Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

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

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

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

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

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

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

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

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

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

4. Конец цикла.

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

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

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

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

Каждый метод сигнала с новой строки выводит:

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

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

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

Каждый метод обработчика с новой строки выводит:

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

Моделировать работу системы, которая выполняет следующие команды с параметрами:

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

целевого объекта» – устанавливает связь;

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

Реализовать алгоритм работы системы:

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

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

1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве

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

После ввода состава дерева иерархии построчно вводится:

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

Ввод информации для построения связей завершается строкой, которая содержит:

```
«end_of_connections»
```

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

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

Команда END присутствует обязательно.

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

Если не найден объект по координате:

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

Если не найден целевой объект по координате:

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

Пример ввода:

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

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

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

Если отработал метод обработчика:

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

Пример вывода:

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

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

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- typedef - определение новых типов данных;
- struct - структура;
- #define - параметризированное макроопределение препроцессора.

Класс base:

- свойства/поля:
    - поле вектор для хранения установленных связей:
        - наименование — connects;
        - тип — vector;
        - модификатор доступа — private;
- функционал:
    - метод setconnect — установка связи между сигналом текущего объекта и обработчикос целевого объекта;
    - метод removeconnect — удаление связи между сигналом текущего объекта и обработчиком целевого объекта;
    - метод emitsignal — выдача сигналов от текущего объекта с передачей строковой переменной;
    - метод getabsoluteway — получение абсолютного пути объекта;
    - метод setstatebranch — установка объекта и его потомкам значения готовности;
    - метод getclassnum — возврат номера класса.

Класс app:

- функционал:
    - метод signal — сигнал;
    - метод handler — обработчиком;

    - метод getclassnum — возврат номера класса.

Класс cl_2:

- функционал:
    - метод signal — сигнал;
    - метод handler — обработчик;
    - метод getclassnum — возврат номера класса.

Класс cl_3:

- функционал:
    - метод signal — сигнал;
    - метод handler — обработчик;
    - метод getclassnum — возврат номера класса.

Класс cl_4:

- функционал:
    - метод signal — сигнал;
    - метод handler — обработчик;
    - метод getclassnum — возврат номера класса.

Класс cl_5:

- функционал:
    - метод signal — сигнал;
    - метод handler — обработчик;
    - метод getclassnum — возврат номера класса.

Класс cl_6:

- функционал:
    - метод signal — сигнал;
    - метод handler — обработчик;
    - метод getclassnum — возврат номера класса.

Таблица 1 – Иерархия наследования классов

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

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода setconnect класса base

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

Параметры: TYPE_SIGNAL signalptr, base* purposeptr, TYPE_HANDLER handlerptr.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода setconnect класса base

№	Предикат	Действия	№ перехода
1		инициализация переменной i типа int	2
2	i<размера вектора connects		3
			4
3	переданные параметры совпадают с полями i-ой связи вектора connects		∅
		i++	2
4		создание объекта структуры connect с помощью оператора new и присваивание указателю newconnect адрес этого объекта	5
5		присваивание полю signalptr объекта показателю newconnect значение параметра signalptr	6

№	Предикат	Действия	№ перехода
6		присваивание полю purposeptr объекта по указателю newconnect значение параметра purposeptr	7
7		присваивание полю handlerptr объекта показателю newconnect значение параметра handlerptr	8
8		добавление указателя newconnect в вектор connects	∅

3.2 Алгоритм метода removeconnect класса base

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

Параметры: TYPE_SIGNAL signalptr, base* purposeptr, TYPE_HANDLER handlerptr.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода removeconnect класса base

№	Предикат	Действия	№ перехода
1		инициализация переменной i типа int со значением 0	2
2	i<размера вектора connects		3
			∅
3	переданные параметры совпадают с полями i-ой связи вектора connects	вызов деструктора для объекта по i-му указателю вектора connects	4
		i++	2
4		удаление i-го элемента вектора connects	∅

3.3 Алгоритм метода emitsignal класса base

Функционал: выдача сигналов от текущего объекта с передачей строковой переменной.

Параметры: TYPE_SIGNAL signalptr, string &command.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода emitsignal класса base

№	Предикат	Действия	№ перехода
1	свойство state = 0		∅
			2
2		вызов метода сигнала по указателю signalptr с параметром command	3
3		инициализация переменной i типа int со значением 0	4
4	i< размера свойства connects		5
			∅
5	свойство signalptr i-ой связи вектора connects = signalptr		7
			6
6		i++	4
7		инициализация указателя на метод обработчика handlerptr значением свойства handler i-го объекта вектора connects	8
8		инициализация указателя purposeptr на объект класса base значением свойства purposeptr i-го объекта вектора connects	9

№	Предикат	Действия	№ перехода
9	свойство state объекта по указателю purposeptr не равно 0	вызов метода обработчика по указателю handlerptr объекта по указателю purposeptr с параметром command	6
			6

3.4 Алгоритм метода getabsoluteway класса base

Функционал: получение абсолютного пути объекта.

Параметры: none.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода getabsoluteway класса base

№	Предикат	Действия	№ перехода
1		объявление строки result	2
2		объявление стека строк stack	3
3		инициализация указателя rootptr на объект класса base адресом текущего объекта	4
4	результат вызова метода getparent объекта по указателю rootptr не равен нулевому указателю	добавление в stack результата вызова метода getname объекта по указателю rootptr	5
			6
5		присваивание rootptr результат вызова метода getparent объекта по указателю rootptr	4
6	stack не пустой	добавление в строку result символа "/" и строку на вершине стека stack	7
			8
7		удаление элемента на вершине стека stack	6

№	Предикат	Действия	№ перехода
8	строка result пустая	возврат "/"	∅
		возврат result	∅

3.5 Алгоритм метода setstatebranch класса base

Функционал: установка объекта и его потомкам значение готовности.

Параметры: int newstate.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода setstatebranch класса base

№	Предикат	Действия	№ перехода
1	у объекта есть родитель и его свойство state = 0		∅
		вызов метода setstate с параметром newstate	2
2		инициализация переменной i типа int со значением 0	3
3	i< размера вектора descens	вызов метода set_state_branch объекта по i-му указателю вектора descens с параметром newstate	4
			∅
4		i++	3

3.6 Алгоритм метода signal класса app

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода signal класса app

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вызова метода getabsoluteway()	2
2		добавление в конец строки message " (class: {результат вызова метода getclassnumber приведенный к строке})"	∅

3.7 Алгоритм метода handler класса app

Функционал: обработчиком.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода handler класса app

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

3.8 Алгоритм метода getclassnum класса app

Функционал: возврат номера класса.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода getclassnum класса app

№	Предикат	Действия	№ перехода
1		возврат 1	∅

3.9 Алгоритм метода signal класса cl_2

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода signal класса cl_2

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вызова метода getabsoluteway()	2
2		добавление в конец строки message " (class: {результат вызова метода getclassnumber приведенный к строке})"	∅

3.10 Алгоритм метода handler класса cl_2

Функционал: обработчик.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода handler класса cl_2

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

3.11 Алгоритм метода getclassnum класса cl_2

Функционал: возврат номера класса.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода getclassnum класса cl_2

№	Предикат	Действия	№ перехода
1		возврат 2	∅

3.12 Алгоритм метода signal класса cl_3

Функционал: метода сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода signal класса cl_3

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вызова метода getabsoluteway()	2
2		добавление в конец строки message " (class: {результат вызова метода getclassnumber приведенный к строке})"	∅

3.13 Алгоритм метода handler класса cl_3

Функционал: обработчик.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода handler класса cl_3

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

3.14 Алгоритм метода getclassnum класса cl_3

Функционал: возврат номера класса.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода getclassnum класса cl_3

№	Предикат	Действия	№ перехода
1		возврат 3	∅

3.15 Алгоритм метода signal класса cl_4

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода signal класса cl_4

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вызова метода getabsoluteway()	2
2		добавление в конец строки message " (class: {результат вызова метода getclassnumber приведенный к строке})"	∅

3.16 Алгоритм метода handler класса cl_4

Функционал: обработчик.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода handler класса cl_4

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

3.17 Алгоритм метода getclassnum класса cl_4

Функционал: возврат номера класса.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода getclassnum класса cl_4

№	Предикат	Действия	№ перехода
1		возврат 4	∅

3.18 Алгоритм метода signal класса cl_5

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода signal класса cl_5

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вызова метода getabsoluteway()	2
2		добавление в конец строки message " (class: {результат вызова метода getclassnumber приведенный к строке})"	∅

3.19 Алгоритм метода handler класса cl_5

Функционал: обработчик.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода handler класса cl_5

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

3.20 Алгоритм метода getclassnum класса cl_5

Функционал: возврат номера класса.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода getclassnum класса cl_5

№	Предикат	Действия	№ перехода
1		возврат 5	∅

3.21 Алгоритм метода signal класса cl_6

Функционал: метод сигнала.

Параметры: ссылка на строку message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода signal класса cl_6

№	Предикат	Действия	№ перехода
1		вывод с новой строки "Signal from " и результат вызова метода getabsoluteway()	2
2		добавление в конец строки message " (class: {результат вызова метода getclassnumber приведенный к строке})"	∅

3.22 Алгоритм метода handler класса cl_6

Функционал: обработчик.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода handler класса cl_6

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

3.23 Алгоритм метода getclassnum класса cl_6

Функционал: возврат номера класса.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода getclassnum класса cl_6

№	Предикат	Действия	№ перехода
1		возврат 6	∅

3.24 Алгоритм функции main

Функционал: основная функция программы.

Параметры: none.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 25.

Таблица 25 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_app класса app с использованием параметризированного конструктора и передачей в него в качестве параметра пустого указателя	2
2		вызов метода treeobj объекта ob_app	3
3		возвращение результата работы метода start_app() для объекта ob_app	∅

3.25 Алгоритм метода getclassnum класса base

Функционал: возврат номера класса.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода getclassnum класса base

№	Предикат	Действия	№ перехода
1		возврат 0	∅

3.26 Алгоритм деструктора класса base

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

Параметры: none.

Алгоритм деструктора представлен в таблице 27.

Таблица 27 – Алгоритм деструктора класса base

№	Предикат	Действия	№ перехода
1		инициализация указателя rootptr на объект класса base адресом текущего объекта	2
2	у объекта по указателю есть родитель	присваивание rootptr результат вызова метода get_parent объекта по указателю rootptr	2
		объявление стека stack содержащего указатели на объекты класса base	3
3		добавление на вершину стека указатель rootptr	4
4	стек stack содержит элементы	инициализация указателя ptr на класс base значением указателя на вершине стека	5
			12
5		удаление вершины стека	6
6		инициализация переменной i типа int со значением 0	7
7	i< размера вектора connects объекта по указателю ptr		10
		i=0	8
8	i< размера вектора descens	добавление в стек stack i-го элемента вектора	9

№	Предикат	Действия	№ перехода
	объекта по указателю ptr	descens объекта по указателю ptr	
			4
9		i++	7
10	свойство targetptr i-го объекта вектора connects объекта по указателю ptr совпадает с текущим объектом	удаление i-го элемента из массива connects объекта по указателю ptr	11
		i++	7
11		вызов оператора delete для i-го элемента массива ptr объекта по указателю ptr	7
12	вектор descens содержит элементы	удаление нулевого элемента из вектора descens	13
			∅
13		вызов оператора delete для указателя tmpptr	∅

3.27 Алгоритм функции classnumbertohandler

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

Параметры: int classnum.

Возвращаемое значение: указатель на метод обработчика.

Алгоритм функции представлен в таблице 28.

Таблица 28 – Алгоритм функции classnumbertohandler

№	Предикат	Действия	№ перехода
1	classnum = 1	возврат указателя на метод обработчика класса app	∅
	classnum = 2	возврат указателя на метод обработчика класса	∅

№	Предикат	Действия	№ перехода
		cl_2	
	classnum = 3	возврат указателя на метод обработчика класса cl_3	∅
	classnum = 4	возврат указателя на метод обработчика класса cl_4	∅
	classnum = 5	возврат указателя на метод обработчика класса cl_5	∅
	classnum = 6	возврат указателя на метод обработчика класса cl_6	∅
		возврат нулевого вектора	∅

3.28 Алгоритм функции classnumbertosignal

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

Параметры: int classnum.

Возвращаемое значение: указатель на метод сигнала.

Алгоритм функции представлен в таблице 29.

Таблица 29 – Алгоритм функции classnumbertosignal

№	Предикат	Действия	№ перехода
1	classnum = 1	возврат указателя на метод сигнала класса app	∅
	classnum = 2	возврат указателя на метод сигнала класса cl_2	∅
	classnum = 3	возврат указателя на метод сигнала класса cl_3	∅
	classnum = 4	возврат указателя на метод сигнала класса cl_4	∅
	classnum = 5	возврат указателя на метод сигнала класса cl_5	∅
	classnum = 6	возврат указателя на метод сигнала класса cl_6	∅
		возврат нулевого указателя	∅

3.29 Алгоритм метода treeobj класса app

Функционал: построение дерева иерархии.

Параметры: none.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 30.

Таблица 30 – Алгоритм метода treeobj класса app

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2
2		объявление строк way, descenname	3
3		объявление переменной tmp типа int	4
4		ввод descenname	5
5		вызов метода setname с параметром descanname	6
6		объявление указателя parentnodeptr на объект класса base	7
7		инициализация указателя lastcreateptr адресом текущего объекта	8
8		ввод path	9
9	way != "endtree"	ввод descenname и tmp	10
			16
10		присваивание parentnodeptr результат вызова метода getobjbyway с параметром path объекта по указателю lastcreateptr, вывод с новой строки "The head object ", way, "is not found"	11
11	parentnodeptr нулевой указатель	вызов метода printbranch	12
			14
12		вывод с новой строки "The head object ", way, " is not found"	13

№	Предикат	Действия	№ перехода
13		выход с кодом 1	∅
14	у объекта по указателю parentnodeptr нет подчиненного с именем descenname	вывод с новой строки way, " Dubbing the names of subordinate objects"	15
	tmp принимает значение от 1 до 6	создание объекта класса в соответствии со значением переменной tmp с помощью оператора new, конструктора ипараметров parentnodeptr, descenname и присваивание lastcreateptr фдрес этого объекта	15
			15
15		ввод way	9
16		объявление указателя purposeptr на объект класса base	17
17		объявление строки purposeway	18
18		ввод way	19
19	way != "end_of_connections"	ввод purposeway	20
			∅
20		присваивание parentnodeptr результат вызова метода getobjbyway с параметром way	21
21		присваивание purposeptr результат вызова метода getobjbyway с параметром purposeptr	22
22		инициализация указателя signalf на метод сигнала результатом вызова функции classnumbertosignal с параметром, являющимся номером класса объекта по указателю parentnodeptr	23
23		инициализация указателя handlerf на метод обработчика результатом вызова функции classnumbertosignal с параметром, являющимся	24

№	Предикат	Действия	№ перехода
		номером класса объекта по указателю purposeptr	
24		вызов методаsetconnect объекта по указателю parentnodeptr с параметрами signalf, purposeptr, handlerf	25
25		ввод значения way	19

3.30 Алгоритм метода start_app класса app

Функционал: запуск программы.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 31.

Таблица 31 – Алгоритм метода start_app класса app

№	Предикат	Действия	№ перехода
1		объявление указателя signalf на метод сигнала	2
2		объявление указателя handlerf на метод обработчика	3
3		вызов метода setstatebranch с параметром 1	4
4		инициализация пустых строк command, input и message	5
5		объявление переменной newstate типа int	6
6		объявление указателей extraobjectptr, purposeobjectptr на объекты класса base	7
7		вызов метода printbranch	8
8		ввод command	9
9	command!="END"	ввод input	10
		возврат 0	∅
10		присваивание extraobject результат вызова метода	11

№	Предикат	Действия	№ перехода
		getobjbyway с параметром input	
11	extraobjectptr нулевой указатель	вывод с новой строки "Object", input, " not found"	13
	command = "EMIT"	ввод message	14
	command = "SET_CONNECT"	ввод input	16
	command = "DELETE_CONNECT"	ввод input	20
	command = "SET_CONDITION"	ввод newstate	24
			12
12		ввод command	9
13		ввод input	9
14		инициализация переменной n типа int результатом вызова метода getclassnumber объекта по указателю extraobjectptr	15
15		вызов метода emitsignal объекта по указателю extraobjectptr с параметрами classnumbertosignal(n) и message	9
16			17
17	purposeobjectptr нулевой указатель	вывод ""Handler object ", input, "not found	9
		присваивание signalf результат вызова функции classnumbertosignal с параметром, являющимся номером класса объекта по указателю extraobjectptr	18
18		присваивание handlerf результат вызова функции classnumbertohandler с параметром, являющимся номером класса объекта по указателю	19

№	Предикат	Действия	№ перехода
		purposeobjectptr	
19		вызов метода setconnect объекта по указателю extraobjectptr с параметрами signalf, purposeobjectptr, handlerf	9
20		присваивание purposeobjectptr результата вызова метода getobjbyway с параметром input	21
21	purposeobjectptr нулевой указатель	вывод "Handler object ", input, " not found"	9
		присваивание signalf результат вызова функции classnumbertosignal с параметром, являющимся номером класса объекта по указателю extraobject	22
22		присваивание handlerf результат вызова функции classnumbertohandler с параметром, являющимся номером класса объекта по указателю purposeobject	23
23		вызов метода removeconnect объекта по указателю extraobjectptr с параметрами signalf, purposeobjectptr, handlerf	9
24		вызов метода setstate объекта по указателю extraobjectptr с параметром newstate	9

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-22.

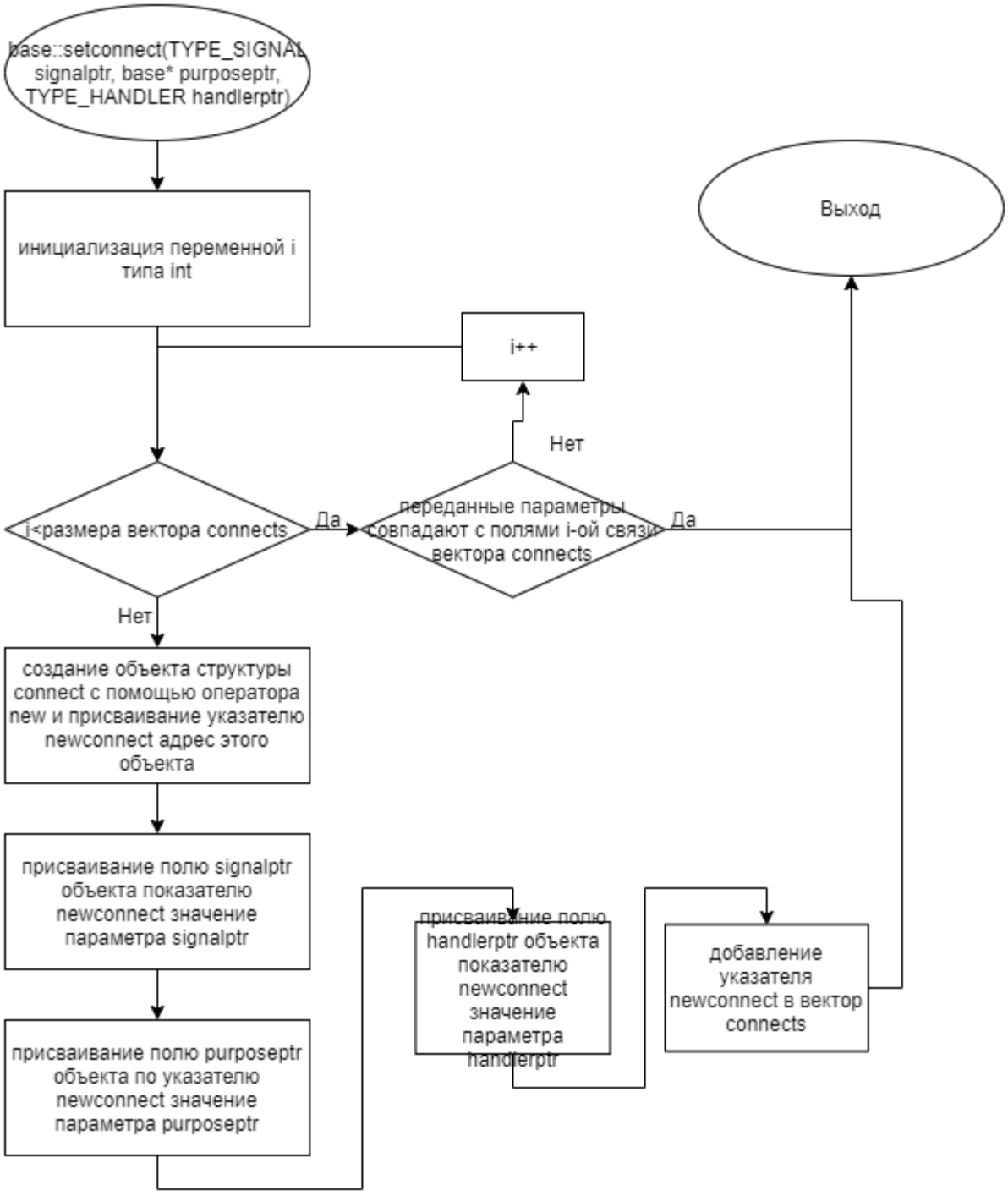

Рисунок 1 – Блок-схема алгоритма

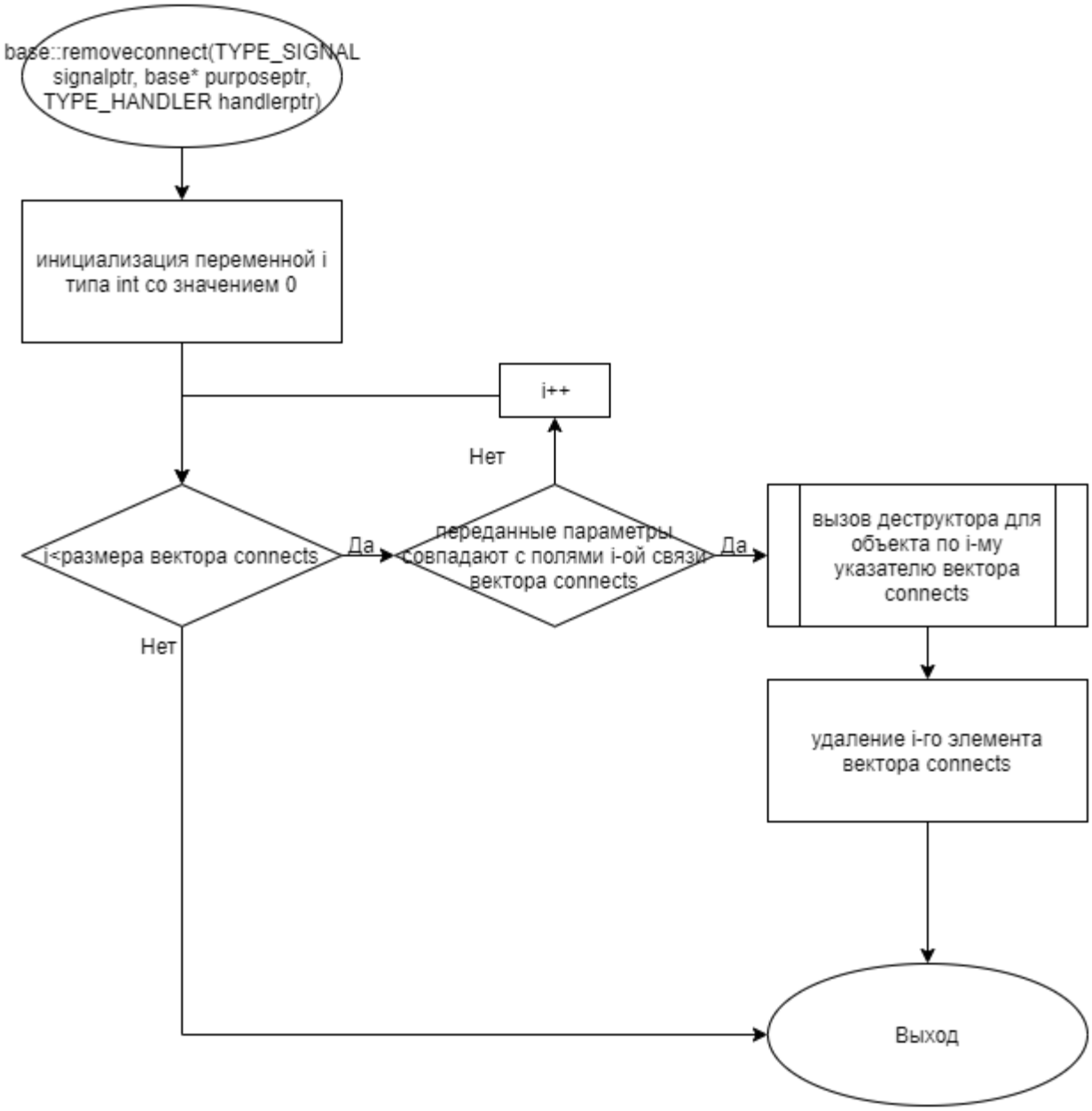

Рисунок 2 – Блок-схема алгоритма

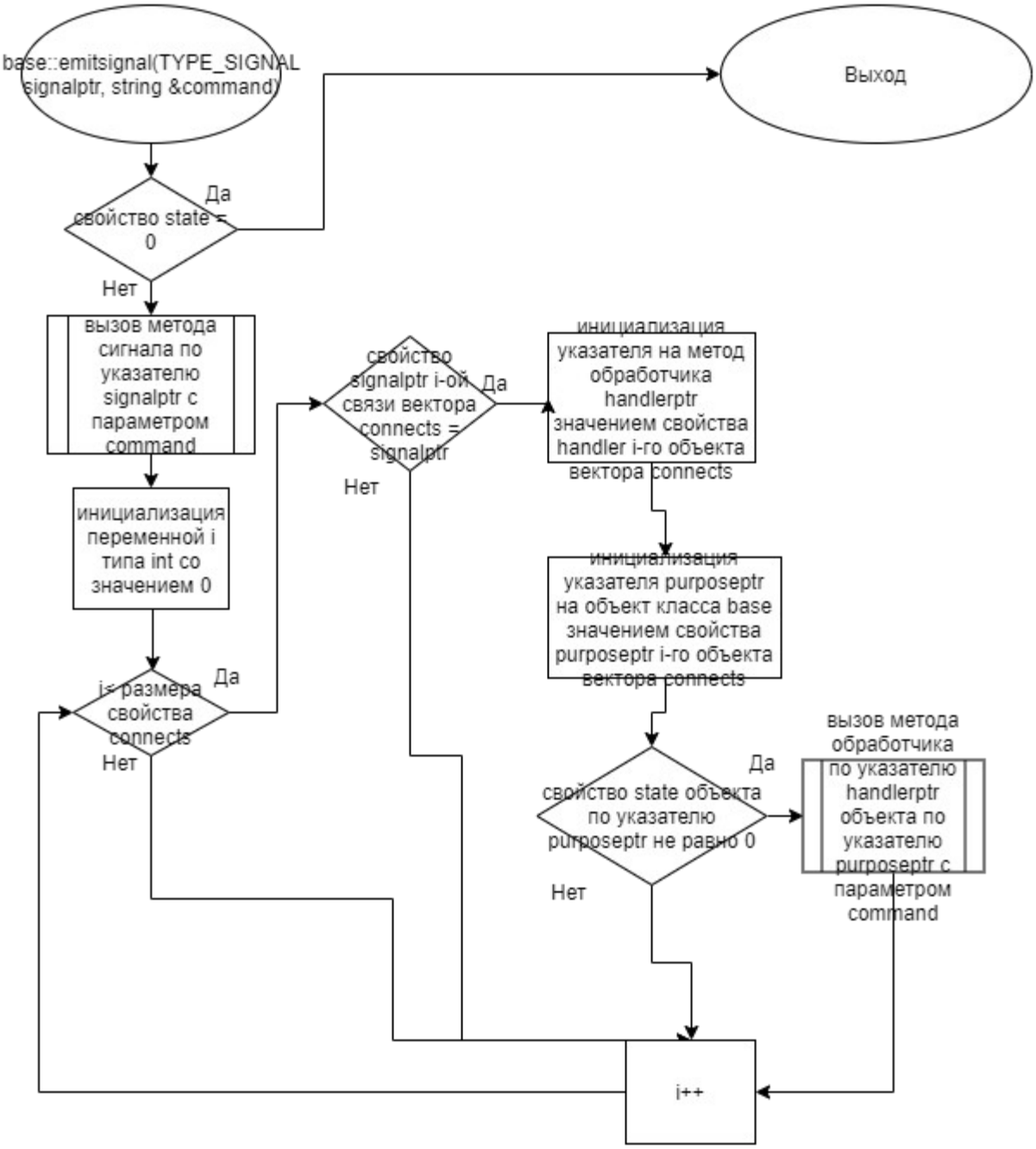

Рисунок 3 – Блок-схема алгоритма

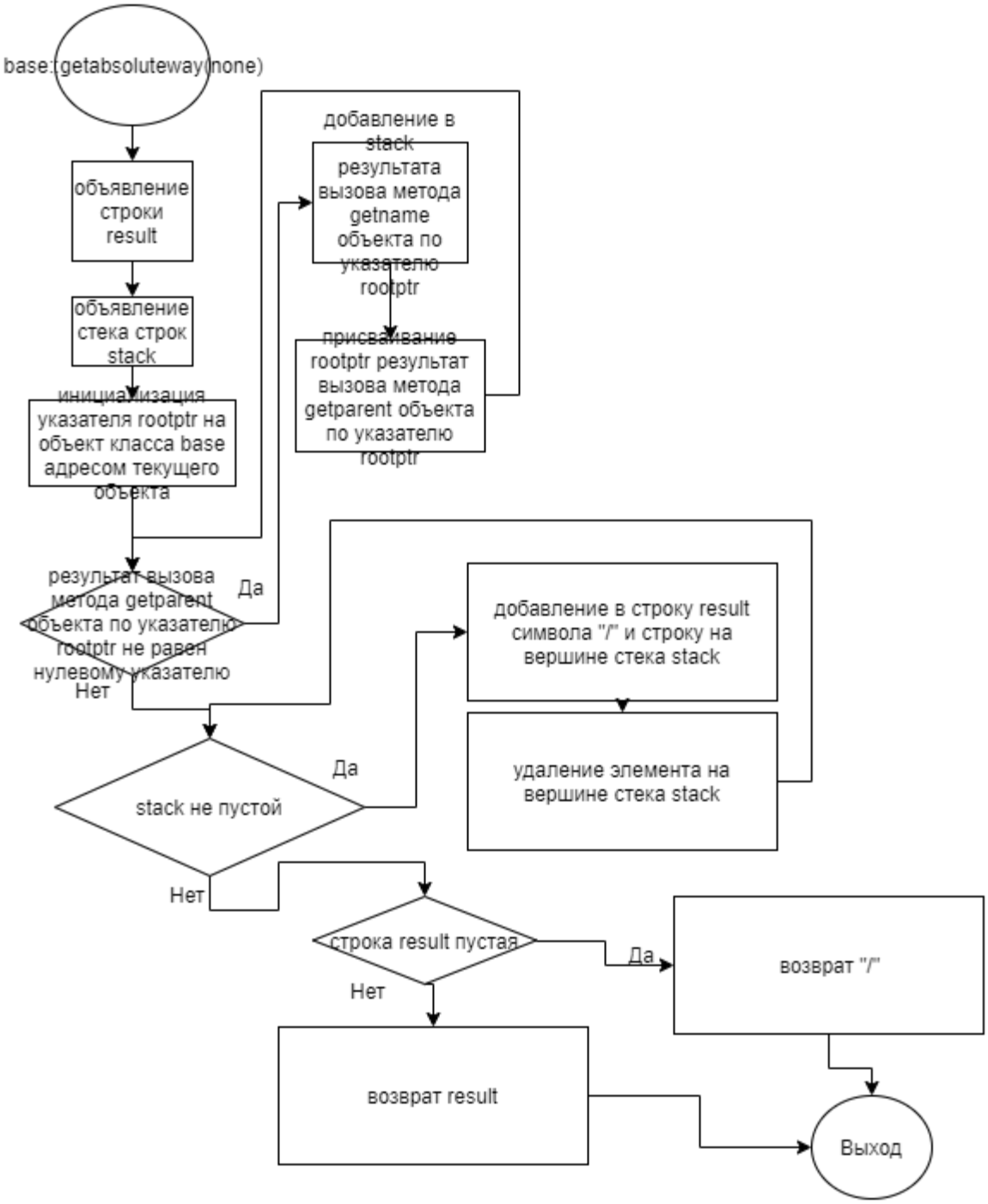

Рисунок 4 – Блок-схема алгоритма

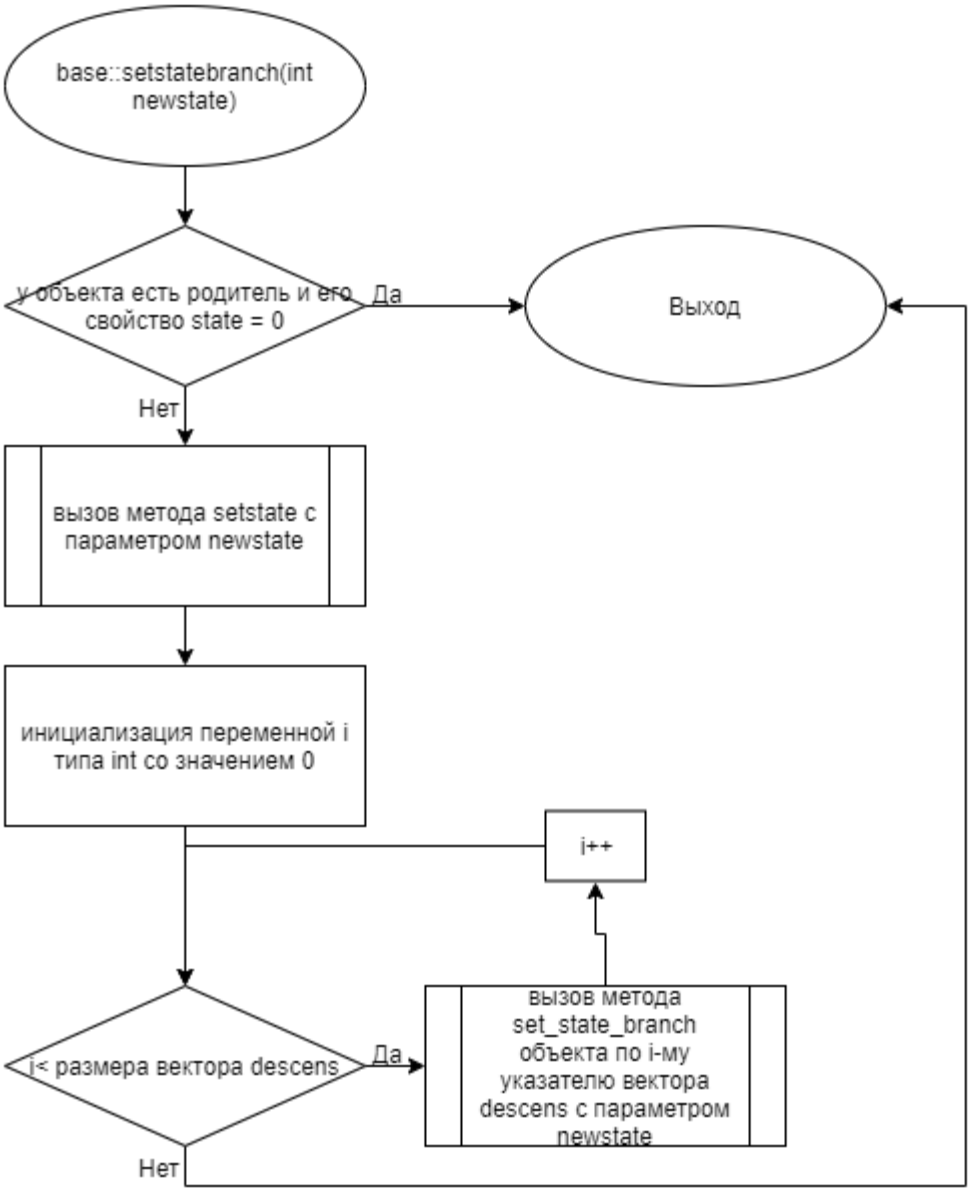

Рисунок 5 – Блок-схема алгоритма

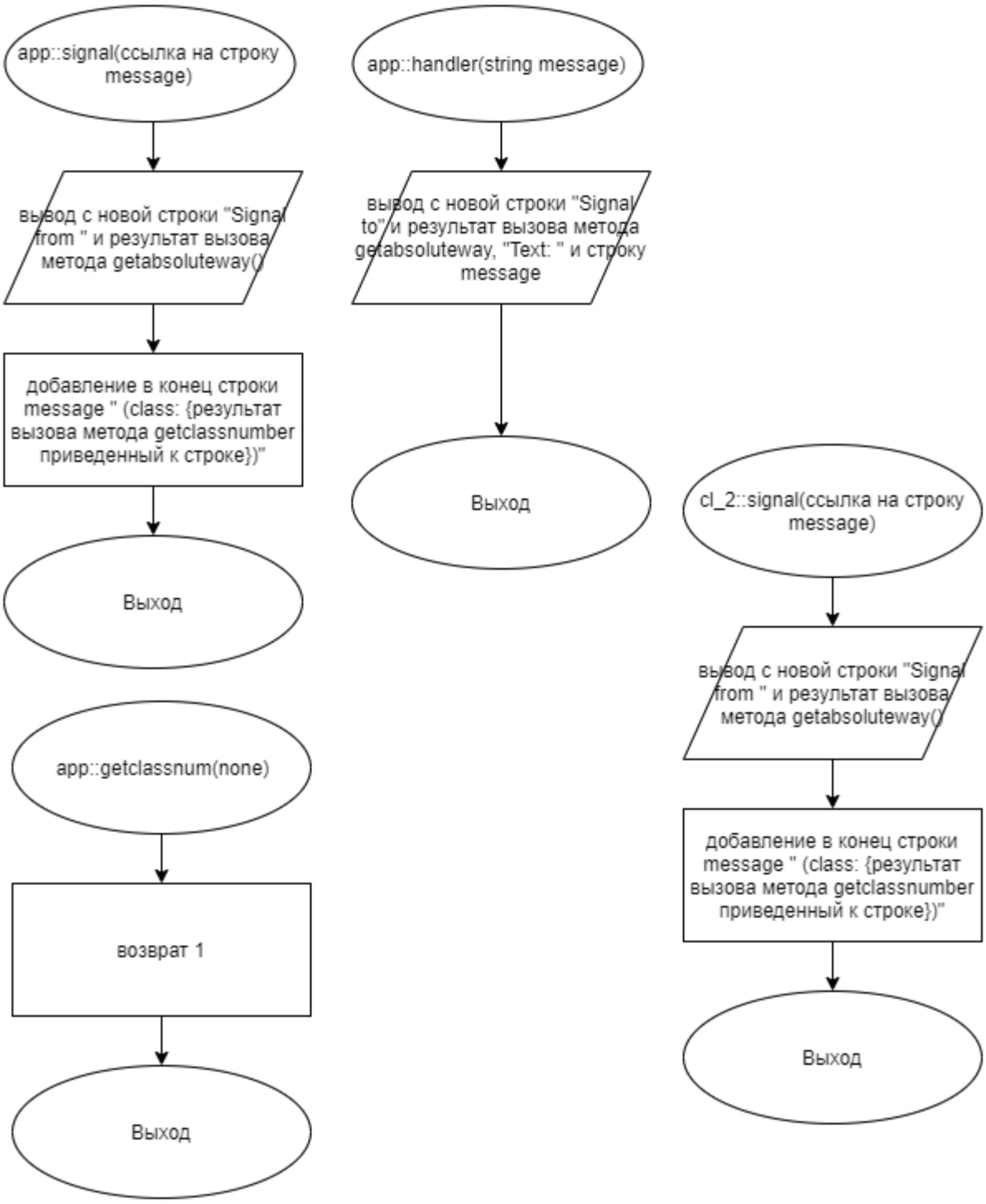

Рисунок 6 – Блок-схема алгоритма

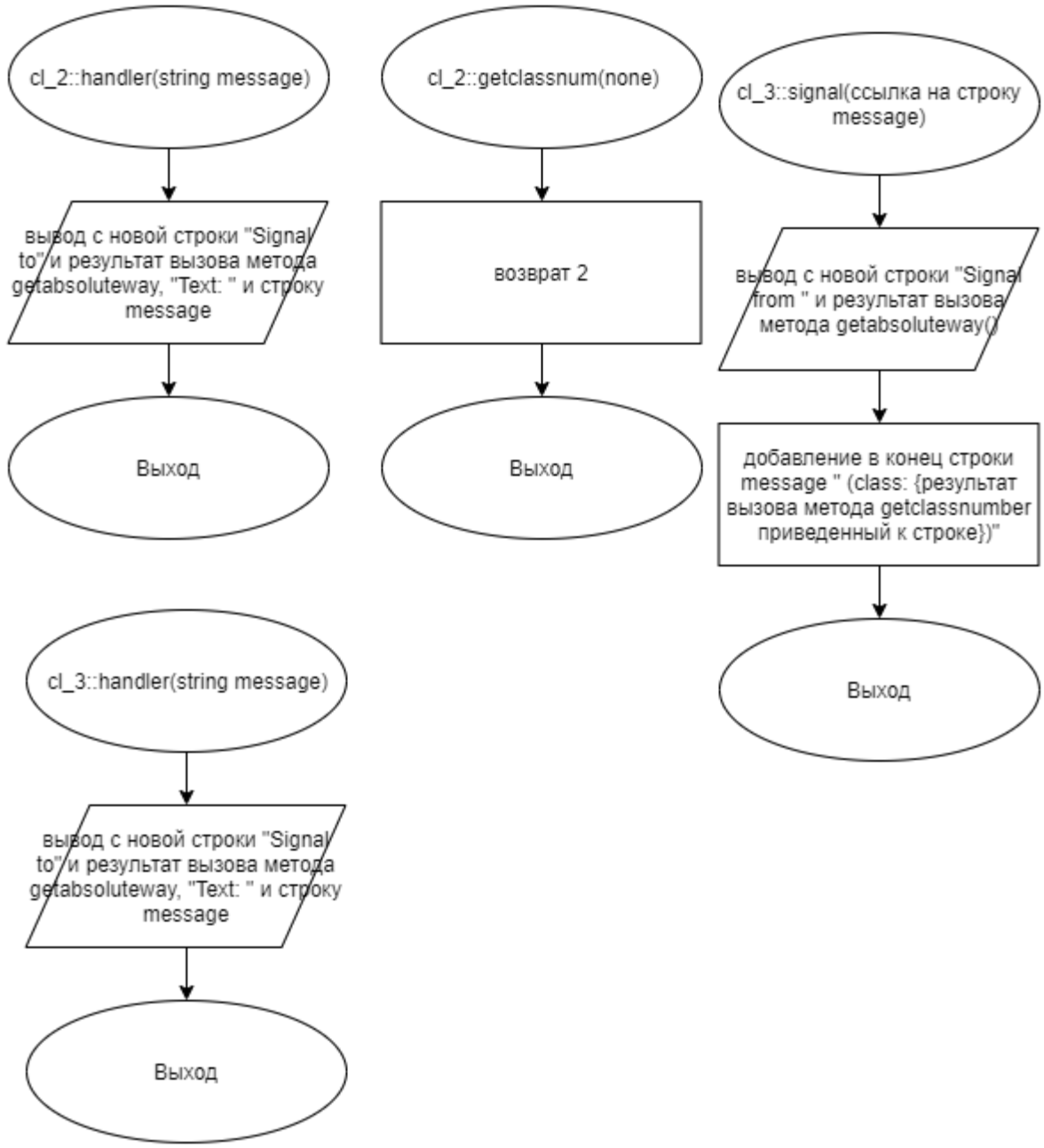

Рисунок 7 – Блок-схема алгоритма

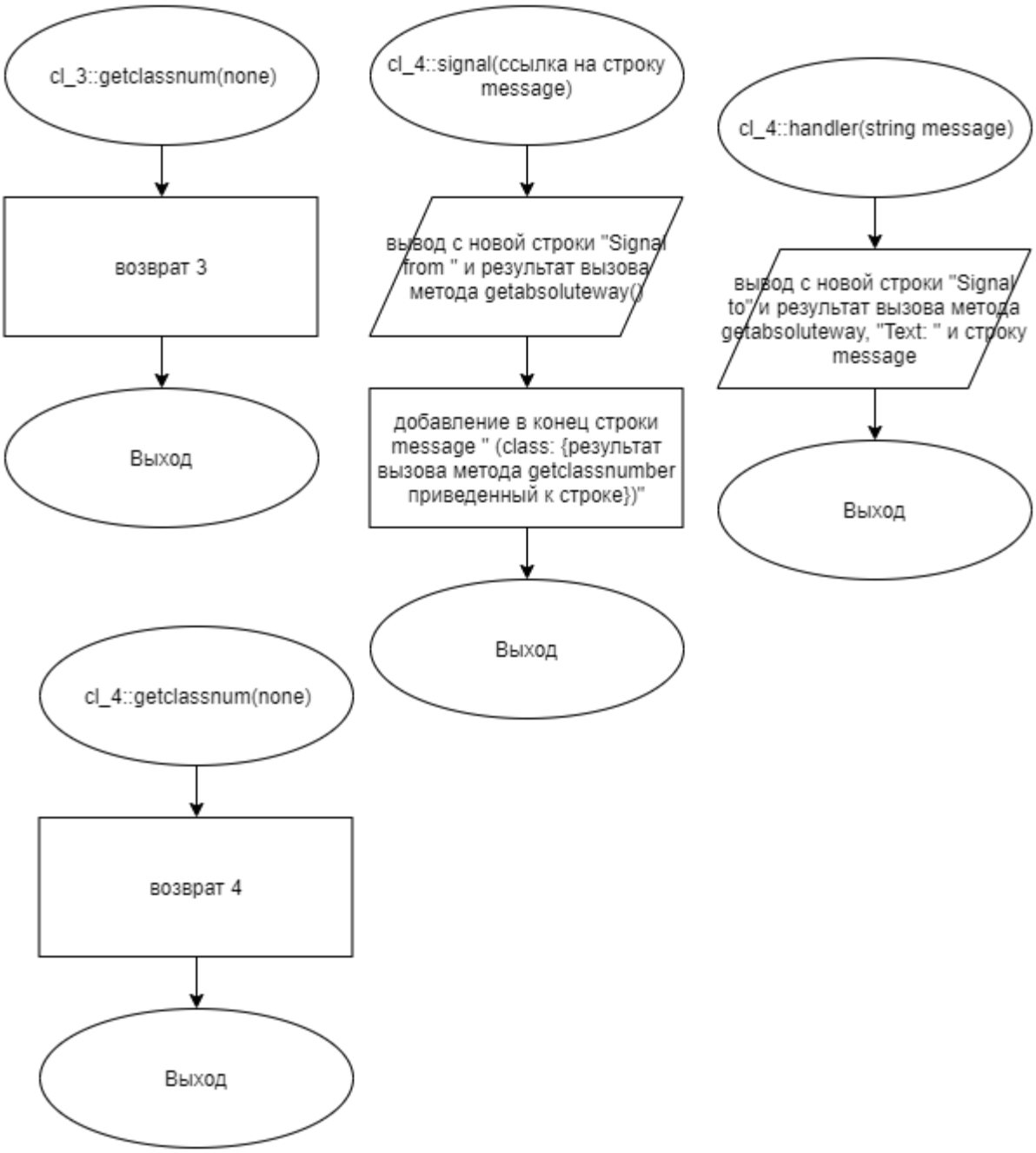

Рисунок 8 – Блок-схема алгоритма

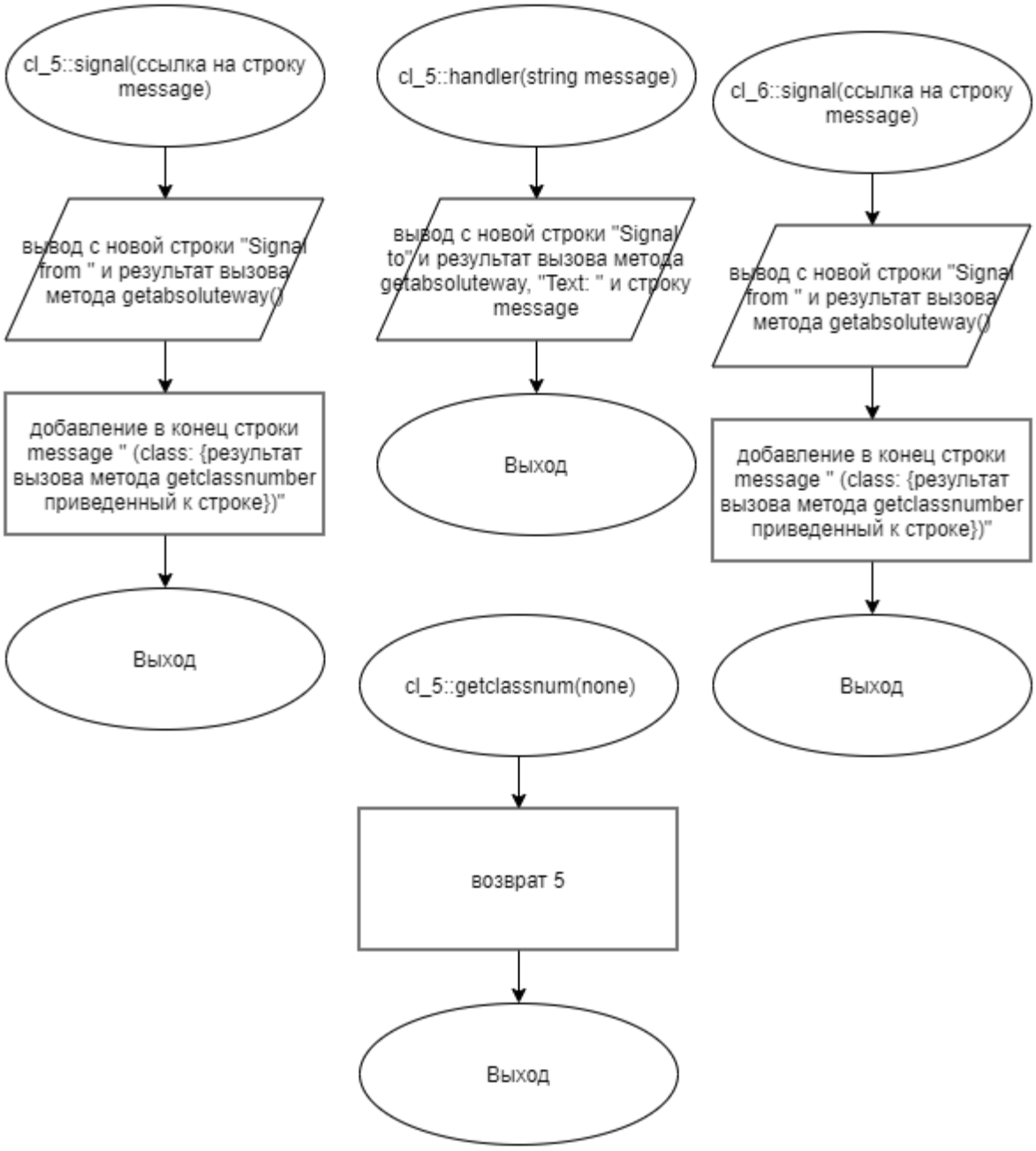

Рисунок 9 – Блок-схема алгоритма

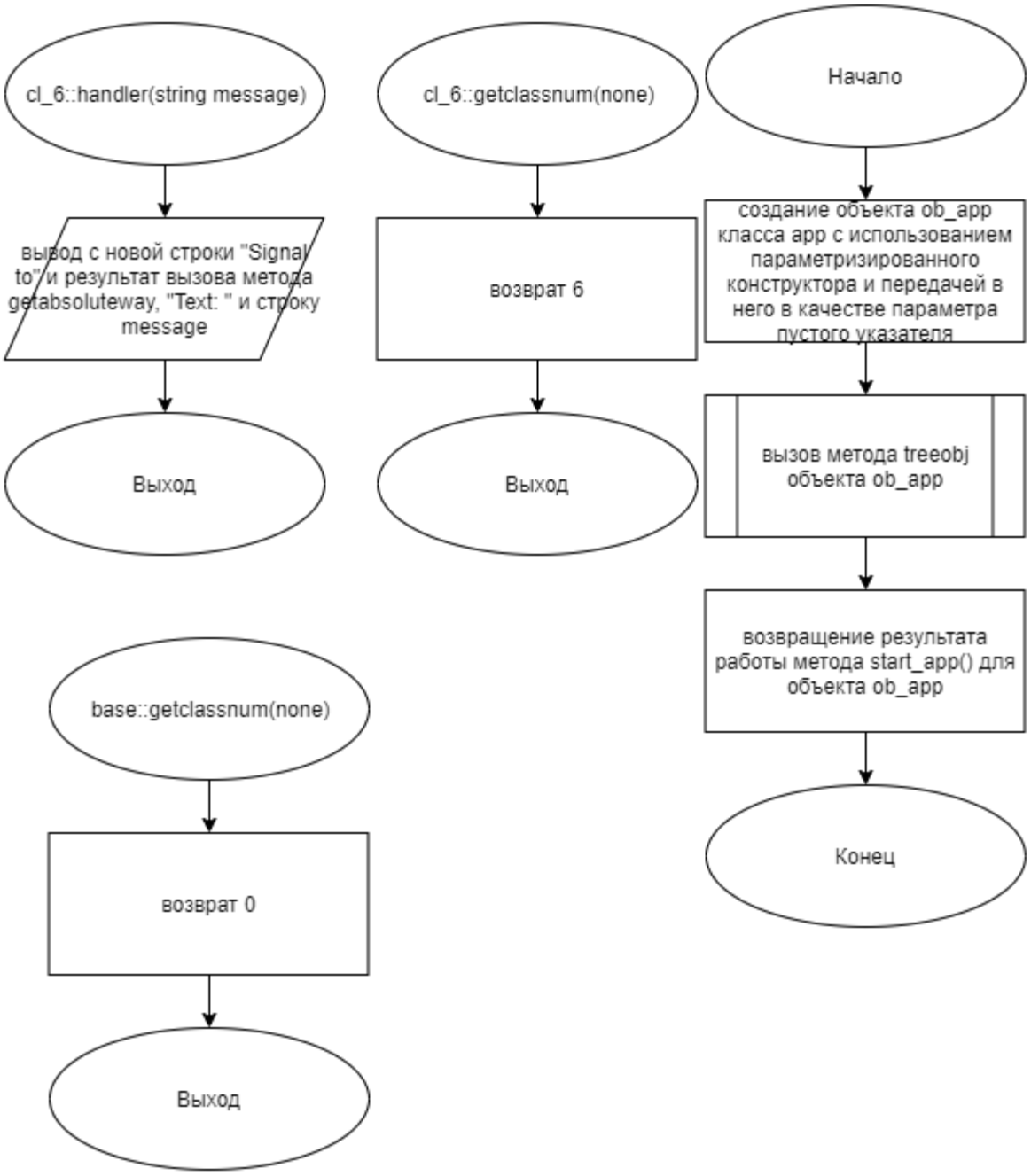

Рисунок 10 – Блок-схема алгоритма

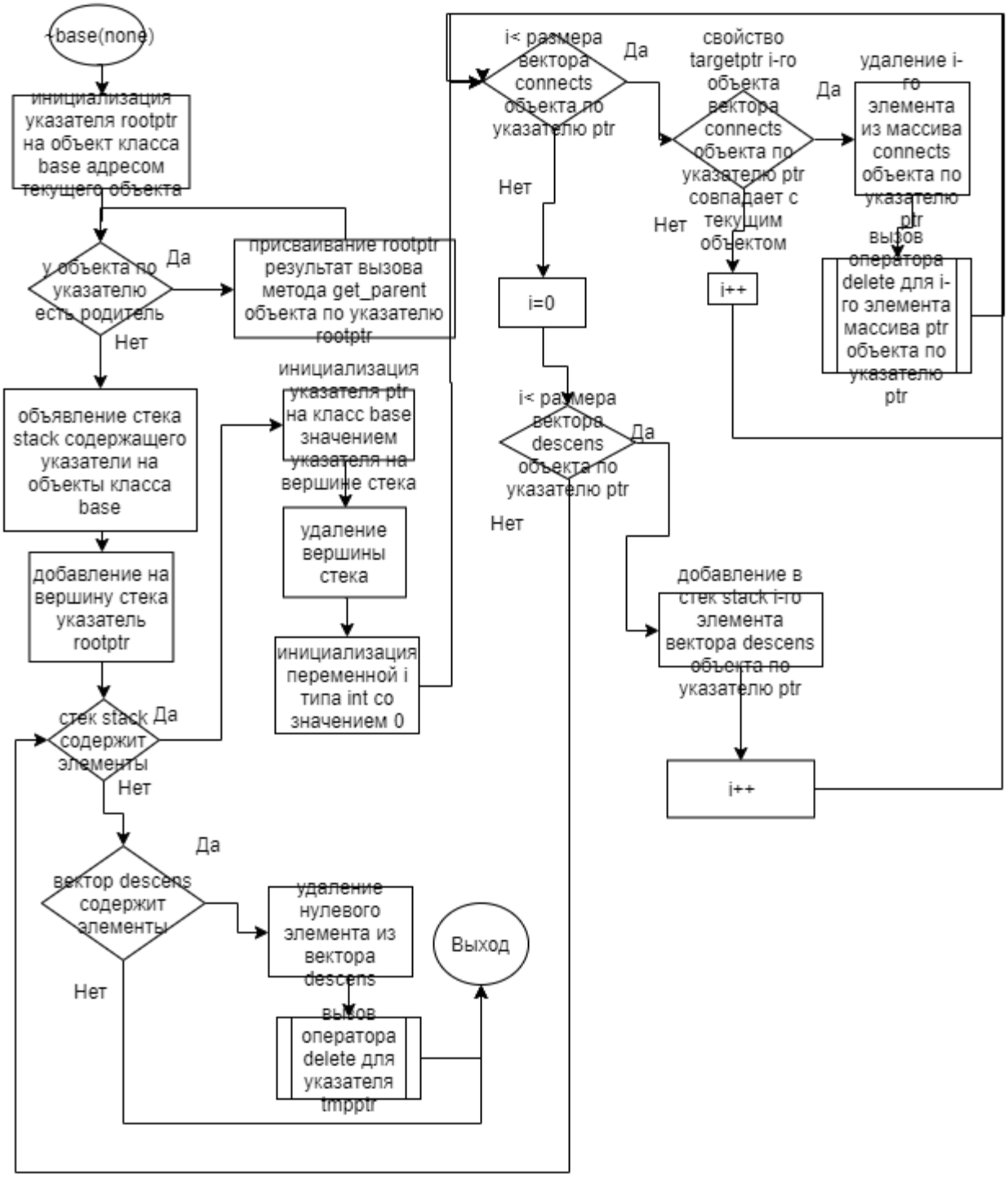

Рисунок 11 – Блок-схема алгоритма

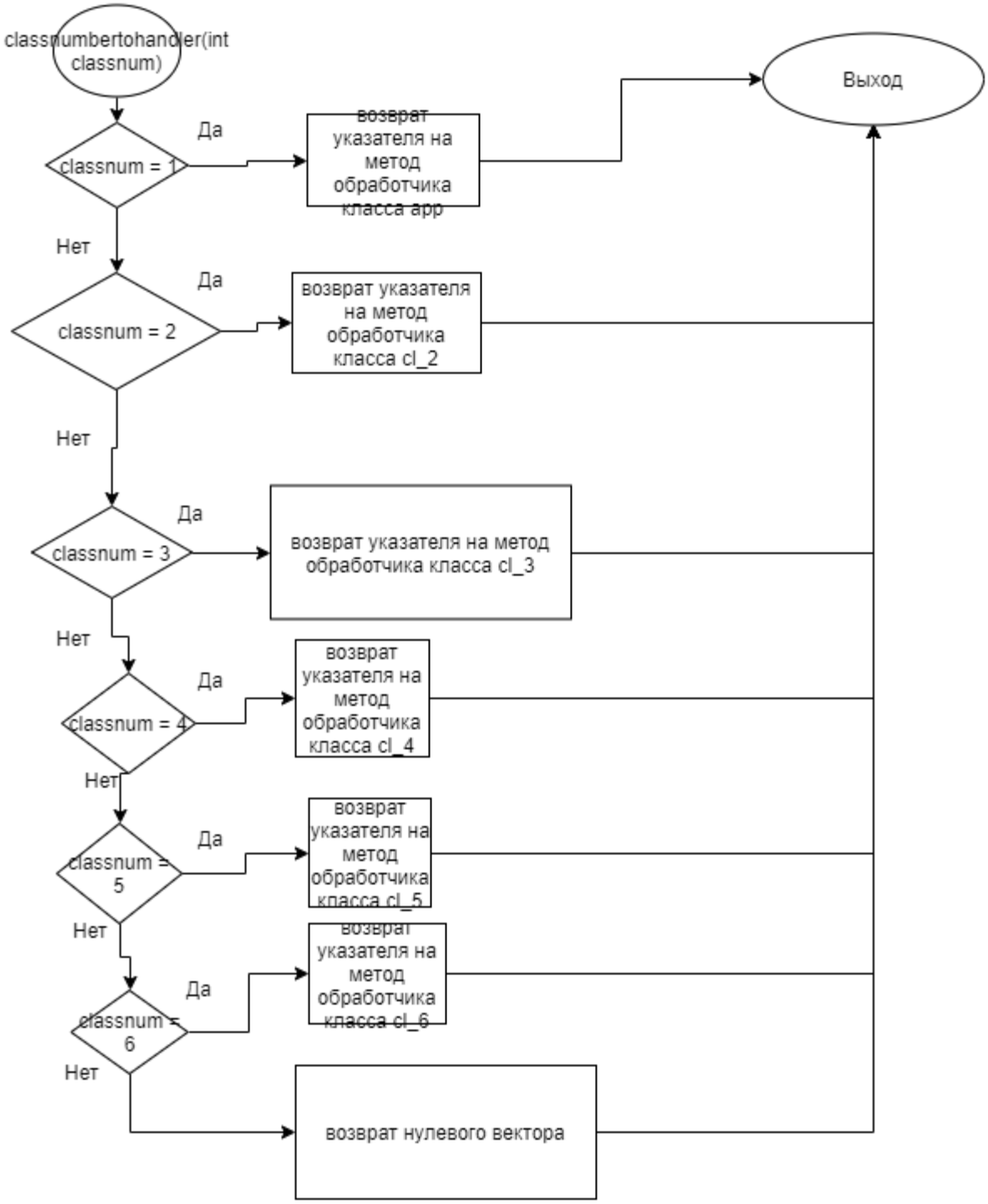

Рисунок 12 – Блок-схема алгоритма

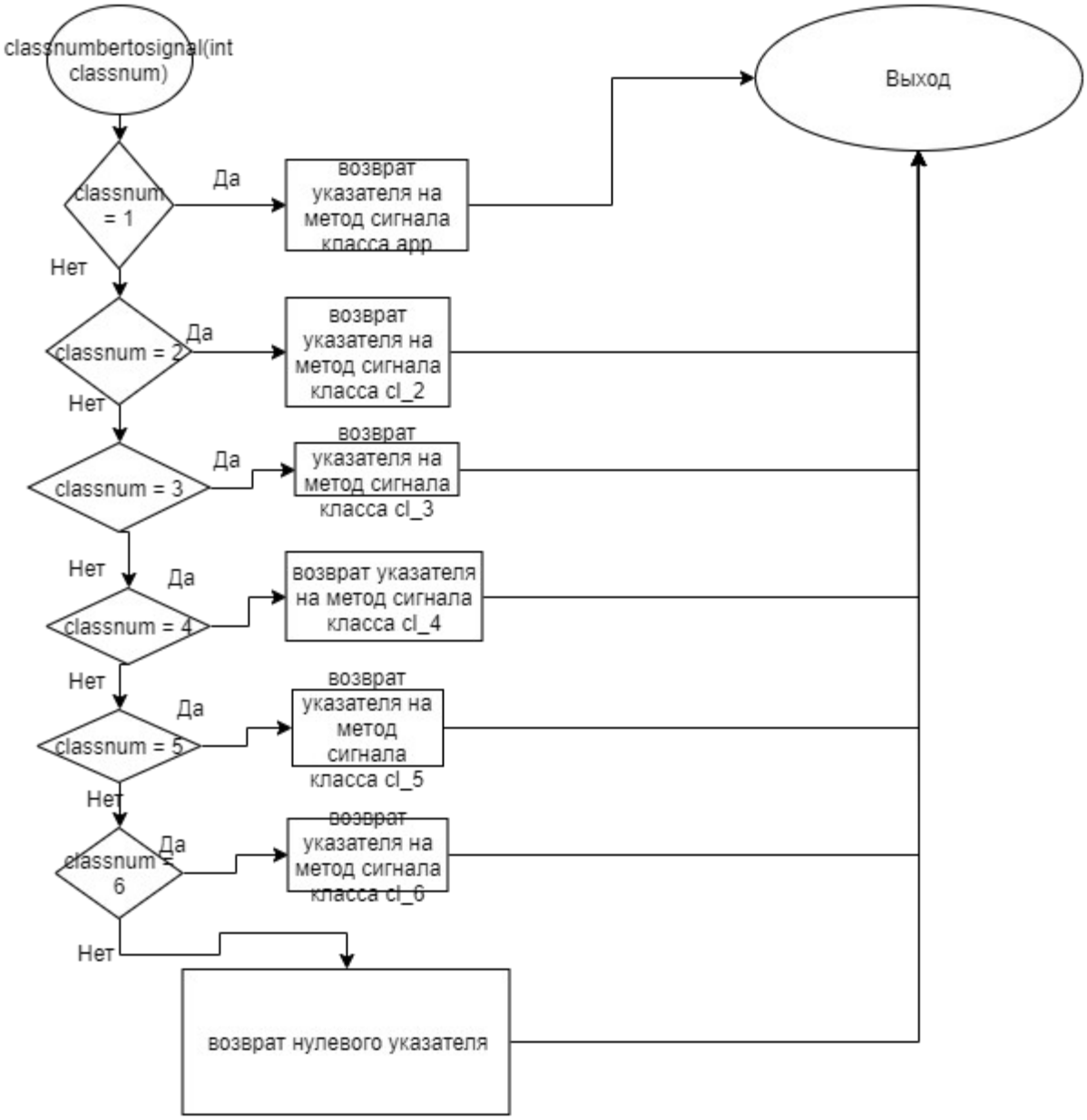

Рисунок 13 – Блок-схема алгоритма

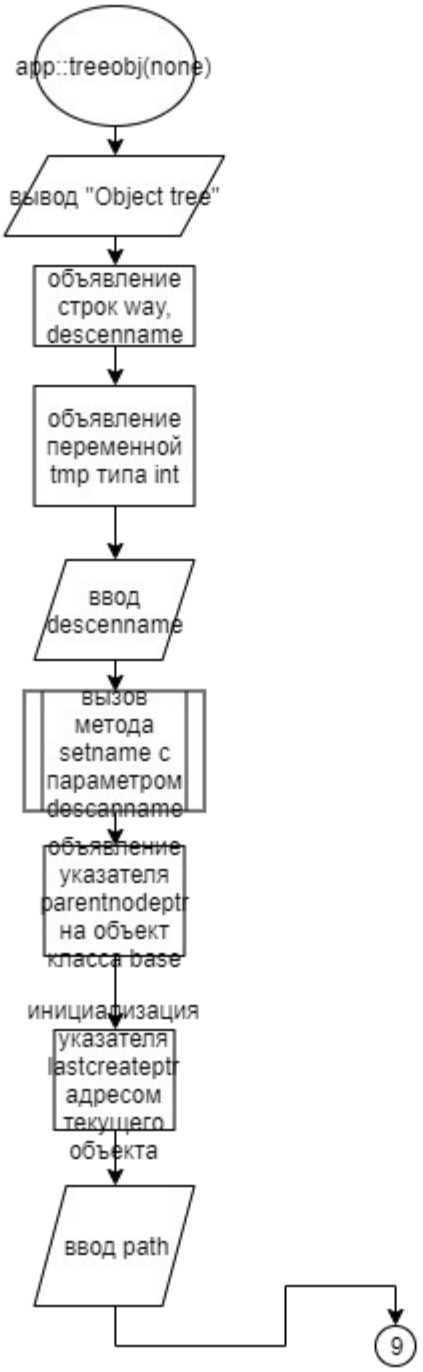

Рисунок 14 – Блок-схема алгоритма

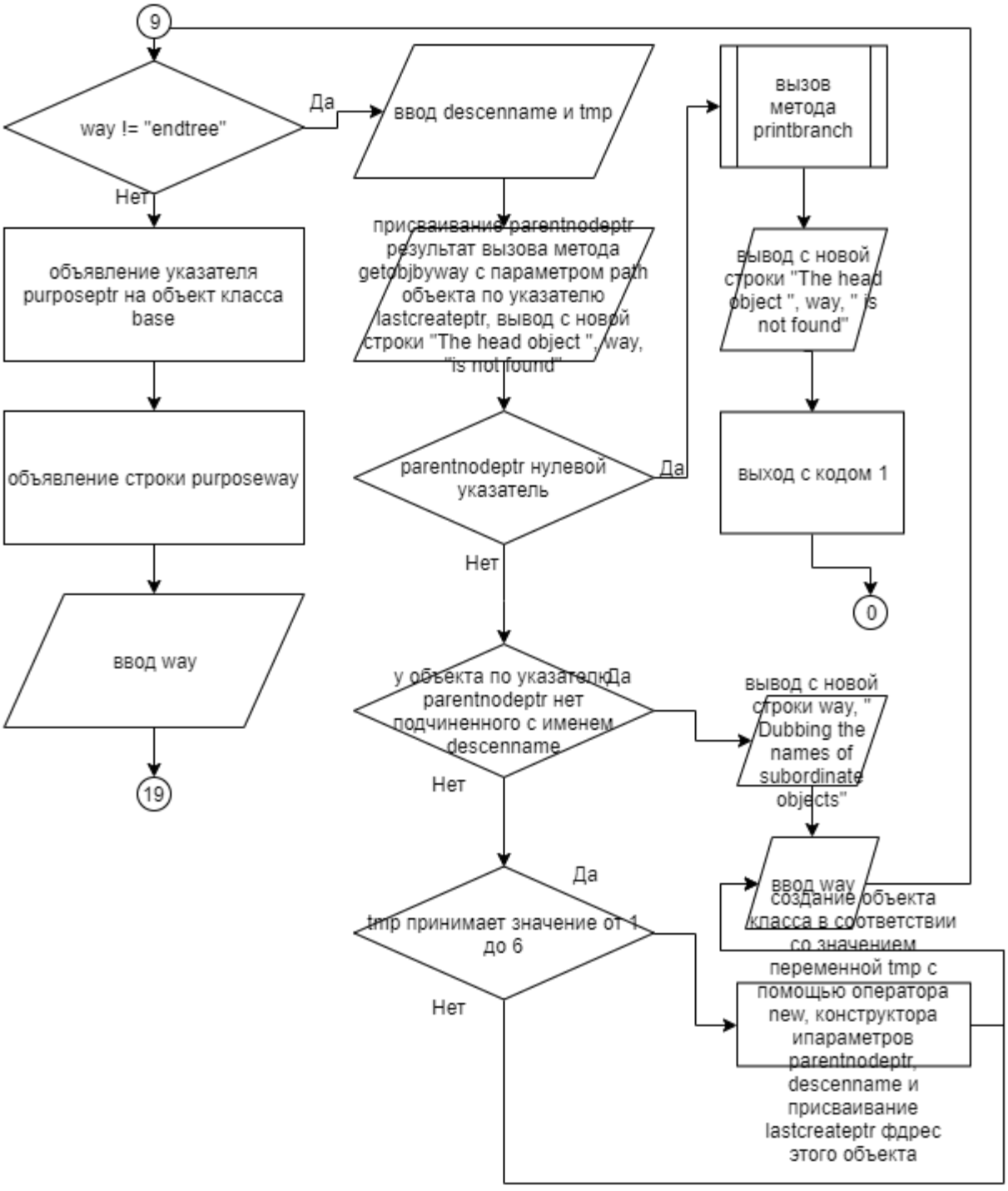

Рисунок 15 – Блок-схема алгоритма

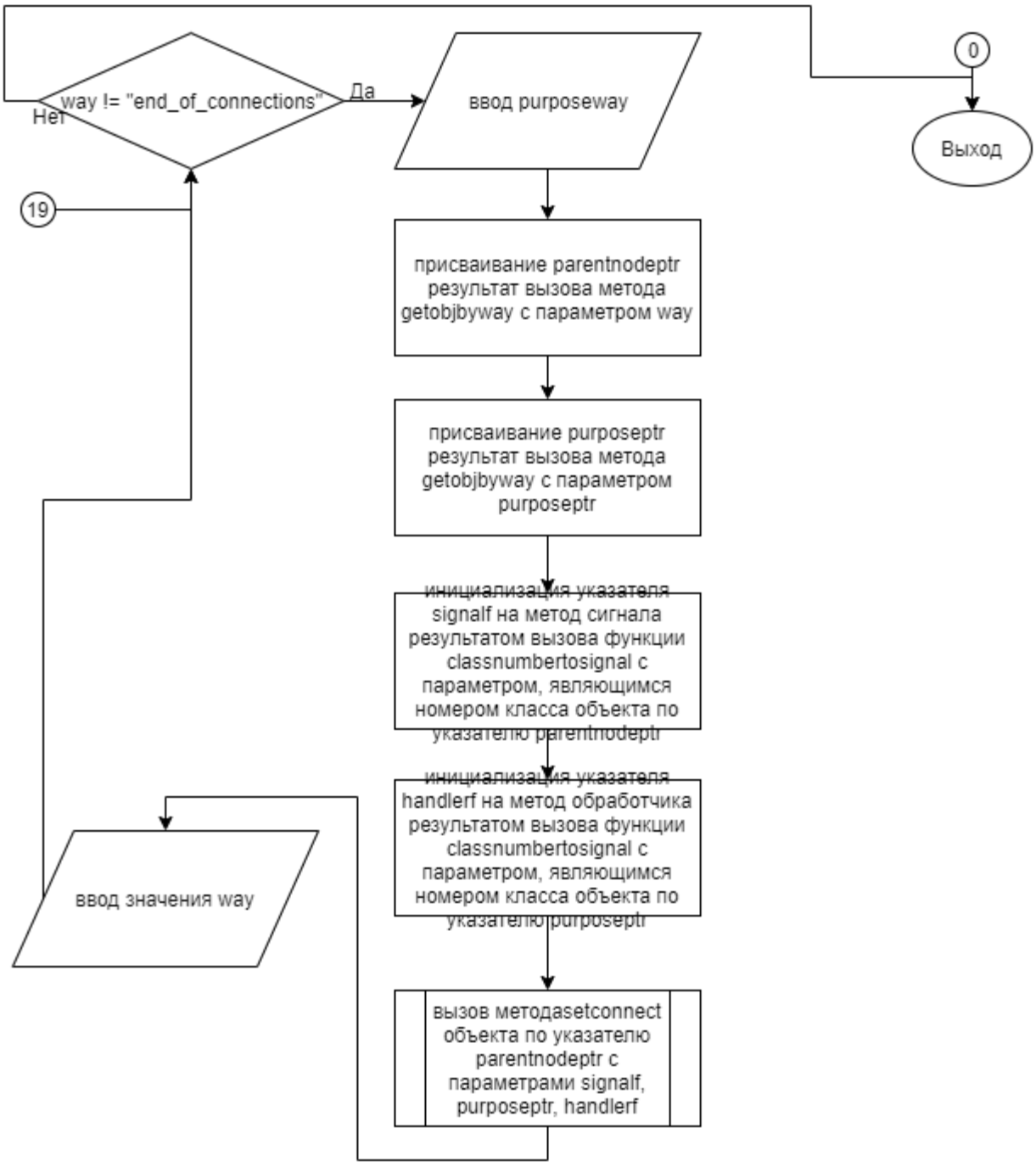

Рисунок 16 – Блок-схема алгоритма

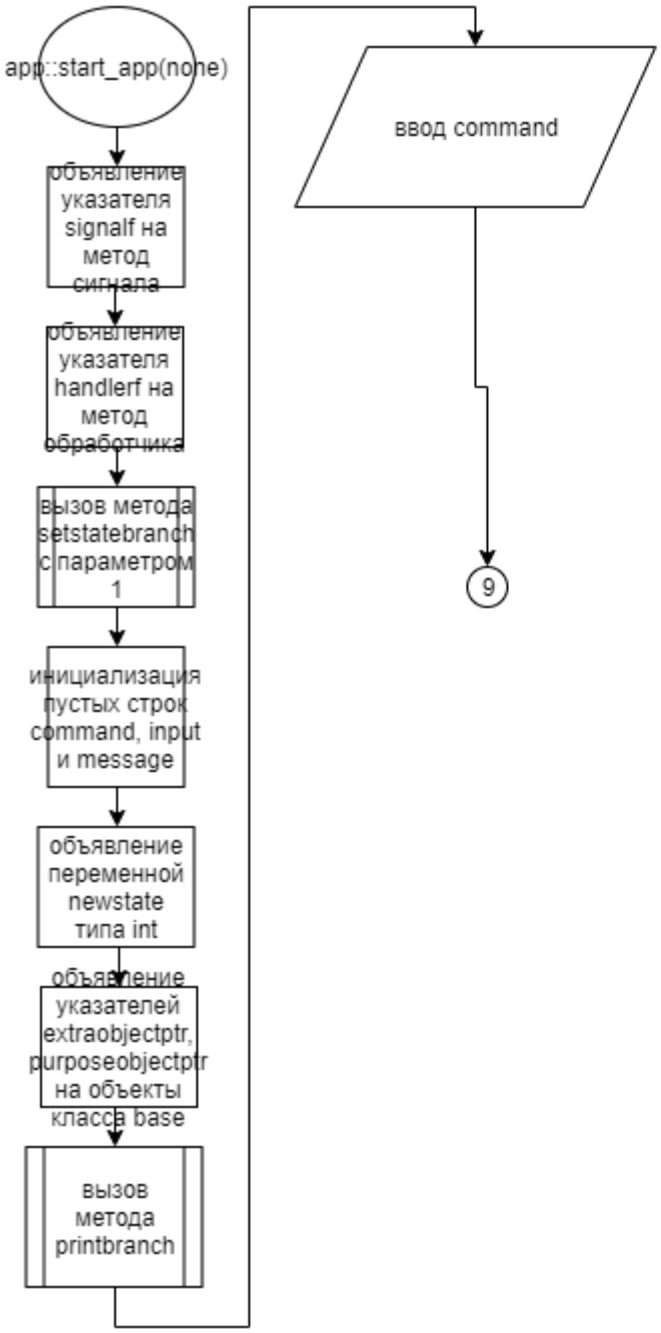

Рисунок 17 – Блок-схема алгоритма

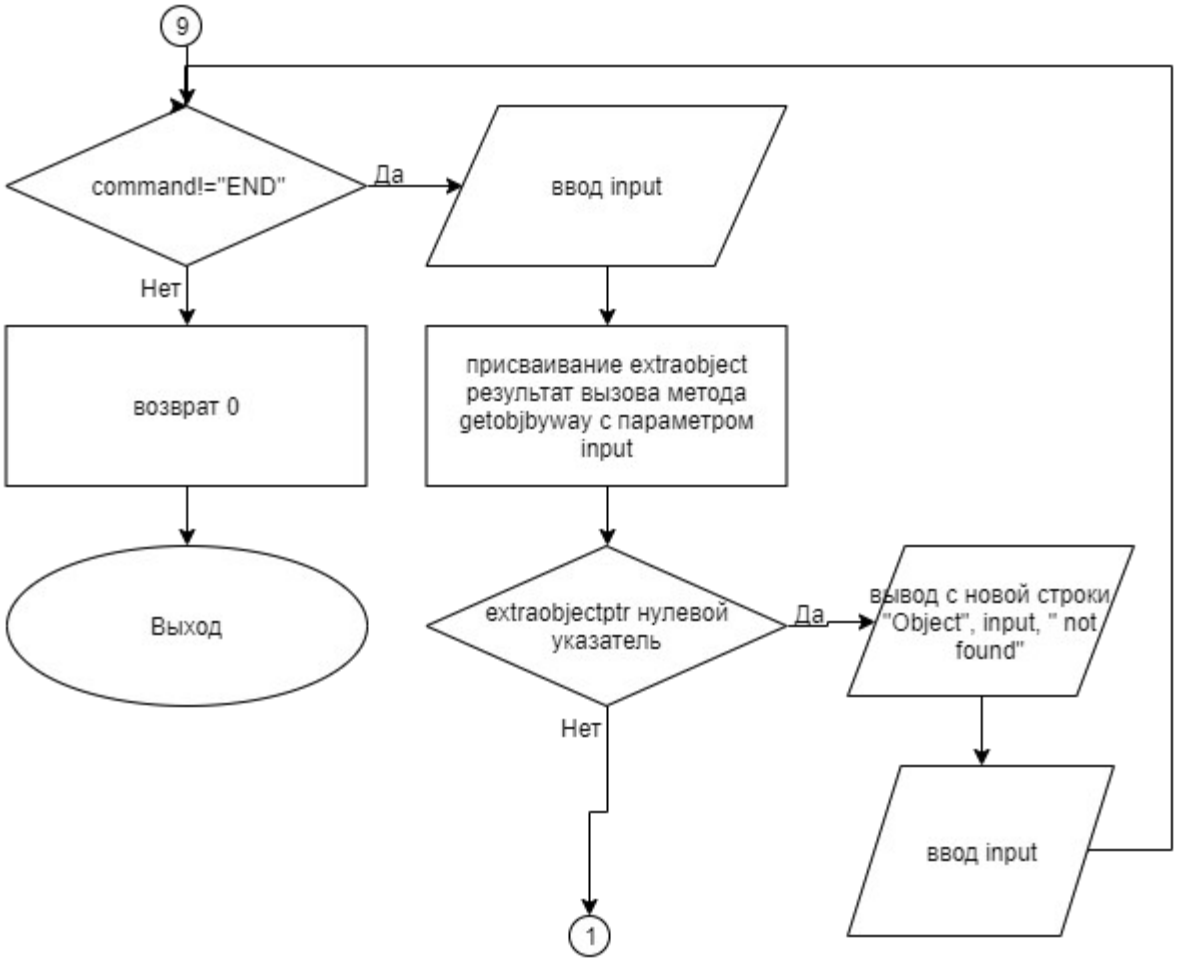

Рисунок 18 – Блок-схема алгоритма

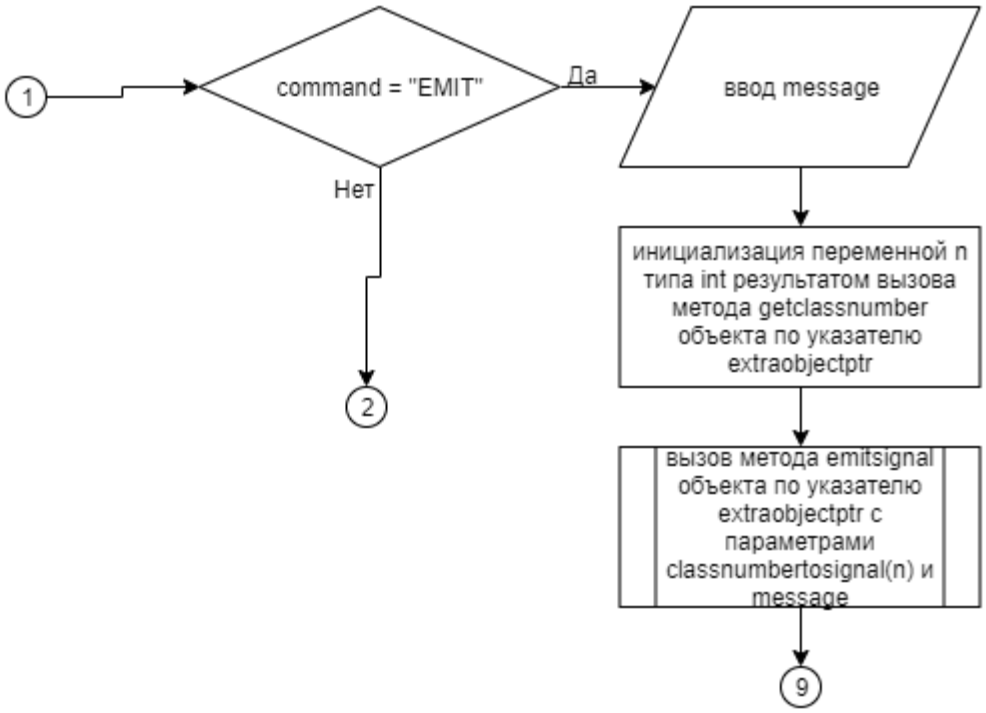

Рисунок 19 – Блок-схема алгоритма

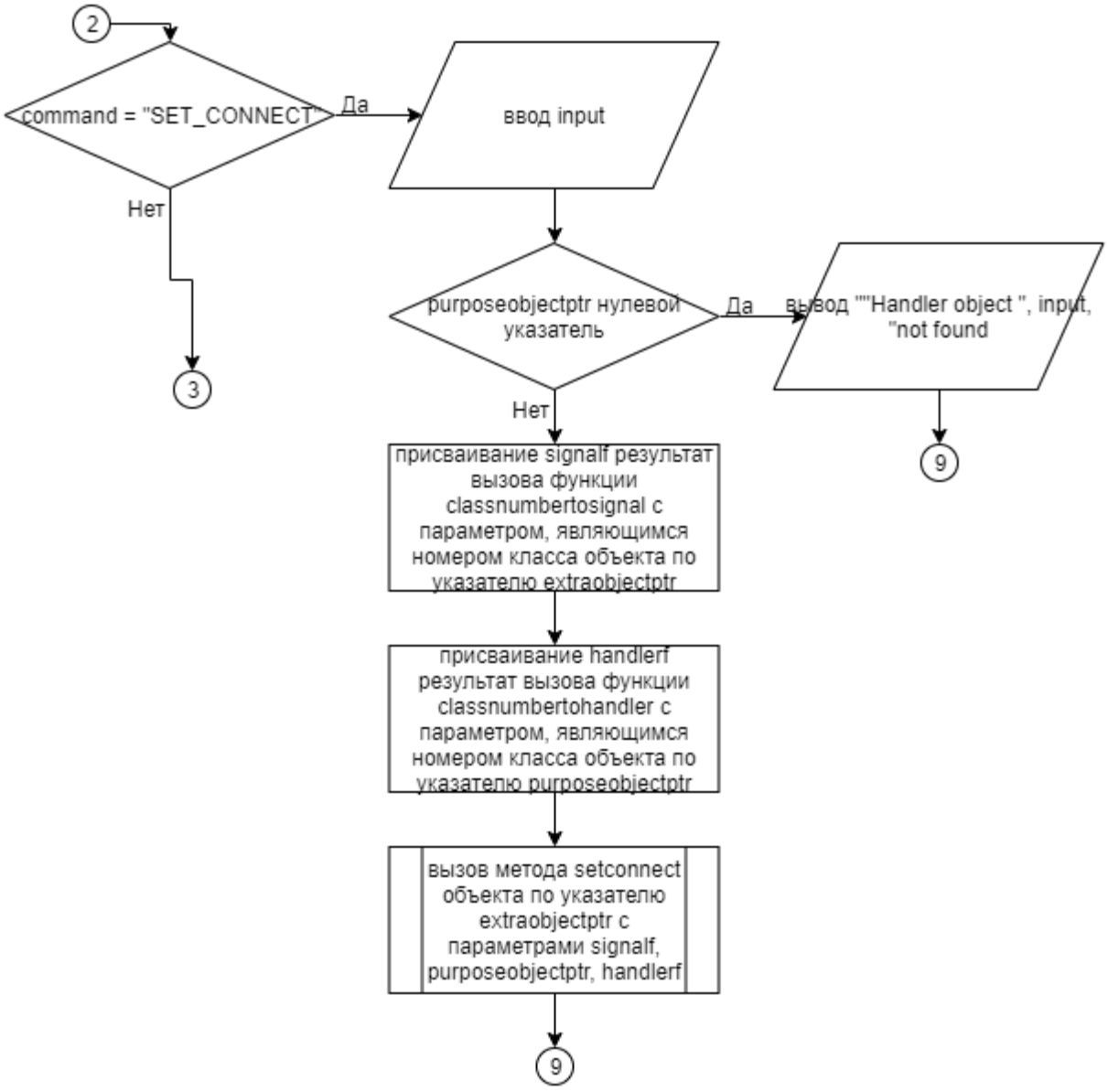

Рисунок 20 – Блок-схема алгоритма

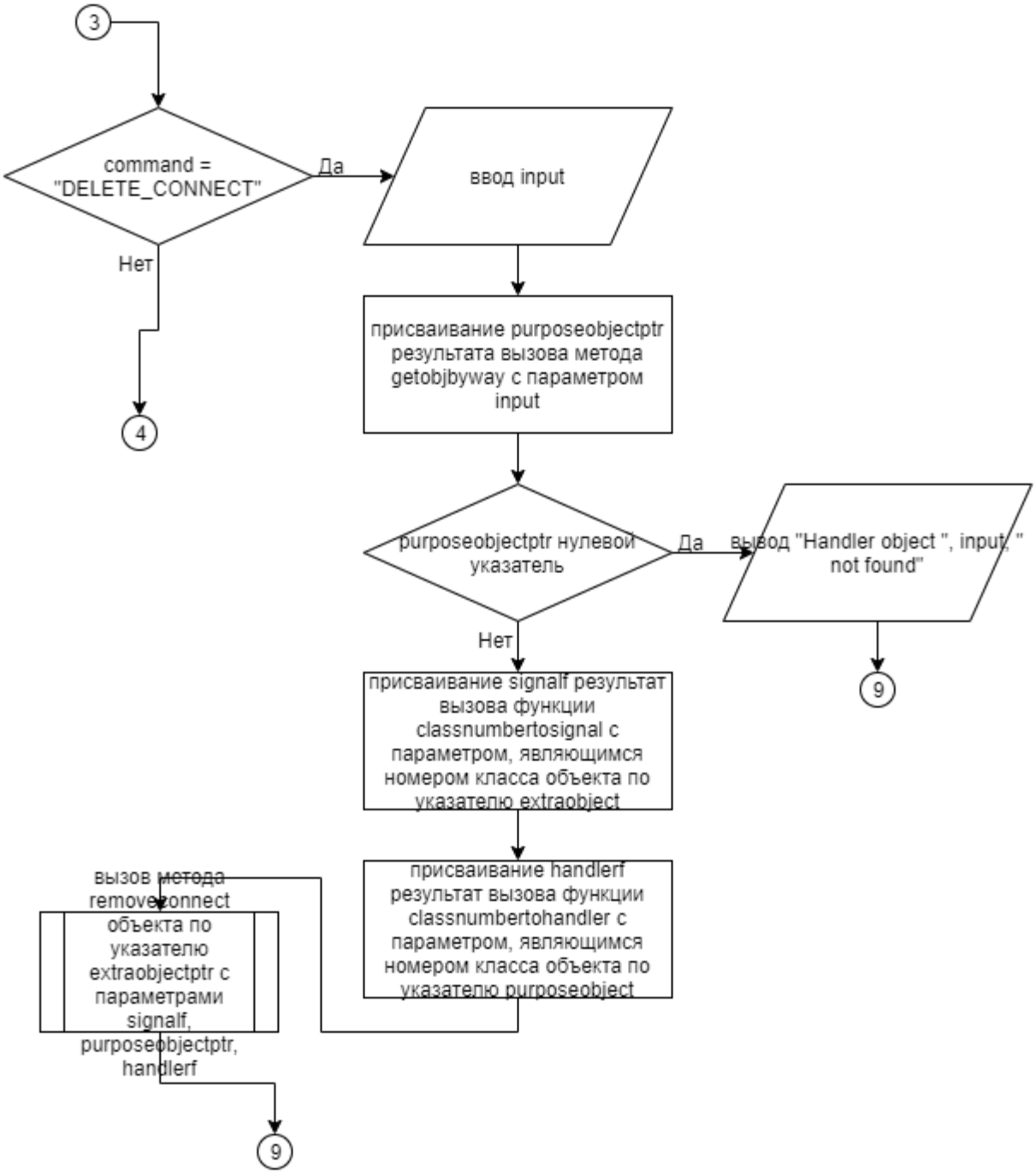

Рисунок 21 – Блок-схема алгоритма

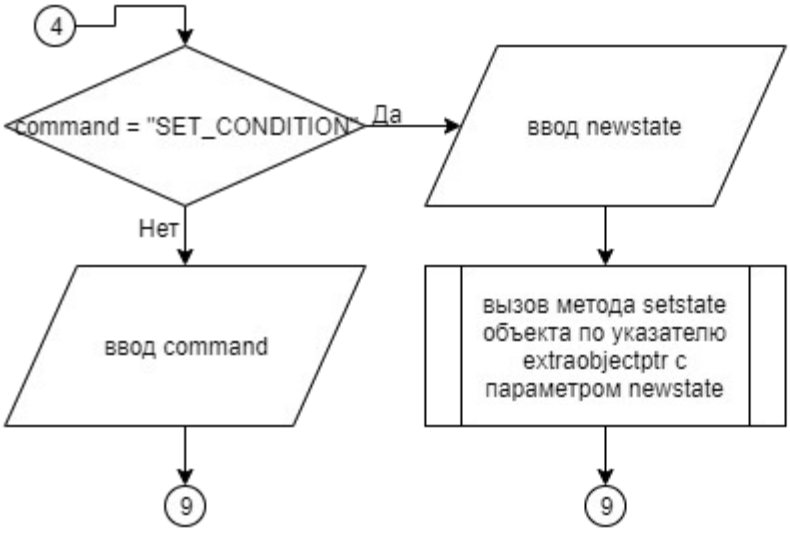

Рисунок 22 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл app.cpp

Листинг 1 – app.cpp

```
#include "app.h"
#include <iostream>
#include <string>
#include <stack>

using namespace std;

app::app(base* parent):base(parent){}

TYPE_SIGNAL classnumbertosignal(int classnum)
{
   switch(classnum)
   {
      case 1:
         return SIGNAL_D(app::signal);
      case 2:
         return SIGNAL_D(cl_2::signal);
      case 3:
         return SIGNAL_D(cl_3::signal);
      case 4:
         return SIGNAL_D(cl_4::signal);
      case 5:
         return SIGNAL_D(cl_5::signal);
      case 6:
         return SIGNAL_D(cl_6::signal);
      default:
         return nullptr;
   }
}

TYPE_HANDLER classnumbertohandler(int classnum)
{
   switch(classnum)
   {
      case 1:
         return HANDLER_D(app::handler);
      case 2:
         return HANDLER_D(cl_2::handler);
      case 3:
```

```
            return HANDLER_D(cl_3::handler);
         case 4:
            return HANDLER_D(cl_4::handler);
         case 5:
            return HANDLER_D(cl_5::handler);
         case 6:
            return HANDLER_D(cl_6::handler);
         default:
            return nullptr;
      }
   }

   int app::start_app() {
      TYPE_SIGNAL signalf;
      TYPE_HANDLER handlerf;
      this->setstatebranch(1);
      string command, input, message;
      int newstate;
      base* extraobjectptrt;
      base* purposeobjectpt;
      this->printbranch();
      cin >> command;
      while (command != "END")
      {
         cin >> input;
         extraobjectptrt = this->getobjbyway(input);
         if (extraobjectptrt == nullptr)
         {
            cout << endl << "Object " << input << " not found";
            cin >> input;
            continue;
         }
         if (command == "EMIT")
         {
            getline(cin, message);
            int n = extraobjectptrt->getclassnumber();
            extraobjectptrt->emitsignal(classnumbertosignal(n), message);
         }
         else if (command == "SET_CONNECT")
         {
            cin >> input;
            purposeobjectpt = this->getobjbyway(input);
            if (purposeobjectpt == nullptr)
            {
               cout << endl << "Handler object " << input << " not found";
               continue;
            }
            signalf = classnumbertosignal(extraobjectptrt->getclassnumber());
            handlerf = classnumbertohandler(purposeobjectpt->getclassnumber());
            extraobjectptrt->setconnect(signalf, purposeobjectpt, handlerf);
         }
         else if (command == "DELETE_CONNECT")
         {
            cin >> input;
            purposeobjectpt = this->getobjbyway(input);
```

```
            if (purposeobjectpt == nullptr)
            {
               cout << endl << "Handler object " << input << " not found";
               continue;
            }
            signalf = classnumbertosignal(extraobjectptrt->getclassnumber());
            handlerf = classnumbertohandler(purposeobjectpt->getclassnumber());
            extraobjectptrt->removeconnect(signalf, purposeobjectpt, handlerf);
         }
         else if (command == "SET_CONDITION")
         {
            cin >> newstate;
            extraobjectptrt->setstate(newstate);
         }
         cin >> command;
      }
      return 0;
   }
   int app::getclassnumber()
   {
      return 1;
   }
   void app::signal(string& message)
   {
      cout << endl << "Signal from " << getabsoluteway();
      message += " (class: " + to_string(getclassnumber()) + ")";
   }
   void app::handler(string message)
   {
      cout << endl << "Signal to " << getabsoluteway() << " Text: " << message;
   }
   void app::treeobj()
   {
      cout << "Object tree";
      string way, descenName;
      int classNum;
      cin >> descenName;
      this->setname(descenName);
      base* parentnodeptr;
      base* ptr_this = this;
      cin >> way;
      while (way != "endtree")
      {
         cin >> descenName >> classNum;
         parentnodeptr = ptr_this->getobjbyway(way);
         if (parentnodeptr == nullptr)
         {
            this->printbranch();
            cout << endl << "The head object " << way << " is not found";
            exit(1);
         }
         if (parentnodeptr->getdescenbyname(descenName) != nullptr)
         {
            cout << endl << way << " Dubbing the names of subordinate objects";
         }
```

```
         else
         {
            switch (classNum)
            {
               case 1:
                  ptr_this = new app(parentnodeptr);
               break;
               case 2:
                  ptr_this = new cl_2(parentnodeptr, descenName);
                  break;
               case 3:
                  ptr_this = new cl_3(parentnodeptr, descenName);
                  break;
               case 4:
                  ptr_this = new cl_4(parentnodeptr, descenName);
                  break;
               case 5:
                  ptr_this = new cl_5(parentnodeptr, descenName);
                  break;
               case 6:
                  ptr_this = new cl_6(parentnodeptr, descenName);
                  break;
               default:
                  break;
            }
         }
         cin >> way;
      }
      base* targetpt;
      string targetway;
      cin >> way;
      while (way != "end_of_connections")
      {
         cin >> targetway;
         parentnodeptr = getobjbyway(way);
         targetpt = getobjbyway(targetway);
         TYPE_SIGNAL      signalf      =      classnumbertosignal(parentnodeptr-
>getclassnumber());
         TYPE_HANDLER       handlerf       =       classnumbertohandler(targetpt-
>getclassnumber());
         parentnodeptr->setconnect(signalf, targetpt, handlerf);
         cin >> way;
      }
   }
```

5.2 Файл app.h

Листинг 2 – app.h

```
   #ifndef __APP__H
```

```
#define __APP__H
#include "base.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"

class app: public base
{
   public:
      app(base* parent);
      void treeobj();
      int start_app();
      int getclassnumber();
      void signal(string & message);
      void handler(string message);
};

#endif
```

5.3 Файл base.cpp

Листинг 3 – base.cpp

```
#include "base.h"
#include <stack>
base::base(base* parent, string name): parent(parent), name(name)
{
   if (parent != nullptr)
   {
      parent->descens.push_back(this);
   }
}
base::~base()
{
   base* rootptr = this;
   while (rootptr->getparent() != nullptr)
   {
   rootptr = rootptr->getparent();
   }
   stack<base*> stack;
   stack.push(rootptr);
   while (!stack.empty())
   {
      base* ptr = stack.top();
      stack.pop();
      int i = 0;
      while (i < ptr->connects.size())
      {
```

```
            if (ptr->connects[i]->purposeptr == this)
            {
               ptr->connects.erase(ptr->connects.begin() + i);
               delete ptr->connects[i];
            }
            else
            {
               i++;
            }
         }
         for (i = 0; i < ptr->descens.size(); ++i)
         {
            stack.push(ptr->descens[i]);
         }
      }
      while (!descens.empty())
      {
         base* tmpptr = descens[0];
         descens.erase(descens.begin());
         delete tmpptr;
      }
   }
   string base::getname() const
   {
      return name;
   }
   base* base::getparent() const
   {
      return parent;
   }
   bool base::setname(string name1)
   {
      if (getparent() != nullptr && getparent() -> getdescenbyname(name1) !
   =nullptr)
      {
         return false;
      }
      name = name1;
      return true;
   }
   base* base::getdescenbyname(string name)
   {
      for (auto descen: descens)
      {
         if (descen->name == name)
         {
            return descen;
         }
      }
      return nullptr;
   }
   base* base::findobjonbranch(string s_object_name)
   {
      base* found = nullptr;
      queue <base*> elementsqueue;
```

```
   elementsqueue.push(this);
   while(!elementsqueue.empty())
   {
      base* elem = elementsqueue.front();
      elementsqueue.pop();
      if (elem->name == s_object_name)
      {
         if (found != nullptr)
         {
            return nullptr;
         }
         else
         {
            found = elem;
         }
      }
      for (int i = 0; i < elem->descens.size();i++)
      {
         elementsqueue.push(elem->descens[i]);
      }
   }
   return found;
}
base* base::findobjontree(string s_object_name)
{
   if (parent != nullptr)
   {
      return parent->findobjontree(s_object_name);
   }
   else
   {
      return findobjonbranch(s_object_name);
   }
}
void base::printbranch(int level)
{
   cout << endl;
   for (int i = 0; i < level; ++i)
   {
      cout << "    ";
   }
   cout << this->getname();
   for (int i = 0; i < descens.size(); ++i)
   {
      descens[i]->printbranch(level + 1);
   }
}
void base::printbranchwithstate(int level)
{
   cout << endl;
   for (int i = 0; i < level; ++i)
   {
      cout << " ";
   }
   if (this->state != 0)
```

```
        {
            cout << this->getname() << " is ready";
        }
        else
        {
            cout << this->getname() << " is not ready";
        }
        for (int i= 0; i < descens.size(); ++i)
        {
            descens[i]->printbranchwithstate(level + 1);
        }
    }
    void base::setstate(int state)
    {
        if (parent == nullptr || parent->state != 0)
        {
            this->state = state;
        }
        if (state == 0)
        {
            this->state = state;
            for (int i = 0; i < descens.size(); i++)
            {
                descens[i]->setstate(state);
            }
        }
    }
    bool base::setparent(base* newparent)
    {
        if (this->getparent() == newparent)
        {
            return true;
        }
        if (this->getparent() == nullptr || newparent == nullptr)
        {
            return false;
        }
        if (newparent->getdescenbyname(this->getname()) != nullptr)
        {
            return false;
        }
        stack<base*> stack;
        stack.push(this);
        while (!stack.empty())
        {
            base* currentnode = stack.top();
            stack.pop();
            if (currentnode == newparent)
            {
                return false;
            }
            for (int i = 0; i < currentnode->descens.size(); ++i)
            {
                stack.push(currentnode->descens[i]);
            }
```

```
        }
        vector<base*> & v = this->getparent()->descens;
        for (int i = 0; i < v.size(); ++i)
        {
           if (v[i]->getname() == this->getname())
           {
              v.erase(v.begin() + i);
              newparent->descens.push_back(this);
              return true;
           }
        }
        return false;
     }
     void base::deletedescenbyname(string descenname)
     {
        vector<base*> & v = this->descens;
        for (int i = 0; i < v.size(); ++i)
        {
           if (v[i]->getname() == descenname)
           {
              delete v[i];
              v.erase(v.begin() + i);
              return;
           }
        }
     }
     base* base::getobjbyway(string way)
     {
        if (way.empty())
        {
           return nullptr;
        }
        if (way == ".")
        {
           return this;
        }
        if (way[0] == '.')
        {
           return findobjonbranch(way.substr(1));
        }
        if (way.substr(0, 2) == "//")
        {
           return this->findobjontree(way.substr(2));
        }
        if (way[0] != '/')
        {
           size_t slashindex = way.find('/');
           base* descenptr = this->getdescenbyname(way.substr(0,slashindex));
           if (descenptr == nullptr || slashindex == string::npos)
           {
              return descenptr;
           }
           return descenptr->getobjbyway(way.substr(slashindex + 1));
        }
        base* rootptr = this;
```

```
   while (rootptr ->getparent() != nullptr)
   {
      rootptr = rootptr->getparent();
   }
   if (way == "/")
   {
      return rootptr ;
   }
   return rootptr->getobjbyway(way.substr(1));
}
void base::setconnect(TYPE_SIGNAL signalptr, base* purposeptr, TYPE_HANDLER
handlerptr)
{
   for (int i = 0; i < connects.size(); ++i)
   {
      if (connects[i]->signalptr == signalptr && connects[i]->purposeptr
==purposeptr && connects[i]->handlerptr == handlerptr)
      {
         return;
      }
   }
   connect * newconnect = new connect();
   newconnect->signalptr = signalptr;
   newconnect->purposeptr = purposeptr;
   newconnect->handlerptr = handlerptr;
   connects.push_back(newconnect);
}
void    base::removeconnect(TYPE_SIGNAL    signalptr,    base*    purposeptr,
TYPE_HANDLER handlerptr)
{
   for (int i = 0; i < connects.size(); ++i)
   {
      if (connects[i]->signalptr == signalptr && connects[i]->purposeptr ==
purposeptr && connects[i]->handlerptr == handlerptr)
      {
         delete connects[i];
         connects.erase(connects.begin() + i);
         return;
      }
   }
}
void base::emitsignal(TYPE_SIGNAL signalptr, string & command)
{
   if (this->state == 0)
   {
      return;
   }
   (this->*signalptr)(command);
   for (int i = 0; i < connects.size(); ++i)
   {
      if (connects[i]->signalptr == signalptr)
      {
         TYPE_HANDLER handlerptr = connects[i]->handlerptr;
         base* purposeptr = connects[i]->purposeptr;
         if (purposeptr->state != 0)
```

```
            {
               (purposeptr->*handlerptr)(command);
            }
         }
      }
   }
   string base::getabsoluteway()
   {
      string result;
      stack<string> stack;
      base* rootptr = this;
      while (rootptr->getparent() != nullptr)
      {
         stack.push(rootptr->getname());
         rootptr = rootptr->getparent();
      }
      while (!stack.empty())
      {
         result += '/' + stack.top();
         stack.pop();
      }
      if (result.empty())
      {
         return "/";
      }
      return result;
   }
   int base::getclassnumber()
   {
      return 0;
   }
   void base::setstatebranch(int newstate)
   {
      if (getparent() != nullptr && getparent()->state == 0)
      {
         return;
      }
      setstate(newstate);
      for (int i = 0; i < descens.size(); ++i)
      descens[i]->setstatebranch(newstate);
   }
```

5.4 Файл base.h

Листинг 4 – base.h

```
   #ifndef __BASECLASS__H
   #define __BASECLASS__H
   #include <iostream>
   #include <vector>
```

```
#include <string>
#include <queue>
#include <stack>

using namespace std;

#define SIGNAL_D(signalf)(TYPE_SIGNAL)(&signalf);
#define HANDLER_D(handlerf)(TYPE_HANDLER)(&handlerf);

class base;
typedef void(base::*TYPE_SIGNAL)(string &);
typedef void (base::*TYPE_HANDLER)(string);
class base
{
   private:
      struct connect
      {
         TYPE_SIGNAL signalptr;
         base* purposeptr;
         TYPE_HANDLER handlerptr;
      };
      int state=0;
      base* parent;
      vector<base*>descens;
      string name;
      vector <connect*> connects;
   public:
      base(base* parent, string name="Object_root");
      ~base();
      bool setname(string name);
      string getname() const;
      base* getparent() const;
      base* getdescenbyname(string name);
      base* findobjonbranch(string name);
      base* findobjontree(string name);
      void printbranch(int level=0);
      void printbranchwithstate(int level=0);
      void setstate(int state);
      bool setparent(base* newparent);
      void deletedescenbyname(string descenname);
      base* getobjbyway(string way);
      void setconnect(TYPE_SIGNAL signalptr, base* purposeptr, TYPE_HANDLER
handlerptr);
      void removeconnect(TYPE_SIGNAL signalptr, base* purposeptr,
TYPE_HANDLER handlerptr);
      void emitsignal(TYPE_SIGNAL signalptr, string & command);
      string getabsoluteway();
      virtual int getclassnumber();
      void setstatebranch(int newstate);
};

#endif
```

5.5 Файл cl_2.cpp

Листинг 5 – cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(base*  p_head_object,  string  s_object_name):base(p_head_object,
s_object_name){}
int cl_2::getclassnumber()
{
   return 2;
}
void cl_2::signal(string &message)
{
   cout<<endl<<"Signal from "<<getabsoluteway();
   message+=" (class: "+to_string(getclassnumber()) + ")";
}
void cl_2::handler(string message)
{
   cout<<endl<<"Signal to "<<getabsoluteway()<<" Text: "<<message;
}
```

5.6 Файл cl_2.h

Листинг 6 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "base.h"

class cl_2:public base
{
   public:
      cl_2(base* p_head_object, string s_object_name);
      int getclassnumber();
      void signal(string & message);
      void handler(string message);
};
#endif
```

5.7 Файл cl_3.cpp

Листинг 7 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(base*  p_head_object,  string  s_object_name):base(p_head_object,
s_object_name){}
int cl_3::getclassnumber()
{
   return 3;
}
void cl_3::signal(string &message)
{
   cout<<endl<<"Signal from "<<getabsoluteway();
   message+=" (class: "+to_string(getclassnumber()) + ")";
}
void cl_3::handler(string message)
{
   cout<<endl<<"Signal to "<<getabsoluteway()<<" Text: "<<message;
}
```

5.8 Файл cl_3.h

Листинг 8 – cl_3.h

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

class cl_3:public base
{
   public:
      cl_3(base* p_head_object, string s_object_name);
      int getclassnumber();
      void signal(string &message);
      void handler(string message);
};
#endif
```

5.9 Файл cl_4.cpp

Листинг 9 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(base*  p_head_object,  string  s_object_name):base(p_head_object,
s_object_name){}
int cl_4::getclassnumber()
{
   return 4;
}
void cl_4::signal(string &message)
{
   cout<<endl<<"Signal from "<<getabsoluteway();
   message+=" (class: "+to_string(getclassnumber()) + ")";
}
void cl_4::handler(string message)
{
   cout<<endl<<"Signal to "<<getabsoluteway()<<" Text: "<<message;
}
```

5.10 Файл cl_4.h

Листинг 10 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include "base.h"

class cl_4:public base
{
   public:
      cl_4(base* p_head_object, string s_object_name);
      int getclassnumber();
      void signal(string &message);
      void handler(string message);
};
#endif
```

5.11 Файл cl_5.cpp

Листинг 11 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(base*  p_head_object,  string  s_object_name):base(p_head_object,
s_object_name){}
int cl_5::getclassnumber()
{
   return 5;
}
void cl_5::signal(string &message)
{
   cout<<endl<<"Signal from "<<getabsoluteway();
   message+=" (class: "+to_string(getclassnumber()) + ")";
}
void cl_5::handler(string message)
{
   cout<<endl<<"Signal to "<<getabsoluteway()<<" Text: "<<message;
}
```

5.12 Файл cl_5.h

Листинг 12 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include "base.h"

class cl_5:public base
{
   public:
      cl_5(base* p_head_object, string s_object_name);
      int getclassnumber();
      void signal(string &message);
      void handler(string message);
};
#endif
```

5.13 Файл cl_6.cpp

Листинг 13 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(base*  p_head_object,  string  s_object_name):base(p_head_object,
s_object_name){}
int cl_6::getclassnumber()
{
   return 6;
}
void cl_6::signal(string &message)
{
   cout<<endl<<"Signal from "<<getabsoluteway();
   message+=" (class: "+to_string(getclassnumber()) + ")";
}
void cl_6::handler(string message)
{
   cout<<endl<<"Signal to "<<getabsoluteway()<<" Text: "<<message;
}
```

5.14 Файл cl_6.h

Листинг 14 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "base.h"

class cl_6:public base
{
   public:
      cl_6(base* p_head_object, string s_object_name);
      int getclassnumber();
      void signal(string &message);
      void handler(string message);
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

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

int main()
{
   app ob_app(nullptr);
   ob_app.treeobj();
   return (ob_app.start_app());
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 32.

Таблица 32 – 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 EMIT /object_s1 Send message 43 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	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_1	Object tree appls_root The head object object_1 is not found	Object tree appls_root The head object object_1 is not found
/ 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 tree / The head object object_s1 is not found	Object tree / The head object object_s1 is not found

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
/ /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_CONDITION /object_s2/object_s4 -3 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		
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	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
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 DELETE_CONNECT /object_s2/object_s4 /object_s2/object_s6 EMIT /object_s2/object_s4 Send message 2 SET_CONNECT /object_s2/object_s4 /object_s2/object_s6 SET_CONDITION / 0 EMIT /object_s2/object_s4 Send message 3 SET_CONDITION / 1 SET_CONDITION /object_s2 1 EMIT /object_s2/object_s4 Send message 4 SET_CONDITION /object_s2/object_s4 1 EMIT /object_s2/object_s4 Send message 5 SET_CONDITION /object_s2/object_s6 1 EMIT /object_s2/object_s4 Send message 6 EMIT /object_s1 Send message 7 SET_CONNECT /object_s1 /object_s2/object_s4 EMIT /object_s1 Send message 8 SET_CONDITION /object_s2/object_s4 1 EMIT /object_s1 Send	/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 / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 5 (class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 6 (class: 4) Signal to /object_s2/object_s6 Text: Send message 6 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s4 Text: Send message 10 (class: 3)	/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 / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 5 (class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 6 (class: 4) Signal to /object_s2/object_s6 Text: Send message 6 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s4 Text: Send message 10 (class: 3)

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
message 9 SET_CONDITION /object_s1 1 EMIT /object_s1 Send message 10 END		

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).