

Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ.....	6
1.1 Описание входных данных.....	8
1.2 Описание выходных данных.....	10
2 МЕТОД РЕШЕНИЯ.....	12
3 ОПИСАНИЕ АЛГОРИТМОВ.....	15
3.1 Алгоритм метода set_connection класса cl_base.....	15
3.2 Алгоритм метода delete_connection класса cl_base.....	16
3.3 Алгоритм метода emit_signal класса cl_base.....	16
3.4 Алгоритм метода setConnections класса cl_application.....	17
3.5 Алгоритм конструктора класса cl_1.....	19
3.6 Алгоритм конструктора класса cl_2.....	19
3.7 Алгоритм конструктора класса cl_3.....	19
3.8 Алгоритм конструктора класса cl_4.....	20
3.9 Алгоритм конструктора класса cl_6.....	20
3.10 Алгоритм конструктора класса cl_5.....	20
3.11 Алгоритм метода signal класса cl_1.....	21
3.12 Алгоритм метода signal класса cl_2.....	21
3.13 Алгоритм метода signal класса cl_3.....	21
3.14 Алгоритм метода signal класса cl_4.....	22
3.15 Алгоритм метода signal класса cl_5.....	22
3.16 Алгоритм метода signal класса cl_6.....	23
3.17 Алгоритм метода handler класса cl_1.....	23
3.18 Алгоритм метода handler класса cl_2.....	23
3.19 Алгоритм метода handler класса cl_3.....	24
3.20 Алгоритм метода handler класса cl_4.....	24
3.21 Алгоритм метода handler класса cl_5.....	25

3.22 Алгоритм метода handler класса cl_6.....	25
3.23 Алгоритм функции main.....	25
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....	27
5 КОД ПРОГРАММЫ.....	40
5.1 Файл cl_1.cpp.....	40
5.2 Файл cl_1.h.....	40
5.3 Файл cl_2.cpp.....	41
5.4 Файл cl_2.h.....	41
5.5 Файл cl_3.cpp.....	42
5.6 Файл cl_3.h.....	42
5.7 Файл cl_4.cpp.....	43
5.8 Файл cl_4.h.....	43
5.9 Файл cl_5.cpp.....	44
5.10 Файл cl_5.h.....	44
5.11 Файл cl_6.cpp.....	45
5.12 Файл cl_6.h.....	45
5.13 Файл cl_application.cpp.....	46
5.14 Файл cl_application.h.....	49
5.15 Файл cl_base.cpp.....	50
5.16 Файл cl_base.h.....	57
5.17 Файл main.cpp.....	58
6 ТЕСТИРОВАНИЕ.....	59
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	61

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 были изменены:

- Класс cl_1
- Структура oSh
- Пользовательские типы данных TYPE_SIGNAL, TYPE_HANDLER
- Параметризированное макроопределение препроцессора
- Класс cl_base
 - Поля
 - Целочисленное поле number
 - Тип данных - целый тип
 - Название - number
 - Модификатор доступа - public
 - Ветор connects структур oSh
 - Тип данных - вектор структур oSh
 - Название - connects
 - Модификатор доступа - public
 - Методы
 - Метод setConnect
 - Функционал - устанавливает связь между двумя объектами
 - Метод deleteConnect
 - Функционал - удаляет связь между двумя объектами
 - Метод emitSignal
 - Функционал-отправляетсообщениеотодногообъектак другому
 - Метод setFullReadiness

- Функционал - устанавливает готовность на все деревья в дереве
- Класс cl_application
 - Методы
 - Метод setConnections
 - Функционал - обработка установления связи между объектами
 - Метод handleCommands
 - Функционал - обработка ввода команд
- Классы cl_1 по класс cl_6:
 - Методы
 - Конструктор cl_1 по cl_6
 - Функционал - создание объекта и присвоение полю number номера текущего класса
 - Метод signal
 - Функционал - отправление сообщения
 - Метод handler
 - Функционал - принятие сообщения
- Структура oSh
 - Поля
 - Поле pSignal
 - Тип данных - пользовательский тип TYPE_SIGNAL
 - Модификатор - public
 - Название - pSignal
 - Поле pClObject
 - Тип данных - указатель на объект класса cl_base
 - Модификатор - public

- Название - pClObject
- Поле pHandler
 - Тип данных - пользовательский тип TYPE_HANDLER
 - Модификатор - public
 - Название - pHandler

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода `set_connection` класса `cl_base`

Функционал: Устанавливает связь между двумя объектами.

Параметры: пользовательский тип `TYPE_SIGNAL`, указатель на объект класса `cl_base`, пользовательский тип `TYPE_HANDLER`.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода `set_connection` класса `cl_base`

№	Предикат	Действия	№ перехода
1		Инициализация <code>pValue</code> указателя на структуру <code>oSh</code>	2
2	<code>i</code> меньше длины вектора <code>connects</code>		3
			4
3	Все поля текущей связи совпадают с переданными параметрами		∅
			2
4		Присвоение <code>pValue</code> значения указателя на структуру <code>oSh</code>	5
5		Присвоение полю <code>p_signal</code> значения параметра <code>p_signal</code>	6
6		Присвоение полю <code>p_handler</code> значения параметра	7

№	Предикат	Действия	№ перехода
		p_handler	
7		Присвоение полю p_target значения параметра p_target	8
8		Добавление в вектор connects переменной p_value	∅

3.2 Алгоритм метода delete_connection класса cl_base

Функционал: Удаляет связь между двумя объектами.

Параметры: пользовательский тип TYPE_SIGNAL, указатель на объект класса cl_base, пользовательский тип TYPE_HANDLER.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода delete_connection класса cl_base

№	Предикат	Действия	№ перехода
1		создание итератора p_it	2
2	i меньше длины вектора connects		3
			∅
3	Все поля текущей связи совпадают с переданными параметрами	Удаление из списка connects текущей связи	2
			2

3.3 Алгоритм метода emit_signal класса cl_base

Функционал: Отправка сообщения от одного объекта к другому.

Параметры: пользовательский тип TYPE_SIGNAL, адрес строкового типа.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 3.

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

№	Предикат	Действия	№ перехода
1		вызываем указанный сигнал, передавая ему сообщение s_masseye.	1
2	начинаем перебор всех соединений, которые установлены для данного объекта		2
			3
3	проверяем, соответствует ли текущее соединение вызванному сигналу.		4
			Ø
4		сохраняем указатель на объект-получатель, который был связан с данным соединением	5
5		сохраняем указатель на метод-обработчик, который был связан с данным соединением	6
6		вызываем метод-обработчик на объекте-получателе, передавая ему сообщение s_masseye	Ø

3.4 Алгоритм метода *setConnections* класса *cl_application*

Функционал: Управляет установкой связью между объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода *setConnections* класса *cl_application*

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной sender_coordinate	2
2		Инициализация строковой переменной reciever- coordinate	3
3		Инициализация указателя на объект класса Base p_sender	4
4		Инициализация указателя на объект класса Base p_reciever	5
5		Создание списка методов сигнала всех классов	6
6		Создание списка обработчиков сигнала всех классов	7
7		Ввод переменной sender_coordinate	8
8	sender-coordinate равно "end_of_connections"		∅
			9
9		Ввод переменной reciever_coordinate	10
10		Присвоение p_sender значения вызова метода find_obj_by_coord	11
11		Присвоение p_reciever значения вызова метода find_obj_by_coord	12
12		Присвоение переменной signal значения элемента списка SIGNALS_LIST под индексом текущего класса минус 1	13
13		Присвоение переменной handler значения элемента списка HANDLERS_LIST под индексом текущего класса минус 1	14
14		Вызов метода set_connection от p_sender	7

3.5 Алгоритм конструктора класса cl_1

Функционал: Параметризированный конструктор объекта.

Параметры: указатель на объект класса Base, строка.

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм конструктора класса cl_1

№	Предикат	Действия	№ перехода
1		Присвоение полю cl_number номера текущего класса	Ø

3.6 Алгоритм конструктора класса cl_2

Функционал: Параметризированный конструктор объекта.

Параметры: указатель на объект класса Base, строка.

Алгоритм конструктора представлен в таблице 6.

Таблица 6 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		Присвоение полю cl_number номера текущего класса	Ø

3.7 Алгоритм конструктора класса cl_3

Функционал: Параметризированный конструктор объекта.

Параметры: указатель на объект класса Base, строка.

Алгоритм конструктора представлен в таблице 7.

Таблица 7 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		Присвоение полю cl_number номера текущего класса	Ø

3.8 Алгоритм конструктора класса cl_4

Функционал: Параметризированный конструктор объекта.

Параметры: указатель на объект класса Base, строка.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		Присвоение полю cl_number номера текущего класса	Ø

3.9 Алгоритм конструктора класса cl_6

Функционал: Параметризированный конструктор объекта.

Параметры: указатель на объект класса Base, строка.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		Присвоение полю cl_number номера текущего класса	Ø

3.10 Алгоритм конструктора класса cl_5

Функционал: Параметризированный конструктор объекта.

Параметры: указатель на объект класса Base, строка.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		Присвоение полю cl_number номера текущего класса	Ø

3.11 Алгоритм метода **signal** класса **cl_1**

Функционал: Метод сигнала.

Параметры: Адрес строковой переменной.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода *signal* класса *cl_1*

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from" координата текущего объекта	2
2		Добавление к строке текста "class: " номер класса текущего объекта	Ø

3.12 Алгоритм метода **signal** класса **cl_2**

Функционал: Метод сигнала.

Параметры: Адрес строковой переменной.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода *signal* класса *cl_2*

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from" координата текущего объекта	2
2		Добавление к строке текста "class: " номер класса текущего объекта	Ø

3.13 Алгоритм метода **signal** класса **cl_3**

Функционал: Метод сигнала.

Параметры: Адрес строковой переменной.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода *signal* класса *cl_3*

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from" координата текущего объекта	2
2		Добавление к строке текста "class: " номер класса текущего объекта	∅

3.14 Алгоритм метода *signal* класса *cl_4*

Функционал: Метод сигнала.

Параметры: Адрес строковой переменной.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода *signal* класса *cl_4*

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from" координата текущего объекта	2
2		Добавление к строке текста "class: " номер класса текущего объекта	∅

3.15 Алгоритм метода *signal* класса *cl_5*

Функционал: Метод сигнала.

Параметры: Адрес строковой переменной.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода *signal* класса *cl_5*

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from" координата текущего объекта	2
2		Добавление к строке текста "class: " номер класса текущего объекта	∅

3.16 Алгоритм метода **signal** класса **cl_6**

Функционал: Метод сигнала.

Параметры: Адрес строковой переменной.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода *signal* класса *cl_6*

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal from" координата текущего объекта	2
2		Добавление к строке текста "class: " номер класса текущего объекта	Ø

3.17 Алгоритм метода **handler** класса **cl_1**

Функционал: Обработчик сигнала.

Параметры: строка.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода *handler* класса *cl_1*

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to " координата текущего объекта "Text: " значение параметра	Ø

3.18 Алгоритм метода **handler** класса **cl_2**

Функционал: Обработчик сигнала.

Параметры: строка.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода handler класса cl_2

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to " координата текущего объекта "Text: " значение параметра	Ø

3.19 Алгоритм метода handler класса cl_3

Функционал: Обработчик сигнала.

Параметры: строка.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода handler класса cl_3

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to " координата текущего объекта "Text: " значение параметра	Ø

3.20 Алгоритм метода handler класса cl_4

Функционал: Обработчик сигнала.

Параметры: строка.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода handler класса cl_4

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to " координата текущего объекта "Text: " значение параметра	Ø

3.21 Алгоритм метода handler класса cl_5

Функционал: Обработчик сигнала.

Параметры: строка.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода handler класса cl_5

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to " координата текущего объекта "Text: " значение параметра	Ø

3.22 Алгоритм метода handler класса cl_6

Функционал: Обработчик сигнала.

Параметры: строка.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода handler класса cl_6

№	Предикат	Действия	№ перехода
1		Вывод на экран "Signal to " координата текущего объекта "Text: " значение параметра	Ø

3.23 Алгоритм функции main

Функционал: Основная функция.

Параметры: отсутствует.

Возвращаемое значение: Целый тип - индикация успешной работы.

Алгоритм функции представлен в таблице 23.

Таблица 23 – Алгоритм функции *main*

№	Предикат	Действия	№ перехода
1		Инициализация application	2
2		Вызов метода buildTree()	3
3		Вызов метода exesApp()	Ø

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

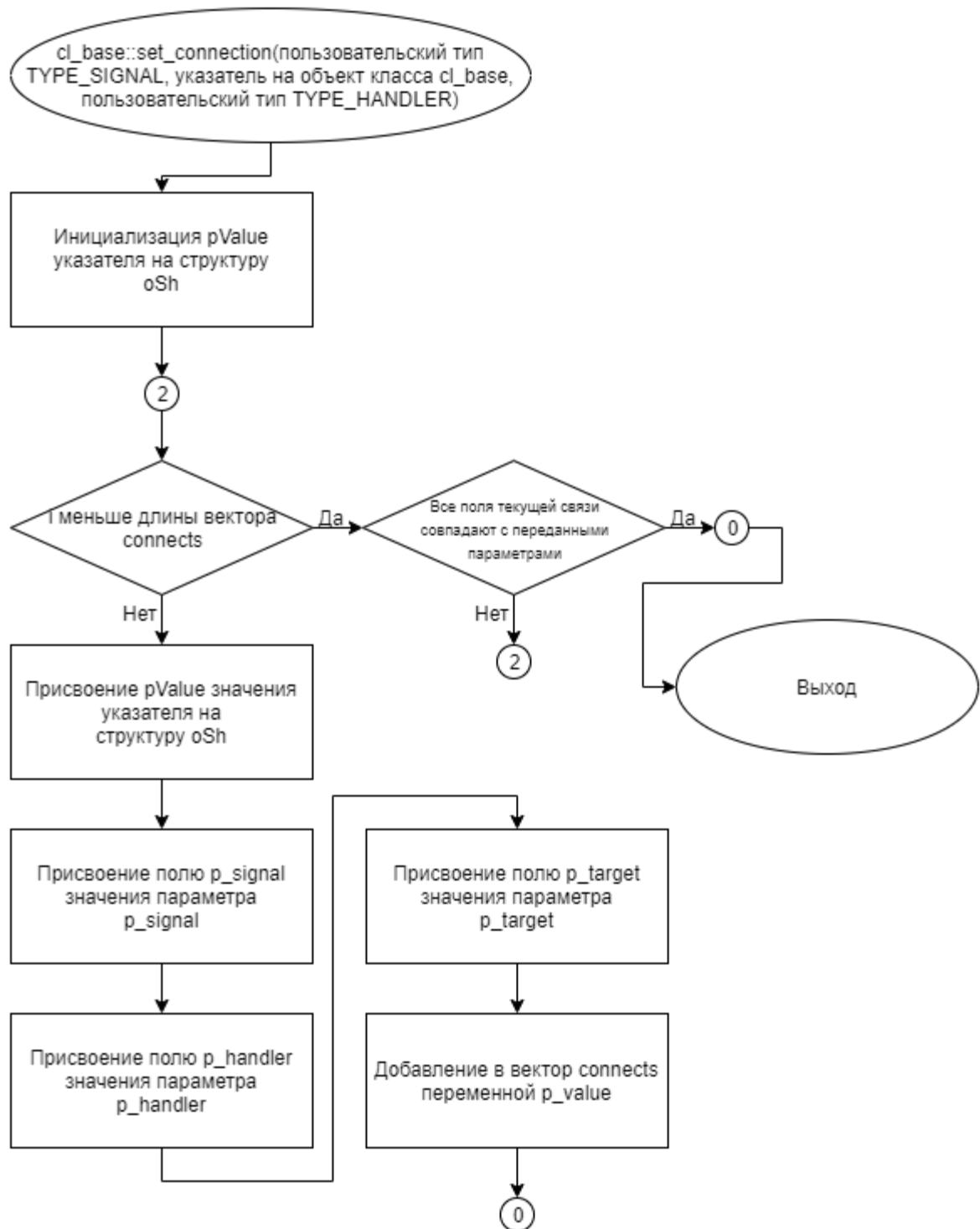


Рисунок 1 – Блок-схема алгоритма

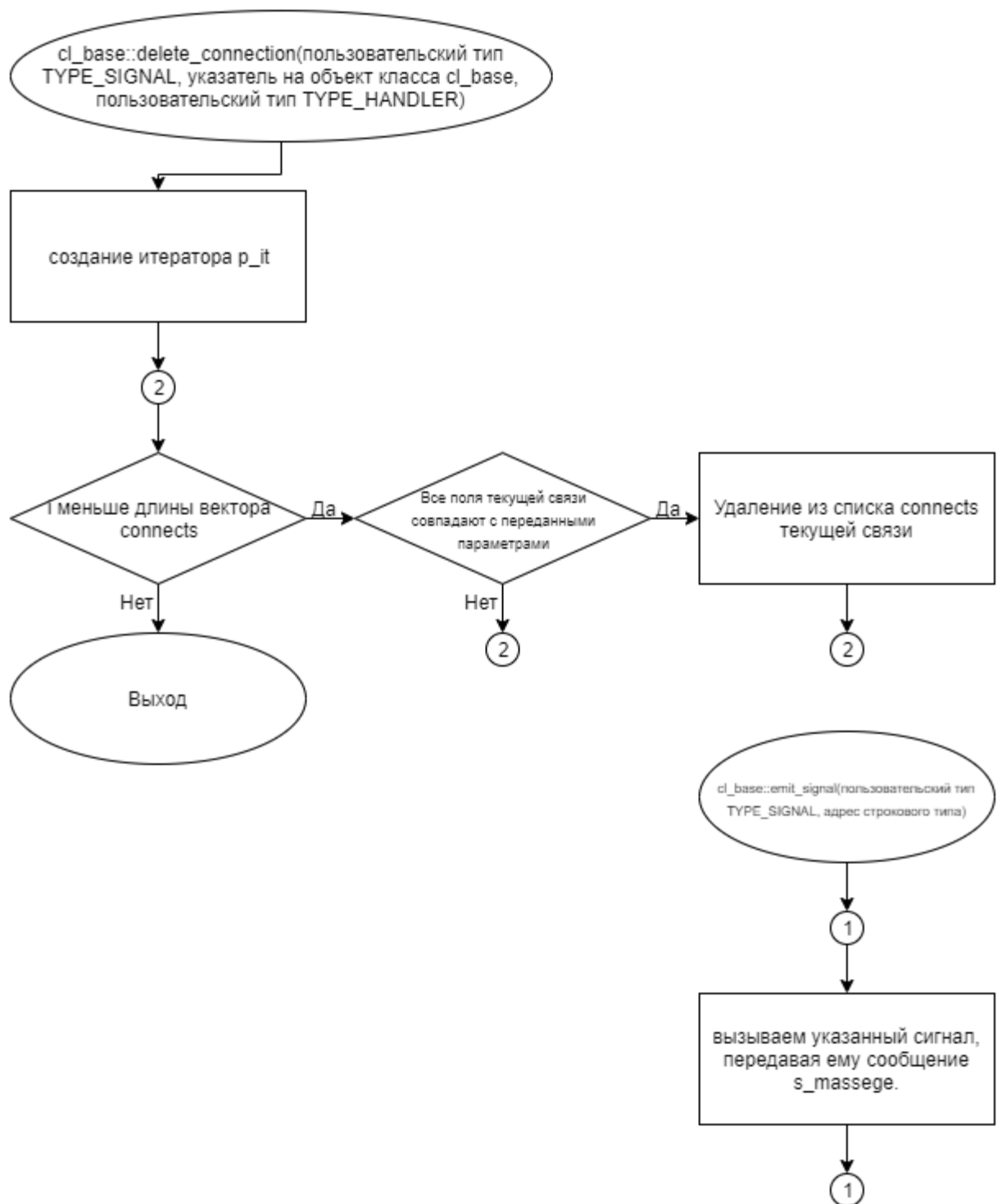


Рисунок 2 – Блок-схема алгоритма

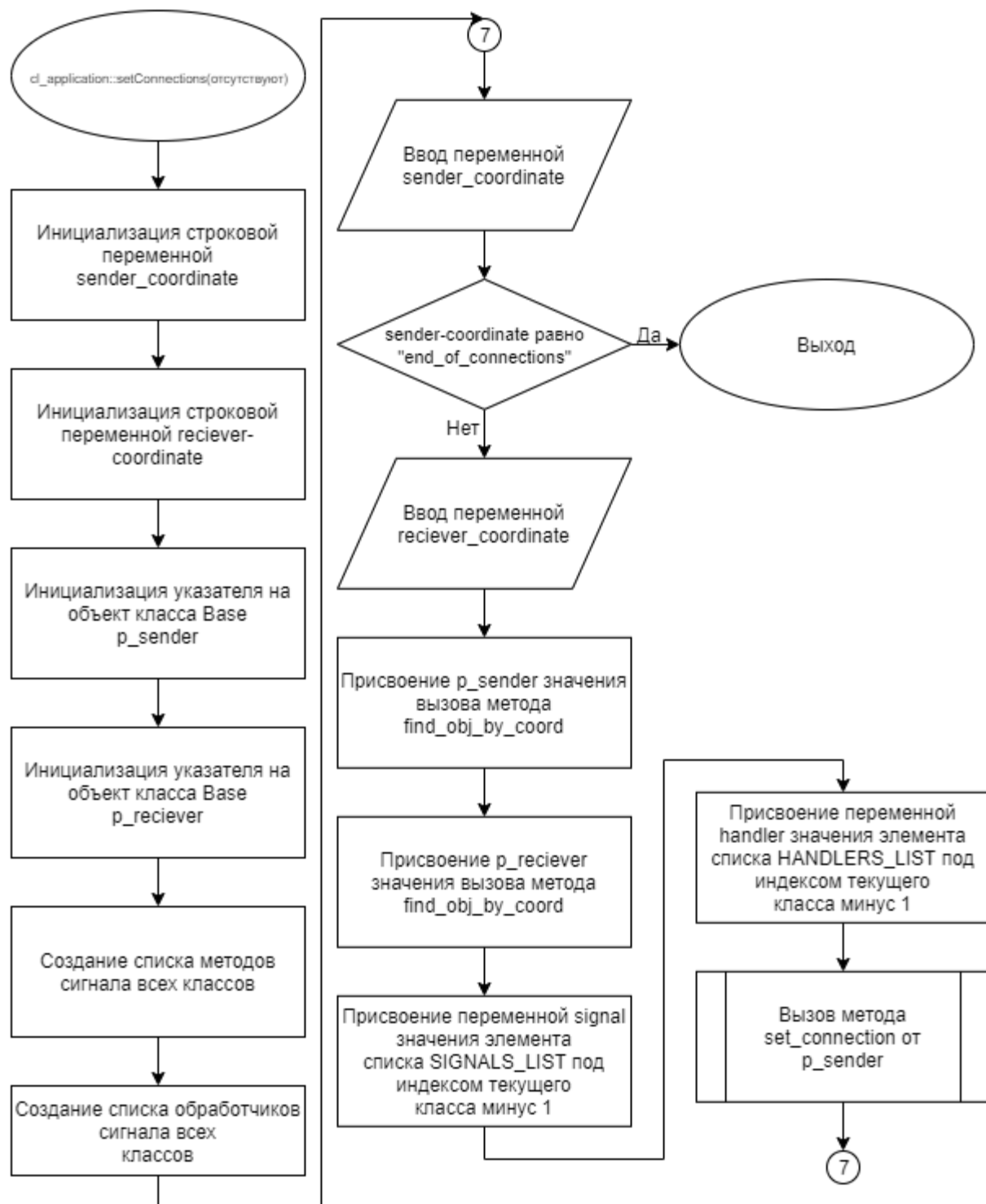


Рисунок 3 – Блок-схема алгоритма

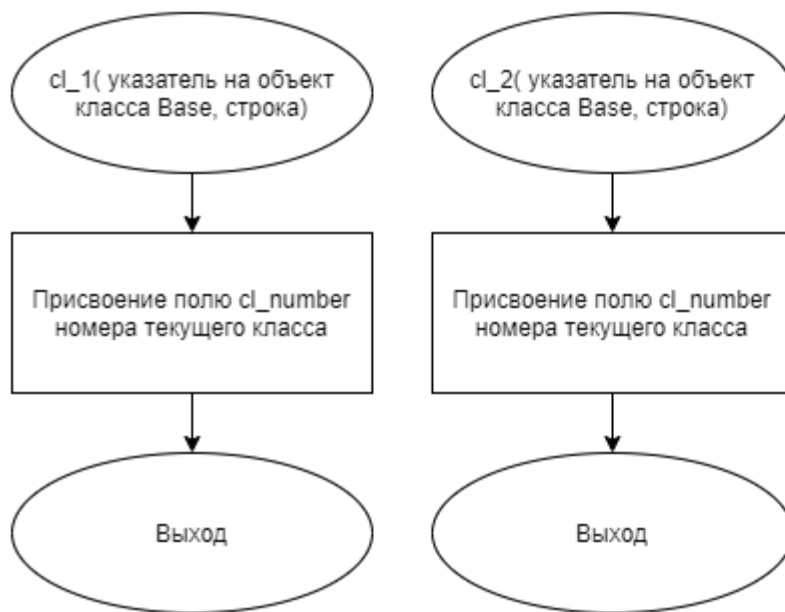


Рисунок 4 – Блок-схема алгоритма

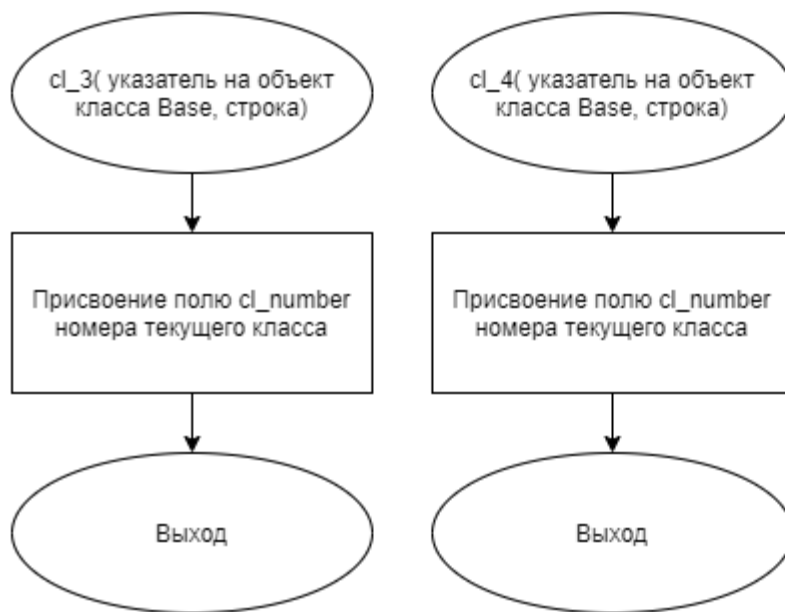


Рисунок 5 – Блок-схема алгоритма

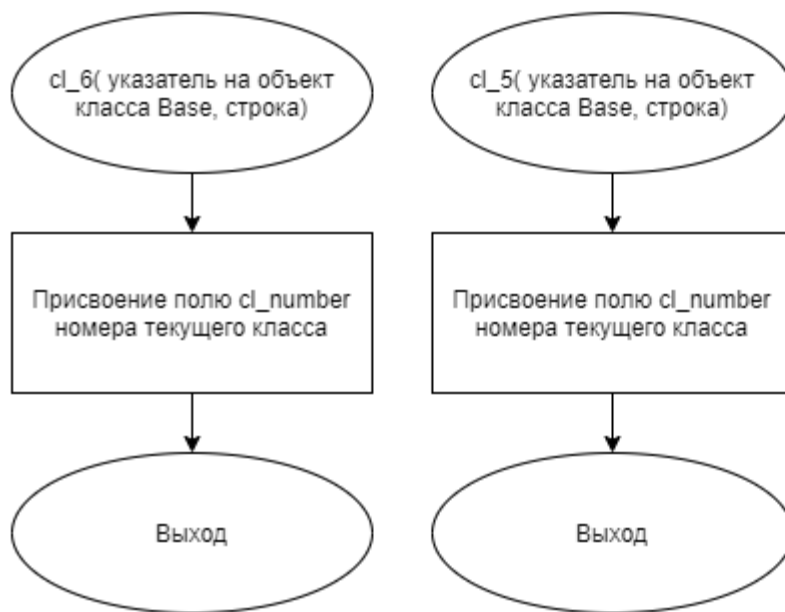


Рисунок 6 – Блок-схема алгоритма

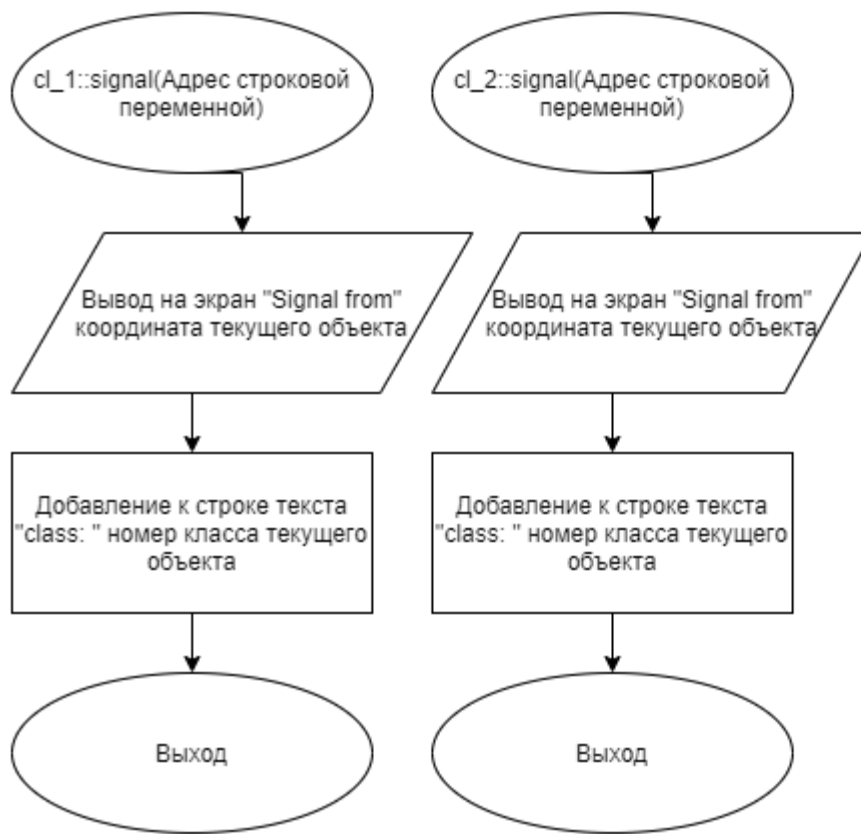


Рисунок 7 – Блок-схема алгоритма

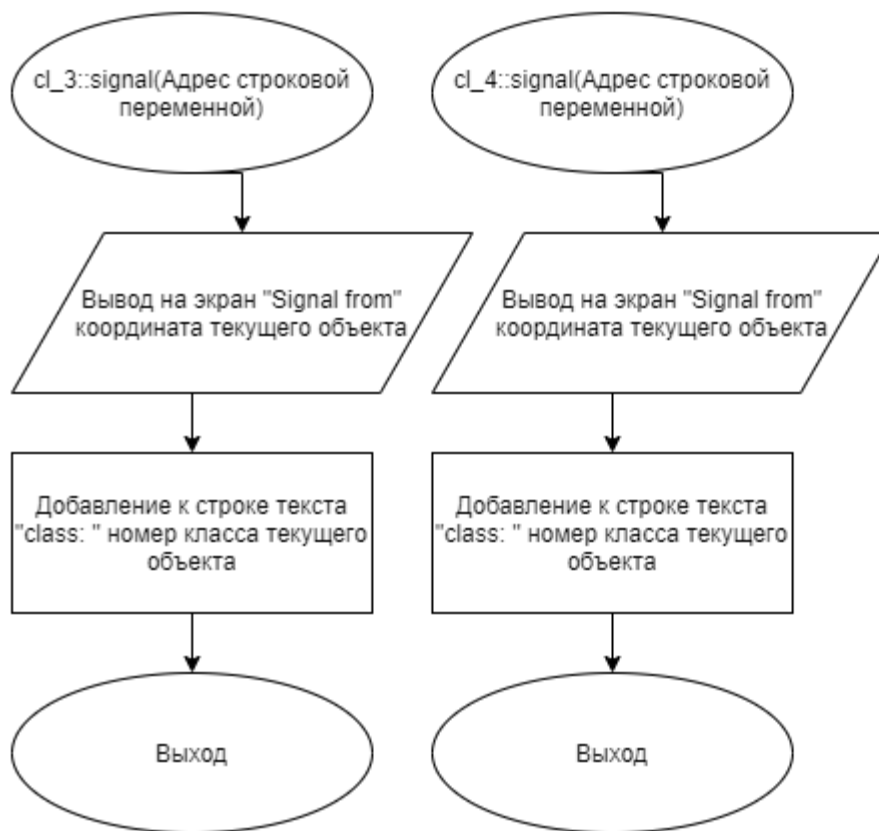


Рисунок 8 – Блок-схема алгоритма

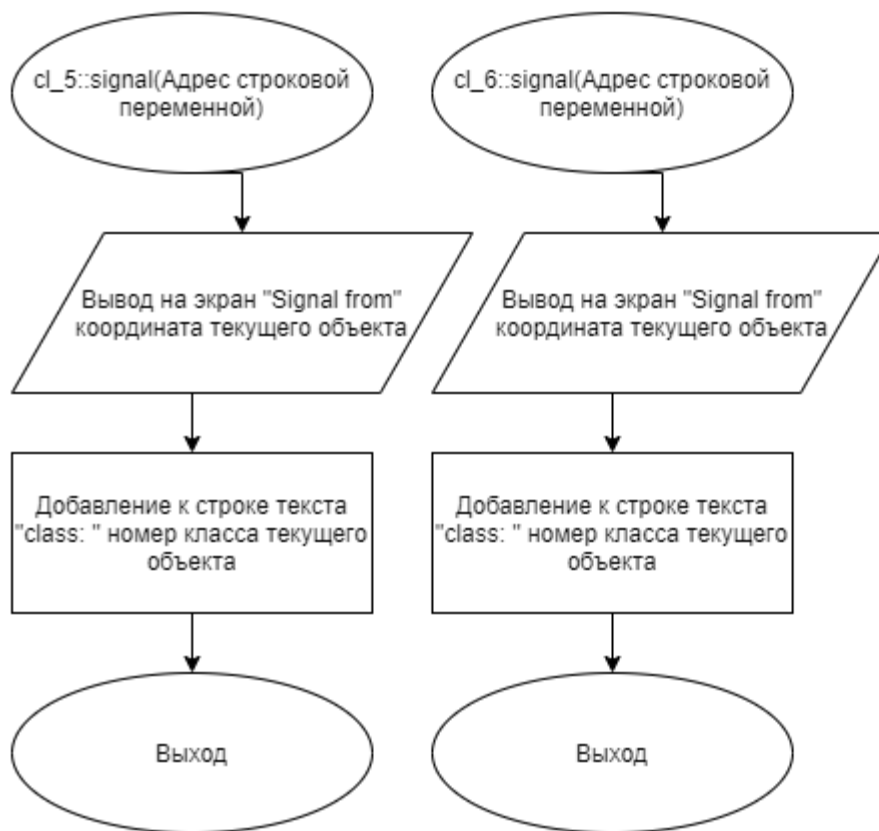


Рисунок 9 – Блок-схема алгоритма

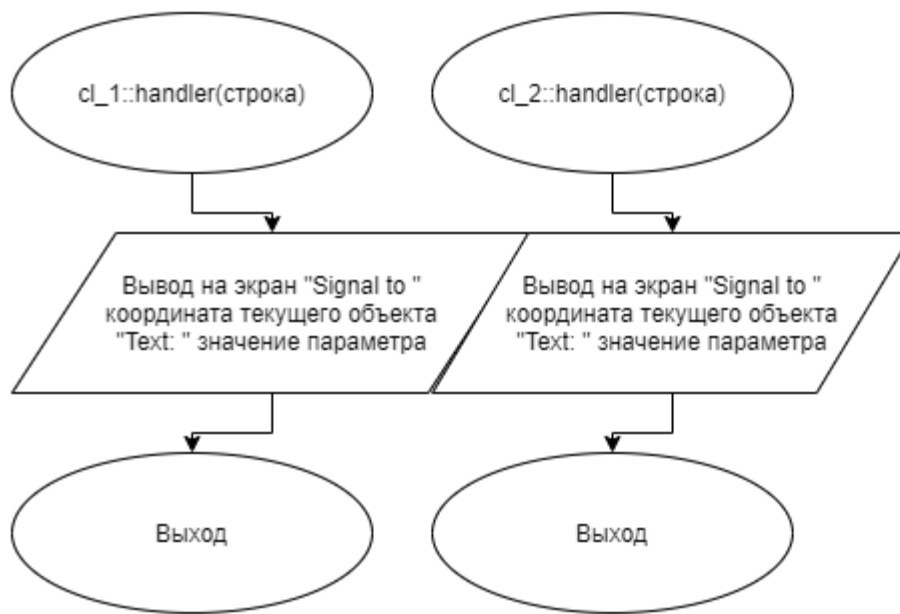


Рисунок 10 – Блок-схема алгоритма

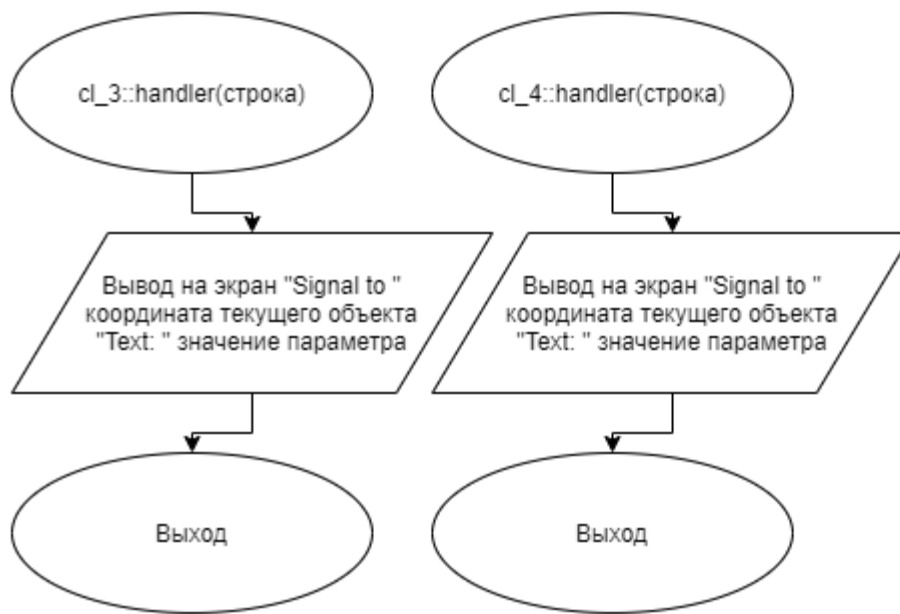


Рисунок 11 – Блок-схема алгоритма

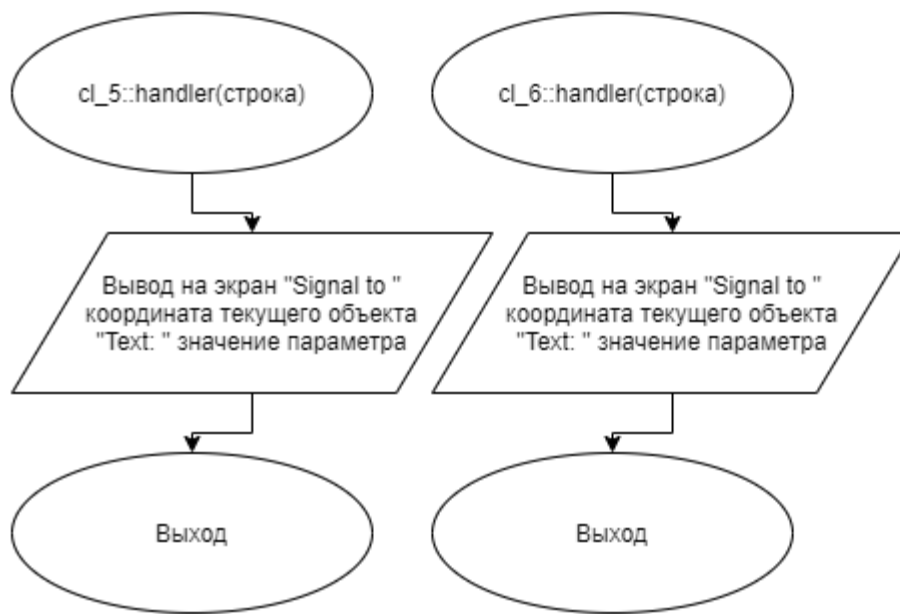


Рисунок 12 – Блок-схема алгоритма

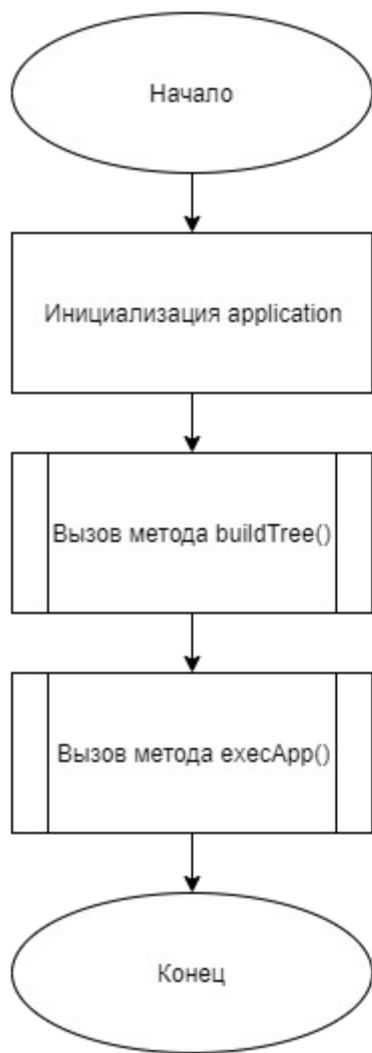


Рисунок 13 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"

cl_1::cl_1(cl_base* p_head_object, string s_name) :cl_base(p_head_object,
s_name)
{
    this->number = 1;
}

void cl_1::signal_f(string& msg)
{
    cout << endl << "Signal from " << this->get_path();
    msg += " (class: 1)";
}

void cl_1::handler_f(string msg)
{
    cout << endl << "Signal to " << get_path() << " Text:  " << msg;
}
```

5.2 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* p_head_object, string s_name);
    void signal_f(string& msg);
    void handler_f(string msg);
};
```

```
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(cl_base* p_head_object, string s_name) :cl_base(p_head_object,
s_name)
{
    this->number = 2;
}

void cl_2::signal_f(string& msg)
{
    cout << endl << "Signal from " << this->get_path();
    msg += " (class: 2)";
}

void cl_2::handler_f(string msg)
{
    cout << endl << "Signal to " << get_path() << " Text:  " << msg;
}
```

5.4 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* p_head_object, string s_name);
    void signal_f(string& msg);
    void handler_f(string msg);
};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* p_head_object, string s_name) :cl_base(p_head_object,
s_name)
{
    this->number = 3;
}

void cl_3::signal_f(string& msg)
{
    cout << endl << "Signal from " << this->get_path();
    msg += " (class: 3)";
}

void cl_3::handler_f(string msg)
{
    cout << endl << "Signal to " << get_path() << " Text:  " << msg;
}
```

5.6 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* p_head_object, string s_name);
    void signal_f(string& msg);
    void handler_f(string msg);
};
#endif
```


5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* p_head_object, string s_name) :cl_base(p_head_object,
s_name)
{
    this->number = 4;
}

void cl_4::signal_f(string& msg)
{
    cout << endl << "Signal from " << this->get_path();
    msg += " (class: 4)";
}

void cl_4::handler_f(string msg)
{
    cout << endl << "Signal to " << get_path() << " Text:  " << msg;
}
```

5.8 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* p_head_object, string s_name);
    void signal_f(string& msg);
    void handler_f(string msg);
};
#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* p_head_object, string s_name) :cl_base(p_head_object,
s_name)
{
    this->number = 5;
}

void cl_5::signal_f(string& msg)
{
    cout << endl << "Signal from " << this->get_path();
    msg += " (class: 5)";
}

void cl_5::handler_f(string msg)
{
    cout << endl << "Signal to " << get_path() << " Text:  " << msg;
}
```

5.10 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* p_head_object, string s_name);
    void signal_f(string& msg);
    void handler_f(string msg);
};
#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* p_head_object, string s_name) :cl_base(p_head_object,
s_name)
{
    this->number = 6;
}

void cl_6::signal_f(string& msg)
{
    cout << endl << "Signal from " << this->get_path();
    msg += " (class: 6)";
}

void cl_6::handler_f(string msg)
{
    cout << endl << "Signal to " << get_path() << " Text:  " << msg;
}
```

5.12 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* p_head_object, string s_name);
    void signal_f(string& msg);
    void handler_f(string msg);
};
#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.h"

void setConnections();

cl_application::cl_application(cl_base* p_head_object) :
cl_base(p_head_object) {}

void cl_application::build_tree_objects()
{
    string s_head, s_sub;
    int s_num_obj;
    cl_base* p_head = this, * p_sub = nullptr;
    cin >> s_head;
    set_name(s_head);
    while (true)
    {
        cin >> s_head;
        if (s_head == "endtree")
        {
            break;
        }
        cin >> s_sub >> s_num_obj;
        if (p_head != nullptr)
        {
            p_head = find_obj_by_coord(s_head);
            switch (s_num_obj)
            {
                case 1:
                    p_sub = new cl_1(p_head, s_sub);
                    break;
                case 2:
                    p_sub = new cl_2(p_head, s_sub);
                    break;
                case 3:
                    p_sub = new cl_3(p_head, s_sub);
                    break;
                case 4:
                    p_sub = new cl_4(p_head, s_sub);
                    break;
                case 5:
                    p_sub = new cl_5(p_head, s_sub);
                    break;
                case 6:
                    p_sub = new cl_6(p_head, s_sub);
                    break;
            }
        }
        else
        {
            cout << "Object tree";
        }
    }
}
```

```

        print_from_current();
        cout << endl << "The head object " << s_head << " is not found";
        exit(1);
    }
}

void cl_application::build_commands()
{
    string line, command, coord, text;
    vector <TYPE_SIGNAL> SIGNALS_LIST =
    {
        SIGNAL_D(cl_1::signal_f),
        SIGNAL_D(cl_2::signal_f),
        SIGNAL_D(cl_3::signal_f),
        SIGNAL_D(cl_4::signal_f),
        SIGNAL_D(cl_5::signal_f),
        SIGNAL_D(cl_6::signal_f)
    };
    vector<TYPE_HANDLER> HANDLERS_LIST =
    {
        HANDLER_D(cl_1::handler_f),
        HANDLER_D(cl_2::handler_f),
        HANDLER_D(cl_3::handler_f),
        HANDLER_D(cl_4::handler_f),
        HANDLER_D(cl_5::handler_f),
        HANDLER_D(cl_6::handler_f)
    };
    while (true)
    {
        getline(cin, line);
        command = line.substr(0, line.find(' '));
        line = line.substr(line.find(' ') + 1, line.size() - 1);
        coord = line.substr(0, line.find(' '));
        text = line.substr(line.find(' ') + 1);
        if (command == "END")
        {
            break;
        }
        if (line == "")
        {
            continue;
        }
        cl_base* pSender = this->find_obj_by_coord(coord);
        if (pSender == nullptr)
        {
            cout << endl << "Object " << coord << " not found";
            continue;
        }
        if (command == "EMIT")
        {
            TYPE_SIGNAL signal = SIGNALS_LIST[pSender->number - 1];
            pSender->emit_signal(signal, text);
        }
        if (command == "SET_CONNECT")
    }
}

```

```

        {
            cl_base* pReceiver = this->find_obj_by_coord(text);
            if (pReceiver == nullptr)
            {
                cout << endl << "Handler object " << text << " not found";
            }
            TYPE_SIGNAL signal = SIGNALS_LIST[pSender->number - 1];
            TYPE_HANDLER handler = HANDLERS_LIST[pReceiver->number - 1];
            pSender->set_connection(signal, pReceiver, handler);
        }
        if (command == "DELETE_CONNECT")
        {
            cl_base* pReceiver = this->find_obj_by_coord(text);
            if (pReceiver == nullptr)
            {
                cout << endl << "Handler object " << text << " not found";
            }
            else
            {
                TYPE_SIGNAL signal = SIGNALS_LIST[pSender->number - 1];
                TYPE_HANDLER handler = HANDLERS_LIST[pReceiver->number - 1];
                pSender->delete_connection(signal, pReceiver, handler);
            }
        }
        if (command == "SET_CONDITION") {
            int state = stoi(text);
            pSender->setState(state);
        }
    }
}

int cl_application::exec_app()
{
    cout << "Object tree";
    print_from_current();
    this->setConnections();
    build_commands();
    return 0;
}

void cl_application::setConnections()
{
    string senderCoord;
    string receiverCoord;
    cl_base* pSender;
    cl_base* pReceiver;
    vector<TYPE_SIGNAL> SIGNALS_LIST =
    {
        SIGNAL_D(cl_1::signal_f),
        SIGNAL_D(cl_2::signal_f),
        SIGNAL_D(cl_3::signal_f),
        SIGNAL_D(cl_4::signal_f),
        SIGNAL_D(cl_5::signal_f),
        SIGNAL_D(cl_6::signal_f)
    };
};

```

```

vector<TYPE_HANDLER> HANDLERS_LIST =
{
    HANDLER_D(cl_1::handler_f),
    HANDLER_D(cl_2::handler_f),
    HANDLER_D(cl_3::handler_f),
    HANDLER_D(cl_4::handler_f),
    HANDLER_D(cl_5::handler_f),
    HANDLER_D(cl_6::handler_f)
};
while (true)
{
    cin >> senderCoord;
    if (senderCoord == "end_of_connections") break;
    cin >> receiverCoord;
    pSender = this->find_obj_by_coord(senderCoord);
    pReceiver = this->find_obj_by_coord(receiverCoord);
    TYPE_SIGNAL signal = SIGNALS_LIST[pSender->number - 1];
    TYPE_HANDLER handler = HANDLERS_LIST[pReceiver->number - 1];
    pSender->set_connection(signal, pReceiver, handler);
}
}

```

5.14 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* p_head_object);
    void build_tree_objects();
    int exec_app();
    void build_commands();
    void setConnections();
    void handleCommands();
};
#endif

```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base* p_head_object, string s_name)
{
    this->s_name = s_name;
    this->p_head_object = p_head_object;
    if (p_head_object != nullptr)
    {
        p_head_object->p_sub_objects.push_back(this);
    }
}

cl_base::~~cl_base()
{
    get_root()->delete_links(this);
    for (int i = 0; i < p_sub_objects.size(); i++)
    {
        delete p_sub_objects[i];
    }
}

bool cl_base::set_name(string s_new_name)
{
    if (get_head() != nullptr)
    {
        for (int i = 0; i < get_head()->p_sub_objects.size(); i++)
        {
            if (get_head()->p_sub_objects[i]->get_name() == s_new_name)
            {
                return false;
            }
        }
    }
    s_name = s_new_name;
    return true;
}

void cl_base::print_tree(string delay)
{
    cout << endl << delay << get_name();
    for (auto p_sub : p_sub_objects)
    {
        p_sub->print_tree(delay + "    ");
    }
}

void cl_base::print_ready(string delay)
{
    cout << endl << delay;
    get_ready(get_name());
}
```



```

        for (auto p_sub : p_sub_objects)
        {
            p_sub->print_ready(delay + "    ");
        }
    }

    string cl_base::get_name()
    {
        return s_name;
    }

    cl_base* cl_base::get_head()
    {
        return p_head_object;
    }

    cl_base* cl_base::get_sub_obj(string s_name)
    {
        for (int i = 0; i < p_sub_objects.size(); i++)
        {
            if (p_sub_objects[i]->s_name == s_name)
            {
                return p_sub_objects[i];
            }
        }
        return nullptr;
    }

    int cl_base::count(string name)
    {
        int count = 0;
        if (get_name() == name)
        {
            count++;
        }
        for (int i = 0; i < p_sub_objects.size(); i++)
        {
            count += p_sub_objects[i]->count(name);
        }
        return count;
    }

    cl_base* cl_base::search_by_name(string name)
    {
        if (s_name == name)
        {
            return this;
        }
        cl_base* p_result = nullptr;
        for (int i = 0; i < p_sub_objects.size(); i++)
        {
            p_result = p_sub_objects[i]->search_by_name(name);
            if (p_result != nullptr)
            {
                return p_result;
            }
        }
    }

```

```

    }
    }
    return nullptr;
}

cl_base* cl_base::search_cur(string name)
{
    if (count(name) != 1)
    {
        return nullptr;
    }
    return search_by_name(name);
}

cl_base* cl_base::search_from_root(string name)
{
    if (p_head_object != nullptr)
    {
        return p_head_object->search_from_root(name);
    }
    else
    {
        return search_cur(name);
    }
}

void cl_base::set_ready(int s_new_ready)
{
    if (s_new_ready != 0)
    {
        if (p_head_object == nullptr || p_head_object != nullptr &&
p_head_object->p_ready != 0)
        {
            p_ready = s_new_ready;
        }
    }
    else
    {
        p_ready = s_new_ready;
        for (int i = 0; i < p_sub_objects.size(); i++)
        {
            p_sub_objects[i]->set_ready(s_new_ready);
        }
    }
}

void cl_base::get_ready(string name)
{
    if (get_name() == name)
    {
        if (p_ready != 0)
        {
            cout << get_name() << " is ready";
        }
        else
    }
}

```

```

        {
            cout << get_name() << " is not ready";
        }
    }
    else
    {
        for (int i = 0; i < p_sub_objects.size(); i++)
        {
            return p_sub_objects[i]->get_ready(name);
        }
    }
}

bool cl_base::change_head_obj(cl_base* new_head_obj)
{
    if (new_head_obj != nullptr)
    {
        cl_base* temp = new_head_obj;
        while (temp != nullptr)
        {
            temp = temp->p_head_object;
            if (temp == this)
            {
                return false;
            }
        }
        if (new_head_obj->get_sub_obj(get_name()) == nullptr && p_head_object != nullptr)
        {
            p_head_object->p_sub_objects.erase(find(p_head_object->p_sub_objects.begin(), p_head_object->p_sub_objects.end(), this));
            new_head_obj->p_sub_objects.push_back(this);
            p_head_object = new_head_obj;
            return true;
        }
    }
    return false;
}

void cl_base::delete_subordinate_obj(string name)
{
    cl_base* subordinate_obj = get_sub_obj(name);
    if (subordinate_obj != nullptr)
    {
        p_sub_objects.erase(find(p_sub_objects.begin(), p_sub_objects.end(), subordinate_obj));
        delete subordinate_obj;
    }
}

cl_base* cl_base::find_obj_by_coord(string s_object_path)
{
    if (s_object_path == "")
    {
        return nullptr;
    }
}

```

```

    }
    cl_base* head_obj = this;
    string s_path_item;
    if (s_object_path == "." || s_object_path == "/")
    {
        return head_obj;
    }
    if (s_object_path[0] == '.')
    {
        s_object_path.erase(s_object_path.begin());
        return search_by_name(s_object_path);
    }
    if (s_object_path[1] == '/' && s_object_path[0] == '/')
    {
        s_object_path.erase(s_object_path.begin());
        s_object_path.erase(s_object_path.begin());
        return this->search_from_root(s_object_path);
    }
    if (s_object_path[0] == '/')
    {
        s_object_path.erase(s_object_path.begin());
        while (head_obj->p_head_object != nullptr)
        {
            head_obj = head_obj->p_head_object;
        }
    }
    stringstream ss_path(s_object_path);
    while (getline(ss_path, s_path_item, '/'))
    {
        head_obj = head_obj->get_sub_obj(s_path_item);
        if (head_obj == nullptr)
        {
            return nullptr;
        }
    }
    return head_obj;
}

void cl_base::print_from_current(int n)
{
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "    ";
    }
    cout << s_name;
    for (auto p_subordinate_object : p_sub_objects)
    {
        p_subordinate_object->print_from_current(n + 1);
    }
}

void cl_base::set_connection(TYPE_SIGNAL p_signal, cl_base* p_target,
TYPE_HANDLER p_handler)
{

```

```

    o_sh* p_value;
    for (int i = 0; i < connects.size(); i++)
    {
        if (connects[i]->p_signal == p_signal &&
            connects[i]->p_handler == p_handler &&
            connects[i]->p_target == p_target)
        {
            return;
        }
    }
    p_value = new o_sh();
    p_value->p_signal = p_signal;
    p_value->p_handler = p_handler;
    p_value->p_target = p_target;

    connects.push_back(p_value);
}

void cl_base::delete_connection(TYPE_SIGNAL p_signal, cl_base* p_target,
TYPE_HANDLER p_handler)
{
    vector<o_sh*>::iterator p_it;
    for (p_it = connects.begin(); p_it != connects.end(); p_it++)
    {
        if ((*p_it)->p_signal == p_signal &&
            (*p_it)->p_target == p_target &&
            (*p_it)->p_handler == p_handler)
        {
            delete* p_it;
            p_it = connects.erase(p_it);
            p_it--;
        }
    }
}

void cl_base::emit_signal(TYPE_SIGNAL p_signal, string s_masseege)
{
    if (p_ready != 0)
    {
        TYPE_HANDLER pHandler;
        cl_base* pObject;
        (this->*p_signal)(s_masseege);
        for (int i = 0; i < connects.size(); i++)
        {
            if (connects[i]->p_signal == p_signal)
            {
                pHandler = connects[i]->p_handler;
                pObject = connects[i]->p_target;
                if (pObject->p_ready != 0)
                {
                    (pObject->*pHandler)(s_masseege);
                }
            }
        }
    }
}

```

```

}
string cl_base::get_path()
{
    cl_base* p_head_object = this->get_head();
    if (p_head_object != nullptr)
    {
        if (p_head_object->get_head() == nullptr)
        {
            return p_head_object->get_path() + s_name;
        }
        else
        {
            return p_head_object->get_path() + "/" + s_name;
        }
    }
    return "/";
}

void cl_base::setState(int state)
{
    if (state == 0)
    {
        this->p_ready = 0;
        for (int i = 0; i < p_sub_objects.size(); i++) {
            p_sub_objects[i]->setState(0);
        }
        return;
    }
    if (this->p_head_object == nullptr || this->p_head_object->p_ready != 0) {
        this->p_ready = state;
    }
}

void cl_base::delete_links(cl_base* targ)
{
    for (auto p_it = connects.begin(); p_it != connects.end(); p_it++)
    {
        if ((*p_it)->p_target == targ)
        {
            delete (*p_it);
            connects.erase(p_it);
            p_it--;
        }
    }

    for (auto p_sub : p_sub_objects)
    {
        p_sub->delete_links(targ);
    }
}

cl_base* cl_base::get_root()
{

```

```

        if (p_head_object != nullptr)
        {
            p_head_object->get_root();
        }
        return this;
    }

```

5.16 Файл cl_base.h

Листинг 16 – cl_base.h

```

#ifndef __CL_BASE_H__
#define __CL_BASE_H__
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <algorithm>
#define SIGNAL_D(signal_f) (TYPE_SIGNAL)(&signal_f)
#define HANDLER_D(handler_f) (TYPE_HANDLER)(&handler_f)
using namespace std;
class cl_base;

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

struct o_sh
{
    TYPE_SIGNAL p_signal;
    TYPE_HANDLER p_handler;
    cl_base* p_target;
};

class cl_base
{
private:
    string s_name;
    cl_base* p_head_object;
    vector <cl_base*> p_sub_objects;
    int p_ready = 1;
    vector<o_sh*> connects;
public:
    cl_base(cl_base* p_head_object, string s_name = "Base Object");
    bool set_name(string s_new_name);
    string get_name();
    cl_base* get_head();
    void print_tree(string delay = "");
    cl_base* get_sub_obj(string s_name);
    ~cl_base();
    int count(string name);

```

```

    cl_base* search_by_name(string name);
    cl_base* search_cur(string name);
    cl_base* search_from_root(string name);
    void set_ready(int s_new_ready);
    void get_ready(string name);
    void print_ready(string delay = "");
    bool change_head_obj(cl_base* new_head_obj);
    void delete_subordinate_obj(string name);
    cl_base* find_obj_by_coord(string s_object_path);
    void print_from_current(int n = 0);
    void set_connection(TYPE_SIGNAL p_signal, cl_base* p_target, TYPE_HANDLER
p_handler);
    void delete_connection(TYPE_SIGNAL p_signal, cl_base* p_target,
TYPE_HANDLER p_handler);
    void emit_signal(TYPE_SIGNAL p_signal, string massege);
    string get_path();
    int number = 1;
    typedef void (cl_base::* TYPE_SIGNAL)(string&);
    typedef void (cl_base::* TYPE_HANDLER)(string);
    void setState(int state);
    void delete_links(cl_base* targ);
    cl_base* get_root();
};
#endif

```

5.17 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 24.

Таблица 24 – 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>
<pre> appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 </pre>	<pre> Object tree appls_root object_s1 object_s7 object_s2 object_s4 </pre>	<pre> Object tree appls_root object_s1 object_s7 object_s2 object_s4 </pre>

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 EMIT /object_s2/object_s4 Send message 2 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 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_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_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

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).