СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....7
1 ПОСТАНОВКА ЗАДАЧИ.....9
1.1 Описание входных данных.....11
1.2 Описание выходных данных.....13
2 МЕТОД РЕШЕНИЯ.....15
3 ОПИСАНИЕ АЛГОРИТМОВ.....18
3.1 Алгоритм метода set_connect класса cl_base.....18
3.2 Алгоритм метода remove_connect класса cl_base.....19
3.3 Алгоритм метода emit_signal класса cl_base.....20
3.4 Алгоритм метода get_absolute_path класса cl_base.....21
3.5 Алгоритм метода set_state_branch класса cl_base.....22
3.6 Алгоритм метода signal класса cl_application.....23
3.7 Алгоритм метода handler класса cl_application.....23
3.8 Алгоритм метода get_class_number класса cl_application.....24
3.9 Алгоритм метода signal класса cl_2.....24
3.10 Алгоритм метода handler класса cl_2.....25
3.11 Алгоритм метода get_class_number класса cl_2.....25
3.12 Алгоритм метода signal класса cl_3.....25
3.13 Алгоритм метода handler класса cl_3.....26
3.14 Алгоритм метода get_class_number класса cl_3.....26
3.15 Алгоритм метода signal класса cl_4.....27
3.16 Алгоритм метода handler класса cl_4.....27
3.17 Алгоритм метода get_class_number класса cl_4.....27
3.18 Алгоритм метода signal класса cl_5.....28
3.19 Алгоритм метода handler класса cl_5.....28
3.20 Алгоритм метода get_class_number класса cl_5.....29

3.21 Алгоритм метода signal класса cl_6.....29
3.22 Алгоритм метода handler класса cl_6.....30
3.23 Алгоритм метода get_class_number класса cl_6.....30
3.24 Алгоритм функции class_number_to_signal.....30
3.25 Алгоритм метода build_tree_objects класса cl_application.....31
3.26 Алгоритм метода exec_app класса cl_application.....34
3.27 Алгоритм функции main.....37
3.28 Алгоритм функции class_number_to_handler.....37
3.29 Алгоритм деструктора класса cl_base.....38
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....41
5 КОД ПРОГРАММЫ.....56
5.1 Файл cl_2.cpp.....56
5.2 Файл cl_2.h.....56
5.3 Файл cl_3.cpp.....57
5.4 Файл cl_3.h.....57
5.5 Файл cl_4.cpp.....58
5.6 Файл cl_4.h.....58
5.7 Файл cl_5.cpp.....59
5.8 Файл cl_5.h.....59
5.9 Файл cl_6.cpp.....60
5.10 Файл cl_6.h.....60
5.11 Файл cl_application.cpp.....61
5.12 Файл cl_application.h.....64
5.13 Файл cl_base.cpp.....65
5.14 Файл cl_base.h.....72
5.15 Файл main.cpp.....73
6 ТЕСТИРОВАНИЕ.....75

ЗАКЛЮЧЕНИЕ.....78
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....79

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

Объектно-ориентированное программирование (ООП) — это парадигма разработки программного обеспечения, в которой основная концепция заключается в понятии объекта. Объекты представляют собой сущности, обладающие состоянием и поведением, которые определяются классами. Класс выступает как шаблон, по которому создаются объекты, наделенные конкретными атрибутами и методами. ООП позволяет моделировать реальные сущности, создавая программы, которые легко понимаются и поддерживаются.

Появление ООП было обусловлено необходимостью решения сложных задач, связанных с построением больших и многоуровневых систем. Использование объектов и классов позволяет разработчикам писать код, который можно многократно использовать и адаптировать под различные данные. Это снижает количество повторного кода и облегчает сопровождение программ. Основные принципы ООП включают абстракцию, полиморфизм, наследование и инкапсуляцию.

Абстракция позволяет выделять значимые характеристики объектов, игнорируя несущественные детали. Полиморфизм дает возможность работать с различными объектами через единый интерфейс, не зная их конкретного типа и внутренней структуры. Наследование позволяет создавать новые классы на основе

существующих, используя их функциональность и добавляя новые возможности. Инкапсуляция объединяет данные и методы в классах, скрывая внутренние детали реализации и предоставляя удобный интерфейс для взаимодействия с объектами.

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:

- структура connect имеет указатель на метод сигнала signal_ptr, указатель на целевой объект target_ptr, указатель на метод обработчика handler_ptr.

Класс cl_base:

- свойства/поля:
    - поле вектор для хранения установленных связей:
        - наименование — connects;
        - тип — ;
        - модификатор доступа — private;
- 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 об;
    - 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 установки объекту и всем его потомкам значение готовности.

Класс cl_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.

Таблица 1 – Иерархия наследования классов

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

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода set_connect класса cl_base

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

Параметры: указатель на функцию сигнал, указатель на целевой объект который обрабатывает сигнал, указатель на функцию обработчик.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_connect класса cl_base

№	Предикат	Действия	№ перехода
1		объявление и инициализация целочисленной переменной i значением 0	2
2	i меньше размера вектора connects		3
			4
3	переданные параметры совпадают с полями связи вектора connects по индексу i		∅
		к значению i прибавляется 1	2
4		создание объекта структуры connect с помощью оператора new и присваивание указателю new_connect адрес этого объекта	5

№	Предикат	Действия	№ перехода
5		присваивание 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		присваивание полю handler_ptr объекта по указателю new_connect значение параметра handler_ptr	8
8		добавление указателя new_connect в вектор connects	∅

3.2 Алгоритм метода remove_connect класса cl_base

Функционал: метод удаления (разрыва) связи между сигналом текущего об.

Параметры: указатель на функцию сигнала signal_ptr, указатель на целевой объект target_ptr, указатель на обработчик handler_ptr.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода remove_connect класса cl_base

№	Предикат	Действия	№ перехода
1		объявление и инициализация целочисленной переменной i значением 0	2
2	i меньше размера вектора connects		3
			∅
3	переданные параметры совпадают с полями связи	вызов деструктора для объекта по i-му указателю вектора connects	4

№	Предикат	Действия	№ перехода
	вектора connects по индексу i		
		к значению i прибавляется 1	2
4		удаление элемента вектора connects по индексу i	∅

3.3 Алгоритм метода emit_signal класса cl_base

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

Параметры: указатель на функцию сигнала signal_ptr, строка command содержащая команду или данные которые должны быть переданы вместе с сигналом.

Возвращаемое значение: ничего не возвращает.

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 по указателю signal_ptr с параметром command	3
3		объявление и инициализация целочисленной переменной i значением 0	4
4	i меньше размера вектора connects		5
			∅
5	свойство signal_ptr связи по индексу i вектора connects равно signal_ptr		7

№	Предикат	Действия	№ перехода
			6
6		к значению i прибавляется 1	4
7		объявление и инициализация указателя на метод обработчика handler_ptr значением свойства handler_ptr объекта вектора connects по индексу i	8
8		объявление и инициализация указателя target_ptr на объект класса cl_base значением свойства target_ptr объекта вектора connects по индексу i	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

3.4 Алгоритм метода get_absolute_path класса cl_base

Функционал: метод получения абсолютного пути объекта.

Параметры: нет.

Возвращаемое значение: строковый тип.

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		объявление и инициализация указателя root_ptr на объект класса cl_base адресом текущего объекта	4
4	результат вызова метода get_parent (возвращает имя объекта)объекта по	добавить в стек st результат вызова метода get_name(возвращает имя объекта) объекта по указателю root_ptr	5

№	Предикат	Действия	№ перехода
	указателю root_ptr не равен нулевому указателю		
			6
5		присваивание root_ptr результата вызова метода get_parent(возвращает имя объекта) 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	∅

3.5 Алгоритм метода set_state_branch класса cl_base

Функционал: метод установки объекту и всем его потомкам значение готовности.

Параметры: целочисленная переменная new_state(новое состояние, которое должно быть установлено для текущего объекта и его дочерних объектов).

Возвращаемое значение: ничего не возвращает.

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		объявление и инициализация целочисленной	4

№	Предикат	Действия	№ перехода
		переменной i значением 0	
4	i меньше размера вектора children		5
			∅
5		вызов метода set_state_branch объекта по указателю вектора children по индексу i с параметром new_state	6
6		к значению i прибавляется 1	4

3.6 Алгоритм метода signal класса cl_application

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода signal класса cl_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 приведенный к строке})"	∅

3.7 Алгоритм метода handler класса cl_application

Функционал: метод обработчика.

Параметры: строка message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода handler класса cl_application

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

3.8 Алгоритм метода get_class_number класса cl_application

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: целочисленное значение.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_class_number класса cl_application

№	Предикат	Действия	№ перехода
1		вернуть 1	∅

3.9 Алгоритм метода signal класса cl_2

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 10.

Таблица 10 – 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 приведенный к строке})"	∅

3.10 Алгоритм метода handler класса cl_2

Функционал: метод обработчика.

Параметры: строка message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 11.

Таблица 11 – 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	∅

3.11 Алгоритм метода get_class_number класса cl_2

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: целочисленное значение.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода get_class_number класса cl_2

№	Предикат	Действия	№ перехода
1		вернуть 2	∅

3.12 Алгоритм метода signal класса cl_3

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 13.

Таблица 13 – 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 приведенный к строке})"	∅

3.13 Алгоритм метода handler класса cl_3

Функционал: метод обработчика.

Параметры: строка message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 14.

Таблица 14 – 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	∅

3.14 Алгоритм метода get_class_number класса cl_3

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: целочисленное значение.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода get_class_number класса cl_3

№	Предикат	Действия	№ перехода
1		вернуть 3	∅

3.15 Алгоритм метода signal класса cl_4

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 16.

Таблица 16 – 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 приведенный к строке})"	∅

3.16 Алгоритм метода handler класса cl_4

Функционал: метод обработчика.

Параметры: строка message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 17.

Таблица 17 – 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	∅

3.17 Алгоритм метода get_class_number класса cl_4

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: целочисленное значение.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода get_class_number класса cl_4

№	Предикат	Действия	№ перехода
1		вернуть 4	∅

3.18 Алгоритм метода signal класса cl_5

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 19.

Таблица 19 – 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 приведенный к строке})"	∅

3.19 Алгоритм метода handler класса cl_5

Функционал: метод обработчика.

Параметры: строка message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 20.

Таблица 20 – 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	∅

3.20 Алгоритм метода get_class_number класса cl_5

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: целочисленное значение.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода get_class_number класса cl_5

№	Предикат	Действия	№ перехода
1		вернуть 5	∅

3.21 Алгоритм метода signal класса cl_6

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 22.

Таблица 22 – 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 приведенный к строке})"	∅

3.22 Алгоритм метода handler класса cl_6

Функционал: метод обработчика.

Параметры: строка message.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 23.

Таблица 23 – 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	∅

3.23 Алгоритм метода get_class_number класса cl_6

Функционал: метод возврата номера класса.

Параметры: нет.

Возвращаемое значение: целочисленное значение.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода get_class_number класса cl_6

№	Предикат	Действия	№ перехода
1		вернуть 6	∅

3.24 Алгоритм функции class_number_to_signal

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

Параметры: целочисленное значение параметра class_number.

Возвращаемое значение: указатель на метод сигнала.

Алгоритм функции представлен в таблице 25.

Таблица 25 – Алгоритм функции class_number_to_signal

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

3.25 Алгоритм метода build_tree_objects класса cl_application

Функционал: строит дерево иерархии и устанавливает связь между объектами.

Параметры: нет.

Возвращаемое значение: ничего не возвращает.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2
2		объявление строк path, child_name	3
3		объявление целочисленной переменной tmp	4
4		ввод child_name	5

№	Предикат	Действия	№ перехода
5		вызов метода setName(устанавливает имя объекта) с параметром child_name	6
6		объявление указателя parent_node_ptr на объект класса cl_base	7
7		объявление и инициализация указателя last_created_node_ptr адресом текущего объекта	8
8		ввод path	9
9	строка path не равно "endtree"	ввод child_name и tmp	10
			16
10		присваивание parent_node_ptr результат вызова метода get_object_by_path(метод получения указателя на любой объект в составе дерева иерархии согласно пути) с параметром path объекта по указателю last_created_node_ptr	11
11	parent_node_ptr нулевой указатель	вызов метода printBranch(метод вывода дерева иерархии объектов)	12
			14
12		вывод с новой строки "The head object ", path, " is not found"	13
13		выход из метода с кодом 1	∅
14	у объекта по указателю parent_node_ptr нет подчиненного объекта с именем child_name	вывод с новой строки path, " Dubbing the names of subordinate objects"	15
	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	15

№	Предикат	Действия	№ перехода
		адрес этого объекта	
			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(метод получения указателя на любой объект в составе дерева иерархии согласно пути) с параметром path	21
21		присваивание target_ptr результат вызова метода get_object_by_path(метод получения указателя на любой объект в составе дерева иерархии согласно пути) с параметром target_ptr	22
22		объявление и 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
23		объявление и 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,	25

№	Предикат	Действия	№ перехода
		handler_ptr	
25		ввод path	19

3.26 Алгоритм метода exec_app класса cl_application

Функционал: обрабатывает команды пользователя.

Параметры: нет.

Возвращаемое значение: целочисленное значение.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода exec_app класса cl_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		объявление и инициализация пустых строк command, input, message	5
5		объявление целочисленной переменной new_state	6
6		объявление указателей extra_object_ptr и target_object_ptr на объекты класса cl_base	7
7		вызов метода printBranch(метод вывода дерева иерархии объектов)	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(метод получения указателя на любой объект в составе дерева	11

№	Предикат	Действия	№ перехода
		иерархии согласно пути) с параметром input	
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	13
13		вывод с новой строки "Object ", input, " not found"	14
14		ввод input	9
15		ввод message	16
16		объявление и инициализация целочисленной переменной n результатом вызова метода get_class_number объекта по указателю extra_object_ptr	17
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(метод получения указателя на любой объект в составе дерева иерархии согласно пути) с параметром input	20
20	target_object_ptr нулевой указатель	вывод "Handler object ", input, " not found"	9

№	Предикат	Действия	№ перехода
		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_signal 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(метод получения указателя на любой объект в составе дерева иерархии согласно пути) 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 номером класса объекта по указателю extra_object_ptr	26
26		присваивание handler_f результат вызова функции class_number_to_signal 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

№	Предикат	Действия	№ перехода
28		ввод new_state	29
29		вызов метода setState(метод устанавливает готовность объекта) объекта по указателю extra_object_ptr с параметром new_state	9

3.27 Алгоритм функции main

Функционал: основная функция.

Параметры: нет.

Возвращаемое значение: целочисленный флаг выхода из программы.

Алгоритм функции представлен в таблице 28.

Таблица 28 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_application с использованием параметризированного конструктора и передачей в него в качестве аргумента пустого указателя	2
2		вызов метода build_tree_objects объекта ob_application	3
3		возвращение результата работы метода exec_app() для объекта ob_application	∅

3.28 Алгоритм функции class_number_to_handler

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

Параметры: целочисленное значение параметра 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	значение class_number равно 1	вернуть указатель на метод обработчика класса cl_application	∅
	значение class_number равно 2	вернуть указатель на метод обработчика класса cl_2	∅
	значение class_number равно 3	вернуть указатель на метод обработчика класса cl_3	∅
	значение class_number равно 4	вернуть указатель на метод обработчика класса cl_4	∅
	значение class_number равно 5	вернуть указатель на метод обработчика класса cl_5	∅
	значение class_number равно 6	вернуть указатель на метод обработчика класса cl_6	∅
		вернуть нулевой указатель	∅

3.29 Алгоритм деструктора класса cl_base

Функционал: уничтожает объект и его потомков, удаляет все вхождения объекта и его потомков в связях объектов дерева.

Параметры: нет.

Алгоритм деструктора представлен в таблице 30.

Таблица 30 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		объявление и 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

№	Предикат	Действия	№ перехода
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	объявление и инициализация указателя ptr на класс cl_base значением указателя на вершине стека	6
			14
6		удаление вершины стека	7
7		объявление и 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 элемента по индексу i вектора children объекта по указателю ptr	11
			5
11		i += 1	10
12	свойство target_ptr объекта по индексу i вектора connects объекта по указателю ptr совпадает с текущим объектом	удаление элемента по индексу i из массива connects объекта по указателю ptr	13
		i += 1	8
13		вызов оператора delete для элемента по индексу i массива ptr объекта по указателю ptr	8

№	Предикат	Действия	№ перехода
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 1-15.

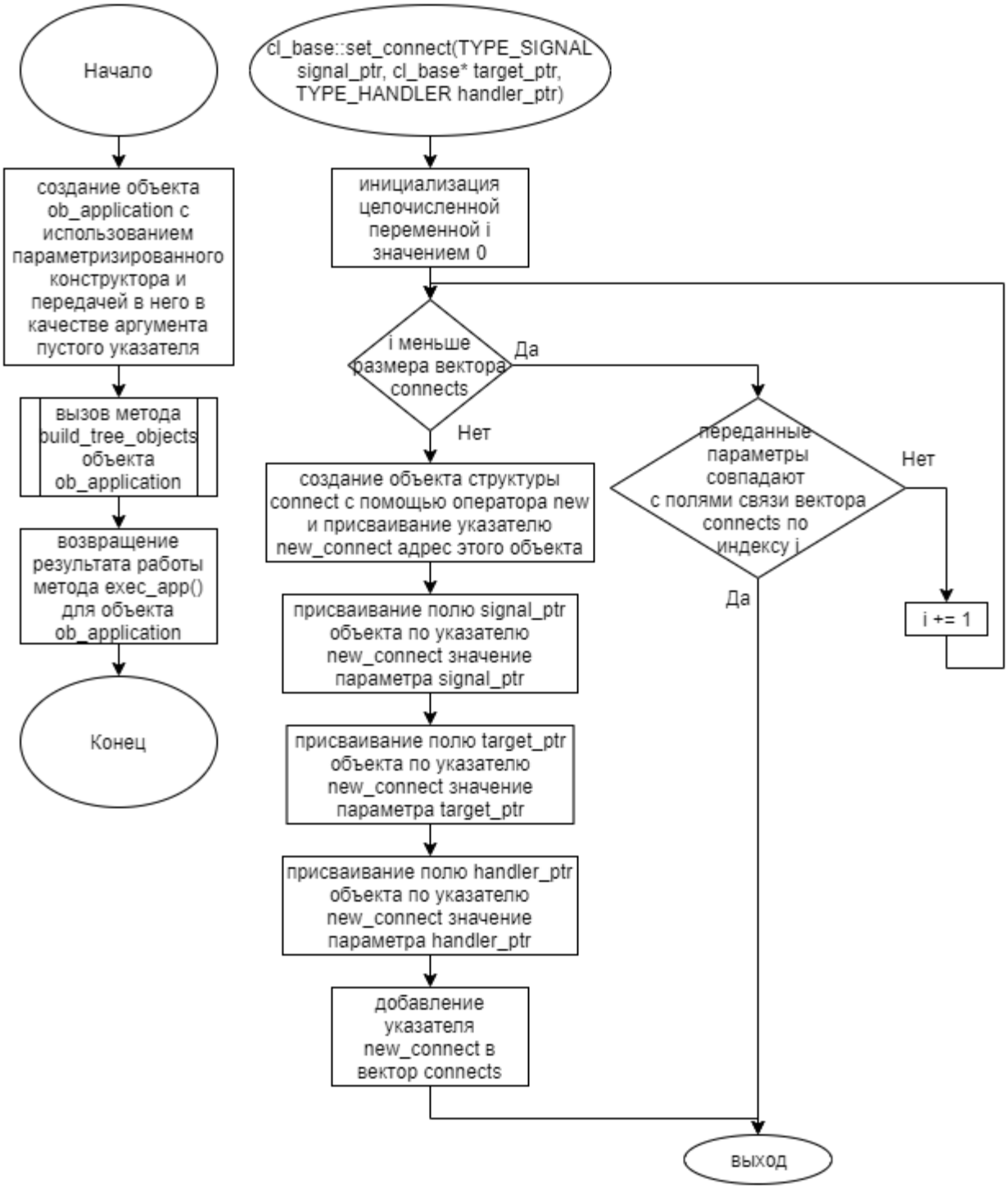

Рисунок 1 – Блок-схема алгоритма

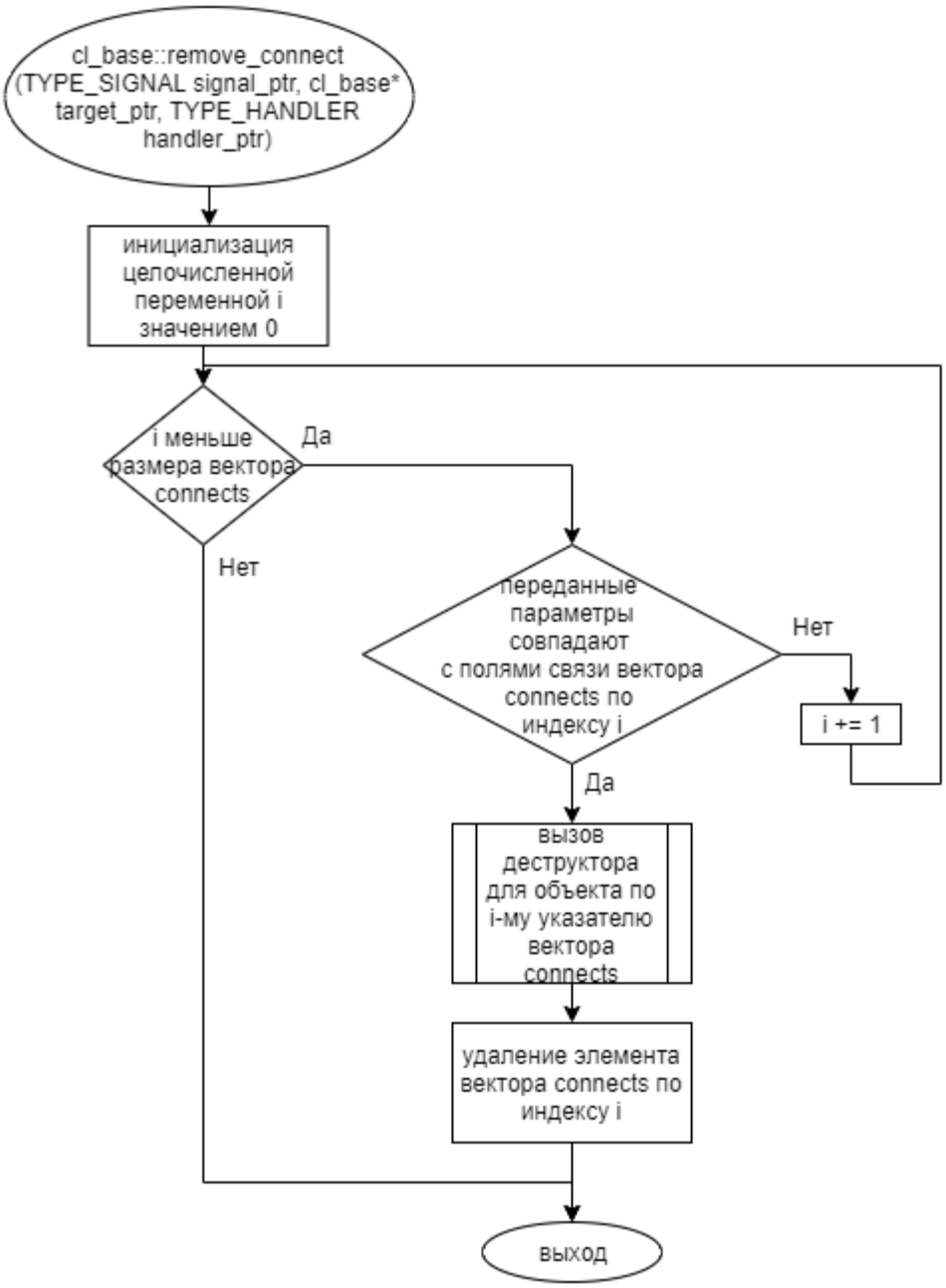

Рисунок 2 – Блок-схема алгоритма

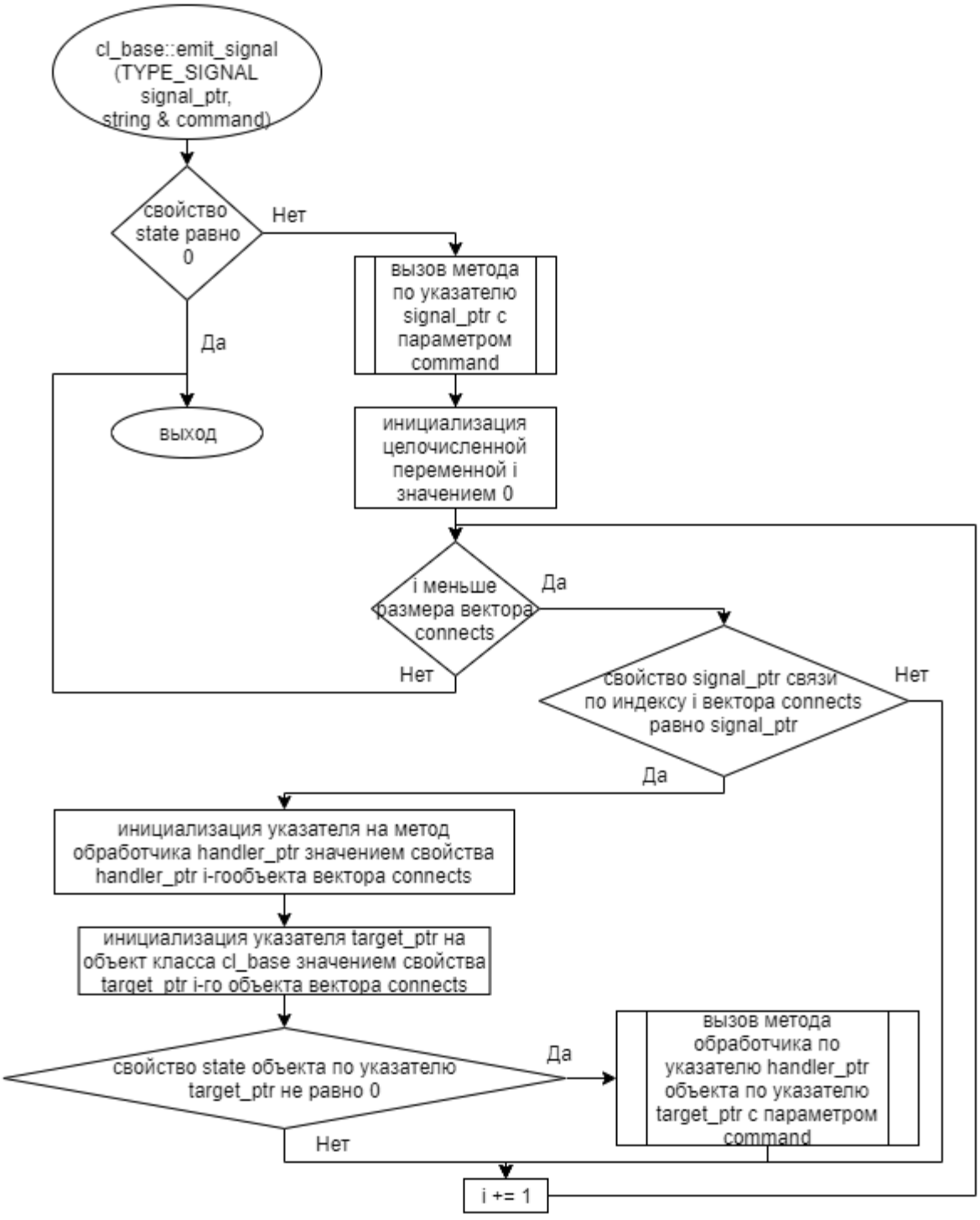

Рисунок 3 – Блок-схема алгоритма

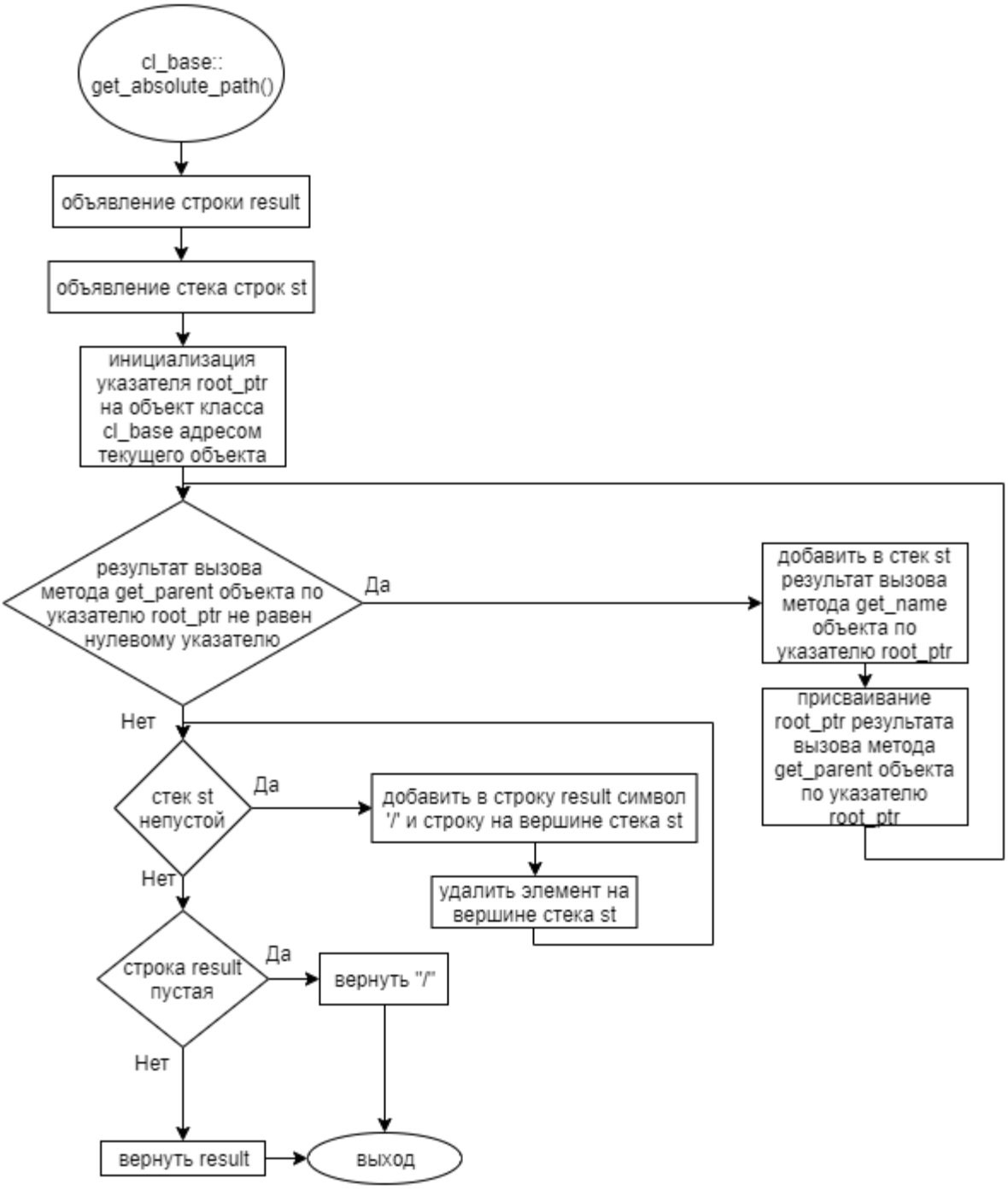

Рисунок 4 – Блок-схема алгоритма

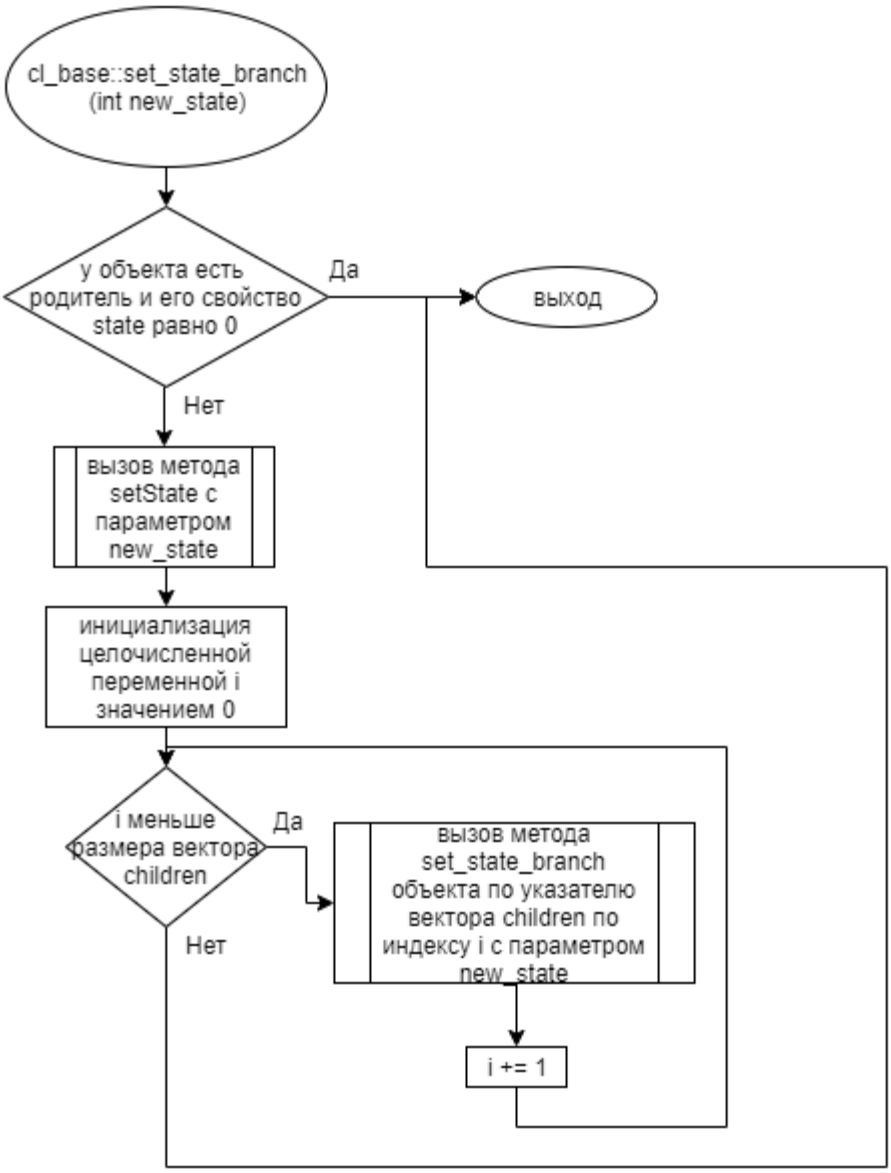

Рисунок 5 – Блок-схема алгоритма

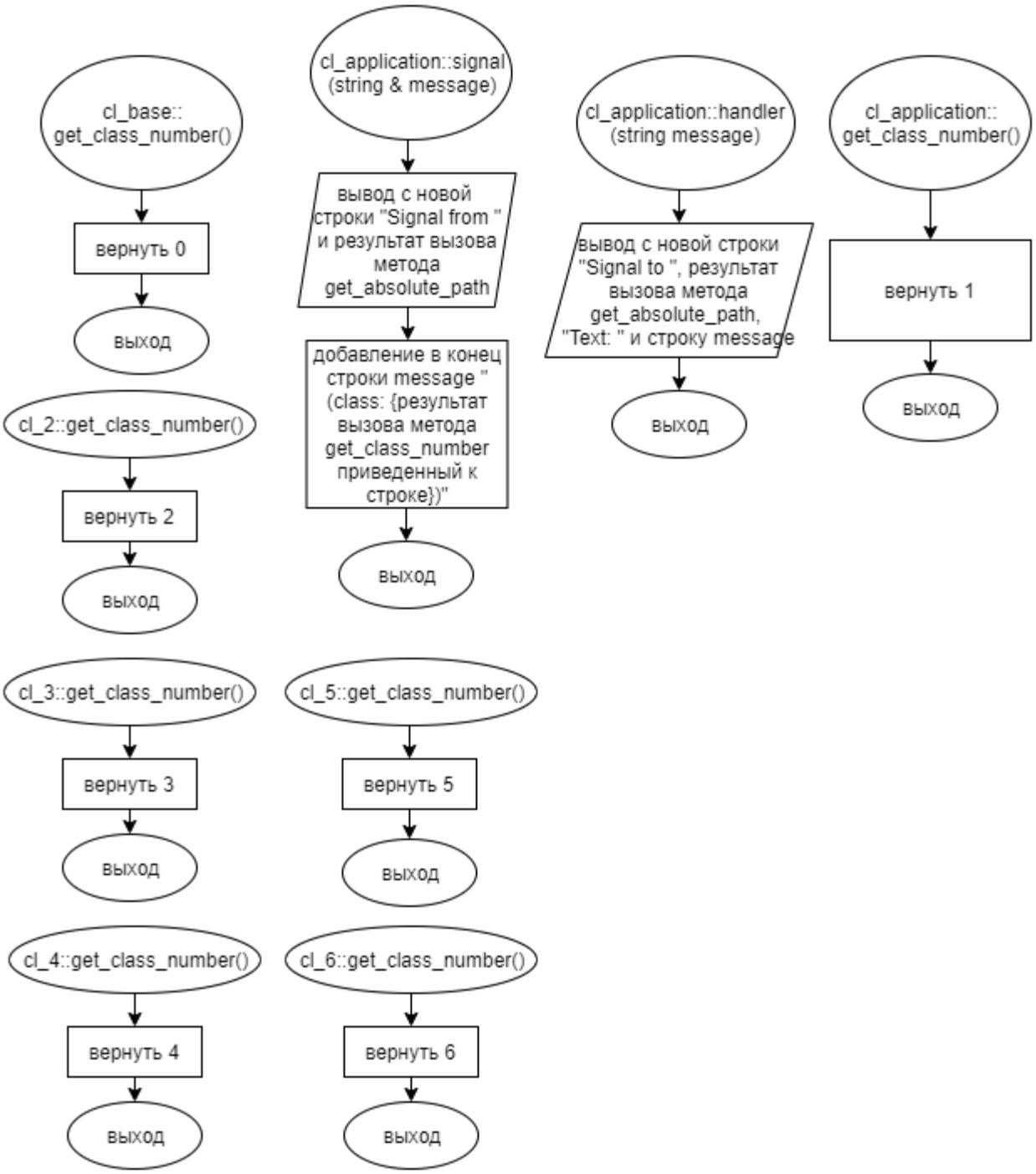

Рисунок 6 – Блок-схема алгоритма

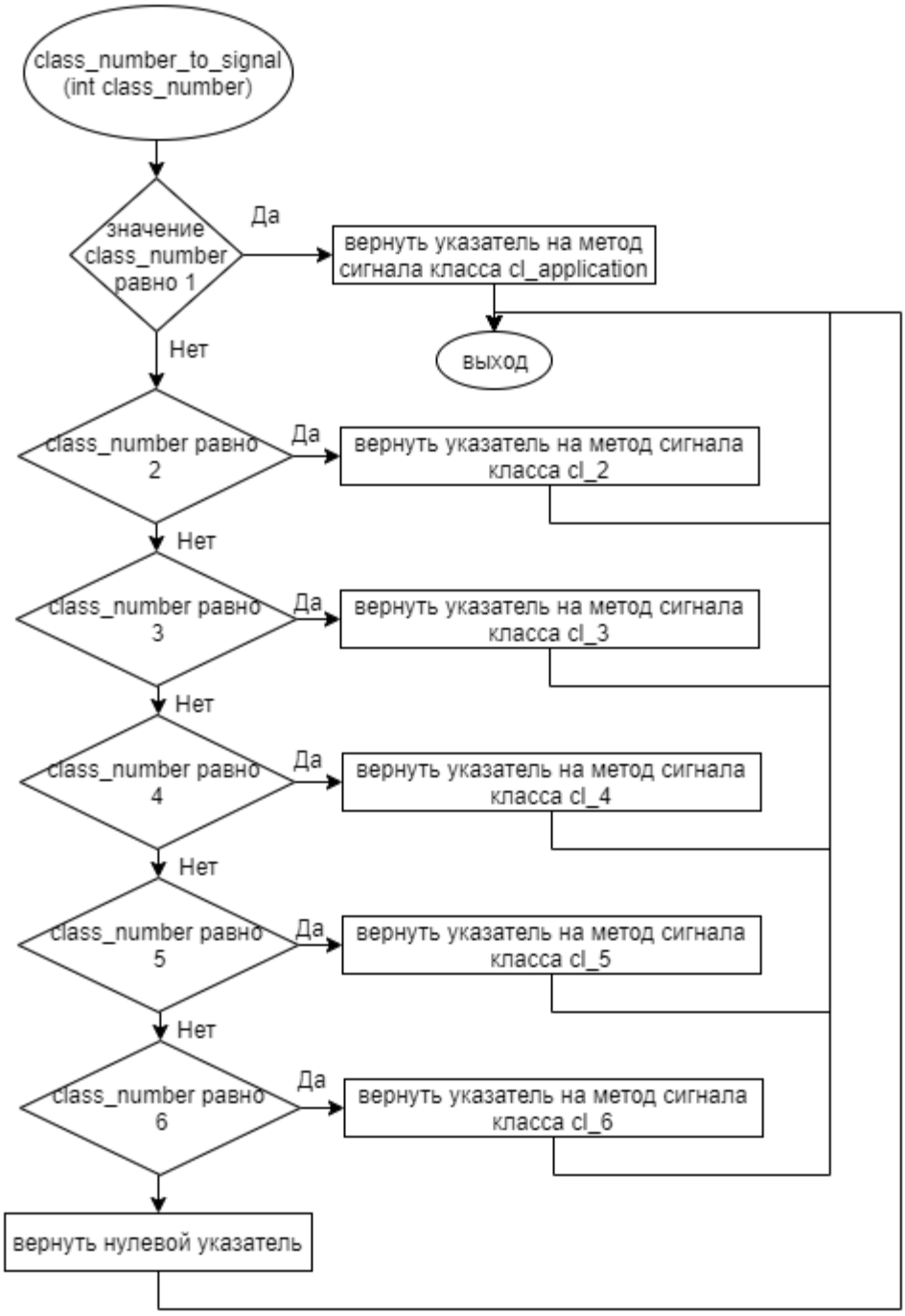

Рисунок 7 – Блок-схема алгоритма

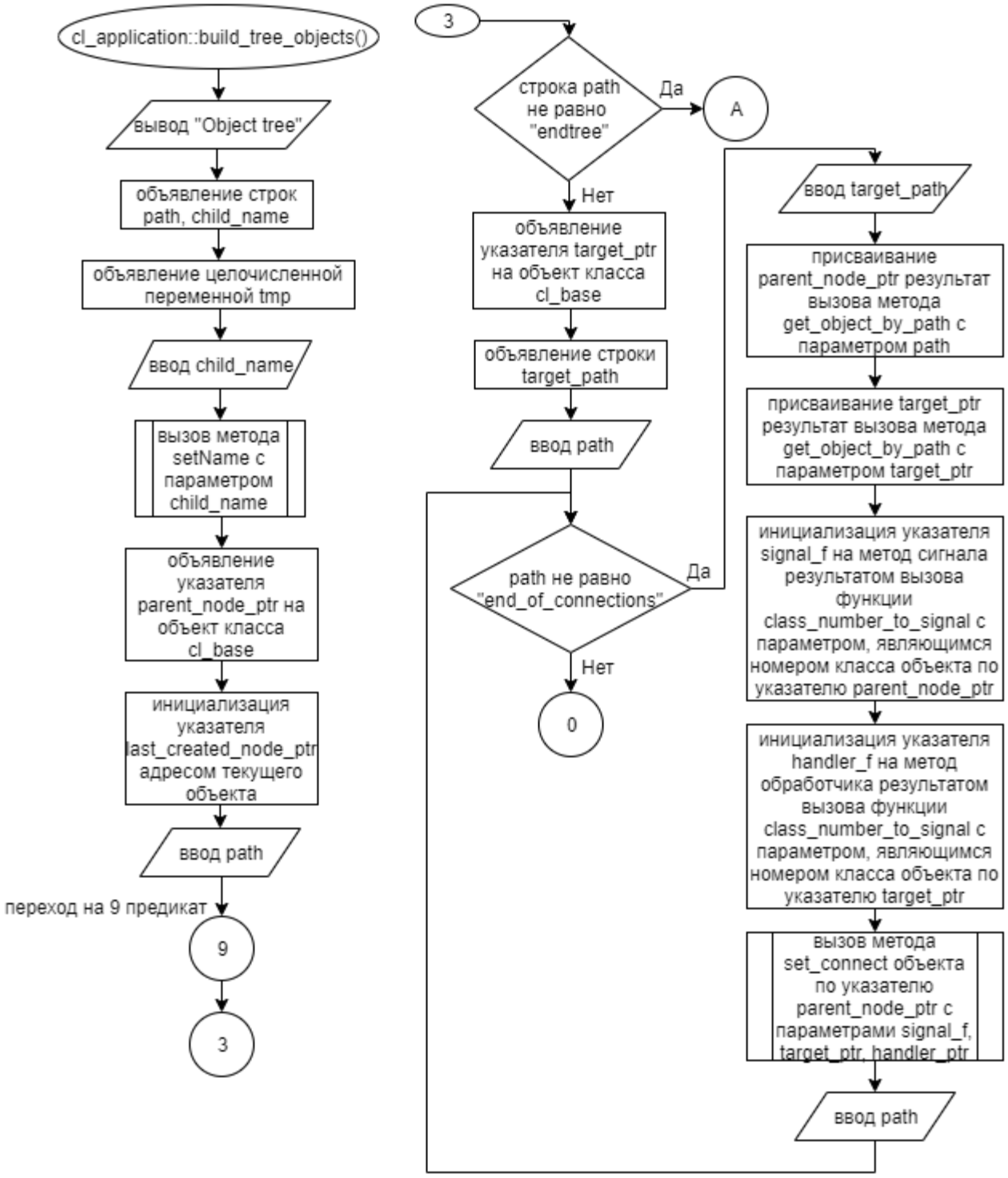

Рисунок 8 – Блок-схема алгоритма

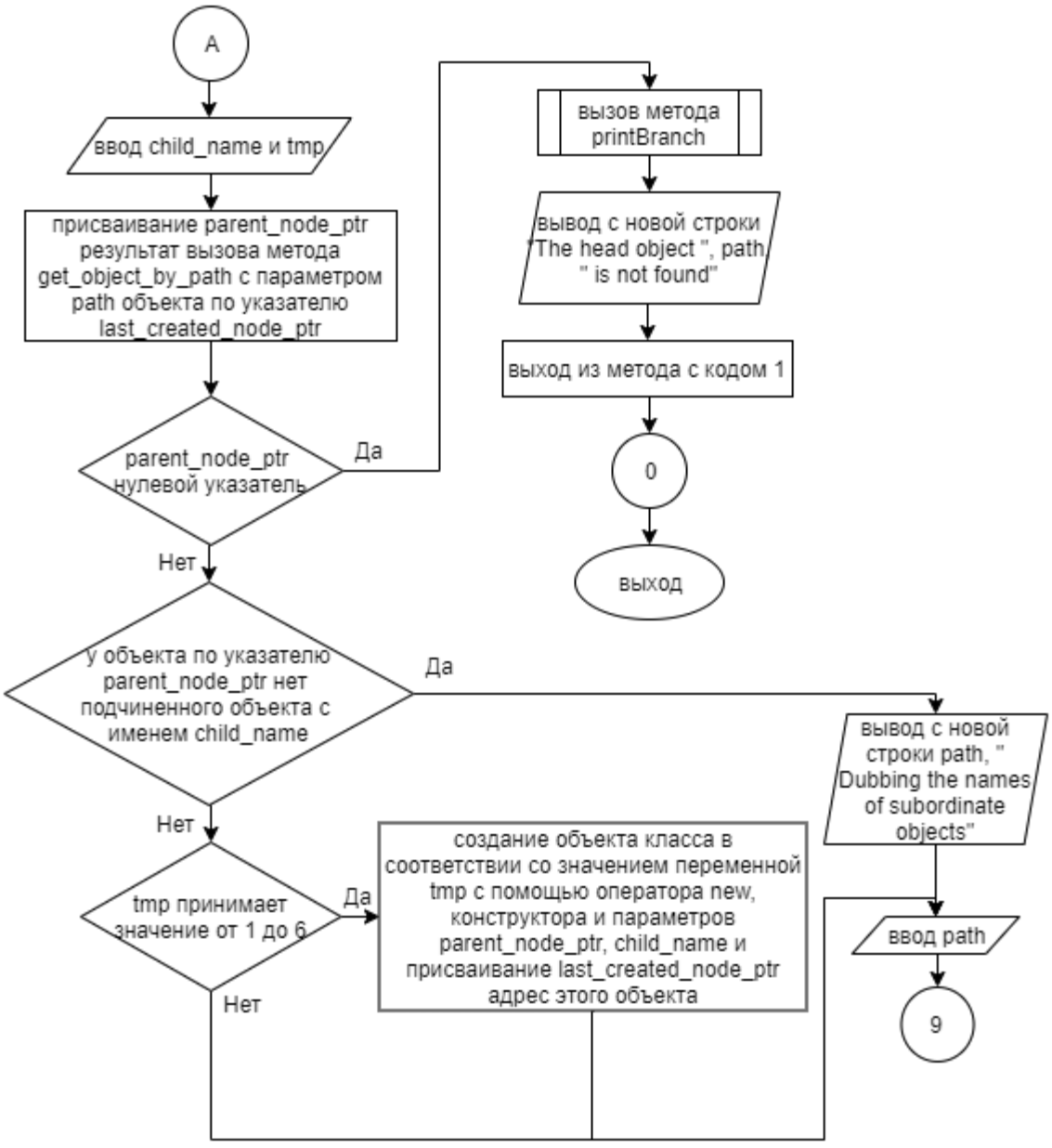

Рисунок 9 – Блок-схема алгоритма

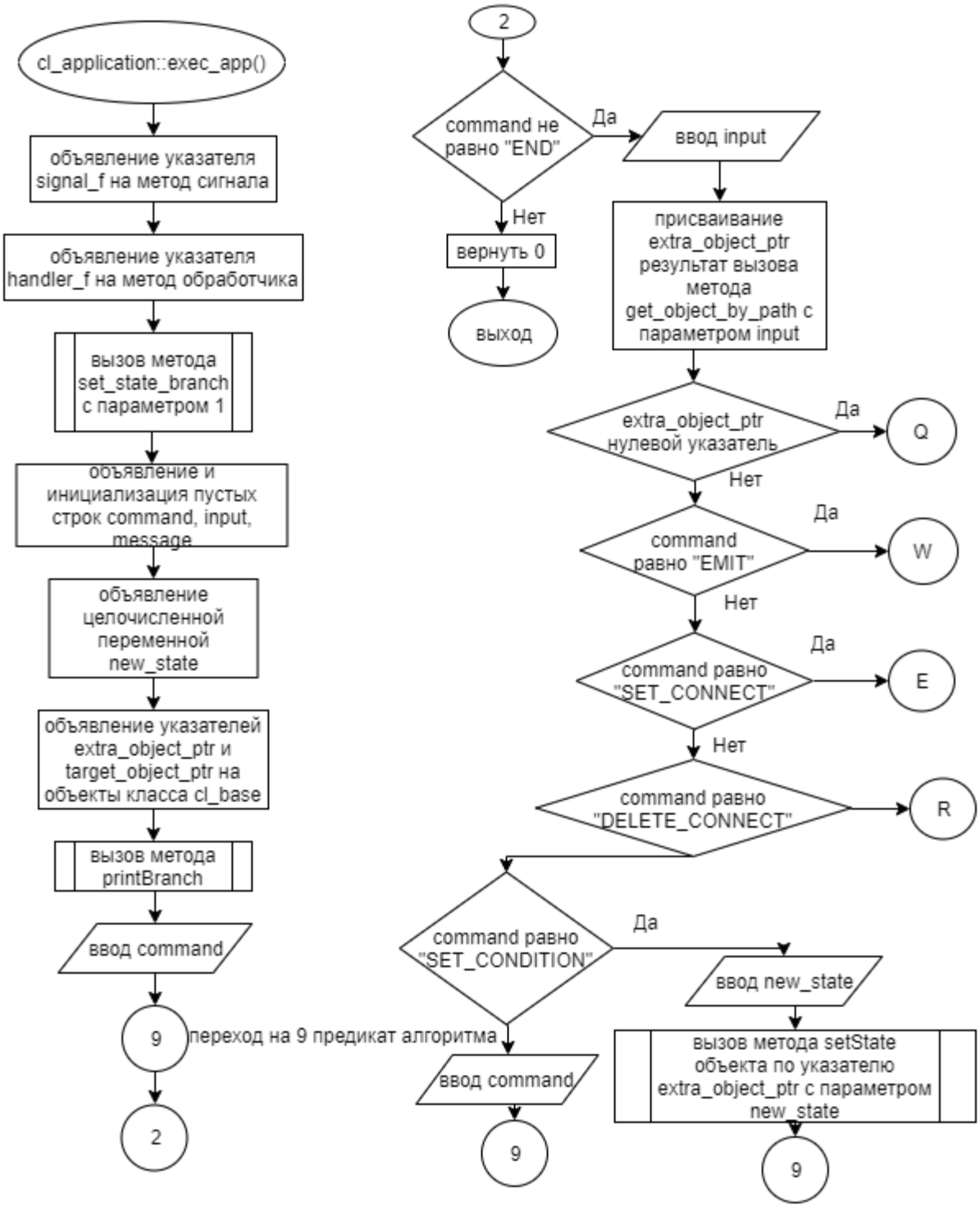

Рисунок 10 – Блок-схема алгоритма

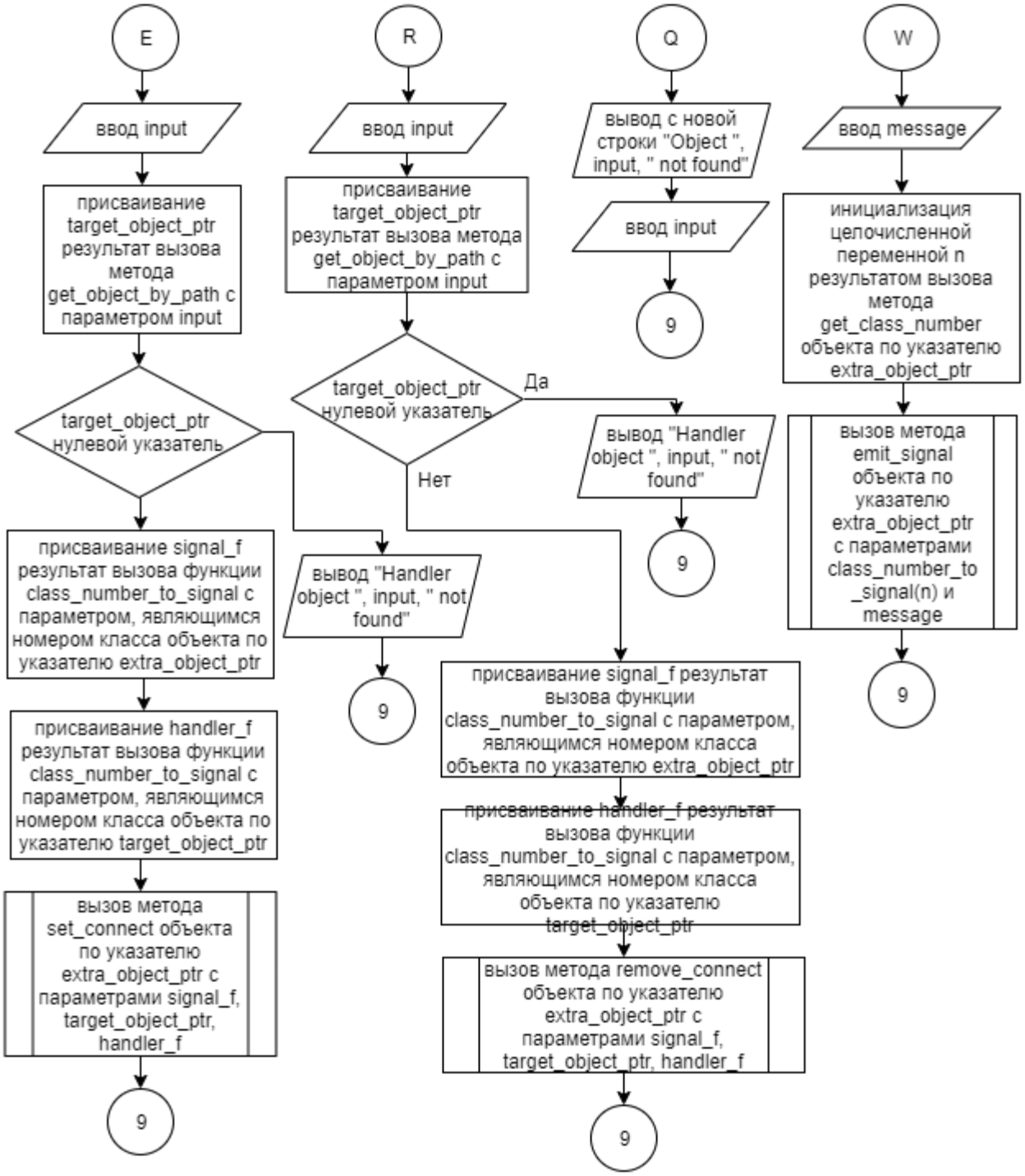

Рисунок 11 – Блок-схема алгоритма

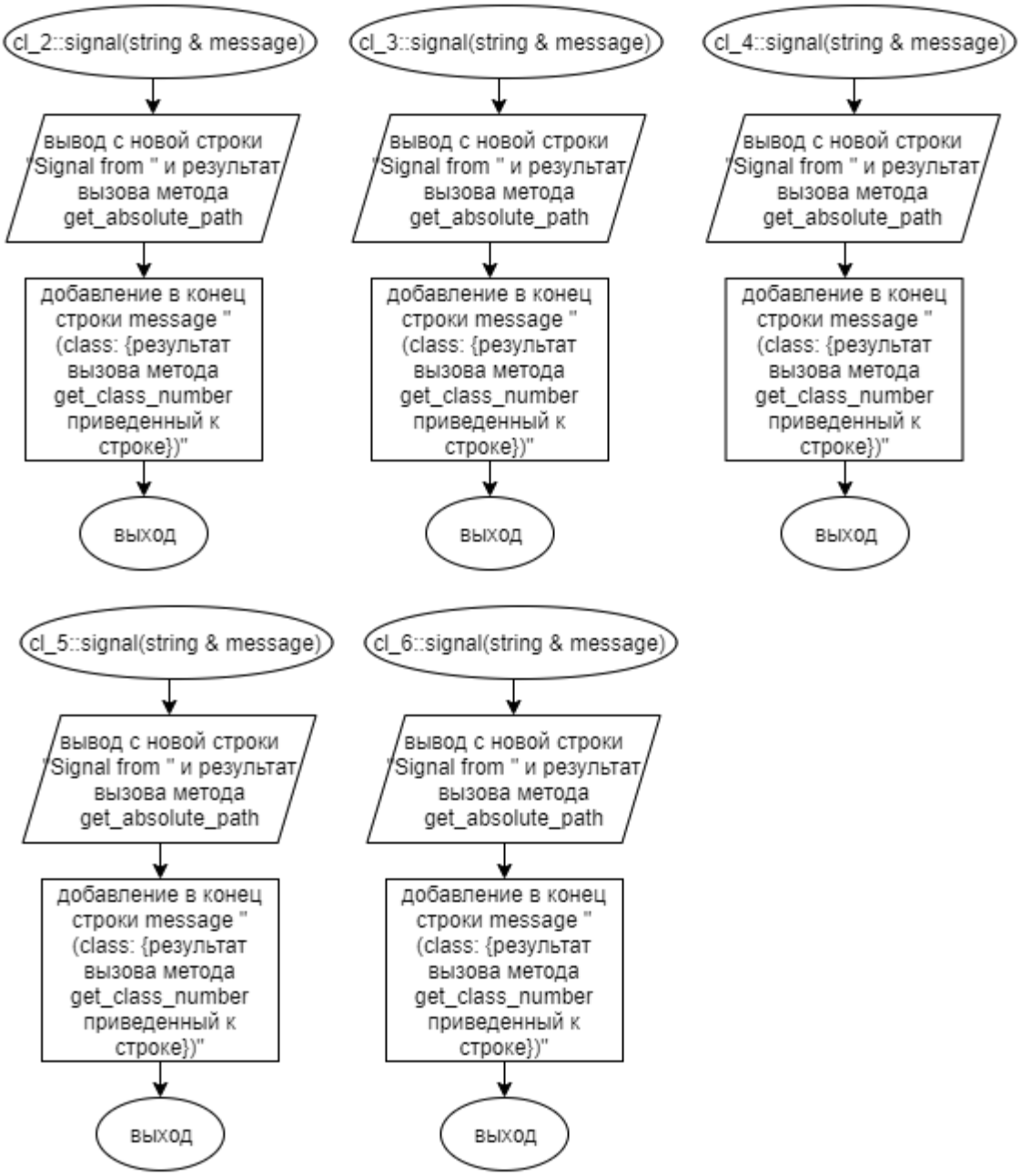

Рисунок 12 – Блок-схема алгоритма

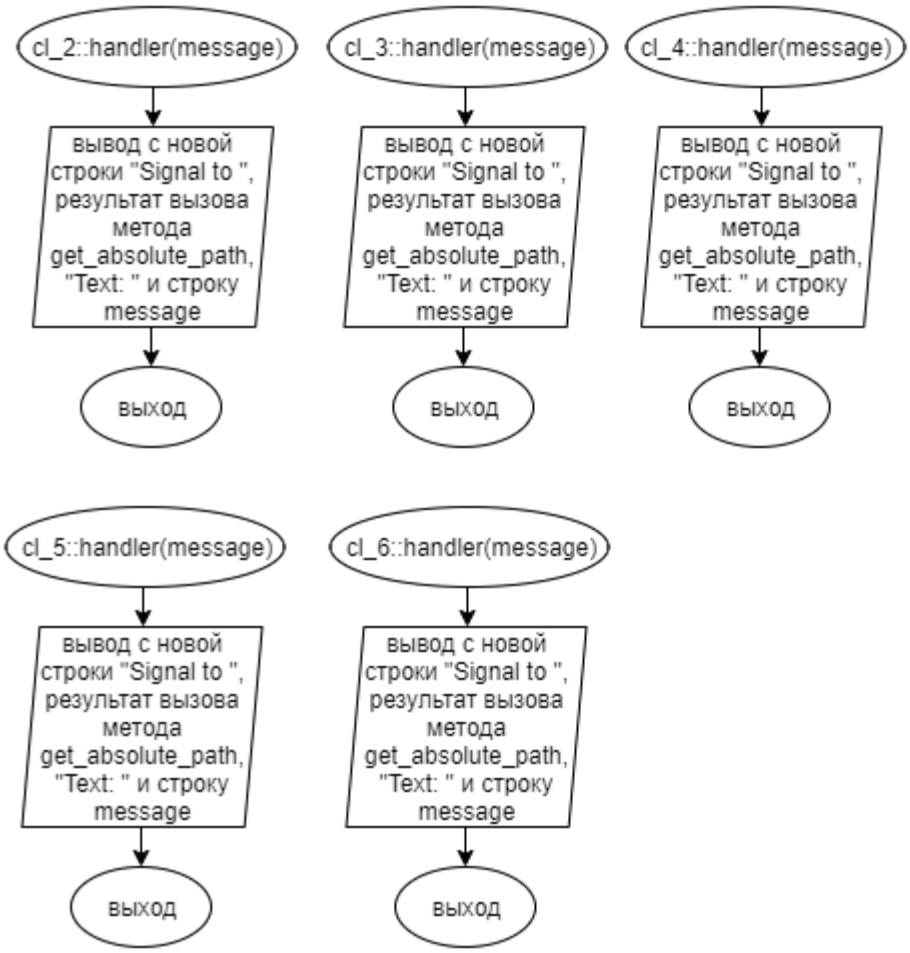

Рисунок 13 – Блок-схема алгоритма

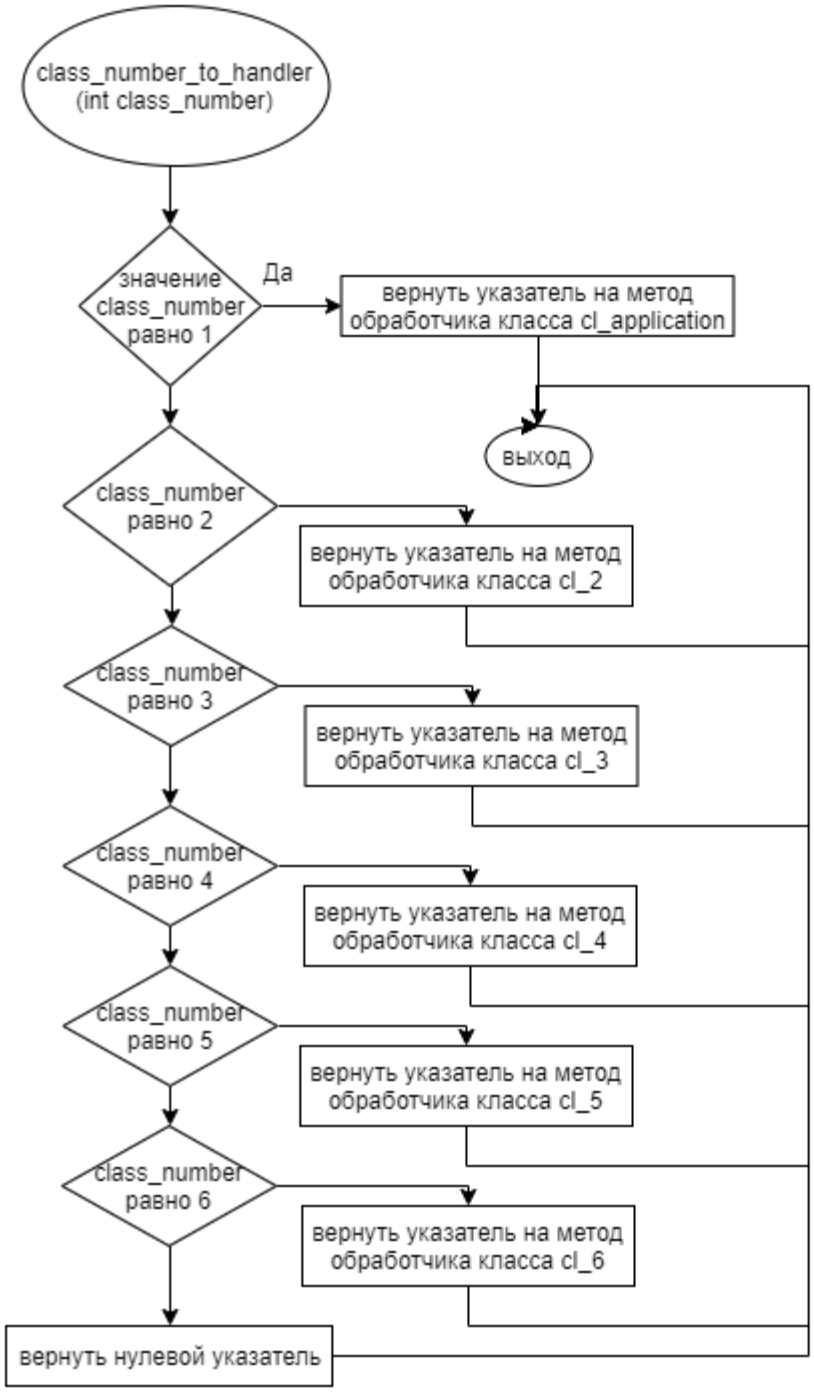

Рисунок 14 – Блок-схема алгоритма

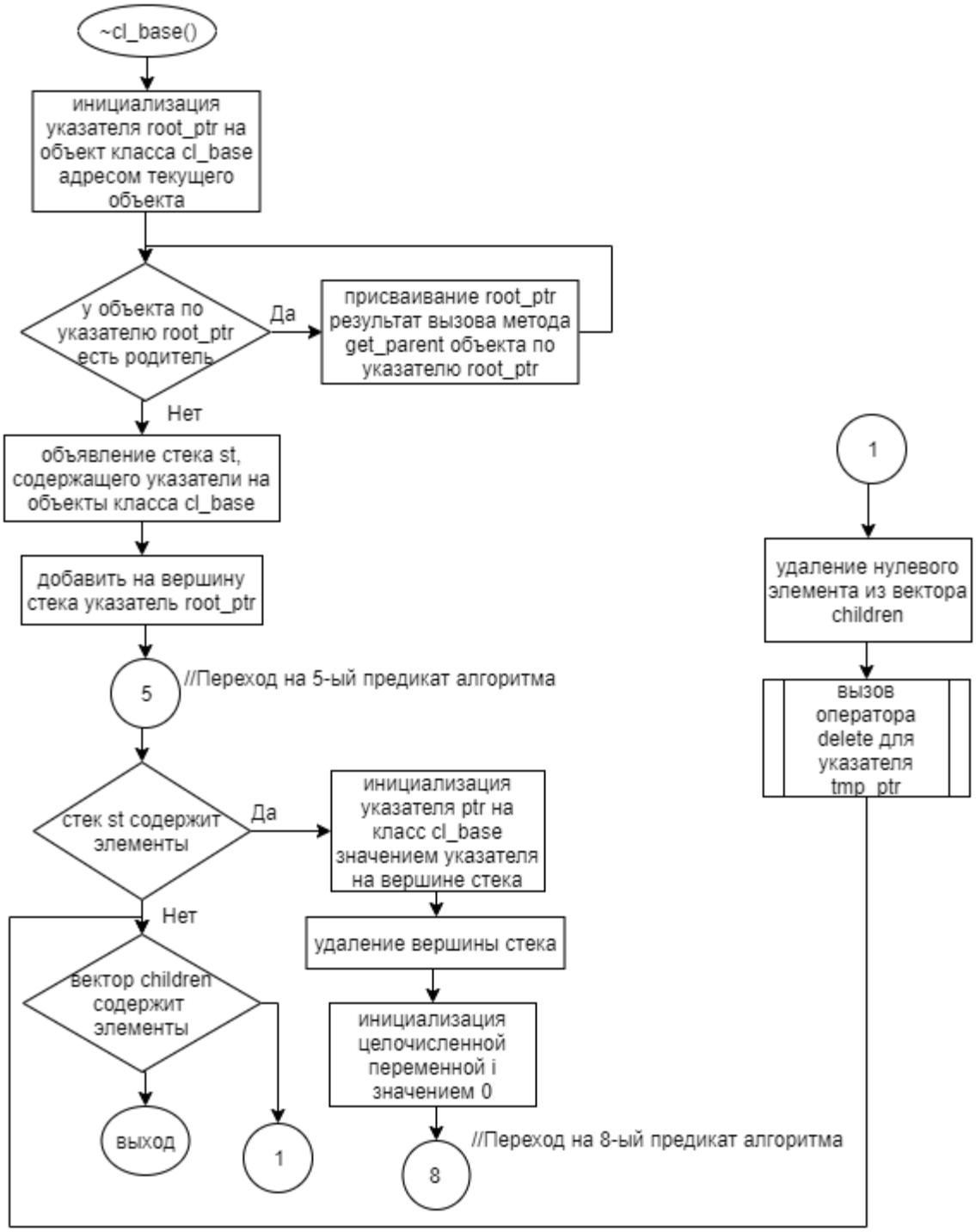

Рисунок 15 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"

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;
}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H

#include "cl_base.h"
class cl_2: public cl_base
{
public:
   cl_2(cl_base* p_head_object, string s_object_name);
   int get_class_number();
```

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

5.3 Файл cl_3.cpp

Листинг 3 – 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;
}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

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

class cl_3: public cl_base
{
public:
   cl_3(cl_base* p_head_object, string s_object_name);
   int get_class_number();
   void signal(string & message);
   void handler(string message);
};
```

```
#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – 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;
}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H

#include "cl_base.h"

class cl_4: public cl_base
{
public:
   cl_4(cl_base* p_head_object, string s_object_name);
   int get_class_number();
   void signal(string & message);
   void handler(string message);
};
```

```
#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – 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;
}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H

#include "cl_base.h"
class cl_5: public cl_base
{
public:
   cl_5(cl_base* p_head_object, string s_object_name);
   int get_class_number();
   void signal(string & message);
   void handler(string message);
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base*                    p_head_object,                    string
s_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;
}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H

#include "cl_base.h"

class cl_6: public cl_base
{
public:
   cl_6(cl_base* p_head_object, string s_object_name);
   int get_class_number();
   void signal(string & message);
   void handler(string message);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
#include <iostream>
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <stack>
using namespace std;
TYPE_SIGNAL class_number_to_signal(int class_number)
{
   switch (class_number)
   {
      case 1:
         return SIGNAL_D(cl_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(cl_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;
}

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

```
int cl_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;
   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 cl_application::get_class_number()
      {
        return 1;
      }

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

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

      void cl_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 cl_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;
         }

      }
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
      #ifndef __CL_APPLICATION__H
      #define __CL_APPLICATION__H
```

```
#include "cl_base.h"

class cl_application : public cl_base
{
public:
   cl_application(cl_base* parent);
   void build_tree_objects();
   int exec_app();
   int get_class_number();
   void signal(string & message);
   void handler(string message);
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
#include <iostream>
#include <stack>
using namespace std;
cl_base::cl_base(cl_base* parent, string name): parent(parent), name(name)
{
   if (parent != nullptr)/*
                        т.к по условию задачи указатель на головной объект
может быть нулевым
                        у корневого объекта
                  */
   {
      parent -> children.push_back(this);//организуем связь вышестоящих
объектов с новым объектом
   }
}

cl_base::~cl_base()
/*
деструктор, отвечает за корректное удаление объекта
и освобождение всех ресурсов, связанных с ним и его
дочерними объектами
*/
{
   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)
              {
                 delete ptr -> connects[i];
                 ptr -> connects.erase(ptr -> connects.begin() + 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;
     }

     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);
      }
   }
}

//

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 задача

void   cl_base::set_connect(TYPE_SIGNAL   signal_ptr,   cl_base*   target_ptr,
TYPE_HANDLER handler_ptr)
/*
устанавливает соединение между сигналом и обработчиком
*/
{
   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)
/*
удаляет соединение между сигналом, целевым объектом
и обработчиком, если такое соединение сушествует
*/
```

```
{
   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)
/*
инициирует сигнал и передачу его всем целевым объектам,
которые должны обработать этот сигнал,если они активны
*/
{
   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()
/*
возвращает абсолютный путь текущего объекта
в дереве
*/
{
   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)
    /*
    устанавливает состояние для текущего объекта
    и его дочерних объектов
    */
    {
        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);

      cl_base* findObjOnBranch(string name);
      cl_base* findObjOnTree(string name);
      void printBranch(int level = 0);
      void printBranchWithState(int level = 0);
      void setState(int state);

      bool set_parent(cl_base* new_parent);
      void remove_child_by_name(string child_name);
      cl_base* get_object_by_path(string path);

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

```
#include "cl_application.h"
int main()
{
   cl_application ob_application(nullptr);
   //создание объекта приложение
   ob_application.build_tree_objects();
   //конструирование системы
   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
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_s2 object_s4 object_s6	Object tree appls_root object_s1 object_s2 object_s4 object_s6

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
6 endtree /object_s2/object_s4 /object_s2/object_s6 end_of_connections SET_CONNECT /object_s2/object_s6 /object_s13 EMIT /object_s2/object_s6 Test message 1 DELETE_CONNECT /object_s2/object_s4 /object_s2/object_s6 EMIT /object_s2/object_s4 Test message 2 END	object_s13 Signal from /object_s2/object_s6 Signal to /object_s13 Text: Test message 1 (class: 6) Signal from /object_s2/object_s4	object_s13 Signal from /object_s2/object_s6 Signal to /object_s13 Text: Test message 1 (class: 6) Signal from /object_s2/object_s4
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 endtree /object_s2/object_s4 /object_s13 /object_s2 /object_s1 end_of_connections SET_CONDITION /object_s2 0 EMIT /object_s2/object_s4 Test message EMIT /object_s1 Test message SET_CONNECT /object_s1 /object_s13 DELETE_CONNECT /object_s1 /object_s2 END	Object tree appls_root object_s1 object_s2 object_s4 object_s13 Signal from /object_s1	Object tree appls_root object_s1 object_s2 object_s4 object_s13 Signal from /object_s1
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5	Object tree appls_root object_s1 object_s2 object_s4 object_s13 Signal from	Object tree appls_root object_s1 object_s2 object_s4 object_s13 Signal from

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
endtree /object_s2/object_s4 /object_s13 /object_s1 /object_s2 end_of_connections SET_CONDITION /object_s2 1 EMIT /object_s2/object_s4 Message 1 SET_CONNECT /object_s1 /object_s13 EMIT /object_s1 Message 2 DELETE_CONNECT /object_s1 /object_s2 EMIT /object_s1 Message 3 END	/object_s2/object_s4 Signal to /object_s13 Text: Message 1 (class: 4) Signal from /object_s1 Signal to /object_s2 Text: Message 2 (class: 3) Signal to /object_s13 Text: Message 2 (class: 3) Signal from /object_s1 Signal to /object_s13 Text: Message 3 (class: 3)	/object_s2/object_s4 Signal to /object_s13 Text: Message 1 (class: 4) Signal from /object_s1 Signal to /object_s2 Text: Message 2 (class: 3) Signal to /object_s13 Text: Message 2 (class: 3) Signal from /object_s1 Signal to /object_s13 Text: Message 3 (class: 3)

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы я значительно углубил свои знания и навыки в области объектно-ориентированного программирования. Я научился определять указатели на объекты по их координатам и использовать их для эффективного управления данными. Работа с инструментами разработки, такими как АРМ Аврора, сделала процесс создания и тестирования программ более удобным и структурированным.

Разработка алгоритмов и блок-схем способствовала более глубокому пониманию принципов ООП. Создание отчетов и документации облегчило анализ и сопровождение кода. В результате выполнения данной курсовой работы я закрепил теоретические знания, полученные в ходе изучения курса, и убедился в значимости ООП для разработки сложных и надежных программных систем.

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).