - signal - функция void с сигналом отправки; cl_base* - target - указатель на получателя сигнала; TYPE_HANDLER - handler -

функция void получения сигнала.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода delete_connection класса cl_base

№	Предикат	Действия	№ перехода
1		создание итератора it для vector <o_sh*>	2
2	поля итератора среди массива connects текущего элемента равны параметрам	удалить связь из connects; удалить указатель на итератор	∅
			∅

3.19 Алгоритм метода emit_signal класса cl_base

Функционал: Отправка сигнала от объекта к объекту.

Параметры: TYPE_SIGNAL - signal - функция void с сигналом отправки; string - msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода emit_signal класса cl_base

№	Предикат	Действия	№ перехода
1	состояние готовности текущего объекта равно 0		∅
		вызвать сигнал signal для текущего объекта с параметром msg	2
2	среди связей текущего объекта есть сигнал совпадающий с параметром		3
			∅

№	Предикат	Действия	№ перехода
3	состояние готовности объекта среди получателей сигнала равно 1	вызвать сигнал handler для получателя с параметром msg	∅
			∅

3.20 Алгоритм метода delete_links класса cl_base

Функционал: Удаление всех связей объекта.

Параметры: cl_base* - target - указатель на целевой объект.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода delete_links класса cl_base

№	Предикат	Действия	№ перехода
1	Существует целевой объект среди связей текущего объекта	удалить связь с целевым объектом из connects	2
			∅
2	Существуют подчиненные текущеу объекты	рекурсивный вызов метода delete_links для каждого подчиненного объекта с параметром target	2
			∅

3.21 Алгоритм деструктора класса cl_base

Функционал: Деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 22.

Таблица 22 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Вызов метода delete_links от корня с параметром this	2
2	существуют подчиненные объекты у текущего	удалить подчиненный объект	2
			∅

3.22 Алгоритм конструктора класса cl_application

Функционал: Конструктор.

Параметры: cl_base * p_head_object - указатель на головной объект.

Алгоритм конструктора представлен в таблице 23.

Таблица 23 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		создание объекта	∅

3.23 Алгоритм метода Tree класса cl_application

Функционал: Создание дерева объектов.

Параметры: нет.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода Tree класса cl_application

№	Предикат	Действия	№ перехода
1		Присвоить корневому объекту имя System	2
2		Создание объектов printer, Command_panel, coils, PC, Display, Details соответствующих классов	3

№	Предикат	Действия	№ перехода
3		Установка всех состояний готовности на 1; перевод принтера в состояние 5	4
4		Установка связей между приложением и командной панелью; командной панелью и компьютером, принтером, Дисплеем; компьютером и принтером; принтером и компьютером, приложением	5
5		Ввод начальных данных с клавиатуры (настройка принтера)	6
6		Перевод принтера в состояние 1	∅

3.24 Алгоритм метода Start класса cl_application

Функционал: Начало работы приложения.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности завершения алгоритма.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода Start класса cl_application

№	Предикат	Действия	№ перехода
1		Ввод команды с клавиатуры и ее запись в переменную com	2
2	com не равна "Turn off the system"		3
		Перевод принтера в состояние 0; Вывод на дисплей "Turn off the system"; вернуть 0	∅
3	com равна "PC condition"	Вывод на дисплей занятость компьютера	9
			4

№	Предикат	Действия	№ перехода
4	com равна "PC идентификатор катушки, объем изделия, имя изделиия"	Создание объекта Details с именем совпадающим с именем детали; Добавление детали в очередь на компьютер	8
			5
5	com равна "Filament coil condition"	Вывод на дисплей идентификатора катушки и остатка нити филамента	9
			6
6	com равна "SHOWTREE"	Вывод на дисплей дерева иерархии объектов	9
			7
7	com равна "System status"	Вывод на дисплей такта системы, температуры принтера, состояние принтера, напечатанного объема детали, количества деталей в очереди, имен и объемов деталей в очереди	9
			9
8	Состояние принтера равно 1	Передача сигнала от компьютера принтеру с именем изделия	9
			9
9	такт не изменился после выполнения команды и состояние принтера не равно 1		10
			14
10	состояние принтера равно 2	Передача сигнала принтеру от компьютера с параметром имени текущей детали	14
			11
11	состояние принтера равно 3	Передача сигнала принтеру от компьютера с параметром имени текущей детали	14

№	Предикат	Действия	№ перехода
			12
12	состояние принтера равно 4	увеличение такта заполнения катушки на 1	13
			14
13	такт заполнения катушки равен 3	приравнять поле still_l пустой катушки полю lenght	14
			14
14	такт не изменился после выполнения команд	увеличение такта на 1	1
			1

3.25 Алгоритм метода signal_app_to_panel класса cl_application

Функционал: Сигнал для командной панели.

Параметры: string - &msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода signal_app_to_panel класса cl_application

№	Предикат	Действия	№ перехода
1		оправка сигнала	∅

3.26 Алгоритм метода handler_app_from_printer класса cl_application

Функционал: Получение сигнала от принтера.

Параметры: string - &msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода handler_app_from_printer класса cl_application

№	Предикат	Действия	№ перехода
1		увеличение такта на 1	∅

3.27 Алгоритм метода Set_state класса printer

Функционал: Установить состояние.

Параметры: int - i - новое состояние.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода Set_state класса printer

№	Предикат	Действия	№ перехода
1		Перевод принтера в состоянии i	∅

3.28 Алгоритм конструктора класса printer

Функционал: Конструктор.

Параметры: cl_base* - p_head_object - указатель на головной объект; string - s_object_name - имя объекта .

Алгоритм конструктора представлен в таблице 29.

Таблица 29 – Алгоритм конструктора класса printer

№	Предикат	Действия	№ перехода
1		Создание объекта	∅

3.29 Алгоритм метода signal_printer_to_PC класса printer

Функционал: Сигнал компьютеру.

Параметры: string - &msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 30.

Таблица 30 – Алгоритм метода signal_printer_to_PC класса printer

№	Предикат	Действия	№ перехода
1		отправка сигнала	∅

3.30 Алгоритм метода signal_printer_to_app класса printer

Функционал: Сигнал приложению.

Параметры: string - &msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 31.

Таблица 31 – Алгоритм метода signal_printer_to_app класса printer

№	Предикат	Действия	№ перехода
1		отправка сигнала	∅

3.31 Алгоритм метода signal_printer_to_panel класса printer

Функционал: Сигнал командной панели.

Параметры: string - &msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 32.

Таблица 32 – Алгоритм метода signal_printer_to_panel класса printer

№	Предикат	Действия	№ перехода
1		отправка сигнала	∅

3.32 Алгоритм метода signal_printer_to_display класса printer

Функционал: Сигнал дисплею.

Параметры: string - &msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 33.

Таблица 33 – Алгоритм метода signal_printer_to_display класса printer

№	Предикат	Действия	№ перехода
1		отправка сигнала	∅

3.33 Алгоритм метода handler_printer_from_panel класса printer

Функционал: Сигнал от командной панели.

Параметры: string - &msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 34.

Таблица 34 – Алгоритм метода handler_printer_from_panel класса printer

№	Предикат	Действия	№ перехода
1	msg равно "Write"	Ввод с клавиатуры значений полей принтера	∅
			2
2	msg равно "Coils"	Создание катушки с филаментом с вводом полей с клавиатуры; добавление катушки в массив всех катушек	∅
			∅

3.34 Алгоритм метода handler_printer_from_PC класса printer

Функционал: Сигнал от компьютера.

Параметры: string - &msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 35.

Таблица 35 – Алгоритм метода handler_printer_from_PC класса printer

№	Предикат	Действия	№ перехода
1	Состояние принтера равно 2	создание объекта детали и катушки по свойству name_of_detail	2
			4
2	Напечатанный объем равен объему детали	Обнуление напечатанного объема; обнулевние name_of_detail = ""; перевод принтера в состояние 1; Увеличение такта	∅
		Начало печати	3
3	Филамент в катушке кончился	Перевод принтера в состояние 4	∅
		Напечатать speed_of_print филамента	∅
4	Состояние принтера равно 3	создание объектов катушки и детали по имени name_of_detail	5
			6
5	Температура принтера равна температуре плавления катушки	Перевод принтера в состояние 2; передача тактового сигнала	∅
		Начало нагрева принтера	∅
6	Состояние принтера равно 1	Присвоить name_of_detail - msg; Удалить деталь из очереди компьютера; Передача тактового сигнала	7

№	Предикат	Действия	№ перехода
			∅
7	Температура принтера равна температуре плавления катушки	Перевод принтера в состояние 2	∅
		Перевод принтера в состояние 3	∅

3.35 Алгоритм конструктора класса PC

Функционал: Конструктор.

Параметры: cl_base* - p_head_object - указатель на головной объект; string - s_object_name - имя объекта .

Алгоритм конструктора представлен в таблице 36.

Таблица 36 – Алгоритм конструктора класса PC

№	Предикат	Действия	№ перехода
1		Создание объекта	∅

3.36 Алгоритм метода signal_PC_to_printer класса PC

Функционал: Сигнал принтеру.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 37.

Таблица 37 – Алгоритм метода signal_PC_to_printer класса PC

№	Предикат	Действия	№ перехода
1		Отправка сигнала	∅

3.37 Алгоритм метода handler_PC_from_panel класса PC

Функционал: Сигнал от командной панели.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 38.

Таблица 38 – Алгоритм метода handler_PC_from_panel класса PC

№	Предикат	Действия	№ перехода
1	Сообщение равно "condition"	Вывод на дисплей загруженность компьютера	∅
			2
2	Сообщение равно "queue"	Вывод на дисплей размера очереди	∅
			3
3	Сообщение равно "PC"	Вывод на дисплей деталей в очереди	∅
		Добавить деталь с именем msg в очередь	∅

3.38 Алгоритм метода handler_PC_from_printer класса PC

Функционал: Сигнал от принтера.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 39.

Таблица 39 – Алгоритм метода handler_PC_from_printer класса PC

№	Предикат	Действия	№ перехода
1	Сообщение равно "Delete"	Удалить деталь из очереди	∅
			∅

3.39 Алгоритм конструктора класса Display

Функционал: Конструктор.

Параметры: cl_base* - p_head_object - указатель на головной объект; string - s_object_name - имя объекта .

Алгоритм конструктора представлен в таблице 40.

Таблица 40 – Алгоритм конструктора класса Display

№	Предикат	Действия	№ перехода
1		Создание объекта	∅

3.40 Алгоритм метода handler_display_from_panel класса Display

Функционал: Сигнал от командной панели.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 41.

Таблица 41 – Алгоритм метода handler_display_from_panel класса Display

№	Предикат	Действия	№ перехода
1	msg = "SHOWTREE"	Вывод на экран дерева иерархии объектов	∅
		выод на экран msg	∅

3.41 Алгоритм конструктора класса Details

Функционал: Конструктор.

Параметры: cl_base* - p_head_object - указатель на головной объект; string - s_object_name - имя объекта .

Алгоритм конструктора представлен в таблице 42.

Таблица 42 – Алгоритм конструктора класса Details

№	Предикат	Действия	№ перехода
1		Создание объекта	∅

3.42 Алгоритм конструктора класса coils

Функционал: Конструктор.

Параметры: cl_base* - p_head_object - указатель на головной объект; string - s_object_name - имя объекта; string - id - идентификатор катушки; int - l - температура нагрева филамента; int - t - длина нити филамента.

Алгоритм конструктора представлен в таблице 43.

Таблица 43 – Алгоритм конструктора класса coils

№	Предикат	Действия	№ перехода
1		Присвоить полям катушки значения соответствующих параметров	∅

3.43 Алгоритм конструктора класса Command_panel

Функционал: Конструктор.

Параметры: cl_base* - p_head_object - указатель на головной объект; string - s_object_name - имя объекта .

Алгоритм конструктора представлен в таблице 44.

Таблица 44 – Алгоритм конструктора класса Command_panel

№	Предикат	Действия	№ перехода
1		создание объекта	∅

3.44 Алгоритм метода signal_panel_to_PC класса Command_panel

Функционал: Сигнал компьютеру.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 45.

Таблица 45 – Алгоритм метода signal_panel_to_PC класса Command_panel

№	Предикат	Действия	№ перехода
1		Отправка сигнала	∅

3.45 Алгоритм метода signal_panel_to_printer класса Command_panel

Функционал: Сигнал принтеру.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 46.

Таблица 46 – Алгоритм метода signal_panel_to_printer класса Command_panel

№	Предикат	Действия	№ перехода
1		Отправка сигнала	∅

3.46 Алгоритм метода signal_panel_to_display класса Command_panel

Функционал: Сигнал дисплею.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 47.

Таблица 47 – Алгоритм метода signal_panel_to_display класса Command_panel

№	Предикат	Действия	№ перехода
1		Отправка сигнала	∅

3.47 Алгоритм метода handler_panel_from_app класса Command_panel

Функционал: Сигнал от приложения.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 48.

Таблица 48 – Алгоритм метода handler_panel_from_app класса Command_panel

№	Предикат	Действия	№ перехода
1		Ввод с клавиатуры количества катушек с филаментом - n	2
2		отправка сигнала принтеру с сообщением Write	3
3		n раз отправка сигнала принтеру с сообщением "Coils"	4
4		Ввод команды завершения с клавиатуры - end	5
5	end = "End of settings"	Вывод на дисплей "Ready to work"	∅
			∅

3.48 Алгоритм метода handler_panel_from_printer класса Command_panel

Функционал: Сигнал от принтера.

Параметры: string& - msg - адрес сообщения.

Возвращаемое значение: void - не возвращает значения.

Алгоритм метода представлен в таблице 49.

Таблица 49 – Алгоритм метода handler_panel_from_printer класса Command_panel

№	Предикат	Действия	№ перехода
1		Получение сигнала	∅

3.49 Алгоритм функции main

Функционал: Основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности завершения алгоритма.

Алгоритм функции представлен в таблице 50.

Таблица 50 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_cl_application класса cl_application	2
2		Вызов метода Tree для объекта ob_cl_application	3
3		Вернуть значение метода Start для объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-38.

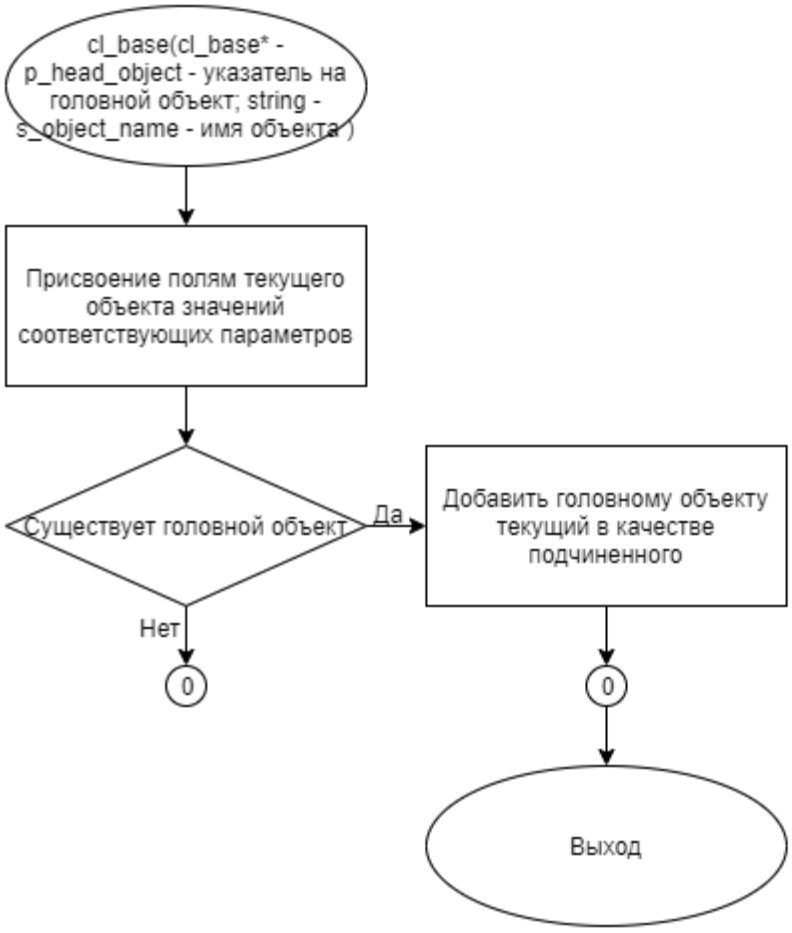

Рисунок 1 – Блок-схема алгоритма

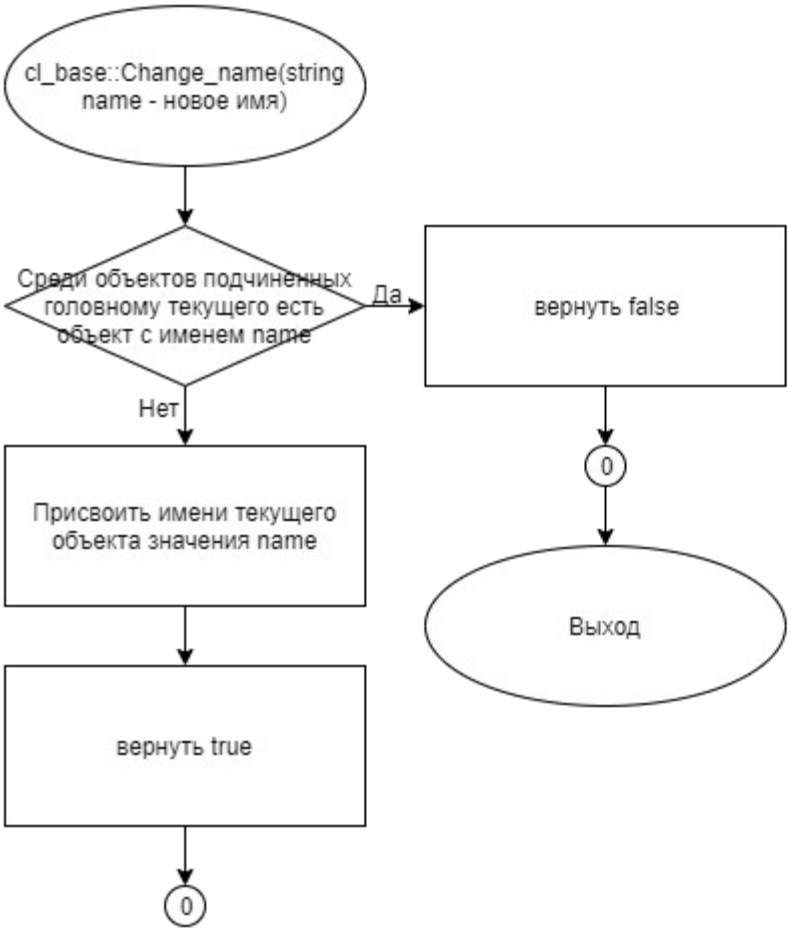

Рисунок 2 – Блок-схема алгоритма

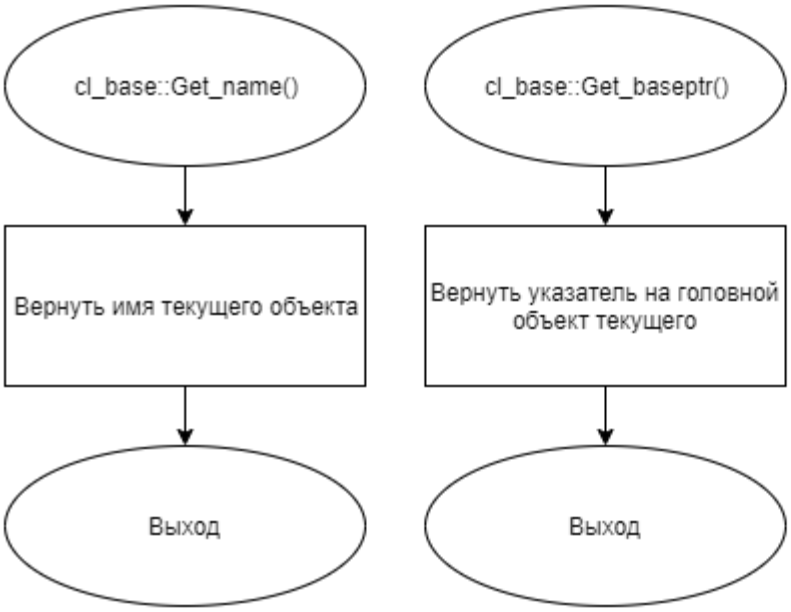

Рисунок 3 – Блок-схема алгоритма

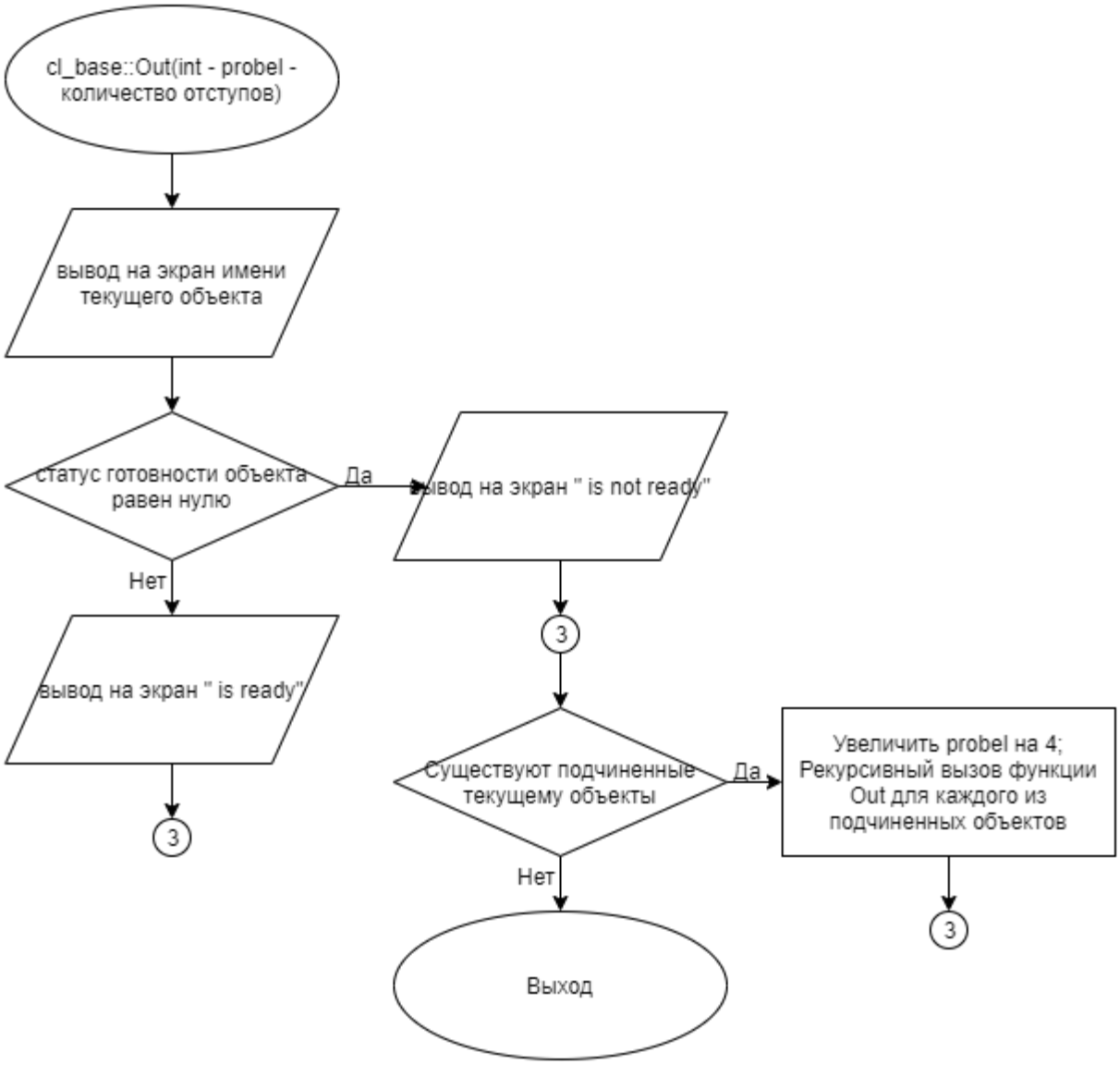

Рисунок 4 – Блок-схема алгоритма

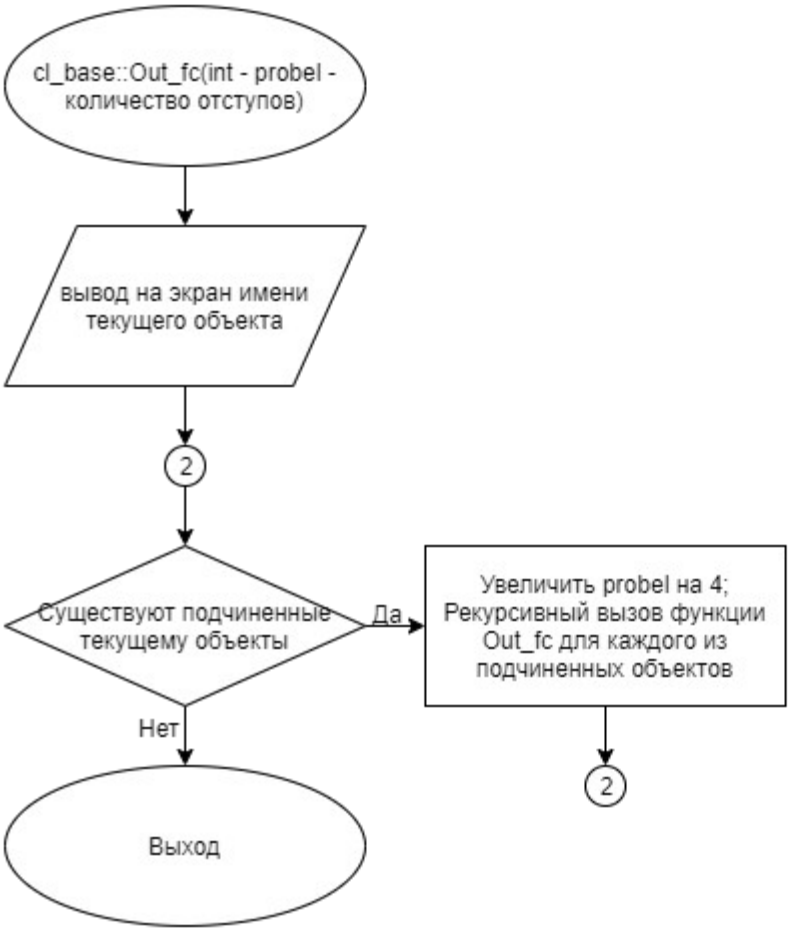

Рисунок 5 – Блок-схема алгоритма

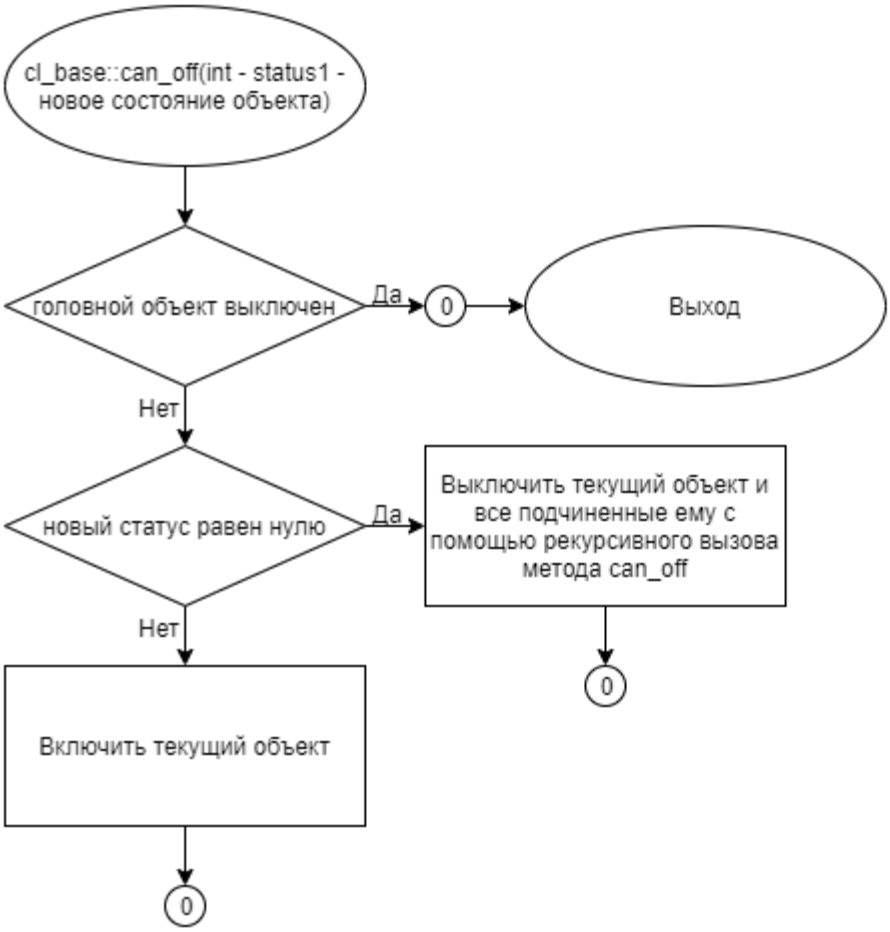

Рисунок 6 – Блок-схема алгоритма

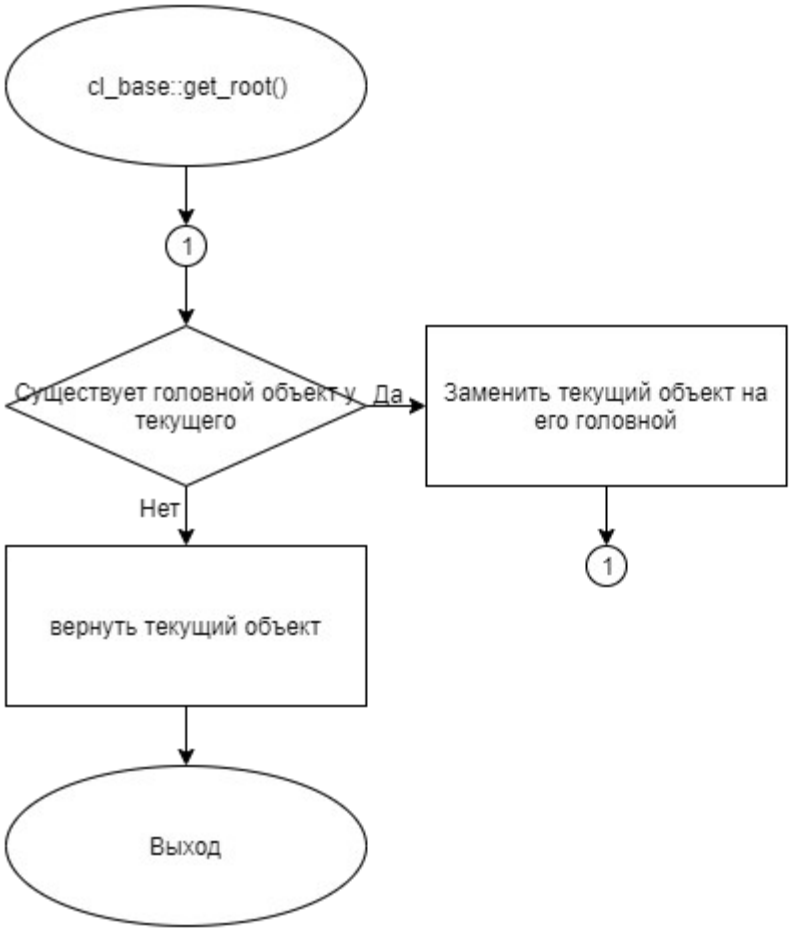

Рисунок 7 – Блок-схема алгоритма

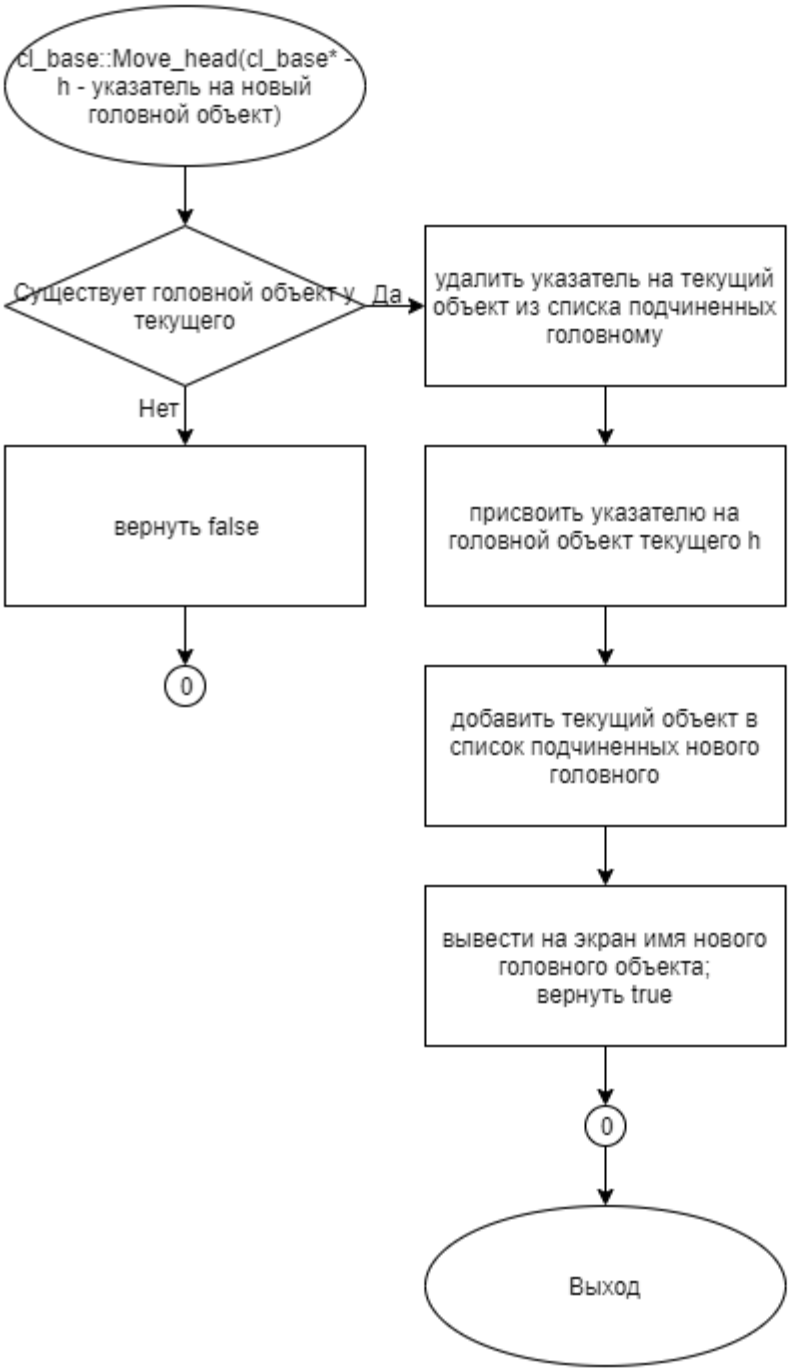

Рисунок 8 – Блок-схема алгоритма

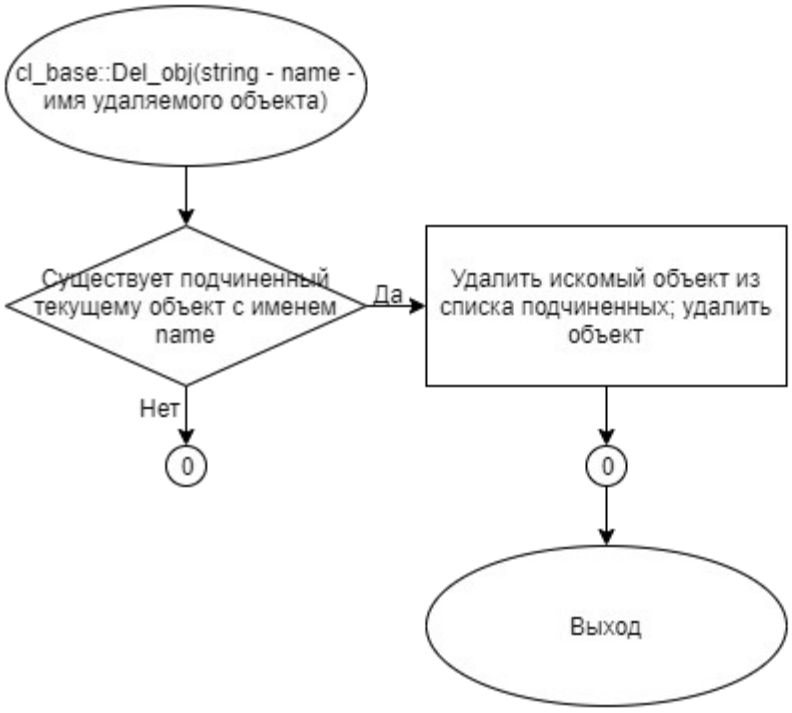

Рисунок 9 – Блок-схема алгоритма

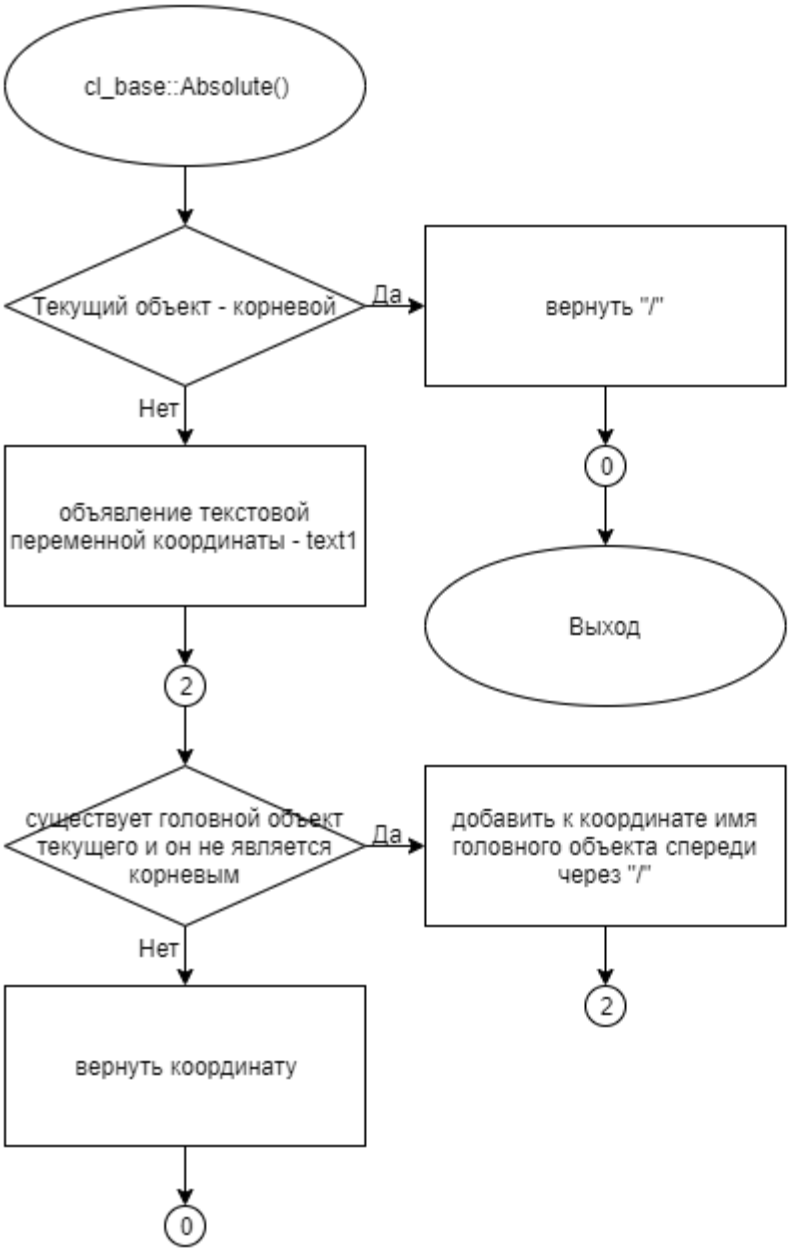

Рисунок 10 – Блок-схема алгоритма

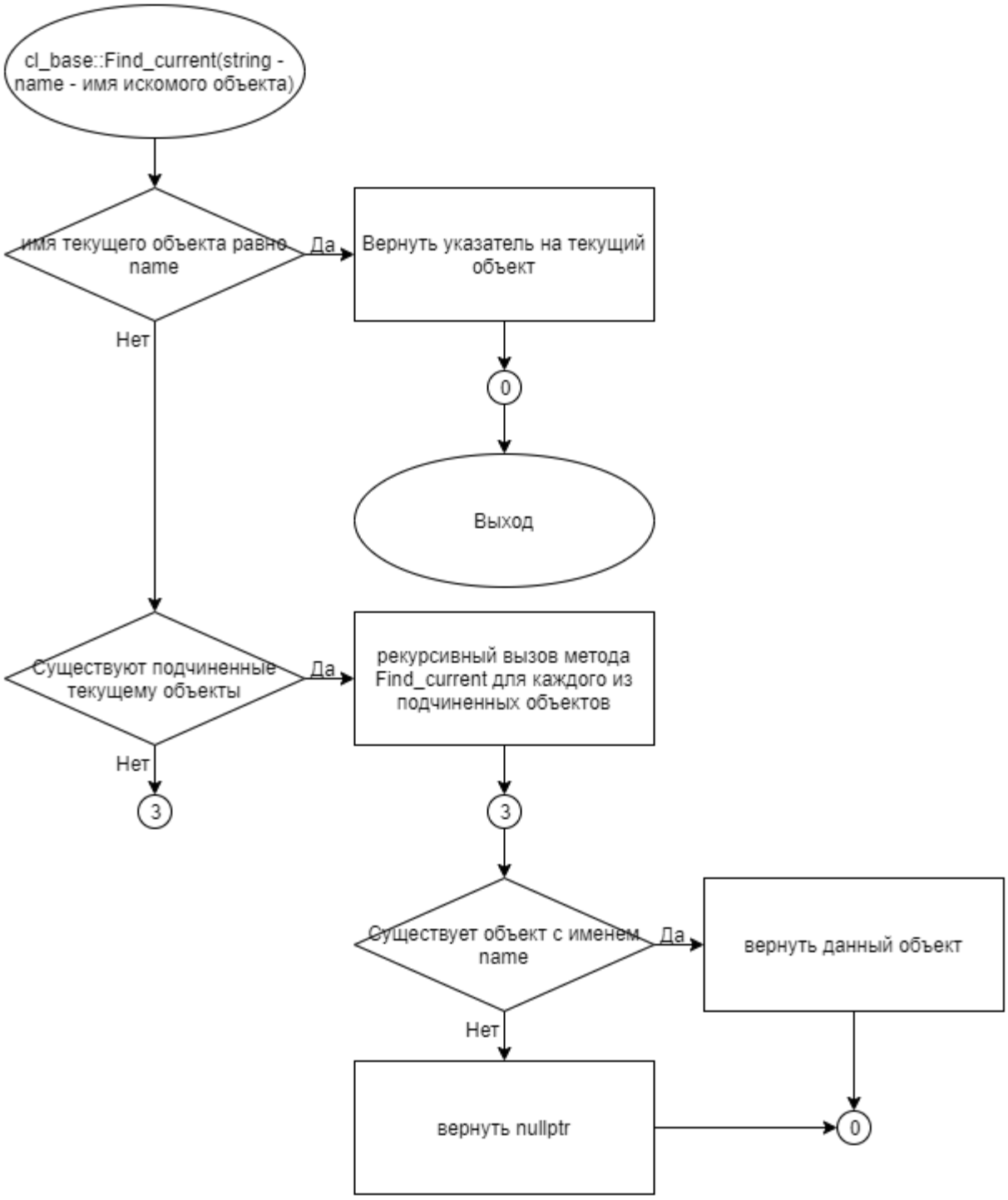

Рисунок 11 – Блок-схема алгоритма

Рисунок 12 – Блок-схема алгоритма

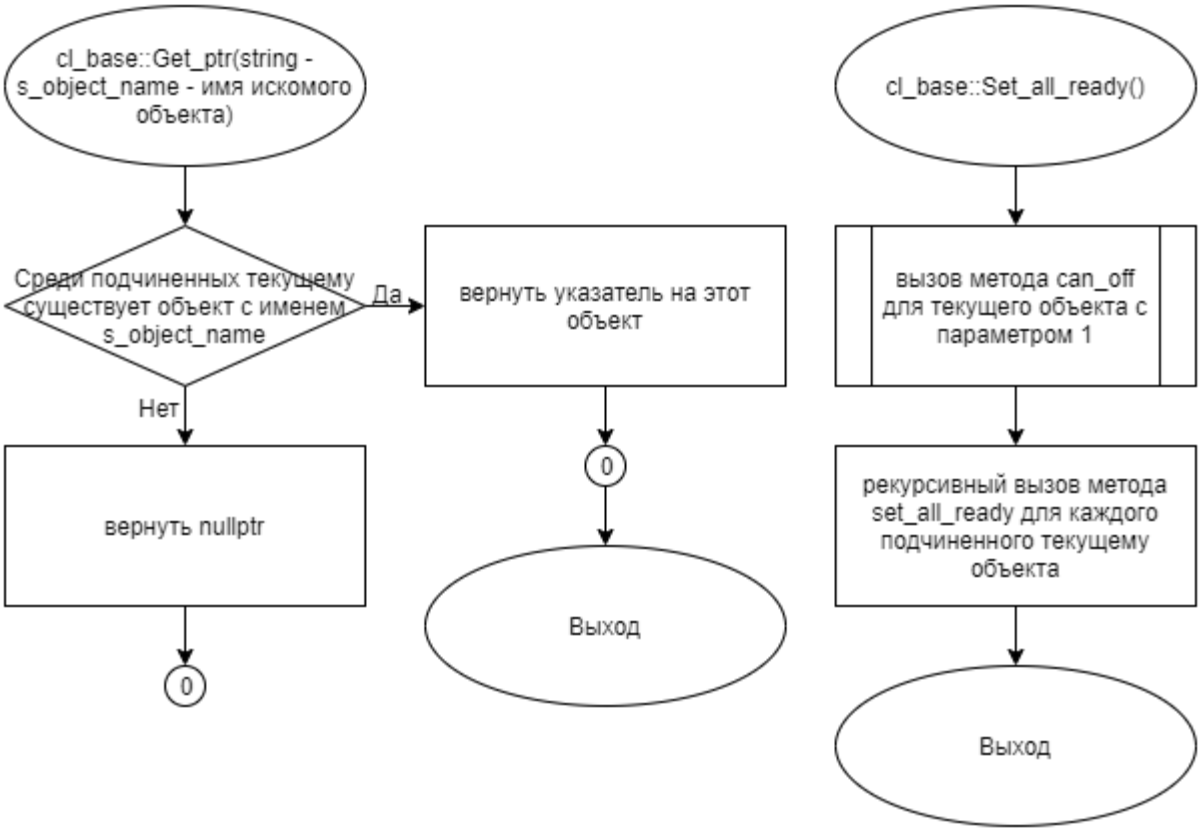

Рисунок 13 – Блок-схема алгоритма

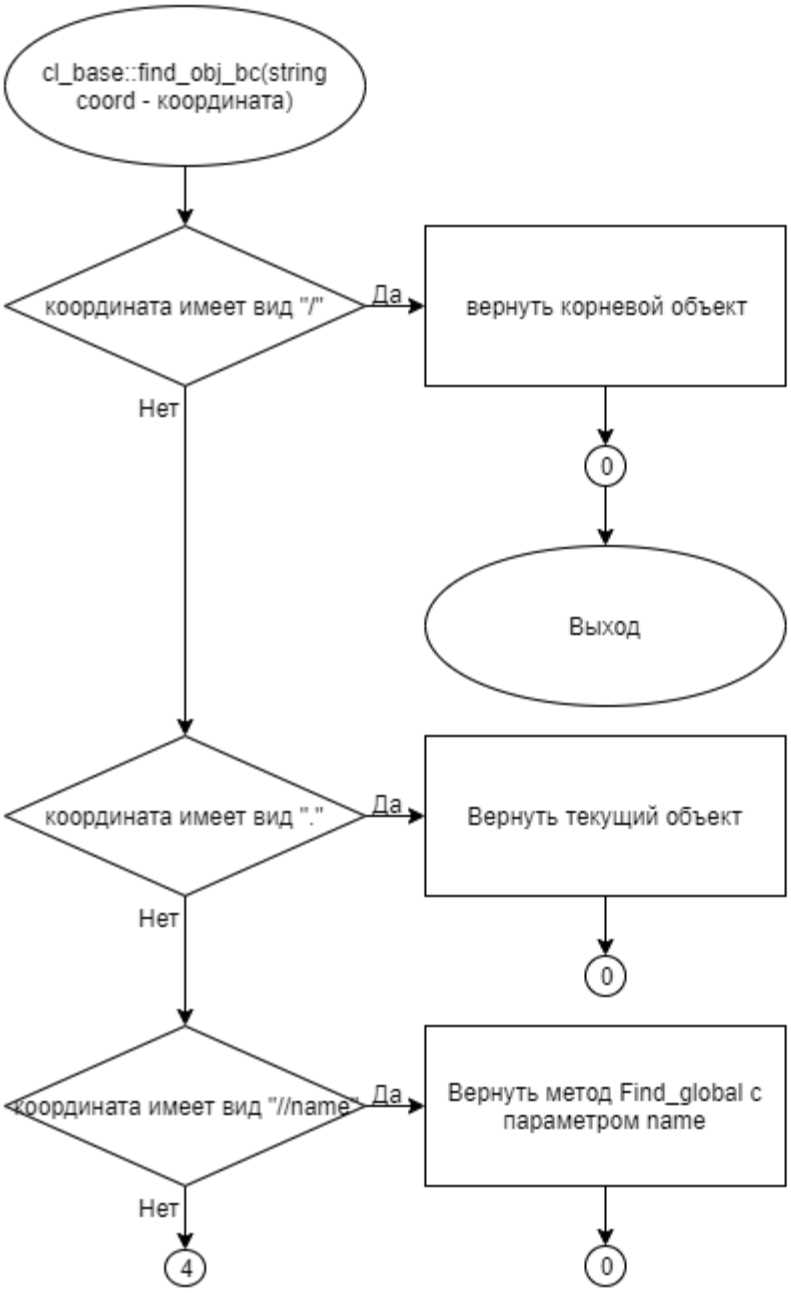

Рисунок 14 – Блок-схема алгоритма

Рисунок 15 – Блок-схема алгоритма

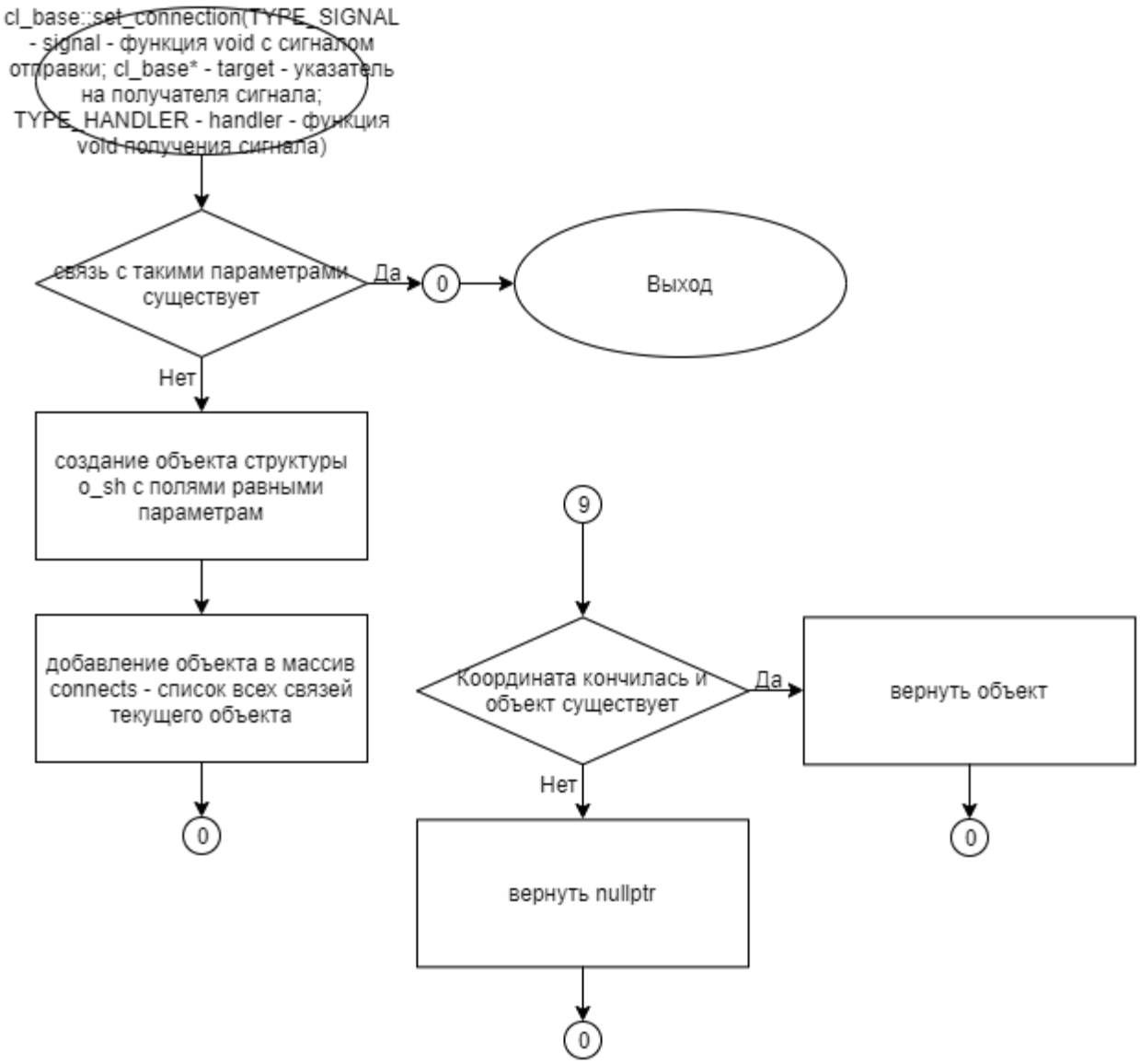

Рисунок 16 – Блок-схема алгоритма

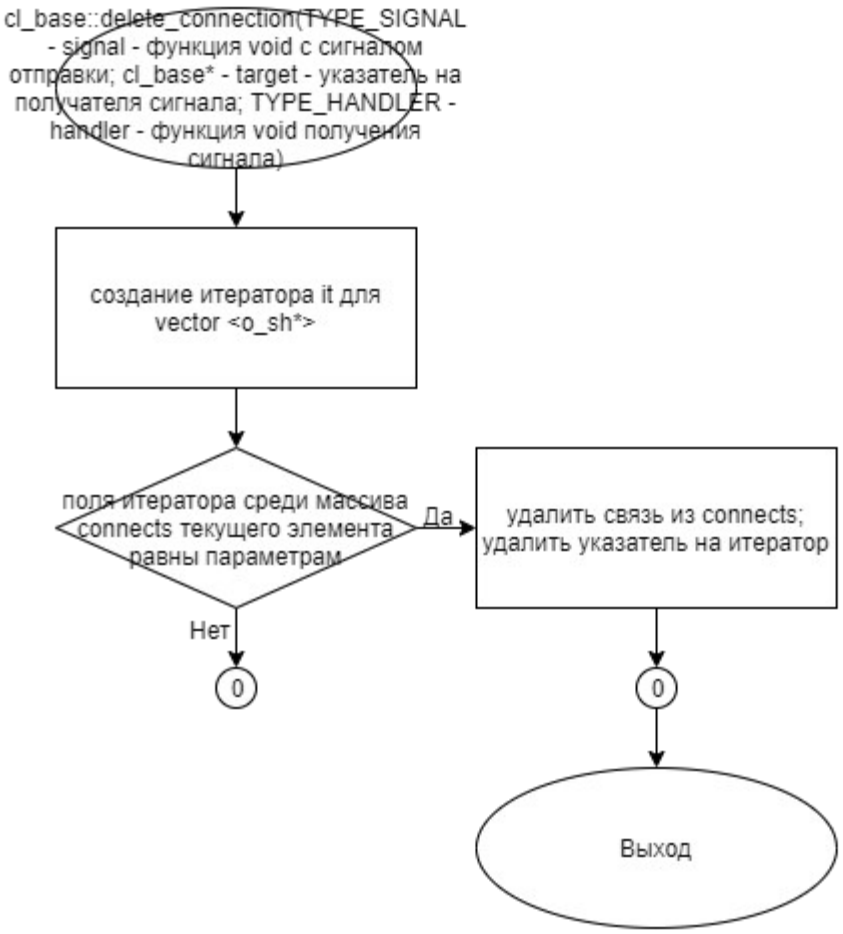

Рисунок 17 – Блок-схема алгоритма

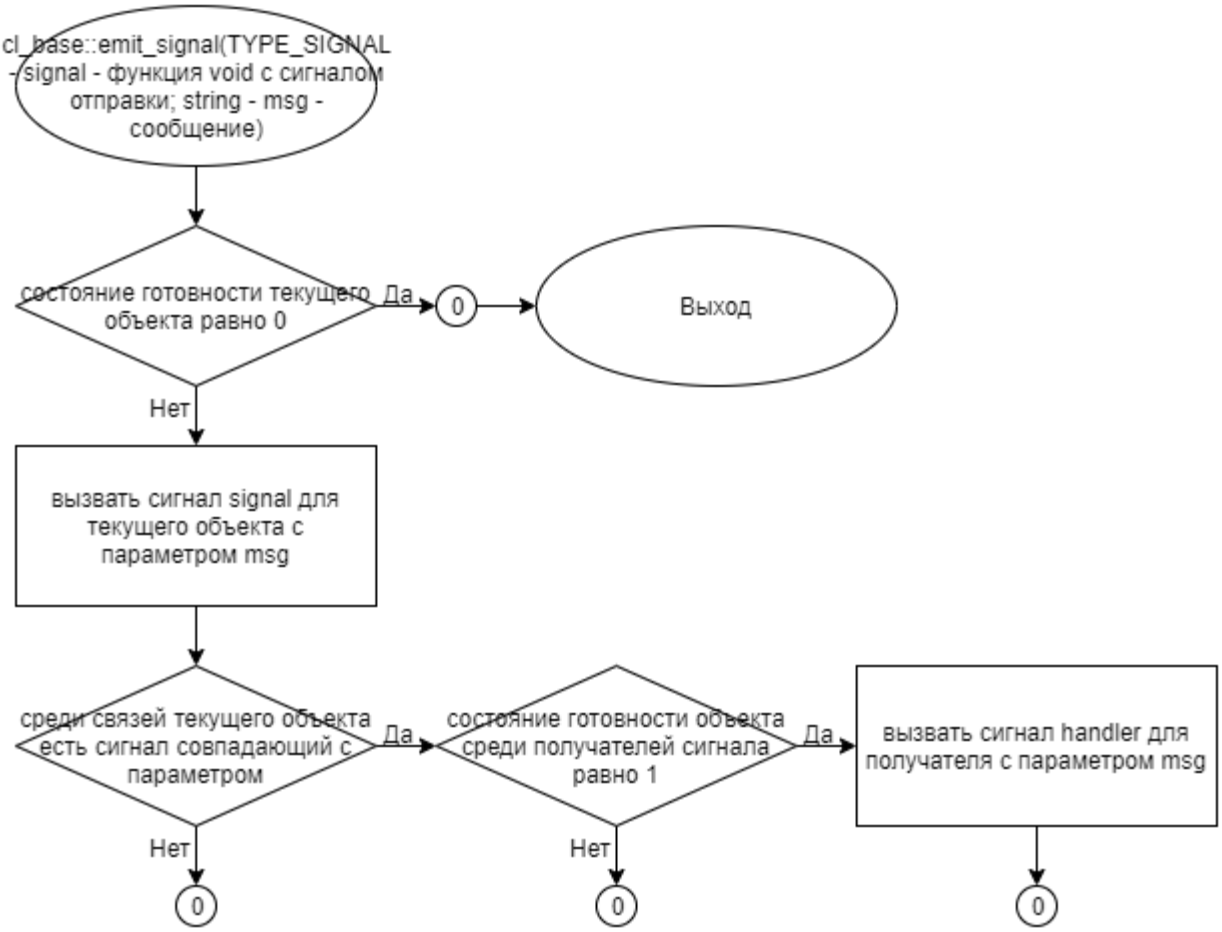

Рисунок 18 – Блок-схема алгоритма

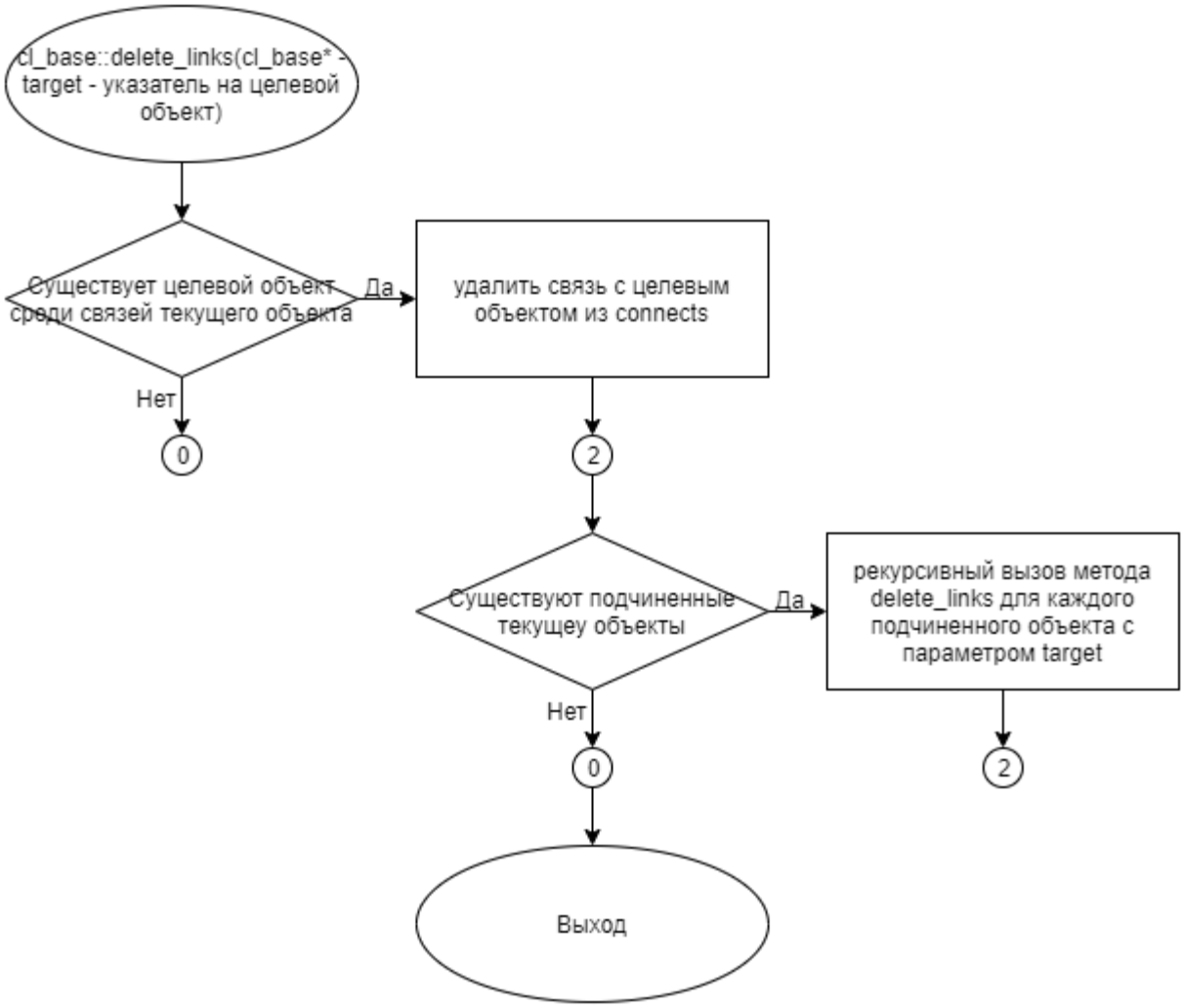

Рисунок 19 – Блок-схема алгоритма

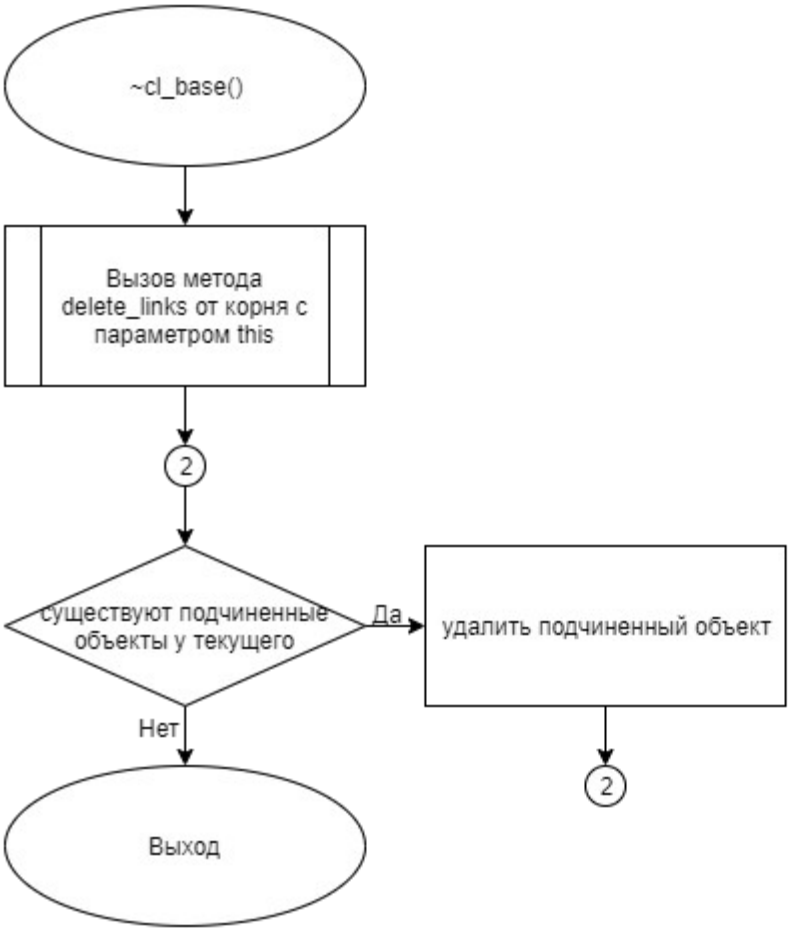

Рисунок 20 – Блок-схема алгоритма

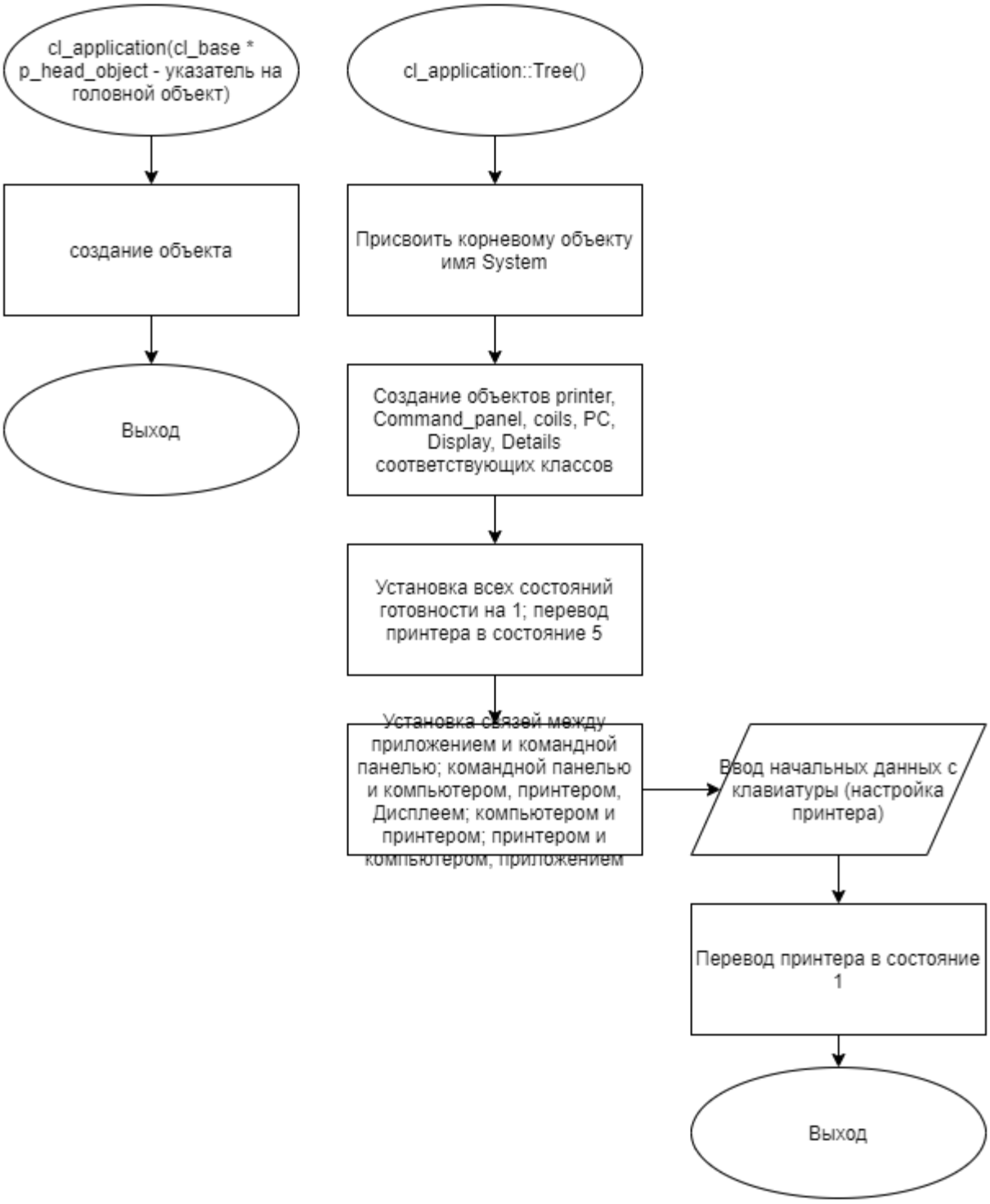

Рисунок 21 – Блок-схема алгоритма

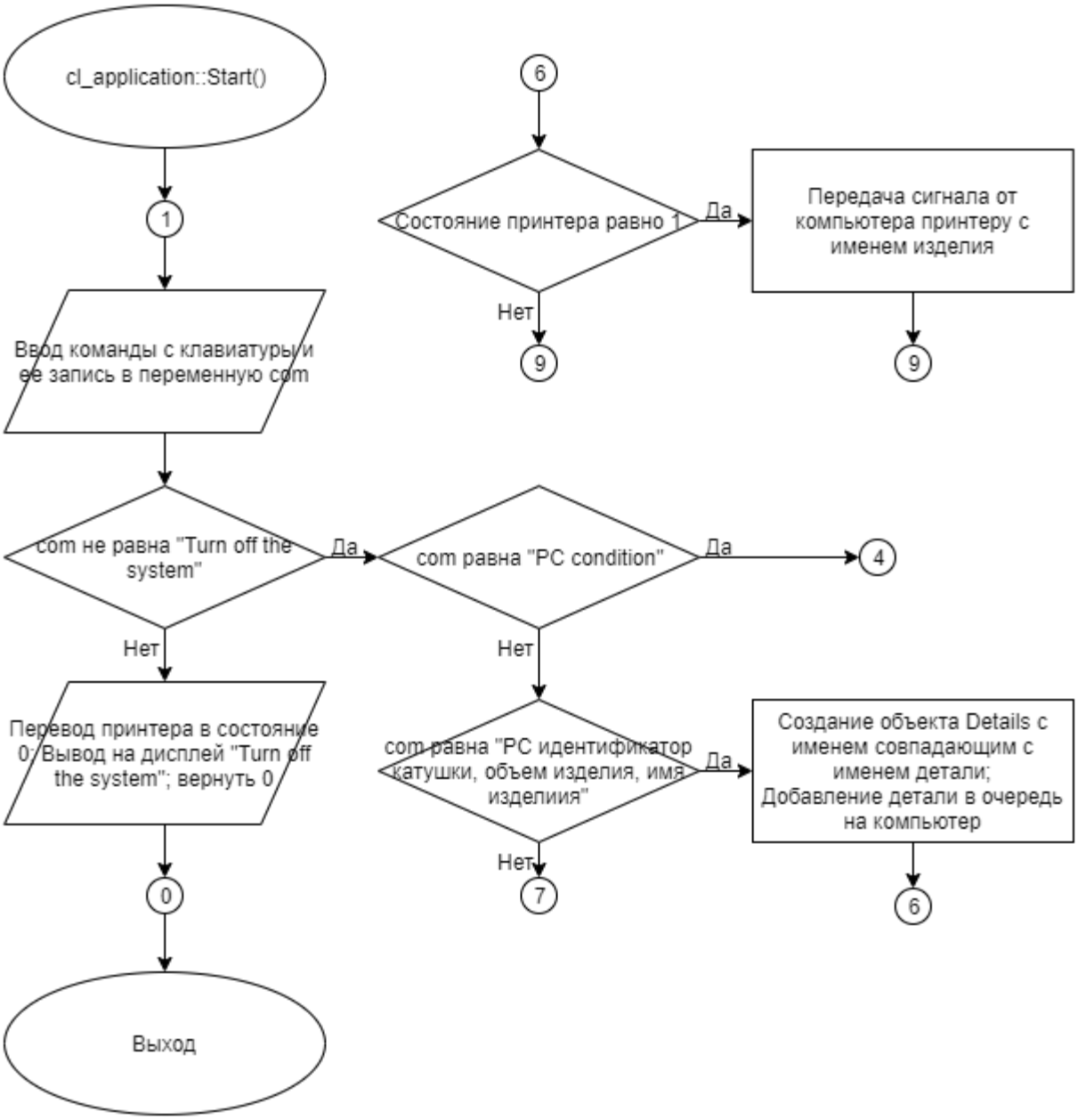

Рисунок 22 – Блок-схема алгоритма

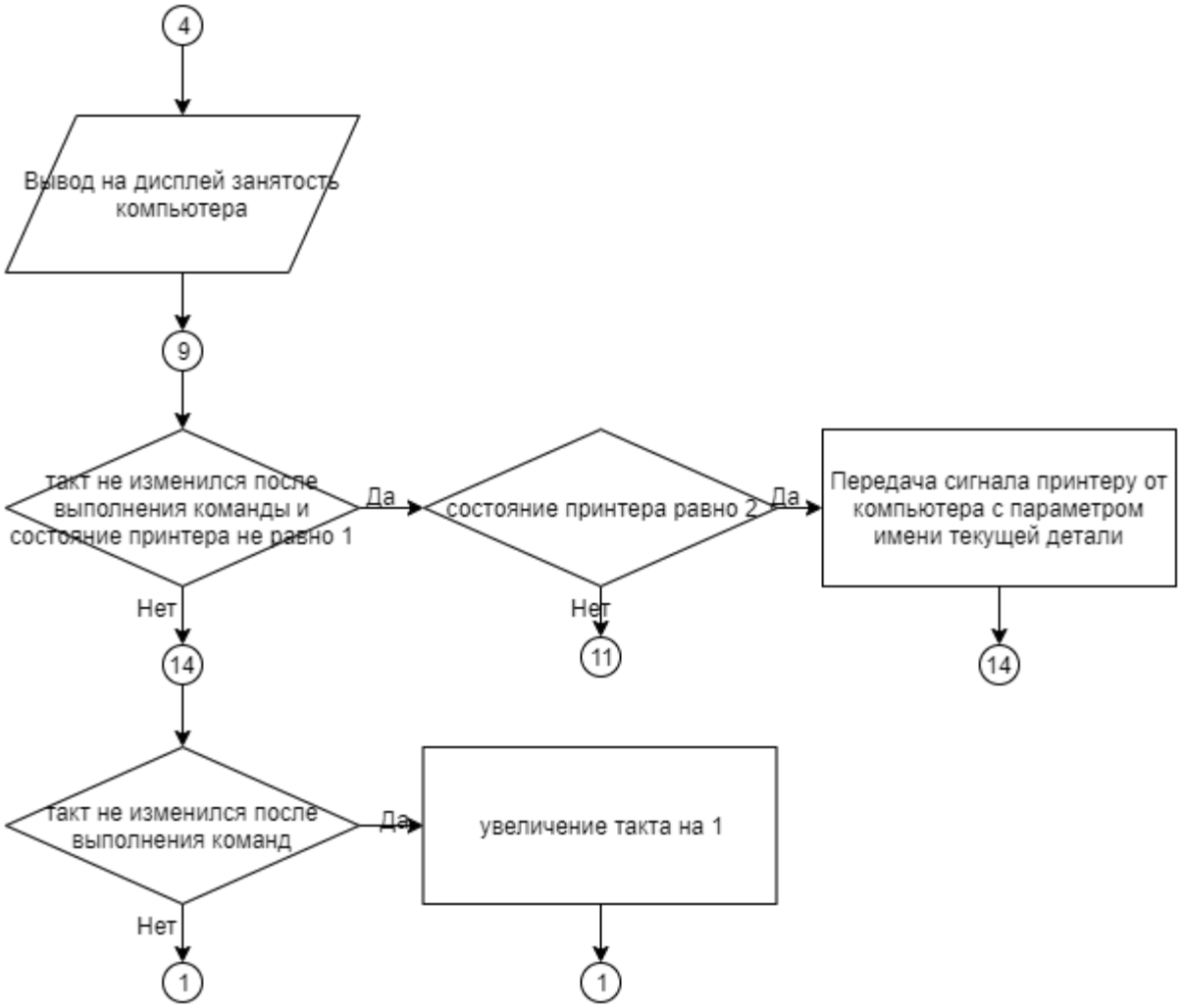

Рисунок 23 – Блок-схема алгоритма

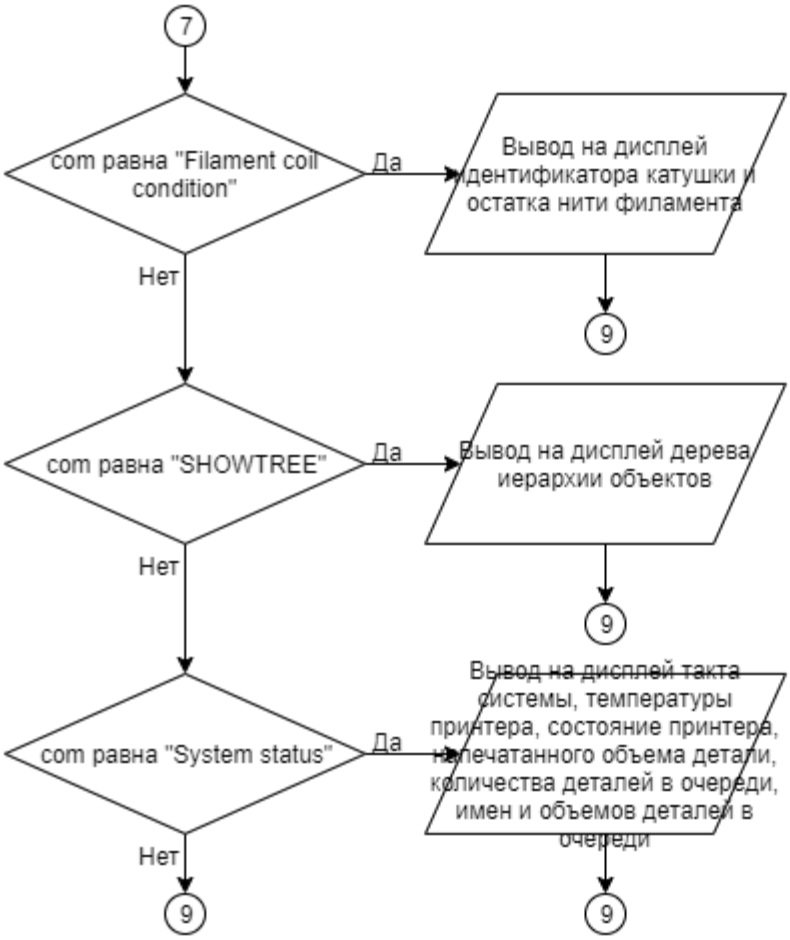

Рисунок 24 – Блок-схема алгоритма

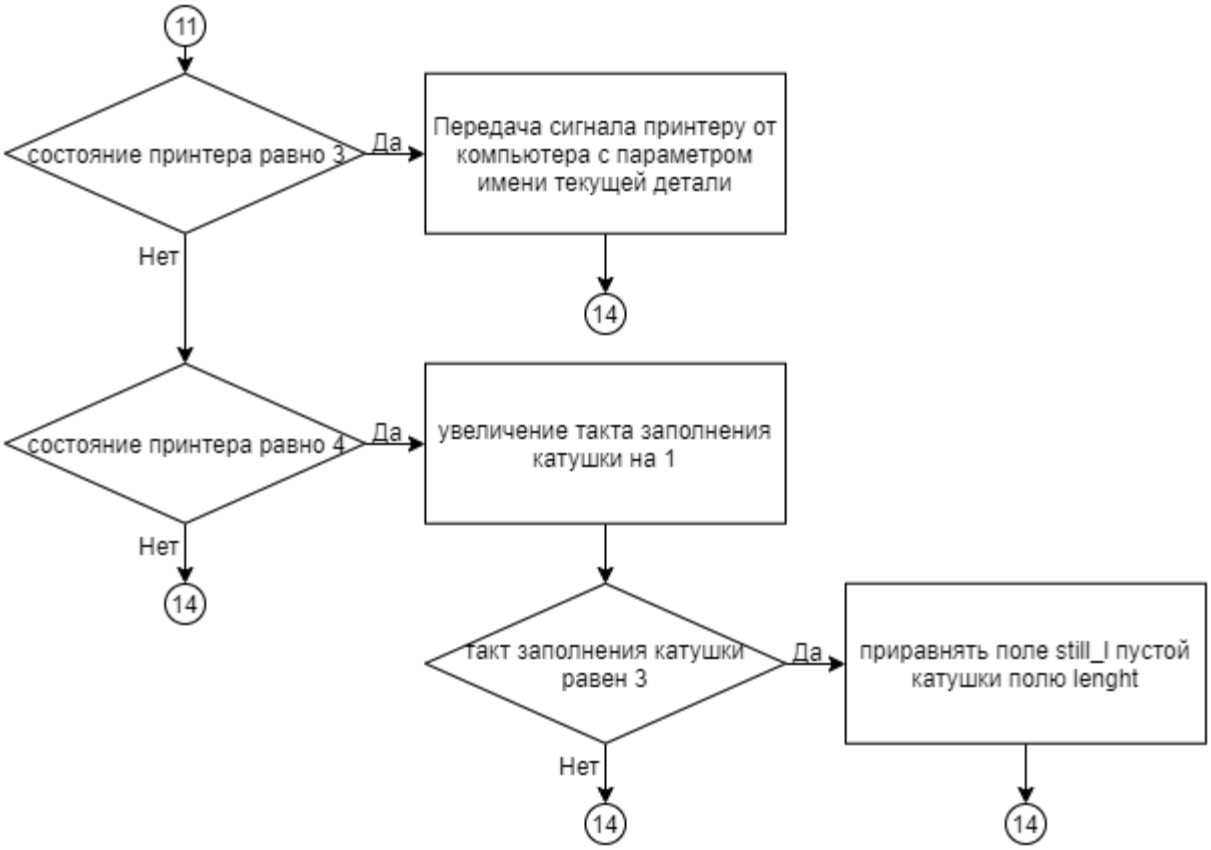

Рисунок 25 – Блок-схема алгоритма

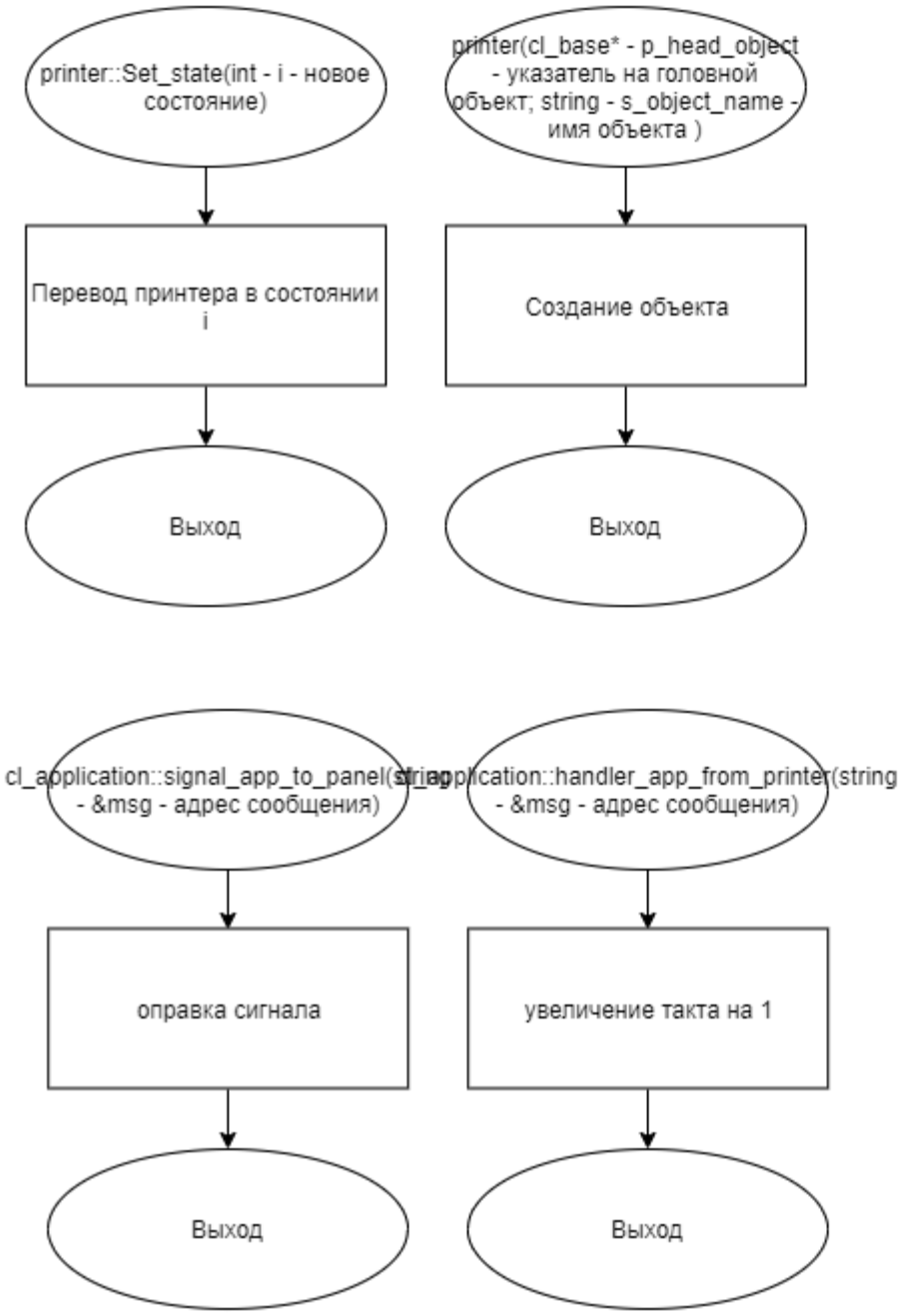

Рисунок 26 – Блок-схема алгоритма

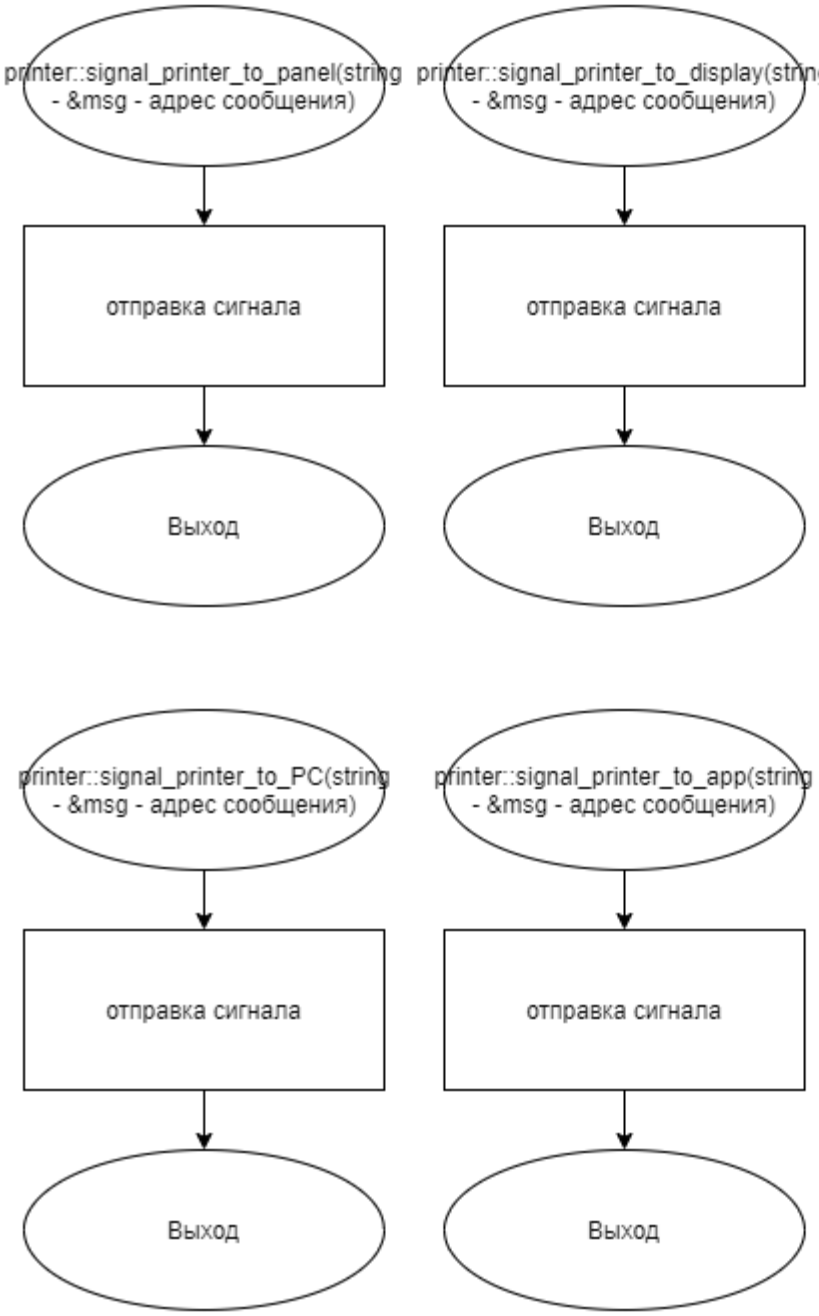

Рисунок 27 – Блок-схема алгоритма

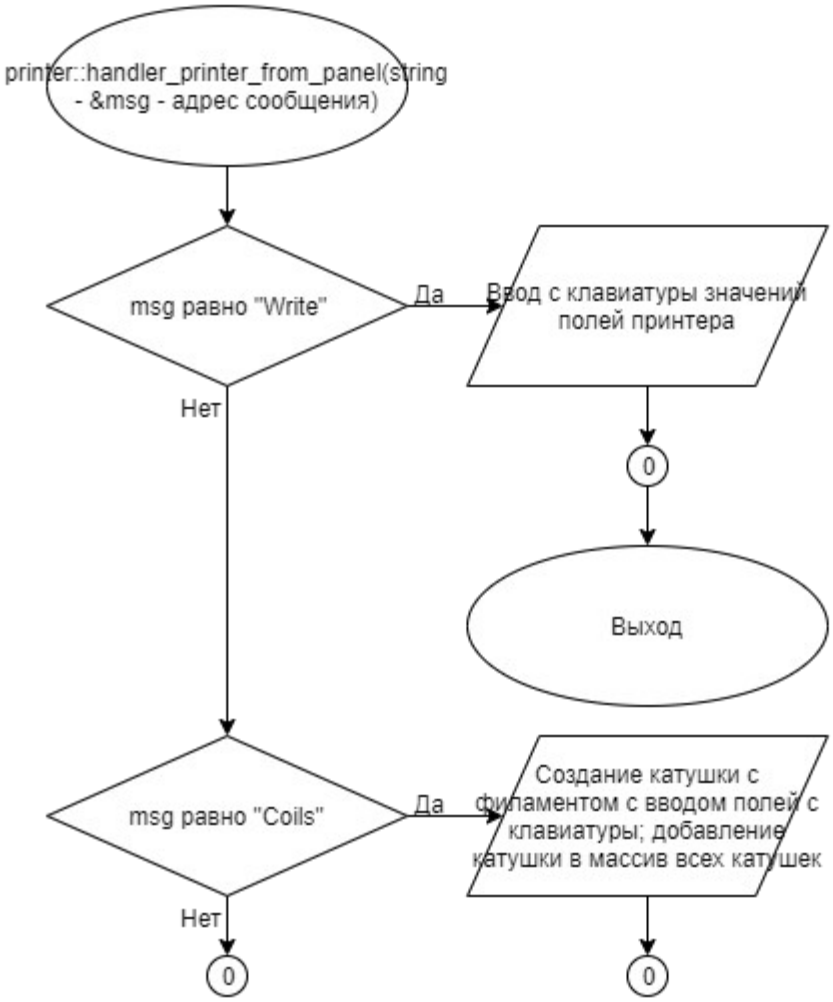

Рисунок 28 – Блок-схема алгоритма

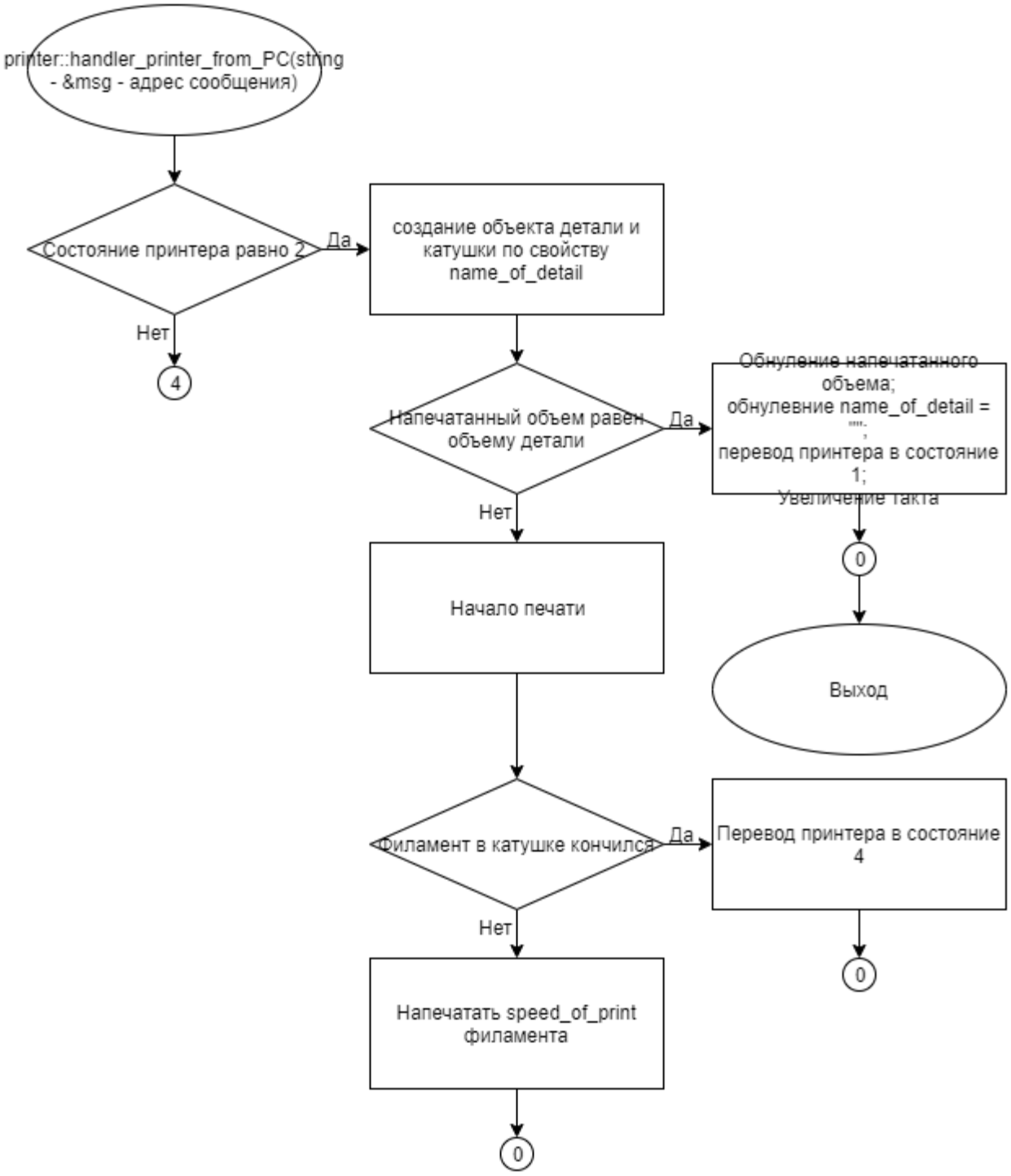

Рисунок 29 – Блок-схема алгоритма

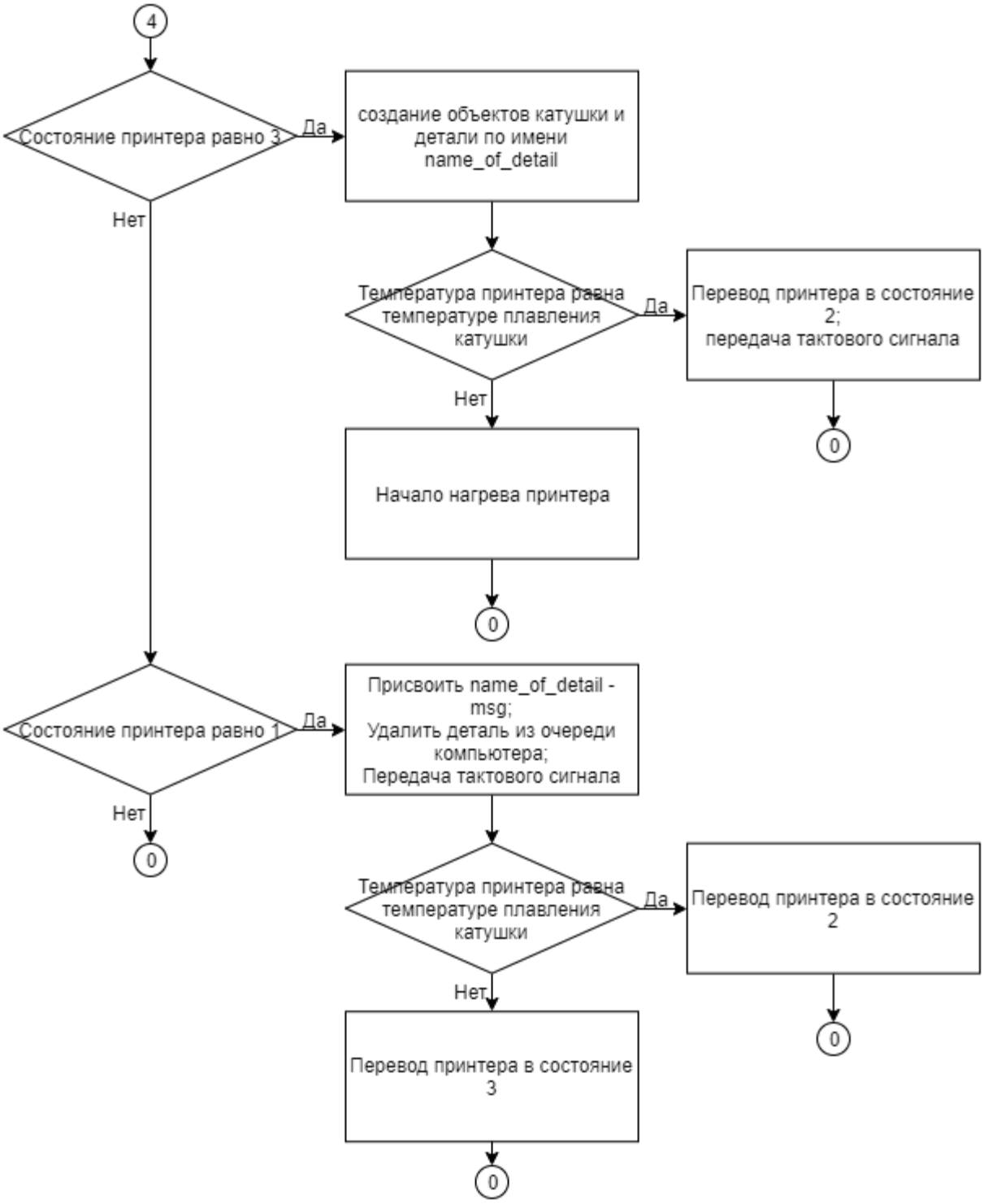

Рисунок 30 – Блок-схема алгоритма

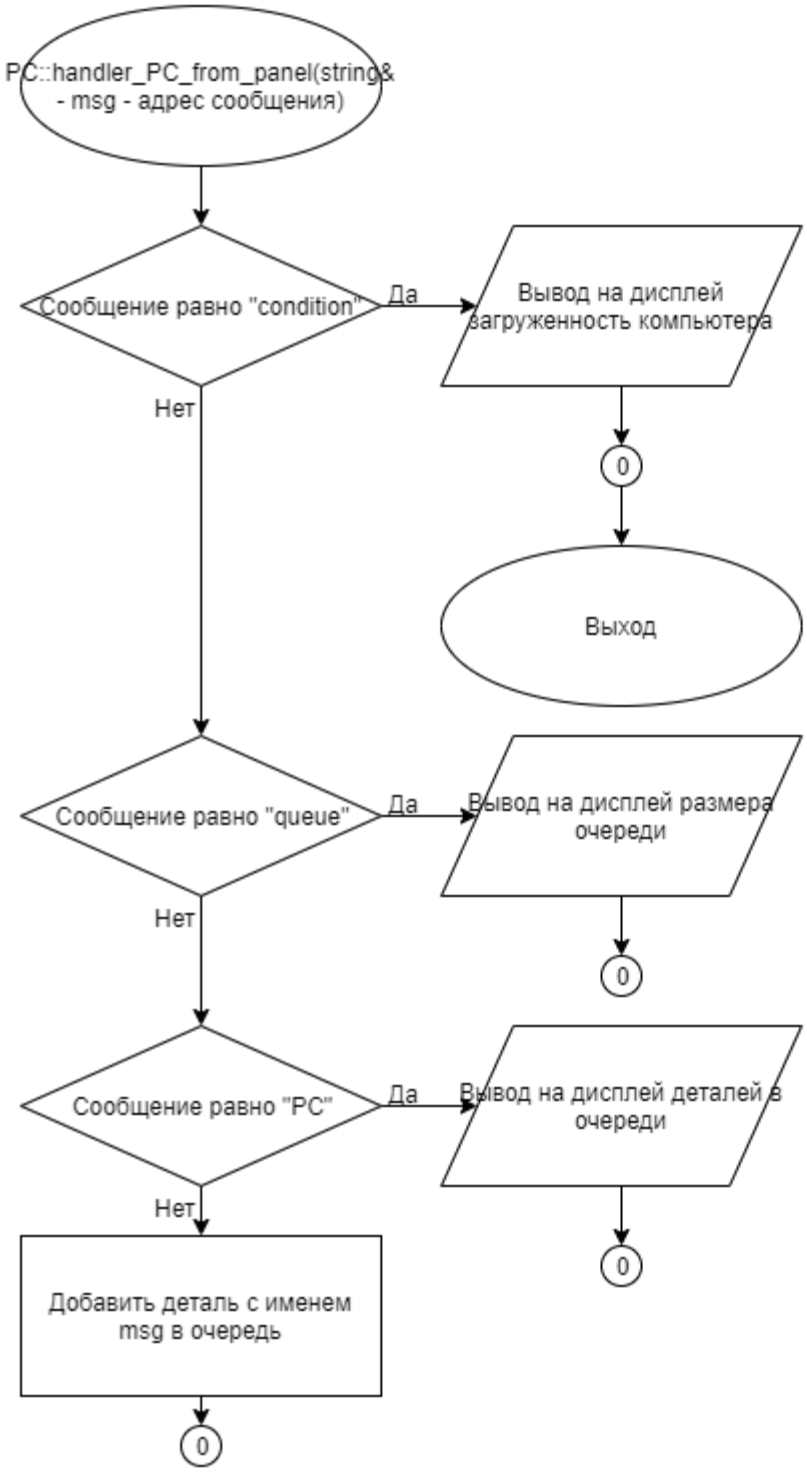

Рисунок 31 – Блок-схема алгоритма

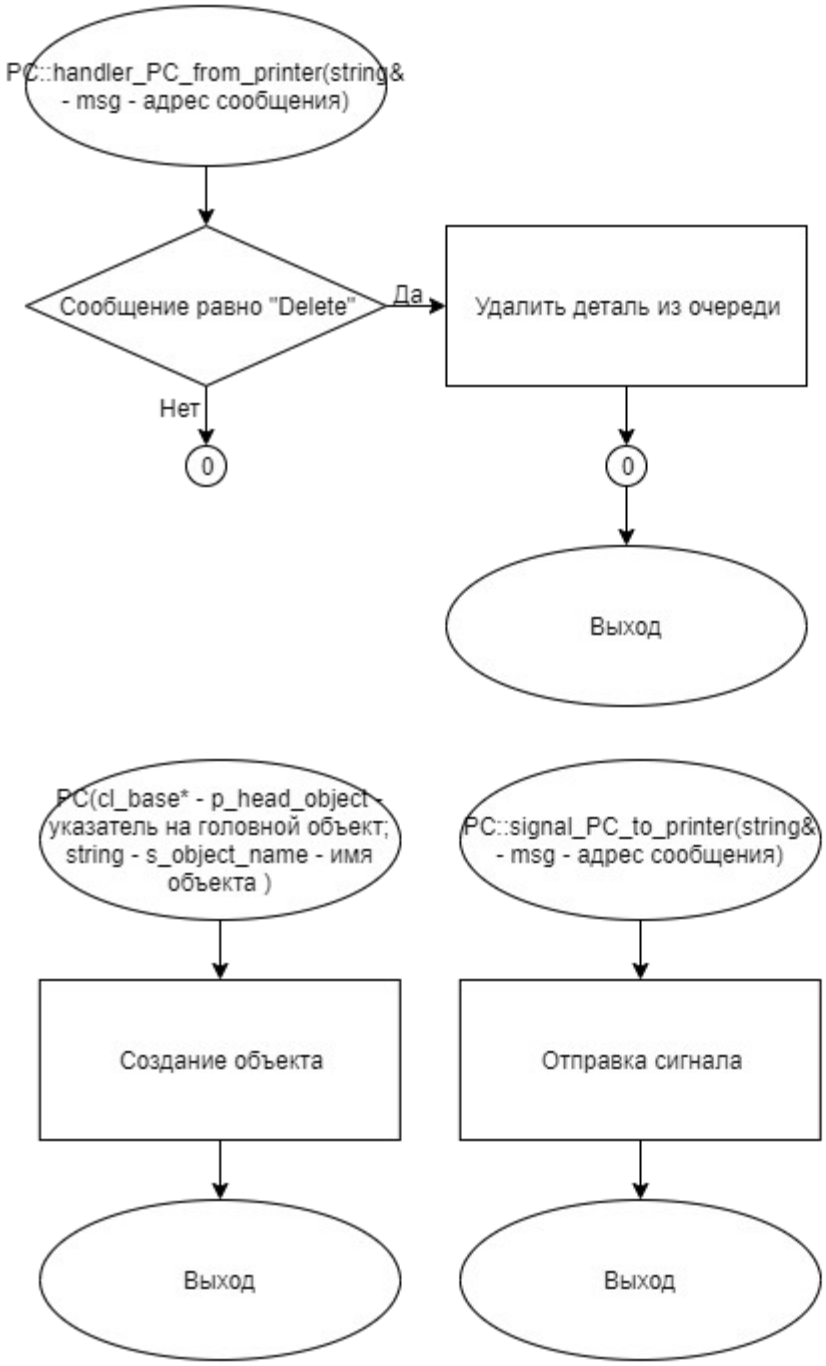

Рисунок 32 – Блок-схема алгоритма

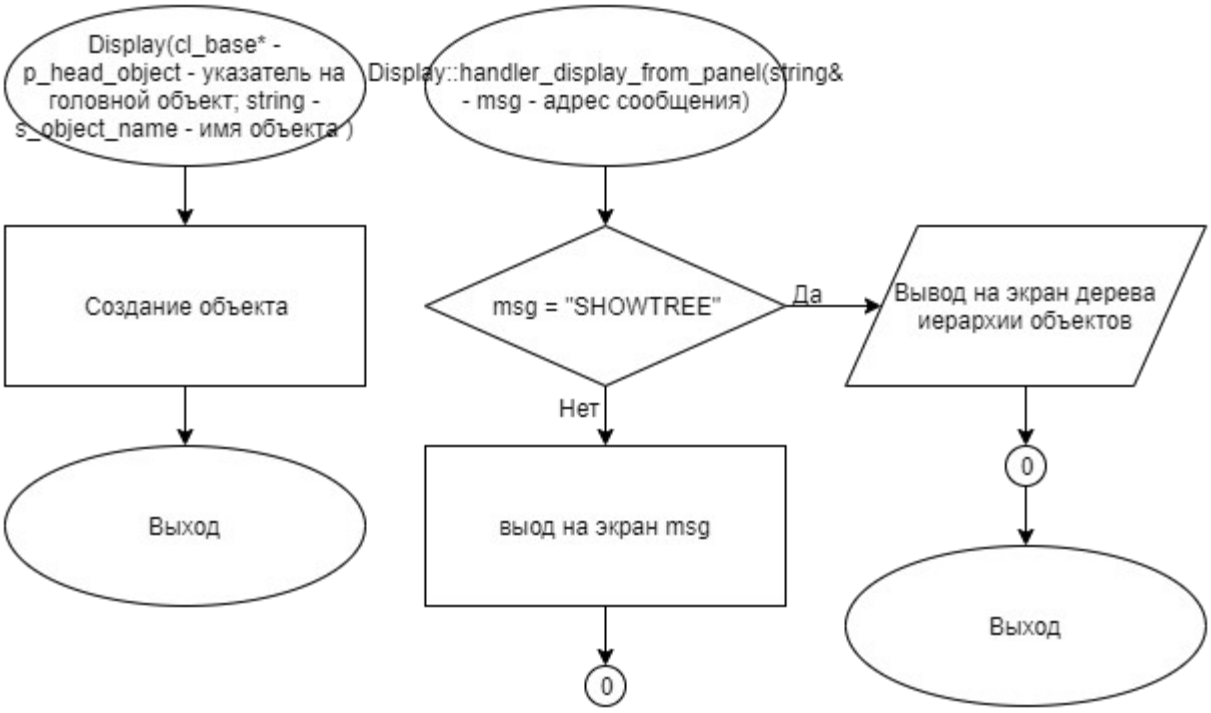

Рисунок 33 – Блок-схема алгоритма

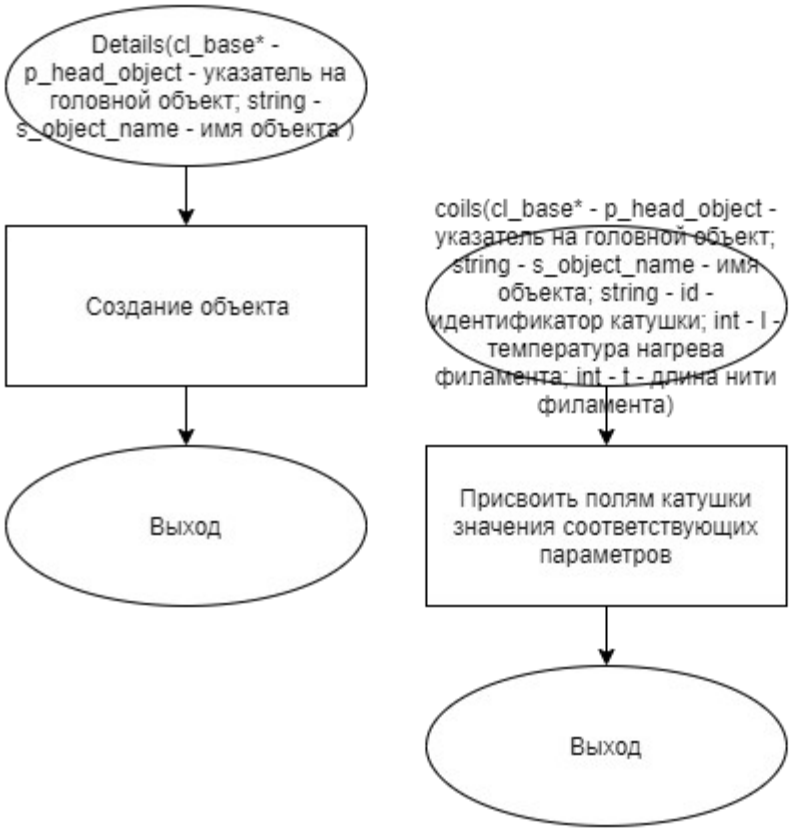

Рисунок 34 – Блок-схема алгоритма

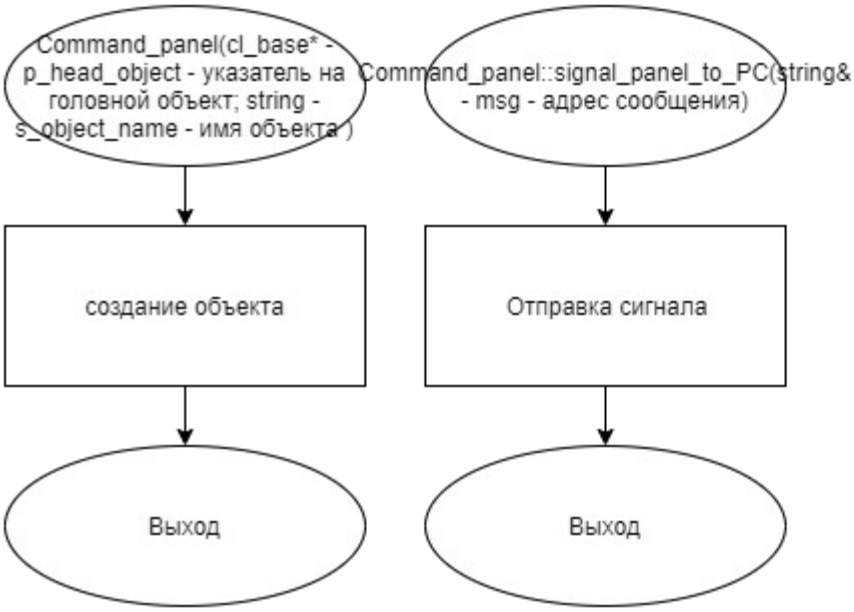

Рисунок 35 – Блок-схема алгоритма

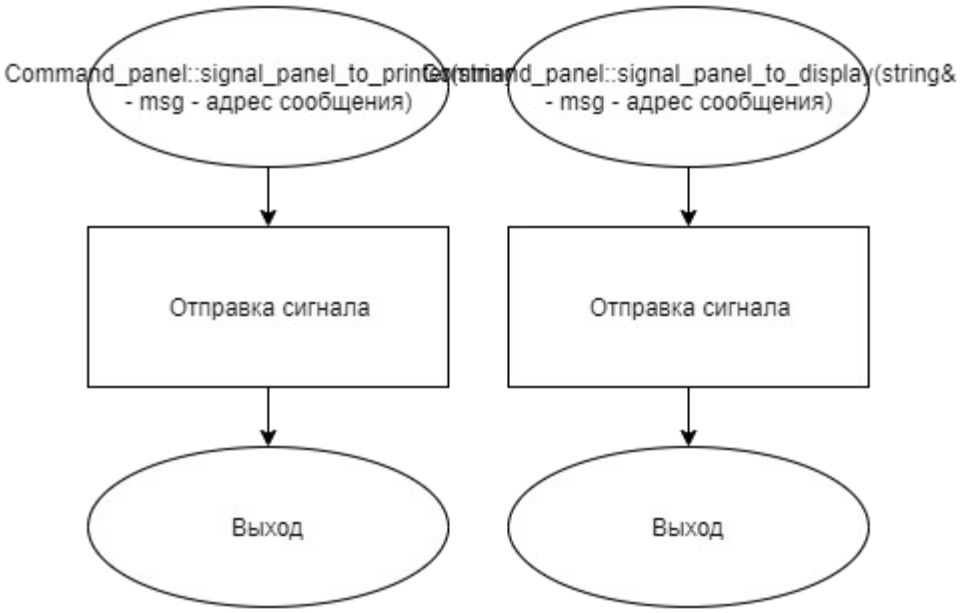

Рисунок 36 – Блок-схема алгоритма

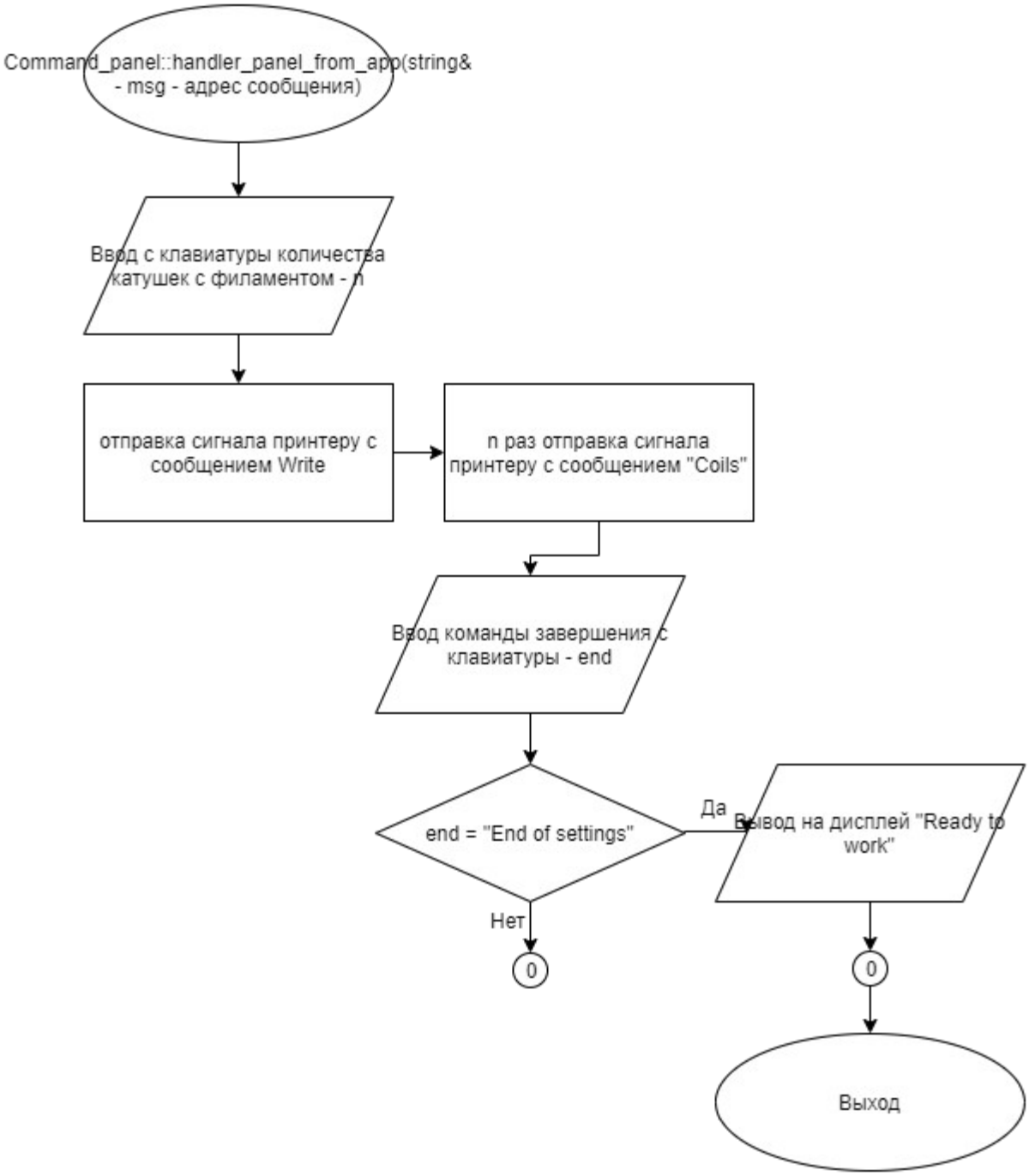

Рисунок 37 – Блок-схема алгоритма

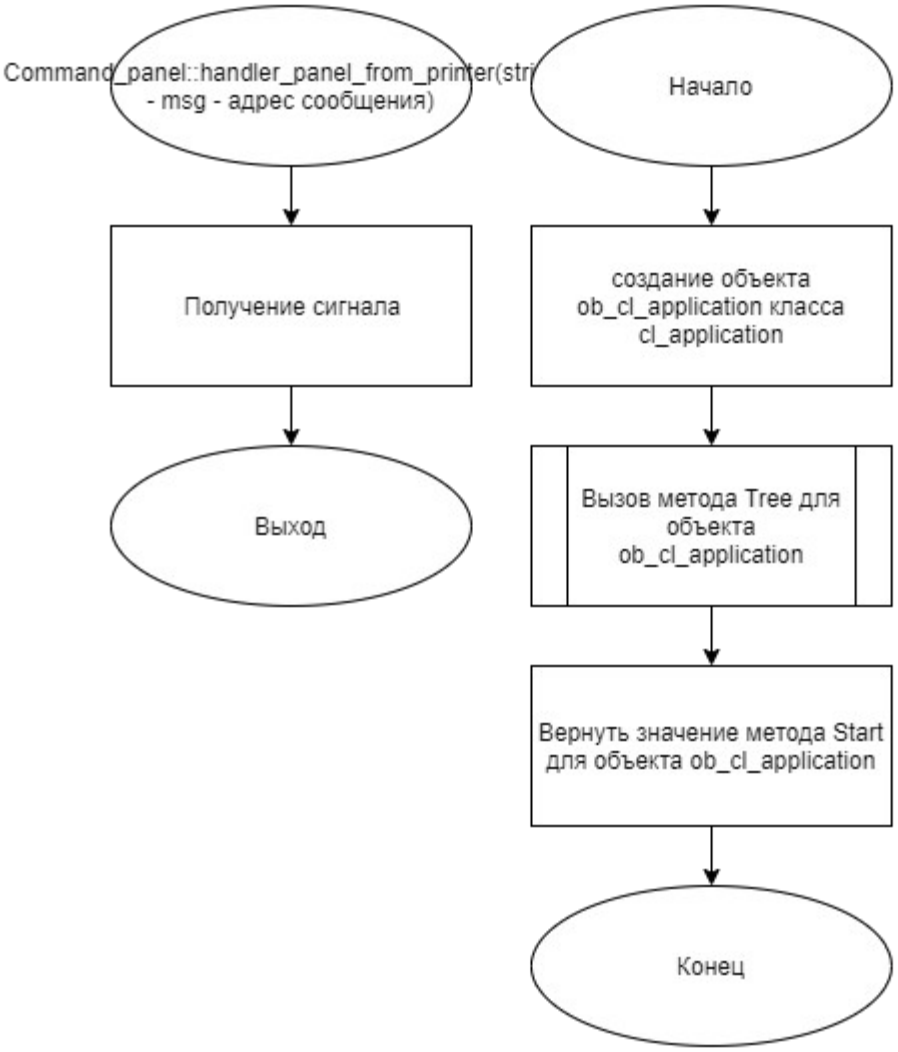

Рисунок 38 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_application.cpp

Листинг 1 – cl_application.cpp

```
#include "cl_application.h"
#include <vector>
#include <string>
cl_application::cl_application(cl_base*
p_head_object):cl_base(p_head_object){}
void cl_application::Tree(){

   //Дерево
   cl_base* sub = this;
   Change_name("System");

   //Создание объектов
   sub=new printer(this, "3D-printer");
   sub=new Command_panel(this, "Command_panel");
   sub=new coils(this, "Coils","Coils",0,0);
   sub=new PC(this, "Computer");
   sub=new Display(this, "Display");
   sub=new Details(this, "Details");
   cl_base* p_root=get_root();

   //Установка всех готовностей на 1
   p_root->Set_all_ready();
   ((printer*) Get_ptr("3D-printer"))->Set_state(5);

   //Связи
   this-
>set_connection(SIGNAL_D(cl_application::signal_app_to_panel),Get_ptr("Comma
nd_panel"), HANDLER_D(Command_panel::handler_panel_from_app));
   Get_ptr("Command_panel")-
>set_connection(SIGNAL_D(Command_panel::signal_panel_to_PC),Get_ptr("Compute
r"), HANDLER_D(PC::handler_PC_from_panel));
   Get_ptr("Command_panel")-
>set_connection(SIGNAL_D(Command_panel::signal_panel_to_printer),Get_ptr("3D
-printer"), HANDLER_D(printer::handler_printer_from_panel));
   Get_ptr("Computer")-
>set_connection(SIGNAL_D(PC::signal_PC_to_printer),Get_ptr("3D-printer"),
HANDLER_D(printer::handler_printer_from_PC));
   Get_ptr("3D-printer")-
>set_connection(SIGNAL_D(printer::signal_printer_to_PC),Get_ptr("Computer"),
```

```
HANDLER_D(PC::handler_PC_from_printer));
   Get_ptr("Command_panel")-
>set_connection(SIGNAL_D(Command_panel::signal_panel_to_display),Get_ptr("Di
splay"), HANDLER_D(Display::handler_display_from_panel));
   Get_ptr("3D-printer")-
>set_connection(SIGNAL_D(printer::signal_printer_to_app),this,
HANDLER_D(cl_application::handler_app_from_printer));

   //Начальные данные
   string message= "";
   this->emit_signal(SIGNAL_D(cl_application::signal_app_to_panel), message);
   ((printer*) Get_ptr("3D-printer"))->Set_state(1);
}

void cl_application::signal_app_to_panel(string &msg){}

int cl_application::Start(){
   int i_state=tact;
   int count = 0;
   string head,com;
   cl_base* p_root=get_root();
   while (true){
      i_state=tact;
      getline(cin,com);

      if (com=="Turn off the system") {
         ((printer*) Find_global("3D-printer"))->Set_state(0);
         Find_global("Command_panel")-
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_display),"Turn  off  the
system");
         return 0;}
      if (com.substr(0,2)=="PC"){

         if(com.substr(3,9)=="condition"){
            Find_global("Command_panel")-
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_PC),"condition");
         }
         else {
            int ind1,ind2,ind3;
            ind1=com.find(' ',0);
            ind2=com.find(' ',ind1+1);
            ind3=com.find(' ',ind2+1);
            string id = com.substr(ind2+1, ind3-ind2-1);
            string name = com.substr(ind3+1);
            int volume = stoi(com.substr(ind1+1, ind2-ind1-1));
            if(Find_global(name)){
               Find_global("Command_panel")-
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_display),"Failed     to
create product");
            }
            else{
               Details* d = new Details(Get_ptr("Details"),name);
               d->name = name;
               d->id = id;
               d->Volume = volume;
               Find_global("Command_panel")-
```

```
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_PC),name);
              if (((printer*)Find_global("3D-printer"))->state == 1){
                 Find_global("Computer")-
>emit_signal(SIGNAL_D(PC::signal_PC_to_printer),
((PC*)Find_global("Computer"))->Q_Details[0]->name);
              }
           }
        }
     }
     if (com.substr(0,23)=="Filament coil condition"){
        coils* k = ((coils*) Find_global(com.substr(24)));
        if(k){
           cout<<"Filament coil condition: "<<k->id<<" "<<k->still_l<<endl;
        }
     }
     if (com == "SHOWTREE"){
        Find_global("Command_panel")-
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_display),"SHOWTREE");
     }
     if (com=="System status"){
        cout<<"3D printer tact: "<<tact<<"; temp: "<<((printer*)Get_ptr("3D-
printer"))->Celsium<<"; status: "<<((printer*)Get_ptr("3D-printer"))->state;
        cout<<"; print product: ";
        if(((printer*)Find_global("3D-printer"))->name_of_detail   ==""   ||
((printer*)Find_global("3D-printer"))->len==-1 ){cout<<0;}
        else        cout<<((Details*)Find_global(((printer*)Find_global("3D-
printer"))->name_of_detail))->Volume      -      ((printer*)Find_global("3D-
printer"))->len;
        cout<<"; queue products: ";
        Find_global("Command_panel")-
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_PC),"queue");
        cout<<"; PC: ";
        Find_global("Command_panel")-
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_PC),"PC");
        cout<<endl;
     }
     if (i_state==tact && ((printer*)Find_global("3D-printer"))->state !=1 )
{
        string name;
        name = ((printer*)Find_global("3D-printer"))->name_of_detail;

        if(((printer*)Find_global("3D-printer"))->state == 2){
           Find_global("Computer")-
>emit_signal(SIGNAL_D(PC::signal_PC_to_printer),name);
        }

        if (((printer*)Find_global("3D-printer"))->state == 3){
           Find_global("Computer")-
>emit_signal(SIGNAL_D(PC::signal_PC_to_printer),name);
        }

        if(((printer*)Find_global("3D-printer"))->state == 4){
           count++;
           if(count == 3){
              name = ((printer*)Find_global("3D-printer"))->name_of_detail;
              coils* k = ((coils*)Find_global(((Details*)Find_global(name))-
```

```
>id));
                k->still_l=k->lenght;
                count=0;
                ((printer*)Find_global("3D-printer"))->state = 2;
             }
             tact++;
          }

       }
       if (i_state==tact && ((printer*)Find_global("3D-printer"))->state == 1
&& ((PC*)Find_global("Computer"))->Q_Details.size()!=0){
          string name;
          name = ((PC*)Find_global("Computer"))->Q_Details[0]->name;
          Find_global("Computer")-
>emit_signal(SIGNAL_D(PC::signal_PC_to_printer),name);
       }
       if (i_state == tact) tact++;
    }
    return 0;
}
void cl_application::handler_app_from_printer(string& msg){
    if (msg=="tact"){
       tact++;
    }
}
```

5.2 Файл cl_application.h

Листинг 2 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
#include "printer.h"
#include "Command_panel.h"
#include "coils.h"
#include "Details.h"
#include "Display.h"
#include "PC.h"

class cl_application:public cl_base{
   public:

   int tact=1;

   cl_application(cl_base* p_head_object);
   void Tree();
   int Start();
   void signal_app_to_panel(string &msg);
   void handler_app_from_printer(string &msg);
```

```
};
#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"
#include <stack>
using namespace std;
cl_base::cl_base(cl_base* p_head_object, string s_object_name){
   this -> s_object_name = s_object_name;
   this -> p_head_object = p_head_object;
   if (p_head_object!=nullptr){
      p_head_object->subordinate_objects.push_back(this);
   }
}

//Получение имени объекта
string cl_base::Get_name(){
   return s_object_name;
}


//Поменять имя объекта
bool cl_base::Change_name(string name){
   if(Get_baseptr()!=nullptr){
      for (int i=0; i<p_head_object->subordinate_objects.size();i++){
         if (p_head_object->subordinate_objects[i]->Get_name()==name){
            return false;
         }
      }
   }
   this->s_object_name=name;
   return true;
}

//Получение указателя на объект
cl_base* cl_base::Get_ptr(string s_object_name){
   for (int i=0; i<subordinate_objects.size();i++){
      if (subordinate_objects[i]->Get_name()==s_object_name){
         return subordinate_objects[i];
      }
   }
   return nullptr;
}

//Получение головного указателя
cl_base* cl_base::Get_baseptr(){
   return p_head_object;
```

```
}


//Статус
void cl_base::can_off(int status1){
   if (status1==0){
      status=0;
      for (auto sub:subordinate_objects){
         sub->can_off(0);
      }
   }
   if (Get_baseptr()&&(Get_baseptr()->status==0)){
      return;
   }
   else if (status1!=0){
      status=1;
   }
}

void cl_base::Set_all_ready(){
   this->can_off(1);
   for (auto sub:this->subordinate_objects){
      sub->Set_all_ready();
   }
}
//Вывод дерева
void cl_base::Out_fc(int probel){
   cout<<Get_name();
   if(subordinate_objects.size()!=0){
      for (auto sub:subordinate_objects){
         cout<<endl;
         for(int i =0; i<probel;i++) cout<<" ";
         sub->Out_fc(probel+4);
      }
   }
}

//Вывод дерева с готовностью
void cl_base::Out(int probel){
   cout<< Get_name();
   if (status==0)cout<<" is not ready";
   else if (status!=0){cout<<" is ready";}
   if (subordinate_objects.size()!=0){
      for(int i=0; i<subordinate_objects.size();i++){
         cout<<endl;
         for(int i =0; i<probel;i++) cout<<" ";
         subordinate_objects[i]->Out(probel+4);
      }
   }
}

//Поиск от текущего элемента
cl_base* cl_base::Find_current(string name){
   if (Get_name() == name)
   {
```

```
            return this;
         }
         for (auto sub : subordinate_objects)
         {
            cl_base* p_sub = sub->Find_current(name);
            if (p_sub)
            {
               return p_sub;
            }
         }
         return nullptr;
      }


      //Поиск на всем дереве
      cl_base* cl_base::Find_global(string name){
         cl_base* p_found = get_root();
         int count =0;
         stack <cl_base*> stack;
         stack.push(p_found);
         while(!stack.empty()){
            cl_base * cur = stack.top();
            stack.pop();
            if (cur->Get_name()==name) count++;
            for (auto sub:cur->subordinate_objects) stack.push(sub);
         }
         if (count!=1) return nullptr;
         return p_found->Find_current(name);
      }

      //Метод нахождения корня
      cl_base* cl_base::get_root(){
         cl_base* p_root=this;
         while(p_root->Get_baseptr()){
            p_root=p_root->Get_baseptr();
         }
         return p_root;
      }

      //Метод поиска объекта по координате
      cl_base* cl_base::find_obj_bc(string coord){
         string s_name;
         int index;
         cl_base* ob = nullptr;
         cl_base* p_root=get_root();
         //головной объект
         if(coord=="/"){
            return p_root;
         }
         //текущий
         if(coord=="."){
            return this;
         }
         //Поиск от корня по имени
         if(coord[0]=='/' && coord[1]=='/'){
```

```
            s_name=coord.substr(2);
            return this->Find_global(s_name);
         }
         //Поиск от текущего по имени
         if(coord[0]=='.'){
            s_name=coord.substr(1);
            return this->Find_current(s_name);
         }
         index=coord.find("/",1);
         //Абсолютный путь
         if(coord[0]=='/'){
            if (index!=-1){
               s_name= coord.substr(1,index-1);
               ob=p_root->Get_ptr(s_name);
               if(ob!=nullptr){
                  return ob->find_obj_bc(coord.substr(index+1));
               }
               else return ob;
            }
            else {
               s_name= coord.substr(1);
               return p_root->Get_ptr(s_name);
            }
         }
         //Относительный путь
         if(coord[0]!='/'){
            if (index!=-1){
               s_name= coord.substr(0,index);
               ob=this->Get_ptr(s_name);
               if(ob!=nullptr){
                  return ob->find_obj_bc(coord.substr(index+1));
               }
               else return ob;
            }
            else{
               s_name=coord;
               return this->Get_ptr(s_name);
            }
         }
         return nullptr;
      }
      //Метод переопределения головного объекта
      bool cl_base::Move_head(cl_base* h){
         if (this->Get_baseptr()){
            for (int i=0;i<this->Get_baseptr()->subordinate_objects.size();i++){
               if(this->Get_baseptr()->subordinate_objects[i]==this){
                  Get_baseptr()->subordinate_objects.erase(this->Get_baseptr()-
      >subordinate_objects.begin()+i);
                  break;
               }
            }
            this->p_head_object=h;
            h->subordinate_objects.push_back(this);
            cout<<endl<<"New head object: "<<h->Get_name();
            return true;
```

```
   }
   return false;
}
//Метод удаления подчиненного объекта
void cl_base::Del_obj(string name){
   cl_base* p = Get_ptr(name);
   int i = 0;
   for(auto sub:subordinate_objects){
      if (sub==p){
         subordinate_objects.erase(subordinate_objects.begin()+i);
         delete p;
         return;
      }
      i++;
   }
}
//Абсолютный путь до объекта
string cl_base::Absolute(){
   cl_base* p_root =get_root();
   string coord="";
   if (p_root == this) return "/";
   vector <string> text;
   string text1;
   coord+=this->Get_name();
   cl_base* h = this->Get_baseptr();
   while(h){
      if(h->Get_name()!=p_root->Get_name()){
         text.push_back(h->Get_name());

      }
      h=h->Get_baseptr();
   }
   for (int i = text.size()-1;i>=0;i--) text1+="/"+text[i];
   if(coord!=p_root->Get_name()) text1+="/"+coord;
   return text1;
}
//Метод создания связи
void    cl_base::set_connection(TYPE_SIGNAL    signal,    cl_base*    target,
TYPE_HANDLER handler){
   o_sh  * p_value;
   //-----------------------------------------------------------
   // Цикл для исключения повторного установления связи
   for (int i = 0; i < connects.size ( );i++)
   {
      if (connects [ i ] -> signal  == signal &&
         connects [ i ] -> target == target &&
         connects [ i ] -> handler == handler )return;
   }
   p_value  =    new  o_sh  ();//  создание  объекта  структуры  для  хранения
информации о новой связи
   p_value -> signal  = signal;
   p_value -> target = target;
   p_value -> handler = handler;
   connects.push_back (p_value);// добавление новой связи
}
```

```
//Метод удаления связи
void   cl_base::delete_connection(TYPE_SIGNAL   signal,   cl_base*   target,
TYPE_HANDLER handler){
   vector <o_sh*>::iterator it;
   for (it = connects.begin(); it<=connects.end(); it++){
      for (int i = 0; i < connects.size ( );i++){
         if ((*it)-> signal  == signal &&
            (*it)-> target == target &&
            (*it)-> handler == handler )
         {

            delete *it;
            it = connects.erase(it);
            it--;
         }
      }
   }
}
//Метод подачи сигнала
void cl_base::emit_signal(TYPE_SIGNAL signal, string msg){
   if (this->status==0) return;
   (this ->*signal)(msg);
   for (auto con:connects){
      if(con->signal==signal){
         cl_base* target=con->target;
         if (target->status!=0){
            TYPE_HANDLER handler=con->handler;
            (target->*handler)(msg);
         }
      }
   }

}

//Удаление связей
void cl_base::delete_links(cl_base* target){
   for (int i =0; i<this->connects.size();i++){
      if(this->connects[i]->target == target){
         delete this->connects[i];
         this->connects.erase(connects.begin()+i);
         i--;
      }

   }

   for (auto sub:subordinate_objects){
      sub->delete_links(target);
   }
}


cl_base::~cl_base(){
   get_root()->delete_links(this);
   for (int i=0; i<subordinate_objects.size();i++){
      delete subordinate_objects[i];
```

```
            subordinate_objects.erase(subordinate_objects.begin()+i);
        }
    }
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
    #ifndef __CL_BASE__H
    #define __CL_BASE__H
    #include <iostream>
    #include <string>
    #include <vector>
    #include <queue>
    class cl_base;
    using namespace std;
    #define SIGNAL_D(signal_f)(TYPE_SIGNAL)(&signal_f)
    #define HANDLER_D(handler_f)(TYPE_HANDLER)(&handler_f)
    typedef void ( cl_base :: * TYPE_SIGNAL ) ( string & msg);
    typedef void ( cl_base :: * TYPE_HANDLER ) ( string );
    struct o_sh                 // Структура задания одной связи
    {
       TYPE_SIGNAL  signal;    // Указатель на метод сигнала
       cl_base* target;   // Указатель на целевой объект
       TYPE_HANDLER handler;   // Указатель на метод обработчика

    };

    class cl_base{
       private:
       int status=0;
       string s_object_name;
       cl_base * p_head_object;
       vector <cl_base*> subordinate_objects;
       vector <o_sh*> connects;
       public:
       cl_base(cl_base* p_head_object,string s_object_name="Base_object");
       bool Change_name(string name);
       string Get_name();
       cl_base* Get_baseptr();

       void Out(int probel=4);
       void Out_fc(int probel=4);

       void can_off(int status);

       cl_base* get_root();

       bool Move_head(cl_base*);
       void Del_obj(string name);
```

```
   string Absolute();


   cl_base* Find_current(string name);
   cl_base* Find_global(string name);
   cl_base* Get_ptr(string s_object_name);
   cl_base* find_obj_bc(string coord);


   void Set_all_ready();

   void  set_connection(TYPE_SIGNAL  signal,  cl_base*  target,  TYPE_HANDLER
handler);
   void delete_connection(TYPE_SIGNAL signal, cl_base* target, TYPE_HANDLER
handler);
   void emit_signal(TYPE_SIGNAL signal, string msg);
   ~cl_base();

   //Удаление связей
   void delete_links(cl_base* target);
};
#endif
```

5.5 Файл coils.cpp

Листинг 5 – coils.cpp

```
#include "coils.h"
coils::coils(cl_base* p_head, string s_object_name, string id, int l, int
t):cl_base(p_head, s_object_name){
   this->id=id;
   this->lenght=t;
   this->p_temp=l;
   this->still_l=t;
}
```

5.6 Файл coils.h

Листинг 6 – coils.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include "cl_base.h"
class coils:public cl_base{
```

```
   public:

   int p_temp;
   int lenght;
   int still_l;
   string id;

   coils(cl_base* p_head, string s_object_name, string id, int l, int t);
};
#endif
```

5.7 Файл Command.cpp

Листинг 7 – Command.cpp

```
#include "Command_panel.h"
Command_panel::Command_panel(cl_base*             p_head,             string
s_object_name):cl_base(p_head, s_object_name){
}

void Command_panel::handler_panel_from_app(string& msg){
   int n;
   cin>>n;
   this-
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_printer),"Write");
   for (int i =0; i<n; i++){
      this-
>emit_signal(SIGNAL_D(Command_panel::signal_panel_to_printer),"Coils");
   }
   string end;
   getline(cin,end);
   getline(cin,end);
   if (end=="End of settings") cout<<"Ready to work\n";
}
void Command_panel::handler_panel_from_printer(string& msg){
}
void Command_panel::signal_panel_to_PC(string& msg){
}
void Command_panel::signal_panel_to_printer(string& msg){
}
void Command_panel::signal_panel_to_display(string& msg){
}
```

5.8 Файл Command_panel.h

Листинг 8 – Command_panel.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"

class Command_panel:public cl_base{
   public:
   Command_panel(cl_base* p_head, string s_object_name);
   void signal_panel_to_PC(string& msg);
   void signal_panel_to_printer(string& msg);
   void handler_panel_from_app(string &msg);

   void handler_panel_from_printer(string &msg);
   void signal_panel_to_display(string& msg);
};


#endif
```

5.9 Файл Details.cpp

Листинг 9 – Details.cpp

```
#include "Details.h"
Details::Details(cl_base*   p_head,   string   s_object_name):cl_base(p_head,
s_object_name){
}
```

5.10 Файл Details.h

Листинг 10 – Details.h

```
#ifndef __DETAILS__H
#define __DETAILS__H
#include "cl_base.h"
class Details:public cl_base{
   public:

   string name;
   string id;
```

```
   int Volume;

   Details(cl_base* p_head, string s_object_name);
};


#endif
```

5.11 Файл Display.cpp

Листинг 11 – Display.cpp

```
#include "Display.h"
Display::Display(cl_base*   p_head,   string   s_object_name):cl_base(p_head,
s_object_name){
}

void Display::handler_display_from_panel(string& msg){
   if (msg=="SHOWTREE"){
      get_root()->Out_fc();
      cout<<endl;
   }
   else {cout<<msg<<endl;}
}
```

5.12 Файл Display.h

Листинг 12 – Display.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "cl_base.h"
class Display:public cl_base{
   public:
   Display(cl_base* p_head, string s_object_name);
   void handler_display_from_panel(string& msg);
};


#endif
```

5.13 Файл main.cpp

Листинг 13 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "cl_base.h"
#include "cl_application.h"
int main()
{
   cl_application  ob_cl_application ( nullptr );
   ob_cl_application.Tree();
   return ob_cl_application.Start();
}
```

5.14 Файл PC.cpp

Листинг 14 – PC.cpp

```
#include "PC.h"
PC::PC(cl_base* p_head, string s_object_name):cl_base(p_head, s_object_name)
{
}

void PC::signal_PC_to_printer(string& msg){
}
void PC::handler_PC_from_panel(string& msg){
   if (msg=="condition"){
      if(Q_Details.size()!=0){

         cout<<"PC condition turned on";
         string name = ((printer*)Find_global("3D-printer"))->name_of_detail;
         if(name!=""){
            cout<<" "<<name<<";";
         }

         for (int i=0; i<Q_Details.size();i++){
            cout<<" "<<Q_Details[i]->name<<";";
         }
         cout<<endl;
      }
      else cout<<"PC condition turned off"<<endl;
   }
   else if (msg=="queue"){
      int count=0;
      if (((printer*)Find_global("3D-printer"))->name_of_detail!=""){
         count++;
      }
```

```
            cout<<count+Q_Details.size();
         }
         else if (msg=="PC"){
            string name = ((printer*)Find_global("3D-printer"))->name_of_detail;
            if (name!=""){
               cout<<"("<<name<<":"<<((Details*)Find_global(name))->Volume<<") ";
            }
            if(Q_Details.size()!=0){
               for (int i=0; i<Q_Details.size();i++){

                  cout<<"("<<Q_Details[i]->name<<":"<<Q_Details[i]->Volume<<") ";
               }
            }
         }
         else{
            Q_Details.push_back(((Details*)Find_global(msg)));
         }
      }
      void PC::handler_PC_from_printer(string& msg){
         if (msg=="Delete"){
            Q_Details.erase(Q_Details.begin());
         }
      }
```

5.15 Файл PC.h

Листинг 15 – PC.h

```
      #ifndef __CL_5__H
      #define __CL_5__H
      #include <vector>
      #include "cl_base.h"
      #include "Details.h"
      #include "printer.h"
      class PC:public cl_base{
         public:

         vector <Details*> Q_Details;

         PC(cl_base* p_head, string s_object_name);
         void signal_PC_to_printer(string& msg);
         void handler_PC_from_panel(string& msg);
         void handler_PC_from_printer(string& msg);
      };
      #endif
```

5.16 Файл printer.cpp

Листинг 16 – printer.cpp

```
#include "printer.h"
printer::printer(cl_base*   p_head,   string   s_object_name):cl_base(p_head,
s_object_name){
}

void printer::handler_printer_from_panel(string& msg){
   if (msg=="Write"){
      cin>>speed_of_print;
      cin>>speed_of_temp;
      cin>>Celsium;
   }
   else if (msg=="Coils"){
      int a, b;
      string id;
      cin>>id>>a>>b;
      if (this->Find_global(id)){
         cout<<"Failed to create filament coil\n";
         return;}
      coils* k= new coils(Find_global("Coils"),id, id, a, b);
      All_coils.push_back(k);
   }
}
void printer::handler_printer_from_PC(string& msg){
   Details* det = ((Details*)Find_global(msg));
   coils* k = ((coils*)Find_global(det->id));
   if (state==1){
      name_of_detail=msg;
      Find_global("3D-printer")-
>emit_signal(SIGNAL_D(printer::signal_printer_to_PC),"Delete");

      if (Celsium == k->p_temp) {
         state = 2;
         Find_global("3D-printer")-
>emit_signal(SIGNAL_D(printer::signal_printer_to_app),"tact");}
      else state = 3;
   }
   else if (state==2){
      if (len==-1){
         len =0;
         return;
      }
      det = ((Details*)Find_global(name_of_detail));
      k = ((coils*)Find_global(det->id));
      if (len!= det->Volume){
         if (k->still_l!=0){
            if (det->Volume - len <= speed_of_print){
               if(det->Volume - len <= k->still_l){
                  k->still_l -= (det->Volume - len);
                  len = det->Volume;
               }
```

```
                else {
                   len+=k->still_l;
                   k->still_l=0;
                   state = 4;
                }
             }
             else if (speed_of_print<=k->still_l){
                len+=speed_of_print;
                k->still_l-=speed_of_print;
                Find_global("3D-printer")-
>emit_signal(SIGNAL_D(printer::signal_printer_to_app),"tact");
             }
             else if (speed_of_print>k->still_l){
                len+=k->still_l;
                k->still_l=0;
                state=4;
             }
          }
          else{
             state = 4;
          }
       }

       if (len == det->Volume){
          len=0;
          name_of_detail = "";
          state = 1;
          Find_global("3D-printer")-
>emit_signal(SIGNAL_D(printer::signal_printer_to_app),"tact");
          return;
       }
    }

    if (state==3){
       det = ((Details*)Find_global(name_of_detail));
       k = ((coils*)Find_global(det->id));
       if (Celsium > k->p_temp){
          if (Celsium - speed_of_temp <= k->p_temp){
             Celsium = k->p_temp;
          }
          else { Celsium -= speed_of_temp;}
       }
       else if(Celsium < k->p_temp){
          if (Celsium + speed_of_temp >= k->p_temp){
             Celsium = k->p_temp;
          }
          else { Celsium += speed_of_temp;}
       }
       if(Celsium == k->p_temp) {
          state = 2;
          len = -1;
       }
       Find_global("3D-printer")-
>emit_signal(SIGNAL_D(printer::signal_printer_to_app),"tact");
    }
```

```
}
void printer::Set_state(int i){
   state=i;
}
void printer::signal_printer_to_PC(string& msg){}
void printer::signal_printer_to_app(string& msg){}
void printer::signal_printer_to_panel(string& msg){}
void printer::signal_printer_to_display(string& msg){}
```

5.17 Файл printer.h

Листинг 17 – printer.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"
#include "Details.h"
#include "coils.h"
#include "PC.h"
class printer:public cl_base{
   public:
   int speed_of_print;
   int speed_of_temp;
   int Celsium;
   int state=0;
   int len = 0;

   vector <coils*> All_coils;
   string name_of_detail= "";

   void Set_state(int i);
   printer(cl_base* p_head, string s_object_name);
   void signal_printer_to_PC(string& msg);
   void signal_printer_to_app(string& msg);
   void signal_printer_to_panel(string& msg);
   void signal_printer_to_display(string& msg);
   void handler_printer_from_panel(string& msg);
   void handler_printer_from_PC(string &msg);
};
#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 51.

Таблица 51 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
1 5 100 0 abs 200 50 End of settings PC 10 abs Product 2 System status Turn off the system	Ready to work 3D printer tact: 4; temp: 200; status: 2; print product: 10; queue products: 1; PC: (Product 2:10) Turn off the system	Ready to work 3D printer tact: 4; temp: 200; status: 2; print product: 10; queue products: 1; PC: (Product 2:10) Turn off the system
3 5 100 0 abs 200 50 pla 160 100 petg 190 5 End of settings System status PC 5 pla Product 1 PC 10 abs Product 2 PC condition Filament coil condition pla PC 10 petg Product 3 System status PC condition Filament coil condition petg PC condition System status Turn off the system	Ready to work 3D printer tact: 1; temp: 0; status: 1; print product: 0; queue products: 0; PC: PC condition turned on Product 1; Product 2; Filament coil condition: pla 100 3D printer tact: 7; temp: 200; status: 2; print product: 0; queue products: 2; PC: (Product 2:10) (Product 3:10) PC condition turned on Product 2; Product 3; Filament coil condition: petg 0 PC condition turned off 3D printer tact: 17; temp: 190; status: 1; print product: 0; queue products: 0; PC: Turn off the system	Ready to work 3D printer tact: 1; temp: 0; status: 1; print product: 0; queue products: 0; PC: PC condition turned on Product 1; Product 2; Filament coil condition: pla 100 3D printer tact: 7; temp: 200; status: 2; print product: 0; queue products: 2; PC: (Product 2:10) (Product 3:10) PC condition turned on Product 2; Product 3; Filament coil condition: petg 0 PC condition turned off 3D printer tact: 17; temp: 190; status: 1; print product: 0; queue products: 0; PC: Turn off the system
2 12 500 0 1 500 11 2 114 234 End of settings	Ready to work 3D printer tact: 4; temp: 500; status: 4; print product: 1; queue products: 2;	Ready to work 3D printer tact: 4; temp: 500; status: 4; print product: 1; queue products: 2;

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
PC 12 1 Detalka PC 29 2 NeDetalka System status Turn off the system	PC: (Detalka:12) (NeDetalka:29) Turn off the system	PC: (Detalka:12) (NeDetalka:29) Turn off the system

ЗАКЛЮЧЕНИЕ

В процессе выполнения поставленных целей курсовой работы были преобретены знания и практические навыки в области объектно-ориентированного программирования. Освоение основных концепций ООП, таких как работа с древовидной иерархией объектов, описание базового класса, а также реализация сигналов и их обработчиков, позволило понять структуры данных, архитектуру программ и методологию проектирования. Эти навыки являются ключевыми для успешной разработки надежного и эффективного программного обеспечения.

Выполнение данной работы дало возможность не только ознакомиться с теоретическими аспектами программирования, но и применить их на практике, разработав алгоритм моделирования 3D-принтера.

По завершении работы были выполнены все поставленные постановкой курсовой работы цели и задачи.

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).