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:

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

- 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 синтаксический 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

Используем параметризированное макроопределение препроцессора для получения указателей на методы сигнала и обработчика объектов.

Структура connect:

- Свойства/поля:
  - Указатель на метод сигнала signal_ptr;
  - Указатель на целевой объект target_ptr;
  - Указатель на метод обработчика handler_ptr;

Класс cl_base:

- Свойства / поля:
  - Вектор для хранения установленных связей connects;
- Функционал:
  - Метод set_connect - метод установки связи между сигналом текущего объекта и обработчиком целевого объекта;
  - Метод remove_connect - метод удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
  - Метод emit_signal - метод выдачи сигнала от текущего объекта с передачей строковой переменной;
  - Метод get_absolute_path - метод получения абсолютного пути объекта;
  - Метод set_state_branch - Метод устанавки объекту и всем его потомкам значение готовности;

Класс application:

- Функционал:
  - Метод signal - метод сигнала;
  - Метод handler - метод обработчика;
  - Метод get_class_number - метод возврата номера класса;

Класс cl_2:

- Функционал:
  - Метод signal - метод сигнала;
  - Метод handler - метод обработчика;
  - Метод get_class_number - метод возврата номера класса;

Класс cl_3:

- Функционал:
  - Метод signal - метод сигнала;
  - Метод handler - метод обработчика;
  - Метод get_class_number - метод возврата номера класса;

Класс cl_4:

- Функционал:
  - Метод signal - метод сигнала;
  - Метод handler - метод обработчика;
  - Метод get_class_number - метод возврата номера класса;

Класс cl_5:

- Функционал:
  - Метод signal - метод сигнала;
  - Метод handler - метод обработчика;
  - Метод get_class_number - метод возврата номера класса;

Класс cl_6:

- Функционал:
  - Метод signal - метод сигнала;
  - Метод handler - метод обработчика;
  - Метод get_class_number - метод возврата номера класса.

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: Основная программа.

Параметры: .

Возвращаемое значение: int - код возврата.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

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

3.2 Алгоритм метода set_connect класса cl_base

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

Параметры: TYPE_SIGNAL signal_ptr, cl_base* target_ptr, TYPE_HANDLER handler_ptr.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_connect класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i значением 0	2
2	i меньше размера вектора connects		3
			4
3	Переданные параметры совпадают с полями i-ой связи вектора connects		∅
		i += 1	2
4		Создание объекта структуры connect с помощью оператора new и присваивание указателю new_connect адрес этого объекта	5
5		Присваиваивание полю signal_ptr объекта по указателю new_connect значение параметра signal_ptr	6
6		Присваивание полю target_ptr объекта по указателю new_connect значение параметра target_ptr	7
7		Присваиваивание полю handler_ptr объекта по указателю new_connect значение параметра handler_ptr	8
8		Добавление указателя new_connect в вектор connects	∅

3.3 Алгоритм метода remove_connect класса cl_base

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

Параметры: TYPE_SIGNAL signal_ptr, cl_base* target_ptr, TYPE_HANDLER handler_ptr.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода remove_connect класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i значением 0	2
2	i меньше размера вектора connects		3
			∅
3	Переданные параметры совпадают с полями i-ой связи вектора connects	Вызов деструктора для объекта по i-му указателю вектора connects	4
		i += 1	2
4		Удаление i-го элемента веткора connects	∅

3.4 Алгоритм метода emit_signal класса cl_base

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

Параметры: TYPE_SIGNAL signal_ptr, string & command.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода emit_signal класса cl_base

№	Предикат	Действия	№ перехода
1	свойство state равно 0		∅
			2
2		Вызов метода сигнала по указателю signal_ptr с параметром command	3
3		Инициализация целочисленной переменной i значением 0	4
4	i меньше размера свойства connects		5
			∅
5	свойство signal_ptr i-й связи вектора connects равно signal_ptr		7
			6
6		i += 1	4
7		Инициализация указателя на метод обработчика handler_ptr значением свойства handler_ptr i-го объекта вектора connects	8
8		Инициализация указателя target_ptr на объект класса cl_base значением свойства target_ptr i-го объекта вектора connects	9
9	свойство state объекта по указателю target_ptr не равно 0	Вызов метода обработчика по указателю handler_ptr объекта по указателю target_ptr с параметром command	6
			6

3.5 Алгоритм метода get_absolute_path класса cl_base

Функционал: Возвращает абсолютный путь к объекту.

Параметры: .

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_absolute_path класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление строки result	2
2		Объявление стека строк st	3
3		Инициализация указателя root_ptr на объект класса cl_base адресом текущего объекта	4
4	Результат вызова метода get_parent объекта по указателю root_ptr не равен нулевому указателю	Добавить в стек st результат вызова метода get_name объекта по указателю root_ptr	5
			6
5		Присваивание root_ptr результат вызова метода get_parent объекта по указателю root_ptr	4
6	Стек st непустой	Добавить в строку result символ '/' и строку на вершине стека st	7
			8
7		Удалить элемент на вершине стека st	6
8	Строка result пустая	Вернуть "/"	∅
		Вернуть result	∅

3.6 Алгоритм метода set_state_branch класса cl_base

Функционал: Устанавливает объекту и всем его потомкам значение готовности.

Параметры: int new_state.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода set_state_branch класса cl_base

№	Предикат	Действия	№ перехода
1	У объекта есть родитель и его свойство state равно 0		∅
			2
2		Вызов метода setState с параметром new_state	3
3		Инициализация целочисленной переменной i значением 0	4
4	i меньше размера вектора children		5
			∅
5		Вызов метода set_state_branch объекта по i-му указателю вектора children с параметром new_state	6
6		i += 1	4

3.7 Алгоритм метода get_class_number класса cl_base

Функционал: Возвращает номер класса.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_class_number класса cl_base

№	Предикат	Действия	№ перехода
1		Вернуть 0	∅

3.8 Алгоритм метода signal класса application

Функционал: Метод сигнала.

Параметры: Ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода signal класса application

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal from " и результат вызова метода get_absolute_path()	2
2		Добавление в конец строки message " (class: {результат вызова метода get_class_number приведённый к строке})"	∅

3.9 Алгоритм метода handler класса application

Функционал: Метод обработчика.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода handler класса application

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal to ", результат вызова метода get_absolute_path, " Text: " и строку message	∅

3.10 Алгоритм метода get_class_number класса application

Функционал: Возвращает номер класса.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода get_class_number класса application

№	Предикат	Действия	№ перехода
1		Вернуть 1	∅

3.11 Алгоритм метода get_class_number класса cl_2

Функционал: Возвращает номер класса.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода get_class_number класса cl_2

№	Предикат	Действия	№ перехода
1		Вернуть 2	∅

3.12 Алгоритм метода get_class_number класса cl_3

Функционал: Возвращает номер класса.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода get_class_number класса cl_3

№	Предикат	Действия	№ перехода
1		Вернуть 3	∅

3.13 Алгоритм метода get_class_number класса cl_4

Функционал: Возвращает номер класса.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода get_class_number класса cl_4

№	Предикат	Действия	№ перехода
1		Вернуть 4	∅

3.14 Алгоритм метода get_class_number класса cl_5

Функционал: Возвращает номер класса.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода get_class_number класса cl_5

№	Предикат	Действия	№ перехода
1		Вернуть 5	∅

3.15 Алгоритм метода get_class_number класса cl_6

Функционал: Возвращает номер класса.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода get_class_number класса cl_6

№	Предикат	Действия	№ перехода
1		Вернуть 6	∅

3.16 Алгоритм функции class_number_to_signal

Функционал: Возвращает указатель на метод сигнала, в зависимости от номера класса.

Параметры: int class_number.

Возвращаемое значение: Указатель на метод сигнала.

Алгоритм функции представлен в таблице 16.

Таблица 16 – Алгоритм функции class_number_to_signal

№	Предикат	Действия	№ перехода
1	class_number равно 1	Вернуть указатель на метод сигнала класса application	∅
	class_number равно 2	Вернуть указатель на метод сигнала класса cl_2	∅
	class_number равно 3	Вернуть указатель на метод сигнала класса cl_3	∅
	class_number равно 4	Вернуть указатель на метод сигнала класса cl_4	∅
	class_number равно 5	Вернуть указатель на метод сигнала класса cl_5	∅
	class_number равно 6	Вернуть указатель на метод сигнала класса cl_6	∅
		Вернуть нулевой указатель	∅

3.17 Алгоритм метода build_tree_objects класса application

Функционал: Строит дерево и устанавливает связи между объектами.

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода build_tree_objects класса application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree"	2
2		Объявление строк path, child_name	3
3		Объявление целочисленной переменной tmp	4
4		Ввод chlid_name	5
5		Вызов метода setName с параметром child_name	6
6		Объявление указателя parent_node_ptr на объект класса cl_base	7
7		Инициализация указателя last_created_node_ptr адресом текущего объекта	8
8		Ввод path	9
9	path не равно "endtree"	ввод child_name и tmp	10
			16
10		Присваивание parent_node_ptr результат вызова метода get_object_by_path с параметром path объекта по указателю last_created_node_ptr	11
11	parent_node_ptr нулевой указатель	вызов метода printBranch	12
			14
12		Вывод с новой строки "The head object ", path, " is not found"	13
13		Выход с кодом 1	∅
14	У объекта по указателю parent_node_ptr нет подчинённого с именем	Вывод с новой строки path, " Dubbing the names of subordinate objects"	15

№	Предикат	Действия	№ перехода
	child_name		
	tmp принимает значение от 1 до 6	Создание объекта класса в соответствии со значением переменной tmp с помощью оператора new, конструктора и параметров parent_node_ptr, child_name и присваивание last_created_node_ptr адрес этого объекта	15
			15
15		Ввод path	9
16		Объявление указателя target_ptr на объект класса cl_base	17
17		Объявление строки target_path	18
18		Ввод path	19
19	path не равно "end_of_connections"	Ввод target_path	20
			∅
20		Присваивание parent_node_ptr результат вызова метода get_object_by_path с параметром path	21
21		Присваивание target_ptr результат вызова метода get_object_by_path с параметром target_ptr	22
22		Инициализация указателя signal_f на метод сигнала результатом вызова функции class_number_to_signal с параметром, являющимся номером класса объекта по указателю parent_node_ptr	23

№	Предикат	Действия	№ перехода
23		Инициализация указателя handler_f на метод обработчика результатом вызова функции class_number_to_signal с параметром, являющимся номером класса объекта по указателю target_ptr	24
24		Вызов метода set_connect объекта по указателю parent_node_ptr с параметрами signal_f, target_ptr, handler_f	25
25		Ввод path	19

3.18 Алгоритм метода exec_app класса application

Функционал: Обрабатывает комманды пользователя.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		Объявление указателя signal_f на метод сигнала	2
2		Объявление указателя handler_f на метод обработчика	3
3		Вызов метода set_state_branch с параметром 1	4
4		Инициализация пустых строк command, input и message	5
5		Объявление целочисленной переменной new_state	6
6		Объявление указателей extra_object_ptr и	7

№	Предикат	Действия	№ перехода
		target_object_ptr на объекты класса cl_base	
7		Вызов метода printBranch	8
8		Ввод command	9
9	command не равно "END"	Ввод input	10
		Вернуть 0	∅
10		Присваивание extra_object_ptr результат вызова метода get_object_by_path с параметром input	11
11	extra_object_ptr нулевой указатель		13
	command равно "EMIT"		15
	command равно "SET_CONNECT"		18
	command равно "DELETE_CONNECT"		23
	command равно "SET_CONDITION"		28
			12
12		Ввод command	9
13		Вывод с новой строки "Object ", input, " not found"	14
14		Ввод input	9
15		Ввод message	16
16		Инициализация целочисленной переменной n результатом вызова метода get_class_number объекта по указателю extra_object_ptr	17

№	Предикат	Действия	№ перехода
17		Вызов метода emit_signal объекта по указателю extra_object_ptr с параметрами class_number_to_signal(n) и message	9
18		Ввод input	19
19		Присваивание target_object_ptr результат вызова метода get_object_by_path с параметром input	20
20	target_object_ptr нулевой указатель	Вывод "Handler object ", input, " not found"	9
		Присваивание signal_f результат вызова функции class_number_to_signal с параметром, являющимся номером класса объекта по указателю extra_object_ptr	21
21		Присваивание handler_f результат вызова функции class_number_to_handler с параметром, являющимся номером класса объекта по указателю target_object_ptr	22
22		Вызов метода set_connect объекта по указателю extra_object_ptr с параметрами signal_f, target_object_ptr, handler_f	9
23		Ввод input	24
24		Присваивание target_object_ptr результат вызова метода get_object_by_path с параметром input	25
25	target_object_ptr нулевой указатель	Вывод "Handler object ", input, " not found"	9
		Присваивание signal_f результат вызова функции class_number_to_signal с параметром, являющимся	26

№	Предикат	Действия	№ перехода
		номером класса объекта по указателю extra_object_ptr	
26		Присваивание handler_f результат вызова функции class_number_to_handler с параметром, являющимся номером класса объекта по указателю target_object_ptr	27
27		Вызов метода remove_connect объекта по указателю extra_object_ptr с параметрами signal_f, target_object_ptr, handler_f	9
28		Ввод new_state	29
29		Вызов метода setState объекта по указателю extra_object_ptr с параметром new_state	9

3.19 Алгоритм метода signal класса cl_2

Функционал: Метод сигнала.

Параметры: Ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода signal класса cl_2

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal from " и результат вызова метода get_absolute_path()	2
2		Добавление в конец строки message " (class: {результат вызова метода get_class_number приведённый к строке})"	∅

3.20 Алгоритм метода signal класса cl_3

Функционал: Метод сигнала.

Параметры: Ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода signal класса cl_3

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal from " и результат вызова метода get_absolute_path()	2
2		Добавление в конец строки message " (class: {результат вызова метода get_class_number приведённый к строке})"	∅

3.21 Алгоритм метода signal класса cl_4

Функционал: Метод сигнала.

Параметры: Ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода signal класса cl_4

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal from " и результат вызова метода get_absolute_path()	2
2		Добавление в конец строки message " (class: {результат вызова метода get_class_number приведённый к строке})"	∅

3.22 Алгоритм метода signal класса cl_5

Функционал: Метод сигнала.

Параметры: Ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода signal класса cl_5

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal from " и результат вызова метода get_absolute_path()	2
2		Добавление в конец строки message " (class: {результат вызова метода get_class_number приведённый к строке})"	∅

3.23 Алгоритм метода signal класса cl_6

Функционал: Метод сигнала.

Параметры: Ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода signal класса cl_6

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal from " и результат вызова метода get_absolute_path()	2
2		Добавление в конец строки message " (class: {результат вызова метода get_class_number приведённый к строке})"	∅

3.24 Алгоритм метода handler класса cl_2

Функционал: Метод обработчика.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода handler класса cl_2

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal to ", результат вызова метода get_absolute_path, " Text: " и строку message	∅

3.25 Алгоритм метода handler класса cl_3

Функционал: Метод обработчика.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода handler класса cl_3

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal to ", результат вызова метода get_absolute_path, " Text: " и строку message	∅

3.26 Алгоритм метода handler класса cl_4

Функционал: Метод обработчика.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода handler класса cl_4

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal to ", результат вызова метода get_absolute_path, " Text: " и строку message	∅

3.27 Алгоритм метода handler класса cl_5

Функционал: Метод обработчика.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода handler класса cl_5

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal to ", результат вызова метода get_absolute_path, " Text: " и строку message	∅

3.28 Алгоритм метода handler класса cl_6

Функционал: Метод обработчика.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода handler класса cl_6

№	Предикат	Действия	№ перехода
1		Вывод с новой строки "Signal to ", результат вызова метода get_absolute_path, " Text: " и строку message	∅

3.29 Алгоритм функции class_number_to_handler

Функционал: Возвращает указатель на метод обработчика, в зависимости от номера класса.

Параметры: int class_number.

Возвращаемое значение: Указатель на метод обработчика.

Алгоритм функции представлен в таблице 29.

Таблица 29 – Алгоритм функции class_number_to_handler

№	Предикат	Действия	№ перехода
1	class_number равно 1	Вернуть указатель на метод обработчика класса application	∅
	class_number равно 2	Вернуть указатель на метод обработчика класса cl_2	∅
	class_number равно 3	Вернуть указатель на метод обработчика класса cl_3	∅
	class_number равно 4	Вернуть указатель на метод обработчика класса cl_4	∅
	class_number равно 5	Вернуть указатель на метод обработчика класса cl_5	∅
	class_number равно 6	Вернуть указатель на метод обработчика класса cl_6	∅
		Вернуть нулевой указатель	∅

3.30 Алгоритм деструктора класса cl_base

Функционал: Уничтожает объект и его потомков, удаляет всех вхождения объекта и его потомков в связях объектов дерева.

Параметры: .

Алгоритм деструктора представлен в таблице 30.

Таблица 30 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя root_ptr на объект класса cl_base адресом текущего объекта	2
2	У объекта по указателю root_ptr есть родитель	Присваивание root_ptr результат вызова метода get_parent объекта по указателю root_ptr	2
			3
3		Объявление стека st, содержащего указатели на объекты класса cl_base	4
4		Добавить на вершину стека указатель root_ptr	5
5	Стек st содержит элементы	Инициализация указателя ptr на класс cl_base значением указателя на вершине стека	6
			14
6		Удаление вершины стека	7
7		Инициализация целочисленной переменной i значением 0	8
8	i меньше размера вектора connects объекта по указателю ptr		12
			9
9		i = 0	10
10	i меньше размера вектора children объекта по указателю ptr	Добавление в стек st i-го элемента вектора children объекта по указателю ptr	11
			5
11		i += 1	10
1	свойство target_ptr i-го	Удаление i-го элемента из массива connects объекта	13

№	Предикат	Действия	№ перехода
2	объекта вектора connects объекта по указателю ptr совпадает с текущим объектом	по указателю ptr	
		i += 1	8
13		Вызов оператора delete для i-го элемента массива ptr объекта по указателю ptr	8
14	Вектор children содержит элементы		15
			∅
15		Удаление нулевого элемента из вектора children	16
16		Вызов оператора delete для указателя tmp_ptr	14

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-17.

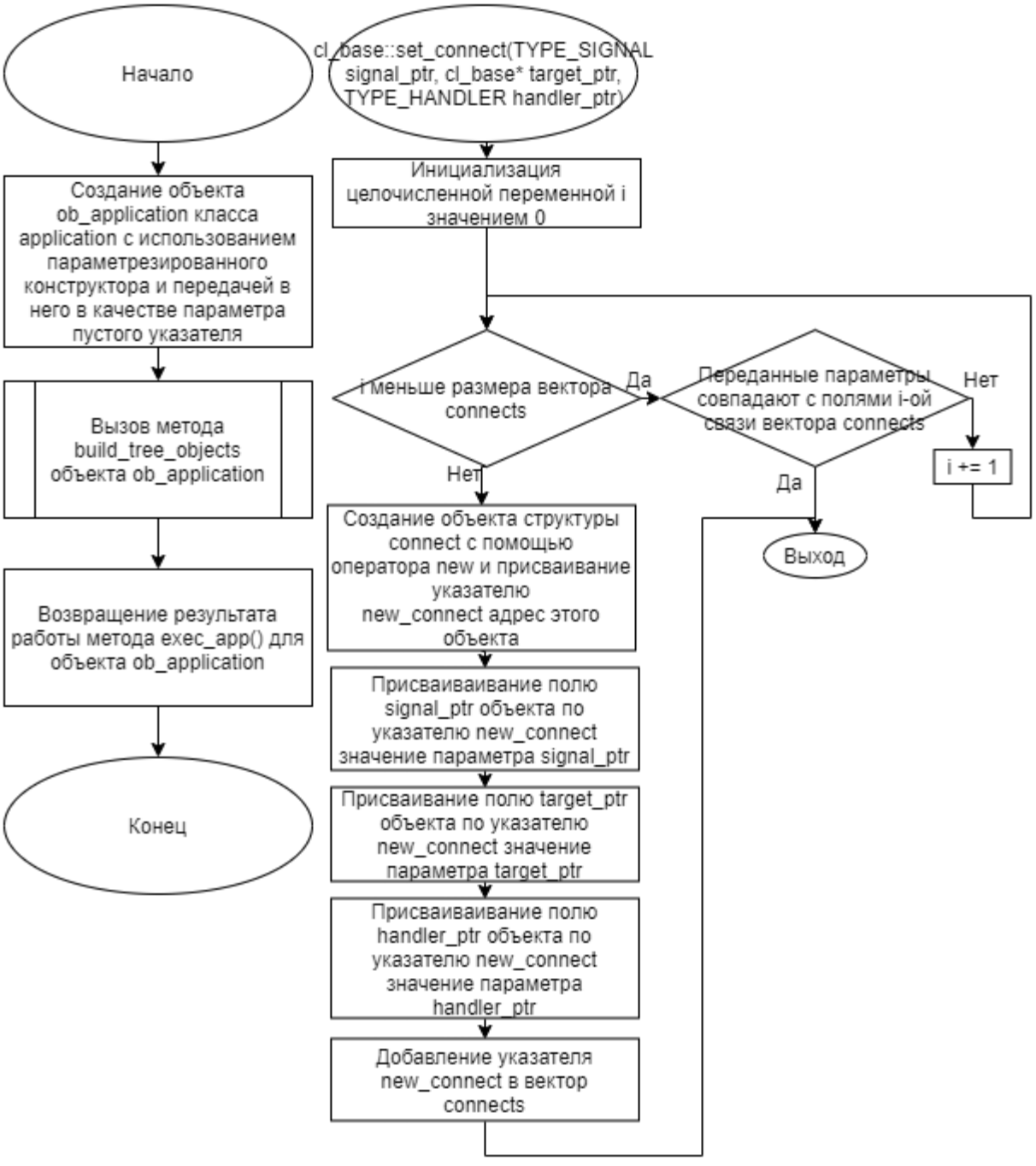

Рисунок 1 – Блок-схема алгоритма

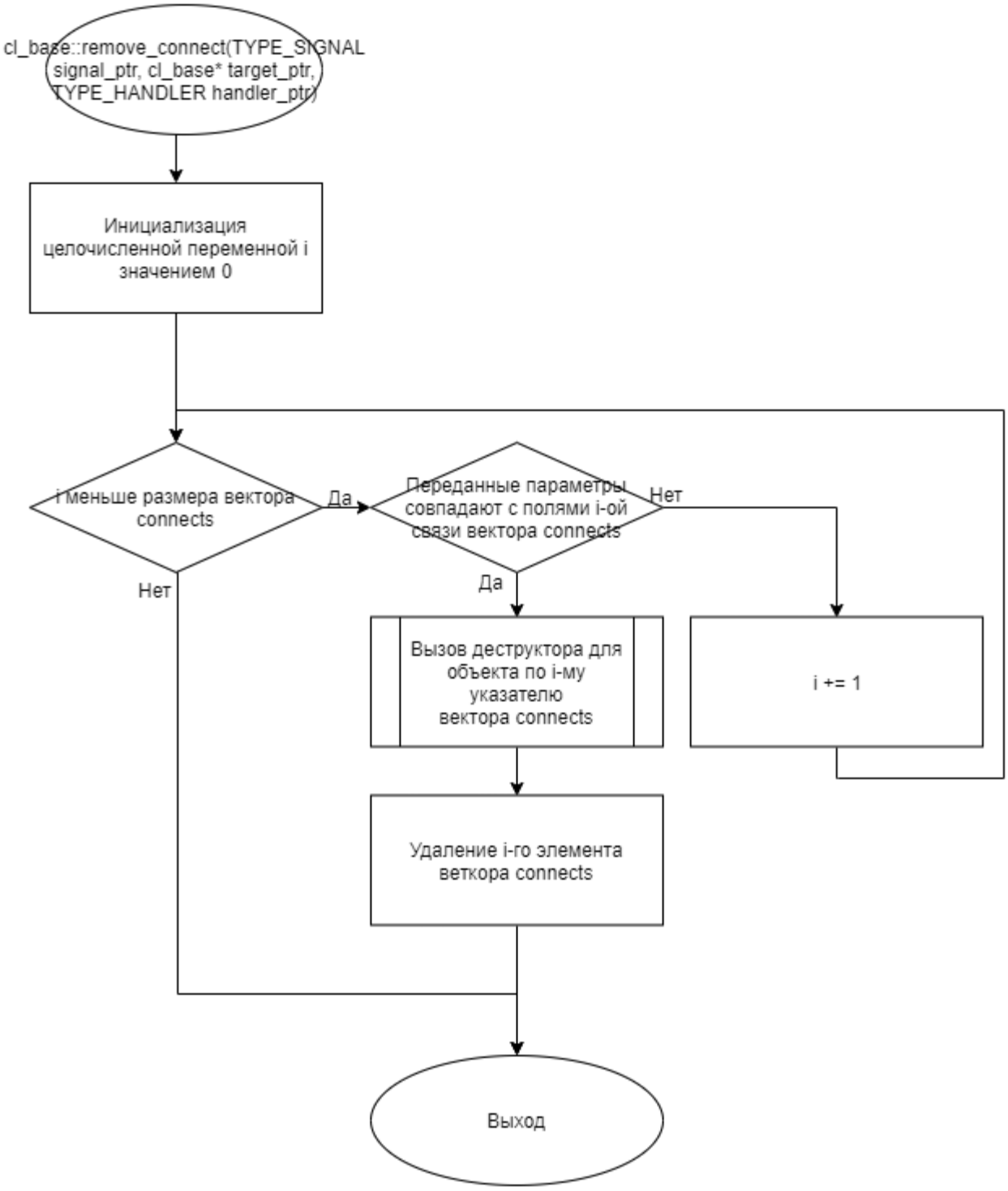

Рисунок 2 – Блок-схема алгоритма

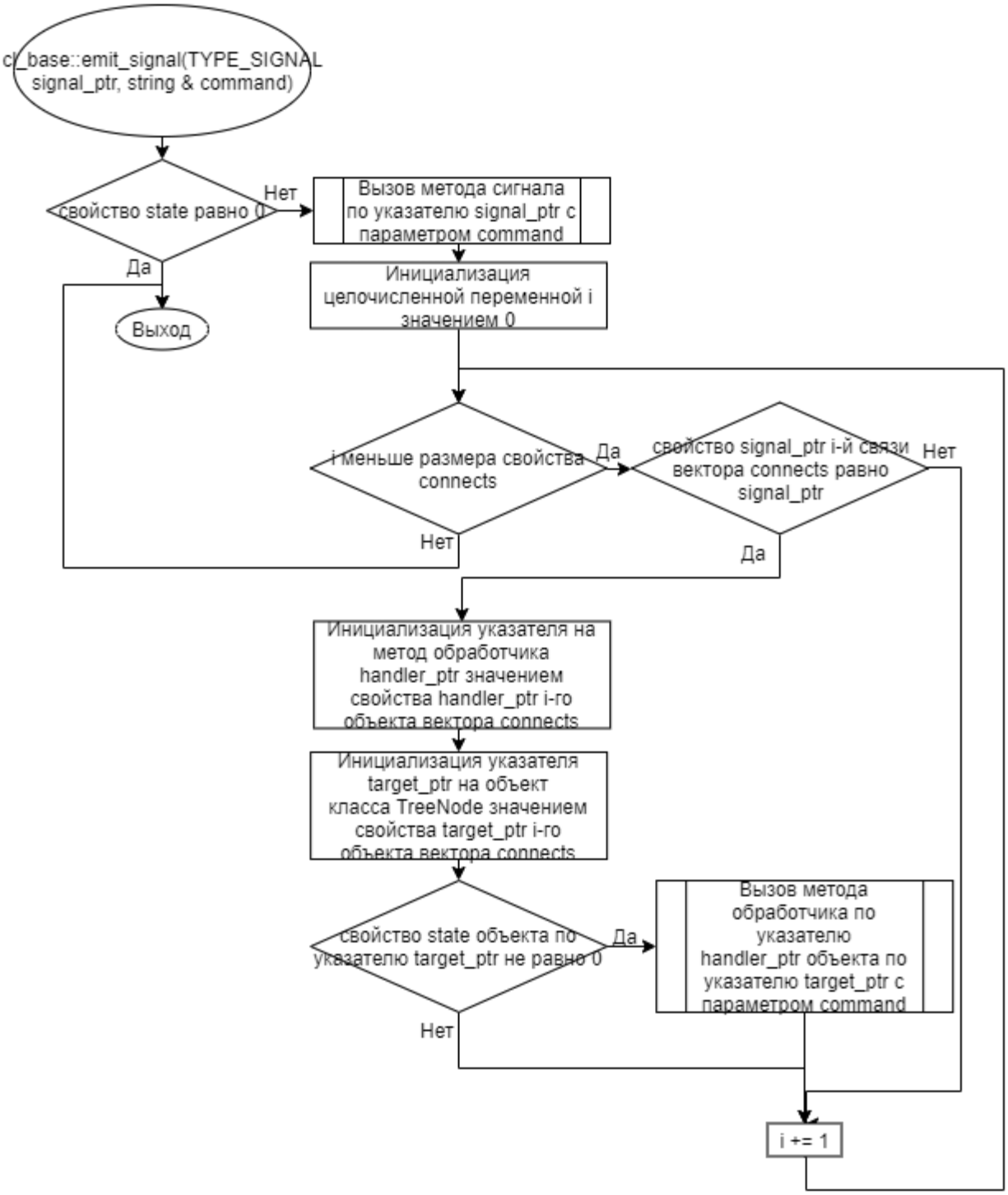

Рисунок 3 – Блок-схема алгоритма

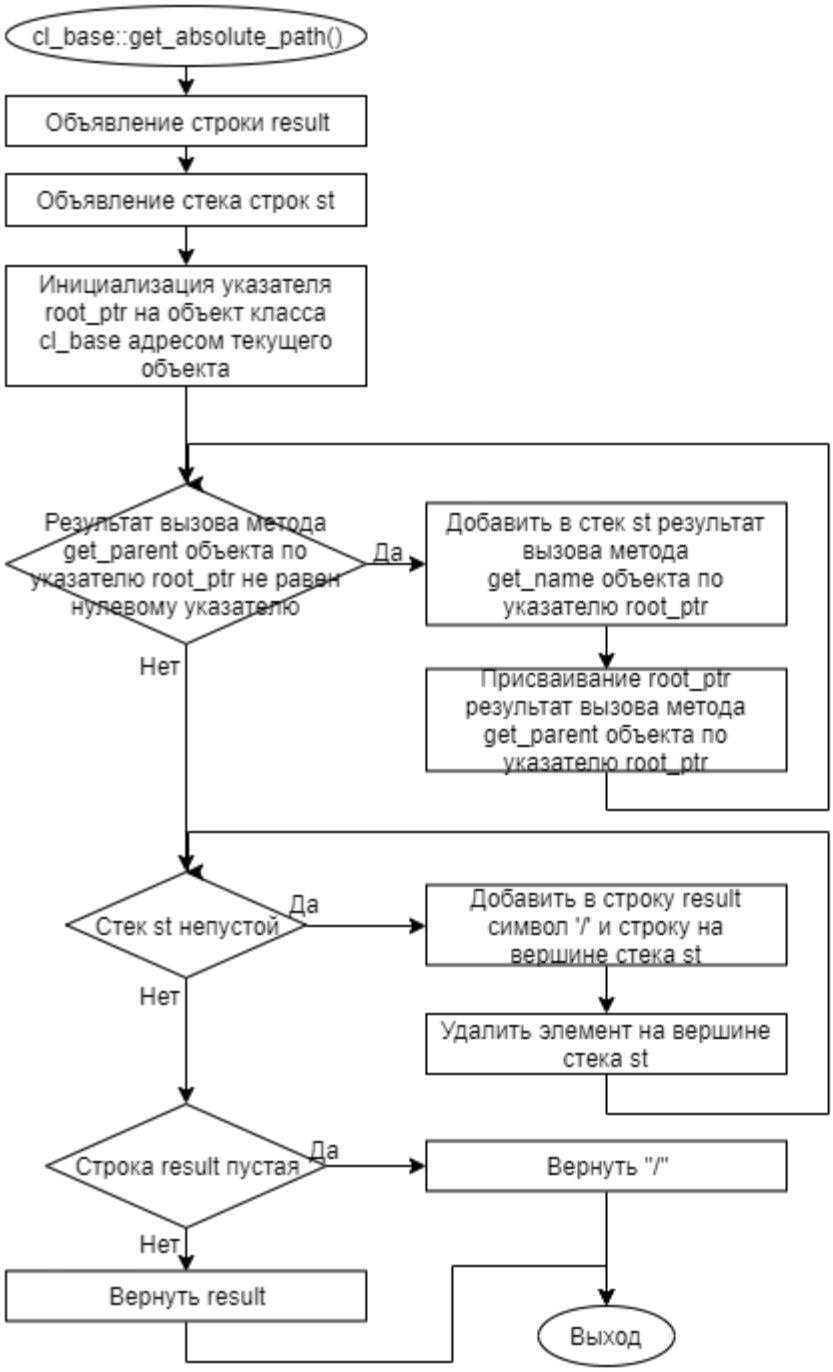

Рисунок 4 – Блок-схема алгоритма

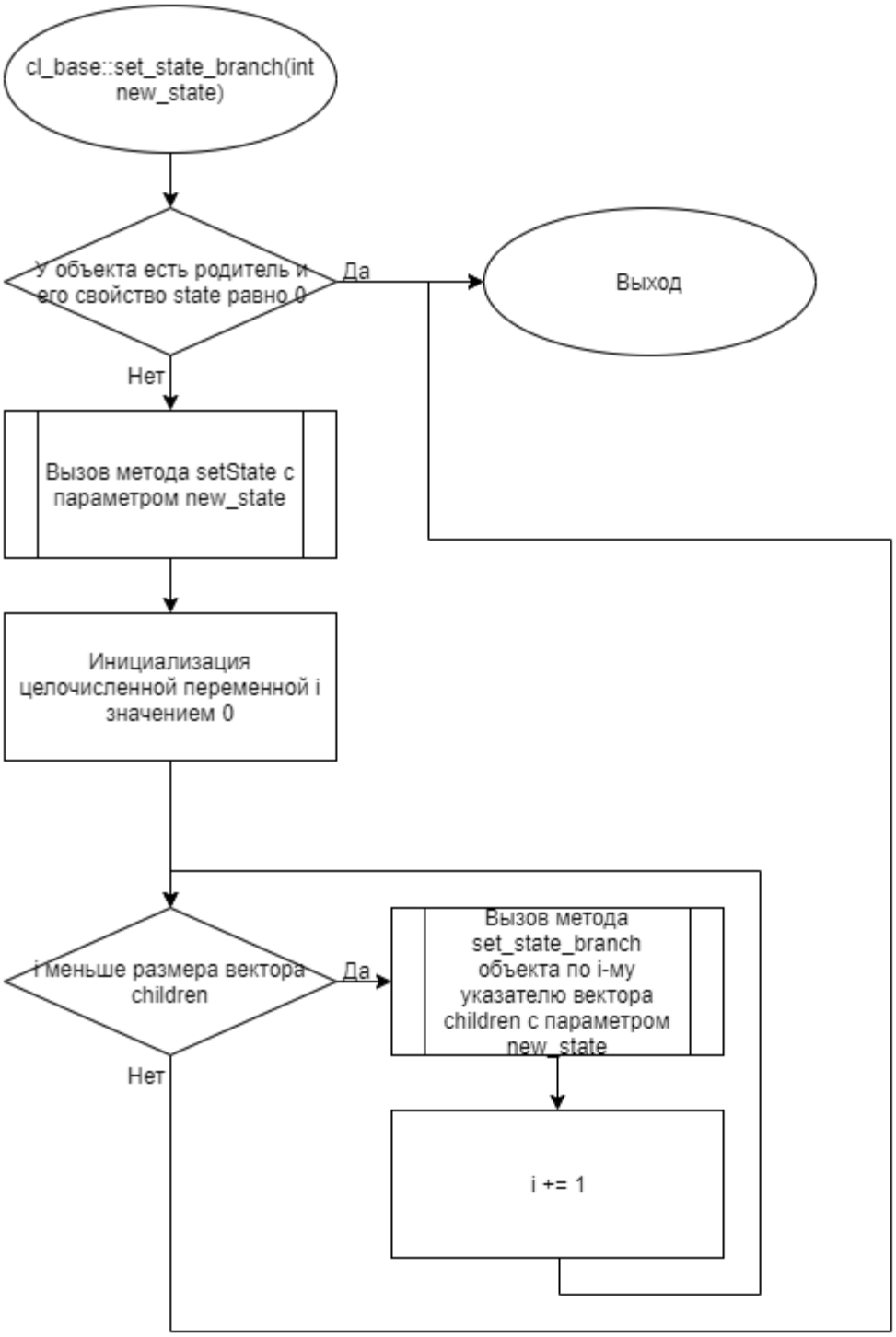

Рисунок 5 – Блок-схема алгоритма

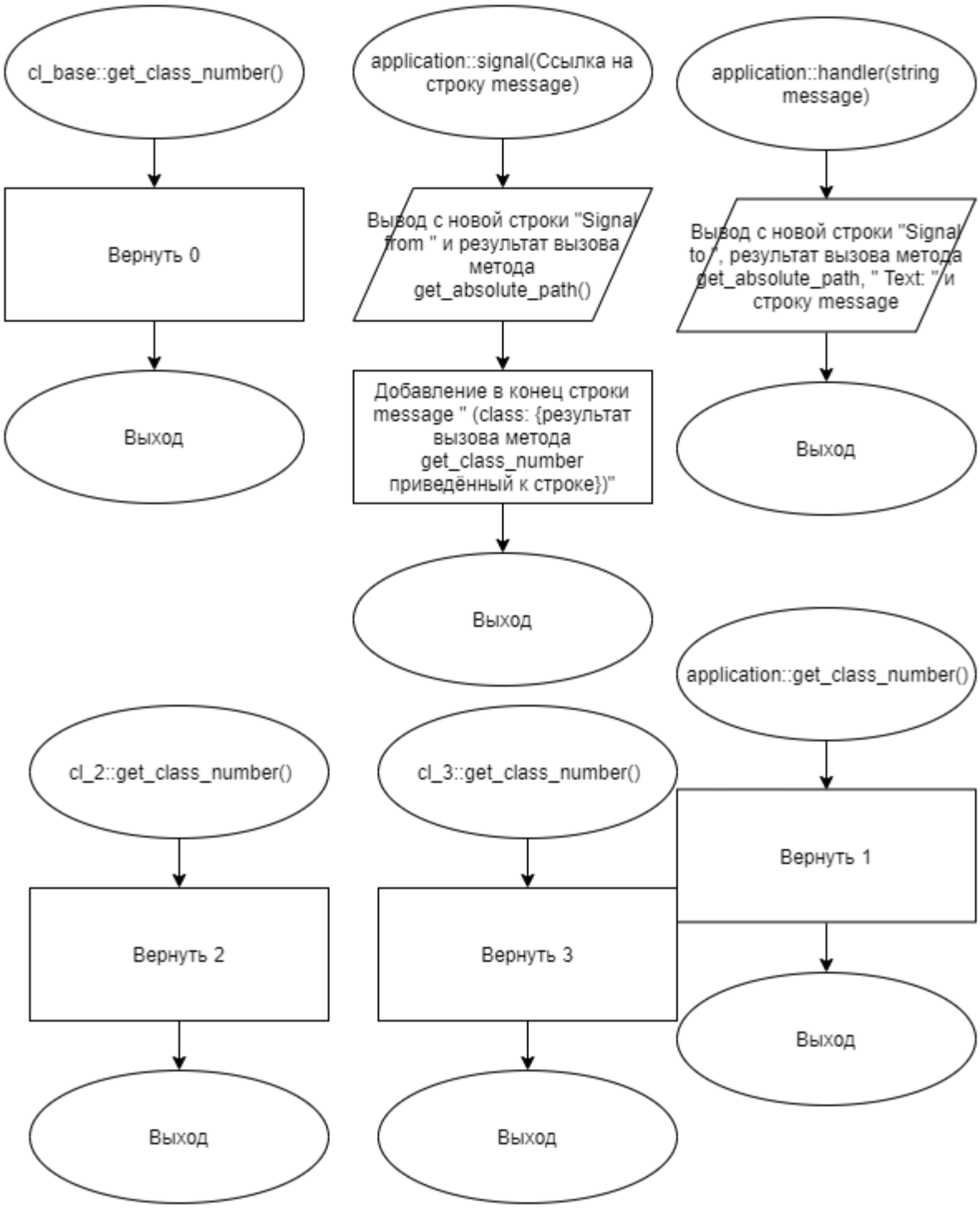

Рисунок 6 – Блок-схема алгоритма

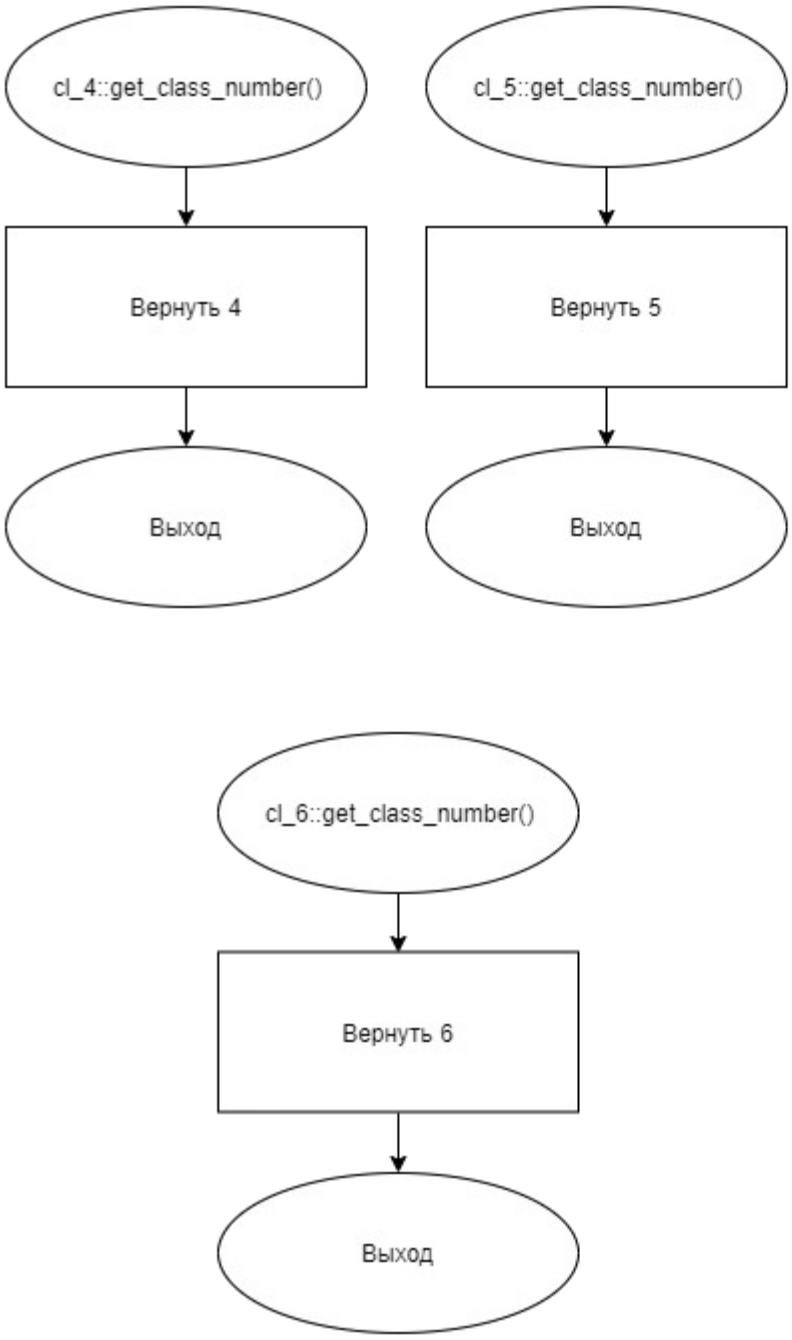

Рисунок 7 – Блок-схема алгоритма

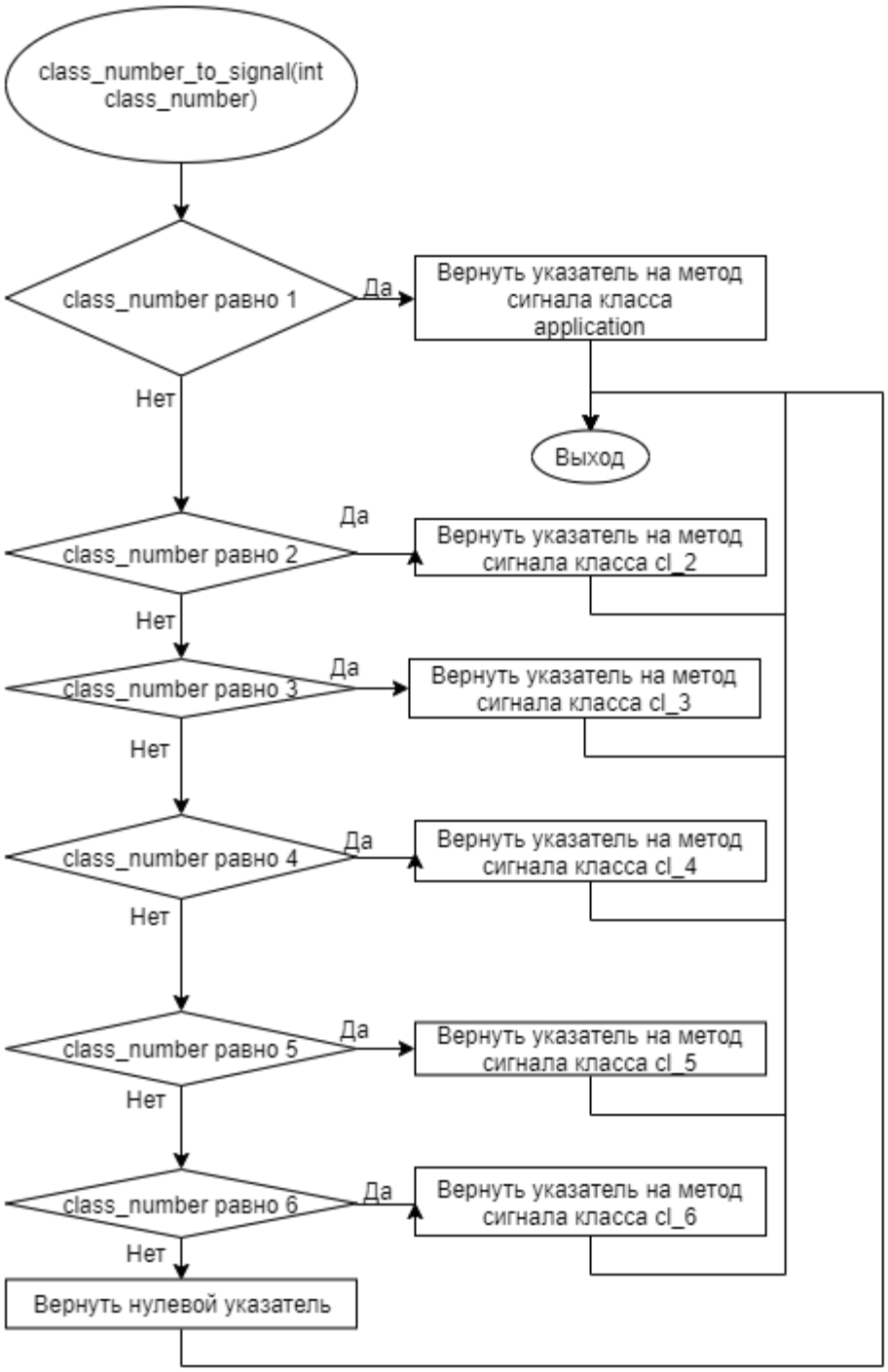

Рисунок 8 – Блок-схема алгоритма

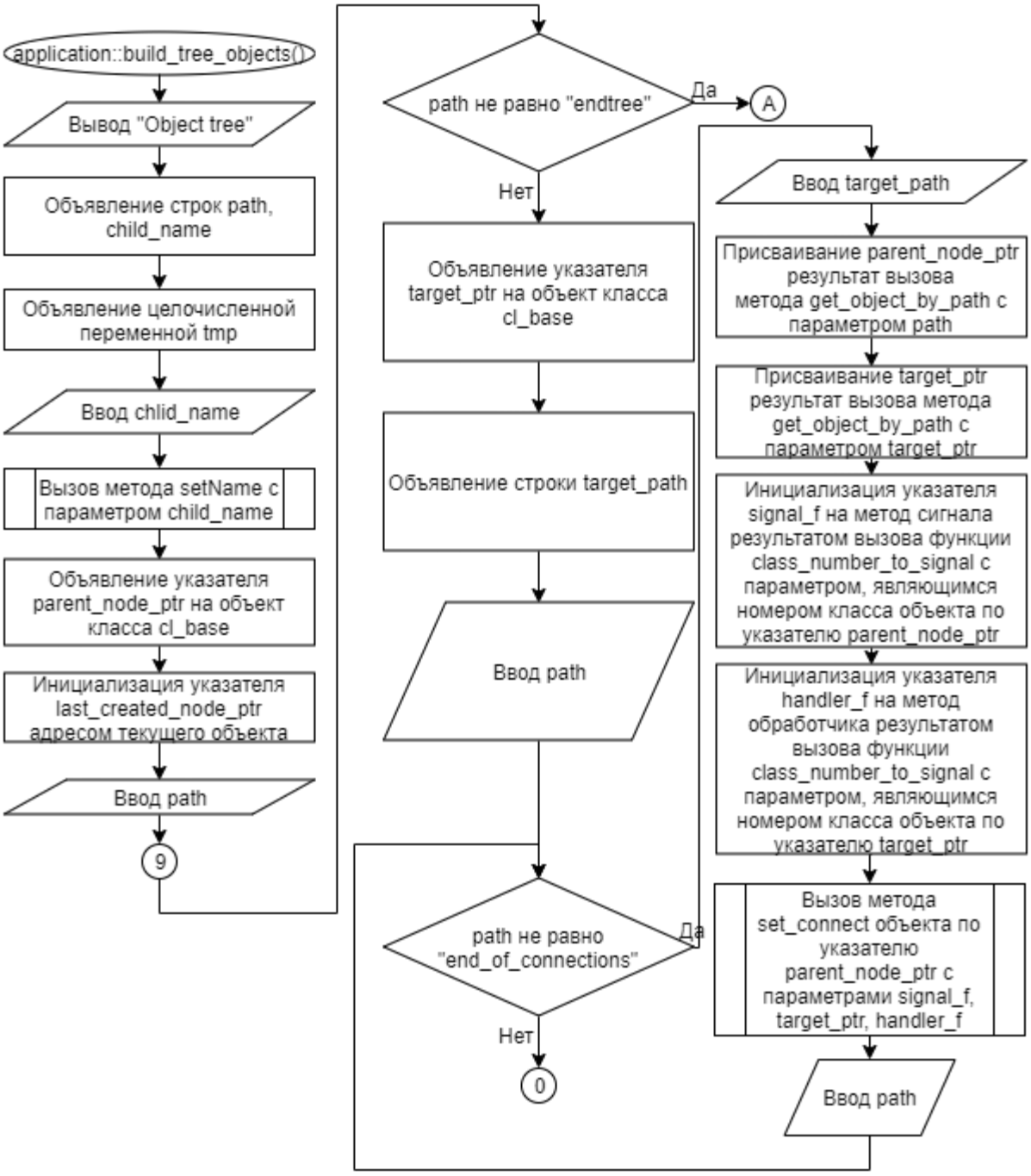

Рисунок 9 – Блок-схема алгоритма

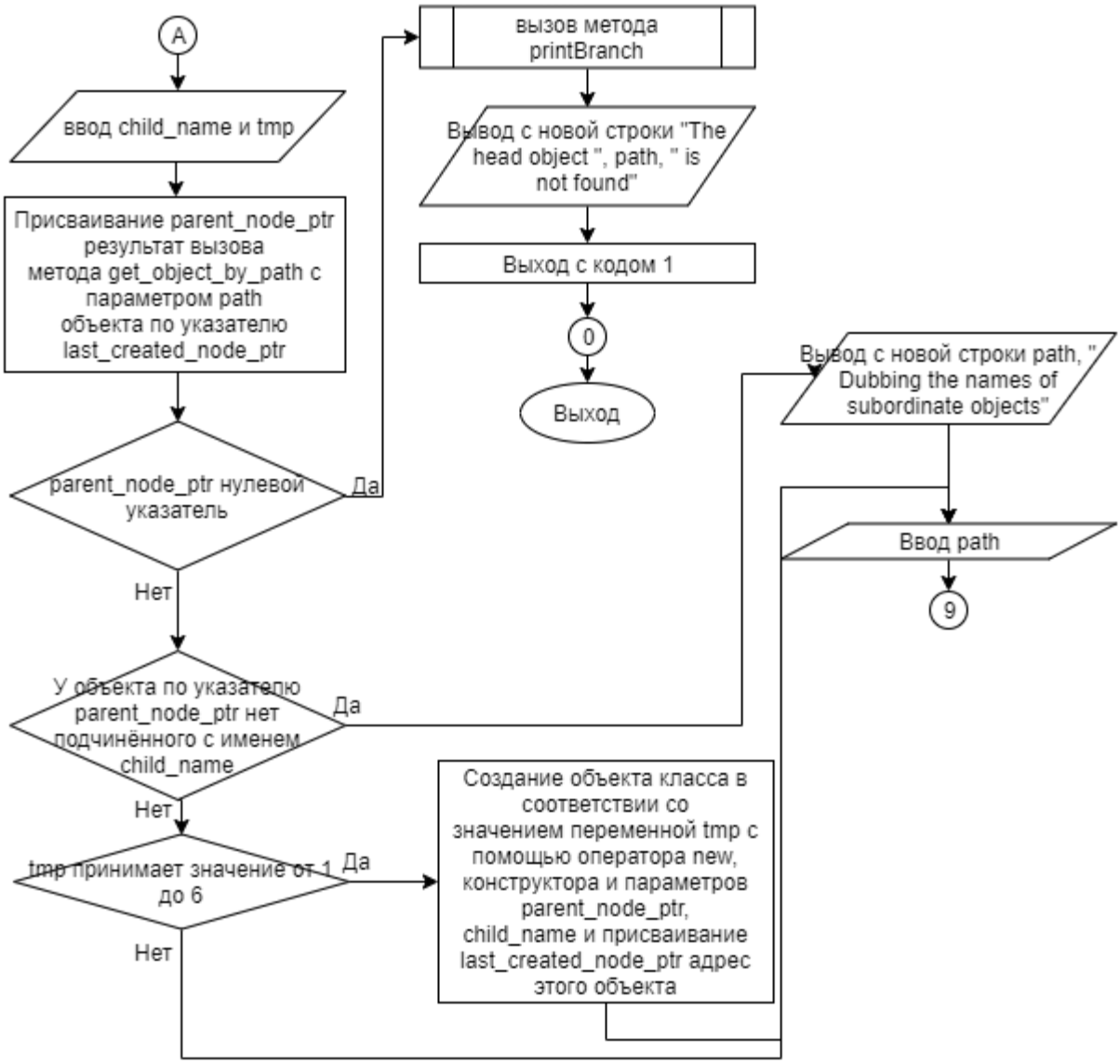

Рисунок 10 – Блок-схема алгоритма

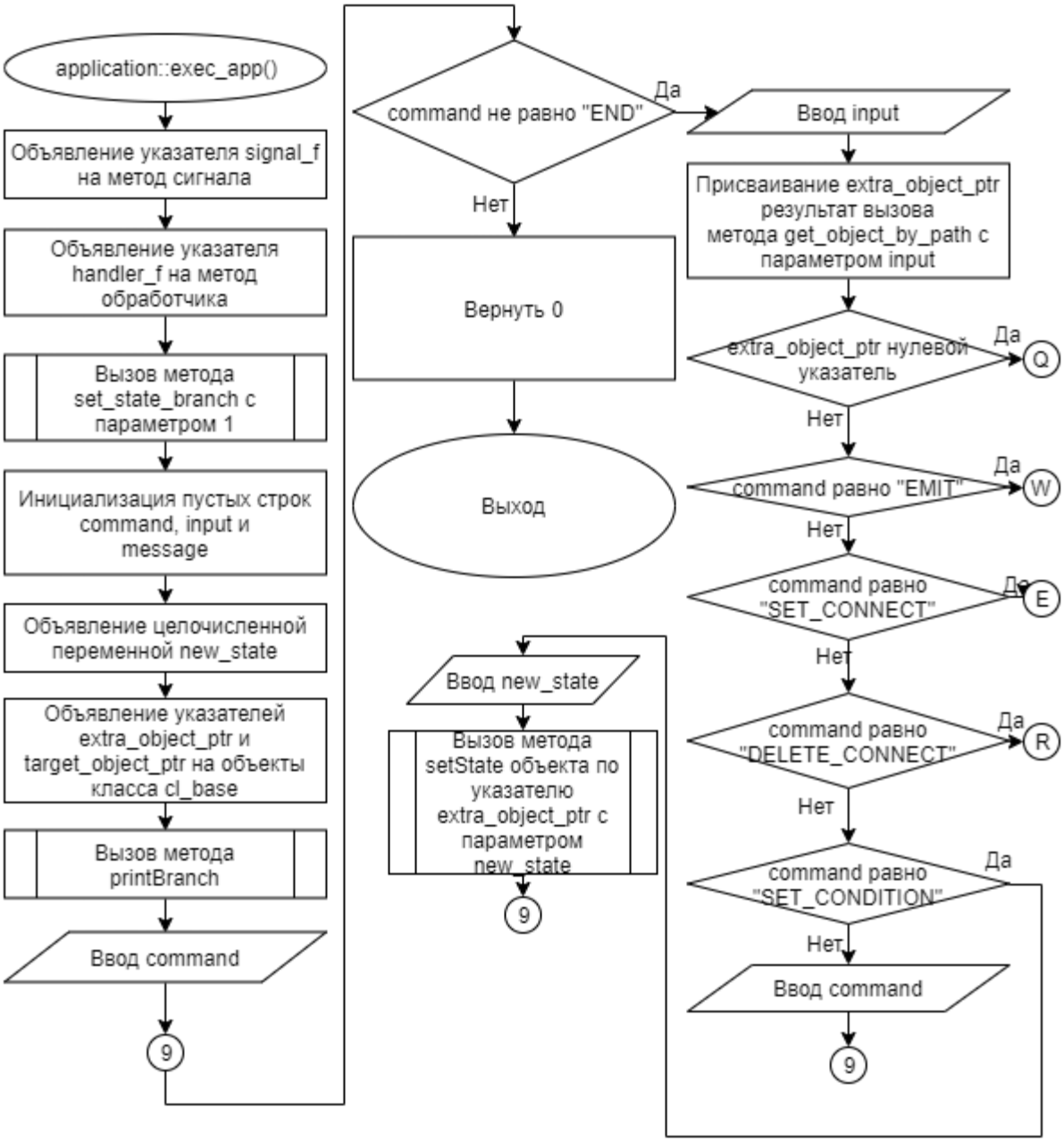

Рисунок 11 – Блок-схема алгоритма

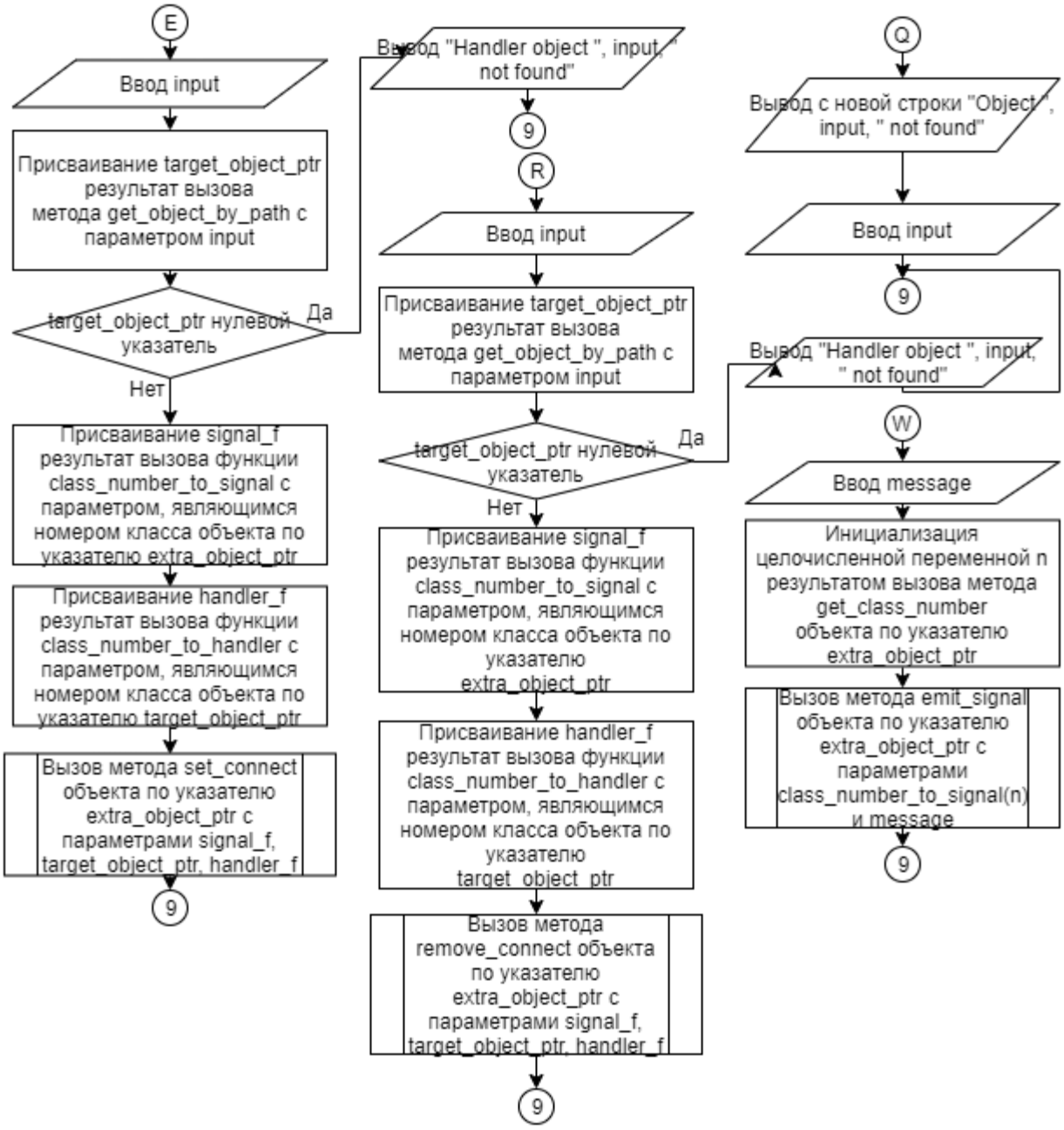

Рисунок 12 – Блок-схема алгоритма

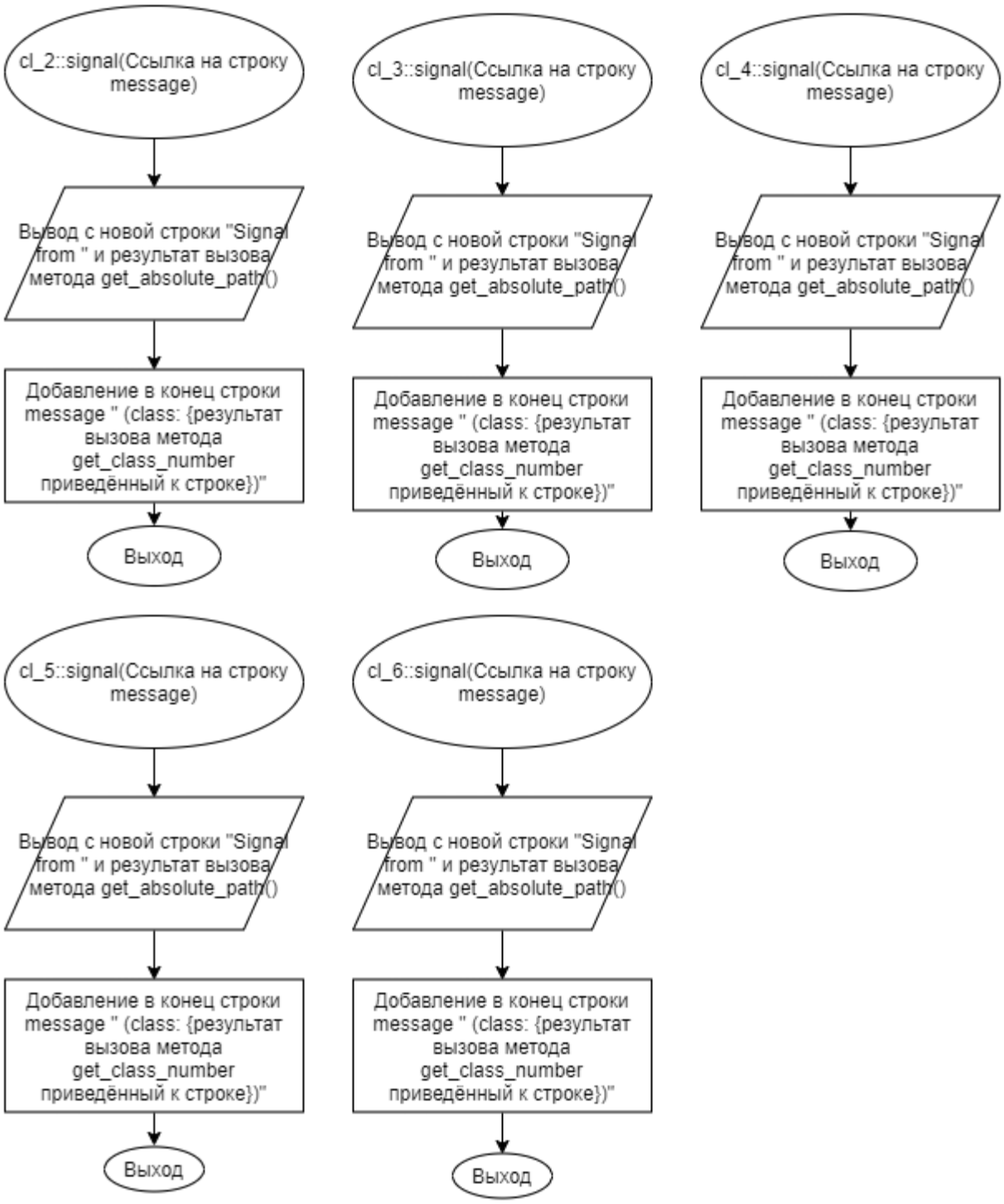

Рисунок 13 – Блок-схема алгоритма

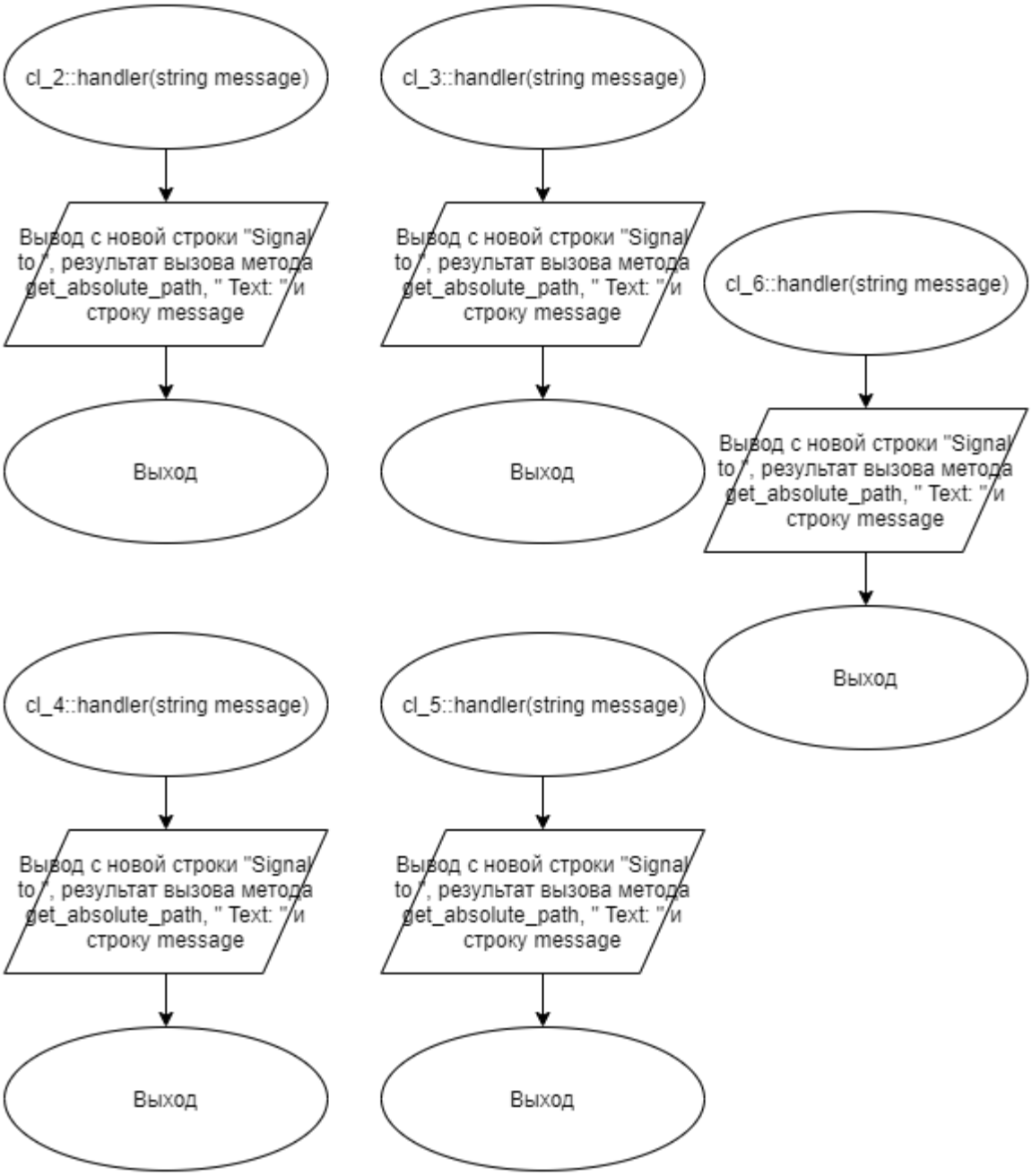

Рисунок 14 – Блок-схема алгоритма

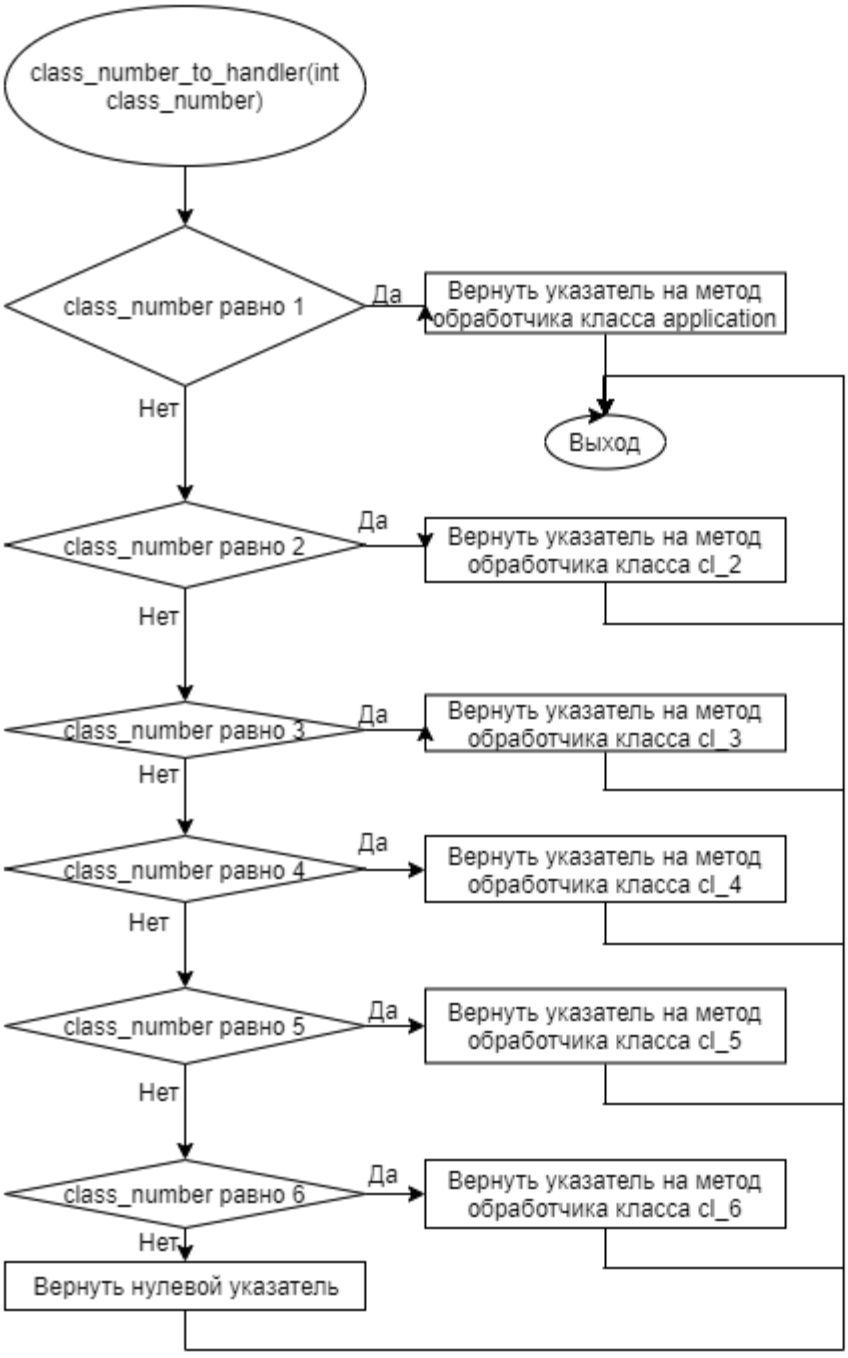

Рисунок 15 – Блок-схема алгоритма

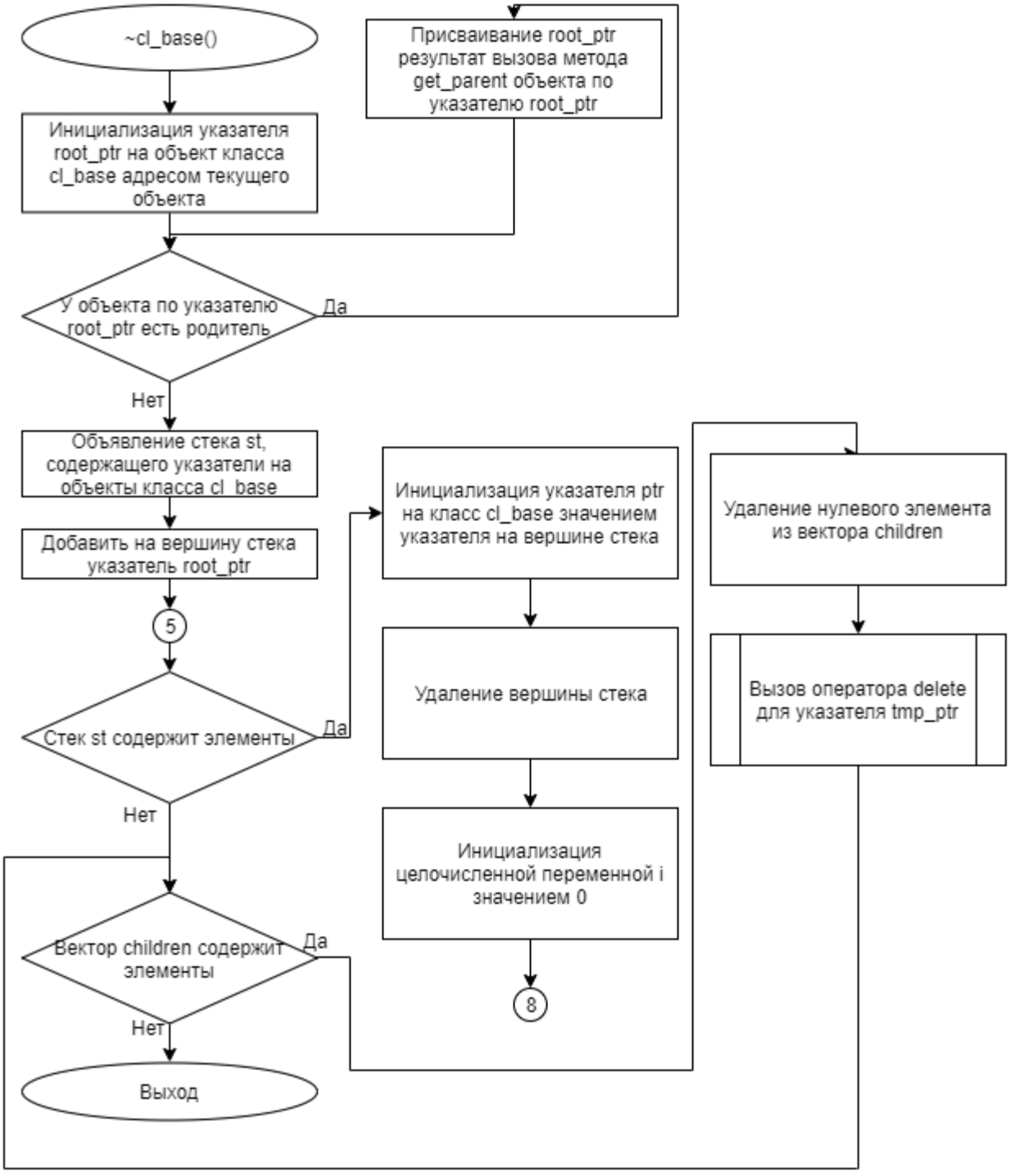

Рисунок 16 – Блок-схема алгоритма

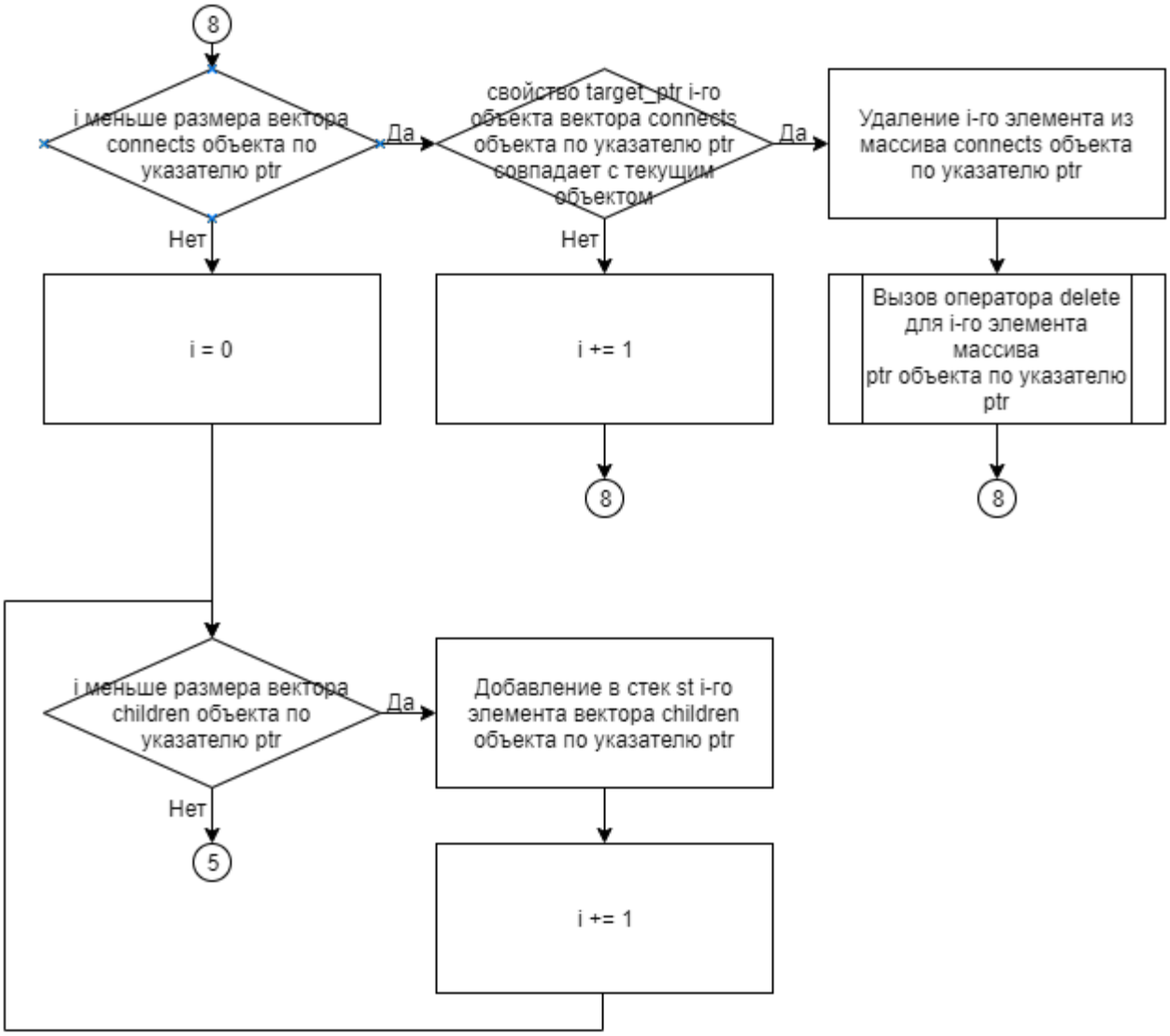

Рисунок 17 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <stack>
TYPE_SIGNAL class_number_to_signal(int class_number) {
	switch (class_number) {
		case 1:
			return SIGNAL_D(application::signal);
		case 2:
			return SIGNAL_D(cl_2::signal);
		case 3:
			return SIGNAL_D(cl_3::signal);
		case 4:
			return SIGNAL_D(cl_4::signal);
		case 5:
			return SIGNAL_D(cl_5::signal);
		case 6:
			return SIGNAL_D(cl_6::signal);
	}
	return nullptr;
}
TYPE_HANDLER class_number_to_handler(int class_number) {
	switch (class_number) {
		case 1:
			return HANDLER_D(application::handler);
		case 2:
			return HANDLER_D(cl_2::handler);
		case 3:
			return HANDLER_D(cl_3::handler);
		case 4:
			return HANDLER_D(cl_4::handler);
		case 5:
			return HANDLER_D(cl_5::handler);
		case 6:
			return HANDLER_D(cl_6::handler);
	}
	return nullptr;
}
```

```
application::application(cl_base* parent): cl_base(parent) {}

int application::exec_app() {
      TYPE_SIGNAL signal_f;
      TYPE_HANDLER handler_f;
      this->set_state_branch(1);
      string command, input, message;
      int new_state;
      cl_base* extra_object_ptr;
      cl_base* target_object_ptr;
      // stack<string> st;
      this->printBranch();
      cin >> command;
      while (command != "END") {
            cin >> input;
            extra_object_ptr = this->get_object_by_path(input);
            if (extra_object_ptr == nullptr) {
                  cout << endl << "Object " << input << " not found";
                  cin >> input;
                  continue;
            }
            if (command == "EMIT") {
                  getline(cin, message);
                  int n = extra_object_ptr->get_class_number();
                  extra_object_ptr->emit_signal(class_number_to_signal(n),
message);
            } else if (command == "SET_CONNECT") {
                  cin >> input;
                  target_object_ptr = this->get_object_by_path(input);
                  if (target_object_ptr == nullptr) {
                        cout << endl << "Handler object " << input << " not
found";
                        continue;
                  }
                  signal_f       =       class_number_to_signal(extra_object_ptr-
>get_class_number());
                  handler_f      =      class_number_to_handler(target_object_ptr-
>get_class_number());
                  extra_object_ptr->set_connect(signal_f,       target_object_ptr,
handler_f);
            } else if (command == "DELETE_CONNECT") {
                  cin >> input;
                  target_object_ptr = this->get_object_by_path(input);
                  if (target_object_ptr == nullptr) {
                        cout << endl << "Handler object " << input << " not
found";
                        continue;
                  }
                  signal_f       =       class_number_to_signal(extra_object_ptr-
>get_class_number());
                  handler_f      =      class_number_to_handler(target_object_ptr-
>get_class_number());
                  extra_object_ptr->remove_connect(signal_f,    target_object_ptr,
handler_f);
            } else if (command == "SET_CONDITION") {
                  cin >> new_state;
```

```
                  extra_object_ptr->setState(new_state);
            }
            cin >> command;
      }
      return 0;
}

int application::get_class_number() {
      return 1;
}
void application::signal(string & message) {
      cout << endl << "Signal from " << get_absolute_path();
      message += " (class: " + to_string(get_class_number()) + ")";
}
void application::handler(string message) {
      cout << endl << "Signal to " << get_absolute_path() << " Text: " << message;
}


void application::build_tree_objects() {
      cout << "Object tree";
      string path, child_name;
      int tmp;
      cin >> child_name;
      this->setName(child_name);
      cl_base* parent_node_ptr;
      cl_base* last_created_node_ptr = this;
      cin >> path;
      while (path != "endtree") {
            cin >> child_name >> tmp;
            parent_node_ptr = last_created_node_ptr->get_object_by_path(path);
            if (parent_node_ptr == nullptr) {
                  this->printBranch();
                  cout << endl << "The head object " << path << " is not found";
                  exit(1);
            }
            if (parent_node_ptr->get_child_by_name(child_name) != nullptr)
                  cout << endl << path << " Dubbing the names of subordinate
objects";
            else {
                  switch (tmp) {
                        case 1:
                              last_created_node_ptr             =             new
application(parent_node_ptr);
                              break;
                        case 2:
                              last_created_node_ptr  =  new  cl_2(parent_node_ptr,
child_name);
                              break;
                        case 3:
                              last_created_node_ptr  =  new  cl_3(parent_node_ptr,
child_name);
                              break;
                        case 4:
                              last_created_node_ptr  =  new  cl_4(parent_node_ptr,
child_name);
                              break;
```

```
                        case 5:
                              last_created_node_ptr  =  new  cl_5(parent_node_ptr,
child_name);
                              break;
                        case 6:
                              last_created_node_ptr  =  new  cl_6(parent_node_ptr,
child_name);
                              break;
                  }
            }
            cin >> path;
      }
      cl_base* target_ptr;
      string target_path;
      cin >> path;
      while (path != "end_of_connections") {
            cin >> target_path;
            parent_node_ptr = get_object_by_path(path);
            target_ptr = get_object_by_path(target_path);
            TYPE_SIGNAL    signal_f    =    class_number_to_signal(parent_node_ptr-
>get_class_number());
            TYPE_HANDLER    handler_f    =    class_number_to_handler(target_ptr-
>get_class_number());
            parent_node_ptr->set_connect(signal_f, target_ptr, handler_f);
            cin >> path;
      }
}
```

5.2 Файл application.h

Листинг 2 – application.h

```
#ifndef __APPLICATION__H
#define __APPLICATION__H

#include "cl_base.h"
class application: public cl_base {
public:
      application(cl_base* parent);
      void build_tree_objects();
      int exec_app();
      int get_class_number();
      void signal(string & message);
      void handler(string message);

};


#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}

int cl_2::get_class_number() {
      return 2;
}
void cl_2::signal(string & message) {
      cout << endl << "Signal from " << get_absolute_path();
      message += " (class: " + to_string(get_class_number()) + ")";
}
void cl_2::handler(string message) {
      cout << endl << "Signal to " << get_absolute_path() << " Text: " << message;
}
```

5.4 Файл cl_2.h

Листинг 4 – 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_object_name);
      int get_class_number();
      void signal(string & message);
      void handler(string message);
};

#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"
```

```
cl_3::cl_3(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
int cl_3::get_class_number() {
      return 3;
}
void cl_3::signal(string & message) {
      cout << endl << "Signal from " << get_absolute_path();
      message += " (class: " + to_string(get_class_number()) + ")";
}
void cl_3::handler(string message) {
      cout << endl << "Signal to " << get_absolute_path() << " Text: " << message;
}
```

5.6 Файл cl_3.h

Листинг 6 – 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_object_name);
      int get_class_number();
      void signal(string & message);
      void handler(string message);
};


#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
int cl_4::get_class_number() {
      return 4;
}
void cl_4::signal(string & message) {
      cout << endl << "Signal from " << get_absolute_path();
```

```
	message += " (class: " + to_string(get_class_number()) + ")";
}
void cl_4::handler(string message) {
	cout << endl << "Signal to " << get_absolute_path() << " Text: " << message;
}
```

5.8 Файл cl_4.h

Листинг 8 – 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_object_name);
	int get_class_number();
	void signal(string & message);
	void handler(string message);
};

#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
int cl_5::get_class_number() {
	return 5;
}
void cl_5::signal(string & message) {
	cout << endl << "Signal from " << get_absolute_path();
	message += " (class: " + to_string(get_class_number()) + ")";
}
void cl_5::handler(string message) {
	cout << endl << "Signal to " << get_absolute_path() << " Text: " << message;
}
```

5.10 Файл cl_5.h

Листинг 10 – 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_object_name);
      int get_class_number();
      void signal(string & message);
      void handler(string message);
};

#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
int cl_6::get_class_number() {
      return 6;
}
void cl_6::signal(string & message) {
      cout << endl << "Signal from " << get_absolute_path();
      message += " (class: " + to_string(get_class_number()) + ")";
}
void cl_6::handler(string message) {
      cout << endl << "Signal to " << get_absolute_path() << " Text: " << message;
}
```

5.12 Файл cl_6.h

Листинг 12 – 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_object_name);
	int get_class_number();
	void signal(string & message);
	void handler(string message);
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
#include <stack>

cl_base::cl_base(cl_base* parent, string name): parent(parent), name(name) {
	if (parent != nullptr) {
		parent->children.push_back(this);
	}
}
cl_base::~cl_base() {
	cl_base* root_ptr = this;
	while (root_ptr->get_parent() != nullptr) {
		root_ptr = root_ptr->get_parent();
	}
	stack<cl_base*> st;
	st.push(root_ptr);
	while (!st.empty()) {
		cl_base* ptr = st.top();
		st.pop();
		int i = 0;
		while (i < ptr->connects.size()) {
			if (ptr->connects[i]->target_ptr == this) {
				ptr->connects.erase(ptr->connects.begin() + i);
				delete ptr->connects[i];
			}
			else {
				i++;
			}
		}
		for (i = 0; i < ptr->children.size(); ++i)
			st.push(ptr->children[i]);
	}
	while (!children.empty()) {
		cl_base* tmp_ptr = children[0];
		children.erase(children.begin());
		delete tmp_ptr;
	}
}

string cl_base::get_name() const {return name;}
```

```
cl_base* cl_base::get_parent() const {return parent;}

bool cl_base::setName(string name1) {
	if (get_parent() != nullptr && get_parent() -> get_child_by_name(name1) != nullptr) {
		return false;
	}
	name = name1;
	return true;
}

cl_base* cl_base::get_child_by_name(string name) {
	for (auto child: children) {
		if (child->name == name) return child;
	}
	return nullptr;
}

//2 часть

cl_base* cl_base::findObjOnBranch(string s_object_name) {
	cl_base* found = nullptr;
	queue <cl_base*> elementsQueue;
	elementsQueue.push(this);
	while(!elementsQueue.empty()) {
		cl_base* elem = elementsQueue.front();
		elementsQueue.pop();
		if (elem->name == s_object_name) {
			if (found != nullptr) {
				return nullptr;
			}
			else {
				found = elem;
			}
		}
		for (int i = 0; i < elem->children.size();i++) {
			elementsQueue.push(elem->children[i]);
		}
	}
	return found;
}
cl_base* cl_base::findObjOnTree(string s_object_name) {
	if (parent != nullptr) {
		return parent->findObjOnTree(s_object_name);
	}
	else {
		return findObjOnBranch(s_object_name);
	}
}
void cl_base::printBranch(int level) {
	cout << endl;
	for (int i = 0; i < level; ++i) {
		cout << "    ";
	}
	cout << this->get_name();
	for (int i = 0; i < children.size(); ++i) {
		children[i]->printBranch(level + 1);
```

```
	}
}
void cl_base::printBranchWithState(int level) {
	cout << endl;
	for (int i = 0; i < level; ++i) {
		cout << "    ";
	}
	if (this->state != 0) {
		cout << this->get_name() << " is ready";
	}
	else {
		cout << this->get_name() << " is not ready";
	}
	for (int i= 0; i < children.size(); ++i) {
		children[i]->printBranchWithState(level + 1);
	}
}
void cl_base::setState(int state) {
	if (parent == nullptr || parent->state != 0) {
		this->state = state;
	}
	if (state == 0) {
		this->state = state;
		for (int i = 0; i < children.size(); i++) {
			children[i]->setState(state);
		}
	}
}

//3 часть

bool cl_base::set_parent(cl_base* new_parent) {
	if (this->get_parent() == new_parent) {
		return true;
	}
	if (this->get_parent() == nullptr || new_parent == nullptr) {
		return false;
	}
	if (new_parent->get_child_by_name(this->get_name()) != nullptr) {
		return false;
	}
	stack<cl_base*> st;
	st.push(this);
	while (!st.empty()) {
		cl_base* current_node_ptr = st.top();
		st.pop();
		if (current_node_ptr == new_parent) {
			return false;
		}
		for (int i = 0; i < current_node_ptr->children.size(); ++i) {
			st.push(current_node_ptr->children[i]);
		}
	}
	vector<cl_base*> & v = this->get_parent()->children;
	for (int i = 0; i < v.size(); ++i) {
		if (v[i]->get_name() == this->get_name()) {
			v.erase(v.begin() + i);
```

```
                  new_parent->children.push_back(this);
                  return true;
            }
      }
      return false;
}
void cl_base::remove_child_by_name(string child_name) {
      vector<cl_base*> & v = this->children;
      for (int i = 0; i < v.size(); ++i) {
            if (v[i]->get_name() == child_name) {
                  delete v[i];
                  v.erase(v.begin() + i);
                  return;
            }
      }
}
cl_base* cl_base::get_object_by_path(string path) {
      if (path.empty()) {
            return nullptr;
      }
      if (path == ".") {
            return this;
      }
      if (path[0] == '.') {
            return findObjOnBranch(path.substr(1));
      }
      if (path.substr(0, 2) == "//") {
            return this->findObjOnTree(path.substr(2));
      }
      if (path[0] != '/') {
            size_t slash_index = path.find('/');
            cl_base*     child_ptr     =     this->get_child_by_name(path.substr(0,
slash_index));
            if (child_ptr == nullptr || slash_index == string::npos) {
                  return child_ptr;
            }
            return child_ptr->get_object_by_path(path.substr(slash_index + 1));
      }
      cl_base* root_ptr = this;
      while (root_ptr->get_parent() != nullptr) {
            root_ptr = root_ptr->get_parent();
      }
      if (path == "/") {
            return root_ptr;
      }
      return root_ptr->get_object_by_path(path.substr(1));
}

//4 часть

void    cl_base::set_connect(TYPE_SIGNAL    signal_ptr,    cl_base*    target_ptr,
TYPE_HANDLER handler_ptr) {
      for (int i = 0; i < connects.size(); ++i) {
            if (connects[i]->signal_ptr == signal_ptr && connects[i]->target_ptr
== target_ptr && connects[i]->handler_ptr == handler_ptr) {
                  return;
            }
```

```
	}
	connect * new_connect = new connect();
	new_connect->signal_ptr = signal_ptr;
	new_connect->target_ptr = target_ptr;
	new_connect->handler_ptr = handler_ptr;
	connects.push_back(new_connect);
}
void   cl_base::remove_connect(TYPE_SIGNAL   signal_ptr,   cl_base*   target_ptr,
TYPE_HANDLER handler_ptr) {
	for (int i = 0; i < connects.size(); ++i) {
		if (connects[i]->signal_ptr == signal_ptr && connects[i]->target_ptr
== target_ptr && connects[i]->handler_ptr == handler_ptr) {
			delete connects[i];
			connects.erase(connects.begin() + i);
			return;
		}
	}
}
void cl_base::emit_signal(TYPE_SIGNAL signal_ptr, string & command) {
	if (this->state == 0) {
		return;
	}
	(this->*signal_ptr)(command);
	for (int i = 0; i < connects.size(); ++i) {
		if (connects[i]->signal_ptr == signal_ptr) {
			TYPE_HANDLER handler_ptr = connects[i]->handler_ptr;
			cl_base* target_ptr = connects[i]->target_ptr;
			if (target_ptr->state != 0) {
				(target_ptr->*handler_ptr)(command);
			}
		}
	}
}
string cl_base::get_absolute_path() {
	string result;
	stack<string> st;
	cl_base* root_ptr = this;
	while (root_ptr->get_parent() != nullptr) {
		st.push(root_ptr->get_name());
		root_ptr = root_ptr->get_parent();
	}
	while (!st.empty()) {
		result += '/' + st.top();
		st.pop();
	}
	if (result.empty()) {
		return "/";
	}
	return result;
}
int cl_base::get_class_number() {
	return 0;
}
void cl_base::set_state_branch(int new_state) {
	if (get_parent() != nullptr && get_parent()->state == 0) {
		return;
	}
```

```
	setState(new_state);
	for (int i = 0; i < children.size(); ++i) {
		children[i]->set_state_branch(new_state);
	}
}
```

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>
#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);

class cl_base {
	struct connect {
		TYPE_SIGNAL signal_ptr;
		cl_base* target_ptr;
		TYPE_HANDLER handler_ptr;
	};
private:
	int state = 0;
	cl_base* parent;
	vector <cl_base*> children;
	string name;
	vector<connect*> connects;
public:
	cl_base(cl_base* parent, string name = "Object_root");
	~cl_base();
	bool setName(string name);
	string get_name() const;
	cl_base* get_parent() const;
	cl_base* get_child_by_name(string name);
	//2 часть
	cl_base* findObjOnBranch(string name);
	cl_base* findObjOnTree(string name);
	void printBranch(int level = 0);
	void printBranchWithState(int level = 0);
	void setState(int state);
	//3 часть
	bool set_parent(cl_base* new_parent);
```

```
	void remove_child_by_name(string child_name);
	cl_base* get_object_by_path(string path);
	//4 часть
	void set_connect(TYPE_SIGNAL signal_ptr, cl_base* target_ptr, TYPE_HANDLER
handler_ptr);
	void remove_connect(TYPE_SIGNAL signal_ptr, cl_base* target_ptr,
TYPE_HANDLER handler_ptr);
	void emit_signal(TYPE_SIGNAL signal_ptr, string & command);
	string get_absolute_path();
	virtual int get_class_number();
	void set_state_branch(int new_state);
};


#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "application.h"
int main() {
	application ob_application(nullptr);
	ob_application.build_tree_objects();
	return (ob_application.exec_app());
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 31.

Таблица 31 – 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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. 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).

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).