Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....7
1 ПОСТАНОВКА ЗАДАЧИ.....8
1.1 Описание входных данных.....10
1.2 Описание выходных данных.....12
2 МЕТОД РЕШЕНИЯ.....14
3 ОПИСАНИЕ АЛГОРИТМОВ.....19
3.1 Алгоритм метода set_connection класса cl_base.....19
3.2 Алгоритм метода delete_connection класса cl_base.....20
3.3 Алгоритм метода emit_signal класса cl_base.....21
3.4 Алгоритм метода get_status класса cl_base.....22
3.5 Алгоритм метода get_abs_path класса cl_base.....22
3.6 Алгоритм метода delete_links класса cl_base.....23
3.7 Алгоритм метода set_status_tree класса cl_base.....24
3.8 Алгоритм метода signal_f класса cl_application.....25
3.9 Алгоритм метода handler_f класса cl_application.....25
3.10 Алгоритм метода get_class_num класса cl_application.....26
3.11 Алгоритм метода signal_f класса cl_2.....26
3.12 Алгоритм метода handler_f класса cl_2.....26
3.13 Алгоритм метода get_class_num класса cl_2.....27
3.14 Алгоритм метода signal_f класса cl_3.....27
3.15 Алгоритм метода handler_f класса cl_3.....28
3.16 Алгоритм метода get_class_num класса cl_3.....28
3.17 Алгоритм метода signal_f класса cl_4.....28
3.18 Алгоритм метода handler_f класса cl_4.....29
3.19 Алгоритм метода get_class_num класса cl_4.....29
3.20 Алгоритм метода signal_f класса cl_5.....30

3.21 Алгоритм метода handler_f класса cl_5.....30
3.22 Алгоритм метода get_class_num класса cl_5.....30
3.23 Алгоритм метода signal_f класса cl_6.....31
3.24 Алгоритм метода handler_f класса cl_6.....31
3.25 Алгоритм метода get_class_num класса cl_6.....32
3.26 Алгоритм метода build_tree_objects класса cl_application.....32
3.27 Алгоритм метода exec_app класса cl_application.....35
3.28 Алгоритм функции main.....37
3.29 Алгоритм конструктора класса cl_base.....38
3.30 Алгоритм деструктора класса cl_base.....38
3.31 Алгоритм метода get_name класса cl_base.....39
3.32 Алгоритм метода get_p_head класса cl_base.....39
3.33 Алгоритм метода set_name класса cl_base.....40
3.34 Алгоритм метода get_subordinate_object класса cl_base.....41
3.35 Алгоритм метода print_tree класса cl_base.....41
3.36 Алгоритм метода search_object_from_current класса cl_base.....42
3.37 Алгоритм метода search_object_from_tree класса cl_base.....43
3.38 Алгоритм метода set_status класса cl_base.....44
3.39 Алгоритм метода find_obj_by_coord класса cl_base.....45
3.40 Алгоритм конструктора класса cl_application.....47
3.41 Алгоритм конструктора класса cl_2.....47
3.42 Алгоритм конструктора класса cl_3.....47
3.43 Алгоритм конструктора класса cl_4.....48
3.44 Алгоритм конструктора класса cl_5.....48
3.45 Алгоритм конструктора класса cl_6.....49
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....50
5 КОД ПРОГРАММЫ.....86

5.1 Файл cl_2.cpp.....86
5.2 Файл cl_2.h.....86
5.3 Файл cl_3.cpp.....87
5.4 Файл cl_3.h.....87
5.5 Файл cl_4.cpp.....88
5.6 Файл cl_4.h.....88
5.7 Файл cl_5.cpp.....89
5.8 Файл cl_5.h.....89
5.9 Файл cl_6.cpp.....90
5.10 Файл cl_6.h.....90
5.11 Файл cl_application.cpp.....91
5.12 Файл cl_application.h.....95
5.13 Файл cl_base.cpp.....95
5.14 Файл cl_base.h.....102
5.15 Файл main.cpp.....103
6 ТЕСТИРОВАНИЕ.....104
ЗАКЛЮЧЕНИЕ.....107
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....108

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) - является парадигм разработки. Парадигмой называют набор правил и критериев. Методология ООП основана на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования. Объектно-ориентированное программирование актуально на данный момент, так как является востребованной парадигмой программирования при разработки программного обеспечения.

Цель работы: получить практические навыки разработки на языке программирования C++, создать программное обеспечение с функционалом проектирования дерева и управлением объектами. При получении навыков разработки на C++ нужно усвоить материал основы работы с классами.

Поставленной задачей является построение дерева на языке C++ и внедрение функционала для пользователя по управлению деревом. Для выполнения поставленной задачи с деревом нужно изучить теоритическую часть создания иерархии. Для создания управления понадобится знание в правильной аллокации памяти и реализация взаимодействия объектов по средствам сигналов и обработчиков.

1 ПОСТАНОВКА ЗАДАЧИ

Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков, с передачей вместе сигналом текстового сообщения (строковой переменной).

Для организации взаимосвязи по механизму сигналов и обработчиков в базовый класс добавить три метода:

- установления связи между сигналом текущего объекта и обработчиком целевого объекта;
- удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
- выдачи сигнала от текущего объекта с передачей строковой переменной.

Включенный объект может выдать или обработать сигнал.

Методу установки связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу удаления (разрыва) связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу выдачи сигнала передать указатель на метод сигнала и строковую переменную. В данном методе реализовать алгоритм:

1. Если текущий объект отключен, то выход, иначе к пункту 2.
2. Вызов метода сигнала с передачей строковой переменной по ссылке.
3. Цикл по всем связям сигнал-обработчик текущего объекта:

    3.1. Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то проверить готовность целевого объекта. Если целевой объект готов, то вызвать метод обработчика

целевого объекта указанной в связи и передать в качестве аргумента строковую переменную по значению.

4. Конец цикла.

Для приведения указателя на метод сигнала и на метод обработчика использовать параметризированное макроопределение препроцессора.

В базовый класс добавить метод определения абсолютной пути до текущего объекта. Этот метод возвращает абсолютный путь текущего объекта.

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 3 курсовой работы. Если при построении дерева иерархии возникает ситуация дубляжа имен среди починенных у текущего головного объекта, то новый объект не создается.

Система содержит объекты шести классов с номерами: 1, 2, 3, 4, 5, 6. Классу корневого объекта соответствует номер 1. В каждом производном классе реализовать один метод сигнала и один метод обработчика.

Каждый метод сигнала с новой строки выводит:

```
Signal from «абсолютная координата объекта»
```

Каждый метод сигнала добавляет переданной по параметру строке текста номер класса принадлежности текущего объекта по форме:

```
«пробел»(class: «номер класса»)
```

Каждый метод обработчика с новой строки выводит:

```
Signal to «абсолютная координата объекта»   Text: «переданная строка»
```

Моделировать работу системы, которая выполняет следующие команды с параметрами:

- EMIT «координата объекта» «текст» – выдает сигнал от заданного по координате объекта;
- SET_CONNECT «координата объекта выдающего сигнал» «координата

целевого объекта» – устанавливает связь;

- DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаляет связь;
- SET_CONDITION «координата объекта» «значение состояния» – устанавливает состояние объекта.
- END – завершает функционирование системы (выполнение программы).

Реализовать алгоритм работы системы:

- в методе построения системы:
  - построение дерева иерархии объектов согласно вводу;
  - ввод и построение множества связей сигнал-обработчик для заданных пар объектов.
- в методе отработки системы:
  - привести все объекты в состоянии готовности;
  - цикл до признака завершения ввода:
    - ввод наименования объекта и текста сообщения;
    - вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной, содержащей текст сообщения.
  - конец цикла.

Допускаем, что все входные данные вводятся синтаксически корректно. Контроль корректности входных данных можно реализовать для самоконтроля работы программы. Не оговоренные, но необходимые функции и элементы классов добавляются разработчиком.

1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве

иерархии. Структура данных для ввода согласно изложенному в версии № 3 курсовой работы.

После ввода состава дерева иерархии построчно вводится:

```
«координата объекта выдающего сигнал» «координата целевого объекта»
```

Ввод информации для построения связей завершается строкой, которая содержит:

```
«end_of_connections»
```

В методе запуска (отработки) системы построчно вводятся множество команд в производном порядке:

- EMIT «координата объекта» «текст» – выдать сигнал от заданного по координате объекта;
- SET_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – установка связи;
- DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаление связи;
- SET_CONDITION «координата объекта» «значение состояния» – установка состояния объекта.
- END – завершить функционирование системы (выполнение программы).

Команда END присутствует обязательно.

Если координата объекта задана некорректно, то соответствующая операция не выполняется и с новой строки выдается сообщение об ошибке.

Если не найден объект по координате:

```
Object «координата объекта» not found
```

Если не найден целевой объект по координате:

```
Handler object «координата целевого объекта» not found
```

Пример ввода:

```
appls_root
/ object_s1 3
/ object_s2 2
/object_s2 object_s4 4
/ object_s13 5
/object_s2 object_s6 6
/object_s1 object_s7 2
endtree
/object_s2/object_s4 /object_s2/object_s6
/object_s2 /object_s1/object_s7
/ /object_s2/object_s4
/object_s2/object_s4 /
end_of_connections
EMIT /object_s2/object_s4 Send message 1
EMIT /object_s2/object_s4 Send message 2
EMIT /object_s2/object_s4 Send message 3
EMIT /object_s1 Send message 4
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

```
Signal from «абсолютная координата объекта»
```

Если отработал метод обработчика:

```
Signal to «абсолютная координата объекта»   Text: «переданная строка»
```

Пример вывода:

```
Object tree
appls_root
    object_s1
        object_s7
    object_s2
        object_s4
        object_s6
    object_s13
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text:  Send message 1 (class: 4)
Signal to / Text:  Send message 1 (class: 4)
Signal from /object_s2/object_s4
```

```
Signal to /object_s2/object_s6 Text:  Send message 2 (class: 4)
Signal to / Text:  Send message 2 (class: 4)
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text:  Send message 3 (class: 4)
Signal to / Text:  Send message 3 (class: 4)
Signal from /object_s1
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект ob_cl_application класса cl_application предназначен для построения дерева, обработки комманд и запуска приложения;
- в классе cl_base были использованы параметризированные макроопределения препроцессора для приведения указателя на метод сигнала и на метод обработчика, а также были объявлены новые типы данных TYPE_SIGNAL и TYPE_HANDLER для определения указателей на методы сигнала и обработчика.;
- также была добавлена дополнительная структура o_sh, представляющая собой пользовательский тип данных для хранения соединений "сигнал-обрпаботчик".

Класс cl_base:

- свойства/поля:
  - поле отвечающее за хранение соединений "сигнал-обработчик" для определения объекта:
    - наименование — connects;
    - тип — вектор объектов структуры o_sh;
    - модификатор доступа — private;
  - поле для наименования объекта:
    - наименование — s_object_name;
    - тип — string;
    - модификатор доступа — private;
  - поле указатель на головной объект для текущего объекта:
    - наименование — p_head_object;
    - тип — cl_base*;

        - модификатор доступа — private;
    - поле динамический массив указателей на объекты, подчиненные текущему объекту в дереве иерархии:
        - наименование — subordinate_objects;
        - тип — vector<cl_base*>;
        - модификатор доступа — private;
    - поле индикатор состояния объекта:
        - наименование — status;
        - тип — int;
        - модификатор доступа — private;
- функционал:
    - метод set_connection — метод установки связи между сигналом и обработчиком целевого объекта;
    - метод delete_connection — метод для разрыва связи между сигналом текущего объекта и обработчиком целевого объекта;
    - метод emit_signal — метод для выдачи сигнала от текущего объекта с передачей строковой переменной;
    - метод get_status — метод получения статуса объекта;
    - метод get_abs_path — метод для возвращения полного пути текущего объекта иерархии;
    - метод delete_links — метод удаляет связи, идущие к целевому объекту;
    - метод set_status_tree — метод установки статуса у всех объектов в дереве;
    - метод cl_base — параметризированный конструктор класса;
    - метод ~cl_base — деструктор;
    - метод get_name — метод получения имени объекта;

- метод get_p_head — метод получения указателя на родительский объект;
- метод set_name — метод редактирования имени объекта;
- метод get_subordinate_object — получение указателя на непосредственно подчиненный объект по имени;
- метод print_tree — метод вывода дерева;
- метод search_object_from_current — поиск объекта на ветке иерархии от текущего по имени;
- метод search_object_from_tree — поиск по дереву (в корне) иерархии;
- метод set_status — метод установки статуса объекта;
- метод find_obj_by_coord — метод поиска объекта по заданной координате.

Класс cl_application:

- функционал:
  - метод signal_f — метод сигнала;
  - метод handler_f — метод обработчика;
  - метод get_class_num — метод возврата номера класса;
  - метод build_tree_objects — метод, создающий иерархию объекта и устанавливает связи между объектами;
  - метод exec_app — метод запуска приложения;
  - метод cl_application — параметризированный конструктор класса.

Класс cl_2:

- функционал:
  - метод signal_f — метод сигнала;
  - метод handler_f — метод обработчика;
  - метод get_class_num — метод возврата номера класса;

- метод cl_2 — параметризированный конструктор.

Класс cl_3:

- функционал:
  - метод signal_f — метод сигнала;
  - метод handler_f — метод обработчика;
  - метод get_class_num — метод возврата номера класса;
  - метод cl_3 — параметризированный конструктор.

Класс cl_4:

- функционал:
  - метод signal_f — метод сигнала;
  - метод handler_f — метод обработчика;
  - метод get_class_num — метод возврата номера класса;
  - метод cl_4 — параметризированный конструктор.

Класс cl_5:

- функционал:
  - метод signal_f — метод сигнала;
  - метод handler_f — метод обработчика;
  - метод get_class_num — метод возврата номера класса;
  - метод cl_5 — параметризированный конструктор.

Класс cl_6:

- функционал:
  - метод signal_f — метод сигнала;
  - метод handler_f — метод обработчика;
  - метод get_class_num — метод возврата номера класса;
  - метод cl_6 — параметризированный конструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			базовый класс	
		cl_application	public		2
		cl_2	public		3
		cl_3	public		4
		cl_4	public		5
		cl_5	public		6
		cl_6	public		7
2	cl_application			класс приложения	
3	cl_2			класс объекта древа	
4	cl_3			класс объекта древа	
5	cl_4			класс объекта древа	
6	cl_5			класс объекта древа	
7	cl_6			класс объекта древа	

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода set_connection класса cl_base

Функционал: метод установки связи между сигналом и обработчиком целевого объекта.

Параметры: TYPE_SIGNAL p_signal, cl_base* p_target, TYPE_HANDLER p_handler.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_connection класса cl_base

№	Предикат	Действия	№ перехода
1		объявление p_value - указатель на элемент структуры o_sh	2
2	переменная "c" списка connects не равна nullptr		3
			4
3	поля p_signal, p_target, p_handler переменной c равны параметрам p_signal, p_target, p_handler		∅
			4
4		p_value присваивается новая структура по умолчанию	5

№	Предикат	Действия	№ перехода
5		присваивание полю p_signal объекта p_value значение параметра p_signal	6
6		присваивание полю p_target объекта p_value значение параметра p_target	7
7		присваивание полю p_handler объекта p_value значение параметра p_handler	8
8		добавление в конец списка connects объект p_value	∅

3.2 Алгоритм метода delete_connection класса cl_base

Функционал: метод для разрыва связи между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: TYPE_SIGNAL p_signal, cl_base* p_target, TYPE_HANDLER p_handler.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода delete_connection класса cl_base

№	Предикат	Действия	№ перехода
1		объявление p_it типа итератор по элементам вектора o_sh	2
2	p_it не равен указателю на следующий элемент connects после последнего		3
			∅
3	поля p_signal, p_target, p_handler разыменованного указателя p_it равны параметрам p_signal,	вызов деструктора для p_it	4

№	Предикат	Действия	№ перехода
	p_target, p_handler		
		p_it++	2
4		p_it присваивается следущий элемент ocnnects, текущий элемент connects удаляется	5
5		p_it--	2

3.3 Алгоритм метода emit_signal класса cl_base

Функционал: метод для выдачи сигнала от текущего объекта с передачей строковой переменной.

Параметры: TYPE_SIGNAL p_signal, string s_msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода emit_signal класса cl_base

№	Предикат	Действия	№ перехода
1	свойство status равно 0		∅
			2
2		от текущего объектва вызывается разыменованный метод p_signal в качестве аргумента передается s_msg	3
3	переменная "c" списка connects не равна nullptr		4
			∅
4	поле p_signal переменной "c" равен параметру p_signal	объявление указателя p_target и присваивание поле p_target переменной "c"	5
			7
5		объявление указателя на метод обработчика p_handler значением поля p_handler переменной	6

№	Предикат	Действия	№ перехода
		"c"	
6	статус объекта p_target не равен 0	от p_target вызывается разыменованный метод p_handler в качестве аргумента передается s_msg	7
			7
7		"c" присваивается следующий элемент connects	3

3.4 Алгоритм метода get_status класса cl_base

Функционал: метод получения статуса объекта.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_status класса cl_base

№	Предикат	Действия	№ перехода
1		возвращает status текущего объекта	∅

3.5 Алгоритм метода get_abs_path класса cl_base

Функционал: метод для возвращения полного пути текущего объекта иерархии.

Параметры: нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_abs_path класса cl_base

№	Предикат	Действия	№ перехода
1		объявление строки s_abs_path	2

№	Предикат	Действия	№ перехода
2		объявление указателя p_obj на объект класса cl_base адресом текущего объекта	3
3	результат вызова метода get_p_head объекта по указателю p_obj не равен нулевому указателю	s_abs_path присваивается "/" + имя p_obj + s_abs_path	4
			5
4		p_obj присваивается результат вывода метода get_p_head объекта p_obj	3
5	s_abs_path пустой	s_abs_path добавляется символ "/"	6
			6
6		возврат s_abs_path	∅

3.6 Алгоритм метода delete_links класса cl_base

Функционал: метод удаляет связи, идущие к целевому объекту.

Параметры: cl_base* p_target.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода delete_links класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация i = 0	2
2	i меньше размера вектора connects текущего объекта		3
			6
3	поле p_target i-ого элемента connects текущего объекта равен параметру p_target	вызов деструктора для i-ого элемента connects текущего объекта	4

№	Предикат	Действия	№ перехода
			2
4		удаление i-ого элемента connects текущего объекта	5
5		i--	2
6	переменная "sub" вектора subordinate_objects не равна nullptr	вызывается метод delete_links, в качестве аргумента передается p_target, от переменной sub	7
			∅
7		sub присваивается следующий элемент subordinate_objects	6

3.7 Алгоритм метода set_status_tree класса cl_base

Функционал: метод установки статуса у всех объектов в дереве.

Параметры: int status.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода set_status_tree класса cl_base

№	Предикат	Действия	№ перехода
1	у объекта есть головной объект и его свойство status равен 0		∅
			2
2		вызов метода set_status с параметром status	3
3	переменная "sub" вектора subordinate_objects не равна nullptr	вызывается метод set_status_tree, с параметром status, от переменной sub	4
			∅
4		sub присваивается следующий элемент	3

№	Предикат	Действия	№ перехода
		subordinate_objects	

3.8 Алгоритм метода signal_f класса cl_application

Функционал: метод сигнала.

Параметры: ссылка на строку msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода signal_f класса cl_application

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вывода метода get_abs_path	2
2		добавление в конец строки msg " (class: 1)"	∅

3.9 Алгоритм метода handler_f класса cl_application

Функционал: метод обработчика.

Параметры: string msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода handler_f класса cl_application

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результат вывода get_abs_path, " Text: " и строку msg	∅

3.10 Алгоритм метода get_class_num класса cl_application

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода get_class_num класса cl_application

№	Предикат	Действия	№ перехода
1		возврат 1	∅

3.11 Алгоритм метода signal_f класса cl_2

Функционал: метод сигнала.

Параметры: ссылка на строку msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода signal_f класса cl_2

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вывода метода get_abs_path	2
2		добавление в конец строки msg " (class: 2)"	∅

3.12 Алгоритм метода handler_f класса cl_2

Функционал: метод обработчика.

Параметры: string msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода handler_f класса cl_2

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результат вывода get_abs_path, " Text: " и строку msg	∅

3.13 Алгоритм метода get_class_num класса cl_2

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода get_class_num класса cl_2

№	Предикат	Действия	№ перехода
1		возврат 2	∅

3.14 Алгоритм метода signal_f класса cl_3

Функционал: метод сигнала.

Параметры: ссылка на строку msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода signal_f класса cl_3

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вывода метода get_abs_path	2
2		добавление в конец строки msg " (class: 3)"	∅

3.15 Алгоритм метода handler_f класса cl_3

Функционал: метод обработчика.

Параметры: string msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода handler_f класса cl_3

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результат вывода get_abs_path, " Text: " и строку msg	∅

3.16 Алгоритм метода get_class_num класса cl_3

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода get_class_num класса cl_3

№	Предикат	Действия	№ перехода
1		возврат 3	∅

3.17 Алгоритм метода signal_f класса cl_4

Функционал: метод сигнала.

Параметры: ссылка на строку msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода signal_f класса cl_4

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вывода метода get_abs_path	2
2		добавление в конец строки msg " (class: 4)"	∅

3.18 Алгоритм метода handler_f класса cl_4

Функционал: метод обработчика.

Параметры: string msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода handler_f класса cl_4

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результат вывода get_abs_path, " Text: " и строку msg	∅

3.19 Алгоритм метода get_class_num класса cl_4

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода get_class_num класса cl_4

№	Предикат	Действия	№ перехода
1		возврат 4	∅

3.20 Алгоритм метода signal_f класса cl_5

Функционал: метод сигнала.

Параметры: ссылка на строку msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода signal_f класса cl_5

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вывода метода get_abs_path	2
2		добавление в конец строки msg " (class: 5)"	∅

3.21 Алгоритм метода handler_f класса cl_5

Функционал: метод обработчика.

Параметры: string msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода handler_f класса cl_5

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результат вывода get_abs_path, " Text: " и строку msg	∅

3.22 Алгоритм метода get_class_num класса cl_5

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода get_class_num класса cl_5

№	Предикат	Действия	№ перехода
1		возврат 5	∅

3.23 Алгоритм метода signal_f класса cl_6

Функционал: метод сигнала.

Параметры: ссылка на строку msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода signal_f класса cl_6

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вывода метода get_abs_path	2
2		добавление в конец строки msg " (class: 6)"	∅

3.24 Алгоритм метода handler_f класса cl_6

Функционал: метод обработчика.

Параметры: string msg.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода handler_f класса cl_6

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результат вывода get_abs_path, " Text: " и строку msg	∅

3.25 Алгоритм метода get_class_num класса cl_6

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода get_class_num класса cl_6

№	Предикат	Действия	№ перехода
1		возврат 6	∅

3.26 Алгоритм метода build_tree_objects класса cl_application

Функционал: метод, создающий иерархию объекта и устанавливает связи между объектами.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2
2		объявление строк s_head_name, s_sub_name	3
3		объявление указателя p_head на объект класса cl_base с адресом текущего объекта	4
4		объявление указателя p_sub на объект класса cl_base	5
5		объявление int class_number	6
6		объявление строк s_sender, s_reciever	7
7		объявление указателя p_sender на объект класса	8

№	Предикат	Действия	№ перехода
		cl_base	
8		объявление указателя p_reciever на объект класса cl_base	9
9		ввод s_head_name	10
10		вызов метода set_name текущего объекта с параметром s_head_name	11
11		ввод s_head_name	12
12	s_head_name не равен "endtree"	ввод s_sub_name и class_number	13
			26
13		присваивание p_head результат вызова метода find_obj_by_coord с параметром s_head_name	14
14	p_head равен нулевому указателю	вызов print_tree	15
			17
15		вывод с новой строки "The head object " s_head_name " is not found"	16
16		выход с кодом 1	∅
17	у объекта по указателю p_head нет подчиненного с именем s_sub_name	вывод с новой строки s_head_name " Dubbing the names of subordinate objects"	24
			18
18	class_number равен 1	создание объекта класса cl_application, с помощью оператора new и параметром p_head	24
			19
19	class_number равен 2	создание объекта класса cl_2 ,с помощью оператора new с и параметрами p_head, s_sub_name	24
			20

№	Предикат	Действия	№ перехода
20	class_number равен 3	создание объекта класса cl_3 ,с помощью оператора new с и параметрами p_head, s_sub_name	24
			21
21	class_number равен 4	создание объекта класса cl_4 ,с помощью оператора new с и параметрами p_head, s_sub_name	24
			22
22	class_number равен 5	создание объекта класса cl_5 ,с помощью оператора new с и параметрами p_head, s_sub_name	24
			23
23	class_number равен 6	создание объекта класса cl_6 ,с помощью оператора new с и параметрами p_head, s_sub_name	24
			24
24		ввод s_head_name	25
25		объявление вектора элементов TYPE_SIGNALS signals хранящий сигналы классов 1-6 и указатель на метод signal_f	26
26		объявление вектора элементов TYPE_HANDLER handlers хранящий обработчик классов 1-6 и указатель на метод handler_f	27
27		ввод s_sender	28
28	s_sender не равен "end_of_connections"	p_sender присваивается значение метода find_obj_by_coord с параметром s_sender	29
			∅
29	p_sender равен нулевому указателю	вывод с новой строки "Object " s_sender " not found"	27

№	Предикат	Действия	№ перехода
			30
30		ввод s_reciever	31
31		p_reciever присваивается значение метода find_obj_by_coord с параметром s_reciever	32
32	p_reciever равен нулевому указателю	вывод с новой строки "Handler object " s_reciever " not found"	27
			33
33		вызов метода set_connection указателя p_sender, с аргументами в виде: сигнал класса p_sender, указателя p_reciever, обработчик класса p_reciever	34
34		ввод s_sender	28

3.27 Алгоритм метода exec_app класса cl_application

Функционал: запуск приложения, считывание комманд.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		объявление строк s_command, s_coordinate, s_text	2
2		объявление строк s_sender, s_reciever	3
3		объявление указателя p_sender и p_reciever на объект класса cl_base	4
4		объявление int object_status	5
5		объявление вектора элементов TYPE_SIGNALS signals хранящий сигналы классов 1-6 и указатель на метод signal_f	6

№	Предикат	Действия	№ перехода
6		объявление вектора элементов TYPE_HANDLER handlers хранящий обработчик классов 1-6 и указатель на метод handler_f	7
7		вызов от текущего объекта метод set_status_tree с аргументом 1	8
8		вызов метода print_tree	9
9		ввод s_command	10
10	s_command не равен "END"	ввод s_sender	11
		возврат 0	∅
11		присваивание указателю p_sender значение выполнения метода find_obj_by_coord с параметром s_sender	12
12	p_sender равен пустому указателю	вывод с новой строки "Object " s_sender " not found"	13
			14
13		ввод s_sender	9
14	s_command равен "EMIT"	считывание строки в переменную s_text	15
			16
15		вызов метода emit_signal указателя p_sender, с аргументами в виде: сигнал класса p_sender, s_text	14
16	7 индекс s_command равен 'N'	ввод s_reciever	17
			19
17		присваивание указателю p_reciever значение выполнения метода find_obj_by_coord с параметром s_reciever	18
18	p_reciever равен нулевому указателю	вывод с новой строки "Handler object " s_reciever " not found"	9
		вызов метода set_connection указателя p_sender, с	19

№	Предикат	Действия	№ перехода
		аргументами в виде: сигнал класса p_sender, указателя p_reciever, обработчик класса p_reciever	
19	s_command равен "DELETE_CONNECT"	ввод s_reciever	20
			22
20		присваивание указателю p_reciever значение выполнения метода find_obj_by_coord с параметром s_reciever	21
21	p_reciever равен нулевому указателю	вывод с новой строки "Handler object " s_reciever " not found"	9
		вызов метода delete_connection указателя p_sender, с аргументами в виде: сигнал класса p_sender, указателя p_reciever, обработчик класса p_reciever	22
22	s_command равен "SET_CONDITION"	ввод object_status	23
			24
23		вызов метода set_status с параметром object_status объекта p_sender	24
24		ввод s_command	∅

3.28 Алгоритм функции main

Функционал: основная программа.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 29.

Таблица 29 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_cl_application класса cl_application в качестве параметра подается нулевой указатель	2
2		вызов метода build_tree_objects объекта ob_cl_application	3
3		вызов метода exec_app объекта ob_cl_application	∅

3.29 Алгоритм конструктора класса cl_base

Функционал: параметризированный конструктор класса.

Параметры: cl_base* p_head_object - указатель на головной объект, string s_object_name - имя узла дерева, по умолчанию подается "Base object".

Алгоритм конструктора представлен в таблице 30.

Таблица 30 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		полю p_head_object этого объекта присваивается параметр p_head_object	2
2		полю s_object_name этого объекта присваивается параметр s_object_name	3
3	p_head_object ненулевой	в вектор указателей подчиненных объектов головного объекта добавляется указатель на этот объект	∅
			∅

3.30 Алгоритм деструктора класса cl_base

Функционал: деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 31.

Таблица 31 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		от текущего объекта вызывается метод delete_links с аргументом текущего объекта	2
2	переменная p_sub вектора subordinate_objects не равна nullptr	вызов деструктора для p_sub, освобождается память	3
			∅
3		p_sub присваивается следующий элемент subordinate_objects	2

3.31 Алгоритм метода get_name класса cl_base

Функционал: метод получения имени объекта.

Параметры: нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 32.

Таблица 32 – Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода
1		возвращает s_object_name текущего объекта	∅

3.32 Алгоритм метода get_p_head класса cl_base

Функционал: метод получения указателя на родительский объект.

Параметры: нет.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 33.

Таблица 33 – Алгоритм метода get_p_head класса cl_base

№	Предикат	Действия	№ перехода
1		возвращает указатель p_head_object текущего объекта	∅

3.33 Алгоритм метода set_name класса cl_base

Функционал: метод редактирования имени объекта.

Параметры: string new_name - новое имя узла дерева.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 34.

Таблица 34 – Алгоритм метода set_name класса cl_base

№	Предикат	Действия	№ перехода
1	p_head_object этого объекта не равен пустому указателю		2
		s_object_name текущего объекта присваивается new_name	4
2	переменная p_sub вектора subordinate_objects указателя p_head_objects не равна nullptr		3
			4
3	результат вызова метода get_name объекта p_sub равен new_name	возврат false	∅
		p_sub присваивается следующий элемент subordinate_objects	2
4		возврат true	∅

3.34 Алгоритм метода get_subordinate_object класса cl_base

Функционал: получение указателя на непосредственно подчиненный объект по имени.

Параметры: string search_name - имя искомого объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 35.

Таблица 35 – Алгоритм метода get_subordinate_object класса cl_base

№	Предикат	Действия	№ перехода
1	переменная p_sub вектора subordinate_objects не равна nullptr		2
			3
2	результат вызова метода get_name объекта p_sub равен search_name	возврат p_sub	∅
		p_sub присваивается следующий элемент subordinate_objects	1
3		возврат пустого указателя	∅

3.35 Алгоритм метода print_tree класса cl_base

Функционал: метод вывода дерева.

Параметры: int layer - уровень объекта на дереве.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 36.

Таблица 36 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1		вывод со следующей строки, вывод layer пробелов << результат вызова метода get_name текущего объекта	2
2	переменная p_sub вектора subordinate_objects не равна nullptr	вызов метода print_tree объекта p_sub с параметром layer+4	3
			∅
3		p_sub присваивается следующий элемент subordinate_objects	2

3.36 Алгоритм метода search_object_from_current класса cl_base

Функционал: поиск объекта на ветке иерархии от текущего по имени.

Параметры: string s_name - имя искомого объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 37.

Таблица 37 – Алгоритм метода search_object_from_current класса cl_base

№	Предикат	Действия	№ перехода
1		объявление очереди q, которая принимает указатели на объекта класса cl_base	2
2		инициализация пустого указателя p_found на объекты класса cl_base	3
3		добавление в конец очереди q указателя на текущий объект	4
4	очередь q не пустая	инициализация указателя p_front на объект класса cl_base значением первого элемента в очереди q	5

№	Предикат	Действия	№ перехода
			10
5		удаление элемент из начала очереди q	6
6	результат вызова метода get_name объекта p_front равен параметру s_name		7
			8
7	p_found равен пустому указателю	p_found присваивается значение p_front	8
		возврат пустого указателя	∅
8	переменная p_sub вектора subordinate_objects указателя p_front не равна nullptr	добавление в конец очереди q элемента p_sub	9
			4
9		p_sub присваивается следующий элемент subordinate_objects	8
10		возврат p_found	∅

3.37 Алгоритм метода search_object_from_tree класса cl_base

Функционал: поиск по дереву (в корне) иерархии.

Параметры: string s_name - имя искомого объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 38.

Таблица 38 – Алгоритм метода search_object_from_tree класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация указателя p_root и присваивание текущего объекта	2
2	головной объект p_root не	p_root присваивается результат вызова метода	2

№	Предикат	Действия	№ перехода
	равен пустому указателю	get_p_head указателя p_root	
			3
3		возврат вызова метода search_object_from_current указателя p_root с параметром s_name	∅

3.38 Алгоритм метода set_status класса cl_base

Функционал: метод установки статуса объекта.

Параметры: int status - номер состояния.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 39.

Таблица 39 – Алгоритм метода set_status класса cl_base

№	Предикат	Действия	№ перехода
1	p_head_object равен пустому указателю или поле status указателя p_head_object не равен 0	присваивание status текущего объекта параметр status	2
			2
2	status равен 0	присваивание status текущего объекта параметр status	3
			∅
3		инициализация i =0	4
4	i < размера вектора subordinate_objects	вызов метода set_status i-ого элемента subordinate_objects с параметром status	5
			∅
5		i++	4

3.39 Алгоритм метода find_obj_by_coord класса cl_base

Функционал: метод поиска объекта по координате.

Параметры: string s_coord - координата искомого объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 40.

Таблица 40 – Алгоритм метода find_obj_by_coord класса cl_base

№	Предикат	Действия	№ перехода
1		p_root присваивается указатель на текущий объект. Инициализация i_slash_2 = 0. Строковой тип s_name. Инициализация указателя p_obj	2
2	s_coord равен "/"		3
			4
3	p_root не равен пустому указателю	p_root присваивается значение вызываемого метода get_p_head от p_root	3
		возврат p_root	∅
4	s_coord равен "."	возврат текущего объекта	∅
			5
5	s_coord[0] равен '/' и s_coord[1] равен '/'	возврат вызываемого метода search_object_from_tree от текущего объекта с параметром s_coord с встроенном методом substr(2) (начиная с 3 символа)	∅
			6
6	s_coord равен "."	возврат вызываемого метода search_object_from_current от текущего объекта с параметром s_coord с встроенном методом substr(1) (начиная со 2 символа)	∅
			7
7		i_slash_2 присваивается индекс второго слэша	8

№	Предикат	Действия	№ перехода
8	s_coord[0] равен "/"		9
			14
9	головной объект не равен пустому указателю	p_root присваивается значение вызываемого метода get_p_head от p_root	9
			10
10	i_slash_2 не равен -1	s_name присваивается значение объекта до следущего слэша из переменной s_coord	11
		s_name присваивается значение объекта с 1 символа s_coord	13
11		p_obj присваивается значение вызываемого метода get_subordinate_objects с параметром s_name объекта p_root	12
12	p_obj не равен пустому указателю	вовзрат вызываемого метода find_obj_by_coord с параметром s_coord (отсекается строка от найденного индекса слэша) объекта p_obj	∅
		возврат p_obj	∅
13		возврат вызываемого метода get_subordinate_object с параметром s_name объекта p_root	∅
14	i_slash_2 не равен -1	s_name присваивается значение объекта до следующего слэша из переменной s_coord	15
		возврат вызываемого метода get_subordinate_object с параметром s_coord текущего объекта	∅
15		p_obj присваивается значение вызываемого метода get_subordinate_object с параметром s_name текущего объекта	16
16	p_obj не равен пустому указателю	возврат вызываемого метода find_obj_by_coord с параметром s_coord (строка, после найденного	∅

№	Предикат	Действия	№ перехода
		первого слэша) объекта p_obj	
		возврат p_obj	∅

3.40 Алгоритм конструктора класса cl_application

Функционал: параметризированный конструктор класса.

Параметры: cl_base* p_head_object.

Алгоритм конструктора представлен в таблице 41.

Таблица 41 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметром p_head_object	∅

3.41 Алгоритм конструктора класса cl_2

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

Алгоритм конструктора представлен в таблице 42.

Таблица 42 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head_objects и s_object_name	∅

3.42 Алгоритм конструктора класса cl_3

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

Алгоритм конструктора представлен в таблице 43.

Таблица 43 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head_objects и s_object_name	∅

3.43 Алгоритм конструктора класса cl_4

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

Алгоритм конструктора представлен в таблице 44.

Таблица 44 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head_objects и s_object_name	∅

3.44 Алгоритм конструктора класса cl_5

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

Алгоритм конструктора представлен в таблице 45.

Таблица 45 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head_objects и s_object_name	∅

3.45 Алгоритм конструктора класса cl_6

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

Алгоритм конструктора представлен в таблице 46.

Таблица 46 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head_objects и s_object_name	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-36.

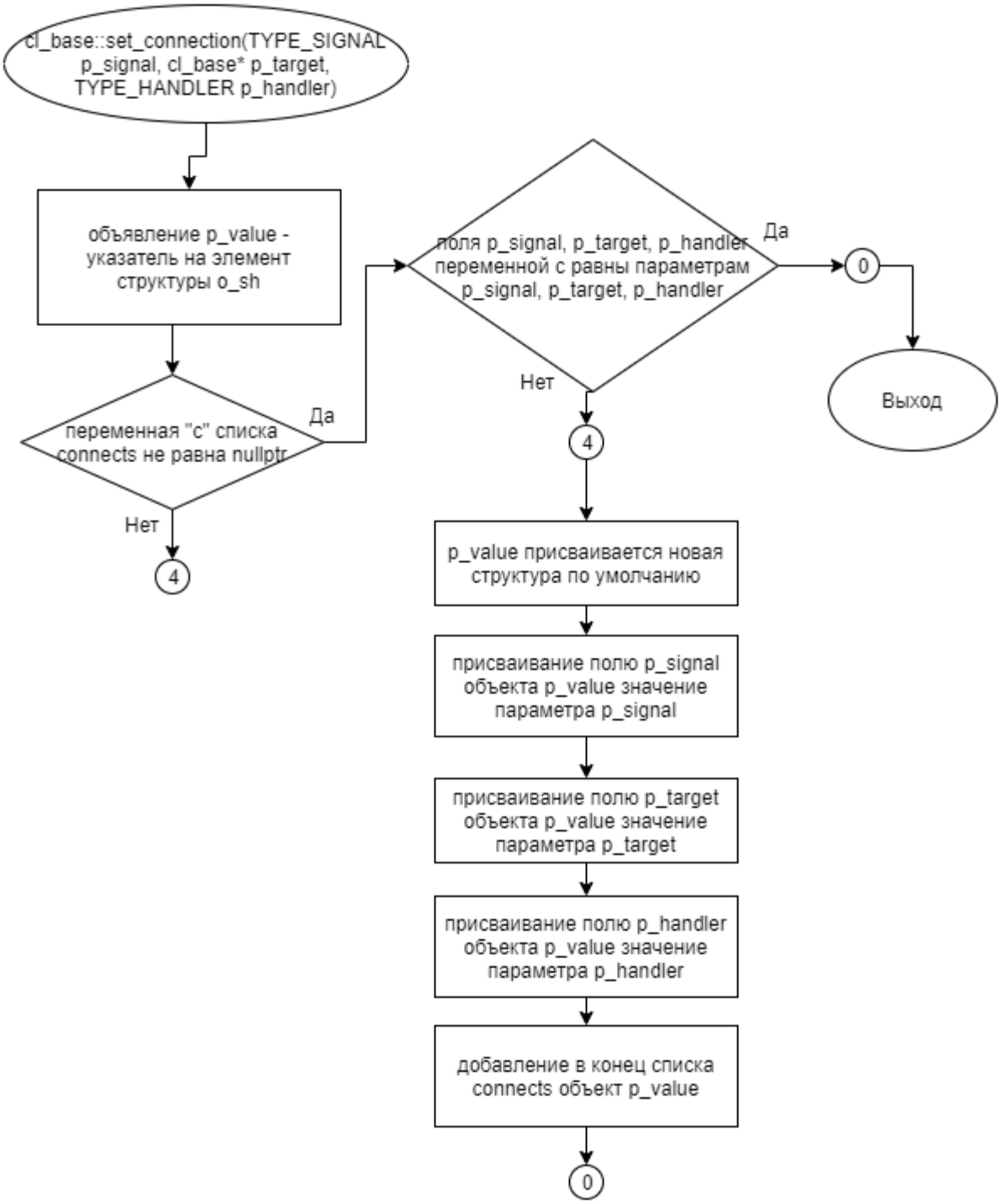


Рисунок 1 – Блок-схема алгоритма

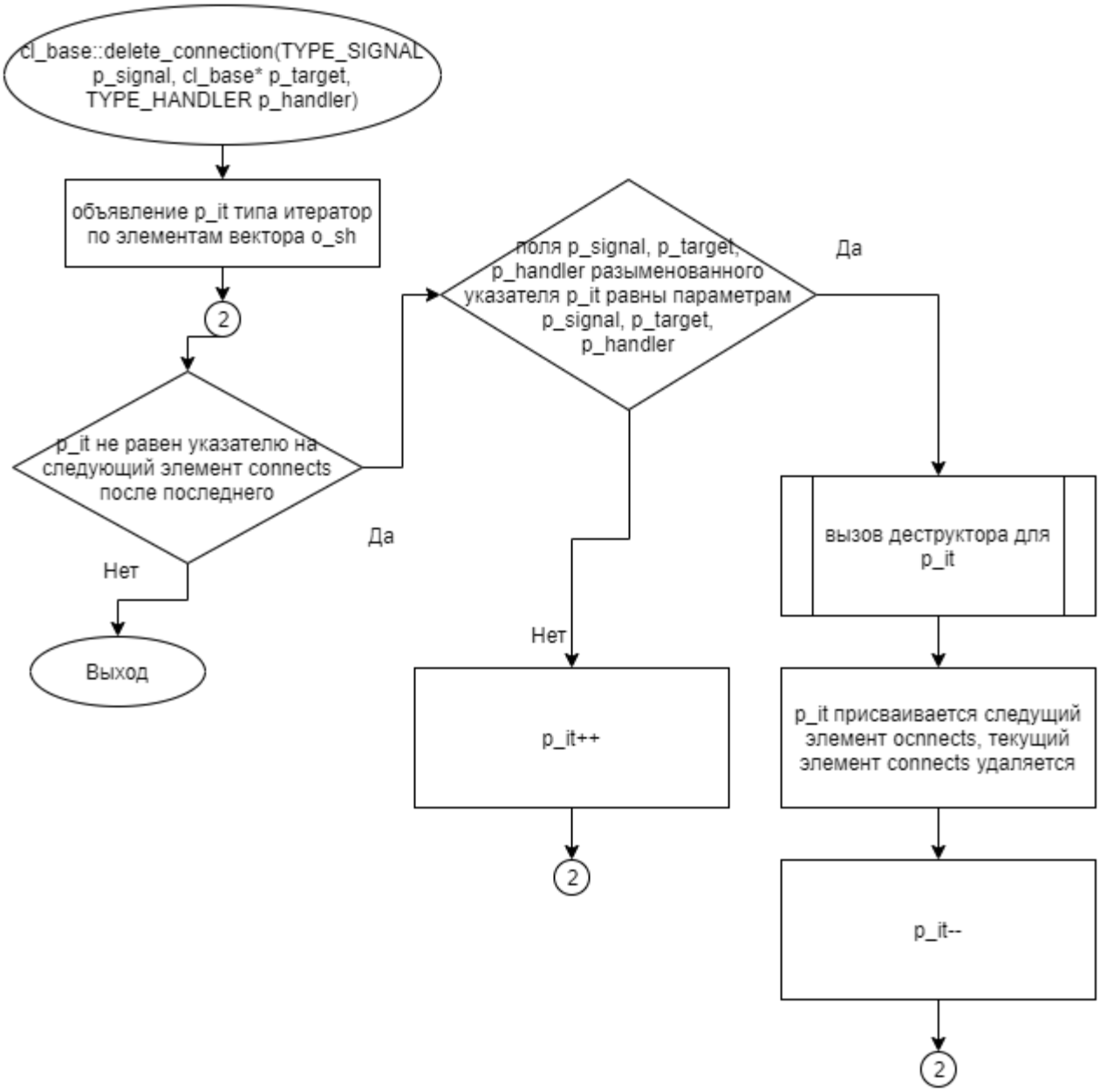


Рисунок 2 – Блок-схема алгоритма

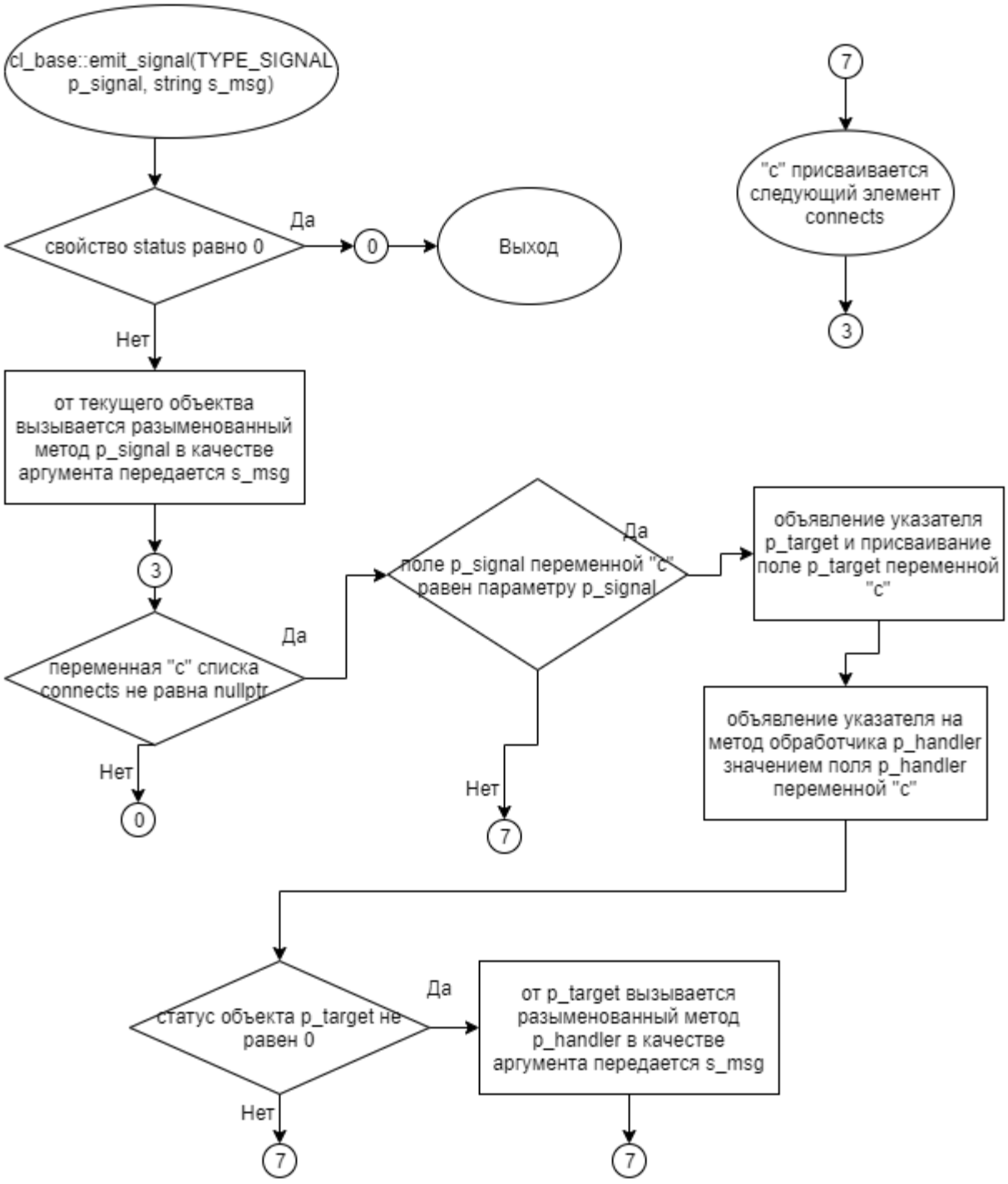


Рисунок 3 – Блок-схема алгоритма

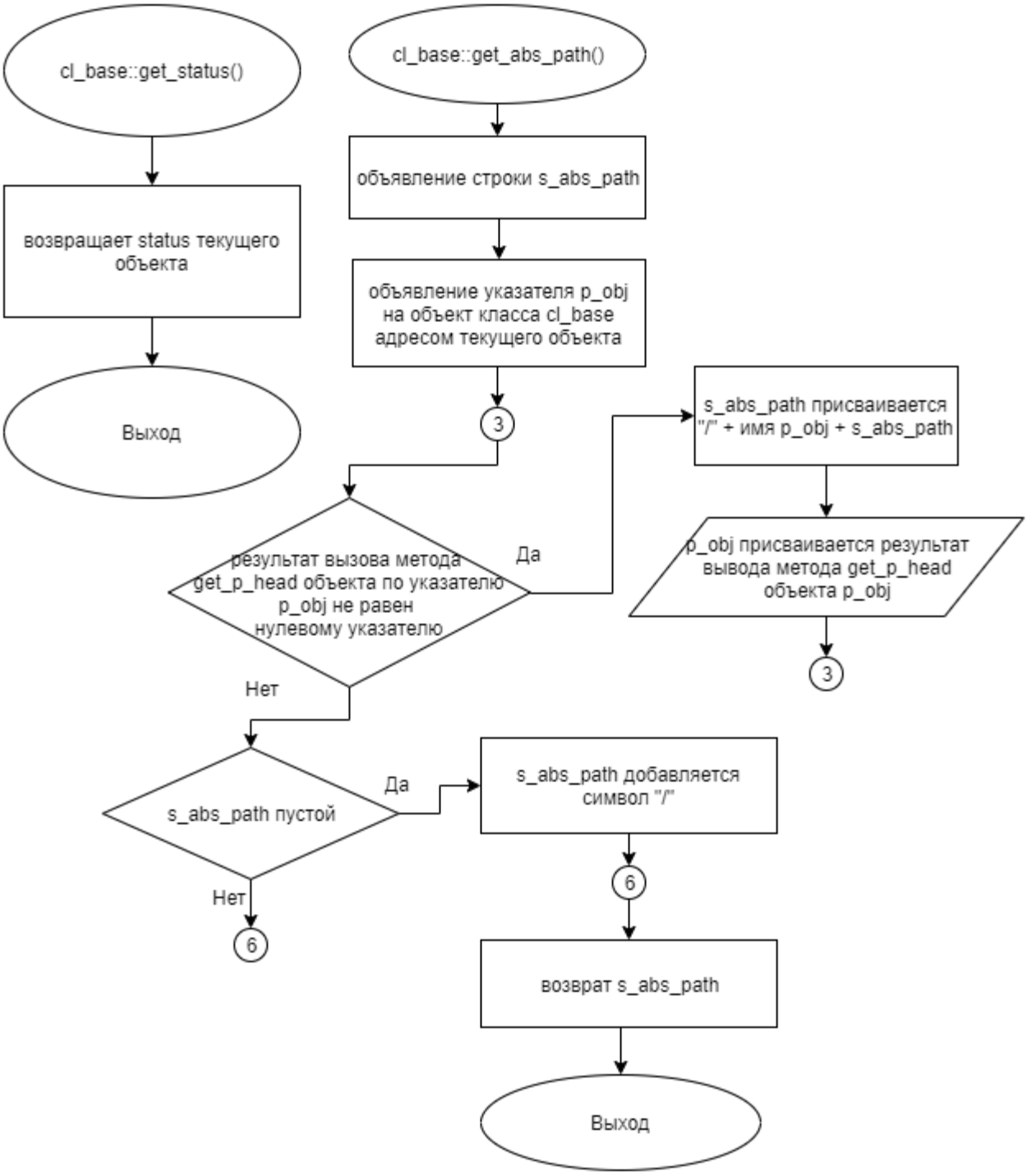


Рисунок 4 – Блок-схема алгоритма

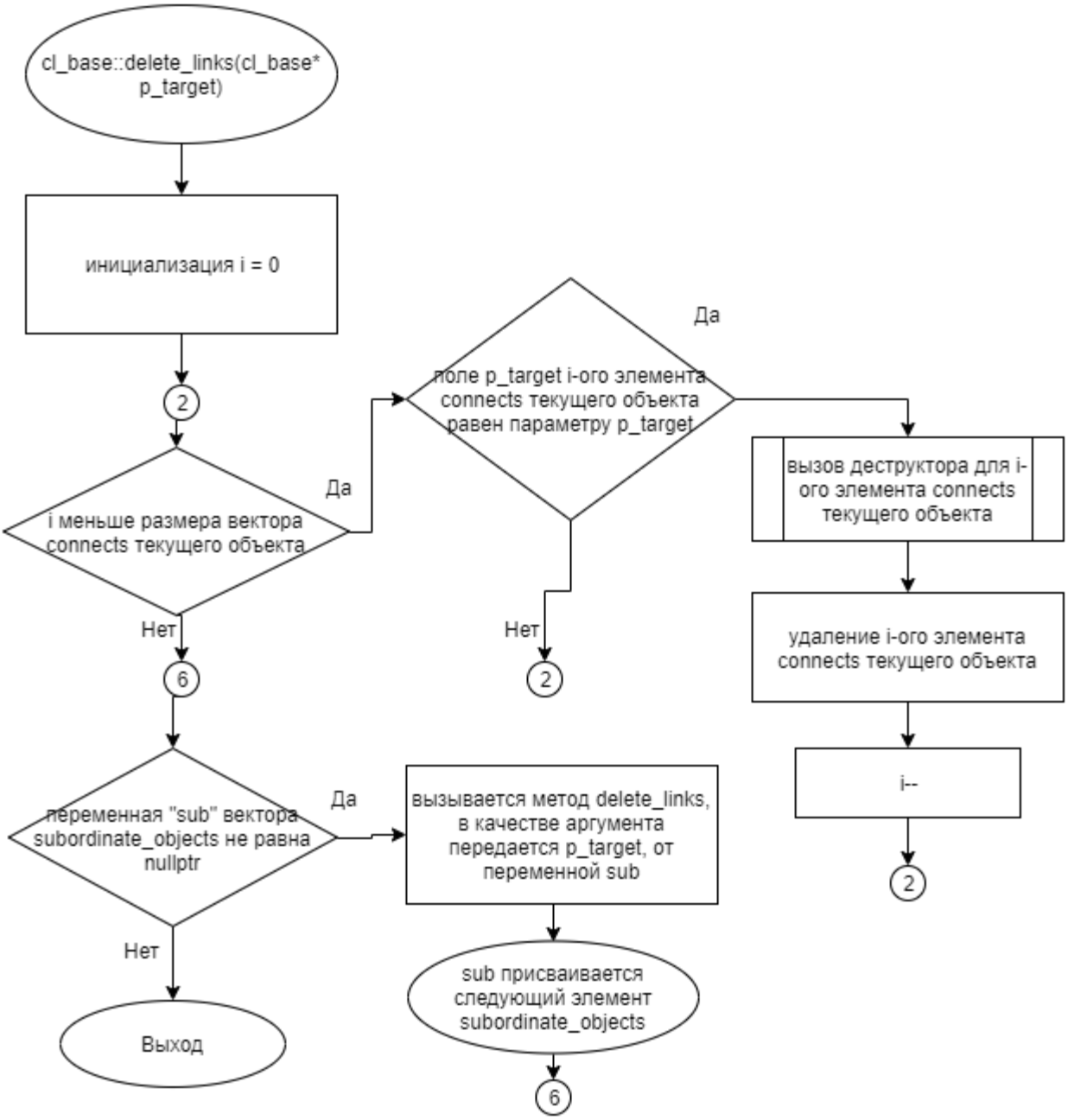


Рисунок 5 – Блок-схема алгоритма

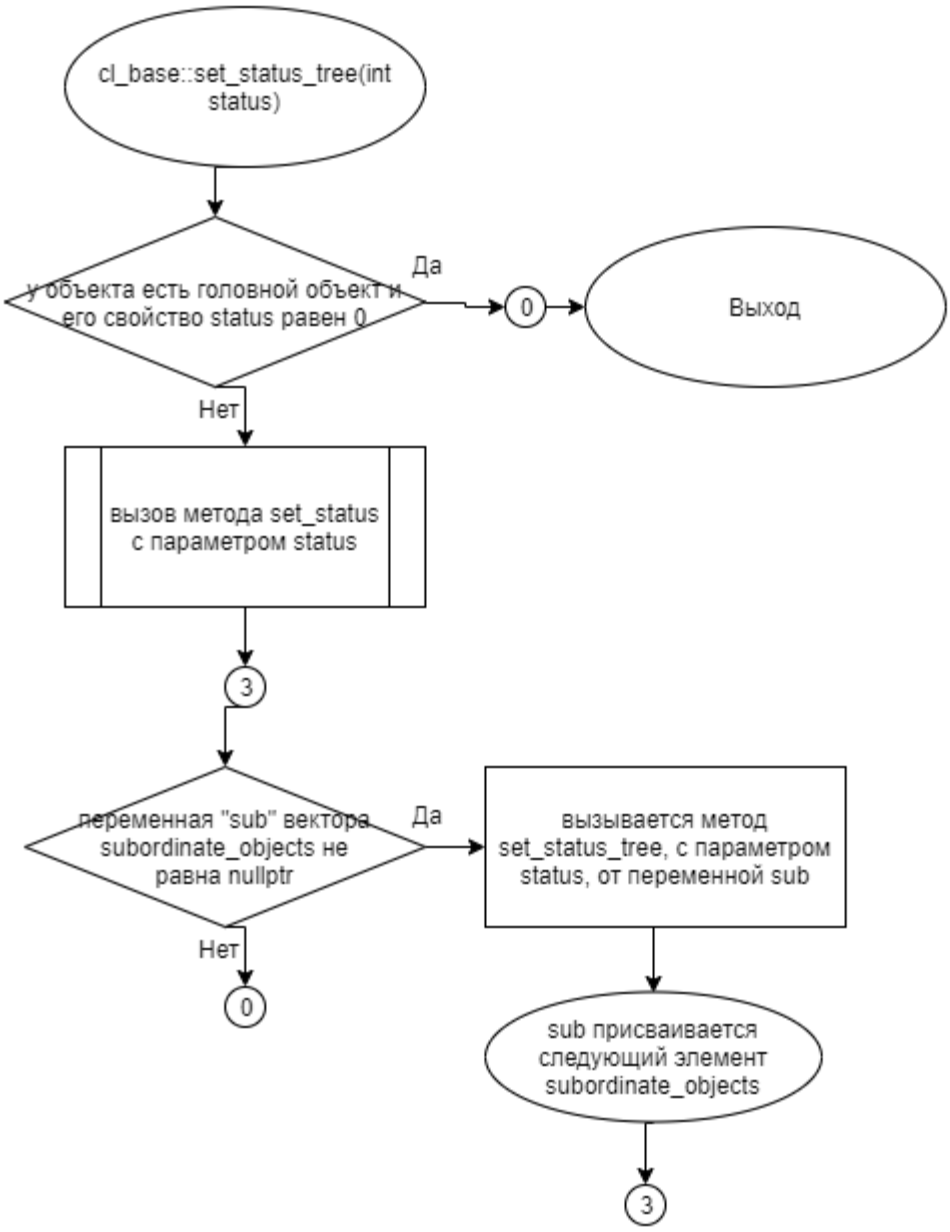


Рисунок 6 – Блок-схема алгоритма

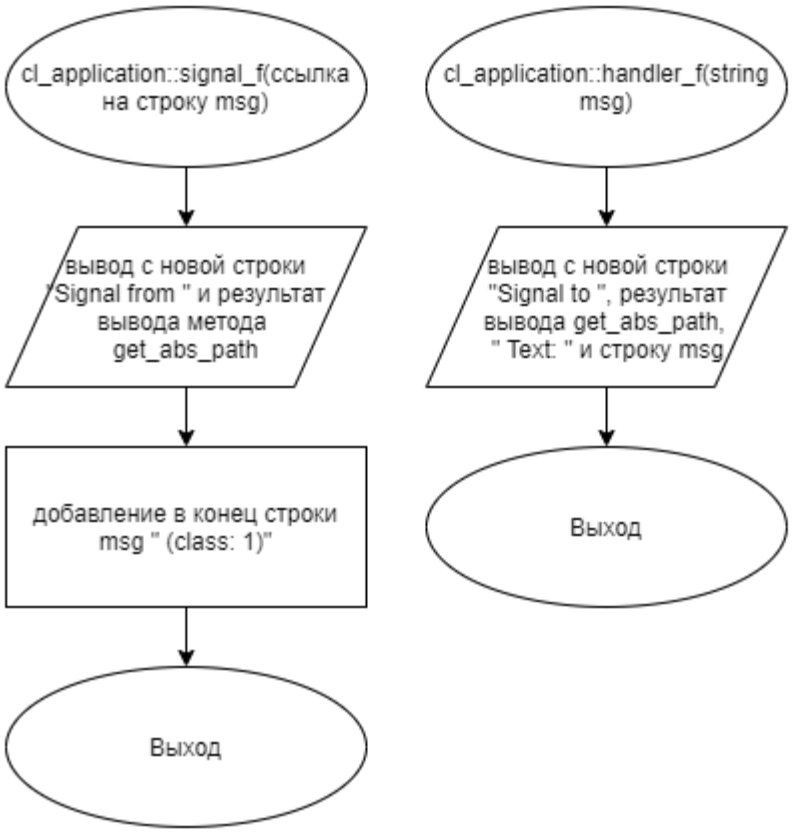


Рисунок 7 – Блок-схема алгоритма

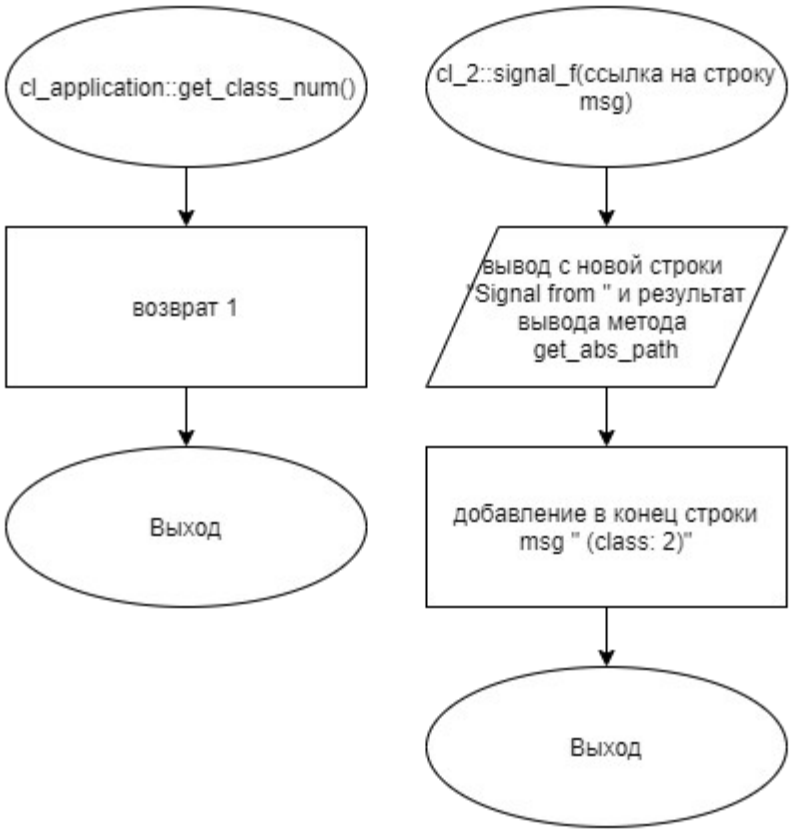


Рисунок 8 – Блок-схема алгоритма

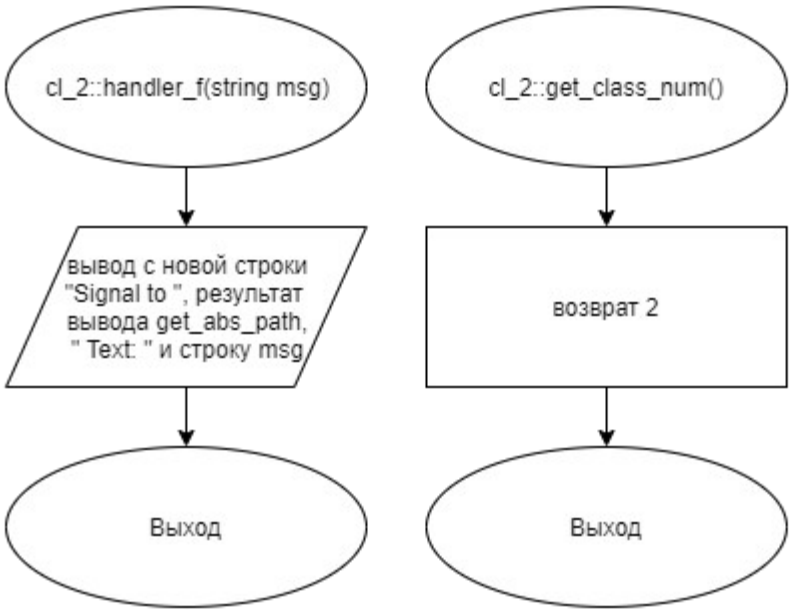


Рисунок 9 – Блок-схема алгоритма

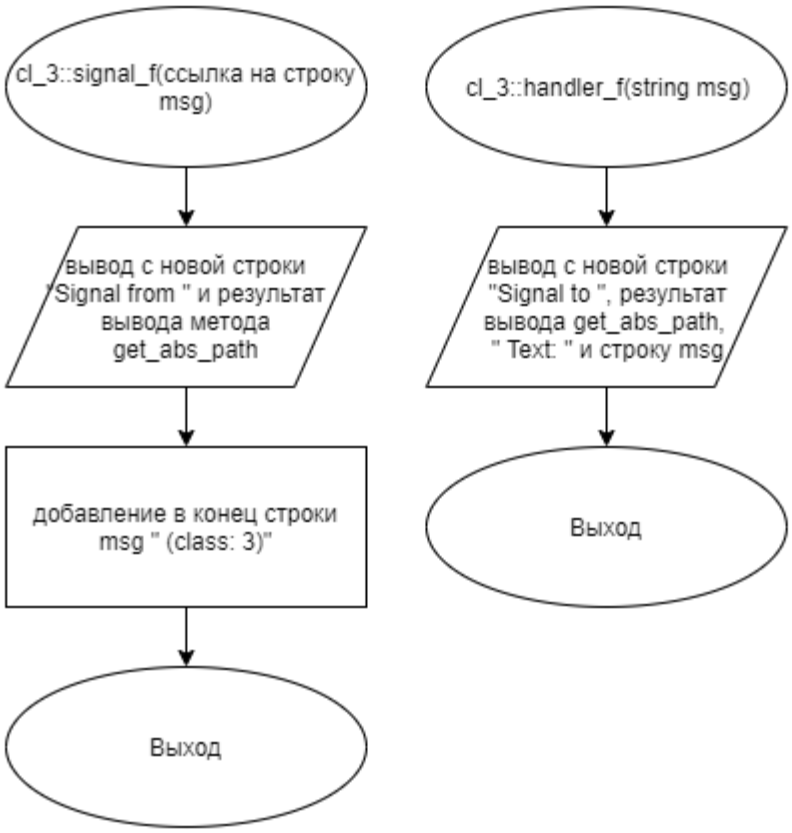


Рисунок 10 – Блок-схема алгоритма

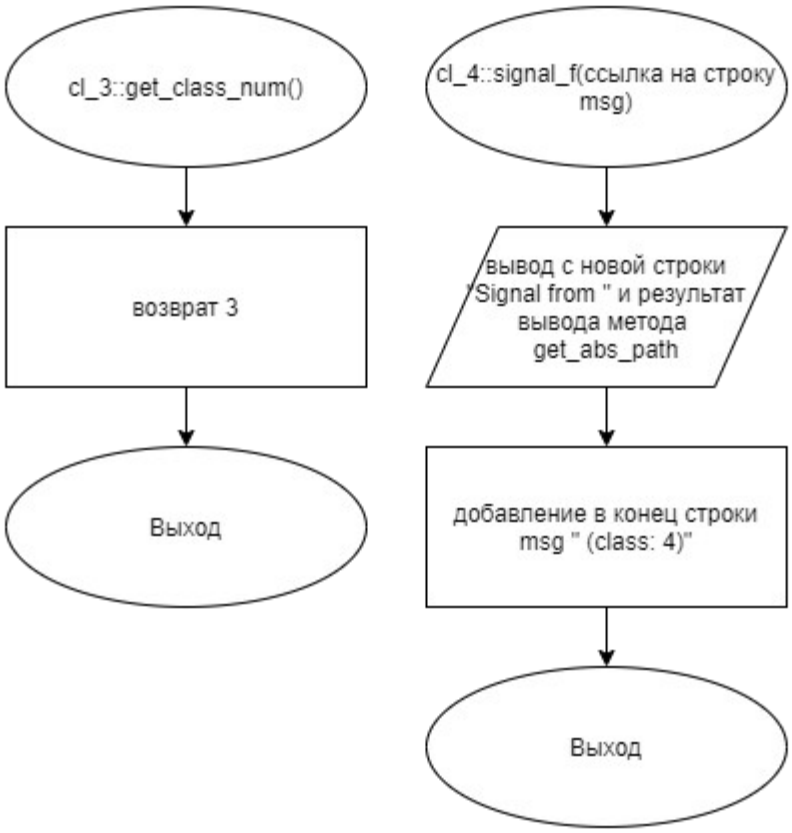


Рисунок 11 – Блок-схема алгоритма

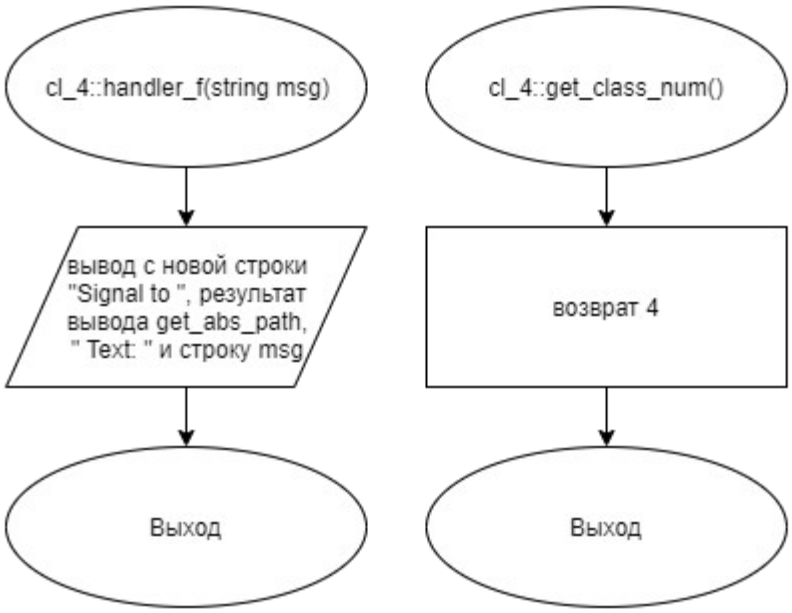


Рисунок 12 – Блок-схема алгоритма

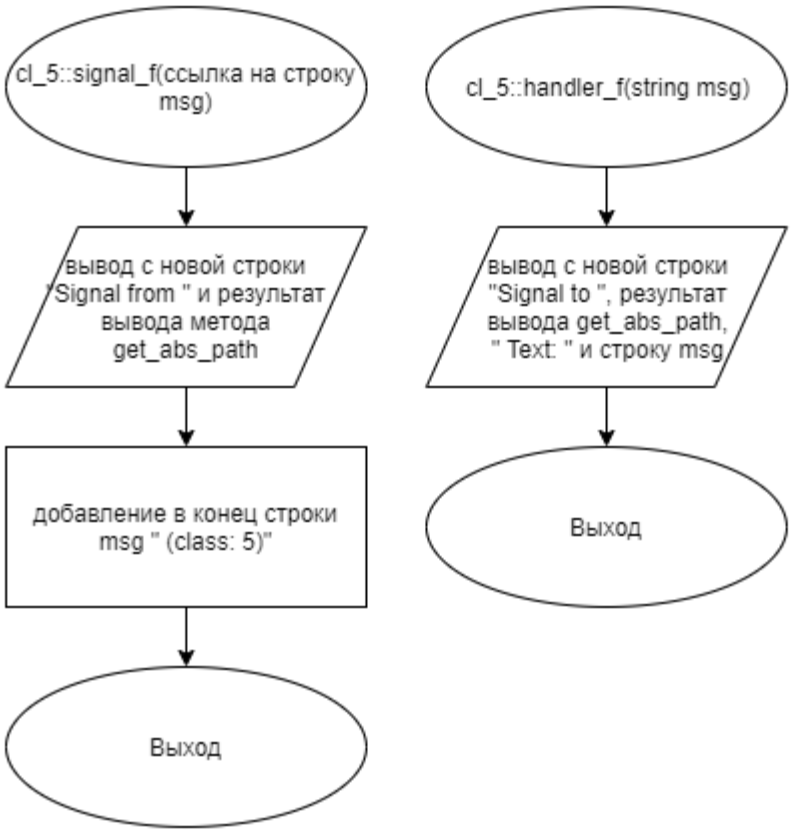


Рисунок 13 – Блок-схема алгоритма

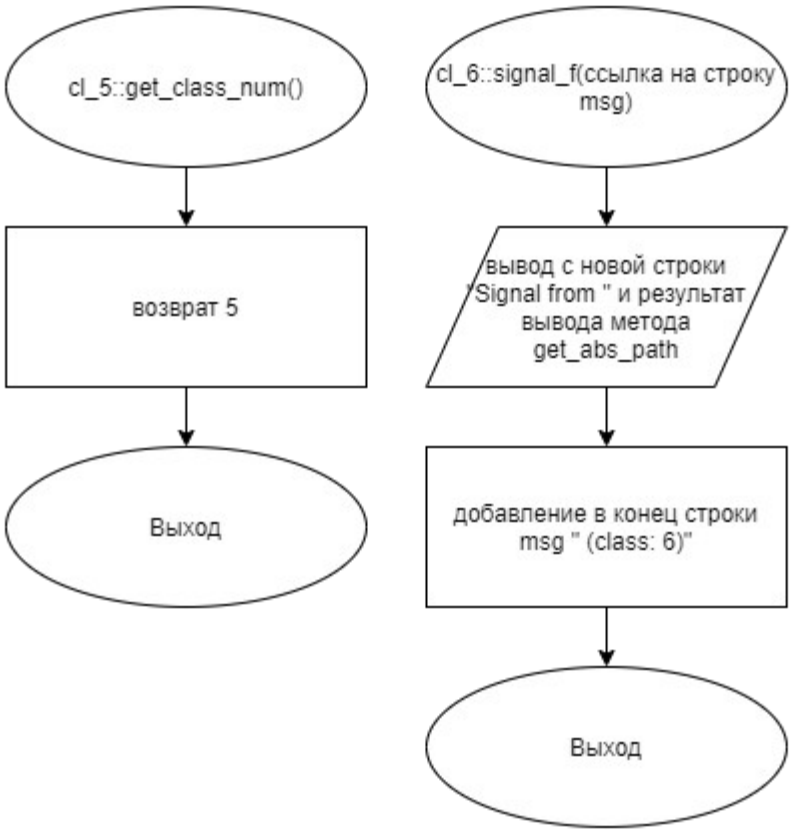


Рисунок 14 – Блок-схема алгоритма

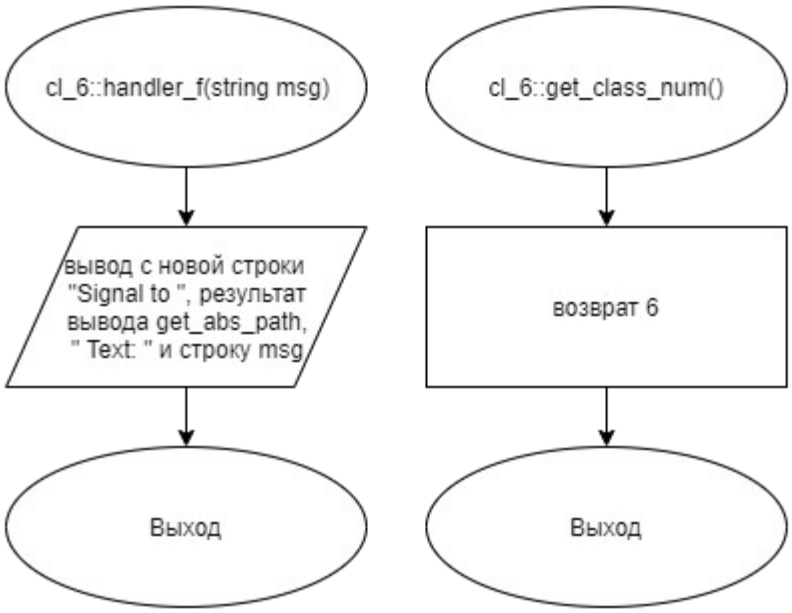


Рисунок 15 – Блок-схема алгоритма

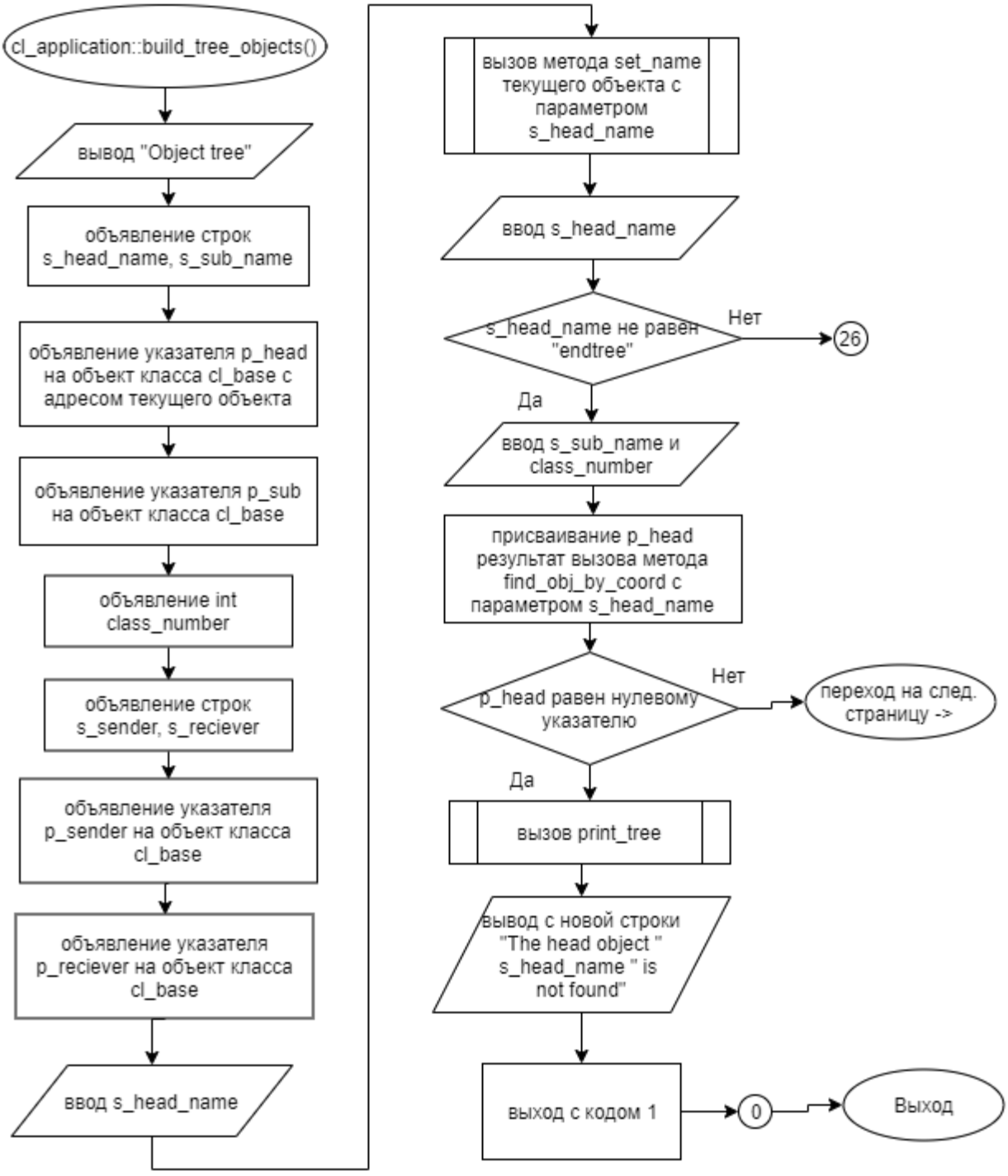


Рисунок 16 – Блок-схема алгоритма

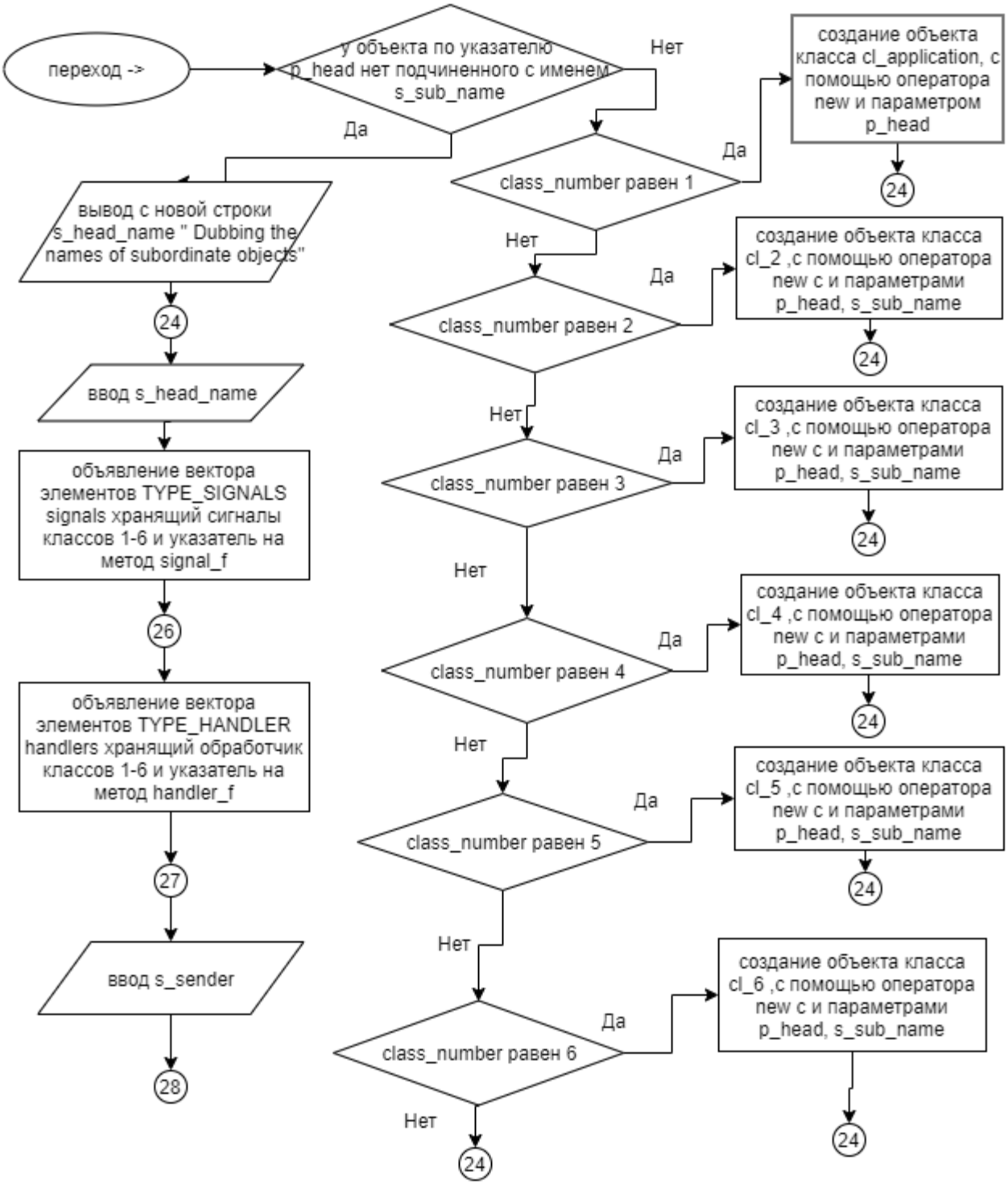


Рисунок 17 – Блок-схема алгоритма

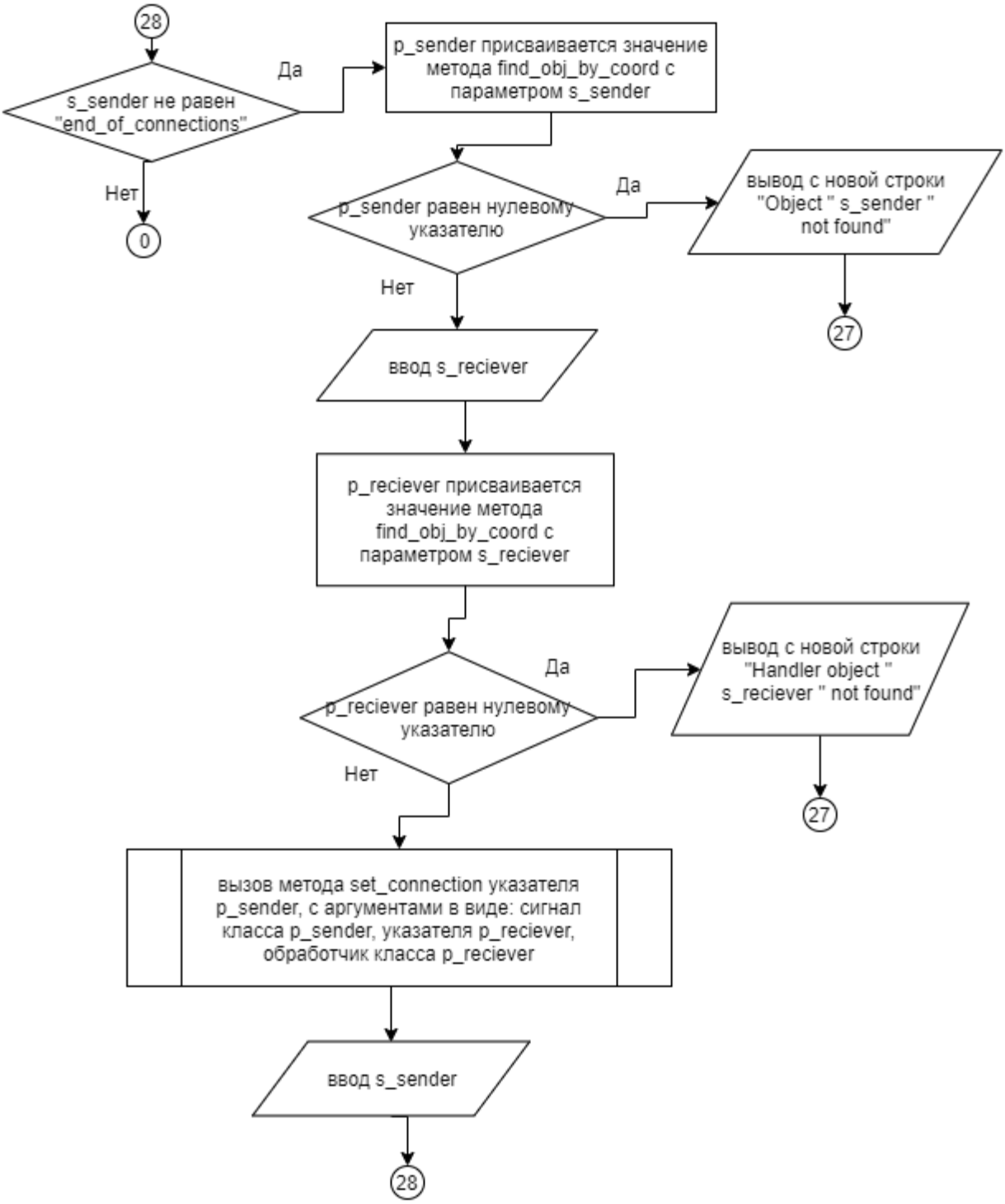


Рисунок 18 – Блок-схема алгоритма

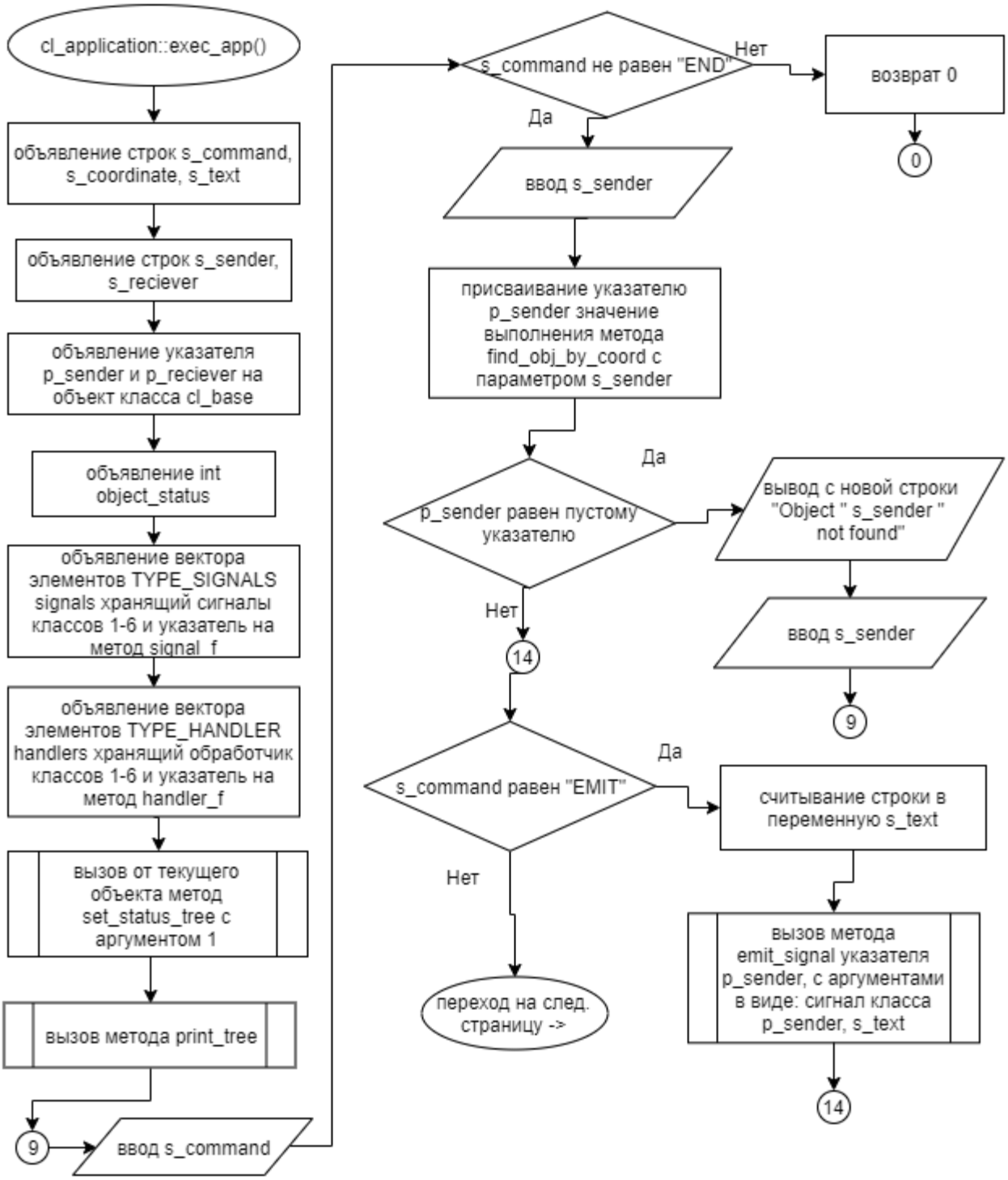
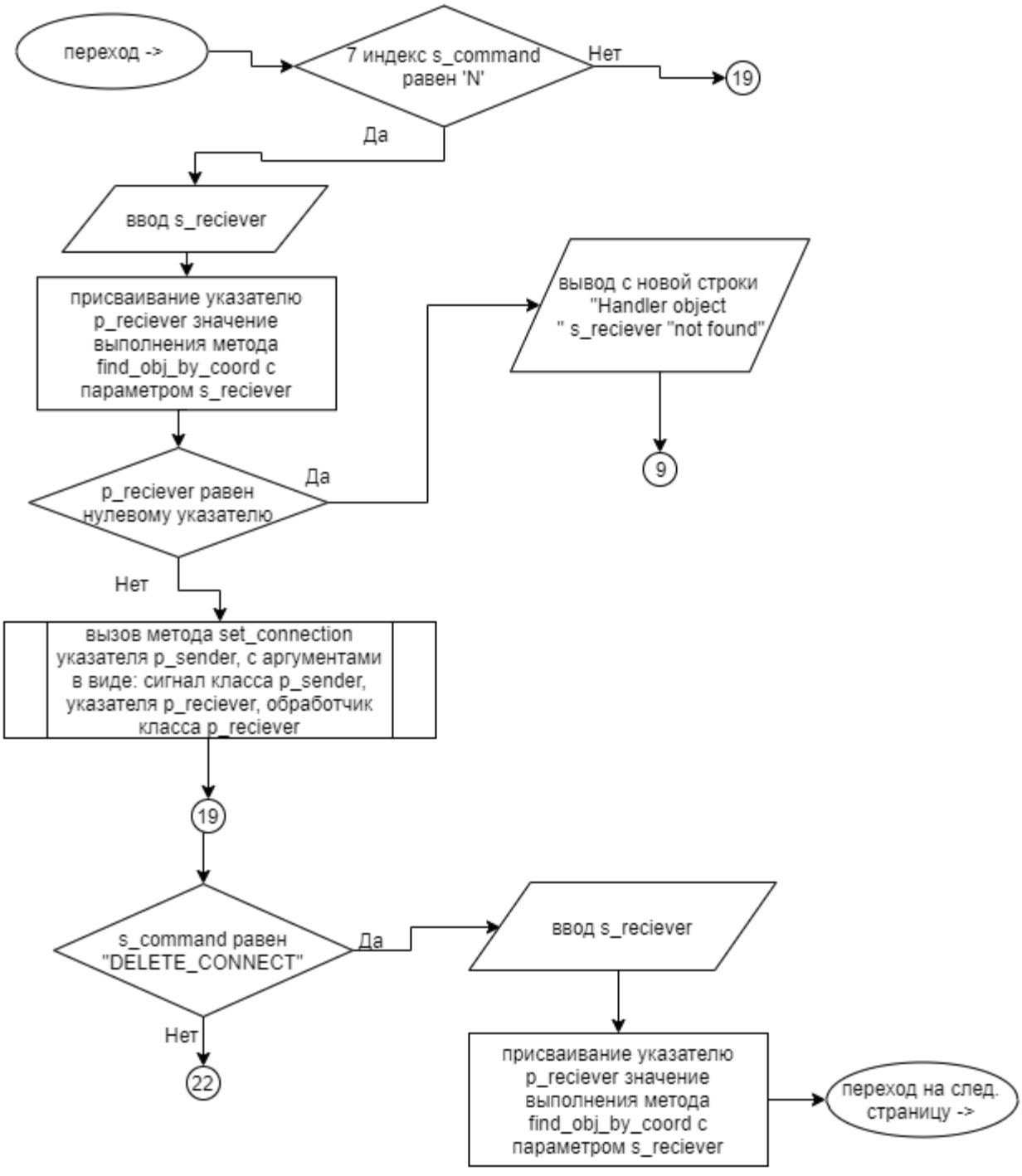


Рисунок 19 – Блок-схема алгоритма


переход ->
7 индекс s_command равен 'N'
Нет
19
Да
ввод s_reciever
присваивание указателю p_reciever значение выполнения метода find_obj_by_coord с параметром s_reciever
вывод с новой строки "Handler object " s_reciever "not found"
9
p_reciever равен нулевому указателю
Да
Нет
вызов метода set_connection указателя p_sender, с аргументами в виде: сигнал класса p_sender, указателя p_reciever, обработчик класса p_reciever
19
s_command равен "DELETE_CONNECT"
Да
ввод s_reciever
Нет
22
присваивание указателю p_reciever значение выполнения метода find_obj_by_coord с параметром s_reciever
переход на след. страницу ->
Рисунок 20 – Блок-схема алгоритма

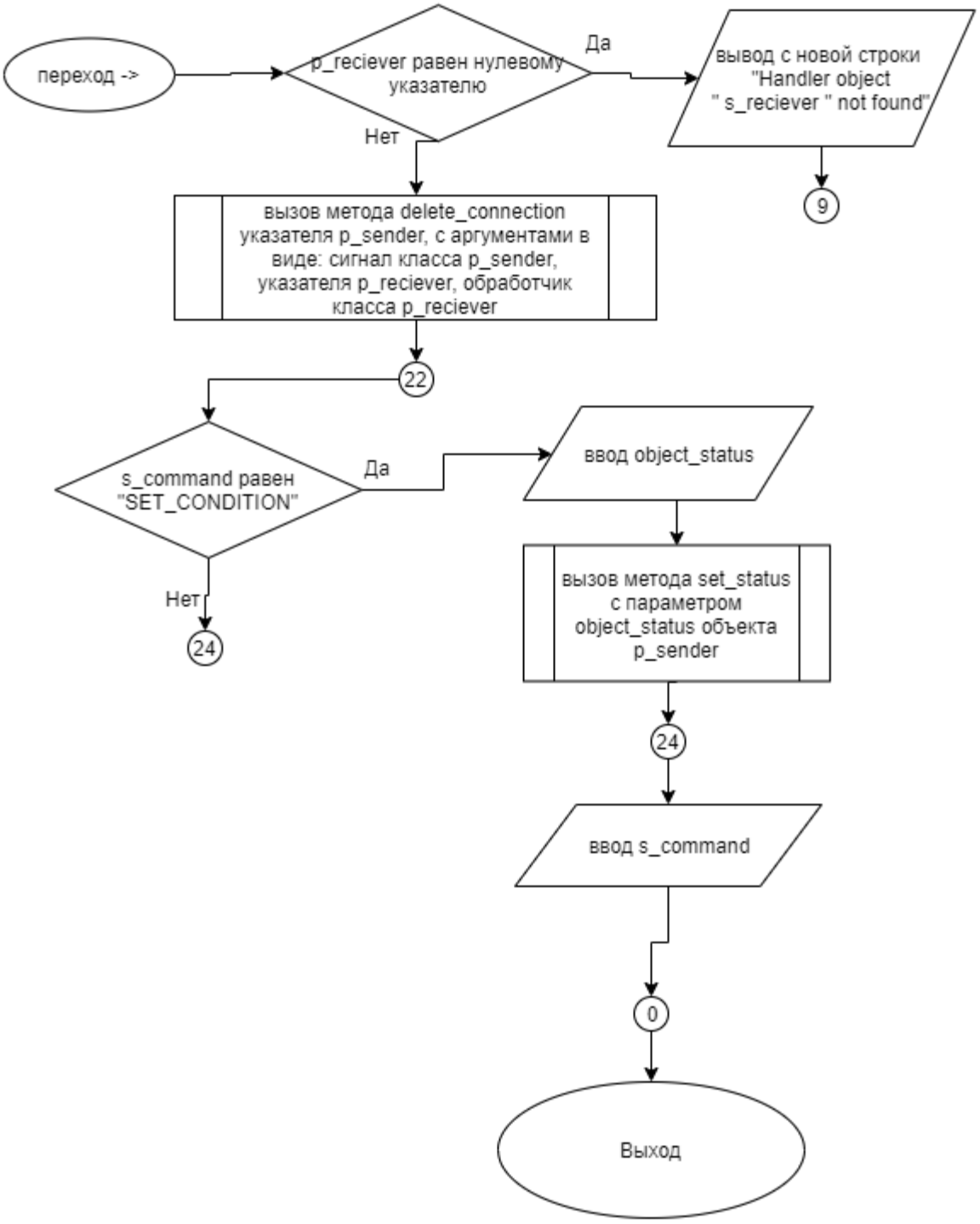


Рисунок 21 – Блок-схема алгоритма

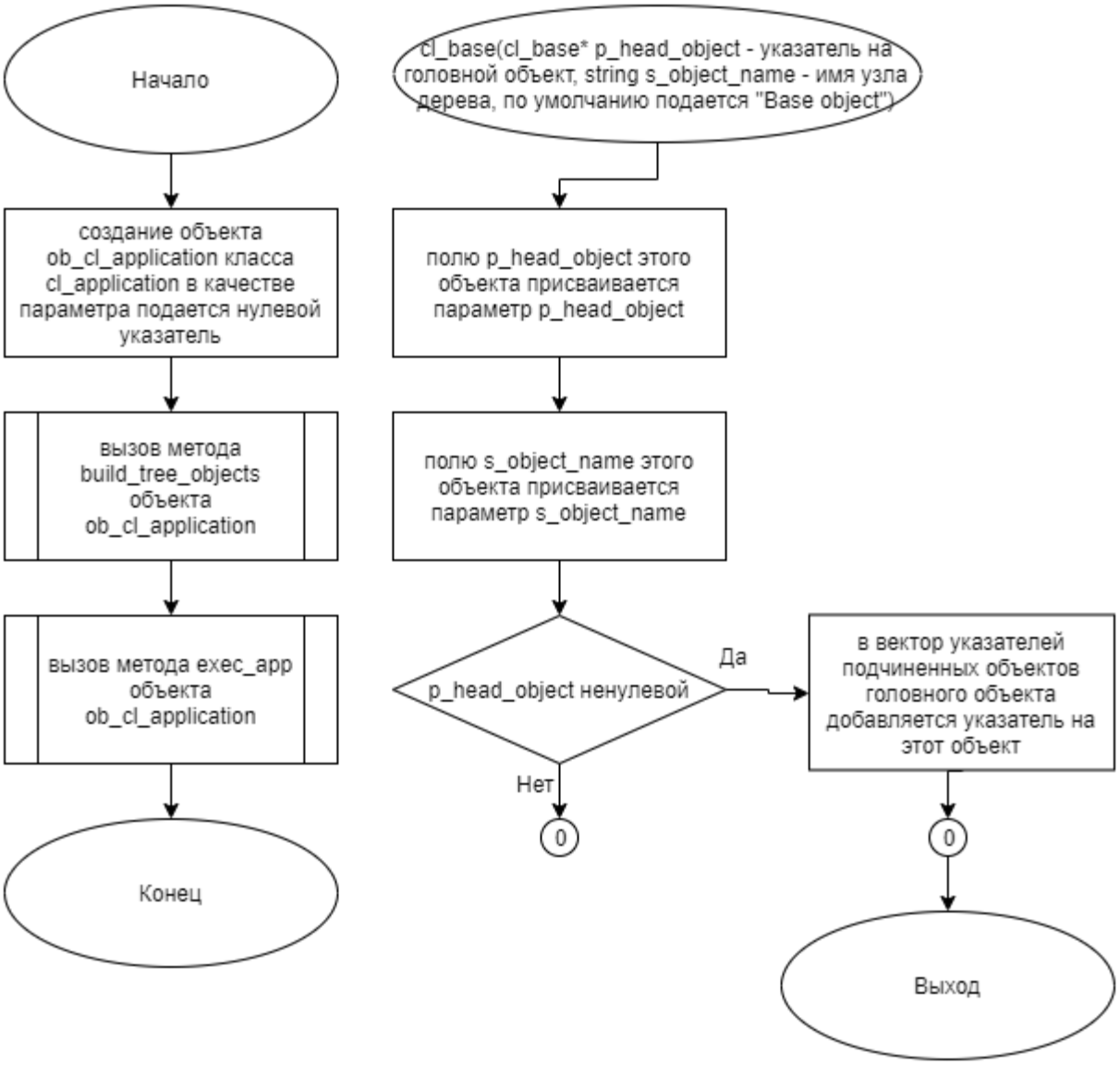


Рисунок 22 – Блок-схема алгоритма

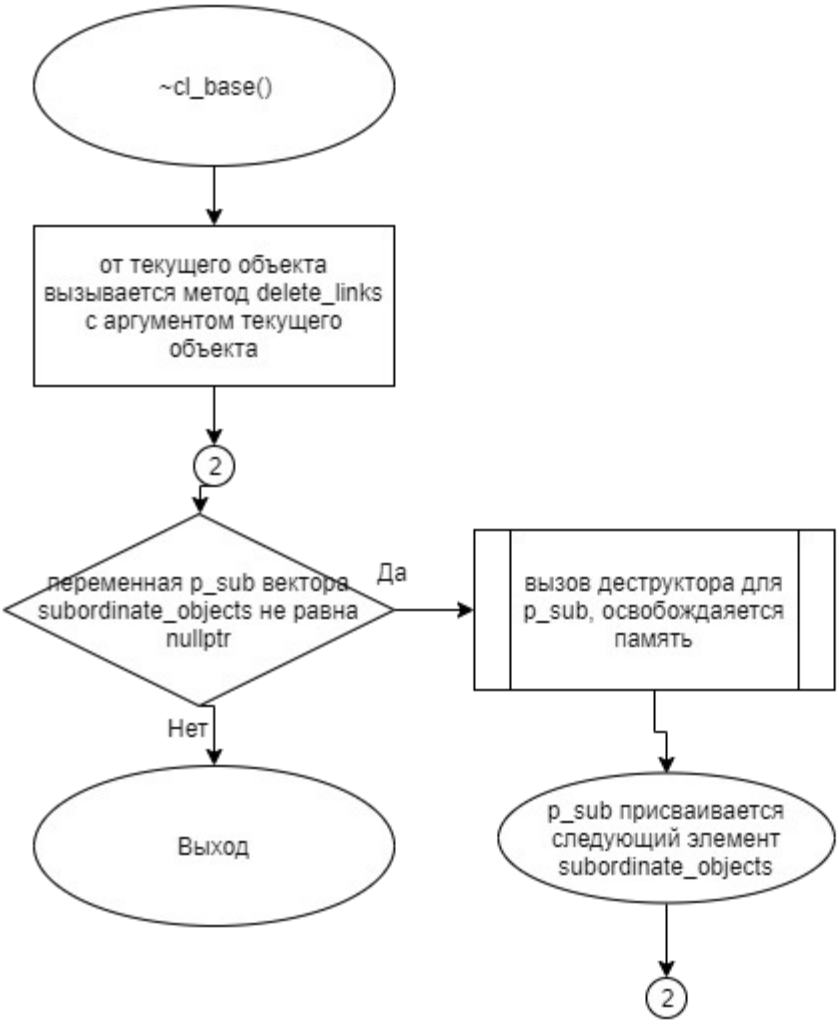


Рисунок 23 – Блок-схема алгоритма

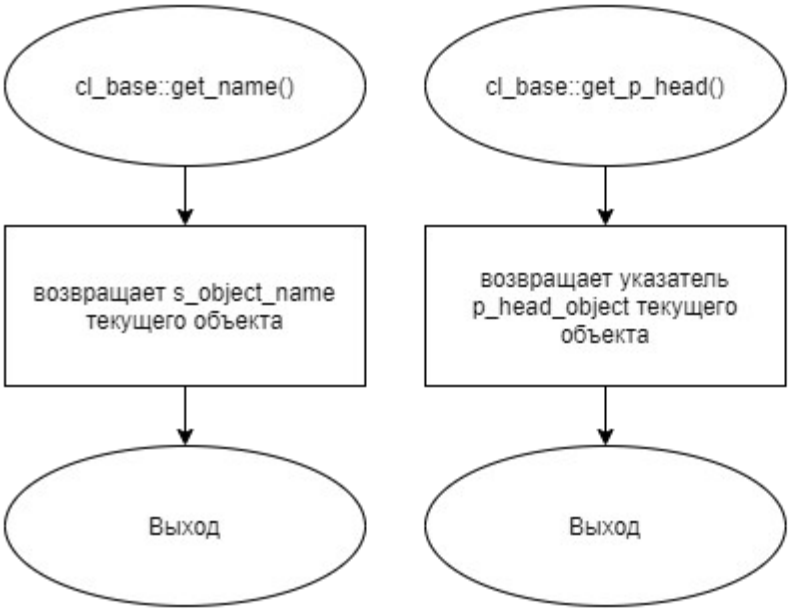


Рисунок 24 – Блок-схема алгоритма

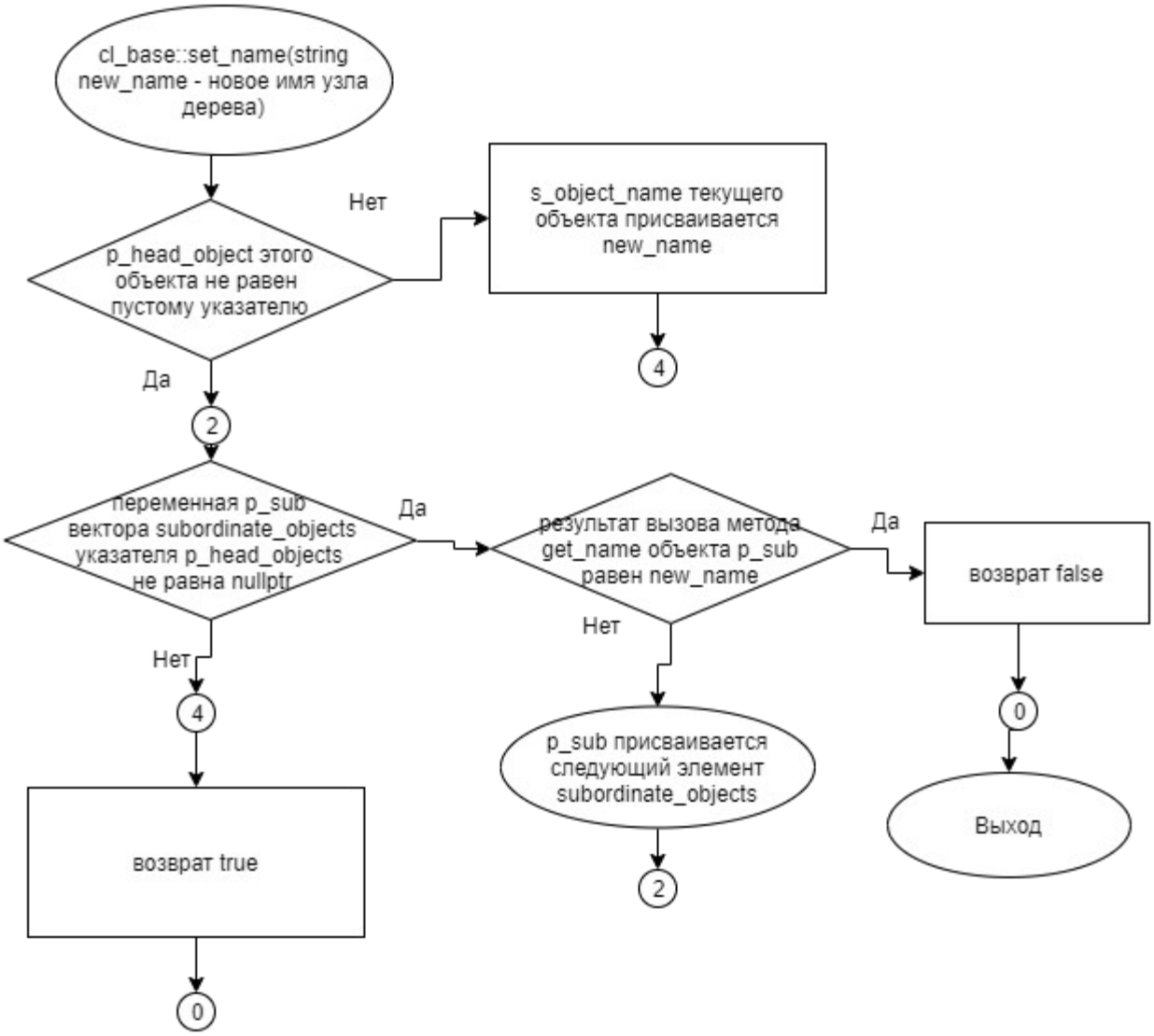


Рисунок 25 – Блок-схема алгоритма

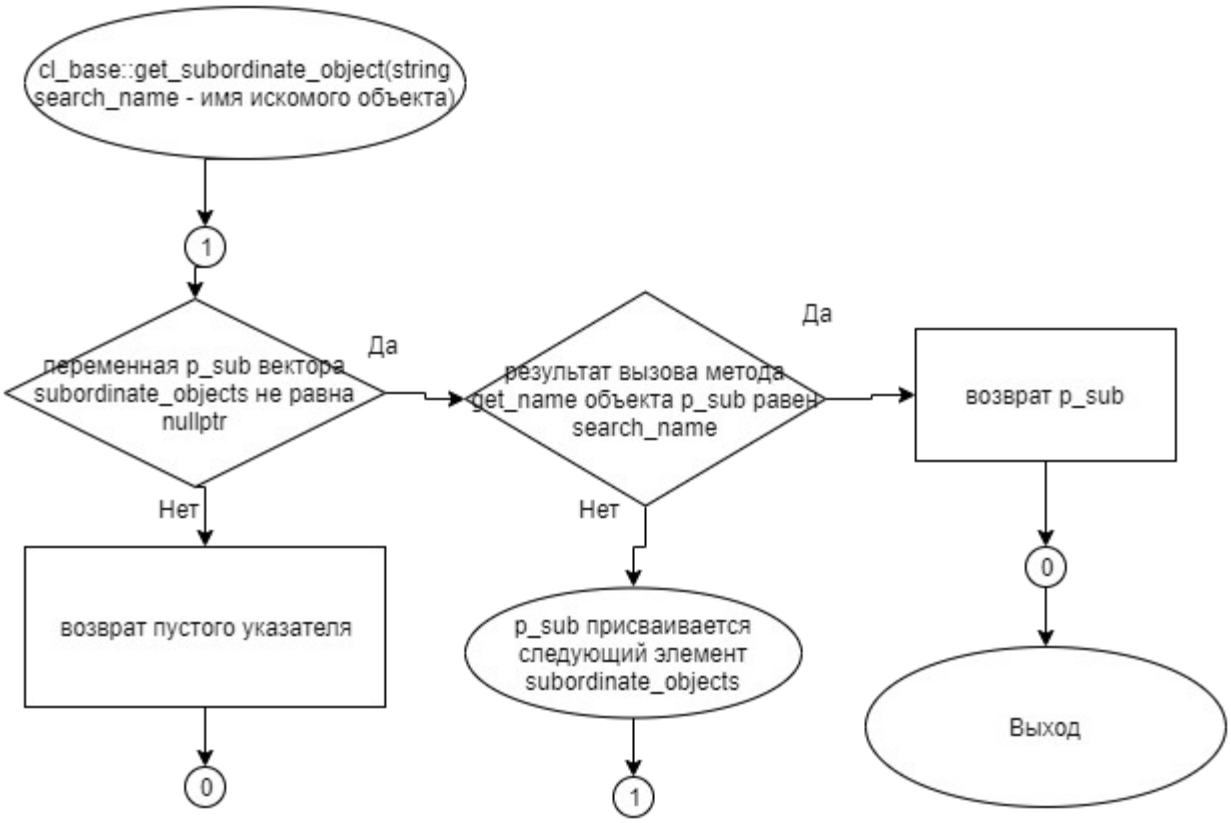


Рисунок 26 – Блок-схема алгоритма

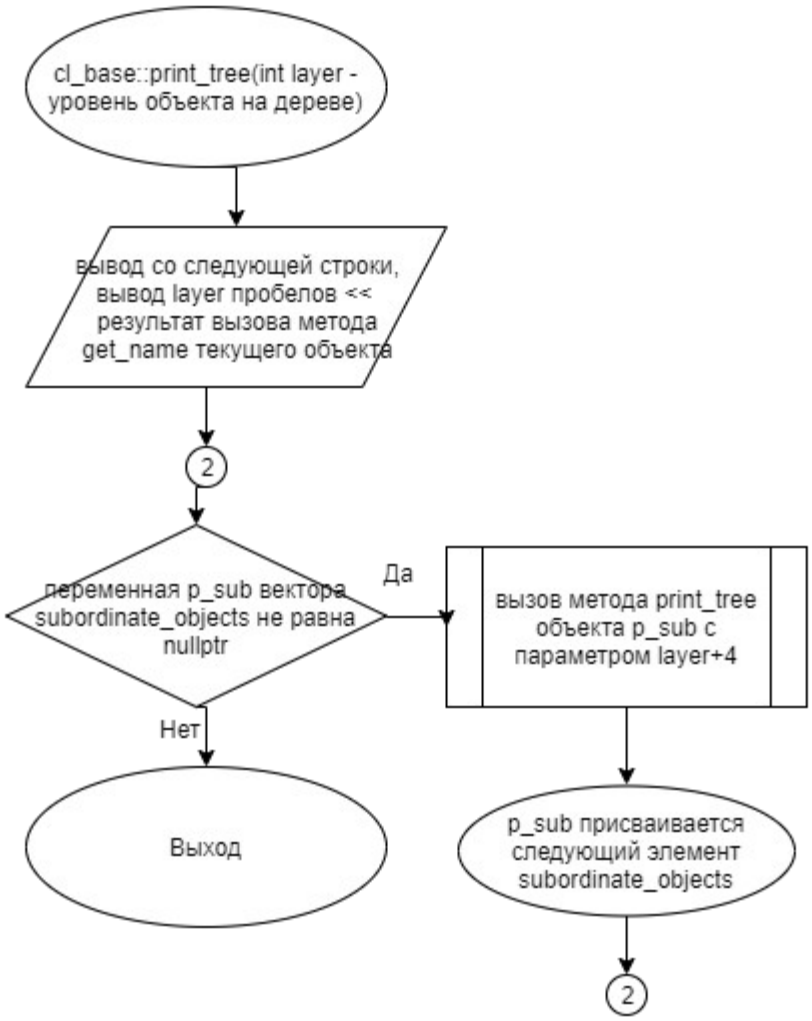


Рисунок 27 – Блок-схема алгоритма

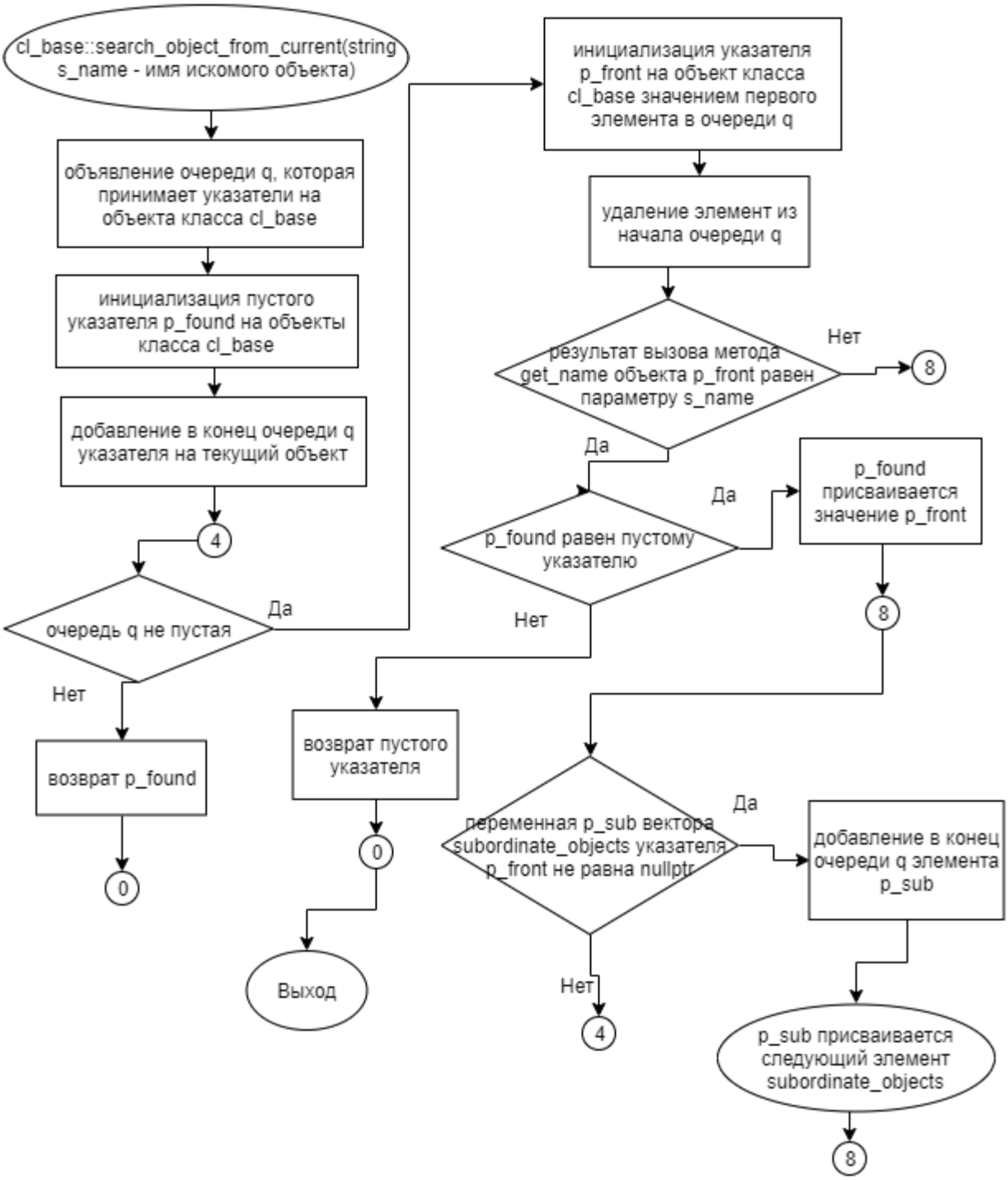


Рисунок 28 – Блок-схема алгоритма

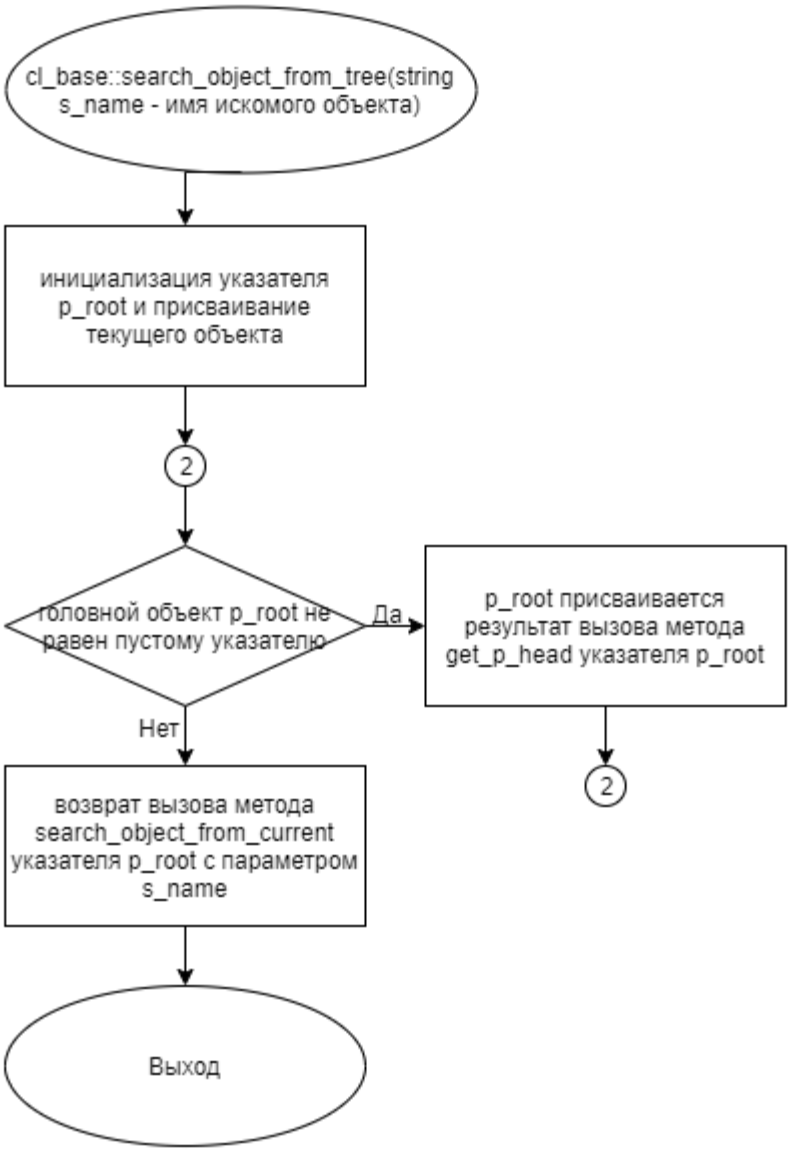


Рисунок 29 – Блок-схема алгоритма

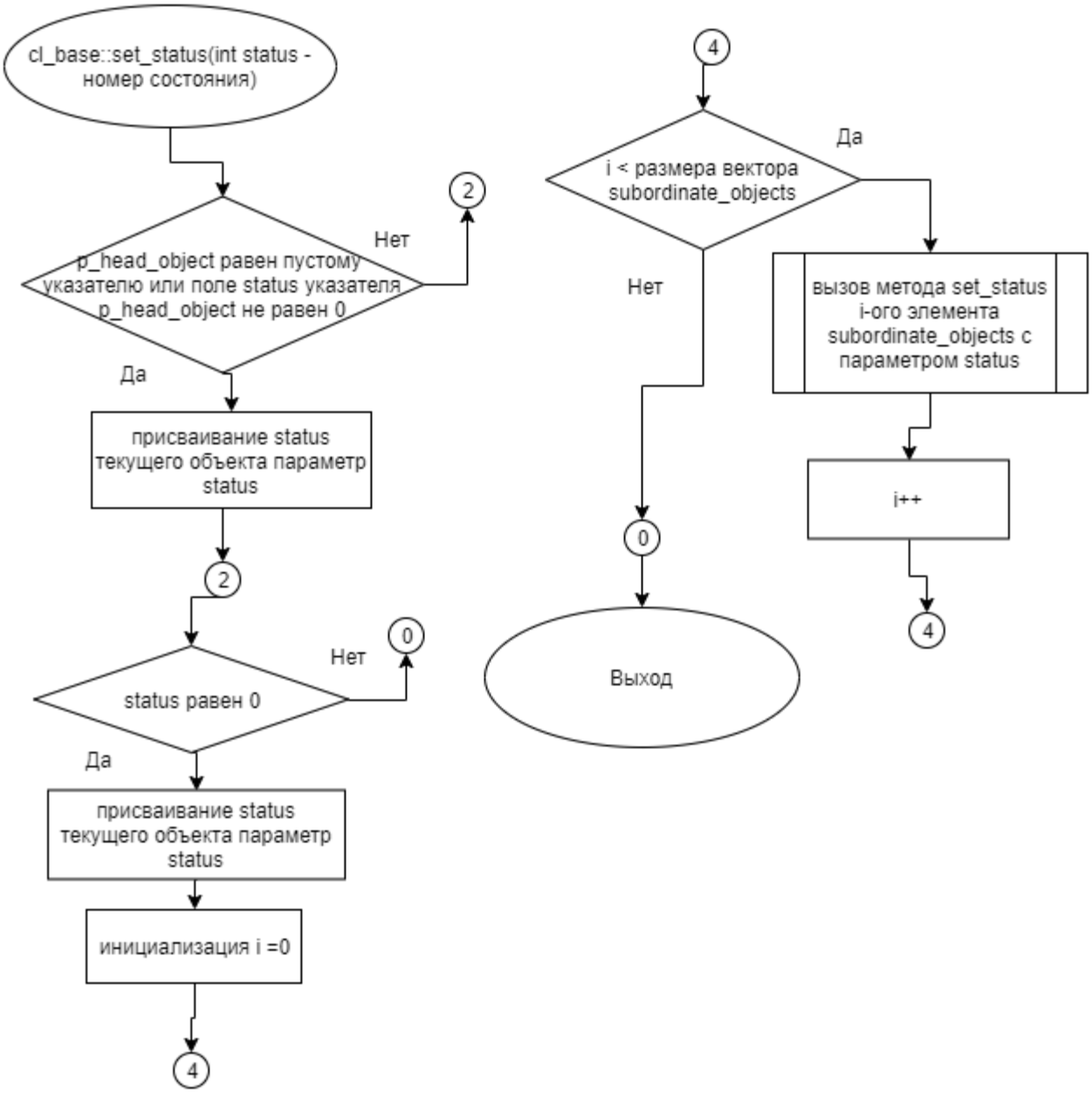


Рисунок 30 – Блок-схема алгоритма

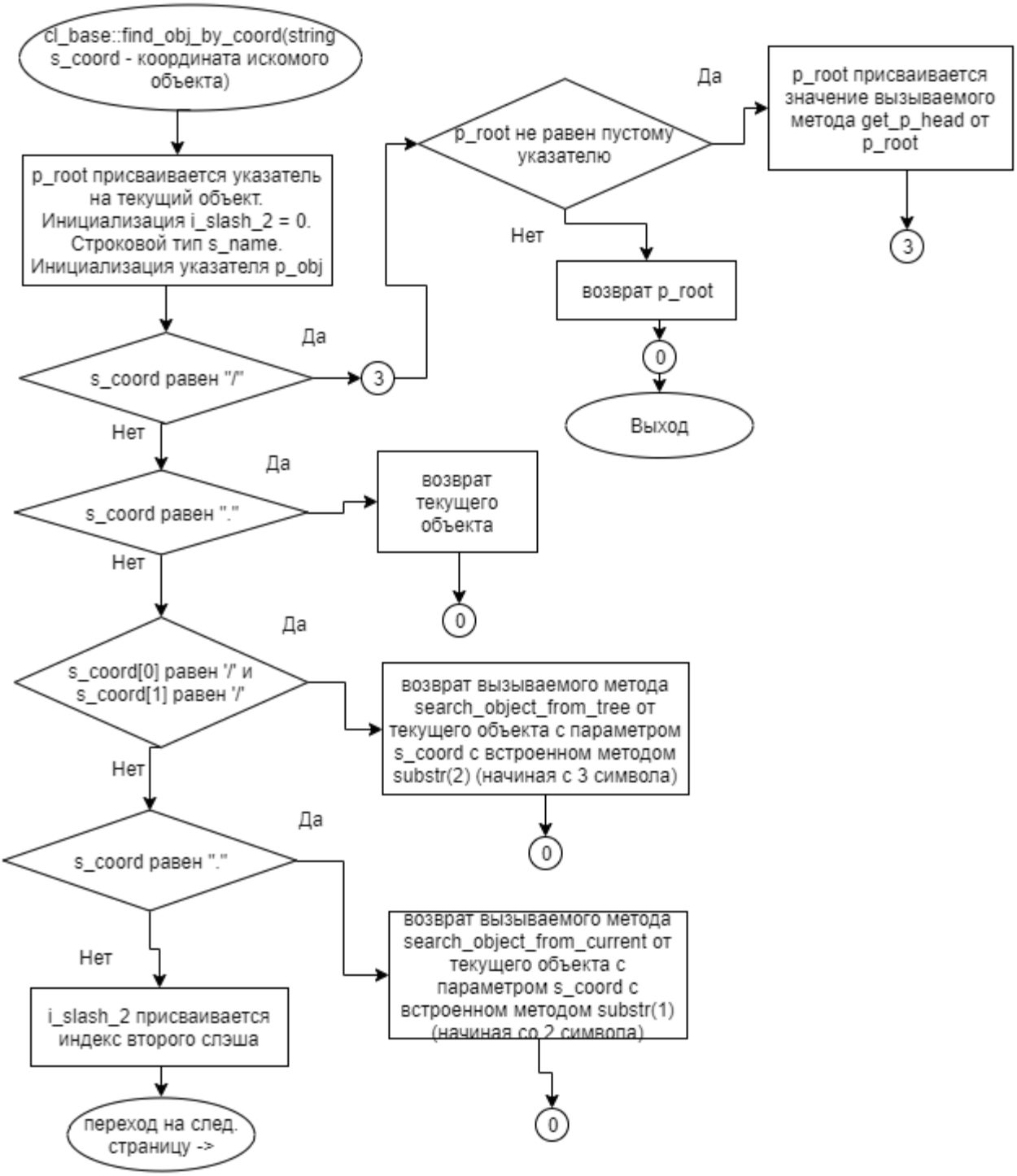


Рисунок 31 – Блок-схема алгоритма

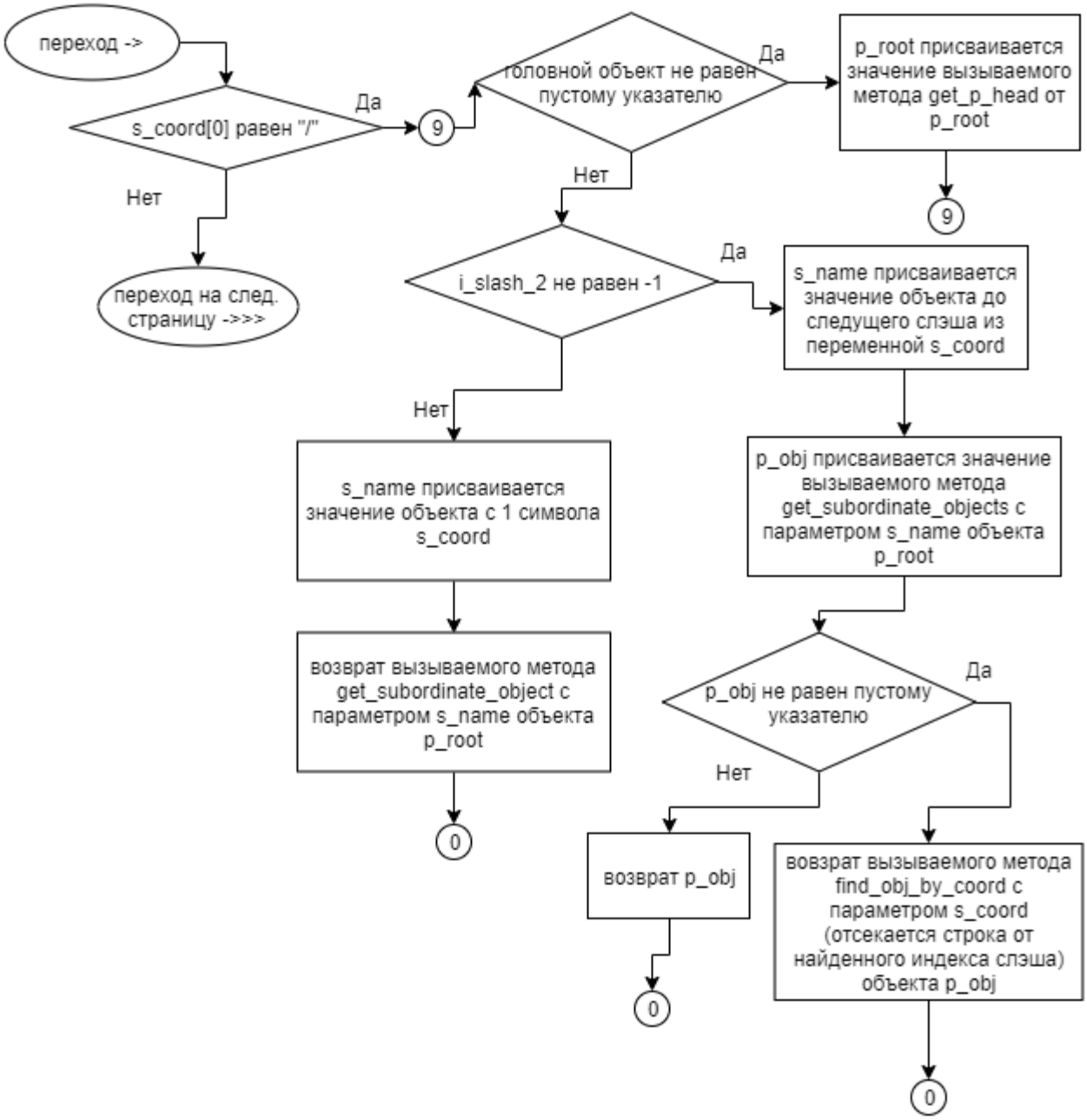


Рисунок 32 – Блок-схема алгоритма

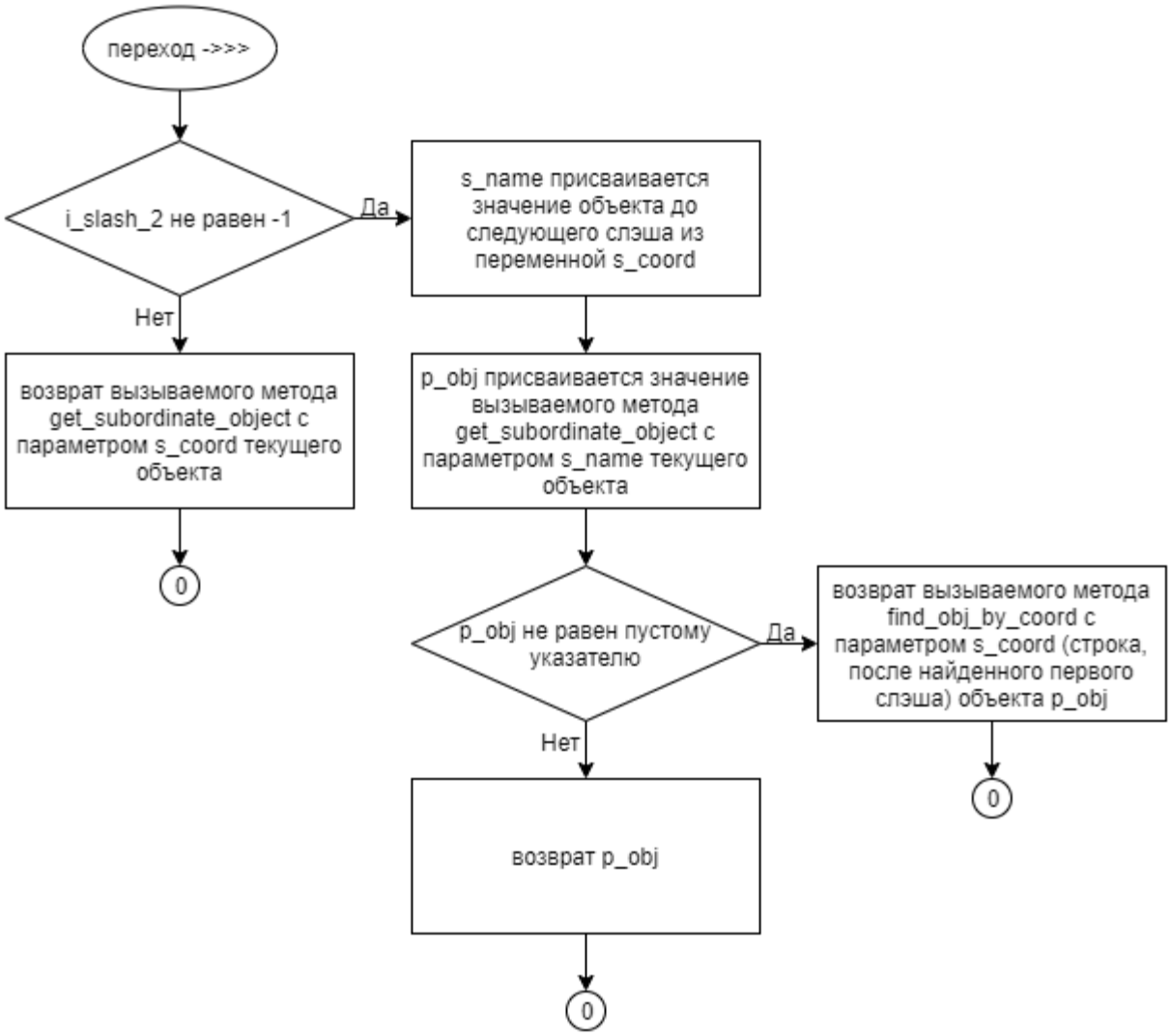


Рисунок 33 – Блок-схема алгоритма

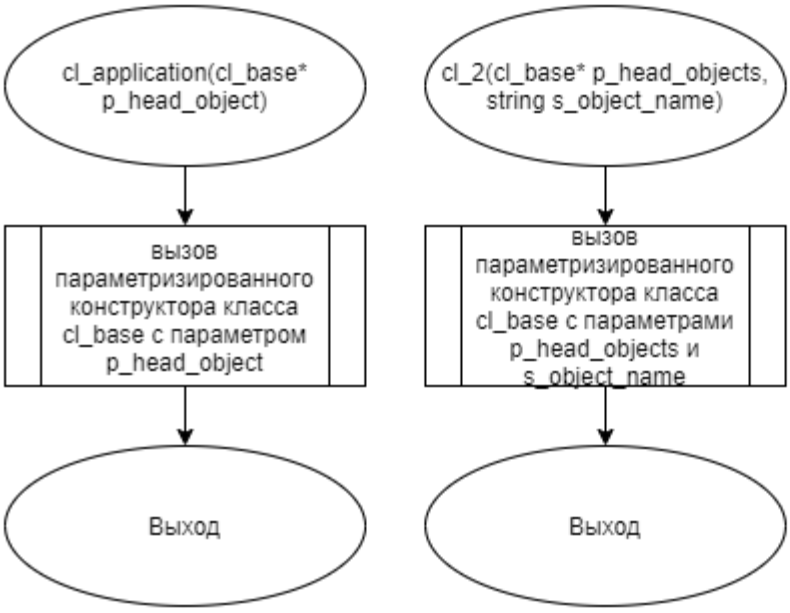


Рисунок 34 – Блок-схема алгоритма

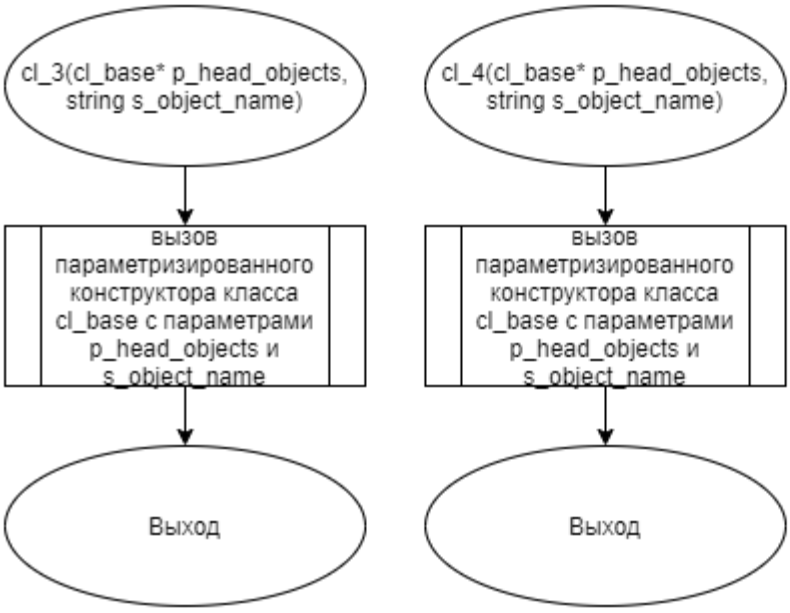


Рисунок 35 – Блок-схема алгоритма

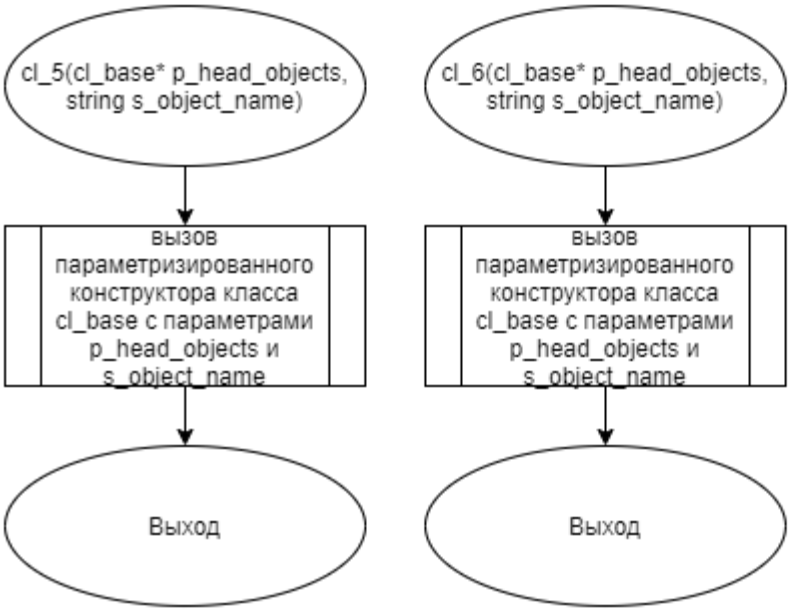


Рисунок 36 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_2::cl_2(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_name){}

int cl_2::get_class_num(){
   return 2;
}

void cl_2::signal_f(string& msg){
   cout << endl << "Signal from " << get_abs_path();
   msg += " (class: 2)";
}
void cl_2::handler_f(string msg){
   cout << endl << "Signal to " << get_abs_path() << " Text: "<< msg;
}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H

#include "cl_base.h"


class cl_2: public cl_base{
public:
   // конструктор, создающий объект класса cl_2
   cl_2(cl_base * p_head_object, string s_object_name);
   void signal_f(string& msg);
```

```
   void handler_f(string msg);
   int get_class_num();
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_3::cl_3(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_name){}

int cl_3::get_class_num(){
   return 3;
}

void cl_3::signal_f(string& msg){
   cout << endl << "Signal from " << get_abs_path();
   msg += " (class: 3)";
}
void cl_3::handler_f(string msg){
   cout << endl << "Signal to " << get_abs_path() << " Text: "<< msg;
}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H

#include "cl_base.h"


class cl_3: public cl_base{
public:
   // конструктор, создающий объект класса cl_3
   cl_3(cl_base * p_head_object, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string msg);
```

```
   int get_class_num();
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_4::cl_4(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_name){}

int cl_4::get_class_num(){
   return 4;
}

void cl_4::signal_f(string& msg){
   cout << endl << "Signal from " << get_abs_path();
   msg += " (class: 4)";
}
void cl_4::handler_f(string msg){
   cout << endl << "Signal to " << get_abs_path() << " Text: "<< msg;
}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H

#include "cl_base.h"


class cl_4: public cl_base{
public:
   // конструктор, создающий объект класса cl_4
   cl_4(cl_base * p_head_object, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string msg);
   int get_class_num();
```

```
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_5::cl_5(cl_base*     p_head_object,     string     s_object_name)     :
cl_base(p_head_object, s_object_name){}

int cl_5::get_class_num(){
   return 5;
}

void cl_5::signal_f(string& msg){
   cout << endl << "Signal from " << get_abs_path();
   msg += " (class: 5)";
}
void cl_5::handler_f(string msg){
   cout << endl << "Signal to " << get_abs_path() << " Text: "<< msg;
}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H

#include "cl_base.h"


class cl_5: public cl_base{
public:
   // конструктор, создающий объект класса cl_5
   cl_5(cl_base * p_head_object, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string msg);
   int get_class_num();
};
```

```
#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_6::cl_6(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_name){}

int cl_6::get_class_num(){
   return 6;
}

void cl_6::signal_f(string& msg){
   cout << endl << "Signal from " << get_abs_path();
   msg += " (class: 6)";
}
void cl_6::handler_f(string msg){
   cout << endl << "Signal to " << get_abs_path() << " Text: "<< msg;
}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H

#include "cl_base.h"


class cl_6: public cl_base{
public:
   // конструктор, создающий объект класса cl_6
   cl_6(cl_base * p_head_object, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string msg);
   int get_class_num();
};
```

```
#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"

// вызов параметризированного конструктора класса cl_base с параметром p_head_object
cl_application::cl_application(cl_base * p_head_object) : cl_base(p_head_object){}

int cl_application::get_class_num(){
   return 1;
}
void cl_application::signal_f(string& msg){
   cout << endl << "Signal from " << get_abs_path();
   msg += " (class: 1)";
}
void cl_application::handler_f(string msg){
   cout << endl << "Signal to " << get_abs_path() << " Text: "<< msg;
}

void cl_application::build_tree_objects(){
   /*
   метод построения дерева иерархии объектов
   */

   cout << "Object tree";
   //-------------------------------------------------------------
   string s_head_name, s_sub_name; // имя головного объекта, подчиненного объекта
   cl_base* p_head = this;
   cl_base* p_sub = nullptr;
   int class_number; // номер класса, номер состояния

   string s_sender, s_reciever; // отправитель, получатель сигнала
   cl_base* p_sender = nullptr;
   cl_base* p_reciever = nullptr;
   //-------------------------------------------------------------

   cin >> s_head_name;
   this -> set_name(s_head_name);// вызов метода set_name этого объекта с параметром s_head_name

   // ввод иерархии объектов
   cin >> s_head_name;
   while(s_head_name != "endtree"){
      cin >> s_sub_name >> class_number;
```

```
      p_head = find_obj_by_coord(s_head_name); //поиск головного объекта

      if (p_head == nullptr){ // если головной объект ненулевой
         print_tree();
         cout << endl << "The head object " << s_head_name << " is not
found";
         exit(1);
      }
      if (p_head->get_subordinate_object(s_sub_name) != nullptr){ // если
подчиненные объекты не равен пустому указателю
         cout << endl << s_head_name << "         Dubbing the names of
subordinate objects";
      }
      else{
         switch(class_number){
            case 1:
               p_sub = new cl_application(p_head);
               break;
            case 2:
               p_sub = new cl_2(p_head, s_sub_name);
               break;
            case 3:
               p_sub = new cl_3(p_head, s_sub_name);
               break;
            case 4:
               p_sub = new cl_4(p_head, s_sub_name);
               break;
            case 5:
               p_sub = new cl_5(p_head, s_sub_name);
               break;
            case 6:
               p_sub = new cl_6(p_head, s_sub_name);
               break;
         }
      }
      cin >> s_head_name;
   }
   // вектор элементов TYPE_SIGNAL хранит сигналы классов 1-6, указатель на
метод signal_f
   vector<TYPE_SIGNAL>   signals   =   {SIGNAL_D(cl_application::signal_f),
SIGNAL_D(cl_2::signal_f),                        SIGNAL_D(cl_3::signal_f),
SIGNAL_D(cl_4::signal_f),                        SIGNAL_D(cl_5::signal_f),
SIGNAL_D(cl_6::signal_f)};
   // вектор элементов TYPE_HANDLER хранит обработчик классов 1-6, указатель
на метод handler_f
   vector<TYPE_HANDLER>  handlers  =  {HANDLER_D(cl_application::handler_f),
HANDLER_D(cl_2::handler_f),                     HANDLER_D(cl_3::handler_f),
HANDLER_D(cl_4::handler_f),                     HANDLER_D(cl_5::handler_f),
HANDLER_D(cl_6::handler_f)};

   // установка соединений
   cin >> s_sender; // получаем путь текущего объекта, от которого будем
отправлять сигнал
   while(s_sender != "end_of_connections"){
      p_sender  =  find_obj_by_coord(s_sender);  //  вызываем  метод  поиска
```

```
объекта и присваиваем полученный путь к указателю

      if (p_sender == nullptr){
         cout << endl << "Object " << s_sender << " not found";
         continue;
      }

      cin >> s_reciever;
      p_reciever = find_obj_by_coord(s_reciever); // вызываем метод поиска
объекта и присваиваем полученный путь к указателю

      if (p_reciever == nullptr){
         cout << endl << "Handler object " << s_reciever << " not found";
         continue;
      }
      // для объекта передающего сигнал вызываем метод установки соединения
      // передаем указатель на метод сигнала на объекте который посылает
сигнал
      // отнимаем 1, из-за индексов в векторе
      p_sender->set_connection(signals[p_sender->get_class_num()    -    1],
p_reciever, handlers[p_reciever->get_class_num() - 1]);

      cin >> s_sender;
   }
}

int cl_application::exec_app(){
   //---------------------------------------
   string s_command, s_coordinate, s_text; // команда, координата, текст
   string s_sender, s_reciever; // отправитель, получатель
   cl_base* p_sender = nullptr;
   cl_base* p_reciever = nullptr;
   int object_status;

   // вектор элементов TYPE_SIGNAL хранит сигналы классов 1-6, указатель на
метод signal_f
   vector<TYPE_SIGNAL>   signals   =   {SIGNAL_D(cl_application::signal_f),
SIGNAL_D(cl_2::signal_f), SIGNAL_D(cl_3::signal_f),
                           SIGNAL_D(cl_4::signal_f),
SIGNAL_D(cl_5::signal_f), SIGNAL_D(cl_6::signal_f)};
   // вектор элементов TYPE_HANDLER хранит обработчик классов 1-6, указатель
на метод handler_f
   vector<TYPE_HANDLER>  handlers  =  {HANDLER_D(cl_application::handler_f),
HANDLER_D(cl_2::handler_f), HANDLER_D(cl_3::handler_f),
                           HANDLER_D(cl_4::handler_f),
HANDLER_D(cl_5::handler_f), HANDLER_D(cl_6::handler_f)};
   //---------------------------------------
   this->set_status_tree(1); // установка состояний готовности для всех
объектов

   this->print_tree();

   // цикл обработки комманд
   cin >> s_command;
   while(s_command != "END"){
```

```
      cin >> s_sender;
      p_sender  =  find_obj_by_coord(s_sender);  //  находим  отправителя,  с
помощью метода поиска
      if (p_sender == nullptr){
         cout << endl << "Object " << s_sender << " not found";
         cin >> s_sender;
         continue;
      }
      // |E|MIT
      if (s_command == "EMIT"){
         getline(cin, s_text);
         // передаем номер сигнала отправителя и текст
         p_sender->emit_signal(signals[p_sender->get_class_num()     -     1],
s_text);
      }
      // SET_CON|N|ECT
      else if (s_command[7] == 'N'){
         cin >> s_reciever;
         p_reciever = find_obj_by_coord(s_reciever); // находим получателя, с
помощью метода поиска

         if (p_reciever == nullptr){
            cout << endl << "Handler object " << s_reciever << " not found";
            continue;
         }
         else{
            // вызываем метод установки состояния
            p_sender->set_connection(signals[p_sender->get_class_num()  -  1],
p_reciever, handlers[p_reciever->get_class_num() - 1]);
         }
      }
      // |D|ELETE_CONNECT
      else if (s_command == "DELETE_CONNECT"){
         cin >> s_reciever;
         p_reciever = find_obj_by_coord(s_reciever); // находим получателя, с
помощью метода поиска

         if (p_reciever == nullptr){
            cout << endl << "Handler object " << s_reciever << " not found";
            continue;
         }
         else{
            // вызываем метод удаления состояния
            p_sender->delete_connection(signals[p_sender->get_class_num()    -
1], p_reciever,
                                        handlers[p_reciever->get_class_num()      -
1]);
         }
      }
      // SET_CON|D|ITION
      else if (s_command == "SET_CONDITION"){
         cin >> object_status;
         p_sender-> set_status(object_status);
      }
      cin >> s_command;
```

```
    }
    return 0;
}
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H

#include "cl_base.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"


class cl_application : public cl_base{
public:
   cl_application(cl_base * p_head_object); // конструктор, создающий объект
класса cl_app..
   int exec_app(); // метод запуска приложения
   void build_tree_objects(); // метод, создающий иерархию объекта

   void signal_f(string& msg);
   void handler_f(string msg);
   int get_class_num();
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base * p_head_object, string s_object_name)
: p_head_object(p_head_object) , s_object_name(s_object_name){
   /*
   параметризированный конструктор
   p_head_object - указатель на головной объект
   s_object_name - имя узла дерева
```

```
   */

   //полю p_head_object этого объекта присваивается параметр p_head_object
   //полю s_object_name этого объекта присваивается параметр s_object_name

   // если p_head_object ненулевой, то в subordinate_objects добавляется
указатель на этот объект
   if (p_head_object){
      p_head_object -> subordinate_objects.push_back(this);
   }
}
cl_base::~cl_base(){
   this->delete_links(this); // удаляем соединения
   // проходимся по каждому элементу subordinate_objects и удаляем его
   for  (auto p_sub: subordinate_objects){
      delete p_sub;
   }
}

bool cl_base::set_name(string new_name){
   /*
   метод редактирования имени объекта
   new_name - новое имя узла дерева
   */
   // проходимся по каждому элементу вектора указателей subordinate_objects
объекта по указателю p_head_object
   // если он равен new_name, то возвращаем false
   if (p_head_object != nullptr){
      for (auto p_sub: p_head_object -> subordinate_objects){
         if (p_sub-> get_name() == new_name){
            return false;
         }
      }
   }
   // полю s_object_name этого объекта присваивается new_name
   this -> s_object_name = new_name;
   return true;
}

string cl_base::get_name(){
   // возвращаем s_object_name
   return this->s_object_name;
}

cl_base * cl_base::get_p_head(){
   // возвращаем p_head_object
   return this->p_head_object;
}

cl_base * cl_base::get_subordinate_object(string search_name){
   /*
   получение указателя на непосредственно подчиненный объект по имени
   search_name - имя искомого объекта
   */
```

```
   // проходимся по элементам subordinate_objects, если он равен search_name
   // возвращаем i-ый subordinate_objects
   for (auto p_sub: subordinate_objects){
      if (p_sub -> get_name() == search_name){
         return p_sub;
      }
   }
   // иначе возвращается нулевой указатель
   return nullptr;
}



cl_base* cl_base::search_object_from_current(string s_name){
   /*
   метод поиска объекта по имене в поддереве (в ветке) (обход графа в ширину)
   s_name - имя искомого объекта
   */
   cl_base* p_found = nullptr; // указатель на объект, который был найден
   queue<cl_base*> q; // очередь элементов

   q.push(this); // добавляем в очередь текущий элемент

   // пока очередь не пустая
   while(!q.empty()){
      cl_base* p_front = q.front(); // хранится указатель на наш объект

      q.pop(); // удаляем элемент из начала очереди, чтобы пройтись по всем
элементам
      if (p_front -> get_name() == s_name){ // если имя указателя на элемент
в очереди совпадает с имкомым объектом
         if (p_found     ==  nullptr)  //  указатель  не  пустой  и  объект  не
уникальный
            p_found = p_front; // присваиваем указатель на элемент в очереди
         else // нашли дубликат
            return nullptr;
      }
      // добавляем дочерние элементы p_front в очередь
      for (auto p_sub : p_front->subordinate_objects){
         q.push(p_sub);
      }
   }
   return p_found;
}

cl_base* cl_base::search_object_from_tree(string s_name){
   /*
   поиск объекта по имене во всем дереве
   s_name - имя искомого объекта
   */

   cl_base* p_root = this;
   // пока вышестоящий объект в дереве не пустой, поднимаемся по дереву
   while (p_root-> get_p_head() != nullptr){
      p_root = p_root-> get_p_head();
```

```
        }

        return p_root -> search_object_from_current(s_name);
    }
    void cl_base::print_tree(int layer){
        /*
        метод вывода иерархии объектов (дерева/ветки) от текущего объекта
        layer - уровень на дереве иерархии
        */

        // выводим layer-ое кол-во ' ' и имя объекта
        cout << endl << string(layer, ' ') << this->get_name();
        // проходимся по элемента subordinate_objects и вызываем рекурсию
        for (auto p_sub : subordinate_objects)
            p_sub->print_tree(layer + 4);

    }

    void cl_base::set_status(int status){
        /*
        метод установки статуса объекта
        status - номер состояния
        */

        // если значение status ненулевое у головного объекта
        if (p_head_object == nullptr || p_head_object -> status != 0){
            this -> status = status;
        }
        if (status == 0){
            this ->status = status;
            for (int i = 0; i < subordinate_objects.size(); i++){
                subordinate_objects[i] -> set_status(status);
            }
        }
    }

    int cl_base::get_status(){
        return this->status;
    }

    cl_base* cl_base::find_obj_by_coord(string s_coord){
        /*
        метод поиска объекта по координате
        s_coord - координата искомого объекта
        */

        cl_base* p_root = this; // указатель на текущий объект
        int i_slash_2 = 0; // хранит индекс 2-ого слэша
        string s_name = "";
        cl_base* p_obj;

        if (s_coord == "/"){
            //поднимаемся по корню дереву
            while(p_root -> get_p_head() != nullptr){ // пока головной объект не
    равен пустому указателю
```

```
            p_root = p_root -> get_p_head();
         }
         return p_root; // нашли корневой объект
      }
      if (s_coord == "."){
         return this; // вовзращаем текущий объект, т.к. считаем, что метод
вызван от текущего объекта
      }
      if (s_coord[0] == '/' && s_coord[1] == '/'){ // второй символ строки равен
/ (/ '/' )
         return  this->search_object_from_tree(s_coord.substr(2));  //  текущему
объекту вызывваем метод с парам. s_coord (начиная с 3 символа)
      }
      if (s_coord[0] == '.'){
         return this->search_object_from_current(s_coord.substr(1)); // текущему
объекту вызывваем метод с парам. s_coord (начиная со 2 символа)
      }

      i_slash_2 = s_coord.find("/",1);
      if (s_coord[0] == '/'){
         //поднимаемся по корню дереву
         while(p_root -> get_p_head() != nullptr){ // пока головной объект не
равен пустому указателю
            p_root = p_root -> get_p_head();
         }

         if (i_slash_2 != -1){ // нашли индекс слэша
            s_name = s_coord.substr(1, i_slash_2 - 1); // получили имя объекта,
после слэша и до след. слэша
            p_obj  =  p_root  ->  get_subordinate_object(s_name);  //  корневому
объекту вызываем метод поиска подчиненных объектов с парам. s_name
            if (p_obj != nullptr) // если p_obj не равен нулевому указателю
               return p_obj->find_obj_by_coord(s_coord.substr(i_slash_2 + 1));
            else
               return p_obj;
         }
         else{
            s_name = s_coord.substr(1);
            return  p_root->get_subordinate_object(s_name);  //  от  корневого
объекта ищем подчиненные
         }
      } // ob1/ob2/ob3
      else{
         if (i_slash_2 != -1){ // если слэш был найден в s_coord
            s_name = s_coord.substr(0, i_slash_2); // получили имя объекта
            p_obj = this -> get_subordinate_object(s_name); // текущему объекту
вызываем метод с парам. s_name
            if (p_obj != nullptr) // если p_obj не равен нулевому указателю
               return    p_obj->find_obj_by_coord(s_coord.substr(i_slash_2    +
1)); // рекурсивно вызываем этот метод, передавая строку, после / в s_coord
            else
               return p_obj;
         }
         else{ // второго слэша не нашлось (ob3)
            return this->get_subordinate_object(s_coord); // от текущего объекта
```

```
ищем подчиненные
      }
   }

   return nullptr;
}

void   cl_base::set_connection(TYPE_SIGNAL   p_signal,   cl_base*   p_target,
TYPE_HANDLER p_handler){
   // новое соединение устанавливает

   o_sh* p_value; // новый указатель на элемент структуры
   // проверка в векторе соединений для исключения повторного установления
связи
   for (auto c: connects){
      if (c->p_signal == p_signal && c->p_target == p_target && c->p_handler
== p_handler)
         return; // выходим из метода, т.к есть такое соединение
   }
   p_value = new o_sh(); // вызываем структуру по умолчанию, создаст новый
экземпляр структуры

   // присваивание переменных
   p_value->p_signal = p_signal;
   p_value->p_target = p_target;
   p_value->p_handler = p_handler;

   connects.push_back(p_value); // добавляем p_value в вектор
}
void  cl_base::delete_connection(TYPE_SIGNAL  p_signal,  cl_base*  p_target,
TYPE_HANDLER p_handler){

   vector<o_sh*>::iterator p_it; // объявляем p_it типа итератор по элементам
вектора o_sh
   // проходимся по элементам connects до (указателя на следующий элемент
после последнего)
   for (p_it = connects.begin(); p_it != connects.end(); p_it++)
      // p_it (просто указатель) (*p_it) - получаем элемент, на который он
указывает (разыменовали итератор)
      if ( (*p_it) -> p_signal == p_signal && (*p_it) -> p_target == p_target
&& (*p_it) -> p_handler == p_handler){
         // удаляем соединение
         delete *p_it;
         p_it = connects.erase(p_it);
         p_it--;
      }
}
void cl_base::emit_signal(TYPE_SIGNAL p_signal, string s_msg){
   if (this->get_status() == 0)
      return;

   (this->*p_signal)(s_msg); // от текущего объекта вызывается разыменованный
метод p_signal в качестве аргумента получает s_msg
   //цикл по всем соединениям
   for (auto c: connects){
```

```
      if (c->p_signal == p_signal){
         cl_base* p_target = c->p_target; // объявляем p_target и сохраняем
целевой объект из connects
         TYPE_HANDLER p_handler = c->p_handler; // объявляем p_handler и
значение p_handler из connects
         // проверка целевого объекта на готовность
         if(p_target->get_status() != 0)
            (p_target->*p_handler)(s_msg);  //  объекту  p_target  вызываем
разыменованный метод p_handler в качестве аргумента получает s_msg
      }
   }
}

string cl_base::get_abs_path(){
   string s_abs_path = "";
   cl_base* p_obj = this; // присваиваем указатель на текущий объект

   // пока головной объект не равен пустому указателю
   while (p_obj->get_p_head() != nullptr){
      s_abs_path = "/" + p_obj->get_name() + s_abs_path; // записываем путь
текущего объекта
      p_obj = p_obj->get_p_head(); // вызов геттера головного объекта
   }
   if (s_abs_path == "")
      s_abs_path = "/";

   return s_abs_path;
}

void cl_base::delete_links(cl_base* p_target){
   // метод удаляет связяи, идущие к p_target - целевой объект
   // из всех других элементов дерева

   // для текущего объекта удаляем соединения
   for (int i = 0; i < this->connects.size(); i++){
      if (this->connects[i]->p_target == p_target){
         delete this->connects[i];
         this->connects.erase(connects.begin()+1);
         i--;
      }
   }
   // для всех векторов подчиненных вызываем этот метод
   for(auto sub: subordinate_objects){
      sub->delete_links(p_target);
   }
}
void cl_base::set_status_tree(int status){
   if (get_p_head() != nullptr && get_p_head()-> status == 0)
      return;
   set_status(status);
   for (auto sub: subordinate_objects){
      sub->set_status_tree(status);
   }
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H

#include <string>
#include <vector>
#include <iostream>
#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 msg);

struct o_sh{
   TYPE_SIGNAL p_signal; // указатель на метод сигнала
   cl_base* p_target; // указатель на целевой объект
   TYPE_HANDLER p_handler; // указатель на метод обработчик
};

class cl_base {
private:
   string s_object_name; // имя объекта
   cl_base * p_head_object; // указатель на родительский объект
   vector <cl_base *> subordinate_objects; // вектор подчиненных объектов
   int status = 0; // статус состояния объекта
   vector <o_sh*> connects; // вектор соединений
public:
   cl_base(cl_base * p_head_object, string s_object_name = "Base object"); //
параметризированный конструктор
   string get_name(); // метод получения имени
   cl_base * get_p_head(); // метод получения указателя на родительский
объект
   bool set_name(string new_name); //метод редактирования имени объекта
   cl_base * get_subordinate_object(string search_name); // получение
указателя на непосредственно подчиненный объект по имени
   ~cl_base(); // деструктор

   cl_base* search_object_from_current(string); // поиск объекта на ветке
иерархии от текущего по имени
   cl_base* search_object_from_tree(string); // поиск по дереву (в корне)
иерархии
   void set_status(int status); // метод установки статуса объекта
   void print_tree(int layer = 0); // метод вывода дерева иерархии

   cl_base* find_obj_by_coord(string); // метод поиска объекта по заданной
координате
```

```
   void set_connection(TYPE_SIGNAL p_signal, cl_base* p_target, TYPE_HANDLER
p_handler); // метод установки соединений между объектов
   void    delete_connection(TYPE_SIGNAL    p_signal,    cl_base*    p_target,
TYPE_HANDLER p_handler); // метод удаления соединения
   void emit_signal(TYPE_SIGNAL p_signal, string s_msg); // метод проверки
соединения
   int get_status(); // метод получения статуса
   string get_abs_path(); // метод получения абсолютного пути объекта
   virtual int get_class_num() = 0; // виртуальная функция. если равно 0, то
нет тела у метода, метод реализован в производных классах
   void delete_links(cl_base* p_target); // метод удаляет связи, идущие к
p_target - целевой объект
   void  set_status_tree(int  status);  //  метод  установки  статуса  у  всех
объектов в дереве
};



#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_application.h"

int main()
{
   cl_application ob_cl_application(nullptr); // создание корневого объекта
   ob_cl_application.build_tree_objects();   //   конструирование   системы,
построение дерева иерархии

   return (ob_cl_application.exec_app()); // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 47.

Таблица 47 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 EMIT /object_s2/object_s4 Send message 2 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 4 END	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6	Object tree appls_root object_s1 object_s7 object_s2 object_s4	Object tree appls_root object_s1 object_s7 object_s2 object_s4

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 EMIT /object_s2/object_s4 Send message 2 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 4 SET_CONNECT /object_s1 /object_s2/object_s6 SET_CONNECT /object_s1 /object_s13 EMIT /object_s1 Send message 4 END	object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3) Signal to /object_s13 Text: Send message 4 (class: 3)	object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3) Signal to /object_s13 Text: Send message 4 (class: 3)
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 6 /object_s1 object_s7 2	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 EMIT /object_s2/object_s4 Send message 2 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 4 DELETE_CONNECT /object_s2/object_s4 / EMIT /object_s2/object_s4 Send message 2 SET_CONDITION /object_s2/object_s4 0 EMIT /object_s2/object_s4 Send message 3 SET_CONNECT /object_s1 /object_s2/object_s6 EMIT /object_s1 Send message 4 SET_CONDITION /object_s1/object_s7 0 EMIT /object_s2 Send message 12 END	/object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3) Signal from /object_s2	/object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3) Signal from /object_s2

ЗАКЛЮЧЕНИЕ

Поставленная задача по созданию программного обеспечения с функционалом построения и управления деревом была реализована в полном объёме. Полученный результат работы показывает простое дерево иерархии и лимитированное управление деревом иерархии. Управление деревом включает в себя: установку объекта на дереве, установка связей между объектами, создание дерева иерархии, смена имени объекта на дереве, отправление и получение сигналов среди объектов по дереву иерархии, удаление объектов на дереве иерархии.

Объектно-ориентированное программирование полезно в сфере программирования своим функционалом. На объектно-ориентированных языках программирования можно долго поддерживать код. В объектно-ориентированных языках программирования легче реализована модульность, организация программы на совокупность небольших блоков. Модернизация программы является ключом успешной программы и в сфере ООП обновление продукта упрощенна. ООП объединяет данные и связанное с ними поведение в объект, что помогает программистам легче понять, как работает программа.

Программное обеспечение АСО Avrora, выполненное инженерами Федерального государственного бюджетного образовательного учреждения высшего образования "МИРЭА - Российский технологический университет", упрощает выполнение практических и курсовых работ для студентов.

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).