Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков, с передачей вместе сигналом текстового сообщения (строковой переменной).

Для организации взаимосвязи по механизму сигналов и обработчиков в базовый класс добавить три метода:

- установления связи между сигналом текущего объекта и обработчиком целевого объекта;
- удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
- выдачи сигнала от текущего объекта с передачей строковой переменной. Включенный объект может выдать или обработать сигнал.

Методу установки связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу удаления (разрыва) связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу выдачи сигнала передать указатель на метод сигнала и строковую переменную. В данном методе реализовать алгоритм:

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

    3.1. Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то проверить готовность целевого объекта. Если целевой объект готов, то вызвать метод обработчика

целевого объекта указанной в связи и передать в качестве аргумента строковую переменную по значению.

4. Конец цикла.

Для приведения указателя на метод сигнала и на метод обработчика использовать параметризированное макроопределение препроцессора.

В базовый класс добавить метод определения абсолютной пути до текущего объекта. Этот метод возвращает абсолютный путь текущего объекта.

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 3 курсовой работы. Если при построении дерева иерархии возникает ситуация дубляжа имен среди починенных у текущего головного объекта, то новый объект не создается.

Система содержит объекты шести классов с номерами: 1, 2, 3, 4, 5, 6. Классу корневого объекта соответствует номер 1. В каждом производном классе реализовать один метод сигнала и один метод обработчика.

Каждый метод сигнала с новой строки выводит:

```
Signal from «абсолютная координата объекта»
```

Каждый метод сигнала добавляет переданной по параметру строке текста номер класса принадлежности текущего объекта по форме:

```
«пробел»(class: «номер класса»)
```

Каждый метод обработчика с новой строки выводит:

```
Signal to «абсолютная координата объекта»   Text: «переданная строка»
```

Моделировать работу системы, которая выполняет следующие команды с параметрами:

- EMIT «координата объекта» «текст» – выдает сигнал от заданного по координате объекта;
- SET_CONNECT «координата объекта выдающего сигнал» «координата

целевого объекта» – устанавливает связь;

- DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаляет связь;
- SET_CONDITION «координата объекта» «значение состояния» – устанавливает состояние объекта.
- END – завершает функционирование системы (выполнение программы).

Реализовать алгоритм работы системы:

- в методе построения системы:
  - построение дерева иерархии объектов согласно вводу;
  - ввод и построение множества связей сигнал-обработчик для заданных пар объектов.
- в методе отработки системы:
  - привести все объекты в состоянии готовности;
  - цикл до признака завершения ввода:
    - ввод наименования объекта и текста сообщения;
    - вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной, содержащей текст сообщения.
  - конец цикла.

Допускаем, что все входные данные вводятся синтаксически корректно. Контроль корректности входных данных можно реализовать для самоконтроля работы программы. Не оговоренные, но необходимые функции и элементы классов добавляются разработчиком.

1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве

иерархии. Структура данных для ввода согласно изложенному в версии № 3 курсовой работы.

После ввода состава дерева иерархии построчно вводится:

```
«координата объекта выдающего сигнал» «координата целевого объекта»
```

Ввод информации для построения связей завершается строкой, которая содержит:

```
«end_of_connections»
```

В методе запуска (отработки) системы построчно вводятся множество команд в производном порядке:

- EMIT «координата объекта» «текст» – выдать сигнал от заданного по координате объекта;
- SET_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – установка связи;
- DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаление связи;
- SET_CONDITION «координата объекта» «значение состояния» – установка состояния объекта.
- END – завершить функционирование системы (выполнение программы).

Команда END присутствует обязательно.

Если координата объекта задана некорректно, то соответствующая операция не выполняется и с новой строки выдается сообщение об ошибке.

Если не найден объект по координате:

```
Object «координата объекта» not found
```

Если не найден целевой объект по координате:

```
Handler object «координата целевого объекта» not found
```

Пример ввода:

```
appls_root
/ object_s1 3
/ object_s2 2
/object_s2 object_s4 4
/ object_s13 5
/object_s2 object_s6 6
/object_s1 object_s7 2
endtree
/object_s2/object_s4 /object_s2/object_s6
/object_s2 /object_s1/object_s7
/ /object_s2/object_s4
/object_s2/object_s4 /
end_of_connections
EMIT /object_s2/object_s4 Send message 1
EMIT /object_s2/object_s4 Send message 2
EMIT /object_s2/object_s4 Send message 3
EMIT /object_s1 Send message 4
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

```
Signal from «абсолютная координата объекта»
```

Если отработал метод обработчика:

```
Signal to «абсолютная координата объекта»   Text: «переданная строка»
```

Пример вывода:

```
Object tree
appls_root
    object_s1
        object_s7
    object_s2
        object_s4
        object_s6
    object_s13
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text:  Send message 1 (class: 4)
Signal to / Text:  Send message 1 (class: 4)
Signal from /object_s2/object_s4
```

```
Signal to /object_s2/object_s6 Text:  Send message 2 (class: 4)
Signal to / Text:  Send message 2 (class: 4)
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text:  Send message 3 (class: 4)
Signal to / Text:  Send message 3 (class: 4)
Signal from /object_s1
```

2 МЕТОД РЕШЕНИЯ

Класс kv_base:

- функционал:
    - метод SetChildstate — используется для смены состояний всех подчиненных объектов текущего на значение параметра;
    - метод SetConnect — используется для установления связи между сигналом текущего объекта и обработчиком целевого объекта;
    - метод DeleteConnect — используется для удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
    - метод EmitSignal — используется для выдачи сигнала от текущего объекта с передачей строковой переменной;
    - метод GetFullPath — используется для возращения полного пути текущего объекта иерархии;
    - метод SetClassNum — используется для установки номера класса текущего объекта;
    - метод GetClassNum — используется для получения номера класса текущего объекта.

Класс kv_1:

- функционал:
    - метод Signal — используется как сигнал текущего объекта;
    - метод Handler — используется как обработчик текущего объекта.

Класс kv_2:

- функционал:
    - метод Signal — используется как сигнал текущего объекта;
    - метод Handler — используется как обработчик текущего объекта.

Класс kv_3:

- функционал:
    - метод Signal — используется как сигнал текущего объекта;
    - метод Handler — используется как обработчик текущего объекта.

Класс kv_4:

- функционал:
    - метод Signal — используется как сигнал текущего объекта;
    - метод Handler — используется как обработчик текущего объекта.

Класс kv_5:

- функционал:
    - метод Signal — используется как сигнал текущего объекта;
    - метод Handler — используется как обработчик текущего объекта.

Класс kv_6:

- функционал:
    - метод Signal — используется как сигнал текущего объекта;
    - метод Handler — используется как обработчик текущего объекта.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	kv_base			основной класс	
		kv_1	public		2
		kv_2	public		3
		kv_3	public		4
		kv_4	public		5
		kv_5	public		6
		kv_6	public		7
2	kv_1			параметризированный конструктор	
3	kv_2			параметризированный конструктор	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
4	kv_3			параметризированный конструктор	
5	kv_4			параметризированный конструктор	
6	kv_5			параметризированный конструктор	
7	kv_6			параметризированный конструктор	

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 SetChildstate класса kv_base

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

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода SetChildstate класса kv_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	∅
			∅

3.2 Алгоритм метода SetConnect класса kv_base

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

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода SetConnect класса kv_base

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

3.3 Алгоритм метода DeleteConnect класса kv_base

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

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода DeleteConnect класса kv_base

№	Предикат	Действия	№ перехода
1		Инициализация итератора i значением указателя на начальный элемент	2
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

3.4 Алгоритм метода EmitSignal класса kv_base

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

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода EmitSignal класса kv_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		3

№	Предикат	Действия	№ перехода
	nullptr		
			∅
3	Поле Signal переменной pos равно signal	Инициализация указателя handler на объект типа TYPE_HANDLER значением поля Handler переменной pos	4
			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

3.5 Алгоритм метода GetFullPath класса kv_base

Функционал: используется для возвращения полного пути текущего объекта иерархии.

Параметры: нет.

Возвращаемое значение: string - полный путь текущего объекта.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода GetFullPath класса kv_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	4

№	Предикат	Действия	№ перехода
		temp	
4	Возвращаемое значение метода GetParent равно nullptr	Возврат "/"	∅
			5
5		Вызов метода GetParent через указатель на объект temp	6
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

3.6 Алгоритм метода SetClassNum класса kv_base

Функционал: Установка номера класса текущего объекта.

Параметры: int num - значение номера класса текущего объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода SetClassNum класса kv_base

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

3.7 Алгоритм метода GetClassNum класса kv_base

Функционал: используется для получения номера класса текущего объекта.

Параметры: нет.

Возвращаемое значение: int - значение номера класса текущего объекта.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода GetClassNum класса kv_base

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

3.8 Алгоритм метода Signal класса kv_1

Функционал: Метод сигнала текущего объекта.

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода Signal класса kv_1

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

3.9 Алгоритм метода Handler класса kv_1

Функционал: используется как обработчик текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода Handler класса kv_1

№	Предикат	Действия	№ перехода
1		Вывод "Signal to ", результата выполнения метода GetFullPath и "	∅

№	Предикат	Действия	№ перехода
		Text: " с значением data	

3.10 Алгоритм метода Signal класса kv_2

Функционал: используется как сигнал текущего объекта.

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода Signal класса kv_2

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

3.11 Алгоритм метода Handler класса kv_2

Функционал: используется как обработчик текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода Handler класса kv_2

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

3.12 Алгоритм метода Signal класса kv_3

Функционал: используется как сигнал текущего объекта.

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода Signal класса kv_3

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

3.13 Алгоритм метода Handler класса kv_3

Функционал: используется как обработчик текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода Handler класса kv_3

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

3.14 Алгоритм метода Signal класса kv_4

Функционал: используется как сигнал текущего объекта.

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода Signal класса kv_4

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

3.15 Алгоритм метода Handler класса kv_4

Функционал: используется как обработчик текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода Handler класса kv_4

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

3.16 Алгоритм метода Signal класса kv_5

Функционал: используется как сигнал текущего объекта.

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода Signal класса kv_5

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

3.17 Алгоритм метода Handler класса kv_5

Функционал: используется как обработчик текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода Handler класса kv_5

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

3.18 Алгоритм метода Signal класса kv_6

Функционал: используется как сигнал текущего объекта.

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

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода Signal класса kv_6

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

3.19 Алгоритм метода Handler класса kv_6

Функционал: используется как обработчик текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода Handler класса kv_6

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

3.20 Алгоритм функции main

Функционал: Основной алгоритм работы программы.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности завершения программы.

Алгоритм функции представлен в таблице 21.

Таблица 21 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Инициализация объекта ob_kv_application класа kv_application передав значение nullptr	2
2		Вызов метода build_tree_objects() объекта ob_kv_application	3
3		Вызов метода exec_app() для запуска программы	∅

3.21 Алгоритм метода build_tree_objects класса kv_application

Функционал: Построение исходного дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода build_tree_objects класса kv_application

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

№	Предикат	Действия	№ перехода
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
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	18

№	Предикат	Действия	№ перехода
		childPtr	

3.22 Алгоритм метода exec_app класса kv_application

Функционал: Запуск приложения.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности завершения метода.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода exec_app класса kv_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	10

№	Предикат	Действия	№ перехода
		переменную message	
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
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,	6

№	Предикат	Действия	№ перехода
		и GetHandler, с аргументом temp, и указателя temp	
			6

3.23 Алгоритм метода GetSignal класса kv_application

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

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

Возвращаемое значение: TYPE_SIGNAL - указатель на метод сигнала 22 одного из подчиненных классов класса kv_base.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода GetSignal класса kv_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	∅

3.24 Алгоритм метода GetHandler класса kv_application

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

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

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

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода GetHandler класса kv_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	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

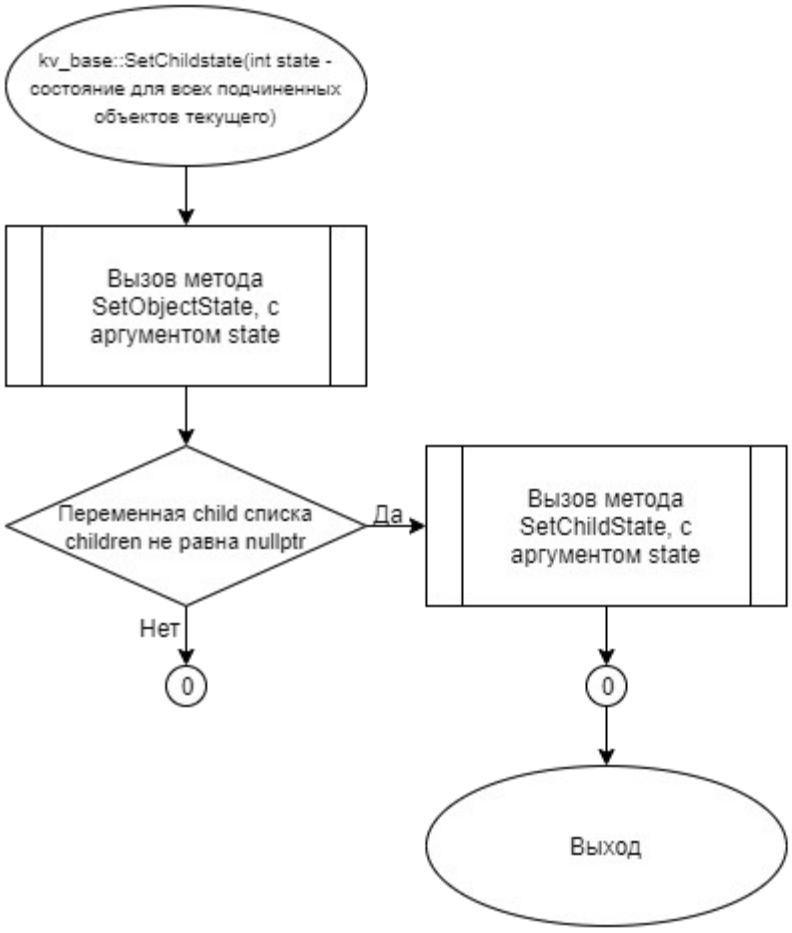

Рисунок 1 – Блок-схема алгоритма

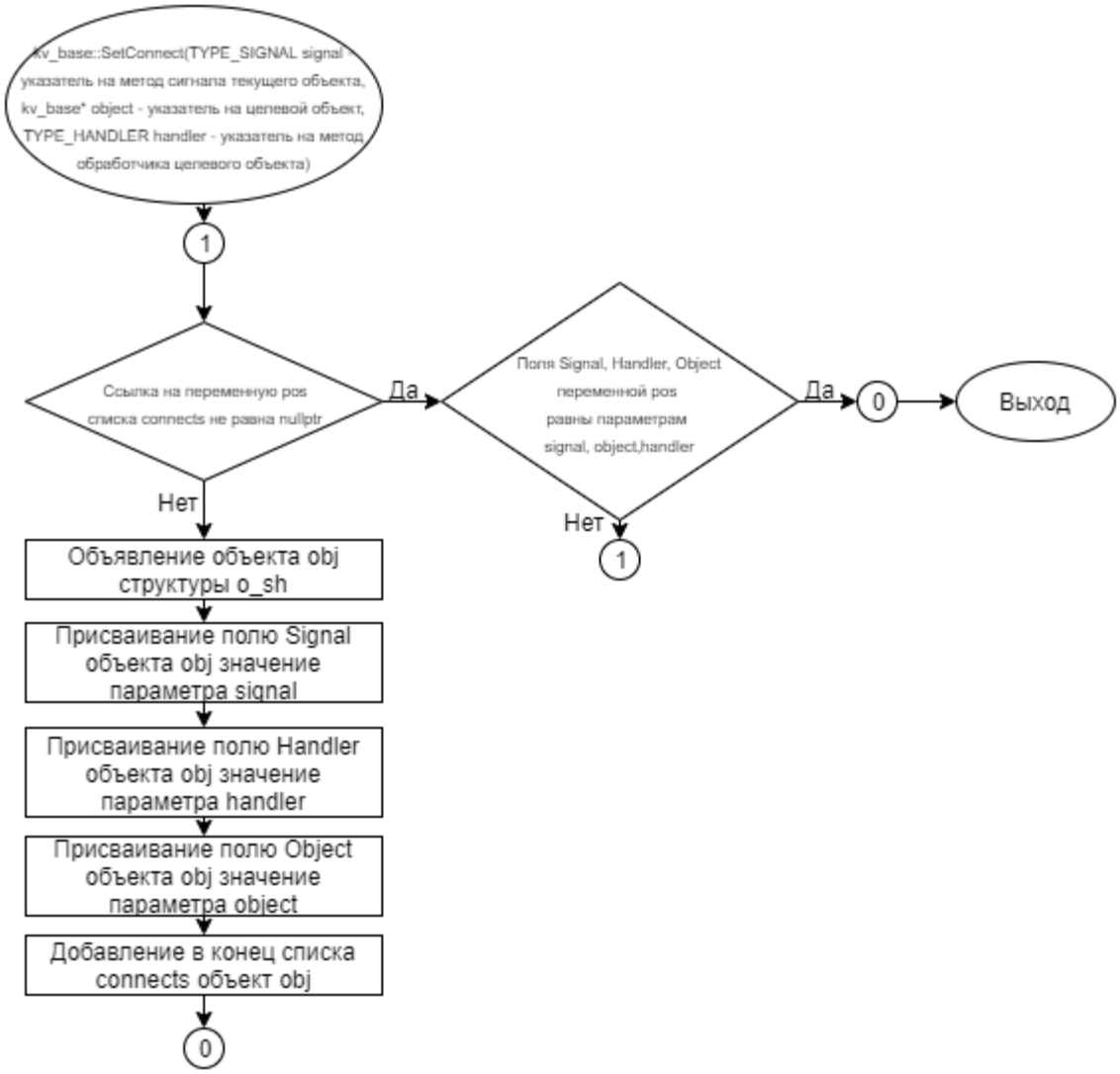

Рисунок 2 – Блок-схема алгоритма

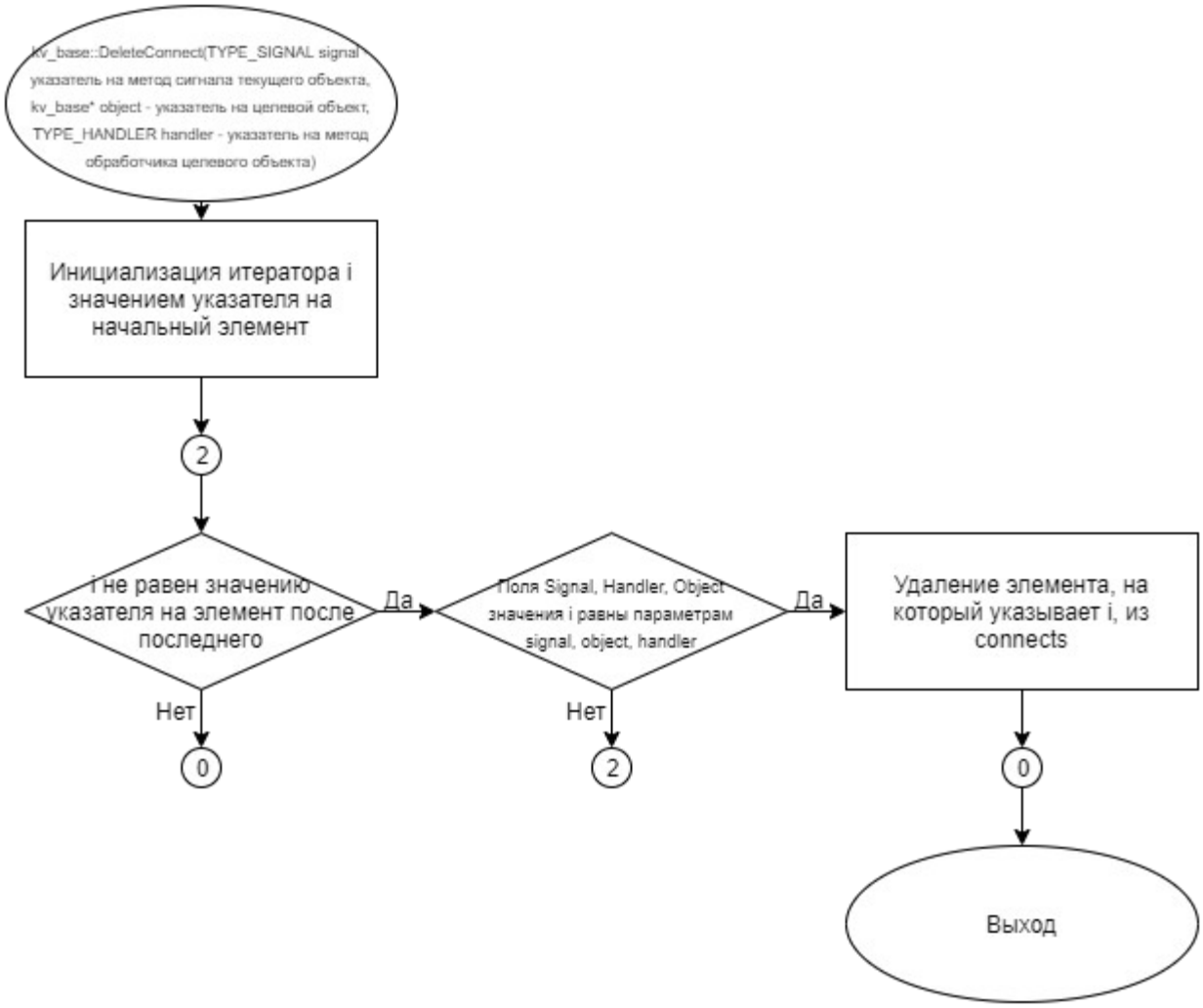

Рисунок 3 – Блок-схема алгоритма

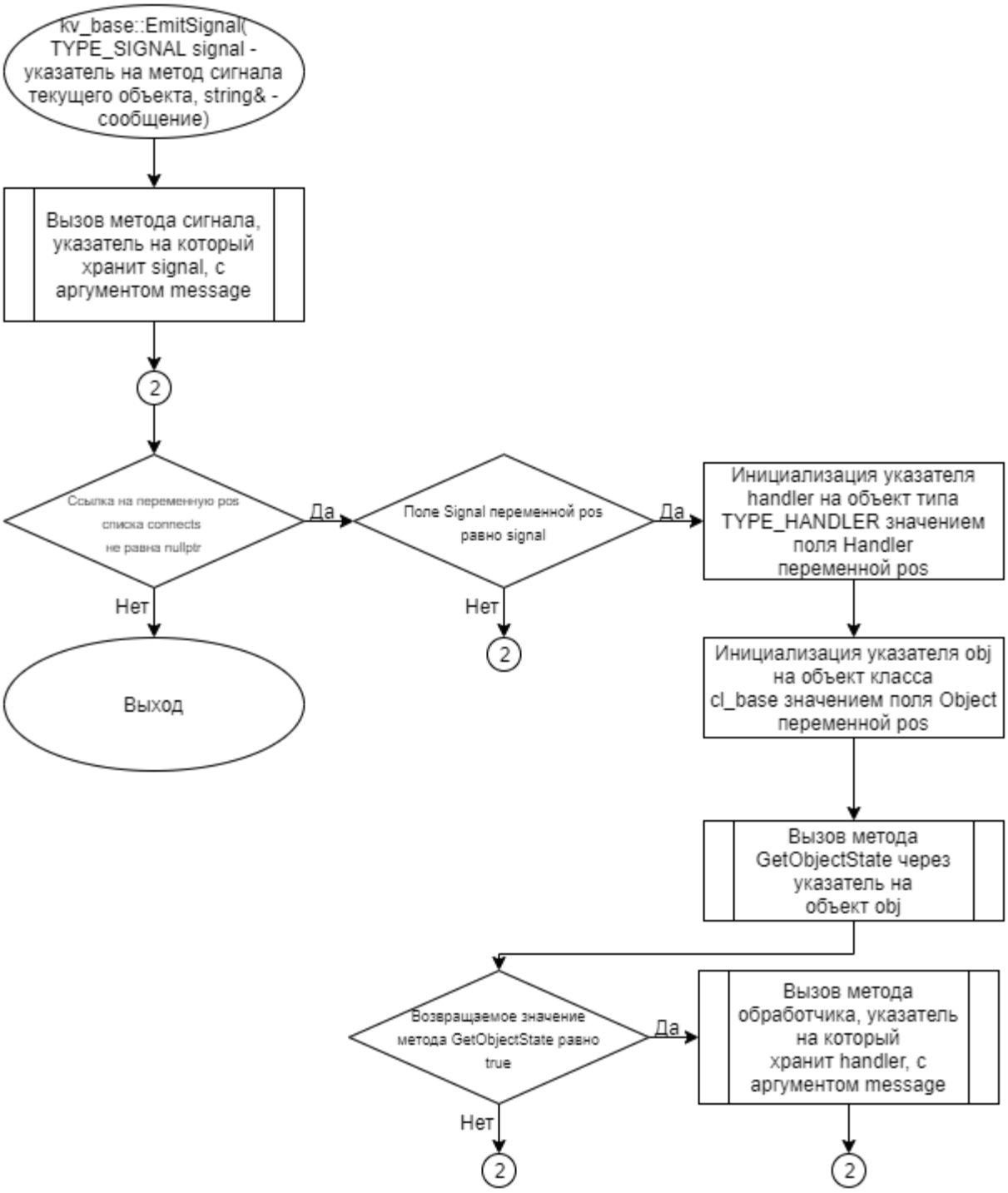

Рисунок 4 – Блок-схема алгоритма

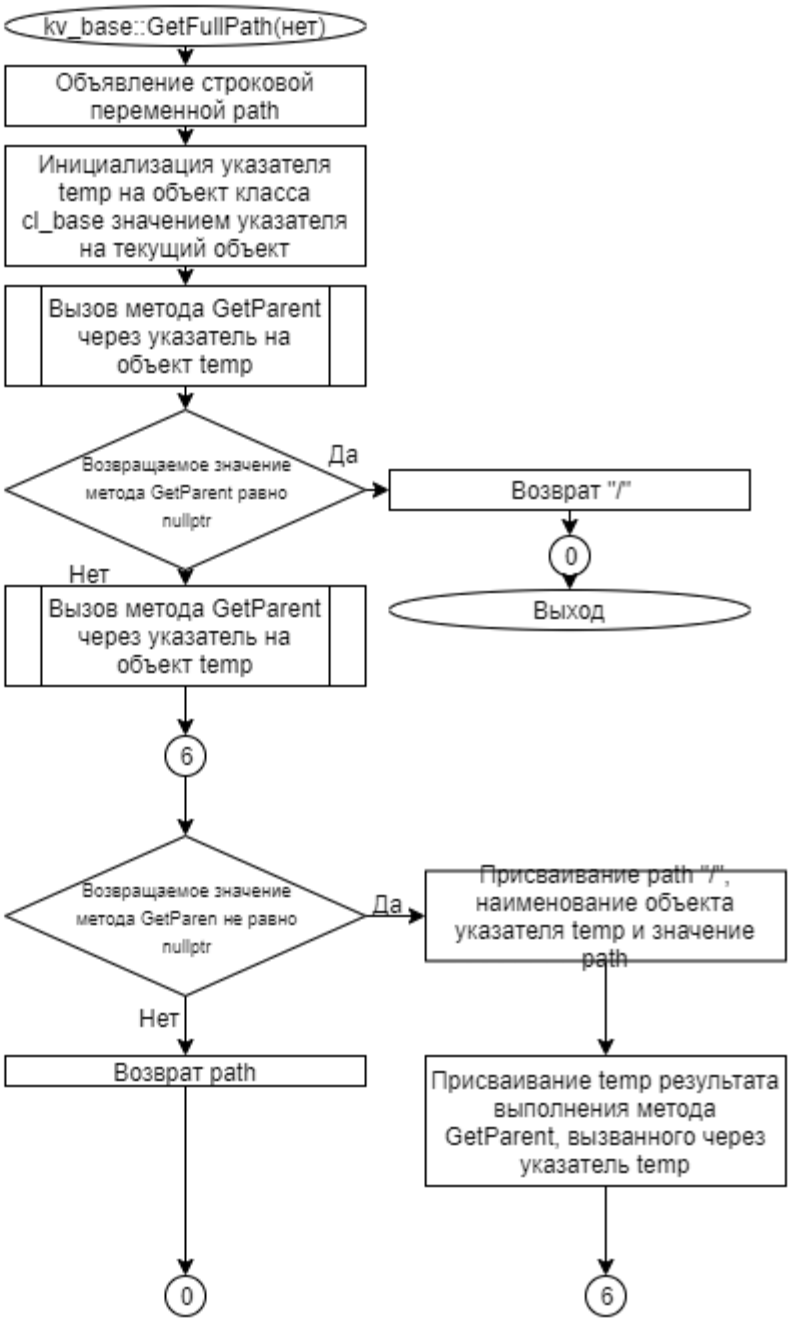

Рисунок 5 – Блок-схема алгоритма

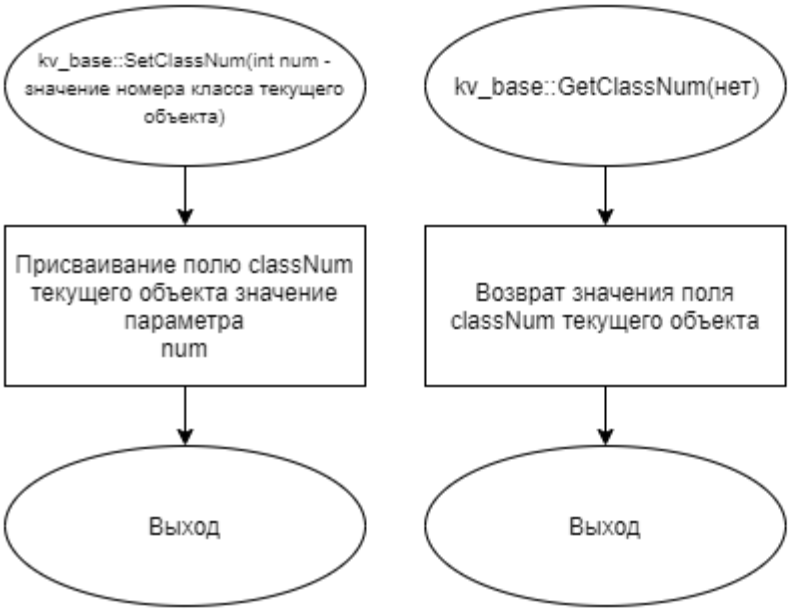

Рисунок 6 – Блок-схема алгоритма

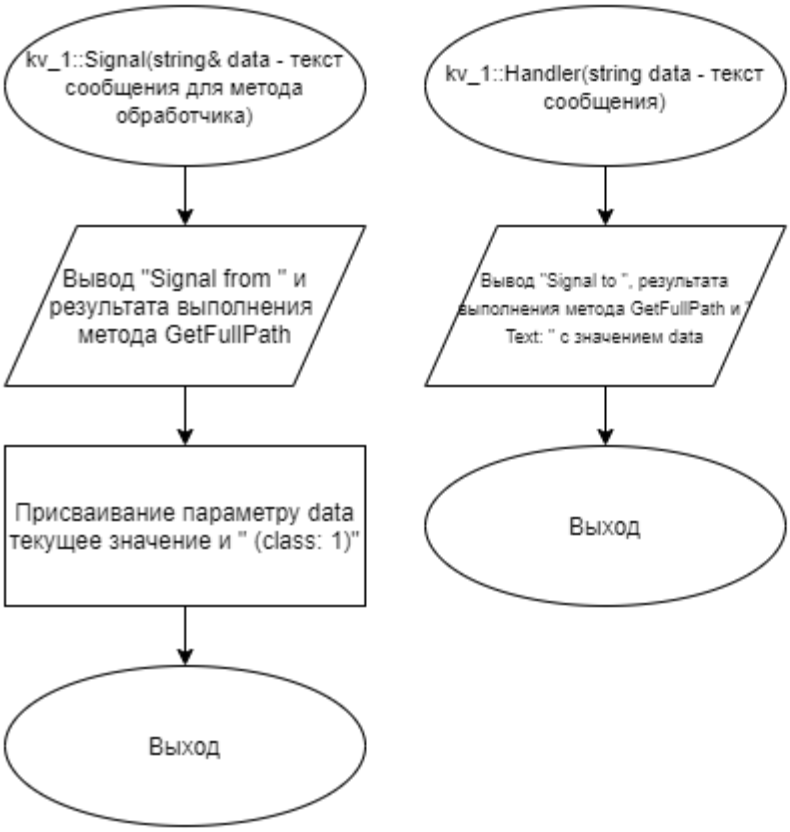

Рисунок 7 – Блок-схема алгоритма

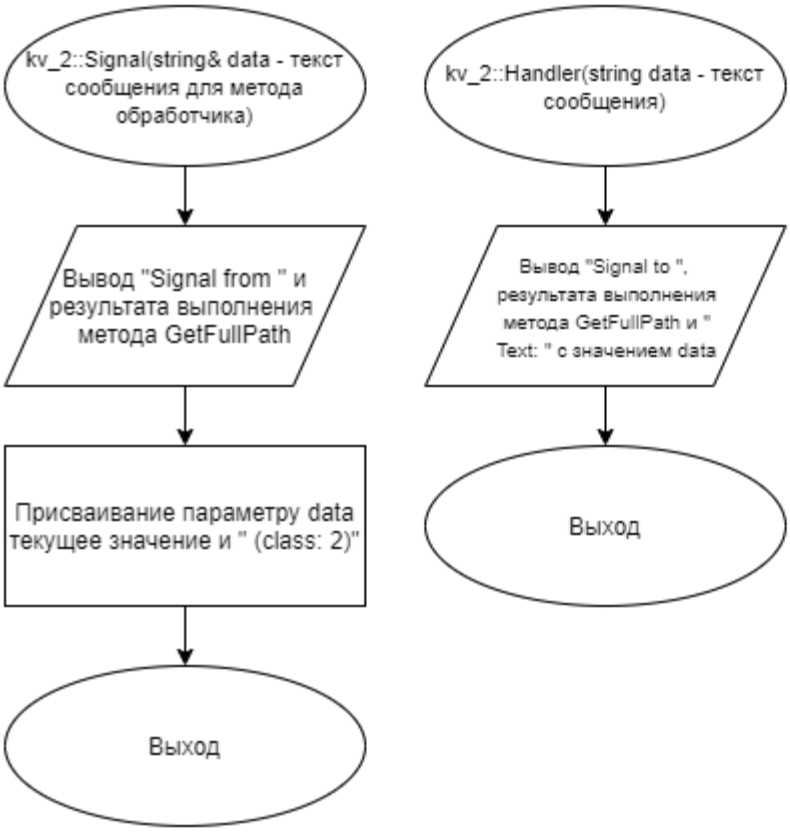

Рисунок 8 – Блок-схема алгоритма

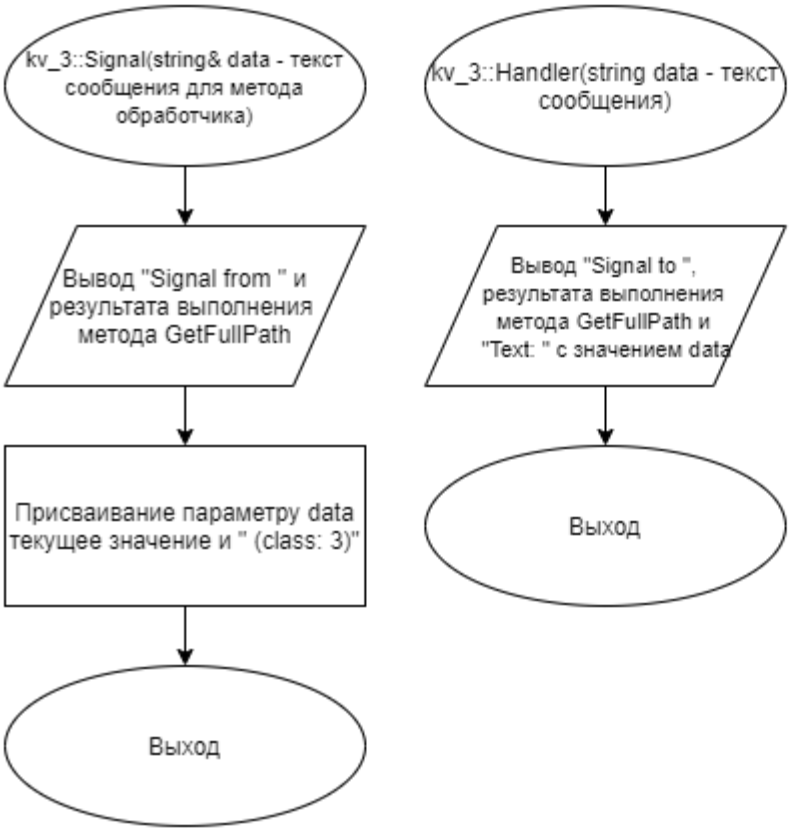

Рисунок 9 – Блок-схема алгоритма

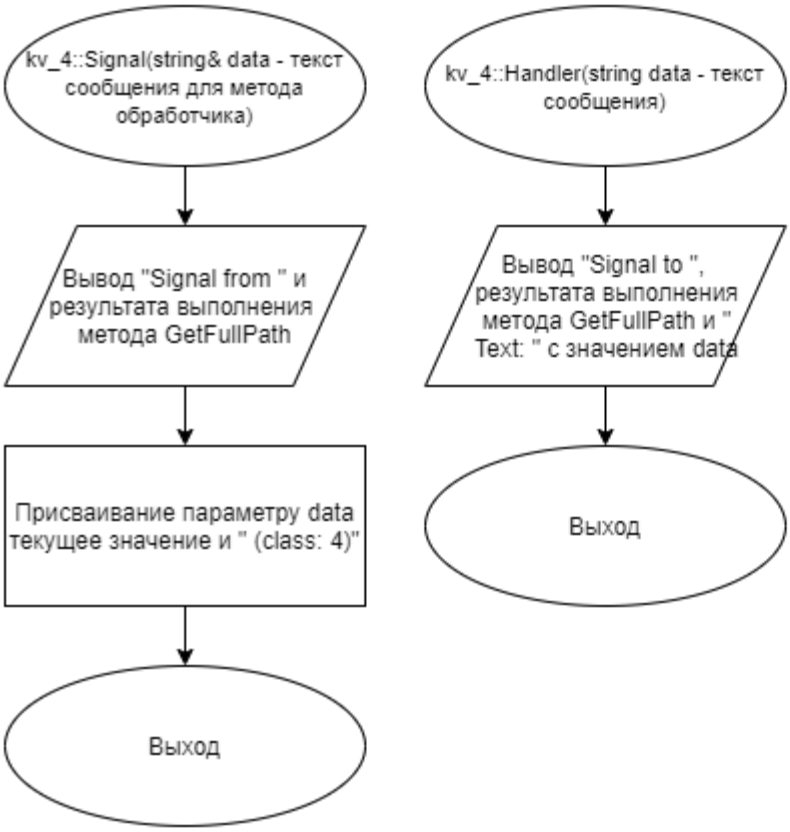

Рисунок 10 – Блок-схема алгоритма

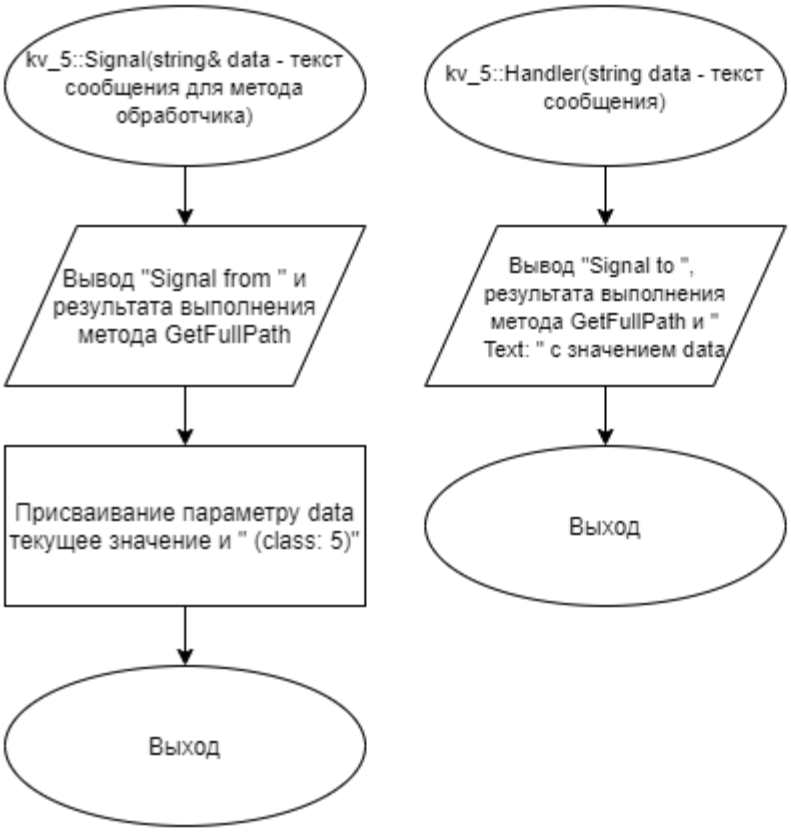

Рисунок 11 – Блок-схема алгоритма

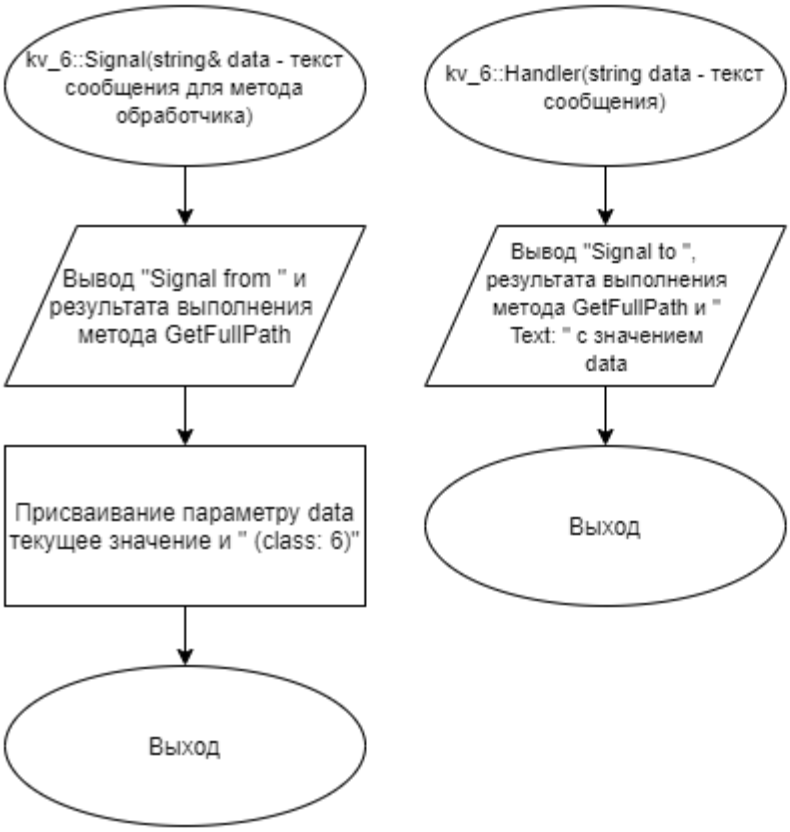

Рисунок 12 – Блок-схема алгоритма

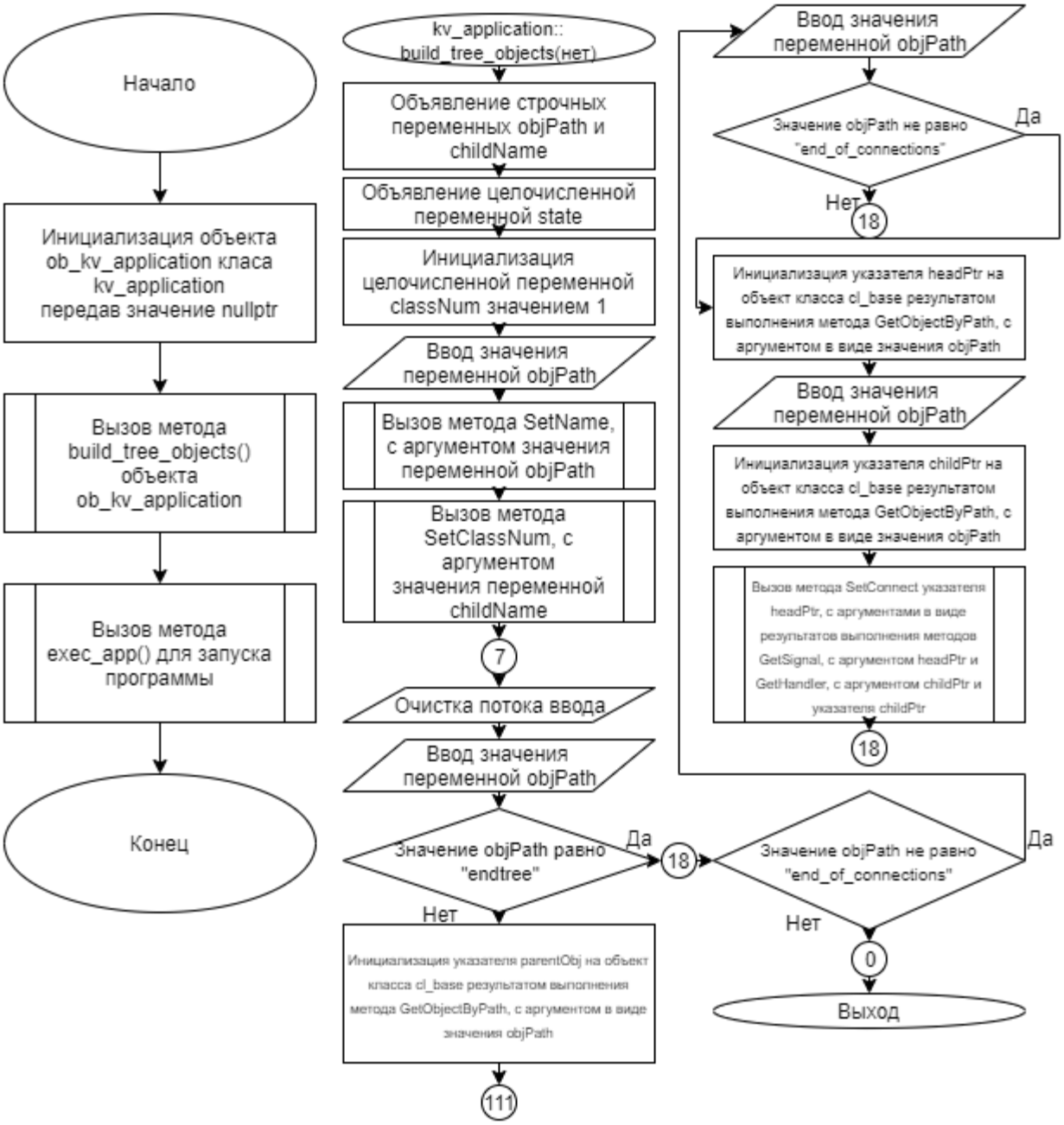

Рисунок 13 – Блок-схема алгоритма

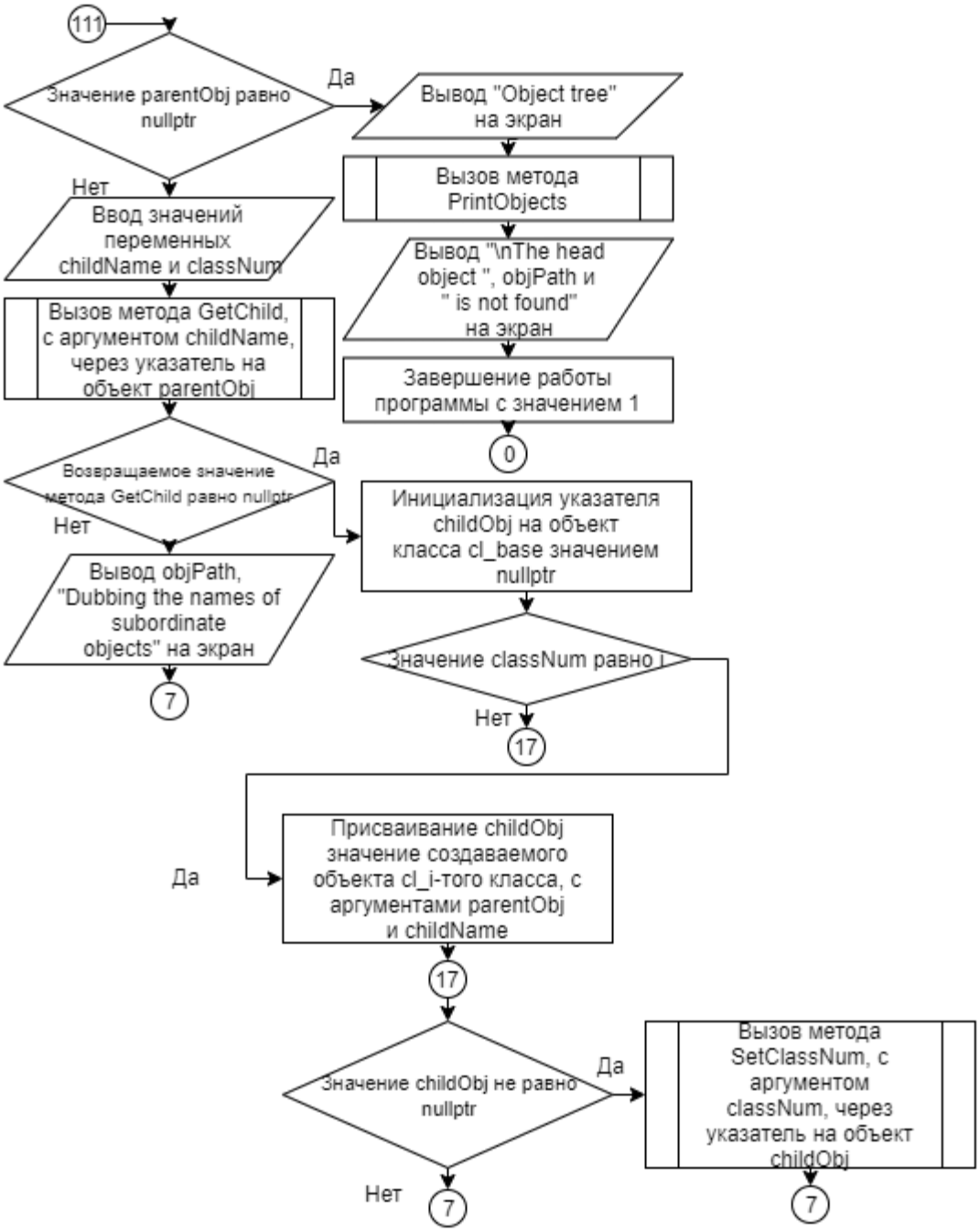

Рисунок 14 – Блок-схема алгоритма

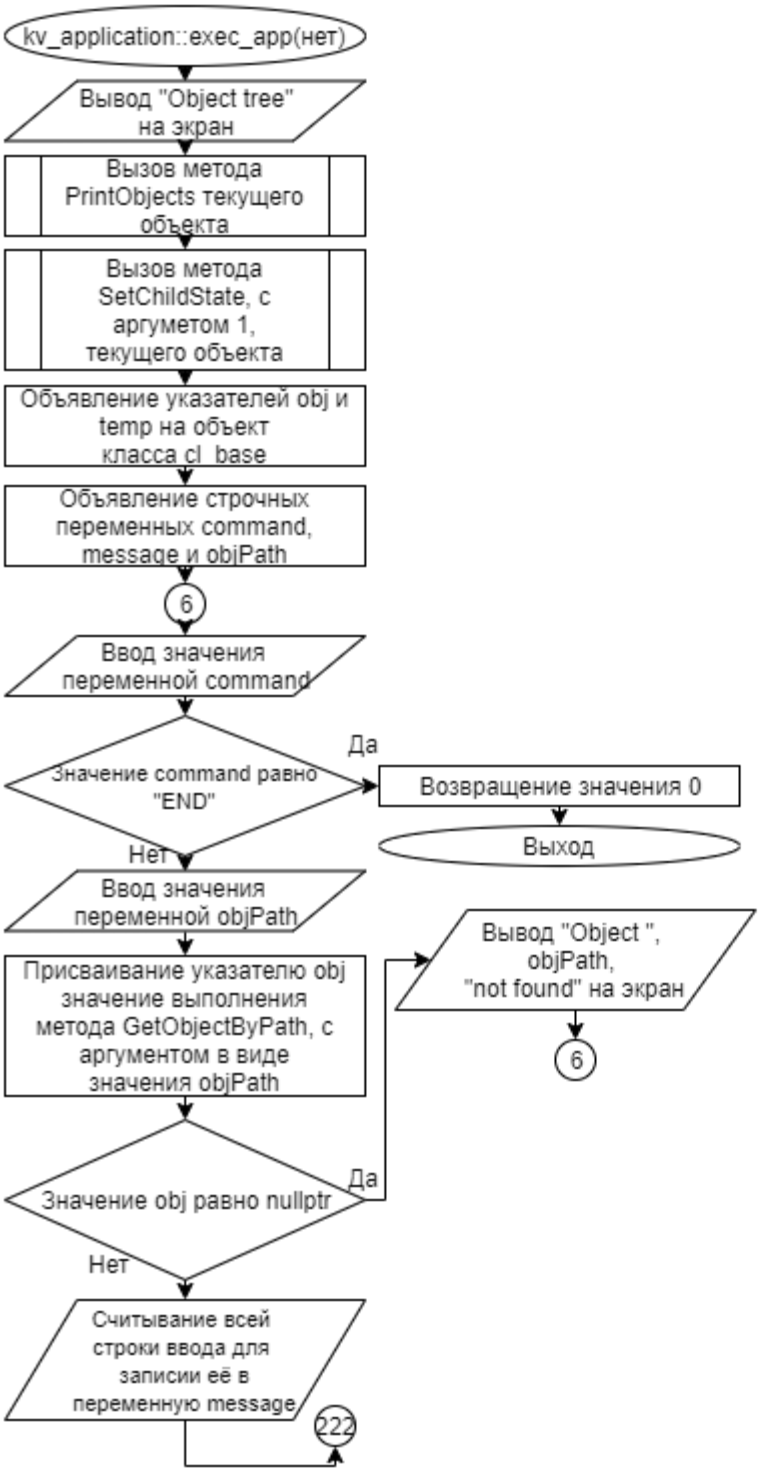

Рисунок 15 – Блок-схема алгоритма

Рисунок 16 – Блок-схема алгоритма

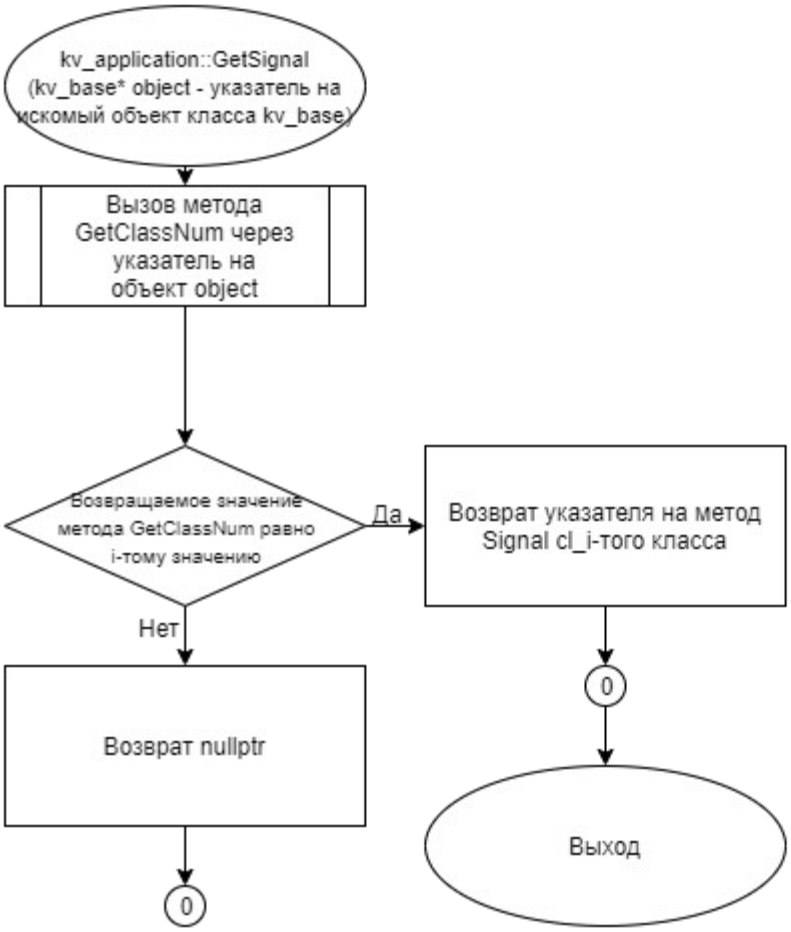

Рисунок 17 – Блок-схема алгоритма

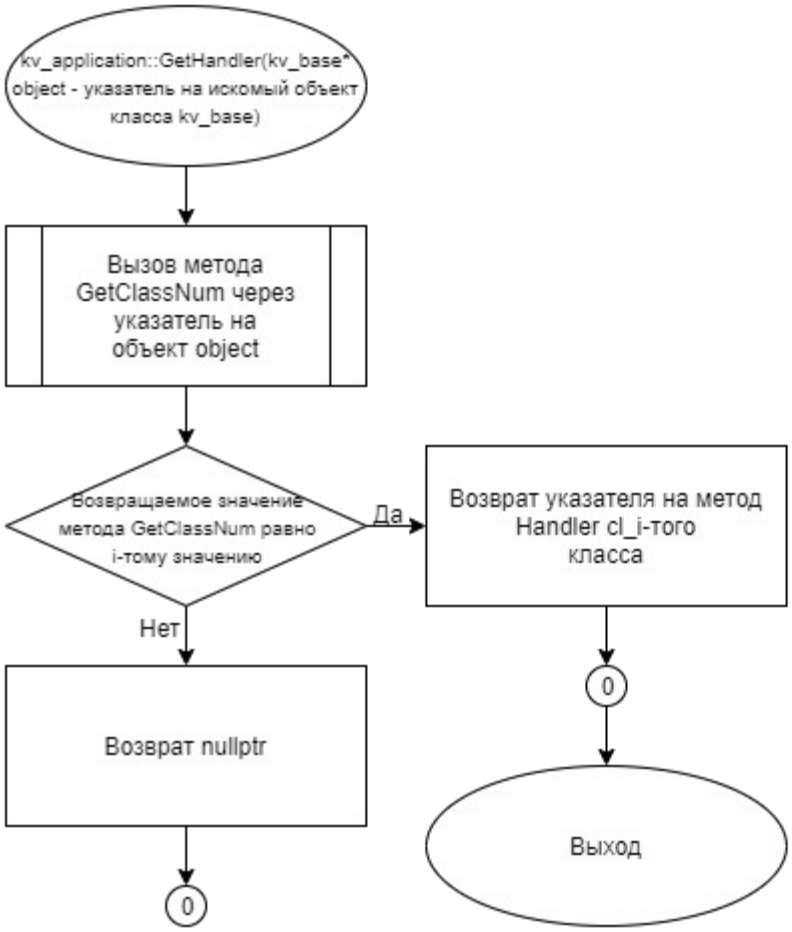

Рисунок 18 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл kv_1.cpp

Листинг 1 – kv_1.cpp

```
#include "kv_1.h" // Включает заголовочный файл kv_1.h, который содержит объявление класса kv_1.

// Определение конструктора класса kv_1. Этот конструктор инициализирует объект kv_1,
// вызывая конструктор базового класса kv_base с параметрами parent и name.
kv_1::kv_1(kv_base* parent, string name) : kv_base(parent, name){}

// Определение метода Signal класса kv_1. Этот метод принимает ссылку на строку и модифицирует ее,
// добавляя информацию о классе. Также выводит путь объекта, от которого исходит сигнал.
void kv_1::Signal(string& dat){
   cout << endl << "Signal from " + GetFullPath(); // Выводит новую строку и путь объекта, от которого исходит сигнал.
   dat = dat + " (class: 1)"; // Добавляет к строке dat информацию о том, что сигнал исходит из класса 1.
}

// Определение метода Handler класса kv_1. Этот метод принимает строку и выводит информацию о том,
// куда направлен сигнал и какой текст он несет.
void kv_1::Handler(string dat){
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat; // Выводит новую строку, путь объекта, к которому направлен сигнал, и текст сигнала.
}
```

5.2 Файл kv_1.h

Листинг 2 – kv_1.h

```
#ifndef __KV_1__H// Проверяет, определен ли макрос __KV_1__H, чтобы предотвратить повторное включение кода
#define __KV_1__H// Определяет макрос __KV_1__H, если он еще не определен
#include "kv_base.h"// Включает заголовочный файл kv_base.h, который,
```

```
вероятно, содержит объявление базового класса kv_base

class kv_1 : public kv_base{ // Объявляет класс kv_1, который наследуется от класса kv_base
  public: // Следующие члены класса являются публичными и доступны извне класса
     kv_1(kv_base* parent, string name); // Объявляет конструктор класса с двумя параметрами: указателем на kv_base и строкой
     void Signal(string& dat);// Объявляет метод Signal, который принимает ссылку на строку в качестве параметра
     void Handler(string dat);// Объявляет метод Handler, который принимает строку в качестве параметра
}; // Заканчивает условную директиву препроцессора, начатую с #ifndef
#endif
```

5.3 Файл kv_2.cpp

Листинг 3 – kv_2.cpp

```
#include "kv_2.h" // Включает заголовочный файл kv_2.h, который содержит объявление класса kv_2.

// Определение конструктора класса kv_2. Этот конструктор инициализирует объект kv_2, вызывая конструктор базового класса kv_base с параметрами parent и name.
kv_2::kv_2(kv_base* parent, string name) : kv_base(parent, name){}

// Определение метода Signal класса kv_2. Этот метод принимает ссылку на строку и модифицирует ее, добавляя информацию о классе. Также выводит путь объекта, от которого исходит сигнал.
void kv_2::Signal(string& dat){
  cout << endl << "Signal from " + GetFullPath(); // Выводит новую строку и путь объекта, от которого исходит сигнал.
  dat = dat + " (class: 2)"; // Добавляет к строке dat информацию о том, что сигнал исходит из класса 2
}

// Определение метода Handler класса kv_2. Этот метод принимает строку и выводит информацию о том, куда направлен сигнал и какой текст он несет.
void kv_2::Handler(string dat){
  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat; // Выводит новую строку, путь объекта, к которому направлен сигнал, и текст сигнала.
}
```

5.4 Файл kv_2.h

Листинг 4 – kv_2.h

```
#ifndef __KV_2__H// Проверяет, определен ли макрос __KV_2__H, чтобы предотвратить повторное включение кода
#define __KV_2__H// Определяет макрос __KV_2__H, если он еще не определен
#include "kv_base.h"// Включает заголовочный файл kv_base.h, который, вероятно, содержит объявление базового класса kv_base

class kv_2 : public kv_base{// Объявляет класс kv_2, который наследуется от класса kv_base
  public:// Следующие члены класса являются публичными и доступны извне класса
     kv_2(kv_base* parent, string name);// Объявляет конструктор класса с двумя параметрами: указателем на kv_base и строкой
     void Signal(string& dat);// Объявляет метод Signal, который принимает ссылку на строку в качестве параметра
     void Handler(string dat);// Объявляет метод Handler, который принимает строку в качестве параметра
};
#endif// Заканчивает условную директиву препроцессора, начатую с #ifndef
```

5.5 Файл kv_3.cpp

Листинг 5 – kv_3.cpp

```
#include "kv_3.h" // Включает заголовочный файл kv_3.h, который содержит объявление класса kv_3

// Определение конструктора класса kv_3. Этот конструктор инициализирует объект kv_3, вызывая конструктор базового класса kv_base с параметрами parent и name.
kv_3::kv_3(kv_base* parent, string name) : kv_base(parent, name){}

// Определение метода Signal класса kv_3. Этот метод принимает ссылку на строку и модифицирует ее, добавляя информацию о классе. Также выводит путь объекта, от которого исходит сигнал.
void kv_3::Signal(string& dat){
  cout << endl << "Signal from " + GetFullPath(); // Выводит новую строку и путь объекта, от которого исходит сигнал.
  dat = dat + " (class: 3)"; // Добавляет к строке dat информацию о том, что сигнал исходит из класса 3
}

// Определение метода Handler класса kv_3. Этот метод принимает строку и выводит информацию о том, куда направлен сигнал и какой текст он несет.
void kv_3::Handler(string dat){
  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat; // Выводит
```

```
новую строку, путь объекта, к которому направлен сигнал, и текст сигнала.
}
```

5.6 Файл kv_3.h

Листинг 6 – kv_3.h

```
#ifndef __KV_3__H// Проверяет, определен ли макрос __KV_3__H, чтобы предотвратить повторное включение кода
#define __KV_3__H// Определяет макрос __KV_3__H, если он еще не определен
#include "kv_base.h"// Включает заголовочный файл kv_base.h, который, вероятно, содержит объявление базового класса kv_base

class kv_3 : public kv_base{// Объявляет класс kv_3, который наследуется от класса kv_base
  public:// Следующие члены класса являются публичными и доступны извне класса
     kv_3(kv_base* parent, string name);// Объявляет конструктор класса с двумя параметрами: указателем на kv_base и строкой
     void Signal(string& dat);// Объявляет метод Signal, который принимает ссылку на строку в качестве параметра
     void Handler(string dat);// Объявляет метод Handler, который принимает строку в качестве параметра
};
#endif// Заканчивает условную директиву препроцессора, начатую с #ifndef
```

5.7 Файл kv_4.cpp

Листинг 7 – kv_4.cpp

```
#include "kv_4.h" // Включает заголовочный файл kv_4.h, который содержит объявление класса kv_4

// Определение конструктора класса kv_4. Этот конструктор инициализирует объект kv_4, вызывая конструктор базового класса kv_base с параметрами parent и name.
kv_4::kv_4(kv_base* parent, string name) : kv_base(parent, name){}

// Определение метода Signal класса kv_4. Этот метод принимает ссылку на строку и модифицирует ее, добавляя информацию о классе. Также выводит путь объекта, от которого исходит сигнал.
void kv_4::Signal(string& dat){
  cout << endl << "Signal from " + GetFullPath(); // Выводит новую строку и путь объекта, от которого исходит сигнал.
  dat = dat + " (class: 4)"; // Добавляет к строке dat информацию о том, что
```

```
сигнал исходит из класса 4
}

// Определение метода Handler класса kv_4. Этот метод принимает строку и выводит информацию о том, куда направлен сигнал и какой текст он несет.
void kv_4::Handler(string dat){
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat; // Выводит новую строку, путь объекта, к которому направлен сигнал, и текст сигнала.
}
```

5.8 Файл kv_4.h

Листинг 8 – kv_4.h

```
#ifndef __KV_4__H// Проверяет, определен ли макрос __KV_4__H, чтобы предотвратить повторное включение кода
#define __KV_4__H// Определяет макрос __KV_4__H, если он еще не определен
#include "kv_base.h"// Включает заголовочный файл kv_base.h, который, вероятно, содержит объявление базового класса kv_base

class kv_4 : public kv_base{// Объявляет класс kv_4, который наследуется от класса kv_base
   public:// Следующие члены класса являются публичными и доступны извне класса
      kv_4(kv_base* parent, string name);// Объявляет конструктор класса с двумя параметрами: указателем на kv_base и строкой
      void Signal(string& dat);// Объявляет метод Signal, который принимает ссылку на строку в качестве параметра
      void Handler(string dat);// Объявляет метод Handler, который принимает строку в качестве параметра
};
#endif// Заканчивает условную директиву препроцессора, начатую с #ifndef
```

5.9 Файл kv_5.cpp

Листинг 9 – kv_5.cpp

```
#include "kv_5.h" // Включает заголовочный файл kv_5.h, который содержит объявление класса kv_5

// Определение конструктора класса kv_5. Этот конструктор инициализирует объект kv_5, вызывая конструктор базового класса kv_base с параметрами parent и name.
kv_5::kv_5(kv_base* parent, string name) : kv_base(parent, name){}

// Определение метода Signal класса kv_5. Этот метод принимает ссылку на
```

```
строку и модифицирует ее, добавляя информацию о классе. Также выводит путь
объекта, от которого исходит сигнал.
void kv_5::Signal(string& dat){
   cout << endl << "Signal from " + GetFullPath(); // Выводит новую строку и
путь объекта, от которого исходит сигнал.
   dat = dat + " (class: 5)"; // Добавляет к строке dat информацию о том, что
сигнал исходит из класса 5
}

// Определение метода Handler класса kv_5. Этот метод принимает строку и
выводит информацию о том, куда направлен сигнал и какой текст он несет.
void kv_5::Handler(string dat){
   cout << endl << "Signal to " + GetFullPath() + " Text: " + dat; // Выводит
новую строку, путь объекта, к которому направлен сигнал, и текст сигнала.
}
```

5.10 Файл kv_5.h

Листинг 10 – kv_5.h

```
#ifndef __KV_5__H// Проверяет, определен ли макрос __KV_5__H, чтобы
предотвратить повторное включение кода
#define __KV_5__H// Определяет макрос __KV_5__H, если он еще не определен
#include "kv_base.h"// Включает заголовочный файл kv_base.h, который,
вероятно, содержит объявление базового класса kv_base

class kv_5 : public kv_base{// Объявляет класс kv_5, который наследуется от
класса kv_base
   public:// Следующие члены класса являются публичными и доступны извне
класса
      kv_5(kv_base* parent, string name);// Объявляет конструктор класса с
двумя параметрами: указателем на kv_base и строкой
      void Signal(string& dat);// Объявляет метод Signal, который принимает
ссылку на строку в качестве параметра
      void Handler(string dat);// Объявляет метод Handler, который принимает
строку в качестве параметра
};
#endif// Заканчивает условную директиву препроцессора, начатую с #ifndef
```

5.11 Файл kv_6.cpp

Листинг 11 – kv_6.cpp

```
#include "kv_6.h" // Включает заголовочный файл kv_6.h, который содержит
объявление класса kv_6
```

```
// Определение конструктора класса kv_6. Этот конструктор инициализирует объект kv_6, вызывая конструктор базового класса kv_base с параметрами parent и name.
kv_6::kv_6(kv_base* parent, string name) : kv_base(parent, name){}

// Определение метода Signal класса kv_4. Этот метод принимает ссылку на строку и модифицирует ее, добавляя информацию о классе. Также выводит путь объекта, от которого исходит сигнал.
void kv_6::Signal(string& dat){
  cout << endl << "Signal from " + GetFullPath(); // Выводит новую строку и путь объекта, от которого исходит сигнал.
  dat = dat + " (class: 6)"; // Добавляет к строке dat информацию о том, что сигнал исходит из класса 6
}

// Определение метода Handler класса kv_6. Этот метод принимает строку и выводит информацию о том, куда направлен сигнал и какой текст он несет.
void kv_6::Handler(string dat){
  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat; // Выводит новую строку, путь объекта, к которому направлен сигнал, и текст сигнала.
}
```

5.12 Файл kv_6.h

Листинг 12 – kv_6.h

```
#ifndef __KV_6__H// Проверяет, определен ли макрос __KV_6__H, чтобы предотвратить повторное включение кода
#define __KV_6__H// Определяет макрос __KV_6__H, если он еще не определен
#include "kv_base.h"// Включает заголовочный файл kv_base.h, который, вероятно, содержит объявление базового класса kv_base

class kv_6 : public kv_base{// Объявляет класс kv_6, который наследуется от класса kv_base
  public:// Следующие члены класса являются публичными и доступны извне класса
     kv_6(kv_base* parent, string name);// Объявляет конструктор класса с двумя параметрами: указателем на kv_base и строкой
     void Signal(string& dat);// Объявляет метод Signal, который принимает ссылку на строку в качестве параметра
     void Handler(string dat);// Объявляет метод Handler, который принимает строку в качестве параметра
};
#endif// Заканчивает условную директиву препроцессора, начатую с #ifndef
```

5.13 Файл kv_application.cpp

Листинг 13 – kv_application.cpp

```
#include "kv_application.h" // Включает заголовочный файл kv_application.h, который содержит объявление класса kv_application.

// Определение конструктора класса kv_application. Этот конструктор инициализирует объект kv_application,
// вызывая конструктор базового класса kv_base с параметром parent.
kv_application::kv_application(kv_base* parent) : kv_base(parent){}

// Определение метода build_tree_objects класса kv_application. Этот метод считывает данные для построения дерева объектов.
void kv_application::build_tree_objects(){
   string objPath, childName; // Объявление строковых переменных для пути объекта и имени ребенка.
   int state; // Объявление переменной для состояния объекта.
   int classNum = 1; // Инициализация переменной classNum значением 1.
   cin >> objPath; // Считывание пути объекта из стандартного ввода.
   SetName(objPath); // Установка имени объекта.
   SetClassNum(classNum); // Установка номера класса объекта.
   while(true){ // Начало бесконечного цикла для построения дерева объектов.
      cin.clear(); // Очистка потока ввода.
      cin >> objPath; // Считывание пути объекта из стандартного ввода.
      if (objPath == "endtree") // Проверка на ключевое слово "endtree" для завершения построения дерева.
         break; // Выход из цикла.
      kv_base*   parentObj   =   GetObjectByPath(objPath);   //   Получение родительского объекта по пути.
      if (!parentObj){ // Проверка на существование родительского объекта.
         cout << "Object tree" << endl; // Вывод сообщения о дереве объектов.
         PrintObjects(); // Печать объектов.
         cout << "\nThe head object " << objPath << " is not found"; // Вывод сообщения об отсутствии головного объекта.
         exit(1); // Выход из программы с кодом 1.
      }
      cin >> childName >> classNum; // Считывание имени ребенка и номера класса из стандартного ввода.
      if (!parentObj->GetChild(childName)){ // Проверка на отсутствие ребенка с таким именем.
         kv_base* childObj = nullptr; // Инициализация указателя на дочерний объект как nullptr.
         switch(classNum){ // Начало оператора switch для создания объектов разных классов.
            case 2: // В случае, если номер класса равен 2.
            {
               childObj = new kv_2(parentObj, childName); // Создание объекта класса kv_2.
               break; // Выход из блока switch.
            }
            case 3: // В случае, если номер класса равен 3.
            {
               childObj = new kv_3(parentObj, childName); // Создание объекта
```

```
класса kv_3.
                break; // Выход из блока switch.
            }
            case 4: // В случае, если номер класса равен 4.
            {
                childObj = new kv_4(parentObj, childName); // Создание объекта
класса kv_4.
                break; // Выход из блока switch.
            }
            case 5: // В случае, если номер класса равен 5.
            {
                childObj = new kv_5(parentObj, childName); // Создание объекта
класса kv_5.
                break; // Выход из блока switch.
            }
            case 6: // В случае, если номер класса равен 6.
            {
                childObj = new kv_6(parentObj, childName); // Создание объекта
класса kv_6.
                break; // Выход из блока switch.
            }
        }
        if (childObj != nullptr) childObj->SetClassNum(classNum); // Если
дочерний объект был создан, установить его номер класса.
     }
     else{ // Если ребенок с таким именем уже существует.
        cout << objPath << " Dubbing the names of subordinate objects" <<
endl; // Вывод сообщения о дублировании имен подчиненных объектов.
     }
  }
  while(objPath != "end_of_connections"){ // Начало цикла для установления
связей между объектами.
     cin >> objPath; // Считывание пути объекта из стандартного ввода.
     if (objPath != "end_of_connections"){ // Проверка на ключевое слово
"end_of_connections" для завершения установления связей.
        kv_base* headPtr = GetObjectByPath(objPath); // Получение указателя
на головной объект.
        cin >> objPath; // Считывание пути объекта из стандартного ввода.
        kv_base* childPtr = GetObjectByPath(objPath); // Получение указателя
на дочерний объект.
        headPtr->SetConnect(GetSignal(headPtr),                    childPtr,
GetHandler(childPtr)); // Установление связи между головным и дочерним
объектами.
     }
  }
}
// Определение метода exec_app класса kv_application. Этот метод запускает
основной цикл приложения.
int kv_application::exec_app(){
  cout << "Object tree" << endl; // Выводит сообщение "Object tree" в
стандартный вывод.
  PrintObjects(); // Вызывает метод PrintObjects для вывода текущего
состояния дерева объектов.
  SetChildState(1); // Устанавливает состояние всех дочерних объектов в 1.
  kv_base* obj; // Объявляет указатель на базовый класс kv_base для текущего
```

```
объекта.
   kv_base* temp; // Объявляет временный указатель на базовый класс kv_base.
   string command, message, objPath; // Объявляет строки для команды,
сообщения и пути объекта.
   while(true){ // Начинает бесконечный цикл.
      cin >> command; // Считывает команду из стандартного ввода.
      if (command == "END"){ // Проверяет, является ли команда "END".
         break; // Прерывает цикл, если команда "END".
      }
      cin >> objPath; // Считывает путь объекта из стандартного ввода.
      obj = GetObjectByPath(objPath); // Получает объект по указанному пути.
      if (obj == nullptr){ // Проверяет, существует ли объект.
         cout << endl << "Object " << objPath << " not found"; // Выводит
сообщение, если объект не найден.
         continue; // Продолжает цикл, пропуская оставшуюся часть тела цикла.
      }
      getline(cin, message); // Считывает оставшуюся часть строки ввода в
переменную message.
      if (command == "EMIT"){ // Проверяет, является ли команда "EMIT".
         if (obj->GetObjectState()){ // Проверяет состояние объекта.
            obj->EmitSignal(GetSignal(obj), message); // Если состояние
объекта активно, вызывает метод EmitSignal.
         }
      }
      else if (command == "SET_CONDITION") // Проверяет, является ли команда
"SET_CONDITION".
         obj->SetObjectState(stoi(message)); // Устанавливает состояние
объекта в значение, преобразованное из строки message.
      else{ // Если команда не "EMIT" и не "SET_CONDITION".
         message = message.substr(1); // Удаляет первый символ из сообщения
(предположительно пробел).
         temp = GetObjectByPath(message); // Получает объект по новому пути,
указанному в message.
         if (temp == nullptr){ // Проверяет, существует ли объект handler.
            cout << endl << "Handler object " << message << " not found"; //
Выводит сообщение, если объект handler не найден.
            continue; // Продолжает цикл, пропуская оставшуюся часть тела
цикла.
         }
         if (command == "SET_CONNECT") // Проверяет, является ли команда
"SET_CONNECT".
            obj->SetConnect(GetSignal(obj), temp, GetHandler(temp)); //
Устанавливает связь между текущим объектом и объектом handler.
         else if (command == "DELETE_CONNECT") // Проверяет, является ли
команда "DELETE_CONNECT".
            obj->DeleteConnect(GetSignal(obj), temp, GetHandler(temp)); //
Удаляет связь между текущим объектом и объектом handler.
      }
   }
   return 0; // Возвращает 0, что обычно означает успешное завершение
программы.
}

// Определение метода GetSignal класса kv_application. Этот метод возвращает
функцию сигнала для объекта.
```

```
TYPE_SIGNAL kv_application::GetSignal(kv_base* object){
   switch(object->GetClassNum()){  //  Оператор  switch,  который  выбирает
функцию сигнала на основе номера класса объекта.
      case 1: // Если номер класса равен 1.
         return  SIGNAL_D(kv_1::Signal);  //  Возвращает  указатель  на  функцию
сигнала класса kv_1.
         break; // Выход из блока switch.
      case 2: // Если номер класса равен 2.
         return  SIGNAL_D(kv_2::Signal);  //  Возвращает  указатель  на  функцию
сигнала класса kv_2.
         break; // Выход из блока switch.
      case 3: // Если номер класса равен
         return  SIGNAL_D(kv_3::Signal);//  Возвращает  указатель  на  функцию
сигнала класса kv_3.
         break;
      case 4:
         return  SIGNAL_D(kv_4::Signal);//  Возвращает  указатель  на  функцию
сигнала класса kv_4.
         break;
      case 5:
         return  SIGNAL_D(kv_5::Signal);//  Возвращает  указатель  на  функцию
сигнала класса kv_5
         break;
      case 6:
         return  SIGNAL_D(kv_6::Signal);//  Возвращает  указатель  на  функцию
сигнала класса kv_6
         break;
   }
   return nullptr;// Если ни один из кейсов не совпал, возвращается nullptr.
}
TYPE_HANDLER   kv_application::GetHandler(kv_base*   object){//   Определение
метода  GetHandler  класса  kv_application.  Этот  метод  возвращает  функцию
обработчика для объекта.
   switch(object->GetClassNum()){  //  Оператор  switch,  который  выбирает
функцию обработчика на основе номера класса объекта.
      case 1: // Если номер класса равен 1.
         return HANDLER_D(kv_1::Handler); // Возвращает указатель на функцию
обработчика класса kv_1.
         break;  //  Выход  из  блока  switch.  (Этот  break  никогда  не  будет
достигнут, так как return уже вышел из функции)

      case 2: // Если номер класса равен 2.
         return HANDLER_D(kv_2::Handler); // Возвращает указатель на функцию
обработчика класса kv_2.
         break;  //  Выход  из  блока  switch.  (Этот  break  никогда  не  будет
достигнут, так как return уже вышел из функции)

      case 3: // Если номер класса равен 3.
         return HANDLER_D(kv_3::Handler); // Возвращает указатель на функцию
обработчика класса kv_3.
         break;  //  Выход  из  блока  switch.  (Этот  break  никогда  не  будет
достигнут, так как return уже вышел из функции)

      case 4: // Если номер класса равен 4.
         return HANDLER_D(kv_4::Handler); // Возвращает указатель на функцию
```

```
обработчика класса kv_4.
        break; // Выход из блока switch. (Этот break никогда не будет
достигнут, так как return уже вышел из функции)

     case 5: // Если номер класса равен 5.
        return HANDLER_D(kv_5::Handler); // Возвращает указатель на функцию
обработчика класса kv_5.
        break; // Выход из блока switch. (Этот break никогда не будет
достигнут, так как return уже вышел из функции)

     case 6: // Если номер класса равен 6.
        return HANDLER_D(kv_6::Handler); // Возвращает указатель на функцию
обработчика класса kv_6.
        break; // Выход из блока switch. (Этот break никогда не будет
достигнут, так как return уже вышел из функции)
   }
   return nullptr; // Если ни один из кейсов не совпал, возвращается nullptr.
}
```

5.14 Файл kv_application.h

Листинг 14 – kv_application.h

```
#ifndef KV_APPLICATION_H // Проверяет, определен ли макрос KV_APPLICATION_H,
чтобы избежать повторного включения этого заголовочного файла.
#define KV_APPLICATION_H // Определяет макрос KV_APPLICATION_H, если он еще
не был определен.

#include "kv_base.h" // Включает заголовочный файл kv_base.h, который,
вероятно, содержит базовый класс kv_base.
#include "kv_1.h" // Включает заголовочный файл kv_1.h для доступа к
определениям класса kv_1.
#include "kv_2.h" // Включает заголовочный файл kv_2.h для доступа к
определениям класса kv_2.
#include "kv_3.h" // Включает заголовочный файл kv_3.h для доступа к
определениям класса kv_3.
#include "kv_4.h" // Включает заголовочный файл kv_4.h для доступа к
определениям класса kv_4.
#include "kv_5.h" // Включает заголовочный файл kv_5.h для доступа к
определениям класса kv_5.
#include "kv_6.h" // Включает заголовочный файл kv_6.h для доступа к
определениям класса kv_6.

class kv_application : public kv_base{ // Объявляет класс kv_application,
который наследуется от класса kv_base.
  public: // Следующие члены класса являются публичными и доступны извне
класса.
     kv_application(kv_base* parent); // Объявляет конструктор класса с
одним параметром: указателем на kv_base.
     void build_tree_objects(); // Объявляет метод build_tree_objects,
```

```
который, вероятно, строит дерево объектов в приложении.
     int exec_app(); // Объявляет метод exec_app, который, вероятно,
выполняет основную логику приложения и возвращает целочисленное значение.
     // KB4
     TYPE_SIGNAL GetSignal(kv_base*); // Объявляет метод GetSignal, который
принимает указатель на kv_base и возвращает значение типа TYPE_SIGNAL.
     TYPE_HANDLER GetHandler(kv_base*); // Объявляет метод GetHandler,
который принимает указатель на kv_base и возвращает значение типа
TYPE_HANDLER.
};
#endif // Заканчивает условную директиву препроцессора, начатую с #ifndef.
```

5.15 Файл kv_base.cpp

Листинг 15 – kv_base.cpp

```
#include "kv_base.h" // Подключение заголовочного файла kv_base.h, который
содержит объявление класса kv_base.

// Конструктор класса kv_base.
kv_base::kv_base(kv_base* parent, string name){
   this->parent = parent; // Инициализация указателя на родительский объект.
   this->name = name; // Инициализация имени объекта.
   if (GetParent() != nullptr) // Проверка, существует ли родительский
объект.
     GetParent()->children.push_back(this); // Если родитель существует,
добавление текущего объекта в список детей родителя.
}

// Деструктор класса kv_base.
kv_base::~kv_base(){
   for (auto child : children){ // Перебор всех дочерних объектов.
     delete child; // Удаление каждого дочернего объекта для освобождения
памяти.
   }
}

// Метод для установки нового имени объекта.
bool kv_base::SetName(string newName){
   if(GetParent() != nullptr && GetParent()->GetChild(newName) != nullptr) //
Проверка, не занято ли новое имя среди объектов-братьев.
     return false; // Если имя занято, возвращается false.
   name = newName; // Установка нового имени.
   return true; // Возвращается true, указывая на успешное изменение имени.
}

// Метод для получения имени объекта.
string kv_base::GetName() const{
   return name; // Возвращение имени объекта.
}
```

```
// Метод для получения родительского объекта.
kv_base* kv_base::GetParent() const{
   return parent; // Возвращение указателя на родительский объект.
}

// Метод для получения дочернего объекта по имени.
kv_base* kv_base::GetChild(string objName) const{
   for (auto child : children){ // Перебор всех дочерних объектов.
      if (child->GetName() == objName){ // Проверка, совпадает ли имя
дочернего объекта с заданным.
         return child; // Возвращение дочернего объекта с совпадающим именем.
      }
   }
   return nullptr; // Если объект не найден, возвращается nullptr.
}

// Метод для подсчета количества объектов с заданным именем в иерархии.
int kv_base::ObjNameCount(string objName){
   int count = 0; // Инициализация счетчика.
   if(GetName() == objName) // Проверка, совпадает ли имя текущего объекта с
заданным.
      count++; // Увеличение счетчика.
   for (auto child : children){ // Перебор всех дочерних объектов.
      count += child->ObjNameCount(objName); // Рекурсивный подсчет объектов
с заданным именем в поддереве.
   }
   return count; // Возвращение общего количества объектов с заданным именем.
}

// Метод для проверки уникальности объекта с заданным именем в иерархии.
kv_base* kv_base::CheckingObjUniq(string objName){
   if (ObjNameCount(objName) != 1) // Если объект с таким именем не уникален
(их больше одного).
      return nullptr; // Возвращается nullptr.
   return SearchObjOnBranch(objName); // Иначе выполняется поиск объекта в
ветке.
}

// Метод для поиска объекта с заданным именем в ветке.
kv_base* kv_base::SearchObjOnBranch(string objName){
   if (GetName() == objName) // Если имя текущего объекта совпадает с
заданным.
      return this; // Возвращается текущий объект.
   for (auto child : children){ // Перебор всех дочерних объектов.
      kv_base* subChild = child->SearchObjOnBranch(objName); // Рекурсивный
поиск объекта в поддереве.
      if (subChild) // Если объект найден.
         return subChild; // Возвращается найденный объект.
   }
   return nullptr; // Если объект не найден, возвращается nullptr.
}

// Метод для поиска объекта с заданным именем во всем дереве.
kv_base* kv_base::SearchObjOnTree(string objName){
   return GetRoot()->CheckingObjUniq(objName); // Поиск уникального объекта
```

```
начинается с корня дерева.
}

// Метод для печати иерархии объектов.
void kv_base::PrintObjects(int spaces) const{
   cout << GetName(); // Печать имени текущего объекта.
   if (!children.empty()){ // Если у текущего объекта есть дочерние объекты.
      for (auto child : children){ // Перебор всех дочерних объектов.
         cout << endl; // Печать символа новой строки.
         for (int i = 0; i < spaces; i++) // Печать пробелов для отступа.
            cout << " ";
         child->PrintObjects(spaces+4);   //   Рекурсивная   печать   дочерних
объектов с увеличенным отступом.
      }
   }
}

// Метод для печати состояний объектов в иерархии.
void kv_base::PrintObjectsStates(int spaces) const{
   cout << GetName(); // Печать имени текущего объекта.
   cout << (GetObjectState() ? " is ready" : " is not ready"); // Печать
состояния объекта.
   if (!children.empty()){ // Если у текущего объекта есть дочерние объекты.
      for (auto child : children){ // Перебор всех дочерних объектов.
         cout << endl; // Печать символа новой строки.
         for (int i = 0; i < spaces; i++) // Печать пробелов для отступа.
            cout << " ";
         child->PrintObjectsStates(spaces+4); // Рекурсивная печать состояний
дочерних объектов с увеличенным отступом.
      }
   }
}

// Метод для установки состояния объекта и его распространения на дочерние
объекты.
void kv_base::SetObjectState(bool state){
   if (GetParent() && !GetParent()->GetObjectState()){ // Если у объекта есть
родитель и состояние родителя не "готово".
      this->state = false; // Установка состояния текущего объекта в "не
готово".
   }
   else{
      this->state = state; // В противном случае установка состояния,
переданного в метод.
   }
   if (!state){ // Если состояние установлено в "не готово".
      for (auto child : children){ // Перебор всех дочерних объектов.
         child->SetObjectState(state); // Рекурсивная установка состояния в
"не готово" для всех дочерних объектов.
      }
   }
}

// Метод для получения состояния объекта.
bool kv_base::GetObjectState() const{
```

```
   return state; // Возвращение состояния объекта.
}

// Метод для получения корневого объекта дерева.
kv_base* kv_base::GetRoot(){
   kv_base* obj = this; // Начало с текущего объекта.
   while(obj->GetParent()){ // Пока у объекта есть родитель.
      obj = obj->GetParent(); // Перемещение вверх по дереву.
   }
   return obj; // Возвращение корневого объекта.
}
// Метод для получения объекта по заданному пути.
kv_base* kv_base::GetObjectByPath(string path){
   kv_base* currentObj = this; // Начинаем с текущего объекта.
   string nextObjName; // Строка для хранения следующего имени объекта в
пути.
   if (path.substr(0,1) == "/"){ // Если путь начинается с "/", это
абсолютный путь.
      if (path.substr(1,1) == "/"){ // Если путь начинается с "//", это
специальный путь для поиска в дереве.
         return SearchObjOnTree(path.substr(2)); // Используем метод поиска
объекта в дереве.
      }
      currentObj = GetRoot(); // Получаем корневой объект дерева.
      path = path.substr(1); // Обрезаем первый символ пути.
   }
   else if (path.substr(0,1) == "."){ // Если путь начинается с ".", это
относительный путь.
      return path == "." ? currentObj : CheckingObjUniq(path.substr(1)); //
Возвращаем текущий объект или проверяем уникальность объекта в ветке.
   }
   stringstream streamPath(path); // Создаем потоковый объект для работы со
строкой пути.
   while(getline(streamPath, nextObjName, '/')){ // Разбиваем путь на части
по символу "/".
      currentObj = currentObj->GetChild(nextObjName); // Получаем дочерний
объект по имени следующего элемента пути.
      if (!currentObj){ // Если такого дочернего объекта нет,
         return nullptr; // возвращаем nullptr.
      }
   }
   return currentObj; // Возвращаем найденный объект.
}

// Метод для изменения родительского объекта текущего объекта.
bool kv_base::ChangeHeadObj(kv_base* newHead){
   if (GetParent()){ // Если у текущего объекта есть родитель,
      for (auto i = (GetParent()->children).begin(); i != (GetParent()-
>children).end(); i++){ // ищем текущий объект среди детей родителя.
   if (*i == this){ // Когда находим,
      (GetParent()->children).erase(i); // удаляем его из списка детей
родителя.
      break; // Прерываем цикл.
   }
   }
   this->parent = newHead; // Устанавливаем нового родителя для текущего
```

```
объекта.
   (GetParent()->children).push_back(this);  //  Добавляем  текущий  объект  в
список детей нового родителя.
   return true; // Возвращаем true, так как операция прошла успешно.
   }
   return false; // Если у текущего объекта нет родителя, возвращаем false.
}

// Метод для удаления дочернего объекта по имени.
bool kv_base::DeleteSubObj(string objName){
   kv_base* subObj = GetChild(objName); // Получаем дочерний объект по имени.
   for (auto i = children.begin(); i != children.end(); i++){ // Ищем этот
объект среди детей текущего объекта.
      if (*i == subObj){ // Когда находим,
         children.erase(i); // удаляем его из списка детей.
         delete subObj; // Освобождаем память, занимаемую дочерним объектом.
         return true; // Возвращаем true, так как объект успешно удален.
      }
   }
   return false; // Если объект не найден, возвращаем false.
}

//  Метод  для  установки  состояния  текущего  объекта  и  всех  его  дочерних
объектов.
void kv_base::SetChildState(int state){
   SetObjectState(state); // Устанавливаем состояние для текущего объекта.
   for (auto child : children){ // Для каждого дочернего объекта
      child->SetChildState(state);   //   рекурсивно   устанавливаем   то   же
состояние.
   }
}

// Метод для установки соединения между сигналом и обработчиком объекта.
void  kv_base::SetConnect(TYPE_SIGNAL  signal,  kv_base*  object,  TYPE_HANDLER
handler){
   for(auto& pos : connects){ // Перебираем все существующие соединения.
      if (pos.Signal==signal && pos.Handler==handler && pos.Object==object)
{ // Если такое соединение уже существует,
         return; // выходим из метода.
      }
}
   o_sh  obj;  //  Создаем  временный  объект  для  хранения  информации  о
соединении.
   obj.Signal = signal; // Устанавливаем сигнал.
   obj.Handler = handler; // Устанавливаем обработчик.
   obj.Object = object; // Устанавливаем объект.
   connects.push_back(obj); // Добавляем информацию о соединении в список
соединений.
}

// Метод для удаления соединения между сигналом и обработчиком.
void    kv_base::DeleteConnect(TYPE_SIGNAL    signal,    kv_base*    object,
TYPE_HANDLER handler){
   for (auto i = connects.begin(); i != connects.end(); i++){ // Перебираем
все соединения.
      if      ((*i).Signal==signal      &&      (*i).Handler==handler      &&
```

```
(*i).Object==object){ // Если находим соединение, которое нужно удалить,
        connects.erase(i); // удаляем его из списка.
        return; // Выходим из метода.
     }
   }
}

// Метод для отправки сигнала всем подключенным обработчикам.
void kv_base::EmitSignal(TYPE_SIGNAL signal, string& message){
   (this->*signal)(message); // Вызываем метод, соответствующий сигналу, для
текущего объекта.
   for(auto& pos : connects){ // Перебираем все соединения.
      if  (pos.Signal==signal){  //  Если  сигнал  соединения  совпадает  с
отправляемым,
        TYPE_HANDLER handler = pos.Handler; // получаем обработчик.
        kv_base* obj = pos.Object; // Получаем объект.
        if (obj->GetObjectState()){ // Если состояние объекта позволяет,
           (obj->*handler)(message); // вызываем обработчик.
        }
     }
   }
}

// Метод для получения полного пути текущего объекта в иерархии.
string kv_base::GetFullPath(){
   string path; // Строка для хранения пути.
   kv_base* temp = this; // Начинаем с текущего объекта.
   if(temp->GetParent() == nullptr){ // Если у объекта нет родителя,
      return "/"; // возвращаем "/", так как это корневой объект.
   }
   while(temp->GetParent() != nullptr){ // Пока у объекта есть родитель,
      path = "/" + temp->GetName() + path; // добавляем имя объекта к пути.
      temp = temp->GetParent(); // Переходим к родительскому объекту.
   }
   return path; // Возвращаем полный путь.
}

// Метод для установки номера класса текущего объекта.
void kv_base::SetClassNum(int num){
   classNum = num; // Присваиваем номер класса.
}

// Метод для получения номера класса текущего объекта.
int kv_base::GetClassNum(){
   return classNum; // Возвращаем номер класса.
}
```

5.16 Файл kv_base.h

Листинг 16 – kv_base.h

```
#ifndef KV_BASE_H // Проверяет, определен ли макрос KV_BASE_H, чтобы избежать повторного включения этого заголовочного файла.
#define KV_BASE_H // Определяет макрос KV_BASE_H, если он еще не был определен.

#include <iostream> // Включает стандартную библиотеку ввода-вывода.
#include <string> // Включает стандартную библиотеку строк.
#include <sstream> // Включает стандартную библиотеку строковых потоков.
#include <vector> // Включает стандартную библиотеку векторов.
using namespace std; // Использует стандартное пространство имен, чтобы избежать необходимости предварять стандартные типы и функции префиксом std::.

class kv_base; // Предварительное объявление класса kv_base.

// Макросы для приведения типов функций-членов класса kv_base, используемых как сигналы и обработчики.
#define SIGNAL_D(signal_f)(TYPE_SIGNAL)(&signal_f) // Макрос для создания указателя на функцию-член класса kv_base, которая используется как сигнал.
#define HANDLER_D(handler_f)(TYPE_HANDLER)(&handler_f) // Макрос для создания указателя на функцию-член класса kv_base, которая используется как обработчик.

// Определение типов для указателей на функции-члены класса kv_base.
typedef void (kv_base :: *TYPE_SIGNAL)(string&); // Определяет тип TYPE_SIGNAL для указателя на функцию-член класса kv_base, которая принимает ссылку на строку.
typedef void (kv_base :: *TYPE_HANDLER)(string); // Определяет тип TYPE_HANDLER для указателя на функцию-член класса kv_base, которая принимает строку.

class kv_base{ // Объявление класса kv_base.
   string name; // Приватный член данных для хранения имени объекта.
   kv_base* parent; // Приватный член данных для хранения указателя на родительский объект.
   vector<kv_base*> children; // Приватный член данных для хранения вектора указателей на дочерние объекты.
   bool state = false; // Приватный член данных для хранения состояния объекта, инициализированного как false.
   int classNum; // Приватный член данных для хранения номера класса объекта.
   struct o_sh{ // Вложенная структура для хранения связей между сигналами и обработчиками.
      TYPE_SIGNAL Signal; // Указатель на функцию-член, используемую как сигнал.
      kv_base* Object; // Указатель на объект, с которым связан сигнал или обработчик.
      TYPE_HANDLER Handler; // Указатель на функцию-член, используемую как обработчик.
   };
   vector<o_sh> connects; // Приватный член данных для хранения вектора
```

```
связей между сигналами и обработчиками.
   public: // Публичные члены класса kv_base.
      kv_base(kv_base* parent, string name = "Base_object"); // Конструктор с параметрами для родителя и имени, с именем по умолчанию "Base_object".
      ~kv_base(); // Деструктор класса.
      bool SetName(string newName); // Функция для установки нового имени объекта.
      string GetName() const; // Функция для получения имени объекта.
      kv_base* GetParent() const; // Функция для получения указателя на родительский объект.
      kv_base* GetChild(string objName) const; // Функция для получения указателя на дочерний объект по имени.
      int ObjNameCount(string objName); // Функция для подсчета количества объектов с заданным именем.
      kv_base* CheckingObjUniq(string objName); // Функция для проверки уникальности объекта с заданным именем.
      kv_base* SearchObjOnBranch(string objName); // Функция для поиска объекта по ветке.
      kv_base* SearchObjOnTree(string objName); // Функция для поиска объекта по всему дереву объектов.
      void PrintObjects(int spaces = 4) const; // Функция для печати объектов с заданным количеством пробелов для отступа.
      void PrintObjectsStates(int spaces = 4) const; // Функция для печати состояний объектов.
      void SetObjectState(bool state); // Функция для установки состояния объекта.
      bool GetObjectState() const; // Функция для получения состояния объекта.
      kv_base* GetRoot(); // Функция для получения корневого объекта дерева.
      kv_base* GetObjectByPath(string path); // Функция для получения объекта по пути.
      bool ChangeHeadObj(kv_base* newHead); // Функция для изменения головного объекта.
      bool DeleteSubObj(string objName); // Функция для удаления подобъекта по имени.
      void SetChildState(int); // Функция для установки состояния дочернего объекта.
      void SetConnect(TYPE_SIGNAL, kv_base*, TYPE_HANDLER); // Функция для установки связи между сигналом и обработчиком.
      void DeleteConnect(TYPE_SIGNAL, kv_base*, TYPE_HANDLER); // Функция для удаления связи между сигналом и обработчиком.
      void EmitSignal(TYPE_SIGNAL, string&); // Функция для генерации сигнала.
      string GetFullPath(); // Функция для получения полного пути объекта.
      void SetClassNum(int); // Функция для установки номера класса объекта.
      int GetClassNum(); // Функция для получения номера класса объекта.
};
#endif // Заканчивает условную директиву препроцессора, начатую с #ifndef.
```

5.17 Файл main.cpp

Листинг 17 – main.cpp

```
#include "kv_application.h" // Включает заголовочный файл kv_application.h, который предположительно содержит объявление класса kv_application и его методов.

int main(){ // Определение главной функции программы, с которой начинается выполнение любой C++ программы.
  kv_application ob_kv_application(nullptr); // Создание объекта ob_kv_application класса kv_application. Конструктору передается nullptr, что может означать отсутствие родительского объекта или другой контекст в зависимости от реализации конструктора.
  ob_kv_application.build_tree_objects(); // Вызов метода build_tree_objects() для объекта ob_kv_application. Этот метод, вероятно, строит дерево объектов или инициализирует структуры данных, необходимые для работы приложения.
  return ob_kv_application.exec_app(); // Вызов метода exec_app() для объекта ob_kv_application, который запускает основной цикл приложения. Возвращаемое значение этого метода используется как код возврата для операционной системы, где 0 обычно означает успешное заверш<span class="last-character"></span>
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 26.

Таблица 26 – Результат тестирования программы

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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 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).