Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

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

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

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

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

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

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

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

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

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

4. Конец цикла.

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

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

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

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

Каждый метод сигнала с новой строки выводит:

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

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

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

Каждый метод обработчика с новой строки выводит:

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

Моделировать работу системы, которая выполняет следующие команды с параметрами:

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

целевого объекта» – устанавливает связь;

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

Реализовать алгоритм работы системы:

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

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

1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве

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

После ввода состава дерева иерархии построчно вводится:

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

Ввод информации для построения связей завершается строкой, которая содержит:

```
«end_of_connections»
```

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

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

Команда END присутствует обязательно.

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

Если не найден объект по координате:

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

Если не найден целевой объект по координате:

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

Пример ввода:

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

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

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

Если отработал метод обработчика:

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

Пример вывода:

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

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

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- функция override для перезапись;
- функция typedef для определение нового типа;
- функция getline для Запись строки с носителя;
- iterator - изменяемый объект внутри vector.

Класс cl_1:

- функционал:
    - метод signal_f — Отправитель сигнала;
    - метод handler_f — Получатель сигнала;
    - метод get_class — возвращает номер класса.

Класс cl_base:

- свойства/поля:
    - поле структура хранения функций сигналов:
        - наименование — struct;
        - тип — o_sh;
        - модификатор доступа — private;
    - поле Массив установленных соединений:
        - наименование — connects;
        - тип — vector <o_sh*>;
        - модификатор доступа — private;
- функционал:
    - метод Set_all_ready — Установить состояния всех объектов на 1;
    - метод Absolute — Получить абсолютный адрес объекта;
    - метод get_class — Получить номер класса;
    - метод emit_signal — Отправить сообщение;

- метод set_connection — Установить связь;
- метод delete_connection — Удалить связь.

Класс cl_application:

- функционал:
  - метод Tree — Создание дерева иерархии;
  - метод Start — Запуск программы.

Класс cl_2:

- функционал:
  - метод signal_f — Отправитель сигнала;
  - метод handler_f — Получатель сигнала;
  - метод get_class — возвращает номер класса.

Класс cl_3:

- функционал:
  - метод signal_f — Отправитель сигнала;
  - метод handler_f — Получатель сигнала;
  - метод get_class — возвращает номер класса.

Класс cl_4:

- функционал:
  - метод signal_f — Отправитель сигнала;
  - метод handler_f — Получатель сигнала;
  - метод get_class — возвращает номер класса.

Класс cl_5:

- функционал:
  - метод signal_f — Отправитель сигнала;
  - метод handler_f — Получатель сигнала;
  - метод get_class — возвращает номер класса.

Класс cl_6:

- функционал:
    - метод signal_f — Отправитель сигнала;
    - метод handler_f — Получатель сигнала;
    - метод get_class — возвращает номер класса.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_1			Производный класс от cl_base	
2	cl_base			базовый класс	
		cl_1	public		1
		cl_application	public		3
		cl_2	public		4
		cl_3	public		5
		cl_4	public		6
		cl_5	public		7
		cl_6	public		8
3	cl_application			Программа	
4	cl_2			Производный класс от cl_base	
5	cl_3			Производный класс от cl_base	
6	cl_4			Производный класс от cl_base	
7	cl_5			Производный класс от cl_base	
8	cl_6			Производный класс от cl_base	

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 signal_f класса cl_1

Функционал: Отправитель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода signal_f класса cl_1

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal from абсолютная координата объекта	2
2		msg+= " (class: 2)"	∅

3.2 Алгоритм метода handler_f класса cl_1

Функционал: Получатель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода handler_f класса cl_1

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal to абсолютная координата объекта msg	∅

3.3 Алгоритм метода get_class класса cl_1

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: int - номер класса.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_class класса cl_1

№	Предикат	Действия	№ перехода
1		Вернуть 2	∅

3.4 Алгоритм метода Set_all_ready класса cl_base

Функционал: Установить состояния всех объектов на 1.

Параметры: нет.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода Set_all_ready класса cl_base

№	Предикат	Действия	№ перехода
1		Вызов метода can_off с параметром 1 для объекта this	2
2	auto sub:this->subordinate_objects	Вызов метода Set_all_ready с параметром 1 для объекта sub	2
			∅

3.5 Алгоритм метода Absolute класса cl_base

Функционал: Получить абсолютный адрес объекта.

Параметры: нет.

Возвращаемое значение: string - абсолютная координата.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода Absolute класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация p_root - корня	2
2	this==p_root	Вернуть "/"	∅
		инициализация cl_base* h = this->Get_baseptr(); string text = ""	3
3	h	text+="/"+ h->Get_name()	4
		text+="/" + this->Get_name()	5
4		h=h->Get_baseptr()	3
5		Вернуть text	∅

3.6 Алгоритм метода get_class класса cl_base

Функционал: Получить номер класса.

Параметры: нет.

Возвращаемое значение: int - номер класса.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_class класса cl_base

№	Предикат	Действия	№ перехода
1		вернуть 0	∅

3.7 Алгоритм метода emit_signal класса cl_base

Функционал: Отправить сообщение.

Параметры: TYPE_SIGNAL signal - функция отправления сигнала; string msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода emit_signal класса cl_base

№	Предикат	Действия	№ перехода
1	this->status ==0		∅
		вызвать метод *signal с параметром msg для this	2
2		для каждой связи в connects с совпадающим signal и status !=0 вызвать метод *(TYPE_HANDLER handler = con->handler) с параметром msg для объекта this	∅

3.8 Алгоритм метода set_connection класса cl_base

Функционал: Установить связь.

Параметры: TYPE_SIGNAL signal - функция отправления сигнала; cl_base * target - объект получатель; TYPE_HANDLER handler - функция принятия сигнала.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода set_connection класса cl_base

№	Предикат	Действия	№ перехода
1		объявление o_sh *p_value	2
2		инициализация i=0	3
3	i<connects.size()	i++	4
		добавить новое соединение в connects	∅
4	signal, target, handler connects[i] == переданным параметрам	закончить метод	∅
			3

3.9 Алгоритм метода delete_connection класса cl_base

Функционал: Удалить связь.

Параметры: TYPE_SIGNAL signal - функция отправления сигнала; cl_base * target - объект получатель; TYPE_HANDLER handler - функция принятия сигнала.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода delete_connection класса cl_base

№	Предикат	Действия	№ перехода
1		объявление it - iterator vector <o_sh*>; it=connects.begin()	2
2	it<= connects.end()	it++; инициализация i=0	3
			∅
3	i<connects.size()	i++;	4
			2
4	параметры it == переданные параметры	connects.erase(it)	3
			3

3.10 Алгоритм метода Tree класса cl_application

Функционал: Создание дерева иерархии.

Параметры: нет.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода Tree класса cl_application

№	Предикат	Действия	№ перехода
1		создать иерархию объектов	2

№	Предикат	Действия	№ перехода
2		ввод координат объектов отдающего и прнимающего сигнал	3
3		вызов для объекта отдающего сигнал метод создания связи с параметром объекта принимающего объекта пока есть ввод	∅

3.11 Алгоритм метода Start класса cl_application

Функционал: Запуск программы.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности выполнения алгоритма.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода Start класса cl_application

№	Предикат	Действия	№ перехода
1		установка status всех объектов на 1	2
2	true	ввод string command с клавиатуры	3
		break;	5
3	command!="END"	ввод координаты coord1 с клавиатуры и создание по ней объекта ob	4
		break;	5
4	command == "EMIT"	ввод msg с клавиатуры; вызов метода emit_signal для объекта ob с параметром msg	2
			6
5		вывод дерева	∅
6	command == "SET_CONNECT"	ввод координаты coord2 с клавиатуры и создание по ней объекта target; вызов метода set_connection для объекта ob с параметром target	2
			7
7	command ==	ввод координаты coord2 с клавиатуры и создание	2

№	Предикат	Действия	№ перехода
	"DELETE_CONNECT"	по ней объекта target; вызов метода delete_connection для объекта ob с параметром target	
			8
8	command == "SET_CONDITION"	ввод i_state с клавиатуры; вызов метода can_off с параметро i_state для объекта ob	2
			2

3.12 Алгоритм метода signal_f класса cl_2

Функционал: Отправитель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода signal_f класса cl_2

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal from абсолютная координата объекта	2
2		msg+= " (class: 3)"	∅

3.13 Алгоритм метода handler_f класса cl_2

Функционал: Получатель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода handler_f класса cl_2

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal to абсолютная координата объекта msg	∅

3.14 Алгоритм метода get_class класса cl_2

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: int - номер класса.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода get_class класса cl_2

№	Предикат	Действия	№ перехода
1		вернуть 3	∅

3.15 Алгоритм метода signal_f класса cl_3

Функционал: Отправитель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода signal_f класса cl_3

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal from абсолютная координата объекта	2
2		msg+= " (class: 4)"	∅

3.16 Алгоритм метода handler_f класса cl_3

Функционал: Получатель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода handler_f класса cl_3

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal to абсолютная координата объекта msg	∅

3.17 Алгоритм метода get_class класса cl_3

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: int - номер класса.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода get_class класса cl_3

№	Предикат	Действия	№ перехода
1		вернуть 4	∅

3.18 Алгоритм метода signal_f класса cl_4

Функционал: Отправитель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода signal_f класса cl_4

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal from абсолютная координата объекта	2
2		msg+= " (class: 5)"	∅

3.19 Алгоритм метода handler_f класса cl_4

Функционал: Получатель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода handler_f класса cl_4

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal to абсолютная координата объекта msg	∅

3.20 Алгоритм метода get_class класса cl_4

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: int - номер класса.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода get_class класса cl_4

№	Предикат	Действия	№ перехода
1		вернуть 5	∅

3.21 Алгоритм метода signal_f класса cl_5

Функционал: Отправитель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода signal_f класса cl_5

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal from абсолютная координата объекта	2
2		msg+= " (class: 6)"	∅

3.22 Алгоритм метода handler_f класса cl_5

Функционал: Получатель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода handler_f класса cl_5

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal to абсолютная координата объекта msg	∅

3.23 Алгоритм метода get_class класса cl_5

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: int - номер класса.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода get_class класса cl_5

№	Предикат	Действия	№ перехода
1		вернуть 6	∅

3.24 Алгоритм метода signal_f класса cl_6

Функционал: Отправитель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода signal_f класса cl_6

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal from абсолютная координата объекта	2
2		msg+= " (class: 1)"	∅

3.25 Алгоритм метода handler_f класса cl_6

Функционал: Получатель сигнала.

Параметры: string& msg - сообщение.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода handler_f класса cl_6

№	Предикат	Действия	№ перехода
1		Вывести с новой строки Signal to абсолютная координата объекта msg	∅

3.26 Алгоритм метода get_class класса cl_6

Функционал: возвращает номер класса.

Параметры: нет.

Возвращаемое значение: int - номер класса.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода get_class класса cl_6

№	Предикат	Действия	№ перехода
1		вернуть 1	∅

3.27 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности выполнения алгоритма.

Алгоритм функции представлен в таблице 28.

Таблица 28 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		вернуть метод Start	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-18.

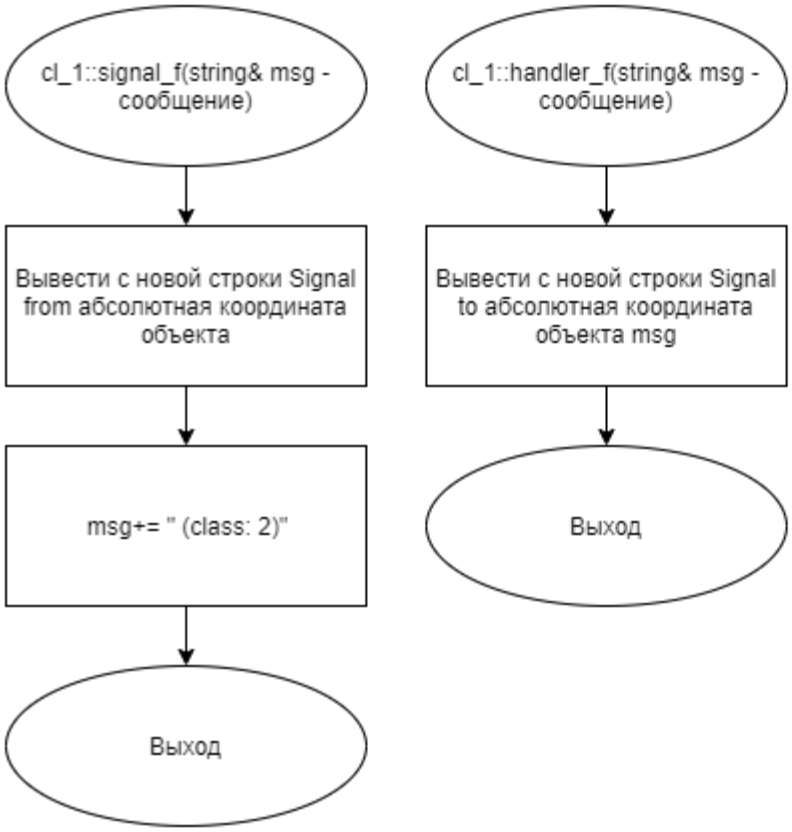

Рисунок 1 – Блок-схема алгоритма

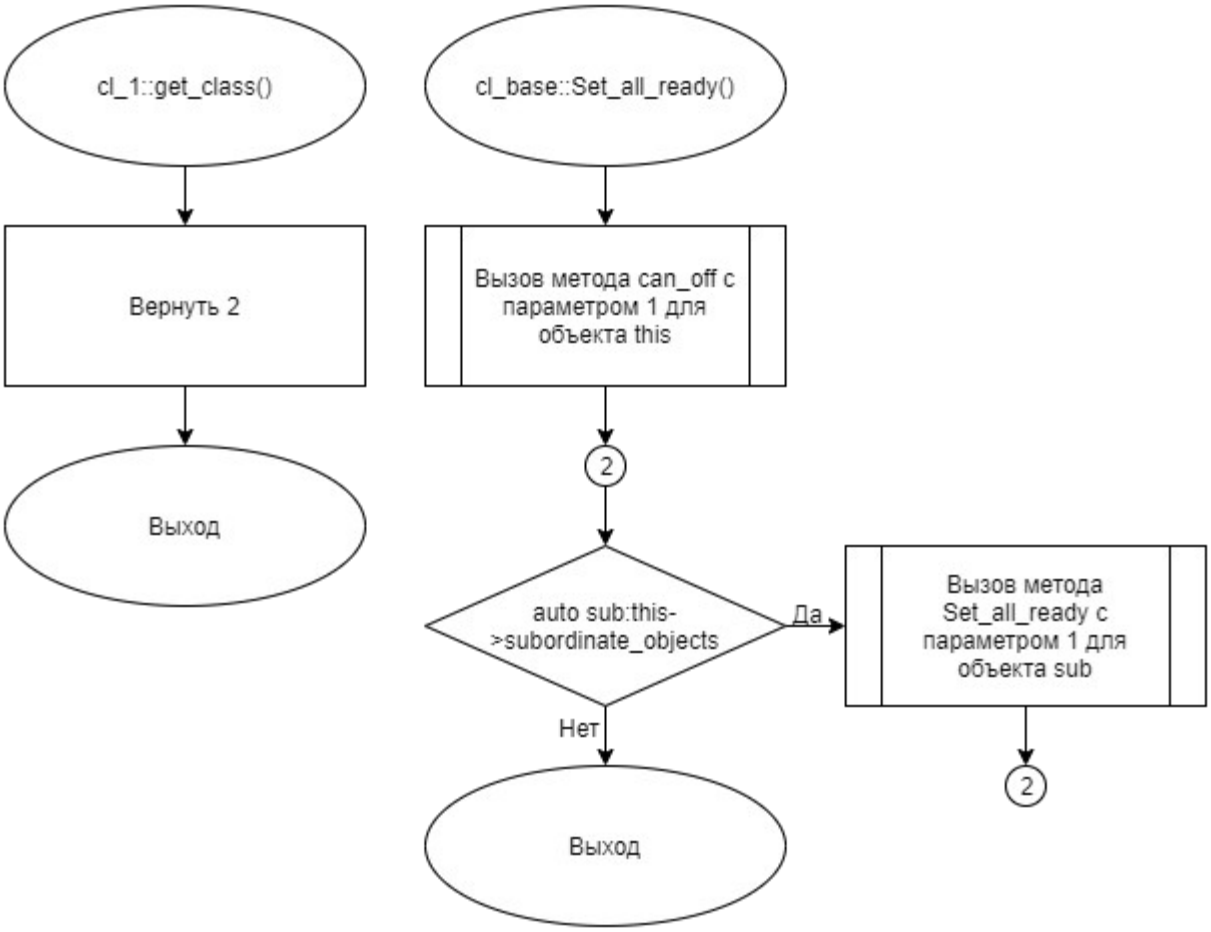

Рисунок 2 – Блок-схема алгоритма

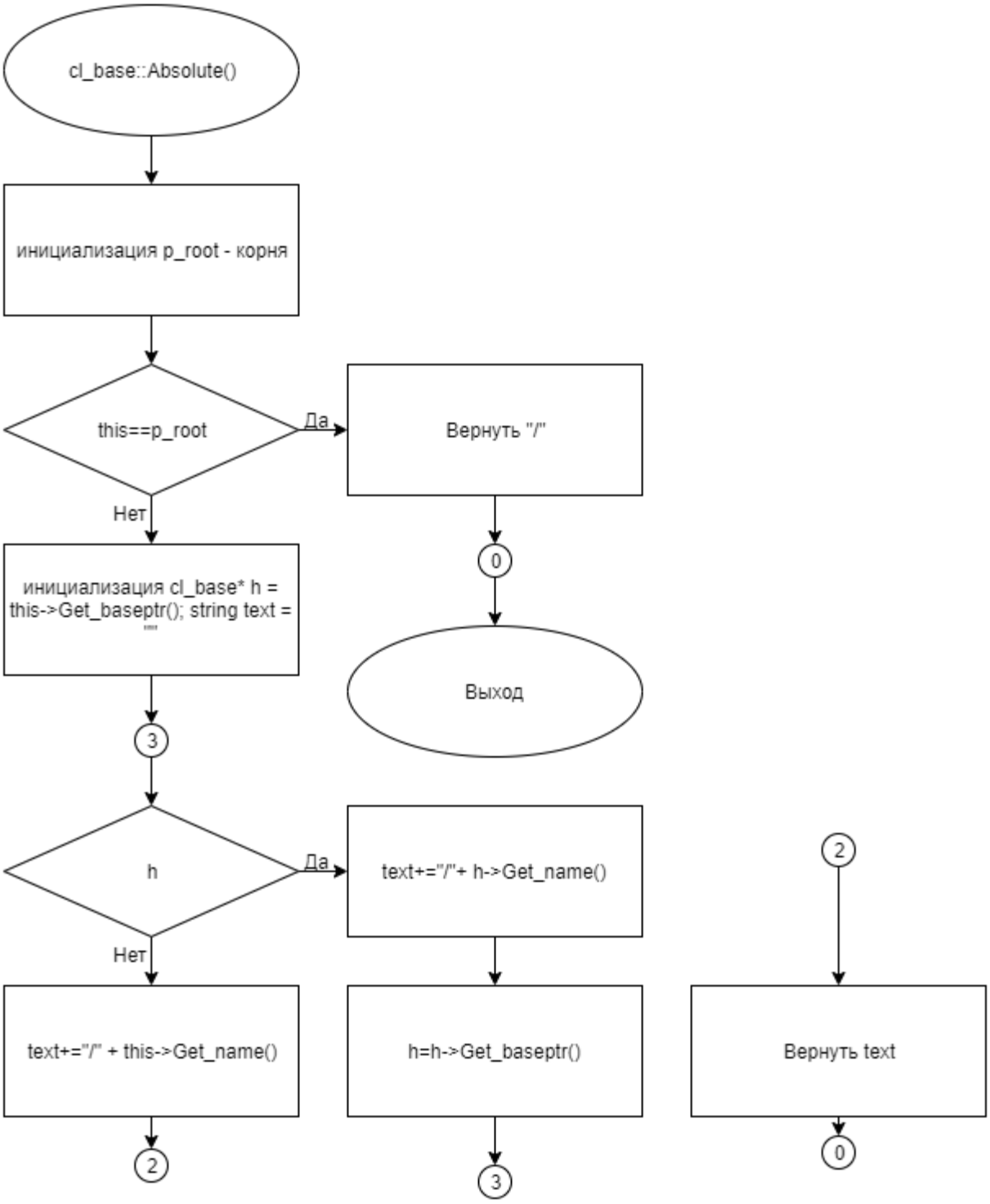

Рисунок 3 – Блок-схема алгоритма

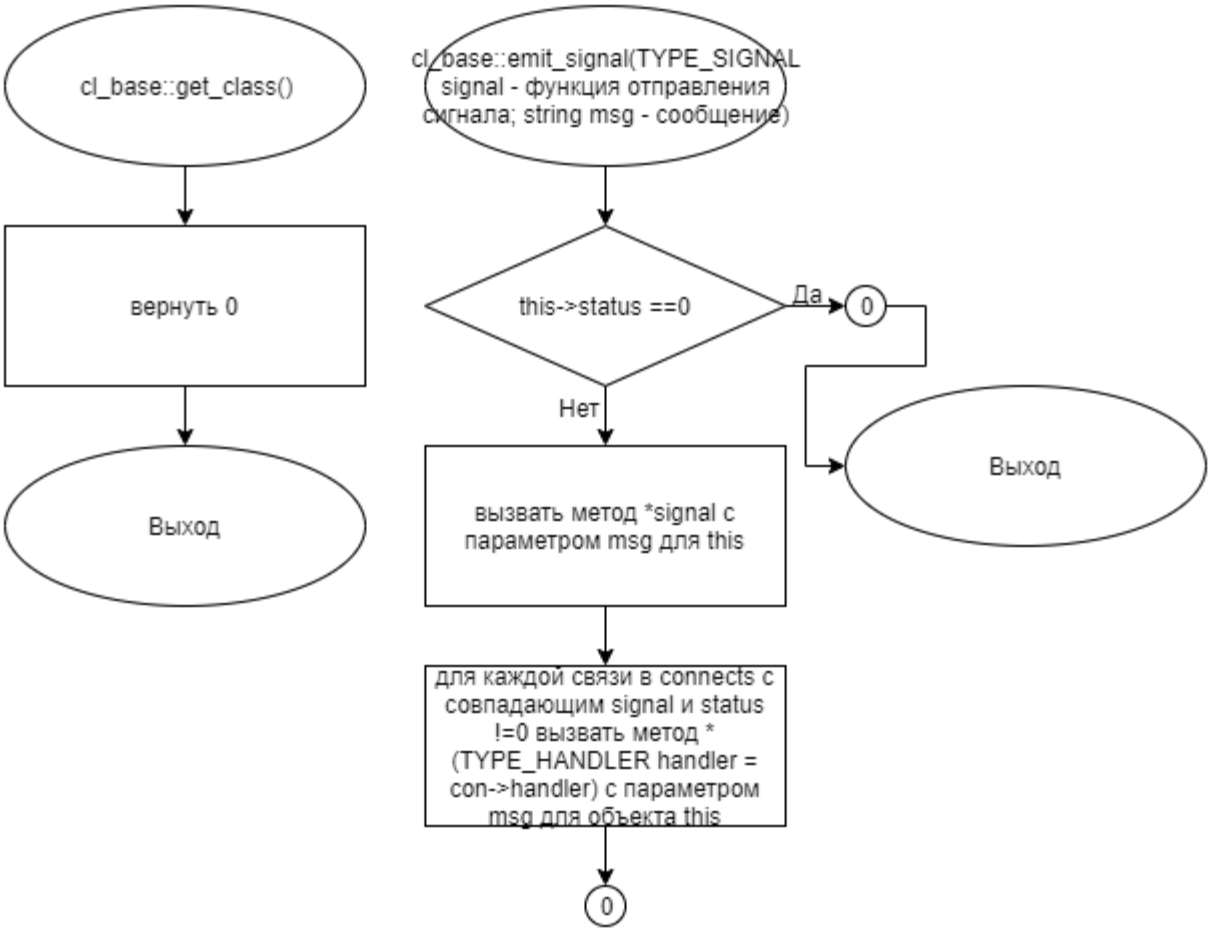

Рисунок 4 – Блок-схема алгоритма

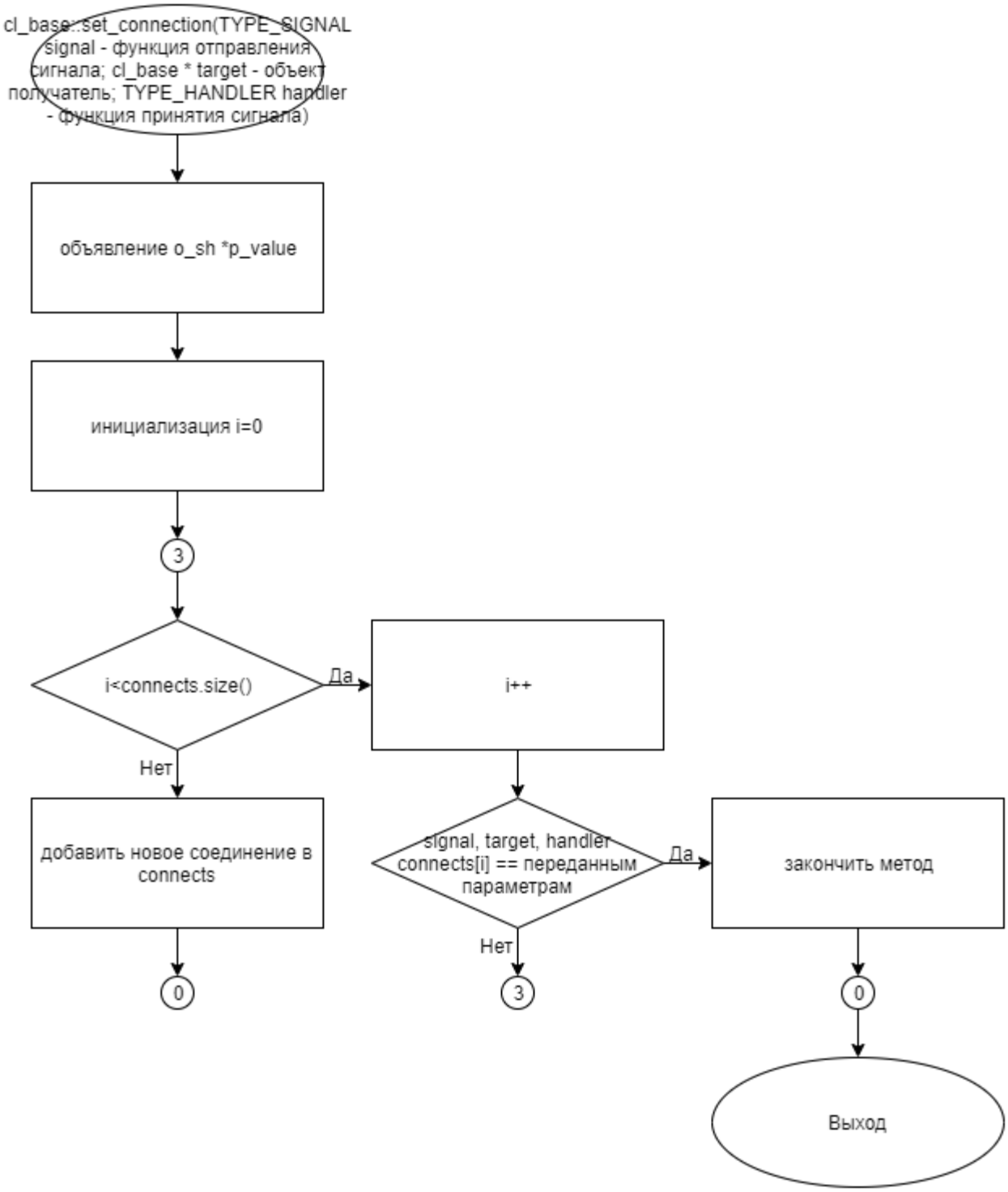

Рисунок 5 – Блок-схема алгоритма

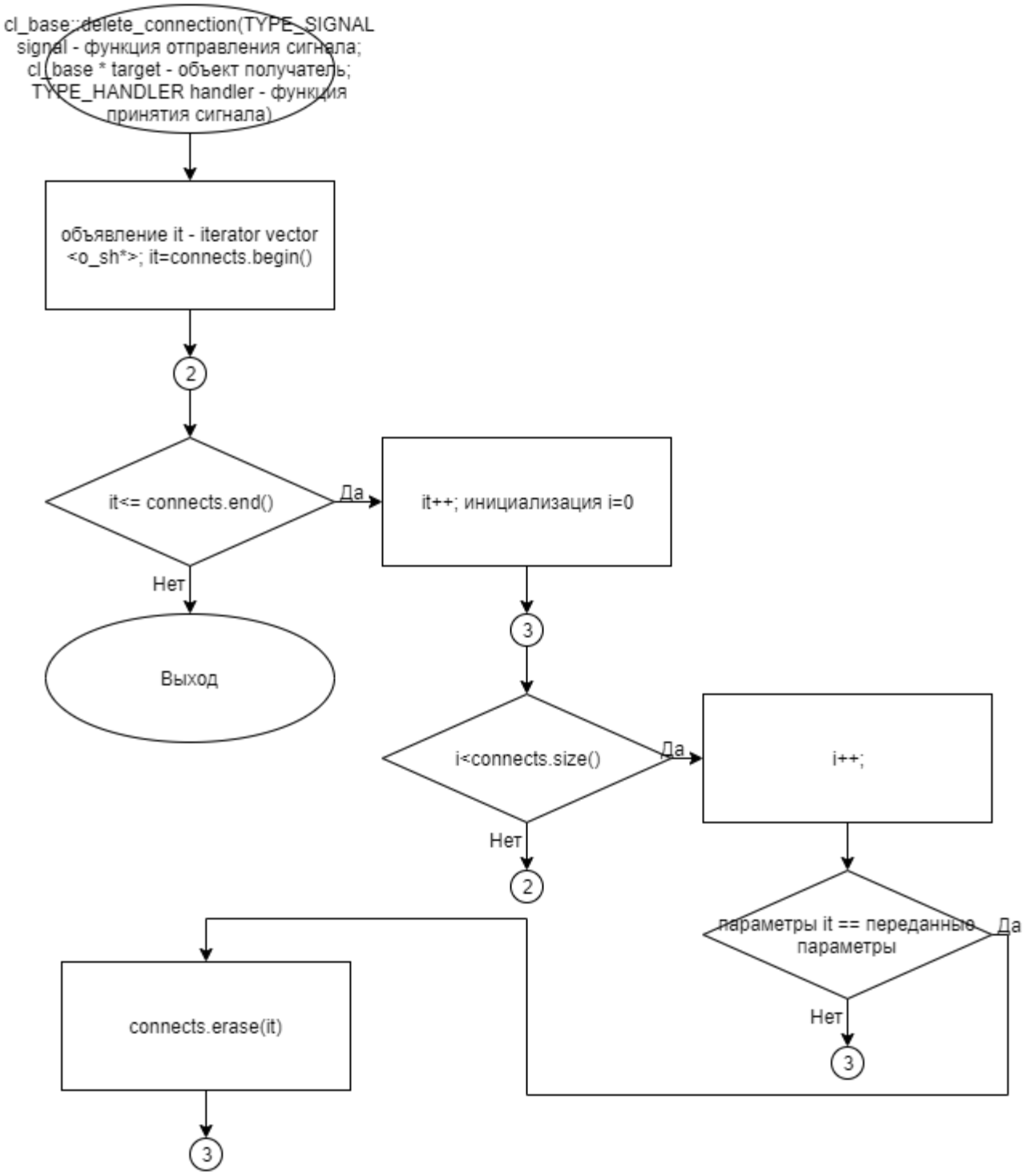

Рисунок 6 – Блок-схема алгоритма

Рисунок 7 – Блок-схема алгоритма

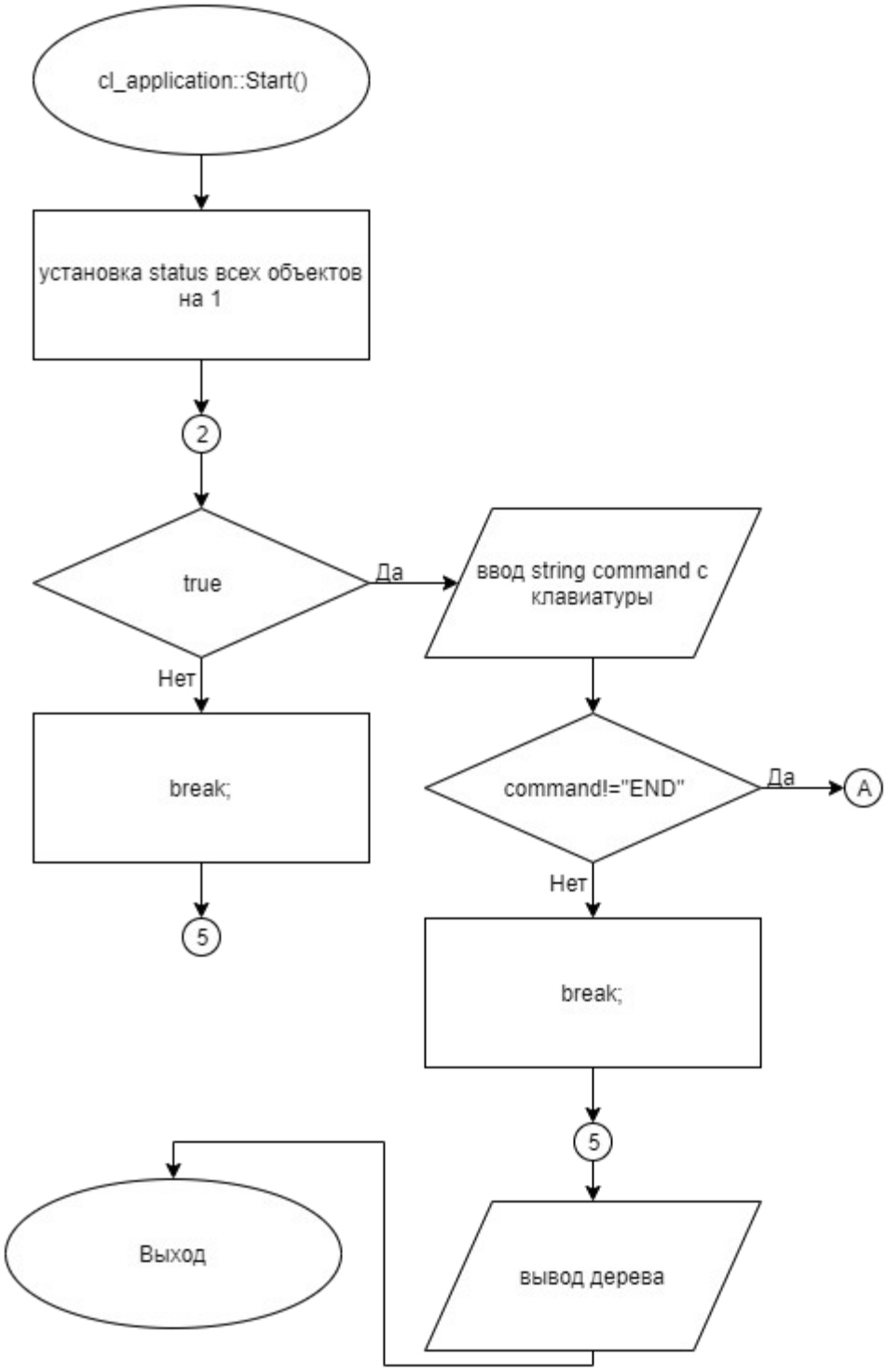

Рисунок 8 – Блок-схема алгоритма

Рисунок 9 – Блок-схема алгоритма

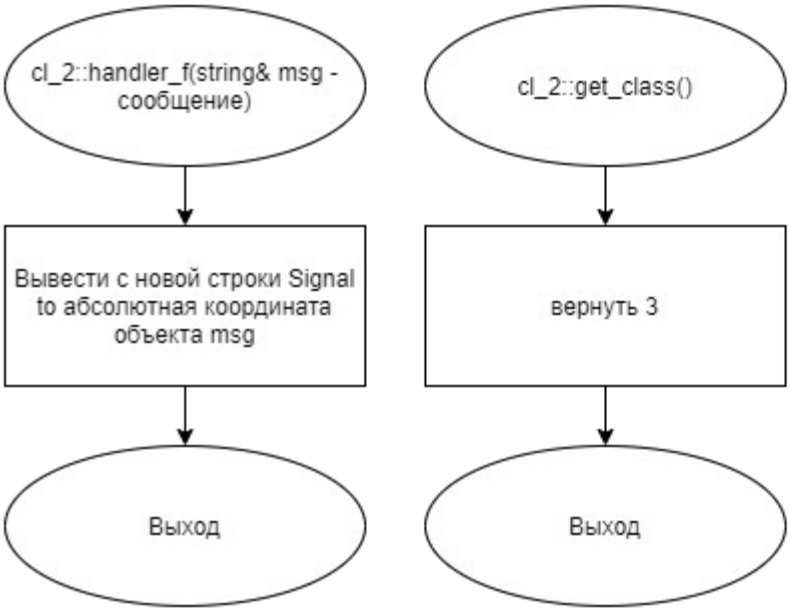

Рисунок 10 – Блок-схема алгоритма

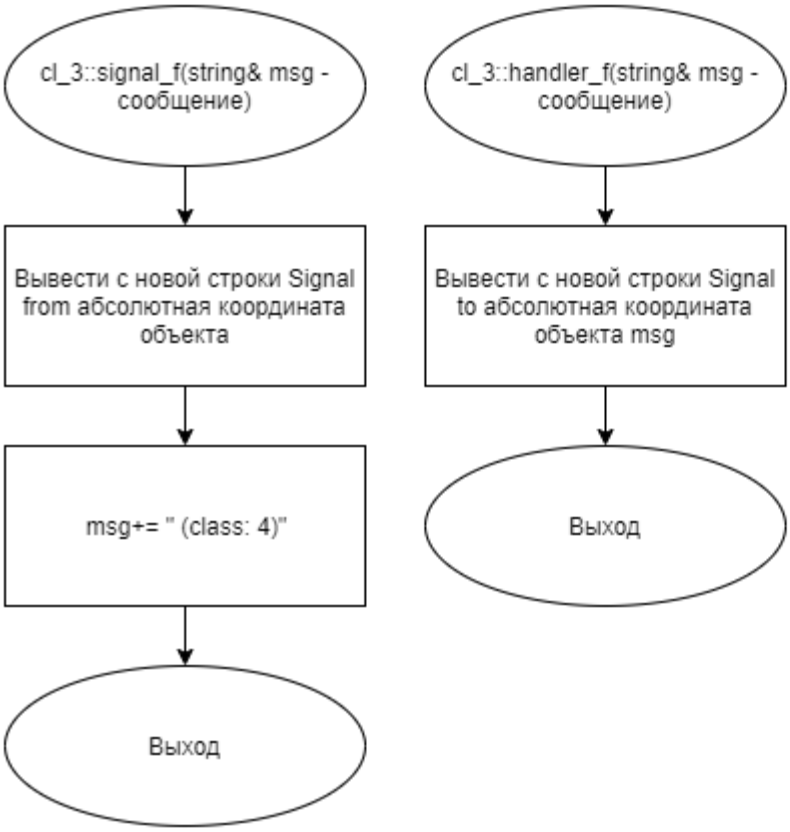

Рисунок 11 – Блок-схема алгоритма

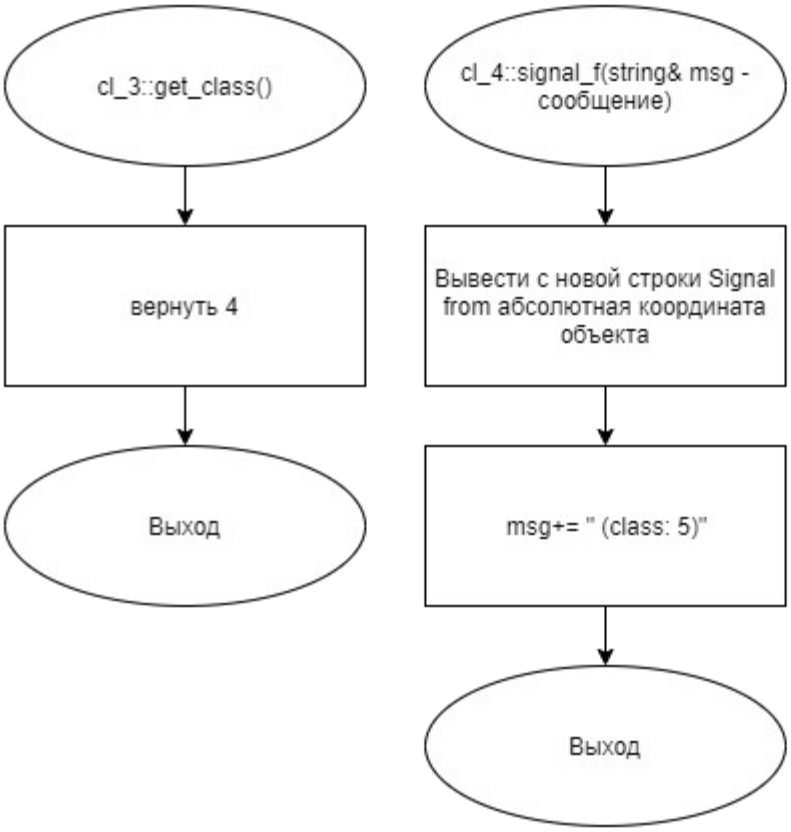

Рисунок 12 – Блок-схема алгоритма

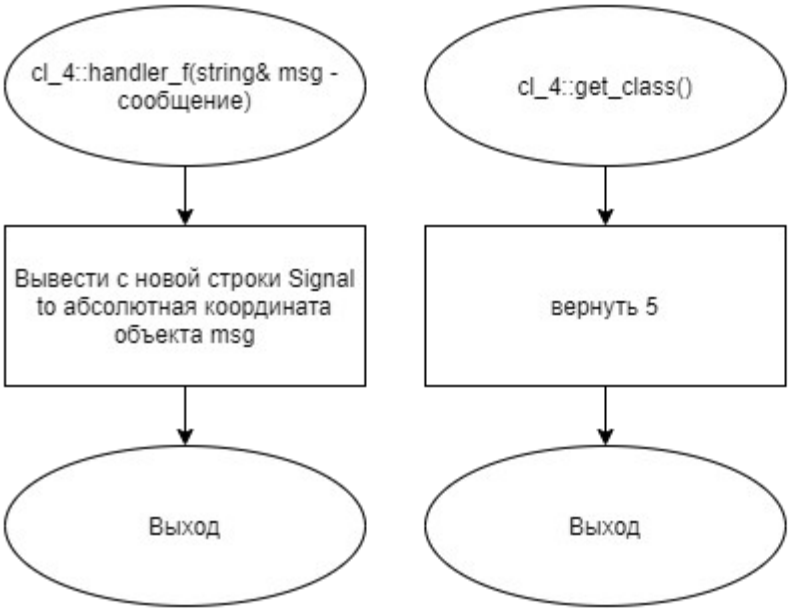

Рисунок 13 – Блок-схема алгоритма

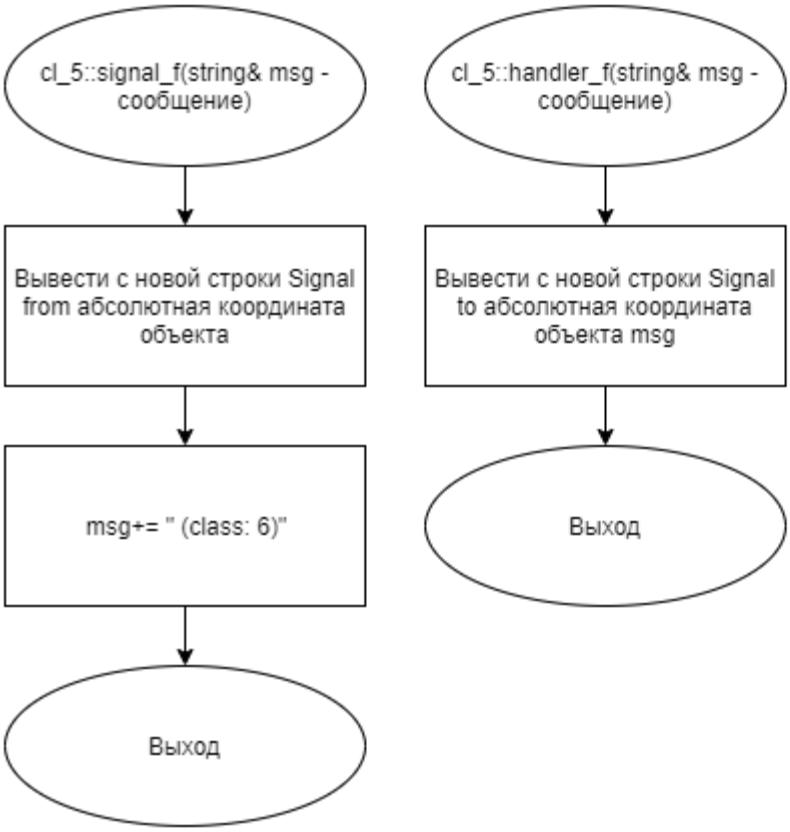

Рисунок 14 – Блок-схема алгоритма

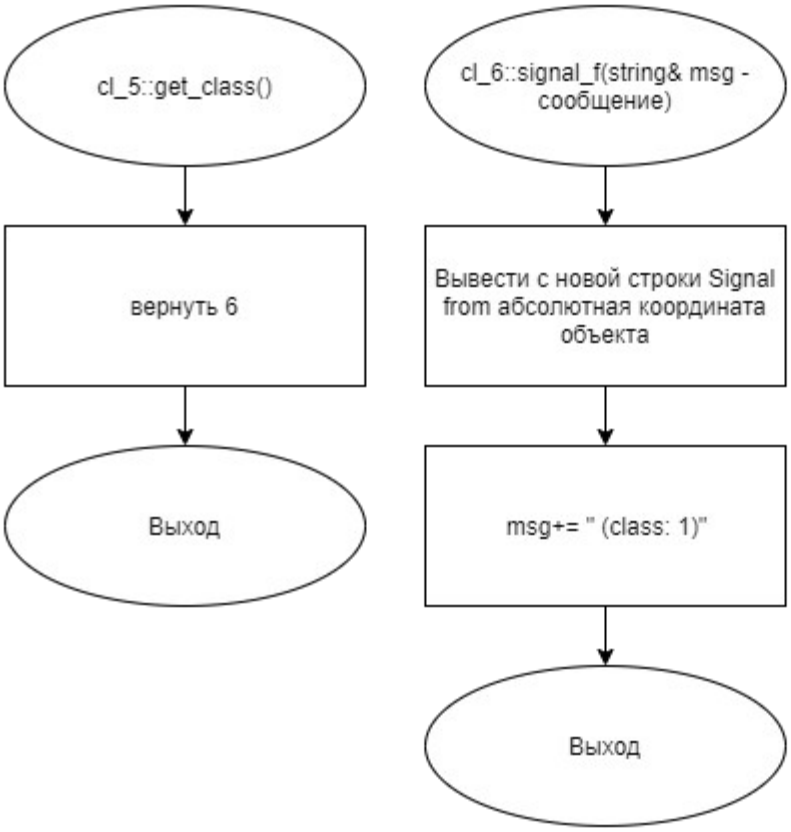

Рисунок 15 – Блок-схема алгоритма

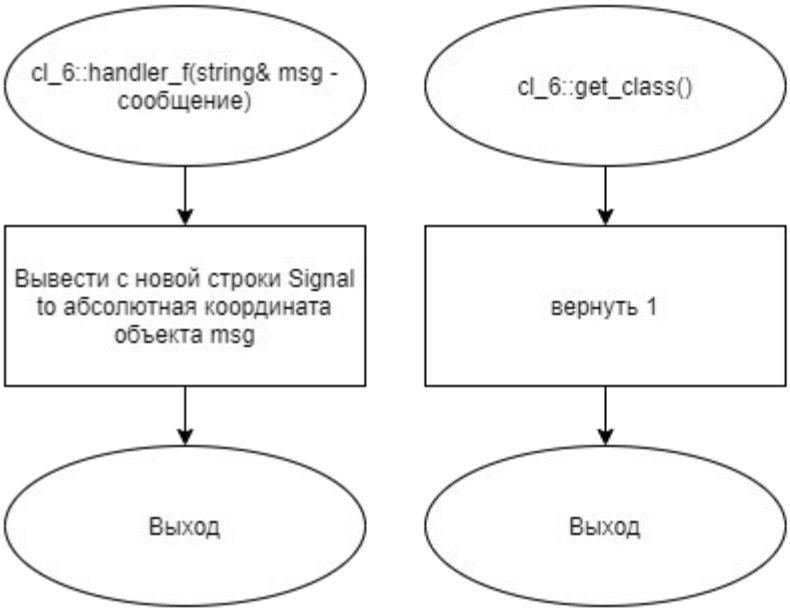

Рисунок 16 – Блок-схема алгоритма

Рисунок 17 – Блок-схема алгоритма

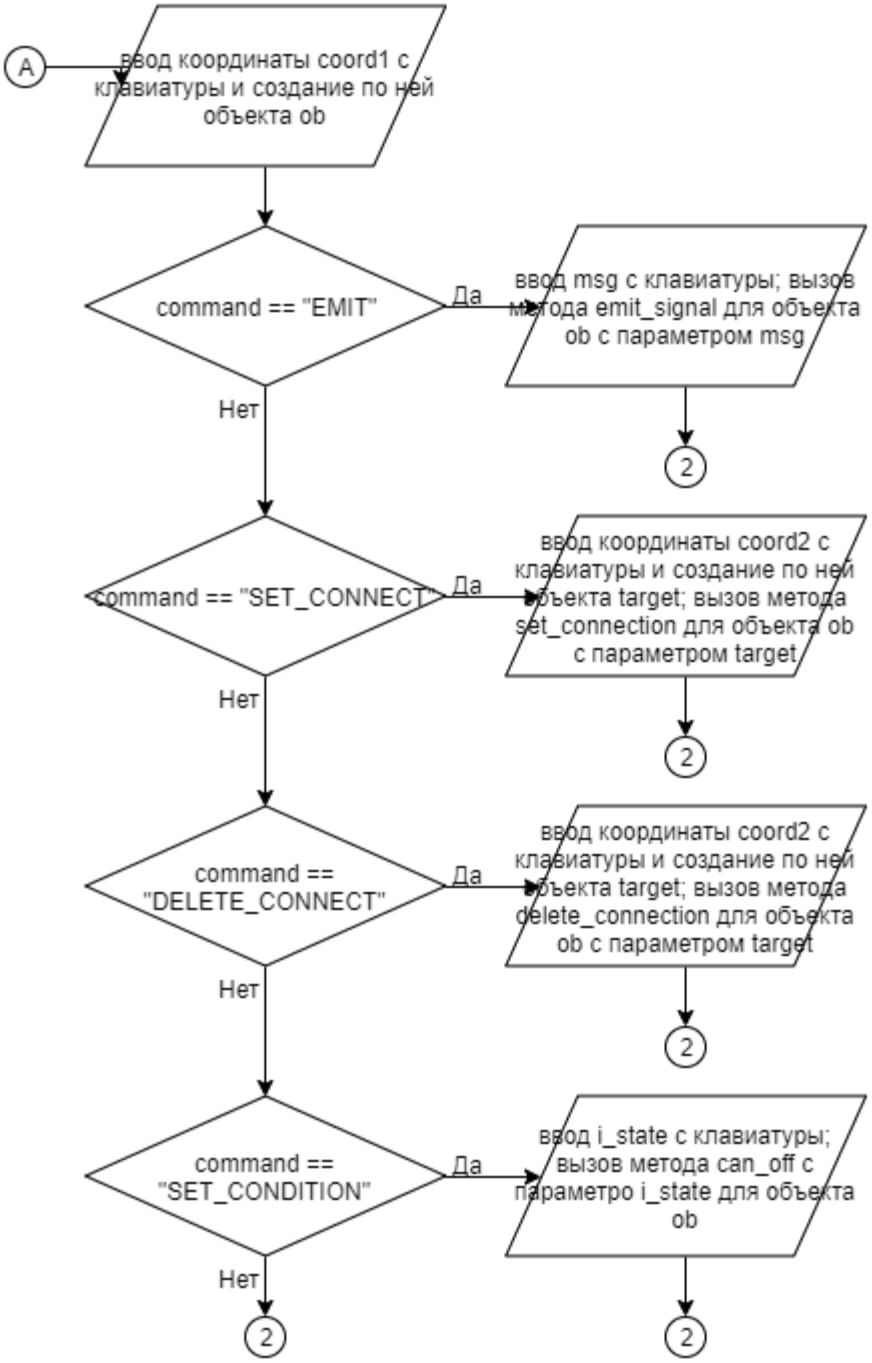

Рисунок 18 – Блок-схема алгоритма

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,     string     s_object_name):cl_base(p_head,
s_object_name){
}

int cl_1::get_class()const{return 1;}
void cl_1::signal_f(string& msg){
   cout<< "Signal from "<<this->Absolute()<<endl;
   msg+= " (class: 2)";
}
void cl_1::handler_f(string& msg){
   cout<< "Signal to "<<this->Absolute()<<" Text: "<<msg<<endl;
}
```

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, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string& msg);
   int get_class() const override;
};
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"
cl_2::cl_2(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}

int cl_2::get_class()const{return 2;}
void cl_2::signal_f(string& msg){
   cout<< "Signal from "<<this->Absolute()<<endl;
   msg+= " (class: 3)";
}
void cl_2::handler_f(string& msg){
   cout<< "Signal to "<<this->Absolute()<<" Text: "<<msg<<endl;
}
```

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, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string& msg);
   int get_class()const override;
};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"
cl_3::cl_3(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

```
int cl_3::get_class()const{return 3;}
void cl_3::signal_f(string& msg){
   cout<< "Signal from "<<this->Absolute()<<endl;
   msg+= " (class: 4)";
}
void cl_3::handler_f(string& msg){
   cout<< "Signal to "<<this->Absolute()<<" Text: "<<msg<<endl;
}
```

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, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string& msg);
   int get_class()const override;
};


#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}

int cl_4::get_class()const{return 4;}
void cl_4::signal_f(string& msg){
   cout<< "Signal from "<<this->Absolute()<<endl;
   msg+= " (class: 5)";
}
void cl_4::handler_f(string& msg){
   cout<< "Signal to "<<this->Absolute()<<" Text: "<<msg<<endl;
}
```

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, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string& msg);
   int get_class()const override;
};
#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"
cl_5::cl_5(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}

int cl_5::get_class()const{return 5;}
void cl_5::signal_f(string& msg){
   cout<< "Signal from "<<this->Absolute()<<endl;
   msg+= " (class: 6)";
}
void cl_5::handler_f(string& msg){
   cout<< "Signal to "<<this->Absolute()<<" Text: "<<msg<<endl;
}
```

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, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string& msg);
   int get_class()const override;
};
#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"
cl_6::cl_6(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}

int cl_6::get_class()const{return 0;}
void cl_6::signal_f(string& msg){
   cout<< "Signal from "<<this->Absolute()<<endl;
   msg+= " (class: 1)";
}
void cl_6::handler_f(string& msg){
   cout<< "Signal to "<<this->Absolute()<<" Text: "<<msg<<endl;
}
```

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, string s_object_name);
   void signal_f(string& msg);
   void handler_f(string& msg);
   int get_class() const override;
};


#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.h"
#include <vector>
cl_application::cl_application(cl_base*
p_head_object):cl_base(p_head_object){}
void cl_application::Tree(){
   string head, coord;

   int i_class, i_state;
   cin>>head;
   cl_base* p_head=this;
   cl_base* p_sub=nullptr;
   Change_name(head);
   //Ввод иерархии
   while(true){
      cin>>head;
      if (head=="endtree"){
         break;
      }
      cin>>coord>> i_class;
      p_head = this->find_obj_bc(head);
      if(!p_head){
         cout<<"Object tree"<<endl;
         Out_fc();
         cout<<endl<<"The head object "<<head<<" is not found";
         cl_base* p_root=this;
         while(p_root->Get_baseptr()){
            p_root=p_root->Get_baseptr();
         }
         p_root->Change_name("EEERROR");
         return;
      }
         /*
         if(this->Find_global(head.substr(1))){
            p_head=this->Find_global(head.substr(1));
         }
         else {
            Out_fc();
            cout<<endl<<"The head object "<<coord<<" is not found";
            return 1;
         }
         */
      /*
      if(p_sub!=nullptr && head==p_sub->Get_name()){
         p_head=p_sub;
      }
      if(head==p_head->Get_name() && p_head->Get_ptr(sub)==nullptr){
         p_sub=new cl_1 (p_head,sub);
      }
      */
      if (p_head && !p_head->Get_ptr(coord)){
```

```
            switch (i_class)
            {
               case 1:
                  p_sub = new cl_6 (p_head, coord);
                  break;
               case 2:
                  p_sub=new cl_1(p_head,coord);
                  break;
               case 3:
                  p_sub=new cl_2(p_head,coord);
                  break;
               case 4:
                  p_sub=new cl_3(p_head,coord);
                  break;
               case 5:
                  p_sub=new cl_4(p_head,coord);
                  break;
               case 6:
                  p_sub=new cl_5(p_head,coord);
                  break;
               default:
                  break;
            }
         }
         else cout<<endl<<coord<<"    Dubbing the name of subordinate objects";
      }
      cout<<"Object tree"<<endl;
      Out_fc();
      cout<<endl;
      cl_base* p_root=this;
      while(p_root->Get_baseptr()){
         p_root=p_root->Get_baseptr();
      }
      cl_base* cur_ob=p_root;
      string coord1, coord2;
      while (true){
         cl_base* ob = nullptr;
         cl_base* target = nullptr;
         cin>>coord1;
         if (coord1 == "end_of_connections") break;
         cin>>coord2;
         ob = find_obj_bc(coord1);
         target = find_obj_bc(coord2);
         if(ob!=nullptr){
            if(target){
               int i_class1 = ob->get_class();
               int i_class2 = target->get_class();
               switch (i_class1){
                  case 0:
                     switch(i_class2){
                           case 0:
                              ob->set_connection(SIGNAL_D(cl_6::signal_f),
   target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->set_connection(SIGNAL_D(cl_6::signal_f),
```

```
      target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->set_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->set_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->set_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->set_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                              break;
                     }
                     break;
                  case 1:
                     switch(i_class2){
                           case 0:
                              ob->set_connection(SIGNAL_D(cl_1::signal_f),
      target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->set_connection(SIGNAL_D(cl_1::signal_f),
      target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->set_connection(SIGNAL_D(cl_1::signal_f),
      target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->set_connection(SIGNAL_D(cl_1::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->set_connection(SIGNAL_D(cl_1::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->set_connection(SIGNAL_D(cl_1::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                              break;
                     }
                     break;
                  case 2:
                     switch(i_class2){
                           case 0:
                              ob->set_connection(SIGNAL_D(cl_2::signal_f),
      target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->set_connection(SIGNAL_D(cl_2::signal_f),
```

```
      target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->set_connection(SIGNAL_D(cl_2::signal_f),
      target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->set_connection(SIGNAL_D(cl_2::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->set_connection(SIGNAL_D(cl_2::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->set_connection(SIGNAL_D(cl_2::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                              break;
                     }
                     break;
                  case 3:
                     switch(i_class2){
                           case 0:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                              break;
                     }
                     break;
                  case 4:
                     switch(i_class2){
                           case 0:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
```

```
      target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                              break;
                     }
                     break;
                  case 5:
                     switch(i_class2){
                           case 0:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                              break;
                     }
                     break;
               }
               //cout<<i_class1<<" "<<i_class2<<endl;
            }
            else cout<<"Handler object "<<coord2<<" not found"<<endl;
         }
         else cout<<"Object "<<coord1<<" not found"<<endl;
      }
```

```
      /*
      while(cin>>head>>i_state){
         p_head = Find_global(head);
         if(p_head!=nullptr) {p_head->can_off(i_state);}
      }
      */
      //cout<<"Статусы заданы\n";

   }
   int cl_application::Start(){
      int i_state;
      string head;
      cl_base* p_root=this;
      while(p_root->Get_baseptr()){
         p_root=p_root->Get_baseptr();
      }
      if(p_root->Get_name()=="EEERROR") return 1;

      //Установка всех готовностей на 1
      p_root->Set_all_ready();
      cl_base * cur_ob=p_root;


      //Команды
      while(true){
         string command, coord1, text, coord2;
         text="";
         coord2="";
         cin>>command;
         if(command=="END") break;
         cin>>coord1;
         //cout<<endl<<coord1<<endl;
         cl_base* ob = cur_ob->find_obj_bc(coord1);
         if (ob!=nullptr){
            if (command=="EMIT"){
               getline(cin,text);
               //cout<<endl<<text<<endl;
               int i_class = ob->get_class();
               switch (i_class){
                  case 0:
                     ob->emit_signal(SIGNAL_D(cl_6::signal_f),text);
                     break;
                  case 1:
                     ob->emit_signal(SIGNAL_D(cl_1::signal_f),text);
                     break;
                  case 2:
                     ob->emit_signal(SIGNAL_D(cl_2::signal_f),text);
                     break;
                  case 3:
                     ob->emit_signal(SIGNAL_D(cl_3::signal_f),text);
                     break;
                  case 4:
                     ob->emit_signal(SIGNAL_D(cl_4::signal_f),text);
                     break;
                  case 5:
```

```
                    ob->emit_signal(SIGNAL_D(cl_5::signal_f),text);
                    break;
               }

            }
            if (command=="SET_CONNECT"){
               cin>>coord2;
               cl_base* target = cur_ob->find_obj_bc(coord2);
               if (target!=nullptr){
                  int i_class1 = ob->get_class();
                  int i_class2 = target->get_class();
                  switch (i_class1){
                     case 0:
                           switch(i_class2){
                              case 0:
                                 ob->set_connection(SIGNAL_D(cl_6::signal_f),
     target, HANDLER_D(cl_6::handler_f));
                                 break;
                              case 1:
                                 ob->set_connection(SIGNAL_D(cl_6::signal_f),
     target, HANDLER_D(cl_1::handler_f));
                                 break;
                              case 2:
                                 ob->set_connection(SIGNAL_D(cl_6::signal_f),
     target, HANDLER_D(cl_2::handler_f));
                                 break;
                              case 3:
                                 ob->set_connection(SIGNAL_D(cl_6::signal_f),
     target, HANDLER_D(cl_3::handler_f));
                                 break;
                              case 4:
                                 ob->set_connection(SIGNAL_D(cl_6::signal_f),
     target, HANDLER_D(cl_4::handler_f));
                                 break;
                              case 5:
                                 ob->set_connection(SIGNAL_D(cl_6::signal_f),
     target, HANDLER_D(cl_5::handler_f));
                                 break;
                           }
                           break;
                     case 1:
                           switch(i_class2){

                              case 0:
                                 ob->set_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_6::handler_f));
                                 break;
                              case 1:
                                 ob->set_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_1::handler_f));
                                 break;
                              case 2:
                                 ob->set_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_2::handler_f));
                                 break;
```

```
                              case 3:
                                 ob->set_connection(SIGNAL_D(cl_1::signal_f),
target, HANDLER_D(cl_3::handler_f));
                                 break;
                              case 4:
                                 ob->set_connection(SIGNAL_D(cl_1::signal_f),
target, HANDLER_D(cl_4::handler_f));
                                 break;
                              case 5:
                                 ob->set_connection(SIGNAL_D(cl_1::signal_f),
target, HANDLER_D(cl_5::handler_f));
                                 break;
                           }
                           break;
                     case 2:
                           switch(i_class2){
                              case 0:
                                 ob->set_connection(SIGNAL_D(cl_2::signal_f),
target, HANDLER_D(cl_6::handler_f));
                                 break;
                              case 1:
                                 ob->set_connection(SIGNAL_D(cl_2::signal_f),
target, HANDLER_D(cl_1::handler_f));
                                 break;
                              case 2:
                                 ob->set_connection(SIGNAL_D(cl_2::signal_f),
target, HANDLER_D(cl_2::handler_f));
                                 break;
                              case 3:
                                 ob->set_connection(SIGNAL_D(cl_2::signal_f),
target, HANDLER_D(cl_3::handler_f));
                                 break;
                              case 4:
                                 ob->set_connection(SIGNAL_D(cl_2::signal_f),
target, HANDLER_D(cl_4::handler_f));
                                 break;
                              case 5:
                                 ob->set_connection(SIGNAL_D(cl_2::signal_f),
target, HANDLER_D(cl_5::handler_f));
                                 break;
                           }
                           break;
                     case 3:
                           switch(i_class2){
                              case 0:
                                 ob->set_connection(SIGNAL_D(cl_3::signal_f),
target, HANDLER_D(cl_6::handler_f));
                                 break;
                              case 1:
                                 ob->set_connection(SIGNAL_D(cl_3::signal_f),
target, HANDLER_D(cl_1::handler_f));
                                 break;
                              case 2:
                                 ob->set_connection(SIGNAL_D(cl_3::signal_f),
target, HANDLER_D(cl_2::handler_f));
```

```
                              break;
                           case 3:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->set_connection(SIGNAL_D(cl_3::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                              break;
                        }
                        break;
                  case 4:
                        switch(i_class2){
                           case 0:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->set_connection(SIGNAL_D(cl_4::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                              break;
                        }
                        break;
                  case 5:
                        switch(i_class2){
                           case 0:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->set_connection(SIGNAL_D(cl_5::signal_f),
```

```
      target, HANDLER_D(cl_2::handler_f));
                                 break;
                              case 3:
                                 ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                                 break;
                              case 4:
                                 ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                                 break;
                              case 5:
                                 ob->set_connection(SIGNAL_D(cl_5::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                                 break;
                           }
                           break;
                     }
                  }
                  else cout<<"Handler object "<<coord2<<" not found"<<endl;
               }
               if (command=="DELETE_CONNECT"){
                  cin>>coord2;
                  cl_base* target = cur_ob->find_obj_bc(coord2);
                  if (target!=nullptr){
                     int i_class1 = ob->get_class();
                     int i_class2 = target->get_class();
                     switch (i_class1){
                        case 0:
                              switch(i_class2){
                                 case 0:
                                    ob->delete_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_6::handler_f));
                                    break;
                                 case 1:
                                    ob->delete_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_1::handler_f));
                                    break;
                                 case 2:
                                    ob->delete_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_2::handler_f));
                                    break;
                                 case 3:
                                    ob->delete_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_3::handler_f));
                                    break;
                                 case 4:
                                    ob->delete_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_4::handler_f));
                                    break;
                                 case 5:
                                    ob->delete_connection(SIGNAL_D(cl_6::signal_f),
      target, HANDLER_D(cl_5::handler_f));
                                    break;
                              }
                              break;
```

```
                    case 1:
                          switch(i_class2){
                             case 0:
                                ob->delete_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_6::handler_f));
                                break;
                             case 1:
                                ob->delete_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_1::handler_f));
                                break;
                             case 2:
                                ob->delete_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_2::handler_f));
                                break;
                             case 3:
                                ob->delete_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_3::handler_f));
                                break;
                             case 4:
                                ob->delete_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_4::handler_f));
                                break;
                             case 5:
                                ob->delete_connection(SIGNAL_D(cl_1::signal_f),
     target, HANDLER_D(cl_5::handler_f));
                                break;
                          }
                          break;
                    case 2:
                          switch(i_class2){
                             case 0:
                                ob->delete_connection(SIGNAL_D(cl_2::signal_f),
     target, HANDLER_D(cl_6::handler_f));
                                break;
                             case 1:
                                ob->delete_connection(SIGNAL_D(cl_2::signal_f),
     target, HANDLER_D(cl_1::handler_f));
                                break;
                             case 2:
                                ob->delete_connection(SIGNAL_D(cl_2::signal_f),
     target, HANDLER_D(cl_2::handler_f));
                                break;
                             case 3:
                                ob->delete_connection(SIGNAL_D(cl_2::signal_f),
     target, HANDLER_D(cl_3::handler_f));
                                break;
                             case 4:
                                ob->delete_connection(SIGNAL_D(cl_2::signal_f),
     target, HANDLER_D(cl_4::handler_f));
                                break;
                             case 5:
                                ob->delete_connection(SIGNAL_D(cl_2::signal_f),
     target, HANDLER_D(cl_5::handler_f));
                                break;
                          }
```

```
                        break;
                  case 3:
                        switch(i_class2){
                           case 0:
                              ob->delete_connection(SIGNAL_D(cl_3::signal_f),
     target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->delete_connection(SIGNAL_D(cl_3::signal_f),
     target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->delete_connection(SIGNAL_D(cl_3::signal_f),
     target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->delete_connection(SIGNAL_D(cl_3::signal_f),
     target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->delete_connection(SIGNAL_D(cl_3::signal_f),
     target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->delete_connection(SIGNAL_D(cl_3::signal_f),
     target, HANDLER_D(cl_5::handler_f));
                              break;
                        }
                        break;
                  case 4:
                        switch(i_class2){
                           case 0:
                              ob->delete_connection(SIGNAL_D(cl_4::signal_f),
     target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->delete_connection(SIGNAL_D(cl_4::signal_f),
     target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->delete_connection(SIGNAL_D(cl_4::signal_f),
     target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->delete_connection(SIGNAL_D(cl_4::signal_f),
     target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->delete_connection(SIGNAL_D(cl_4::signal_f),
     target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->delete_connection(SIGNAL_D(cl_4::signal_f),
     target, HANDLER_D(cl_5::handler_f));
                              break;
```

```
                        }
                        break;
                  case 5:
                        switch(i_class2){
                           case 0:
                              ob->delete_connection(SIGNAL_D(cl_5::signal_f),
target, HANDLER_D(cl_6::handler_f));
                              break;
                           case 1:
                              ob->delete_connection(SIGNAL_D(cl_5::signal_f),
target, HANDLER_D(cl_1::handler_f));
                              break;
                           case 2:
                              ob->delete_connection(SIGNAL_D(cl_5::signal_f),
target, HANDLER_D(cl_2::handler_f));
                              break;
                           case 3:
                              ob->delete_connection(SIGNAL_D(cl_5::signal_f),
target, HANDLER_D(cl_3::handler_f));
                              break;
                           case 4:
                              ob->delete_connection(SIGNAL_D(cl_5::signal_f),
target, HANDLER_D(cl_4::handler_f));
                              break;
                           case 5:
                              ob->delete_connection(SIGNAL_D(cl_5::signal_f),
target, HANDLER_D(cl_5::handler_f));
                              break;
                        }
                        break;
                  }
               }
               else cout<<"Handler object "<<coord2<<" not found"<<endl;
            }
            if (command=="SET_CONDITION"){
               cin>>i_state;
               ob->can_off(i_state);
            }
         }
         else {cout<<"Object "<<coord1<<" not found"<<endl;}
      }
      return 0;
   }
   int cl_application::get_class()const{
      return 0;
   }
   /*
   void cl_application::Tree(){
      string head, sub;
      cl_base* p_head=this;
      cl_base* p_sub=nullptr;
      int i_class, i_state;
      cin>>head;
      Change_name(head);
      //Ввод иерархии
```

```
        while(true){
           cin>>head;
           if (head=="endtree"){
              break;
           }
           cin>> sub >> i_class;
           p_head=Find_global(head);

           if(p_sub!=nullptr && head==p_sub->Get_name()){
              p_head=p_sub;
           }
           if(head==p_head->Get_name() && p_head->Get_ptr(sub)==nullptr){
              p_sub=new cl_1 (p_head,sub);
           }

           if (p_head&&!p_head->Get_ptr(sub)){
              switch (i_class)
              {
                 case 1:
                    p_sub=new cl_1(p_head,sub);
                    break;
                 case 2:
                    p_sub=new cl_2(p_head,sub);
                    break;
                 case 3:
                    p_sub=new cl_3(p_head,sub);
                    break;
                 case 4:
                    p_sub=new cl_4(p_head,sub);
                    break;
                 case 5:
                    p_sub=new cl_5(p_head,sub);
                    break;
                 default:
                    break;
              }
           }
        }
        //Установка состояний объекта
        while(cin>>head>>i_state){
           p_head = Find_global(head);
           if(p_head!=nullptr) {p_head->can_off(i_state);}
        }
        //cout<<"Статусы заданы\n";
     }
     */
     /*
     void cl_application::Tree_1(){
        string head, sub;
        cl_base* p_head=this;
        cl_base* p_sub=nullptr;
        cin>>head;
        Change_name(head);
        while(true){
           cin>>head>>sub;
```

```
            if (head==sub){
               break;
            }
            if(p_sub!=nullptr && head==p_sub->Get_name()){
               p_head=p_sub;
            }
            if(head==p_head->Get_name() && p_head->Get_ptr(sub)==nullptr){
               p_sub=new cl_1 (p_head,sub);
            }
         }
      }
      */
      /*
      int cl_application::Start(){
         cl_base* p_root=this;
         while(p_root->Get_baseptr()){
            p_root=p_root->Get_baseptr();
         }
         if(p_root->Get_name()=="EEERROR") return 1;
         cout<<"Object tree"<<endl;
         Out_fc();
         //cout<<endl<<"The tree of objects and their readiness"<<endl;
         //Out();
         cl_base* cur_ob=p_root;
         //Команды
         while(true){
            string command, coord;
            cin>>command;
            if(command=="END") break;
            cin>>coord;
            if (command=="SET"){
               if(cur_ob->find_obj_bc(coord)){
                  cur_ob=cur_ob->find_obj_bc(coord);
                  cout<<endl<<"Object is set: "<<cur_ob->Get_name();
               }
               else cout<<endl<<"The object was not found in specified coordinate:
      "<<coord;
            }
            if (command=="FIND"){
               cout<<endl<<coord<<"     ";
               if(cur_ob->find_obj_bc(coord)){
                  cout<<"Object name: "<<cur_ob->find_obj_bc(coord)->Get_name();
               }
               else cout<<"Object is not found";
            }
            if (command=="MOVE"){
               cl_base* ob = cur_ob->find_obj_bc(coord);
               if(!ob) cout<<endl<<coord<<"     Head object is not found";
               else  if(ob->Get_baseptr()  !=  cur_ob->Get_baseptr()  &&  cur_ob-
      >Find_current(ob->Get_name())) cout<<endl<<coord<<"      Redefining the head
      object failed";
               else   if   (ob->Get_ptr(cur_ob->Get_name()))   cout<<endl<<coord<<"
      Dubbing the names of subordinate objects";
               else      if(cur_ob->Move_head(ob)==false)      cout<<endl<<coord<<"
      Redefining the head object failed";
```

```
            }
            if (command=="DELETE"){
               if(cur_ob->Get_ptr(coord)){
                  vector <string> text;
                  string text1;
                  cl_base* h = cur_ob->Get_ptr(coord)->Get_baseptr();
                  cur_ob->Del_obj(coord);
                  while(h){
                     if(h->Get_name()!=p_root->Get_name()){
                        text.push_back(h->Get_name());
                     }
                     h=h->Get_baseptr();
                  }
                  for (int i = text.size()-1;i>=0;i--) text1+="/"+text[i];
                  cout<<endl<<"The object "<<text1+"/"+coord<<" has been deleted";
               }
            }
         }
         cout<<endl<<"Current object hierarchy tree"<<endl;
         Out_fc();
         return 0;
      }
      */
```

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 Tree();
         //void Tree_1();
         int Start();
         virtual int get_class() const override;
      };
      #endif
```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
#include "cl_base.h"
#include <stack>
using namespace std;
cl_base::cl_base(cl_base* p_head_object, string s_object_name){
   this -> s_object_name = s_object_name;
   this -> p_head_object = p_head_object;
   if (p_head_object!=nullptr){
      p_head_object->subordinate_objects.push_back(this);
   }
}
string cl_base::Get_name(){
   return s_object_name;
}//Получение имени объекта

//int cl_base::get_class()const{return 1;}

bool cl_base::Change_name(string name){
   if(Get_baseptr()!=nullptr){
      for (int i=0; i<p_head_object->subordinate_objects.size();i++){
         if (p_head_object->subordinate_objects[i]->Get_name()==name){
            return false;
         }
      }
   }
   this->s_object_name=name;
   return true;
}//Поменять имя объекта
cl_base* cl_base::Get_ptr(string s_object_name){
   for (int i=0; i<subordinate_objects.size();i++){
      if (subordinate_objects[i]->Get_name()==s_object_name){
         return subordinate_objects[i];
      }
   }
   return nullptr;
}//Получение указателя на объект
cl_base* cl_base::Get_baseptr(){
   return p_head_object;
}//Получение головного указателя
//Статус
/*
int cl_base::Get_status(){
   return status;
}
*/
/*Сет статус
void cl_base::Set_status(int i){
   this->status=i;
}
*/
//Статус
```

```
void cl_base::can_off(int status1){
   if (status1==0){
      status=0;
      for (auto sub:subordinate_objects){
         sub->can_off(0);
      }
      //cout<<"Поставил фальш\n";
   }
   if (Get_baseptr()&&(Get_baseptr()->status==0)){
      return;
   }
   else if (status1!=0){
      status=1;
   }
}

void cl_base::Set_all_ready(){
   this->can_off(1);
   for (auto sub:this->subordinate_objects){
      sub->Set_all_ready();
   }
}

void cl_base::Out_fc(int probel){
   cout<<Get_name();
   if(subordinate_objects.size()!=0){
      for (auto sub:subordinate_objects){
         cout<<endl;
         for(int i =0; i<probel;i++) cout<<" ";
         sub->Out_fc(probel+4);
      }
   }
}//Вывод дерева
/*
void cl_base::Out(int probel){
   string ready="is ready";
   string nready="is not ready";
   cout<<"    "<<this->Get_name();
   if(this->Get_status()!=0){
      cout<<ready<<endl;
   }
   else{cout<<nready<<endl;}
}*/
//Вывод дерева
void cl_base::Out(int probel){
   cout<< Get_name();
   if (status==0)cout<<" is not ready";
   else if (status!=0){cout<<" is ready";}
   if (subordinate_objects.size()!=0){
      for(int i=0; i<subordinate_objects.size();i++){
         cout<<endl;
         for(int i =0; i<probel;i++) cout<<" ";
         subordinate_objects[i]->Out(probel+4);
      }
   }
```

```
}//Вывод дерева

//Поиск не на основе очереди, был на ней, но не судьба
cl_base* cl_base::Find_current(string name){
   if (Get_name() == name)
   {
      //cout<<"Нашел\n";
      return this;
   }
   for (auto sub : subordinate_objects)
   {
      cl_base* p_sub = sub->Find_current(name);
      if (p_sub)
      {
         return p_sub;
      }
   }
   return nullptr;
}
/*
int cl_base::Count(string name){
   int count=0;
   if (Get_name()==name) count++;
   for(auto sub:subordinate_objects)count+=sub->Count(name);
   return count;
}
*/

cl_base* cl_base::Find_global(string name){
   cl_base* p_root=this;
   while(p_root->Get_baseptr()){
      p_root=p_root->Get_baseptr();
   }
   cl_base* p_found = p_root;
   int count =0;
   stack <cl_base*> stack;
   stack.push(p_found);
   while(!stack.empty()){
      cl_base * cur = stack.top();
      stack.pop();
      if (cur->Get_name()==name) count++;
      for (auto sub:cur->subordinate_objects) stack.push(sub);
   }
   if (count!=1) return nullptr;
   //cout<<"Нащел родича \n";
   //cout<<p_found->Find_current(name)->Get_name();
   return p_found->Find_current(name);
}
/*
cl_base* cl_base::get_root(){
   cl_base* p_root=this;
   while(p_root->Get_baseptr()){
      p_root=p_root->Get_baseptr();
   }
   return p_root;
```

```
}
*/
//Метод поиска объекта по координате
cl_base* cl_base::find_obj_bc(string coord){
   string s_name;
   int index;
   cl_base* ob = nullptr;
   cl_base* p_root=this;
   while(p_root->Get_baseptr()){
      p_root=p_root->Get_baseptr();
   }
   //головной объект
   if(coord=="/"){
      return p_root;
   }
   //текущий
   if(coord=="."){
      return this;
   }
   //Поиск от корня по имени
   if(coord[0]=='/' && coord[1]=='/'){
      s_name=coord.substr(2);
      return this->Find_global(s_name);
   }
   //Поиск от текущего по имени
   if(coord[0]=='.'){
      s_name=coord.substr(1);
      return this->Find_current(s_name);
   }
   index=coord.find("/",1);
   //Абсолютный путь
   if(coord[0]=='/'){
      if (index!=-1){
         s_name= coord.substr(1,index-1);
         ob=p_root->Get_ptr(s_name);
         if(ob!=nullptr){
            return ob->find_obj_bc(coord.substr(index+1));
         }
         else return ob;
      }
      else {
         //cout<<"\n ИНДЕКС: "<<index<<endl;
         s_name= coord.substr(1);
         return p_root->Get_ptr(s_name);
      }
   }
   //Относительный путь
   if(coord[0]!='/'){
      if (index!=-1){
         s_name= coord.substr(0,index);
         ob=this->Get_ptr(s_name);
         if(ob!=nullptr){
            return ob->find_obj_bc(coord.substr(index+1));
         }
         else return ob;
```

```
            }
            else{
               s_name=coord;
               return this->Get_ptr(s_name);
            }
         }
         return nullptr;
      }
      //Метод переопределения головного объекта
      bool cl_base::Move_head(cl_base* h){
         if (this->Get_baseptr()){
            /*
            for (auto sub:h->subordinate_objects){
               if (sub==h){
                  cout<<"    Dubbing the names of subordinate objects";
                  return false;
               }
            }
            */
            for (int i=0;i<this->Get_baseptr()->subordinate_objects.size();i++){
               if(this->Get_baseptr()->subordinate_objects[i]==this){
                  Get_baseptr()->subordinate_objects.erase(this->Get_baseptr()-
      >subordinate_objects.begin()+i);
                  break;
               }
            }
            this->p_head_object=h;
            h->subordinate_objects.push_back(this);
            cout<<endl<<"New head object: "<<h->Get_name();
            return true;
         }
         return false;
      }
      //Метод удаления подчиненного объекта
      void cl_base::Del_obj(string name){
         cl_base* p = Get_ptr(name);
         /*
         if (p->subordinate_objects.size()!=0){
            return false;
         }
         */
         int i = 0;
         for(auto sub:subordinate_objects){
            if (sub==p){
               subordinate_objects.erase(subordinate_objects.begin()+i);
               delete p;
               return;
            }
            i++;
         }
         /*
         string text;
         cl_base* h = p->Get_baseptr();
         while(h){
            text+="/"+h->Get_name();
```

```
            h=h->Get_baseptr();
         }
         cout<<endl<<"The object "<< text+"/"+name<<" has been deleted";
         */
      }
      //Абсолютный путь до объекта
      string cl_base::Absolute(){
         cl_base* p_root =this;
         string coord="";
         while(p_root->Get_baseptr()){
            p_root=p_root->Get_baseptr();
         }
         if (p_root == this) return "/";
         vector <string> text;
         string text1;
         coord+=this->Get_name();
         cl_base* h = this->Get_baseptr();
         while(h){
            if(h->Get_name()!=p_root->Get_name()){
               text.push_back(h->Get_name());

            }
            h=h->Get_baseptr();
         }
         for (int i = text.size()-1;i>=0;i--) text1+="/"+text[i];
         if(coord!=p_root->Get_name()) text1+="/"+coord;
         return text1;
      }
      //Метод создания связи
      void    cl_base::set_connection(TYPE_SIGNAL    signal,    cl_base*    target,
      TYPE_HANDLER handler){
         o_sh  * p_value;
         //-------------------------------------------------------
         // Цикл для исключения повторного установления связи
         for (int i = 0; i < connects.size ( );i++)
         {
            if (connects [ i ] -> signal  == signal &&
               connects [ i ] -> target == target &&
               connects [ i ] -> handler == handler )return;
         }
         p_value  =    new  o_sh  ();//  создание  объекта  структуры  для  хранения
      информации о новой связи
         p_value -> signal  = signal;
         p_value -> target = target;
         p_value -> handler = handler;
         connects.push_back (p_value);// добавление новой связи
      }
      //Метод удаления связи
      void   cl_base::delete_connection(TYPE_SIGNAL   signal,   cl_base*   target,
      TYPE_HANDLER handler){
         vector <o_sh*>::iterator it;
         for (it = connects.begin(); it<=connects.end(); it++){
            for (int i = 0; i < connects.size ( );i++){
               if ((*it)-> signal  == signal &&
                  (*it)-> target == target &&
```

```
            (*it)-> handler == handler ){
            connects.erase(it);
         }
      }
   }
}
//Метод подачи сигнала
void cl_base::emit_signal(TYPE_SIGNAL signal, string msg){
   if (this->status==0) return;
   (this ->*signal)(msg);
   //cout<< "Signal from "<<this->Absolute()<<endl;
   for (auto con:connects){
      if(con->signal==signal){
         cl_base* target=con->target;
         if (target->status!=0){
            TYPE_HANDLER handler=con->handler;
            (target->*handler)(msg);
            //cout<<   "Signal   to   "<<target->Absolute()<<"         Text:
"<<msg<<endl;
         }
      }
   }

}


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

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 <queue>
class cl_base;
using namespace std;
#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 & msg);
typedef void ( cl_base :: * TYPE_HANDLER ) ( string );
struct o_sh                // Структура задания одной связи
{
```

```
   TYPE_SIGNAL  signal;    // Указатель на метод сигнала
   cl_base* target;   // Указатель на целевой объект
   TYPE_HANDLER handler;   // Указатель на метод обработчика

};

class cl_base{
   private:
   int status=0;
   string s_object_name;
   cl_base * p_head_object;
   vector <cl_base*> subordinate_objects;
   vector <o_sh*> connects;
   public:
   cl_base(cl_base* p_head_object,string s_object_name="Base_object");
   bool Change_name(string name);
   string Get_name();
   cl_base* Get_baseptr();

   void Out(int probel=4);
   void Out_fc(int probel=4);

   void can_off(int status);

   //void Out1();
   //int Get_status();
   //void Set_status(int i);
   //int Count(string name);
   //cl_base* get_root();

   bool Move_head(cl_base*);
   void Del_obj(string name);

   string Absolute();
   virtual int get_class()const =0;

   cl_base* Find_current(string name);
   cl_base* Find_global(string name);
   cl_base* Get_ptr(string s_object_name);
   cl_base* find_obj_bc(string coord);
   //cl_base* Search(string name);

   void Set_all_ready();

   void  set_connection(TYPE_SIGNAL  signal,  cl_base*  target,  TYPE_HANDLER
handler);
   void delete_connection(TYPE_SIGNAL signal, cl_base* target, TYPE_HANDLER
handler);
   void emit_signal(TYPE_SIGNAL signal, string msg);
   ~cl_base();
};
#endif
```

5.17 Файл main.cpp

Листинг 17 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "cl_base.h"
#include "cl_application.h"
int main()
{
   cl_application  ob_cl_application ( nullptr );
   ob_cl_application.Tree();
   return ob_cl_application.Start();
   return 1;
   /*
   string a,b;
   int c;
   cin>>a;
   cin>>b>>c;
   cout<<a<<" "<<b<<" "<<c<<endl;
   cout<<a[0]<<endl;
   */
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 29.

Таблица 29 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
k endtree end_of_connections END	Object tree k	Object tree k

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).