Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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

Актуальность работы: Деревья иерархии широко используются для организации и представления данных в различных областях, таких как computer science, биоинформатика, управление проектами и во многих других. Курсовая работа актуальна, так как позволяет освоить принципы построения и использования деревьев иерархии, что является важным навыком для программиста. В эпоху больших данных и сложных информационных систем, умение эффективно организовывать и управлять данным становится ключевым навыком программиста. Полученные навыки будут ценны для работы в сферах, где требуется организованное представление и управление данными, таких как базы данных, файловые системы, графы и сети, а так же реализации сложных алгоритмов и оптимизации процессов.

Объекто-ориентированное программирование(ООП) представляет собой методологию программирования, в которой основными концепциями являются объекты и классы. ООП позволяет создавать программы, моделирующие реальные объекты и их взаимодействие, что делает код более понятным, модульным и легко поддерживаемым. Язык программирования C++ тесно связан с ООП, так как представляет все необходимые инструменты для реализации объектов, наследования, полиморфизма и инкапсуляции.

Цель работы: получение практических навыков в области объекто-ориентированного программирования с использованием алгоритмического языка C++.

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:

- объект класса cl_custom_2 предназначен для создание нового объекта в дереве иерархии;
- объект класса cl_custom_3 предназначен для создание нового объекта в дереве иерархии;
- объект класса cl_custom_4 предназначен для создание нового объекта в дереве иерархии;
- объект класса cl_custom_5 предназначен для создание нового объекта в дереве иерархии;
- объект класса cl_custom_6 предназначен для создание нового объекта в дереве иерархии;
- объект cin класса потокового ввода предназначен для функционирования системы;
- объект cout класса потокового вывода предназначен для функционирования системы;
- объект obj_custom_app класса cl_custom_application предназначен для запуска приложения;
- оператор new - выделение динамической памяти для объектов;
- оператор return - возврат значения из функции;
- оператор delete - освобождение памяти;
- if...else - условные операторы;
- for - оператор цикла с счётчиком;
- while - оператор цикла с предусловием;
- оператор break - прерывания работы цикла;
- метод size - определения длины переменной класса Vector;

- метод push_back - добавления элемента в конец переменной класса Vector.

Класс cl_custom_base:

- свойства/поля:
  - поле , указатель, на объект родитель текущего пользовательского объекта:
    - наименование — p_main_custom_object;
    - тип — cl_custom_base;
    - модификатор доступа — private;
  - поле , поле - хранение указателей на подчиненные объекты:
    - наименование — p_sub_customs;
    - тип — vector<cl_custom_base*>;
    - модификатор доступа — private;
  - поле , вектор связей сигналов и обработчиков:
    - наименование — main_connections;
    - тип — vector<signals*>;
    - модификатор доступа — private;
  - поле , строка, представляющая имя пользовательского объекта:
    - наименование — custom_name;
    - тип — строковый;
    - модификатор доступа — private;
  - поле , хранение состояния объекта:
    - наименование — status;
    - тип — целочисленный;
    - модификатор доступа — private;
- функционал:
  - метод cl_custom_base — конструктор;
  - метод ~cl_custom_base — деструктор;

- метод set_custom_name — установка имени объекта;
- метод get_custom_name — получение имени объекта;
- метод get_main_absolute_coords — метод получения абсолютных координат;
- метод set_head_object — установка нового головного объекта;
- метод get_main_custom — получение главного объекта;
- метод connections_set_main — добавление новой связи;
- метод connections_delete_main — удаление связи;
- метод connections_main_signal — выдача сигнала;
- метод signal_method — передача сигнала;
- метод handler_method — получение сигнала;
- метод indentation_method — вывод имени текущего объекта;
- метод indent_call_method — вызов метода;
- метод return_number_class — возврат номера класса;
- метод set_object_ready — приведение всех объектов в состоянии готовности;
- метод get_object_path — получение указателя на объект в составе дерева иерархии;
- метод search_current — поиск объекта по имени;
- метод search_tree — поиск подчиненного объекта корневого объекта по имени;
- метод get_sub_customs — поиск подчиненного объекта по заданному имени;
- метод set_state — установка состояния объекта и подчиненных объектов;
- метод delete_custom_sub_name — удаление подчинённого объекта с заданным именем.

Класс cl_custom_2:

- функционал:
    - метод cl_custom_2 — конструктор;
    - метод return_number_class — возврат номера класса.

Класс cl_custom_3:

- функционал:
    - метод cl_custom_3 — конструктор;
    - метод return_number_class — возврат номера класса.

Класс cl_custom_4:

- функционал:
    - метод cl_custom_4 — конструктор;
    - метод return_number_class — возврат номера класса.

Класс cl_custom_5:

- функционал:
    - метод cl_custom_5 — конструктор;
    - метод return_number_class — возврат номера класса.

Класс cl_custom_6:

- функционал:
    - метод cl_custom_6 — конструктор;
    - метод return_number_class — возврат номера класса.

Класс cl_custom_application:

- функционал:
    - метод cl_custom_application — конструктор;
    - метод build_tree_objects — создание дерева иерархии;
    - метод exec_custom_app — запуск системы.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_custom_base			основная логика управления пользовательскими объектами и их иерархией	
		cl_custom_2	public		2
		cl_custom_3	public		3
		cl_custom_4	public		4
		cl_custom_5	public		5
		cl_custom_6	public		6
		cl_custom_application	public		7
2	cl_custom_2			создузлы дерева иерархии объектов	
3	cl_custom_3			узлы дерева иерархии объектов	
4	cl_custom_4			узлы дерева иерархии объектов	
5	cl_custom_5			узлы дерева иерархии объектов	
6	cl_custom_6			узлы дерева иерархии объектов	
7	cl_custom_application			класс корневого объекта	

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм конструктора класса cl_custom_base

Функционал: конструктор.

Параметры: p_main_custom_object - указатель на главный объект, custom_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация указателя p_main_custom_object текущего объекта значением p_main_custom_object, которое передано в качестве аргумента	2
2		присвоение значения аргумента custom_name члену класса custom_name текущего объекта	3
3	не является ли p_main_custom_object пустым?	добавление объекта в вектор подпользовательских объектов p_sub_customs родительского класса p_main_custom_object	∅
			∅

3.2 Алгоритм деструктора класса cl_custom_base

Функционал: деструктор.

Параметры: отсутствуют.

Алгоритм деструктора представлен в таблице 3.

Таблица 3 – Алгоритм деструктора класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация целочисленной переменной i = 0	2
2	i < размера вектора p_sub_customs	i++; удаление p_sub_customs[i], освобождение памяти	∅
			∅

3.3 Алгоритм метода set_custom_name класса cl_custom_base

Функционал: установка имени объекта.

Параметры: new_custom_name - новое пользовательское имя.

Возвращаемое значение: если имя уникально - custom_name, True / если нет - False.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_custom_name класса cl_custom_base

№	Предикат	Действия	№ перехода
1	существует ли p_main_custom_object	инициализация целочисленной переменной i = 0	2
			4
2	i < размера вектора p_sub_customs		3
			4
3	совпадает ли имя текущего элемента с new_custom_name	возврат значение false	4
			4
4		присвоение значения new_custom_name переменной custom_name	5

№	Предикат	Действия	№ перехода
5		возврат значения true	∅

3.4 Алгоритм метода get_custom_name класса cl_custom_base

Функционал: получение имени объекта.

Параметры: отсутствуют.

Возвращаемое значение: имя текущего объекта типа cl_custom_base.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_custom_name класса cl_custom_base

№	Предикат	Действия	№ перехода
1		возврат значения поля custom_name текущего объекта	∅

3.5 Алгоритм метода get_main_absolute_coords класса cl_custom_base

Функционал: метод получения абсолютных координат.

Параметры: отсутствуют.

Возвращаемое значение: возврат имени - пути от текущего объекта до главного род объекта.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_main_absolute_coords класса cl_custom_base

№	Предикат	Действия	№ перехода
1		создание указателя ptr и инициализация его текущим объектом(this)	2
2		создание строки text и инициализация ее значением "/" , добавляя ptr->custom_name через	3

№	Предикат	Действия	№ перехода
		конкатенацию	
3	у объекта на который указывает ptr, сущесвует p_main_custom_object	перемещение указателя ptr на объект , на который ссылается p_main_custom_object текущего объекта	3
			4
4	ptr и this совпадают	установление значения text на "/"	8
			5
5		инициализация ptr текущим объектом this	6
6	родительский объект p_main_custom_object у родительского объекта текущего объекта существует	перемещение указателя ptr на объект , на который ссылается p_main_custom_object текущего объекта	8
			7
7		конструирование полного имени, добавляя имя родительского объекта к началу сущесвующий строки	8
8		возврат значения text	∅

3.6 Алгоритм метода set_head_object класса cl_custom_base

Функционал: установка нового головного объекта.

Параметры: new_p_head_object - указатель cl_custom_base.

Возвращаемое значение: возврат булевого значения.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода set_head_object класса cl_custom_base

№	Предикат	Действия	№ перехода
1	текущий объект this является	возврат значения true	2

№	Предикат	Действия	№ перехода
	дочерним элементом new_p_head_object		
			2
2	текущий объект главного объекта get_main_custom не существует	возврат значения false	3
			3
3		создание временного указателя quet того же типа что и cl_custom_object и инициализация его значением new_p_head_object	4
4	у объекта на который указывает quet, есть главный объект get_main_custom		5
			7
5	не указывает ли quet на текущий объект this	возврат значения false	6
			6
6		перемещение указателя quet вверх по иерархии объектов	7
7		создание ссылки s_base_ob на вектор указателей cl_custom_base из p_main_custom_object	8
8		инициализация целочисленной переменной i = 0	9
9	i < количества дочерних объектов главного объекта	i++	10
			13
10	является ли текущий объект this элементом s_base_ob	удаление текущего объекта из списка дочерних элементов своего текущего объекта	11
			13
11		установка нового главного объекта для текущего	12

№	Предикат	Действия	№ перехода
		объекта	
12		добавление текущего объекта в список дочерних элементов нового главного объекта	13
13		возврат значения false	∅

3.7 Алгоритм метода get_main_custom класса cl_custom_base

Функционал: получение главного объекта.

Параметры: отсутствуют.

Возвращаемое значение: значение p_main_custom_object.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода get_main_custom класса cl_custom_base

№	Предикат	Действия	№ перехода
1		возврат указателя на родительский объект текущего объекта	∅

3.8 Алгоритм метода connections_set_main класса cl_custom_base

Функционал: добавление новой связи.

Параметры: p_main_signal - указатель на метод сигнала, p_main_target - указатель на целевой объект , p_main_handler - указатель на метод обработчика.

Возвращаемое значение: отсутствуют.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода connections_set_main класса cl_custom_base

№	Предикат	Действия	№ перехода
1		объявление переменной ob1 типа signals*	2

№	Предикат	Действия	№ перехода
2		инициализация целочисленной переменной i = 0	3
3	i < размера main_connections		4
			5
4	существует ли уже идентичная связь в main_connections, существует ли одна из связей, сравнение типа сигнала , целевой объект, обработчик текущей связи с переданными значениям	возврат return	5
			5
5		создание объекта структуры для хранения информации о новой связи	6
6		устанавливаем тип сигнала для новой связи	7
7		устанавливаем целевой объект для новой связи	8
8		добавление новой связи в список	∅

3.9 Алгоритм метода connections_delete_main класса cl_custom_base

Функционал: удаление связи.

Параметры: p_main_signal - указатель на метод сигнала, p_main_target - указатель на целевой объект , p_main_handler - указатель на метод обработчика.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода connections_delete_main класса cl_custom_base

№	Предикат	Действия	№ перехода
1		создаем итератор для перебора списка связей	2
2	найдена связь с заданными параметрами		3
			∅
3	совпадает ли тип сигнала связи с переданным типом p_main_signal, указывает ли целевой объект связи на тот же объект, что и переданный p_main_target, совпадает ли метод текущей связи с переданным методом p_main_handler	освобождение памяти - выделенную под структуру связи	4
			5
4		удаление связи из списка итератором	∅
5		переходим к следующей связи	∅

3.10 Алгоритм метода connections_main_signal класса cl_custom_base

Функционал: выдача сигнала.

Параметры: p_main_signal - указатель на метод сигнала, строковая переменная text.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода connections_main_signal класса cl_custom_base

№	Предикат	Действия	№ перехода
1		обработчик сигнала	2
2		целевой объект	3
3	текущий объект не находится в состоянии готовности	возврат return	4
			4
4		вызываем метод сигнала у текущего объекта, передавая текст сигнала	5
5		инициализация целочисленной переменной i = 0	6
6		перебираем список связей	7
7	i меньше main_connections.size()		8
			∅
8	найдена связь с совпадающим типом сигнала и целевой объект находится в состоянии готовности	получаем обработчик из связи	9
			∅
9		получаем целевой объект из связи	10
10		вызов метода обработчика у целевого объекта, передавая сигнала	∅

3.11 Алгоритм метода signal_method класса cl_custom_base

Функционал: передача сигнала.

Параметры: text - текст сигнала.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода signal_method класса cl_custom_base

№	Предикат	Действия	№ перехода
1		формируем текст сигнала, вывод на экран " Text: " , значение переменной text, вывод " (class: " , вывод to_string(return_number_class()) , вывод ")"	2
2		вывод "Signal from " и вызов get_main_absolute_coords()	∅

3.12 Алгоритм метода handler_method класса cl_custom_base

Функционал: получение сигнала.

Параметры: text - строковая переменная.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода handler_method класса cl_custom_base

№	Предикат	Действия	№ перехода
1		вывод "Signal to ", вызов get_main_absolute_coords(), вывод text	∅

3.13 Алгоритм метода indentation_method класса cl_custom_base

Функционал: вывод имени текущего объекта.

Параметры: отсутствует.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода indentation_method класса cl_custom_base

№	Предикат	Действия	№ перехода
1	главный объект существует	переход на следующую строку	2

№	Предикат	Действия	№ перехода
			2
2		инициализация переменной i_space = 0	3
3		инициализация указателя на текущий объект	4
4	существует ли главный объект у текущего объекта	переходим к главному объекту	5
			6
5		увеличение счетчиков отступов	6
6		инициализация целочисленной переменной i = 0	7
7	i меньше i_space	выводим отступ в 4 пробела	8
			8
8		выводим имя текущего объекта	9
9	по всем подообъектам текущего объекта	рекурсивный вызов метода для каждого подообъекта	∅
			∅

3.14 Алгоритм метода indent_call_method класса cl_custom_base

Функционал: вызов метода.

Параметры: отсутствует.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода indent_call_method класса cl_custom_base

№	Предикат	Действия	№ перехода
1		вызываем метод indentation_method для отступов	∅

3.15 Алгоритм метода return_number_class класса cl_custom_base

Функционал: возврат номера класса.

Параметры: отсутствует.

Возвращаемое значение: возврат номера класса.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода return_number_class класса cl_custom_base

№	Предикат	Действия	№ перехода
1		возврат номера класса	∅

3.16 Алгоритм метода set_object_ready класса cl_custom_base

Функционал: приведение всех объектов в состояние готовности.

Параметры: отсутствует.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода set_object_ready класса cl_custom_base

№	Предикат	Действия	№ перехода
1		установление состояния готовности текущего объекта	2
2		инициализация целочисленной переменной i = 0	3
3	i меньше размера p_sub_customs	рекурсивный вызов метода set_object_ready() для всех подобъектов текущего объекта	3
			∅

3.17 Алгоритм метода get_object_path класса cl_custom_base

Функционал: получение указателя на объект в составе дерева иерархии.

Параметры: path_custom - путь к искомому объекту в иерархии.

Возвращаемое значение: указатель на объект типа cl_custom_base.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода get_object_path класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация вещественной вспомогательная переменная text для хранения подстроки пути	2
2		инициализация целочисленной переменной command_ver для хранения индекса разделителя в пути	3
3		инициализация base - указатель на объект	4
4	путь пустой	возврат nullptr	5
			5
5	путь path_custom равен "/"	создание указателя ptr, который инициализируется текущим объектом this	6
			8
6	метод get_main_custom указывает на родительский объект	обновляется ptr, чтобы указывать на родительский объект объекта ptr = ptr->get_main_custom()	7
			7
7		возврат ptr	8
8	путь path_custom равен "."	возвращаем указатель на объект	9
			9
9	путь начинается с двух слэшей, начинается с двух первых индексов	создание подстроки начиная с третьего символа строки path_custom	10

№	Предикат	Действия	№ перехода
			11
10		выполняется поиск объекта с значением строки path_custom	11
11	путь начинается с точки	создание подстроки начиная со второго символа	12
			13
12		выполнение поиска объекта с значением строки text начиная с текущего объекта	13
13		находим индекс разделителя в пути	14
14	путь абсолютный		15
			21
15	в пути есть разделитель	получение имени текущего уровня вложенности	16
			18
16		находим объект по имени и продолжаем поиск для остальной части пути	17
17	base существует	создание подстроки path_custom, начиная с позиции command_ver + 1 вызов метода get_object_path на объекте base с созданной подстрокой в качестве аргумента return	18
			18
18	разделитель не найден	присвоение переменной text значение path_custom	19
			∅
19		возврат результата вызова метода get_sub_customs с аргументом text	20
20			∅
21	путь относительный		22
			∅
22	в пути есть разделитель	получение имени текущего уровня вложенности	23
			18
23		нахождение объекта по имени и продолжение	24

№	Предикат	Действия	№ перехода
		поиска для остальной части пути	
24	base существует	вызов метода get_object_path на объекте base с созданной подстрокой в качестве аргумента	18
		возврат base	∅

3.18 Алгоритм метода search_current класса cl_custom_base

Функционал: поиск объекта по имени.

Параметры: s_custom_name - хранение имени объекта - который нужно найти найти в методах поиска.

Возвращаемое значение: path_ind - возврат указателя на найденный объект.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода search_current класса cl_custom_base

№	Предикат	Действия	№ перехода
1		очередь для поиска в ширину	2
2		указатель на найденный объект	3
3		добавление текущего объекта в очередь	4
4	очередь не пуста		5
			11
5	текущего объекта совпадает с искомым именем		6
			7
6	путь еще не найден	установление указатель на текущий объект как найденный путь	7
		возврат nullptr	7
7	проход по каждому подчиненному объекту	добавление всех подобъектов текущего объекта в очередь	8
			10

№	Предикат	Действия	№ перехода
8		проходит по всем подчиненным объектам текущего объекта - находящегося в начале очереди q	9
9		добавление каждого подчиненного объекта в конец очереди q	10
10		удаление текущего объекта из очереди	11
11		возврат указателя на найденный объект	∅

3.19 Алгоритм метода search_tree класса cl_custom_base

Функционал: поиск подчиненного объекта корневого объекта по имени.

Параметры: s_custom_name -хранение имени объекта - который нужно найти найти в методах поиска.

Возвращаемое значение: возвращаем подчинненный объект с именем s_custom_name у корневого объекта.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода search_tree класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация указателя на текущий объект	2
2	у текущего объекта есть главный объект	перемещаем указатель на главный объект	2
			3
3		возвращаем подчинненный объект с именем s_custom_name у корневого объекта	∅

3.20 Алгоритм метода get_sub_customs класса cl_custom_base

Функционал: поиск подчиненного объекта по заданному имени.

Параметры: имя подобъекта, который нужно найти.

Возвращаемое значение: возврат указателя на найденный подобъект или если подобъект с заданным именем не найден - возврат nullptr.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода get_sub_customs класса cl_custom_base

№	Предикат	Действия	№ перехода
1		инициализация целочисленной переменной int i = 0	2
2	i меньше размера p_sub_customs	проверка совпадений имен	3
			4
3	совпадает ли имя подчиненного объекта с заданным именем	возврат указателя на найденный подобъект	4
			4
4		если подобъект с заданным именем не найден - возврат nullptr	∅

3.21 Алгоритм метода set_state класса cl_custom_base

Функционал: установка состояния объекта и подчиненных объектов.

Параметры: status_object - переданное состояние объкта.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода set_state класса cl_custom_base

№	Предикат	Действия	№ перехода
1	переданное состояние не равно нулю	инициализация указателем quet на главный объект	2

№	Предикат	Действия	№ перехода
			6
2	есть главные объекты в иерархии		3
			5
3	у главного объекта нет состояния	завершаем выполнение метода	4
			4
4		переходим к следующему главному объекту	5
5		устанавливаем состояние объекта	6
6	переданное состояние равно нулю	перебираем все подобъекты текущего объекта	7
			∅
7	по каждому элементу в векторе p_sub_customs	устанавливаем состояние для каждого подобъекта	7
			8
8		устанавливаем состояние текущего объекта	∅

3.22 Алгоритм метода delete_custom_sub_name класса cl_custom_base

Функционал: удаление подчиненного объекта с заданным именем.

Параметры: sub_name - хранение имени подчиненного объекта.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода delete_custom_sub_name класса cl_custom_base

№	Предикат	Действия	№ перехода
1		получение указателя на подобъект по его имени	2
2	подобъект найден	инициализация целочисленной переменной int i =	3

№	Предикат	Действия	№ перехода
		0	
			∅
3	по всем подобъектам текущего объекта, i меньше p_sub_customs		4
			∅
4	текущий подобъект равен искомому подобъекту	удаление его из списка подобъектов	5
			∅
5		удаление подобъекта	6
6		прерываем цикл - break	∅

3.23 Алгоритм функции main

Функционал: основной алгоритм работы программы.

Параметры: отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения работы программы.

Алгоритм функции представлен в таблице 24.

Таблица 24 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объект obj_custom_app класса cl_custom_application с родительским объектом , указанным как nullptr	2
2		вызов метода build_tree_objects() для объекта obj_custom_app	3
3		вызов метода exec_custom_app для объекта obj_custom_app и возврат результата выполнения этого метода	∅

3.24 Алгоритм конструктора класса cl_custom_2

Функционал: конструктор.

Параметры: p_main_custom_object - указатель на главный объект, custom_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 25.

Таблица 25 – Алгоритм конструктора класса cl_custom_2

№	Предикат	Действия	№ перехода
1		определение конструктора класса cl_custom_2 , который вызывает конструктор базового класса cl_custom_base с передачей ему указателя на главный объект p_main_custom_object и имя объекта custom_name	∅

3.25 Алгоритм метода return_number_class класса cl_custom_2

Функционал: возврат номера класса.

Параметры: отсутствуют.

Возвращаемое значение: возврат целочисленного значения , которое представляет номер класса cl_custom_2.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода return_number_class класса cl_custom_2

№	Предикат	Действия	№ перехода
1		возврат целочисленного значение 2, которое представляет номер класса cl_custom_2	∅

3.26 Алгоритм конструктора класса cl_custom_3

Функционал: конструктор.

Параметры: p_main_custom_object - указатель на главный объект, custom_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 27.

Таблица 27 – Алгоритм конструктора класса cl_custom_3

№	Предикат	Действия	№ перехода
1		определение конструктора класса cl_custom_3 , который вызывает конструктор базового класса cl_custom_base с передачей ему указателя на главный объект p_main_custom_object и имя объекта custom_name	∅

3.27 Алгоритм метода return_number_class класса cl_custom_3

Функционал: возврат номера класса.

Параметры: отсутствуют.

Возвращаемое значение: возврат целочисленного значения , которое представляет номер класса cl_custom_3.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода return_number_class класса cl_custom_3

№	Предикат	Действия	№ перехода
1		возврат целочисленного значение 3, которое представляет номер класса cl_custom_3	∅

3.28 Алгоритм конструктора класса cl_custom_4

Функционал: конструктор.

Параметры: p_main_custom_object - указатель на главный объект, custom_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 29.

Таблица 29 – Алгоритм конструктора класса cl_custom_4

№	Предикат	Действия	№ перехода
1		определение конструктора класса cl_custom_4 , который вызывает конструктор базового класса cl_custom_base с передачей ему указателя на главный объект p_main_custom_object и имя объекта custom_name	∅

3.29 Алгоритм метода return_number_class класса cl_custom_4

Функционал: возврат номера класса.

Параметры: отсутствуют.

Возвращаемое значение: возврат целочисленного значения , которое представляет номер класса cl_custom_4.

Алгоритм метода представлен в таблице 30.

Таблица 30 – Алгоритм метода return_number_class класса cl_custom_4

№	Предикат	Действия	№ перехода
1		возврат целочисленного значение 4, которое представляет номер класса cl_custom_4	∅

3.30 Алгоритм конструктора класса cl_custom_5

Функционал: конструктор.

Параметры: p_main_custom_object - указатель на главный объект, custom_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 31.

Таблица 31 – Алгоритм конструктора класса cl_custom_5

№	Предикат	Действия	№ перехода
1		определение конструктора класса cl_custom_5 , который вызывает	∅

№	Предикат	Действия	№ перехода
		конструктор базового класса cl_custom_base с передачей ему указателя на главный объект p_main_custom_object и имя объекта custom_name	

3.31 Алгоритм метода return_number_class класса cl_custom_5

Функционал: возврат номера класса.

Параметры: отсутствуют.

Возвращаемое значение: возврат целочисленного значения , которое представляет номер класса cl_custom_5.

Алгоритм метода представлен в таблице 32.

Таблица 32 – Алгоритм метода return_number_class класса cl_custom_5

№	Предикат	Действия	№ перехода
1		возврат целочисленного значение 5, которое представляет номер класса cl_custom_5	∅

3.32 Алгоритм конструктора класса cl_custom_6

Функционал: конструктор.

Параметры: p_main_custom_object - указатель на главный объект, custom_name - строка с именем объекта.

Алгоритм конструктора представлен в таблице 33.

Таблица 33 – Алгоритм конструктора класса cl_custom_6

№	Предикат	Действия	№ перехода
1		определение конструктора класса cl_custom_6 , который вызывает конструктор базового класса cl_custom_base с передачей ему указателя на главный объект p_main_custom_object и имя объекта	∅

№	Предикат	Действия	№ перехода
		custom_name	

3.33 Алгоритм метода return_number_class класса cl_custom_6

Функционал: возврат номера класса.

Параметры: отсутствуют.

Возвращаемое значение: возврат целочисленного значения , которое представляет номер класса cl_custom_6.

Алгоритм метода представлен в таблице 34.

Таблица 34 – Алгоритм метода return_number_class класса cl_custom_6

№	Предикат	Действия	№ перехода
1		возврат целочисленного значение 6, которое представляет номер класса cl_custom_6	∅

3.34 Алгоритм конструктора класса cl_custom_application

Функционал: конструктор.

Параметры: p_main_custom_object - указатель на объкт класса cl_custom_base.

Алгоритм конструктора представлен в таблице 35.

Таблица 35 – Алгоритм конструктора класса cl_custom_application

№	Предикат	Действия	№ перехода
1		определение конструктора класса cl_custom_application, который вызывает конструктор базового класса cl_custom_base, передавая ему указатель p_main_custom_object в качестве параметра	∅

3.35 Алгоритм метода build_tree_objects класса cl_custom_application

Функционал: создания дерева иерархии.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 36.

Таблица 36 – Алгоритм метода build_tree_objects класса cl_custom_application

№	Предикат	Действия	№ перехода
1		инициализация строковых переменных path_custom, sub_name; - переменные для хранения пути и иемни подчиненного объекта	2
2		инициализация указатель на текущий объект state	3
3		инициализация main_state = nullptr указатель на создаваемый подчиненный объект,	4
4		инициализация целочисленной command переменная для хранения команды	5
5		path_custom - чтение имени главного объета	6
6		установка имени главного объекта path_custom	7
7		state = this;// текущий объект - это сам объект приложения	8
8	true - правда	path_custom - ввод значения с клавиатуры	9
			23
9	path_custom равен "endtree"	прерываем цикл - break	23
			10
10		sub_name , command - чтетие имени подчиненного объекта и команды	11
11		получение указателя на объект по пути	12
12	объект не найден	вывод на экран "Object tree" , переход на	13

№	Предикат	Действия	№ перехода
		следующую строку - endl	
			16
13		вызов метода indentation_method	14
14		вывод "The head object ", значение переменной path_custom , вывод " is not found"	15
15		exit(1) - прекращение выполнения программы с кодом ошибки 1 - произошла ошибка	∅
16	подобъект с именем указанным в аргументе sub_name существует	вывод значения path_custom , вывод на экран " Dubbing the names of subordinate objects"	17
			17
17		создание нового подчиненного объекта в зависимости от команды	18
18	command равна 2	объект создается с указанным state в качестве главного объекта и sub_name в качестве имени объекта	23
			19
19	command равна 3	объект создается с указанным state в качестве главного объекта и sub_name в качестве имени объекта	23
			20
20	command равна 4	объект создается с указанным state в качестве главного объекта и sub_name в качестве имени объекта	23
			21
21	command равна 5	объект создается с указанным state в качестве главного объекта и sub_name в качестве имени объекта	23
			22
22	command равна 6	объект создается с указанным state в качестве	23

№	Предикат	Действия	№ перехода
		главного объекта и sub_name в качестве имени объекта	
			23
23		вывод дерева объектов, вывод "Object tree"	24
24		indentation_method() - вызов метода для вывода отступов	25
25		инициализация целочисленных text_class, set_class	26
26		инициализация signal_obj указатель на метод сигнала	27
27		инициализация handler_obj указатель на метод обработчика	28
28		инициализация cl_custom_base* parent - указатель на родительский объект	29
29		инициализация cl_custom_base* p_sub - указатель на подчиненный объект	30
30	true - правда	ввод значения переменной path_custom	31
			∅
31	path_custom равна значению "end_of_connections"	прерываем цикл	32
			∅
32		ввод sub_name - чтение имени подчиненного объекта	33
33		parent = get_object_path(path_custom) - получение указателя на родительский объект	34
34		p_sub = get_object_path(sub_name) - получение указателя на подчиненный объект	35
35	родительский или подчиненный объект не		36

№	Предикат	Действия	№ перехода
	найден или они одинаковы		
			39
36	родительский объект не найден	вывод на экран "Object ", значения path_custom , вывод " not found"	38
			37
37	подчиненный объект не найден	вывод на экран "Handler object ", значения sub_name, вывод " not found"	38
			38
38		продолжаем - continue	39
39		text_class = parent->return_number_class(); // получение номера класса родительского объекта	40
40		set_class = p_sub->return_number_class(); // получение номера класс подчиненного объекта	41
41		определение метода сигнала в зависимости от номера класса родительского объекта	42
42	значение text_class равно 1	инициализация переменной signal_obj= которая является указателем на метод сигнала , используя макрос SIGNAL_D. макрос принимает указатель на метод и приводит его к типу TYPE_SIGNAL	48
			43
43	значение text_class равно 2	инициализация переменной signal_obj= которая является указателем на метод сигнала , используя макрос SIGNAL_D. макрос принимает указатель на метод и приводит его к типу TYPE_SIGNAL	48
			44
44	значение text_class равно 3	инициализация переменной signal_obj= которая является указателем на метод сигнала , используя макрос SIGNAL_D. макрос принимает указатель	48

№	Предикат	Действия	№ перехода
		на метод и приводит его к типу TYPE_SIGNAL	
			45
45	значение text_class равно 4	инициализация переменной signal_obj= которая является указателем на метод сигнала , используя макрос SIGNAL_D. макрос принимает указатель на метод и приводит его к типу TYPE_SIGNAL	48
			46
46	значение text_class равно 5	инициализация переменной signal_obj= которая является указателем на метод сигнала , используя макрос SIGNAL_D. макрос принимает указатель на метод и приводит его к типу TYPE_SIGNAL	48
			47
47	значение text_class равно 6	инициализация переменной signal_obj= которая является указателем на метод сигнала , используя макрос SIGNAL_D. макрос принимает указатель на метод и приводит его к типу TYPE_SIGNAL	48
			49
48		определение метода обработчика в зависимости от номера класса подчиненного объекта	49
49	значение set_class равно 1	инициализация переменной handler_obj = которая является указателем на метод сигнала , используя макрос HANDLER_D. макрос принимает указатель на метод и приводит его к типу TYPE_HANDLER	55
			50
50	значение set_class равно 2	инициализация переменной handler_obj = которая является указателем на метод сигнала , используя макрос HANDLER_D. макрос принимает указатель на метод и приводит его к типу	55

№	Предикат	Действия	№ перехода
		TYPE_HANDLER	
			51
51	значение set_class равно 3	инициализация переменной handler_obj = которая является указателем на метод сигнала , используя макрос HANDLER_D. макрос принимает указатель на метод и приводит его к типу TYPE_HANDLER	55
			52
52	значение set_class равно 4	инициализация переменной handler_obj = которая является указателем на метод сигнала , используя макрос HANDLER_D. макрос принимает указатель на метод и приводит его к типу TYPE_HANDLER	55
			53
53	значение set_class равно 5	инициализация переменной handler_obj = которая является указателем на метод сигнала , используя макрос HANDLER_D. макрос принимает указатель на метод и приводит его к типу TYPE_HANDLER	55
			54
54	значение set_class равно 6	инициализация переменной handler_obj = которая является указателем на метод сигнала , используя макрос HANDLER_D. макрос принимает указатель на метод и приводит его к типу TYPE_HANDLER	55
			55
55		установка связи между сигналом родительского объекта и обработчиком подчиненного объекта, вызывается метод connections_set_main объекта parent для установки связи между сигналом,	∅

№	Предикат	Действия	№ перехода
		обработчкиком и целевым объектом	

3.36 Алгоритм метода exec_custom_app класса cl_custom_application

Функционал: метод запуска системы.

Параметры: отсутствуют.

Возвращаемое значение: индикатор корректности работы алгоритма.

Алгоритм метода представлен в таблице 37.

Таблица 37 – Алгоритм метода exec_custom_app класса cl_custom_application

№	Предикат	Действия	№ перехода
1		вызов метода set_object_ready	2
2		инициализация строковых переменных signal_coord, handler_coord - переменные для хранения координат сигнала и обработчика	3
3		инициализация целочисленной переменной int status - переменная для хранения статуса	4
4		cl_custom_base* parent - указатель на родительский объект	5
5		cl_custom_base* p_sub - указатель на подчиненный объект	6
6		инициализация переменной TYPE_SIGNAL signal_obj - переменная для хранения указателя на метод сигнала	7
7		инициализация переменной TYPE_HANDLER handler_obj - переменная для хранения указателя	8

№	Предикат	Действия	№ перехода
		на метод обработчика	
8		инициализация строковых переменных string command, text - переменные для хранения команды и текста	9
9		инициализация целочисленных переменных - int text_class, set_class - переменные для хранения номеров классов объектов	10
10	true - правда	ввод значения command - чтение команды	11
			53
11	значение переменной command равно "END"	прерываем цикл - break	12
			12
12	значение переменной command равно "EMIT"	ввод значения переменной signal_coord - чтение координаты сигнала	13
			25
13		чтение текста сигнала	14
14		получение указателя на объект по координате сигнала	15
15	не найден объект parent по указанному пути signal_coord	вывод на экран "Object " , вывод значения signal_coord, вывод " not found";	16
			16
16		вызывается метод return_number_class для ообъект parent - который возвращает номер класса объекта parent - полученный номер сохраняется в переменной text_Class - получение номера класса объета	17
17		установка соответсвующего сигнала в	18

№	Предикат	Действия	№ перехода
		зависимости от номера класса	
18	значение переменной text_class равно 1	установка метода сигнала, используется макрос SIGNAL_D - чтобы преобразовать указатель на метод signal_method в соответствующий тип TYPE_SIGNAL , который используется для установки сигнала в методе connections_set_main()	24
			19
19	значение переменной text_class равно 2	установка метода сигнала, используется макрос SIGNAL_D - чтобы преобразовать указатель на метод signal_method в соответствующий тип TYPE_SIGNAL , который используется для установки сигнала в методе connections_set_main()	24
			20
20	значение переменной text_class равно 3	установка метода сигнала, используется макрос SIGNAL_D - чтобы преобразовать указатель на метод signal_method в соответствующий тип TYPE_SIGNAL , который используется для установки сигнала в методе connections_set_main()	24
			21
21	значение переменной text_class равно 4	установка метода сигнала, используется макрос SIGNAL_D - чтобы преобразовать указатель на метод signal_method в соответствующий тип TYPE_SIGNAL , который используется для установки сигнала в методе connections_set_main()	24
			22
22	значение переменной text_class равно 5	установка метода сигнала, используется макрос SIGNAL_D - чтобы преобразовать указатель на метод signal_method в соответствующий тип TYPE_SIGNAL , который используется для	24

№	Предикат	Действия	№ перехода
		установки сигнала в методе connections_set_main()	
			23
23	значение переменной text_class равно 6	установка метода сигнала, используется макрос SIGNAL_D - чтобы преобразовать указатель на метод signal_method в соответствующий тип TYPE_SIGNAL , который используется для установки сигнала в методе connections_set_main()	24
			24
24		вызывается метод connections_main_signal у объекта parent - передая ему signal_obj и текст text для обработки	25
25	значение command равно "SET_CONNECT" или command равно "DELETE_CONNECT"	ввод значений signal_coord и handler_coord - чтение координат сигнала и обработчика	26
			49
26		parent = get_object_path(signal_coord);//получение указателя на объект по координате сигнала	27
27		p_sub = get_object_path(handler_coord);//получение указателя на объект по координате обработчика	28
28	объекты не найдены или совпадают		29
			31
29	родительский объект не найден	вывод на экран "Object ", вывод значения signal_coord , вывод " not found"	31
			30
30	подчинненый объект не найден	вывод на экран "Handler object ", вывод значения handler_coord, вывод " not found"	31

№	Предикат	Действия	№ перехода
			31
31		parent->return_number_class() - получение номера класса родительского объекта	32
32		p_sub->return_number_class() - получение номера класса подчиненного объекта	33
33		установка соответствующего сигнала в зависимости от номера класса родительского объекта	34
34	значение переменной text_class равно 1	установка метода обработчика, определение переменной signal_obj как указатель на метод signal_method с использованием макроса 'SIGNAL_D'	40
			35
35	значение переменной text_class равно 2	установка метода обработчика, определение переменной signal_obj как указатель на метод signal_method с использованием макроса 'SIGNAL_D'	40
			36
36	значение переменной text_class равно 3	установка метода обработчика, определение переменной signal_obj как указатель на метод signal_method с использованием макроса 'SIGNAL_D'	40
			37
37	значение переменной text_class равно 4	установка метода обработчика, определение переменной signal_obj как указатель на метод signal_method с использованием макроса 'SIGNAL_D'	40
			38
38	значение переменной	установка метода обработчика, определение	40

№	Предикат	Действия	№ перехода
	text_class равно 5	переменной signal_obj как указатель на метод signal_method с использованием макроса 'SIGNAL_D'	
			39
39	значение переменной text_class равно 6	установка метода обработчика, определение переменной signal_obj как указатель на метод signal_method с использованием макроса 'SIGNAL_D'	40
			40
40		установка метода обработчика, установка соответствующего обработчика в зависимости от номера класса подчиненного объекта	41
41	значение переменной set_class равно 1	установка метода обработчика, определение переменной handler_obj как указатель на метод handler_method с использованием макроса HANDLER_D	47
			42
42	значение переменной set_class равно 2	установка метода обработчика, определение переменной handler_obj как указатель на метод handler_method с использованием макроса HANDLER_D	47
			43
43	значение переменной set_class равно 3	установка метода обработчика, определение переменной handler_obj как указатель на метод handler_method с использованием макроса HANDLER_D	47
			44
44	значение переменной set_class равно 4	установка метода обработчика, определение переменной handler_obj как указатель на метод handler_method с использованием макроса	47

№	Предикат	Действия	№ перехода
		HANDLER_D	
			45
45	значение переменной set_class равно 5	установка метода обработчика, определение переменной handler_obj как указатель на метод handler_method с использованием макроса HANDLER_D	47
			46
46	значение переменной set_class равно 6	установка метода обработчика, определение переменной handler_obj как указатель на метод handler_method с использованием макроса HANDLER_D	47
			47
47	значение переменной command равно "SET_CONNECT"	вызов метода connections_set_main у объекта parent, signal_obj, p_sub, handler_obj - которые представляют собой указатель на метод сигнала , указатель на целевой объект, указатель на метод обработчика установка связи между сигналом и обработчиком	48
			48
48	значение переменной command равно "DELETE_CONNECT"	вызов метода connections_delete_mainy объекта parent, signal_obj, p_sub, handler_obj - которые представляют собой указатель на метод сигнала , указатель на целевой объект, указатель на метод обработчика установка связи между сигналом и обработчиком	49
			49
49	значение переменной command равно "SET_CONDITION"	ввод значений в переменные signal_coord , status; - чтение координаты сигнала и статуса	50

№	Предикат	Действия	№ перехода
			53
50		получение указателя на объект по координате сигнала	51
51	объект parent не существует	вывод на экран "Object ", значение переменной signal_coord, вывод на экран " not found";	52
			52
52		установка состояния объекта - вызывается метод set_state для объекта parent с аргументом - метод set_state устанавливает состояние объекта в значение status	53
53		return(0) - индикатор корректности работы алгоритма - нет ошибок	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-63.

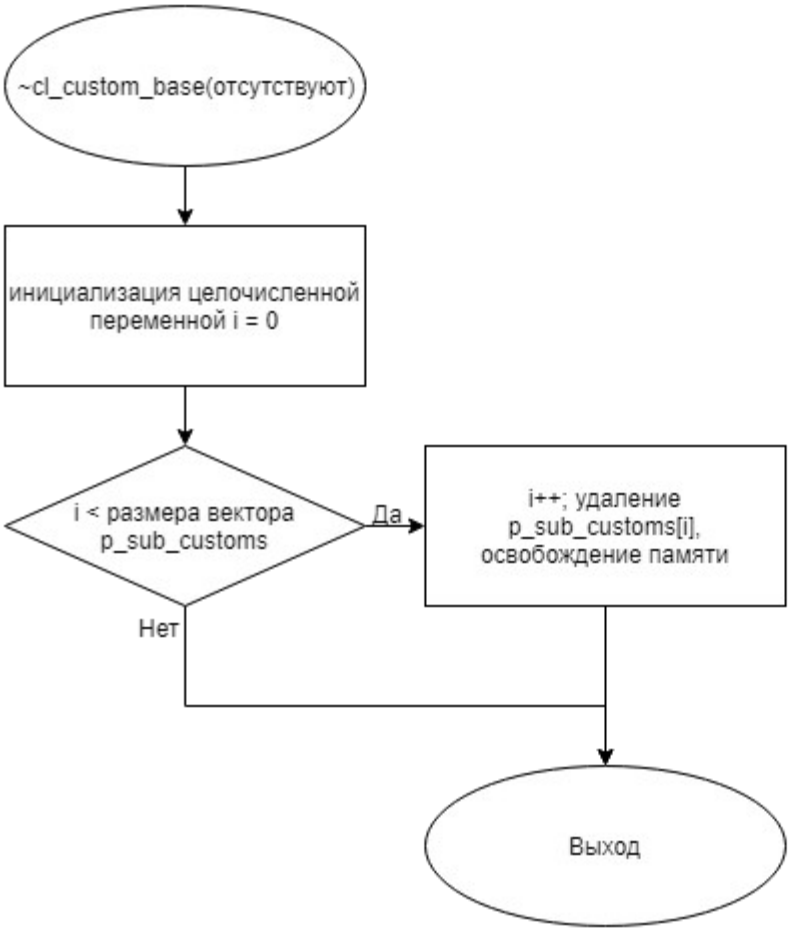

Рисунок 1 – Блок-схема алгоритма

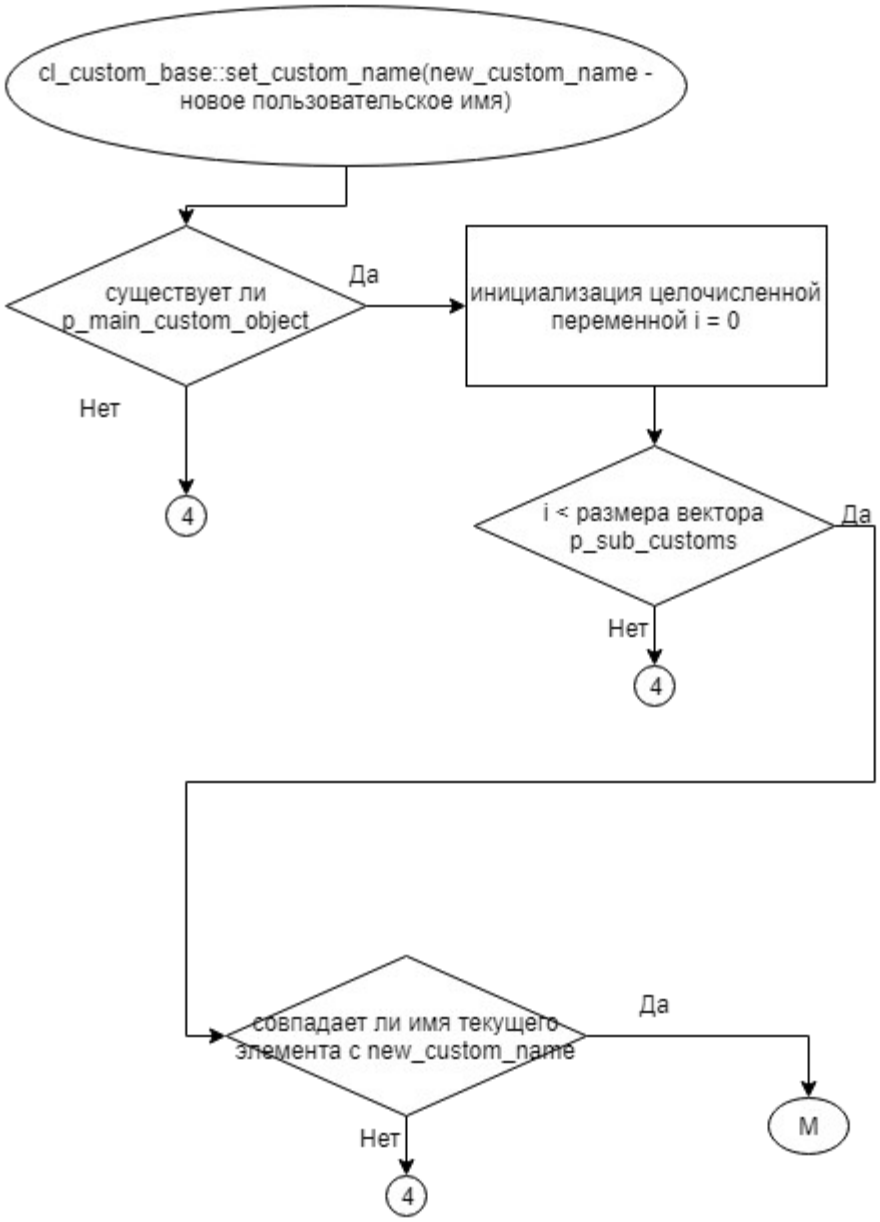

Рисунок 2 – Блок-схема алгоритма

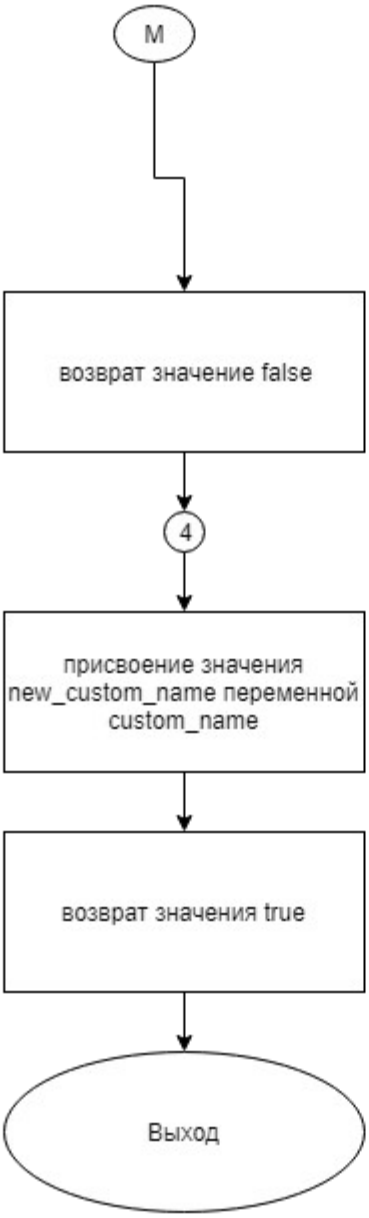

Рисунок 3 – Блок-схема алгоритма

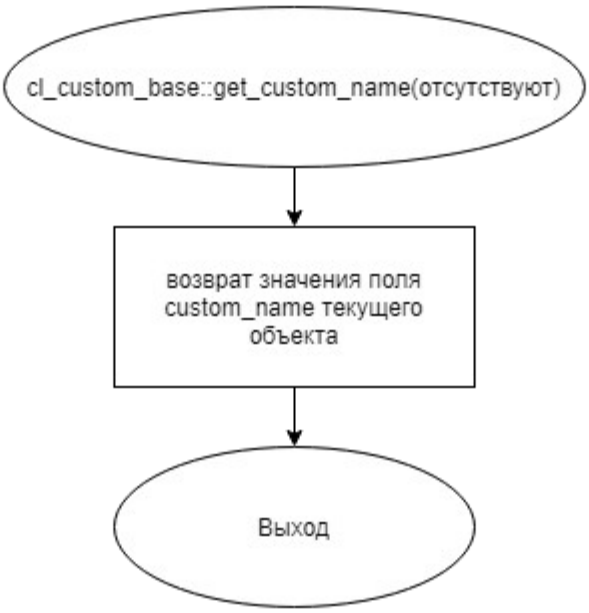

Рисунок 4 – Блок-схема алгоритма

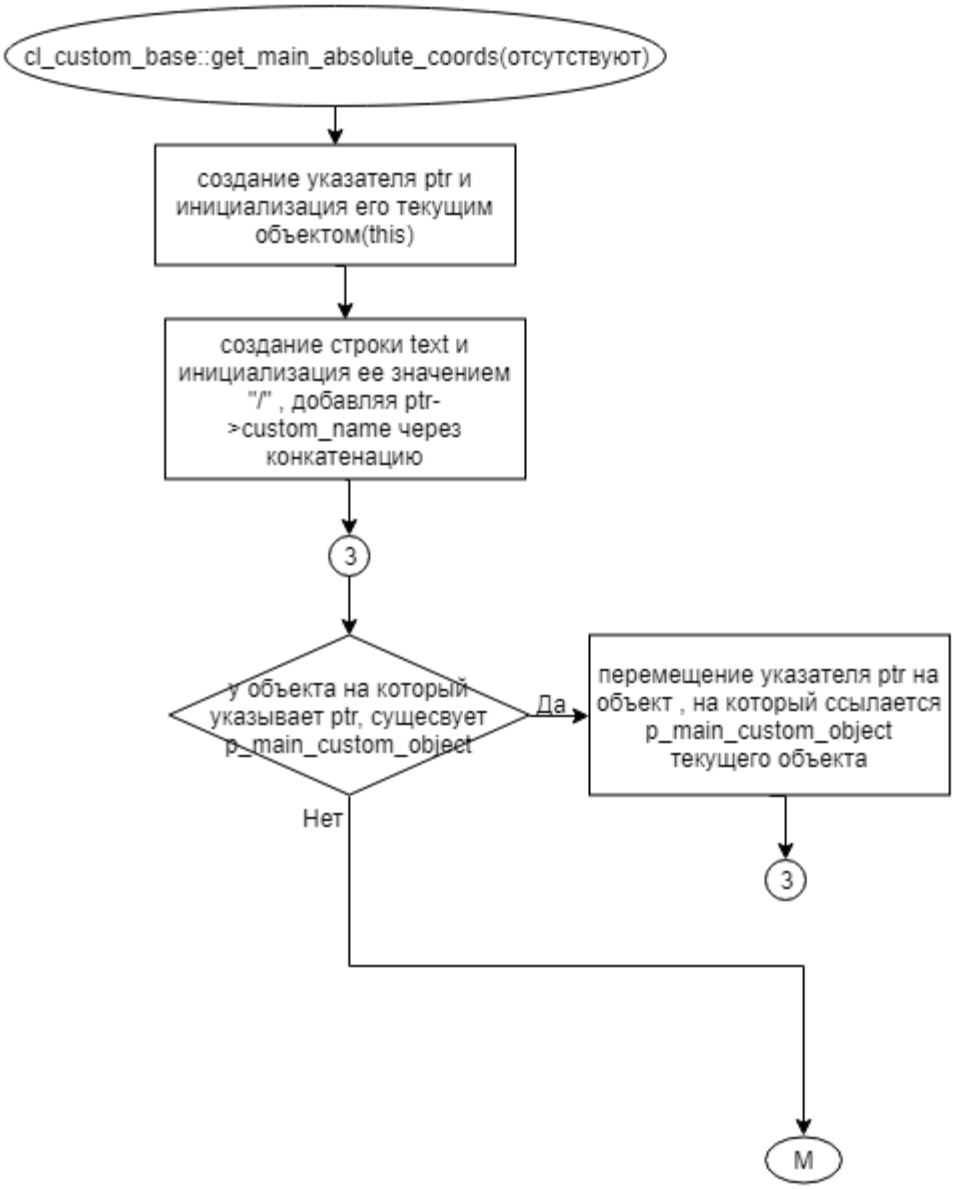

Рисунок 5 – Блок-схема алгоритма

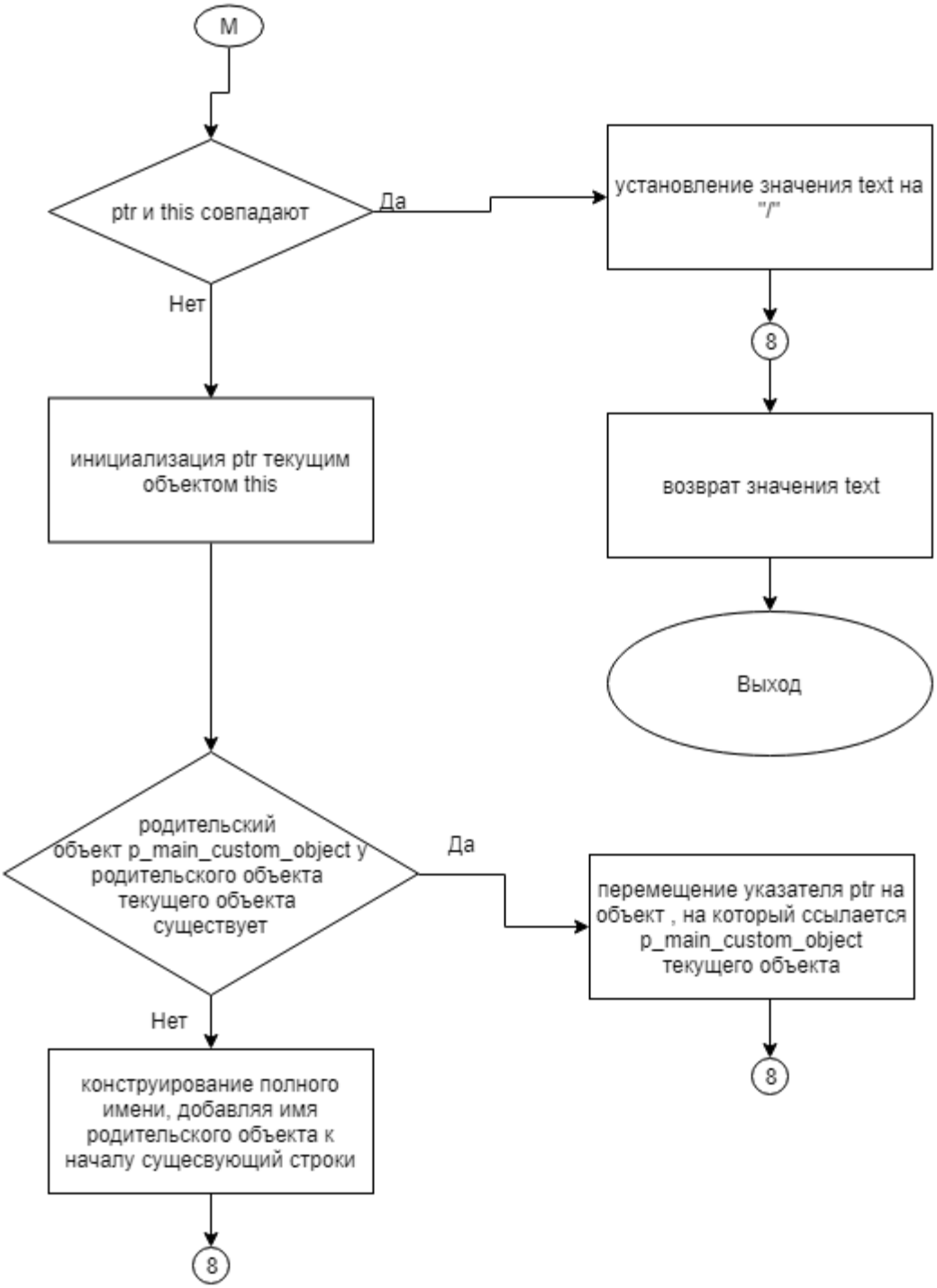

Рисунок 6 – Блок-схема алгоритма

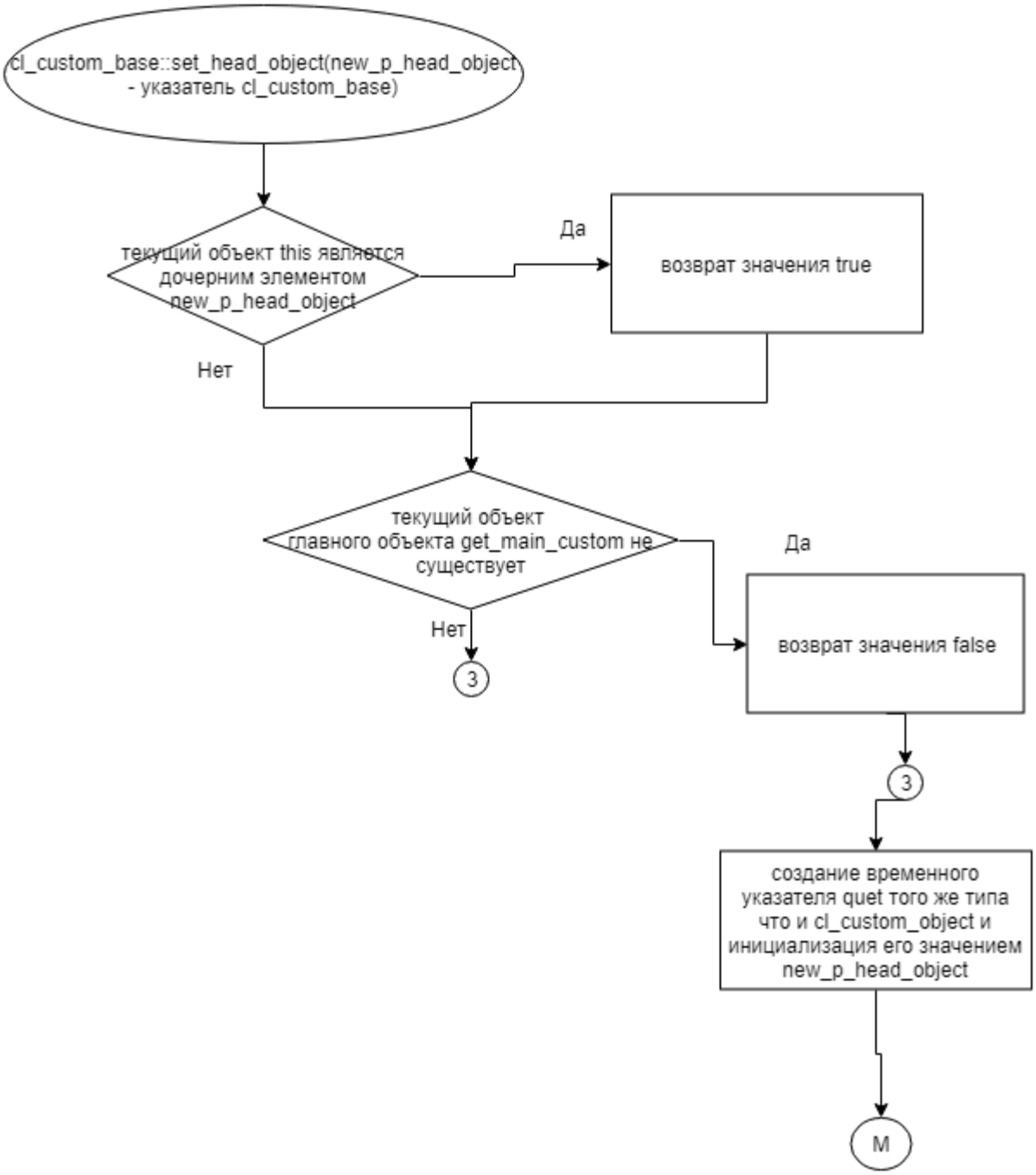

Рисунок 7 – Блок-схема алгоритма

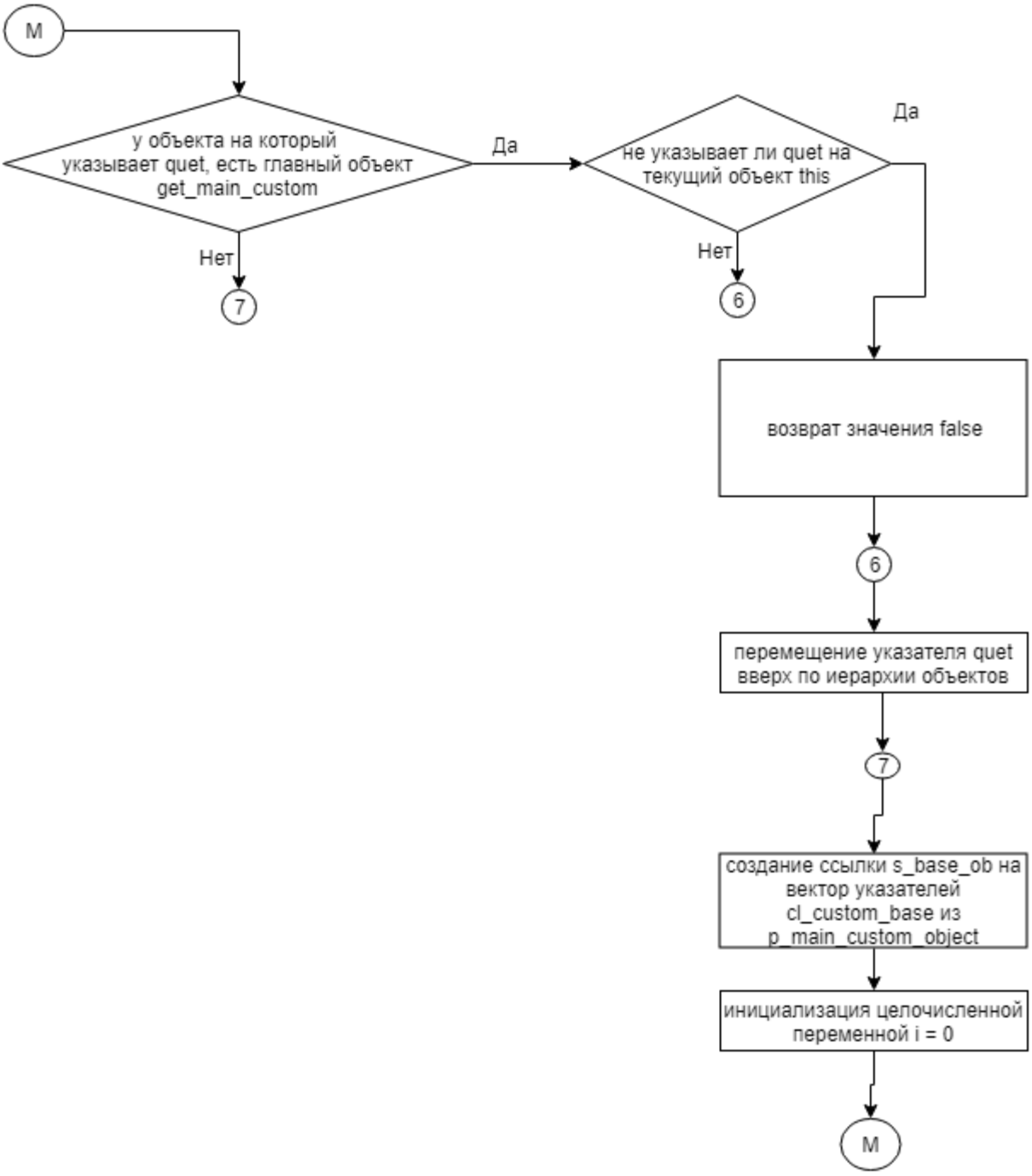

Рисунок 8 – Блок-схема алгоритма

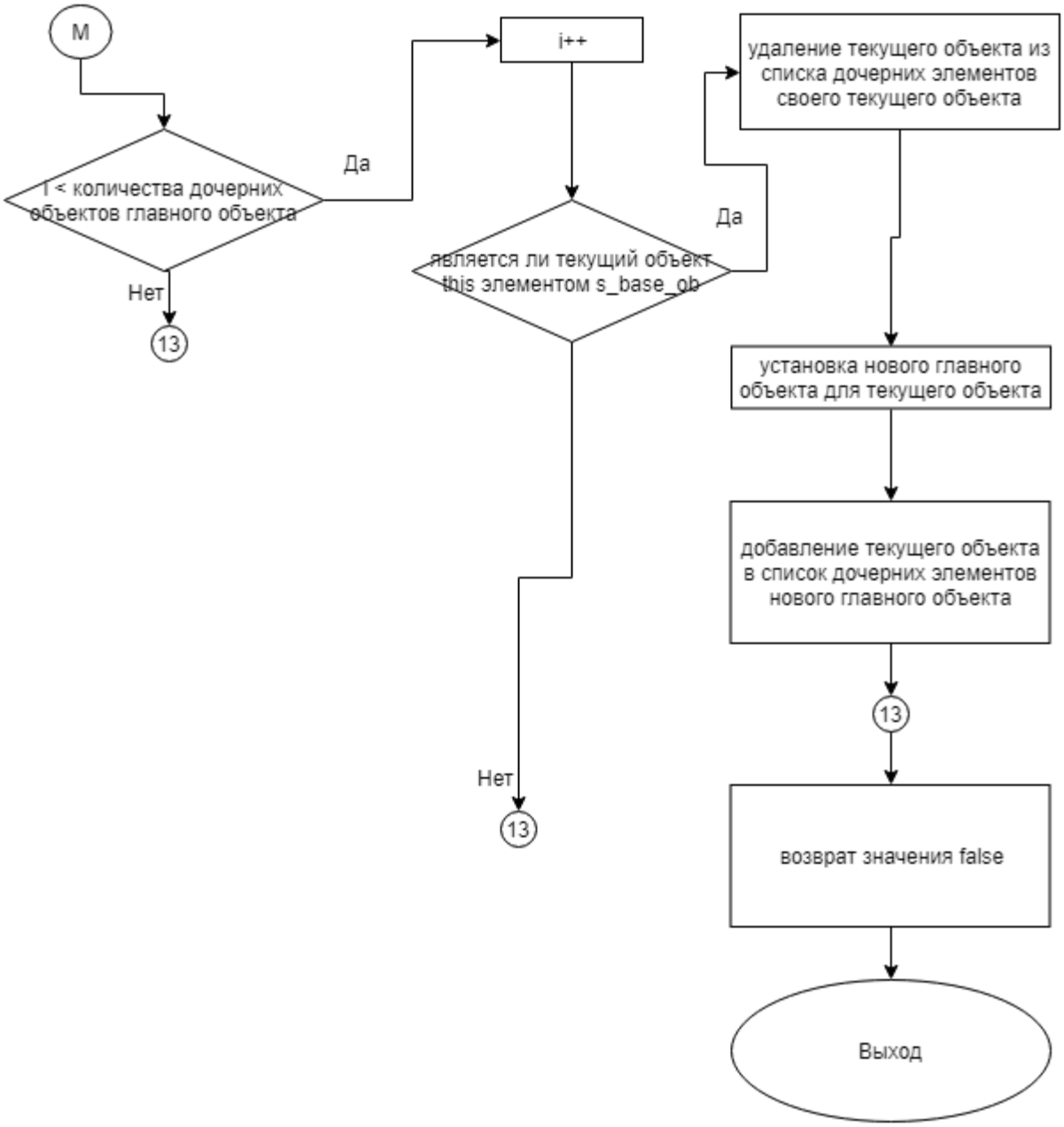

Рисунок 9 – Блок-схема алгоритма

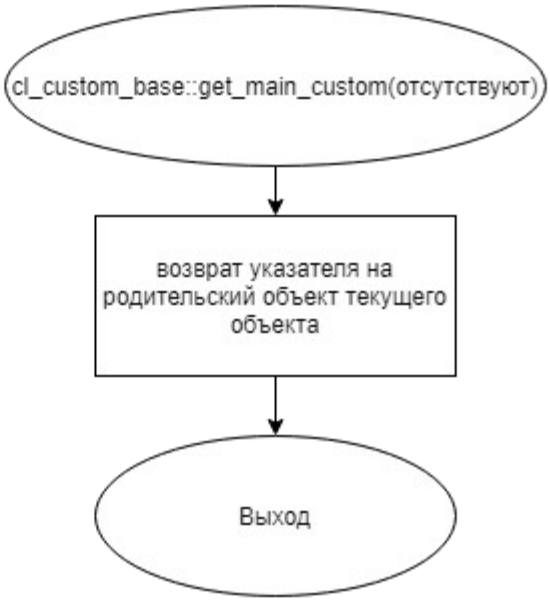

Рисунок 10 – Блок-схема алгоритма

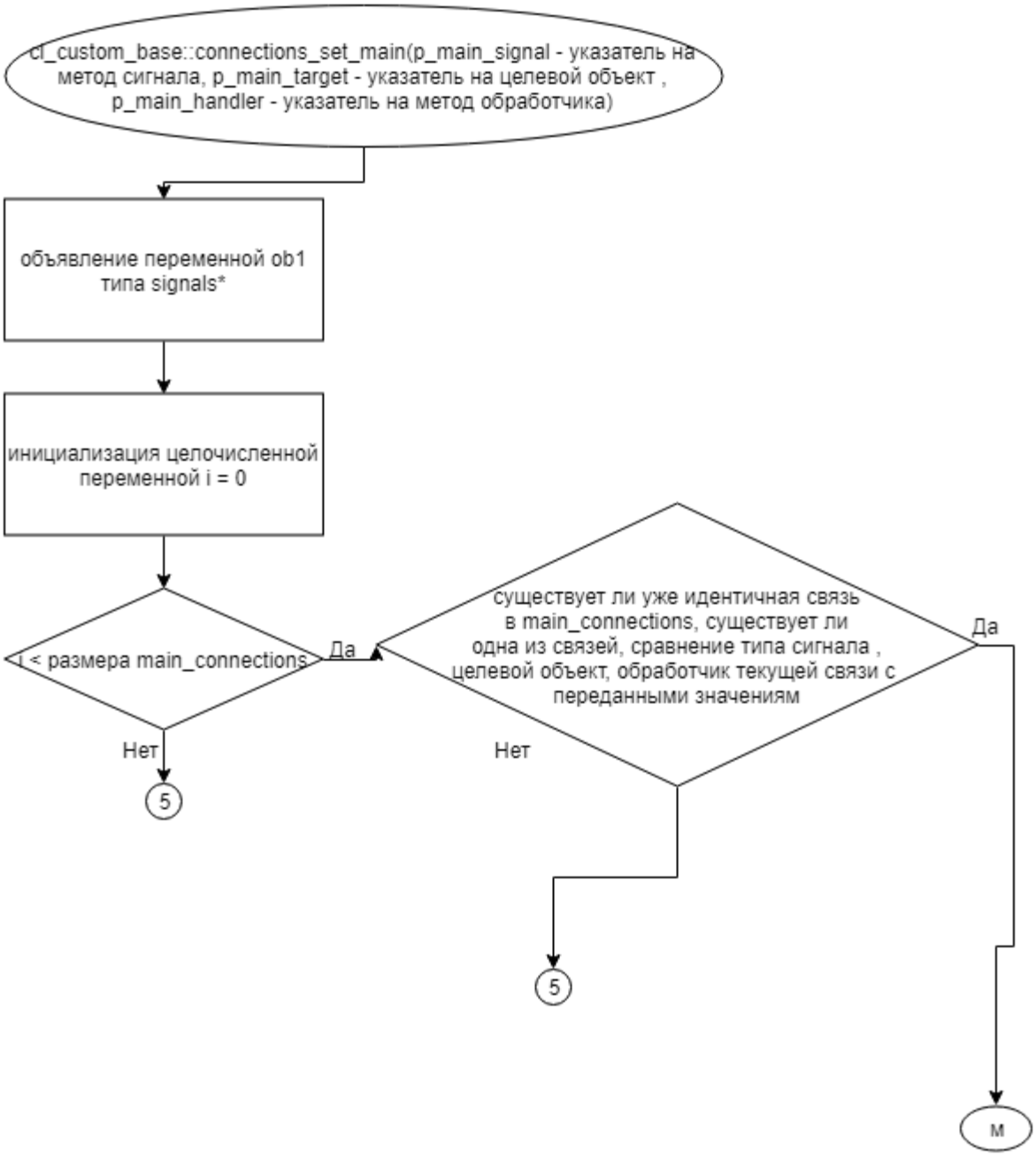

Рисунок 11 – Блок-схема алгоритма

Рисунок 12 – Блок-схема алгоритма

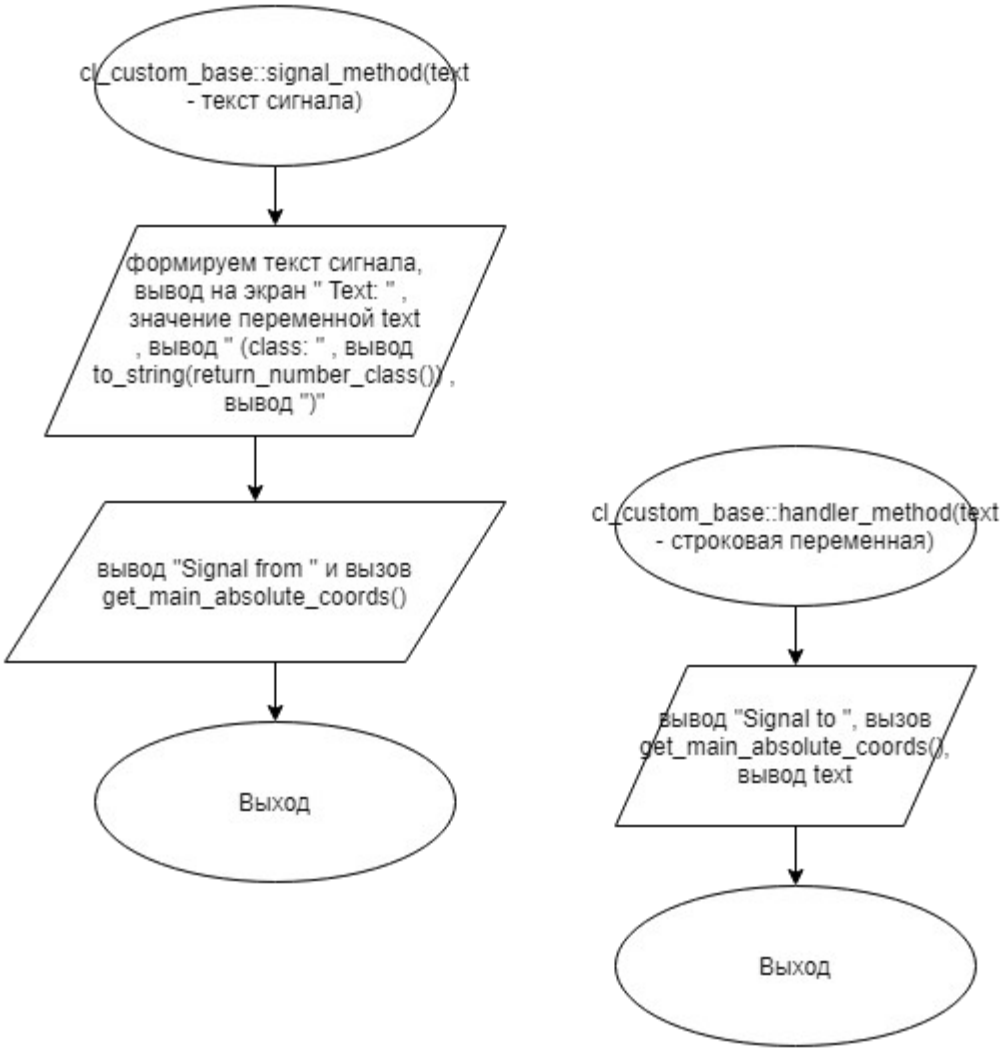

Рисунок 13 – Блок-схема алгоритма

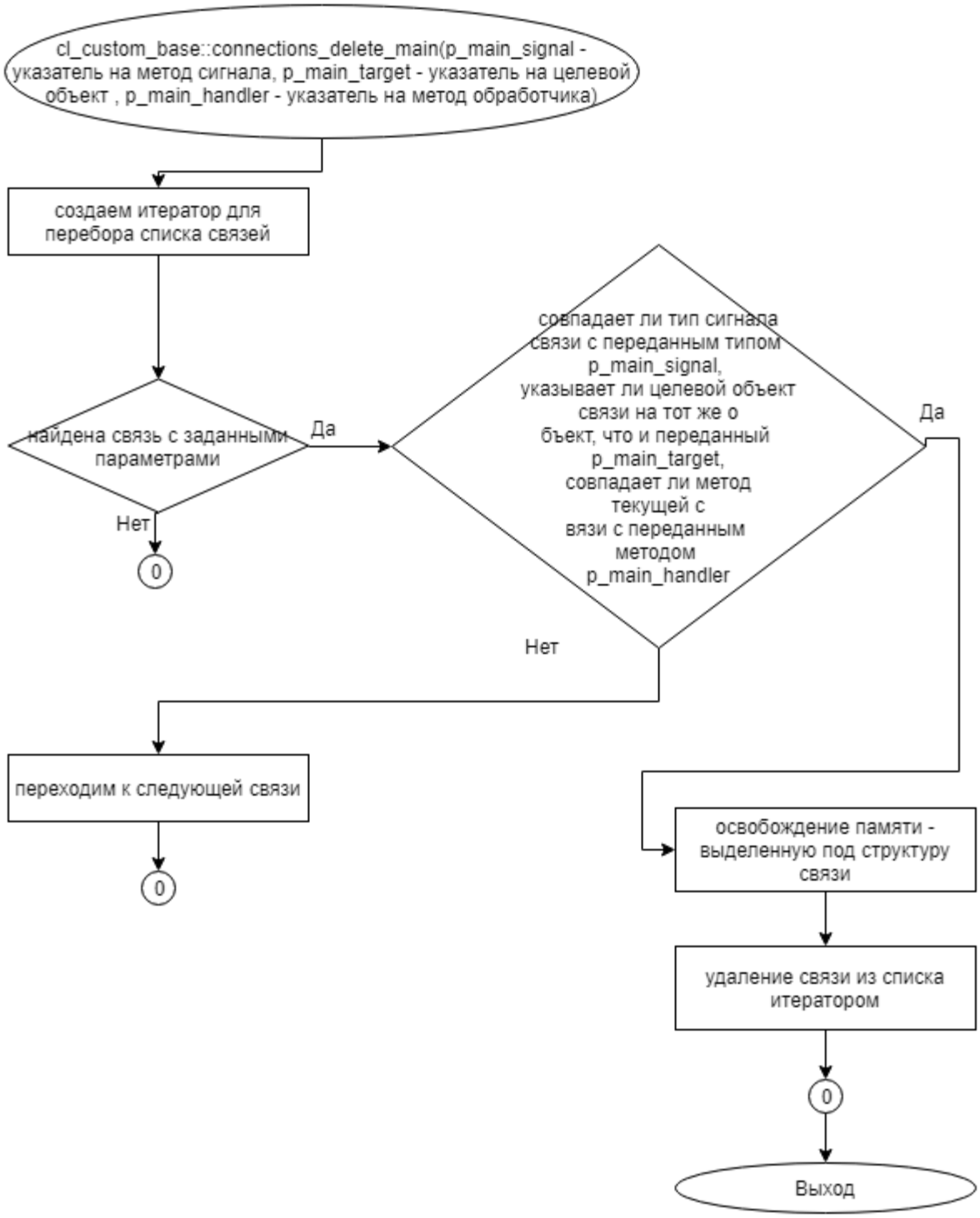

Рисунок 14 – Блок-схема алгоритма

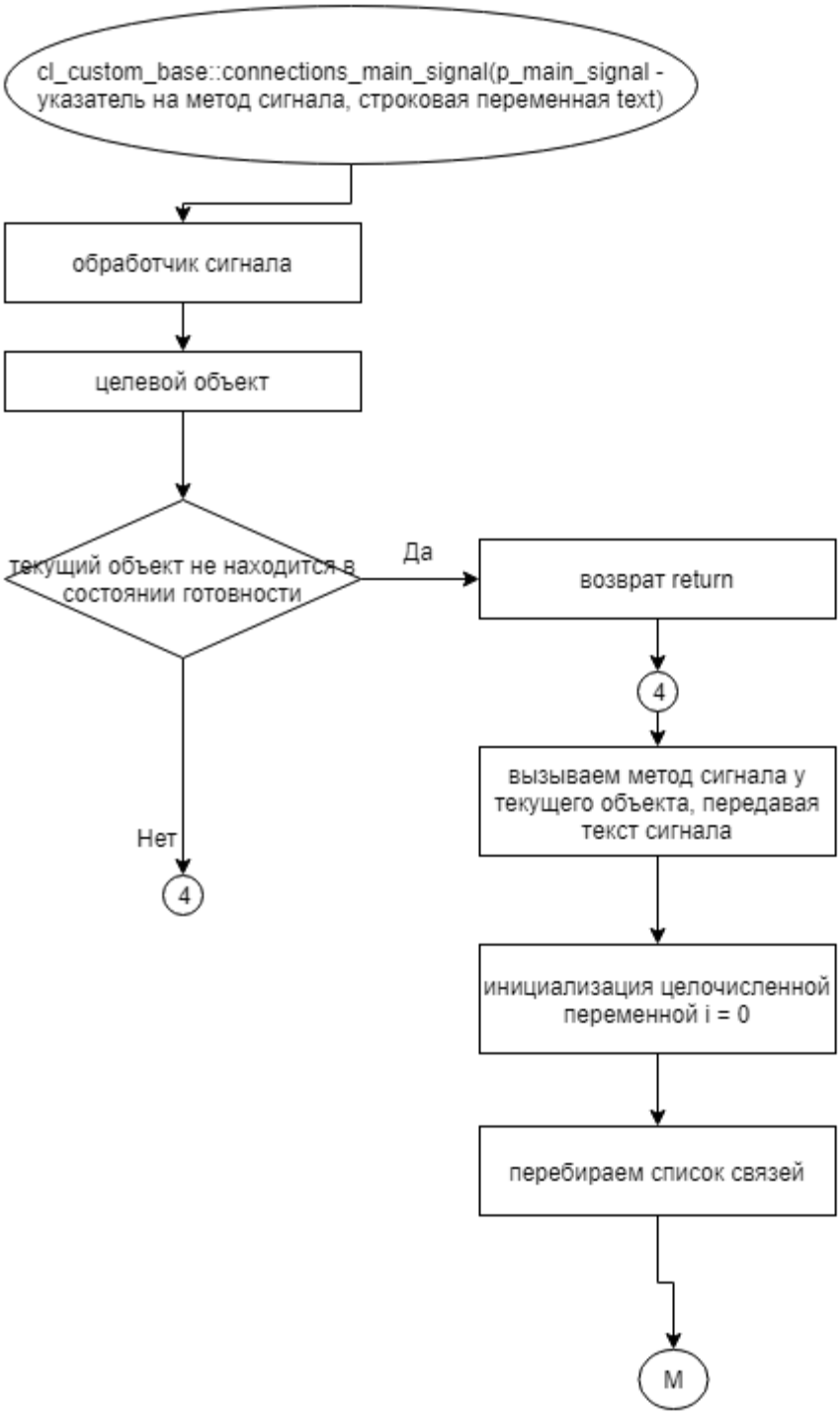

Рисунок 15 – Блок-схема алгоритма

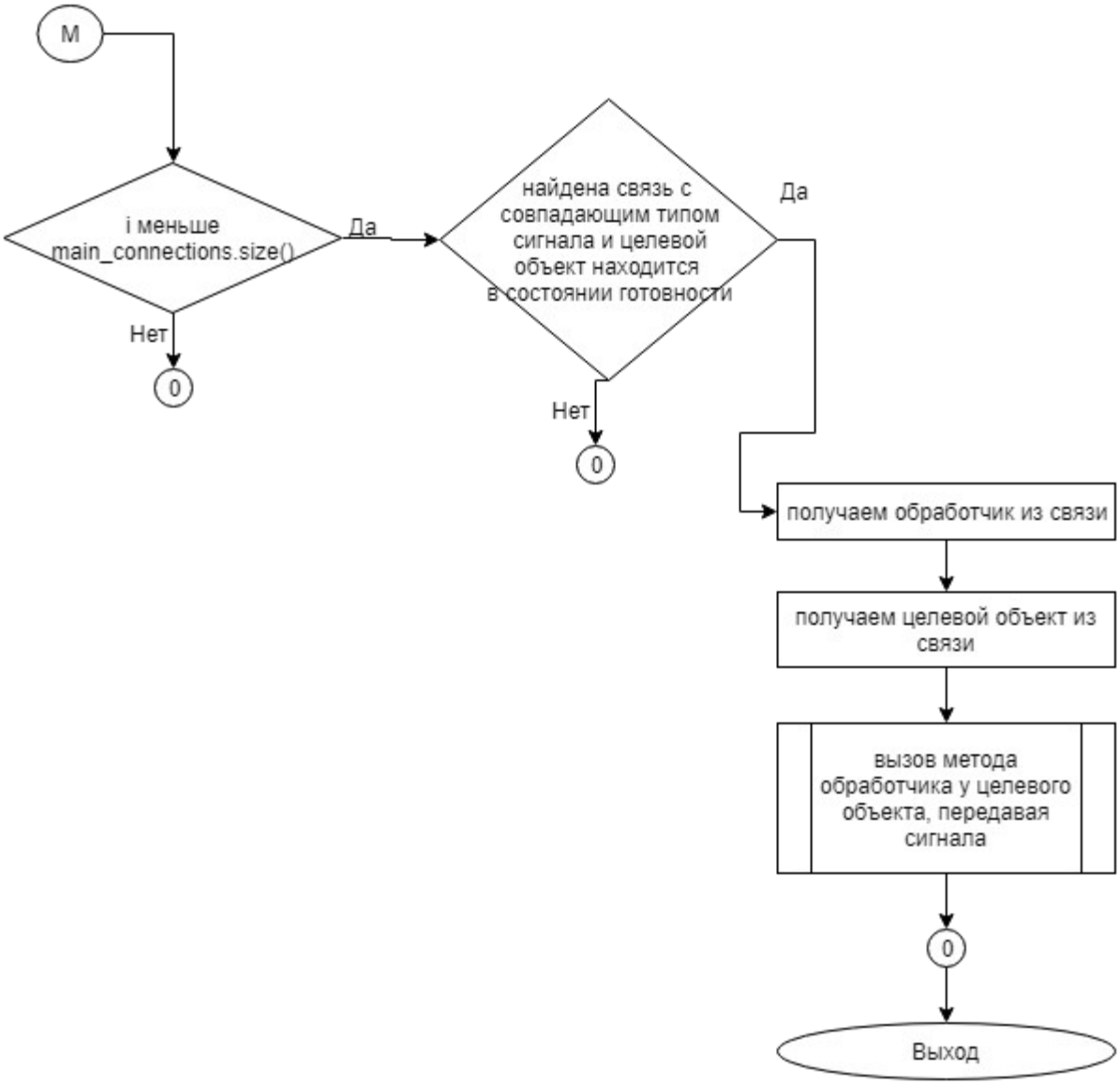

Рисунок 16 – Блок-схема алгоритма

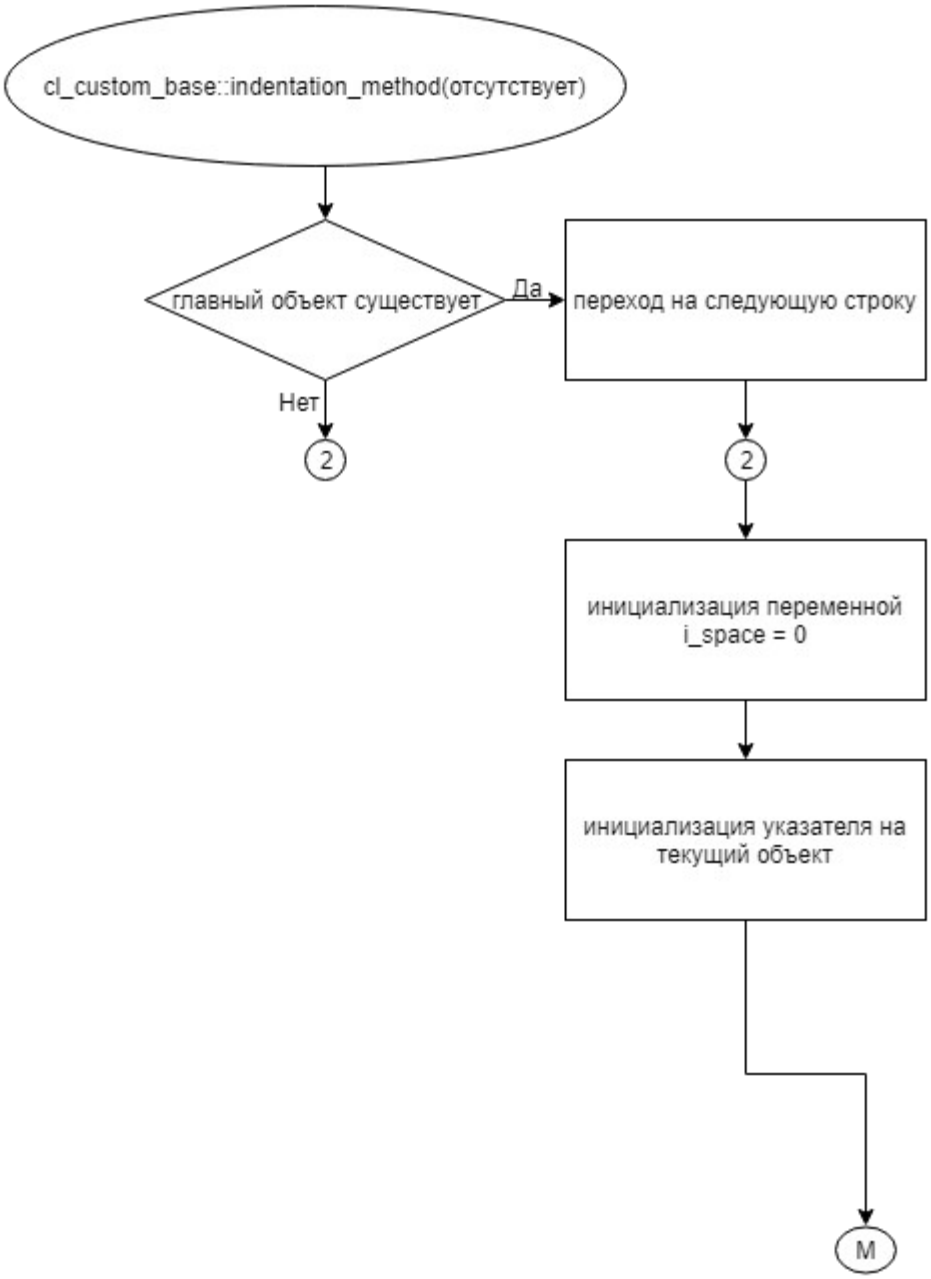

Рисунок 17 – Блок-схема алгоритма

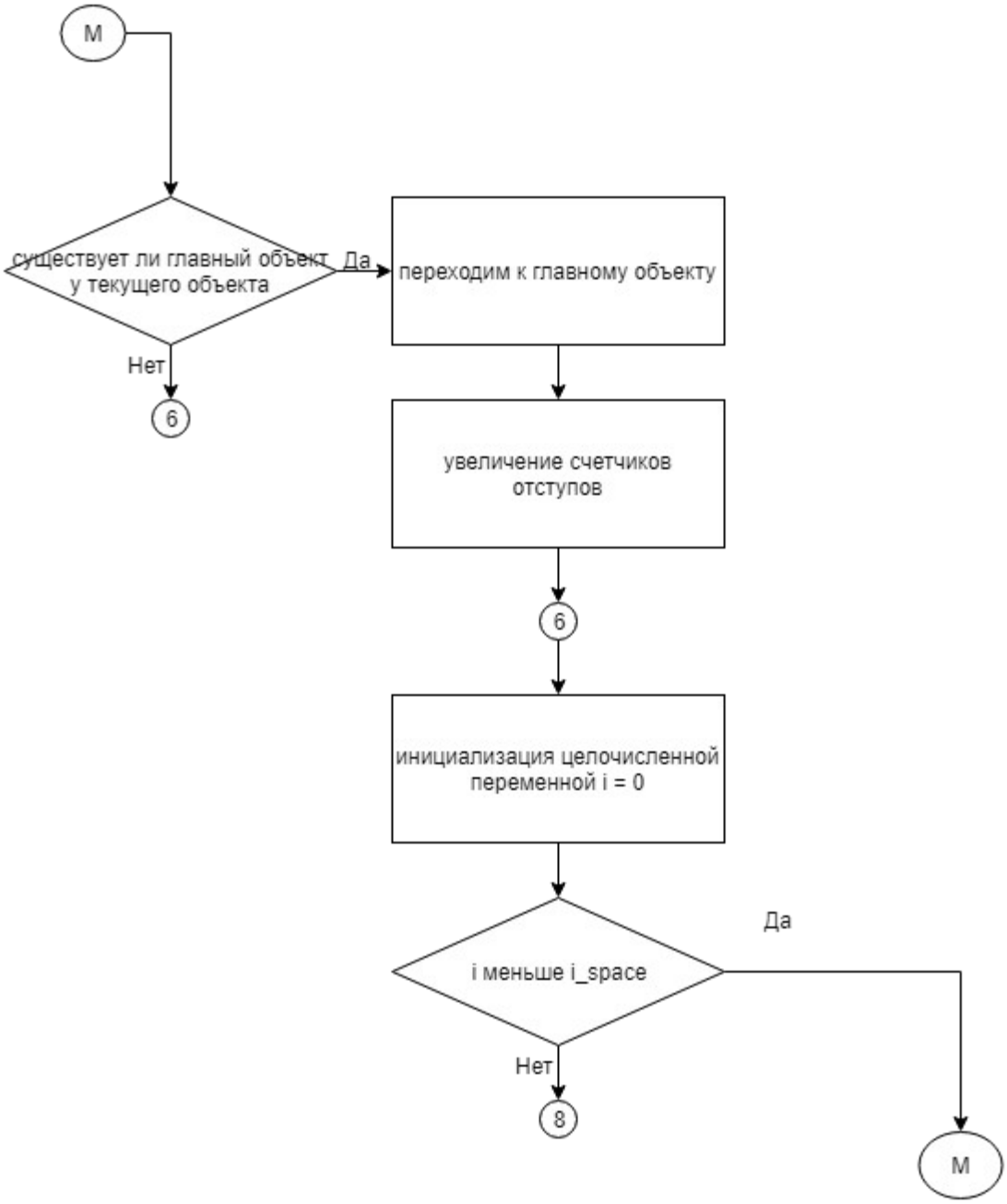

Рисунок 18 – Блок-схема алгоритма

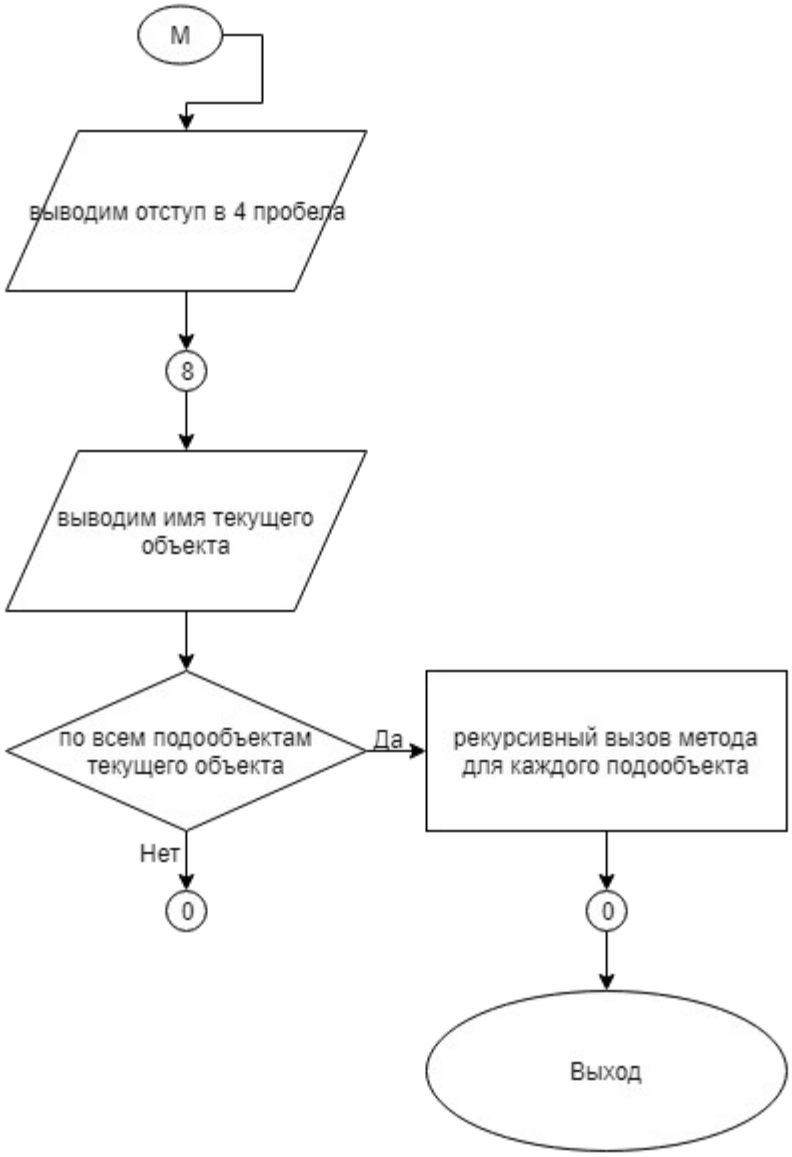

Рисунок 19 – Блок-схема алгоритма

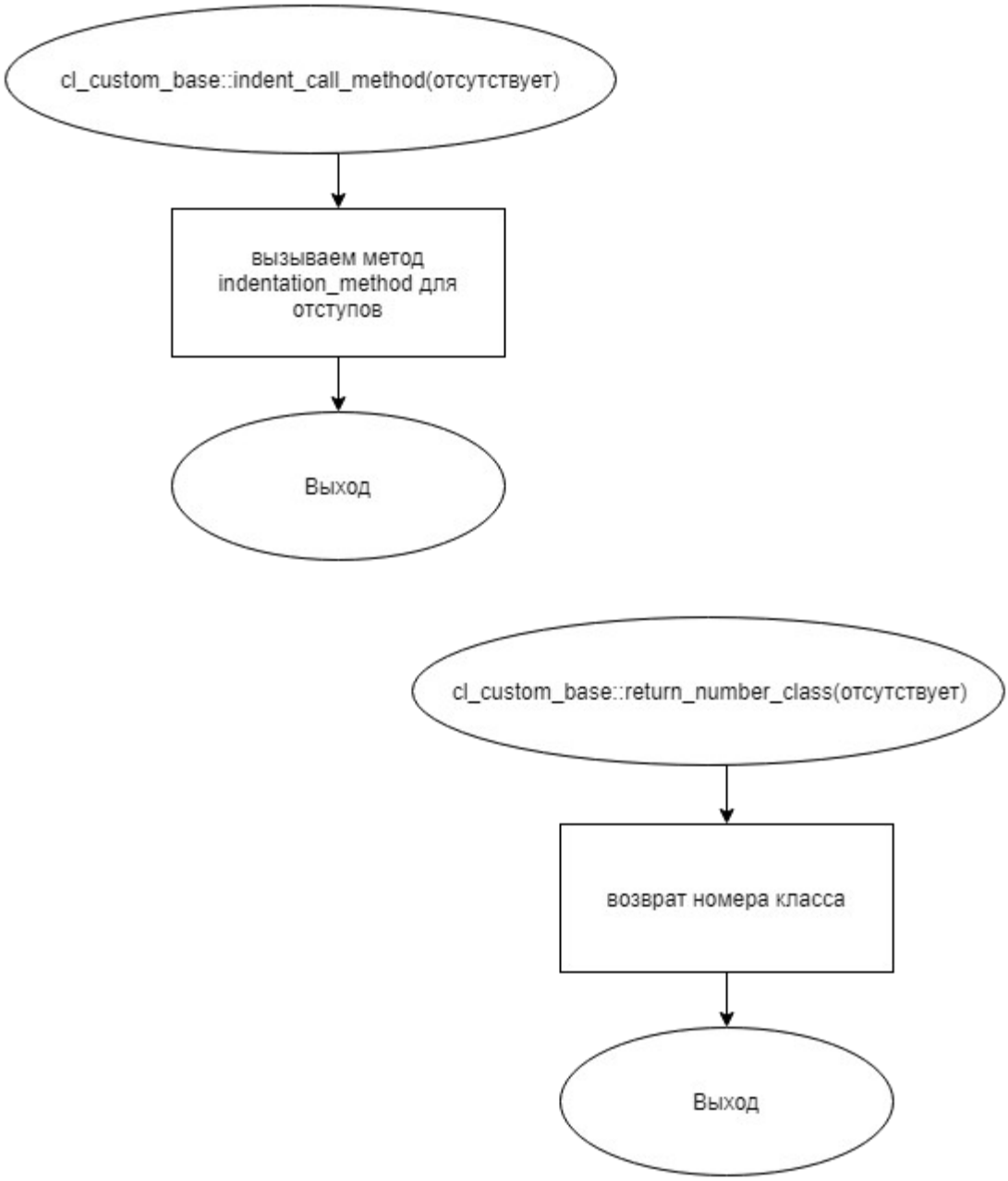

Рисунок 20 – Блок-схема алгоритма

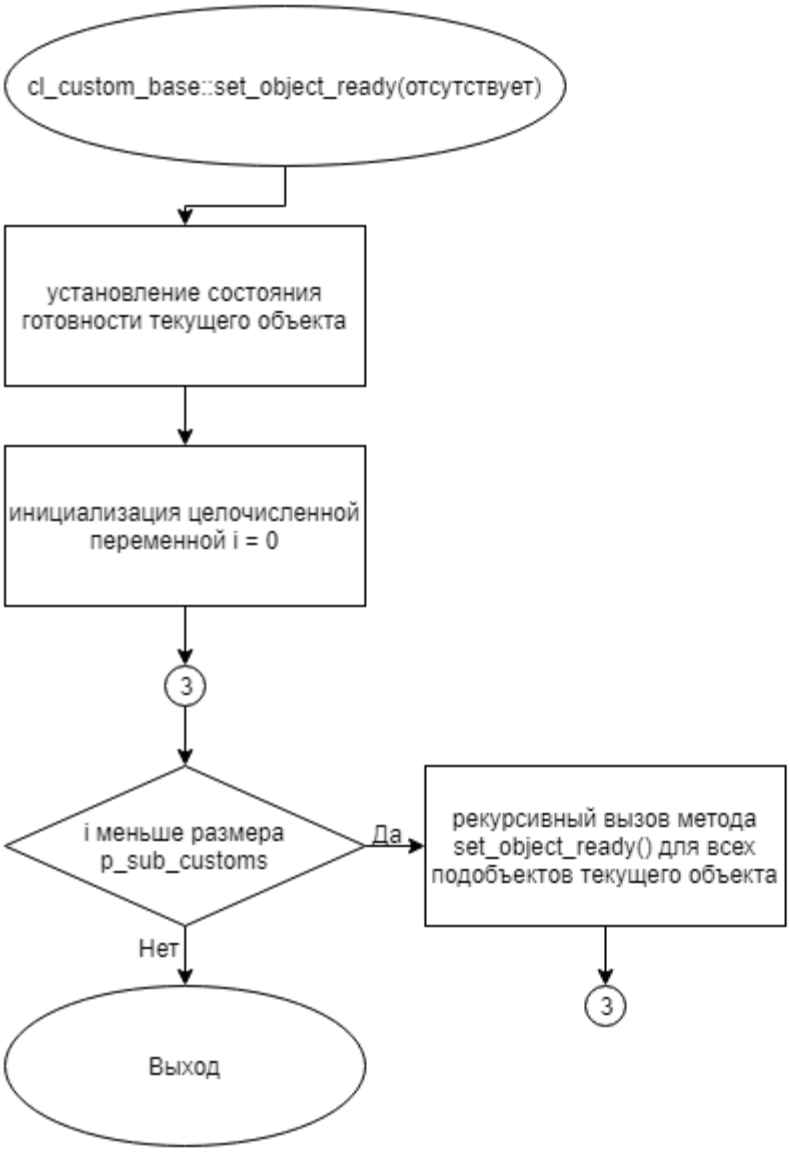

Рисунок 21 – Блок-схема алгоритма

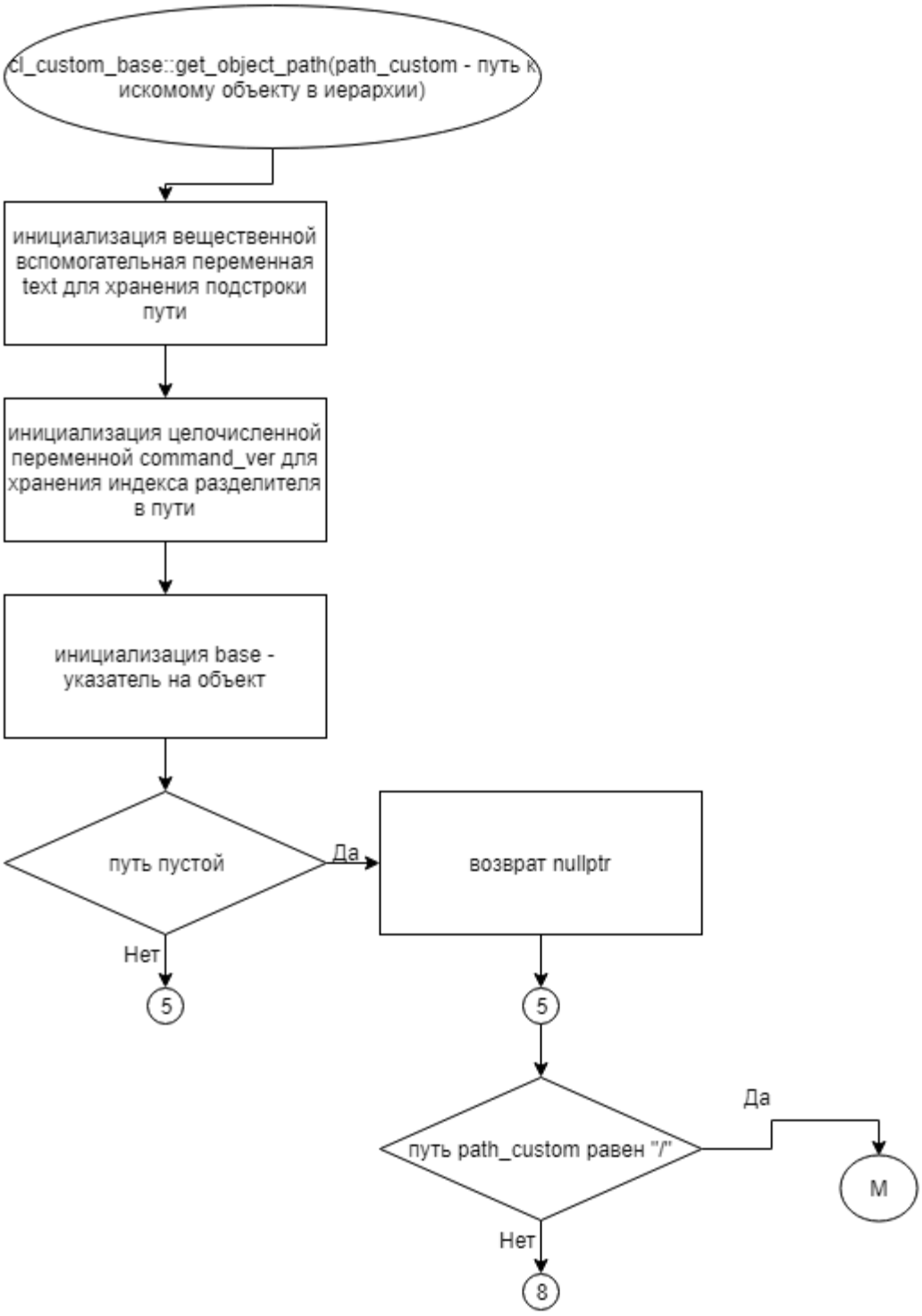

Рисунок 22 – Блок-схема алгоритма

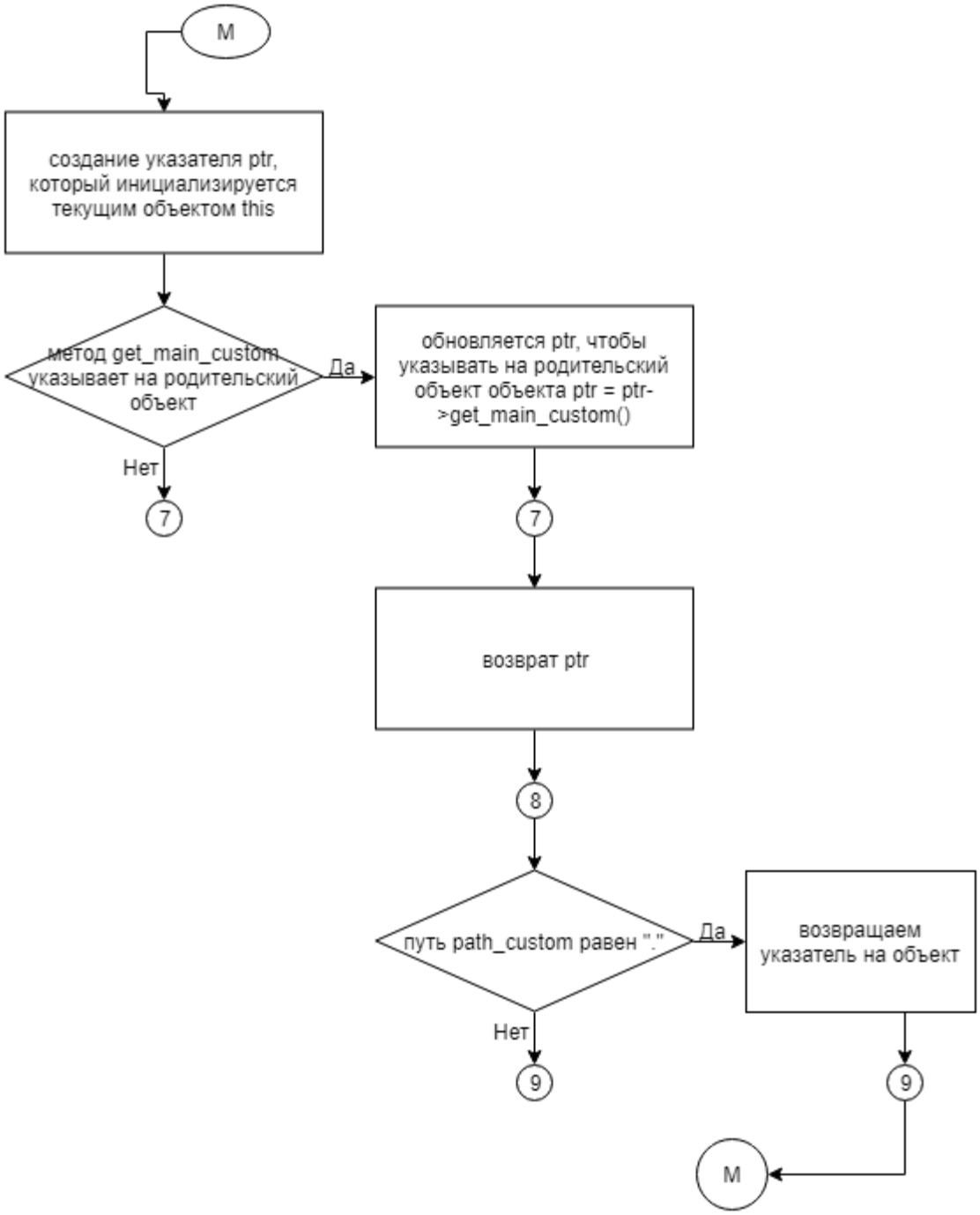

Рисунок 23 – Блок-схема алгоритма

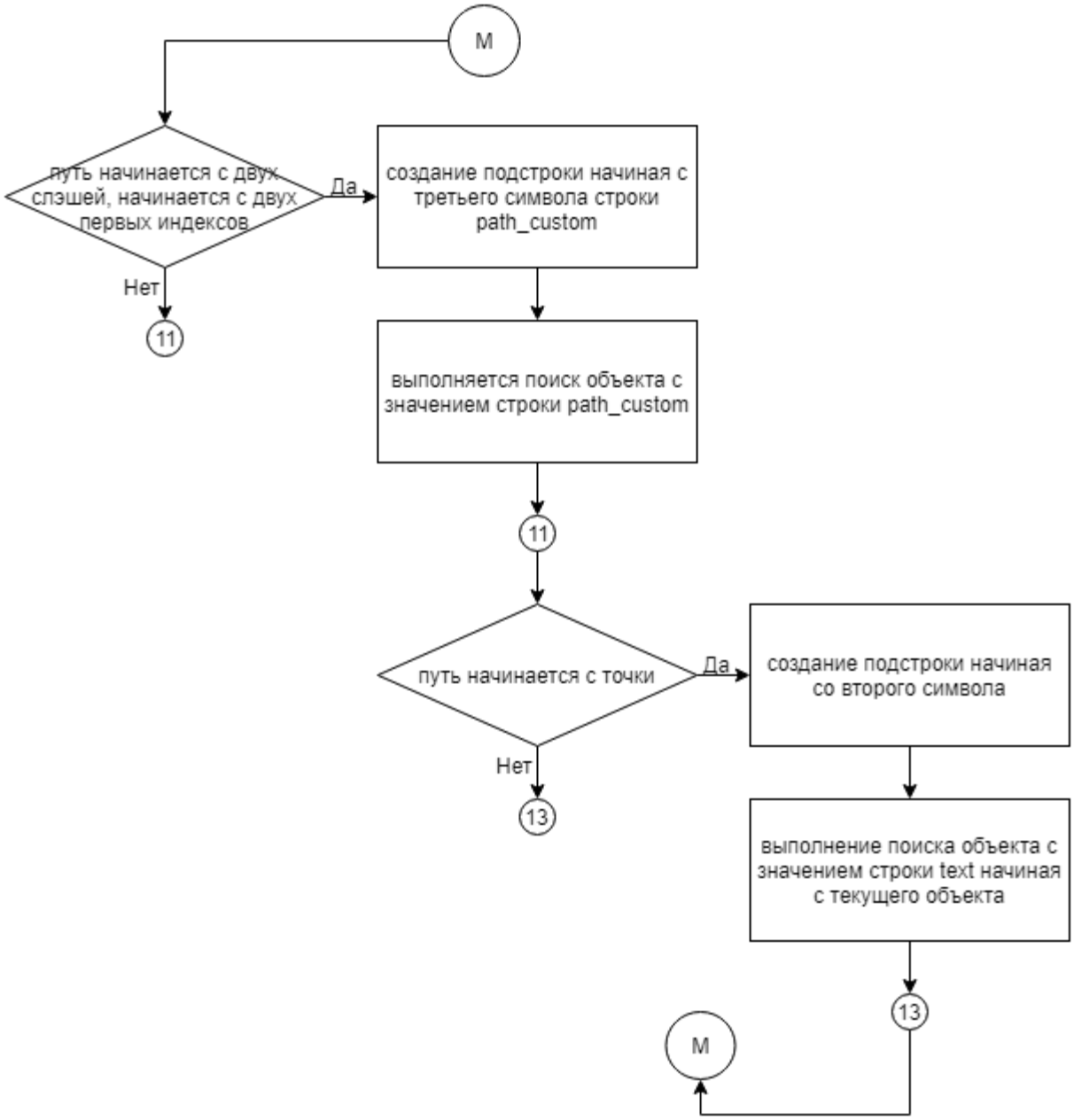

Рисунок 24 – Блок-схема алгоритма

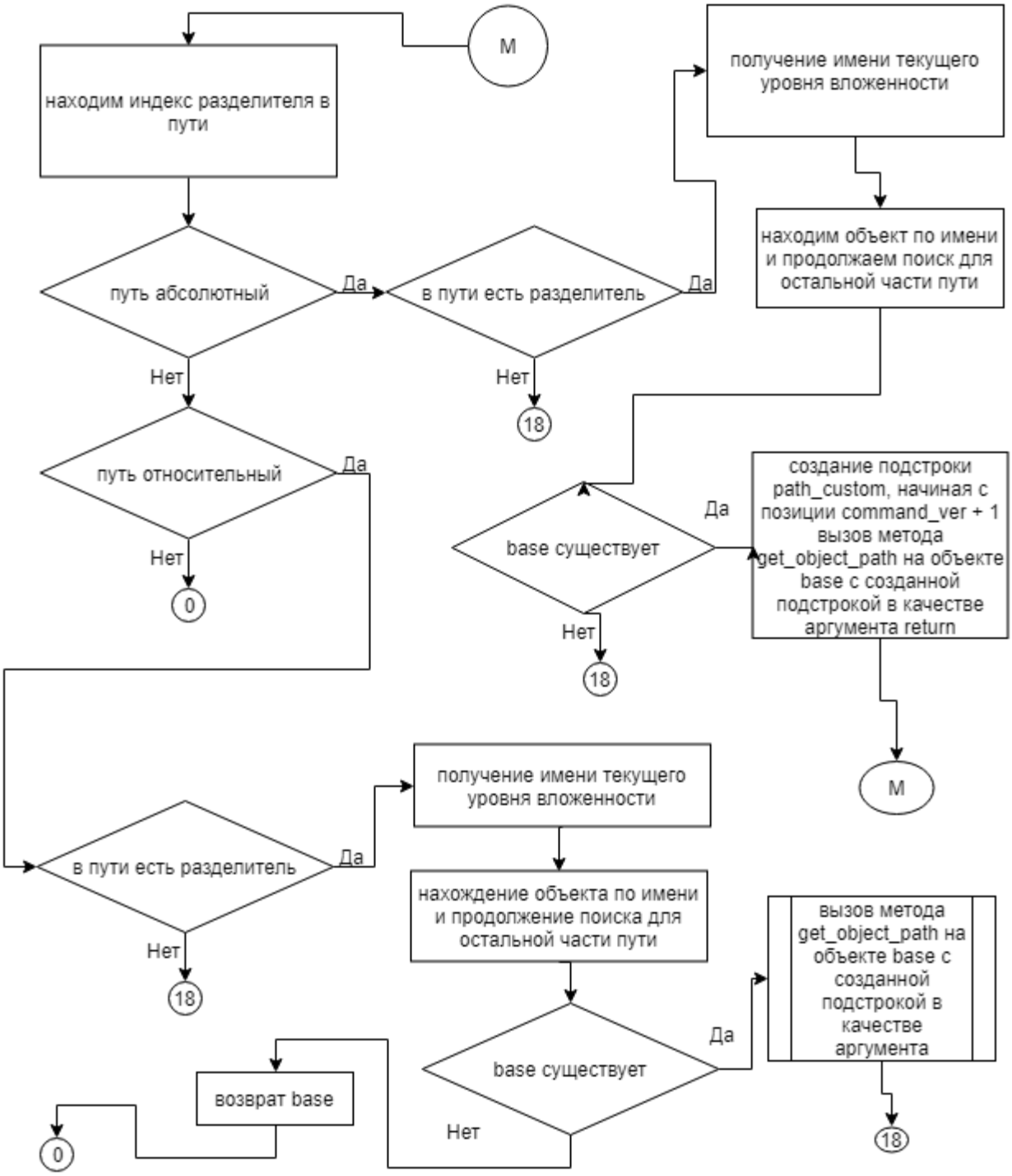

Рисунок 25 – Блок-схема алгоритма

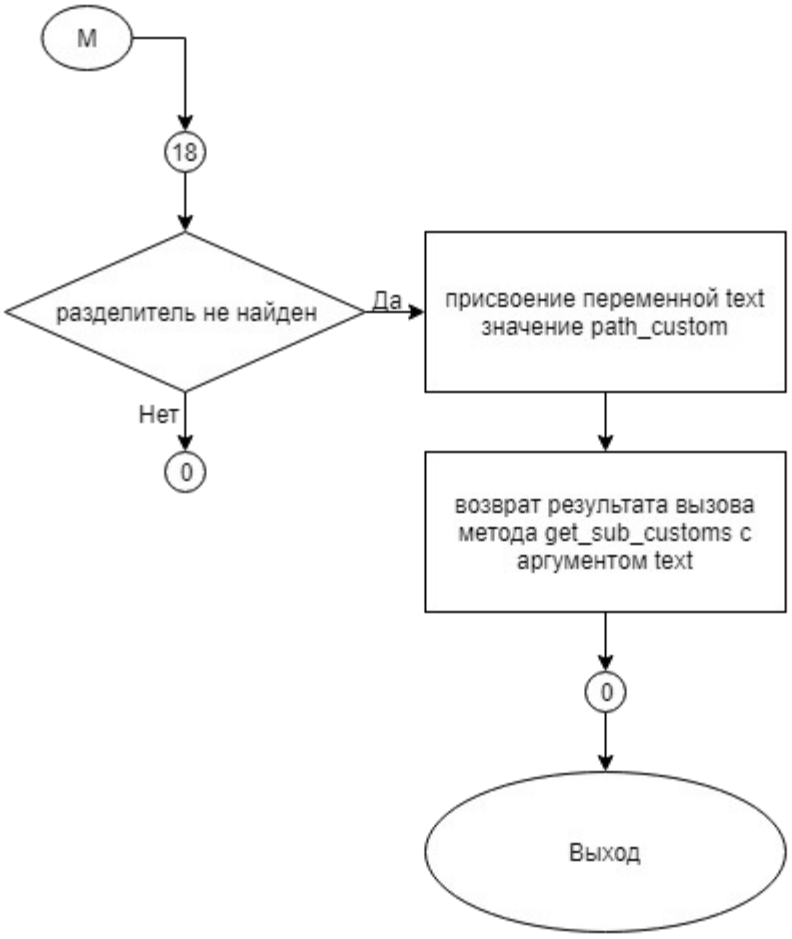

Рисунок 26 – Блок-схема алгоритма

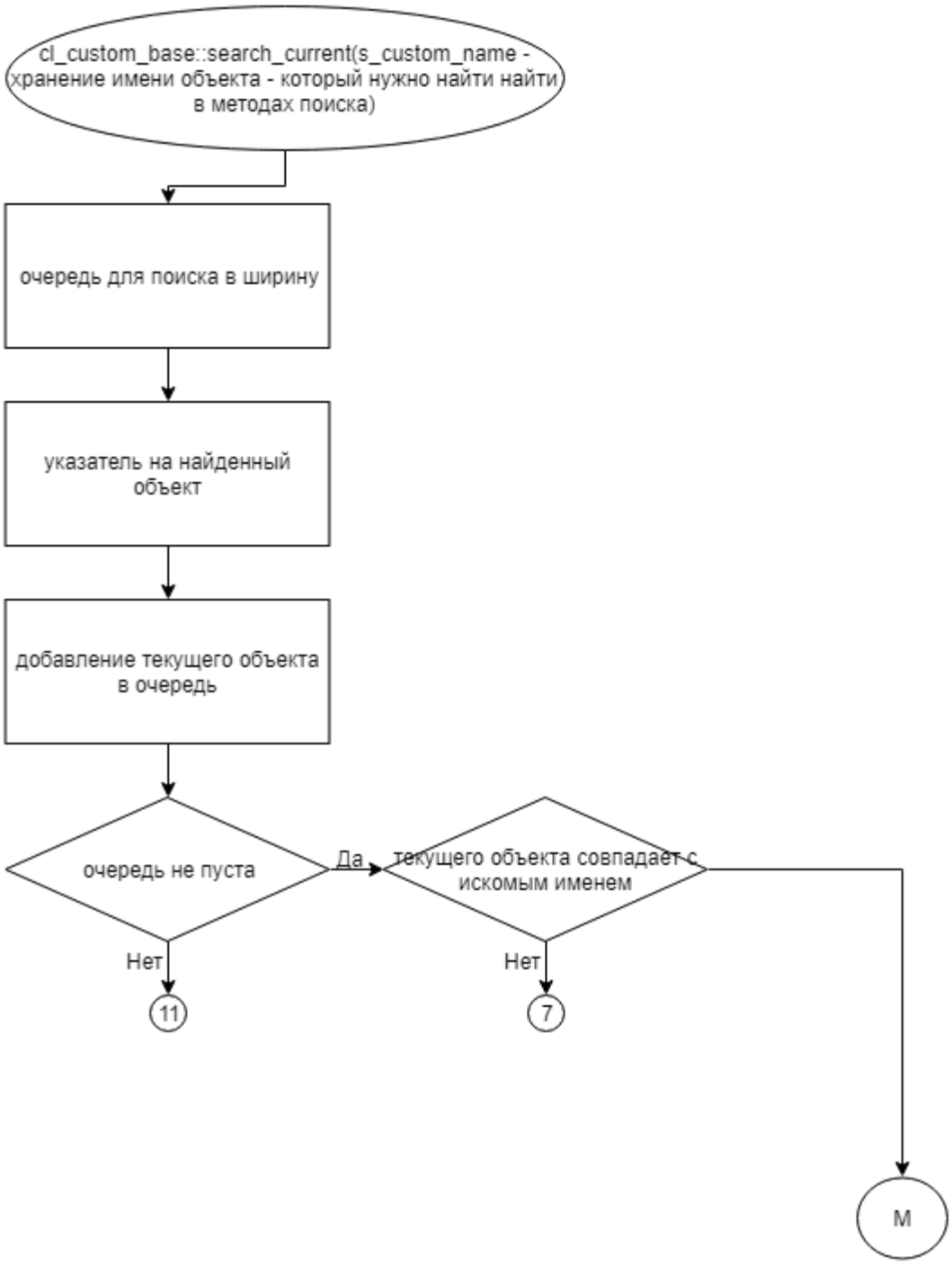

Рисунок 27 – Блок-схема алгоритма

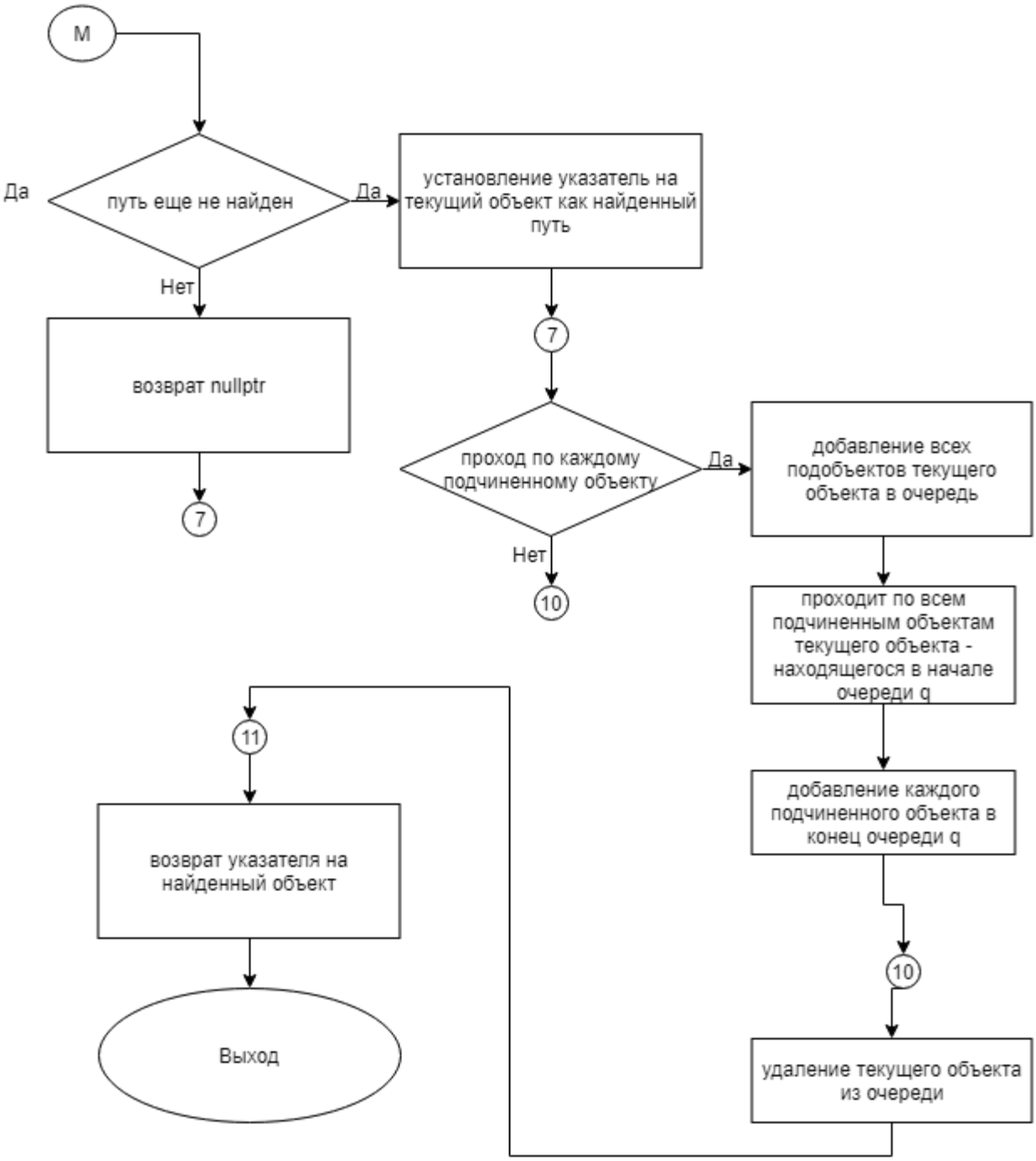

Рисунок 28 – Блок-схема алгоритма

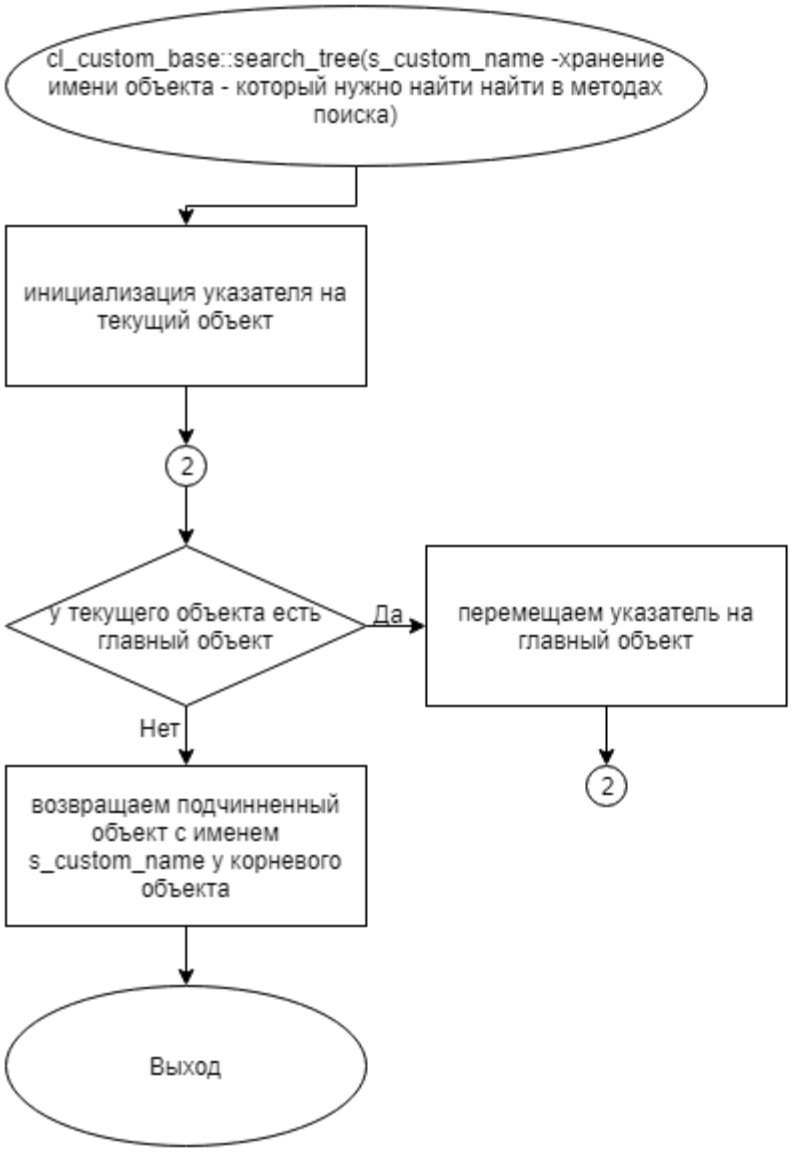

Рисунок 29 – Блок-схема алгоритма

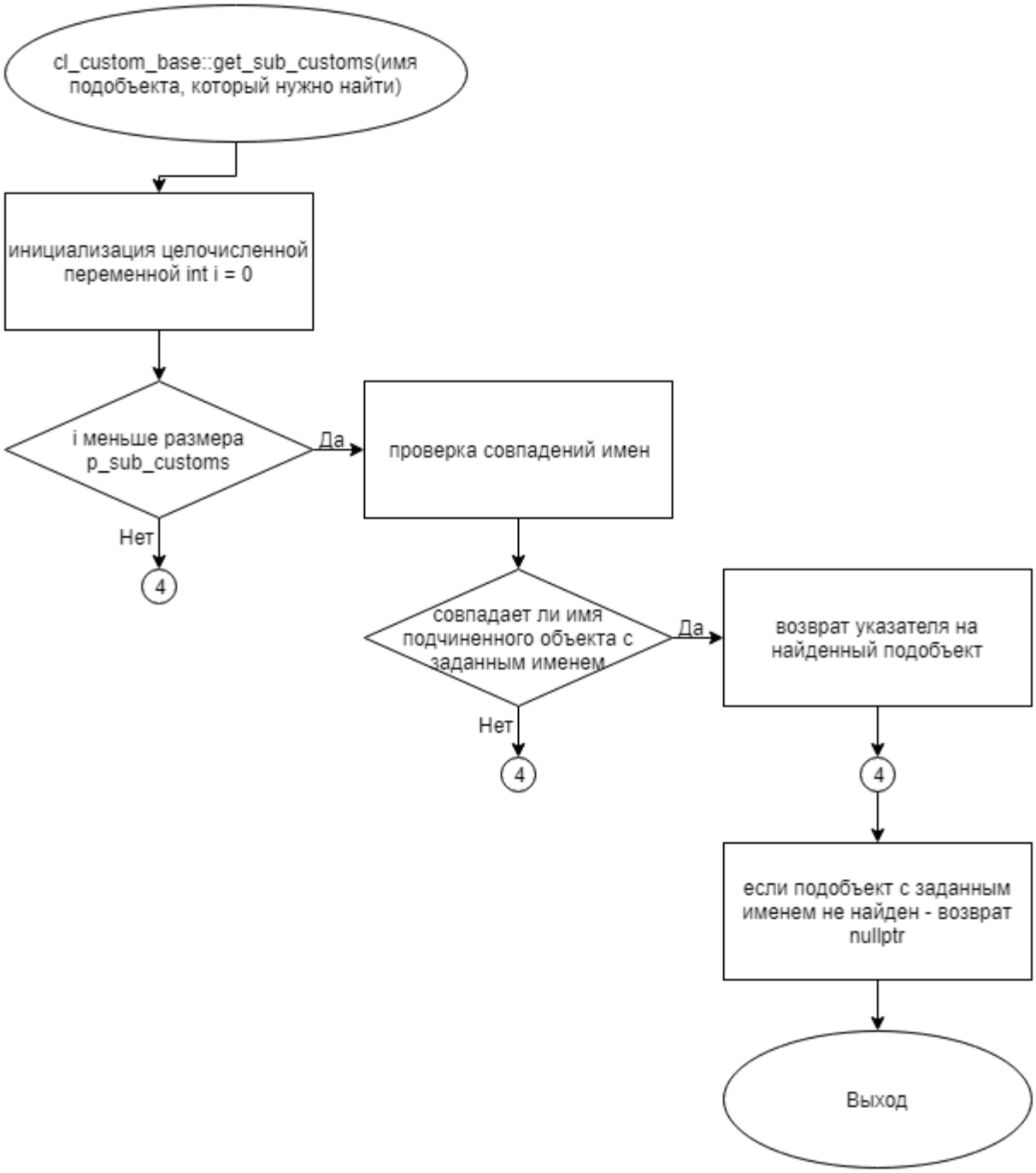

Рисунок 30 – Блок-схема алгоритма

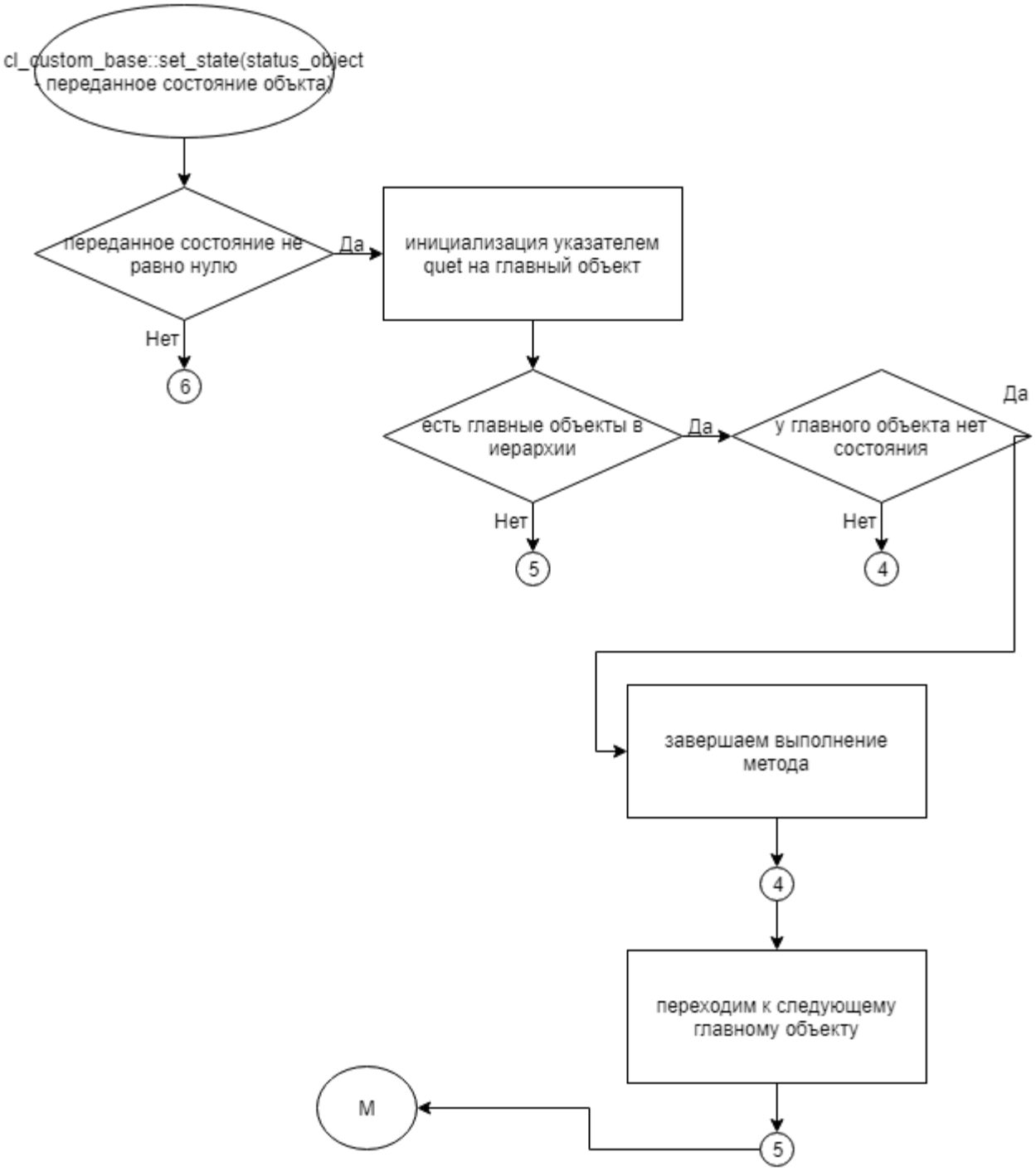

Рисунок 31 – Блок-схема алгоритма

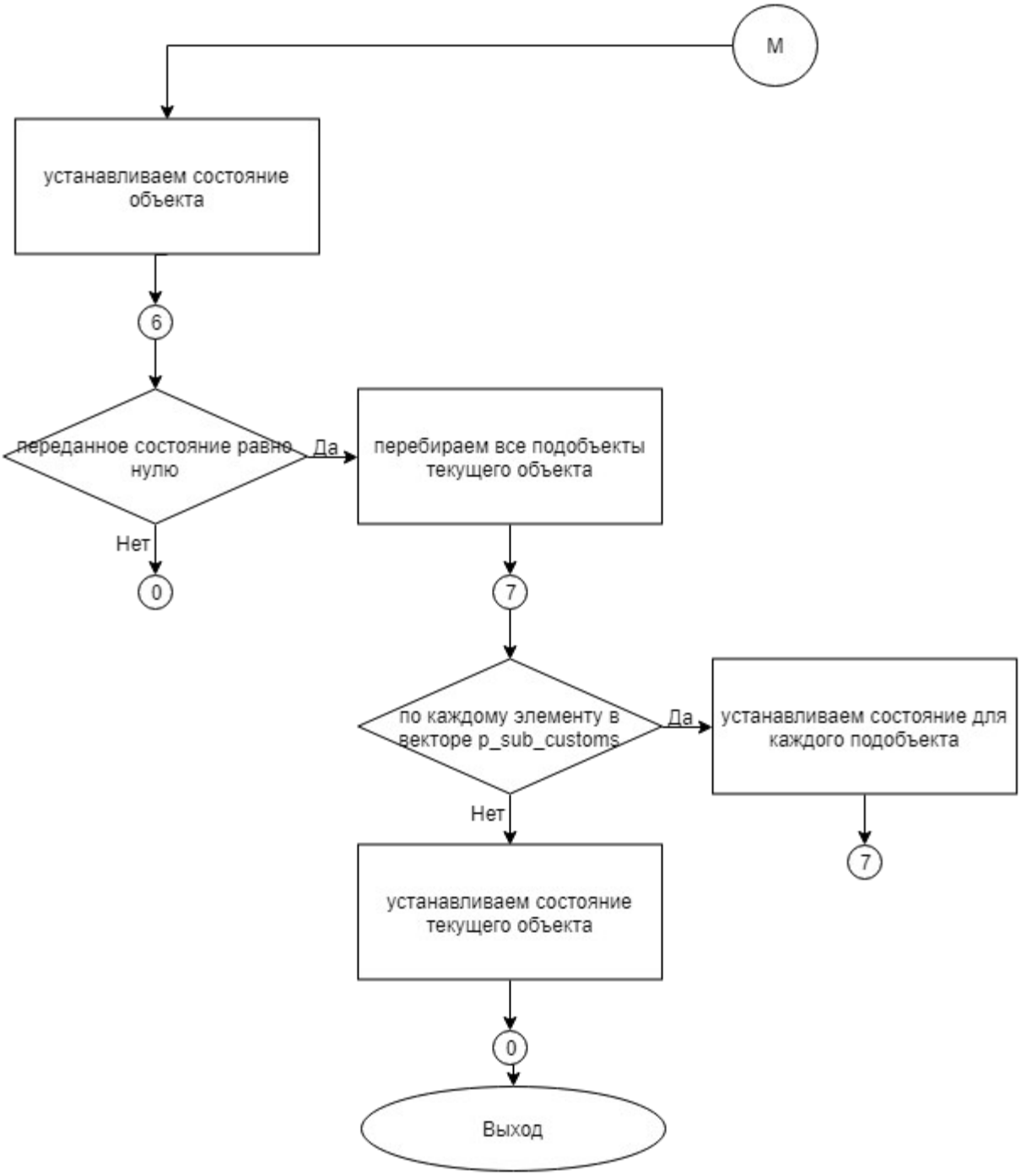

Рисунок 32 – Блок-схема алгоритма

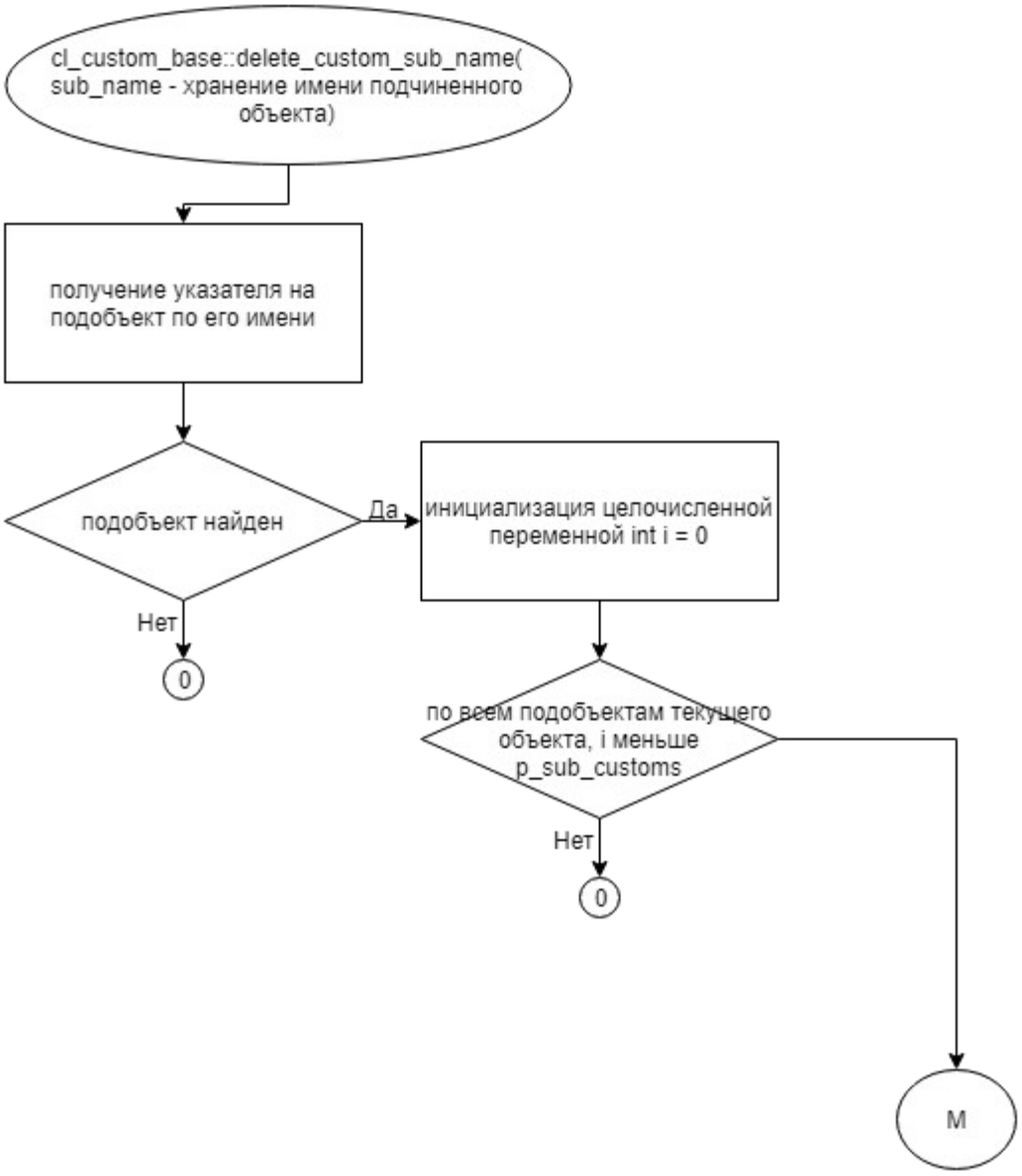

Рисунок 33 – Блок-схема алгоритма

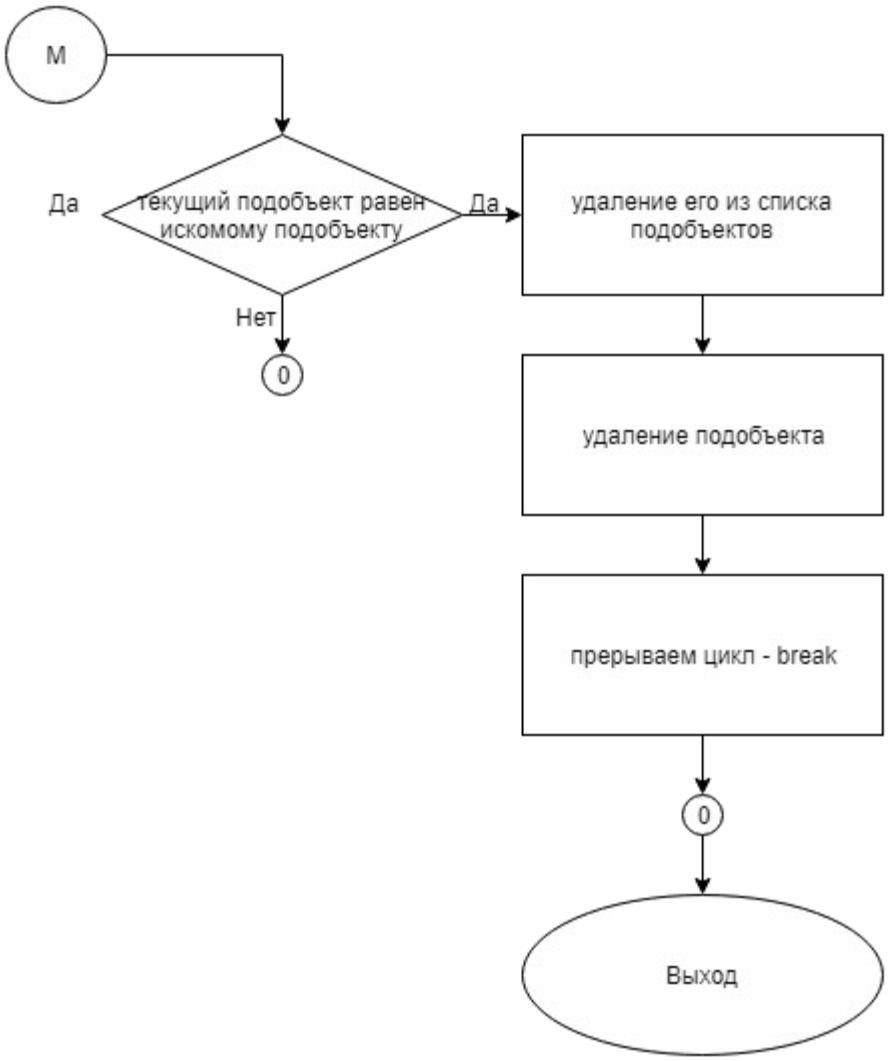

Рисунок 34 – Блок-схема алгоритма

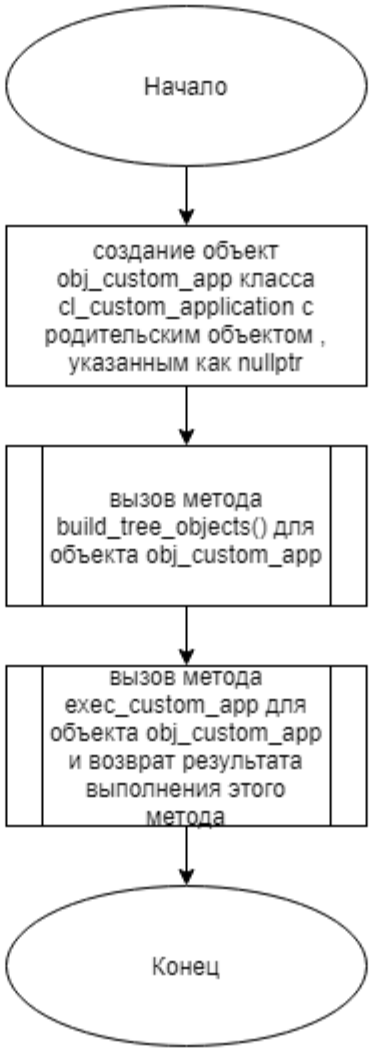

Рисунок 35 – Блок-схема алгоритма

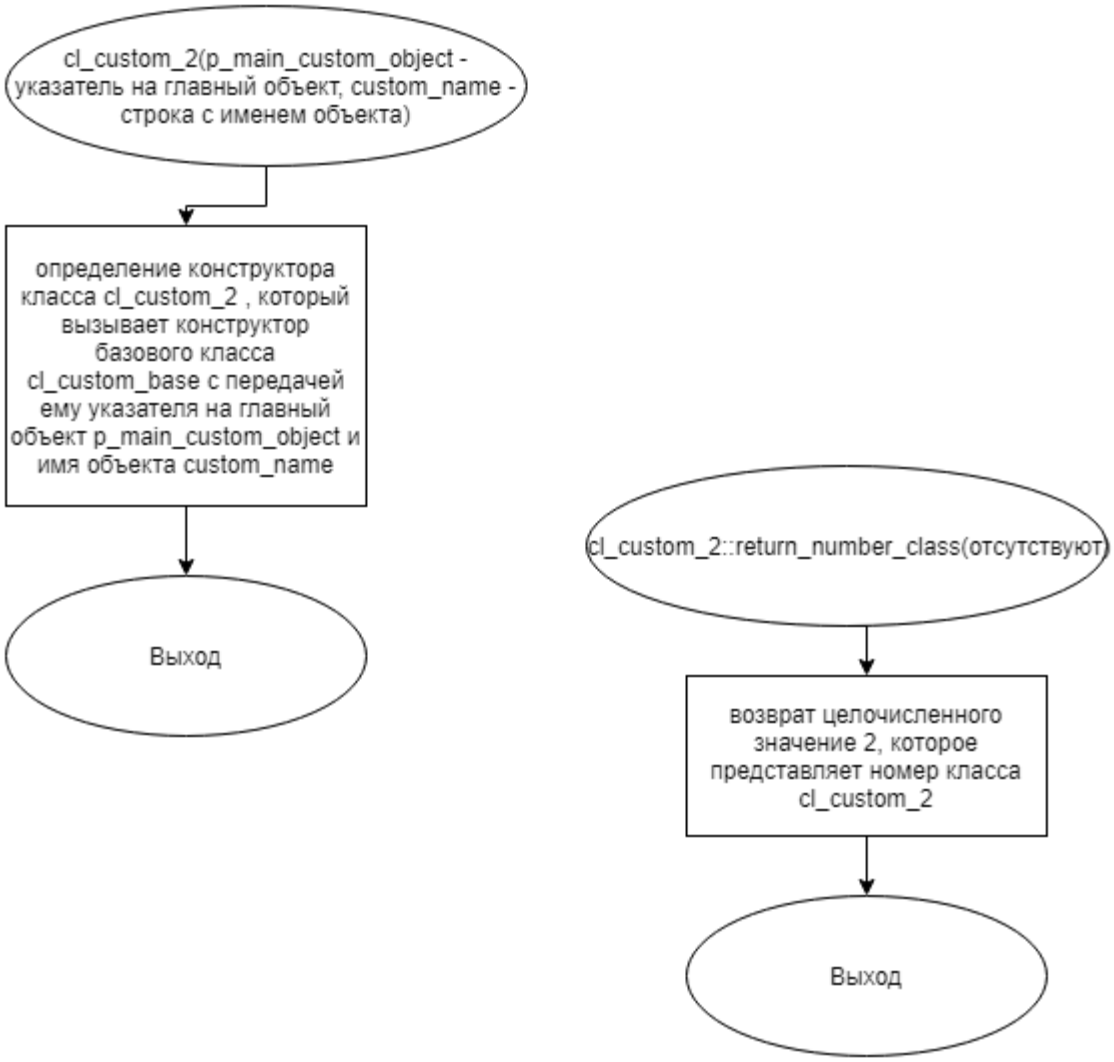

Рисунок 36 – Блок-схема алгоритма

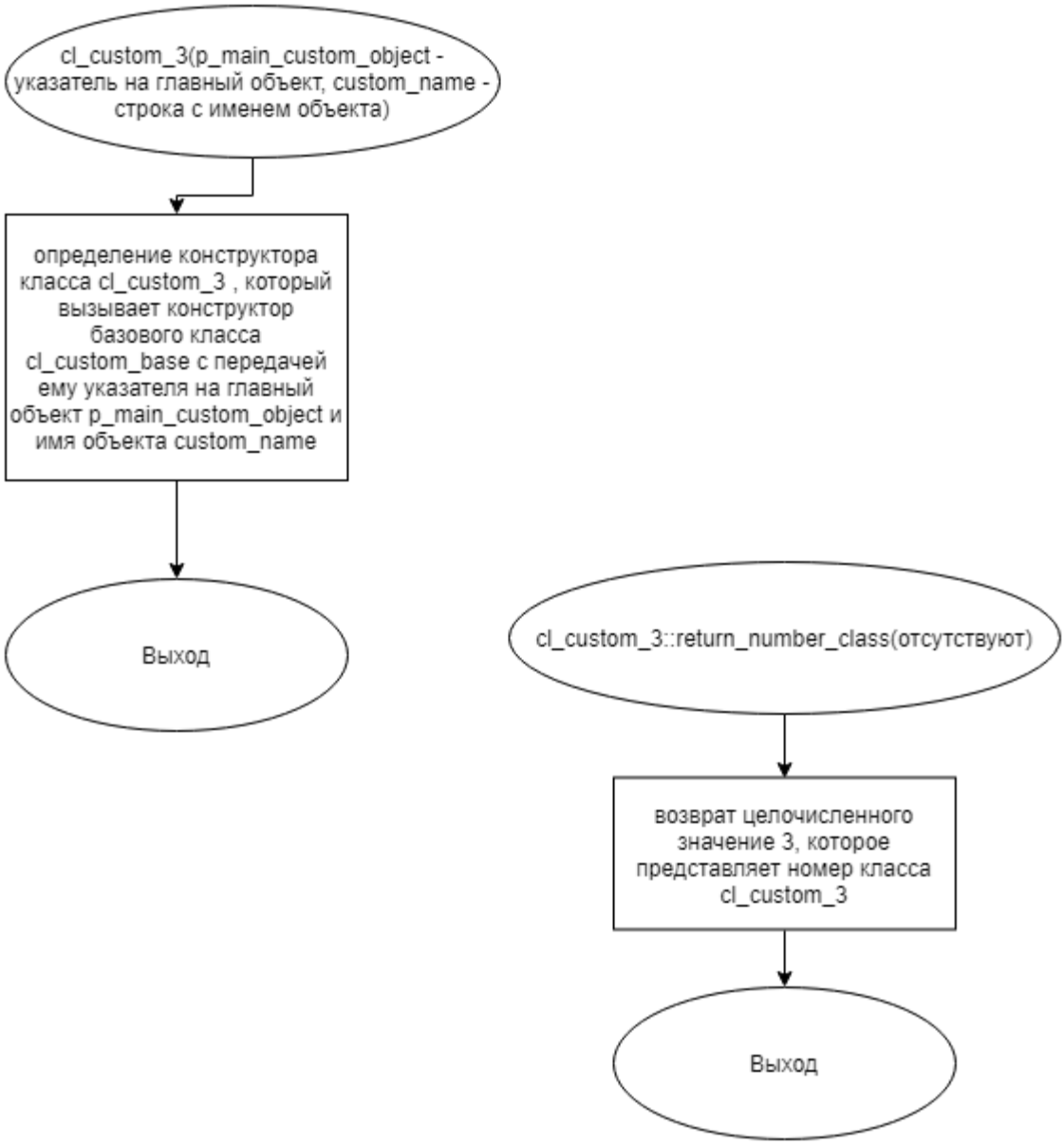

Рисунок 37 – Блок-схема алгоритма

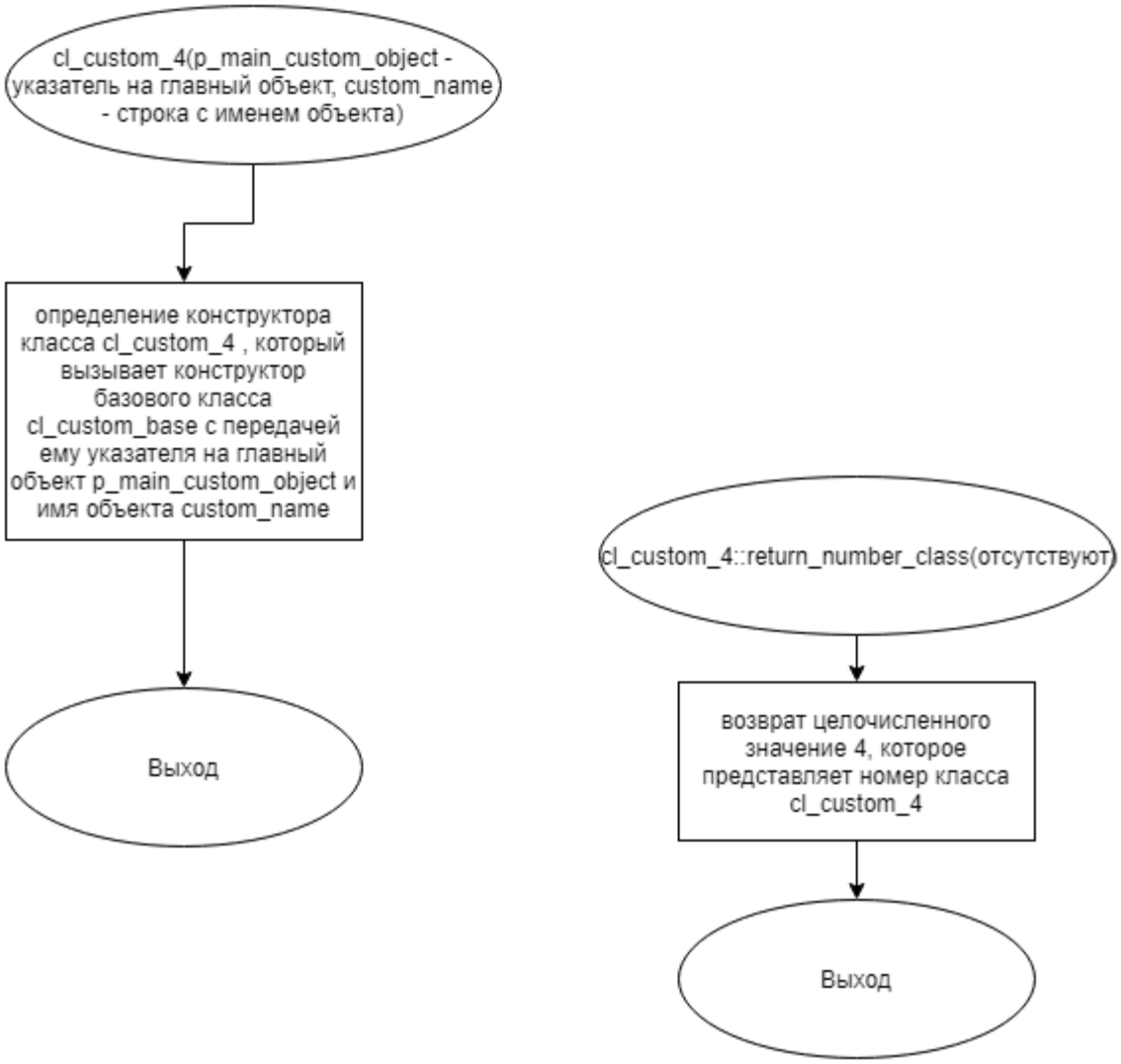

Рисунок 38 – Блок-схема алгоритма

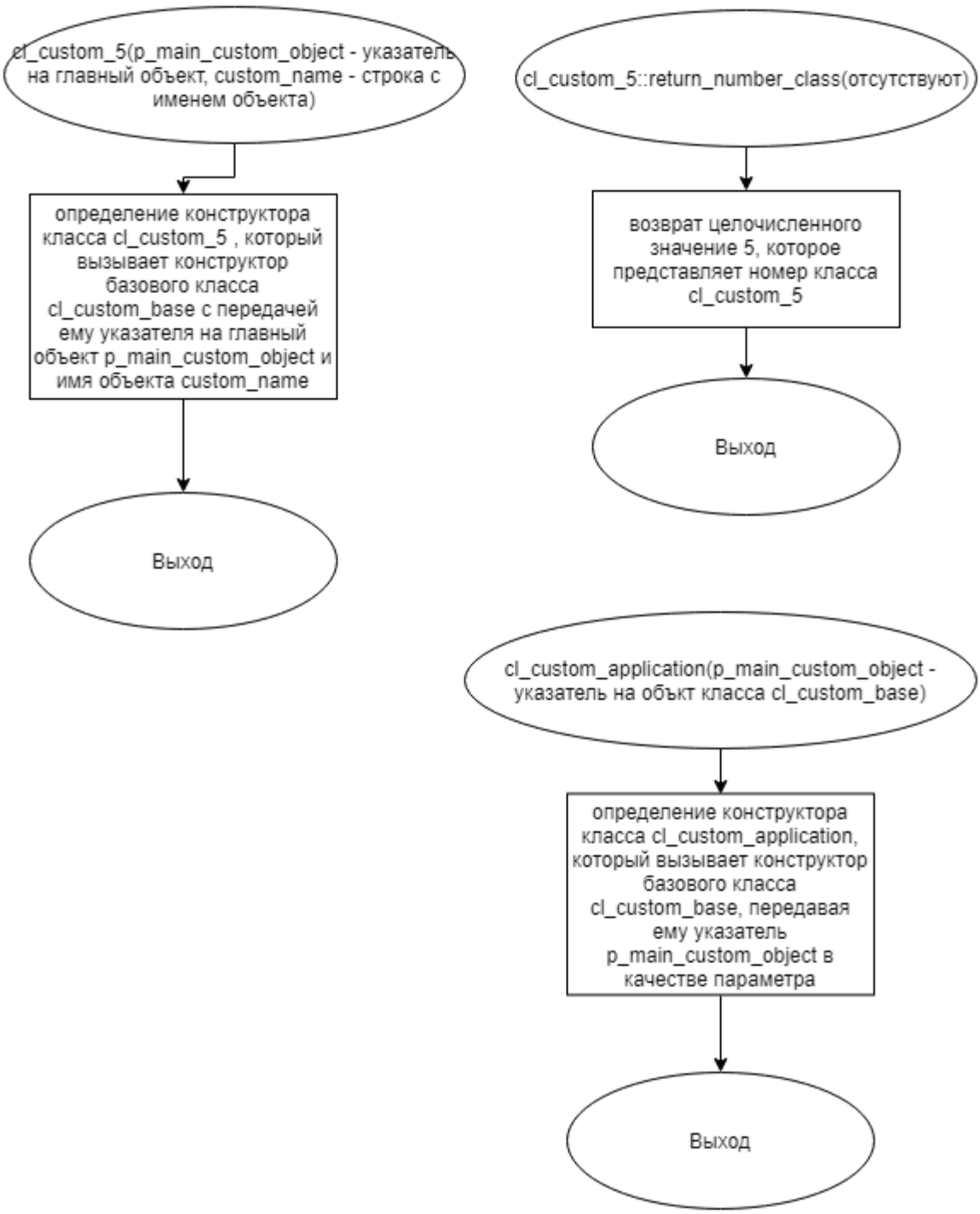

Рисунок 39 – Блок-схема алгоритма

Рисунок 40 – Блок-схема алгоритма

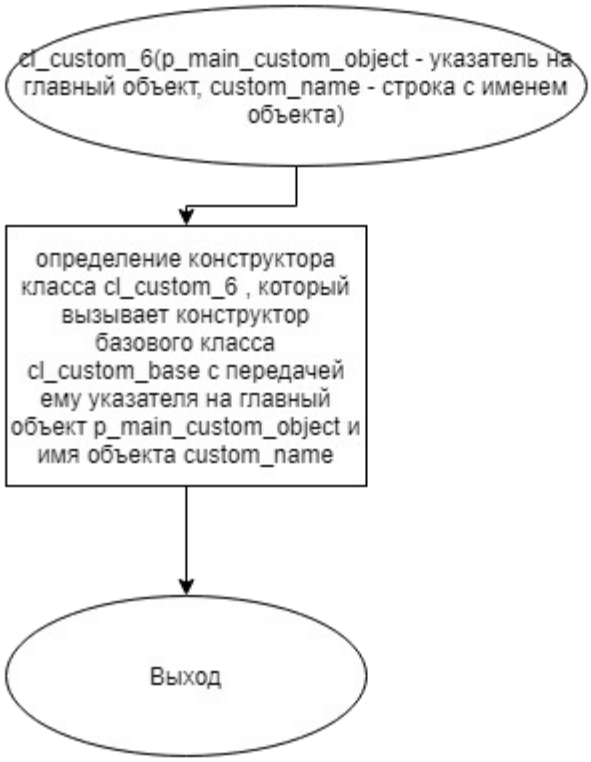

Рисунок 41 – Блок-схема алгоритма

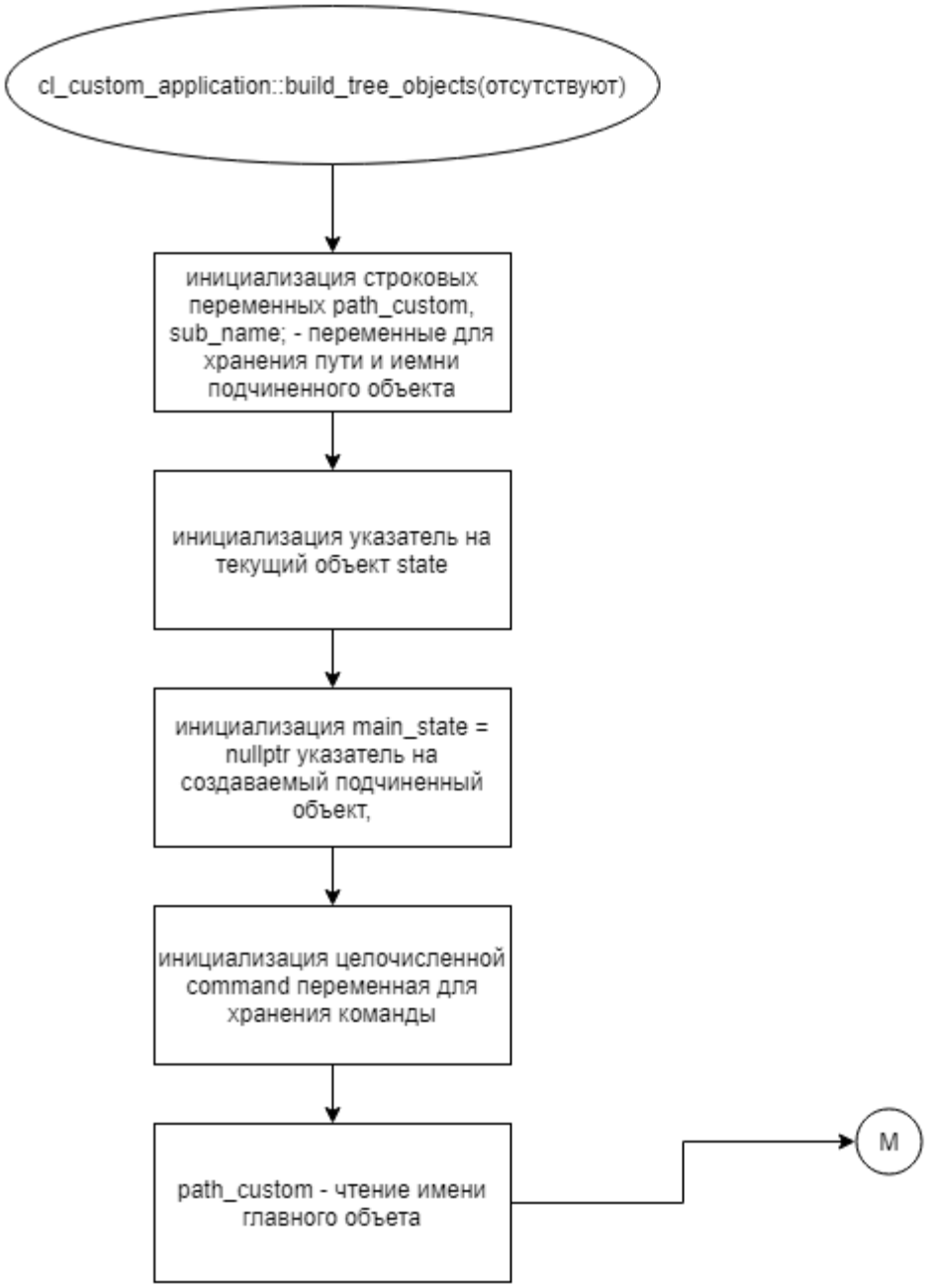

Рисунок 42 – Блок-схема алгоритма

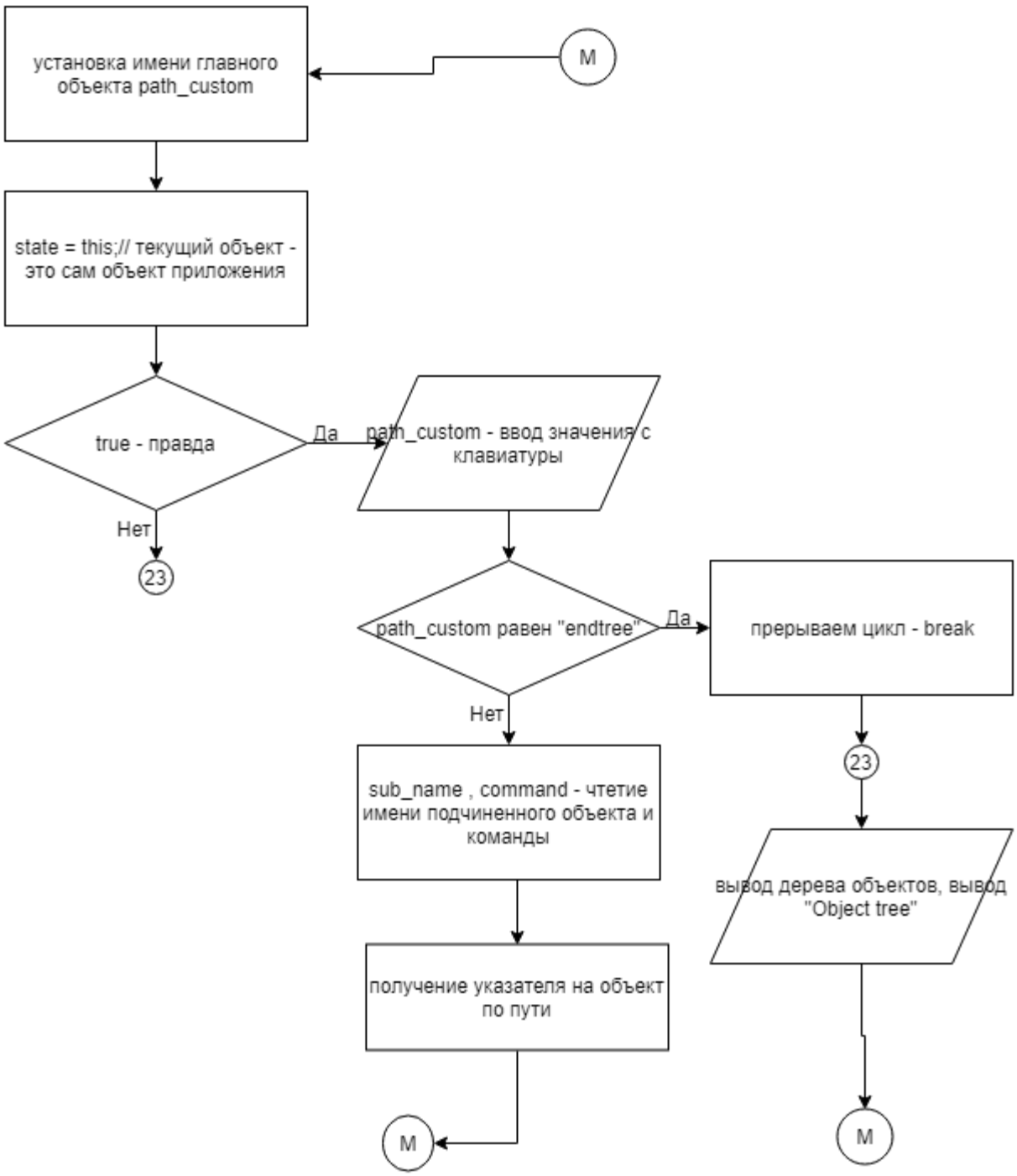

Рисунок 43 – Блок-схема алгоритма

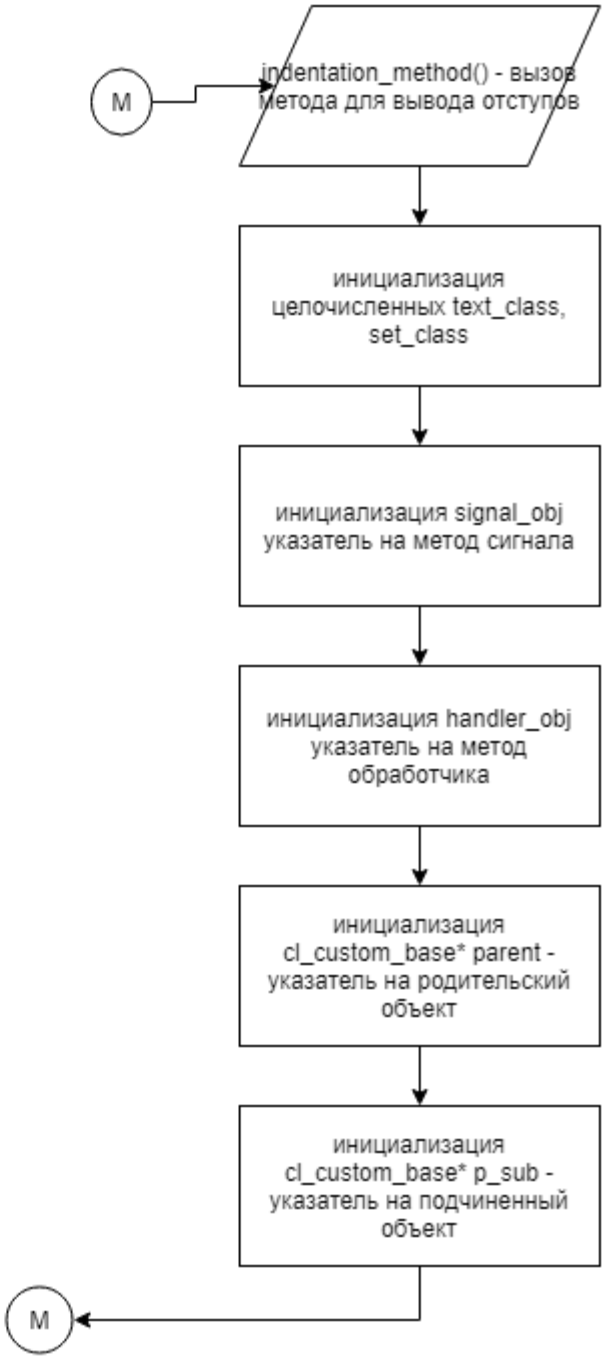

Рисунок 44 – Блок-схема алгоритма

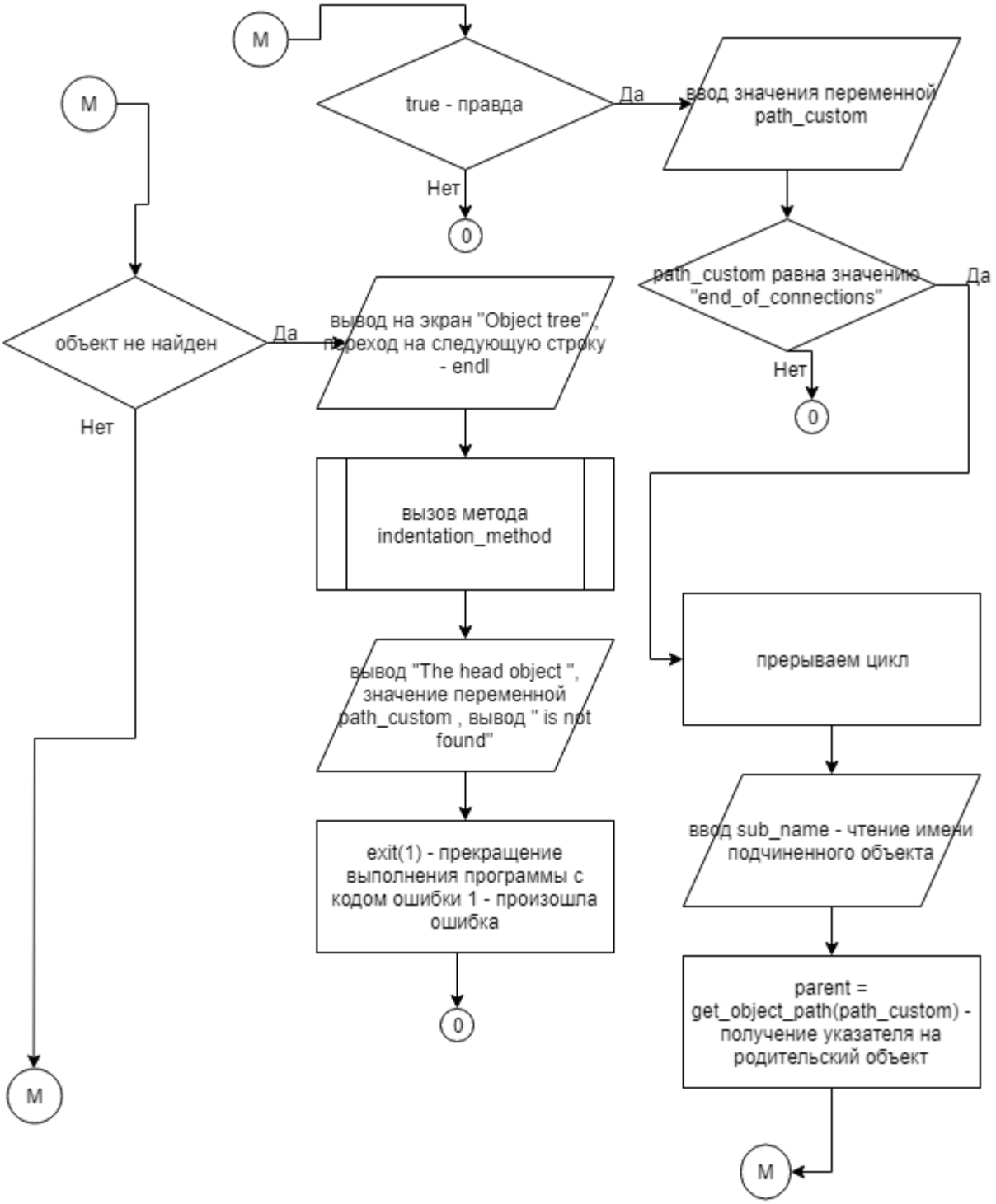

Рисунок 45 – Блок-схема алгоритма

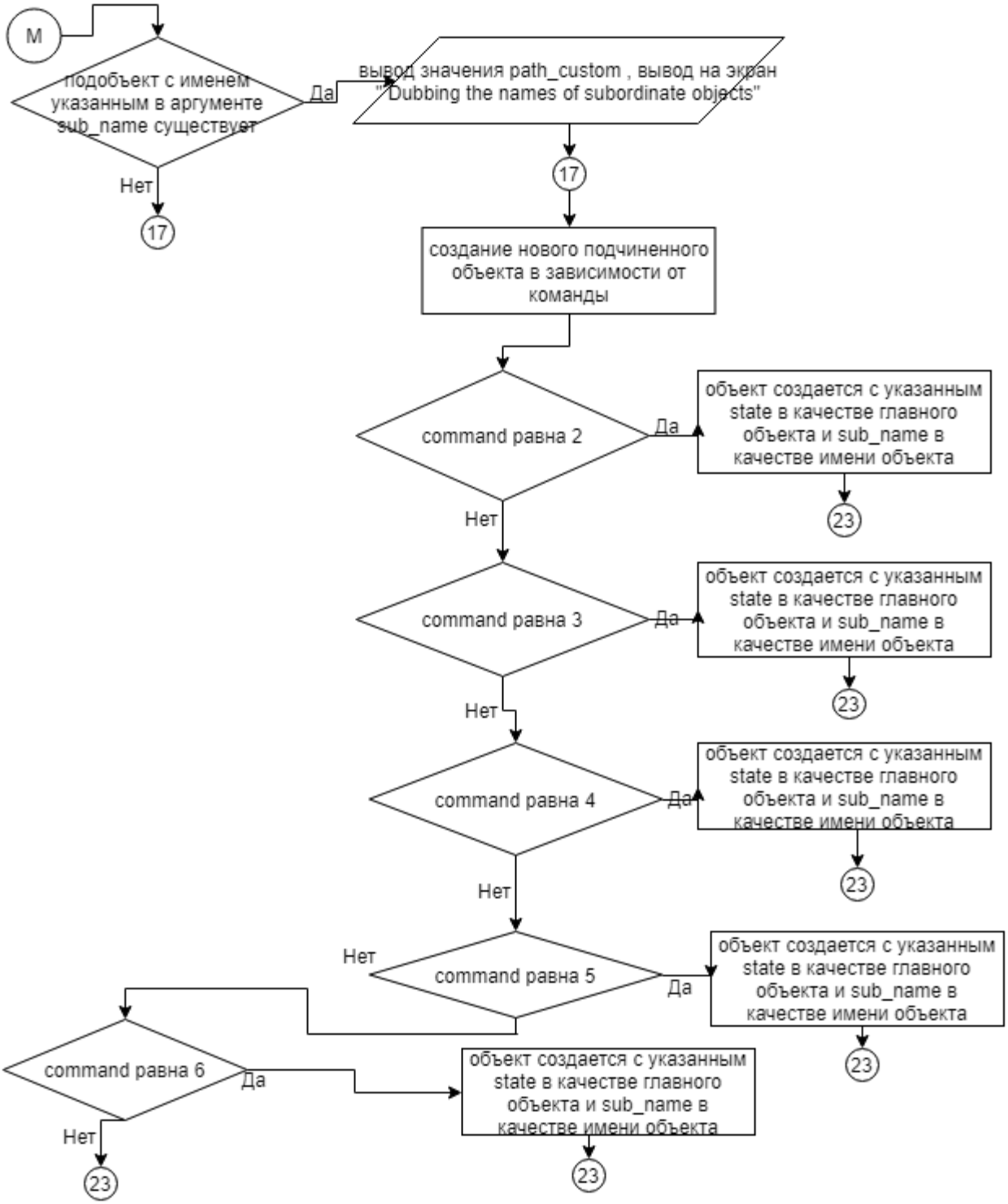

Рисунок 46 – Блок-схема алгоритма

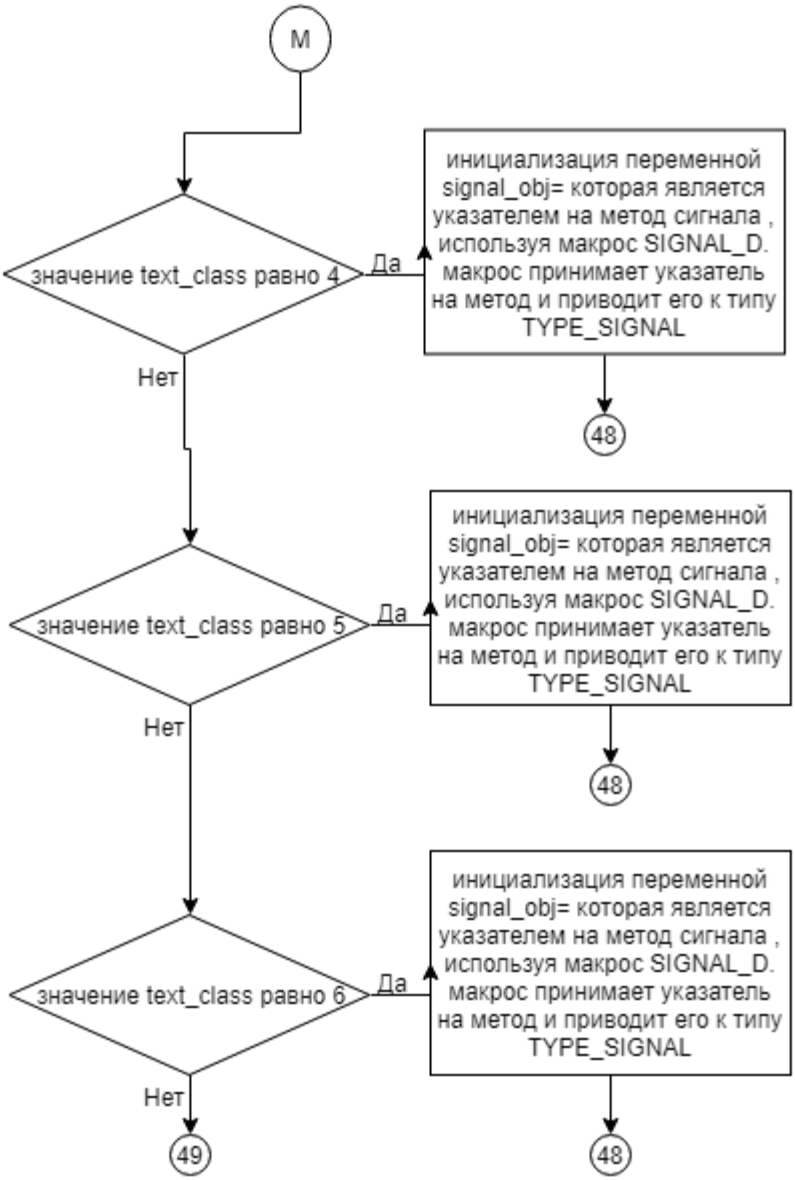

Рисунок 47 – Блок-схема алгоритма

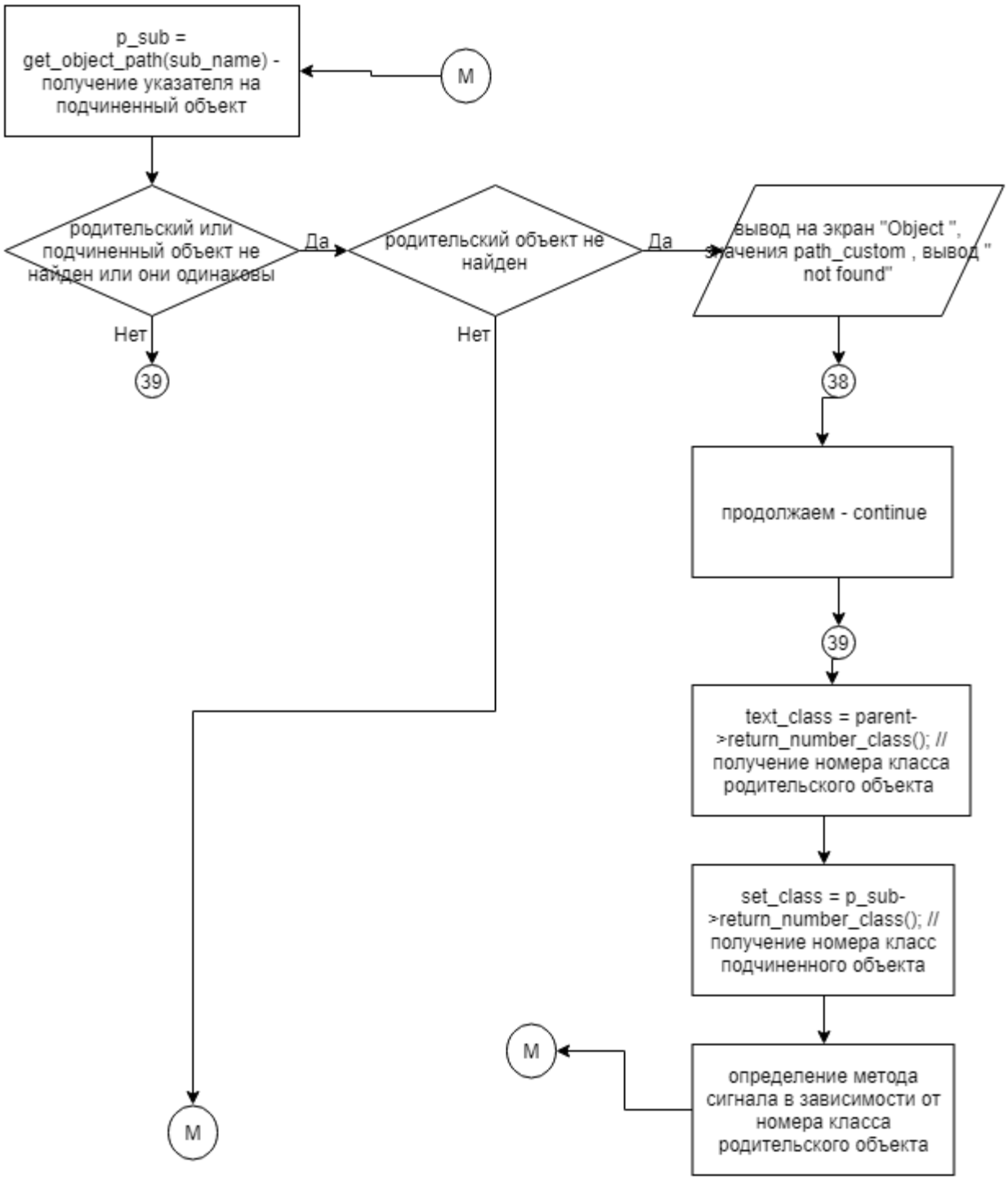

Рисунок 48 – Блок-схема алгоритма

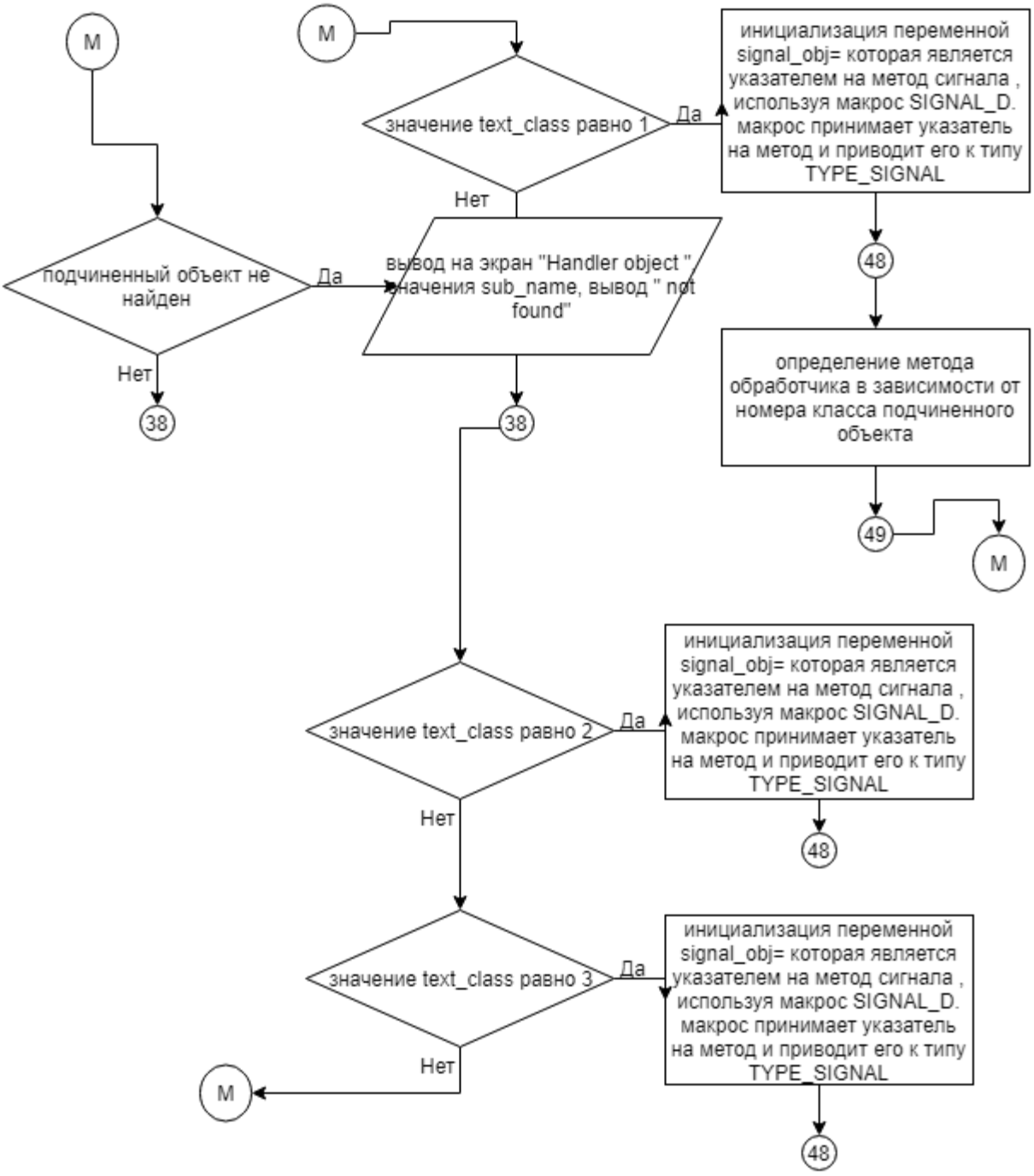

Рисунок 49 – Блок-схема алгоритма

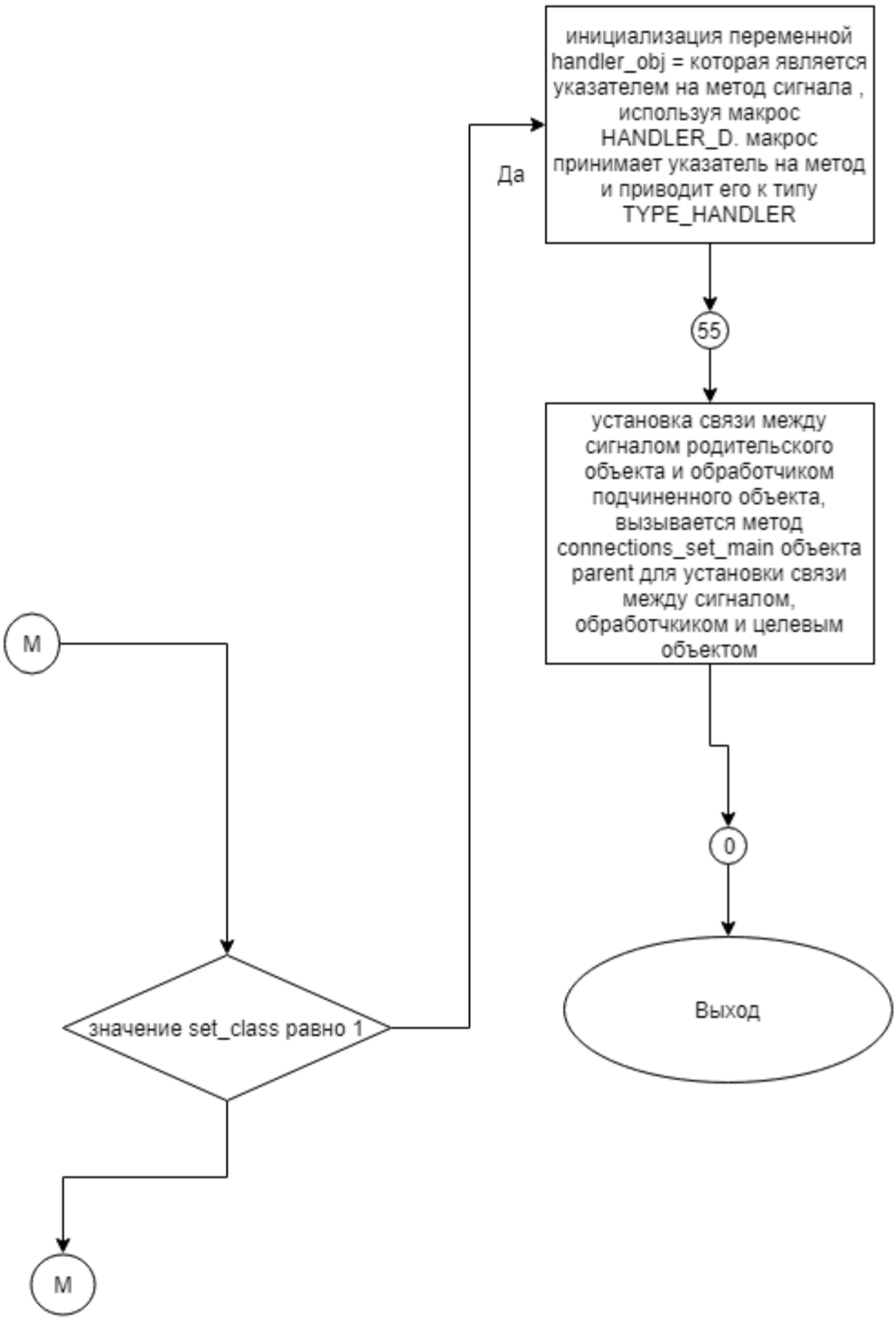

Рисунок 50 – Блок-схема алгоритма

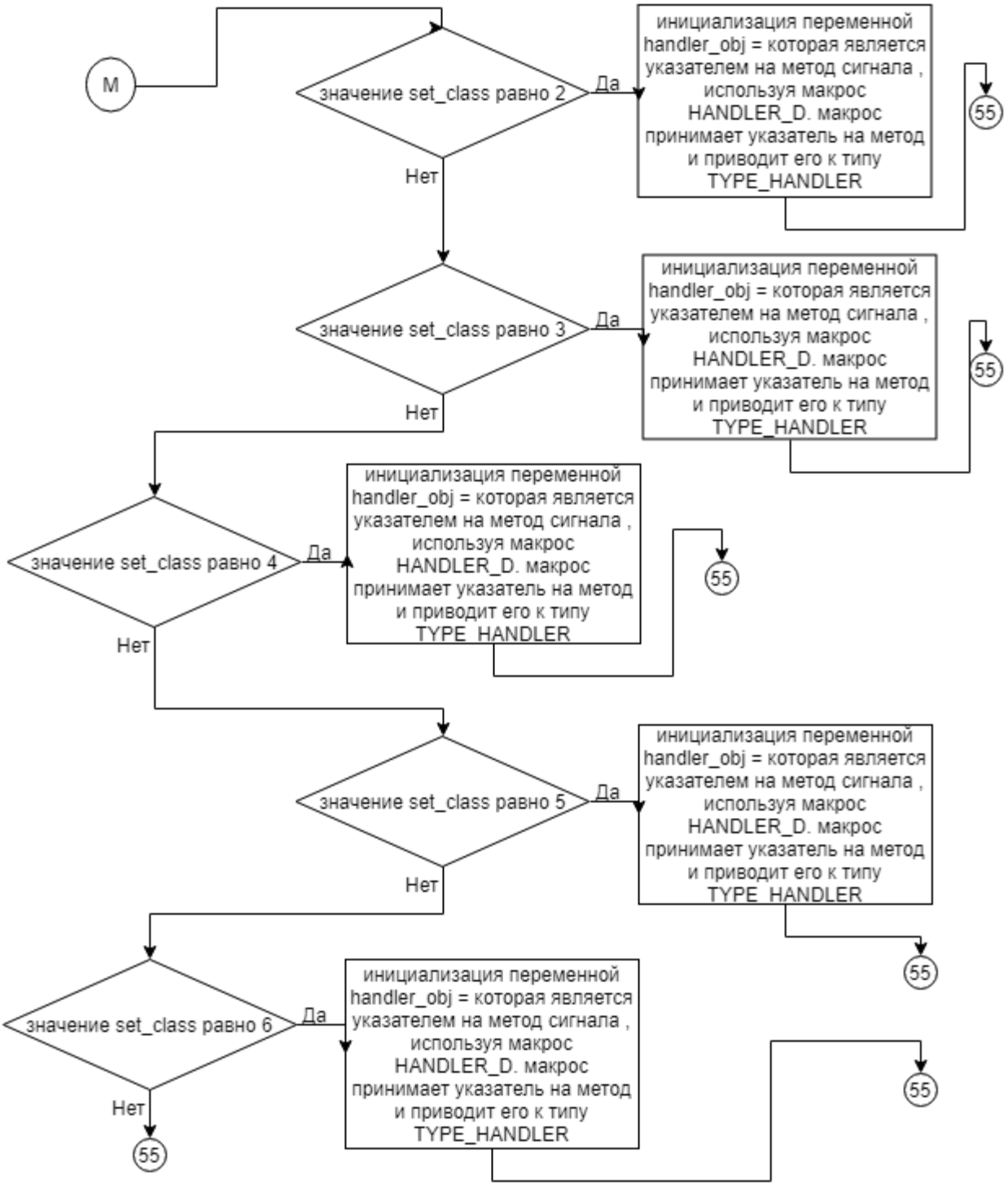

Рисунок 51 – Блок-схема алгоритма

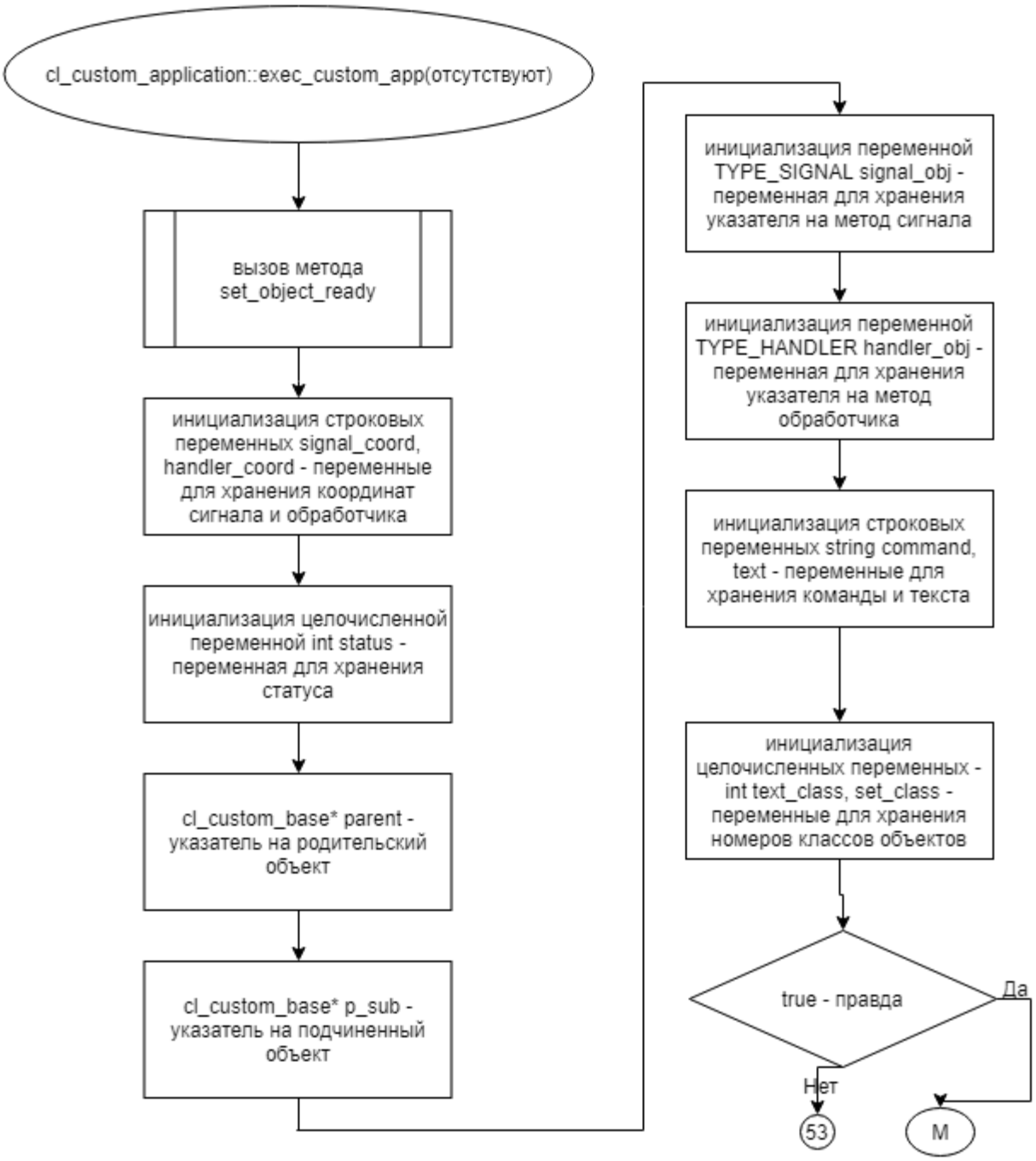

Рисунок 52 – Блок-схема алгоритма

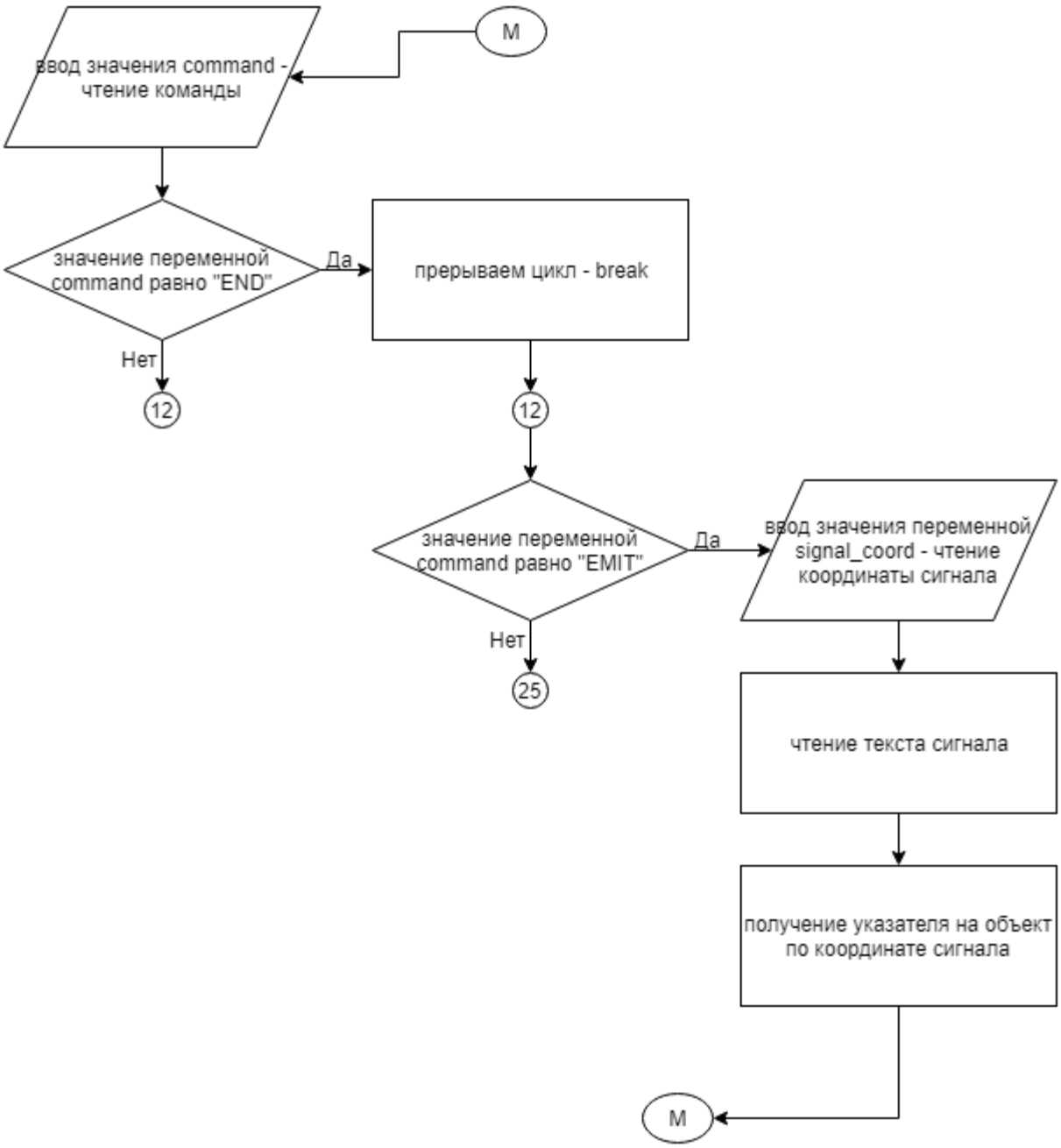

Рисунок 53 – Блок-схема алгоритма

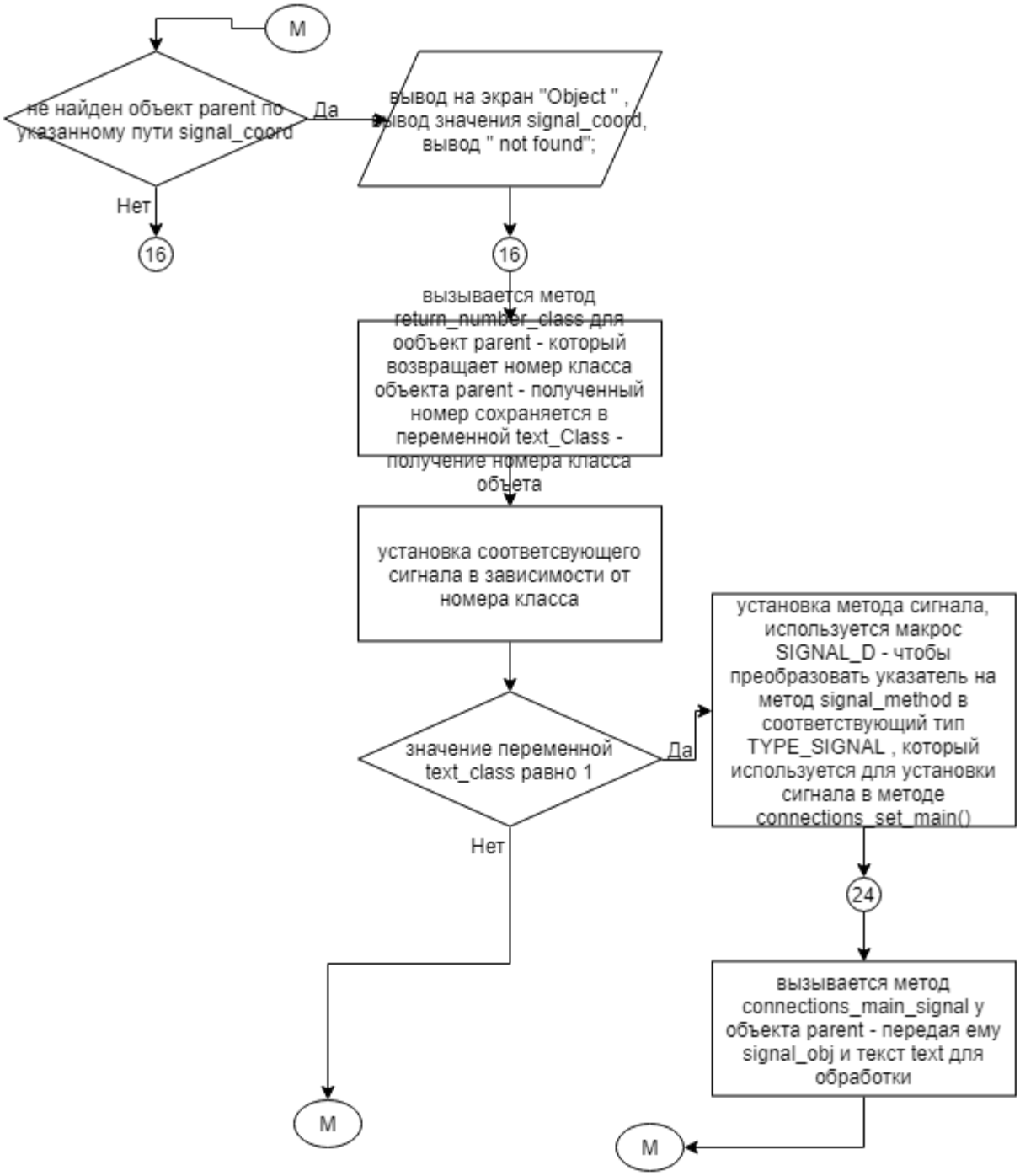

Рисунок 54 – Блок-схема алгоритма

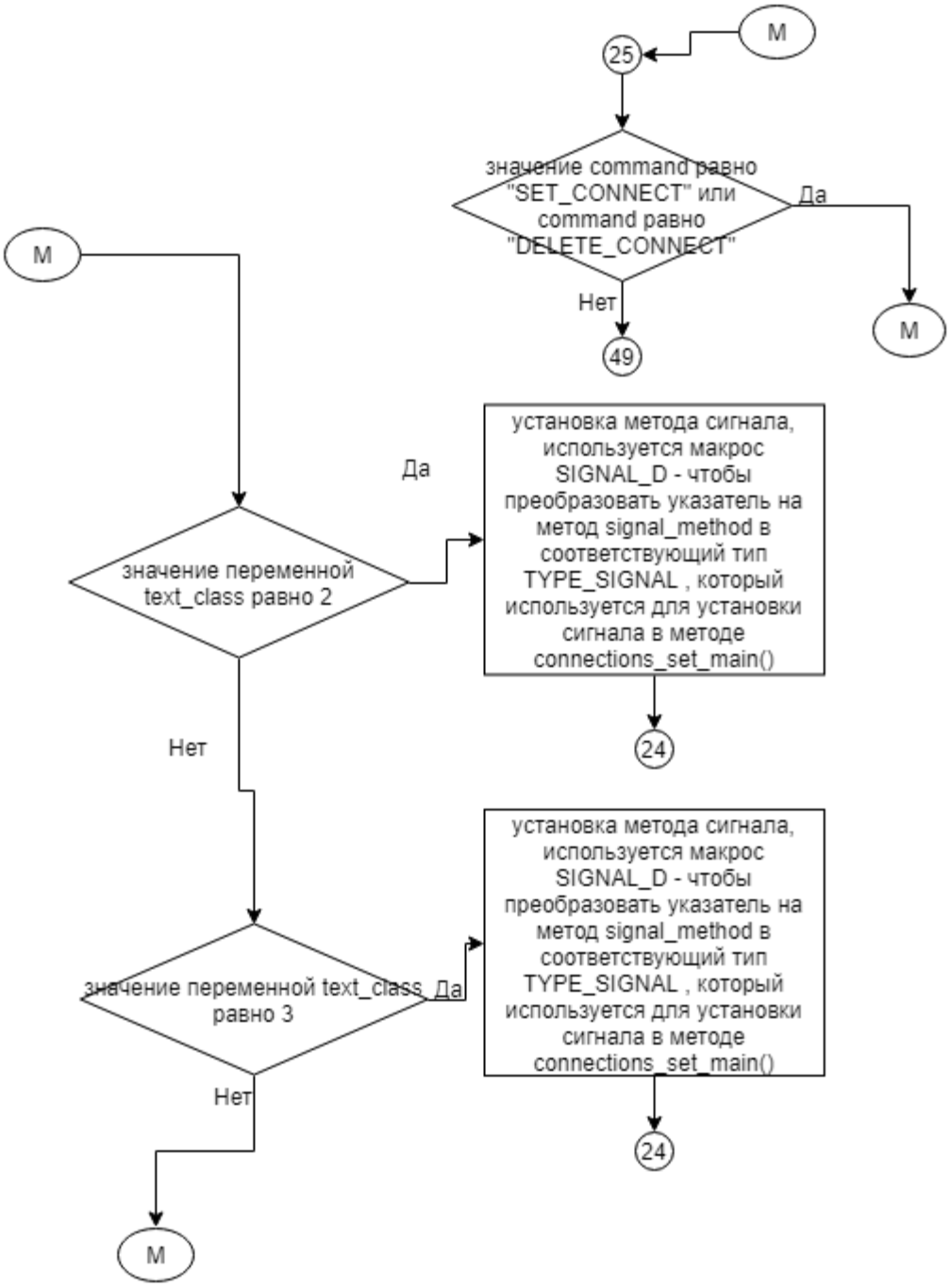

Рисунок 55 – Блок-схема алгоритма

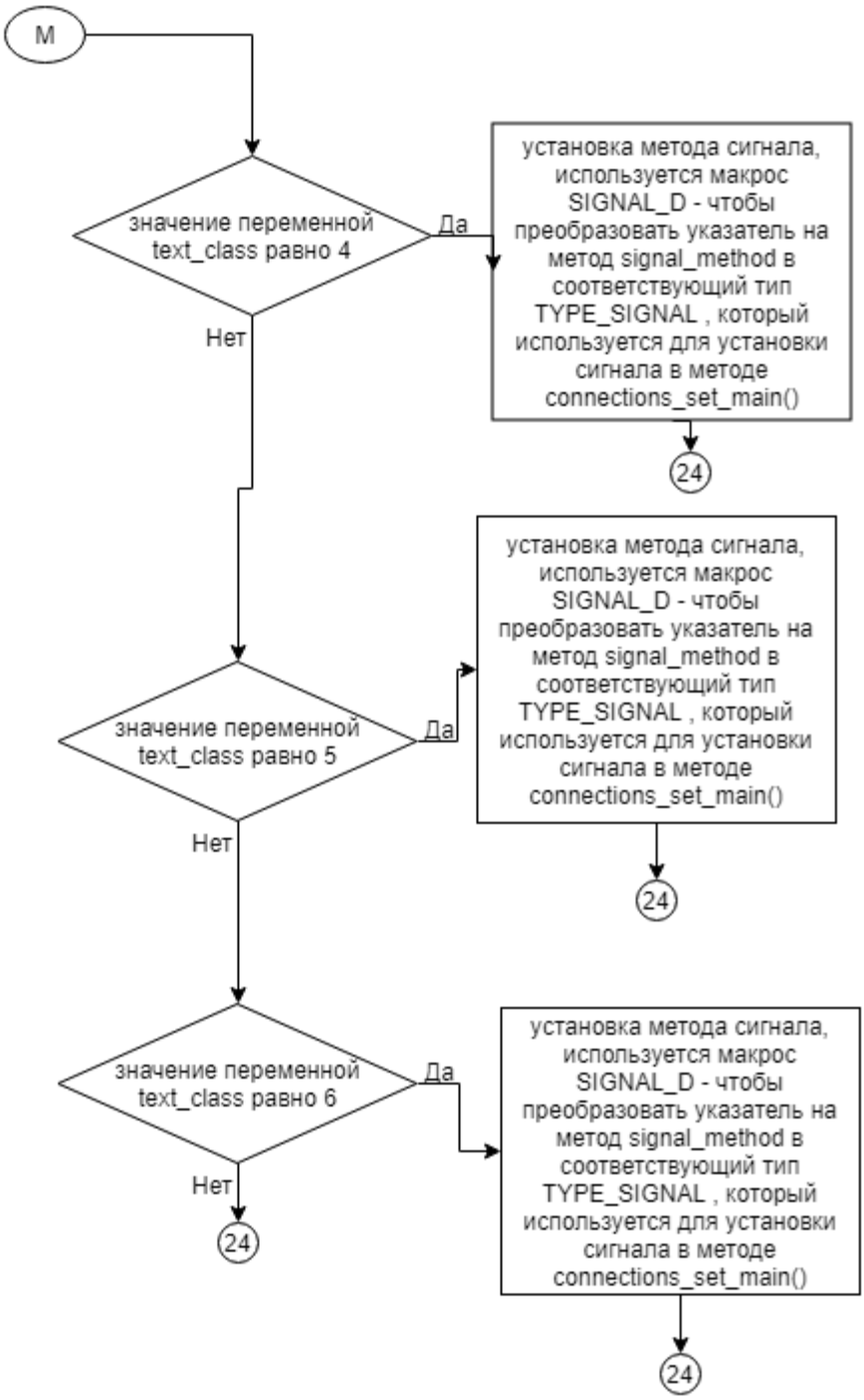

Рисунок 56 – Блок-схема алгоритма

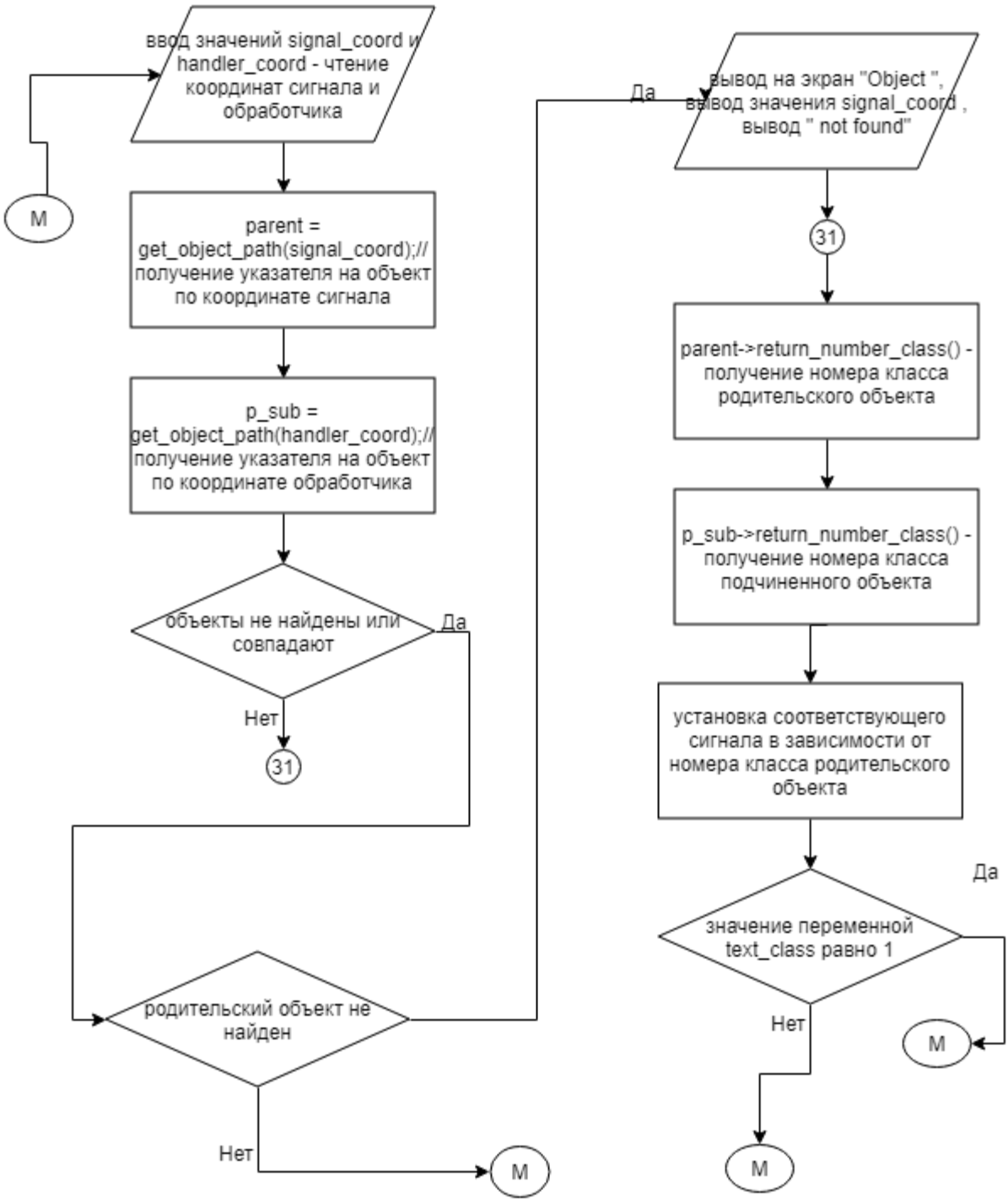

Рисунок 57 – Блок-схема алгоритма

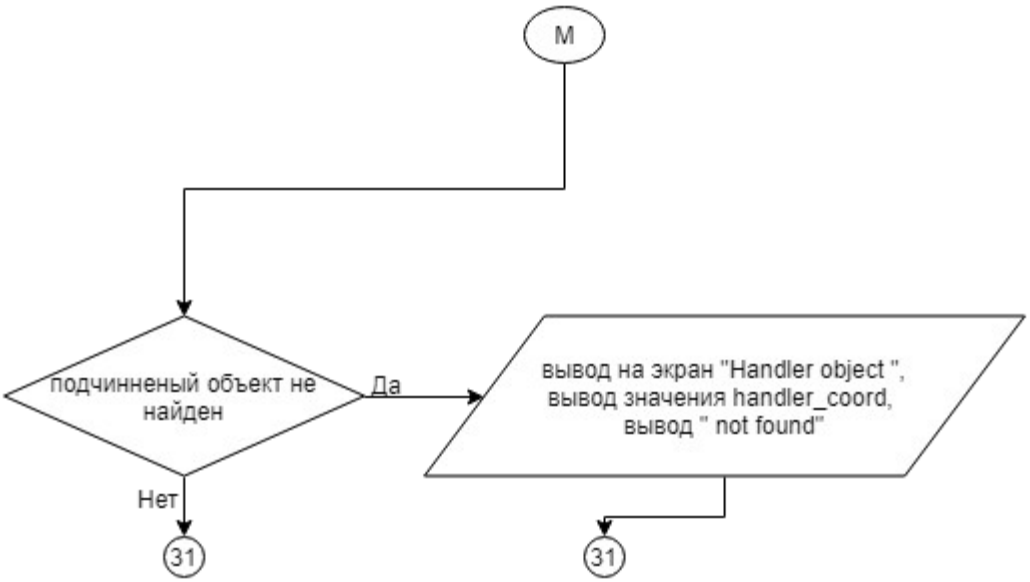

Рисунок 58 – Блок-схема алгоритма

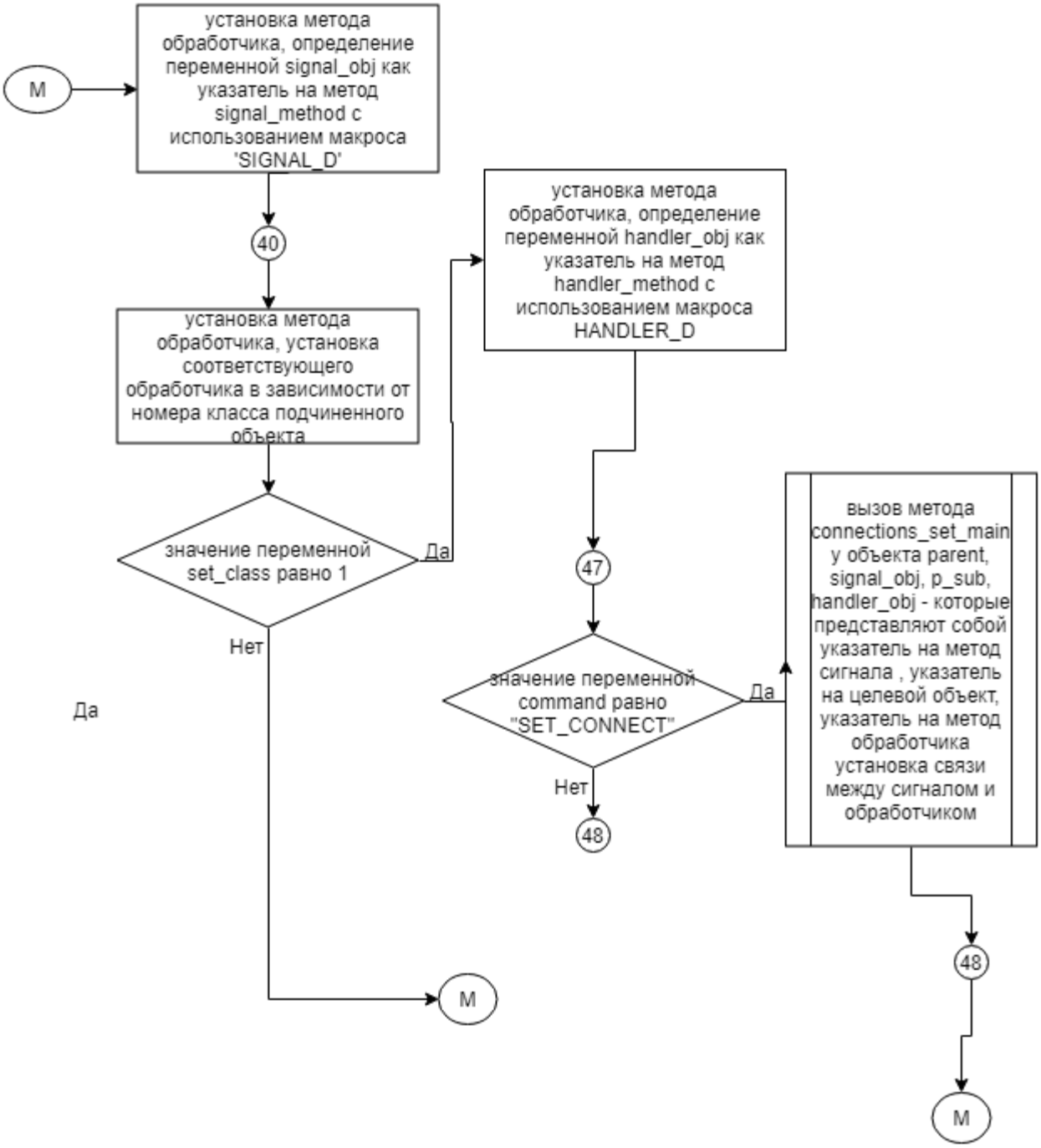

Рисунок 59 – Блок-схема алгоритма

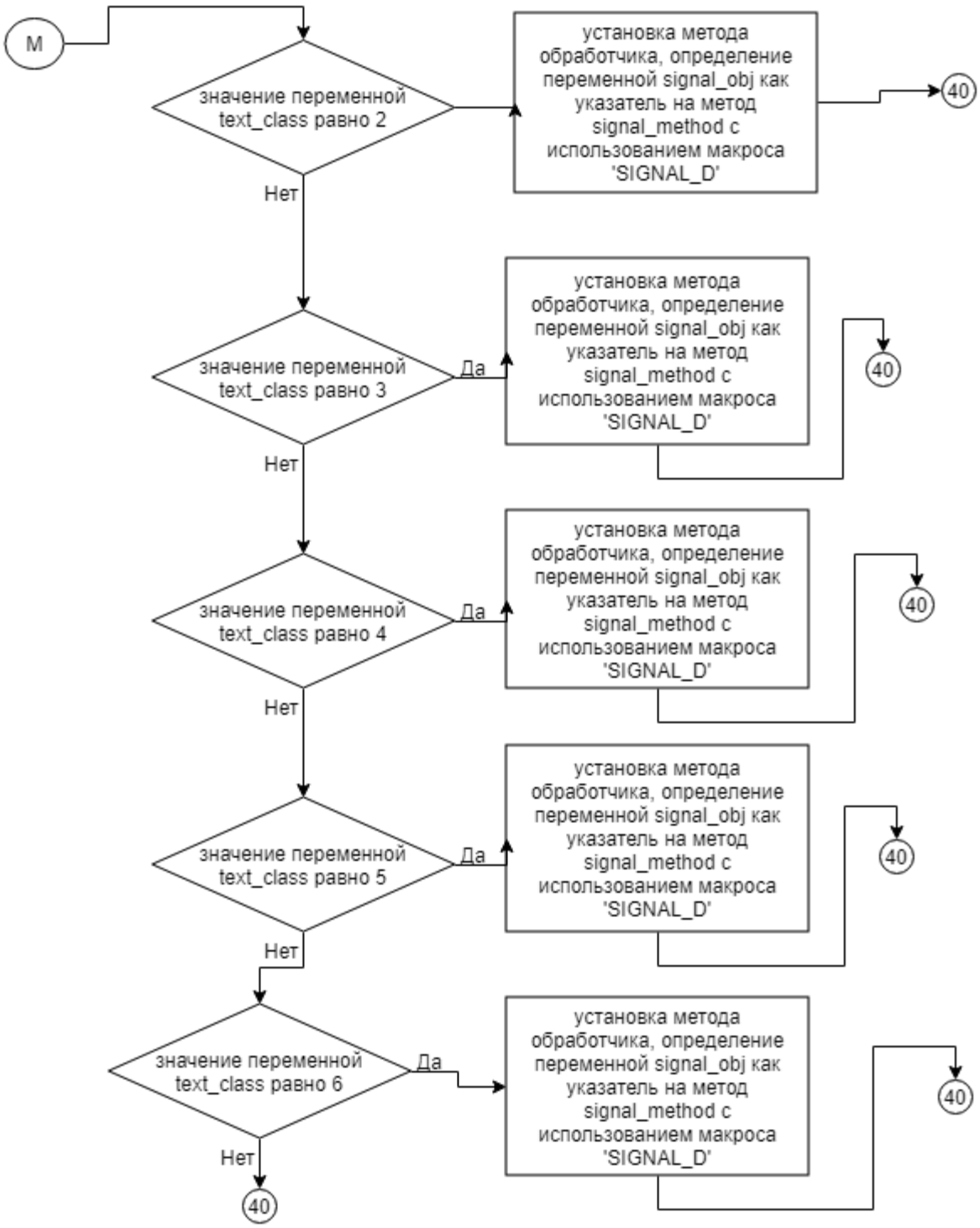

Рисунок 60 – Блок-схема алгоритма

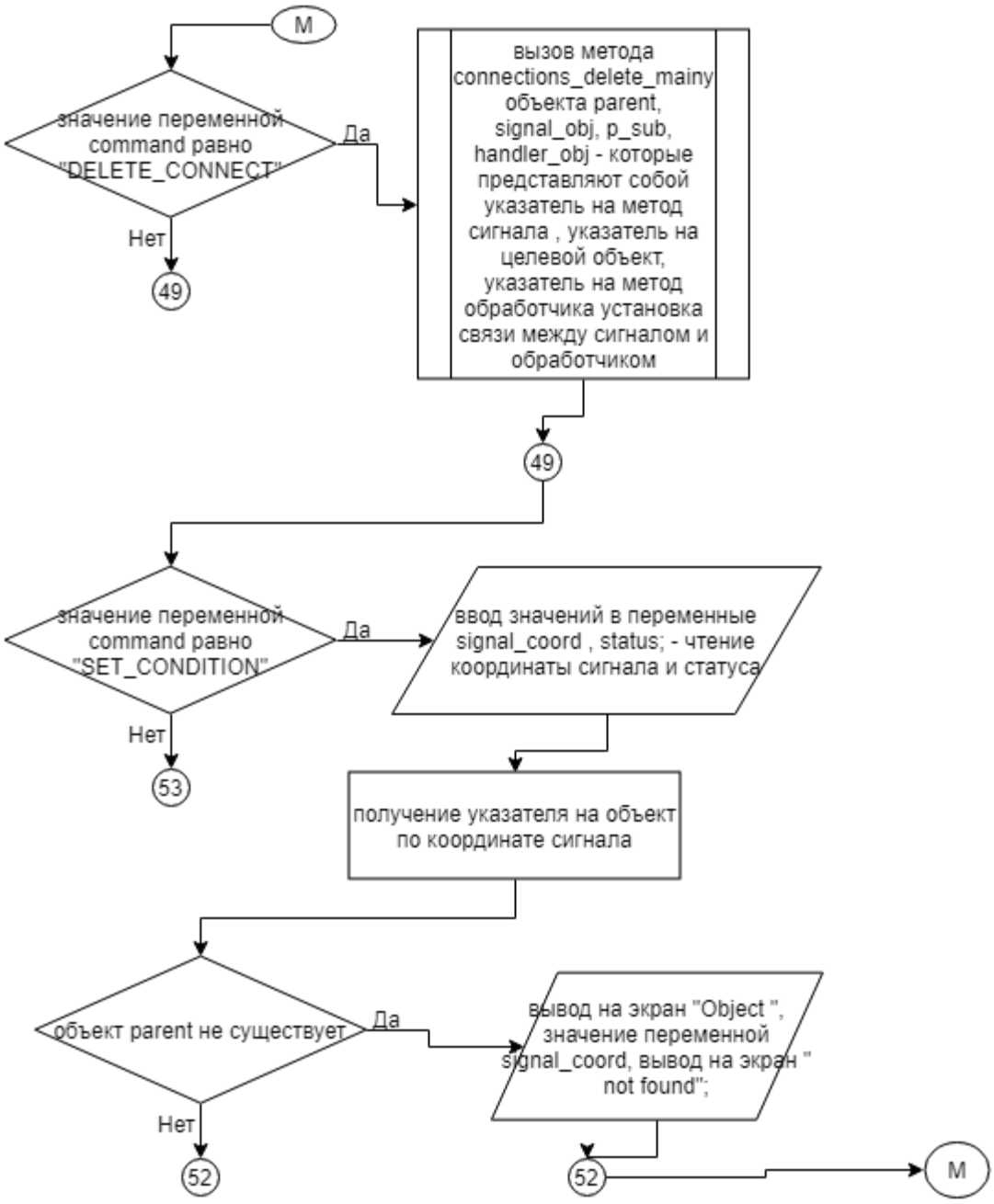

Рисунок 61 – Блок-схема алгоритма

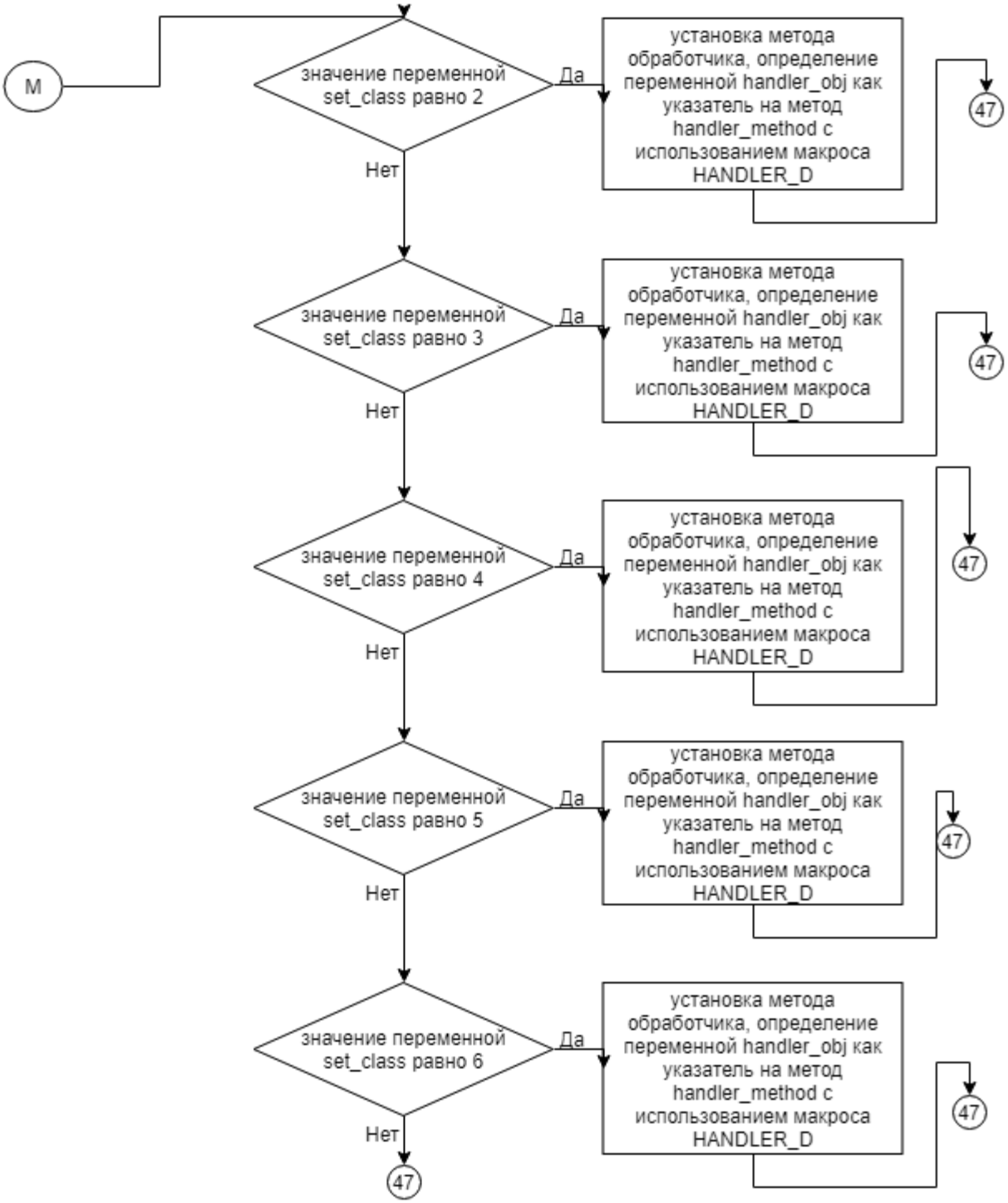

Рисунок 62 – Блок-схема алгоритма

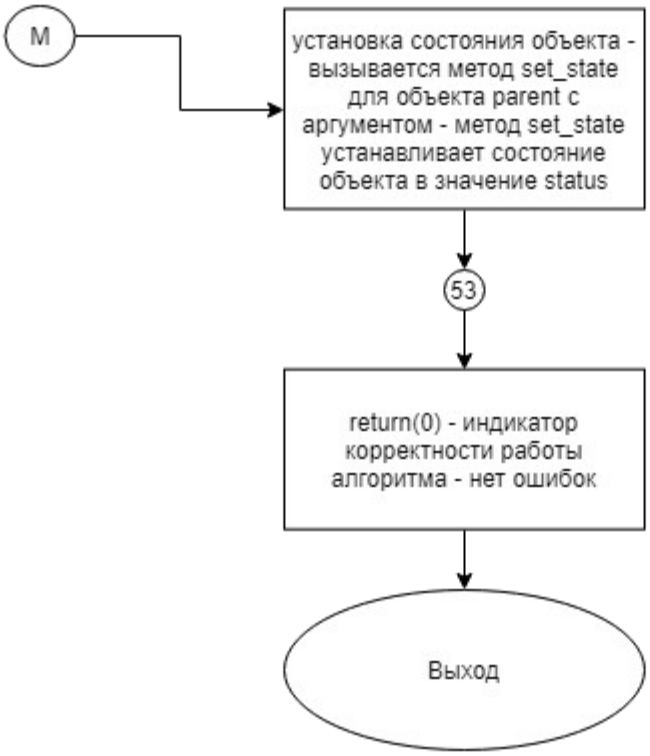

Рисунок 63 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_custom_2.cpp

Листинг 1 – cl_custom_2.cpp

```
#include "cl_custom_base.h"

#include "cl_custom_2.h"

cl_custom_2::cl_custom_2(cl_custom_base*     p_main_custom_object,     string
custom_name) : cl_custom_base(p_main_custom_object, custom_name) {}

// метод - возвращающий номер класса для cl_custom_2
int cl_custom_2::return_number_class() {
   return 2; // возврат целочисленного значение 2, которое представляет номер
класса cl_custom_2
}
```

5.2 Файл cl_custom_2.h

Листинг 2 – cl_custom_2.h

```
#ifndef __CL_CUSTOM_2__H
#define __CL_CUSTOM_2__H

#include "cl_custom_base.h"

class cl_custom_2 : public cl_custom_base {
public:
   cl_custom_2(cl_custom_base* p_main_custom_object, string custom_name);
   // конструктор - указатель на главный объект и строку с именем
   int return_number_class();
   // метод возвращающий номер класса

};

#endif
```

5.3 Файл cl_custom_3.cpp

Листинг 3 – cl_custom_3.cpp

```
#include "cl_custom_base.h"

#include "cl_custom_3.h"

cl_custom_3::cl_custom_3(cl_custom_base*     p_main_custom_object,     string
custom_name) : cl_custom_base(p_main_custom_object, custom_name) {}

// метод - возвращающий номер класса для cl_custom_3
int cl_custom_3::return_number_class() {
   return 3;
}
```

5.4 Файл cl_custom_3.h

Листинг 4 – cl_custom_3.h

```
#ifndef __CL_CUSTOM_3__H
#define __CL_CUSTOM_3__H

#include "cl_custom_base.h"

class cl_custom_3 : public cl_custom_base {
public:
   cl_custom_3(cl_custom_base* p_main_custom_object, string custom_name);
   // конструктор - указатель на главный объект и строку с именем
   int return_number_class();  // метод возвращающий номер класса

};

#endif
```

5.5 Файл cl_custom_4.cpp

Листинг 5 – cl_custom_4.cpp

```
#include "cl_custom_base.h"

#include "cl_custom_4.h"

cl_custom_4::cl_custom_4(cl_custom_base*     p_main_custom_object,     string
```

```
custom_name) : cl_custom_base(p_main_custom_object, custom_name) {}

// метод - возвращающий номер класса для cl_custom_4
int cl_custom_4::return_number_class() {
   return 4;
}
```

5.6 Файл cl_custom_4.h

Листинг 6 – cl_custom_4.h

```
#ifndef __CL_CUSTOM_4__H
#define __CL_CUSTOM_4__H

#include "cl_custom_base.h"

class cl_custom_4 : public cl_custom_base {
public:
   cl_custom_4(cl_custom_base* p_main_custom_object, string custom_name);
   // конструктор - указатель на главный объект и строку с именем
   int return_number_class();  // метод возвращающий номер класса

};

#endif
```

5.7 Файл cl_custom_5.cpp

Листинг 7 – cl_custom_5.cpp

```
#include "cl_custom_base.h"

#include "cl_custom_5.h"

cl_custom_5::cl_custom_5(cl_custom_base*     p_main_custom_object,     string
custom_name) : cl_custom_base(p_main_custom_object, custom_name) {}

// метод - возвращающий номер класса для cl_custom_5
int cl_custom_5::return_number_class() {
   return 5;
}
```

5.8 Файл cl_custom_5.h

Листинг 8 – cl_custom_5.h

```
#ifndef __CL_CUSTOM_5__H
#define __CL_CUSTOM_5__H

#include "cl_custom_base.h"

class cl_custom_5 : public cl_custom_base {
public:
   cl_custom_5(cl_custom_base* p_main_custom_object, string custom_name);
   // конструктор - указатель на главный объект и строку с именем
   int return_number_class();  // метод возвращий номер класса

};

#endif
```

5.9 Файл cl_custom_6.cpp

Листинг 9 – cl_custom_6.cpp

```
#include "cl_custom_base.h"

#include "cl_custom_6.h"

cl_custom_6::cl_custom_6(cl_custom_base*     p_main_custom_object,     string
custom_name) : cl_custom_base(p_main_custom_object, custom_name) {}

// метод - возвращающий номер класса для cl_custom_6
int cl_custom_6::return_number_class() {
   return 6;
}
```

5.10 Файл cl_custom_6.h

Листинг 10 – cl_custom_6.h

```
#ifndef __CL_CUSTOM_6__H
#define __CL_CUSTOM_6__H

#include "cl_custom_base.h"
```

```
class cl_custom_6 : public cl_custom_base {
public:
   cl_custom_6(cl_custom_base* p_main_custom_object, string custom_name);
   // конструктор - указатель на главный объект и строку с именем
   int return_number_class();  // метод возвращий номер класса

};

#endif
```

5.11 Файл cl_custom_application.cpp

Листинг 11 – cl_custom_application.cpp

```
#include "cl_custom_application.h"

// конструктор класса cl_custom_application
cl_custom_application::cl_custom_application(cl_custom_base*
p_main_custom_object) : cl_custom_base(p_main_custom_object) {}

// метод создания дерева иерархии
// path_custom - строка - содержащая путь к текущему объекту
// sub_name - строка - содержащая имя подчиненного объекта
// state - указатель на текущий объект - используется для построения дерева
// main_state - указатель на главный объект - используется для создания
новых
// подчиненных объектов
// command - целочисленное значение - определяющее тип создаваемого
подчиненного объекта
// text_class - целочисленное значение - определяющее класс объекта из
которого исходит сигнал
// set_class - целочисленное значение - определяющее класс объекта
// signal_obj - указатель на метод сигнала - используемый для установки
связи
// handler_obj - указатель на метод обработчика - используемый для установки
связи
// parent - указатель на родительский объект в иерархии
// p_sub - указатель на подчиненный объект в иерархии
void cl_custom_application::build_tree_objects() {
   // объявление переменных для хранения пользовательского ввода и текущего
состояния объекта
   string path_custom, sub_name; // переменные для хранения пути и иемни
подчиненного объекта
   cl_custom_base* state; // указатель на текущий объект
   cl_custom_base*  main_state  =  nullptr;  //  указатель  на  создаваемый
подчиненный объект,
   // инициализируется
   int command; // переменная для хранения команды
```

```
   cin >> path_custom; // чтение имени главного объета
   set_custom_name(path_custom);// установка имени главного объекта

   state = this;// текущий объект - это сам объект приложения !!!!!!!!!!!!!
11

   while(true) { // цикл для построения дерева
      cin >> path_custom; // чтение значения
      if(path_custom == "endtree") {
         break;
      }

      cin >> sub_name >> command; // чтетие имени подчиненного объекта и
команды

      state = get_object_path(path_custom); // получение указателя на объект
по пути

      if(!(state)) { // если объект не найден
         cout << "Object tree" << endl;
         indentation_method();
         cout << endl << "The head object " << path_custom << " is not
found";
         exit(1);
      }
      if(state->get_sub_customs(sub_name)) { // проверка на дублирование имен
подчиненных объектов
         // вызов get_sub_customs для объект statae с передачей в качестве
аргумента sub_name
         //подобъект с именем указанным в аргументе sub_name существует
         cout << path_custom << " Dubbing the names of subordinate objects"
<< endl;
         continue;
      }

      switch(command) { // создание нового подчиненного объекта в зависимости
от команды
         case 2:
            main_state = new cl_custom_2(state, sub_name);// создание объекта
класса cl_custom_2
            //объект создается с указанным state в качестве главного объекта
и sub_name в
            //качестве имени объекта
            break;
         case 3:
            main_state = new cl_custom_3(state, sub_name);// создание объекта
класса cl_custom_3
            //объект создается с указанным state в качестве главного объекта
и sub_name в
            //качестве имени объекта
            break;
         case 4:
            main_state = new cl_custom_4(state, sub_name);// создание объекта
класса cl_custom_4
            //объект создается с указанным state в качестве главного объекта
```

```
и sub_name в
            //качестве имени объекта
            break;
         case 5:
            main_state = new cl_custom_5(state, sub_name);// создание объекта
класса cl_custom_5
            //объект создается с указанным state в качестве главного объекта
и sub_name в
            //качестве имени объекта
            break;
         case 6:
            main_state = new cl_custom_6(state, sub_name);// создание объекта
класса cl_custom_6
            //объект создается с указанным state в качестве главного объекта
и sub_name в
            //качестве имени объекта
            break;
      }
   }

   // вывод дерева объектов
   cout << "Object tree" << endl;
   indentation_method(); //вызов метода для вывода отступов

   int text_class, set_class;

   TYPE_SIGNAL signal_obj; // указатель на метод сигнала
   TYPE_HANDLER handler_obj; // указатель на метод обработчика

   cl_custom_base* parent; // указатель на родительский объект
   cl_custom_base* p_sub; // указатель на подчиненный объект

   while(true) { // установка связей между объектами
      cin >> path_custom;

      if(path_custom == "end_of_connections") {
         break;
      }

      cin >> sub_name; // чтение имени подчиненного объекта

      parent  =  get_object_path(path_custom);  //  получение  указателя  на
родительский объект
      p_sub   =   get_object_path(sub_name);   //   получение   указателя   на
подчиненный объект

      if((!parent) || (!p_sub) || (parent == p_sub)) {
         //  если  родительский  или  подчиненный  объект  не  найден  или  они
одинаковы
         if(!parent) {// если родительский объект не найден
            cout  <<  endl  <<  "Object  "  <<  path_custom  <<  "  not  found";  //
вывод сообщения об ошибке
         } else if(!p_sub) { // если подчиненный объект не найден
            cout << endl << "Handler object " << sub_name << " not found"; //
вывод сообщения об ошибке
```

```
            }
            continue;
         }

         text_class = parent->return_number_class(); // получение номера класса
родительского объекта
         set_class = p_sub->return_number_class(); // получение номера класс
подчиненного объекта

         switch(text_class) { // определение метода сигнала в зависимости от
номера класса родительского объекта
            case 1:
               signal_obj = SIGNAL_D(cl_custom_base::signal_method);//установка
указаетля на метод сигнала
               //класса cl_custom_base
               //   инициализация   переменной   signal_obj=   которая   является
указателем на метод сигнала ,
               // используя макрос SIGNAL_D. макрос принимает указатель на метод
и приводит его к типу
               // TYPE_SIGNAL
               break;
            case 2:
               signal_obj   =   SIGNAL_D(cl_custom_2::signal_method);//установка
указаетля на метод сигнала
               //класса cl_custom_2
               //   инициализация   переменной   signal_obj=   которая   является
указателем на метод сигнала ,
               // используя макрос SIGNAL_D. макрос принимает указатель на метод
и приводит его к типу
               // TYPE_SIGNAL
               break;
            case 3:
               signal_obj   =   SIGNAL_D(cl_custom_3::signal_method);//установка
указаетля на метод сигнала
               //класса cl_custom_3
               //   инициализация   переменной   signal_obj=   которая   является
указателем на метод сигнала ,
               // используя макрос SIGNAL_D. макрос принимает указатель на метод
и приводит его к типу
               // TYPE_SIGNAL
               break;
            case 4:
               signal_obj   =   SIGNAL_D(cl_custom_4::signal_method);//установка
указаетля на метод сигнала
               //класса cl_custom_4
               //   инициализация   переменной   signal_obj=   которая   является
указателем на метод сигнала ,
               // используя макрос SIGNAL_D. макрос принимает указатель на метод
и приводит его к типу
               // TYPE_SIGNAL
               break;
            case 5:
               signal_obj   =   SIGNAL_D(cl_custom_5::signal_method);//установка
указаетля на метод сигнала
               //класса cl_custom_5
               //   инициализация   переменной   signal_obj=   которая   является
```

```
указателем на метод сигнала ,
            // используя макрос SIGNAL_D. макрос принимает указатель на метод
и приводит его к типу
            // TYPE_SIGNAL
            break;
         case 6:
            signal_obj   =   SIGNAL_D(cl_custom_6::signal_method);//установка
указаетля на метод сигнала
            //класса cl_custom_6
            //   инициализация   переменной   signal_obj=   которая   является
указателем на метод сигнала ,
            // используя макрос SIGNAL_D. макрос принимает указатель на метод
и приводит его к типу
            // TYPE_SIGNAL
            break;
      }
      switch(set_class) {//определение метода обработчика в зависимости от
номера класса подчиненного объекта
         case 1:
            handler_obj                                                    =
HANDLER_D(cl_custom_base::handler_method);//установка указателя на метод
            //обработчика класса cl_custom_base
            //  инициализация  переменной  handler_obj  =  которая  является
указателем на метод сигнала ,
            //  используя  макрос  HANDLER_D.  макрос  принимает  указатель  на
метод и приводит его к типу
            // TYPE_HANDLER
            break;
         case 2:
            handler_obj  =  HANDLER_D(cl_custom_2::handler_method);//установка
указателя на метод
            //обработчика класса cl_custom_2
            //  инициализация  переменной  handler_obj  =  которая  является
указателем на метод сигнала ,
            //  используя  макрос  HANDLER_D.  макрос  принимает  указатель  на
метод и приводит его к типу
            // TYPE_HANDLER
            break;
         case 3:
            handler_obj  =  HANDLER_D(cl_custom_3::handler_method);//установка
указателя на метод
            //обработчика класса cl_custom_3
            //  инициализация  переменной  handler_obj  =  которая  является
указателем на метод сигнала ,
            //  используя  макрос  HANDLER_D.  макрос  принимает  указатель  на
метод и приводит его к типу
            // TYPE_HANDLER
            break;
         case 4:
            handler_obj  =  HANDLER_D(cl_custom_4::handler_method);//установка
указателя на метод
            //обработчика класса cl_custom_4
            //  инициализация  переменной  handler_obj  =  которая  является
указателем на метод сигнала ,
            //  используя  макрос  HANDLER_D.  макрос  принимает  указатель  на
```

```
метод и приводит его к типу
            // TYPE_HANDLER
            break;
         case 5:
            handler_obj  =  HANDLER_D(cl_custom_5::handler_method);//установка
указателя на метод
            //обработчика класса cl_custom_5
            //  инициализация  переменной  handler_obj  =  которая  является
указателем на метод сигнала ,
            //  используя  макрос  HANDLER_D.  макрос  принимает  указатель  на
метод и приводит его к типу
            // TYPE_HANDLER
            break;
         case 6:
            handler_obj  =  HANDLER_D(cl_custom_6::handler_method);//установка
указателя на метод
            //обработчика класса cl_custom_6
            //  инициализация  переменной  handler_obj  =  которая  является
указателем на метод сигнала ,
            //  используя  макрос  HANDLER_D.  макрос  принимает  указатель  на
метод и приводит его к типу
            // TYPE_HANDLER
            break;
      }
      parent->connections_set_main(signal_obj, p_sub, handler_obj);
      // вызывается метод connections_set_main объекта parent для установки
связи между сигналом,
      // обработчкиком и целевым объектом
      // установка связи между сигналом родительского объекта и обработчиком
подчиненного объекта
   }
}

// метод запуска системы
// signal_coord -
// handler_coord -
// status -  хранит статус объекта
// parent- указатель на родительский объект
// p_sub-указатель на подчиненный объект
// signal_obj- хранит указатель на метод сигнала
// handler_obj- хранит указатель на метод обработчика
// command- хранит команду - полученную из ввода
// text- хранит текст - полученный из ввода
// text_class- хранит номер класса объекта, полученный из ввода для методов
связаанных с сигналами
// set_class - хранит номер класса объекта, полученный из ввода для методов
связанных с обработчиками
int cl_custom_application::exec_custom_app() {
   set_object_ready();

   string signal_coord, handler_coord; // переменные для хранения координат
сигнала и обработчика
   int status; // переменная для хранения статуса

   cl_custom_base* parent; // указатель на родительский объект
```

```
   cl_custom_base* p_sub; // указатель на подчиненный объект

   TYPE_SIGNAL signal_obj; // переменная для хранения указателя на метод
сигнала
   TYPE_HANDLER handler_obj; // переменная для хранения указателя на метод
обработчика

   string command, text; // переменные для хранения команды и текста
   int text_class, set_class; // переменные для хранения номеров классов
объектов

   while(true) {
      cin >> command; // чтение команды

      if(command == "END") {
         break;
      }

      if(command == "EMIT") {
         cin >> signal_coord;//чтение координаты сигнала
         getline(cin, text);//чтение текста сигнала

         parent = get_object_path(signal_coord);//получение указателя на
объект по координате сигнала

         if(!parent) { //был ли найден объект parent по указанному пути
signal_coord
            cout << "Object " << signal_coord << " not found";
            continue;
         }

         text_class = parent->return_number_class();// получение номера
класса объета
         // вызывается метод return_number_class для ообъект parent - который
возвращает
         // номер класса объекта parent - полученный номер сохраняется в
переменной text_Class

         switch(text_class) {//установка соответсвующего сигнала в
зависимости от номера класса
            case 1:
               signal_obj                                                =
SIGNAL_D(cl_custom_base::signal_method);//установка метода сигнала для
класса
               //cl_custom_base
               // используется макрос SIGNAL_D - чтобы преобразовать
указатель на метод signal_method
               // в соответствующий тип TYPE_SIGNAL , который используется
для установки сигнала
               // в методе connections_set_main()
               break;
            case 2:
               signal_obj = SIGNAL_D(cl_custom_2::signal_method);//установка
метода сигнала для класса
               //cl_custom_2
               // используется макрос SIGNAL_D - чтобы преобразовать
```

```
указатель на метод signal_method
               // в соответствующий тип TYPE_SIGNAL , который используется
для установки сигнала
               // в методе connections_set_main()
               break;
            case 3:
               signal_obj  =  SIGNAL_D(cl_custom_3::signal_method);//установка
метода сигнала для класса
               //cl_custom_3
               //  используется  макрос  SIGNAL_D  -  чтобы  преобразовать
указатель на метод signal_method
               // в соответствующий тип TYPE_SIGNAL , который используется
для установки сигнала
               // в методе connections_set_main()
               break;
            case 4:
               signal_obj  =  SIGNAL_D(cl_custom_4::signal_method);//установка
метода сигнала для класса
               //cl_custom_4
               //  используется  макрос  SIGNAL_D  -  чтобы  преобразовать
указатель на метод signal_method
               // в соответствующий тип TYPE_SIGNAL , который используется
для установки сигнала
               // в методе connections_set_main()
               break;
            case 5:
               signal_obj  =  SIGNAL_D(cl_custom_5::signal_method);//установка
метода сигнала для класса
               //cl_custom_5
               //  используется  макрос  SIGNAL_D  -  чтобы  преобразовать
указатель на метод signal_method
               // в соответствующий тип TYPE_SIGNAL , который используется
для установки сигнала
               // в методе connections_set_main()
               break;
            case 6:
               signal_obj  =  SIGNAL_D(cl_custom_6::signal_method);//установка
метода сигнала для класса
               //cl_custom_6
               //  используется  макрос  SIGNAL_D  -  чтобы  преобразовать
указатель на метод signal_method
               // в соответствующий тип TYPE_SIGNAL , который используется
для установки сигнала
               // в методе connections_set_main()
               break;
         }
         parent->connections_main_signal(signal_obj, text);//отправка сигнала
объекту
         //  вызывается  метод  connections_main_signal  у  объекта  parent  -
передая ему signal_obj и текст text
         // для обработки
      }

      if(command == "SET_CONNECT" || command == "DELETE_CONNECT") {
         cin >> signal_coord >> handler_coord; // чтение координат сигнала и
```

```
обработчика

        parent  =  get_object_path(signal_coord);//получение  указателя  на
объект по координате сигнала
        p_sub  =  get_object_path(handler_coord);//получение  указателя  на
объект по координате обработчика

        if((!parent) || (!p_sub) || (parent == p_sub)) {//если объекты не
найдены или совпадают
           if(!parent) {//если родительский объект не найден
              cout << "Object " << signal_coord << " not found";
           } else if (!p_sub) {//если подчинненый объект не найден
              cout << "Handler object " << handler_coord << " not found";
           }
           continue;
        }

        text_class  =  parent->return_number_class();//  получение  номера
класса родительского объекта
        set_class = p_sub->return_number_class();// получение номера класса
подчиненного объекта

        switch(text_class)  {  //  установка  соответствующего  сигнала  в
зависимости от номера класса родительского объекта
           case 1:
              signal_obj                                                  =
SIGNAL_D(cl_custom_base::signal_method);//установка   метода   сигнала   для
класса
              //cl_custom_base
              //  определение  переменной  signal_obj  как  указатель  на  метод
signal_method с использованием
              // макроса 'SIGNAL_D'
              break;
           case 2:
              signal_obj  =  SIGNAL_D(cl_custom_2::signal_method);//установка
метода сигнала для класса
              //cl_custom_2
              //  определение  переменной  signal_obj  как  указатель  на  метод
signal_method с использованием
              // макроса 'SIGNAL_D'
              break;
           case 3:
              signal_obj  =  SIGNAL_D(cl_custom_3::signal_method);//установка
метода сигнала для класса
              //cl_custom_3
              //  определение  переменной  signal_obj  как  указатель  на  метод
signal_method с использованием
              // макроса 'SIGNAL_D'
              break;
           case 4:
              signal_obj  =  SIGNAL_D(cl_custom_4::signal_method);//установка
метода сигнала для класса
              //cl_custom_4
              //  определение  переменной  signal_obj  как  указатель  на  метод
signal_method с использованием
```

```
                // макроса 'SIGNAL_D'
                break;
            case 5:
                signal_obj  =  SIGNAL_D(cl_custom_5::signal_method);//установка
метода сигнала для класса
                //cl_custom_5
                //  определение  переменной  signal_obj  как  указатель  на  метод
signal_method с использованием
                // макроса 'SIGNAL_D'
                break;
            case 6:
                signal_obj  =  SIGNAL_D(cl_custom_6::signal_method);//установка
метода сигнала для класса
                //cl_custom_6
                //  определение  переменной  signal_obj  как  указатель  на  метод
signal_method с использованием
                // макроса 'SIGNAL_D'
                break;
        }  switch(set_class)  {  //установка  соответствующего  обработчика  в
зависимости от номера класса подчиненного
                // объекта
            case 1:
                handler_obj                                                    =
HANDLER_D(cl_custom_base::handler_method);//установка метода обработчика для
                //класса cl_custom_base
                //  определение  переменной  handler_obj  как  указатель  на  метод
handler_method с использованием
                // макроса HANDLER_D
                break;
            case 2:
                handler_obj                                                    =
HANDLER_D(cl_custom_2::handler_method);//установка метода обработчика для
                //класса cl_custom_2
                //  определение  переменной  handler_obj  как  указатель  на  метод
handler_method с использованием
                // макроса HANDLER_D
                break;
            case 3:
                handler_obj                                                    =
HANDLER_D(cl_custom_3::handler_method);//установка метода обработчика для
                //класса cl_custom_3
                //  определение  переменной  handler_obj  как  указатель  на  метод
handler_method с использованием
                // макроса HANDLER_D
                break;
            case 4:
                handler_obj                                                    =
HANDLER_D(cl_custom_4::handler_method);//установка метода обработчика для
                //класса cl_custom_4
                //  определение  переменной  handler_obj  как  указатель  на  метод
handler_method с использованием
                // макроса HANDLER_D
                break;
            case 5:
                handler_obj                                                    =
```

```
      HANDLER_D(cl_custom_5::handler_method);//установка метода обработчика для
                  //класса cl_custom_5
                  // определение переменной handler_obj как указатель на метод
      handler_method с использованием
                  // макроса HANDLER_D
                  break;
               case 6:
                  handler_obj                                                    =
      HANDLER_D(cl_custom_6::handler_method);//установка метода обработчика для
                  //класса cl_custom_6
                  // определение переменной handler_obj как указатель на метод
      handler_method с использованием
                  // макроса HANDLER_D
                  break;
            }
            if(command == "SET_CONNECT") {
               parent->connections_set_main(signal_obj, p_sub, handler_obj);
               //вызов метода connections_set_main у объекта parent
               //signal_obj,  p_sub,  handler_obj  -  которые  представляют  собой
      указатель на метод сигнала,
               //указатель на целевой объект, указатель на метод обработчика
               //установка связи между сигналом и обработчиком
            }
            if(command == "DELETE_CONNECT") {
               parent->connections_delete_main(signal_obj, p_sub, handler_obj);
               //вызов метода connections_delete_main у объекта parent, передачи
      аргументов
               //signal_obj,  p_sub,  handler_obj  -  которые  представляют  собой
      указатель на метод сигнала,
               //указатель на целевой объект, указатель на метод обработчика
               //удаление связи между сигналом и обработчиком
            }
         }

         if(command == "SET_CONDITION") {
            cin >> signal_coord >> status; //чтение координаты сигнала и статуса

            parent  =  get_object_path(signal_coord);//получение  указателя  на
      объект по координате сигнала


            if(!parent) {//существует ли объект parent
               cout << "Object " << signal_coord << " not found";
               continue;
            }

            parent->set_state(status);//установка состояния объекта
            // вызывается метод set_state для объекта parent с аргументом status
      -
            // метод set_state устанавливает состояние объекта в значение status
         }
      }

      return(0);
   }
```

5.12 Файл cl_custom_application.h

Листинг 12 – cl_custom_application.h

```
#ifndef __CL_CUSTOM_APPLICATION__H
#define __CL_CUSTOM_APPLICATION__H

#include "cl_custom_base.h"
#include "cl_custom_2.h"
#include "cl_custom_3.h"
#include "cl_custom_4.h"
#include "cl_custom_5.h"
#include "cl_custom_6.h"

class cl_custom_application : public cl_custom_base {
public:
   cl_custom_application(cl_custom_base* p_main_custom_object);
   // конструктор класса cl_custom_application
   // p_main_custom_object - указатель на главный объект
   void build_tree_objects();
   // метод для построения дерева объектов
   int exec_custom_app();
   // метод для выполнения пользовательского приложения
   // возврат кода завершения
};

#endif
```

5.13 Файл cl_custom_base.cpp

Листинг 13 – cl_custom_base.cpp

```
#include "cl_custom_base.h"

cl_custom_base::cl_custom_base(cl_custom_base*  p_main_custom_object,  string
custom_name) {
   // конструктор класса cl_custom_base с двумя параметрами:
   // указатель на главный объект пользовательского класса и строковое имя
объекта
   this->p_main_custom_object = p_main_custom_object;
   // присвоение указателю this->p_main_custom_object переданонго указателя
p_main_custom_object
   this->custom_name = custom_name;
   // присвоение строковой переменной this->custom_name переданной строки
custom_name
   if(this->p_main_custom_object) { //проверка что главный объект существует
      p_main_custom_object->p_sub_customs.push_back(this);       //добавление
указателя на текущий объект в список
```

```
      // подообъектов главного объекта
   }
}

// деструктор класс cl_custom_base
cl_custom_base::~cl_custom_base() {
   //освобождение памяти выделенной для каждого подообъекта
   for(int i = 0; i < this->p_sub_customs.size(); i++) { // цикл по всем
элементам списка подообъектов
      delete p_sub_customs[i]; // освобождение памяти для текущего подобъекта
   }
}

// метод установки нового имени для объекта
// new_custom_name - новое имя , которое нужно установить объекту
// p_sub_customs - указатель на подобъект главного объекта, используется для
проверки совпадения имени
bool cl_custom_base::set_custom_name(string new_custom_name) {
   // проверка, что у объекта есть главный объект
   if(this->p_main_custom_object) {
      // проверка - что новое имя не совпадаает с именем другого подобъекта
главного объекта

      for(int i = 0; i < p_main_custom_object->p_sub_customs.size(); i++) {
         // цикл по всем подобъектам главного объекта, проход по всем
подобъектам главного объекта
         if(p_main_custom_object->p_sub_customs[i]->get_custom_name()      ==
new_custom_name){
            // проверка совпадения имен
            return false; // возврат false , если имя уже используется
         }
      }
   }

   // установка нового имени объекту
   custom_name = new_custom_name; // присвоение нового имени объекту
   return true; // возврат true , если имя успешно установлено
}

// метод получения подобъекта по его имени
// name_object - имя подобъекта, который нужно найти
//  p_sub_customs  -  указатель  на  текущий  подобъект  текущего  объекта,
используется для сравнения с искомым именем
cl_custom_base* cl_custom_base::get_sub_customs(string name_object) {

   // проход по всем подообъектам текущего объекта
   for(int i = 0; i < p_sub_customs.size(); i++) {

      // проверка совпадений имен
      if(p_sub_customs[i]->get_custom_name() == name_object) { //проверка -
совпадает
         // ли имя подчиненного объекта с заданным именем
         return p_sub_customs[i]; // возврат указателя на найденный подобъект
      }
   }
   return nullptr; // если подобъект с заданным именем не найден - возврат
```

```
nullptr
}

// метод получения главного объекта
cl_custom_base* cl_custom_base::get_main_custom() {
   return p_main_custom_object; // возврат указателя на главный объект
}

// метод вывода иерархии объектов с отступами
// p_sub_customs - указатель на текущий подобъект текущего объекта
//  i_space -  количество  отступов,  используется  для  определения  уровня
вложенности объекта
//  current  -  указатель  на  текущий  объект,  используется  для  подсчета
количества отступов
void cl_custom_base::indentation_method() {
   if(get_main_custom()) { // проверка наличия главного объекта
      cout << endl; // выводим пустую строку для форматирования
   }

   int  i_space  =  0;  //  инициализация  переменной  для  хранения  количества
отступов
   cl_custom_base*  current  =  this;  //  инициализация  указателя  на  текущий
объект
   while(current->get_main_custom())  {  //  цикл  для  подсчета  количества
отступов
      current = current->get_main_custom(); // переходим к главному объекту
      i_space++; // увеличение счетчиков отступов
   }

   for(int i = 0; i < i_space; i++) { // цикл для вывода отступов
      cout << "    "; // выводим отступ
   }
   cout << get_custom_name(); // выводим имя текущего объекта

   for(auto cl_custom_base_p : p_sub_customs) { // цикл по всем подообъектам
текущего объекта
      cl_custom_base_p->indentation_method(); // рекурсивный вызов метода для
каждого подообъекта
   }
}

// метод получения имени объекта
string cl_custom_base::get_custom_name() {
   return custom_name; // возврат строки с именем объекта
}

// метод поиска текущего объекта по имени в ширину
// q - очередь для обхода дерева объектов
// path_id - указатель на найденный объект
//  s_custom_name  -хранение  имени  объекта  -  который  нужно  найти  найти  в
методах поиска
cl_custom_base* cl_custom_base::search_current(string s_custom_name) {
   queue<cl_custom_base*> q; // очередь для поиска в ширину
   cl_custom_base* path_ind = nullptr; // указатель на найденный объект
```

```
        q.push(this); // добавление текущего объекта в очередь

        while(!q.empty()) {  // пока очередь не пуста
           // если имя текущего объекта совпадает с искомым именем
           if(q.front()->get_custom_name() == s_custom_name) {

              if(!path_ind) {  //  если путь еще не найден
                 path_ind = q.front();  // установление указатель на текущий
                 // объект как найденный путь
              } else {
                 return nullptr; // если путь уже найден - возврат nullptr
                 //(несколько объектов с одинаковым именем)
              }
           }

           // добавление всех подобъектов текущего объекта в очередь
           for(auto cl_custom_base_p : q.front()->p_sub_customs) { // проходит по
     всем подчиненным объектам
              // текущего объекта - находящегося в начале очереди q
              q.push(cl_custom_base_p);
              // добавление каждого подчиненного объекта в конец очереди q
           }

           q.pop(); // удаление текущего объекта из очереди
        }
        return path_ind; // возврат указателя на найденный объект
     }

     // метод получения объекта по указанному пути
     // text - текст в пути
     // command_ver - индекс разделителя в пути
     // base - указатель на объект
     cl_custom_base* cl_custom_base::get_object_path(string path_custom) {
        string text; // вспомогательная переменная для хранения подстроки пути
        int command_ver; //  переменная для хранения индекса разделителя в пути
        cl_custom_base* base = nullptr; // указатель на объект

        // если путь пустой - возврат nullptr
        if(path_custom.empty()) {
           return nullptr;
        }

        // если путь содержит только корневой объект - возврат указателя на
     последний объект
        // в дереве
        if(path_custom == "/") {
           cl_custom_base*  ptr  =  this;  //  создание  указателя  ptr,  который
     инициализируется текущим объектом this
           while(ptr->get_main_custom()) { // пока метод get_main_custom указывает
     на родительский объект
              ptr = ptr->get_main_custom(); // обновляется ptr, чтобы указывать на
     родительский объект текущего
              // объекта ptr = ptr->get_main_custom()
           }
```

```
      return ptr;
   }

   if(path_custom == ".") { // если путь содержит текущий объект, возвращаем
указатель
      // на него
      return this;
   }

   if(path_custom[0] == '/' && path_custom[1] == '/') {
      // если путь начинается с двух слэшей - начинаем поиск с корня
      text = path_custom.substr(2); // создание подстроки начиная с третьего
символа строки path_custom
      return  this->search_tree(text);  //  выполняется  поиск  объекта  с
значением  строки path_custom
   }

   if(path_custom[0] == '.') { // если путь начинается с точки - начинаем
поиск
      // с текущего объекта
      text = path_custom.substr(1); // создание подстроки начиная со второго
символа
      return  this->search_current(text);  //  выполнение  поиска  объекта  с
значением строки text
      // начиная с текущего объекта
   }

   command_ver = path_custom.find("/", 1);
   // находим индекс разделителя в пути

   if(path_custom[0] == '/') { // если путь абсолютный
      if(command_ver != -1) { // если в пути есть разделитель
         text = path_custom.substr(1, command_ver - 1);
         // получение имени текущего уровня вложенности

         // находим объект по имени и продолжаем поиск для остальной части
пути
         base = this->search_tree(text);

         // здесь остановился
         if(base) {
            return   base->get_object_path(path_custom.substr(command_ver   +
1)); //
            // создание подстроки path_custom, начиная с позиции command_ver
+ 1
            //вызов  метода  get_object_path  на  объекте  base  с  созданной
подстрокой в качестве аргумента return
         } else {
            return base;
         }

      } else { // если разделитель не найден - продолжаем поиск с корня для
всего оставшегося пути
         text = path_custom.substr(1);
         return this->search_tree(text);
```

```
      }
   } else { // если путь относительный
      if(command_ver != -1) { // если в пути есть разделитель
         text  =  path_custom.substr(0,  command_ver);  //  получение  имени
текущего уровня вложенности

         // нахождение объекта по имени и продолжение поиска для остальной
части пути
         base = this->get_sub_customs(text);

         if(base) {
            return   base->get_object_path(path_custom.substr(command_ver   +
1));
            //создание подстроки path_custom, начиная с позиции command_ver +
1
            //base->get_object_path - вызов метода get_object_path на объекте
base с
            //созданной подстрокой в качестве аргумента
            //return - возврат результата вызова метода get_object_path
         } else {
            return base;
         }
      } else { // если разделитель не найден - продолжаем поиск в текущем
уровне для всего
         // оставшегося пути
         text  =  path_custom;  //  присвоение  переменной  text  значение
path_custom
         return  this->get_sub_customs(text);  //  возврат  результата  вызова
метода get_sub_customs с аргументом text
      }
   }
};


// метод поиска объекта в дереве по имени
// ptr - указатель на текущий объект для поиска в дереве
cl_custom_base* cl_custom_base::search_tree(string s_custom_name) { //
   cl_custom_base* ptr = this; // инициализация указателя на текущий объект
   while(ptr->get_main_custom()) {// пока у текущего объекта есть главный
объект
      ptr = ptr->get_main_custom();// перемещаем указатель на главный объект
   }
   return    ptr->get_sub_customs(s_custom_name);//возвращаем    подчинненный
объект с именем s
   //s_custom_ame у корневого объекта
}

// метод удаления подобъект по имени
// current - указатель на подобъет по его имени
// sub_name - хранение имени подчиненного объекта - создание связи между
объектами в дереве объектов
void cl_custom_base::delete_custom_sub_name(string sub_name) {
   cl_custom_base*   current   =   get_sub_customs(sub_name);   //   получение
указателя на подобъект по его имени
```

```
   if(current) { // если подобъект найден
      for(int i = 0; i < p_sub_customs.size(); i++) { // проходим по всем
подобъектам текущего объекта
         if(p_sub_customs[i] == current) { // если текущий подобъект равен
искомому подобъекту
            p_sub_customs.erase(p_sub_customs.begin() + i); // удаление его
из списка подобъектов
            delete current; // удаление подобъекта
            break;
         }
      }
   }
}

// метод установки главного объекта
// new_p_head_object - указателль на новый главный объект
// quet - указатель на объект для проверки иерархии нового главного объекта
// s_base_ob - ссылка на список подобъектов главного объекта
bool cl_custom_base::set_head_object(cl_custom_base* new_p_head_object) {
   // проверка - является ли новый главный объект уже текущим главным
объектом
   if(this->get_main_custom() == new_p_head_object) {
      return true; // если да - возврат true
   }

   if(!(get_main_custom())) { // проверяем - имеет ли текущий объект главный
объект
      return false; // если нет - возврат false;
   }

   // проверяем - не является ли текущий объект частью иерархии нового
главного объекта
   cl_custom_base* quet = new_p_head_object; // указатель на новый главный
объект
   while(quet->get_main_custom()) { // пока у нового главного объекта есть
родитель
      if(quet == this) { // если текущий объект является частью иерархии
нового главного объекта
         return false; // возвран false
      }
      quet = quet->get_main_custom(); // переходим к родителю нового главного
объекта
   }

   // переменная для хранения ссылки на список подобъектов главного объекта
   vector<cl_custom_base*>& s_base_ob = p_main_custom_object->p_sub_customs;
   for(int i = 0; i < s_base_ob.size(); ++i) { // перебираем все подобъекты
главного объекта
      if(s_base_ob[i] == this) { // если текущий объект является подобъектов
главного объекта
         s_base_ob.erase(s_base_ob.begin() + i); // удаляем его из списка
подобъектов
         p_main_custom_object = new_p_head_object; // устанавливаем новый
главный объект
         new_p_head_object->p_sub_customs.push_back(this);    //    добавляем
```

```
текущий объект в список
        // подобъектов нового главного объекта
        return true; // возврат true
     }
   }
   return false;// если текущий объект не является подобъектом главнго
объекта - возврат false
}

// метод установки состояния объекта
// status_object - переданное состояние объкта
// quet - указатель на главный объект текущего объекта для проверки
состояний
// cl_custom_base_p - указатель на подобъект текущего объекта для установки
состояния
void cl_custom_base::set_state(int status_object) {
   if(status_object) { // если переданное состояние не равно нулю
      cl_custom_base* quet = p_main_custom_object; // указатель на главный
объект текущего объект
      // инициализация указателем quet на главный объект
      while(quet) { // пока есть главные объекты в иерархии
         if(!(quet->status)){ // если у главного объекта нет состояния
            return; // завершаем выполнение метода
         }
         quet = quet->get_main_custom(); // переходим к следующему главному
объекту
      }
      status = status_object; // устанавливаем состояние объекта
   } else { // если переданное состояние равно нулю
      for(auto cl_custom_base_p : p_sub_customs) { // перебираем все
подобъекты текущего объекта
         // итерируемся по каждому элементу в векторе p_sub_customs
         cl_custom_base_p->set_state(status_object);     //     устанавливаем
состояние для каждого подобъекта
         //вызов метода set_state с передачей значения status_object
         //метод set_state устанавливает состояние объекта на который
указывает cl_custom_base_p,
         //в значение status_object
      }
      status = status_object; //устанавливаем состояние текущего объекта
   }
}

// метод вызова метода для отступов
void cl_custom_base::indent_call_method() {
   indentation_method(); // вызываем метод для отступов
}

// метод получения абсолютных координат главного объекта
string cl_custom_base::get_main_absolute_coords() {
   cl_custom_base* ptr = this; // указатель на текущий объект
   string text = "/" + ptr->custom_name; // инициализация строки с именем
текущего объекта

   // пока у текущего объекта есть главный объект , добавляем его имя в
```

```
начало строки
   while(ptr->p_main_custom_object) {
      ptr = ptr->p_main_custom_object; // переходим к главному объекту
   }
   // если текущий объект является корневым - то абсолютные координаты -
просто "/"
   if(ptr == this) {
      text = "/";
   } else {
      // пока у текущего объекта у главного объекта есть родитель - добавляем
их имена в начало строки
      ptr = this;
      while(ptr->p_main_custom_object->p_main_custom_object) {
         ptr = ptr->p_main_custom_object; // переходим к родителю главного
объекта
         //доступ к члену p_main_custom_object структуры на который указывает
ptr
         //ptr получает значение члена p_main_custom_object этого ообъекта
         text = "/" + ptr->custom_name + text; // добавляем его имя в начало
строки
      }
   }
   return text; // возвращаем абсолютные координаты
}

// метод установки связи между объектами
// конструкторы SIGNAL_D и HANDLER_D - для приведения указателей на функции
к типам
// TYPE_SIGNAL и TYPE_HANDLER
// p_main_signal - указатель на метод сигнала, который будет вызываться при
активации сигнала
// p_main_target - указатель на целевой объект , к которому будет привязана
связи
// p_main_handler - указатель на метод обработчика, который будет вызываться
в целевом объекте при активации сигнала
void     cl_custom_base::connections_set_main(TYPE_SIGNAL     p_main_signal,
cl_custom_base* p_main_target, TYPE_HANDLER p_main_handler) {
   signals *ob1; // указатель на объект  структуры для хранения информации о
связи

   // типы TYPE_SIGNAL и TYPE_HANDLER - определены как указатели на методы
клсса cl_custom_base
   // таким образом связь задается тремя элементами - метод сигнала, целевым
объектом и методом
   // обработчика

   // проверяем - не существует ли уже такой связи
   for(int i = 0; i < main_connections.size(); i++) {

      // сравнение типа сигнала , целевой объект, обработчик текущей связи с
переданными значениям
      if(main_connections[i]->p_main_signal == p_main_signal &&
         // main_connections[i]->p_main_signal : тип сигнала текущей связи в
списке
         // p_main_signal : тип сигнала, переданный в метод для установки
```

```
связи
        main_connections[i]->p_main_target == p_main_target &&
        // main_connections[i]->p_main_target : целевой объект текущей связи
в списке
        // p_main_target : целевой объект, переданный в метод для установки
связи
        main_connections[i]->p_main_handler == p_main_handler) {
        // main_connections[i]->p_main_handler : обработчиу текущей связи в
списке
        //  p_main_handler  :  обработчик,  переданный  в  метод  для  установки
связи
        return; // если связь - уже существует - завершаем
     }
   }

   ob1 = new signals(); // создание объекта структуры для хранения информации
о новой связи

   //ob1 - указатель на объект структуры signals - он указывает на экземпляр
этой структуры
   //члену  p_main_signal  структуры  signals,  на  которую  указывает  ob1,
присваивается значение
   //переменной p_main_signal
   ob1->p_main_signal = p_main_signal; // устанавливаем тип сигнала для новой
связи
   ob1->p_main_target = p_main_target; //  устанавливаем целевой объект для
новой связи
   ob1->p_main_handler = p_main_handler;// устанавливаем обработчик для новой
связи
   main_connections.push_back(ob1); // добавление новой связи в список
}

// метод удаления связи между объектами
void    cl_custom_base::connections_delete_main(TYPE_SIGNAL    p_main_signal,
cl_custom_base* p_main_target, TYPE_HANDLER p_main_handler) {
   auto  ob1  =  main_connections.begin();  //  создаем  итерато  для  перебора
списка связей

   // перебираем список связей
   while(ob1 != main_connections.end()) {
      // если найдена связь с заданными параметрами, удаляем ее

      //(*ob1)-> p_main_signal == p_main_signal - проверка - совпадает ли тип
сигнала текущей
      //связи с переданным типом p_main_signal
      //(*ob1)->p_main_target  ==  p_main_target  -  проверка  -  указывает  ли
целевой объект текущей
      //связи на тот же объект, что и переданный p_main_target
      //(*ob1)->p_main_handler == p_main_handler - проверяет - совпадает ли
метод обработчика
      //текущей связи с переданным методом p_main_handler
      if((*ob1)-> p_main_signal == p_main_signal && (*ob1)->p_main_target ==
p_main_target && (*ob1)->p_main_handler == p_main_handler) {
         delete(*ob1);  //  освобождение  памяти  -  выделенную  под  структуру
связи
         ob1  =  main_connections.erase(ob1);  //  удаление  связи  из  списка
```

```
итератором
      } else {
         ++ob1; // переходим к следующей связи
      }
   }
}

// метод активации связи с передачей сигнала
void    cl_custom_base::connections_main_signal(TYPE_SIGNAL    p_main_signal,
string text) {
   TYPE_HANDLER p_main_handler; // обработчик сигнала
   cl_custom_base* p_main_target; // целевой объект

   // если текущий объект не находится в состоянии готовности, завершаем
метод
   if(!(this->status)) {
      return;
   }

   (this->*p_main_signal)(text);  //  вызываем  метод  сигнала  у  текущего
объекта, передавая текст сигнала
   //p_main_signal - указатель на объект класса cl_custom_base
   //доступ к p_main_handler через указатель на объект p_main_signal
   //(text) - передача аргумента text методу p_main_handler

   // перебираем список связей
   for(int i = 0; i < main_connections.size(); i++) {
      if(main_connections[i]->p_main_signal      ==      p_main_signal      &&
main_connections[i]->p_main_target->status) {
         // если найдена связь с совпадающим типом сигнала и целевой объект
находится в состоянии готовности
         p_main_handler  =  main_connections[i]->p_main_handler;//  получаем
обработчик из связи
         //main_connections - контейнер объектов, которые представляют связи
между сигналами и обработчиками
         //в системе, main_connections обращается к i-тому элементу этого
контейнера
         //main_connections[i]->p_main_handler     обращается     к     члену
p_main_handler структуры на который указывает
         //i-ый элемент main_connections
         p_main_target  =  main_connections[i]->p_main_target;  //  получаем
целевой объект из связи
         //main_connections - контейнер объектов, которые представляют связи
между сигналами и обработчиками
         //в системе, main_connections обращается к i-тому элементу этого
контейнера
         //main_connections[i]->p_main_target     обращается     к     члену
p_main_target структуры на который указывает
         //i-ый элемент main_connections
         (p_main_target->*p_main_handler)(text); // вызов метода обработчика
у целевого объекта, передавая текст
         // сигнала
         //p_main_target - указатель на объект класса cl_custom_base
         //доступ к p_main_handler через указатель на объект p_main_target
         //(text) - передача аргумента text методу p_main_handler
```

```
      }
   }
}

// метод передачи сигнала
// text - текст сигнала
void cl_custom_base::signal_method(string& text) {
   text = " Text: " + text + " (class: " + to_string(return_number_class()) +
")"; // формируем текст сигнала
   //вызов метода return_number_class()-возврат номера класса,
   //стандартная функция - to_string преобразует переданное ей значение в
строку
   cout << endl << "Signal from " << get_main_absolute_coords(); // выводим
информацию о месте отправки сигнала
}

// метод получения сигнала
// text - текст сигнала
void cl_custom_base::handler_method(string text) {
   cout << endl << "Signal to " << get_main_absolute_coords() << text; //
вывод информации о месте
   // приема сигнала и сам текст сигнала
}

// метод возврата номера класса
int cl_custom_base::return_number_class() {
   return 1; // возврат номера класса
}

// метод приведения всех объектов в состоянии готовности
void cl_custom_base::set_object_ready() {
   set_state(1); // установление состояния готовности текущего объекта

   // рекурсивно вызываем этот метод для всех подобъектов текущего объекта
   for(int i = 0; i < p_sub_customs.size(); i++) {
      p_sub_customs[i]->set_object_ready();
   }
}
```

5.14 Файл cl_custom_base.h

Листинг 14 – cl_custom_base.h

```
#ifndef __CL_CUSTOM_BASE__H
#define __CL_CUSTOM_BASE__H

#include <iostream>
#include <vector>
#include <string>
#include <queue>
```

```
using namespace std;

class cl_custom_base;

#define SIGNAL_D(signal_f)(TYPE_SIGNAL)(&signal_f)
#define HANDLER_D(handler_f)(TYPE_HANDLER)(&handler_f)
// макросы для преобразования методов в типы сигналов и обработчиков

typedef void(cl_custom_base::*TYPE_SIGNAL)(string&);
typedef void(cl_custom_base::*TYPE_HANDLER)(string);
// директивы определения типов для сигналов и обработчиков

class cl_custom_base {
public:
   cl_custom_base(cl_custom_base* p_main_custom_object, string custom_name =
"Base object"); // конструткор
   ~cl_custom_base(); // деструктор
   bool set_custom_name(string new_custom_name); // установка имени объекта
   string get_custom_name(); // получение имени объекта
   string get_main_absolute_coords(); // возврат кординат главного объекта
   bool  set_head_object(cl_custom_base*  new_p_head_object);  //  установка
нового головного объекта
   cl_custom_base* get_main_custom(); // возврат указателя главного объекта
   void   connections_set_main(TYPE_SIGNAL   p_main_signal,   cl_custom_base*
p_main_target, TYPE_HANDLER p_main_handler);
   // установление новой связи
   void  connections_delete_main(TYPE_SIGNAL  p_main_signal,  cl_custom_base*
p_main_target, TYPE_HANDLER p_main_handler);
   // удаление существующей связи
   void     connections_main_signal(TYPE_SIGNAL     p_main_signal,     string
p_main_handler);
   // вызов сигнала и обработчика
   void signal_method(string& text); // метод - представляющий сигнал
   void  handler_method(string  text);  //  метод  -  представляющий  обработчик
сигнала
   void  indentation_method();  //  метод  для  вывода  структуры  объектов  с
отступами
   void   indent_call_method();   //   вспомогательный   метод   для   вызова
indentation_method
   virtual int return_number_class(); // возврат номера класса
   void set_object_ready(); // установка состояния готовности для объекта и
его подчиненных
   cl_custom_base* get_object_path(string path_custom); // нахождение объекта
по пути
   cl_custom_base*  search_current(string  s_custom_name);  //  поиск  объекта
среди подчиненных текущего объекта
   cl_custom_base* search_tree(string s_custom_name); // поиск объекта среди
всех объектов в дереве
   cl_custom_base*  get_sub_customs(string  main_custom_name);  //  возврат
указателя на подчиненных объектов по имени
   void set_state(int status_object); // установка состояния объекта
   void  delete_custom_sub_name(string  sub_name);  //  удаление  подчиненных
объектов по имени
```

```
   // в этой структуре
   struct signals { // структура - связь сигнала и обработчика
      TYPE_SIGNAL p_main_signal; // указатель на метод сигнала
      cl_custom_base* p_main_target; // указатель на целевой объект
      TYPE_HANDLER p_main_handler; // указатель на метод обработчика
   };
private:
   string custom_name; // поле - хранение имени объекта
   cl_custom_base*  p_main_custom_object;  //   поле  -  хранение  указателя  на
главный объект
   vector<cl_custom_base*> p_sub_customs; // вектор указателей на подчиненные
объекты
   vector<signals*>   main_connections;   //   вектор   связей   сигналов   и
обработчиков
   int status = 0; // состояние объекта
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_custom_application.h"

#include <string>
#include <iostream>

int main() {
   cl_custom_application   obj_custom_app(nullptr);   //   создание   объекта
obj_custom_app
   obj_custom_app.build_tree_objects(); // вызов метода для построения дерева
объектов
   return(obj_custom_app.exec_custom_app());  //  выполнение  приложения  и
возврат его результата
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 38.

Таблица 38 – 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 DELETE_CONNECT /objects_S2/ object_s4 /object_s2/object_s6 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_s6 object_s13Object /objects_S2/object_s 4 not found 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_s13Object /objects_S2/object_s 4 not found 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

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были получены практические навыки в области объекто-ориентированного программирования с использованием алгоритмического языка C++. Были разработаны древовидная иерархия объектов и базовый класс 'cl_custom_base', обеспечивающий основные функциональные возможности для работы с иерархией. Основные методы класса включают:

Выполненная работа позволила глубже понять и практически применить принципы объекто-ориентированного программирования, что является важным этапом в обучении и профессиональном развитии.

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).