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

Для решения задачи используется:

- SIGNAL_D - Макрос для приведения типа к TYPE_SIGNAL - указателя на метод сигнала;
- HANDLER_D - Макрос для приведения типа к TYPE_HANDLER - указателя на метод обработчика;
- TYPE_SIGNAL - тип класса cl_base указатель на метод сигнала;
- TYPE_HANDLER - тип класса cl_base указатель на метод обработчика;
- typedef - декларатор новых типов;
- define - препроцессор макроса;
- getline - функция получения полной строки из стандартного потока ввода.

Класс cl_base:

- свойства/поля:
    - поле Вектор связей:
        - наименование — connects;
        - тип — vector<o_sh*>;
        - модификатор доступа — private;
    - поле Структура установленной связи:
        - наименование — o_sh;
        - тип — struct;
        - модификатор доступа — private;
- функционал:
    - метод signal — Виртуальный метод сигнала для переопределения наследниками;
    - метод handler — Виртуальный метод обработчика для переопределения наследниками;

- метод get_path — Метод получения строки абсолютной координаты пути до объекта;
- метод set_connect — Метод установки связи;
- метод remove_connect — Метод разрыва связи;
- метод emit_signal — Метод отработки сигнала.

Класс cl_application:

- функционал:
  - метод signal — Метод сигнала;
  - метод handler — Метод обработчика;
  - метод exec_app — Метод запуска системы;
  - метод build_tree_objects — Метод построения дерева объектов.

Класс cl_obj2:

- функционал:
  - метод signal — Метод сигнала;
  - метод handler — Метод обработчика.

Класс cl_obj3:

- функционал:
  - метод signal — Метод сигнала;
  - метод handler — Метод обработчика.

Класс cl_obj4:

- функционал:
  - метод signal — Метод сигнала;
  - метод handler — Метод обработчика.

Класс cl_obj5:

- функционал:
  - метод signal — Метод сигнала;
  - метод handler — Метод обработчика.

Класс cl_obj6:

- функционал:
  - метод signal — Метод сигнала;
  - метод handler — Метод обработчика.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс - объект дерева	
		cl_application	public		2
		cl_obj2	public		3
		cl_obj3	public		4
		cl_obj4	public		5
		cl_obj5	public		6
		cl_obj6	public		7
2	cl_application			Класс с функционалом запуска системы и корень дерева	
3	cl_obj2			Второй класс объект дерева	
4	cl_obj3			Третий класс объект дерева	
5	cl_obj4			Четвёртый класс объект дерева	
6	cl_obj5			Пятый класс объект дерева	
7	cl_obj6			Шестой класс объект дерева	

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 signal класса cl_base

Функционал: Виртуальный метод сигнала для переопределения наследниками.

Параметры: string& message - ссылка на сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода signal класса cl_base

№	Предикат	Действия	№ перехода
1		Виртуальный метод - нулевое определение	∅

3.2 Алгоритм метода handler класса cl_base

Функционал: Виртуальный метод обработчика для переопределения наследниками.

Параметры: string message - сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода handler класса cl_base

№	Предикат	Действия	№ перехода
1		Виртуальный метод - нулевое определение	∅

3.3 Алгоритм метода set_connect класса cl_base

Функционал: Метод установки связи.

Параметры: TYPE_SIGNAL p_signal - тип сигнала, cl_base* p_object - целевой объект, TYPE_HANDLER p_ob_handler - тип обработчика.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_connect класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление указателя на структуру o_sh p_value	2
2		Инициализация переменной i = 0 типа int	3
3	i < размер connects		4
			5
4	Значения структуры в connects[i] совпадают с параметрами p_signal p_object P_ob_handler		∅
		i++	3
5		Выделение памяти структуре p_value	6
6		Значение p_signal структуры p_value = параметру p_signal	7
7		Значение p_signal структуры p_cl_base = параметру p_object	8
8		Значение p_handler структуры p_value = параметру p_ob_handler	9
9		Добавлене p_value в конец connects	∅

3.4 Алгоритм метода remove_connect класса cl_base

Функционал: Метод разрыва связи.

Параметры: TYPE_SIGNAL p_signal - тип сигнала, cl_base* p_object - целевой объект, TYPE_HANDLER p_ob_handler - тип обработчика.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода remove_connect класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация итератора i по вектору connects автоматического типа	2
2	i в connects	Инициализация указателя c = указателем на i типа o_sh	3
			∅
3	Значения структуры с p_cl_base p_handler p_signal совпадают с параметрами p_signal p_object p_ob_handler	Удаление элемента по итератору i из connects	4
		Итерация i	2
4		Освобождение памяти по указателю c с помощью оператора delete	∅

3.5 Алгоритм метода emit_signal класса cl_base

Функционал: Метод отработки сигнала.

Параметры: TYPE_SIGNAL p_signal - тип сигнала, string& message - ссылка на сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода emit_signal класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление переменной p_handler типа TYPE_SIGNAL	2
2		Объявление указателя p_object типа cl_base	3
3	поле state = 0		∅
			4
4		Вызов метода сигнала по указателю p_signal с параметром s_command	5
5		Инициализация переменной i = 0 типа int	6
6	i < размер connects		7
			∅
7	значение p_signal в структуре connects[i] = параметру p_signal	p_handler = значение p_handler в структуре connects[i]	8
			10
8		p_object = значение p_cl_base в структуре connects[i]	9
9	значение поля state у объекта p_object != 0	Вызов метода обработчика по указателю p_hanlder с параметром s_command	10
			10
10		i++	6

3.6 Алгоритм метода signal класса cl_application

Функционал: Метод сигнала.

Параметры: string& message - ссылка на сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода signal класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " + результат вызова метода get_path	2
2		Прибавление строки " (class: 1)" в конец message	∅

3.7 Алгоритм метода handler класса cl_application

Функционал: Метод обработчика.

Параметры: string& message - сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода handler класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Signal to " + результат вызова метода get_path + " Text: " + значение message	∅

3.8 Алгоритм метода get_path класса cl_base

Функционал: Метод получения строки абсолютной координаты пути до объекта.

Параметры: нет.

Возвращаемое значение: string - абсолютная координата до объекта.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_path класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация переменной path типа string пустой	2

№	Предикат	Действия	№ перехода
		строкой	
2		Инициализация указателя p_obj типап cl_base текущим объектом	3
3	Головной объект у p_obj не нулевой	path = "/" + значение вызова метода get_name у объекта p_obj + path	4
			5
4		p_obj = значение вызова get_header у p_obj	3
5	path пустая строка	path = "/"	6
			6
6		Возврат path	∅

3.9 Алгоритм метода signal класса cl_obj2

Функционал: Метод сигнала.

Параметры: string& message - ссылка на сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода signal класса cl_obj2

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " + результат вызова метода get_path	2
2		Прибавление строки " (class: 2)" в конец message	∅

3.10 Алгоритм метода handler класса cl_obj2

Функционал: Метод обработчика.

Параметры: string& message - сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода handler класса cl_obj2

№	Предикат	Действия	№ перехода
1		Вывод "Signal to " + результат вызова метода get_path + " Text: " + значение message	∅

3.11 Алгоритм метода signal класса cl_obj3

Функционал: Метод сигнала.

Параметры: string& message - ссылка на сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода signal класса cl_obj3

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " + результат вызова метода get_path	2
2		Прибавление строки " (class: 3)" в конец message	∅

3.12 Алгоритм метода handler класса cl_obj3

Функционал: Метод обработчика.

Параметры: string& message - сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода handler класса cl_obj3

№	Предикат	Действия	№ перехода
1		Вывод "Signal to " + результат вызова метода get_path + " Text: " + значение message	∅

3.13 Алгоритм метода signal класса cl_obj4

Функционал: Метод сигнала.

Параметры: string& message - ссылка на сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода signal класса cl_obj4

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " + результат вызова метода get_path	2
2		Прибавление строки " (class: 4)" в конец message	∅

3.14 Алгоритм метода handler класса cl_obj4

Функционал: Метод обработчика.

Параметры: string& message - сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода handler класса cl_obj4

№	Предикат	Действия	№ перехода
1		Вывод "Signal to " + результат вызова метода get_path + " Text: " + значение message	∅

3.15 Алгоритм метода signal класса cl_obj5

Функционал: Метод сигнала.

Параметры: string& message - ссылка на сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода signal класса cl_obj5

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " + результат вызова метода get_path	2
2		Прибавление строки " (class: 5)" в конец message	∅

3.16 Алгоритм метода handler класса cl_obj5

Функционал: Метод обработчика.

Параметры: string& message - сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода handler класса cl_obj5

№	Предикат	Действия	№ перехода
1		Вывод "Signal to " + результат вызова метода get_path + " Text: " + значение message	∅

3.17 Алгоритм метода signal класса cl_obj6

Функционал: Метод сигнала.

Параметры: string& message - ссылка на сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода signal класса cl_obj6

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " + результат вызова метода get_path	2
2		Прибавление строки " (class: 6)" в конец message	∅

3.18 Алгоритм метода handler класса cl_obj6

Функционал: Метод обработчика.

Параметры: string& message - сообщение.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода handler класса cl_obj6

№	Предикат	Действия	№ перехода
1		Вывод "Signal to " + результат вызова метода get_path + " Text: " + значение message	∅

3.19 Алгоритм метода exec_app класса cl_application

Функционал: Метод запуска системы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1	<...> Обработка команд из КВ-3 и объявление необходимых переменных		2
			21
2	cmd = "EMIT"		3
			7
3		Объявление переменной message типа string	4
4		Получение строки ввода message с помощью getline	5

№	Предикат	Действия	№ перехода
5		Удаление первого символа в message	6
6	p_obj нулеой указатель	Вывод "Object " + coord + " not found"	1
		Вызов метода emit_signal у объекта p_obj с параметрами: Пердача указателя на метод signal у cl_base макросу SIGNAL_D, message	1
7	cmd = "SET_CONNECT"		8
			12
8		Объявление переменной t_coord типа string	9
9		Ввод значения t_coord	10
10		Инициализация указателя target типа cl_base значение вызова метода find_by_coord с параметром t_coord	11
11	p_obj нулеой указатель	Вывод "Object " + coord + " not found"	1
	target нулевой указатель	Вывод "Handler object " + t_coord + " not found"	1
		Вызов метода set_connect у объекта p_obj с параметрами: Пердача указателя на метод signal у cl_base макросу SIGNAL_D, target, Пердача указателя на метод handler у cl_base макросу HANDLER_D	1
12	cmd = "DELETE_CONNECT"		13
			17
13		Объявление переменной t_coord типа string	14
14		Ввод значения t_coord	15
15		Инициализация указателя target типа cl_base значение вызова метода find_by_coord с параметром t_coord	16
16	p_obj нулеой указатель	Вывод "Object " + coord + " not found"	1
	target нулевой указатель	Вывод "Handler object " + t_coord + " not found"	1
		Вызов метода remove_connect у объекта p_obj с	1

№	Предикат	Действия	№ перехода
		параметрами: Пердача указателя на метод signal у cl_base макросу SIGNAL_D, target, Пердача указателя на метод handler у cl_base макросу HANDLER_D	
17	cmd = "SET_CONDITION"		18
			1
18		Объявление переменной s типа int	19
19		Ввод значения s	20
20	p_obj нулеой указатель	Вывод "Object " + coord + " not found"	1
		Вызов метода set_state у объекта p_obj с параметром s	1
21		Возврат значения 0	∅

3.20 Алгоритм метода build_tree_objects класса cl_application

Функционал: Метод построения дерева объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		<...> Повторение построения дерева из КВ-3	2
2		Установка состояния 1 для создаваемых объектов посредством метода set_state с параметром 1	3
3		Вызов метода set_state с параметром 1	4
4		Объявление переменных emitter_path, target_path типа string	5
5		Ввод значения emitter_path	6

№	Предикат	Действия	№ перехода
6	emitter_path="end_of_connections"		∅
		Ввод значения target_path	7
7		Инициализация указателя emitter типа cl_base значение вызова метода find_by_coord с параметром emitter_path	8
8		Инициализация указателя target типа cl_base значение вызова метода find_by_coord с параметром target_path	9
9		Вызов метода set_connect у объекта emitter с параметрами: Пердача указателя на метод signal у cl_base макросу SIGNAL_D, target, Пердача указателя на метод handler у cl_base макросу HANDLER_D	5

3.21 Алгоритм функции main

Функционал: Главная функиця программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 22.

Таблица 22 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		<...> Повторениие кода из КВ-3	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-11.

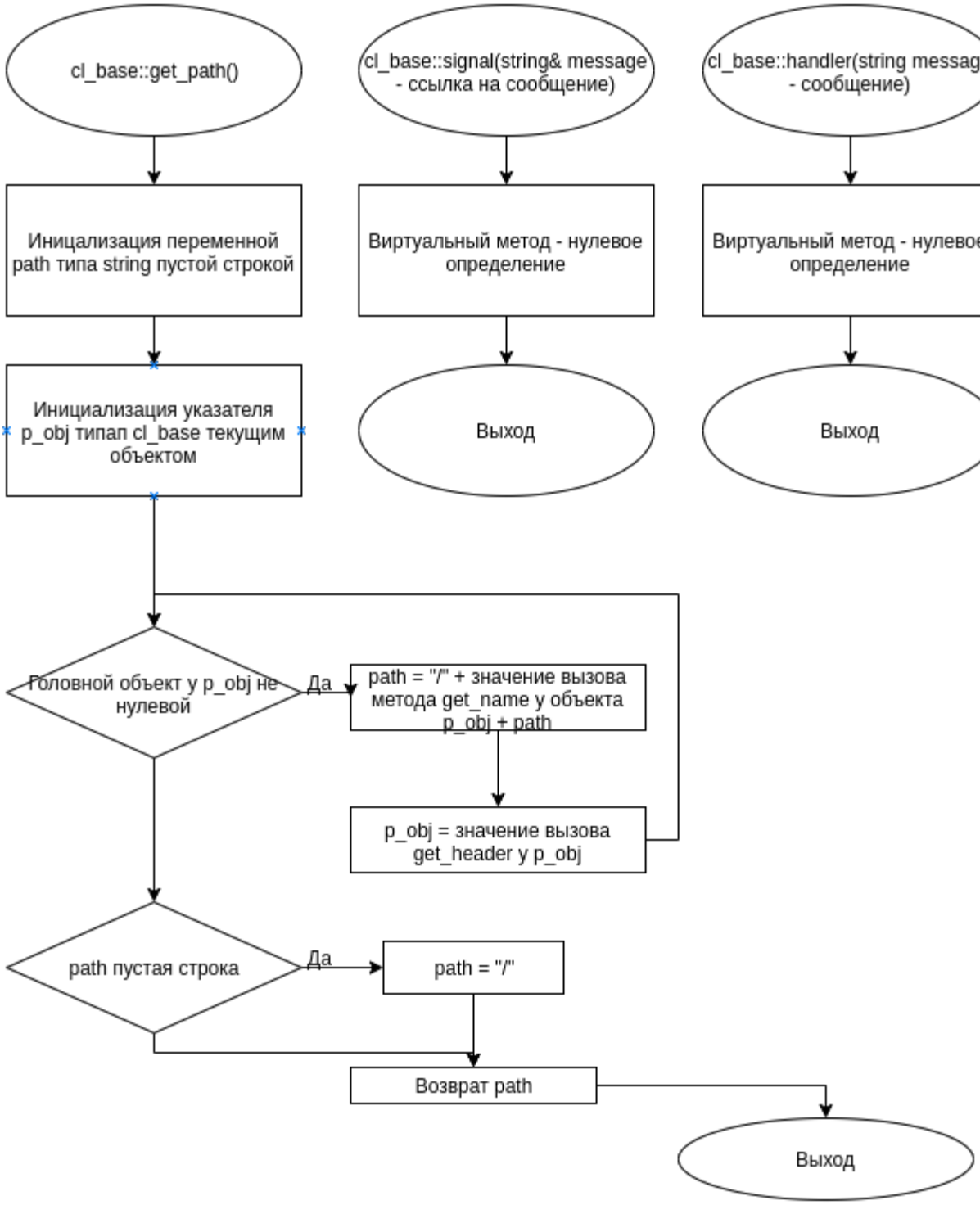

Рисунок 1 – Блок-схема алгоритма

Рисунок 2 – Блок-схема алгоритма

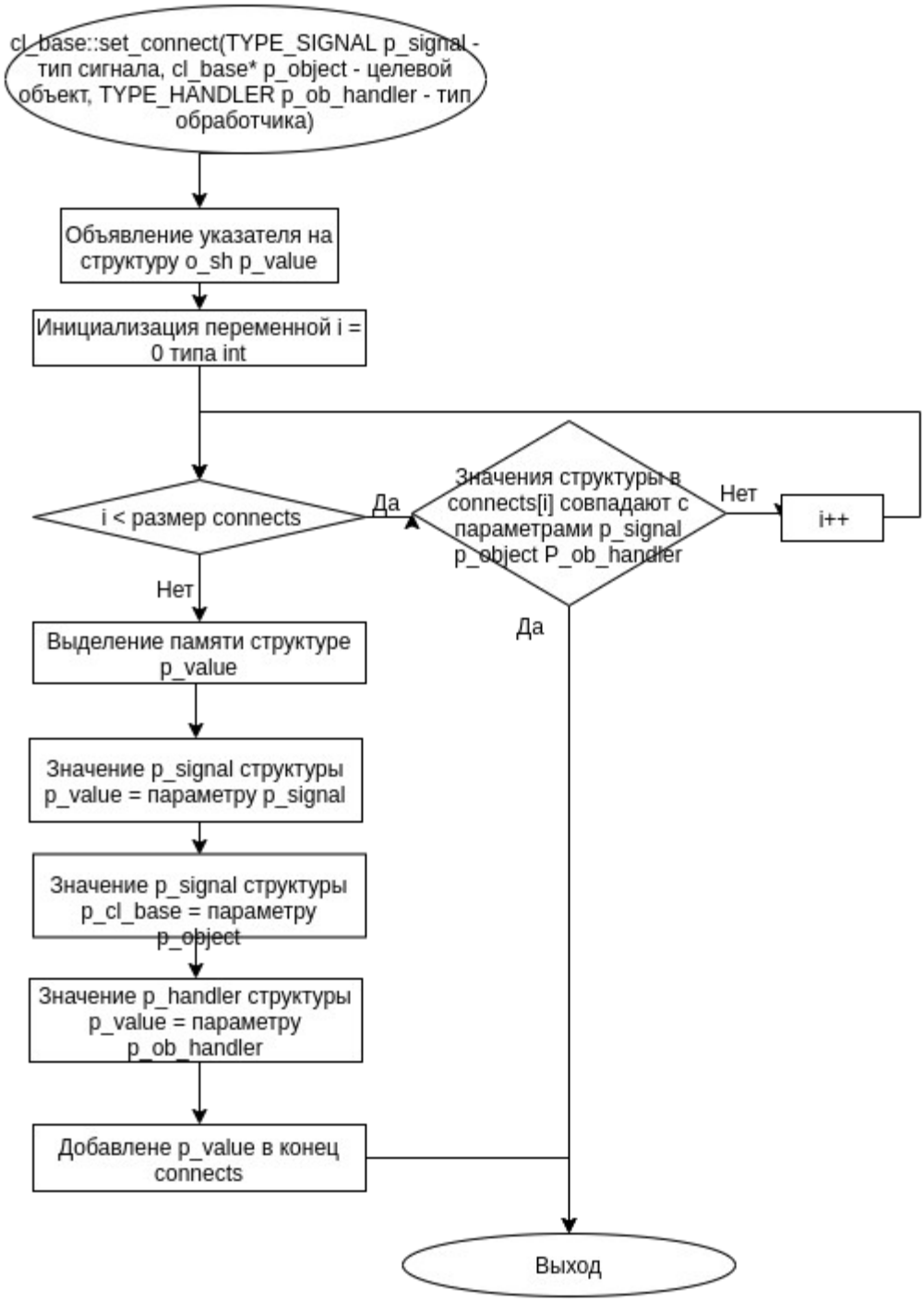

Рисунок 3 – Блок-схема алгоритма

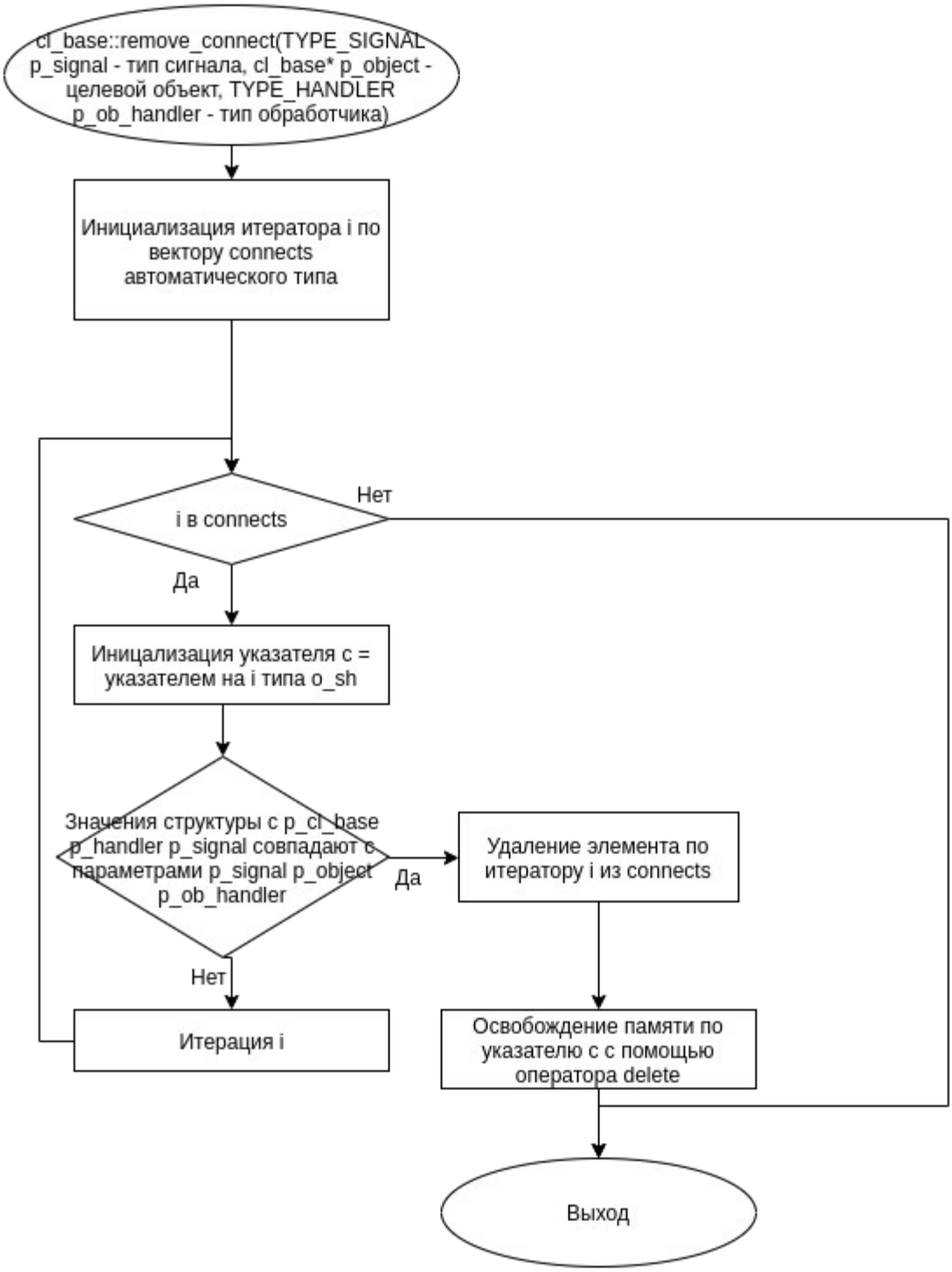

Рисунок 4 – Блок-схема алгоритма

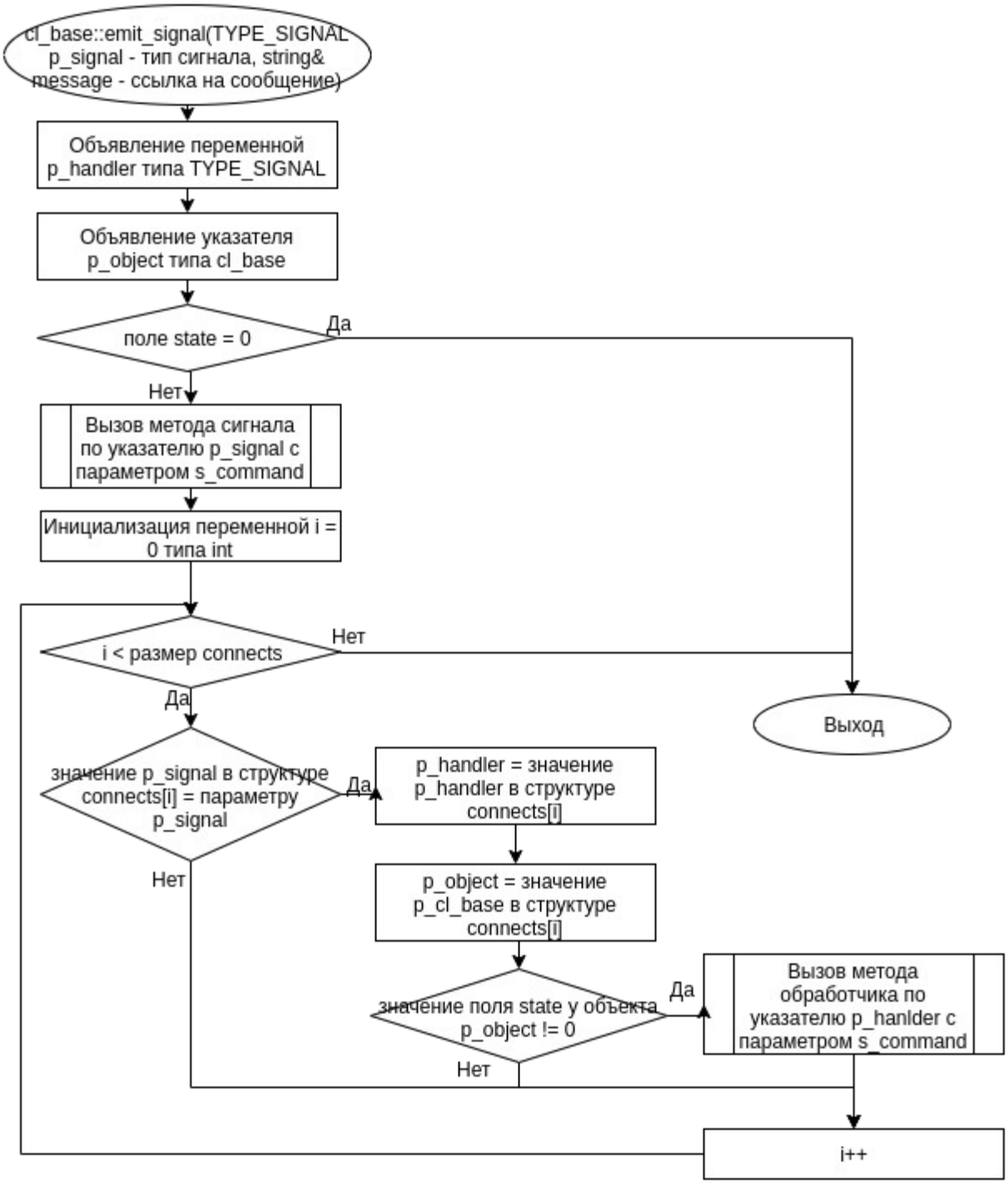

Рисунок 5 – Блок-схема алгоритма

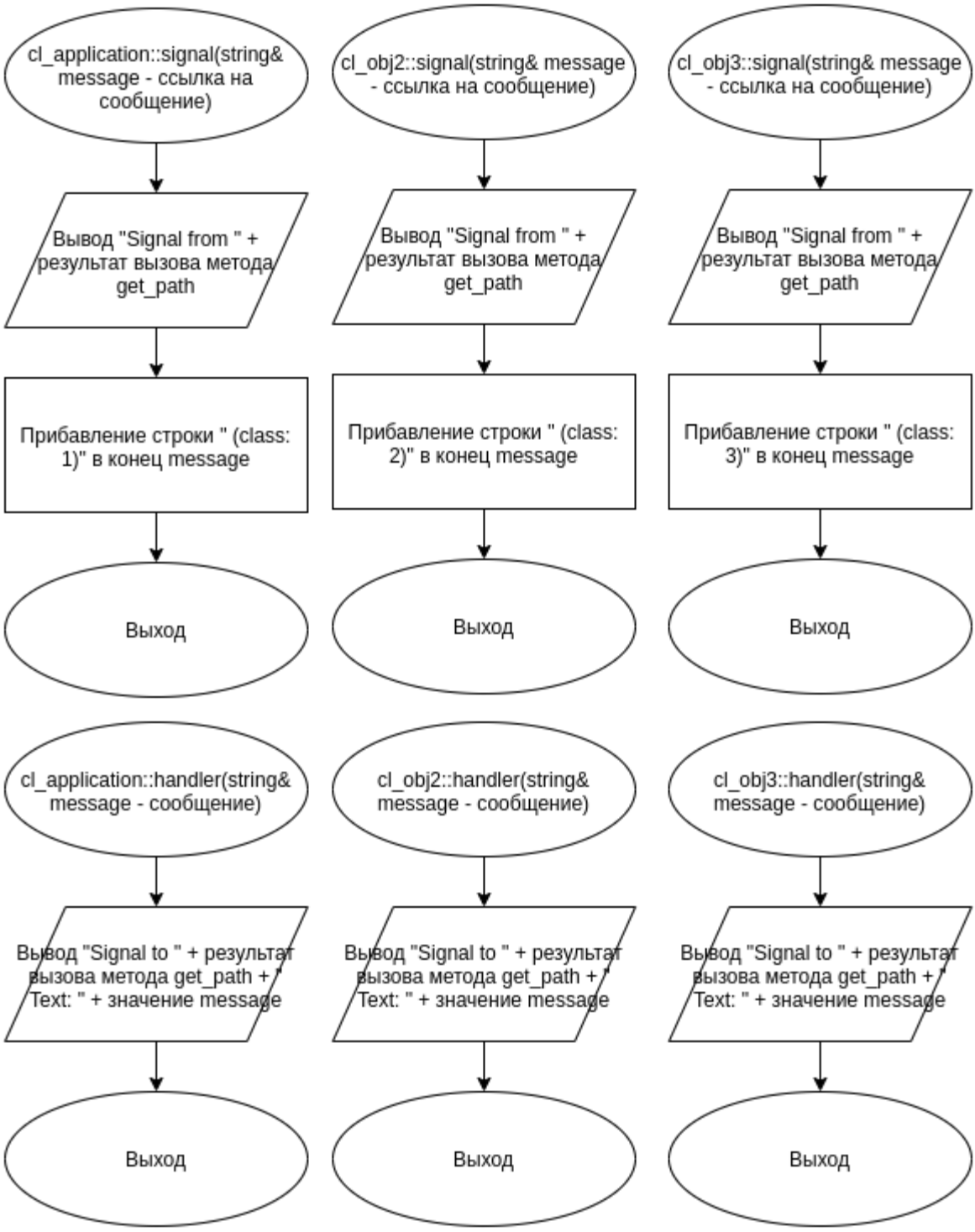

Рисунок 6 – Блок-схема алгоритма

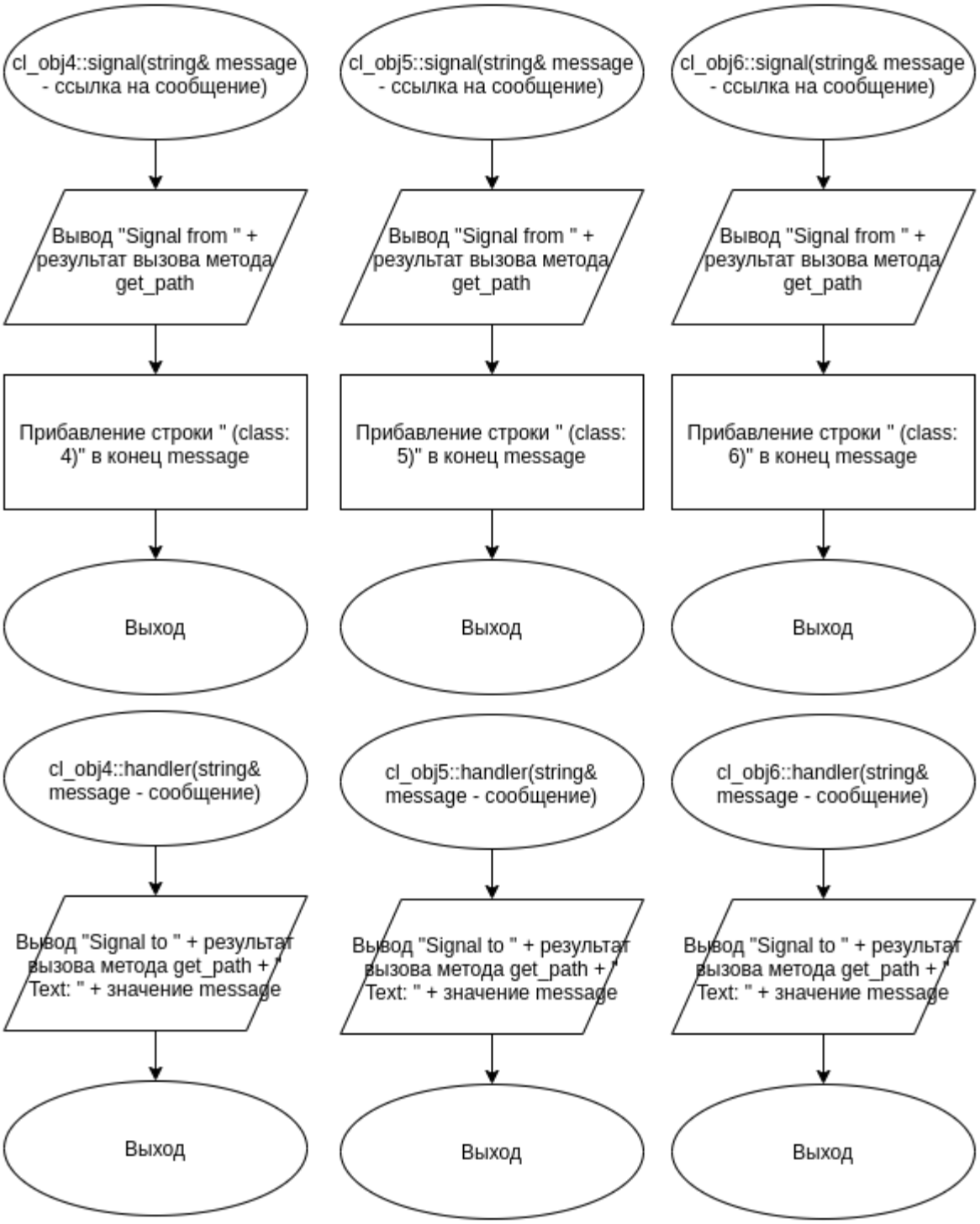

Рисунок 7 – Блок-схема алгоритма

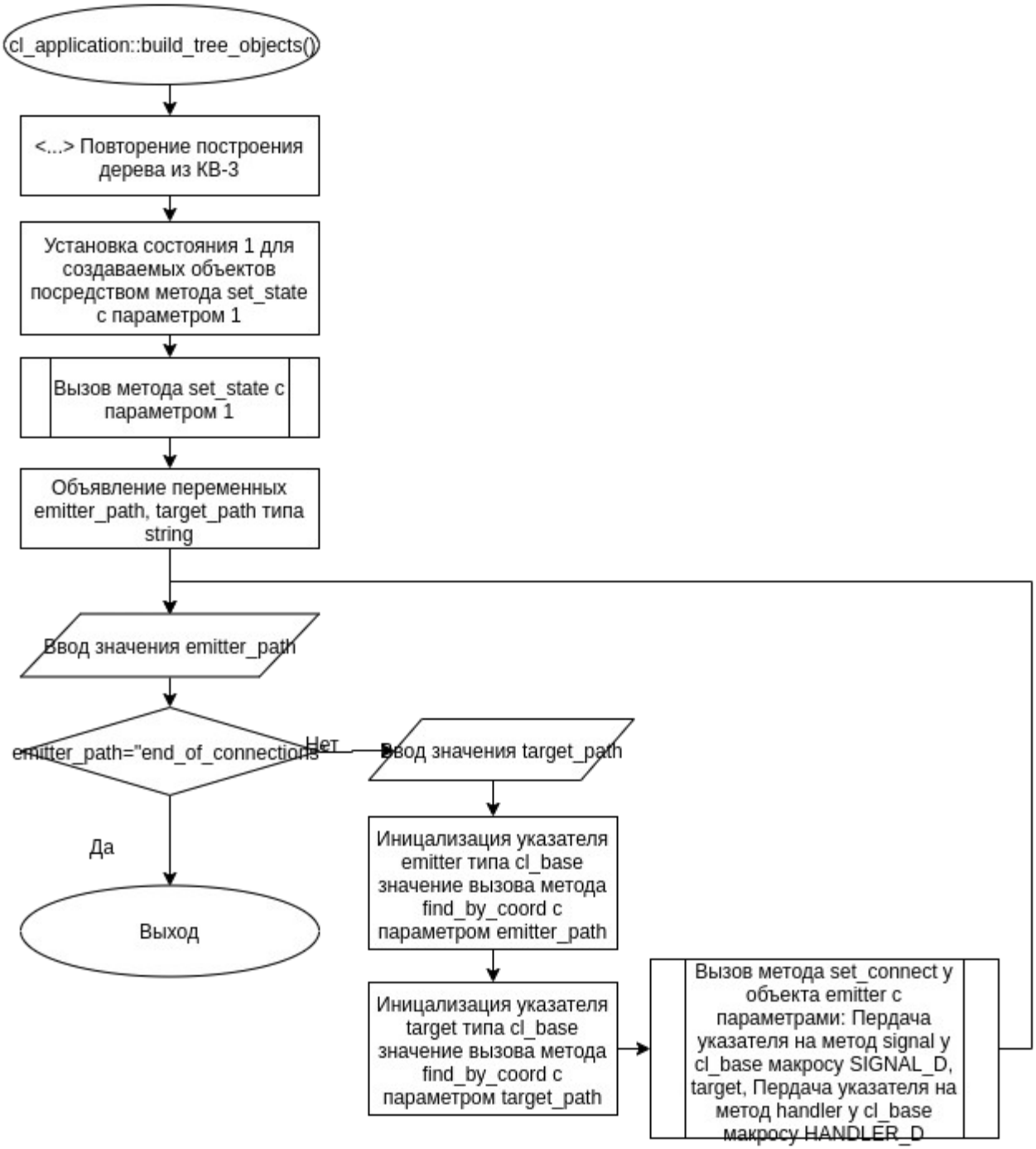

Рисунок 8 – Блок-схема алгоритма

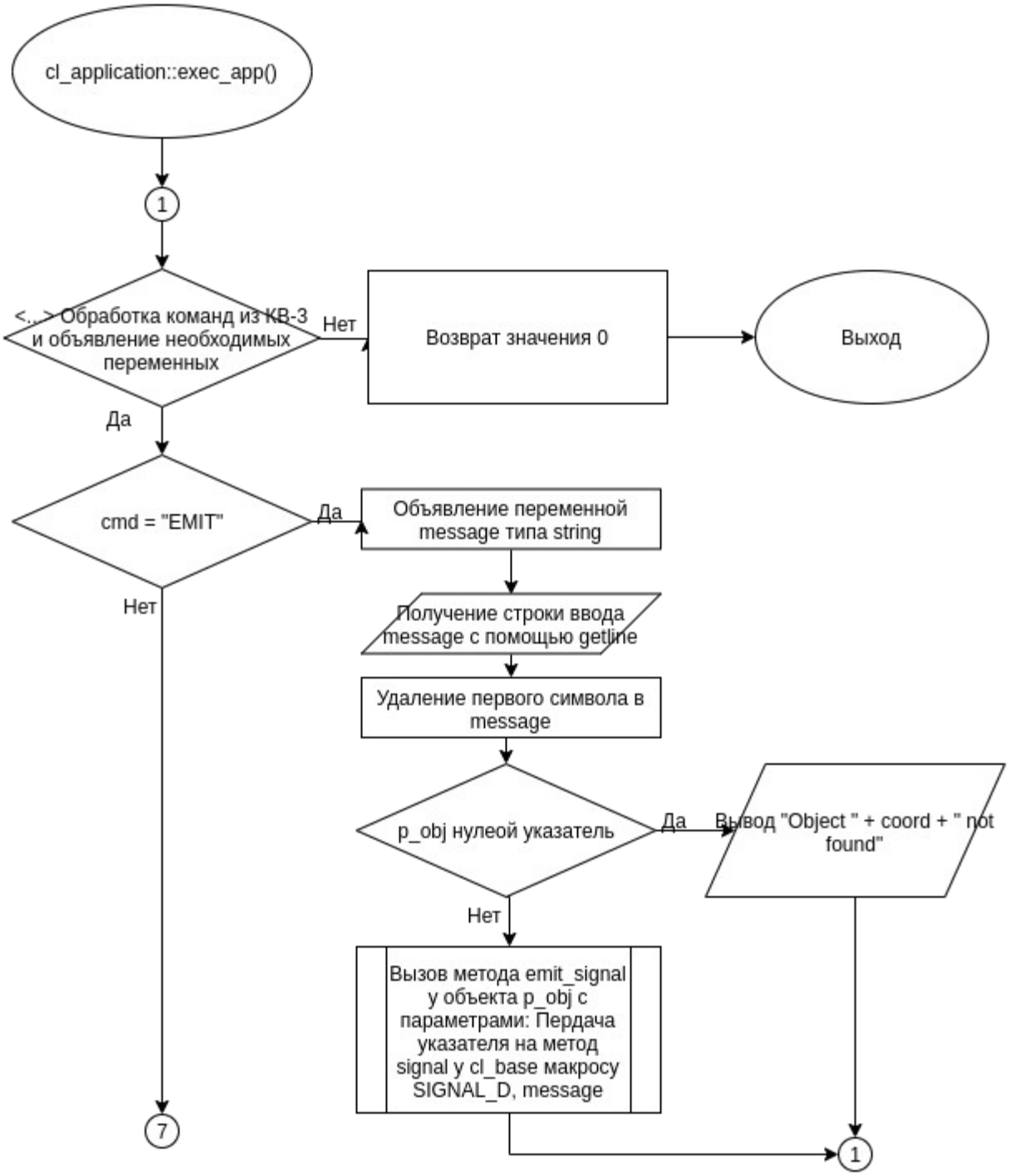

Рисунок 9 – Блок-схема алгоритма

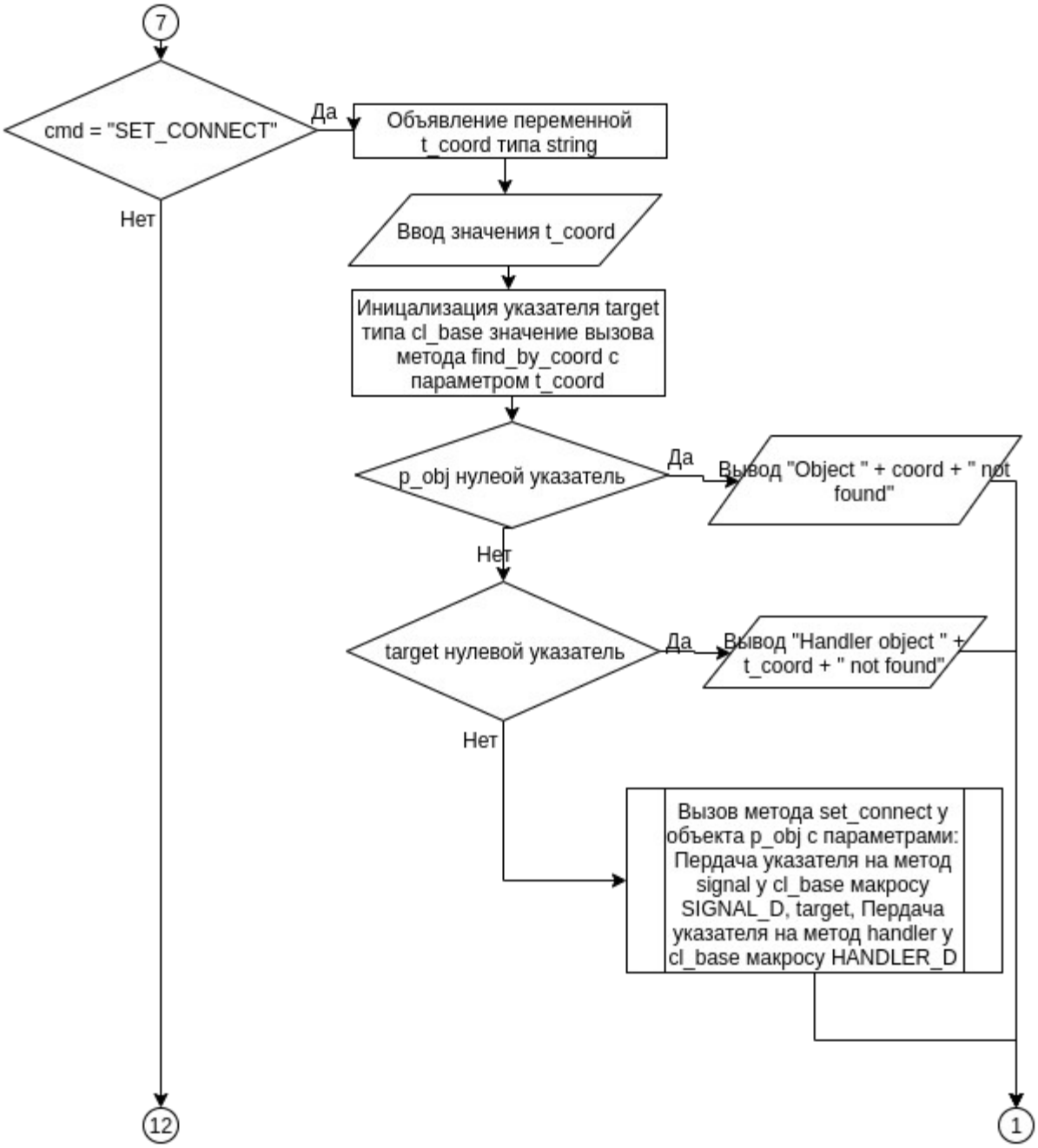

Рисунок 10 – Блок-схема алгоритма

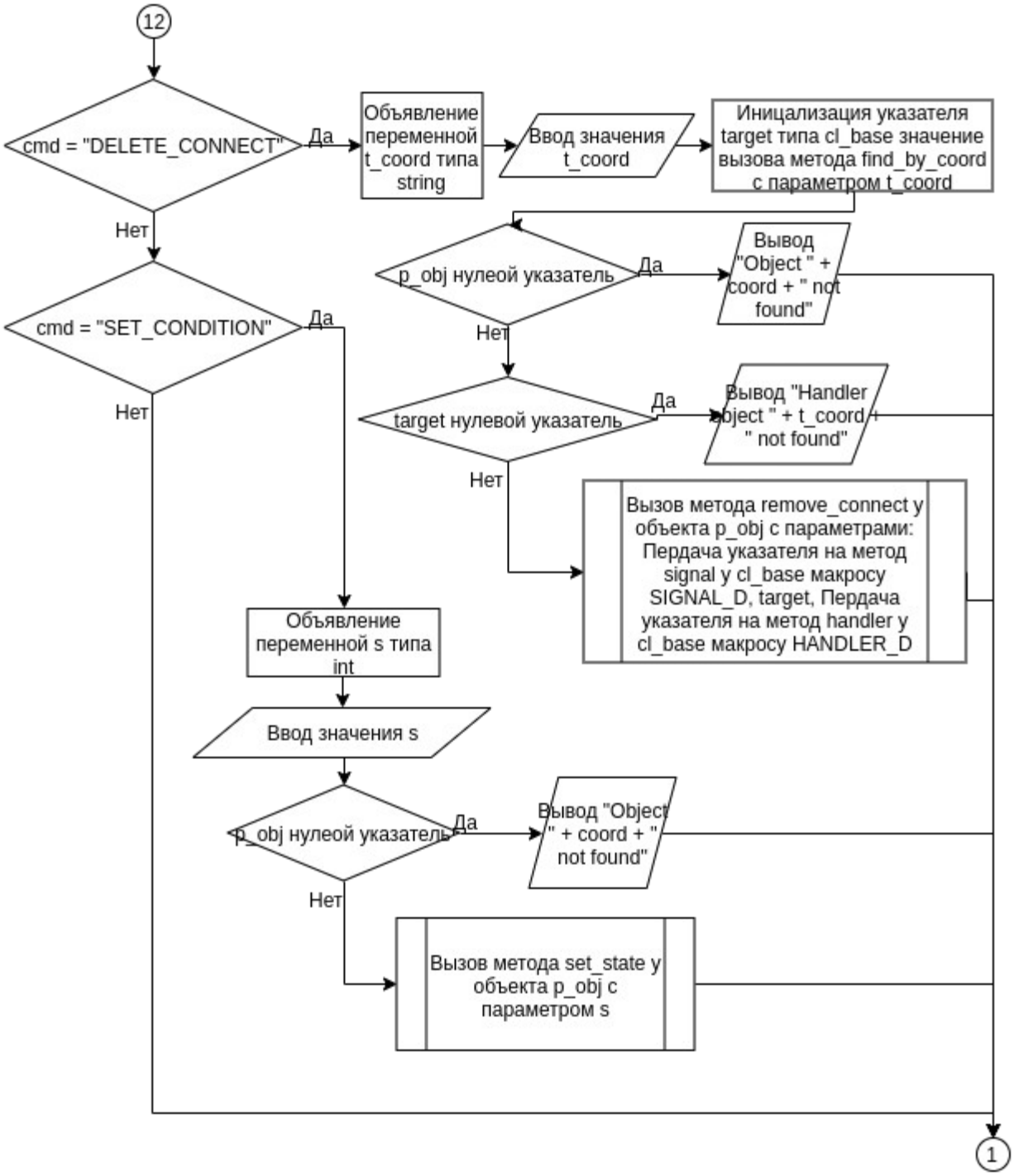

Рисунок 11 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_application.cpp

Листинг 1 – cl_application.cpp

```
#include "cl_application.h"


cl_application::cl_application(cl_base* header) : cl_base(header) {};


void cl_application::signal(string& message) {
   cout << "Signal from " << get_path() << endl;
   message += " (class: 1)";
};

void cl_application::handler(string message) {
   cout << "Signal to " << get_path() << " Text:  " << message << endl;
};


void cl_application::build_tree_objects() {
   cout << "Object tree" << endl;

   string parentCoord, childName;
   int childClass;

   cin >> parentCoord;
   this->edit_name(parentCoord);

   // Ввод дерева
   while(true)
   {
      cin >> parentCoord;
      if(parentCoord == "endtree")
         break;

      cl_base* parent = find_by_coord(parentCoord);

      if(!parent) {
         print();
         cout << endl << "The head object " << parentCoord << " is not
found";
         exit(1);
```

```
         }

         cin >> childName >> childClass;

         if(!parent->get_child(childName))
         {
            cl_base* ch;
            switch(childClass)
            {
               case 2:
               {
                  ch = new cl_obj2(parent, childName);
                  break;
               }
               case 3:
               {
                  ch = new cl_obj3(parent, childName);
                  break;
               }
               case 4:
               {
                  ch = new cl_obj4(parent, childName);
                  break;
               }
               case 5:
               {
                  ch = new cl_obj5(parent, childName);
                  break;
               }
               case 6:
               {
                  ch = new cl_obj6(parent, childName);
                  break;
               }
            }
            ch->set_state(1);
         }
         else {
            cout << parentCoord << "       Dubbing the names of subordinate
objects" << endl;
         }
      }
      // Ввод дерева
      set_state(1); // Установка готовности корневому объекту

      // Ввод связей
      string  emitter_path, target_path;
      while(true) {
         cin >> emitter_path;
         if(emitter_path == "end_of_connections") break;
         cin >> target_path;

         cl_base* emitter = find_by_coord(emitter_path);
         cl_base* target = find_by_coord(target_path);
```

```
      emitter->set_connect(SIGNAL_D(cl_base::signal),                  target,
HANDLER_D(cl_base::handler));
   }
   // Ввод связей
};
int cl_application::exec_app()
{
   print();
   cout << endl;

   cl_base* current = this;

   // Обработка команд
   while(true){
      string cmd, coord;
      cin >> cmd;
      if(cmd == "END") break;
      cin >> coord;

      cl_base* p_obj = current->find_by_coord(coord);
      if(cmd == "SET") {
         if(p_obj){
            current = p_obj;
            cout << "Object is set: " << current->get_name() << endl;
         }
         else {
            cout << "The object was not found at the specified coordinate: "
<< coord << endl;
         }
      }
      else if(cmd == "FIND") {
         if(p_obj){
            cout << coord << "       Object name: " << p_obj->get_name() <<
endl;
         }
         else {
            cout << coord << "     Object is not found" << endl;
         }
      }
      else if(cmd == "MOVE"){
         if(!p_obj) {
            cout << coord << " Head object is not found" << endl;
         }
         else if(p_obj->get_header() != current->get_header()
               && current->find_on_branch(p_obj->get_name()))
         {
            cout << coord << "       Redefining the head object failed" <<
endl;
         }
         else if (p_obj->get_child(current->get_name())) {
            cout << coord << "     Dubbing the names of subordinate objects"
<< endl;
         }
         else if (!current->change_header(p_obj)){
            cout << coord << "       Redefining the head object failed" <<
```

```
      endl;
            }
            else {
               cout << "New head object: " << p_obj->get_name() << endl;
            }
         }
         else if(cmd == "DELETE") {
            if(p_obj) {
               cl_base* fheader = p_obj->get_header();
               string path;
               current->delete_by_name(p_obj->get_name());
               while(fheader != nullptr) {
                  if(fheader->get_header()) path += "/" + fheader->get_name();
                  fheader = fheader->get_header();
               }
               cout << "The object " << path + "/" + coord << " has been
      deleted" << endl;
            }
         }
         // КВ-4 Команды сигналы
         else if(cmd == "EMIT") {
            string message;
            getline(cin, message);
            message.erase(0, 1); // Убираем лишний пробел в начале
            if(!p_obj){
               cout << "Object " << coord << " not found" << endl;
            }
            else {
               p_obj->emit_signal(SIGNAL_D(cl_base::signal), message);
            }

         }
         else if(cmd == "SET_CONNECT") {
            string t_coord;
            cin >> t_coord;
            cl_base* target = find_by_coord(t_coord);
            if(!p_obj){
               cout << "Object " << coord << " not found" << endl;
            }
            else {
               if (!target) {
                  cout << "Handler object " << t_coord << " not found" << endl;
               }
               else {
                  p_obj->set_connect(SIGNAL_D(cl_base::signal),          target,
      HANDLER_D(cl_base::handler));
               }
            }
         }
         else if(cmd == "DELETE_CONNECT") {
            string t_coord;
            cin >> t_coord;
            cl_base* target = find_by_coord(t_coord);
            if(!p_obj) {
               cout << "Object " << coord << " not found" << endl;
```

```
            }
            else {
               if (!target) {
                  cout << "Handler object " << t_coord << " not found" << endl;
               }
               else {
                  p_obj->remove_connect(SIGNAL_D(cl_base::signal),       target,
HANDLER_D(cl_base::handler));
               }
            }
         }
         else if(cmd == "SET_CONDITION") {
            int s;
            cin >> s;
            if(!p_obj) {
               cout << "Object " << coord << " not found" << endl;
            }
            else {
               p_obj->set_state(s);
            }
         }
         // КВ-4 Команды сигналы
      }
      // Обработка команд
      return 0;
};
```

5.2 Файл cl_application.h

Листинг 2 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
#include "cl_obj2.h"
#include "cl_obj3.h"
#include "cl_obj4.h"
#include "cl_obj5.h"
#include "cl_obj6.h"

class cl_application : public cl_base {
public:
   cl_application(cl_base* header);
   void build_tree_objects();
   int exec_app();

   void signal(string& message);
   void handler(string message);
};
```

```
#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"


cl_base::cl_base(cl_base* header, string name){
   this->name = name;
   this->header_ptr = header;
   if(header_ptr){
      header_ptr-> children_ptr.push_back(this);
   }
}

bool cl_base::edit_name(string new_name){
   this->name = new_name;
   return true;
}

string cl_base::get_name(){
   return this->name;
}

cl_base* cl_base::get_header(){
   return this->header_ptr;
}

cl_base* cl_base::get_child(string name){
   for(cl_base* child : children_ptr) {
      if(child->name == name) return child;
   }
   return nullptr;
}

cl_base* cl_base::find_on_branch(string name, int* count) {
   int c = 0;
   if(count == nullptr){
      count = &c;
   }

   cl_base* fchild = nullptr;
   if(this->name == name){
      fchild = this;
      (*count)++;
   }

   for(cl_base* child : children_ptr) {
```

```
            cl_base* f = child->find_on_branch(name, count);
            if(f)
               fchild = f;
         }

         if( (*count)==1 && fchild) {
            return fchild;
         }
         return nullptr;
      }

      cl_base* cl_base::find_on_tree(string name) {
         cl_base* fheader = this;
         while(fheader->get_header()){
            fheader = fheader->get_header();
         }
         return fheader->find_on_branch(name);
      }

      // КВ-3
      bool cl_base::change_header(cl_base* new_header) {
         // Нельзя переопределять корневой и создавать новый корень
         if(header_ptr == nullptr || new_header == nullptr)
            return false;

         // У нового головного нельзя чтобы появились два подчиненных объекта с
      одинаковым наименованием.
         if(new_header->get_child(name))
            return false;


         // Новый объект Не должен принадлежать к ветке текущего
         cl_base* current = new_header;
         while(current->get_header()){
            if(current == this) return false;
            current = current->get_header();
         }


         // Удаляем у текущего родителя текущий объект из списка подчинённых
         this->header_ptr->children_ptr.erase(find(this->header_ptr-
      >children_ptr.begin(), this->header_ptr->children_ptr.end(), this));

         // Переопределяем головной объект
         this->header_ptr = new_header;
         new_header->children_ptr.push_back(this);

         return true;
      }

      void cl_base::delete_by_name(string name) {
         for(int i = 0; i < children_ptr.size(); i++) {
            if(children_ptr[i]->name == name){
               children_ptr.erase(children_ptr.begin() + i);
               return;
```

```
        }
    }
}

cl_base* cl_base::find_by_coord(string coord) {
    string s_name;

    // Пустая координата
    if(coord == "")
        return nullptr;

    cl_base* root = this;
    while(root->get_header()){
        root = root->get_header();
    }

    // Только Корень
    if(coord == "/") {
        return root;
    }

    // Поиск по уникальной имени от корневого
    if(coord.substr(0,2)=="//") {
        s_name = coord.substr(2);
        return this->find_on_tree(s_name);
    }

    // Только текущий объект
    if(coord == ".")
        return this;

    // Поиск по уникальной имени от текущего
    if(coord.substr(0,1)==".") {
        s_name = coord.substr(1);
        return this->find_on_branch(s_name);
    }

    // Если в начале стоит / преобразуем абсолютную координату
    // в относительную и ищем относительную от корня
    cl_base* current = this;

    // Абсолютная координата от корневого объекта
    if(coord.substr(0,1) == "/")
    {
        current = root;
        coord = coord.substr(1);
    }


    // Относительная координата от текущего объекта
    while(coord.find("/") != string::npos)  {
        int nsl = coord.find("/", 1);
        current = current->get_child(coord.substr(0, nsl));

        if(!current) {
```

```
            return nullptr;
         }

         coord = coord.substr(nsl + 1);
      }

      return current->get_child(coord);
   }

   // КВ-3


   // КВ-4
   string cl_base::get_path()
   {
      string path = "";
      cl_base* p_obj = this;
      while(p_obj->get_header()){
         path = "/" + p_obj->get_name() + path;
         p_obj = p_obj->get_header();
      }
      if(path == "") path = "/";
      return path;
   }


   void  cl_base  ::  set_connect  (TYPE_SIGNAL  p_signal,  cl_base*  p_object,
   TYPE_HANDLER p_ob_handler)
   {
      o_sh  * p_value;
      //-------------------------------------------------------
      // Цикл для исключения повторного установления связи
      for (unsigned int i = 0; i < connects.size (); i++)
      {
         if ( connects [ i ] -> p_signal  == p_signal     &&
              connects [ i ] -> p_cl_base == p_object     &&
              connects [ i ] -> p_handler == p_ob_handler    )
         {
            return;
         }
      }

      p_value =  new o_sh ( );          // создание объекта структуры для
                                      // хранения информации о новой связи
      p_value->p_signal = p_signal;
      p_value->p_cl_base = p_object;
      p_value->p_handler = p_ob_handler;

      connects.push_back ( p_value );         // добавление новой связи
   }

   void  cl_base::remove_connect  (TYPE_SIGNAL  p_signal,  cl_base*  p_object,
   TYPE_HANDLER p_ob_handler) {
      for (auto i = connects.begin(); i < connects.end(); i++){
         o_sh* c = *i;
```

```
      if (c->p_signal == p_signal &&
         c->p_cl_base == p_object &&
         c->p_handler == p_ob_handler)
      {
         connects.erase(i);
         delete c;
         return;
      }
   }
}


void cl_base :: emit_signal ( TYPE_SIGNAL p_signal, string & s_command )
{
   TYPE_HANDLER   p_handler;
   cl_base      * p_object;
   //-------------------------------------------------------------------------
-
   if(this->state == 0) return;
   (this->*p_signal) (s_command);  // вызов метода сигнала
   for ( unsigned int i = 0; i < connects.size ( ); i ++ ) // цикл по всем
обработчикам
   {
      if ( connects [ i ] -> p_signal == p_signal )         // определение
допустимого обработчика
      {
         p_handler = connects [ i ] -> p_handler;
         p_object  = connects [ i ] -> p_cl_base;
         if(p_object->state != 0)
            (p_object ->* p_handler ) ( s_command );        // вызов метода
обработчика
      }
   }
}

// КВ-4



void cl_base::print() {
   cout << this->name;

   int tab = 0;
   cl_base* par = this;
   while(par->get_header()){
      par = par->get_header();
      tab += 1;
   }


   if(!children_ptr.empty())
   {
      for(cl_base* child : children_ptr) {
         cout << endl;
         for(int i = 0; i <= tab; i++) cout << "    ";
```

```
            child->print();
         }
      }
   }


   void cl_base::print_state() {
      cout << this->name;
      if(this->state == 0) cout << " is not ready";
      else cout << " is ready";

      int tab = 0;
      cl_base* par = this;
      while(par->get_header()){
         par = par->get_header();
         tab += 1;
      }


      if(!children_ptr.empty())
      {
         for(cl_base* child : children_ptr) {
            cout << endl;
            for(int i = 0; i <= tab; i++) cout << "    ";
            child->print_state();
         }
      }
   }

   void cl_base::set_state(int state) {
      if(header_ptr && header_ptr->state == 0) this->state = 0;
      else this->state = state;

      if(state == 0) {
         for(cl_base* child : children_ptr){
            child->set_state(0);
         }
      }
   }
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
   #ifndef __CL_BASE__H
   #define __CL_BASE__H
   #include <string>
   #include <vector>
   #include <iostream>
   #include <algorithm>
```

```
using namespace std;



class cl_base {
protected:
   string name;
   cl_base *header_ptr;
   vector<cl_base*> children_ptr;
   int state;

   #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 &);
   typedef void (cl_base::*TYPE_HANDLER) (string);

   struct o_sh                 // Структура задания одной связи
   {
      TYPE_SIGNAL  p_signal;    // Указатель на метод сигнала
      cl_base    * p_cl_base;   // Указатель на целевой объект
      TYPE_HANDLER p_handler;   // Указатель на метод обработчика
   };

   vector < o_sh * > connects;


public:
   cl_base(cl_base* base, string name = "Object");
   bool edit_name(string new_name);
   string get_name();
   cl_base* get_header();
   cl_base* get_child(string name);

   cl_base* find_on_branch(string name, int* count = nullptr); // Поиск на
ветке
   cl_base* find_on_tree(string name); // Поиск на всем дереве

   // КВ-3
   bool  change_header(cl_base*  new_header);  //  Переопределение  головного
объекта
   void delete_by_name(string name); // Удаление объекта по наименованию
   cl_base* find_by_coord(string coord); // Поиск по координате
   // КВ-3

   // КВ-4
   virtual void signal(string& message) = 0;
   virtual void handler(string message) = 0;

   string get_path();
   void  set_connect  (TYPE_SIGNAL  p_signal,  cl_base*  p_object,  TYPE_HANDLER
p_ob_handler);
   void remove_connect (TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER
p_ob_handler);
```

```
        void emit_signal (TYPE_SIGNAL p_signal, string& s_command);
        // KB-4


        void print();
        void print_state();

        void set_state(int state);

      };




      #endif
```

5.5 Файл cl_obj2.cpp

Листинг 5 – cl_obj2.cpp

```
      #include "cl_obj2.h"

      cl_obj2::cl_obj2(cl_base* header, string name) : cl_base(header, name) {}


      void cl_obj2::signal(string& message) {
        cout << "Signal from " << get_path() << endl;
        message += " (class: 2)";
      };

      void cl_obj2::handler(string message) {
        cout << "Signal to " << get_path() << " Text:  " << message << endl;
      };
```

5.6 Файл cl_obj2.h

Листинг 6 – cl_obj2.h

```
      #ifndef __CL_OBJ2__H
      #define __CL_OBJ2__H
      #include "cl_base.h"

      class cl_obj2 : public cl_base
      {
      public:
```

```
   cl_obj2(cl_base* header, string name);
   void signal(string& message);
   void handler(string message);
};

#endif
```

5.7 Файл cl_obj3.cpp

Листинг 7 – cl_obj3.cpp

```
#include "cl_obj3.h"

cl_obj3::cl_obj3(cl_base* header, string name) : cl_base(header, name) {}


void cl_obj3::signal(string& message) {
   cout << "Signal from " << get_path() << endl;
   message += " (class: 3)";
};

void cl_obj3::handler(string message) {
   cout << "Signal to " << get_path() << " Text:  " << message << endl;
};
```

5.8 Файл cl_obj3.h

Листинг 8 – cl_obj3.h

```
#ifndef __CL_OBJ3__H
#define __CL_OBJ3__H
#include "cl_base.h"

class cl_obj3 : public cl_base
{
public:
   cl_obj3(cl_base* header, string name);
   void signal(string& message);
   void handler(string message);
};

#endif
```

5.9 Файл cl_obj4.cpp

Листинг 9 – cl_obj4.cpp

```
#include "cl_obj4.h"

cl_obj4::cl_obj4(cl_base* header, string name) : cl_base(header, name) {}


void cl_obj4::signal(string& message) {
   cout << "Signal from " << get_path() << endl;
   message += " (class: 4)";
};

void cl_obj4::handler(string message) {
   cout << "Signal to " << get_path() << " Text:  " << message << endl;
};
```

5.10 Файл cl_obj4.h

Листинг 10 – cl_obj4.h

```
#ifndef __CL_OBJ4__H
#define __CL_OBJ4__H
#include "cl_base.h"

class cl_obj4 : public cl_base
{
public:
   cl_obj4(cl_base* header, string name);
   void signal(string& message);
   void handler(string message);
};

#endif
```

5.11 Файл cl_obj5.cpp

Листинг 11 – cl_obj5.cpp

```
#include "cl_obj5.h"

cl_obj5::cl_obj5(cl_base* header, string name) : cl_base(header, name) {}
```

```
void cl_obj5::signal(string& message) {
   cout << "Signal from " << get_path() << endl;
   message += " (class: 5)";
};

void cl_obj5::handler(string message) {
   cout << "Signal to " << get_path() << " Text:  " << message << endl;
};
```

5.12 Файл cl_obj5.h

Листинг 12 – cl_obj5.h

```
#ifndef __CL_OBJ5__H
#define __CL_OBJ5__H
#include "cl_base.h"

class cl_obj5 : public cl_base
{
public:
   cl_obj5(cl_base* header, string name);
   void signal(string& message);
   void handler(string message);
};


#endif
```

5.13 Файл cl_obj6.cpp

Листинг 13 – cl_obj6.cpp

```
#include "cl_obj6.h"

cl_obj6::cl_obj6(cl_base* header, string name) : cl_base(header, name) {}


void cl_obj6::signal(string& message) {
   cout << "Signal from " << get_path() << endl;
   message += " (class: 6)";
};

void cl_obj6::handler(string message) {
```

```
   cout << "Signal to " << get_path() << " Text:  " << message << endl;
};
```

5.14 Файл cl_obj6.h

Листинг 14 – cl_obj6.h

```
#ifndef __CL_OBJ6__H
#define __CL_OBJ6__H
#include "cl_base.h"

class cl_obj6 : public cl_base
{
public:
   cl_obj6(cl_base* header, string name);
   void signal(string& message);
   void handler(string message);
};



#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.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 23.

Таблица 23 – 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 end_of_connections EMIT /object_s2/object_s4 Send message 1 EMIT /a Message is ded SET_CONNECT / /object_s13 SET_CONNECT /c /object_s13 SET_CONNECT / /c EMIT / Hello! DELETE_CONNECT / /object_s13 DELETE_CONNECT /c /object_s13 DELETE_CONNECT / /c EMIT / Hello! SET_CONDITION / 0 SET_CONDITION /a 45 EMIT /object_s2/object_s4 Non-active message END	object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Object /a not found Object /c not found Handler object /c not found Signal from / Signal to /object_s13 Text: Hello! (class: 1) Object /c not found Handler object /c not found Signal from / Object /a not found	object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Object /a not found Object /c not found Handler object /c not found Signal from / Signal to /object_s13 Text: Hello! (class: 1) Object /c not found Handler object /c not found Signal from / Object /a 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).