Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

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

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

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

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

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

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

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

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

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

4. Конец цикла.

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

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

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

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

Каждый метод сигнала с новой строки выводит:

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

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

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

Каждый метод обработчика с новой строки выводит:

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

Моделировать работу системы, которая выполняет следующие команды с параметрами:

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

целевого объекта» – устанавливает связь;

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

Реализовать алгоритм работы системы:

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

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

1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве

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

После ввода состава дерева иерархии построчно вводится:

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

Ввод информации для построения связей завершается строкой, которая содержит:

```
«end_of_connections»
```

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

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

Команда END присутствует обязательно.

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

Если не найден объект по координате:

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

Если не найден целевой объект по координате:

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

Пример ввода:

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

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

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

Если отработал метод обработчика:

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

Пример вывода:

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

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

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект ob_cl_application класса Cl_application предназначен для конструирования и запуска системы;
- объект стандартного потока ввода с клавиатуры cin;
- объект стандартного потока вывода на экран cout;
- условный оператор if..else;
- оператор цикла for;
- оператор цикла с предусловием while;
- структура connection с полями: p_signal - указатель на метод сигнала, p_target - указатель на объект класса Cl_base, p_handler - указатель на метод обработчика;
- указатель на метод сигнала объекта TYPE_SIGNAL;
- указатель на метод обработчика объекта TYPE_HANDLER.

Класс Cl_base:

- свойства/поля:
    - поле состояние объекта:
        - наименование — object_state;
        - тип — int;
        - модификатор доступа — private;
    - поле класс объекта:
        - наименование — object_class;
        - тип — int;
        - модификатор доступа — public;
    - поле наименование объекта:
        - наименование — s_object_name;

        - тип — string;
        - модификатор доступа — private;
    - поле указатель на головной объект для текущего объекта:
        - наименование — p_head_object;
        - тип — Cl_base*;
        - модификатор доступа — private;
    - поле динамический массив указателей на объекты, подчиненные текущему объекту:
        - наименование — subordinate_objects;
        - тип — vector <Cl_base*>;
        - модификатор доступа — private;
    - поле динамический массив связей сигнал-обработчик:
        - наименование — connects;
        - тип — vector <connection*>;
        - модификатор доступа — private;
- функционал:
    - метод get_absolute_coordinate — метод получения абсолютной координаты объекта;
    - метод set_connect — метод установки связи сигнал-обработчик;
    - метод delete_connect — метод удаления связи сигнал-обработчик;
    - метод emit_signal — метод выдачи сигнала от текущего объекта с передачей строковой переменной.

Класс Cl_application:

- функционал:
    - метод build_tree_object — метод конструирования системы;
    - метод exec_app — метод запуска системы.

Класс Cl_child_2:

- функционал:
    - метод Cl_child_2 — параметризированный конструктор;
    - метод get_signal — метод сигнала;
    - метод get_handler — метод обработчика.

Класс Cl_child_3:

- функционал:
    - метод Cl_child_3 — параметризированный конструктор;
    - метод get_signal — метод сигнала;
    - метод get_handler — метод обработчика.

Класс Cl_child_4:

- функционал:
    - метод Cl_child_4 — параметризированный конструктор;
    - метод get_signal — метод сигнала;
    - метод get_handler — метод обработчика.

Класс Cl_child_5:

- функционал:
    - метод Cl_child_5 — параметризированный конструктор;
    - метод get_signal — метод сигнала;
    - метод get_handler — метод обработчика.

Класс Cl_child_6:

- функционал:
    - метод Cl_child_6 — параметризированный конструктор;
    - метод get_signal — метод сигнала;
    - метод get_handler — метод обработчика.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	Cl_base			Базовый класс	
		Cl_application	public		2
		Cl_child_2	public		3
		Cl_child_3	public		4
		Cl_child_4	public		5
		Cl_child_5	public		6
		Cl_child_6	public		7
2	Cl_application			Класс-приложение	
3	Cl_child_2			Производный класс	
4	Cl_child_3			Производный класс	
5	Cl_child_4			Производный класс	
6	Cl_child_5			Производный класс	
7	Cl_child_6			Производный класс	

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода get_absolute_coordinate класса Cl_base

Функционал: Метод получения абсолютной координаты объекта.

Параметры: нет.

Возвращаемое значение: Строка с абсолютной координатой объекта.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода get_absolute_coordinate класса Cl_base

№	Предикат	Действия	№ перехода
1		Объявление указателя object на текущий объект. Объявление строки absolute и инициализация пустой строкой	2
2	Результат метода get_head_object объекта по адресу object не равен nullptr?	absolute присвоить "/(результат метода get_object_name объекта по адресу object) (absolute)". object присвоить результат метода get_head_object объекта по адресу object	2
			3
3	absolute пустая?	Вернуть "/"	∅
		Вернуть absolute	∅

3.2 Алгоритм метода set_connect класса Cl_base

Функционал: Метод установки связи сигнал-обработчик.

Параметры: Указатель p_signal на метод сигнала, указатель p_target на объект класса Cl_base, указатель p_handler на метод обработчика.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода set_connect класса Cl_base

№	Предикат	Действия	№ перехода
1	Перебираем элементы вектора connects, value - указатель на текущий элемент		2
			3
2	Свойства p_signal, p_target, p_handler объекта по адресу value равны параметрам p_signal, p_target, p_handler соответсвенно?		∅
			1
3		Объявление указателя value на объект структуры connection и инициализация новым объектом структуры connection	4
4		Свойствам p_signal, p_target, p_handler объекта по адресу value присвоить значение параметров p_signal, p_target, p_handler соответственно	5
5		Добавить value в connects	∅

3.3 Алгоритм метода delete_connect класса Cl_base

Функционал: Метод удаления связи сигнал-обработчик.

Параметры: Указатель p_signal на метод сигнала, указатель p_target на

объект класса Cl_base, указатель p_handler на метод обработчика.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода delete_connect класса Cl_base

№	Предикат	Действия	№ перехода
1		Объявление логической переменной f и инициализация значением false	2
2		Объявление целочисленной переменной index и инициализация значением 0	3
3	Перебираем элементы вектора connects, value - указатель на текущий элемент		4
			5
4	Свойства p_signal, p_target, p_handler объекта по адресу value равны параметрам p_signal, p_target, p_handler соответсвенно?	f присвоить значение true	5
		Увеличение index на 1	3
5	Значение f равняется true?	Удалить из вектора connects элемент с индексом index	∅
			∅

3.4 Алгоритм метода emit_signal класса Cl_base

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

Параметры: Указатель p_signal на метод сигнала, строка message.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода emit_signal класса Cl_base

№	Предикат	Действия	№ перехода
1	Свойство object_state равняется 0?		∅
			2
2		Объявление указателя p_handler на метод обработчика. Объявление указателя p_target на объект класса Cl_base	3
3		Вызов метода сигнала текущего объекта с параметром message	4
4	Перебираем элементы вектора connects, value - указатель на текущий элемент		5
			∅
5	Свойство p_signal объекта по адресу value равно параметру p_signal?	p_target присвоить значение свойства p_target объекта по адресу value. p_handler присвоить значение свойства p_handler объекта по адресу value.	6
			4
6	Свойство object_state объекта по адресу p_target не равно 0?	Вызов метода обработчика объекта по адресу p_target с параметром message	4
			4

3.5 Алгоритм метода build_tree_object класса Cl_application

Функционал: Метод конструирования системы.

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода build_tree_object класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных head и sub, указателя head_object на объект класса Cl_base, целочисленной переменной iClass	2
2		Ввод значения head с клавиатуры	3
3	head равно "endtree"?		12
		Ввод значений sub и iClass с клавиатуры	4
4		Присвоение указателю head_object результат метода get_object_by_coordinate с параметром head	5
5	head_object не равен nullptr?		6
			8
6	Результат метода get_sub_object_by_name объекта по адресу head_object с параметром sub не равен nulptr?	Вывод на экран "(head) Dubbing the names of subordinate objects"	2
			7
7		Создание объекта класса Cl_child_i с параметрами head_object и sub, где i - номер класса iClass	2
8		Вывод на экран "Object tree"	9
9		Вызов метода show_object_tree	10

№	Предикат	Действия	№ перехода
10		Вывод на экран "The head object (head) is not found"	11
11		Выход из программы с кодом 1	∅
12		Объявление указателей from и to на объекты класса Cl_base и инициализация this и nullptr соответсвенно	13
13		Объявление вектора указателей на методы сигналов signals. Объявление вектора указателей на методы обработчиков handlers	14
14		Ввод строки head с клавиатуры	15
15	head равно "end_of_connections"?		∅
		Присвоение указателю from результат метода get_object_by_coordinate с параметром подстроки head до первого пробела	16
16		Присвоение указателю to результат метода get_object_by_coordinate с параметром подстроки head после первого пробела	17
17	Свойство object_class объекта по адресу from меньше 2?		14
		Вызов метода set_conect объекта по адресу from с параметрами: элемент signals с индексом object_class - 2, to, элемент handlers с индексом object_class - 2	14

3.6 Алгоритм метода exec_app класса Cl_application

Функционал: Метод запуска системы.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода exec_app класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление строки command, вектора строк args	2
2		Вывод на экран "Object tree"	3
3		Вызов метода show_object_tree	4
4		Объявление указателей from и to на объекты класса Cl_base и инициализация this и nullptr соответсвенно	5
5		Объявление вектора указателей на методы сигналов signals. Объявление вектора указателей на методы обработчиков handlers	6
6		Ввод с клавиатуры command	7
7	command равно "END"?		∅
		Очистить вектор args	8
8	В строке command найден пробел?	Добавление в args подстроки command до первого пробела. command присвоить подстроку command поле первого пробела	8
		Добавление command в args	9
9	Размер args меньше 3?		6
		Присвоение указателю from результат метода get_object_by_coordinate с параметром элемента	10

№	Предикат	Действия	№ перехода
		args с индексом 1	
10		Присвоение указателю to результат метода get_object_by_coordinate с параметром элемента args с индексом 2	11
11	from равен nullptr?	Вывод на экран "Object (элемент args с индексом 1) not found"	6
			12
12	Элемент args с индексом 0 равен "SET_CONDITION"?	Вызов метода change_object_state объекта по адресу from с параметром элемента args с индексом 2	6
			13
13	Свойство object_class объекта по адресу from меньше 2?		6
			14
14	Элемент args с индексом 0 равен "EMIT"?	command присвоить пустую строку	15
			18
15	Перебираем числа от 2 до размера args включительно, i - текущее число		16
			17
16	i не равен размеру args - 1?	В конце command добавить элемент args с индексом i и пробел	15
		В конце command добавить элемент args с индексом i	15
17		Вызов метода emit_signal объекта по адресу from с параметрами: элемент signals с индексом object_class - 2, command	6

№	Предикат	Действия	№ перехода
18	to не равен nullptr?		19
		Вывод на экран "Handler object (элемент args с индексом 2) not found"	6
19	Элемент args с индексом 0 равен "SET_CONNECT"?	Вызов метода set_conect объекта по адресу from с параметрами: элемент signals с индексом object_class - 2, to, элемент handlers с индексом object_class - 2	6
			20
20	Элемент args с индексом 0 равен "DELETE_CONNECT"?	Вызов метода delete_conect объекта по адресу from с параметрами: элемент signals с индексом object_class - 2, to, элемент handlers с индексом object_class - 2	6
			6

3.7 Алгоритм конструктора класса Cl_child_2

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на головной объект, строка s_object_name с наименованием объекта.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса Cl_child_2

№	Предикат	Действия	№ перехода
1		Присвоение свойству object_class значения 2	∅

3.8 Алгоритм метода get_signal класса Cl_child_2

Функционал: Метод сигнала.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_signal класса Cl_child_2

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from (результат метода get_absolute_coordinate)"	2
2		Добавление в конце message " (class: (object_class))"	∅

3.9 Алгоритм метода get_handler класса Cl_child_2

Функционал: Метод обработчика.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода get_handler класса Cl_child_2

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to (результат метода get_absolute_coordinate) Text: (message)"	∅

3.10 Алгоритм конструктора класса Cl_child_3

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на головной объект, строка s_object_name с наименованием объекта.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса Cl_child_3

№	Предикат	Действия	№ перехода
1		Присвоение свойству object_class значения 3	∅

3.11 Алгоритм метода get_signal класса Cl_child_3

Функционал: Метод сигнала.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода get_signal класса Cl_child_3

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from (результат метода get_absolute_coordinate)"	2
2		Добавление в конце message " (class: (object_class))"	∅

3.12 Алгоритм метода get_handler класса Cl_child_3

Функционал: Метод обработчика.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода get_handler класса Cl_child_3

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to (результат метода get_absolute_coordinate) Text: (message)"	∅

3.13 Алгоритм конструктора класса Cl_child_4

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на головной объект, строка s_object_name с наименованием объекта.

Алгоритм конструктора представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса Cl_child_4

№	Предикат	Действия	№ перехода
1		Присвоение свойству object_class значения 4	∅

3.14 Алгоритм метода get_signal класса Cl_child_4

Функционал: Метод сигнала.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода get_signal класса Cl_child_4

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from (результат метода get_absolute_coordinate)"	2
2		Добавление в конце message " (class: (object_class))"	∅

3.15 Алгоритм метода get_handler класса Cl_child_4

Функционал: Метод обработчика.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода get_handler класса Cl_child_4

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to (результат метода get_absolute_coordinate) Text: (message)"	∅

3.16 Алгоритм конструктора класса Cl_child_5

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на головной объект, строка s_object_name с наименованием объекта.

Алгоритм конструктора представлен в таблице 17.

Таблица 17 – Алгоритм конструктора класса Cl_child_5

№	Предикат	Действия	№ перехода
1		Присвоение свойству object_class значения 5	∅

3.17 Алгоритм метода get_signal класса Cl_child_5

Функционал: Метод сигнала.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода get_signal класса Cl_child_5

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from (результат метода get_absolute_coordinate)"	2
2		Добавление в конце message " (class: (object_class))"	∅

3.18 Алгоритм метода get_handler класса Cl_child_5

Функционал: Метод обработчика.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода get_handler класса Cl_child_5

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to (результат метода get_absolute_coordinate) Text: (message)"	∅

3.19 Алгоритм конструктора класса Cl_child_6

Функционал: Параметризированный конструктор.

Параметры: Указатель p_head_object на головной объект, строка s_object_name с наименованием объекта.

Алгоритм конструктора представлен в таблице 20.

Таблица 20 – Алгоритм конструктора класса Cl_child_6

№	Предикат	Действия	№ перехода
1		Присвоение свойству object_class значения 6	∅

3.20 Алгоритм метода get_signal класса Cl_child_6

Функционал: Метод сигнала.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода get_signal класса Cl_child_6

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from (результат метода get_absolute_coordinate)"	2
2		Добавление в конце message " (class: (object_class))"	∅

3.21 Алгоритм метода get_handler класса Cl_child_6

Функционал: Метод обработчика.

Параметры: Строка message с текстом сообщения.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода get_handler класса Cl_child_6

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to (результат метода get_absolute_coordinate) Text: (message)"	∅

3.22 Алгоритм функции main

Функционал: Конструирование и запуск системы.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм функции представлен в таблице 23.

Таблица 23 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Объявление объекта ob_cl_application класса Cl_application с параметром nullptr	2
2		Вызов метода build_tree_object объекта ob_cl_application	3

№	Предикат	Действия	№ перехода
3		Возврат результата метода exec_app объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-9.

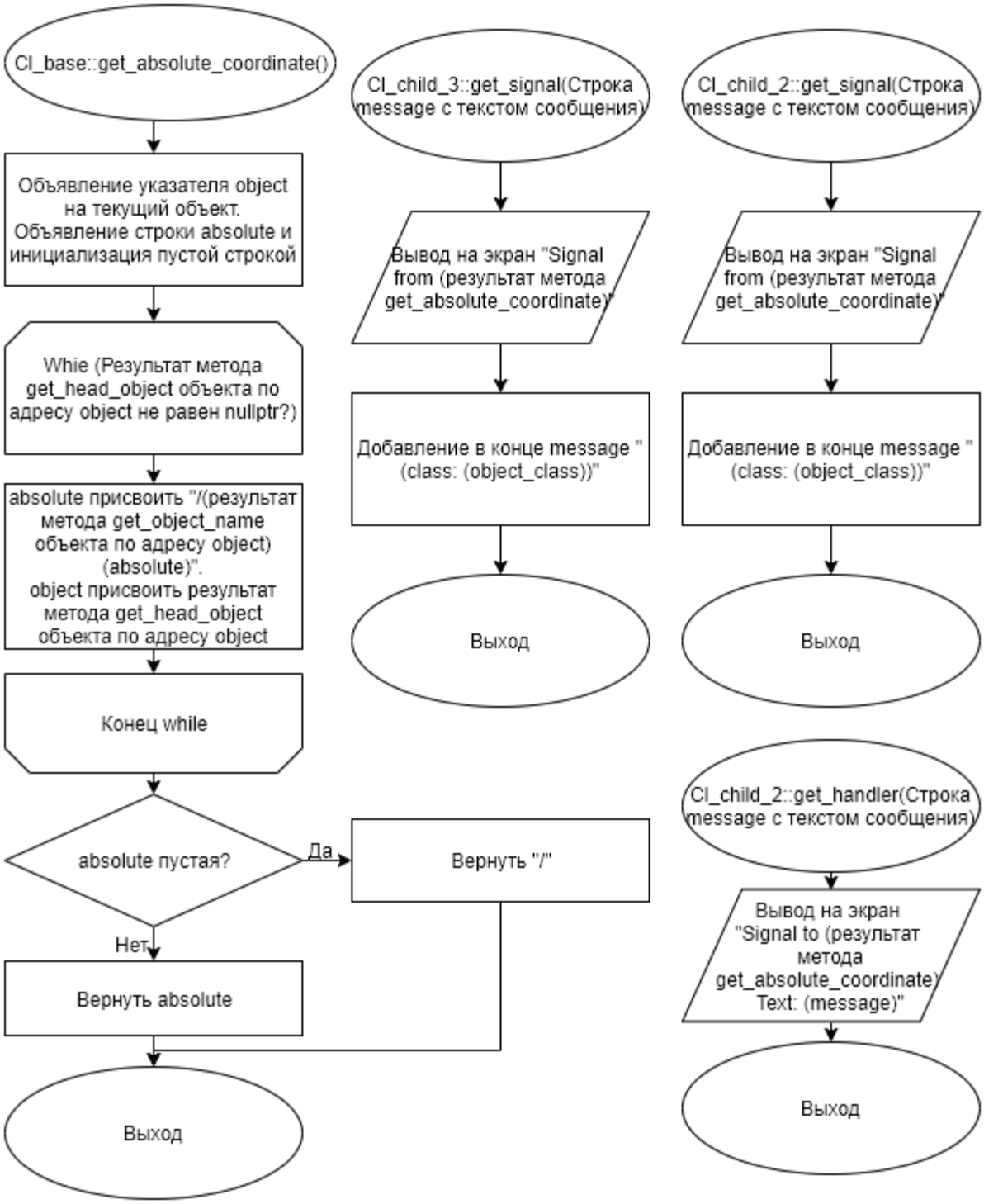

Рисунок 1 – Блок-схема алгоритма

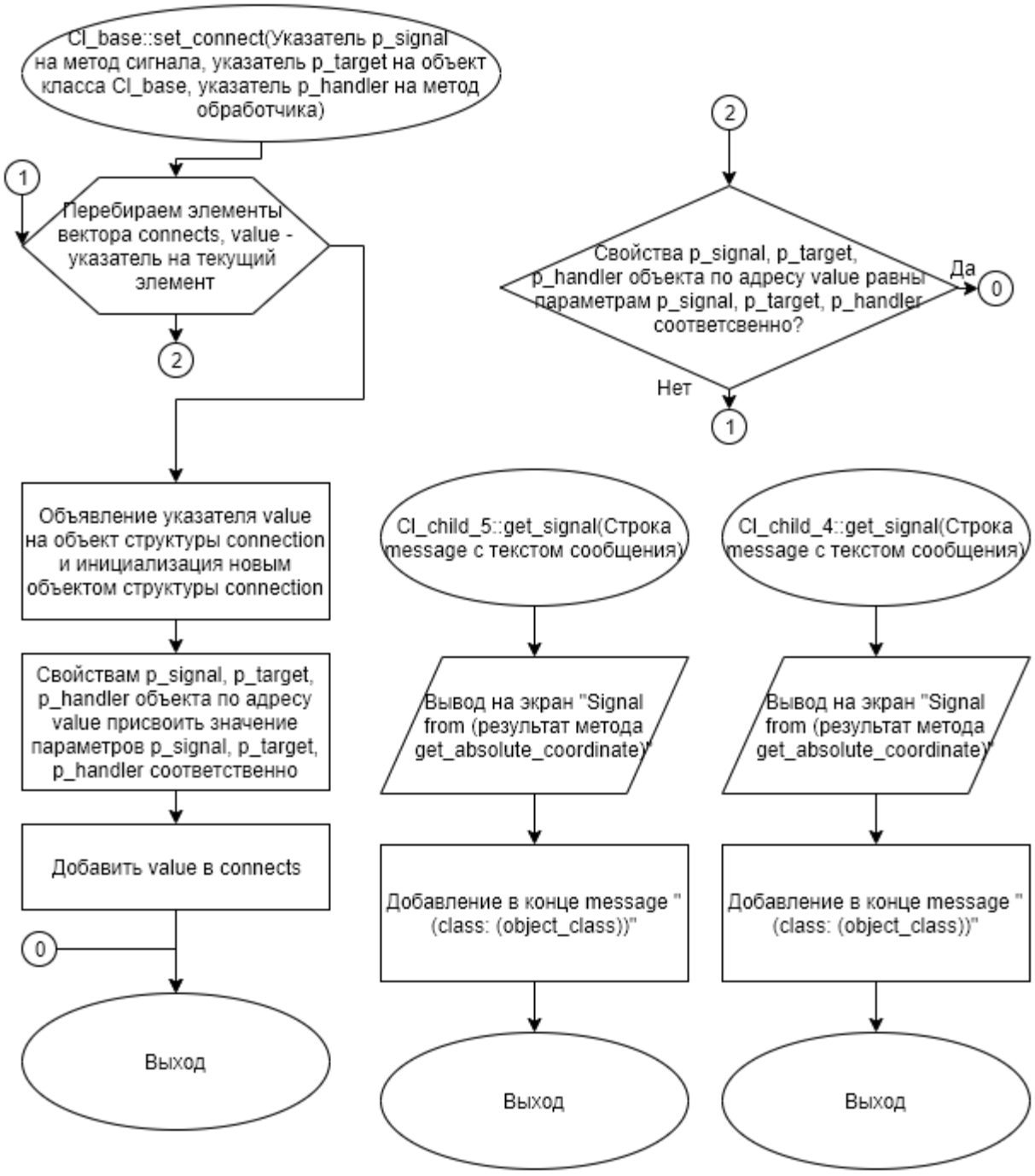

Рисунок 2 – Блок-схема алгоритма

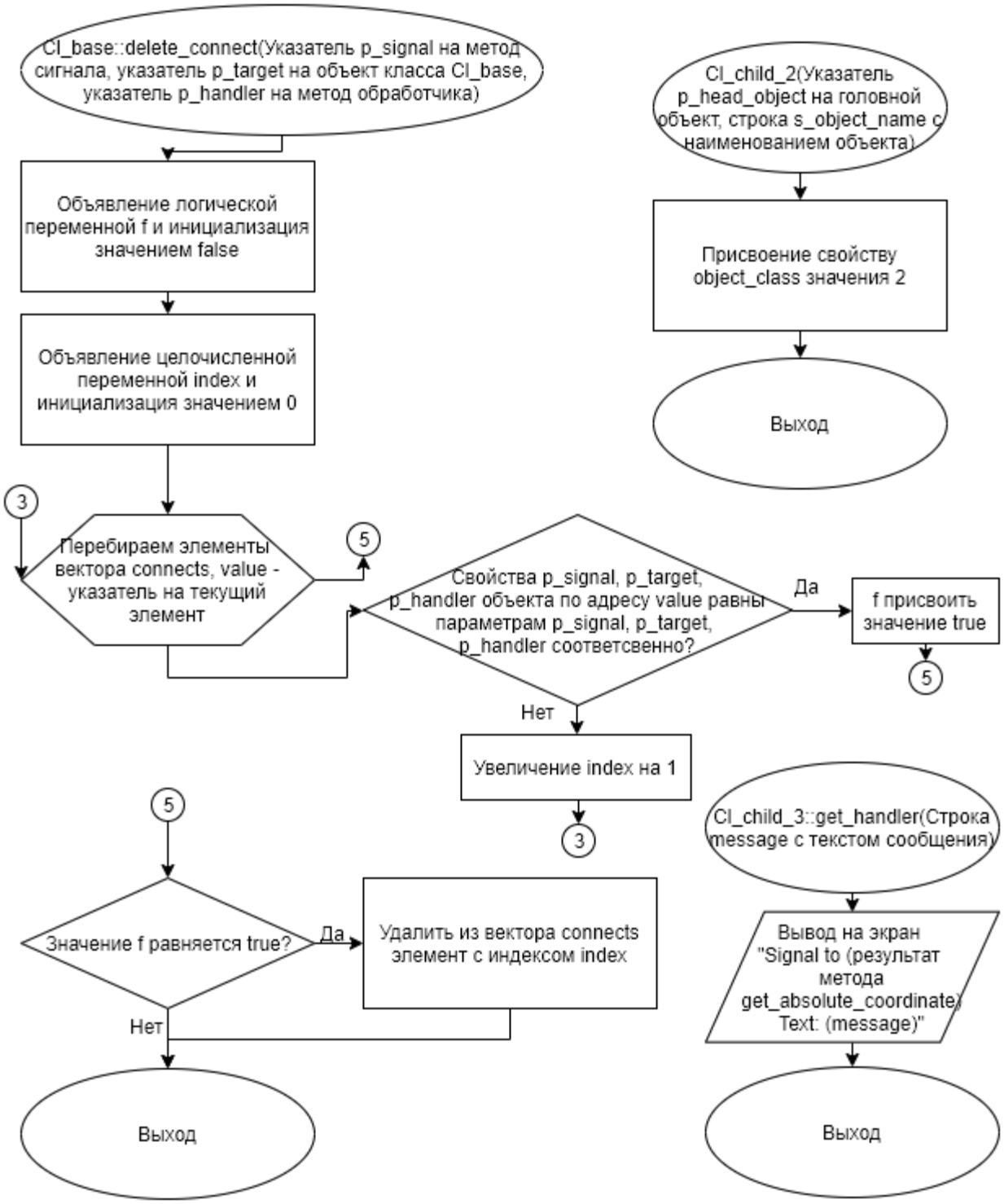

Рисунок 3 – Блок-схема алгоритма

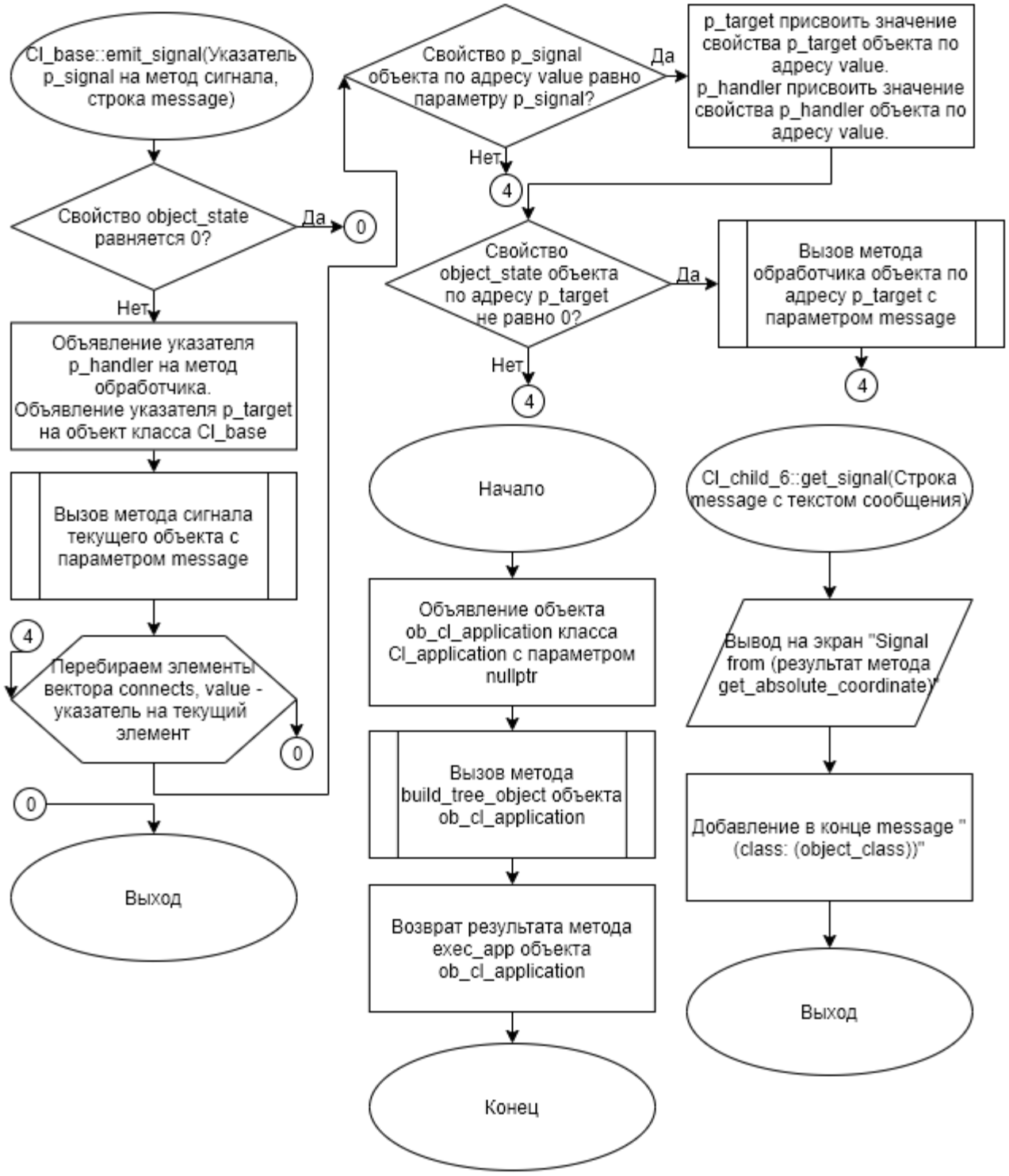

Рисунок 4 – Блок-схема алгоритма

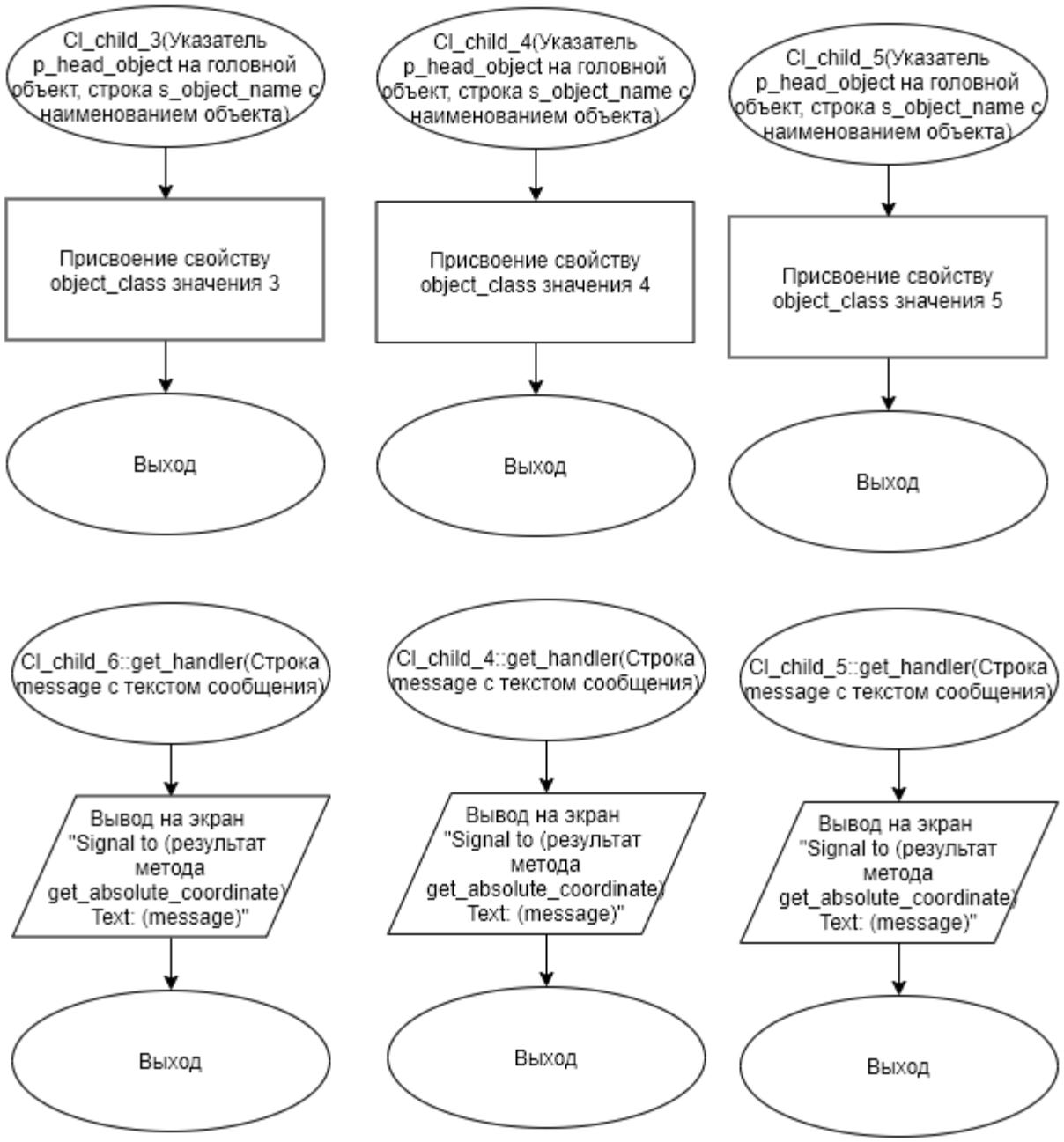

Рисунок 5 – Блок-схема алгоритма

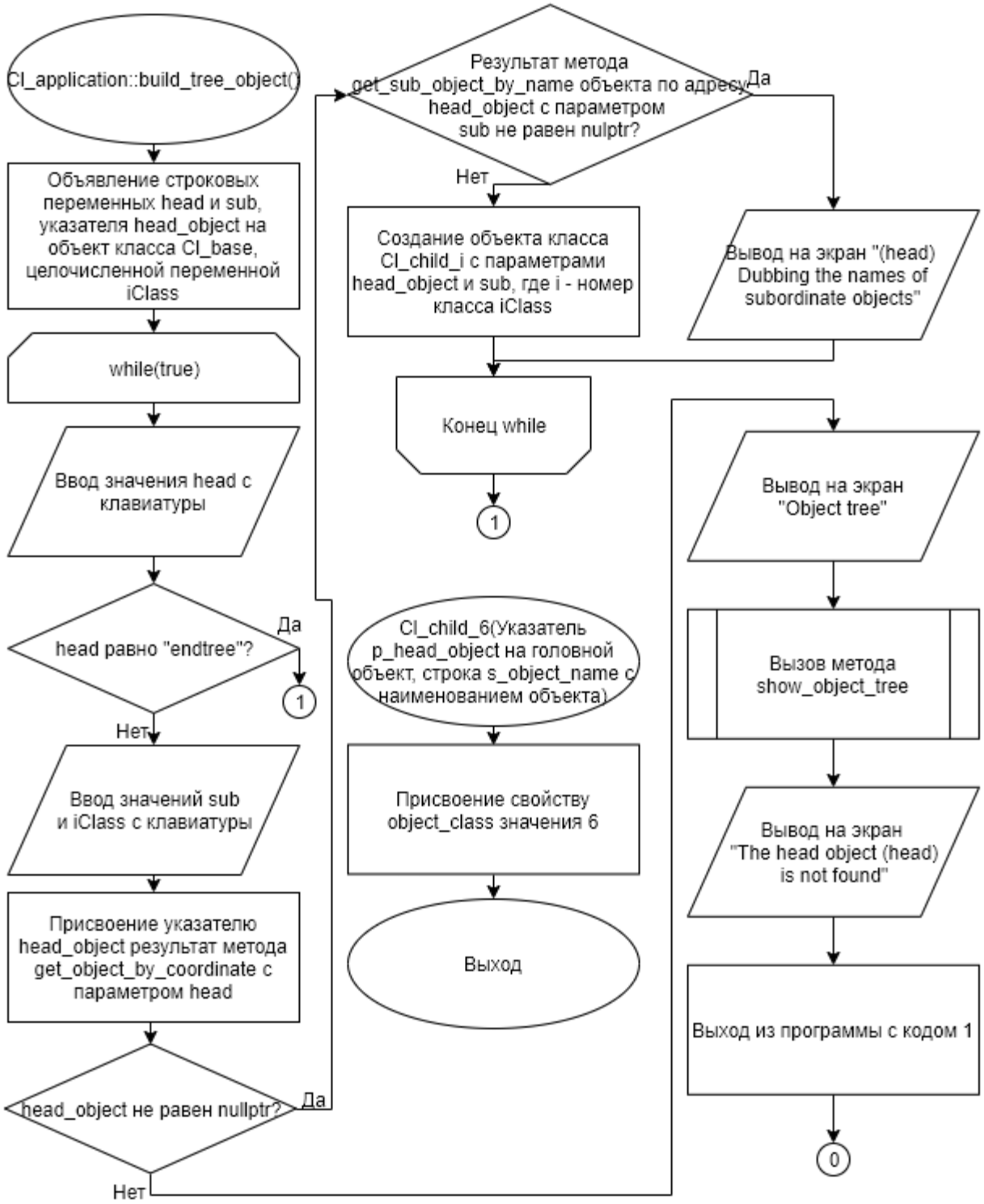

Рисунок 6 – Блок-схема алгоритма

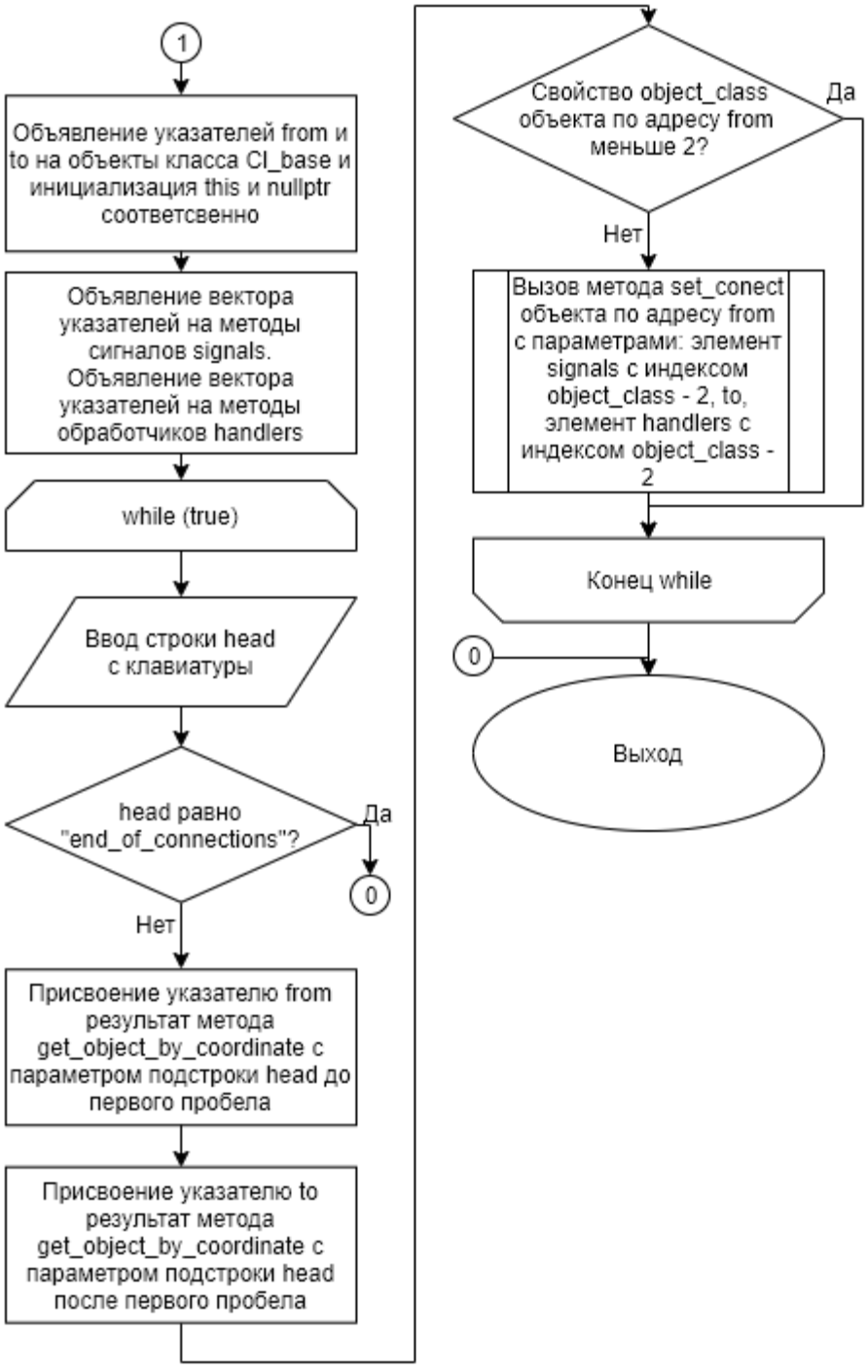

Рисунок 7 – Блок-схема алгоритма

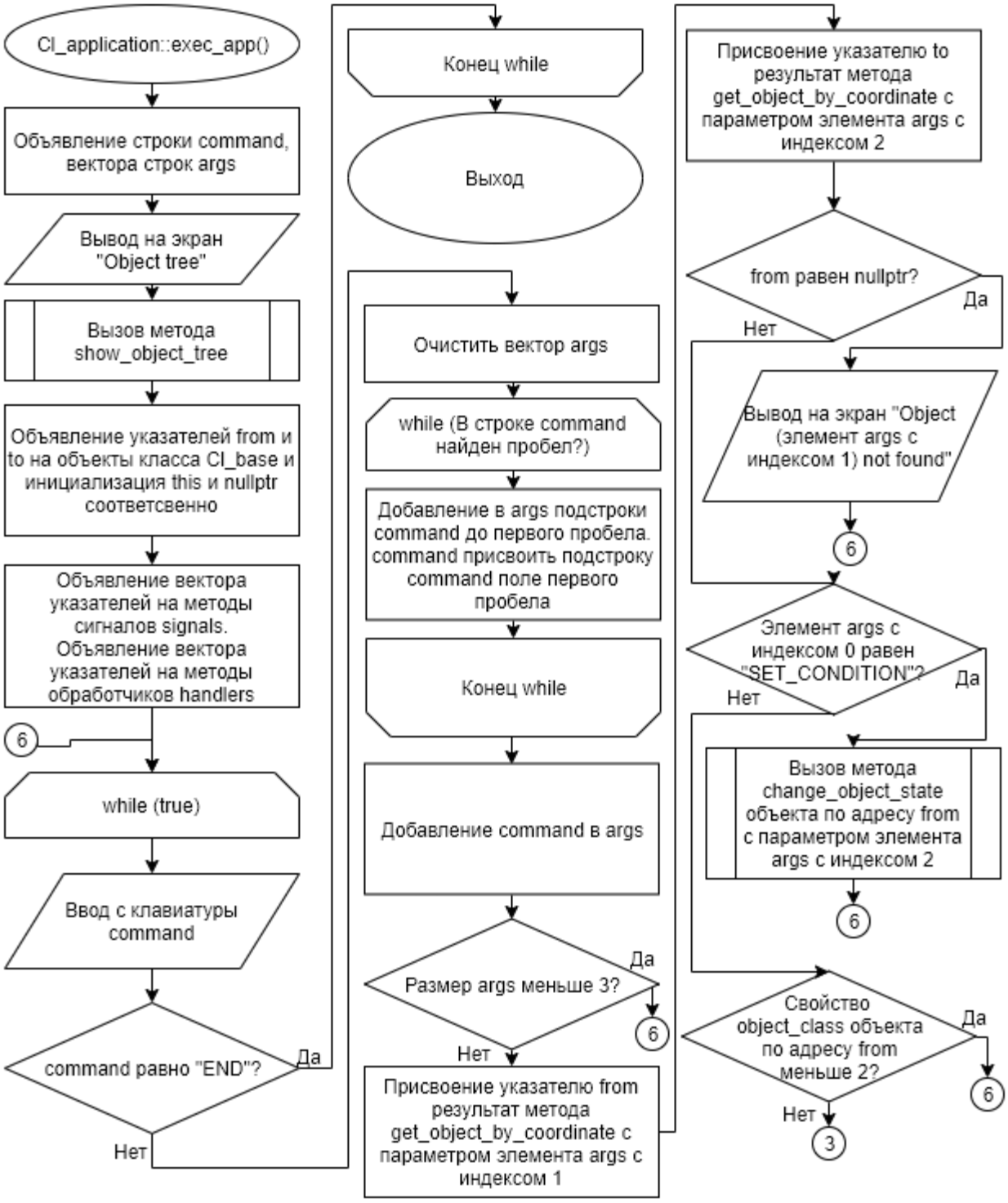

Рисунок 8 – Блок-схема алгоритма

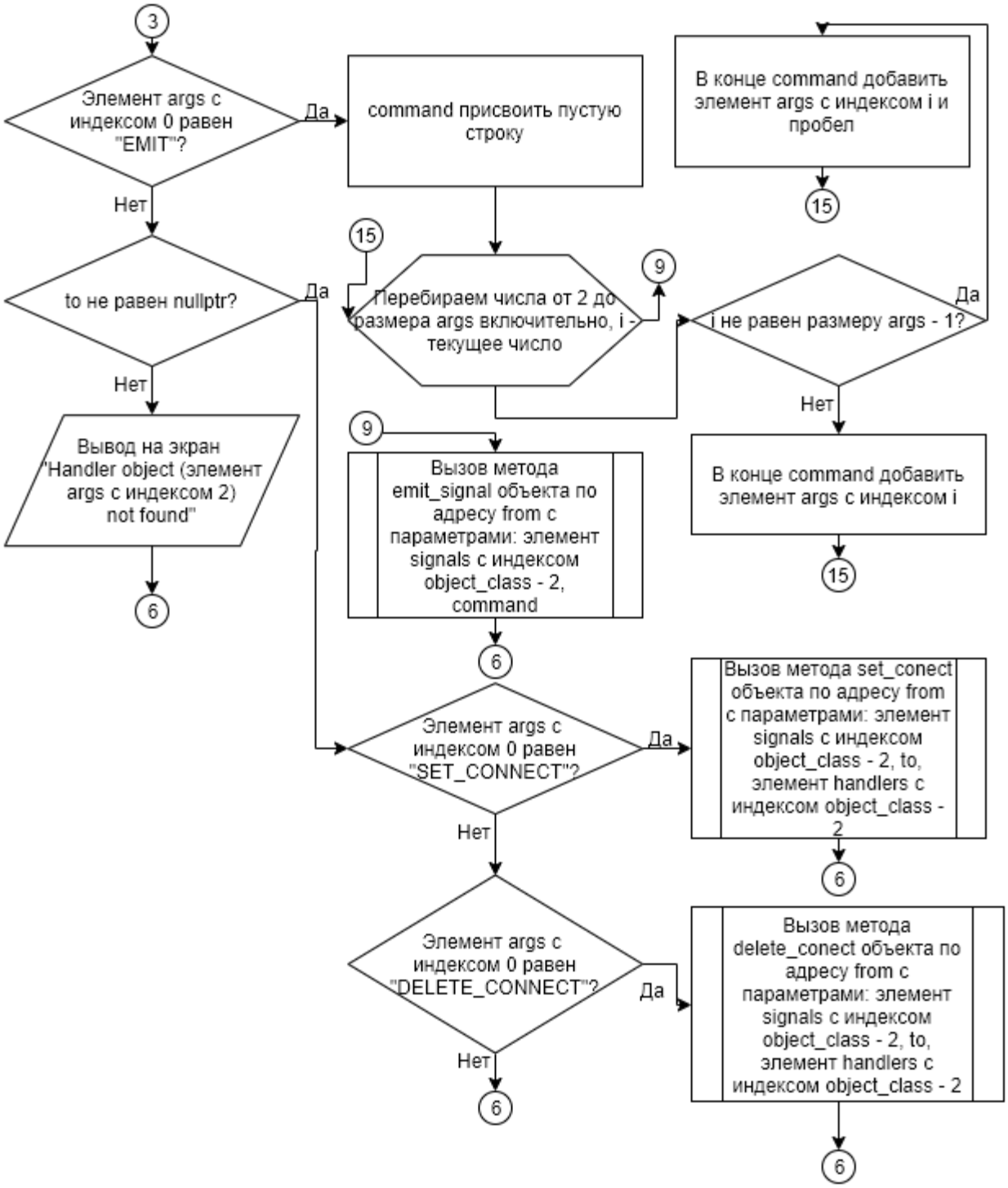

Рисунок 9 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл Cl_application.cpp

Листинг 1 – Cl_application.cpp

```
#include "Cl_application.h"

Cl_application::Cl_application(Cl_base*         p_head_object,         string
s_name_object):Cl_base(p_head_object, s_name_object)
{
   cin >> s_name_object;//ввод имени объекта
   change_object_name(s_name_object);//изменение имени объекта
}

void Cl_application::build_tree_objects()
{
   string head, sub;//переменные имен объектов
   Cl_base* head_object;//указатель на головной объект
   int iClass;//номера классов
   while (true) {//ввод дерева
      cin >> head;//ввод имени головного объекта
      if (head == "endtree") {//если endtree то выход из цикла
         break;
      }
      cin >> sub >> iClass;//ввод номера класса
      head_object = get_object_by_coordinate(head);//поиск годовного объекта
по координате
      if (head_object) {//Координата не nullptr?
         if(head_object -> get_sub_object_by_name(sub)) {//Есть ли дубль?
            cout << head << "      Dubbing the names of subordinate objects"
<< endl;//сообщение о дубле
            continue;
         }
         switch(iClass)//создание объекта по номеру класса
         {
            case 2:
               new Cl_child_2(head_object, sub);
               break;
            case 3:
               new Cl_child_3(head_object, sub);
               break;
            case 4:
               new Cl_child_4(head_object, sub);
               break;
```

```
                case 5:
                   new Cl_child_5(head_object, sub);
                   break;
                case 6:
                   new Cl_child_6(head_object, sub);
                   break;
             }
          }
          else {
             cout << "Object tree" << endl;
             this -> show_object_tree();//Вывод построенного дерева
             cout << "The head object " << head << " is not found";//вывод
сообщения об ошибке
             exit(1);
          }
       }
       Cl_base* from = this, * to = nullptr;//указатели на объекты
       vector   <TYPE_SIGNAL>   signals   =   {SIGNAL_D(Cl_child_2::get_signal),
SIGNAL_D(Cl_child_3::get_signal),          SIGNAL_D(Cl_child_4::get_signal),
SIGNAL_D(Cl_child_5::get_signal),  SIGNAL_D(Cl_child_6::get_signal)};//вектор
сигналов
       vector  <TYPE_HANDLER>  handlers  =  {HANDLER_D(Cl_child_2::get_handler),
HANDLER_D(Cl_child_3::get_handler),      HANDLER_D(Cl_child_4::get_handler),
HANDLER_D(Cl_child_5::get_handler),
HANDLER_D(Cl_child_6::get_handler)};//вектор обработчиков
       getline(cin, head);//чтение строки
       while (true) {
          getline(cin, head);//чтение строки
          if (head == "end_of_connections") {//строка означает завершение?
             break;
          }
          else {
             from    =    get_object_by_coordinate(head.substr(0,    head.find('
')));//извлечение указателя на первый объект
             to   =   get_object_by_coordinate(head.substr(head.find('   ')   +
1));//извлечение указателя на второй объект
             if (from -> object_class < 2) {//класс объекта меньше 2?
                continue;
             }
             else {
                from -> set_connect(signals.at(from -> object_class - 2), to,
handlers.at(from -> object_class - 2));//установка связи
             }
          }
       }
    }

    int Cl_application::exec_app()
    {
       string command;//строка команды
       vector<string> args;//вектор для парсинга строки
       cout << "Object tree";
       show_object_tree();//вывод дерева объектов
       Cl_base* from = this, * to = nullptr;//указатели на объект
       vector   <TYPE_SIGNAL>   signals   =   {SIGNAL_D(Cl_child_2::get_signal),
    SIGNAL_D(Cl_child_3::get_signal),          SIGNAL_D(Cl_child_4::get_signal),
```

```
SIGNAL_D(Cl_child_5::get_signal), SIGNAL_D(Cl_child_6::get_signal)};//вектор
сигналов
   vector <TYPE_HANDLER> handlers = {HANDLER_D(Cl_child_2::get_handler),
HANDLER_D(Cl_child_3::get_handler),      HANDLER_D(Cl_child_4::get_handler),
HANDLER_D(Cl_child_5::get_handler),
HANDLER_D(Cl_child_6::get_handler)};//вектор обработчиков
   while (true) {
      getline(cin, command);//чтение команды
      if (command == "END") {//команда end?
         break;
      }
      args.clear();//очистить вектор
      while (command.find(' ') != string::npos) {//пока есть пробел в строке
         args.push_back(command.substr(0, command.find(' ')));//добавление в
конец вектора
         command = command.substr(command.find(' ') + 1);//обрезание строки
до первого пробела
      }
      args.push_back(command);//добаление в конец вектора
      if (args.size() < 3) {//размер меньше 3?
         continue;
      }
      from = get_object_by_coordinate(args.at(1));//извлечение координаты
первого объекта
      to = get_object_by_coordinate(args.at(2));//извлечение координаты
второго объекта
      if (!from) {//объект по координате не определен?
         cout << endl << "Object " << args.at(1) << " not found";//вывод
сообщения об ошибке
         continue;
      }
      if (args.at(0) == "SET_CONDITION") {//команда set_condition?
         from -> change_object_state(atoi(args.at(2).c_str()));//изменение
состояния объекта
         continue;
      }
      if (from -> object_class < 2) {//класс объекта меньше 2?
         continue;
      }
      if (args.at(0) == "EMIT") {//команда emit?
         command = "";//очистка строки текста сообщения
         for (int i = 2; i < args.size(); i++) {//цикл по элементам вектора
со второго
            if (i != args.size() - 1) {//элемент не последний?
               command += args.at(i) + " ";//добавление слова в текст
сообщения и пробела
            }
            else {
               command += args.at(i);//добавление слова в текст сообщения
            }
         }
         from -> emit_signal(signals.at(from -> object_class - 2),
command);//вызов метода выдачи сигнала
      }
      else {
```

```
            if (to) {//объект по координате определен?
               if (args.at(0) == "SET_CONNECT") {//команда set_connect?
                  from -> set_connect(signals.at(from -> object_class - 2), to,
handlers.at(from -> object_class - 2));//установка связи
               }
               else    if    (args.at(0)    ==    "DELETE_CONNECT")    {//команда
delete_connect?
                  from -> delete_connect(signals.at(from -> object_class - 2),
to, handlers.at(from -> object_class - 2));//удаление связи
               }
            }
            else {
               cout  <<  endl  <<  "Handler  object  "  <<  args.at(2)  <<  "  not
found";//Вывод сообщения об ошибке
            }
         }
      }
      return 0;
   }
```

5.2 Файл Cl_application.h

Листинг 2 – Cl_application.h

```
   #ifndef __CL_APPLICATION__H
   #define __CL_APPLICATION__H
   #include "Cl_child_2.h"
   #include "Cl_child_3.h"
   #include "Cl_child_4.h"
   #include "Cl_child_5.h"
   #include "Cl_child_6.h"

   class Cl_application: public Cl_base//наследование класса
   {
   public:
      Cl_application(Cl_base*          p_head_object,          string          =
   "Base_object");//параметризированный конструктор
      void build_tree_objects();//метод построения дерева
      int exec_app();//метод запуска системы
   };

   #endif
```

5.3 Файл Cl_base.cpp

Листинг 3 – Cl_base.cpp

```
#include "Cl_base.h"

Cl_base::Cl_base(Cl_base* p_head_object, string s_object_name)
{
   this -> p_head_object = p_head_object;//присвоение указателя на
родительский объект
   this -> s_object_name = s_object_name;//присвоение имени объекта
   if ( p_head_object ) {//есть родительский объект?
      p_head_object -> subordinate_objects.push_back(this);//добавить в
производные объекты
   }
}

bool Cl_base::change_object_name(string s_object_name)
{
   if (s_object_name.empty()) {//пустая строка?
      return false;
   }
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
      if (subordinate_object -> get_object_name() == s_object_name) {//если
имя равно искомому
         return false;
      }
   }
   this -> s_object_name = s_object_name;//сменить имя
   return true;
}

string Cl_base::get_object_name()
{
   return s_object_name;//вернуть имя объекта
}

Cl_base* Cl_base::get_head_object()
{
   return p_head_object;//вернуть указатель на родительский объект
}

void Cl_base::show_object_tree()
{
   cout << endl;
   Cl_base* head_object = p_head_object;//указатель на головной объект
   while (head_object != nullptr) {//существует головной объект?
      cout << "    ";//отступ
      head_object = head_object -> p_head_object;//обновление головного
объекта
   }
   cout << s_object_name;//вывод имени объекта
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
```

```
подчиненных объектов
      subordinate_object -> show_object_tree();//уход в рекурсию
   }
}

Cl_base* Cl_base::get_sub_object_by_name(string s_object_name)
{
   if (!s_object_name.empty()) {//строка не пустая?
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
         if (subordinate_object -> get_object_name() == s_object_name) {//имя
равно искомому?
            return subordinate_object;//вернуть указатель на подчиненный
объект
         }
      }
   }
   return nullptr;
}

Cl_base* Cl_base::get_branch_object_by_name(string s_object_name)
{
   if (this -> s_object_name == s_object_name) {//строка совпадает с именем
объекта?
      return this;//вернуть объект
   }
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объеков
      if (subordinate_object -> get_object_name() == s_object_name) {//строка
совпадает с именем объекта?
         return subordinate_object;//вернуть объект
      }
   }
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
      if (subordinate_object -> get_branch_object_by_name(s_object_name))
{//есть в ветви такое имя?
         return                    subordinate_object                    ->
get_branch_object_by_name(s_object_name);//вернуть объект, если есть
      }
   }
   return nullptr;
}

Cl_base* Cl_base::get_object_by_name(string s_object_name)
{
   Cl_base* base = this;//указатель на текущий объект
   while (true) {
      if (base -> get_head_object()) {//существует головной объект?
         base = base -> get_head_object();//обновить текущий объект
      }
      else {
         break;
      }
   }
   if (base -> get_branch_object_by_name(s_object_name)) {//есть в дереве
```

```
такое имя?
      return   base   ->   get_branch_object_by_name(s_object_name);//вернуть
объект, если есть
   }
   return nullptr;
}

void Cl_base::show_object_tree_full()
{
   Cl_base* head_object = p_head_object;//указатель на головной объект
   while (head_object != nullptr) {//головной объект существует?
      cout << "    ";//отступ
      head_object  =  head_object  ->  p_head_object;//обновление  головного
объекта
   }
   cout << s_object_name;//вывод имени объекта
   if (object_state != 0) {//вывод состояния
      cout << " is ready" << endl;
   }
   else {
      cout << " is not ready" << endl;
   }
   for  (Cl_base*  subordinate_object  :  subordinate_objects)  {//для  всех
подчиненных объектов
      subordinate_object -> show_object_tree_full();//уход в рекурсию
   }
}

void Cl_base::change_object_state(int object_state)
{
   if (object_state != 0) {//состояние отлично от 0?
      Cl_base* head_object = p_head_object;//указатель на головной объект
      bool f = true;//объявление флага
      while (head_object != nullptr) {//головной объект существует?
         if (head_object -> object_state == 0) {//состояние головного объекта
0?
            f = false;
            break;
         }
         head_object  =  head_object  ->  p_head_object;//обновление  головного
объекта
      }
      if (f) {
         this -> object_state = object_state;//обновление состояния объекта
      }
   }
   else {
      this -> object_state = 0;//обнуление состояния объекта
      for  (Cl_base*  subordinate_object  :  subordinate_objects)  {//для  всех
подчиненных объектов
         subordinate_object -> change_object_state(0);//обнуление
      }
   }
}
```

```
bool Cl_base::change_head_object(Cl_base* p_head_object)
{
   if (!p_head_object || !get_head_object()) {//Создается новый корневой
объект или переопределяется корневой объект?
      return false;
   }
   if (p_head_object -> get_sub_object_by_name(this -> get_object_name()) !=
nullptr) {//При переопределеии появляется дубль?
      return false;
   }
   if (this -> get_branch_object_by_name(p_head_object -> s_object_name) ==
p_head_object) {//Новый головной объектв ветви текущего объекта?
      return false;
   }
   int k = 0;//переменная счетчик
   for     (Cl_base*     subordinate_object     :     get_head_object()     ->
subordinate_objects) {//для всех подчиненных объектов
      if    (subordinate_object    ->    get_object_name()    ==    this    ->
get_object_name()) {//имя объекта равняетя искомому?
         get_head_object()                                                  ->
subordinate_objects.erase(subordinate_objects.begin()     +     k);//удаление
текущего объекта из списка подчиненных
         break;
      }
      else {
         k++;//увеличение счетчика
      }
   }
   this  ->  p_head_object  =  p_head_object;//Переопределение  указателя  на
головной объект
   this -> p_head_object -> subordinate_objects.push_back(this);//Добавление
объекта в список подчиненных
   return true;
}

void Cl_base::delete_sub_object_by_name(string s_object_name)
{
   if (s_object_name.empty()) {//Строка пустая?
      return;
   }
   int k = 0;//переменная счетчик
   for  (Cl_base*  subordinate_object  :  subordinate_objects)  {//для  всех
подчиненных объектов
      if (subordinate_object -> get_object_name() == s_object_name) {//имя
объекта равняетя искомому?
         subordinate_objects.erase(subordinate_objects.begin()              +
k);//удаление текущего объекта из списка подчиненных
         delete subordinate_object;//очистка памяти
         break;
      }
      else {
         k++;//увеличение счетчика
      }
   }
}
```

```
Cl_base* Cl_base::get_object_by_coordinate(string coordinate)
{
   if (!coordinate.empty()) {//координата не пустая?
      Cl_base* base = this;//Указатель на текущий объект
      if (coordinate[0] == '.') {//Кооринатa начинается с точки?
         if (coordinate.length() == 1) {//Длина равна 1?
            return this;//Вернуь текущий объект
         }
         return     get_branch_object_by_name(coordinate.substr(1));//Вернуть
объект найденный по имени на ветке
      }
      if (coordinate[0] == '/') {//Координата начинается со слеша?
         while (true) {
            if (base -> get_head_object()) {//существует головной объект?
               base = base -> get_head_object();//обновить текущий объект
            }
            else {
               break;
            }
         }
         if (coordinate.length() == 1) {//Длина равна 1?
            return base;//Вернуть корневой объект
         }
         if (coordinate[1] == '/') {//Второй элемент координаты равен слеш?
            return                                base                                ->
get_sub_object_by_name(coordinate.substr(2));//Вернуть    объект    найденный
среди подчиненных по имени
         }
         coordinate    =    coordinate.substr(1);//Отрезать    первый    элемент
координаты
      }
      vector<string> names;//массив для записи имен объектов в координате
      while (true) {
         if (coordinate.find('/') != string::npos) {//В координате есть слеш?
            names.push_back(coordinate.substr(0,
coordinate.find('/')));//добавить в массив имя объекта до слеша
            coordinate     =     coordinate.substr(coordinate.find('/')     +
1);//отрезать первое имя и слеш
         }
         else {
            names.push_back(coordinate);//добавить в массив имя объекта
            break;
         }
      }
      for (string name : names) {//для всех элементов массива
         base  =  base  ->  get_sub_object_by_name(name);//обновить  текущий
объект
         if (!base) {//Нет подчиненного с таким именем?
            break;
         }
      }
      return base;//вернуть текущий объект
   }
   return nullptr;
```

```
}

void   Cl_base::set_connect(TYPE_SIGNAL   p_signal,   Cl_base*   p_target,
TYPE_HANDLER p_handler)
{
   for (connection* value : connects) {//для всех элементов массива
      if (value -> p_signal == p_signal && value -> p_target == p_target &&
value -> p_handler == p_handler) {//существует уже такая связь?
         return;
      }
   }
   connection* value = new connection();//новый элемент
   value -> p_signal = p_signal;//присвоение значений свойств элемента
   value -> p_target = p_target;//присвоение значений свойств элемента
   value -> p_handler = p_handler;//присвоение значений свойств элемента
   connects.push_back(value);//добавление в массив связей

}

void   Cl_base::delete_connect(TYPE_SIGNAL   p_signal,   Cl_base*   p_target,
TYPE_HANDLER p_handler)
{
   bool f = false;//флаг
   int index = 0;//индекс
   connection* del;//указатель на удаляемый элемент
   for (connection* value : connects) {//для всех элементов массива
      if (value -> p_signal == p_signal && value -> p_target == p_target &&
value -> p_handler == p_handler) {//существует такая связь?
         f = true;//смена флага
         del = value;//запоминаем элемент
         break;
      }
      else {
         index++;//увеличение индекса
      }
   }
   if (f) {//true?
      connects.erase(connects.begin() + index);//удаление из списка
      delete del;//очистка памяти
   }
}

void Cl_base::emit_signal(TYPE_SIGNAL p_signal, string& message)
{
   if (object_state == 0) {//объект не активирован?
      return;
   }
   TYPE_HANDLER p_handler;//указатель на метод обраюотчика
   Cl_base* p_target;//указатель на целевой объект
   (this ->* p_signal)(message);//вызов метода сигнала
   for (connection* value : connects) {//для всех элементов массива
      if (value -> p_signal == p_signal) {//нужный нам сигнал?
         p_target = value -> p_target;//извлечение указател на целевой объект
         p_handler  =  value  ->  p_handler;//извлечение  указателя  на  метод
обработчика
```

```
            if (p_target -> object_state != 0) {//объект активирован?
               (p_target ->* p_handler)(message);//вызов метода обработчика
            }
         }
      }
   }

   string Cl_base::get_absolute_coordinate()
   {
      Cl_base* object = this;//указаеь на текущий объект
      string absolute = "";//строка координаты
      while (object -> get_head_object())//пока есть головной объект
      {
         absolute = "/" + object -> get_object_name() + absolute;//добавлене
   имени объекта в координату
         object = object -> get_head_object();//обновление объекта
      }
      if (absolute.empty()) {//строка пустая?
         return "/";
      }
      return absolute;
   }
```

5.4 Файл Cl_base.h

Листинг 4 – Cl_base.h

```
   #ifndef __CL_BASE__H
   #define __CL_BASE__H
   #include <iostream>
   #include <vector>
   #include <string>
   #define SIGNAL_D(signal_f) (TYPE_SIGNAL)(&signal_f)
   #define HANDLER_D(handler_f) (TYPE_HANDLER)(&handler_f)

   using namespace std;

   class Cl_base;
   typedef void(Cl_base::*TYPE_SIGNAL)(string&);//тип сигнала
   typedef void(Cl_base::*TYPE_HANDLER)(string);//тип обработчика
   struct connection//структура связи
   {
      TYPE_SIGNAL p_signal;//указатель на метод сигнала
      Cl_base* p_target;//указатель на целевой объект
      TYPE_HANDLER p_handler;//указатель на метод обработчика
   };

   class Cl_base//наименование класса
   {
   public:
      Cl_base    (Cl_base*    p_head_object,    string    s_object_name    =
```

```
"Base_object");//параметризированный конструктор
   bool change_object_name(string);//метод изменения имени
   string get_object_name();//метод получения имени
   Cl_base*  get_head_object();//метод  получения  указателя  на  родительский
объект
   Cl_base*   get_sub_object_by_name(string);//метод   поиска   подчиненного
объекта по имени

   Cl_base* get_branch_object_by_name(string);//метод поиска объекта на ветке
по имени
   Cl_base* get_object_by_name(string);//метод поиска объекта по имени
   void show_object_tree();//метод вывода дерева объектов
   void show_object_tree_full();//метод вывода дерева объектов и состояния
   void change_object_state(int);//метод установки состояния

   bool   change_head_object(Cl_base*);//метод   переопределения   головного
объекта
   void   delete_sub_object_by_name(string);//метод   удаления   подчиненного
объекта по наименованию
   Cl_base*  get_object_by_coordinate(string);//метод  получения  указателя  на
объект по его координате

   void  set_connect(TYPE_SIGNAL,  Cl_base*,  TYPE_HANDLER);//метод  установки
связи
   void delete_connect(TYPE_SIGNAL, Cl_base*, TYPE_HANDLER);//метод удаления
связи
   void emit_signal(TYPE_SIGNAL, string&);//метод выдачи синала
   string get_absolute_coordinate();//метод получения абсолютной координаты
   int object_class = 1;//класс объекта

private:
   int object_state = 1;//состояние объекта
   string s_object_name;//имя объекта
   Cl_base* p_head_object;//указатель на родительский объект
   vector <Cl_base*> subordinate_objects;//подчиненные объекты
   vector <connection*> connects;//установленные связи
};


#endif
```

5.5 Файл Cl_child_2.cpp

Листинг 5 – Cl_child_2.cpp

```
#include "Cl_child_2.h"

Cl_child_2::Cl_child_2(Cl_base*            p_head_object,            string
s_object_name):Cl_base(p_head_object, s_object_name)
{
```

```
   object_class = 2;//установка номера класса
}

void Cl_child_2::get_signal(string& message)
{
   cout << endl << "Signal from " << get_absolute_coordinate();//вывод
соощения с координатой
   message += " (class: " + to_string(object_class) + ")";//изменение
текстовой строки
}

void Cl_child_2::get_handler(string message)
{
   cout << endl << "Signal to " << get_absolute_coordinate() << " Text:  " <<
message;//вывод сообщения с координатой
}
```

5.6 Файл Cl_child_2.h

Листинг 6 – Cl_child_2.h

```
#ifndef __CL_CHILD__H
#define __CL_CHILD__H
#include "Cl_base.h"

class Cl_child_2 : public Cl_base//наследование класса
{
public:
   Cl_child_2(Cl_base*, string);//параметризированный конструктор

   void get_signal(string&);//метод сигнала
   void get_handler(string);//метод обработчика
};

#endif
```

5.7 Файл Cl_child_3.cpp

Листинг 7 – Cl_child_3.cpp

```
#include "Cl_child_3.h"

Cl_child_3::Cl_child_3(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{
```

```
   object_class = 3;//установка номера класса
}

void Cl_child_3::get_signal(string& message)
{
   cout << endl << "Signal from " << get_absolute_coordinate();//вывод
соощения с координатой
   message += " (class: " + to_string(object_class) + ")";//изменение
текстовой строки
}

void Cl_child_3::get_handler(string message)
{
   cout << endl << "Signal to " << get_absolute_coordinate() << " Text:  " <<
message;//вывод соощения с координатой
}
```

5.8 Файл Cl_child_3.h

Листинг 8 – Cl_child_3.h

```
#ifndef __CL_CHILD_3__H
#define __CL_CHILD_3__H
#include "Cl_base.h"

class Cl_child_3 : public Cl_base//наследование класса
{
public:
   Cl_child_3(Cl_base*, string);//параметризированный конструктор

   void get_signal(string&);//метод сигнала
   void get_handler(string);//метод обработчика
};


#endif
```

5.9 Файл Cl_child_4.cpp

Листинг 9 – Cl_child_4.cpp

```
#include "Cl_child_4.h"

Cl_child_4::Cl_child_4(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
```

```
{
   object_class = 4;//установка номера класса
}

void Cl_child_4::get_signal(string& message)
{
   cout << endl << "Signal from " << get_absolute_coordinate();//вывод
сообщения с координатой
   message += " (class: " + to_string(object_class) + ")";//изменение
текстовой строки
}

void Cl_child_4::get_handler(string message)
{
   cout << endl << "Signal to " << get_absolute_coordinate() << " Text:  " <<
message;//вывод сообщения с координатой
}
```

5.10 Файл Cl_child_4.h

Листинг 10 – Cl_child_4.h

```
#ifndef __CL_CHILD_4__H
#define __CL_CHILD_4__H
#include "Cl_base.h"

class Cl_child_4 : public Cl_base//наследование класса
{
public:
   Cl_child_4(Cl_base*, string);//параметризированный конструктор

   void get_signal(string&);//метод сигнала
   void get_handler(string);//метод обработчика
};


#endif
```

5.11 Файл Cl_child_5.cpp

Листинг 11 – Cl_child_5.cpp

```
#include "Cl_child_5.h"

Cl_child_5::Cl_child_5(Cl_base*            p_head_object,            string
```

```
s_object_name):Cl_base(p_head_object, s_object_name)
{
   object_class = 5;//установка номера класса
}

void Cl_child_5::get_signal(string& message)
{
   cout << endl << "Signal from " << get_absolute_coordinate();//вывод
соощения с координатой
   message += " (class: " + to_string(object_class) + ")";//изменение
текстовой строки
}

void Cl_child_5::get_handler(string message)
{
   cout << endl << "Signal to " << get_absolute_coordinate() << " Text:  " <<
message;//вывод соощения с координатой
}
```

5.12 Файл Cl_child_5.h

Листинг 12 – Cl_child_5.h

```
#ifndef __CL_CHILD_5__H
#define __CL_CHILD_5__H
#include "Cl_base.h"

class Cl_child_5 : public Cl_base//наследование класса
{
public:
   Cl_child_5(Cl_base*, string);//параметризированный конструктор

   void get_signal(string&);//метод сигнала
   void get_handler(string);//метод обработчика
};


#endif
```

5.13 Файл Cl_child_6.cpp

Листинг 13 – Cl_child_6.cpp

```
#include "Cl_child_6.h"

Cl_child_6::Cl_child_6(Cl_base*            p_head_object,            string
```

```
s_object_name):Cl_base(p_head_object, s_object_name)
{
   object_class = 6;//установка номера класса
}

void Cl_child_6::get_signal(string& message)
{
   cout << endl << "Signal from " << get_absolute_coordinate();//вывод
соощения с координатой
   message += " (class: " + to_string(object_class) + ")";//изменение
текстовой строки
}

void Cl_child_6::get_handler(string message)
{
   cout << endl << "Signal to " << get_absolute_coordinate() << " Text:  " <<
message;//вывод соощения с координатой
}
```

5.14 Файл Cl_child_6.h

Листинг 14 – Cl_child_6.h

```
#ifndef __CL_CHILD_6__H
#define __CL_CHILD_6__H
#include "Cl_base.h"

class Cl_child_6 : public Cl_base//наследование класса
{
public:
   Cl_child_6(Cl_base*, string);//параметризированный конструктор

   void get_signal(string&);//метод сигнала
   void get_handler(string);//метод обработчика
};


#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

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

```
#include "Cl_application.h"

int main()
{
   Cl_application ob_cl_application(nullptr);//создание объекта приложения
   ob_cl_application.build_tree_objects();//конструирование системы
   return ob_cl_application.exec_app();//запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 24.

Таблица 24 – 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 / /object_s1 / end_of_connections EMIT /object_s2/object_s4 /5 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 SET_CONDITION /object_s1 1 SET_CONDITION /object_s1 0 DELETE_CONNECT /object_s2/object_s4 /object_s2/object_s6 DELETE_CONNECT /object_s2/object_s4 /5 /object_s2/object_s6 DELETE_CONNECT /object_s2/object_s4 /object_s2/object_s6 /5 SET_CONNECT /object_s2/object_s4 /object_s2/object_s6 SET_CONNECT /object_s2/object_s4 /5 /object_s2/object_s6 SET_CONNECT /object_s2/object_s4 /object_s2/object_s6 /5 END	object_s6 object_s13 Object /object_s2/object_s4 /5 not found Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 Signal to / Text: Send message 4 (class: 3) Object /object_s2/object_s4 /5 not found Handler object /object_s2/object_s6 /5 not found Object /object_s2/object_s4 /5 not found Handler object /object_s2/object_s6 /5 not found	object_s6 object_s13 Object /object_s2/object_s4 /5 not found Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1 Signal to / Text: Send message 4 (class: 3) Object /object_s2/object_s4 /5 not found Handler object /object_s2/object_s6 /5 not found Object /object_s2/object_s4 /5 not found Handler object /object_s2/object_s6 /5 not found

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).