Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

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

Класс chain:

- свойства/поля:
    - поле указатель на метод сигнала:
        - наименование — p_signal;
        - тип — ptr;
        - модификатор доступа — public;
    - поле указатель на метод обработчика:
        - наименование — p_handler;
        - тип — ptr;
        - модификатор доступа — public;
    - поле указатель на целевой объект:
        - наименование — p_destination;
        - тип — ptr;
        - модификатор доступа — public;

Класс cl_base:

- свойства/поля:
    - поле вектор для хранения связей типа "сигнал - обработчик":
        - наименование — chains_of_objects;
        - тип — vector;
        - модификатор доступа — private;
- функционал:
    - метод emit_signal — вызывает сигнал текущего объекта, передает сообщение связанным обработчикам;
    - метод set_chain — устанавливает связь между сигналом текущего объекта и обработчиком целового;

- метод remove_chain — удаляет существующую связь "сигнал - обработчик";
- метод get_coordinate_from_root — возвращает абсолютный путь объекта в иерархии;
- метод set_state_from_actual — устанавливает состояние объекта и всех его подчиненных.

Класс cl_application:

- функционал:
  - метод exec_app — запуск приложения;
  - метод build_tree_objects — построение дерева объектов;
  - метод get_handler_by_class — возврат указателя на обработчика сигнала, соответствующего заданному номеру класса;
  - метод get_signal_by_class — возврат указателя на метод-сигна, соответствующий заданному классу.

Класс cl_2:

- функционал:
  - метод signal_info — выводит сообщение о сигнале и добавляет к строке информацию о классе;
  - метод handler_info — выводит сообщение о полученном сигнале;
  - метод get_object_index — возвращает номер класса.

Класс cl_3:

- функционал:
  - метод signal_info — выводит сообщение о сигнале и добавляет к строке информацию о классе;
  - метод handler_info — выводит сообщение о полученном сигнале;
  - метод get_object_index — возвращает номер класса.

Класс cl_4:

- функционал:
    - метод signal_info — выводит сообщение о сигнале и добавляет к строке информацию о классе;
    - метод handler_info — выводит сообщение о полученном сигнале;
    - метод get_object_index — возвращает номер класса.

Класс cl_5:

- функционал:
    - метод signal_info — выводит сообщение о сигнале и добавляет к строке информацию о классе;
    - метод handler_info — выводит сообщение о полученном сигнале;
    - метод get_object_index — возвращает номер класса.

Класс cl_6:

- функционал:
    - метод signal_info — выводит сообщение о сигнале и добавляет к строке информацию о классе;
    - метод handler_info — выводит сообщение о полученном сигнале;
    - метод get_object_index — возвращает номер класса.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	chain			структрура	
2	cl_base			Базовый класс	
		cl_application	public		3
		cl_2	public		4
		cl_3	public		5
		cl_4	public		6

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
		cl_5	public		7
		cl_6	public		8
3	cl_application			Класс-приложение	
4	cl_2			Дочерний класс	
5	cl_3			Дочерний класс	
6	cl_4			Дочерний класс	
7	cl_5			Дочерний класс	
8	cl_6			Дочерний класс	

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 функции main

Функционал: главная функция программы.

Параметры: нет.

Возвращаемое значение: целое число, идентификатор работоспособности программы.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта cl_ob_application класса cl_application с использованием параметрезированного конструктора и передачей в него в качестве параметра пустого указателя	2
2		Вызов метода build_tree_objects объекта cl_ob_application	3
3		Возвращение результата работы метода exec_app для объекта cl_ob_application	∅

3.2 Алгоритм метода emit_signal класса cl_base

Функционал: вызывает сигнал текущего объекта, передает сообщение связанным обработчикам.

Параметры: TYPE_SIGNAL p_signal, string& information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода emit_signal класса cl_base

№	Предикат	Действия	№ перехода
1	state == 0?		∅
			2
2		метод сигнала вызывается по указателю p_signal параметром information	3
3		инициализация счетчика i со значением 0	4
4	i < chains_of_objects.size?		5
			∅
5	chains_of_objects[i]->p_signal == p_signal?	инициализация указателя на метод обработчика p_handler значением свойства p_handler i-того элемента chains_of_objects	6
			8
6		указатель на объект класса cl_base p_destination инициализируется значением свойства p_destination i-того элемента chains_of_objects	7
7	p_destination->state != 0?	метод обработчика по указателю p_handler вызывается с параметром information	8
			8
8		i+=1	4

3.3 Алгоритм метода set_chain класса cl_base

Функционал: устанавливает связь между сигналом текущего объекта и обработчиком целового.

Параметры: TYPE_SIGNAL p_signal - указатель на метод-сигнал, cl_base* p_destination - целевой объект, TYPE_HANDLER p_handler - указатель на

обработчик.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_chain класса cl_base

№	Предикат	Действия	№ перехода
1		инициализауия счетчками i = 0	2
2	i < chains_of_objects.size?		3
			4
3	связь с такими же p_signal, p_destination, p_handler уже существует?		∅
		i += 1	2
4		объявление указателя new_chain на объект структуры chain	5
5		присвоение свойству p_signal по указателю new_chain значения параметра p_signal	6
6		присвоение свойству p_handler по указателю new_chain значения параметра p_handler	7
7		присвоение свойству p_destination по указателю new_chain значения параметра p_destination	8
8		в вектор chains_of_objects добавляется связь new_chain	∅

3.4 Алгоритм метода remove_chain класса cl_base

Функционал: удаляет существующую связь "сигнал - обработчик".

Параметры: TYPE_SIGNAL p_signal - указатель на метод-сигнал, cl_base* p_destination - целевой объект, TYPE_HANDLER p_handler - указатель на обработчик.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода remove_chain класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация счетчика i = 0	2
2	i < chains_of_objects.size?		3
			∅
3	свойства p_signal, p_destination, p_handler	удаление объекта по i-тому указателю вектора chains_of_objects	4
			5
4		удаление i-того элемента вектора chains_of_objects	5
5		i += 1	2

3.5 Алгоритм метода get_coordinate_from_root класса cl_base

Функционал: возвращает абсолютный путь объекта в иерархии.

Параметры: нет.

Возвращаемое значение: string - абсолютный путь до данного объекта.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_coordinate_from_root класса cl_base

№	Предикат	Действия	№ перехода
1	данный объект - корневой	возврат "/"	∅
			2
2		инициализация указателя на объект класса cl_base ptr текущим объектом	3
3		объявление строки way	4
4	ptr->p_head_object != nullptr?	добавление в строку имени текущего объекта и "/" перед ним	5

№	Предикат	Действия	№ перехода
		возврат way	∅
5		присвоение указателю ptr указателя на головной объект нынешнего объекта	4

3.6 Алгоритм метода set_state_from_actual класса cl_base

Функционал: устанавливает состояние объекта и всех его подчиненных.

Параметры: целое число state.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода set_state_from_actual класса cl_base

№	Предикат	Действия	№ перехода
1	текущий объект не корневой и поле state головного = 0?		∅
		метод set_state вызывается с параметром state	2
2		инициализация счетчика i = 0	3
3	i < p_sub_objects.size?	рекурсивный вызов метода set_state_from_actual по указателю на i-тый объект вектора p_sub_objects	4
			∅
4		i += 1	3

3.7 Алгоритм метода signal_info класса cl_2

Функционал: выводит сообщение о сигнале и добавляет к строке информацию о классе.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода signal_info класса cl_2

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и вызов метода get_coordinate_from_root	2
2		добавление к строке information строк "(class: " и результата вызова get_object_index в строковом формате	∅

3.8 Алгоритм метода handler_info класса cl_2

Функционал: выводит сообщение о полученном сигнале.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода handler_info класса cl_2

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результата работы метода get_coordinate_from_root, строк " Text: " и information	∅

3.9 Алгоритм метода get_object_index класса cl_2

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: целое число - индекс объекта.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода get_object_index класса cl_2

№	Предикат	Действия	№ перехода
1		return 2	∅

3.10 Алгоритм метода signal_info класса cl_3

Функционал: выводит сообщение о сигнале и добавляет к строке информацию о классе.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода signal_info класса cl_3

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и вызов метода get_coordinate_from_root	2
2		добавление к строке information строк "(class: " и результата вызова get_object_index в строковом формате	∅

3.11 Алгоритм метода handler_info класса cl_3

Функционал: выводит сообщение о полученном сигнале.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода handler_info класса cl_3

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результата работы метода get_coordinate_from_root, строк " Text: " и information	∅

3.12 Алгоритм метода get_object_index класса cl_3

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: целое число - индекс объекта.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода get_object_index класса cl_3

№	Предикат	Действия	№ перехода
1		return 3	∅

3.13 Алгоритм метода signal_info класса cl_4

Функционал: выводит сообщение о сигнале и добавляет к строке информацию о классе.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода signal_info класса cl_4

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и вызов метода get_coordinate_from_root	2
2		добавление к строке information строк "(class: " и результата вызова get_object_index в строковом формате	∅

3.14 Алгоритм метода handler_info класса cl_4

Функционал: выводит сообщение о полученном сигнале.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода handler_info класса cl_4

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результата работы метода get_coordinate_from_root, строк " Text: " и information	∅

3.15 Алгоритм метода get_object_index класса cl_4

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: целое число - индекс объекта.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода get_object_index класса cl_4

№	Предикат	Действия	№ перехода
1		return 4	∅

3.16 Алгоритм метода signal_info класса cl_5

Функционал: выводит сообщение о сигнале и добавляет к строке информацию о классе.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода signal_info класса cl_5

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и вызов метода	2

№	Предикат	Действия	№ перехода
		get_coordinate_from_root	
2		добавление к строке information строк "(class: " и результата вызова get_object_index в строковом формате	∅

3.17 Алгоритм метода handler_info класса cl_5

Функционал: выводит сообщение о полученном сигнале.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода handler_info класса cl_5

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результата работы метода get_coordinate_from_root, строк " Text: " и information	∅

3.18 Алгоритм метода get_object_index класса cl_5

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: целое число - индекс объекта.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода get_object_index класса cl_5

№	Предикат	Действия	№ перехода
1		return 5	∅

3.19 Алгоритм метода signal_info класса cl_6

Функционал: выводит сообщение о сигнале и добавляет к строке информацию о классе.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода signal_info класса cl_6

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и вызов метода get_coordinate_from_root	2
2		добавление к строке information строк "(class: " и результата вызова get_object_index в строковом формате	∅

3.20 Алгоритм метода handler_info класса cl_6

Функционал: выводит сообщение о полученном сигнале.

Параметры: string information.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода handler_info класса cl_6

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal to ", результата работы метода get_coordinate_from_root, строк " Text: " и information	∅

3.21 Алгоритм метода get_object_index класса cl_6

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: целое число - индекс объекта.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода get_object_index класса cl_6

№	Предикат	Действия	№ перехода
1		return 6	∅

3.22 Алгоритм метода exec_app класса cl_application

Функционал: запуск приложения.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		обявление строк action, coordinate, information	2
2		объявление указателя destination_object на объект класса cl_base	3
3		объявление указателя actual_object на объект класса cl_base	4
4		объявление целочисленной переменной state	5
5		вызов метода set_states_from_actual с передачей в качестве параметра 1	6
6		объявление указателя на метод сигнала p_signal	7
7		объявление указателя на метод обработчика p_handler	8
8		вызов print_names_recur у текущего объекта	9
9		ввод значения строковой переменной action	10

№	Предикат	Действия	№ перехода
10	action != "END"?	ввод значения строковой переменнй coordinate	11
		return 0	∅
11		присвоение actual_object результата вызова get_object_by_coordinate с параметром coordinate	12
12	указатель actual_object нулевой?	вывод с новой строки "Object", coordinate и " not found"	13
			14
13		ввод значения строковой переменной coordinate	10
14	action "EMIT"?	ввод значения строковой переменной information	15
			17
15		инициализация целочисленной переменной num_class значением вызова метода get_object_index у actual_object	16
16		вызов метода emit_signal у actual_object с параметрами information и указателем на метод сигнала	17
17	action == "SET_CONNECT"?	ввод значения строковой переменной coordinate	18
			22
18		присвоение destination_object результата вызова метода get_object_by_coordinate с параметром coordinate	19
19	destination_object - нулевой?	вывод с новой строки "Handler object ", coordinate, " not found"	10
		присвоение p_signal результата вызова метода функции get_signal_by_class с передачей параметром номера объекта по указателю destination_object	20
20		присвоение p_handler результата вызова метода get_handler_by_class с передачей параметром номера класса объекта по указателю	21

№	Предикат	Действия	№ перехода
		destination_object	
21		вызов метода set_chain у объекта по указателю actual_object с параметрами p_signal, p_destination, p_handler	22
22	action == "DELETE_CONNECT"?	ввод значения строковой переменной coordinate	23
			27
23		присвоение destination_object результата работы метода get_object_by_coordinate с параметром coordinate	24
24	destination_object - нулевой объект	вывод с новой строки "Handler object ", coordinate, " not found"	10
		присвоение p_signal результата вызова метода get_signal_by_class с передачей параметром номера объекта по указателю actual_object	25
25		присвоение p_handler результата вызова метода get_handler_by_class с передачей параметром номера объекта по указателю destination_object	26
26		вызов метода delete_chain у объекта по указателю actual_object с параметром p_signal, p_destination, p_handler	29
27	action == "SET_CONDITION"?	ввод значения переменной state	28
			29
28		вызов метода set_state с параметром state у объекта по указателю actual_object	29
29		ввод значения строковой переменной action	10

3.23 Алгоритм метода build_tree_objects класса cl_application

Функционал: построение дерева объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		объявление указателя на объект класса cl_base destination_object	2
2		ввод значения строковой переменной way	3
3	way != "end_of_connections"?	ввод значения строковой переменной curname	4
			∅
4		присвоение указателю p_head результата работы метода get_obj_by_way с параметром way	5
5		присвоение указателю destination_object результата работы метода get_obj_by_way с параметром curname	6
6		инициализация метода указателя p_signal результатом вызова метода get_signal_by_class с передачей в качестве параметра индекса объекта по указателю p_head	7
7		инициализация метода указателя p_handler результатом вызова метода get_handler_by_class с передачей в качестве параметра индекса объекта по указателю destination_objects	8
8		вызов метода set_chain у объекта по указателю p_head с параметрами p_head, destination_object и p_handler	9

№	Предикат	Действия	№ перехода
9		ввод значения строковой переменной way	3

3.24 Алгоритм метода get_handler_by_class класса cl_application

Функционал: возврат указателя на обработчика сигнала, соответствующего заданному номеру класса.

Параметры: index_obj.

Возвращаемое значение: TYPE_HANDLER - указатель на обработчик сигнала.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода get_handler_by_class класса cl_application

№	Предикат	Действия	№ перехода
1	index_obj == 2?	возврат указателя на метод обработчика класса cl_2	∅
	index_obj == 3?	возврат указателя на метод обработчика класса cl_3	∅
	index_obj == 4?	возврат указателя на метод обработчика класса cl_4	∅
	index_obj == 5?	возврат указателя на метод обработчика класса cl_5	∅
	index_obj == 6?	возврат указателя на метод обработчика класса cl_6	∅
		возврат nullptr	∅

3.25 Алгоритм метода get_signal_by_class класса cl_application

Функционал: возврат указателя на метод-сигна, соответствующий заданному классу.

Параметры: index_obj.

Возвращаемое значение: TYPE_SIGNAL - указатель на функцию-сигнал.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода get_signal_by_class класса cl_application

№	Предикат	Действия	№ перехода
1	index_obj == 2?	возврат указателя на метод сигнала класса cl_2	∅
	index_obj == 3?	возврат указателя на метод сигнала класса cl_3	∅
	index_obj == 4?	возврат указателя на метод сигнала класса cl_4	∅
	index_obj == 5?	возврат указателя на метод сигнала класса cl_5	∅
	index_obj == 6?	возврат указателя на метод сигнала класса cl_6	∅
		возврат nullptr	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-18.

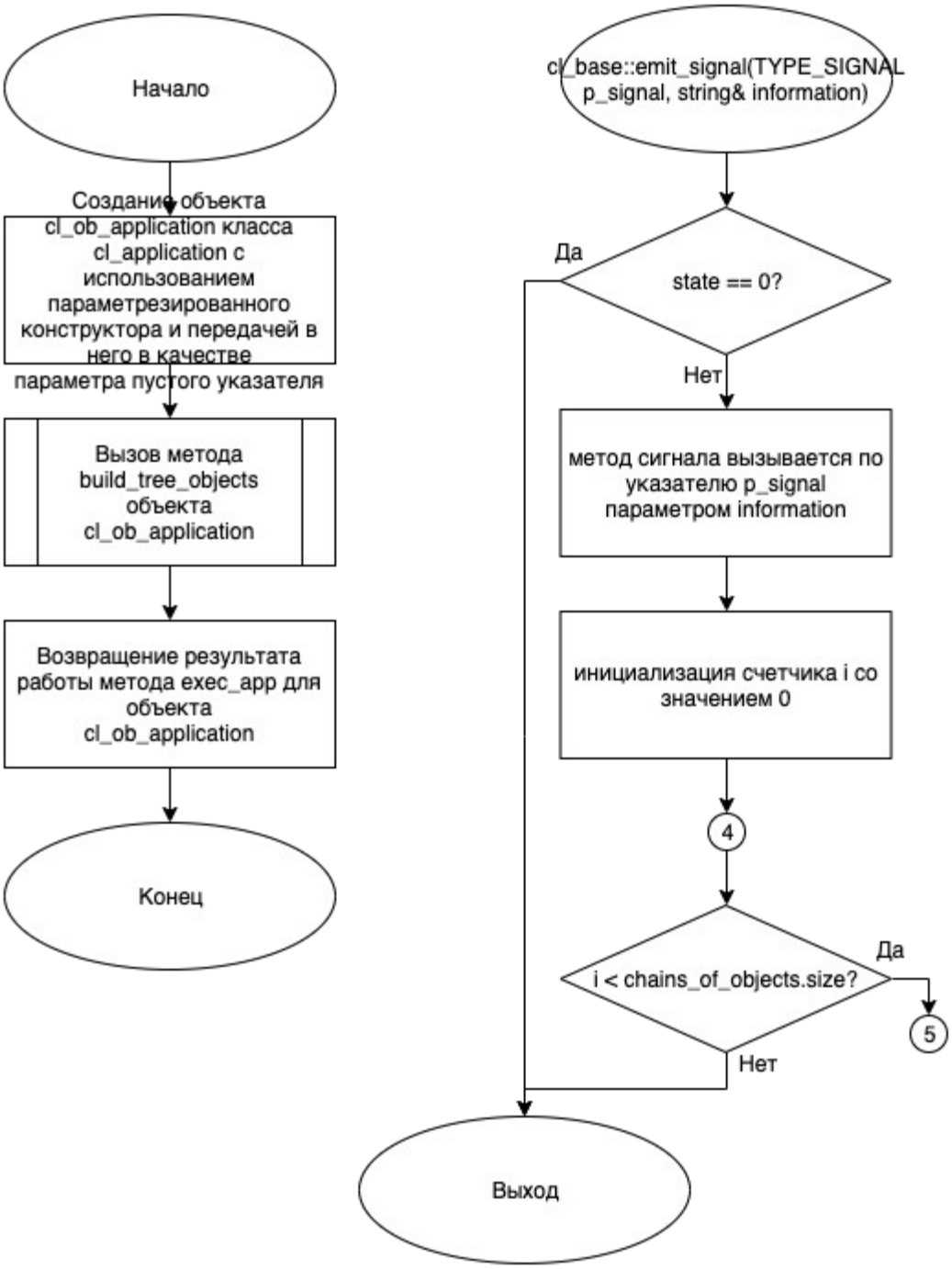

Рисунок 1 – Блок-схема алгоритма

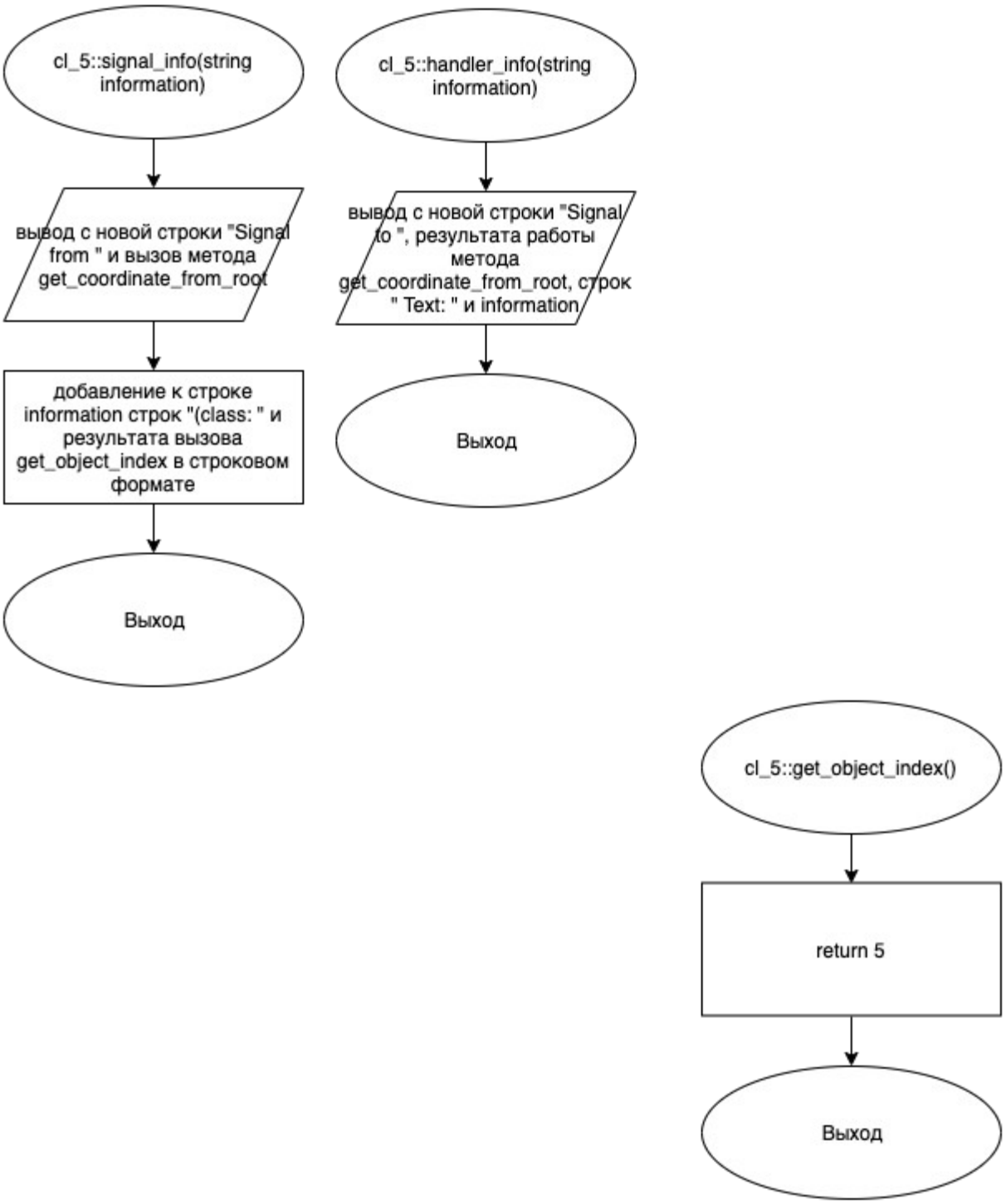

Рисунок 2 – Блок-схема алгоритма

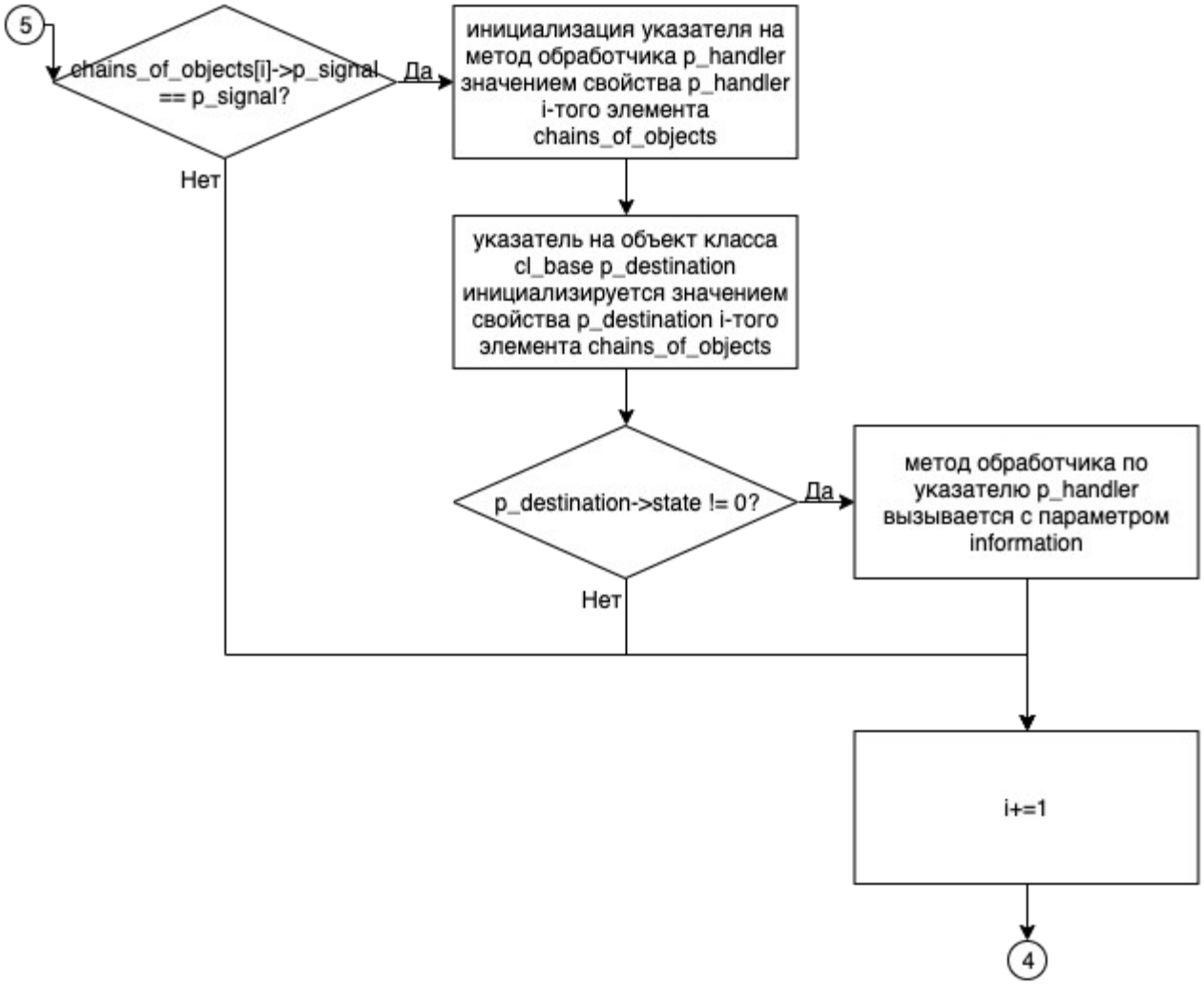

Рисунок 3 – Блок-схема алгоритма

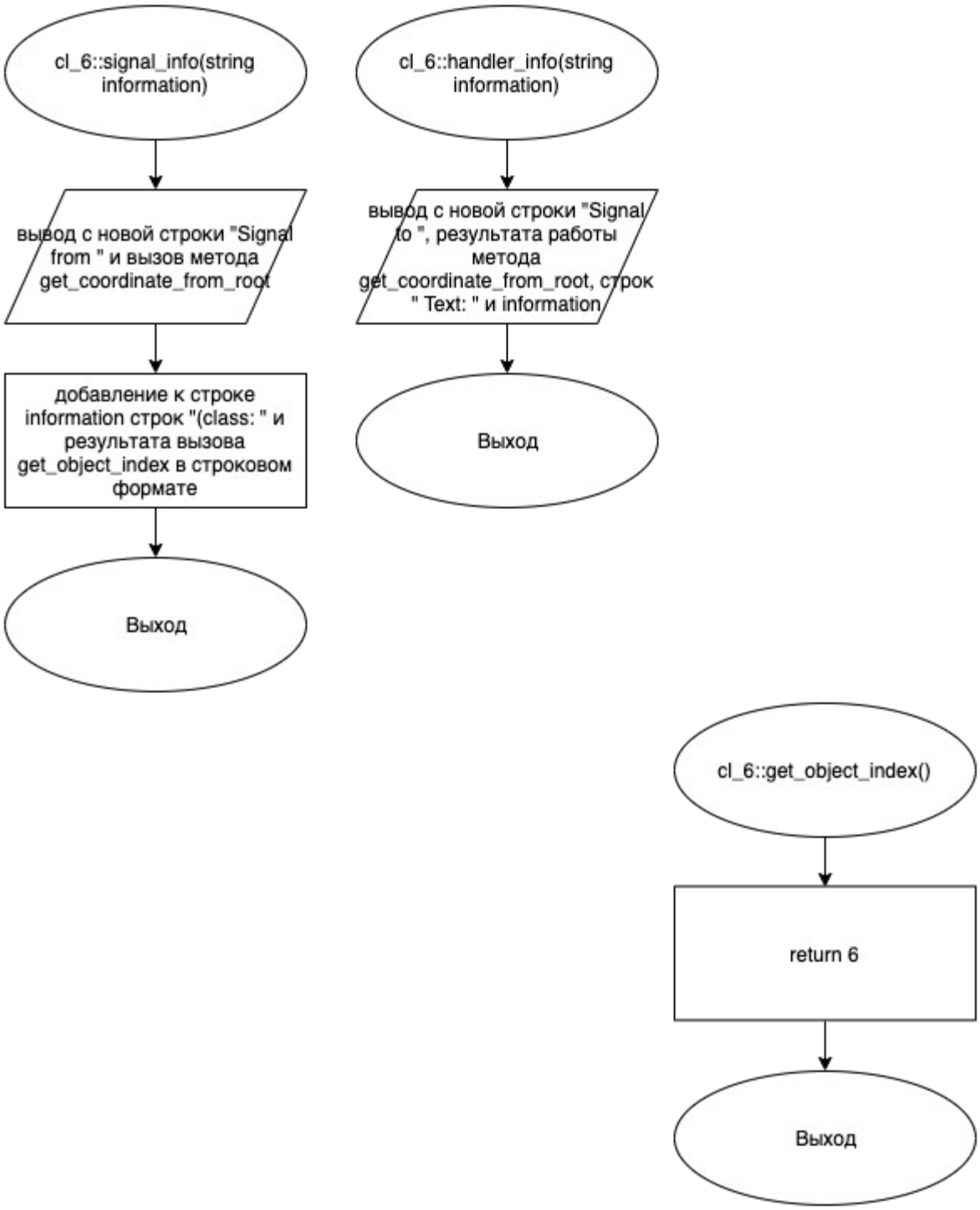

Рисунок 4 – Блок-схема алгоритма

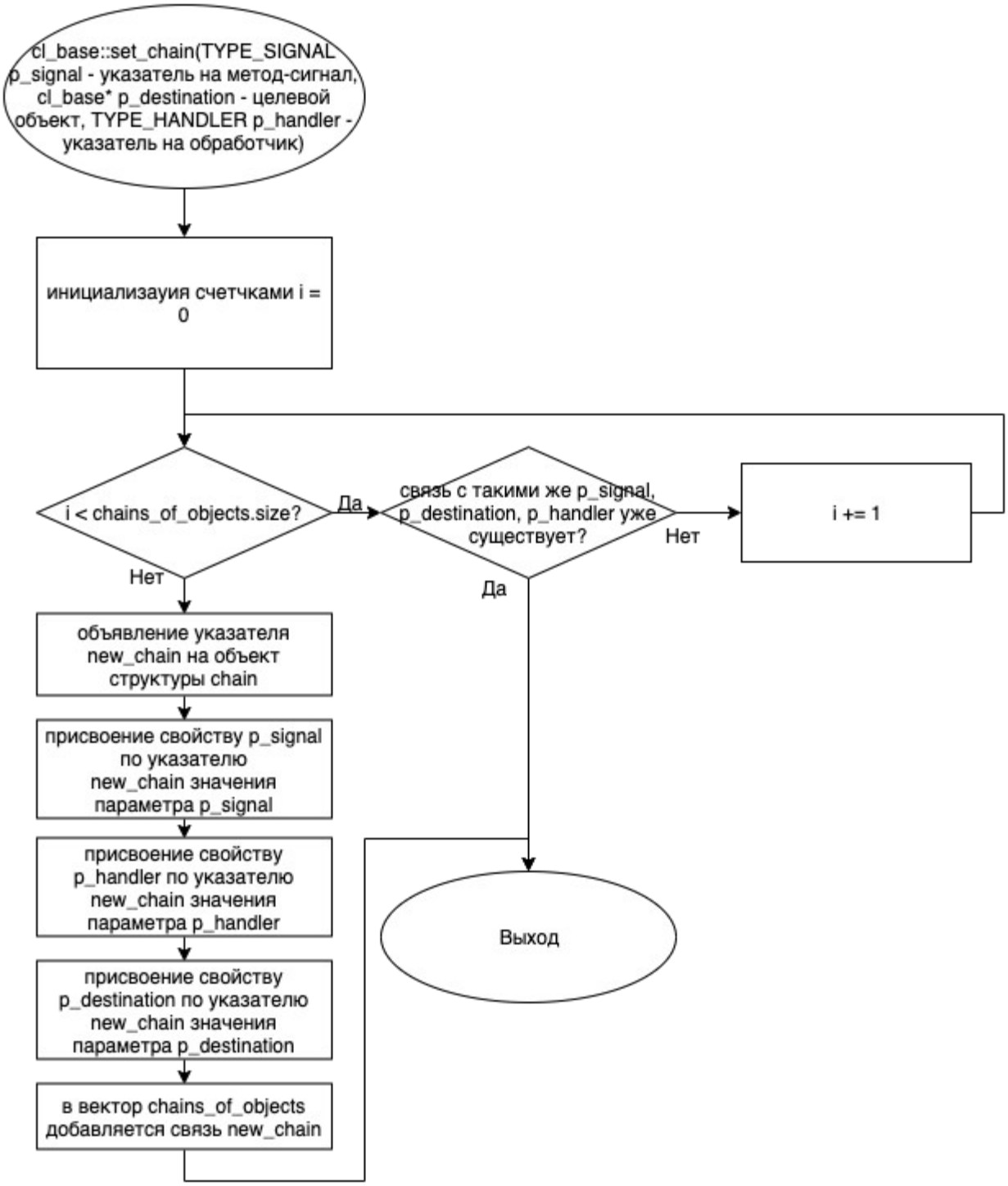

Рисунок 5 – Блок-схема алгоритма

Рисунок 6 – Блок-схема алгоритма

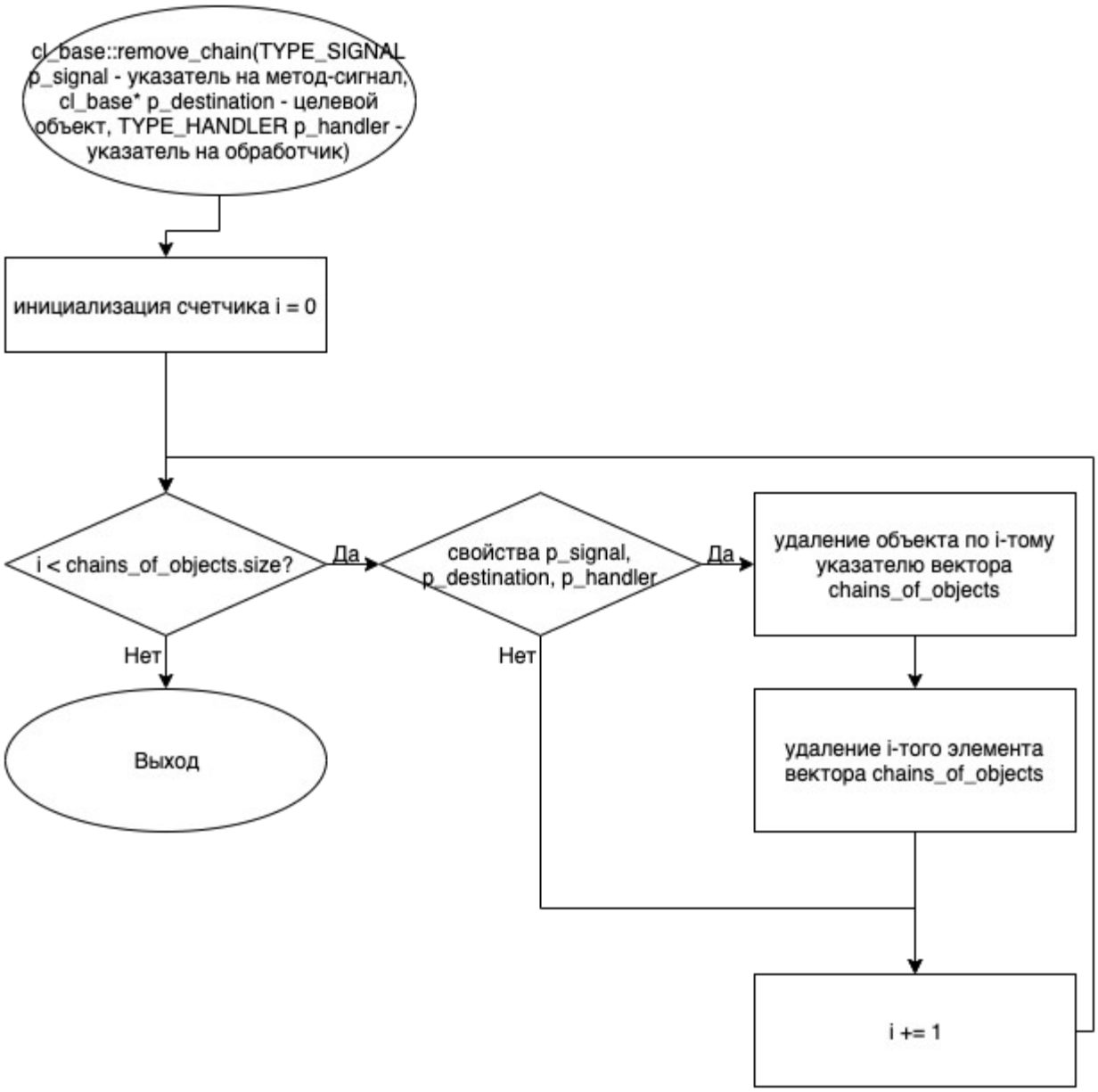

Рисунок 7 – Блок-схема алгоритма

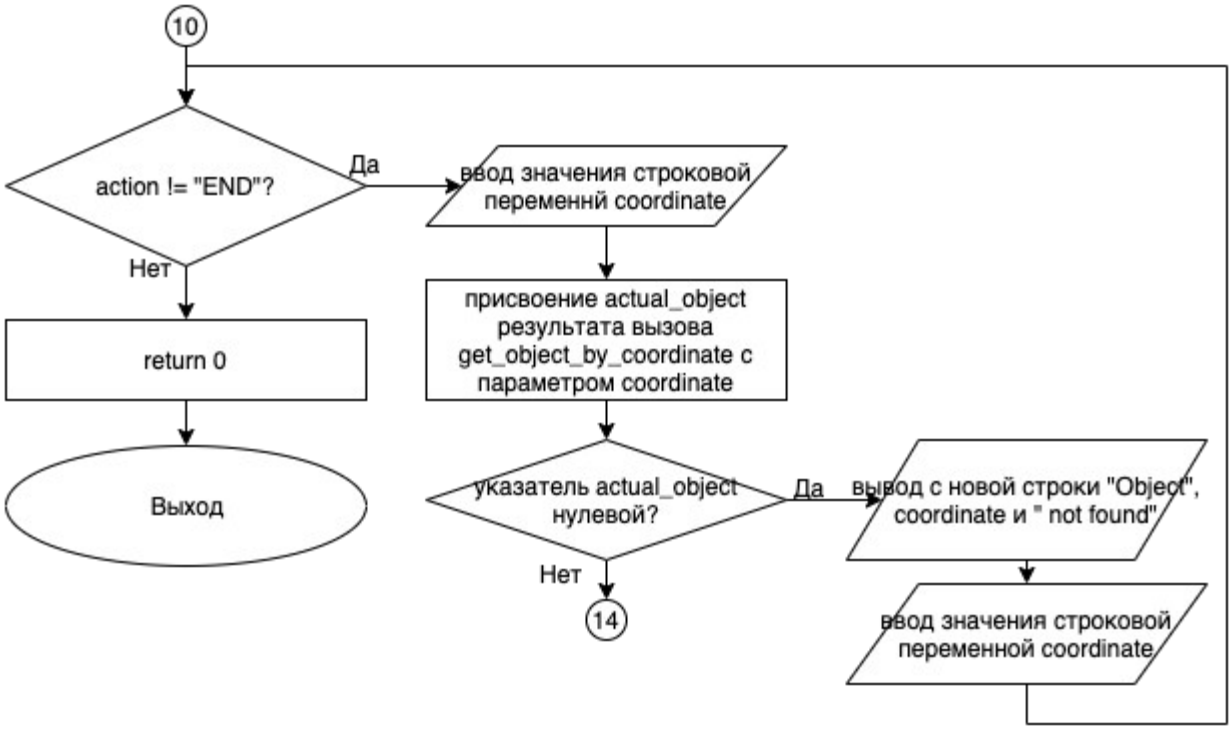

Рисунок 8 – Блок-схема алгоритма

Рисунок 9 – Блок-схема алгоритма

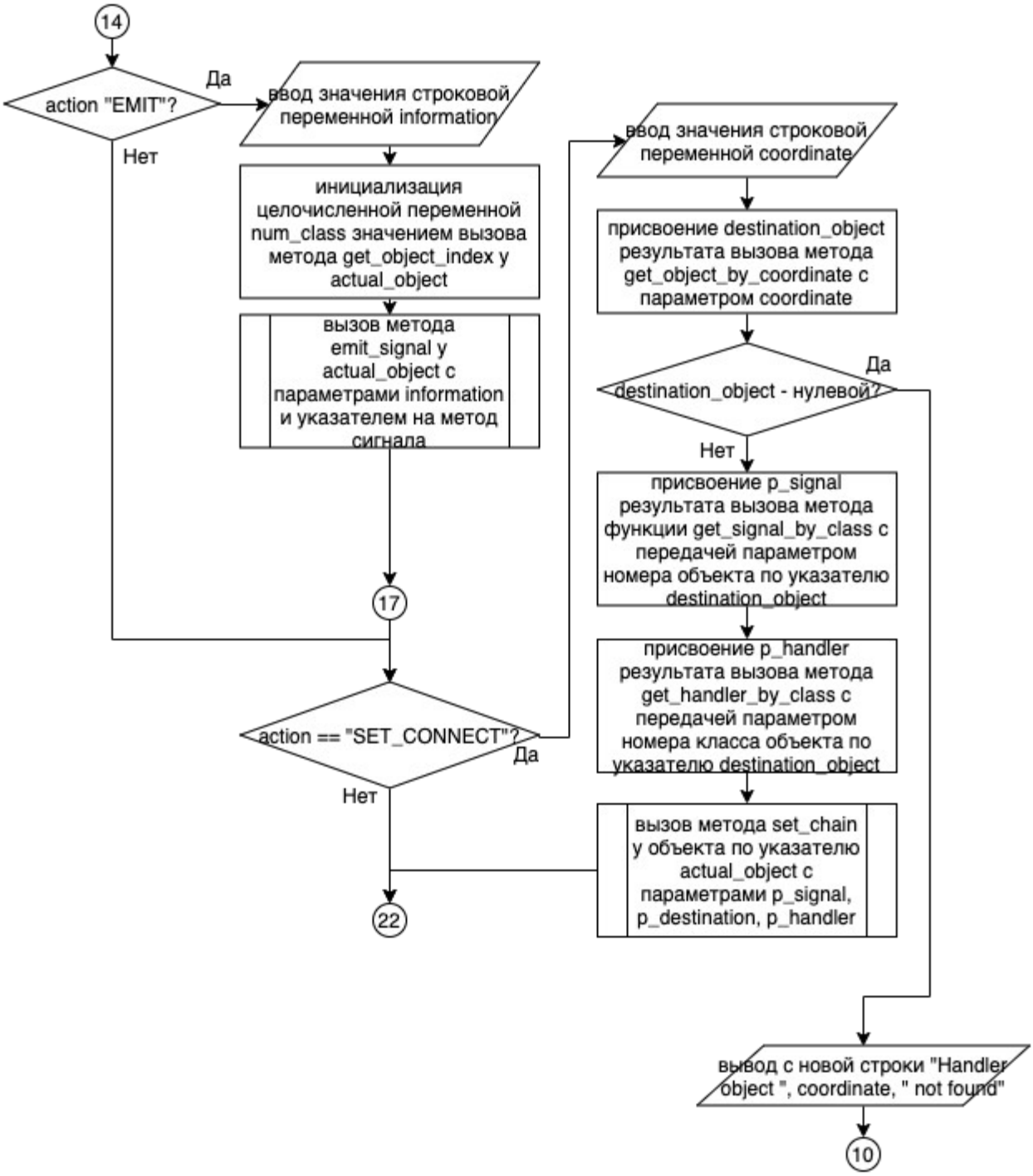

Рисунок 10 – Блок-схема алгоритма

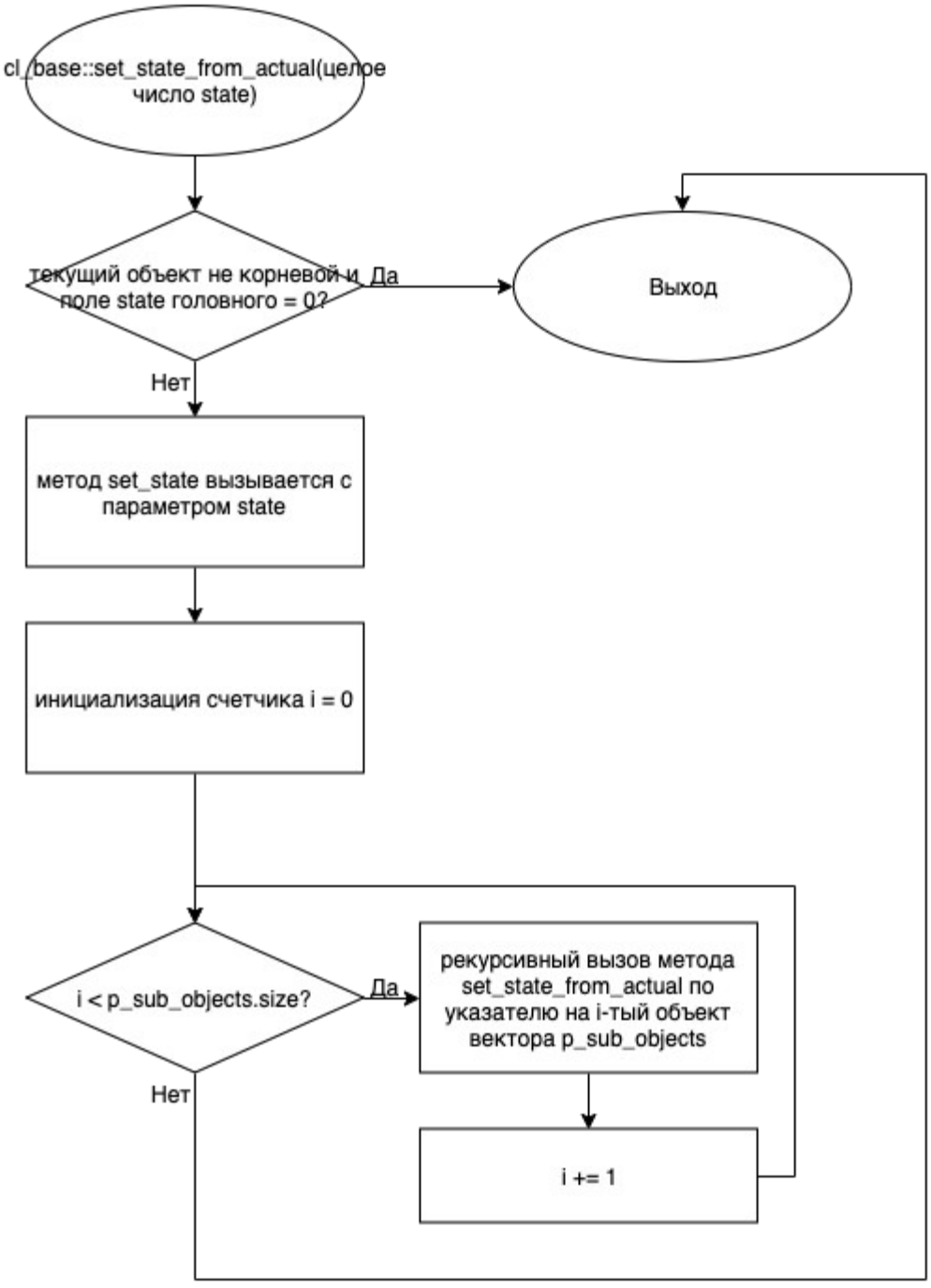

Рисунок 11 – Блок-схема алгоритма

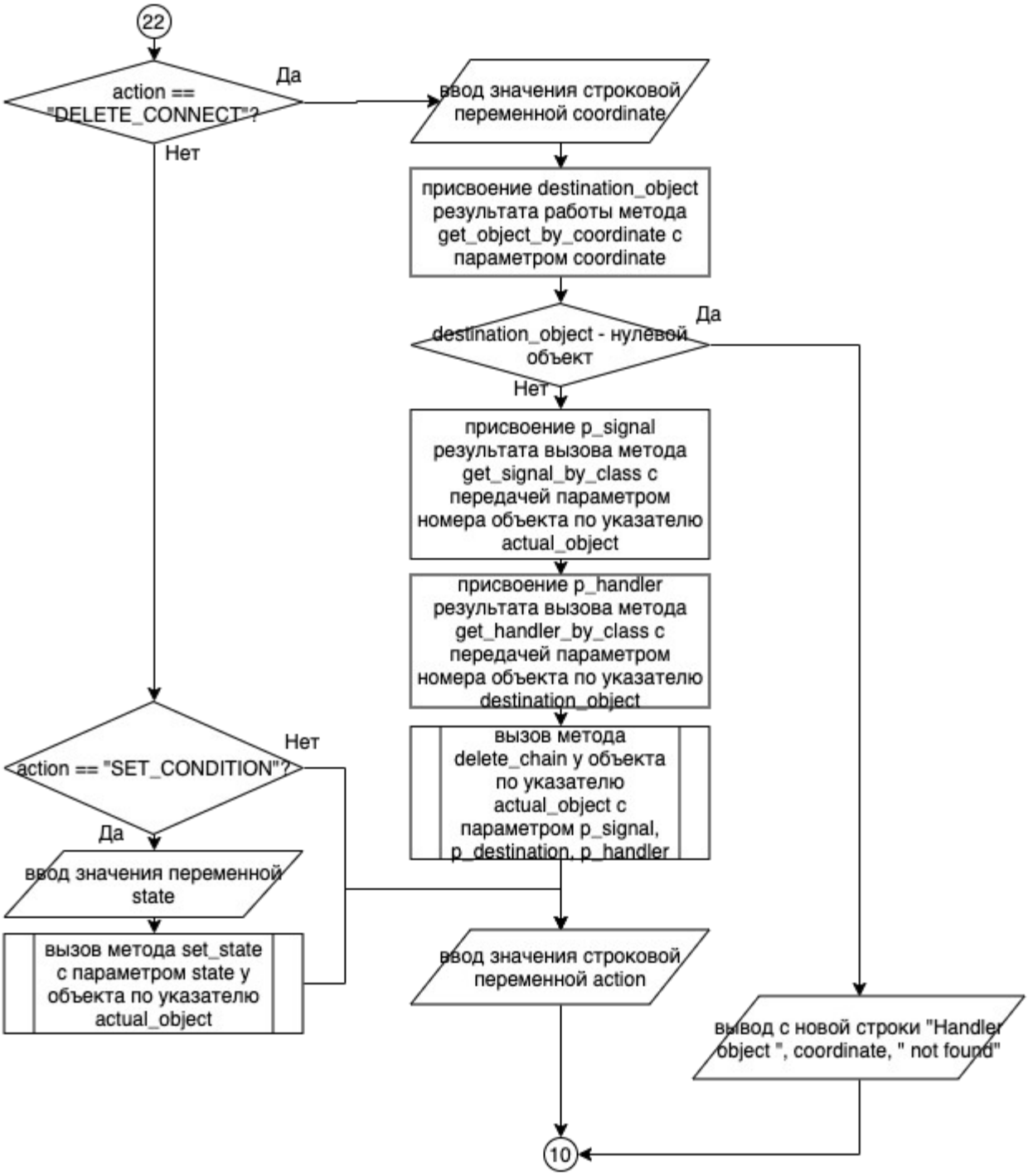

Рисунок 12 – Блок-схема алгоритма

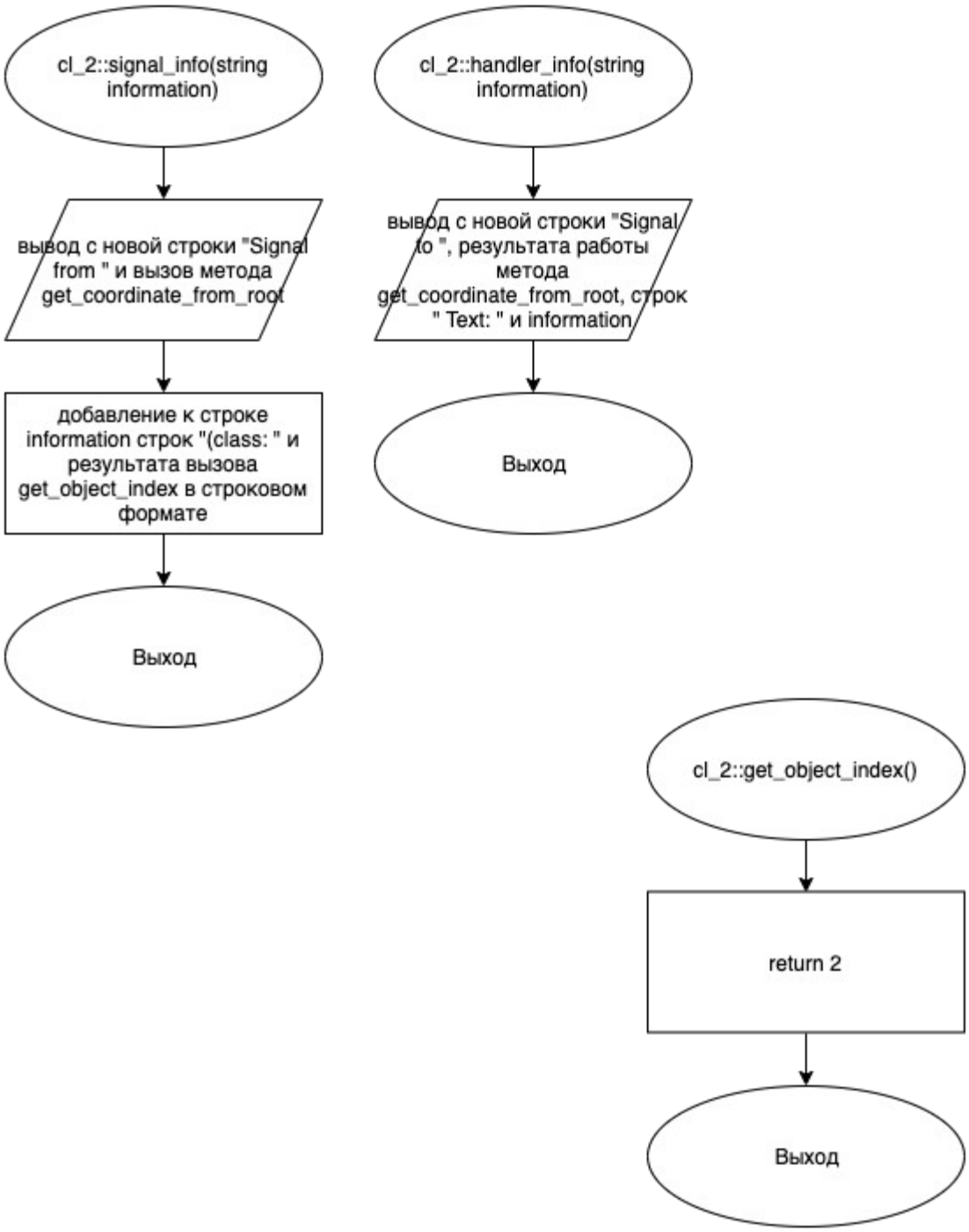

Рисунок 13 – Блок-схема алгоритма

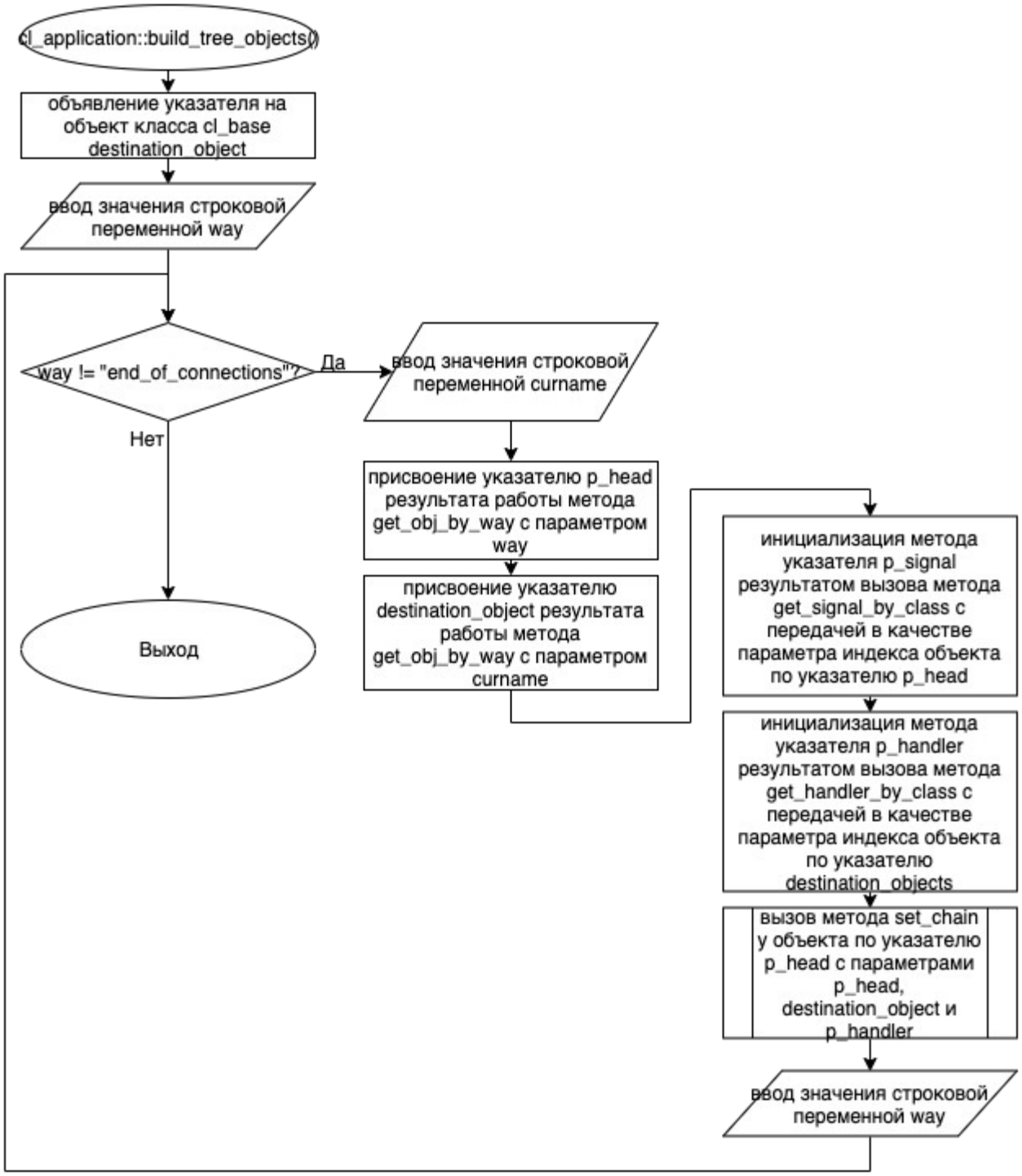

Рисунок 14 – Блок-схема алгоритма

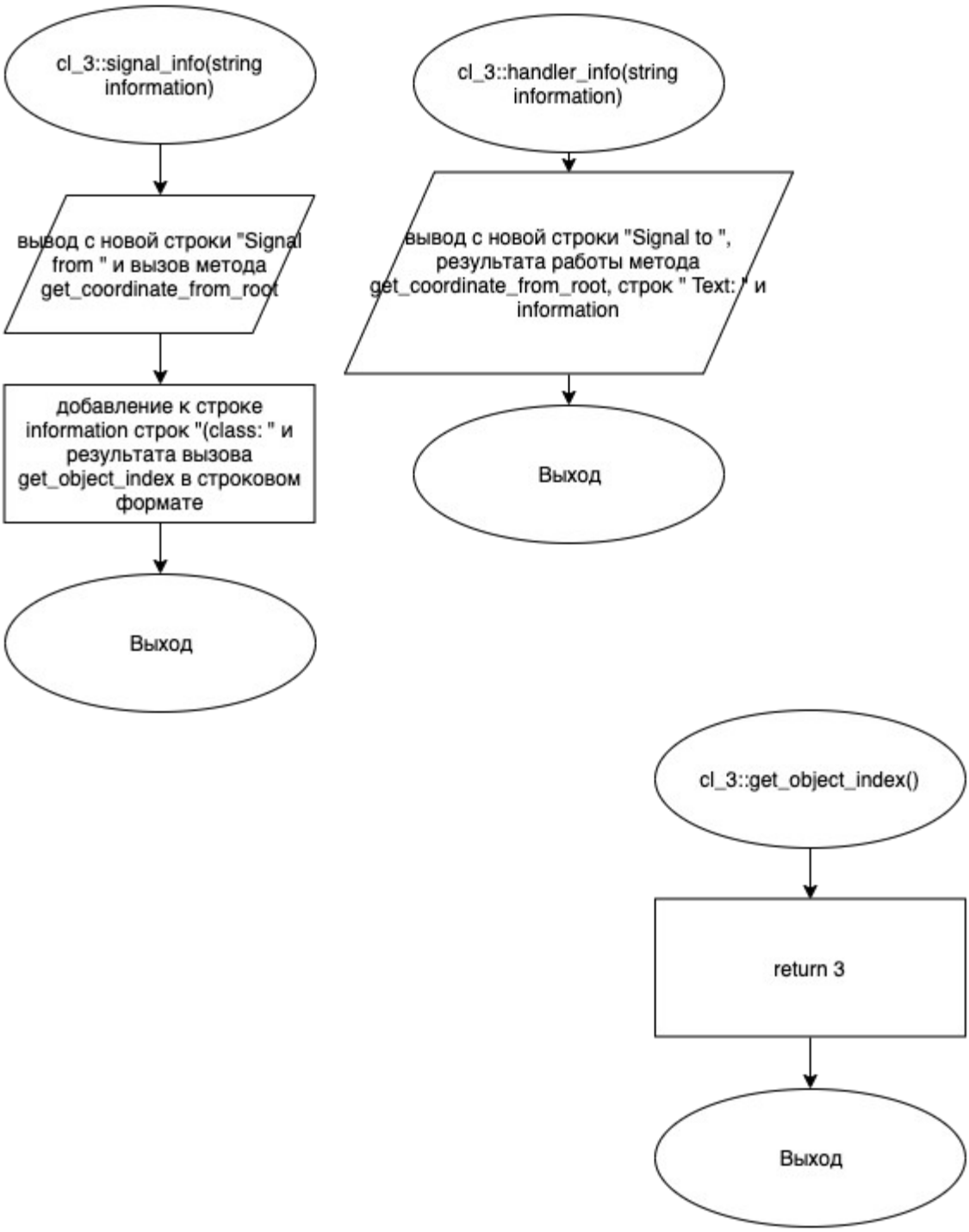

Рисунок 15 – Блок-схема алгоритма

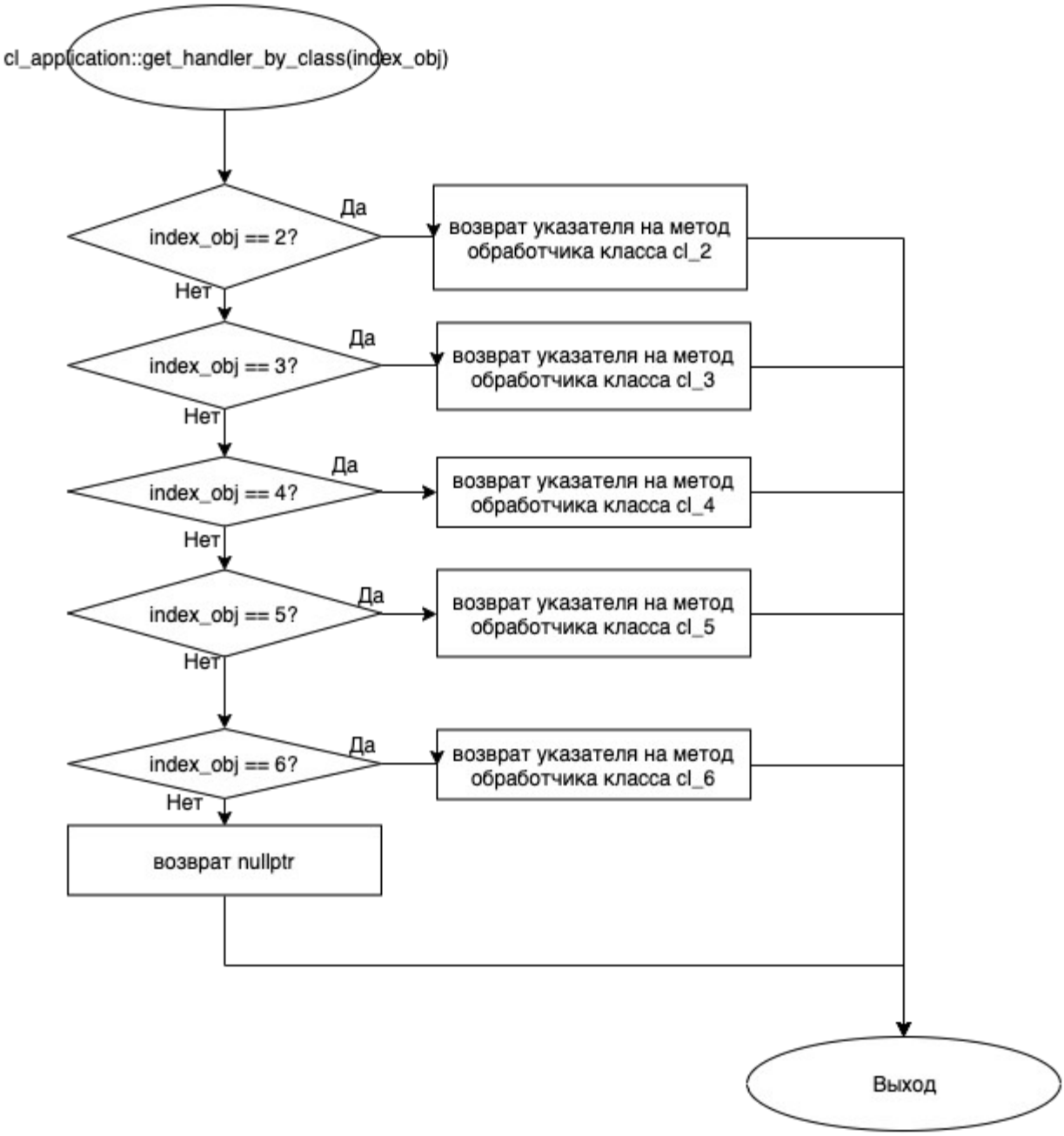

Рисунок 16 – Блок-схема алгоритма

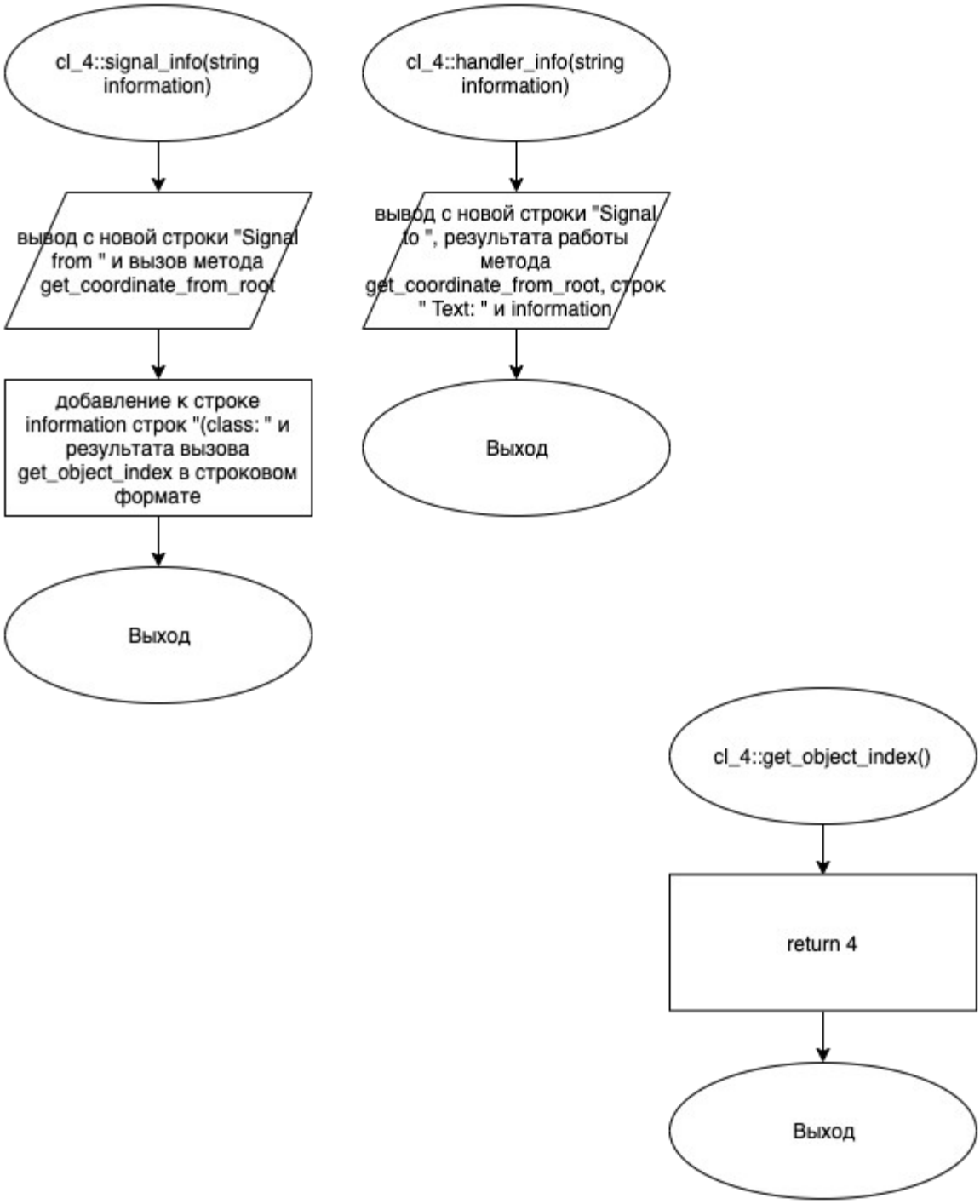

Рисунок 17 – Блок-схема алгоритма

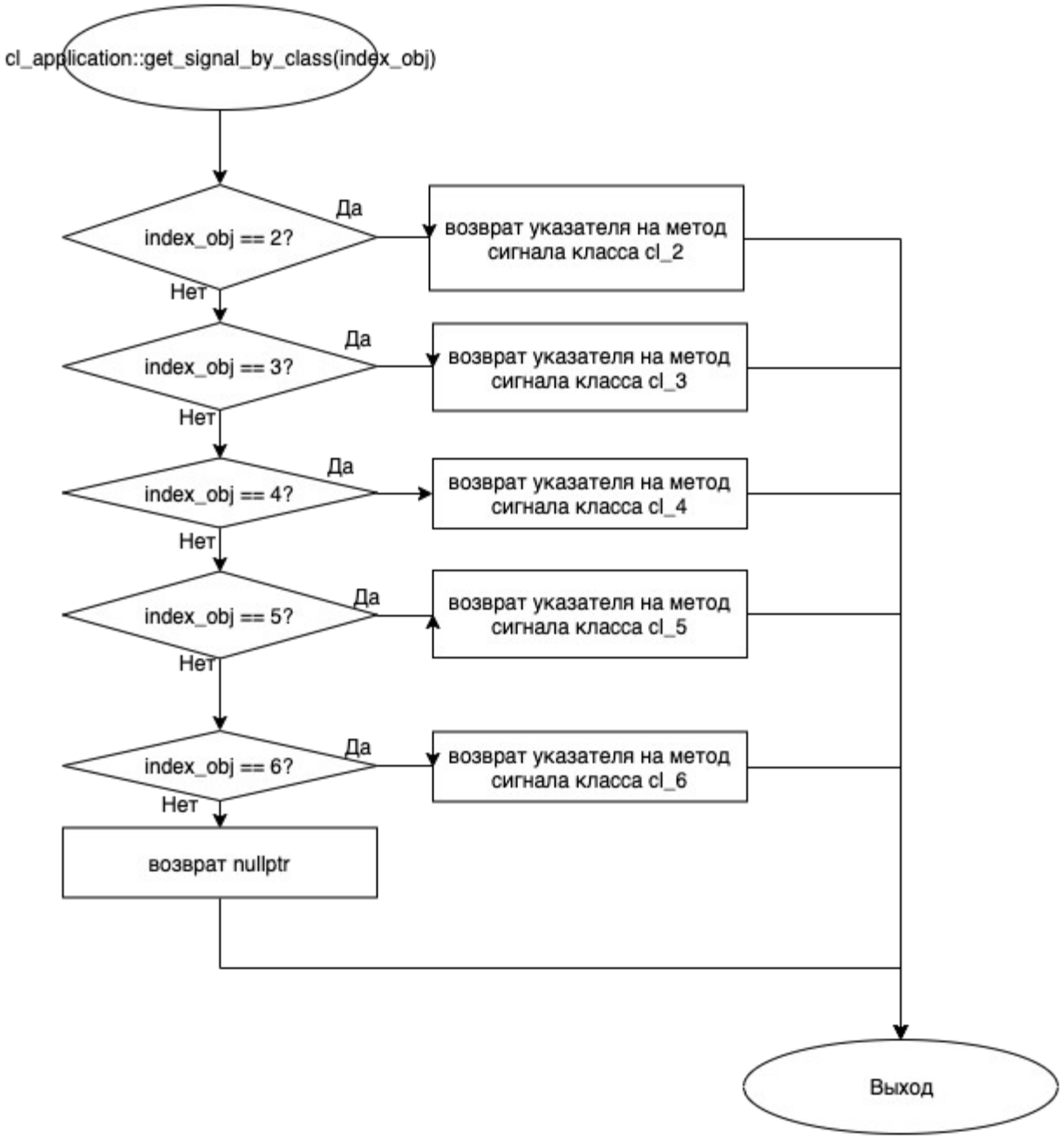

Рисунок 18 – Блок-схема алгоритма

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"
using namespace std;

cl_2::cl_2(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name)
{
   // конструктор объекта cl_2
}

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

void cl_2::handler_info(string information)
{
   cout << endl << "Signal to " << get_coordinate_from_root() << " Text: " <<
information;
   // вывод информации по обработчику
}

void cl_2::signal_info(string& information)
{
   cout  << endl << "Signal from " << get_coordinate_from_root();
   // вывод инфмаорции о сигнале
   information += " (class: " + to_string(get_object_index()) + ")";
   // обновление информации
}
```

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:
   cl_2(cl_base* p_head_object, string s_name);
   int get_object_index();
   void handler_info(string information);
   void signal_info(string& information);
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
using namespace std;

cl_3::cl_3(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_3
}

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

void cl_3::handler_info(string information)
{
   cout << endl << "Signal to " << get_coordinate_from_root() << " Text: " <<
information;
   // вывод информации по обработчику
}

void cl_3::signal_info(string& information)
{
   cout  << endl << "Signal from " << get_coordinate_from_root();
   // вывод инфмаорции о сигнале
   information += " (class: " + to_string(get_object_index()) + ")";
   // обновление информации
}
```

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:
   cl_3(cl_base* p_head_object, string s_name);
   int get_object_index();
   void handler_info(string information);
   void signal_info(string& information);
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"
using namespace std;

cl_4::cl_4(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_4
}

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

void cl_4::handler_info(string information)
{
   cout << endl << "Signal to " << get_coordinate_from_root() << " Text: " <<
information;
   // вывод информации по обработчику
}

void cl_4::signal_info(string& information)
{
   cout  << endl << "Signal from " << get_coordinate_from_root();
   // вывод инфмаорции о сигнале
   information += " (class: " + to_string(get_object_index()) + ")";
   // обновление информации
```

```
}
```

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:
   cl_4(cl_base* p_head_object, string s_name);
   int get_object_index();
   void handler_info(string information);
   void signal_info(string& information);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"
using namespace std;

cl_5::cl_5(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_5
}

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

void cl_5::handler_info(string information)
{
   cout << endl << "Signal to " << get_coordinate_from_root() << " Text: " <<
information;
   // вывод информации по обработчику
}

void cl_5::signal_info(string& information)
```

```
{
   cout  << endl << "Signal from " << get_coordinate_from_root();
   // вывод инфмаорции о сигнале
   information += " (class: " + to_string(get_object_index()) + ")";
   // обновление информации
}
```

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:
   cl_5(cl_base* p_head_object, string s_name);
   int get_object_index();
   void handler_info(string information);
   void signal_info(string& information);
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_6
}

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

void cl_6::handler_info(string information)
{
   cout << endl << "Signal to " << get_coordinate_from_root() << " Text: " <<
information;
```

```
   // вывод информации по обработчику
}

void cl_6::signal_info(string& information)
{
   cout  << endl << "Signal from " << get_coordinate_from_root();
   // вывод инфмаорции о сигнале
   information += " (class: " + to_string(get_object_index()) + ")";
   // обновление информации
}
```

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:
   cl_6(cl_base* p_head_object, string s_name);
   int get_object_index();
   void handler_info(string information);
   void signal_info(string& information);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "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"
#include <iostream>
using namespace std;

cl_application::cl_application(cl_base*           p_head_object)           :
cl_base(p_head_object)
```

```
{

};

TYPE_HANDLER get_handler_by_class(int index_obj)
{
   switch(index_obj)
   {
      case 1:
         return HANDLER_D(cl_application::handler_info);
      case 2:
         return HANDLER_D(cl_2::handler_info);
      case 3:
         return HANDLER_D(cl_2::handler_info);
      case 4:
         return HANDLER_D(cl_2::handler_info);
      case 5:
         return HANDLER_D(cl_2::handler_info);
      case 6:
         return HANDLER_D(cl_2::handler_info);
   }
   return nullptr;
}

TYPE_SIGNAL get_signal_by_class(int index_obj)
{
   switch(index_obj)
   {
      case 1:
         return SIGNAL_D(cl_application::signal_info);
      case 2:
         return SIGNAL_D(cl_2::signal_info);
      case 3:
         return SIGNAL_D(cl_2::signal_info);
      case 4:
         return SIGNAL_D(cl_2::signal_info);
      case 5:
         return SIGNAL_D(cl_2::signal_info);
      case 6:
         return SIGNAL_D(cl_2::signal_info);
   }
   return nullptr;
}

void cl_application::build_tree_objects()
{
   string way, curname;
   int i_class;
   cout << "Object tree";
   cin>>curname;
   set_name(curname);
   cl_base* p_head;
   cl_base* p_sub;
   cin >> way;
   while (way != "endtree")
```

```
        {
           cin >> curname >> i_class;
           p_head = get_obj_by_way(way);
           if (!p_head)
           {
              print_names_recur();
              cout << "\n" << "The head object " << way <<" is not found";
              exit(1);
           }
           if (p_head -> get_sub_object(curname))
           {
              cout << "\n" << way << " Dubbing the names of subordinate objects";
           }
           else
           {
              switch (i_class)
              {
                 case 2:
                    new cl_2(p_head, curname);
                    break;
                 case 3:
                    new cl_3(p_head, curname);
                    break;
                 case 4:
                    new cl_4(p_head, curname);
                    break;
                 case 5:
                    new cl_5(p_head, curname);
                    break;
                 case 6:
                    new cl_6(p_head, curname);
                    break;
              }
           }
        cin >> way;
        }
        cl_base* destination_object;
        cin >> way;
        while (way != "end_of_connections")
        {
           cin >> curname;
           p_head = get_obj_by_way(way);
           destination_object = get_obj_by_way(curname);
           TYPE_SIGNAL      signal_f      =      get_signal_by_class(p_head      ->
     get_object_index());
           TYPE_HANDLER  handler_f  =  get_handler_by_class(destination_object  ->
     get_object_index());
           p_head -> set_chain(signal_f, destination_object, handler_f);
           cin >> way;
        }
     }

     //

     int cl_application::exec_app()
```

```
{
   string action, coordinate, information;
   cl_base* destination_object;
   cl_base* actual_object;
   int state;
   this -> set_state_from_actual(1);
   TYPE_SIGNAL p_signal;
   TYPE_HANDLER p_handler;
   this -> print_names_recur();
   cin>>action;
   while (action != "END"){
      cin >> coordinate;
      actual_object = this -> get_obj_by_way(coordinate);
      if (!actual_object)
      {
         cout << endl << "Object " << coordinate << " not found";
         cin >> action;
            continue;
      }
      if (action == "EMIT")
      {
         getline(cin,information);
         int num_class = actual_object->get_object_index();
         actual_object->emit_signal(get_signal_by_class(num_class),
information);
      }
      if (action == "SET_CONNECT")
      {
         cin >> coordinate;
         destination_object = this -> get_obj_by_way(coordinate);
         if (!destination_object)
         {
            cout << endl << "Handler object " << coordinate << " not found";
            cin >> action;
            continue;
         }
         p_signal = get_signal_by_class(actual_object -> get_object_index());
         p_handler      =      get_handler_by_class(destination_object      ->
get_object_index());
         actual_object -> set_chain(p_signal, destination_object, p_handler);
      }
      else if(action == "DELETE_CONNECT")
      {
         cin >> coordinate;
         destination_object = this -> get_obj_by_way(coordinate);
         if (!destination_object)
         {
            cout << endl << "Handler object " << coordinate << " not found";
            cin >> action;
            continue;
         }
         p_signal = get_signal_by_class(actual_object->get_object_index());
         p_handler         =         get_handler_by_class(destination_object-
>get_object_index());
         actual_object    ->    remove_chain(p_signal,    destination_object,
```

```
p_handler);
      }
      else if (action == "SET_CONDITION")
      {
         cin >> state;
         actual_object->set_state(state);
      }
      cin >> action;
   }
   return 0;
}

int cl_application::get_object_index()
{
   return 1;
}

void cl_application::signal_info(string& information)
{
   cout << endl << "Signal from " << get_coordinate_from_root();
   information += " (class: " + to_string(get_object_index()) + ")";
}

void cl_application::handler_info(string information)
{
   cout << endl << "Signal to " << get_coordinate_from_root() << " Text: " <<
information;
}
```

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"

class cl_application : public cl_base{
public:
   cl_application(cl_base* p_head_object = nullptr);
   void build_tree_objects();
   int exec_app();

   void handler_info(string information);
   void signal_info(string& information);
   int get_object_index();
};
```

```
#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

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

// 1

cl_base::cl_base(cl_base* p_head_object, string s_name){
   this->p_head_object = p_head_object;
   if (p_head_object){
      p_head_object -> p_sub_objects.push_back(this);
   }
   this -> state = 0;
   set_name(s_name);
}

string cl_base::get_name(){
   return s_name; // возврат имени объекта
}

cl_base* cl_base::get_head(){
   return p_head_object;
   // возвращение указателя на головной объект
}

cl_base* cl_base::get_sub_object(string s_name){
   for (auto obj : p_sub_objects){
      if (obj->get_name() == s_name) return obj;
   }
   return nullptr;
}

// 2

cl_base* cl_base::find_obj_current(string name){
   queue <cl_base*> q;
   cl_base* p_found = nullptr;
   q.push(this);
   while (!q.empty()){
      if (q.front() -> get_name() == name){
         if (p_found == nullptr)
            p_found = q.front();
         else
            return nullptr;
      }
```

```
      for (auto p_sub_object : q.front() -> p_sub_objects)
         q.push(p_sub_object);
      q.pop();
   }
   return p_found;
}

// 3

cl_base* cl_base::find_obj_root (string name){
   cl_base* p_root_object = this;
   while (p_root_object -> get_head() != nullptr)
      p_root_object = p_root_object -> get_head();
   return p_root_object -> find_obj_current(name);
}

bool cl_base::set_name(string s_new_name){
   if (p_head_object)
      for (auto obj : p_head_object -> p_sub_objects)
         if(obj -> get_name() == s_new_name)
            return false;
   this -> s_name = s_new_name;
   // присвоение дочернему объекту имени, если не имеется других дочерних
объектов с аналогичным именем
   return true;
}

// 4

void cl_base::set_state(int state){
   if (state == 0){
      this -> state = 0;
      for (auto p_sub_objects : p_sub_objects){
         p_sub_objects -> set_state(0);
      }
   }
   else {
      bool par_ready = true;
      cl_base* root_obj = this;

      while (root_obj -> get_head() != nullptr){
         root_obj = root_obj -> get_head();
         if (root_obj -> state == 0){
            par_ready = false;
            break;
         }
      }
      if (par_ready)
         this -> state = state;
   }
}

// 5

void cl_base::print_names_recur(int level){
```

```
   cout << endl;
   for (int i = 0; i < 4*level; i++){
      cout << " ";
   }
   cout << get_name();

   for (auto p_sub_object : p_sub_objects){
      p_sub_object -> print_names_recur(level+1);
   }
}


void cl_base::print_states_recur(int level){
   cout << endl;
   for (int i = 0; i < 4*level; i++){
      cout << " ";
   }
   cout << get_name() << (state == 0 ? " is not ready" : " is ready");

   for (auto p_sub_object : p_sub_objects)
      p_sub_object -> print_states_recur(level+1);
}
// ____________________
cl_base :: ~cl_base()
{
   for (int i = 0; i < p_sub_objects.size(); i++)
   {
      delete p_sub_objects[i]; // очищение вектора подчиненных объектов
   }
   for (int i = 0; i < chains_of_objects.size(); i++)
   {
      delete chains_of_objects[i]; // очищение вектора связей
   }
}

bool cl_base::set_new_parent(cl_base* new_parent)
{
   cl_base* root = this;
   while (root -> p_head_object)
   {
      root = root -> p_head_object;
   }
   if (this == root || new_parent == nullptr)
   {
      return false;
   }
   if (new_parent -> get_sub_object(this -> s_name) != nullptr)
   {
      return false;
   }
   cl_base* p_obj = new_parent;
   while (p_obj != nullptr)
   {
      if (p_obj == this) return false;
      p_obj=p_obj -> get_head();
```

```
   }
   for (int i = 0; i < p_head_object -> p_sub_objects.size(); i++)
   {
      if (p_head_object -> p_sub_objects[i] == this)
      {
         p_head_object      ->      p_sub_objects.erase(p_head_object      ->
p_sub_objects.begin() + i);
         break;
      }
   }
   this -> p_head_object = new_parent;
   new_parent -> p_sub_objects.push_back(this);
   return true;
}

void cl_base::remove_child (string sub_name)
{
   for (int i = 0; i < p_sub_objects.size(); i++)
   {
      if (p_sub_objects[i] -> s_name == sub_name)
      {
         delete p_sub_objects[i];
         p_sub_objects.erase(p_sub_objects.begin() + i);
         return;
      }
   }
}

cl_base* cl_base::get_obj_by_way(string way)
{
   if (way.empty()) return nullptr;
   if (way == ".") return this;
   if (way[0] == '.') return find_obj_current(way.substr(1));
   if (way.substr(0,2) == "//") return find_obj_root(way.substr(2));
   if (way[0] != '/')
   {
      size_t ind = way.find('/');
      cl_base* p_sub = get_sub_object(way.substr(0, ind));
      if (!p_sub || ind == string::npos) return p_sub;
      return p_sub -> get_obj_by_way(way.substr(ind + 1));
   }
   cl_base* root = this;
   while (root->p_head_object)
   {
      root = root -> p_head_object;
   }
   if (way == "/")
   {
      return root;
   }
   return root -> get_obj_by_way(way.substr(1));
}

// ————————————————
```

```
void cl_base::emit_signal(TYPE_SIGNAL p_signal, string& information)
{
   if (this->state == 0) return;
   (this->*p_signal)(information);
   for (int i=0;i<chains_of_objects.size();i++)
   {
      if (chains_of_objects[i]->p_signal == p_signal)
      {
         TYPE_HANDLER p_handler = chains_of_objects[i]->p_handler;
         cl_base* p_destination = chains_of_objects[i]->p_destination;
         if (p_destination->state != 0)
         {
            (p_destination->*p_handler)(information);
         }
      }
   }
}

void   cl_base::set_chain(TYPE_SIGNAL   p_signal,   cl_base*   p_destination,
TYPE_HANDLER p_handler)
{
   for (int i=0;i<chains_of_objects.size();i++)
   {
      if (chains_of_objects[i]->p_signal == p_signal && chains_of_objects[i]-
>p_destination   ==   p_destination&&   chains_of_objects[i]->p_handler   ==
p_handler) return;
   }
   chain* new_chain = new chain();
   new_chain->p_signal = p_signal;
   new_chain->p_destination = p_destination;
   new_chain->p_handler = p_handler;
   chains_of_objects.push_back(new_chain);
}

void  cl_base::remove_chain(TYPE_SIGNAL  p_signal,  cl_base*  p_destination,
TYPE_HANDLER p_handler)
{
   for (int i=0;i<chains_of_objects.size();i++)
   {
      if (chains_of_objects[i]->p_signal == p_signal && chains_of_objects[i]-
>p_destination  ==  p_destination  &&  chains_of_objects[i]->p_handler  ==
p_handler)
      {
         delete chains_of_objects[i];
         chains_of_objects.erase(chains_of_objects.begin() + i);
         return;
      }
   }
}

int cl_base:: get_object_index()
{
   return 0;
}
```

```
void cl_base::set_state_from_actual(int state)
{
   if (get_head() && !get_head() -> state)
      return;
   set_state(state);
   for (int i=0; i < p_sub_objects.size(); i++)
   {
      p_sub_objects[i] -> set_state_from_actual(state);
   }
}

string cl_base::get_coordinate_from_root()
{
   if (get_head()==nullptr)
   {
      return "/";
   }
   cl_base* ptr=this;
   string way;
   while(ptr->get_head()!=nullptr)
   {
      way="/"+ptr->s_name+way;
      ptr=ptr->get_head();
   }
   return way;
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H

#include <iostream>
#include <vector>
#include <string>
#include <queue>
#include <stack>

#define SIGNAL_D(signal_f)(TYPE_SIGNAL)(&signal_f)
#define HANDLER_D(handler_f)(TYPE_HANDLER)(&handler_f)

using namespace std;

class cl_base;
typedef void(cl_base::*TYPE_SIGNAL)(string&);
typedef void(cl_base::*TYPE_HANDLER)(string);

struct chain
```

```
{
   TYPE_HANDLER p_handler;
   TYPE_SIGNAL p_signal;
   cl_base* p_destination;
};

class cl_base
{
private:
   string s_name;
   cl_base* p_head_object;
   vector <cl_base*> p_sub_objects;
   int state;

   vector<chain*> chains_of_objects;

public:
   cl_base(cl_base* p_head_object, string s_name = "Base object");

   bool set_name(string s_new_name);
   string get_name();
   cl_base* get_head();

   void set_state(int state);

   cl_base* get_sub_object(string s_name);
   cl_base* find_obj_current(string name);
   cl_base* find_obj_root(string name);

   void print_names_recur(int level = 0);
   void print_states_recur(int level = 0);

   ~cl_base();

   bool set_new_parent(cl_base* new_parent);
   void remove_child(string sub_name);
   cl_base* get_obj_by_way(string way);

   void emit_signal(TYPE_SIGNAL p_signal, string& information);
   void set_chain(TYPE_SIGNAL p_signal, cl_base* p_destination, TYPE_HANDLER
p_handler);
   void    remove_chain(TYPE_SIGNAL    p_signal,    cl_base*    p_destination,
TYPE_HANDLER p_handler);
   string get_coordinate_from_root();
   virtual int get_object_index();
   void set_state_from_actual(int state);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_base.h"
#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 объектов
   return ob_cl_application.exec_app(); // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 27.

Таблица 27 – 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
root / A 2 / B 3 endtree / /A /A /B	Object tree root A B Signal from / Signal from /	Object tree root A B Signal from / Signal from /

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
end_of_connections SET_CONDITION /A 0 EMIT / Test message SET_CONDITION /A 1 EMIT / Test again END	Signal to /A Text: Test again (class: 1)	Signal to /A Text: Test again (class: 1)
appls_root / obj_X 4 / obj_Y 5 endtree /obj_X /obj_Y /obj_X /obj_Y end_of_connections EMIT /obj_X "Message before delete" DELETE_CONNECT /obj_X /obj_Y EMIT /obj_X "Message after delete" END	Object tree appls_root obj_X obj_Y Signal from /obj_X Signal to /obj_Y Text: "Message before delete" (class: 4) Signal from /obj_X	Object tree appls_root obj_X obj_Y Signal from /obj_X Signal to /obj_Y Text: "Message before delete" (class: 4) Signal from /obj_X
root / A 2 / B 3 endtree / /A end_of_connections SET_CONNECT / /Z END	Object tree root A B Handler object /Z not found	Object tree root A B Handler object /Z not found

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).