



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»
(наименование дисциплины)

Тема курсовой работы К_34 Моделирование работы автозаправочной станции
(наименование темы)

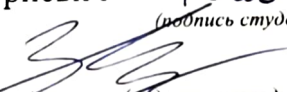
Студент группы ИКБО-08-22
(учебная группа)

Сенькевич Гаврил Дмитриевич
(Фамилия Имя Отчество)


(подпись студента)

Руководитель курсовой работы

доцент Путуридзе З.Ш.
(Должность, звание, ученая степень)


(подпись руководителя)

Консультант

преподаватель Даев А.В.
(Должность, звание, ученая степень)


(подпись консультанта)

Работа представлена к защите «20» мая 2023 г.

Допущен к защите «20» мая 2023 г.

4 (хот)


Москва 2023 г.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой

Подпись

Платонова О.В.

ФИО

«21» февраля 2023 г.

ЗАДАНИЕ

На выполнение курсовой работы

по дисциплине «Объектно-ориентированное программирование»

Студент Сенькевич Гаврил Дмитриевич Группа ИКБО-08-22

Тема К 34 Моделирование работы автозаправочной станции

Исходные данные:

1. Описания исходной иерархии дерева объектов.
2. Описание схемы взаимодействия объектов.
3. Множество команд для управления функционированием моделируемой системы.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Построение версий программ.
2. Построение и работа с деревом иерархии объектов.
3. Взаимодействия объектов посредством интерфейса сигналов и обработчиков.
4. Блок-схемы алгоритмов.
5. Управление функционированием моделируемой системы

Срок представления к защите курсовой работы: до «20» мая 2023 г.

Задание на курсовую работу выдал

(Путуридзе З.И.)

ФИО консультанта

«21» февраля 2023 г.

Задание на курсовую работу получил

(Сенькевич Г.Д.)

ФИО исполнителя

«21» февраля 2023 г.

Москва 2023 г.

ОТЗЫВ

на курсовую работу

по дисциплине «Объектно-ориентированное программирование»

Студент Сенькевич Гаврил Дмитриевич группа ИКБО-08-22
(ФИО студента) (Группа)

Характеристика курсовой работы

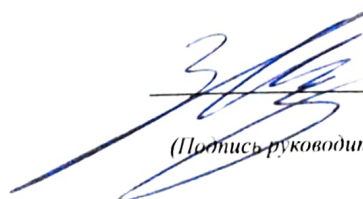
Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме	✓		
2. Соответствие курсовой работы заданию	✓		
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.	✓		
4. Полнота выполнения всех пунктов задания			✓
5. Логичность и системность содержания курсовой работы	✓		
6. Отсутствие фактических грубых ошибок	✓		

Замечаний:

Не все пункты выполнены

Рекомендуемая оценка:

4 (хорошо)



доцент Путуридзе З.Ш.

(Подпись руководителя)

(ФИО руководителя)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1 ПОСТАНОВКА ЗАДАЧИ.....	9
1.1 Описание входных данных.....	11
1.2 Описание выходных данных.....	13
2 МЕТОД РЕШЕНИЯ.....	15
3 ОПИСАНИЕ АЛГОРИТМОВ.....	23
3.1 Алгоритм функции main.....	23
3.2 Алгоритм метода get_coordinate класса cl_base.....	23
3.3 Алгоритм метода get_class_number класса cl_base.....	24
3.4 Алгоритм метода set_class_number класса cl_base.....	25
3.5 Алгоритм метода add_connection класса cl_base.....	25
3.6 Алгоритм метода delete_connection класса cl_base.....	26
3.7 Алгоритм метода emit_signal класса cl_base.....	27
3.8 Алгоритм метода signal класса cl_base.....	29
3.9 Алгоритм метода handler класса cl_base.....	29
3.10 Алгоритм метода get_signal класса cl_application.....	30
3.11 Алгоритм метода get_handler класса cl_application.....	31
3.12 Алгоритм метода activate класса cl_base.....	32
3.13 Алгоритм метода signal класса cl_2.....	33
3.14 Алгоритм метода handler класса cl_2.....	33
3.15 Алгоритм метода signal класса cl_3.....	33
3.16 Алгоритм метода handler класса cl_3.....	34
3.17 Алгоритм метода signal класса cl_4.....	34
3.18 Алгоритм метода handler класса cl_4.....	35
3.19 Алгоритм метода signal класса cl_5.....	35
3.20 Алгоритм метода handler класса cl_5.....	36

3.21 Алгоритм метода signal класса cl_6.....	36
3.22 Алгоритм метода handler класса cl_6.....	37
3.23 Алгоритм метода build_tree_objects класса cl_application.....	37
3.24 Алгоритм метода exes_app класса cl_application.....	40
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....	44
5 КОД ПРОГРАММЫ.....	82
5.1 Файл cl_2.cpp.....	82
5.2 Файл cl_2.h.....	82
5.3 Файл cl_3.cpp.....	83
5.4 Файл cl_3.h.....	83
5.5 Файл cl_4.cpp.....	84
5.6 Файл cl_4.h.....	84
5.7 Файл cl_5.cpp.....	85
5.8 Файл cl_5.h.....	85
5.9 Файл cl_6.cpp.....	86
5.10 Файл cl_6.h.....	86
5.11 Файл cl_application.cpp.....	87
5.12 Файл cl_application.h.....	91
5.13 Файл cl_base.cpp.....	92
5.14 Файл cl_base.h.....	100
5.15 Файл main.cpp.....	102
6 ТЕСТИРОВАНИЕ.....	103
ЗАКЛЮЧЕНИЕ.....	104
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	106

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].

Данная курсовая работа направлена на развитие навыков проектирования и разработки программного обеспечения с использованием объектно-ориентированного подхода.

Цель работы заключается в создании иерархии объектов, взаимодействии между ними с помощью сигналов и обработчиков, а также реализации дополнительных функций, таких как установление связей и определение абсолютного пути объекта на дереве иерархии.

Первая часть работы предполагает разработку базового класса, который будет содержать функционал и свойства для построения иерархии объектов. Этот класс будет использоваться в последующих частях работы в качестве базового для создания других классов.

Во второй части работы необходимо реализовать механизм ввода иерархического дерева объектов, основываясь на входных данных. При вводе проверяется уникальность имен объектов и корректность номера класса.

Третья часть работы расширяет функциональность базового класса и добавляет возможность доступа к объектам системы из текущего объекта. Здесь реализуется метод переопределения головного объекта в дереве иерархии, метод удаления подчиненного объекта по его имени, а также метод получения указателя

на любой объект в дереве иерархии согласно заданному пути (координате).

Четвертая и последняя часть работы посвящена реализации механизма взаимодействия объектов с использованием сигналов и обработчиков. В базовый класс добавляются методы установления связи между сигналом текущего объекта и обработчиком целевого объекта, удаления связи и выдачи сигнала с передачей текстового сообщения. При выдаче сигнала происходит вызов методов обработчиков связанных объектов, если они готовы к приему сигнала.

Работа над данной курсовой работой позволяет погрузиться в процесс разработки иерархической системы объектов, а также ознакомиться с принципами взаимодействия объектов через сигналы и обработчики. Результатом выполненной работы будет полнофункциональная система, демонстрирующая основные принципы ООП и возможности взаимодействия между объектами.

В контексте данной курсовой работы, объектно-ориентированное программирование представляет собой методологию разработки программного обеспечения, в которой основной упор делается на объекты как основные строительные блоки системы. Это позволяет создавать более гибкие, модульные и легко поддерживаемые программы.

Объектно-ориентированное программирование (ООП) — это парадигма программирования, которая организует разработку программного обеспечения вокруг объектов, которые объединяют данные и функциональность в одну сущность. ООП предоставляет набор концепций и инструментов для создания модульных, гибких и легко поддерживаемых программных систем.

Преимущества ООП:

1. Модульность и повторное использование кода: ООП позволяет разбивать программу на независимые модули (классы), которые могут быть повторно использованы в различных частях программы или даже в других проектах. Это способствует улучшению производительности и сокращению времени разработки.

2. Инкапсуляция и скрытие данных: ООП позволяет объединить данные и методы, работающие с этими данными, в одном объекте. Это способствует защите данных от неправильного использования и обеспечивает контролируемый доступ к ним через интерфейс объекта.

3. Наследование и расширяемость: ООП поддерживает концепцию наследования, которая позволяет создавать новые классы на основе существующих, наследуя их свойства и функциональность. Это позволяет расширять функциональность программы без необходимости изменения исходного кода.

4. Полиморфизм и гибкость: ООП позволяет использовать полиморфизм, то есть возможность обрабатывать объекты различных типов с использованием общего интерфейса. Это способствует созданию гибких и масштабируемых систем, где можно легко заменять или расширять функциональность объектов.

ООП применяется на практике в различных областях разработки программного обеспечения, таких как веб-разработка, мобильные приложения, игровая индустрия, научные исследования и другие. Множество популярных языков программирования, таких как Java, C++, Python, C#, разработаны на основе принципов ООП.

ООП является важным инструментом, освоение которого помогает развить навыки анализа, проектирования и разработки программного обеспечения.

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 параметризованное 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 дуближа 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:
 - о построение дерева иерархии объектов согласно вводу;
 - о ввод и построение множества связей сигнал-обработчик для заданных пар объектов.
- в методе отработки системы:
 - о привести все объекты в состоянии готовности;
 - о 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.
 - о 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 используются:

- объект стандартного потока ввода `std::cin` (используется для ввода с клавиатуры);
- объект стандартного потока вывода `std::cout` (используется для вывода на экран);
- оператор цикла со счётчиком `for`;
- оператор цикла с условием `while`;
- условный оператор `if..else`;
- объекты классов `cl_2`, `cl_3`, `cl_4`, `cl_5`, `cl_6`, `cl_application`.

Структура `connection`:

- Свойства (поля):
 - Указатель на метод сигнала `signal` (тип - `TYPE_SIGNAL`);
 - Указатель на целевой объект `target` (тип - указатель на объект класса `cl_base`);
 - Указатель на метод-обработчик `handler` (тип - `TYPE_HANDLER`).

Определение новых типов данных:

- `TYPE_SIGNAL` - указатель на метод сигнала;
- `TYPE_HANDLER` - указатель на метод обработчика.

Макрос `SIGNAL_D`:

- Функционал - возврат указателя на переданный метод сигнала, приведённый к типу `TYPE_SIGNAL`;
- Возвращаемое значение - указатель на метод сигнала, приведённый к типу `TYPE_SIGNAL`;
- Параметры:
 - `signal_f` - метод сигнала.

Макрос HANDLER_D:

- Функционал - возврат указателя на переданный метод обработчика, приведённый к типу TYPE_HANDLER;
- Возвращаемое значение - указатель на метод обработчика, приведённый к типу TYPE_HANDLER;
- Параметры:
 - handler_f - метод обработчика.

Класс cl_base:

- Свойства (поля):
 - Поле, отвечающее за номер класса объекта:
 - Наименование - class_number;
 - Тип - целое число;
 - Модификатор доступа - закрытый;
 - Поле, хранящее список связей объекта:
 - Наименование - connections;
 - Тип - вектор структур connection;
 - Модификатор доступа - закрытый;
- Методы:
 - Метод get_coordinate:
 - Функционал - возврат координаты объекта в дереве иерархии;
 - Возвращаемое значение - строка;
 - Модификатор доступа - открытый;
 - Параметры - отсутствуют;
 - Метод get_class_number:
 - Функционал - возврат значения поля class_number;
 - Возвращаемое значение - целочисленное значение поля класса;

- Модификатор доступа - открытый;
- Параметры - отсутствуют;
- Метод `set_class_number`:
 - Функционал - установка значения поля `class_number`;
 - Возвращаемое значение - отсутствует (`void`);
 - Модификатор доступа - открытый;
 - Параметры:
 - `number` - номер класса (целое число);
- Метод `add_connection`:
 - Функционал - установка связи с объектом, переданным в качестве параметра;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - `signal` - указатель на метод сигнала (`TYPE_SIGNAL`);
 - `target` - указатель на целевой объект (указатель на `cl_base`);
 - `handler` - указатель на метод обработчика (`TYPE_HANDLER`);
- Метод `delete_connection`:
 - Функционал - удаление связи с объектом, переданным в качестве параметра;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - `signal` - указатель на метод сигнала (`TYPE_SIGNAL`);
 - `target` - указатель на целевой объект (указатель на

cl_base);

- handler - указатель на метод обработчика (TYPE_HANDLER);

o Метод emit_signal:

- Функционал - передача сообщения по связи;
- Возвращаемое значение - отсутствует
- Модификатор доступа - открытый;
- Параметры:
 - signal - указатель на метод сигнала (TYPE_SIGNAL);
 - command - сообщение (строка);

o Метод signal:

- Функционал - метод сигнала объекта;
- Возвращаемое значение - отсутствует
- Модификатор доступа - открытый;
- Параметры:
 - message - сообщение (строка);

o Метод handler:

- Функционал - метод обработчика объекта;
- Возвращаемое значение - отсутствует
- Модификатор доступа - открытый;
- Параметры:
 - message - сообщение (строка);

o Метод get_signal:

- Функционал - возврат указателя на метод сигнала объекта;
- Возвращаемое значение - указатель на метод сигнала объекта (TYPE_SIGNAL);
- Модификатор доступа - открытый;

- Параметры:
 - target - указатель на целевой объект (указатель на cl_base);
- o Метод get_handler:
 - Функционал - возврат указателя на метод обработчика объекта;
 - Возвращаемое значение - указатель на метод обработчика объекта (TYPE_HANDLER);
 - Модификатор доступа - открытый;
 - Параметры:
 - target - указатель на целевой объект (указатель на cl_base);
- o Метод activate:
 - Функционал - установка объекта и всех его подчинённых ниже по дереву иерархии в состояние готовности;
 - Возвращаемое значение - отсутствует;
 - Модификатор доступа - открытый;
 - Параметры - отсутствует.

Класс cl_2:

- Методы:
 - o Метод signal:
 - Функционал - метод сигнала объекта;
 - Возвращаемое значение - отсутствует;
 - Модификатор доступа - открытый;
 - Параметры:
 - message - сообщение (строка);
 - o Метод handler:

- Функционал - метод обработчика объекта;
- Возвращаемое значение - отсутствует
- Модификатор доступа - открытый;
- Параметры:
 - message - сообщение (строка).

Класс cl_3:

- Методы:
 - ο Метод signal:
 - Функционал - метод сигнала объекта;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - message - сообщение (строка);
 - ο Метод handler:
 - Функционал - метод обработчика объекта;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - message - сообщение (строка).

Класс cl_4:

- Методы:
 - ο Метод signal:
 - Функционал - метод сигнала объекта;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - message - сообщение (строка);

- o Метод handler:
 - Функционал - метод обработчика объекта;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - message - сообщение (строка).

Класс cl_5:

- Методы:
 - o Метод signal:
 - Функционал - метод сигнала объекта;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - message - сообщение (строка);
 - o Метод handler:
 - Функционал - метод обработчика объекта;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - message - сообщение (строка).

Класс cl_6:

- Методы:
 - o Метод signal:
 - Функционал - метод сигнала объекта;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:

- message - сообщение (строка);
- o Метод handler:
 - Функционал - метод обработчика объекта;
 - Возвращаемое значение - отсутствует
 - Модификатор доступа - открытый;
 - Параметры:
 - message - сообщение (строка).

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм функции `main`

Функционал: основной алгоритм программы.

Параметры: отсутствуют.

Возвращаемое значение: целочисленный код завершения работы программы.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции `main`

№	Предикат	Действия	№ перехода
1		Создание объекта <code>ob_cl_application</code> класса <code>cl_application</code> с использованием параметризованного конструктора и нулевого указателя (<code>nullptr</code>) в качестве параметра	2
2		Вызов метода <code>build_tree_objects</code> объекта <code>ob_cl_application</code>	3
3		Возвращение значения, возвращённого методом <code>exes_app</code> класса <code>ob_cl_application</code>	Ø

3.2 Алгоритм метода `get_coordinate` класса `cl_base`

Функционал: возврат координаты объекта в дереве иерархии.

Параметры: отсутствуют.

Возвращаемое значение: строка.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода *get_coordinate* класса *cl_base*

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной <i>coordinate</i>	2
2		Инициализация указателя на объект класса <i>cl_base</i> <i>curr</i> значением указателя <i>this</i>	3
3	Значение поля <i>parent</i> объекта <i>curr</i> не равно нулевому указателю (<i>nullptr</i>)		4
			6
4		Присваивание переменной <i>coordinate</i> суммы строки <i>"/"</i> , значения поля <i>name</i> объекта <i>curr</i> и значения переменной <i>coordinate</i>	5
5		Присваивание переменной <i>curr</i> значением поля <i>parent</i> объекта <i>curr</i>	3
6	Значение, возвращённое методом <i>length</i> переменной <i>coordinate</i> равно 0	Присваивание переменной <i>coordinate</i> значения <i>"/"</i>	7
			7
7		Возврат значения переменной <i>coordinate</i>	∅

3.3 Алгоритм метода *get_class_number* класса *cl_base*

Функционал: возврат значения поля *class_number*.

Параметры: отсутствуют.

Возвращаемое значение: целочисленное значение поля класса.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода *get_class_number* класса *cl_base*

№	Предикат	Действия	№ перехода
1		Возврат значения поля <i>class_number</i>	∅

3.4 Алгоритм метода `set_class_number` класса `cl_base`

Функционал: установка значения поля `class_number`.

Параметры: `number` - номер класса (целое число).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода `set_class_number` класса `cl_base`

№	Предикат	Действия	№ перехода
1		Присваивание полю <code>class_number</code> значения параметра <code>number</code>	Ø

3.5 Алгоритм метода `add_connection` класса `cl_base`

Функционал: установка связи с объектом, переданным в качестве параметра.

Параметры: `signal` - указатель на метод сигнала; `target` - указатель на целевой объект; `handler` - указатель на метод обработчика.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода `add_connection` класса `cl_base`

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной <code>i</code> значением 0	2
2	Значение переменной <code>i</code> меньше значения, возвращённого методом <code>size</code> поля <code>connections</code>		3
			4
3	Значение поля <code>signal</code> элемента поля <code>connections</code> с		Ø

№	Предикат	Действия	№ перехода
	индексом <i>i</i> равно значению параметра <i>signal</i> и значение поля <i>target</i> элемента поля <i>connections</i> с индексом <i>i</i> равно значению параметра <i>target</i> и значение поля <i>handler</i> элемента поля <i>connections</i> с индексом <i>i</i> равно значению параметра <i>handler</i>		
		Увеличение значения переменной <i>i</i> на 1	2
4		Инициализация указателя на объект структуры <i>connection</i> со адресом нового объекта структуры <i>connection</i>	5
5		Присваивание полю <i>signal</i> объекта <i>con</i> значения параметра <i>signal</i>	6
6		Присваивание полю <i>target</i> объекта <i>con</i> значения параметра <i>target</i>	7
7		Присваивание полю <i>handler</i> объекта <i>con</i> значения параметра <i>handler</i>	8
8		Вызов метода <i>push_back</i> поля <i>connections</i> со значением указателя <i>con</i> в качестве параметра	∅

3.6 Алгоритм метода *delete_connection* класса *cl_base*

Функционал: удаление связи с объектом, переданным в качестве параметра.

Параметры: *signal* - указатель на метод сигнала; *target* - указатель на целевой объект; *handler* - указатель на метод обработчика.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода *delete_connection* класса *cl_base*

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной <i>i</i> значением 0	2
2	Значение переменной <i>i</i> меньше значения, возвращённого методом <i>size</i> поля <i>connections</i>		3
			∅
3	Значение поля <i>signal</i> элемента поля <i>connections</i> с индексом <i>i</i> равно значению параметра <i>signal</i> и значение поля <i>target</i> элемента поля <i>connections</i> с индексом <i>i</i> равно значению параметра <i>target</i> и значение поля <i>handler</i> элемента поля <i>connections</i> с индексом <i>i</i> равно значению параметра <i>handler</i>	Вызов метода <i>erase</i> поля <i>connections</i> со значением суммы значения, возвращённого методом <i>begin</i> поля <i>connections</i> и 1	∅
		Увеличение значения переменной <i>i</i> на 1	2

3.7 Алгоритм метода *emit_signal* класса *cl_base*

Функционал: передача сообщения по связи.

Параметры: *signal* - указатель на метод сигнала; *command* - сообщение.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода *emit_signal* класса *cl_base*

№	Предикат	Действия	№ перехода
1	Значение параметра <i>state</i> равно 0		∅
			2
2		Объявление указателя на метод обработчика <i>handler</i>	3
3		Объявление указателя на объект класса <i>cl_base</i> <i>target</i>	4
4		Вызов метода, на который указывает указатель <i>signal</i> текущего объекта с значением параметра <i>command</i> в качестве аргумента	5
5		Инициализация целочисленной переменной <i>i</i> значением 0	6
6	Значение переменной <i>i</i> меньше значения, возвращённого методом <i>size</i> поля <i>connections</i>		7
			∅
7	Значение поля <i>signal</i> элемента поля <i>connections</i> с индексом <i>i</i> равно значению параметра <i>signal</i> и поле <i>state</i> поля <i>target</i> элемента поля <i>connections</i> с индексом <i>i</i> равно 0		8
			11
8		Присваивание полю <i>handler</i> значения поля <i>handler</i> элемента поля <i>connections</i> с индексом <i>i</i>	9
9		Присваивание полю <i>target</i> значения поля <i>target</i>	10

№	Предикат	Действия	№ перехода
		элемента поля connections с индексом i	
10		Вызов метода, на который указывает указатель handler объекта target со значением параметра command в качестве аргумента	11
11		Увеличение значения параметра i на 1	6

3.8 Алгоритм метода signal класса cl_base

Функционал: метод сигнала объекта.

Параметры: message - сообщения (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода signal класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal from " и значения, возвращённого методом get_coordinate	2
2		Присваивание параметру message значения суммы значения параметра message и строки " (class: 1)"	∅

3.9 Алгоритм метода handler класса cl_base

Функционал: метод обработчика объекта.

Параметры: message - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода handler класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal to ", значения, возвращённого методом get_coordinate, строки " Text: " и значения параметра message	Ø

3.10 Алгоритм метода get_signal класса cl_application

Функционал: возврат указателя на метод сигнала объекта.

Параметры: target - указатель на целевой объект (указатель на объект cl_base).

Возвращаемое значение: указатель на метод сигнала объекта.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода get_signal класса cl_application

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной target_class_number значением, возвращённым методом get_class_number объекта target	2
2	Значение переменной target_class_number равно 1	Возврат указателя на метод signal класса cl_base	Ø
			3
3	Значение переменной target_class_number равно 2	Возврат указателя на метод signal класса cl_2	Ø
			4
4	Значение переменной target_class_number равно 3	Возврат указателя на метод signal класса cl_3	Ø
			5
5	Значение переменной target_class_number равно 4	Возврат указателя на метод signal класса cl_4	Ø
			6

№	Предикат	Действия	№ перехода
6	Значение переменной target_class_number равно 5	Возврат указателя на метод signal класса cl_5	Ø
			7
7	Значение переменной target_class_number равно 6	Возврат указателя на метод signal класса cl_6	Ø
		Возврат значения нулевого указателя	Ø

3.11 Алгоритм метода get_handler класса cl_application

Функционал: возврат указателя на метод обработчика объекта.

Параметры: target - указатель на целевой объект (указатель на cl_base).

Возвращаемое значение: указатель на метод обработчика объекта.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода get_handler класса cl_application

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной target_class_number значением, возвращённым методом get_class_number объекта target	2
2	Значение переменной target_class_number равно 1	Возврат указателя на метод handler класса cl_base	Ø
			3
3	Значение переменной target_class_number равно 2	Возврат указателя на метод handler класса cl_2	Ø
			4
4	Значение переменной target_class_number равно 3	Возврат указателя на метод handler класса cl_3	Ø
			5
5	Значение переменной	Возврат указателя на метод handler класса cl_4	Ø

№	Предикат	Действия	№ перехода
	target_class_number равно 4		
			6
6	Значение переменной target_class_number равно 5	Возврат указателя на метод handler класса cl_5	∅
			7
7	Значение переменной target_class_number равно 6	Возврат указателя на метод handler класса cl_6	∅
		Возврат значения нулевого указателя	∅

3.12 Алгоритм метода activate класса cl_base

Функционал: установка объекта и всех его подчинённых ниже по дереву иерархии в состояние готовности.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода activate класса cl_base

№	Предикат	Действия	№ перехода
1		Вызов метода set_state со значением 1 в качестве параметра	2
2		Инициализация целочисленной переменной i значением 0	3
3	Значение переменной i меньше значения, возвращённого методом size поля children	Вызов метода activate элемента поля children с индексом i	4
			∅
4		Увеличение значения переменной i на 1	3

3.13 Алгоритм метода **signal** класса **cl_2**

Функционал: метод сигнала объекта.

Параметры: message - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода *signal* класса *cl_2*

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal from " и значения, возвращённого методом <code>get_coordinate</code>	2
2		Присваивание параметру <code>message</code> значения суммы значения параметра <code>message</code> и строки " (class: 2)"	Ø

3.14 Алгоритм метода **handler** класса **cl_2**

Функционал: метод обработчика объекта.

Параметры: message - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода *handler* класса *cl_2*

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal to ", значения, возвращённого методом <code>get_coordinate</code> , строки " Text: " и значения параметра <code>message</code>	Ø

3.15 Алгоритм метода **signal** класса **cl_3**

Функционал: метод сигнала объекта.

Параметры: message - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода *signal* класса *cl_3*

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal from " и значения, возвращённого методом <code>get_coordinate</code>	2
2		Присваивание параметру <code>message</code> значения суммы значения параметра <code>message</code> и строки " (class: 3)"	Ø

3.16 Алгоритм метода *handler* класса *cl_3*

Функционал: метод обработчика объекта.

Параметры: `message` - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода *handler* класса *cl_3*

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal to ", значения, возвращённого методом <code>get_coordinate</code> , строки " Text: " и значения параметра <code>message</code>	Ø

3.17 Алгоритм метода *signal* класса *cl_4*

Функционал: метод сигнала объекта.

Параметры: `message` - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода *signal* класса *cl_4*

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal from " и значения, возвращённого методом <i>get_coordinate</i>	2
2		Присваивание параметру <i>message</i> значения суммы значения параметра <i>message</i> и строки " (class: 4)"	Ø

3.18 Алгоритм метода *handler* класса *cl_4*

Функционал: метод обработчика объекта.

Параметры: *message* - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода *handler* класса *cl_4*

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal to ", значения, возвращённого методом <i>get_coordinate</i> , строки " Text: " и значения параметра <i>message</i>	Ø

3.19 Алгоритм метода *signal* класса *cl_5*

Функционал: метод сигнала объекта.

Параметры: *message* - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода *signal* класса *cl_5*

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal from " и значения, возвращённого методом <i>get_coordinate</i>	2

№	Предикат	Действия	№ перехода
2		Присваивание параметру message значения суммы значения параметра message и строки " (class: 5)"	Ø

3.20 Алгоритм метода handler класса cl_5

Функционал: метод обработчика объекта.

Параметры: message - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода handler класса cl_5

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal to ", значения, возвращённого методом get_coordinate, строки " Text: " и значения параметра message	Ø

3.21 Алгоритм метода signal класса cl_6

Функционал: метод сигнала объекта.

Параметры: message - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода signal класса cl_6

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal from " и значения, возвращённого методом get_coordinate	2
2		Присваивание параметру message значения суммы значения параметра message и строки " (class: 6)"	Ø

3.22 Алгоритм метода **handler** класса **cl_6**

Функционал: метод обработчика объекта.

Параметры: message - сообщение (строка).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода *handler* класса *cl_6*

№	Предикат	Действия	№ перехода
1		Вывод строки "\nSignal to ", значения, возвращённого методом get_coordinate, строки " Text: " и значения параметра message	Ø

3.23 Алгоритм метода **build_tree_objects** класса **cl_application**

Функционал: построение исходного дерева иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода *build_tree_objects* класса *cl_application*

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной parent_coordinate	2
2		Объявление строковой переменной child_name	3
3		Инициализация целочисленной переменной class_number значением 0	4
4		Инициализация указателя на объект класса cl_base input_parent значением nullptr	5
5		Инициализация указателя на объект класса cl_base input_child значением nullptr	6

№	Предикат	Действия	№ перехода
6		Ввод значения переменной child_name	7
7		Вызов метода set_name со значением переменной child_name в качестве параметра	8
8		Ввод значения переменной parent_coordinate	9
9	Значение переменной parent_coordinate равно "endtree"		10
			23
10		Ввод значений переменных child_name и class_name	11
11		Присваивание переменной input_parent значением, возвращённым методом get_by_coordinate со значением переменной parent_coordinate в качестве параметра	12
12	Значение указателя input_parent не равно nullptr		13
			20
13	Значение, возвращённое методом get_child объекта input_parent со значением переменной child_name в качестве параметра, равно nullptr		14
			19
14	Значение переменной class_number равно 2	Присваивание указателю input_child адреса нового объекта класса cl_2, созданного с помощью оператора new с помощью параметризованного конструктора	8
			15
15	Значение переменной	Присваивание указателю input_child адреса нового	8

№	Предикат	Действия	№ перехода
	class_number равно 3	объекта класса cl_3, созданного с помощью оператора new с помощью параметризованного конструктора	
			16
16	Значение переменной class_number равно 4	Присваивание указателю input_child адреса нового объекта класса cl_4, созданного с помощью оператора new с помощью параметризованного конструктора	8
			17
17	Значение переменной class_number равно 5	Присваивание указателю input_child адреса нового объекта класса cl_5, созданного с помощью оператора new с помощью параметризованного конструктора	8
			18
18	Значение переменной class_number равно 6	Присваивание указателю input_child адреса нового объекта класса cl_6, созданного с помощью оператора new с помощью параметризованного конструктора	8
			8
19		Вывод значения переменной parent_coordinate и строки " Dubbing the names of subordinate objects"	8
20		Вывод строки "Object tree\n"	21
21		Вызов метода print_tree со строкой " " (4 пробела) в качестве аргумента	22
22		Вывод строки "\nThe head object ", значения переменной parent_coordinate и строки " is not found	∅
23		Объявление строковых переменных	24

№	Предикат	Действия	№ перехода
		from_coordinate и to_coordinate	
24		Инициализация указателя на объект класса cl_base from значением nullptr	25
25		Инициализация указателя на объект класса cl_base to значением nullptr	26
26		Ввод значения переменной from_coordinate	27
27	Значение переменной from_coordinate равно "end_of_connection"		∅
			28
28		Ввод значения переменной to_coordinate	29
29		Присваивание указателю from значения, возвращённого методом get_by_coordinate со значением переменной from_coordinate в качестве параметра	30
30		Присваивание указателю to значения, возвращённого методом get_by_coordinate со значением переменной to_coordinate в качестве параметра	31
31		Вызов метода add_connection объекта from со значением, возвращённым методом get_signal с указателем from в качестве параметра, указателем to и значением, возвращённым методом get_handler с указателем to в качестве параметра, в качестве параметров	26

3.24 Алгоритм метода exec_app класса cl_application

Функционал: запуск приложения.

Параметры: отсутствуют.

Возвращаемое значение: целочисленный код завершения работы приложения.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода `exec_app` класса `cl_application`

№	Предикат	Действия	№ перехода
1		Вывод строки "Object tree\n"	2
2		Вызов метода <code>print_tree</code> со строкой " " (4 пробела) в качестве параметра	3
3		Объявление строковых переменных <code>coordinate</code> , <code>command</code> и <code>info</code>	4
4		Инициализация указателя на объект класса <code>cl_base</code> значением указателя <code>this</code>	5
5		Вызов метода <code>activate</code>	6
6		Ввод значения переменной <code>command</code>	7
7	Значение переменной <code>command</code> равно "END"		21
			8
8		Ввод значения переменной <code>coordinate</code>	9
9		Вызов функции <code>std::getline</code> с параметрами <code>std::cin</code> и значением переменной <code>info</code>	10
10		Вызов метода <code>erase</code> переменной <code>info</code> со значением, возвращённым методом <code>begin</code> переменной <code>info</code> , в качестве параметра	11
11		Инициализация указателя на объект класса <code>cl_base</code> <code>from</code> значением, возвращённым методом <code>get_by_coordinate</code> объекта <code>current_object</code> со значением переменной <code>coordinate</code> в качестве параметра	12

№	Предикат	Действия	№ перехода
12		Инициализация указателя на объект класса cl_base to значением nullptr	13
13	Значение указателя from равно nullptr	Вывод строки "Object ", значения переменной coordinate и строки " not found\n"	6
			14
14	Значение переменной command равно "EMIT"	Вызов метода emit_signal объекта from со значением, возвращённым методом get_signal с указателем from в качестве параметра, и значением переменной info в качестве параметров	6
			15
15	Значение переменной command равно "SET_CONNECT"		16
			18
16		Присваивание указателю to значения, возвращённого методом get_by_coordinate со значением переменной info в качестве параметра	17
17	Значение указателя to равно nullptr	Вывод строки "Handler object ", значения переменной info и строки " not found\n"	6
		Вызов метода add_connection объекта from со значением, возвращённым методом get_signal с указателем from в качестве параметра, указателем to и значением, возвращённым методом get_handler с указателем to в качестве параметра, в качестве параметров	6
18	Значение переменной command равно "DELETE_CONNECT"	Присваивание указателю to значения, возвращённого методом get_by_coordinate со значением переменной info в качестве параметра	19
			20
19	Значение указателя to равно	Вывод строки "Handler object ", значения	6

№	Предикат	Действия	№ перехода
	nullptr	переменной info и строки " not found\n"	
		Вызов метода delete_connection объекта from со значением, возвращённым методом get_signal с указателем from в качестве параметра, указателем to и значением, возвращённым методом get_handler с указателем to в качестве параметра, в качестве параметров	6
20	Значение переменной command равно "SET_CONDITION"	Вызов метода set_state объекта from со значением, возвращённым функцией std::stoi со значением переменной info в качестве параметра, в качестве параметра	6
			6
21		Возврат значения 0	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-38.

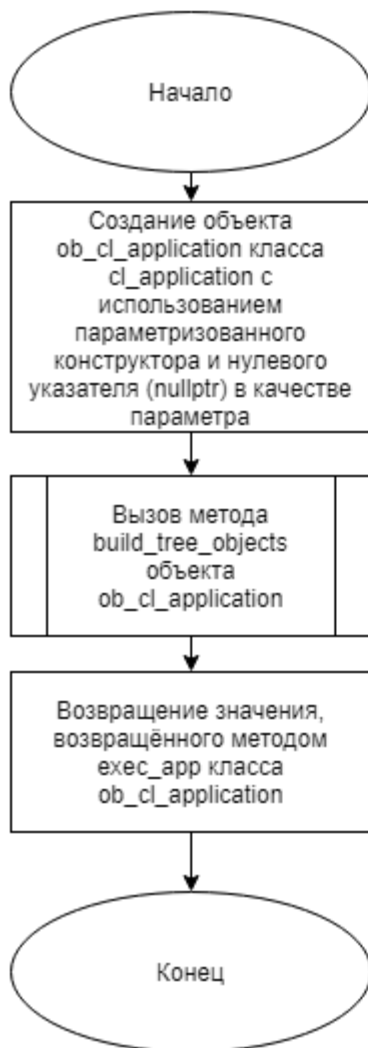


Рисунок 1 – Блок-схема алгоритма



Рисунок 2 – Блок-схема алгоритма



Рисунок 3 – Блок-схема алгоритма



Рисунок 4 – Блок-схема алгоритма

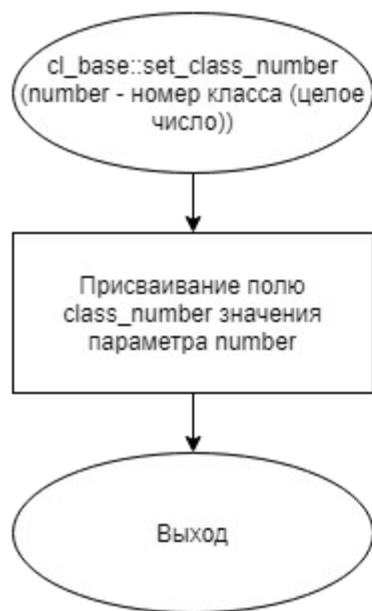


Рисунок 5 – Блок-схема алгоритма

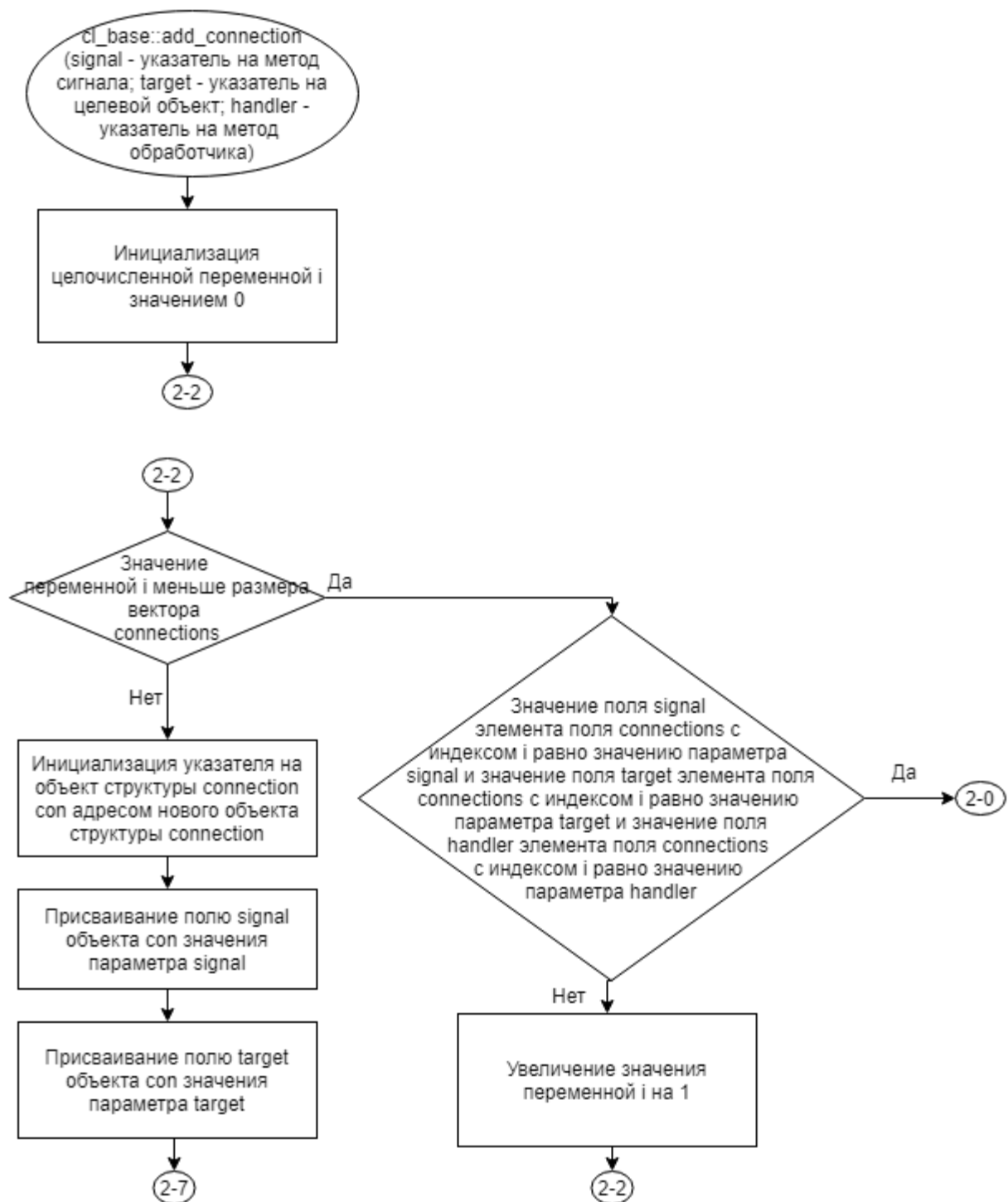


Рисунок 6 – Блок-схема алгоритма

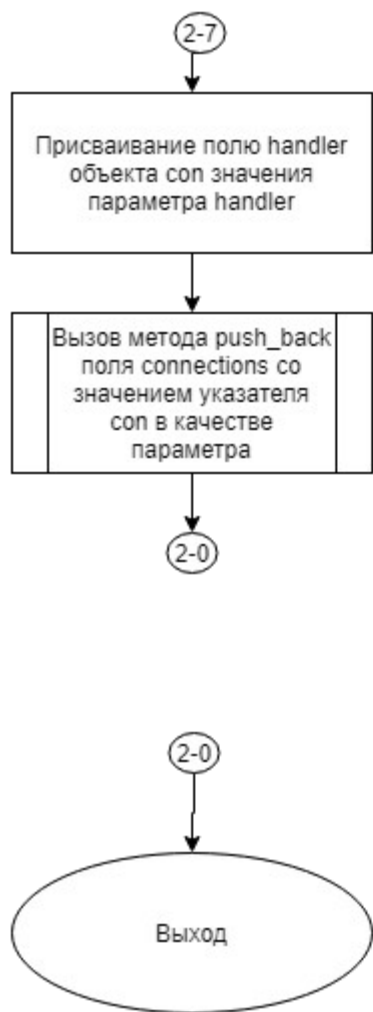


Рисунок 7 – Блок-схема алгоритма

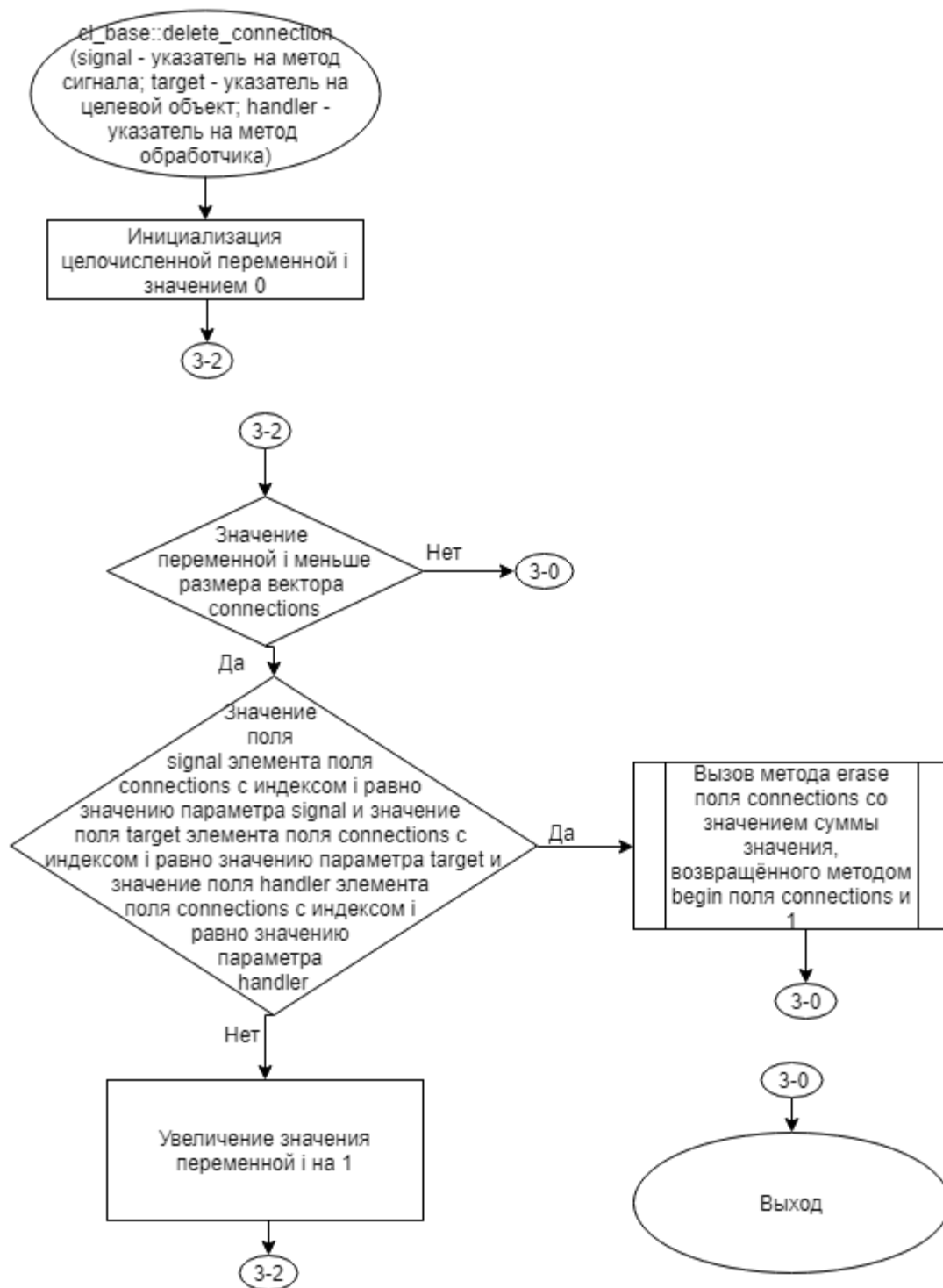


Рисунок 8 – Блок-схема алгоритма

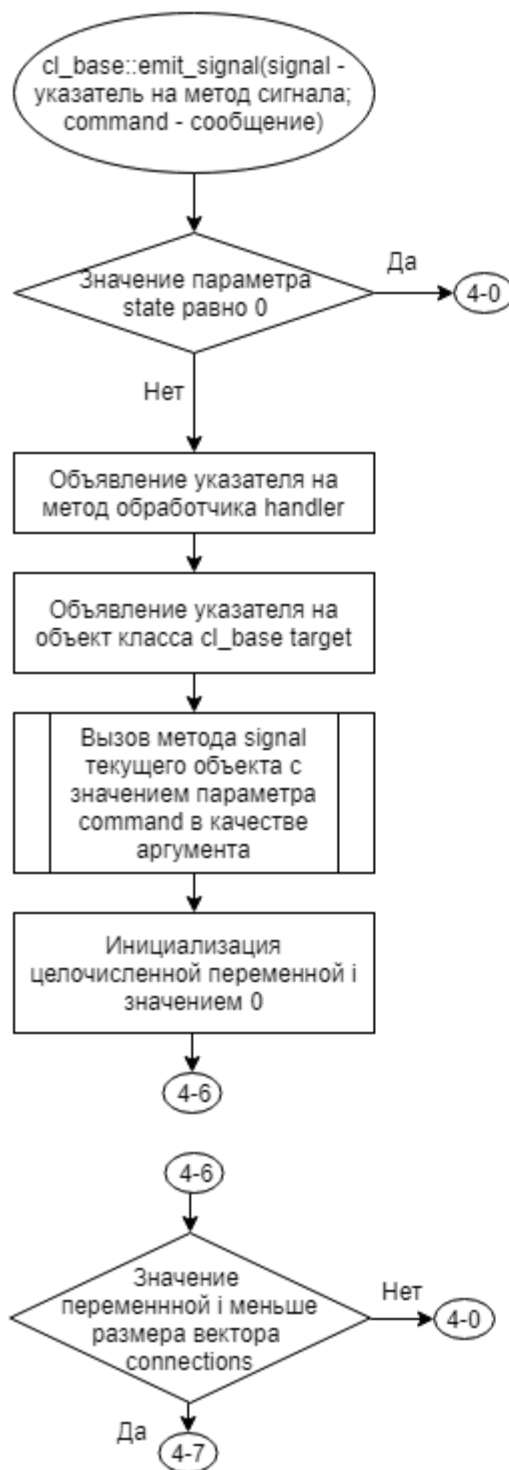


Рисунок 9 – Блок-схема алгоритма

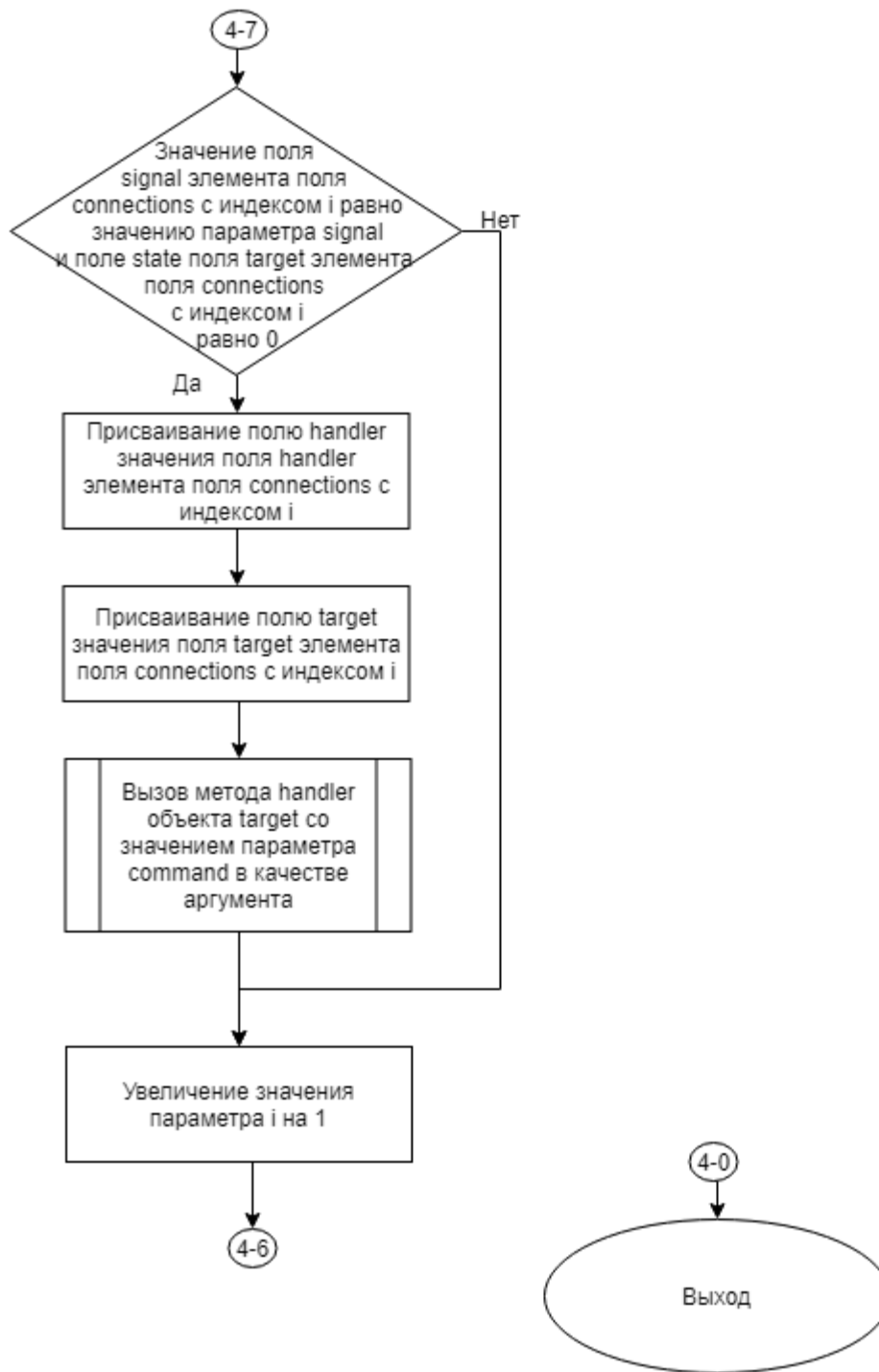


Рисунок 10 – Блок-схема алгоритма



Рисунок 11 – Блок-схема алгоритма

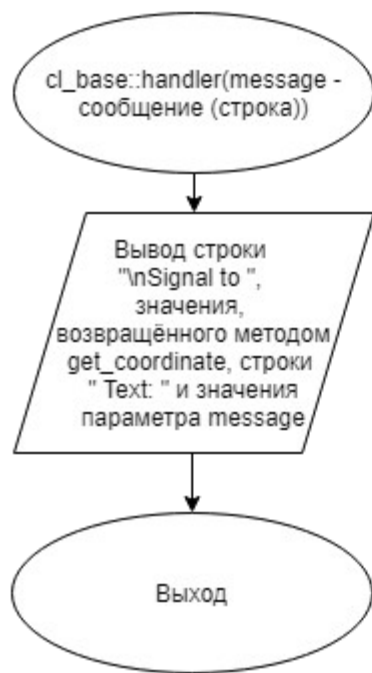


Рисунок 12 – Блок-схема алгоритма

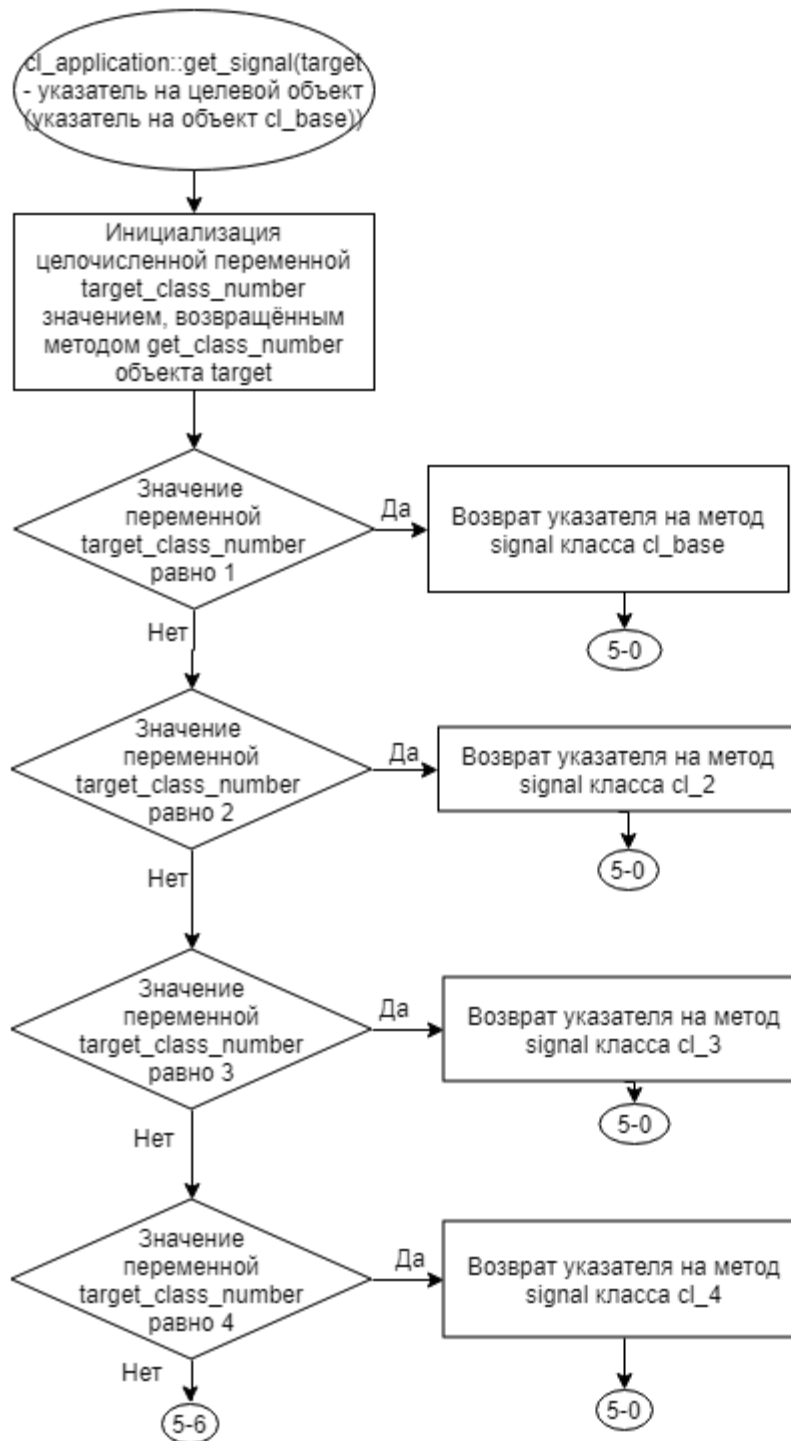


Рисунок 13 – Блок-схема алгоритма

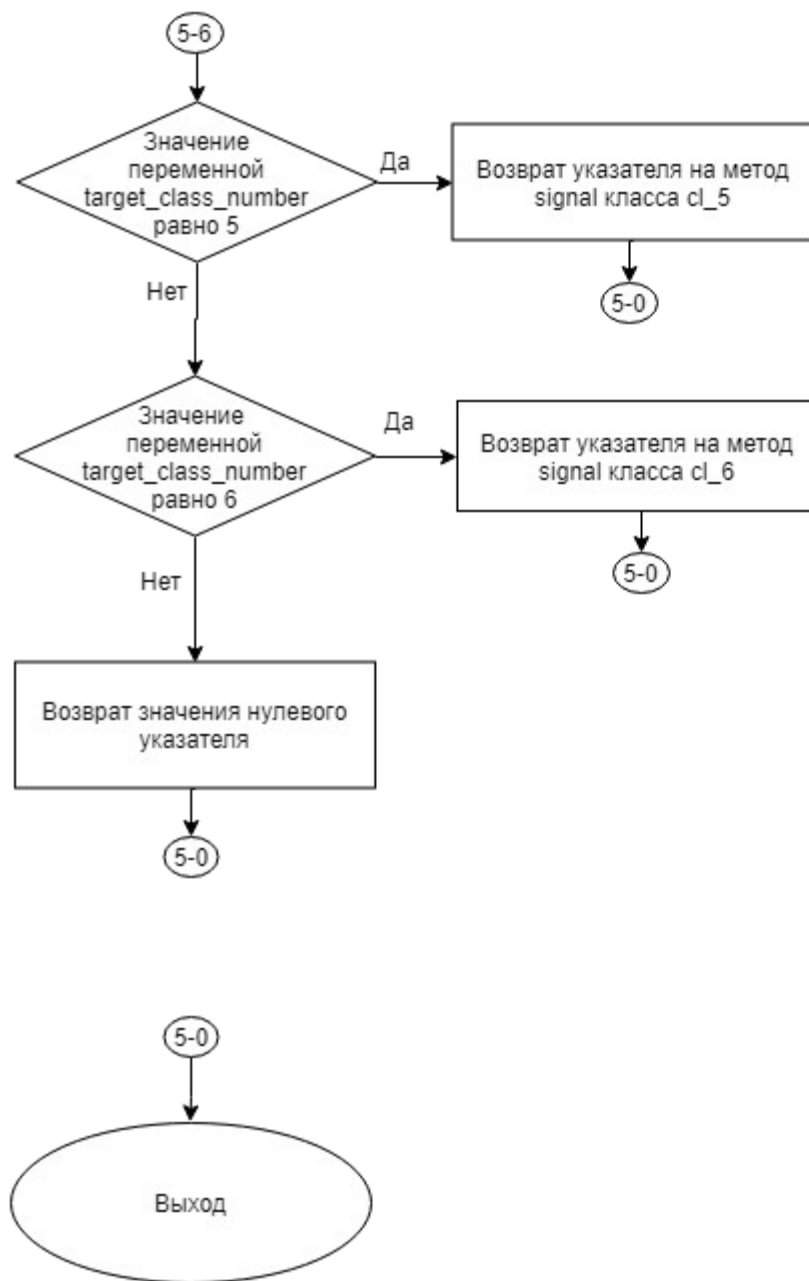


Рисунок 14 – Блок-схема алгоритма

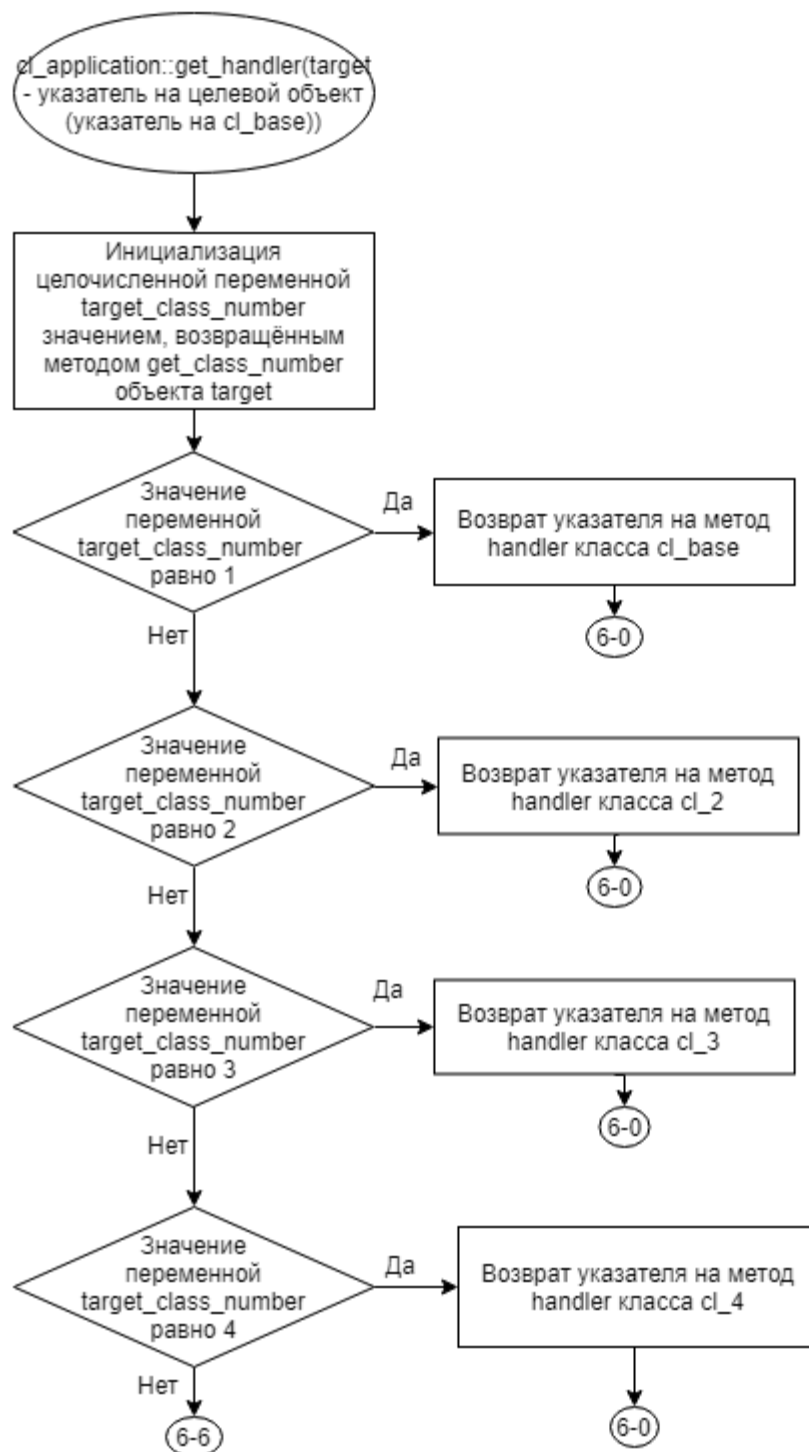


Рисунок 15 – Блок-схема алгоритма

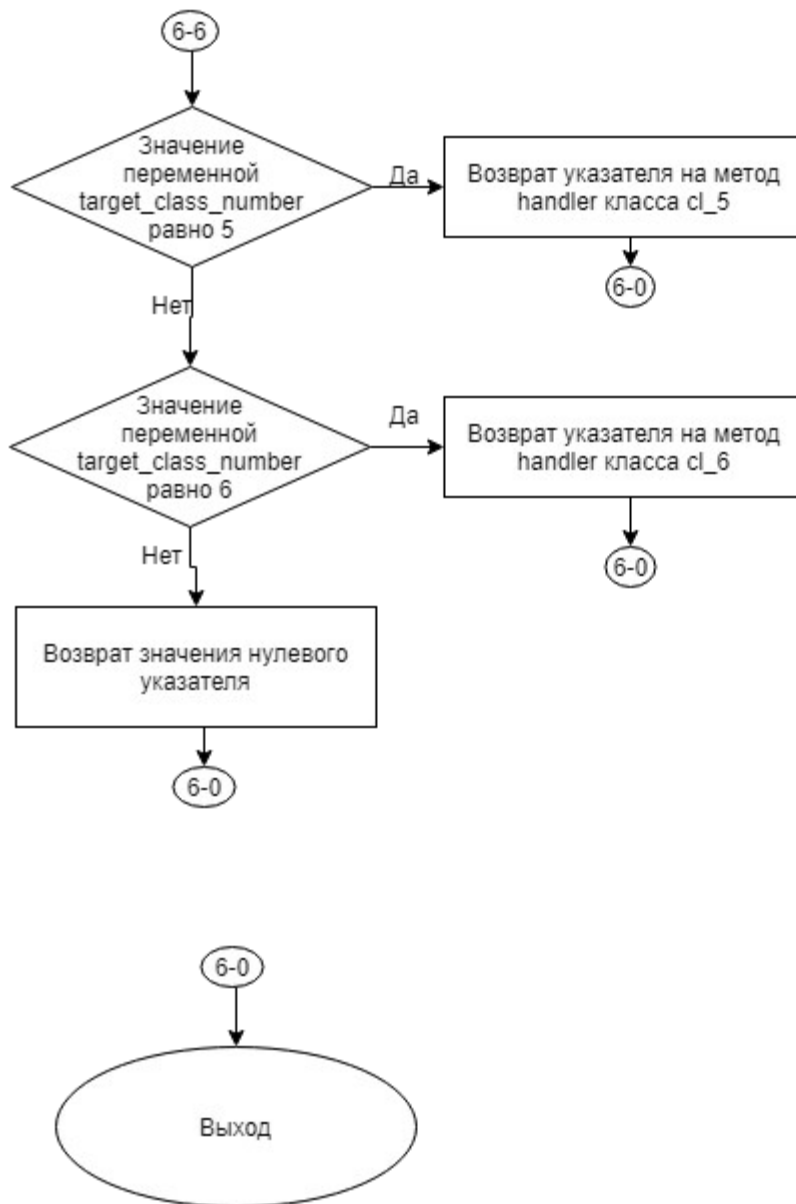


Рисунок 16 – Блок-схема алгоритма

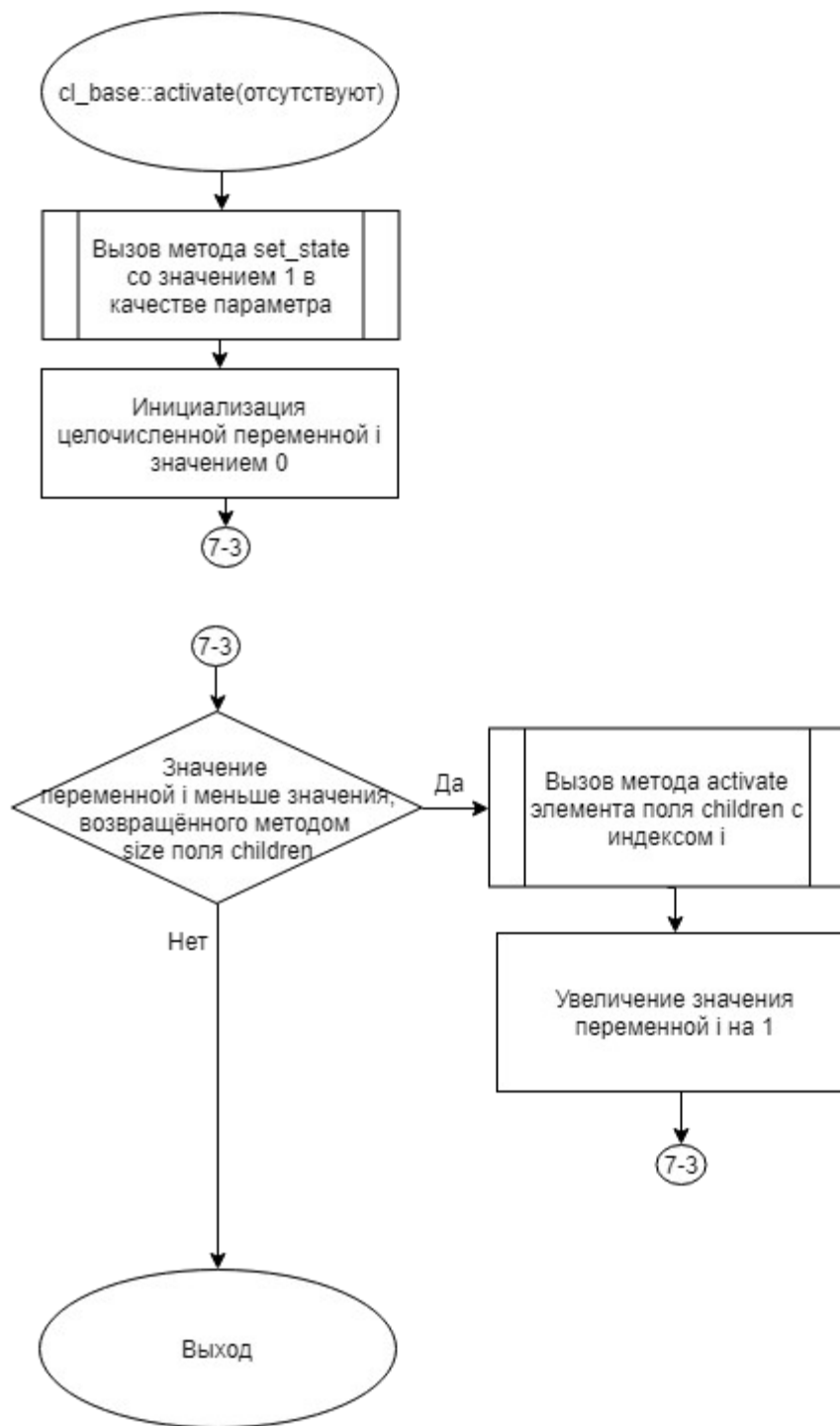


Рисунок 17 – Блок-схема алгоритма



Рисунок 18 – Блок-схема алгоритма

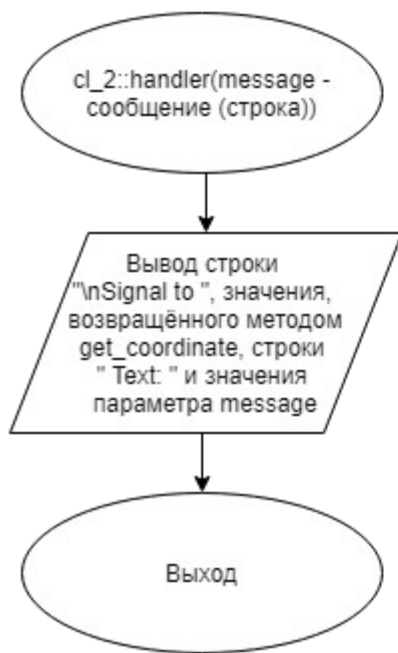


Рисунок 19 – Блок-схема алгоритма



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

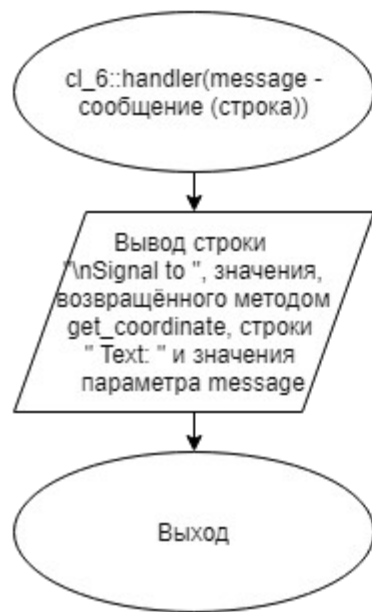


Рисунок 27 – Блок-схема алгоритма



Рисунок 28 – Блок-схема алгоритма

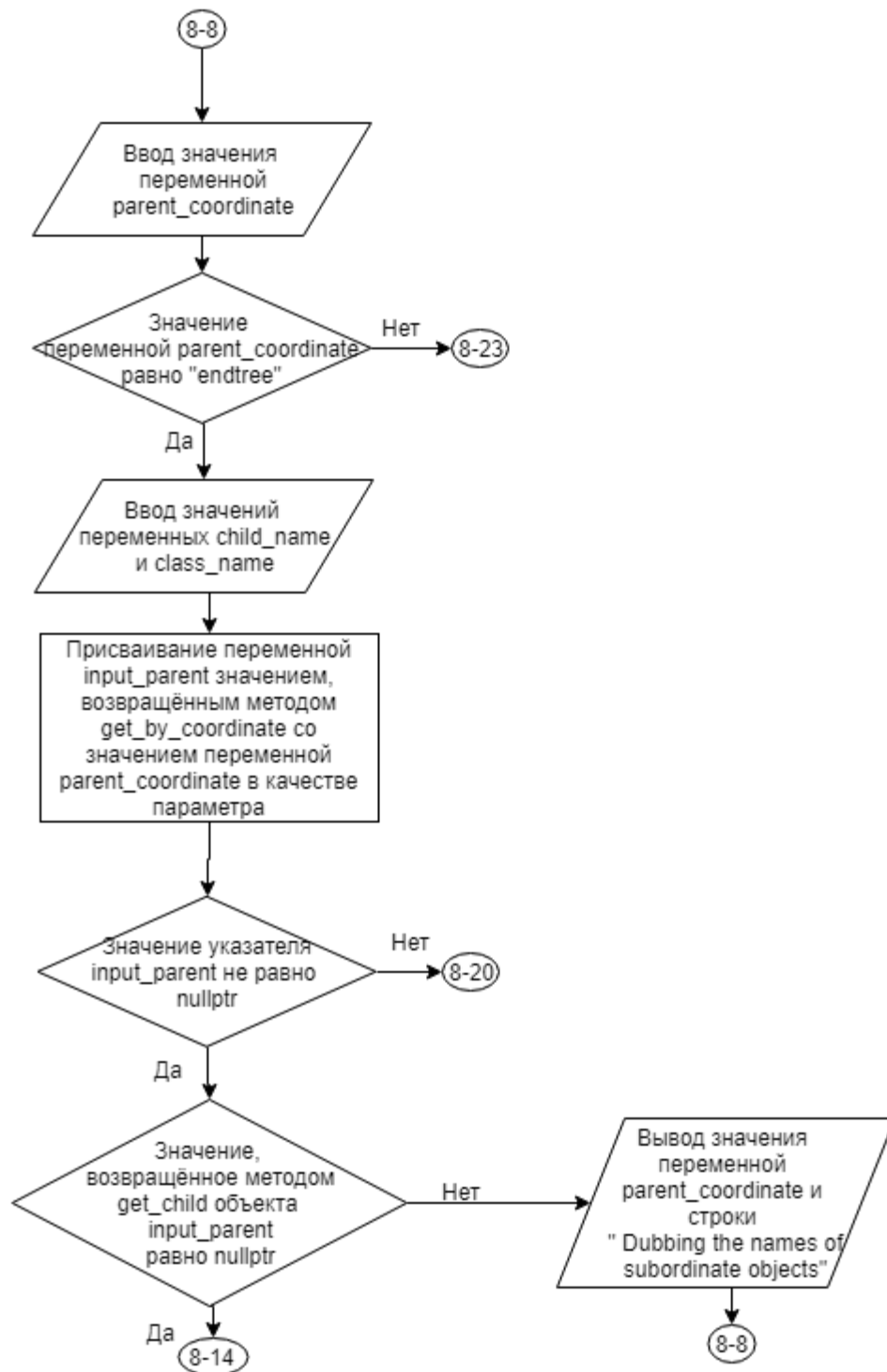


Рисунок 29 – Блок-схема алгоритма

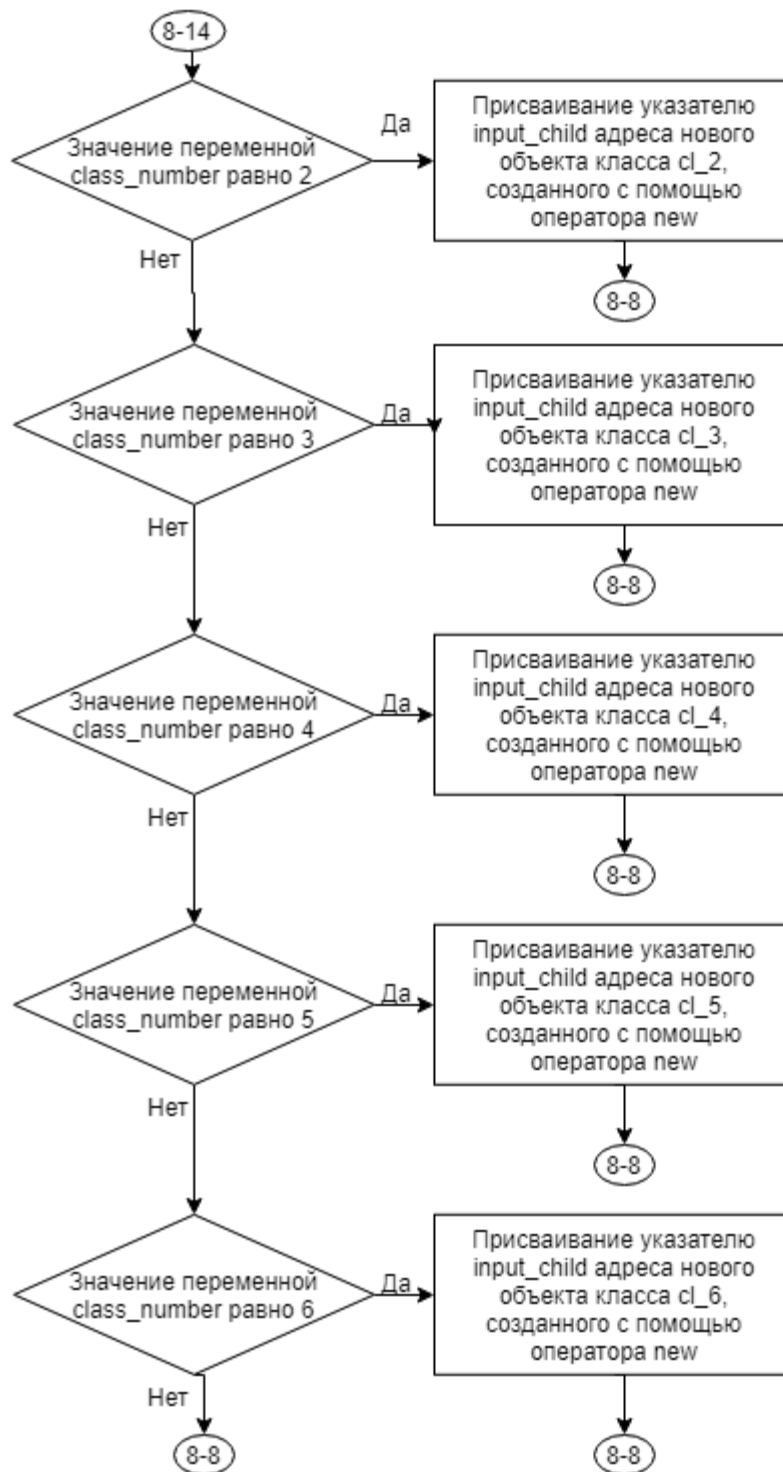


Рисунок 30 – Блок-схема алгоритма



Рисунок 31 – Блок-схема алгоритма

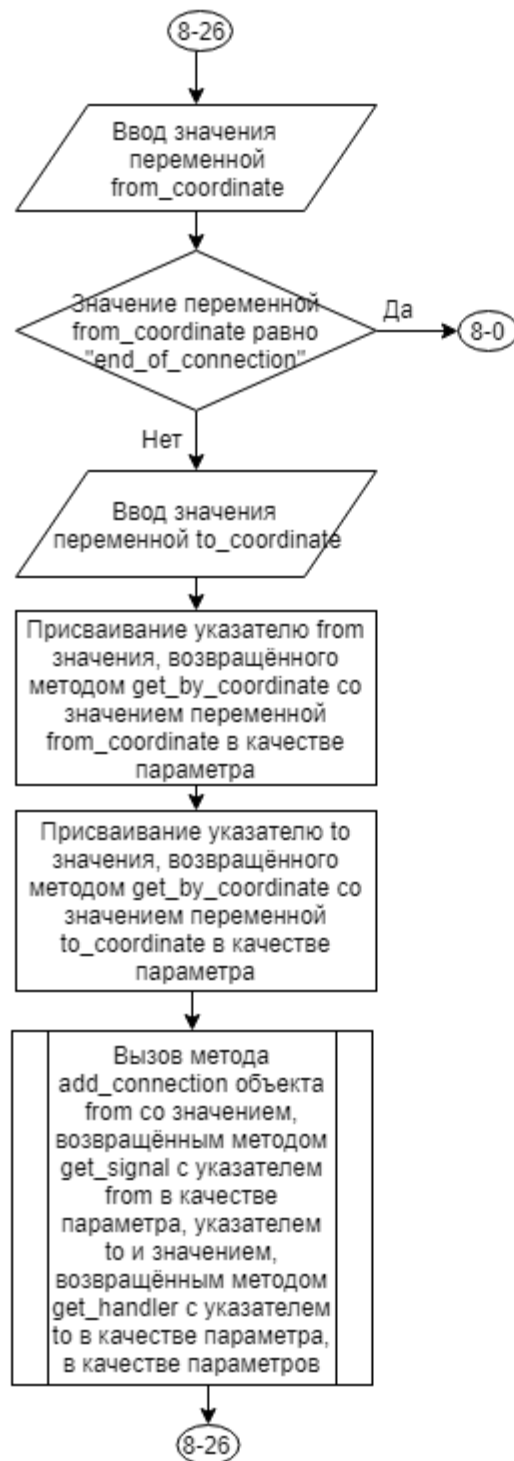


Рисунок 32 – Блок-схема алгоритма

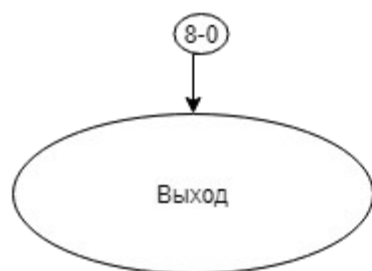


Рисунок 33 – Блок-схема алгоритма



Рисунок 34 – Блок-схема алгоритма

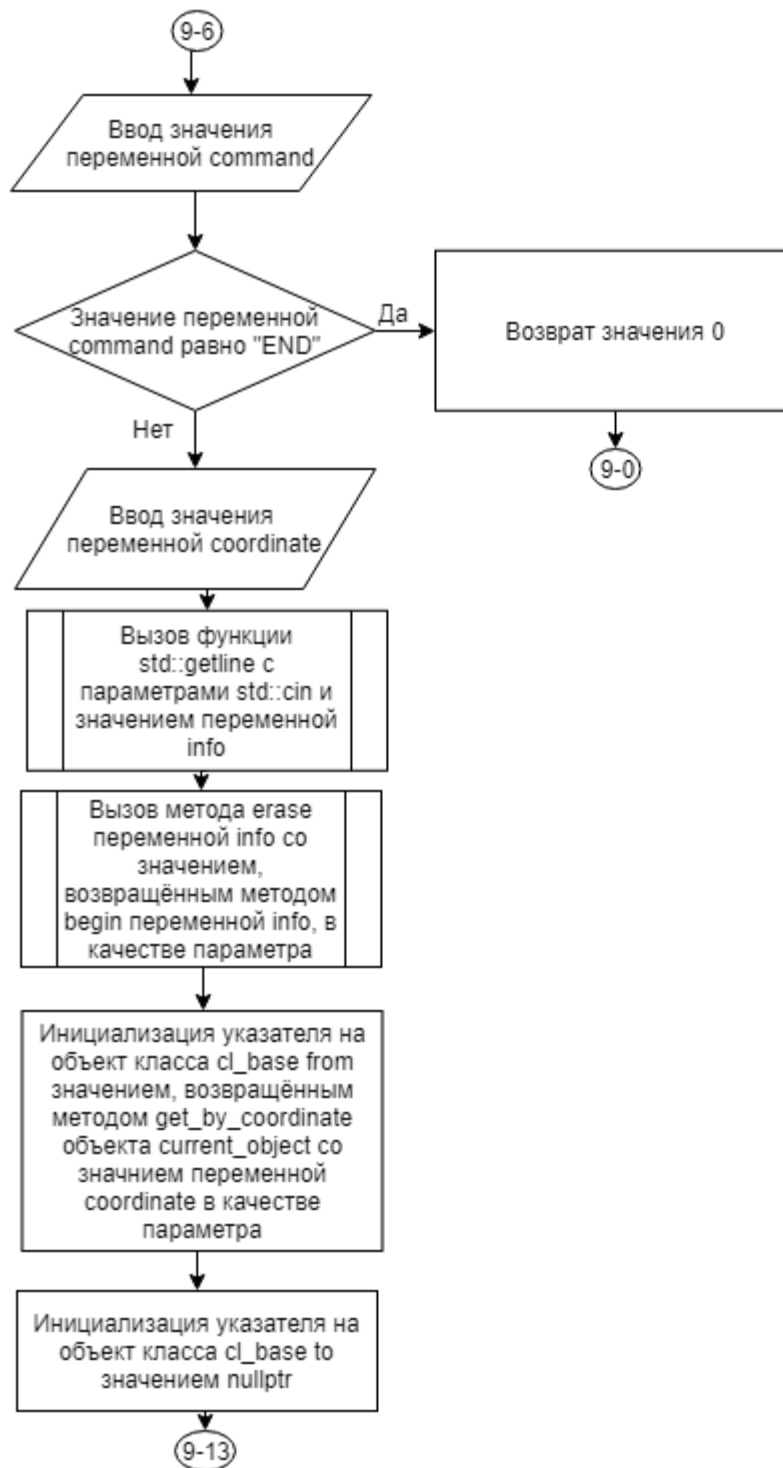


Рисунок 35 – Блок-схема алгоритма

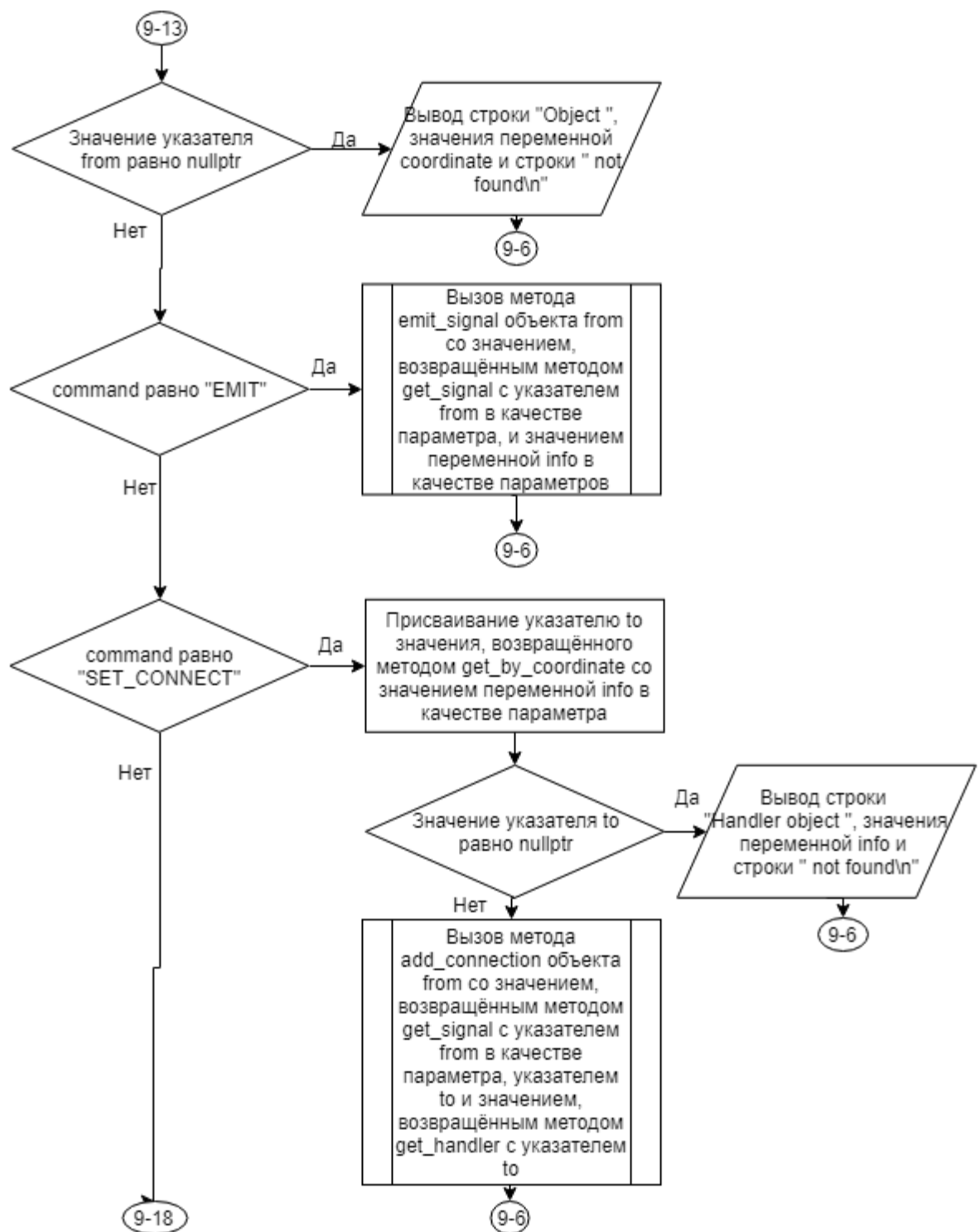


Рисунок 36 – Блок-схема алгоритма

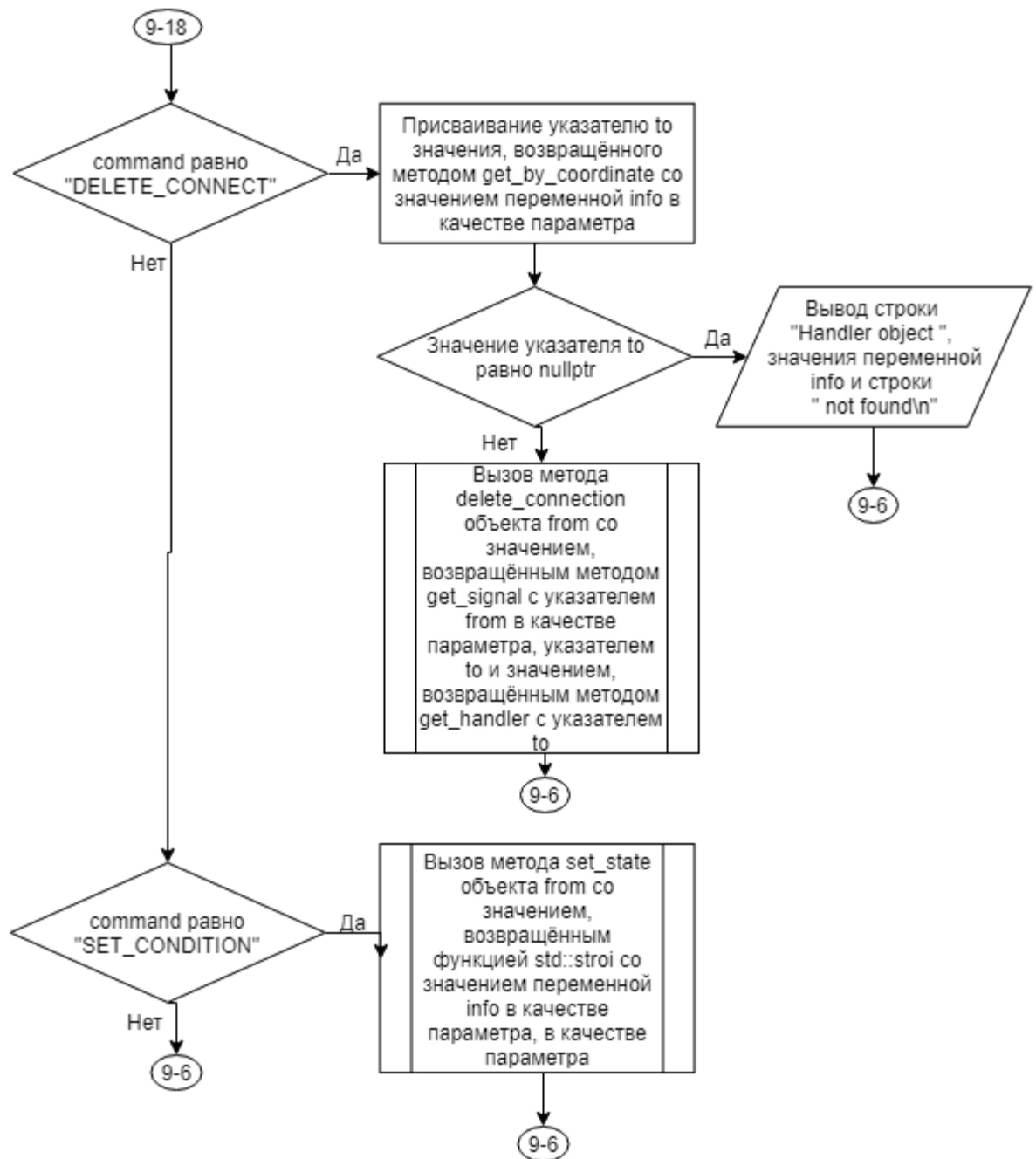


Рисунок 37 – Блок-схема алгоритма

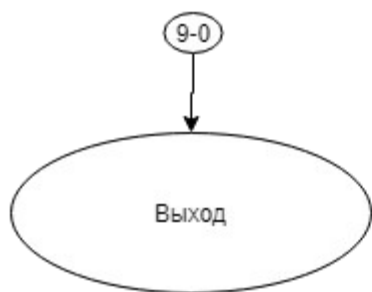


Рисунок 38 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(cl_base* parent, std::string name)
    : cl_base(parent, name)
{
    set_class_number(2);
}

void cl_2::signal(std::string& message)
{
    std::cout << "\nSignal from " << get_coordinate();
    message += " (class: 2)";
}

void cl_2::handler(std::string message)
{
    std::cout << "\nSignal to " << get_coordinate() << " Text:  " << message;
}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H

#include "cl_base.h"
#include <string>

class cl_2 : public cl_base
{
public:
    cl_2(cl_base* parent, std::string name);
    void signal(std::string&);
}
```

```

        void handler(std::string);
    };

#endif

```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```

#include "cl_3.h"

cl_3::cl_3(cl_base* parent, std::string name)
    : cl_base(parent, name)
{
    set_class_number(3);
}

void cl_3::signal(std::string& message)
{
    std::cout << "\nSignal from " << get_coordinate();
    message += " (class: 3)";
}

void cl_3::handler(std::string message)
{
    std::cout << "\nSignal to " << get_coordinate() << " Text:  " << message;
}

```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```

#ifndef __CL_3__H
#define __CL_3__H

#include "cl_base.h"
#include <string>

class cl_3 : public cl_base
{
public:
    cl_3(cl_base* parent, std::string name);
    void signal(std::string&);
    void handler(std::string);
};

```

```
#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* parent, std::string name)
    : cl_base(parent, name)
{
    set_class_number(4);
}

void cl_4::signal(std::string& message)
{
    std::cout << "\nSignal from " << get_coordinate();
    message += " (class: 4)";
}

void cl_4::handler(std::string message)
{
    std::cout << "\nSignal to " << get_coordinate() << " Text:  " << message;
}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H

#include "cl_base.h"
#include <string>

class cl_4 : public cl_base
{
public:
    cl_4(cl_base* parent, std::string name);
    void signal(std::string&);
    void handler(std::string);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* parent, std::string name)
    : cl_base(parent, name)
{
    set_class_number(5);
}

void cl_5::signal(std::string& message)
{
    std::cout << "\nSignal from " << get_coordinate();
    message += " (class: 5)";
}

void cl_5::handler(std::string message)
{
    std::cout << "\nSignal to " << get_coordinate() << " Text:  " << message;
}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H

#include "cl_base.h"
#include <string>

class cl_5 : public cl_base
{
public:
    cl_5(cl_base* parent, std::string name);
    void signal(std::string&);
    void handler(std::string);
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* parent, std::string name)
    : cl_base(parent, name)
{
    set_class_number(6);
}

void cl_6::signal(std::string& message)
{
    std::cout << "\nSignal from " << get_coordinate();
    message += " (class: 6)";
}

void cl_6::handler(std::string message)
{
    std::cout << "\nSignal to " << get_coordinate() << " Text:  " << message;
}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H

#include "cl_base.h"
#include <string>

class cl_6 : public cl_base
{
public:
    cl_6(cl_base* parent, std::string name);
    void signal(std::string&);
    void handler(std::string);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"

// Параметризованный конструктор
cl_application::cl_application(cl_application* parent) : cl_base(parent) {}

// Метод построения исходного дерева иерархии объектов
void cl_application::build_tree_objects()
{
    std::string parent_coordinate;
    std::string child_name;
    int class_number = 0;
    cl_base* input_parent = nullptr;
    cl_base* input_child = this;

    std::cin >> child_name;
    set_name(child_name);

    while (true)
    {
        std::cin >> parent_coordinate;
        if (parent_coordinate == "endtree")
        {
            break;
        }

        std::cin >> child_name >> class_number;
        input_parent = get_by_coordinate(parent_coordinate);

        if (input_parent != nullptr)
        {
            if (input_parent->get_child(child_name) == nullptr)
            {
                if (class_number == 2)
                {
                    input_child = new cl_2(input_parent, child_name);
                }
                else if (class_number == 3)
                {
                    input_child = new cl_3(input_parent, child_name);
                }
                else if (class_number == 4)
                {
                    input_child = new cl_4(input_parent, child_name);
                }
                else if (class_number == 5)
                {
                    input_child = new cl_5(input_parent, child_name);
                }
                else if (class_number == 6)
                {

```

```

        input_child = new cl_6(input_parent, child_name);
    }
}
else
{
    std::cout << parent_coordinate
        << "    Dubbing the names of subordinate objects\n";
}
}
else
{
    std::cout << "Object tree\n";
    print_tree("    ");
    std::cout << "\nThe head object "
        << parent_coordinate << " is not found";
    exit(0);
}
}

std::string from_coordinate, to_coordinate;
cl_base* from = nullptr;
cl_base* to = nullptr;
while (true)
{
    std::cin >> from_coordinate;
    if (from_coordinate == "end_of_connections")
    {
        break;
    }

    std::cin >> to_coordinate;
    from = get_by_coordinate(from_coordinate);
    to = get_by_coordinate(to_coordinate);

    from->add_connection(get_signal(from), to, get_handler(to));
}

// Метод запуска приложения
int cl_application::exec_app()
{
    std::cout << "Object tree\n";
    print_tree("    ");

    std::string coordinate, command, info;
    cl_base* current_object = this;

    activate();

    while (true)
    {
        std::cin >> command;
        if (command == "END")
        {
            break;

```



```

    }

    std::cin >> coordinate;
    std::getline(std::cin, info);
    info.erase(info.begin());
    cl_base* from = current_object->get_by_coordinate(coordinate);
    cl_base* to = nullptr;

    if (from == nullptr)
    {
        std::cout << "Object " << coordinate << " not found\n";
    }
    else if (command == "EMIT")
    {
        from->emit_signal(get_signal(from), info);
    }
    else if (command == "SET_CONNECT")
    {
        to = get_by_coordinate(info);
        if (to == nullptr)
        {
            std::cout << "Handler object " << info << " not found\n";
        }
        else
        {
            from->add_connection(get_signal(from), to, get_handler(to));
        }
    }
    else if (command == "DELETE_CONNECT")
    {
        to = get_by_coordinate(info);
        if (to == nullptr)
        {
            std::cout << "Handler object " << info << " not found\n";
        }
        else
        {
            from->delete_connection(get_signal(from), to, get_handler(to));
        }
    }
    else if (command == "SET_CONDITION")
    {
        from->set_state(std::stoi(info));
    }
}

return 0;
}

void cl_application::read_states()
{
    std::string input_name;
    int input_state = 0;

    while (std::cin >> input_name >> input_state)

```

```

    {
        cl_base* obj = find_in_tree(input_name);
        if (obj != nullptr)
        {
            obj->set_state(input_state);
        }
    }
}

TYPE_SIGNAL cl_application::get_signal(cl_base* target)
{
    int target_class_number = target->get_class_number();

    if (target_class_number == 1)
    {
        return SIGNAL_D(cl_base::signal);
    }
    else if (target_class_number == 2)
    {
        return SIGNAL_D(cl_2::signal);
    }
    else if (target_class_number == 3)
    {
        return SIGNAL_D(cl_3::signal);
    }
    else if (target_class_number == 4)
    {
        return SIGNAL_D(cl_4::signal);
    }
    else if (target_class_number == 5)
    {
        return SIGNAL_D(cl_5::signal);
    }
    else if (target_class_number == 6)
    {
        return SIGNAL_D(cl_6::signal);
    }

    return nullptr;
}

TYPE_HANDLER cl_application::get_handler(cl_base* target)
{
    int target_class_number = target->get_class_number();

    if (target_class_number == 1)
    {
        return HANDLER_D(cl_base::handler);
    }
    else if (target_class_number == 2)
    {
        return HANDLER_D(cl_2::handler);
    }
    else if (target_class_number == 3)
    {

```

```

        return HANDLER_D(cl_3::handler);
    }
    else if (target_class_number == 4)
    {
        return HANDLER_D(cl_4::handler);
    }
    else if (target_class_number == 5)
    {
        return HANDLER_D(cl_5::handler);
    }
    else if (target_class_number == 6)
    {
        return HANDLER_D(cl_6::handler);
    }
    return nullptr;
}

```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```

#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H

#include "cl_base.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <iostream>
#include <string>

class cl_application : public cl_base
{
private:
public:

    // Параметризованный конструктор
    cl_application(cl_application* parent);

    // Метод построения исходного дерева иерархии объектов
    void build_tree_objects();

    // Метод запуска приложения
    int exes_app();

    // Метод для считывания имён объектов и их готовностей
    void read_states();
}

```

```
    TYPE_SIGNAL get_signal(cl_base* target);
    TYPE_HANDLER get_handler(cl_base* target);
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"

// Параметризованный конструктор
cl_base::cl_base(cl_base* parent, std::string name)
{
    // Установка родительского объекта
    this->parent = parent;
    // Установка имени объекта
    this->name = name;

    this->state = 0;

    this->class_number = 1;

    if (parent != nullptr)
    {
        // Добавление объекта к подчинённым
        // объектом родительского объекта
        parent->children.push_back(this);
    }
}

// Деструктор
cl_base::~cl_base()
{
    for (int i = 0; i < children.size(); i++)
    {
        delete children[i];
    }
}

// Метод редактирования имени объекта
bool cl_base::set_name(std::string new_name)
{
    if (parent != nullptr)
    {
        for (int i = 0; i < parent->children.size(); i++)
        {
            if (parent->children[i]->name == new_name)
```

```

        {
            // Объект с именем, совпадающим
            // с новым именем найден среди
            // подчинённых объектов родительского
            return false;
        }
    }
}

// Установка нового имени объекта
this->name = new_name;
return true;
}

// Метод получения имени объекта
std::string cl_base::get_name()
{
    return name;
}

// Метод получения указателя на головной объект текущего объекта
cl_base* cl_base::get_parent()
{
    return parent;
}

// Метод вывода наименований объектов в дереве иерархии слева
// направо и сверху вниз
void cl_base::print_tree(std::string prefix)
{
    if (parent == nullptr)
    {
        // Вывод имени текущего объекта
        std::cout << name;
    }

    // Вывод имён всех подчинённых объектов
    for (int i = 0; i < children.size(); i++)
    {
        std::cout << '\n' << prefix << children[i]->name;
        children[i]->print_tree(prefix + "    ");
    }
}

// Метод получения указателя на непосредственно подчиненный
// объект по его имени
cl_base* cl_base::get_child(std::string child_name)
{
    for (int i = 0; i < children.size(); i++)
    {
        if (children[i]->name == child_name)
        {
            // Среди подчинённых объектов
            // найден объект с именем,
            // совпадающим с искомым

```

```

        return children[i];
    }
}

// Объект не найден
return nullptr;
}

cl_base* cl_base::find_in_tree(std::string wanted_name)
{
    if (name == wanted_name)
    {
        return this;
    }

    for (int i = 0; i < children.size(); i++)
    {
        cl_base* found = children[i]->find_in_tree(wanted_name);
        if (found != nullptr)
        {
            return found;
        }
    }

    return nullptr;
}

void cl_base::print_tree_with_state(std::string prefix)
{
    if (parent == nullptr)
    {
        std::cout << name << (state == 0 ? " is not ready" : " is ready");
    }

    for (int i = 0; i < children.size(); i++)
    {
        std::cout << "\n" << prefix << children[i]->get_name()
            << (children[i]->get_state() == 0 ? " is not ready" : " is ready");
        children[i]->print_tree_with_state(prefix + "    ");
    }
}

int cl_base::get_state()
{
    return state;
}

void cl_base::set_state(int state)
{
    if (parent == nullptr || parent->state != 0)
    {
        this->state = state;
    }
    else
    {

```

```

        this->state = 0;
    }

    if (state == 0)
    {
        for (int i = 0; i < children.size(); i++)
        {
            children[i]->set_state(0);
        }
    }
}

cl_base* cl_base::get_root()
{
    if (parent == nullptr)
    {
        return this;
    }
    else
    {
        return parent->get_root();
    }
}

cl_base* cl_base::find_in_tree_from_root(std::string wanted_name)
{
    int amount = get_root()->count_in_tree(wanted_name);
    if (amount == 1)
    {
        return get_root()->find_in_tree(wanted_name);
    }

    return nullptr;
}

int cl_base::count_in_tree(std::string wanted_name)
{
    int res = 0;
    if (name == wanted_name)
    {
        res++;
    }

    for (int i = 0; i < children.size(); i++)
    {
        res += children[i]->count_in_tree(wanted_name);
    }

    return res;
}

bool cl_base::set_parent(cl_base* new_parent)
{
    if (parent == nullptr || new_parent == nullptr ||
        new_parent->get_child(name) != nullptr)

```

```

    {
        return false;
    }

    cl_base* temp = new_parent;
    while (temp != nullptr)
    {
        if (temp == this)
        {
            return false;
        }
        temp = temp->parent;
    }

    for (int i = 0; i < parent->children.size(); i++)
    {
        if (parent->children[i] == this)
        {
            parent->children.erase(parent->children.begin() + i);
        }
    }

    parent = new_parent;
    new_parent->children.push_back(this);
    return true;
}

std::string cl_base::delete_child(std::string child_name)
{
    cl_base* to_delete = get_child(child_name);
    if (to_delete == nullptr)
    {
        return "";
    }

    std::string delete_coordinate;

    cl_base* temp = to_delete;
    while (temp->get_parent() != nullptr)
    {
        delete_coordinate = '/' + temp->get_name() + delete_coordinate;
        temp = temp->get_parent();
    }

    disconnect_child(child_name);
    delete to_delete;

    return delete_coordinate;
}

cl_base* cl_base::get_by_coordinate(std::string coordinate)
{
    cl_base* root = get_root();

    if (coordinate == "/")

```



```

    {
        return root;
    }

    if (coordinate == ".")
    {
        return this;
    }

    if (coordinate.substr(0, 2) == "//")
    {
        return find_in_tree_from_root(coordinate.substr(2, coordinate.length()
- 1));
    }

    if (coordinate[0] == '.')
    {
        return find_in_tree(coordinate.substr(1, coordinate.length() - 1));
    }

    if (coordinate[0] == '/')
    {
        return root->get_by_coordinate(coordinate.substr(1, coordinate.length()
- 1));
    }

    std::string object_name;
    for (int i = 0; i < coordinate.length(); i++)
    {
        if (coordinate[i] != '/')
        {
            object_name += coordinate[i];
        }
        else if (get_child(object_name) == nullptr)
        {
            return nullptr;
        }
        else
        {
            return get_child(object_name)
->get_by_coordinate(
coordinate.substr(object_name.length() + 1,
coordinate.length() - 1));
        }
    }

    return get_child(object_name);
}

void cl_base::disconnect_child(std::string child_name)
{
    for (int i = 0; i < children.size(); i++)
    {
        if (children[i]->name == child_name)
        {

```

```

        children.erase(children.begin() + i);
        break;
    }
}

std::string cl_base::get_coordinate()
{
    std::string coordinate;

    cl_base* curr = this;
    while (curr->parent != nullptr)
    {
        coordinate = '/' + curr->name + coordinate;
        curr = curr->parent;
    }

    if (coordinate.length() == 0)
    {
        coordinate = "/";
    }

    return coordinate;
}

int cl_base::get_class_number()
{
    return class_number;
}

void cl_base::set_class_number(int number)
{
    class_number = number;
}

void cl_base::add_connection(TYPE_SIGNAL signal, cl_base* target,
TYPE_HANDLER handler)
{
    for (int i = 0; i < connections.size(); i++)
    {
        if (connections[i]->signal == signal && connections[i]->target ==
target
            && connections[i]->handler == handler)
        {
            return;
        }
    }

    connection* con = new connection;
    con->signal = signal;
    con->target = target;
    con->handler = handler;

    connections.push_back(con);
}

```

```

void cl_base::delete_connection(TYPE_SIGNAL signal, cl_base* target,
TYPE_HANDLER handler)
{
    for (int i = 0; i < connections.size(); i++)
    {
        if (connections[i]->signal == signal && connections[i]->target ==
target
        && connections[i]->handler == handler)
        {
            connections.erase(connections.begin() + i);
            return;
        }
    }
}

void cl_base::emit_signal(TYPE_SIGNAL signal, std::string& command)
{
    if (state == 0)
    {
        return;
    }

    TYPE_HANDLER handler;
    cl_base* target;
    (this->*signal)(command);

    for (int i = 0; i < connections.size(); i++)
    {
        if (connections[i]->signal == signal && connections[i]->target->state !
= 0)
        {
            handler = connections[i]->handler;
            target = connections[i]->target;
            (target->*handler)(command);
        }
    }
}

void cl_base::signal(std::string& message)
{
    std::cout << "\nSignal from " << get_coordinate();
    message += " (class: 1)";
}

void cl_base::handler(std::string message)
{
    std::cout << "\nSignal to " << get_coordinate() << " Text: " << message;
}

void cl_base::activate()
{
    set_state(1);
    for (int i = 0; i < children.size(); i++)
    {

```

```

        children[i]->activate();
    }
}

```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```

#ifndef CL_BASE_H
#define CL_BASE_H

#include <iostream>
#include <string>
#include <vector>

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HANDLER_D(handler_f) (TYPE_HANDLER) (&handler_f)

class cl_base;

typedef void (cl_base::*TYPE_SIGNAL)(std::string&);
typedef void (cl_base::*TYPE_HANDLER)(std::string);

struct connection
{
    TYPE_SIGNAL signal;
    cl_base* target;
    TYPE_HANDLER handler;
};

class cl_base
{
private:
    int state;

    // Наименование объекта
    std::string name;

    // Указатель на головной объект для текущего объекта
    cl_base* parent;

    // Динамический массив указателей на объекты,
    // подчиненные к текущему объекту в дереве иерархии
    std::vector<cl_base*> children;

    int class_number;

    std::vector<connection*> connections;

```

```

public:
    // Параметризированный конструктор
    cl_base(cl_base* parent, std::string name="");

    // Деструктор
    ~cl_base();

    // Метод редактирования имени объекта
    bool set_name(std::string new_name);

    // Метод получения имени объекта
    std::string get_name();

    // Метод получения указателя на головной объект текущего объекта
    cl_base* get_parent();

    // Метод вывода наименований объектов в дереве иерархии слева
    // направо и сверху вниз
    void print_tree(std::string prefix);

    // Метод получения указателя на непосредственно подчиненный
    // объект по его имени
    cl_base* get_child(std::string child_name);

    // Метод поиска объекта по заданному имени в дереве иерархии объектов
    cl_base* find_in_tree(std::string wanted_name);

    // Метод вывода наименований объектов в дереве иерархии слева
    // направо и сверху вниз вместе с их готовностью
    void print_tree_with_state(std::string prefix);

    // Метод получения состояния объекта
    int get_state();

    // Метод получения состояния объекта
    void set_state(int state);

    // Метод получения корня дерева
    cl_base* get_root();

    // Метод поиска по всему дереву из произвольной точки
    cl_base* find_in_tree_from_root(std::string wanted_name);

    // Метод подсчёта объектов в дереве с заданным именем
    int count_in_tree(std::string wanted_name);

    // Метод изменения головного объекта
    bool set_parent(cl_base* new_parent);

    // Метод удаления подчинённого объекта
    std::string delete_child(std::string child_name);

    // Метод получения объекта по координате
    cl_base* get_by_coordinate(std::string coordinate);

```

```

    void disconnect_child(std::string child_name);

    std::string get_coordinate();

    int get_class_number();
    void set_class_number(int number);

    void add_connection(TYPE_SIGNAL signal, cl_base* target, TYPE_HANDLER
handler);
    void delete_connection(TYPE_SIGNAL signal, cl_base* target, TYPE_HANDLER
handler);
    void emit_signal(TYPE_SIGNAL signal, std::string& command);
    void signal(std::string& message);
    void handler(std::string message);
    void activate();
};

#endif

```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```

#include "cl_application.h"

int main()
{
    cl_application ob_cl_application(nullptr);
    ob_cl_application.build_tree_objects();
    return ob_cl_application.exec_app();
}

```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 25.

Таблица 25 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
<pre> 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 </pre>	<pre> 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 </pre>	<pre> 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 </pre>

ЗАКЛЮЧЕНИЕ

В заключение данной курсовой работы можно с уверенностью сказать, что все поставленные задачи первых четырёх частей были успешно выполнены. В процессе разработки были реализованы четыре части, каждая из которых внесла свой вклад в расширение функциональности базового класса и обогащение объектно-ориентированного подхода.

Выполнение данной курсовой работы в частности и курс объектно-ориентированного программирования в целом принесли мне ценный опыт и практические навыки в области объектно-ориентированного программирования.

Во время изучения курса по ООП я узнал о ключевых принципах этого подхода, таких как инкапсуляция, наследование и полиморфизм. Я осознал важность разделения программы на независимые объекты, которые могут взаимодействовать друг с другом для выполнения конкретных задач. Я также научился проектировать классы и иерархии объектов, определять их свойства и методы, а также использовать наследование для повторного использования кода и создания иерархических структур.

Курс по ООП помог мне развить навыки абстрактного мышления и проектирования программных решений. Я узнал о важности модульности и расширяемости программного кода, что позволяет легко вносить изменения и добавлять новую функциональность без влияния на остальную часть программы.

Я научился применять основные концепции ООП на практике, создавать классы, методы, объекты и взаимодействовать с ними. Это позволило мне освоить инструменты и технологии, широко используемые в индустрии разработки программного обеспечения.

В результате изучения курса по ООП я получил ценные навыки, которые будут полезны для моей будущей карьеры. Кроме того, я получил хорошую

основу для изучения других программных парадигм и языков программирования, что позволит адаптироваться к изменяющимся требованиям индустрии и успешно развиваться в своей карьере в сфере программирования.

В рамках курса ООП и, в частности, написания курсовой работы, использовалась среда разработки "Аврора", которая является удобным и функциональным инструментом, который оказал мне большую помощь в освоении объектно-ориентированного программирования (ООП).

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-avvora.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-avvora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).
4. Шилдт Г. С++: 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).