Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

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].

Актуальность: реализация механизма взаимодействия объектов с использованием сигналов и обработчиков, передача текстовых сообщений вместе с сигналами - актуальная тема в настоящее время. Это позволяет гибко коммуницировать между компонентами системы и облегчает её расширение и сопровождение. Разработка такой системы с методами установки связи, удаления связи и выдачи сигнала с текстовыми переменными повышает эффективность и модульность кода.

Цель работы: получение практических навыков по разработке системы, состоящей из нескольких объектов, взаимодействующих между собой.

Задачи:

1. Использование на практике уже изученных основных принципов объектно-ориентированного программирования на C++
2. Разработка архитектуры системы
3. Реализация необходимых классов и методов для работы системы
4. Реализация необходимых сигналов и обработчиков
5. Тестирование работы системы на разных входных даннных
6. Написание метода решения, алгоритма и блок-схемы функционирования системы

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 то же, что и в работе КВ_3, однако были изменены/добавлены несколько методов и классов.

В классе cl_base были использованы параметризированные макроопределеня препроцессора для приведения указателя на метод сигнала и на метод обработчика, а также были объявлены новые типы данных TYPE_SIGNAL и TYPE_HANDLER для определения указателей на методы сигнала и обработчика.

Также была добавлена дополнительная структура o_sh, представляющая собой пользовательский тип данных для хранения соединений "сигнал-обработчик".

- Класс cl_application:
  - Свойства/поля:
    - Те же поля, что и в КВ_3 у класса cl_application;
  - Методы
    - Те же методы, что и в КВ_3 у класса cl_application;
    - Метод GetSignal
      - Функционал - используется для получения указателя на метод сигнала определенного класса;
    - Метод GetHandler
      - Функционал - используется для получения указателя на метод обработчика определенного класса;
- Класс cl_base:
  - Свойства/поля:
    - Те же поля, что и в КВ_3 у класса cl_base;
    - Поле, отвечающее за хранение соединений "сигнал-обработчик" для определенного объекта

        - Наименование - connects;
        - Тип - вектор объектов структуры o_sh;
        - Модификатор доступа - private;
    - Поле, отвечающее за хранение номера класса
        - Наименование - classNum;
        - Тип - int;
        - Модификатор доступа - private;
- Методы
    - Те же методы, что и в КВ_3 у класса cl_base;
    - Метод SetChildState
        - Функционал - используется для смены состояний всех подчиненных объектов текущего на значение параметра;
    - Метод SetConnect
        - Функционал - используется для установления связи между сигналом текущего объекта и обработчиком целевого объекта;
    - Метод DeleteConnect
        - Функционал - используется для удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
    - Метод EmitSignal
        - Функционал - используется для выдачи сигнала от текущего объекта с передачей строковой переменной;
    - Метод GetFullPath
        - Функционал - используется для возращения полного пути текущего объекта иерархии;
    - Метод SetClassNum

        - Функционал - используется для установки номера класса текущего объекта;
      - Метод GetClassNum
        - Функционал - используется для получения номера класса текущего объекта;
- Класс cl_1:
  - Свойства/поля:
    - Те же поля, что и в KB_3 у класса cl_1;
  - Методы
    - Метод Signal
      - Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      - Функционал - используется как обработчик текущего объекта;
- Класс cl_2:
  - Свойства/поля:
    - Те же поля, что и в KB_3 у класса cl_2;
  - Методы
    - Метод Signal
      - Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      - Функционал - используется как обработчик текущего объекта;
- Класс cl_3:
  - Свойства/поля:

        - Те же поля, что и в КВ_3 у класса cl_3;
    - Методы
        - Метод Signal
            - Функционал - используется как сигнал текущего объекта;
        - Метод Handler
            - Функционал - используется как обработчик текущего объекта;
- Класс cl_4:
    - Свойства/поля:
        - Те же поля, что и в КВ_3 у класса cl_4;
    - Методы
        - Метод Signal
            - Функционал - используется как сигнал текущего объекта;
        - Метод Handler
            - Функционал - используется как обработчик текущего объекта;
- Класс cl_5:
    - Свойства/поля:
        - Те же поля, что и в КВ_3 у класса cl_5;
    - Методы
        - Метод Signal
            - Функционал - используется как сигнал текущего объекта;
        - Метод Handler
            - Функционал - используется как обработчик текущего

объекта;

- Класс cl_6:
  - Свойства/поля:
    - Те же поля, что и в KB_3 у класса cl_6;
  - Методы
    - Метод Signal
      - Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      - Функционал - используется как обработчик текущего объекта;

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: Основной алгоритм работы программы.

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения программы.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Выполнения алгоритма данной функции из предыдущей работы КВ_3	∅

3.2 Алгоритм метода build_tree_objects класса cl_application

Функционал: Построение исходного дерева иерархии объектов.

Параметры: Отсутствуют.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление строчных переменных objPath и childName	2

№	Предикат	Действия	№ перехода
2		Объявление целочисленной переменной state	3
3		Инициализация целочисленной переменной classNum значением 1	4
4		Ввод значения переменной objPath	5
5		Вызов метода SetName, с аргументом значения переменной objPath	6
6		Вызов метода SetClassNum, с аргументом значения переменной childName	7
7		Очистка потока ввода	8
8		Ввод значения переменной objPath	9
9	Значение objPath равно "endtree"		18
		Инициализация указателя parentObj на объект класса cl_base результатом выполнения метода GetObjectByPath, с аргументом в виде значения objPath	10
10	Значение parentObj равно nullptr	Вывод "Object tree" на экран	11
		Ввод значений переменных childName и classNum	14
11		Вызов метода PrintObjects	12
12		Вывод "\nThe head object ", objPath и " is not found" на экран	13
13		Завершение работы программы с значением 1	∅
14		Вызов метода GetChild, с аргументом childName, через указатель на объект parentObj	15
15	Возвращаемое значение метода GetChild равно nullptr	Инициализация указателя childObj на объект класса cl_base значением nullptr	16
		Вывод objPath, " Dubbing the names of subordinate objects" на экран	7

№	Предикат	Действия	№ перехода
16	Значение classNum равно i	Присваивание childObj значение создаваемого объекта cl_i-того класса, с аргументами parentObj и childName	17
			17
17	Значение childObj не равно nullptr	Вызов метода SetClassNum, с аргументом classNum, через указатель на объект childObj	7
			7
18	Значение objPath не равно "end_of_connections"	Ввод значения переменной objPath	19
			∅
19	Значение objPath не равно "end_of_connections"	Инициализация указателя headPtr на объект класса cl_base результатом выполнения метода GetObjectByPath, с аргументом в виде значения objPath	20
			18
20		Ввод значения переменной objPath	21
21		Инициализация указателя childPtr на объект класса cl_base результатом выполнения метода GetObjectByPath, с аргументом в виде значения objPath	22
22		Вызов метода SetConnect указателя headPtr, с аргументами в виде результатов выполнения методов GetSignal, с аргументом headPtr и GetHandler, с аргументом childPtr и указателя childPtr	18

3.3 Алгоритм метода exec_app класса cl_application

Функционал: Запуск приложения (начало функционирования системы).

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения метода.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree" на экран	2
2		Вызов метода PrintObjects текущего объекта	3
3		Вызов метода SetChildState, с аргуметом 1, текущего объекта	4
4		Объявление указателей obj и temp на объект класса cl_base	5
5		Объявление строчных переменных command, message и objPath	6
6		Ввод значения переменной command	7
7	Значение command равно "END"	Возвращение значения 0	∅
		Ввод значения переменной objPath	8
8		Присваивание указателю obj значение выполнения метода GetObjectByPath, с аргументом в виде значения objPath	9
9	Значение obj равно nullptr	Вывод "Object ", objPath, " not found" на экран	6
		Считывание всей строки ввода для записии её в переменную message	10
10	Значение command равно "EMIT"		11
	Значение command равно "SET_CONDITION"	Вызов метода SetObjectState через указатель на объект obj, с аргументом в виде целой части строковой переменной message	6
			13

№	Предикат	Действия	№ перехода
11		Вызов метода GetObjectState через указатель на объект obj	12
12	Возвращаемое значение метода GetObjectState равно true	Вызов метода EmitSignal указателя obj, с аргументами message и результатом выполнения метода GetSignal, с аргументом obj	6
			6
13		Присваивание message значение message без первого символа	14
14		Присваивание указателю temp значение метода GetObjectByPath, с аргументом в виде значения message	15
15	Значение temp равно nullptr	Вывод "Handler object ", message, " not found" на экран	6
	Значение command равно "SET_CONNECT"	Вызов метода SetConnect через указатель на объект obj, с аргументами в ввиде результатов выполнения методов GetSignal, с аргументом obj, и GetHandler, с аргументом temp, и указателя temp	6
	Значение command равно "DELETE_CONNECT"	Вызов метода DeleteConnect через указатель на объект obj, с аргументами в ввиде результатов выполнения методов GetSignal, с аргументом obj, и GetHandler, с аргументом temp, и указателя temp	6
			6

3.4 Алгоритм метода GetSignal класса cl_application

Функционал: Возврат указателя на метод сигнала одного из подчиненных классов класса cl_base.

Параметры: cl_base* object - указатель на искомый объект класса cl_base.

Возвращаемое значение: TYPE_SIGNAL - указатель на метод сигнала

одного из подчиненных классов класса cl_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода GetSignal класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода GetClassNum через указатель на объект object	2
2	Возвращаемое значение метода GetClassNum равно i-тому значению	Возврат указателя на метод Signal cl_i-того класса	∅
		Возврат nullptr	∅

3.5 Алгоритм метода GetHandler класса cl_application

Функционал: Возврат указателя на метод обработчика одного из подчиненных классов класса cl_base.

Параметры: cl_base* object - указатель на искомый объект класса cl_base.

Возвращаемое значение: TYPE_HANDLER - указатель на метод обработчика одного из подчиненных классов класса cl_base.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода GetHandler класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода GetClassNum через указатель на объект object	2
2	Возвращаемое значение метода GetClassNum равно i-тому значению	Возврат указателя на метод Handler cl_i-того класса	∅
		Возврат nullptr	∅

3.6 Алгоритм метода SetChildState класса cl_base

Функционал: Смена состояний всех подчиненных объектов текущего на значение параметра.

Параметры: int state - состояние для всех подчиненных объектов текущего.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода SetChildState класса cl_base

№	Предикат	Действия	№ перехода
1		Вызов метода SetObjectState, с аргументом state	2
2	Переменная child списка children не равна nullptr	Вызов метода SetChildState, с аргументом state	∅
			∅

3.7 Алгоритм метода SetConnect класса cl_base

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

Параметры: TYPE_SIGNAL signal - указатель на метод сигнала текущего объекта, cl_base* object - указатель на целевой объект, TYPE_HANDLER handler - указатель на метод обработчика целевого объекта.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода SetConnect класса cl_base

№	Предикат	Действия	№ перехода
1	Ссылка на переменную pos списка connects не равна nullptr		2

№	Предикат	Действия	№ перехода
			3
2	Поля Signal, Handler, Object переменной pos равны параметрам signal, object, handler		∅
			1
3		Объявление объекта obj структуры o_sh	4
4		Присваивание полю Signal объекта obj значение параметра signal	5
5		Присваивание полю Handler объекта obj значение параметра handler	6
6		Присваивание полю Object объекта obj значение параметра object	7
7		Добавление в конец списка connects объект obj	∅

3.8 Алгоритм метода DeleteConnect класса cl_base

Функционал: Удаление (разрыв) связи между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: TYPE_SIGNAL signal - указатель на метод сигнала текущего объекта, cl_base* object - указатель на целевой объект, TYPE_HANDLER handler - указатель на метод обработчика целевого объекта.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода DeleteConnect класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация итератора i значением указателя	2

№	Предикат	Действия	№ перехода
		на начальный элемент	
2	i не равен значению указателя на элемент после последнего		3
			∅
3	Поля Signal, Handler, Object значения i равны параметрам signal, object, handler	Удаление элемента, на который указывает i, из connects	∅
			2

3.9 Алгоритм метода EmitSignal класса cl_base

Функционал: Выдача сигнала от текущего объекта.

Параметры: TYPE_SIGNAL signal - указатель на метод сигнала текущего объекта, string& - сообщение.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода EmitSignal класса cl_base

№	Предикат	Действия	№ перехода
1		Вызов метода сигнала, указатель на который хранит signal, с аргументом message	2
2	Ссылка на переменную pos списка connects не равна nullptr		3
			∅
3	Поле Signal переменной pos равно signal	Инициализация указателя handler на объект типа TYPE_HANDLER значением поля Handler переменной pos	4

№	Предикат	Действия	№ перехода
			2
4		Инициализация указателя obj на объект класса cl_base значением поля Object переменной pos	5
5		Вызов метода GetObjectState через указатель на объект obj	6
6	Возвращаемое значение метода GetObjectState равно true	Вызов метода обработчика, указатель на который хранит handler, с аргументом message	2
			2

3.10 Алгоритм метода GetFullPath класса cl_base

Функционал: Возращение полного пути текущего объекта иерархии.

Параметры: Отсутствуют.

Возвращаемое значение: string - полный путь текущего объекта.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода GetFullPath класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной path	2
2		Инициализация указателя temp на объект класса cl_base значением указателя на текущий объект	3
3		Вызов метода GetParent через указатель на объект temp	4
4	Возвращаемое значение метода GetParent равно nullptr	Возврат "/"	∅
			5
5		Вызов метода GetParent через указатель на объект	6

№	Предикат	Действия	№ перехода
		temp	
6	Возвращаемое значение метода GetParen не равно nullptr	Присваивание path "/", наименование объекта указателя temp и значение path	7
		Возврат path	∅
7		Присваивание temp результата выполнения метода GetParent, вызванного через указатель temp	6

3.11 Алгоритм метода SetClassNum класса cl_base

Функционал: Установка номера класса текущего объекта.

Параметры: int num - значение номера класса текущего объекта.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода SetClassNum класса cl_base

№	Предикат	Действия	№ перехода
1		Присваивание полю classNum текущего объекта значение параметра num	∅

3.12 Алгоритм метода GetClassNum класса cl_base

Функционал: Получение номера класса текущего объекта.

Параметры: Отсутствуют.

Возвращаемое значение: int - значение номера класса текущего объекта.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода GetClassNum класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля classNum текущего объекта	∅

3.13 Алгоритм метода Signal класса cl_1

Функционал: Метод сигнала текущего объекта.

Параметры: string& data - текст сообщения для метода обработчика.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода Signal класса cl_1

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " и результата выполнения метода GetFullPath	2
2		Присваивание параметру data текущее значение и " (class: 1)"	∅

3.14 Алгоритм метода Handler класса cl_1

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода Handler класса cl_1

№	Предикат	Действия	№ перехода
1		Вывод "Signal to ", результата выполнения метода GetFullPath и " Text: " с значением data	∅

3.15 Алгоритм метода Signal класса cl_2

Функционал: Метод сигнала текущего объекта.

Параметры: string& data - текст сообщения для метода обработчика.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода Signal класса cl_2

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " и результата выполнения метода GetFullPath	2
2		Присваивание параметру data текущее значение и " (class: 2)"	∅

3.16 Алгоритм метода Handler класса cl_2

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода Handler класса cl_2

№	Предикат	Действия	№ перехода
1		Вывод "Signal to ", результата выполнения метода GetFullPath и " Text: " с значением data	∅

3.17 Алгоритм метода Signal класса cl_3

Функционал: Метод сигнала текущего объекта.

Параметры: string& data - текст сообщения для метода обработчика.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода Signal класса cl_3

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " и результата выполнения метода GetFullPath	2
2		Присваивание параметру data текущее значение и " (class: 3)"	∅

3.18 Алгоритм метода Handler класса cl_3

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода Handler класса cl_3

№	Предикат	Действия	№ перехода
1		Вывод "Signal to ", результата выполнения метода GetFullPath и " Text: " с значением data	∅

3.19 Алгоритм метода Signal класса cl_4

Функционал: Метод сигнала текущего объекта.

Параметры: string& data - текст сообщения для метода обработчика.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода Signal класса cl_4

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " и результата выполнения метода GetFullPath	2
2		Присваивание параметру data текущее значение и " (class: 4)"	∅

3.20 Алгоритм метода Handler класса cl_4

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода Handler класса cl_4

№	Предикат	Действия	№ перехода
1		Вывод "Signal to ", результата выполнения метода GetFullPath и " Text: " с значением data	∅

3.21 Алгоритм метода Signal класса cl_5

Функционал: Метод сигнала текущего объекта.

Параметры: string& data - текст сообщения для метода обработчика.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода Signal класса cl_5

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " и результата выполнения метода GetFullPath	2
2		Присваивание параметру data текущее значение и " (class: 5)"	∅

3.22 Алгоритм метода Handler класса cl_5

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода Handler класса cl_5

№	Предикат	Действия	№ перехода
1		Вывод "Signal to ", результата выполнения метода GetFullPath и " Text: " с значением data	∅

3.23 Алгоритм метода Signal класса cl_6

Функционал: Метод сигнала текущего объекта.

Параметры: string& data - текст сообщения для метода обработчика.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода Signal класса cl_6

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " и результата выполнения метода GetFullPath	2
2		Присваивание параметру data текущее значение и " (class: 6)"	∅

3.24 Алгоритм метода Handler класса cl_6

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода Handler класса cl_6

№	Предикат	Действия	№ перехода
1		Вывод "Signal to ", результата выполнения метода GetFullPath и " Text: " с значением data	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-13.

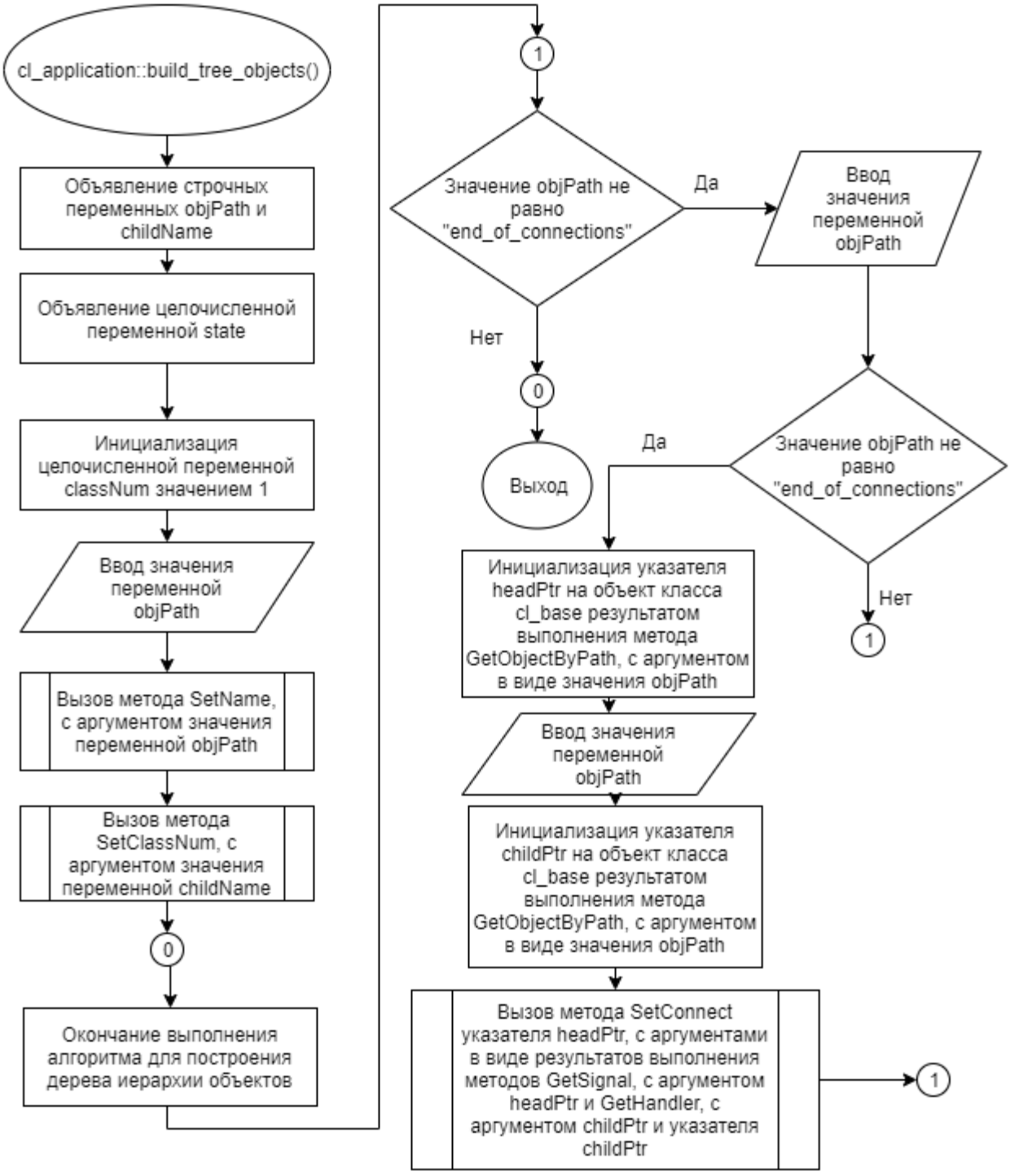

Рисунок 1 – Блок-схема алгоритма

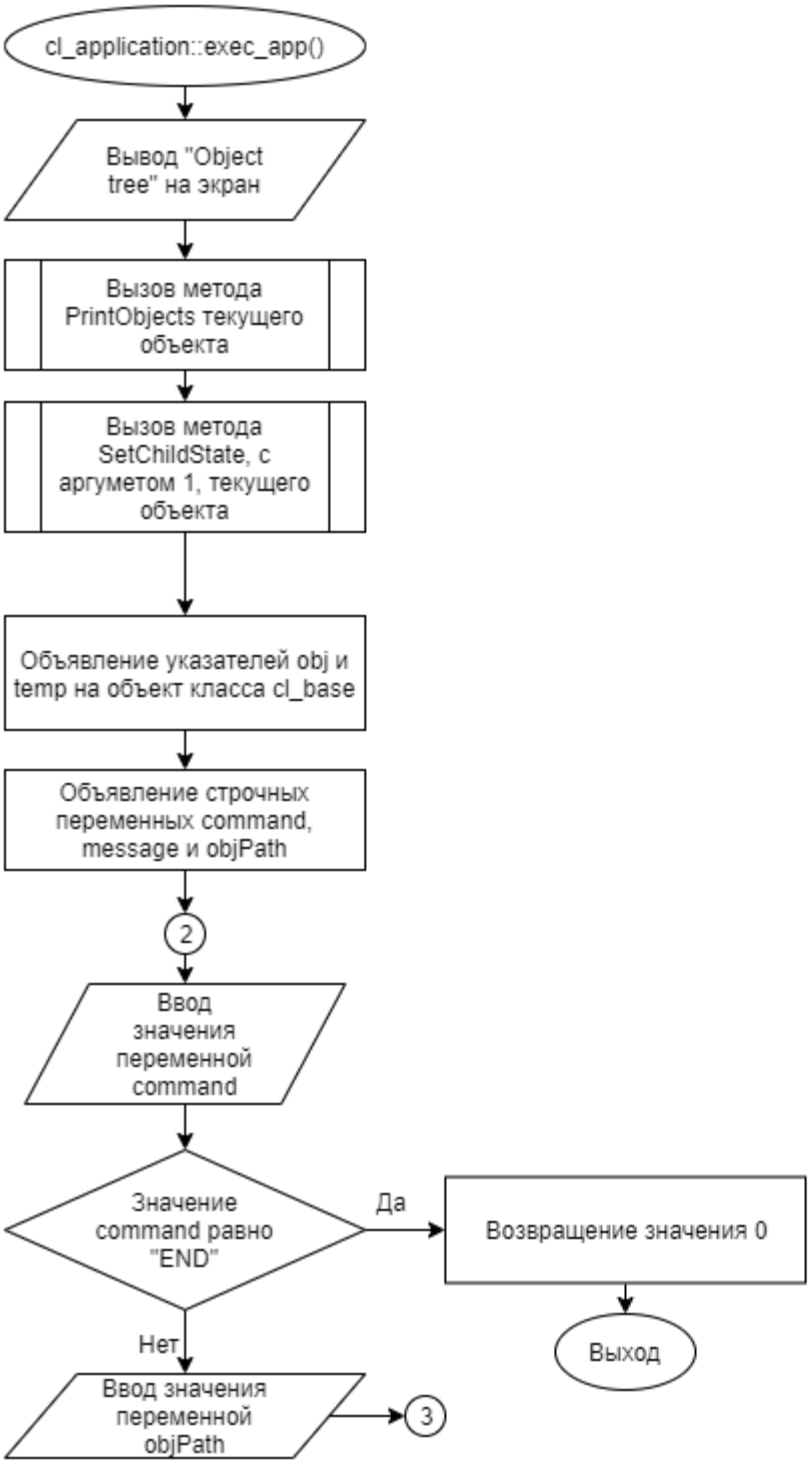

Рисунок 2 – Блок-схема алгоритма

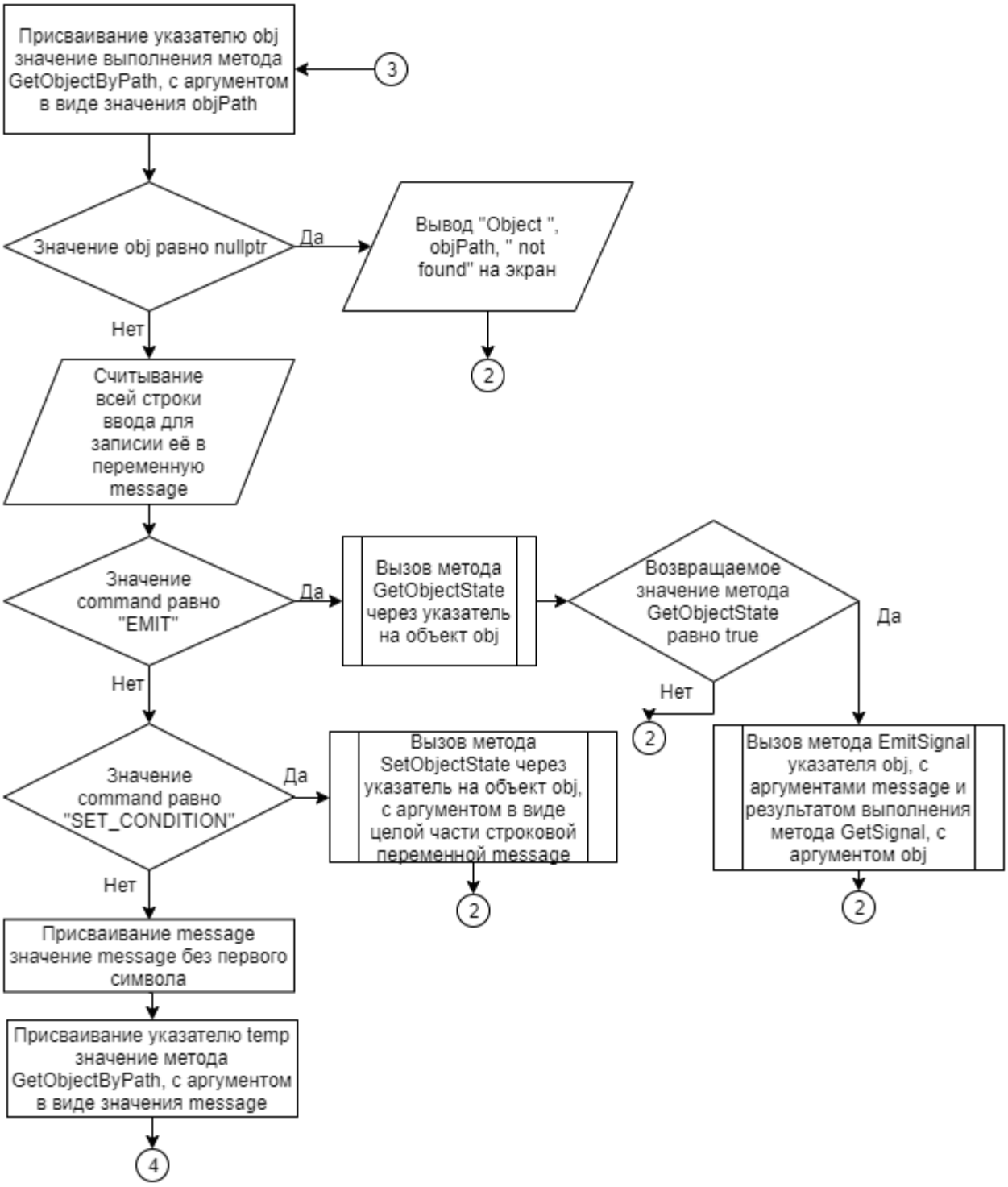

Рисунок 3 – Блок-схема алгоритма

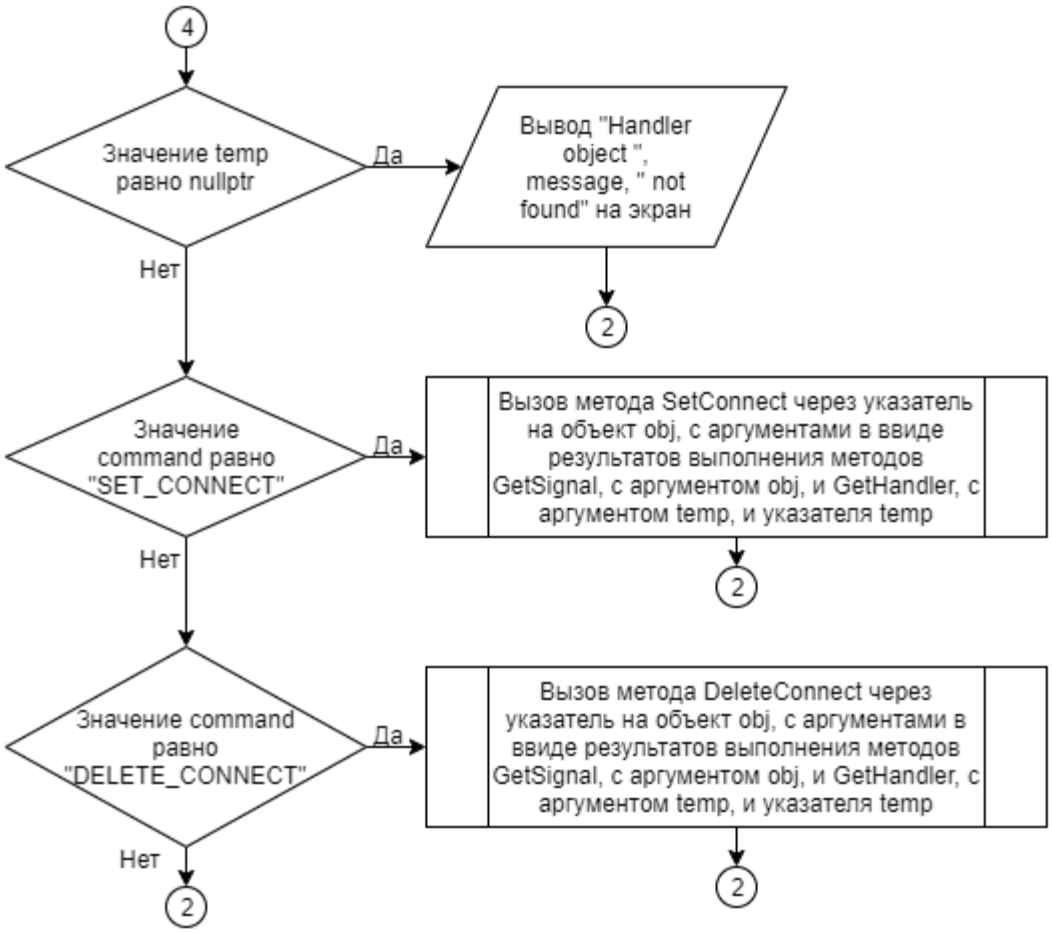

Рисунок 4 – Блок-схема алгоритма

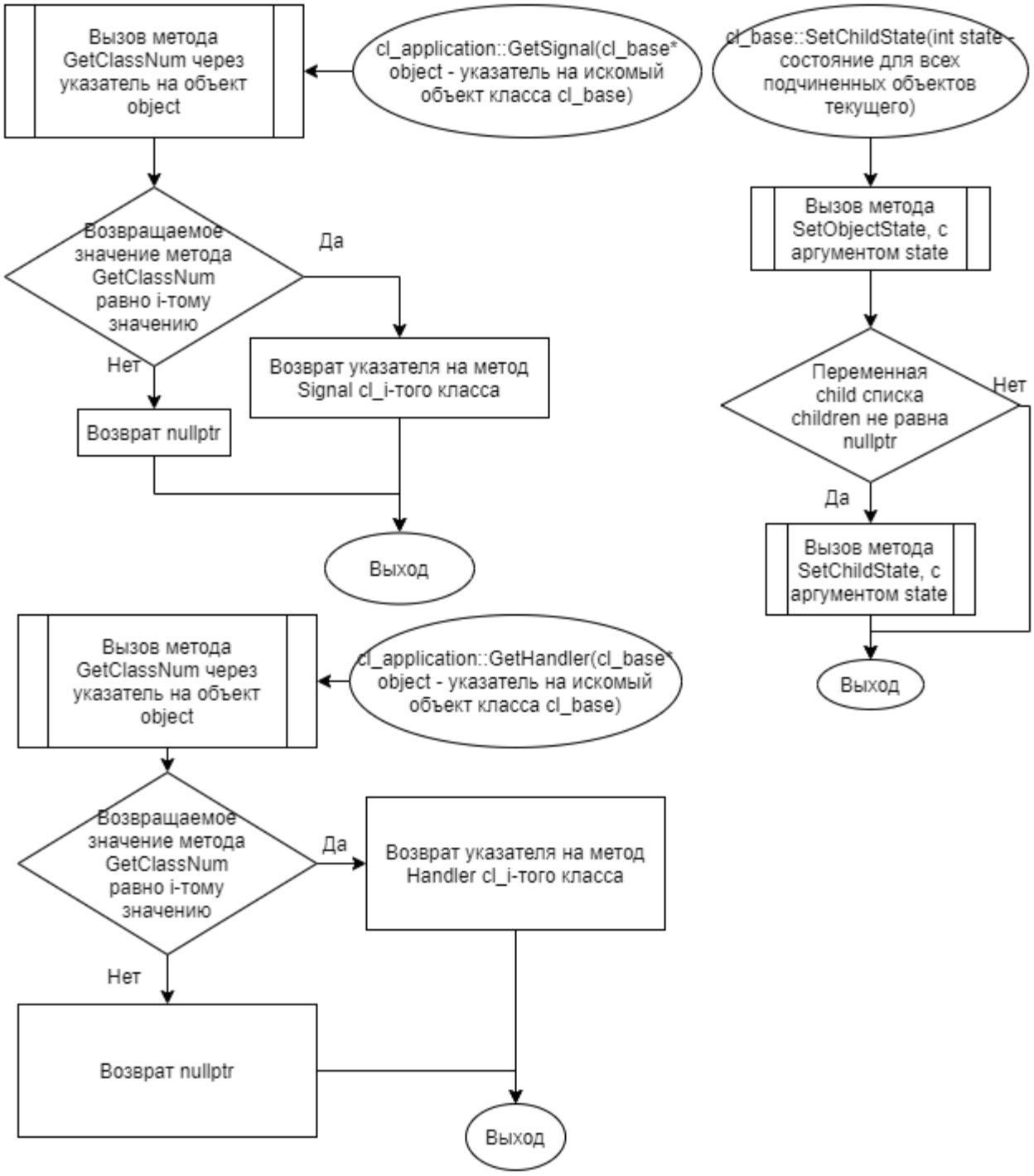

Рисунок 5 – Блок-схема алгоритма

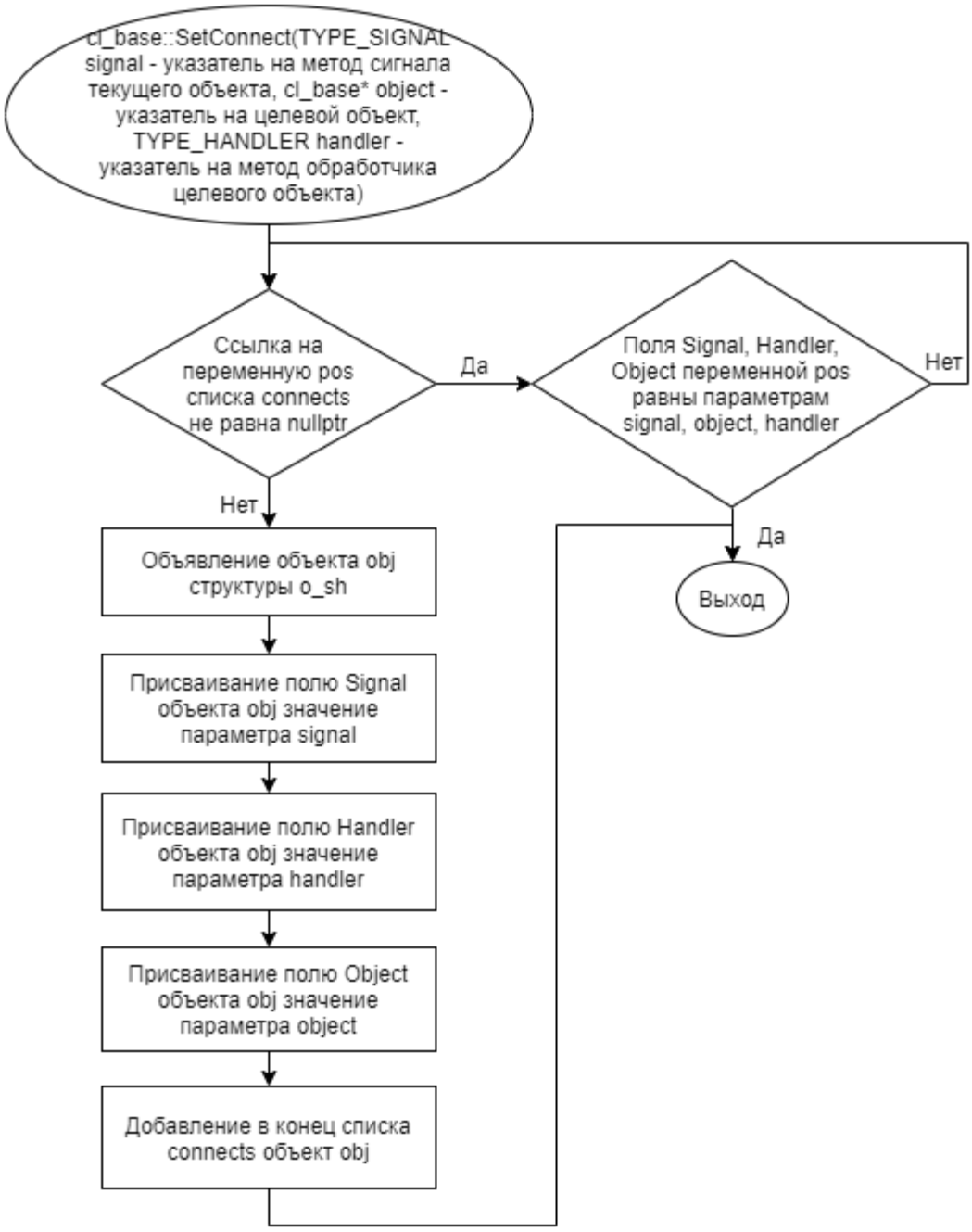

Рисунок 6 – Блок-схема алгоритма

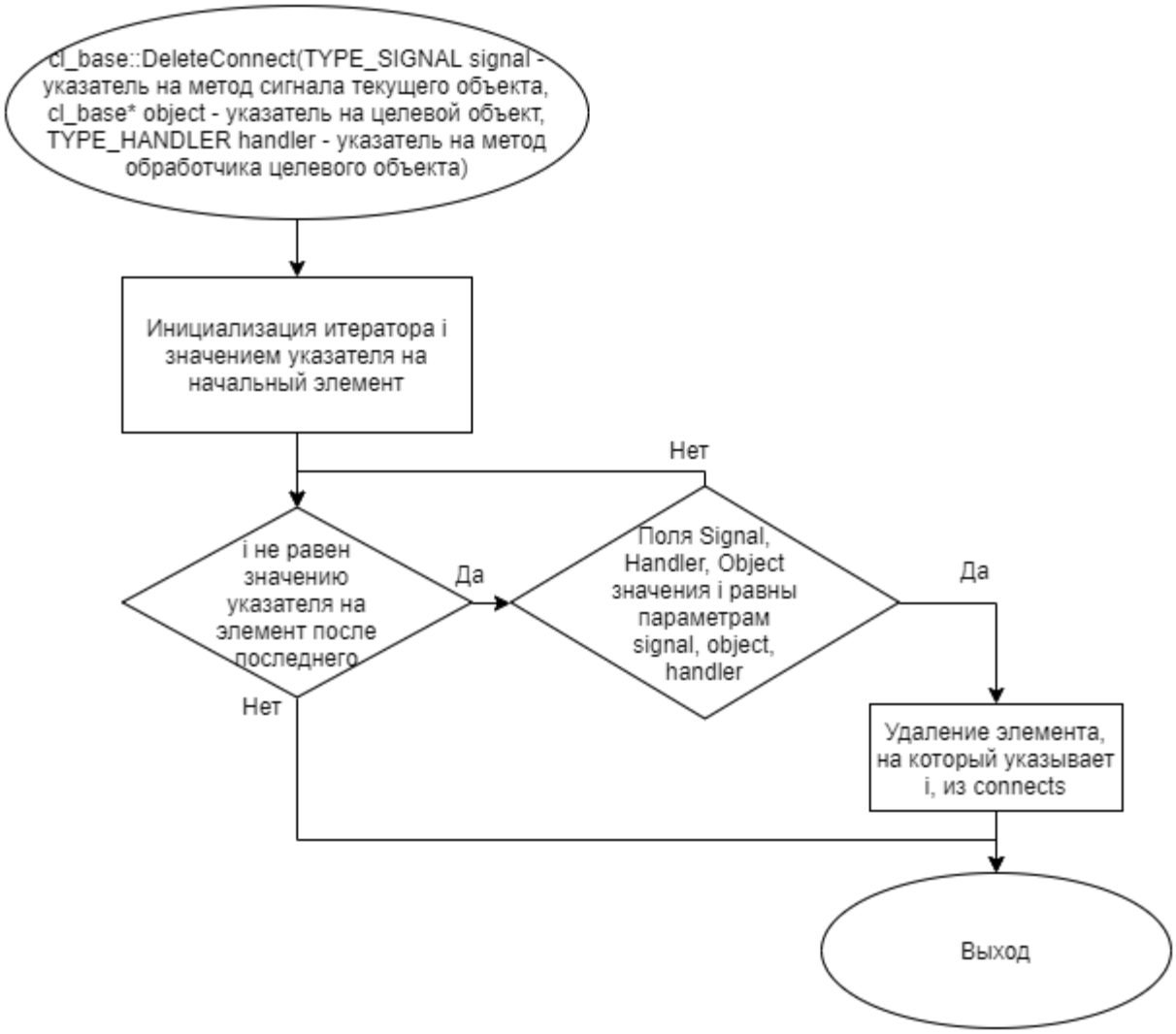

Рисунок 7 – Блок-схема алгоритма

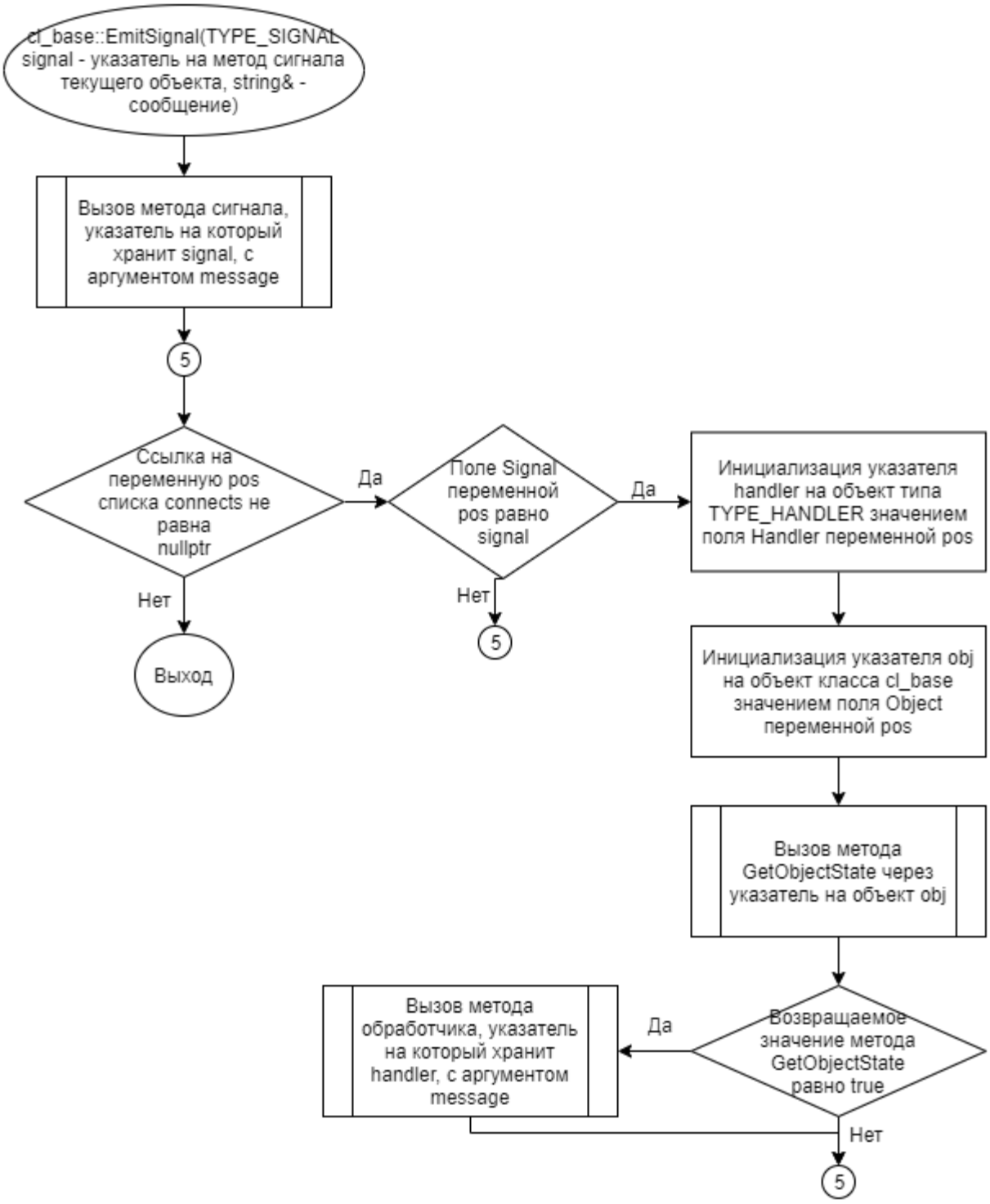

Рисунок 8 – Блок-схема алгоритма

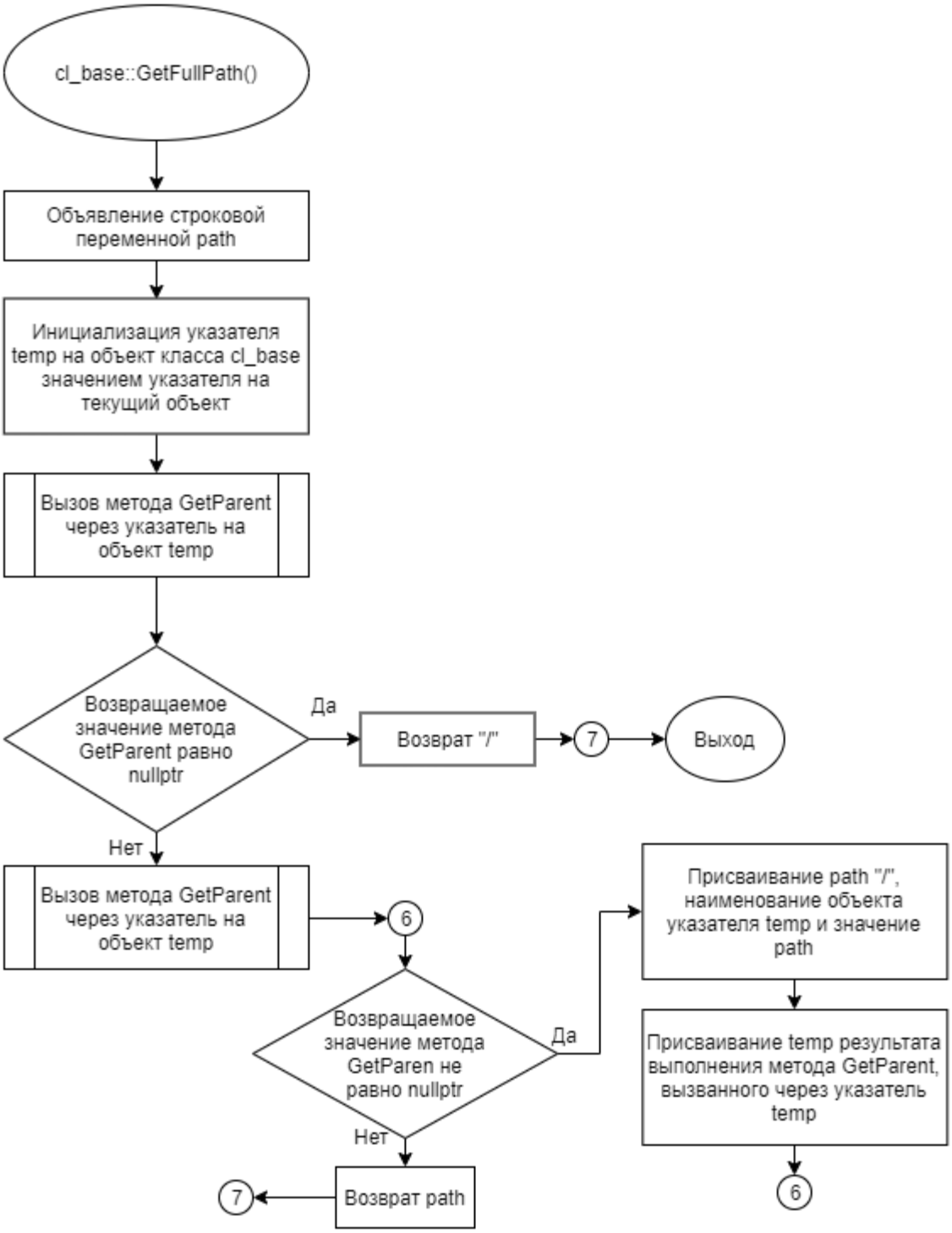

Рисунок 9 – Блок-схема алгоритма

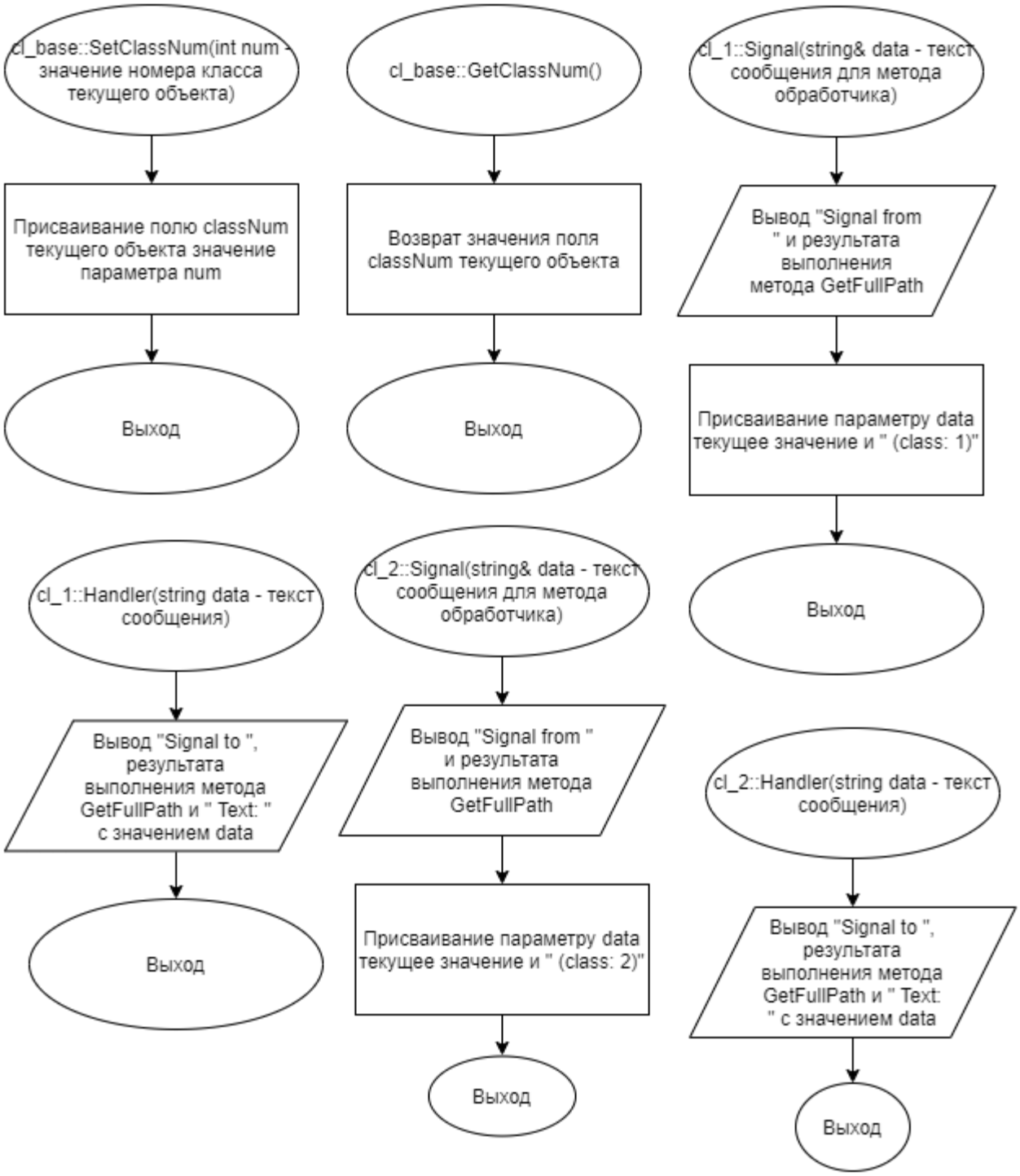

Рисунок 10 – Блок-схема алгоритма

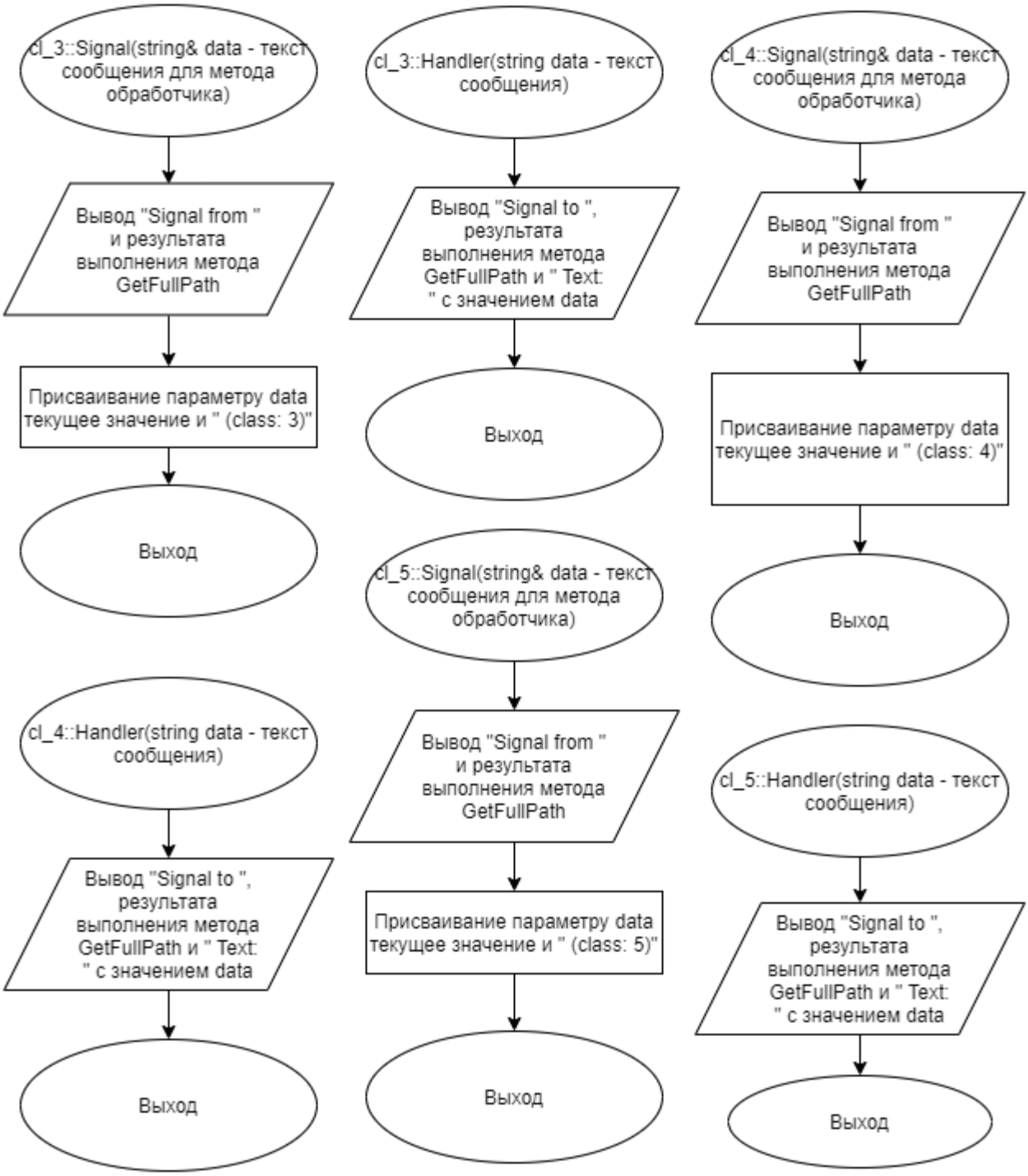

Рисунок 11 – Блок-схема алгоритма

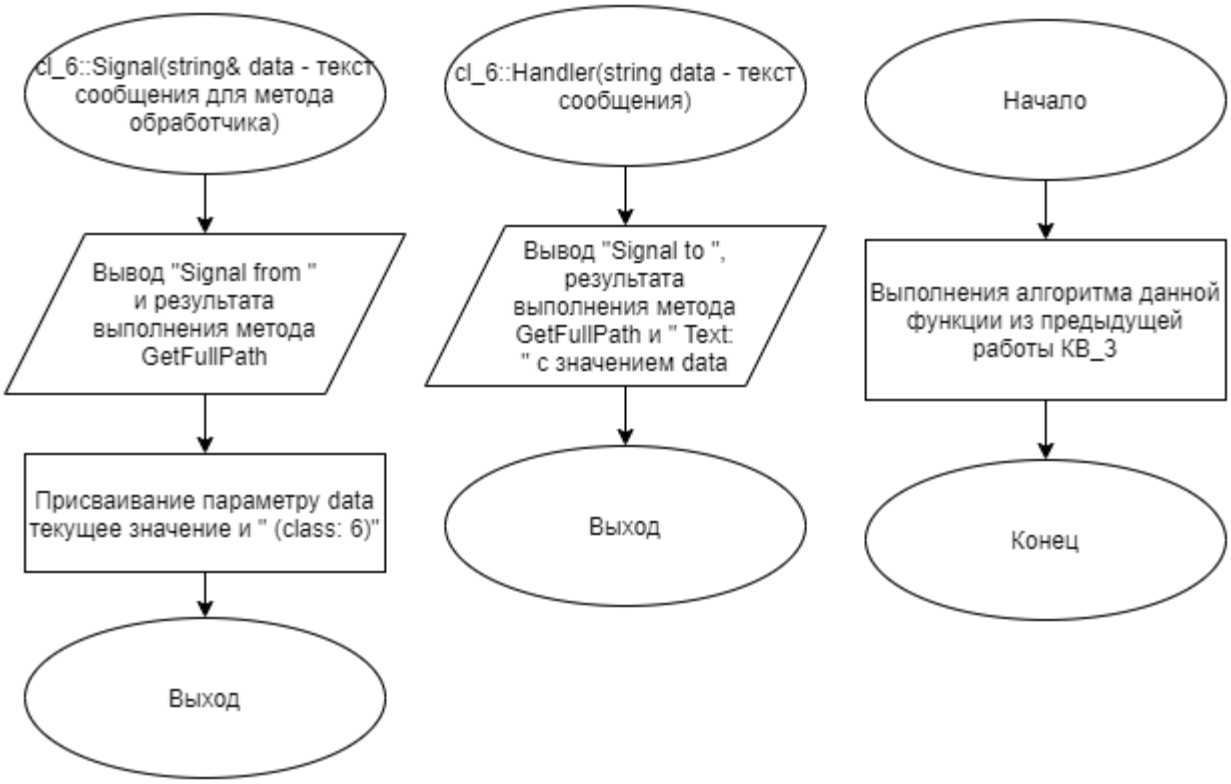

Рисунок 12 – Блок-схема алгоритма

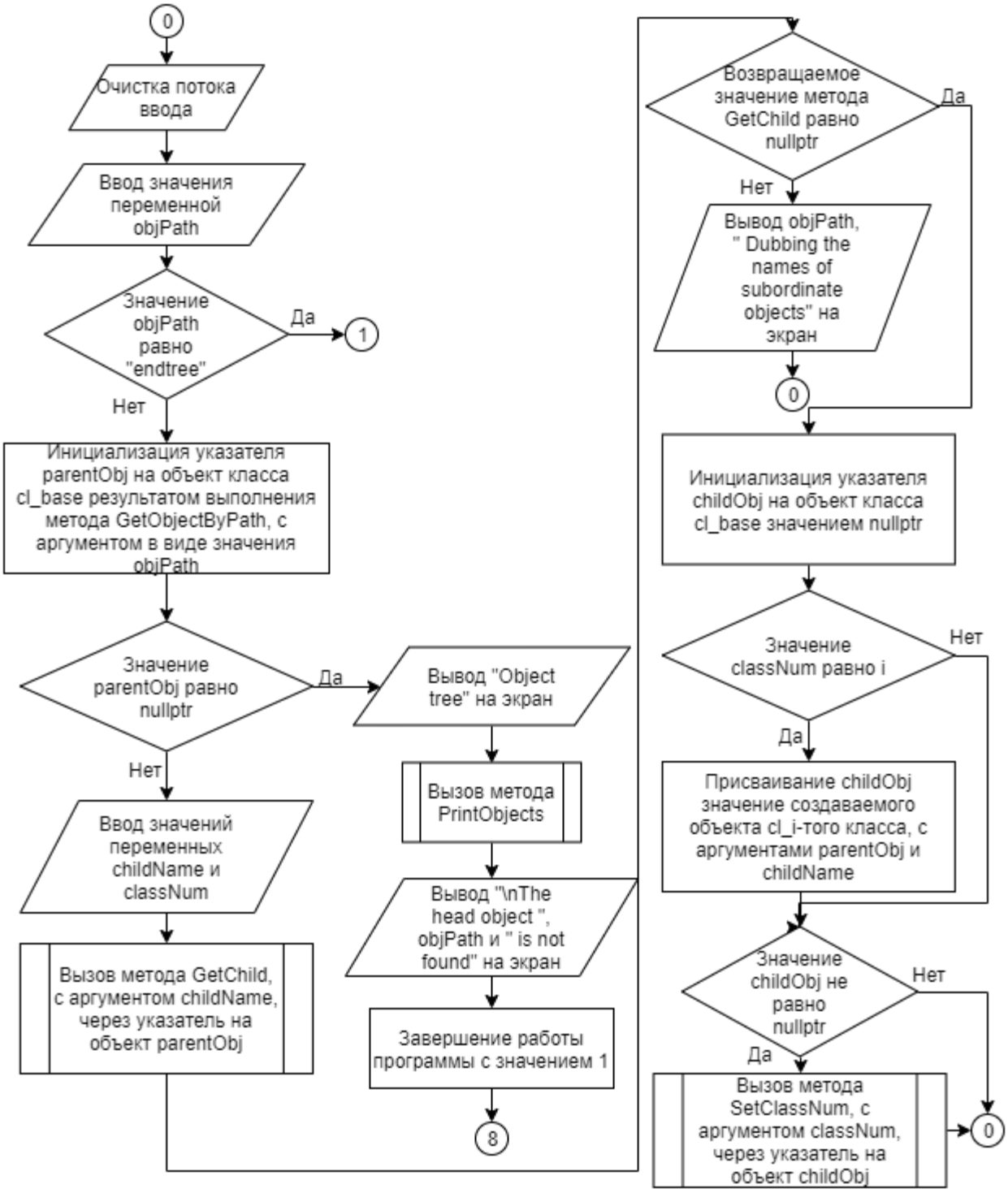

Рисунок 13 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"
cl_1::cl_1(cl_base* parent, string name) : cl_base(parent, name)
{}

// Метод сигнала
void cl_1::Signal(string& dat)
{
   cout << endl << "Signal from " + GetFullPath();
   dat = dat + " (class: 1)";
}
// Метод обработчика
void cl_1::Handler(string dat)
{
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;
}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef __CL_1__H
#define __CL_1__H
#include "cl_base.h"
class cl_1 : public cl_base
{
   public:
      cl_1(cl_base* parent, string name);
      // КВ4
      // Сигнал
      void Signal(string& dat);
      // Обработчик
      void Handler(string dat);
};
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"
cl_2::cl_2(cl_base* parent, string name) : cl_base(parent, name)
{}
void cl_2::Signal(string& dat)
{
   cout << endl << "Signal from " + GetFullPath();
   dat = dat + " (class: 2)";
}
void cl_2::Handler(string dat)
{
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;
}
```

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* parent, string name);
      // КВ4
      // Сигнал
      void Signal(string& dat);
      // Обработчик
      void Handler(string dat);
};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

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

```
cl_3::cl_3(cl_base* parent, string name) : cl_base(parent, name)
{}
void cl_3::Signal(string& dat)
{
   cout << endl << "Signal from " + GetFullPath();
   dat = dat + " (class: 3)";
}
void cl_3::Handler(string dat)
{
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;
}
```

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* parent, string name);
      // КВ4
      // Сигнал
      void Signal(string& dat);
      // Обработчик
      void Handler(string dat);
};
#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(cl_base* parent, string name) : cl_base(parent, name)
{}
void cl_4::Signal(string& dat)
{
   cout << endl << "Signal from " + GetFullPath();
   dat = dat + " (class: 4)";
}
void cl_4::Handler(string dat)
```

```
{
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;
}
```

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* parent, string name);
      // КВ4
      // Сигнал
      void Signal(string& dat);
      // Обработчик
      void Handler(string dat);
};
#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"
cl_5::cl_5(cl_base* parent, string name) : cl_base(parent, name)
{}
void cl_5::Signal(string& dat)
{
   cout << endl << "Signal from " + GetFullPath();
   dat = dat + " (class: 5)";
}
void cl_5::Handler(string dat)
{
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;
}
```

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* parent, string name);
      // КВ4
      // Сигнал
      void Signal(string& dat);
      // Обработчик
      void Handler(string dat);
};
#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"
cl_6::cl_6(cl_base* parent, string name) : cl_base(parent, name)
{}
void cl_6::Signal(string& dat)
{
   cout << endl << "Signal from " + GetFullPath();
   dat = dat + " (class: 6)";
}
void cl_6::Handler(string dat)
{
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;
}
```

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* parent, string name);
      // КВ4
      // Сигнал
      void Signal(string& dat);
      // Обработчик
      void Handler(string dat);
};
#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.h"
cl_application::cl_application(cl_base* parent) : cl_base(parent)
{}
void cl_application::build_tree_objects()
{
   string objPath, childName;
   int state;
   int classNum = 1;
   cin >> objPath;
   SetName(objPath);
   SetClassNum(classNum);
   while(true)
   {
      cin.clear();
      cin >> objPath;
      if (objPath == "endtree")
      {
         break;
      }
      cl_base* parentObj = GetObjectByPath(objPath);
      // Если указатель головной не найден
      if (!parentObj)
      {
         // Выводим готовое дерево иерархии и завершаем выполнение программы
         cout << "Object tree" << endl;
         PrintObjects();
         cout << "\nThe head object " << objPath << " is not found";
         exit(1);
      }
      cin >> childName >> classNum;
      // Если у головного объекта в подчиненных нет объекта с наименованием
childName
      if (!parentObj->GetChild(childName))
      {
```

```
            cl_base* childObj = nullptr;
            switch(classNum)
            {
               case 2:
               {
                  childObj = new cl_2(parentObj, childName);
                  break;
               }
               case 3:
               {
                  childObj = new cl_3(parentObj, childName);
                  break;
               }
               case 4:
               {
                  childObj = new cl_4(parentObj, childName);
                  break;
               }
               case 5:
               {
                  childObj = new cl_5(parentObj, childName);
                  break;
               }
               case 6:
               {
                  childObj = new cl_6(parentObj, childName);
                  break;
               }
            }
            if (childObj != nullptr) childObj->SetClassNum(classNum);
         }
         else
         {
            // Если обнаружен дубликат имени
            cout << objPath << " Dubbing the names of subordinate objects" <<
endl;
         }
      }
      while(objPath != "end_of_connections")
      {
         cin >> objPath;
         if (objPath != "end_of_connections")
         {
            // Находим два пути
            cl_base* headPtr = GetObjectByPath(objPath);
            cin >> objPath;
            cl_base* childPtr = GetObjectByPath(objPath);
            // Создаем соединение
            headPtr->SetConnect(GetSignal(headPtr), childPtr,
            GetHandler(childPtr));
         }
      }
   }
   int cl_application::exec_app()
   {
```

```
        // Вывод готового дерева
        cout << "Object tree" << endl;
        PrintObjects();
        SetChildState(1);
        cl_base* obj;
        cl_base* temp;
        string command, message, objPath;
        while(true)
        {
           // Ввод команды для взаимодействия с объектами дерева
           cin >> command;
           if (command == "END")
           {
              break;
           }
           // Ввод пути объекта
           cin >> objPath;
           // Находим объект по введенному пути
           obj = GetObjectByPath(objPath);
           if (obj == nullptr)
           {
              cout << endl << "Object " << objPath << " not found";
              continue;
           }
           // Считываем всю строку
           getline(cin, message);
           if (command == "EMIT")
           {
              if (obj->GetObjectState())
              {
                 // Посылаем сигнал
                 obj->EmitSignal(GetSignal(obj), message);
              }
           }
           else if (command == "SET_CONDITION")
           {
              obj->SetObjectState(stoi(message));
           }
           else
           {
              // Удаляем первый символ, т.к пробел
              message = message.substr(1);
              temp = GetObjectByPath(message);
              if (temp == nullptr)
              {
                 cout << endl << "Handler object " << message << " not found";
                 continue;
              }
              if (command == "SET_CONNECT")
              {
                 obj->SetConnect(GetSignal(obj), temp, GetHandler(temp));
              }
              else if (command == "DELETE_CONNECT")
              {
                 obj->DeleteConnect(GetSignal(obj), temp,
```

```
                GetHandler(temp));
            }
         }
      }
      return 0;
   }
   // КВ4
   // Определяем нужный метод сигнала
   TYPE_SIGNAL cl_application::GetSignal(cl_base* object)
   {
      switch(object->GetClassNum())
      {
         case 1:
            return SIGNAL_D(cl_1::Signal);
            break;
         case 2:
            return SIGNAL_D(cl_2::Signal);
            break;
         case 3:
            return SIGNAL_D(cl_3::Signal);
            break;
         case 4:
            return SIGNAL_D(cl_4::Signal);
            break;
         case 5:
            return SIGNAL_D(cl_5::Signal);
            break;
         case 6:
            return SIGNAL_D(cl_6::Signal);
            break;
      }
      return nullptr;
   }
   // Определяем нужный метод обработчика
   TYPE_HANDLER cl_application::GetHandler(cl_base* object)
   {
      switch(object->GetClassNum())
      {
         case 1:
            return HANDLER_D(cl_1::Handler);
            break;
         case 2:
            return HANDLER_D(cl_2::Handler);
            break;
         case 3:
            return HANDLER_D(cl_3::Handler);
            break;
         case 4:
            return HANDLER_D(cl_4::Handler);
            break;
         case 5:
            return HANDLER_D(cl_5::Handler);
            break;
         case 6:
            return HANDLER_D(cl_6::Handler);
```

```
            break;
      }
      return nullptr;
   }
```

5.14 Файл cl_application.h

Листинг 14 – cl_application.h

```
   #ifndef CL_APPLICATION_H
   #define CL_APPLICATION_H
   #include "cl_base.h"
   #include "cl_1.h"
   #include "cl_2.h"
   #include "cl_3.h"
   #include "cl_4.h"
   #include "cl_5.h"
   #include "cl_6.h"
   class cl_application : public cl_base
   {
      public:
         cl_application(cl_base* parent);
         void build_tree_objects();
         int exec_app();
         // KB4
         TYPE_SIGNAL GetSignal(cl_base*);
         TYPE_HANDLER GetHandler(cl_base*);
   };
   #endif
```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
   #include "cl_base.h"
   cl_base::cl_base(cl_base* parent, string name)
   {
      this->parent = parent;
      this->name = name;
      if (GetParent() != nullptr)
      {
         GetParent()->children.push_back(this);
      }
   }
   cl_base::~cl_base()
```

```
{
   for (auto child : children)
   {
      delete child;
   }
}
bool cl_base::SetName(string newName)
{
   // Если родитель для текущего найден и
   // он не имеет подчиненнго с именем newName
   if(GetParent() != nullptr && GetParent()->GetChild(newName) != nullptr)
   {
      return false;
   }
   name = newName;
   return true;
}
string cl_base::GetName() const
{
   return name;
}
cl_base* cl_base::GetParent() const
{
   return parent;
}
cl_base* cl_base::GetChild(string objName) const
{
   // Проход по всем подчиненным объектам
   for (auto child : children)
   {
      if (child->GetName() == objName)
      {
         return child;
      }
   }
   return nullptr;
}
//КВ2
// Поиск количества объектов на ветке по имени
// через рекурсию
int cl_base::ObjNameCount(string objName)
{
   int count = 0;
   if(GetName() == objName)
   {
      count++;
   }
   for (auto child : children)
   {
      count += child->ObjNameCount(objName);
   }
   return count;
}
// Проверка на уникальность в ветке по имени
cl_base* cl_base::CheckingObjUniq(string objName)
```

```
{
   if (ObjNameCount(objName) != 1)
   {
      return nullptr;
   }
   return SearchObjOnBranch(objName);
}
cl_base* cl_base::SearchObjOnBranch(string objName)
{
   // Если имя объекта равно параметру objname
   if (GetName() == objName)
   {
      return this;
   }
   for (auto child : children)
   {
      cl_base* subChild = child->SearchObjOnBranch(objName);
      // Если объект не равен nullptr
      if (subChild)
      {
         return subChild;
      }
   }
   return nullptr;
}
// Поиск по всему дереву от корневого
cl_base* cl_base::SearchObjOnTree(string objName)
{
   return GetRoot()->CheckingObjUniq(objName);
}
//Вывод
void cl_base::PrintObjects(int spaces) const
{
   cout << GetName();
   if (!children.empty())
   {
      for (auto child : children)
      {
         cout << endl;
         for (int i = 0; i < spaces; i++)
         cout << " ";
         child->PrintObjects(spaces+4);
      }
   }
}
void cl_base::PrintObjectsStates(int spaces) const
{
   cout << GetName();
   cout << (GetObjectState() ? " is ready" : " is not ready");
   if (!children.empty())
   {
      for (auto child : children)
      {
         cout << endl;
         for (int i = 0; i < spaces; i++)
```

```
            cout << " ";
            child->PrintObjectsStates(spaces+4);
         }
      }
   }
   //Состояние объекта
   void cl_base::SetObjectState(bool state)
   {
      if (GetParent() && !GetParent()->GetObjectState())
      {
         this->state = false;
      }
      else
      {
         this->state = state;
      }
      if (!state)
      {
         for (auto child : children)
         {
            child->SetObjectState(state);
         }
      }
   }
   bool cl_base::GetObjectState() const
   {
      return state;
   }
   //КВЗ
   cl_base* cl_base::GetRoot()
   {
      cl_base* obj = this;
      // Доходим до корневого объекта
      while(obj->GetParent())
      {
         obj = obj->GetParent();
      }
      return obj;
   }
   // Поиск объекта по пути
   cl_base* cl_base::GetObjectByPath(string path)
   {
      // Указатель на текущий
      cl_base* currentObj = this;
      string nextObjName;
      if (path.substr(0,1) == "/")
      {
         if (path.substr(1,1) == "/")
         {
            return SearchObjOnTree(path.substr(2));
         }
         // Текущий равен корневому
         currentObj = GetRoot();
         path = path.substr(1);
      }
```

```
   else if (path.substr(0,1) == ".")
   {
      return path == "." ? currentObj : CheckingObjUniq(path.substr(1));
   }
   stringstream streamPath(path);
   // Пока можно разбивать строку на подстроки по '/'
   while(getline(streamPath, nextObjName, '/'))
   {
      currentObj = currentObj->GetChild(nextObjName);
      // Если такой объект не найден
      if (!currentObj)
      {
         return nullptr;
      }
   }
   return currentObj;
}
// Смена головного объекта
bool cl_base::ChangeHeadObj(cl_base* newHead)
{
   // Если головной объект найден
   if (GetParent())
   {
      for (auto i = (GetParent()->children).begin(); i != (GetParent()-
>children).end(); i++)
      {
         if (*i == this)
         {
            // Удаляем у головного объекта текущего сам текущий
            (GetParent()->children).erase(i);
            break;
         }
      }
      // Переопределяем головной объект
      this->parent = newHead;
      // Добавление объекта к новому головному
      (GetParent()->children).push_back(this);
      return true;
   }
   return false;
}
// Удаление объекта
bool cl_base::DeleteSubObj(string objName)
{
   // Находим в подчиненных по имени
   cl_base* subObj = GetChild(objName);
   for (auto i = children.begin(); i != children.end(); i++)
   {
      if (*i == subObj)
      {
         // Удаление объекта по его итератору в списке
         children.erase(i);
         delete subObj;
         return true;
      }
```

```
   }
   return false;
}
//KB4
// Метод смены состояния всех подчиненных объектов
void cl_base::SetChildState(int state)
{
   SetObjectState(state);
   for (auto child : children)
   {
      child->SetChildState(state);
   }
}
// Установка связи между сигналом текущего объекта и обработчиком целевого
объекта
// signal - указатель на метод сигнала текущего объекта
// object - указатель целевой объект
// handler - указатель на метод обработчика целевого объекта
void cl_base::SetConnect(TYPE_SIGNAL signal, cl_base* object, TYPE_HANDLER
handler)
{
   for(auto& pos : connects)
   {
      // Если соединение найдено, то нет смысла устанавливать соединение
      if (pos.Signal==signal && pos.Handler==handler && pos.Object==object)
      {
         return;
      }
   }
   // Создаем новое соединение
   o_sh obj;
   obj.Signal = signal;
   obj.Handler = handler;
   obj.Object = object;
   connects.push_back(obj);
}
// Удаление связи между сигналом текущего объекта и обработчиком целевого
объекта
void    cl_base::DeleteConnect(TYPE_SIGNAL    signal,    cl_base*    object,
TYPE_HANDLER handler)
{
   for (auto i = connects.begin(); i != connects.end(); i++)
   {
      // Если соединение найдено
      if ((*i).Signal==signal
      && (*i).Handler==handler
      && (*i).Object==object)
      {
         // Удаляем
         connects.erase(i);
         return;
      }
   }
}
// Выдача сигнала от текущего объекта с передачей строковой переменной
```

```
void cl_base::EmitSignal(TYPE_SIGNAL signal, string& message)
{
   // Вызов метода сигнала с передачей в него message
   (this->*signal)(message);
   for(auto& pos : connects)
   {
      // Если сигналы сходятся
      if (pos.Signal==signal)
      {
         TYPE_HANDLER handler = pos.Handler;
         cl_base* obj = pos.Object;
         if (obj->GetObjectState())
         {
            // Вызов метода обработчика с передачей в него message
            (obj->*handler)(message);
         }
      }
   }
}
string cl_base::GetFullPath()
{
   // Переменная для полного пути объекта
   string path;
   // Инициализируем временный указатель на текущий
   cl_base* temp = this;
   // Если род. нет, то возвращаем путь на корень
   if(temp->GetParent() == nullptr)
   {
      return "/";
   }
   // Пока родитель есть
   while(temp->GetParent() != nullptr)
   {
      // Строим полный путь к объекту
      path = "/" + temp->GetName() + path;
      temp = temp->GetParent();
   }
   return path;
}
// Установка номера класса
void cl_base::SetClassNum(int num)
{
   classNum = num;
}
// Возврат номера класса
int cl_base::GetClassNum()
{
   return classNum;
}
```

5.16 Файл cl_base.h

Листинг 16 – cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
using namespace std;
class cl_base;
#define SIGNAL_D(signal_f)(TYPE_SIGNAL)(&signal_f)
#define HANDLER_D(handler_f)(TYPE_HANDLER)(&handler_f)
typedef void (cl_base :: *TYPE_SIGNAL)(string&);
typedef void (cl_base :: *TYPE_HANDLER)(string);
class cl_base
{
   string name;
   cl_base* parent;
   vector<cl_base*> children;
   bool state = false;
   int classNum;
   // Структура соединения между объектами
   struct o_sh
   {
   TYPE_SIGNAL Signal;
   // Куда идет сигнал
   cl_base* Object;
   // Какой метод у объекта обработчик
   TYPE_HANDLER Handler;
   };
   // Список соединений объекта
   vector<o_sh> connects;
   public:
      //KB1
      cl_base(cl_base* parent, string name = "Base_object");
      ~cl_base();
      // Установка наименования для текущего объекта
      bool SetName(string newName);
      // Возврат значения наименования объекта
      string GetName() const;
      // Возврат указателя на головной объект
      cl_base* GetParent() const;
      // Возврат указателя на подчиненный объект
      cl_base* GetChild(string objName) const;
      //KB2
      // Проверка на уникальность
      int ObjNameCount(string objName);
      // Проверка объекта на уникальность на ветке
      cl_base* CheckingObjUniq(string objName);
      // Поиск объекта на ветке по наименованию
      cl_base* SearchObjOnBranch(string objName);
      // Поиск объекта на дереве по наименованию
```

```
        cl_base* SearchObjOnTree(string objName);
        // Вывод дерева иерархии объектов
        void PrintObjects(int spaces = 4) const;
        // Методы для состояния объекта - КВ2
        void PrintObjectsStates(int spaces = 4) const;
        // Установка состояния объекта
        void SetObjectState(bool state);
        // Возврат состояния объекта
        bool GetObjectState() const;
        //КВ3
        // Возврат указателя на корневой объект иерархии
        cl_base* GetRoot();
        // Возврат указателя на объект иерархии по передаваемому пути
        cl_base* GetObjectByPath(string path);
        // Переопределение головного объекта для текущего в дереве иерархии
        bool ChangeHeadObj(cl_base* newHead);
        // Удаление объекта у текущего в дереве иерархии
        bool DeleteSubObj(string objName);
        //КВ4
        // Установка состояния всем объектам иерархии
        void SetChildState(int);
        // Установка соединения
        void SetConnect(TYPE_SIGNAL, cl_base*, TYPE_HANDLER);
        // Удаления соединения
        void DeleteConnect(TYPE_SIGNAL, cl_base*, TYPE_HANDLER);
        // Выдача сигнала
        void EmitSignal(TYPE_SIGNAL, string&);
        // Получение полного пути
        string GetFullPath();
        // Установка номера класса
        void SetClassNum(int);
        // Получение номера класса
        int GetClassNum();
};
#endif
```

5.17 Файл main.cpp

Листинг 17 – main.cpp

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

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 25.

Таблица 25 – Результат тестирования программы

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

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

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
/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 SET_CONNECT /object_s2/object_s4 /object_s13 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 4 SET_CONDITION /object_s2/object_s4 0 EMIT /object_s2/object_s4 Send message 5 END	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 to /object_s13 Text: Send message 3 (class: 4) Signal from /object_s1	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 to /object_s13 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 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	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	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

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
EMIT /object_s2/object_s4 Send message 2 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 4 DELETE_CONNECT /object_s2/object_s4 / EMIT /object_s2/object_s4 Send message 2 SET_CONDITION /object_s2/object_s4 0 EMIT /object_s2/object_s4 Send message 3 SET_CONNECT /object_s1 /object_s2/object_s6 EMIT /object_s1 Send message 4 SET_CONDITION /object_s1/object_s7 0 EMIT /object_s2 Send message 48 END	2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3) Signal from /object_s2	2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3) Signal from /object_s2

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были реализованы все необходимые классы и методы для формирования работы системы, а также методы сигналов и обработчиков для взаимодействия объектов системы между собой. При написании кода были соблюдены основные парадигмы объектно-ориентированного программирования.

В процессе написания программы были закреплены знания и практические навыки, полученные в течение обучения дисциплине "Объектно-ориентированное программирование". Также был получен опыт разработки системы, состоящей из нескольких объектов, выстроенных в дерево иерархии.

Работоспособность программы подтверждена тестированием на множестве тестов.

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).