Здесь будет титульник, листай ниже

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#__RefHeading___Toc13086_2623716724)

[1 ПОСТАНОВКА ЗАДАЧИ 7](#__RefHeading___Toc13088_2623716724)

[1.1 Описание входных данных 9](#__RefHeading___Toc13090_2623716724)

[1.2 Описание выходных данных 11](#__RefHeading___Toc13092_2623716724)

[2 МЕТОД РЕШЕНИЯ 13](#__RefHeading___Toc13094_2623716724)

[3 ОПИСАНИЕ АЛГОРИТМОВ 18](#__RefHeading___Toc13096_2623716724)

[3.1 Алгоритм функции main 18](#__RefHeading___Toc13098_2623716724)

[3.2 Алгоритм метода build\_tree\_objects класса cl\_application 18](#__RefHeading___Toc13100_2623716724)

[3.3 Алгоритм метода exec\_app класса cl\_application 20](#__RefHeading___Toc13102_2623716724)

[3.4 Алгоритм метода GetSignal класса cl\_application 22](#__RefHeading___Toc13104_2623716724)

[3.5 Алгоритм метода GetHandler класса cl\_application 23](#__RefHeading___Toc13106_2623716724)

[3.6 Алгоритм метода SetChildState класса cl\_base 24](#__RefHeading___Toc13108_2623716724)

[3.7 Алгоритм метода SetConnect класса cl\_base 24](#__RefHeading___Toc13110_2623716724)

[3.8 Алгоритм метода DeleteConnect класса cl\_base 25](#__RefHeading___Toc13112_2623716724)

[3.9 Алгоритм метода EmitSignal класса cl\_base 26](#__RefHeading___Toc13114_2623716724)

[3.10 Алгоритм метода GetFullPath класса cl\_base 27](#__RefHeading___Toc13116_2623716724)

[3.11 Алгоритм метода SetClassNum класса cl\_base 28](#__RefHeading___Toc13118_2623716724)

[3.12 Алгоритм метода GetClassNum класса cl\_base 28](#__RefHeading___Toc13120_2623716724)

[3.13 Алгоритм метода Signal класса cl\_1 29](#__RefHeading___Toc13122_2623716724)

[3.14 Алгоритм метода Handler класса cl\_1 29](#__RefHeading___Toc13124_2623716724)

[3.15 Алгоритм метода Signal класса cl\_2 30](#__RefHeading___Toc13126_2623716724)

[3.16 Алгоритм метода Handler класса cl\_2 30](#__RefHeading___Toc13128_2623716724)

[3.17 Алгоритм метода Signal класса cl\_3 30](#__RefHeading___Toc13130_2623716724)

[3.18 Алгоритм метода Handler класса cl\_3 31](#__RefHeading___Toc13132_2623716724)

[3.19 Алгоритм метода Signal класса cl\_4 31](#__RefHeading___Toc13134_2623716724)

[3.20 Алгоритм метода Handler класса cl\_4 32](#__RefHeading___Toc13136_2623716724)

[3.21 Алгоритм метода Signal класса cl\_5 32](#__RefHeading___Toc13138_2623716724)

[3.22 Алгоритм метода Handler класса cl\_5 32](#__RefHeading___Toc13140_2623716724)

[3.23 Алгоритм метода Signal класса cl\_6 33](#__RefHeading___Toc13142_2623716724)

[3.24 Алгоритм метода Handler класса cl\_6 33](#__RefHeading___Toc13144_2623716724)

[4 БЛОК-СХЕМЫ АЛГОРИТМОВ 34](#__RefHeading___Toc13146_2623716724)

[5 КОД ПРОГРАММЫ 47](#__RefHeading___Toc13148_2623716724)

[5.1 Файл cl\_1.cpp 47](#__RefHeading___Toc13150_2623716724)

[5.2 Файл cl\_1.h 47](#__RefHeading___Toc13152_2623716724)

[5.3 Файл cl\_2.cpp 48](#__RefHeading___Toc13154_2623716724)

[5.4 Файл cl\_2.h 48](#__RefHeading___Toc13156_2623716724)

[5.5 Файл cl\_3.cpp 48](#__RefHeading___Toc13158_2623716724)

[5.6 Файл cl\_3.h 49](#__RefHeading___Toc13160_2623716724)

[5.7 Файл cl\_4.cpp 49](#__RefHeading___Toc13162_2623716724)

[5.8 Файл cl\_4.h 50](#__RefHeading___Toc13164_2623716724)

[5.9 Файл cl\_5.cpp 50](#__RefHeading___Toc13166_2623716724)

[5.10 Файл cl\_5.h 51](#__RefHeading___Toc13168_2623716724)

[5.11 Файл cl\_6.cpp 51](#__RefHeading___Toc13170_2623716724)

[5.12 Файл cl\_6.h 51](#__RefHeading___Toc13172_2623716724)

[5.13 Файл cl\_application.cpp 52](#__RefHeading___Toc13174_2623716724)

[5.14 Файл cl\_application.h 56](#__RefHeading___Toc13176_2623716724)

[5.15 Файл cl\_base.cpp 56](#__RefHeading___Toc13178_2623716724)

[5.16 Файл cl\_base.h 63](#__RefHeading___Toc13180_2623716724)

[5.17 Файл main.cpp 64](#__RefHeading___Toc13182_2623716724)

[6 ТЕСТИРОВАНИЕ 65](#__RefHeading___Toc13184_2623716724)

[ЗАКЛЮЧЕНИЕ 69](#__RefHeading___Toc13186_2623716724)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 70](#__RefHeading___Toc13188_2623716724)

# ВВЕДЕНИЕ

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

**Актуальность**: реализация механизма взаимодействия объектов с использованием сигналов и обработчиков, передача текстовых сообщений вместе с сигналами - актуальная тема в настоящее время. Это позволяет гибко коммуницировать между компонентами системы и облегчает её расширение и сопровождение. Разработка такой системы с методами установки связи, удаления связи и выдачи сигнала с текстовыми переменными повышает эффективность и модульность кода.

**Цель работы**: получение практических навыков по разработке системы, состоящей из нескольких объектов, взаимодействующих между собой.

**Задачи**:

1. Использование на практике уже изученных основных принципов объектно-ориентированного программирования на C++
2. Разработка архитектуры системы
3. Реализация необходимых классов и методов для работы системы
4. Реализация необходимых сигналов и обработчиков
5. Тестирование работы системы на разных входных даннных
6. Написание метода решения, алгоритма и блок-схемы функционирования системы

# 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:
   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 то же, что и в работе КВ\_3, однако были изменены/добавлены несколько методов и классов.

В классе cl\_base были использованы параметризированные макроопределеня препроцессора для приведения указателя на метод сигнала и на метод обработчика, а также были объявлены новые типы данных TYPE\_SIGNAL и TYPE\_HANDLER для определения указателей на методы сигнала и обработчика.

Также была добавлена дополнительная структура o\_sh, представляющая собой пользовательский тип данных для хранения соединений "сигнал-обработчик".

* Класс cl\_application:
  + Свойства/поля:
    - Те же поля, что и в КВ\_3 у класса cl\_application;
  + Методы
    - Те же методы, что и в КВ\_3 у класса cl\_application;
    - Метод GetSignal
      * Функционал - используется для получения указателя на метод сигнала определенного класса;
    - Метод GetHandler
      * Функционал - используется для получения указателя на метод обработчика определенного класса;
* Класс cl\_base:
  + Свойства/поля:
    - Те же поля, что и в КВ\_3 у класса cl\_base;
    - Поле, отвечающее за хранение соединений "сигнал-обработчик" для определенного объекта
      * Наименование - connects;
      * Тип - вектор объектов структуры o\_sh;
      * Модификатор доступа - private;
    - Поле, отвечающее за хранение номера класса
      * Наименование - classNum;
      * Тип - int;
      * Модификатор доступа - private;
  + Методы
    - Те же методы, что и в КВ\_3 у класса cl\_base;
    - Метод SetChildState
      * Функционал - используется для смены состояний всех подчиненных объектов текущего на значение параметра;
    - Метод SetConnect
      * Функционал - используется для установления связи между сигналом текущего объекта и обработчиком целевого объекта;
    - Метод DeleteConnect
      * Функционал - используется для удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
    - Метод EmitSignal
      * Функционал - используется для выдачи сигнала от текущего объекта с передачей строковой переменной;
    - Метод GetFullPath
      * Функционал - используется для возращения полного пути текущего объекта иерархии;
    - Метод SetClassNum
      * Функционал - используется для установки номера класса текущего объекта;
    - Метод GetClassNum
      * Функционал - используется для получения номера класса текущего объекта;
* Класс cl\_1:
  + Свойства/поля:
    - Те же поля, что и в КВ\_3 у класса cl\_1;
  + Методы
    - Метод Signal
      * Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      * Функционал - используется как обработчик текущего объекта;
* Класс cl\_2:
  + Свойства/поля:
    - Те же поля, что и в КВ\_3 у класса cl\_2;
  + Методы
    - Метод Signal
      * Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      * Функционал - используется как обработчик текущего объекта;
* Класс cl\_3:
  + Свойства/поля:
    - Те же поля, что и в КВ\_3 у класса cl\_3;
  + Методы
    - Метод Signal
      * Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      * Функционал - используется как обработчик текущего объекта;
* Класс cl\_4:
  + Свойства/поля:
    - Те же поля, что и в КВ\_3 у класса cl\_4;
  + Методы
    - Метод Signal
      * Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      * Функционал - используется как обработчик текущего объекта;
* Класс cl\_5:
  + Свойства/поля:
    - Те же поля, что и в КВ\_3 у класса cl\_5;
  + Методы
    - Метод Signal
      * Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      * Функционал - используется как обработчик текущего объекта;
* Класс cl\_6:
  + Свойства/поля:
    - Те же поля, что и в КВ\_3 у класса cl\_6;
  + Методы
    - Метод Signal
      * Функционал - используется как сигнал текущего объекта;
    - Метод Handler
      * Функционал - используется как обработчик текущего объекта;

# 3 ОПИСАНИЕ АЛГОРИТМОВ

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

## Алгоритм функции main

Функционал: Основной алгоритм работы программы.

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения программы.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Выполнения алгоритма данной функции из предыдущей работы КВ\_3 | ∅ |

## Алгоритм метода build\_tree\_objects класса cl\_application

Функционал: Построение исходного дерева иерархии объектов.

Параметры: Отсутствуют.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода build\_tree\_objects класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление строчных переменных objPath и childName | 2 |
| 2 |  | Объявление целочисленной переменной state | 3 |
| 3 |  | Инициализация целочисленной переменной classNum значением 1 | 4 |
| 4 |  | Ввод значения переменной objPath | 5 |
| 5 |  | Вызов метода SetName, с аргументом значения переменной objPath | 6 |
| 6 |  | Вызов метода SetClassNum, с аргументом значения переменной childName | 7 |
| 7 |  | Очистка потока ввода | 8 |
| 8 |  | Ввод значения переменной objPath | 9 |
| 9 | Значение objPath равно "endtree" |  | 18 |
|  | Инициализация указателя parentObj на объект класса cl\_base результатом выполнения метода GetObjectByPath, с аргументом в виде значения objPath | 10 |
| 10 | Значение parentObj равно nullptr | Вывод "Object tree" на экран | 11 |
|  | Ввод значений переменных childName и classNum | 14 |
| 11 |  | Вызов метода PrintObjects | 12 |
| 12 |  | Вывод "\nThe head object ", objPath и " is not found" на экран | 13 |
| 13 |  | Завершение работы программы с значением 1 | ∅ |
| 14 |  | Вызов метода GetChild, с аргументом childName, через указатель на объект parentObj | 15 |
| 15 | Возвращаемое значение метода GetChild равно nullptr | Инициализация указателя childObj на объект класса cl\_base значением nullptr | 16 |
|  | Вывод objPath, " Dubbing the names of subordinate objects" на экран | 7 |
| 16 | Значение classNum равно i | Присваивание childObj значение создаваемого объекта cl\_i-того класса, с аргументами parentObj и childName | 17 |
|  |  | 17 |
| 17 | Значение childObj не равно nullptr | Вызов метода SetClassNum, с аргументом classNum, через указатель на объект childObj | 7 |
|  |  | 7 |
| 18 | Значение objPath не равно "end\_of\_connections" | Ввод значения переменной objPath | 19 |
|  |  | ∅ |
| 19 | Значение objPath не равно "end\_of\_connections" | Инициализация указателя headPtr на объект класса cl\_base результатом выполнения метода  GetObjectByPath, с аргументом в виде значения objPath | 20 |
|  |  | 18 |
| 20 |  | Ввод значения переменной objPath | 21 |
| 21 |  | Инициализация указателя childPtr на объект класса cl\_base результатом выполнения метода  GetObjectByPath, с аргументом в виде значения objPath | 22 |
| 22 |  | Вызов метода SetConnect указателя headPtr, с аргументами в виде результатов выполнения  методов GetSignal, с аргументом headPtr и GetHandler, с аргументом childPtr и указателя  childPtr | 18 |

## Алгоритм метода exec\_app класса cl\_application

Функционал: Запуск приложения (начало функционирования системы).

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения метода.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода exec\_app класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод "Object tree" на экран | 2 |
| 2 |  | Вызов метода PrintObjects текущего объекта | 3 |
| 3 |  | Вызов метода SetChildState, с аргуметом 1, текущего объекта | 4 |
| 4 |  | Объявление указателей obj и temp на объект класса cl\_base | 5 |
| 5 |  | Объявление строчных переменных command, message и objPath | 6 |
| 6 |  | Ввод значения переменной command | 7 |
| 7 | Значение command равно "END" | Возвращение значения 0 | ∅ |
|  | Ввод значения переменной objPath | 8 |
| 8 |  | Присваивание указателю obj значение выполнения метода GetObjectByPath, с аргументом в виде значения objPath | 9 |
| 9 | Значение obj равно nullptr | Вывод "Object ", objPath, " not found" на экран | 6 |
|  | Считывание всей строки ввода для записии её в переменную message | 10 |
| 10 | Значение command равно "EMIT" |  | 11 |
| Значение command равно "SET\_CONDITION" | Вызов метода SetObjectState через указатель на объект obj, с аргументом в виде целой части строковой переменной message | 6 |
|  |  | 13 |
| 11 |  | Вызов метода GetObjectState через указатель на объект obj | 12 |
| 12 | Возвращаемое значение метода GetObjectState равно true | Вызов метода EmitSignal указателя obj, с аргументами message и результатом выполнения  метода GetSignal, с аргументом obj | 6 |
|  |  | 6 |
| 13 |  | Присваивание message значение message без первого символа | 14 |
| 14 |  | Присваивание указателю temp значение метода GetObjectByPath, с аргументом в виде значения message | 15 |
| 15 | Значение temp равно nullptr | Вывод "Handler object ", message, " not found" на экран | 6 |
| Значение command равно "SET\_CONNECT" | Вызов метода SetConnect через указатель на объект obj, с аргументами в ввиде результатов выполнения методов GetSignal, с аргументом obj, и GetHandler, с аргументом temp, и указателя temp | 6 |
| Значение command равно "DELETE\_CONNECT" | Вызов метода DeleteConnect через указатель на объект obj, с аргументами в ввиде результатов выполнения методов GetSignal, с аргументом obj, и GetHandler, с аргументом temp, и указателя temp | 6 |
|  |  | 6 |

## Алгоритм метода GetSignal класса cl\_application

Функционал: Возврат указателя на метод сигнала одного из подчиненных классов класса cl\_base.

Параметры: cl\_base\* object - указатель на искомый объект класса cl\_base.

Возвращаемое значение: TYPE\_SIGNAL - указатель на метод сигнала одного из подчиненных классов класса cl\_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода GetSignal класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов метода GetClassNum через указатель на объект object | 2 |
| 2 | Возвращаемое значение метода GetClassNum равно i-тому значению | Возврат указателя на метод Signal cl\_i-того класса | ∅ |
|  | Возврат nullptr | ∅ |

## Алгоритм метода GetHandler класса cl\_application

Функционал: Возврат указателя на метод обработчика одного из подчиненных классов класса cl\_base.

Параметры: cl\_base\* object - указатель на искомый объект класса cl\_base.

Возвращаемое значение: TYPE\_HANDLER - указатель на метод обработчика одного из подчиненных классов класса cl\_base.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода GetHandler класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов метода GetClassNum через указатель на объект object | 2 |
| 2 | Возвращаемое значение метода GetClassNum равно i-тому значению | Возврат указателя на метод Handler cl\_i-того класса | ∅ |
|  | Возврат nullptr | ∅ |

## Алгоритм метода SetChildState класса cl\_base

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

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

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода SetChildState класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов метода SetObjectState, с аргументом state | 2 |
| 2 | Переменная child списка children не равна nullptr | Вызов метода SetChildState, с аргументом state | ∅ |
|  |  | ∅ |

## Алгоритм метода SetConnect класса cl\_base

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

Параметры: TYPE\_SIGNAL signal - указатель на метод сигнала текущего объекта, cl\_base\* object - указатель на целевой объект, TYPE\_HANDLER handler - указатель на метод обработчика целевого объекта.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода SetConnect класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Ссылка на переменную pos списка connects не равна nullptr |  | 2 |
|  |  | 3 |
| 2 | Поля Signal, Handler, Object переменной pos равны параметрам signal, object, handler |  | ∅ |
|  |  | 1 |
| 3 |  | Объявление объекта obj структуры o\_sh | 4 |
| 4 |  | Присваивание полю Signal объекта obj значение параметра signal | 5 |
| 5 |  | Присваивание полю Handler объекта obj значение параметра handler | 6 |
| 6 |  | Присваивание полю Object объекта obj значение параметра object | 7 |
| 7 |  | Добавление в конец списка connects объект obj | ∅ |

## Алгоритм метода DeleteConnect класса cl\_base

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

Параметры: TYPE\_SIGNAL signal - указатель на метод сигнала текущего объекта, cl\_base\* object - указатель на целевой объект, TYPE\_HANDLER handler - указатель на метод обработчика целевого объекта.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода DeleteConnect класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация итератора i значением указателя на начальный элемент | 2 |
| 2 | i не равен значению указателя на элемент после  последнего |  | 3 |
|  |  | ∅ |
| 3 | Поля Signal, Handler, Object значения i равны параметрам  signal, object, handler | Удаление элемента, на который указывает i, из connects | ∅ |
|  |  | 2 |

## Алгоритм метода EmitSignal класса cl\_base

Функционал: Выдача сигнала от текущего объекта.

Параметры: TYPE\_SIGNAL signal - указатель на метод сигнала текущего объекта, string& - сообщение.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода EmitSignal класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов метода сигнала, указатель на который хранит signal, с аргументом message | 2 |
| 2 | Ссылка на переменную pos списка connects не равна  nullptr |  | 3 |
|  |  | ∅ |
| 3 | Поле Signal переменной pos равно signal | Инициализация указателя handler на объект типа TYPE\_HANDLER значением поля Handler переменной pos | 4 |
|  |  | 2 |
| 4 |  | Инициализация указателя obj на объект класса cl\_base значением поля Object переменной pos | 5 |
| 5 |  | Вызов метода GetObjectState через указатель на объект obj | 6 |
| 6 | Возвращаемое значение метода GetObjectState равно true | Вызов метода обработчика, указатель на который хранит handler, с аргументом message | 2 |
|  |  | 2 |

## Алгоритм метода GetFullPath класса cl\_base

Функционал: Возращение полного пути текущего объекта иерархии.

Параметры: Отсутствуют.

Возвращаемое значение: string - полный путь текущего объекта.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода GetFullPath класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление строковой переменной path | 2 |
| 2 |  | Инициализация указателя temp на объект класса cl\_base значением указателя на текущий объект | 3 |
| 3 |  | Вызов метода GetParent через указатель на объект temp | 4 |
| 4 | Возвращаемое значение метода GetParent равно nullptr | Возврат "/" | ∅ |
|  |  | 5 |
| 5 |  | Вызов метода GetParent через указатель на объект temp | 6 |
| 6 | Возвращаемое значение метода GetParen не равно nullptr | Присваивание path "/", наименование объекта указателя temp и значение path | 7 |
|  | Возврат path | ∅ |
| 7 |  | Присваивание temp результата выполнения метода GetParent, вызванного через указатель temp | 6 |

## Алгоритм метода SetClassNum класса cl\_base

Функционал: Установка номера класса текущего объекта.

Параметры: int num - значение номера класса текущего объекта.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода SetClassNum класса cl\_base

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

## Алгоритм метода GetClassNum класса cl\_base

Функционал: Получение номера класса текущего объекта.

Параметры: Отсутствуют.

Возвращаемое значение: int - значение номера класса текущего объекта.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода GetClassNum класса cl\_base

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

## Алгоритм метода Signal класса cl\_1

Функционал: Метод сигнала текущего объекта.

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

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода Signal класса cl\_1

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

## Алгоритм метода Handler класса cl\_1

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода Handler класса cl\_1

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

## Алгоритм метода Signal класса cl\_2

Функционал: Метод сигнала текущего объекта.

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

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода Signal класса cl\_2

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

## Алгоритм метода Handler класса cl\_2

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода Handler класса cl\_2

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

## Алгоритм метода Signal класса cl\_3

Функционал: Метод сигнала текущего объекта.

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

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода Signal класса cl\_3

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

## Алгоритм метода Handler класса cl\_3

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода Handler класса cl\_3

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

## Алгоритм метода Signal класса cl\_4

Функционал: Метод сигнала текущего объекта.

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

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода Signal класса cl\_4

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

## Алгоритм метода Handler класса cl\_4

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода Handler класса cl\_4

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

## Алгоритм метода Signal класса cl\_5

Функционал: Метод сигнала текущего объекта.

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

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода Signal класса cl\_5

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

## Алгоритм метода Handler класса cl\_5

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода Handler класса cl\_5

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

## Алгоритм метода Signal класса cl\_6

Функционал: Метод сигнала текущего объекта.

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

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода Signal класса cl\_6

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

## Алгоритм метода Handler класса cl\_6

Функционал: Метод обработчика текущего объекта.

Параметры: string data - текст сообщения.

Возвращаемое значение: Отсутствует.

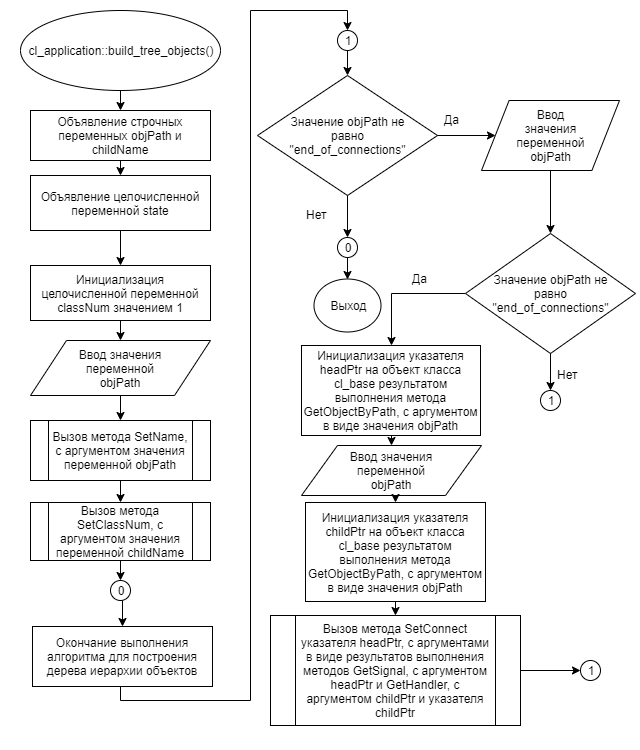
Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода Handler класса cl\_6

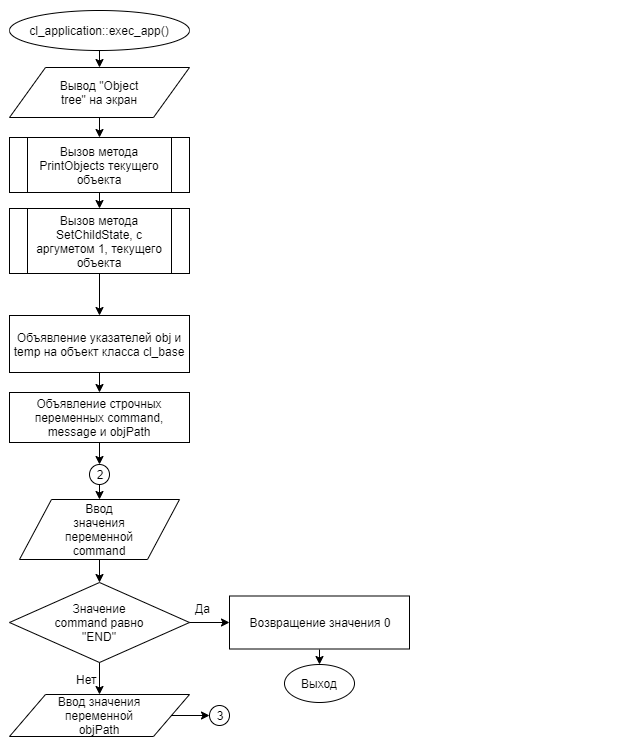
| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод "Signal to ", результата выполнения метода GetFullPath и " Text: " с значением data | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

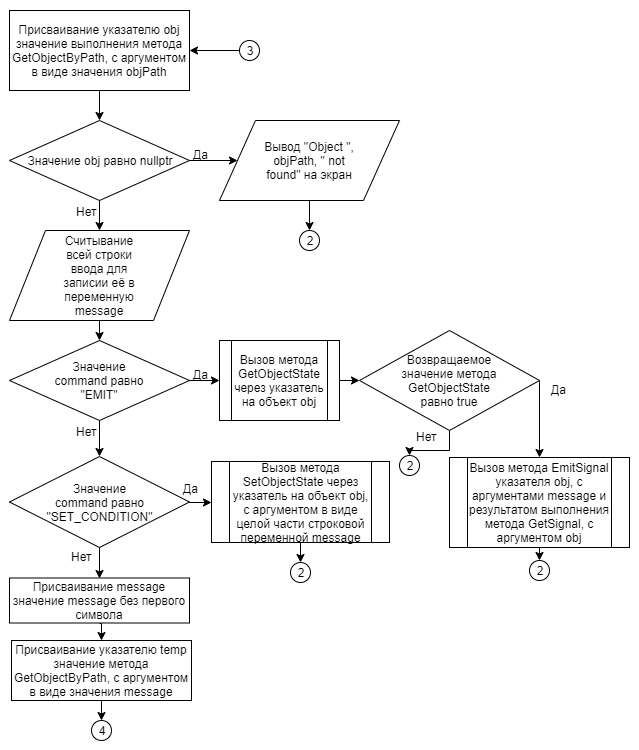
Представим описание алгоритмов в графическом виде на рисунках 1-13.



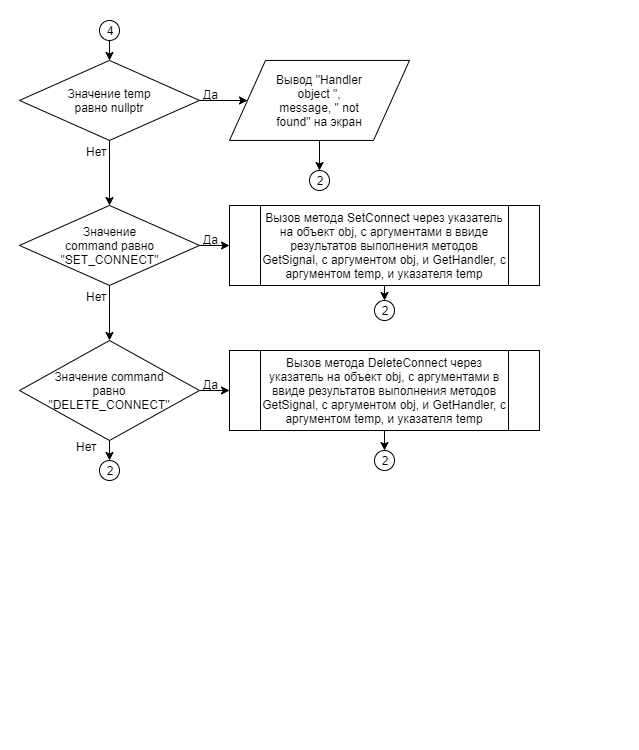
**Рисунок 1 – Блок-схема алгоритма**



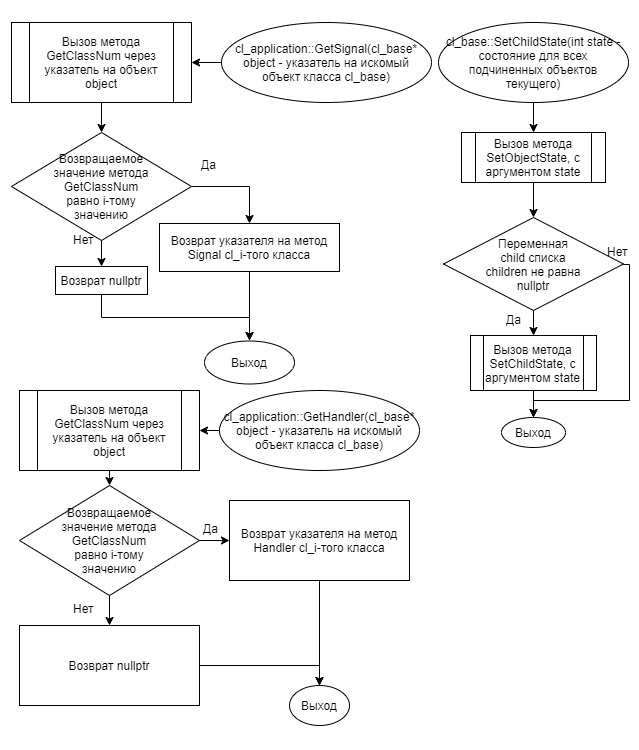
**Рисунок 2 – Блок-схема алгоритма**



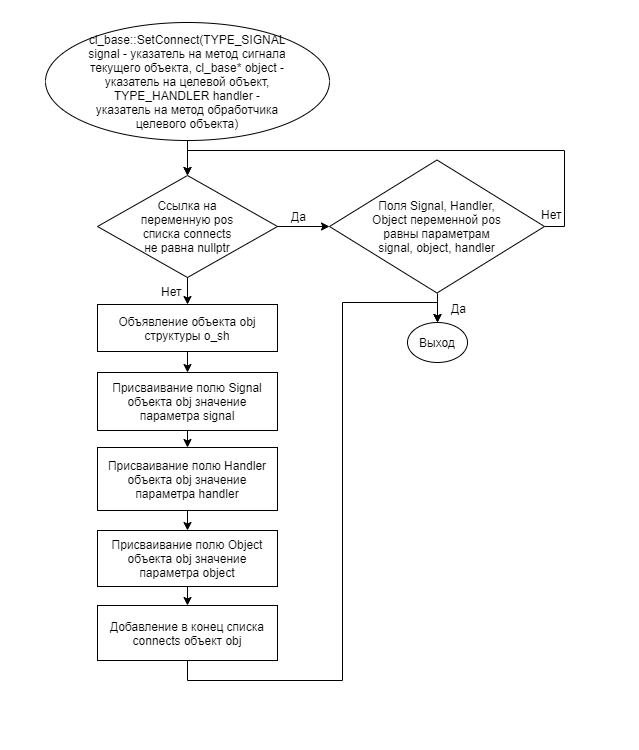
**Рисунок 3 – Блок-схема алгоритма**



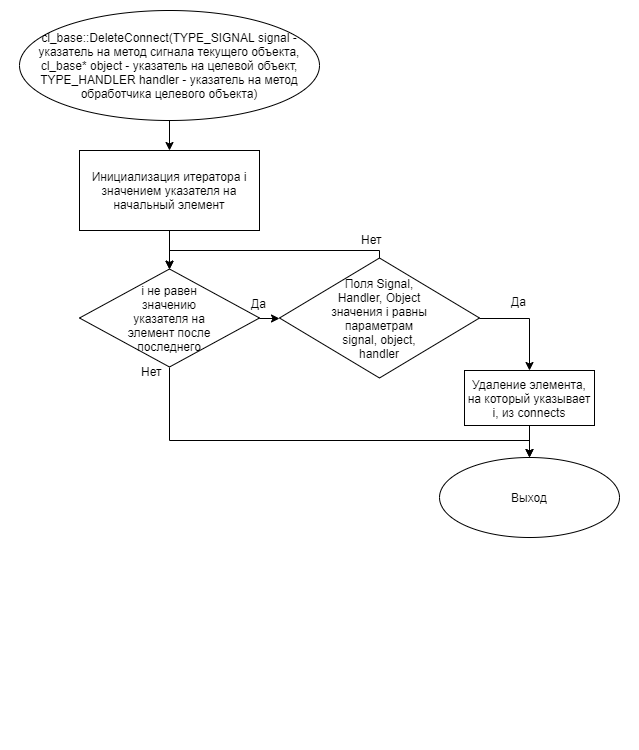
**Рисунок 4 – Блок-схема алгоритма**



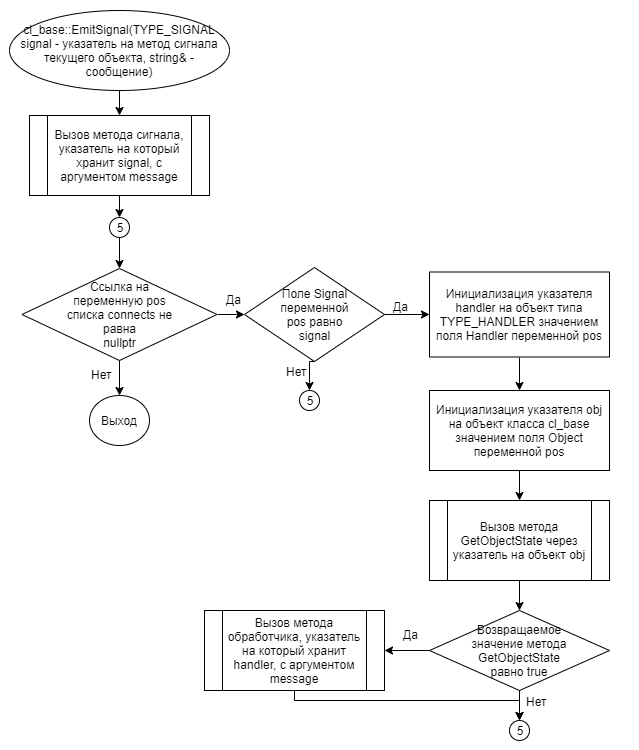
**Рисунок 5 – Блок-схема алгоритма**



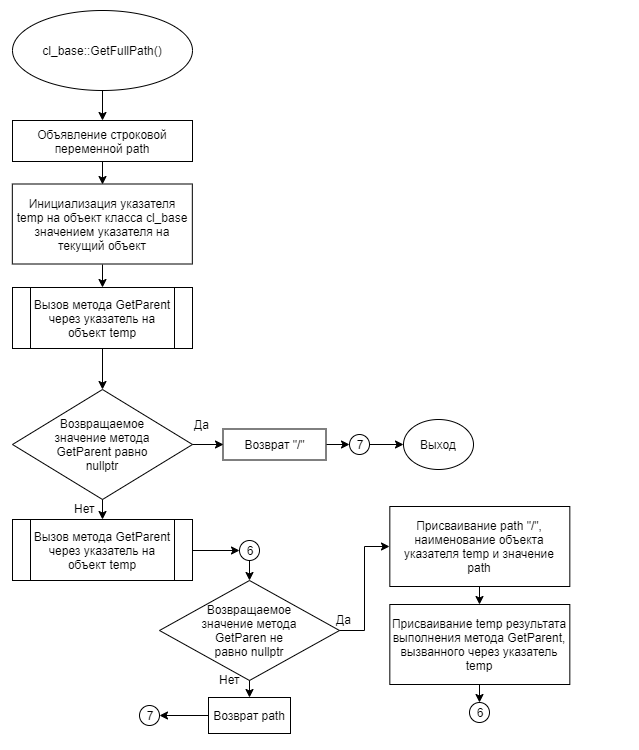
**Рисунок 6 – Блок-схема алгоритма**



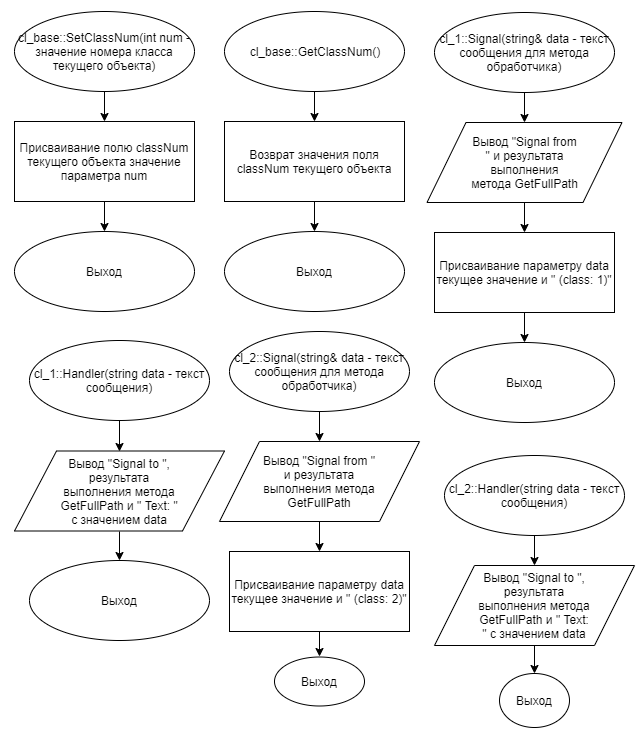
**Рисунок 7 – Блок-схема алгоритма**



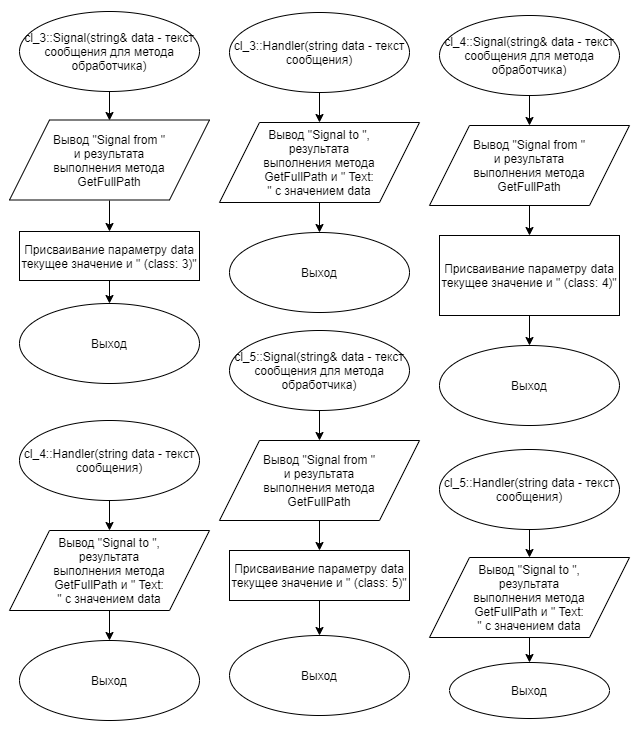
**Рисунок 8 – Блок-схема алгоритма**



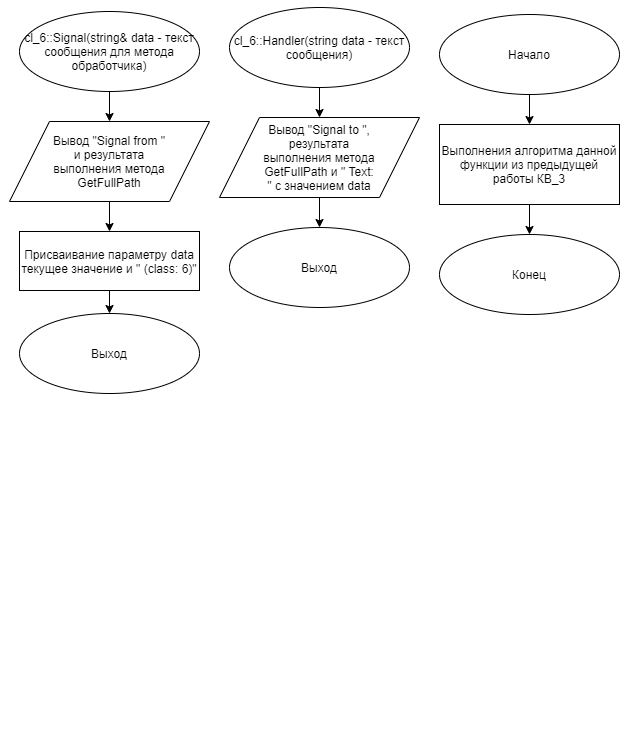
**Рисунок 9 – Блок-схема алгоритма**



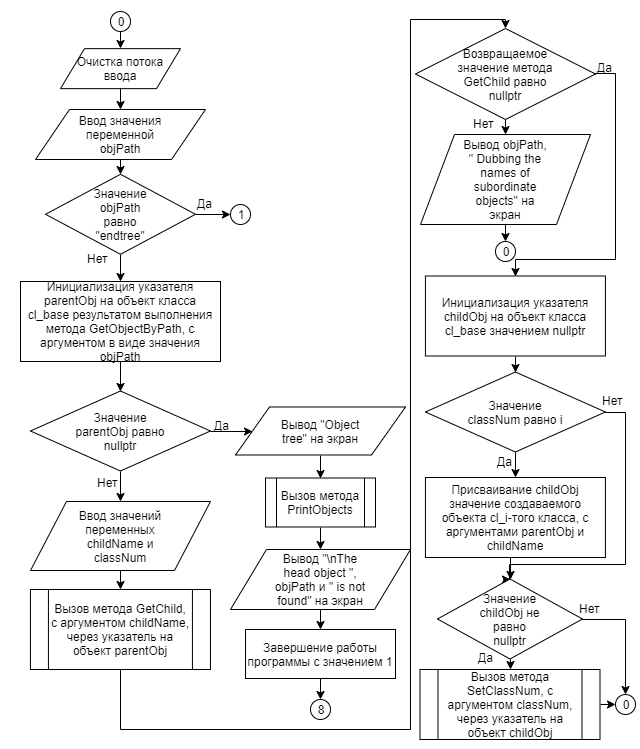
**Рисунок 10 – Блок-схема алгоритма**



**Рисунок 11 – Блок-схема алгоритма**



**Рисунок 12 – Блок-схема алгоритма**



**Рисунок 13 – Блок-схема алгоритма**

# 5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

## **Файл** **cl\_1.cpp**

Листинг 1 – cl\_1.cpp

|  |
| --- |
| #include "cl\_1.h"  cl\_1::cl\_1(cl\_base\* parent, string name) : cl\_base(parent, name)  {}  // Метод сигнала  void cl\_1::Signal(string& dat)  {  cout << endl << "Signal from " + GetFullPath();  dat = dat + " (class: 1)";  }  // Метод обработчика  void cl\_1::Handler(string dat)  {  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;  } |

## **Файл** **cl\_1.h**

Листинг 2 – cl\_1.h

|  |
| --- |
| #ifndef \_\_CL\_1\_\_H  #define \_\_CL\_1\_\_H  #include "cl\_base.h"  class cl\_1 : public cl\_base  {  public:  cl\_1(cl\_base\* parent, string name);  // КВ4  // Сигнал  void Signal(string& dat);  // Обработчик  void Handler(string dat);  };  #endif |

## **Файл** **cl\_2.cpp**

Листинг 3 – cl\_2.cpp

|  |
| --- |
| #include "cl\_2.h"  cl\_2::cl\_2(cl\_base\* parent, string name) : cl\_base(parent, name)  {}  void cl\_2::Signal(string& dat)  {  cout << endl << "Signal from " + GetFullPath();  dat = dat + " (class: 2)";  }  void cl\_2::Handler(string dat)  {  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;  } |

## **Файл** **cl\_2.h**

Листинг 4 – cl\_2.h

|  |
| --- |
| #ifndef \_\_CL\_2\_\_H  #define \_\_CL\_2\_\_H  #include "cl\_base.h"  class cl\_2 : public cl\_base  {  public:  cl\_2(cl\_base\* parent, string name);  // КВ4  // Сигнал  void Signal(string& dat);  // Обработчик  void Handler(string dat);  };  #endif |

## **Файл** **cl\_3.cpp**

Листинг 5 – cl\_3.cpp

|  |
| --- |
| #include "cl\_3.h"  cl\_3::cl\_3(cl\_base\* parent, string name) : cl\_base(parent, name)  {}  void cl\_3::Signal(string& dat)  {  cout << endl << "Signal from " + GetFullPath();  dat = dat + " (class: 3)";  }  void cl\_3::Handler(string dat)  {  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;  } |

## **Файл** **cl\_3.h**

Листинг 6 – cl\_3.h

|  |
| --- |
| #ifndef \_\_CL\_3\_\_H  #define \_\_CL\_3\_\_H  #include "cl\_base.h"  class cl\_3 : public cl\_base  {  public:  cl\_3(cl\_base\* parent, string name);  // КВ4  // Сигнал  void Signal(string& dat);  // Обработчик  void Handler(string dat);  };  #endif |

## **Файл** **cl\_4.cpp**

Листинг 7 – cl\_4.cpp

|  |
| --- |
| #include "cl\_4.h"  cl\_4::cl\_4(cl\_base\* parent, string name) : cl\_base(parent, name)  {}  void cl\_4::Signal(string& dat)  {  cout << endl << "Signal from " + GetFullPath();  dat = dat + " (class: 4)";  }  void cl\_4::Handler(string dat)  {  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;  } |

## **Файл** **cl\_4.h**

Листинг 8 – cl\_4.h

|  |
| --- |
| #ifndef \_\_CL\_4\_\_H  #define \_\_CL\_4\_\_H  #include "cl\_base.h"  class cl\_4 : public cl\_base  {  public:  cl\_4(cl\_base\* parent, string name);  // КВ4  // Сигнал  void Signal(string& dat);  // Обработчик  void Handler(string dat);  };  #endif |

## **Файл** **cl\_5.cpp**

Листинг 9 – cl\_5.cpp

|  |
| --- |
| #include "cl\_5.h"  cl\_5::cl\_5(cl\_base\* parent, string name) : cl\_base(parent, name)  {}  void cl\_5::Signal(string& dat)  {  cout << endl << "Signal from " + GetFullPath();  dat = dat + " (class: 5)";  }  void cl\_5::Handler(string dat)  {  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;  } |

## **Файл** **cl\_5.h**

Листинг 10 – cl\_5.h

|  |
| --- |
| #ifndef \_\_CL\_5\_\_H  #define \_\_CL\_5\_\_H  #include "cl\_base.h"  class cl\_5 : public cl\_base  {  public:  cl\_5(cl\_base\* parent, string name);  // КВ4  // Сигнал  void Signal(string& dat);  // Обработчик  void Handler(string dat);  };  #endif |

## **Файл** **cl\_6.cpp**

Листинг 11 – cl\_6.cpp

|  |
| --- |
| #include "cl\_6.h"  cl\_6::cl\_6(cl\_base\* parent, string name) : cl\_base(parent, name)  {}  void cl\_6::Signal(string& dat)  {  cout << endl << "Signal from " + GetFullPath();  dat = dat + " (class: 6)";  }  void cl\_6::Handler(string dat)  {  cout << endl << "Signal to " + GetFullPath() + " Text: " + dat;  } |

## **Файл** **cl\_6.h**

Листинг 12 – cl\_6.h

|  |
| --- |
| #ifndef \_\_CL\_6\_\_H  #define \_\_CL\_6\_\_H  #include "cl\_base.h"  class cl\_6 : public cl\_base  {  public:  cl\_6(cl\_base\* parent, string name);  // КВ4  // Сигнал  void Signal(string& dat);  // Обработчик  void Handler(string dat);  };  #endif |

## **Файл** **cl\_application.cpp**

Листинг 13 – cl\_application.cpp

|  |
| --- |
| #include "cl\_application.h"  cl\_application::cl\_application(cl\_base\* parent) : cl\_base(parent)  {}  void cl\_application::build\_tree\_objects()  {  string objPath, childName;  int state;  int classNum = 1;  cin >> objPath;  SetName(objPath);  SetClassNum(classNum);  while(true)  {  cin.clear();  cin >> objPath;  if (objPath == "endtree")  {  break;  }  cl\_base\* parentObj = GetObjectByPath(objPath);  // Если указатель головной не найден  if (!parentObj)  {  // Выводим готовое дерево иерархии и завершаем выполнение программы  cout << "Object tree" << endl;  PrintObjects();  cout << "\nThe head object " << objPath << " is not found";  exit(1);  }  cin >> childName >> classNum;  // Если у головного объекта в подчиненных нет объекта с наименованием childName  if (!parentObj->GetChild(childName))  {  cl\_base\* childObj = nullptr;  switch(classNum)  {  case 2:  {  childObj = new cl\_2(parentObj, childName);  break;  }  case 3:  {  childObj = new cl\_3(parentObj, childName);  break;  }  case 4:  {  childObj = new cl\_4(parentObj, childName);  break;  }  case 5:  {  childObj = new cl\_5(parentObj, childName);  break;  }  case 6:  {  childObj = new cl\_6(parentObj, childName);  break;  }  }  if (childObj != nullptr) childObj->SetClassNum(classNum);  }  else  {  // Если обнаружен дубликат имени  cout << objPath << " Dubbing the names of subordinate objects" << endl;  }  }  while(objPath != "end\_of\_connections")  {  cin >> objPath;  if (objPath != "end\_of\_connections")  {  // Находим два пути  cl\_base\* headPtr = GetObjectByPath(objPath);  cin >> objPath;  cl\_base\* childPtr = GetObjectByPath(objPath);  // Создаем соединение  headPtr->SetConnect(GetSignal(headPtr), childPtr,  GetHandler(childPtr));  }  }  }  int cl\_application::exec\_app()  {  // Вывод готового дерева  cout << "Object tree" << endl;  PrintObjects();  SetChildState(1);  cl\_base\* obj;  cl\_base\* temp;  string command, message, objPath;  while(true)  {  // Ввод команды для взаимодействия с объектами дерева  cin >> command;  if (command == "END")  {  break;  }  // Ввод пути объекта  cin >> objPath;  // Находим объект по введенному пути  obj = GetObjectByPath(objPath);  if (obj == nullptr)  {  cout << endl << "Object " << objPath << " not found";  continue;  }  // Считываем всю строку  getline(cin, message);  if (command == "EMIT")  {  if (obj->GetObjectState())  {  // Посылаем сигнал  obj->EmitSignal(GetSignal(obj), message);  }  }  else if (command == "SET\_CONDITION")  {  obj->SetObjectState(stoi(message));  }  else  {  // Удаляем первый символ, т.к пробел  message = message.substr(1);  temp = GetObjectByPath(message);  if (temp == nullptr)  {  cout << endl << "Handler object " << message << " not found";  continue;  }  if (command == "SET\_CONNECT")  {  obj->SetConnect(GetSignal(obj), temp, GetHandler(temp));  }  else if (command == "DELETE\_CONNECT")  {  obj->DeleteConnect(GetSignal(obj), temp,  GetHandler(temp));  }  }  }  return 0;  }  // КВ4  // Определяем нужный метод сигнала  TYPE\_SIGNAL cl\_application::GetSignal(cl\_base\* object)  {  switch(object->GetClassNum())  {  case 1:  return SIGNAL\_D(cl\_1::Signal);  break;  case 2:  return SIGNAL\_D(cl\_2::Signal);  break;  case 3:  return SIGNAL\_D(cl\_3::Signal);  break;  case 4:  return SIGNAL\_D(cl\_4::Signal);  break;  case 5:  return SIGNAL\_D(cl\_5::Signal);  break;  case 6:  return SIGNAL\_D(cl\_6::Signal);  break;  }  return nullptr;  }  // Определяем нужный метод обработчика  TYPE\_HANDLER cl\_application::GetHandler(cl\_base\* object)  {  switch(object->GetClassNum())  {  case 1:  return HANDLER\_D(cl\_1::Handler);  break;  case 2:  return HANDLER\_D(cl\_2::Handler);  break;  case 3:  return HANDLER\_D(cl\_3::Handler);  break;  case 4:  return HANDLER\_D(cl\_4::Handler);  break;  case 5:  return HANDLER\_D(cl\_5::Handler);  break;  case 6:  return HANDLER\_D(cl\_6::Handler);  break;  }  return nullptr;  } |

## **Файл** **cl\_application.h**

Листинг 14 – cl\_application.h

|  |
| --- |
| #ifndef CL\_APPLICATION\_H  #define CL\_APPLICATION\_H  #include "cl\_base.h"  #include "cl\_1.h"  #include "cl\_2.h"  #include "cl\_3.h"  #include "cl\_4.h"  #include "cl\_5.h"  #include "cl\_6.h"  class cl\_application : public cl\_base  {  public:  cl\_application(cl\_base\* parent);  void build\_tree\_objects();  int exec\_app();  // КВ4  TYPE\_SIGNAL GetSignal(cl\_base\*);  TYPE\_HANDLER GetHandler(cl\_base\*);  };  #endif |

## **Файл** **cl\_base.cpp**

Листинг 15 – cl\_base.cpp

|  |
| --- |
| #include "cl\_base.h"  cl\_base::cl\_base(cl\_base\* parent, string name)  {  this->parent = parent;  this->name = name;  if (GetParent() != nullptr)  {  GetParent()->children.push\_back(this);  }  }  cl\_base::~cl\_base()  {  for (auto child : children)  {  delete child;  }  }  bool cl\_base::SetName(string newName)  {  // Если родитель для текущего найден и  // он не имеет подчиненнго с именем newName  if(GetParent() != nullptr && GetParent()->GetChild(newName) != nullptr)  {  return false;  }  name = newName;  return true;  }  string cl\_base::GetName() const  {  return name;  }  cl\_base\* cl\_base::GetParent() const  {  return parent;  }  cl\_base\* cl\_base::GetChild(string objName) const  {  // Проход по всем подчиненным объектам  for (auto child : children)  {  if (child->GetName() == objName)  {  return child;  }  }  return nullptr;  }  //КВ2  // Поиск количества объектов на ветке по имени  // через рекурсию  int cl\_base::ObjNameCount(string objName)  {  int count = 0;  if(GetName() == objName)  {  count++;  }  for (auto child : children)  {  count += child->ObjNameCount(objName);  }  return count;  }  // Проверка на уникальность в ветке по имени  cl\_base\* cl\_base::CheckingObjUniq(string objName)  {  if (ObjNameCount(objName) != 1)  {  return nullptr;  }  return SearchObjOnBranch(objName);  }  cl\_base\* cl\_base::SearchObjOnBranch(string objName)  {  // Если имя объекта равно параметру objname  if (GetName() == objName)  {  return this;  }  for (auto child : children)  {  cl\_base\* subChild = child->SearchObjOnBranch(objName);  // Если объект не равен nullptr  if (subChild)  {  return subChild;  }  }  return nullptr;  }  // Поиск по всему дереву от корневого  cl\_base\* cl\_base::SearchObjOnTree(string objName)  {  return GetRoot()->CheckingObjUniq(objName);  }  //Вывод  void cl\_base::PrintObjects(int spaces) const  {  cout << GetName();  if (!children.empty())  {  for (auto child : children)  {  cout << endl;  for (int i = 0; i < spaces; i++)  cout << " ";  child->PrintObjects(spaces+4);  }  }  }  void cl\_base::PrintObjectsStates(int spaces) const  {  cout << GetName();  cout << (GetObjectState() ? " is ready" : " is not ready");  if (!children.empty())  {  for (auto child : children)  {  cout << endl;  for (int i = 0; i < spaces; i++)  cout << " ";  child->PrintObjectsStates(spaces+4);  }  }  }  //Состояние объекта  void cl\_base::SetObjectState(bool state)  {  if (GetParent() && !GetParent()->GetObjectState())  {  this->state = false;  }  else  {  this->state = state;  }  if (!state)  {  for (auto child : children)  {  child->SetObjectState(state);  }  }  }  bool cl\_base::GetObjectState() const  {  return state;  }  //КВ3  cl\_base\* cl\_base::GetRoot()  {  cl\_base\* obj = this;  // Доходим до корневого объекта  while(obj->GetParent())  {  obj = obj->GetParent();  }  return obj;  }  // Поиск объекта по пути  cl\_base\* cl\_base::GetObjectByPath(string path)  {  // Указатель на текущий  cl\_base\* currentObj = this;  string nextObjName;  if (path.substr(0,1) == "/")  {  if (path.substr(1,1) == "/")  {  return SearchObjOnTree(path.substr(2));  }  // Текущий равен корневому  currentObj = GetRoot();  path = path.substr(1);  }  else if (path.substr(0,1) == ".")  {  return path == "." ? currentObj : CheckingObjUniq(path.substr(1));  }  stringstream streamPath(path);  // Пока можно разбивать строку на подстроки по '/'  while(getline(streamPath, nextObjName, '/'))  {  currentObj = currentObj->GetChild(nextObjName);  // Если такой объект не найден  if (!currentObj)  {  return nullptr;  }  }  return currentObj;  }  // Смена головного объекта  bool cl\_base::ChangeHeadObj(cl\_base\* newHead)  {  // Если головной объект найден  if (GetParent())  {  for (auto i = (GetParent()->children).begin(); i != (GetParent()->children).end(); i++)  {  if (\*i == this)  {  // Удаляем у головного объекта текущего сам текущий  (GetParent()->children).erase(i);  break;  }  }  // Переопределяем головной объект  this->parent = newHead;  // Добавление объекта к новому головному  (GetParent()->children).push\_back(this);  return true;  }  return false;  }  // Удаление объекта  bool cl\_base::DeleteSubObj(string objName)  {  // Находим в подчиненных по имени  cl\_base\* subObj = GetChild(objName);  for (auto i = children.begin(); i != children.end(); i++)  {  if (\*i == subObj)  {  // Удаление объекта по его итератору в списке  children.erase(i);  delete subObj;  return true;  }  }  return false;  }  //КВ4  // Метод смены состояния всех подчиненных объектов  void cl\_base::SetChildState(int state)  {  SetObjectState(state);  for (auto child : children)  {  child->SetChildState(state);  }  }  // Установка связи между сигналом текущего объекта и обработчиком целевого объекта  // signal - указатель на метод сигнала текущего объекта  // object - указатель целевой объект  // handler - указатель на метод обработчика целевого объекта  void cl\_base::SetConnect(TYPE\_SIGNAL signal, cl\_base\* object, TYPE\_HANDLER handler)  {  for(auto& pos : connects)  {  // Если соединение найдено, то нет смысла устанавливать соединение  if (pos.Signal==signal && pos.Handler==handler && pos.Object==object)  {  return;  }  }  // Создаем новое соединение  o\_sh obj;  obj.Signal = signal;  obj.Handler = handler;  obj.Object = object;  connects.push\_back(obj);  }  // Удаление связи между сигналом текущего объекта и обработчиком целевого объекта  void cl\_base::DeleteConnect(TYPE\_SIGNAL signal, cl\_base\* object, TYPE\_HANDLER handler)  {  for (auto i = connects.begin(); i != connects.end(); i++)  {  // Если соединение найдено  if ((\*i).Signal==signal  && (\*i).Handler==handler  && (\*i).Object==object)  {  // Удаляем  connects.erase(i);  return;  }  }  }  // Выдача сигнала от текущего объекта с передачей строковой переменной  void cl\_base::EmitSignal(TYPE\_SIGNAL signal, string& message)  {  // Вызов метода сигнала с передачей в него message  (this->\*signal)(message);  for(auto& pos : connects)  {  // Если сигналы сходятся  if (pos.Signal==signal)  {  TYPE\_HANDLER handler = pos.Handler;  cl\_base\* obj = pos.Object;  if (obj->GetObjectState())  {  // Вызов метода обработчика с передачей в него message  (obj->\*handler)(message);  }  }  }  }  string cl\_base::GetFullPath()  {  // Переменная для полного пути объекта  string path;  // Инициализируем временный указатель на текущий  cl\_base\* temp = this;  // Если род. нет, то возвращаем путь на корень  if(temp->GetParent() == nullptr)  {  return "/";  }  // Пока родитель есть  while(temp->GetParent() != nullptr)  {  // Строим полный путь к объекту  path = "/" + temp->GetName() + path;  temp = temp->GetParent();  }  return path;  }  // Установка номера класса  void cl\_base::SetClassNum(int num)  {  classNum = num;  }  // Возврат номера класса  int cl\_base::GetClassNum()  {  return classNum;  } |

## **Файл** **cl\_base.h**

Листинг 16 – cl\_base.h

|  |
| --- |
| #ifndef CL\_BASE\_H  #define CL\_BASE\_H  #include <iostream>  #include <string>  #include <sstream>  #include <vector>  using namespace std;  class cl\_base;  #define SIGNAL\_D(signal\_f)(TYPE\_SIGNAL)(&signal\_f)  #define HANDLER\_D(handler\_f)(TYPE\_HANDLER)(&handler\_f)  typedef void (cl\_base :: \*TYPE\_SIGNAL)(string&);  typedef void (cl\_base :: \*TYPE\_HANDLER)(string);  class cl\_base  {  string name;  cl\_base\* parent;  vector<cl\_base\*> children;  bool state = false;  int classNum;  // Структура соединения между объектами  struct o\_sh  {  TYPE\_SIGNAL Signal;  // Куда идет сигнал  cl\_base\* Object;  // Какой метод у объекта обработчик  TYPE\_HANDLER Handler;  };  // Список соединений объекта  vector<o\_sh> connects;  public:  //КВ1  cl\_base(cl\_base\* parent, string name = "Base\_object");  ~cl\_base();  // Установка наименования для текущего объекта  bool SetName(string newName);  // Возврат значения наименования объекта  string GetName() const;  // Возврат указателя на головной объект  cl\_base\* GetParent() const;  // Возврат указателя на подчиненный объект  cl\_base\* GetChild(string objName) const;  //КВ2  // Проверка на уникальность  int ObjNameCount(string objName);  // Проверка объекта на уникальность на ветке  cl\_base\* CheckingObjUniq(string objName);  // Поиск объекта на ветке по наименованию  cl\_base\* SearchObjOnBranch(string objName);  // Поиск объекта на дереве по наименованию  cl\_base\* SearchObjOnTree(string objName);  // Вывод дерева иерархии объектов  void PrintObjects(int spaces = 4) const;  // Методы для состояния объекта - КВ2  void PrintObjectsStates(int spaces = 4) const;  // Установка состояния объекта  void SetObjectState(bool state);  // Возврат состояния объекта  bool GetObjectState() const;  //КВ3  // Возврат указателя на корневой объект иерархии  cl\_base\* GetRoot();  // Возврат указателя на объект иерархии по передаваемому пути  cl\_base\* GetObjectByPath(string path);  // Переопределение головного объекта для текущего в дереве иерархии  bool ChangeHeadObj(cl\_base\* newHead);  // Удаление объекта у текущего в дереве иерархии  bool DeleteSubObj(string objName);  //КВ4  // Установка состояния всем объектам иерархии  void SetChildState(int);  // Установка соединения  void SetConnect(TYPE\_SIGNAL, cl\_base\*, TYPE\_HANDLER);  // Удаления соединения  void DeleteConnect(TYPE\_SIGNAL, cl\_base\*, TYPE\_HANDLER);  // Выдача сигнала  void EmitSignal(TYPE\_SIGNAL, string&);  // Получение полного пути  string GetFullPath();  // Установка номера класса  void SetClassNum(int);  // Получение номера класса  int GetClassNum();  };  #endif |

## **Файл** **main.cpp**

Листинг 17 – 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 |
| --- | --- | --- |
| 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 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  SET\_CONNECT /object\_s1 /object\_s2/object\_s6  SET\_CONNECT /object\_s1 /object\_s13  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  Signal to /object\_s2/object\_s6 Text: Send message 4 (class: 3)  Signal to /object\_s13 Text: Send message 4 (class: 3) | 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  Signal to /object\_s2/object\_s6 Text: Send message 4 (class: 3)  Signal to /object\_s13 Text: Send message 4 (class: 3) |
| 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  SET\_CONNECT /object\_s2/object\_s4 /object\_s13  EMIT /object\_s2/object\_s4 Send message 3  EMIT /object\_s1 Send message 4  SET\_CONDITION /object\_s2/object\_s4 0  EMIT /object\_s2/object\_s4 Send message 5  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 to /object\_s13 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 to /object\_s13 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 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  DELETE\_CONNECT  /object\_s2/object\_s4 /  EMIT /object\_s2/object\_s4 Send message 2  SET\_CONDITION /object\_s2/object\_s4 0  EMIT /object\_s2/object\_s4 Send message 3  SET\_CONNECT /object\_s1 /object\_s2/object\_s6  EMIT /object\_s1 Send message 4  SET\_CONDITION  /object\_s1/object\_s7 0  EMIT /object\_s2 Send message 48  END | 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  Signal from /object\_s2/object\_s4  Signal to /object\_s2/object\_s6 Text: Send message 2 (class: 4)  Signal from /object\_s1  Signal to /object\_s2/object\_s6 Text: Send message 4 (class: 3)  Signal from /object\_s2 | 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  Signal from /object\_s2/object\_s4  Signal to /object\_s2/object\_s6 Text: Send message 2 (class: 4)  Signal from /object\_s1  Signal to /object\_s2/object\_s6 Text: Send message 4 (class: 3)  Signal from /object\_s2 |

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были реализованы все необходимые классы и методы для формирования работы системы, а также методы сигналов и обработчиков для взаимодействия объектов системы между собой. При написании кода были соблюдены основные парадигмы объектно-ориентированного программирования.

В процессе написания программы были закреплены знания и практические навыки, полученные в течение обучения дисциплине "Объектно-ориентированное программирование". Также был получен опыт разработки системы, состоящей из нескольких объектов, выстроенных в дерево иерархии.

Работоспособность программы подтверждена тестированием на множестве тестов.

# 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. 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).