

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.....	14
3.1 Алгоритм функции main.....	14
3.2 Алгоритм метода build_tree_objects класса cl_application.....	14
3.3 Алгоритм метода exec_app класса cl_application.....	16
3.4 Алгоритм метода emit класса cl_base.....	17
3.5 Алгоритм метода connect класса cl_base.....	18
3.6 Алгоритм метода disconnect класса cl_base.....	18
3.7 Алгоритм метода signal_f класса cl_base.....	19
3.8 Алгоритм метода handler_f класса cl_base.....	19
3.9 Алгоритм метода signal_f класса cl_2.....	19
3.10 Алгоритм метода handler_f класса cl_2.....	20
3.11 Алгоритм метода signal_f класса cl_3.....	20
3.12 Алгоритм метода handler_f класса cl_3.....	21
3.13 Алгоритм метода signal_f класса cl_4.....	21
3.14 Алгоритм метода handler_f класса cl_4.....	21
3.15 Алгоритм метода signal_f класса cl_5.....	22
3.16 Алгоритм метода handler_f класса cl_5.....	22
3.17 Алгоритм метода signal_f класса cl_6.....	23
3.18 Алгоритм метода handler_f класса cl_6.....	23
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....	24
5 КОД ПРОГРАММЫ.....	37
5.1 Файл cl_2.cpp.....	37

5.2	Файл cl_2.h.....	37
5.3	Файл cl_3.cpp.....	38
5.4	Файл cl_3.h.....	38
5.5	Файл cl_4.cpp.....	39
5.6	Файл cl_4.h.....	39
5.7	Файл cl_5.cpp.....	40
5.8	Файл cl_5.h.....	40
5.9	Файл cl_6.cpp.....	41
5.10	Файл cl_6.h.....	42
5.11	Файл cl_application.cpp.....	42
5.12	Файл cl_application.h.....	45
5.13	Файл cl_base.cpp.....	46
5.14	Файл cl_base.h.....	52
5.15	Файл main.cpp.....	53
6	ТЕСТИРОВАНИЕ.....	55
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	56

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 необходимы те же объекты, циклы и операторы, которые были

использованы для решения работы KB2.

Класс cl_base (с изменениями)

Поля:

Добавлено:

Открытое поле:

connections с типом vector<Connection*> - хранит список соединений

Открытые методы: void emit(TYPE_SIGNAL, string&) - выполняет обработку сигнала от текущего объекта

void connect(TYPE_SIGNAL, cl_base*, TYPE_HANDLER) - устанавливает связь между сигналом текущего объекта и обработчиком целевого объекта

void disconnect(TYPE_SIGNAL, cl_base*, TYPE_HANDLER) - разрывает связь между сигналом текущего объекта и обработчиком целевого объекта

virtual void signal_f(string&) - метод сигнала

virtual void handler_f(string) - метод обработчика

Структура Connection

Поля:

TYPE_SIGNAL p_signal - указатель на метод сигнала

cl_base* ob_target - указатель на объект

TYPE_HANDLER p_handler - указатель на метод обработчика

Класс cl_application (с изменениями)

Методы с изменениями:

`void build_tree_objects()` - строит иерархию объектов и устанавливает связи

`int exes_app()` - обрабатывает команды системы

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм функции `main`

Функционал: главный метод программы.

Параметры: нет.

Возвращаемое значение: `int`.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции `main`

№	Предикат	Действия	№ перехода
1		создание объекта	2
2		вызов метода <code>build_tree_objects</code>	3
3		вызов метода <code>exes_app</code>	Ø

3.2 Алгоритм метода `build_tree_objects` класса `cl_application`

Функционал: строение иерархии и установление связи.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода `build_tree_objects` класса `cl_application`

№	Предикат	Действия	№ перехода
1		ввод имени корневого объекта	2

№	Предикат	Действия	№ перехода
2		установка имени корневого объекта	3
3		установка состояния корневого объекта в 1	4
4		ввод координаты головного объекта	5
5	root_name равно "endtree"		9
		ввод имени дочернего объекта и номера класса	6
6		поиск головного объекта и номера класса	7
7	class_num равно 2	создание нового объекта класса cl_2 с child_name и p	8
	class_num равно 3	создание нового объекта класса cl_3 с child_name и p	8
	class_num равно 4	создание нового объекта класса cl_4 с child_name и p	8
	class_num равно 5	создание нового объекта класса cl_5 с child_name и p	8
	class_num равно 6	создание нового объекта класса cl_6 с child_name и p	8
			4
8		установка состояния нового объекта в 1	4
9		вызов метода print_object_tree	10
10		ввод координаты объекта, выдающего сигнал	11
11	координат объекта равно "end_of_connections"		∅
		ввод координаты целевого объекта	12
12		поиск объекта, выдающего сигнал, по его координате	13
13		поиск целевого объекта по его координате	14
14	объект, выдающий сигнал, или целевой объект не найден		10

№	Предикат	Действия	№ перехода
		установление связи между сигналом объекта и обработчиком целевого объекта	10

3.3 Алгоритм метода `exes_app` класса `cl_application`

Функционал: обрабатывает команды системы.

Параметры: нет.

Возвращаемое значение: `int`.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода `exes_app` класса `cl_application`

№	Предикат	Действия	№ перехода
1		ввод команды	2
2	команда равно "END"		Ø
		ввод координаты	3
3		поиск объекта по координате	4
4	объект не найден	вывод координата	1
			5
5	команда равно "SET_CONNECT" или команда равно "DELETE_CONNECT"	ввод "Object координаты not found"	6
			8
6		поиск целевого объекта по координате	7
7	целевой объект не найден	ввод "Handler object координаты not found"	1
			8
8	команда равно "EMIT"	ввод текста	9
	команда равно "SET_CONDITION"	ввод состояния	10

№	Предикат	Действия	№ перехода
	команда равно "SET_CONNECT"	установить связь между объектом и целевым объектом	1
	команда равно "DELETE_CONNECT"	разорвать связь между объектом и целевым объектом	1
			1
9		вызов метода emit объекта с передачей в него сигнала и текста	1
10		установить состояние объекта	1

3.4 Алгоритм метода emit класса cl_base

Функционал: выполняет обработку сигнала от текущего объекта.

Параметры: TYPE_SIGNAL, string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода emit класса cl_base

№	Предикат	Действия	№ перехода
1	объект не готов		∅
		выполнить полученный метод signal, передав в него cmd	2
2	с принадлежит connections		3
			∅
3	signal с равно переданному signal	выполнить метод handler из структуры с, передав в него cmd	2
			2

3.5 Алгоритм метода connect класса cl_base

Функционал: устанавливает связь между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: TYPE_SIGNAL, cl_base*, TYPE_HANDLER.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода connect класса cl_base

№	Предикат	Действия	№ перехода
1	в connections уже есть структура со значениями, равными значениям из параметров		Ø
		добавить в connections новую структуру с переданными параметрами	Ø

3.6 Алгоритм метода disconnect класса cl_base

Функционал: разрывает связь между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: TYPE_SIGNAL, cl_base*, TYPE_HANDLER.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода disconnect класса cl_base

№	Предикат	Действия	№ перехода
1	it принадлежит connections		2
			Ø
2	значения, содержащиеся в	удалить it из connections	Ø

№	Предикат	Действия	№ перехода
	структуре *it, совпадают со значениями из параметров		
			1

3.7 Алгоритм метода `signal_f` класса `cl_base`

Функционал: метод сигнала.

Параметры: `string& cmd`.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода `signal_f` класса `cl_base`

№	Предикат	Действия	№ перехода
1		вывод <code>signal from</code> "абсолютного пути до текущего объекта"	Ø

3.8 Алгоритм метода `handler_f` класса `cl_base`

Функционал: метод обработчика.

Параметры: `string cmd`.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода `handler_f` класса `cl_base`

№	Предикат	Действия	№ перехода
1		вывод <code>signal to</code> "абсолютный путь до текущего объекта"	Ø

3.9 Алгоритм метода `signal_f` класса `cl_2`

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода *signal_f* класса *cl_2*

№	Предикат	Действия	№ перехода
1		добавить к cmd строку "(class: 2)"	2
2		вывод signal from "абсолютного пути до текущего объекта"	Ø

3.10 Алгоритм метода *handler_f* класса *cl_2*

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода *handler_f* класса *cl_2*

№	Предикат	Действия	№ перехода
1		вывод signal to "абсолютный путь до текущего объекта"	Ø

3.11 Алгоритм метода *signal_f* класса *cl_3*

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода *signal_f* класса *cl_3*

№	Предикат	Действия	№ перехода
1		добавить к cmd строку "(class: 3)"	2

№	Предикат	Действия	№ перехода
2		вывод signal from "абсолютного пути до текущего объекта"	Ø

3.12 Алгоритм метода handler_f класса cl_3

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода handler_f класса cl_3

№	Предикат	Действия	№ перехода
1		вывод signal to "абсолютный путь до текущего объекта"	Ø

3.13 Алгоритм метода signal_f класса cl_4

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода signal_f класса cl_4

№	Предикат	Действия	№ перехода
1		добавить к cmd строку "(class: 4)"	2
2		вывод signal from "абсолютного пути до текущего объекта"	Ø

3.14 Алгоритм метода handler_f класса cl_4

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 14.

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

№	Предикат	Действия	№ перехода
1		вывод signal to "абсолютный путь до текущего объекта"	Ø

3.15 Алгоритм метода *signal_f* класса *cl_5*

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 15.

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

№	Предикат	Действия	№ перехода
1		добавить к cmd строку "(class: 5)"	2
2		вывод signal from "абсолютного пути до текущего объекта"	Ø

3.16 Алгоритм метода *handler_f* класса *cl_5*

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода *handler_f* класса *cl_5*

№	Предикат	Действия	№ перехода
1		вывод signal to "абсолютный путь до текущего объекта"	Ø

3.17 Алгоритм метода `signal_f` класса `cl_6`

Функционал: метод сигнала.

Параметры: `string& cmd`.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода `signal_f` класса `cl_6`

№	Предикат	Действия	№ перехода
1		добавить к <code>cmd</code> строку "(class: 2)"	2
2		вывод <code>signal from</code> "абсолютного пути до текущего объекта"	Ø

3.18 Алгоритм метода `handler_f` класса `cl_6`

Функционал: метод обработчика.

Параметры: `string cmd`.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода `handler_f` класса `cl_6`

№	Предикат	Действия	№ перехода
1		вывод <code>signal to</code> "абсолютный путь до текущего объекта"	Ø

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

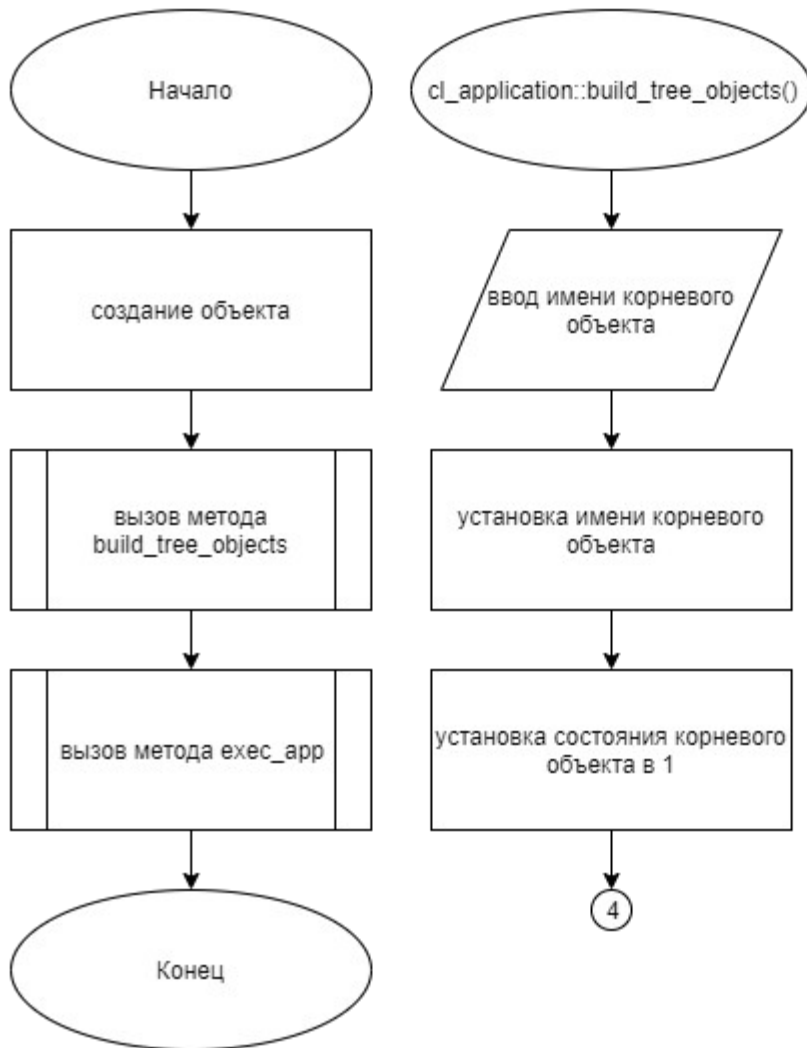


Рисунок 1 – Блок-схема алгоритма

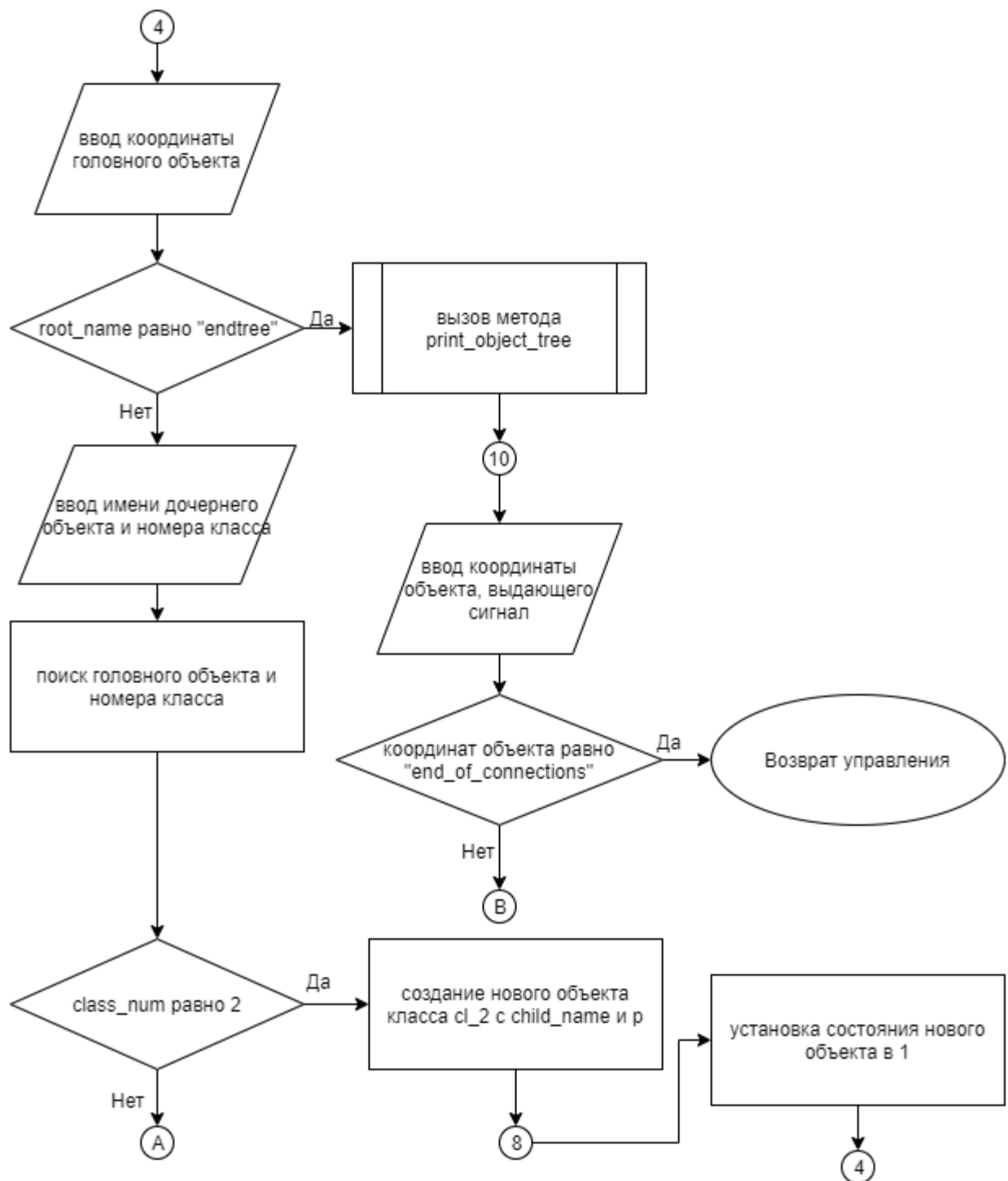


Рисунок 2 – Блок-схема алгоритма

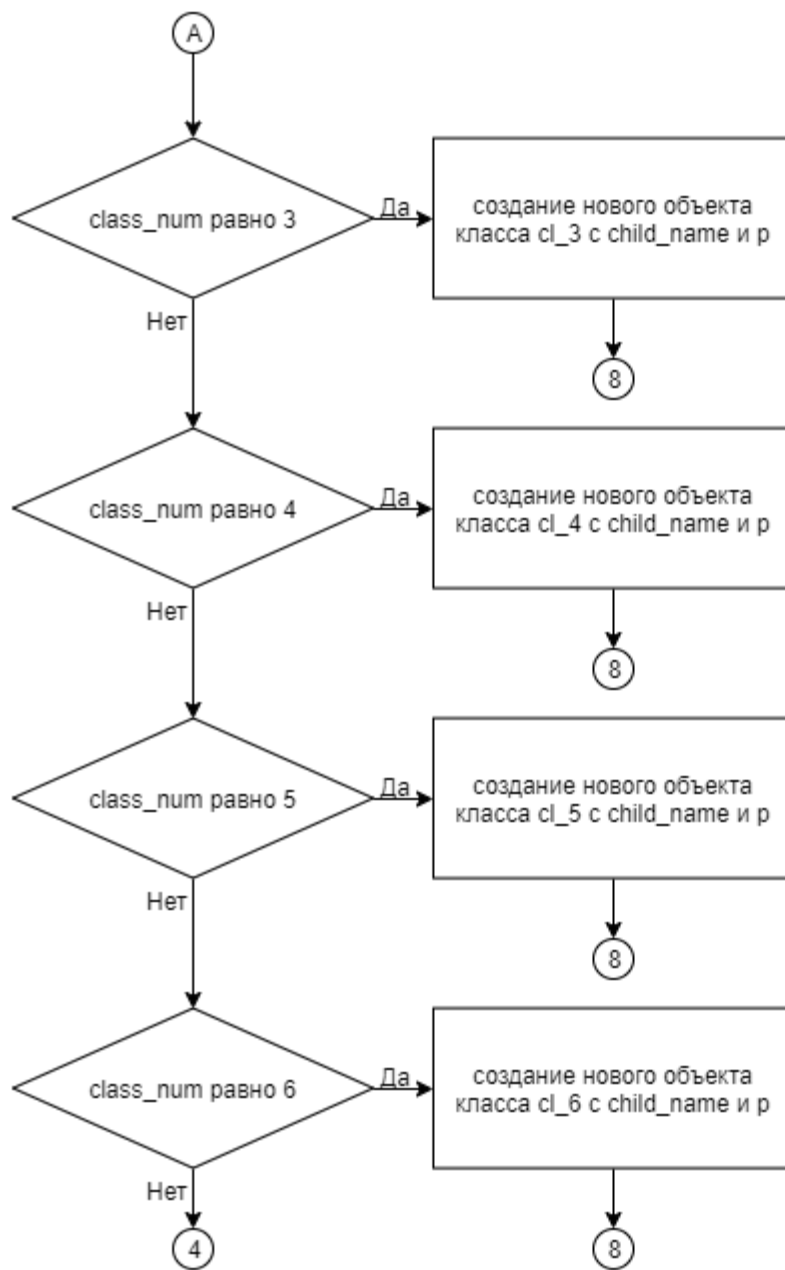


Рисунок 3 – Блок-схема алгоритма

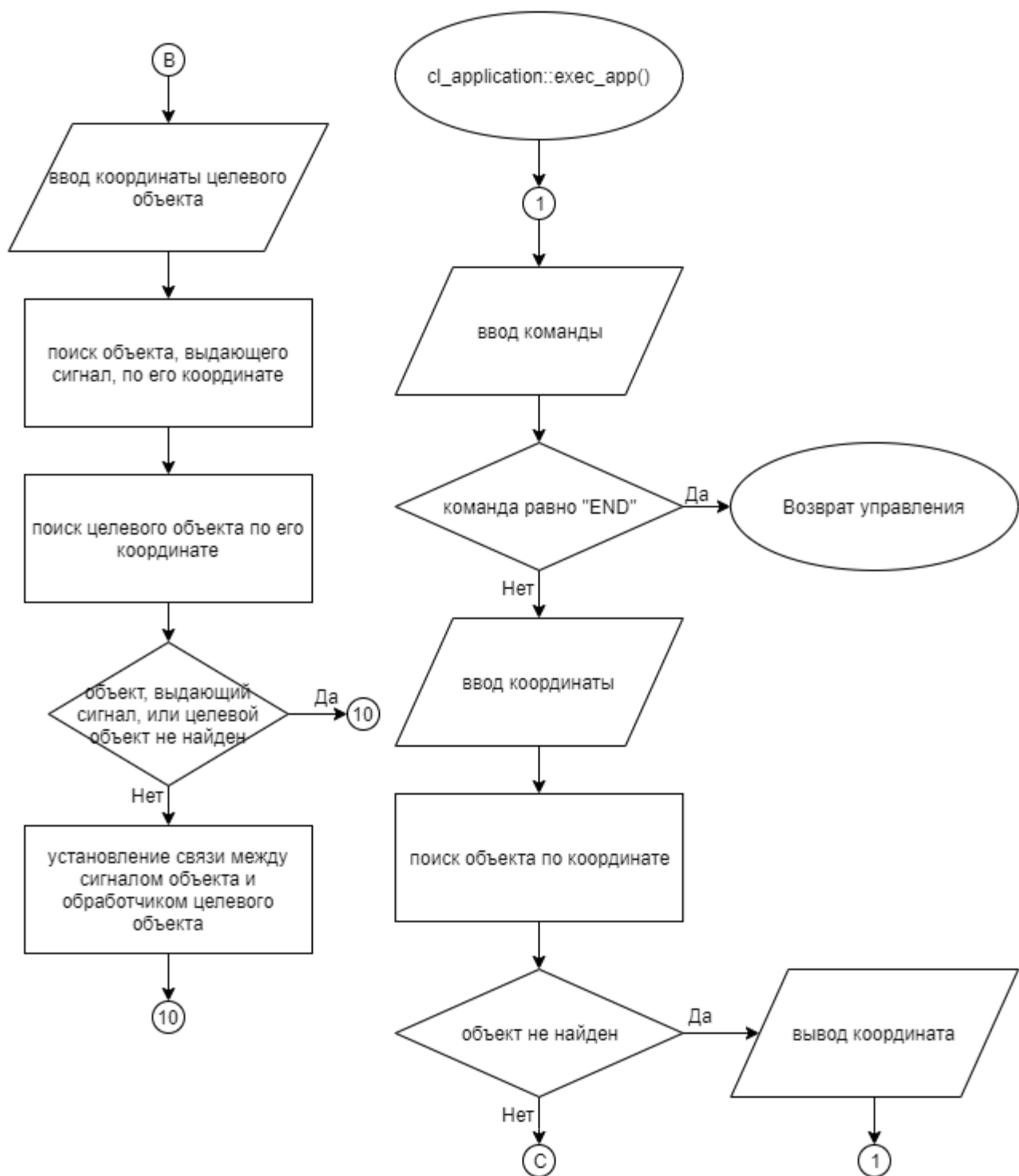


Рисунок 4 – Блок-схема алгоритма

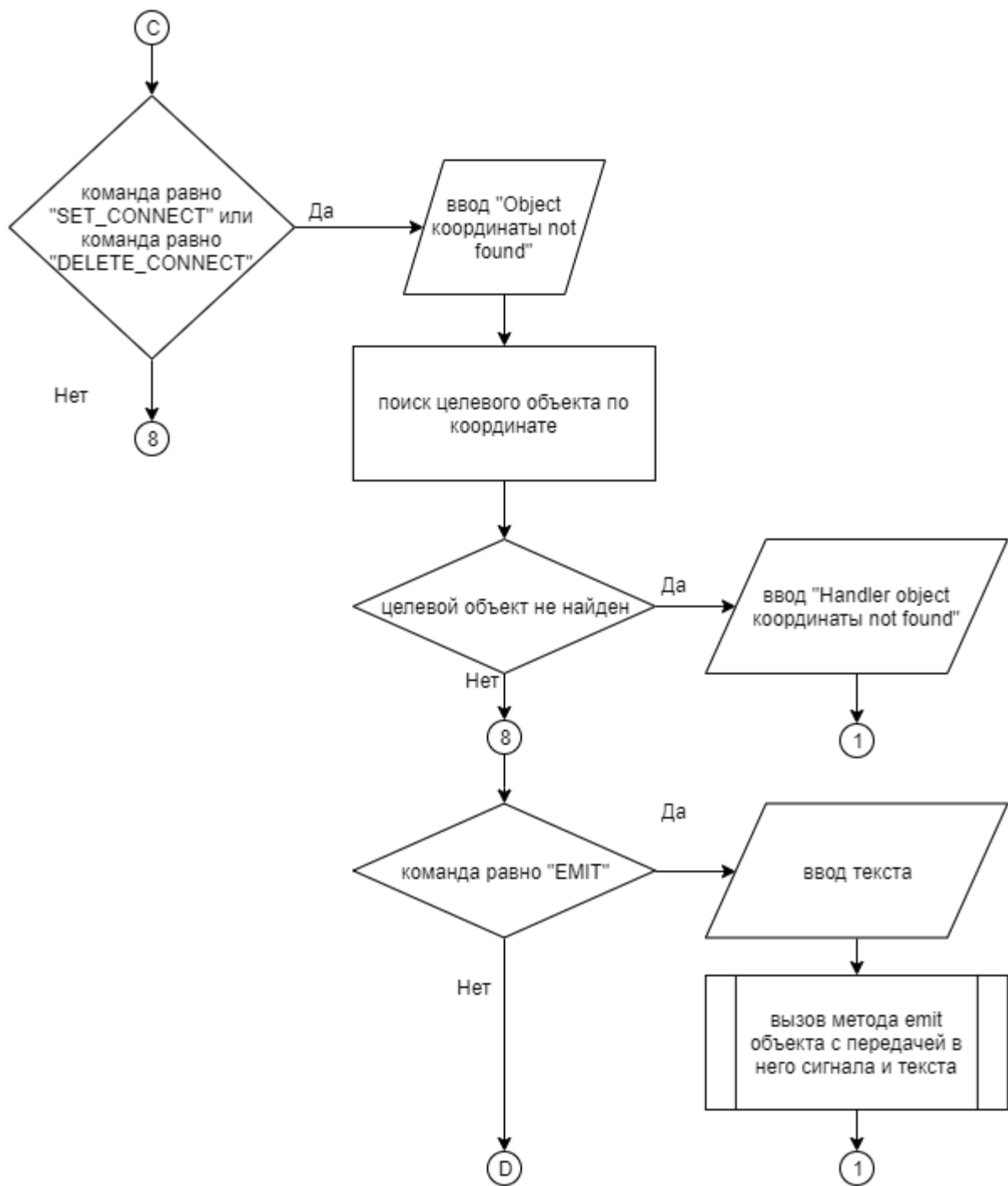


Рисунок 5 – Блок-схема алгоритма

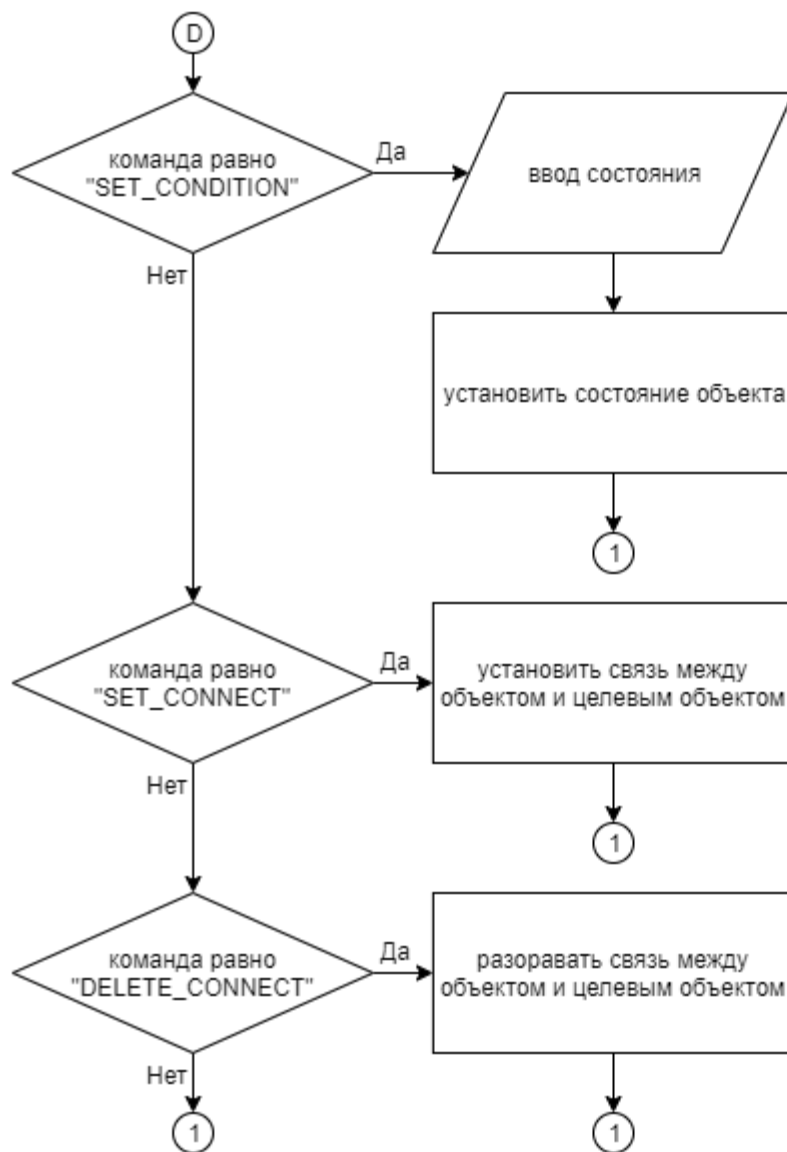


Рисунок 6 – Блок-схема алгоритма

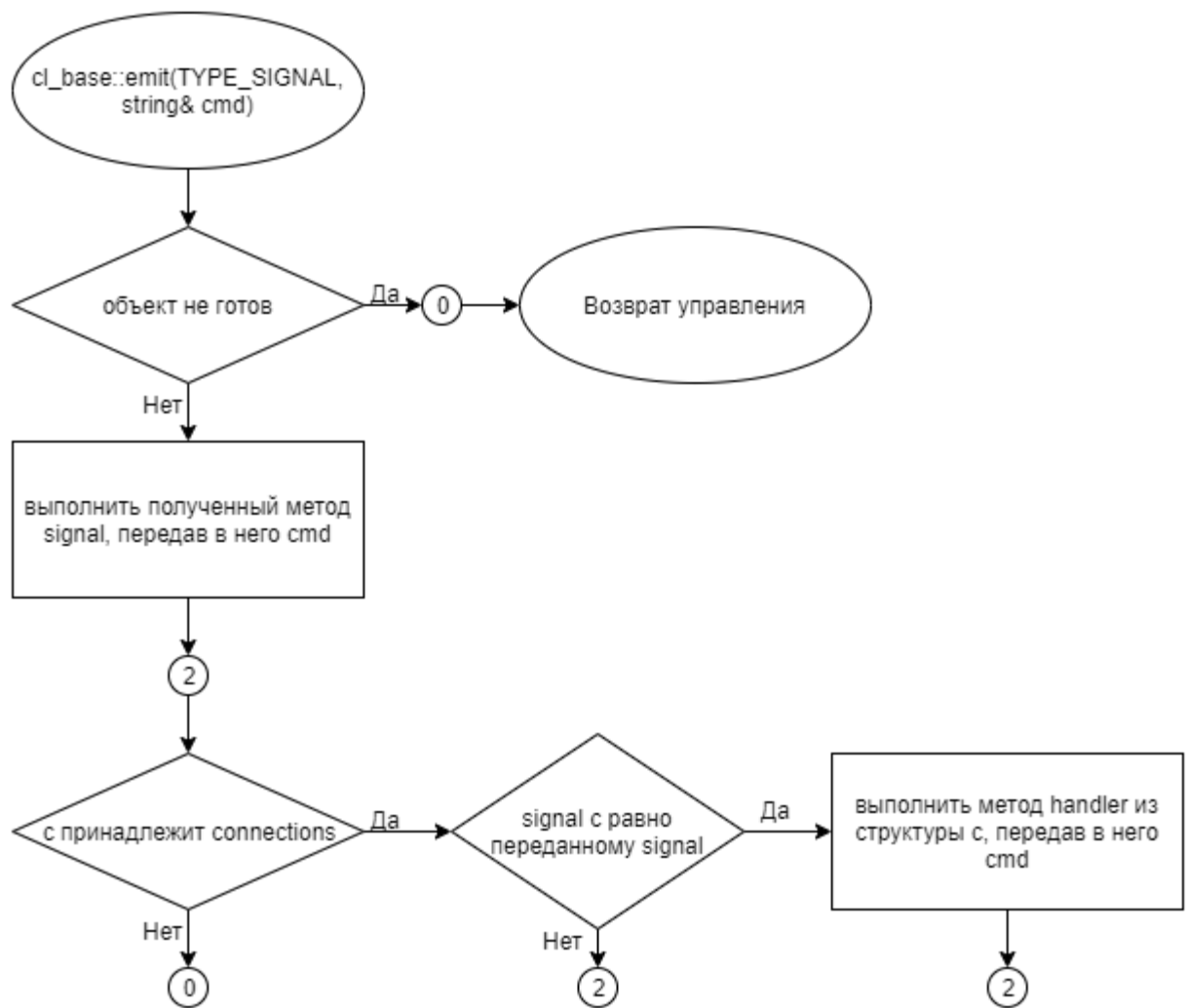


Рисунок 7 – Блок-схема алгоритма

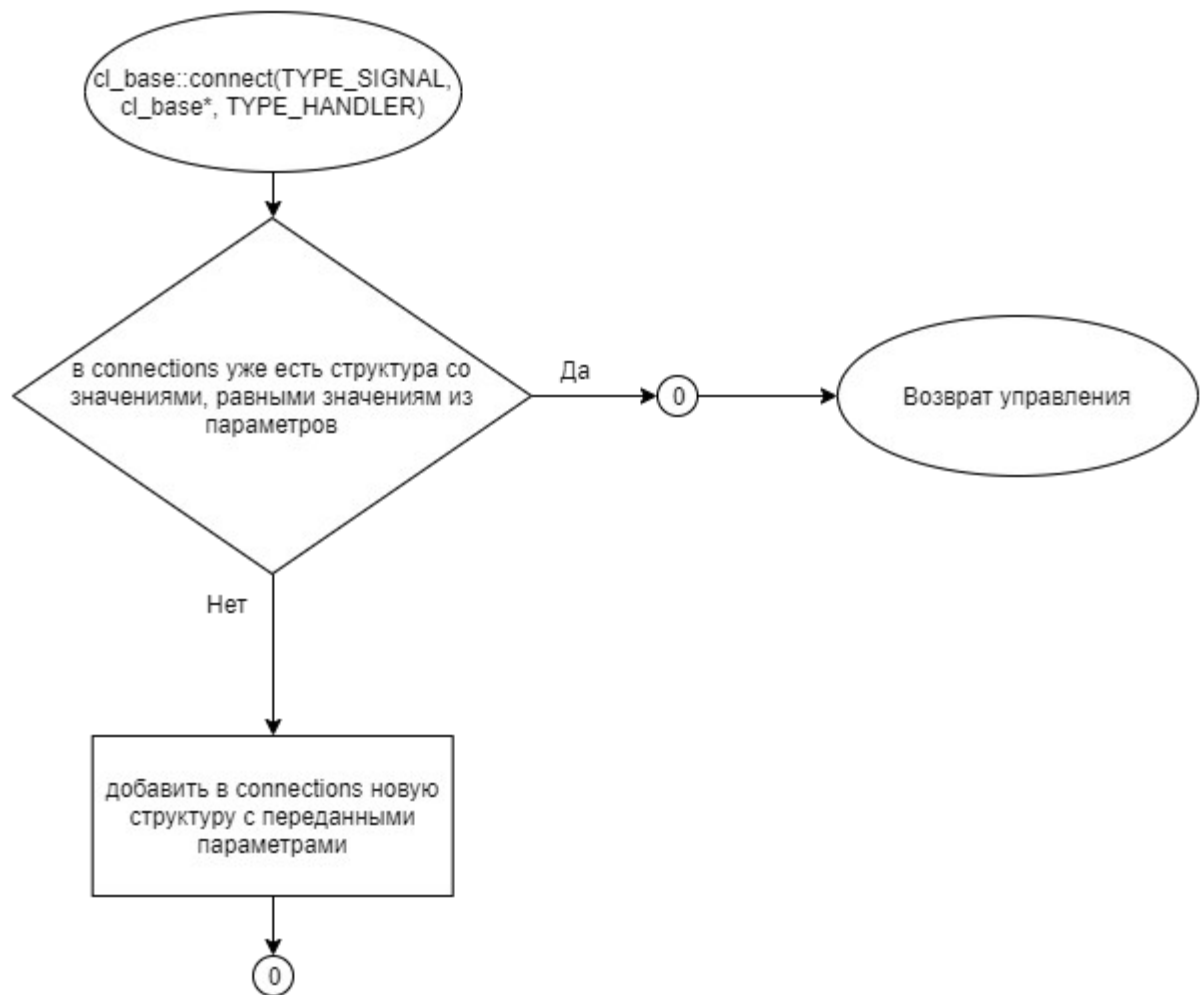


Рисунок 8 – Блок-схема алгоритма

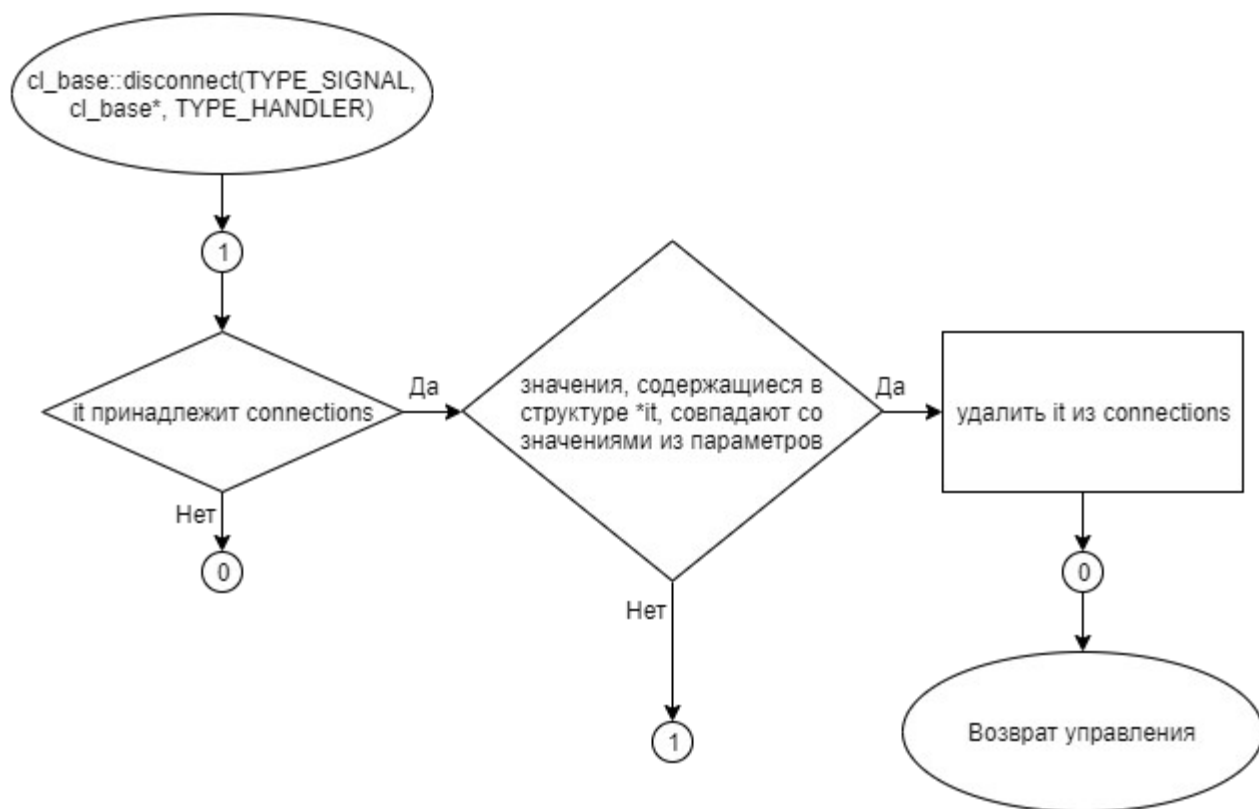


Рисунок 9 – Блок-схема алгоритма

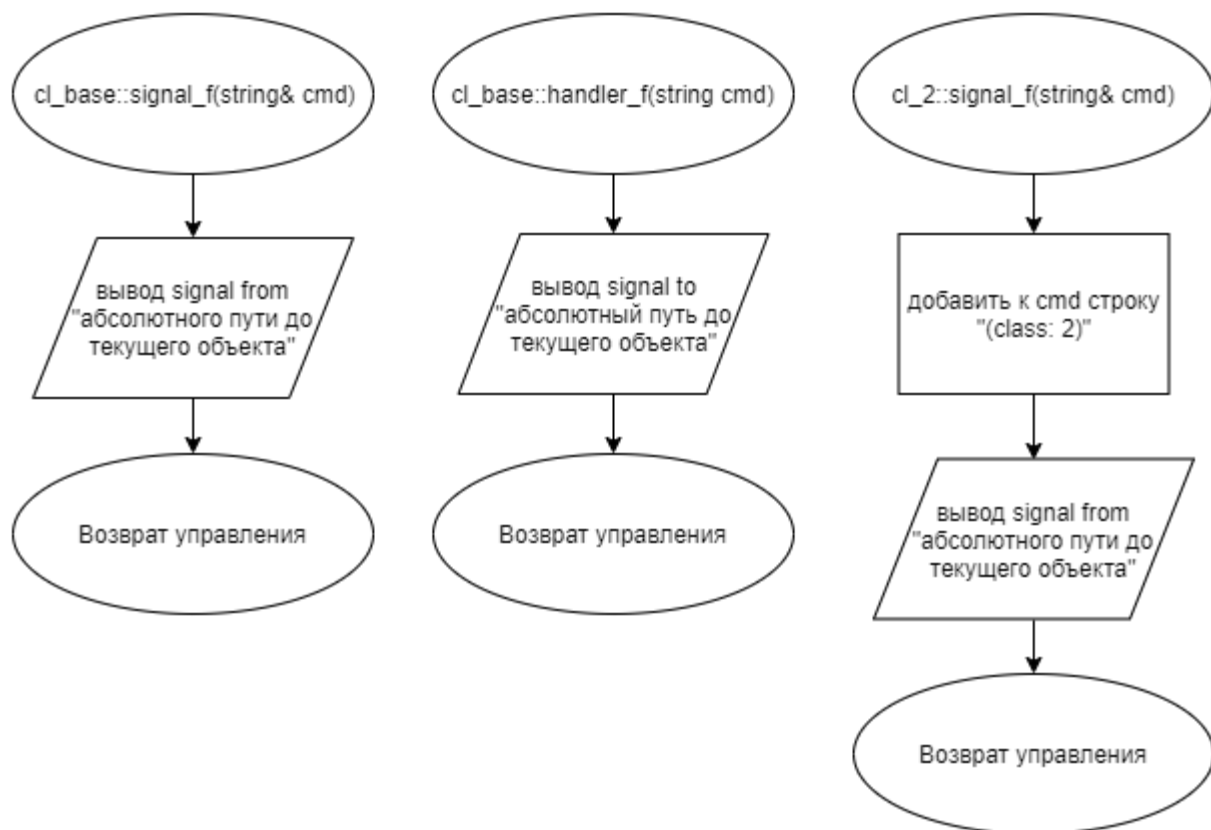


Рисунок 10 – Блок-схема алгоритма

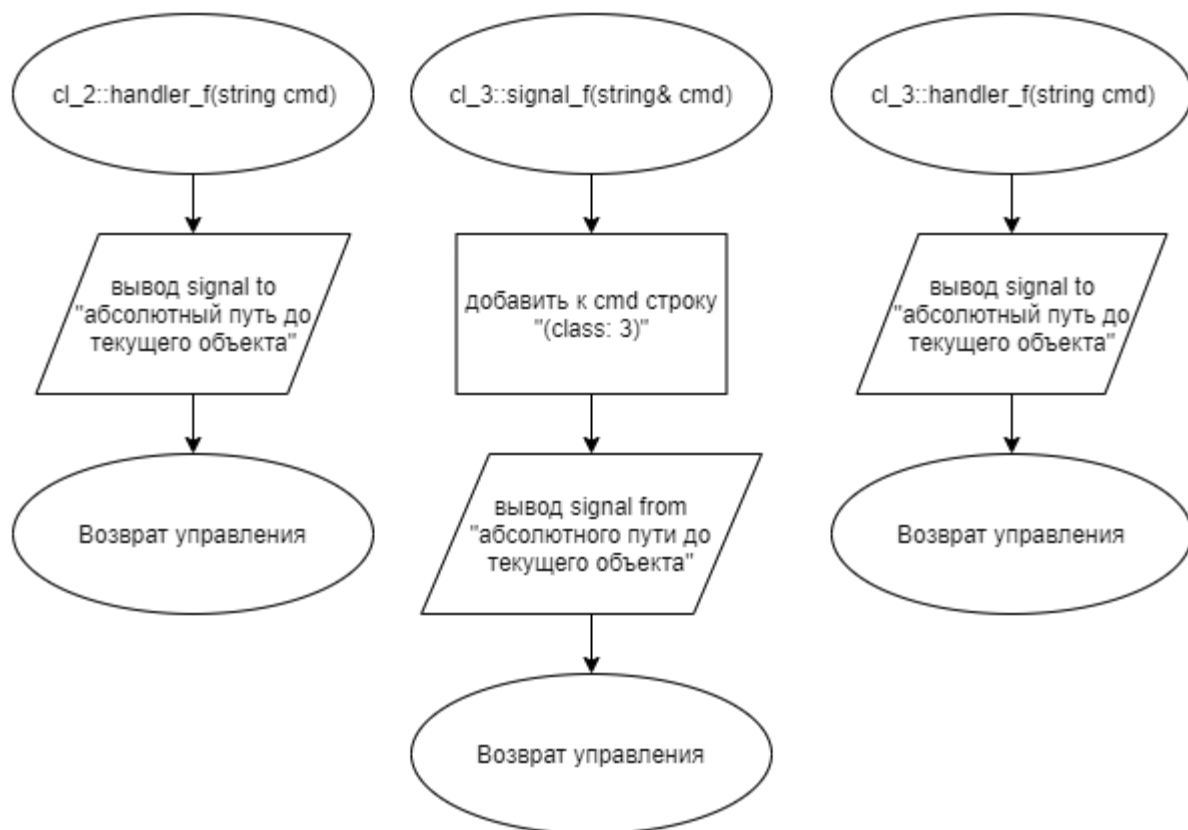


Рисунок 11 – Блок-схема алгоритма

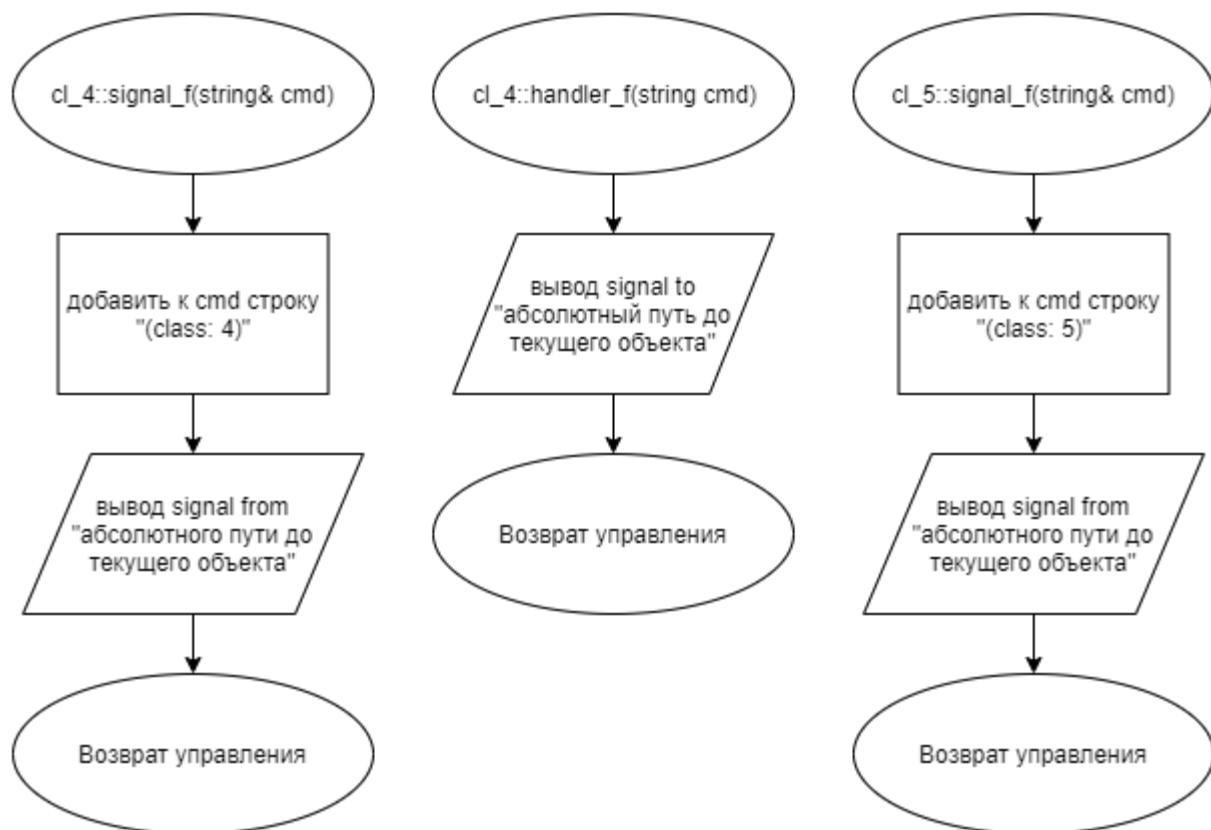


Рисунок 12 – Блок-схема алгоритма

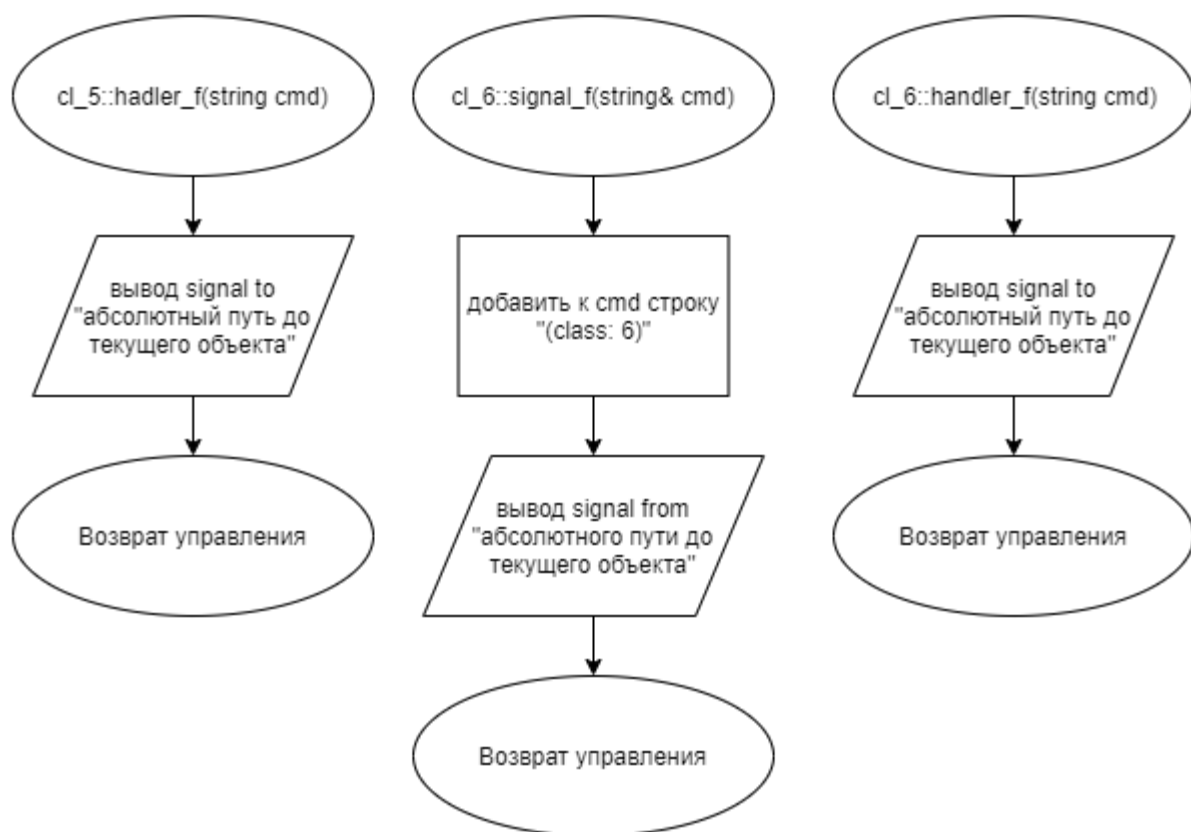


Рисунок 13 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

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

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

void cl_2::handler_f(string cmd)
{
    cout << endl << "Signal to " << this->get_abs_path() << " Text: " << cmd;
}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```

class cl_2 : public cl_base
{
public:
    cl_2(cl_base* parent=nullptr, string name="");
    void signal_f(string&);
    void handler_f(string);
};

#endif

```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```

#include "cl_3.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

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

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

void cl_3::handler_f(string cmd)
{
    cout << endl << "Signal to " << this->get_abs_path() << " Text: " << cmd;
}

```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```

#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

```

```

using namespace std;

class cl_3 : public cl_base
{
public:
    cl_3(cl_base* parent=nullptr, string name="");
    void signal_f(string&);
    void handler_f(string);
};

#endif

```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```

#include "cl_4.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

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

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

void cl_4::handler_f(string cmd)
{
    cout << endl << "Signal to " << this->get_abs_path() << " Text: " << cmd;
}

```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```

#ifndef __CL_4__H
#define __CL_4__H
#include "cl_base.h"

```

```

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

using namespace std;

class cl_4 : public cl_base
{
public:
    cl_4(cl_base* parent=nullptr, string name="");
    void signal_f(string&);
    void handler_f(string);
};

#endif

```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```

#include "cl_5.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

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

void cl_5::signal_f(string& cmd)
{
    cmd += " (class: 2)";
    cout << endl << "Signal from " << this->get_abs_path();
}

void cl_5::handler_f(string cmd)
{
    cout << endl << "Signal to " << this->get_abs_path() << " Text: " << cmd;
}

```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```

#ifndef __CL_5__H

```

```

#define __CL_5__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_5 : public cl_base
{
public:
    cl_5(cl_base* parent=nullptr, string name="");
    void signal_f(string&);
    void handler_f(string);
};

#endif

```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```

#include "cl_6.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

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

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

void cl_6::handler_f(string cmd)
{
    cout << endl << "Signal to " << this->get_abs_path() << " Text: " << cmd;
}

```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_6 : public cl_base
{
public:
    cl_6(cl_base* parent=nullptr, string name="");
    void signal_f(string&);
    void handler_f(string);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_application::cl_application() : cl_base(nullptr, "") {}

void cl_application::build_tree_objects()
{
    string root_name, child_name;
    int class_num = 0;
    cin >> root_name;

    set_object_name(root_name);
```



```

set_state(1);

while (true)
{
    cin >> root_name;
    if (root_name == "endtree")
    {
        break;
    }

    cin >> child_name >> class_num;

    auto p = get_by_path(root_name);

    cl_base* obj;

    switch (class_num)
    {
        case 2:
            obj = new cl_2(p, child_name);
            break;
        case 3:
            obj = new cl_3(p, child_name);
            break;
        case 4:
            obj = new cl_4(p, child_name);
            break;
        case 5:
            obj = new cl_5(p, child_name);
            break;
        case 6:
            obj = new cl_6(p, child_name);
            break;
        default:
            obj = nullptr;
    }
    if (obj)
    {
        obj->set_state(1);
    }
}

print_object_tree();

while (true)
{
    cin >> root_name;
    if (root_name == "end_of_connections")
    {
        break;
    }

    cin >> child_name;

    auto found = get_by_path(root_name);

```

```

        auto target = get_by_path(child_name);

        if (!found || !target)
        {
            continue;
        }
        found->connect(SIGNAL_D(cl_base::signal_f),
HANDLER_D(cl_base::handler_f));
    }
}

void cl_application::print_object_tree()
{
    cout << "Object tree" << endl;
    show_object_tree(false);
}

int cl_application::exec_app()
{
    string cmd, path;
    int state = 0;

    while (true)
    {
        cin >> cmd;
        if (cmd == "END")
        {
            break;
        }

        cin >> path;
        auto found = get_by_path(path);
        if (!found)
        {
            cout << endl << "Object " << path << " not found";
            continue;
        }

        cl_base* target{};
        if (cmd == "SET_CONNECT" || cmd == "DELETE_CONNECT")
        {
            cin >> path;
            target = get_by_path(path);
            if (!target)
            {
                cout << endl << "Handler object " << path << " not found";
                continue;
            }
        }
        if (cmd == "EMIT")
        {
            getline(cin, path);
            found->emit(SIGNAL_D(cl_base::signal_f), path);
        }
        else if (cmd == "SET_CONNECT")

```

```

        {
            found->connect(SIGNAL_D(cl_base::signal_f),           target,
HANDLER_D(cl_base::handler_f));
        }
        else if (cmd == "DELETE_CONNECT")
        {
            found->disconnect(SIGNAL_D(cl_base::signal_f),       target,
HANDLER_D(cl_base::handler_f));
        }
        else if (cmd == "SET_CONDITION")
        {
            cin >> state;
            found->set_state(state);
        }
    }

    return 0;
}

```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```

#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_application : public cl_base
{
public:
    cl_application();
    void build_tree_objects();
    void print_object_tree();
    int exec_app();
};

#endif

```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_base::cl_base(cl_base* parent, string name): p_parent(parent),
object_name(name)
{
    if (parent != nullptr)
    {
        parent->children.push_back(this);
    }
}

cl_base::~~cl_base()
{
    for (auto c: children)
    {
        delete c;
    }
}

bool cl_base::set_object_name(string object_name)
{
    if (!p_parent)
    {
        this->object_name = object_name;
        return true;
    }

    it_child = children.begin();

    while (it_child != children.end())
    {
        if ((*it_child)->get_object_name() == object_name) && ((*it_child) !=
this))
        {
            return false;
        }
        it_child++;
    }

    this->object_name = object_name;
    return true;
}

string cl_base::get_object_name()
{

```

```

    return object_name;
}

cl_base* cl_base::get_parent()
{
    return p_parent;
}

void cl_base::show_object_tree(bool show_state)
{
    show_object_next(0, show_state);
}

void cl_base::show_object_next(int i_level, bool show_state)
{
    string s_space = "";
    if (i_level > 0)
    {
        s_space.append(4 * i_level, ' ');
    }
    cout << s_space << get_object_name();

    if (show_state)
    {
        if (get_state())
        {
            cout << " is ready";
        }
        else
        {
            cout << " is not ready";
        }
    }

    for (auto c : children)
    {
        cout << endl;
        c->show_object_next(i_level + 1, show_state);
    }
}

cl_base* cl_base::get_child(string child_name)
{
    for (auto child : children)
    {
        if (child->get_object_name() == child_name)
        {
            return child;
        }
    }

    return nullptr;
}

```

```

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

void cl_base::set_state(bool state)
{
    if (get_parent() && !get_parent()->get_state())
    {
        this->state = false;
    }
    else
    {
        this->state = state;
    }
    if (!state)
    {
        for (auto c : children)
        {
            c->set_state(state);
        }
    }
}

cl_base* cl_base::find(string object_name)
{
    int count = 0;
    auto found = find_internal(object_name, count);
    if (count > 1)
    {
        return nullptr;
    }
    return found;
}

cl_base* cl_base::find_internal(string object_name, int& count)
{
    cl_base* found = nullptr;
    if (object_name == get_object_name())
    {
        count++;
        found = this;
    }

    for (auto c : children)
    {
        auto obj = c->find_internal(object_name, count);
        if (obj)
        {
            found = obj;
        }
    }

    return found;
}

```

```

cl_base* cl_base::find_on_whole_tree(string object_name)
{
    auto p = this;
    while (p->get_parent())
    {
        p = p->get_parent();
    }
    return p->find(object_name);
}

bool cl_base::is_subordinate(cl_base* obj)
{
    for (auto child: children)
    {
        if (child == obj)
        {
            return true;
        }

        if (child->is_subordinate(obj))
        {
            return true;
        }
    }
    return false;
}

bool cl_base::move(cl_base* new_parent)
{
    if (!get_parent() || !new_parent || new_parent-
>get_child(get_object_name()) || is_subordinate(new_parent))
    {
        return false;
    }
    get_parent()->remove_child(this);
    new_parent->children.push_back(this);
    p_parent = new_parent;
    return true;
}

void cl_base::delete_child(string child_name)
{
    for (auto it = children.begin(); it != children.end(); it++)
    {
        if ((*it)->get_object_name() == child_name)
        {
            delete *it;
            children.erase(it);
            break;
        }
    }
}

cl_base* cl_base::get_by_path(string path)

```

```

{
    if (path.substr(0, 2) == "//")
    {
        return find_on_whole_tree(path.substr(2));
    }
    else if (path == ".")
    {
        return this;
    }
    else if (path[0] == '.')
    {
        return find(path.substr(1));
    }
    else if (path[0] == '/')
    {
        auto new_path = path.substr(1);
        auto cur = this;
        while (cur->get_parent())
        {
            cur = cur->get_parent();
        }
        if (path == "/")
        {
            return cur;
        }
        return cur->get_by_path(new_path);
    }
    auto pos = path.find('/');
    auto child = get_child(path.substr(0, pos));
    if (!child || pos == string::npos) // -1
    {
        return child;
    }
    return child->get_by_path(path.substr(pos + 1));
}

void cl_base::remove_child(cl_base* child)
{
    for (auto it = children.begin(); it != children.end(); it++)
    {
        if ((*it) == child)
        {
            children.erase(it);
            break;
        }
    }
}

string cl_base::get_abs_path()
{
    if (!get_parent())
    {
        return "/";
    }
    auto path = get_parent()->get_abs_path() + "/" + get_object_name();
}

```



```

        if (path.substr(0, 2) == "//")
        {
            return path.substr(1);
        }
        return path;
    }

void cl_base::emit(TYPE_SIGNAL p_signal, string& cmd)
{
    TYPE_HANDLER p_handler;
    cl_base* ob_target;
    if (!get_state())
    {
        return;
    }
    (this->*p_signal) (cmd);

    for (unsigned int i = 0; i < connections.size(); i++)
    {
        if (connections[i]->p_signal == p_signal && connections[i]->ob_target-
>get_state())
        {
            p_handler = connections[i]->p_handler;
            ob_target = connections[i]->ob_target;
            (ob_target->*p_handler) (cmd);
        }
    }
}

void cl_base::connect(TYPE_SIGNAL p_signal, cl_base* ob_target, TYPE_HANDLER
p_handler)
{
    for (unsigned int i = 0; i < connections.size(); i++)
    {
        if (connections[i]->p_signal == p_signal && connections[i]->ob_target
== ob_target && connections[i]->p_handler == p_handler)
        {
            return;
        }
    }

    Connection* sh_temp = new Connection;
    sh_temp->p_signal = p_signal;
    sh_temp->ob_target = ob_target;
    sh_temp->p_handler = p_handler;
    connections.push_back(sh_temp);
}

void cl_base::disconnect(TYPE_SIGNAL p_signal, cl_base* ob_target,
TYPE_HANDLER p_handler)
{
    for (auto it = connections.begin(); it != connections.end(); it++)
    {
        if ((*it)->p_signal == p_signal && (*it)->ob_target == ob_target &&
(*it)->p_handler == p_handler)

```

```

        {
            connections.erase(it);
            break;
        }
    }

void cl_base::signal_f(string& cmd)
{
    cout << endl << "Signal from " << this->get_abs_path();
}

void cl_base::handler_f(string cmd)
{
    cout << endl << "Signal to " << this->get_abs_path() << " Text: " << cmd;
}

```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```

#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <string>
#include <vector>
#include <typeinfo>

using namespace std;

class cl_base;

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

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

struct Connection
{
    TYPE_SIGNAL p_signal;
    cl_base* ob_target;
    TYPE_HANDLER p_handler;
};

class cl_base
{
private:
    string object_name;
    cl_base* p_parent;

```

```

    bool state;
    vector <Connection*> connections;

public:
    vector <cl_base*> children;
    vector <cl_base*> :: iterator it_child;
    cl_base(cl_base* parent=nullptr, string name="");
    ~cl_base();

    bool set_object_name(string object_name);
    string get_object_name();
    cl_base* get_parent();
    void show_object_tree(bool show_state=false);
    void show_object_next(int i_level, bool show_state);
    cl_base* get_child(string child_name);
    bool get_state();
    void set_state(bool state);
    cl_base* find(string object_name);
    cl_base* find_on_whole_tree(string object_name);
    void remove_child(cl_base* child);
    cl_base* find_internal(string object_name, int& count);

    bool move(cl_base* new_parent);
    void delete_child(string child_name);
    cl_base* get_by_path(string path);
    string get_abs_path();
    bool is_subordinate(cl_base* obj);

    void emit(TYPE_SIGNAL p_signal, string& cmd);
    void connect(TYPE_SIGNAL p_signal, cl_base* ob_target, TYPE_HANDLER
p_handler);
    void disconnect(TYPE_SIGNAL p_signal, cl_base* ob_target, TYPE_HANDLER
p_handler);
    virtual void signal_f(string& cmd);
    virtual void handler_f(string cmd);
};

#endif

```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```

#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "cl_application.h"

using namespace std;

```

```
int main()
{
    cl_application ob_cl_application;
    ob_cl_application.build_tree_objects();
    ob_cl_application.exec_app();
    return(0);
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 19.

Таблица 19 – 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>

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).