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 getline для считывание строки из потока ввода;
- функция typedef для оператор для определения нового типа данных;
- функция #define для параметризированное макроопределение препроцессора.

Класс base_class:

- свойства/поля:
    - поле указатели на данные типа connect:
        - наименование — connects;
        - тип — vector<connect*>;
        - модификатор доступа — private;
    - поле номер принадлежности объекта к классу:
        - наименование — number;
        - тип — const int;
        - модификатор доступа — public;
- функционал:
    - метод set_connect — установление связи между сигналом текущего объекта и обработчиком целевого объекта;
    - метод del_connect — удаление (разрыв) связи между сигналом текущего объекта и обработчиком целевого объекта;
    - метод emit_signal — выдача сигнала от текущего объекта с передачей строковой переменной;
    - метод get_path — получение абсолютного пути объекта;
    - метод base_class — параметризированыйй конструктор;
    - метод set_status — установка готовности объекта;

- метод set_status_on_branch — установки готовности объектов на ветке.

Класс application:

- функционал:
  - метод application — параметризированный конструктор;
  - метод signal_f — запуск сигнала;
  - метод handler_f — запуск обработчика;
  - метод build_tree — постороение дерева иерархии;
  - метод exec_app — запуск программы.

Класс cl2:

- функционал:
  - метод cl2 — параметризированный конструктор;
  - метод signal_f — запуск сигнала;
  - метод handler_f — запуск обработчика.

Класс cl3:

- функционал:
  - метод cl3 — параметризированный конструктор;
  - метод signal_f — запуск сигнала;
  - метод handler_f — запуск обработчика.

Класс cl4:

- функционал:
  - метод cl4 — параметризированный конструктор;
  - метод signal_f — запуск сигнала;
  - метод handler_f — запуск обработчика.

Класс cl5:

- функционал:
  - метод cl5 — параметризированный конструктор;

- метод signal_f — запуск сигнала;
- метод handler_f — запуск обработчика.

Класс cl6:

- функционал:
  - метод cl6 — параметризированный конструктор;
  - метод signal_f — запуск сигнала;
  - метод handler_f — запуск обработчика.

Класс connect:

- свойства/поля:
  - поле указатель на метод сигнала:
    - наименование — signal;
    - тип — TYPE_SIGNAL;
    - модификатор доступа — public;
  - поле указатель на целевой объект:
    - наименование — object;
    - тип — base_class*;
    - модификатор доступа — public;
  - поле указатель на метод обработчика:
    - наименование — handler;
    - тип — TYPE_HANDLER;
    - модификатор доступа — public;
- функционал:
  - метод connect — параметризированный конструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	base_class			базовый класс программы	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
		application	public		2
		cl2	public		3
		cl3	public		4
		cl4	public		5
		cl5	public		6
		cl6	public		7
2	application			класс приложения	
3	cl2			второй производный класс	
4	cl3			третий производный класс	
5	cl4			четвертый производный класс	
6	cl5			пятый производный класс	
7	cl6			шестой производный класс	
8	connect			структура для хранения информации о связи объектов	

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: основная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта tree класса application с помощью параметризированного конструктора с параметром nullptr	2
2		вызов метода build_tree объекта tree класса application	3
3		возврат результата работы метода exec_app объекта tree класса application	∅

3.2 Алгоритм метода set_connect класса base_class

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

Параметры: TYPE_SIGNAL signal - указатель на метод сигнала текущего объекта, base_class* object - указатель на целевой объект, TYPE_HANDLER handler - указатель на метод обработчика целевого объекта.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода set_connect класса base_class

№	Предикат	Действия	№ перехода
1		инициализация переменной i типа int значением 0	2
2	значение переменной i меньше размера списка connects?		3
			4
3	значение signal элемента списка connects с индексом i равен значению параметра signal И значение object элемента списка connects с индексом i равен значению параметра object И значение handler элемента списка connects с индексом i равен значению параметра handler?	возврат	∅
		i++	2
4		добавить в конец списка connects новую структуру connect с параметрами signal, object, handler	∅

3.3 Алгоритм метода del_connect класса base_class

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

Параметры: TYPE_SIGNAL signal - указатель на метод сигнала текущего объекта, base_class* object - указатель на целевой объект, TYPE_HANDLER handler - указатель на метод обработчика целевого объекта.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода del_connect класса base_class

№	Предикат	Действия	№ перехода
1		инициализация переменной i типа int значением 0	2
2	значение переменной i меньше размера списка connects?		3
			∅
3	значение signal элемента списка connects с индексом i равен значению параметра signal И значение object элемента списка connects с индексом i равен значению параметра object И значение handler элемента списка connects с индексом i равен значению параметра handler?	удаление из списка connects структуры с индексом i	4
		i++	2
4		возврат	∅

3.4 Алгоритм метода get_path класса base_class

Функционал: получение абсолютного пути объекта.

Параметры: нет.

Возвращаемое значение: string - абсолютный путь к объекту.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_path класса base_class

№	Предикат	Действия	№ перехода
1		инициализация указателя current класса base_class* на родителя данного объекта	2
2	current не существует?	возврат "/"	∅
		инициализация строковой переменной значением '/' + результат метода get_name	3
3	результат метода get_parent от указателя current существует?	присваивание переменной path значение '/' + результат вызова метода get_name от объекта current + значение переменной path	4
		возврат значения переменнной path	∅
4		вызов метода get_parent объекта current и присваивание ему результата этого метода	3

3.5 Алгоритм метода emit_signal класса base_class

Функционал: выдача сигнала от текущего объекта с передачей строковой переменной.

Параметры: TYPE_SIGNAL signal - указатель на метод сигнала текущего объекта, string& command - строка для вывода.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода emit_signal класса base_class

№	Предикат	Действия	№ перехода
1	значение переменной status равно 0?	возврат	∅
		объявление указателя handler типа TYPE_HANDLER и указателя object типа base_class*	2

№	Предикат	Действия	№ перехода
2		вызов метода сигнала от текущего объекта через указатель signal с аргументом command	3
3		инициализация переменной i типа int значением 0	4
4	значение переменной i меньше размера списка connects?		5
			∅
5	поле signal элемента списка connects с индексом i равно значению параметра signal?	присваивание указателю handler значение поля handler структуры списка connects с индексом i	6
			8
6		присваивание указателю object значение поля object структуры списка connects с индексом i	7
7	поле status указателя object не равно 0?	вызов метода обработчика текущего объекта через указатель handler с аргументом command	8
			8
8		i++	4

3.6 Алгоритм конструктора класса base_class

Функционал: параметризированыний конструктор.

Параметры: base_class* parent - указатель на родительский объект, string name - имя объекта, int number = 0 - номер класса объекта.

Алгоритм конструктора представлен в таблице 7.

Таблица 7 – Алгоритм конструктора класса base_class

№	Предикат	Действия	№ перехода
1		присваивание полю parent объекта значение параметра parent	2

№	Предикат	Действия	№ перехода
2		присваивание полю name объекта значение параметра name	3
3	parent существует?	добавление указателя на данный объект в конец списка children объекта parent	∅
			∅

3.7 Алгоритм метода set_status класса base_class

Функционал: установка готовности объекта.

Параметры: int status.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода set_status класса base_class

№	Предикат	Действия	№ перехода
1	status равен 0?		2
			4
2		присваивание полю status данного объекта значение 0	3
3	проход по подчиненным объекта	вызов метода set_status с параметром 0 от указателя child	3
			∅
4	результат метода get_parent существует И поле status результата этого метода равно 0?	присваивание полю status данного объекта значение 0	∅
		присваивание полю status данного объекта значение параметра status	∅

3.8 Алгоритм метода set_status_on_branch класса base_class

Функционал: установки готовности объектов на ветке.

Параметры: int status - готовность объектов.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода set_status_on_branch класса base_class

№	Предикат	Действия	№ перехода
1		вызов метода set_status с аргументом status	2
2	status не равен 0?	возврат	∅
			3
3	проход по всем подчиненным объекта	вызов метода set_status_on_branch с аргументом status от указателя child	3
			∅

3.9 Алгоритм конструктора класса application

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с параметрами root, name, 1.

Параметры: base_class* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса application

№	Предикат	Действия	№ перехода
1			∅

3.10 Алгоритм метода signal_f класса application

Функционал: запуск сигнала.

Параметры: string& command - строка для вывода в handler_f.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода signal_f класса application

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal from " + результат метода get_path + "\n"	2
2		добавление к переменной command строки " (class: 1)"	∅

3.11 Алгоритм метода handler_f класса application

Функционал: запуск обработчика.

Параметры: string command - сообщение для вывода.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода handler_f класса application

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal to " + результат метода get_path + " Text: " + значенеи переменной command + "\n"	∅

3.12 Алгоритм метода build_tree класса application

Функционал: постороение дерева иерархии.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода build_tree класса application

№	Предикат	Действия	№ перехода
1		создание целочисленной переменной number и строковых переменных parent_name, child_name, parent_coords	2
2		ввод значения переменной parent_name с клавиатуры	3
3		вызов метода set_name с параметром parent_name	4
4		инициализация указателя parent указанием на данный объект	5
5	ввод значения переменной parent_coords И её значение не равно строке "endtree"?	ввод значений переменных child_name и number с клавиатуры	6
		создание строковых переменных emit_path и object_path	13
6		присваивание указателю parent указание на результат метода find объекта parent с параметром parent_coords	7
7	parent не существует?	вывод на экран сообщения "Object tree\n"	8
			11
8		вызов метода print_branch	9
9		вывод на экран сообщения "The head object " + значение переменной parent_coords + " is not found\n"	10
10		завершение работы программы	∅
11	результат метода get_child указателя parent с параметром child_name существует?	вывод на экран значение переменной parent_coords и сообщения " Dubbing the names of subordinate objects\n"	5

№	Предикат	Действия	№ перехода
			12
12	значение переменной number равно 2?	создание нового объекта класса cl2 с параметрами parent и child_name	5
	значение переменной number равно 3?	создание нового объекта класса cl3 с параметрами parent и child_name	5
	значение переменной number равно 4?	создание нового объекта класса cl4 с параметрами parent и child_name	5
	значение переменной number равно 5?	создание нового объекта класса cl5 с параметрами parent и child_name	5
	значение переменной number равно 6?	создание нового объекта класса cl6 с параметрами parent и child_name	5
			5
13		создание массива signals из элементов типа TYPE_SIGNAL, в который входят методы-сигналы каждого класса	14
14		создание массива handlers из элементов типа TYPE_HANDLER, в который входят методы-обработчики каждого класса	15
15	ввод значения переменной emit_path И её значение не равно строке "end_of_condition"?	ввод значения переменной object_path	16
			∅
16		инициализация указателя emit типа base_class* результатом метода find с параметром emit_path	17
17		инициализация указателя object типа base_class* результатом метода find с параметром object_path	18
18	emit существует И object существует?	вызов метода set_connect от указателя emit с параметрами signals[emit->number-1], object,	15

№	Предикат	Действия	№ перехода
		handlers[object->number-1]	
			15

3.13 Алгоритм метода exec_app класса application

Функционал: запуск программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Object tree\n"	2
2		вызов метода print_branch	3
3		вызов метода set_status_on_branch с параметром 1	4
4		создание строковой переменной command	5
5		создание массива signals из элементов типа TYPE_SIGNAL, в который входят методы-сигналы каждого класса	6
6		создание массива handlers из элементов типа TYPE_HANDLER, в который входят методы-обработчики каждого класса	7
7	ввод значения переменной command И её значение не равно строке "END"?	создание строковой переменной object_path	8
		возврат 0	∅
8	значение переменной command равно "EMIT"?	ввод значения переменной object_path	9
			13

№	Предикат	Действия	№ перехода
9		создание строковой переменной message	10
10		ввод значения переменной message с клавиатуры	11
11		инициализация указателя object типа base_class* значением результата метода find с координатами object_path	12
12	object существует?	вызов метода emit_signal от указателя object с параметрами signals[object->number-1], message	7
			7
13	значение переменной command равно "SET_CONNECT"?	создание строковой переменной target_path	14
			18
14		ввод значений переменных object_path и target_path с клавиатуры	15
15		инициализация указателя object типа base_class* значением результата метода find с параметром object_path	16
16		инициализация указателя target значением результата метода find с параметром target_path	17
17	object существует И target существует?	вызов метода set_connect от указателя object с параметрами signals[object->number-1], target, handlers[target->number-1]	7
			7
18	значение переменной command равно "DELETE_CONNECT"?	создание строковой переменной target_path	19
			23
19		ввод значений переменных object_path и target_path	20

№	Предикат	Действия	№ перехода
20		инициализация указателя object типа base_class* значением результата метода find с параметром object_path	21
21		инициализация указателя target значением результата метода find с параметром target_path	22
22	object существует И target существует?	вызов метода del_connect от указателя object с параметрами signals[object->number-1], target, handlers[target->number-1]	7
			7
23	значение переменной command равно "SET_CONDITION"?	создание целочисленной переменной status	24
			7
24		ввод значений переменных object_path и status	25
25		инициализация указателя object типа base_class* значением результата метода find с параметром object_path	26
26	object существует?	вызов метода set_status от указателя object с параметром status	7
			7

3.14 Алгоритм конструктора класса cl2

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с параметрами root, name, 2.

Параметры: base_class* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 15.

Таблица 15 – Алгоритм конструктора класса cl2

№	Предикат	Действия	№ перехода
1			∅

3.15 Алгоритм метода signal_f класса cl2

Функционал: запуск сигнала.

Параметры: string& command - строка для вывода в handler_f.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода signal_f класса cl2

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal from " + результат метода get_path + "\n"	2
2		добавление к переменной command строки " (class: 2)"	∅

3.16 Алгоритм метода handler_f класса cl2

Функционал: запуск обработчика.

Параметры: string command - сообщение для вывода.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода handler_f класса cl2

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal to " + результат метода get_path + " Text: " + значение переменной command + "\n"	∅

3.17 Алгоритм конструктора класса cl3

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с параметрами root, name, 3.

Параметры: base_class* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 18.

Таблица 18 – Алгоритм конструктора класса cl3

№	Предикат	Действия	№ перехода
1			∅

3.18 Алгоритм метода signal_f класса cl3

Функционал: запуск сигнала.

Параметры: string& command - строка для вывода в handler_f.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода signal_f класса cl3

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal from " + результат метода get_path + "\n"	2
2		добавление к переменной command строки " (class: 3)"	∅

3.19 Алгоритм метода handler_f класса cl3

Функционал: запуск обработчика.

Параметры: string command - сообщение для вывода.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода handler_f класса cl3

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal to " + результат метода get_path + " Text: " + значение переменной command + "\n"	∅

3.20 Алгоритм конструктора класса cl4

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с параметрами root, name, 4.

Параметры: base_class* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 21.

Таблица 21 – Алгоритм конструктора класса cl4

№	Предикат	Действия	№ перехода
1			∅

3.21 Алгоритм метода signal_f класса cl4

Функционал: запуск сигнала.

Параметры: string& command - строка для вывода в handler_f.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода signal_f класса cl4

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal from " + результат метода get_path + "\n"	2

№	Предикат	Действия	№ перехода
2		добавление к переменной command строки " (class: 4)"	∅

3.22 Алгоритм метода handler_f класса cl4

Функционал: запуск обработчика.

Параметры: string command - сообщение для вывода.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода handler_f класса cl4

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal to " + результат метода get_path + " Text: " + значение переменной command + "\n"	∅

3.23 Алгоритм конструктора класса cl5

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с параметрами root, name, 5.

Параметры: base_class* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 24.

Таблица 24 – Алгоритм конструктора класса cl5

№	Предикат	Действия	№ перехода
1			∅

3.24 Алгоритм метода signal_f класса cl5

Функционал: запуск сигнала.

Параметры: string& command - строка для вывода в handler_f.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода signal_f класса cl5

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal from " + результат метода get_path + "\n"	2
2		добавление к переменной command строки " (class: 5)"	∅

3.25 Алгоритм метода handler_f класса cl5

Функционал: запуск обработчика.

Параметры: string command - сообщение для вывода.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода handler_f класса cl5

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal to " + результат метода get_path + " Text: " + значение переменной command + "\n"	∅

3.26 Алгоритм конструктора класса cl6

Функционал: параметризированный конструктор, вызывающий конструктор базового класса с параметрами root, name, 6.

Параметры: base_class* parent - указатель на родительский объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 27.

Таблица 27 – Алгоритм конструктора класса cl6

№	Предикат	Действия	№ перехода
1			∅

3.27 Алгоритм метода signal_f класса cl6

Функционал: запуск сигнала.

Параметры: string& command - строка для вывода в handler_f.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода signal_f класса cl6

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal from " + результат метода get_path + "\n"	2
2		добавление к переменной command строки " (class: 6)"	∅

3.28 Алгоритм метода handler_f класса cl6

Функционал: запуск обработчика.

Параметры: string command - сообщение для вывода.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 29.

Таблица 29 – Алгоритм метода handler_f класса cl6

№	Предикат	Действия	№ перехода
1		вывод на экран сообщения "Signal to " + результат метода get_path + " Text: " + значение переменной command + "\n"	∅

3.29 Алгоритм конструктора класса connect

Функционал: параметризированный конструктор.

Параметры: TYPE_SIGNAL signal - указатель на метод сигнала текущего объекта, base_class* object - указатель на целевой объект, TYPE_HANDLER handler - указатель на метод обработчика целевого объекта.

Алгоритм конструктора представлен в таблице 30.

Таблица 30 – Алгоритм конструктора класса connect

№	Предикат	Действия	№ перехода
1		присваивание полю signal объекта значение параметра signal	2
2		присваивание полю object объекта значение параметра object	3
3		присваивание полю handler объекта значение параметра handler	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

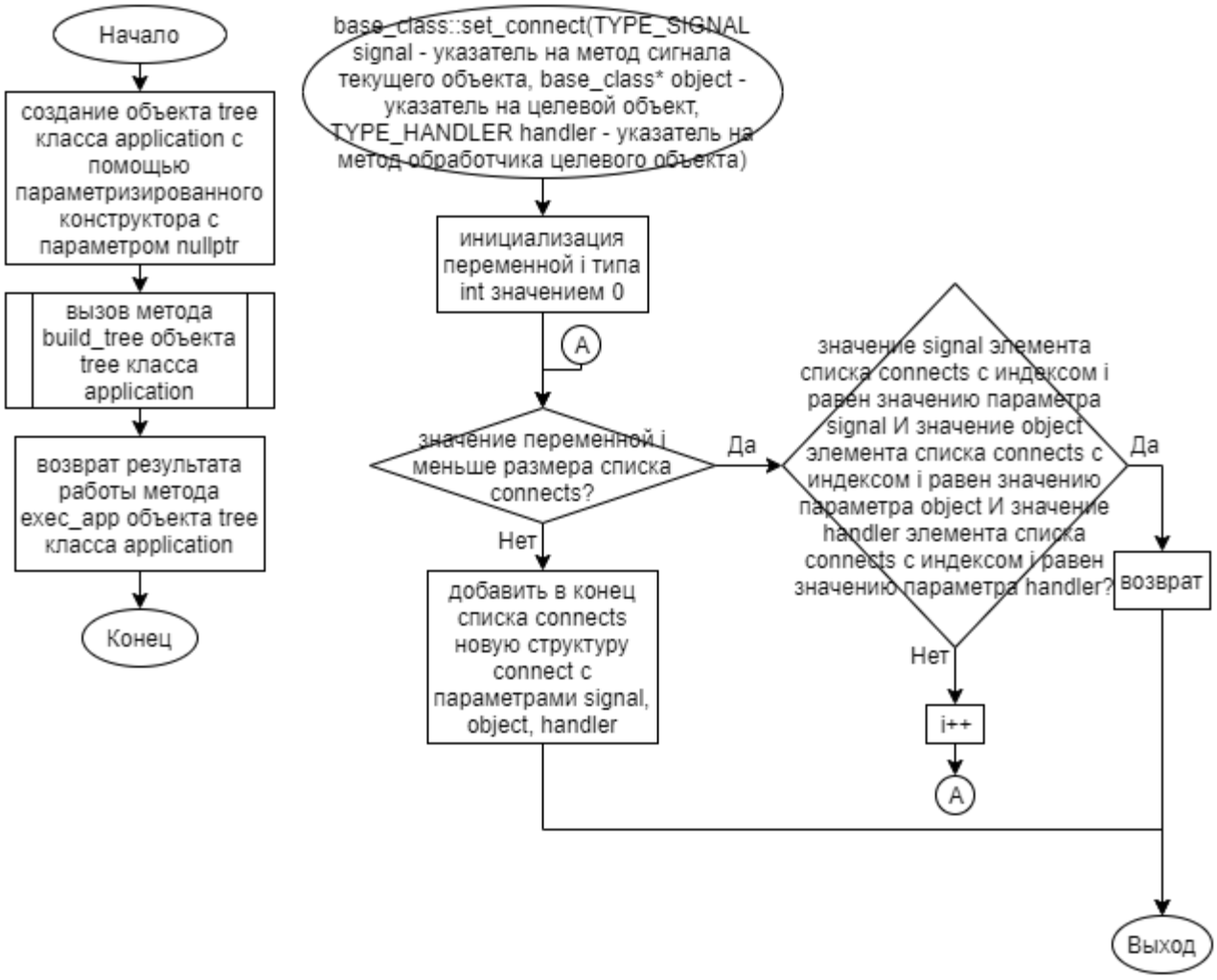

Рисунок 1 – Блок-схема алгоритма

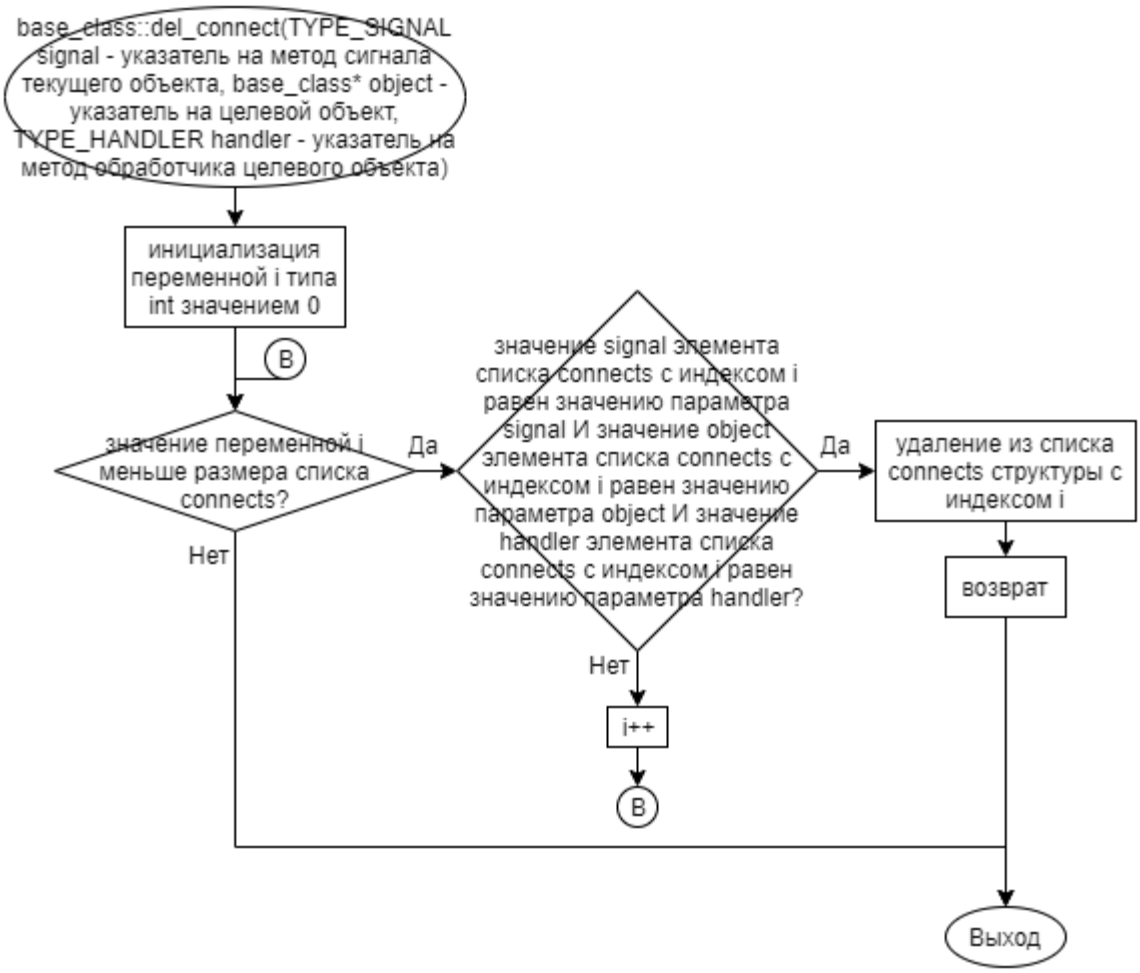

Рисунок 2 – Блок-схема алгоритма

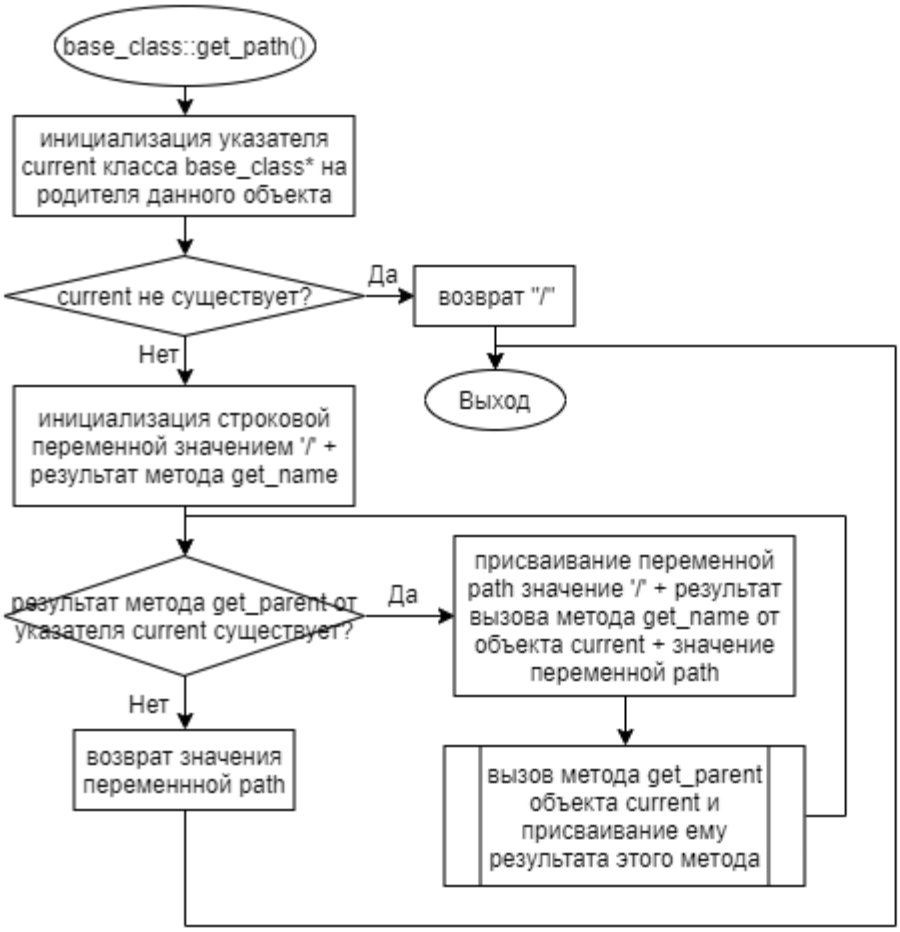

Рисунок 3 – Блок-схема алгоритма

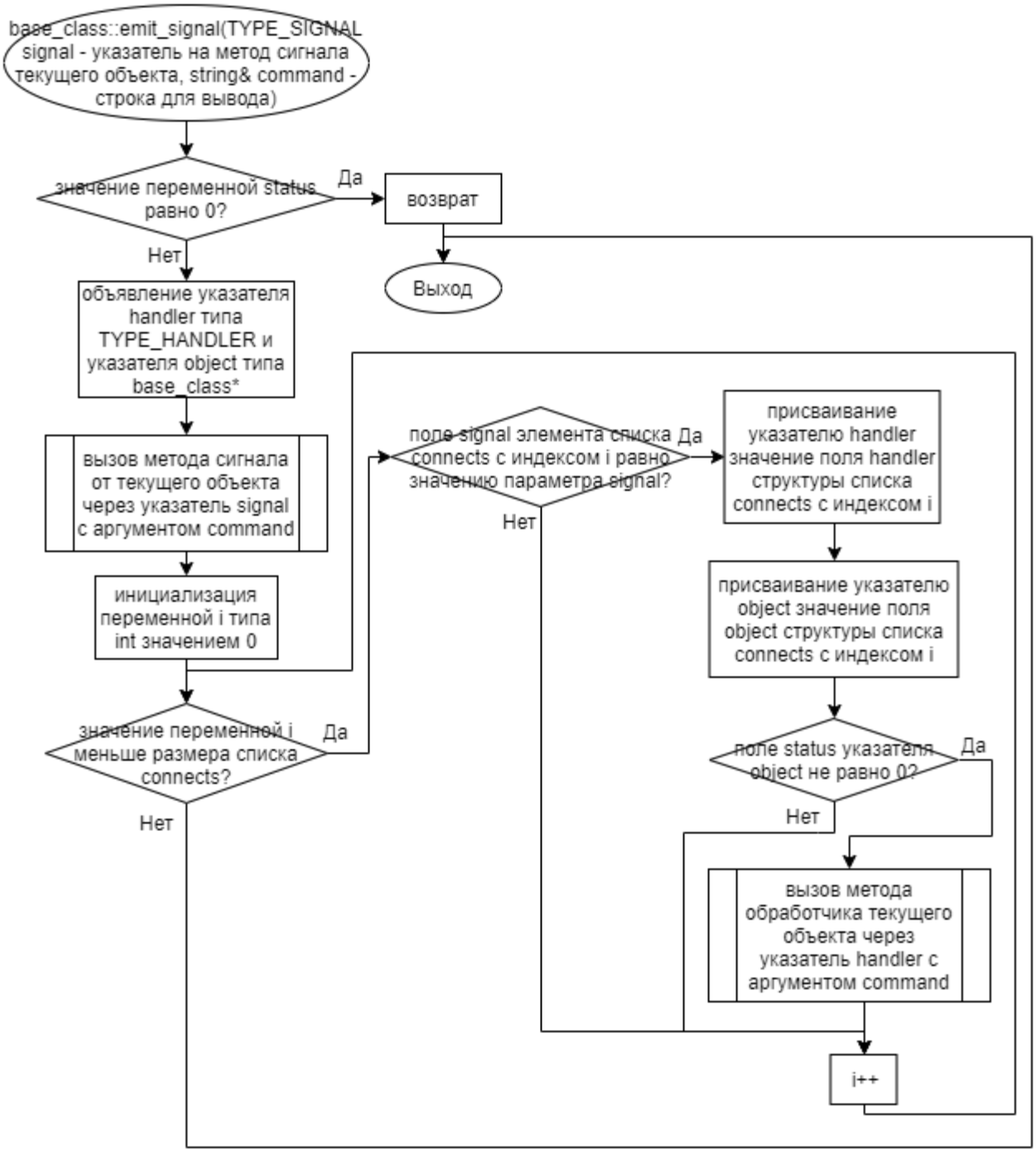

Рисунок 4 – Блок-схема алгоритма

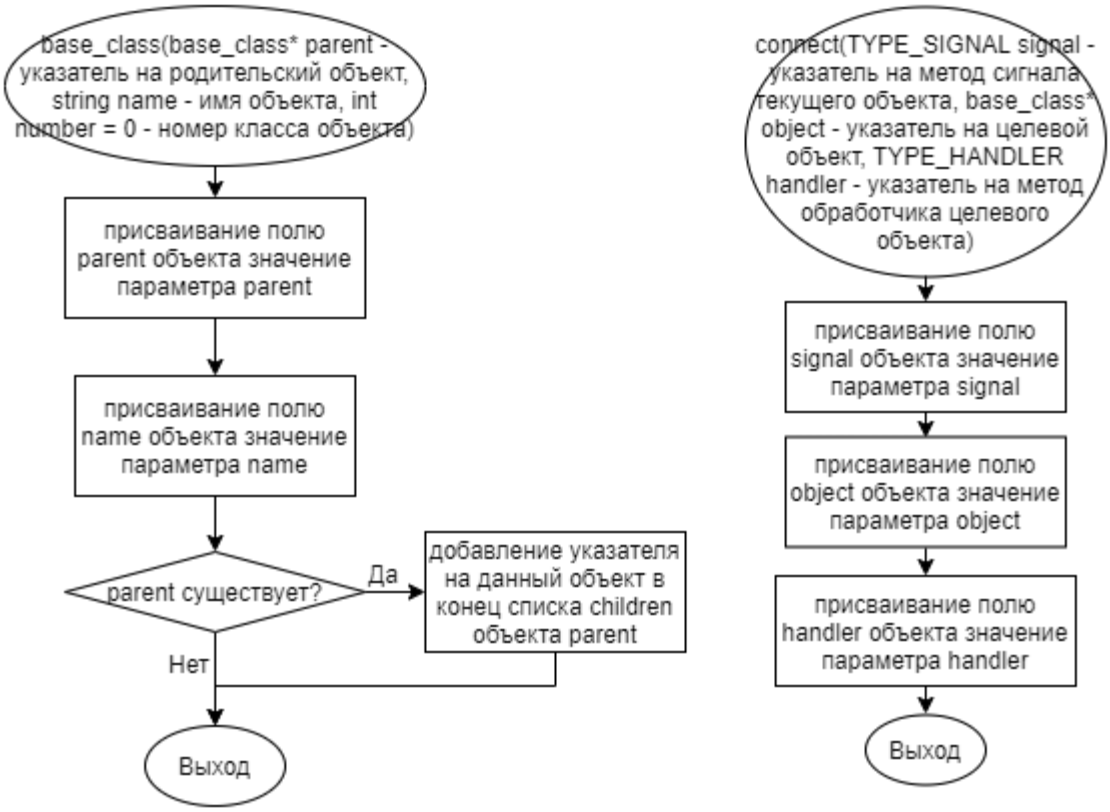

Рисунок 5 – Блок-схема алгоритма

Рисунок 6 – Блок-схема алгоритма

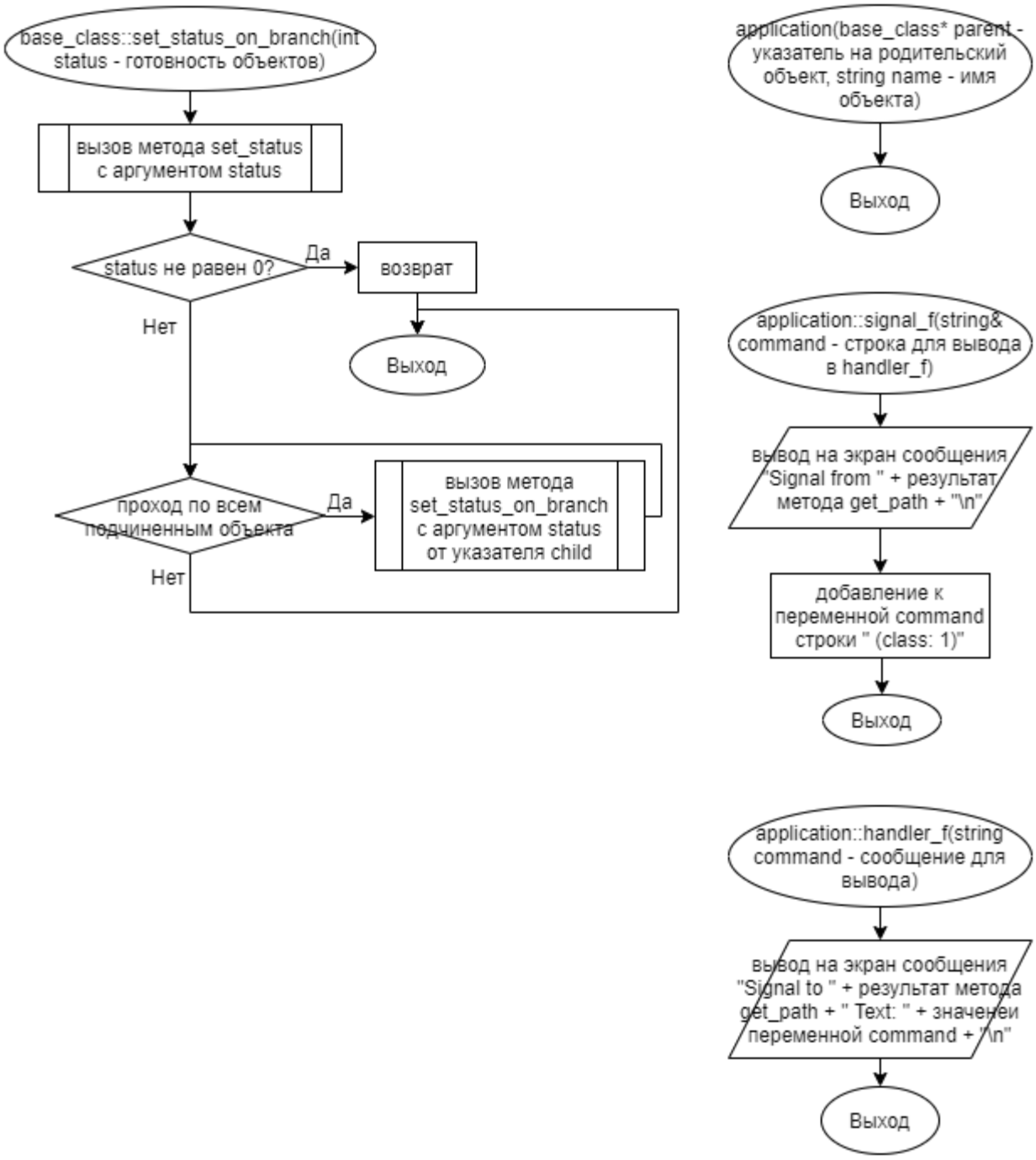

Рисунок 7 – Блок-схема алгоритма

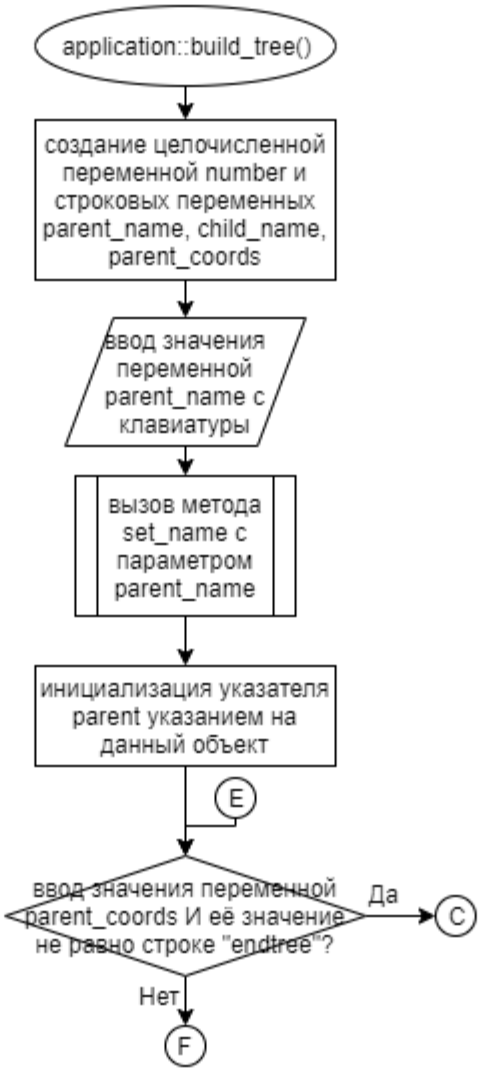

Рисунок 8 – Блок-схема алгоритма

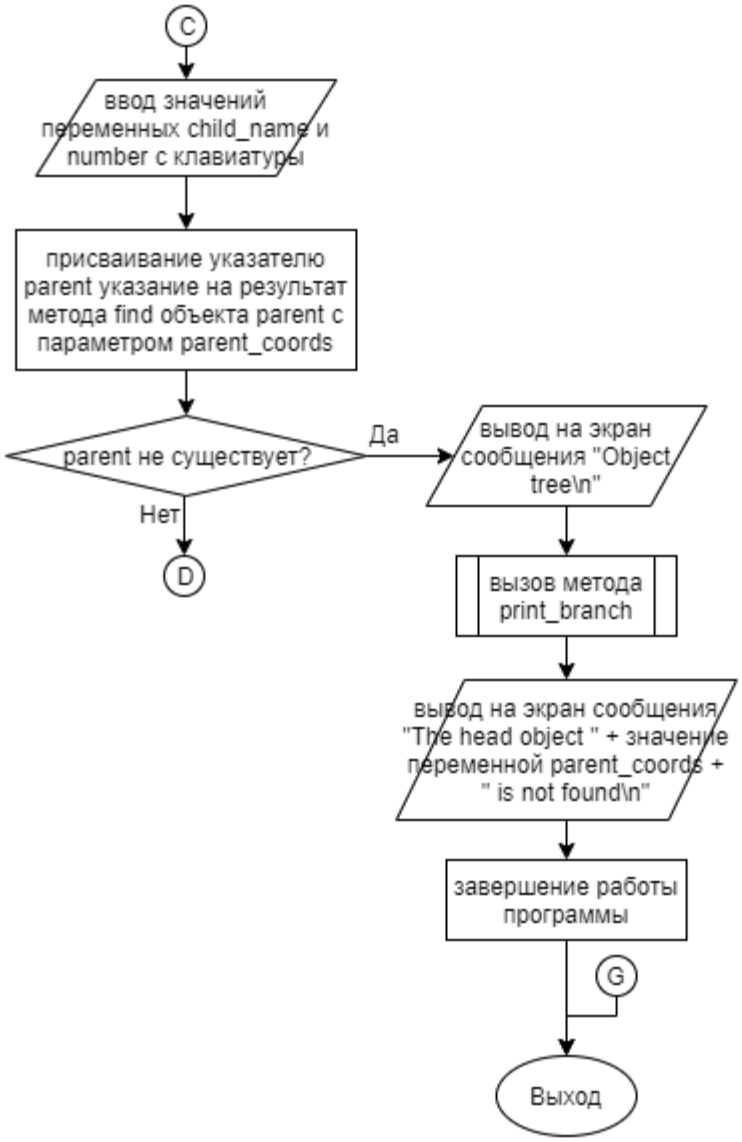

Рисунок 9 – Блок-схема алгоритма

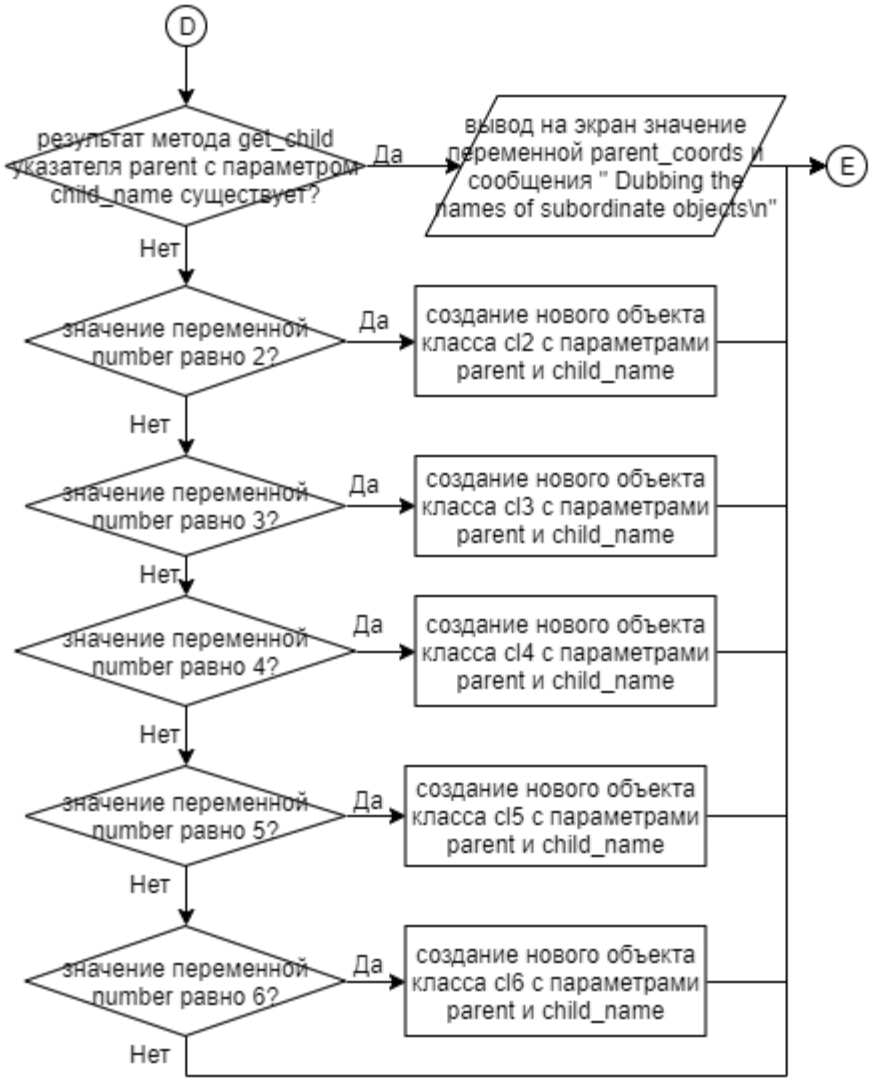

Рисунок 10 – Блок-схема алгоритма

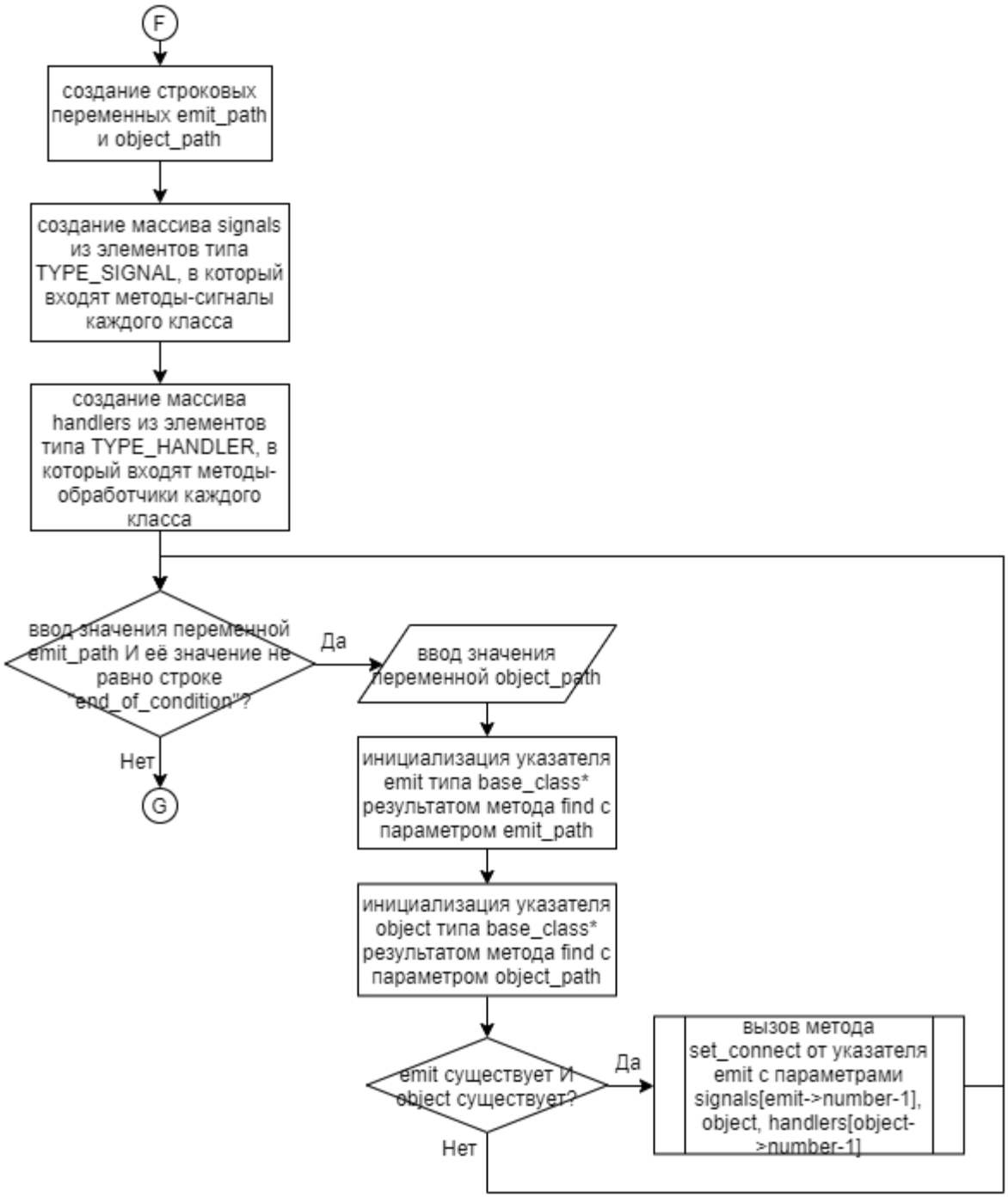

Рисунок 11 – Блок-схема алгоритма

Рисунок 12 – Блок-схема алгоритма

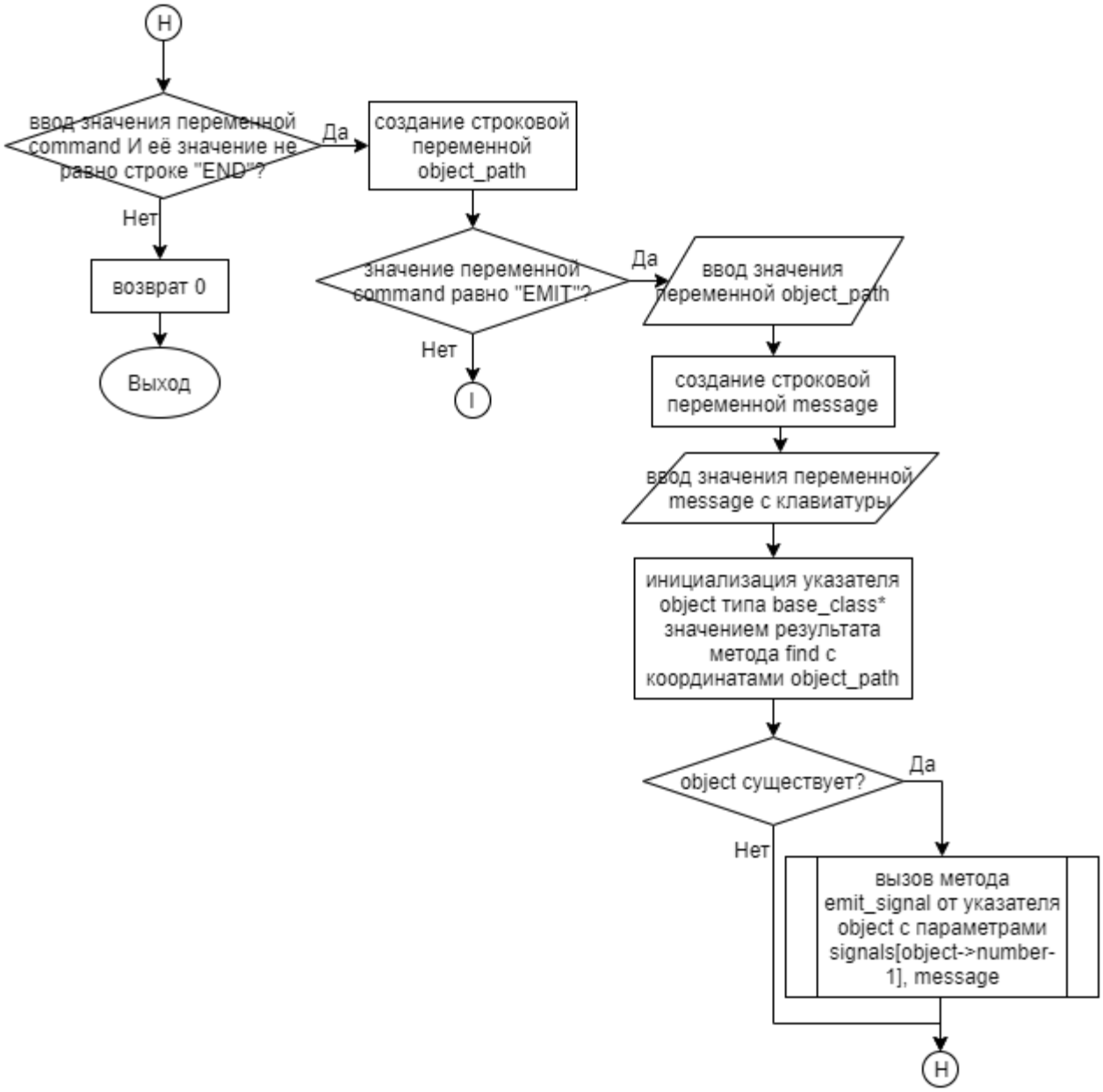

Рисунок 13 – Блок-схема алгоритма

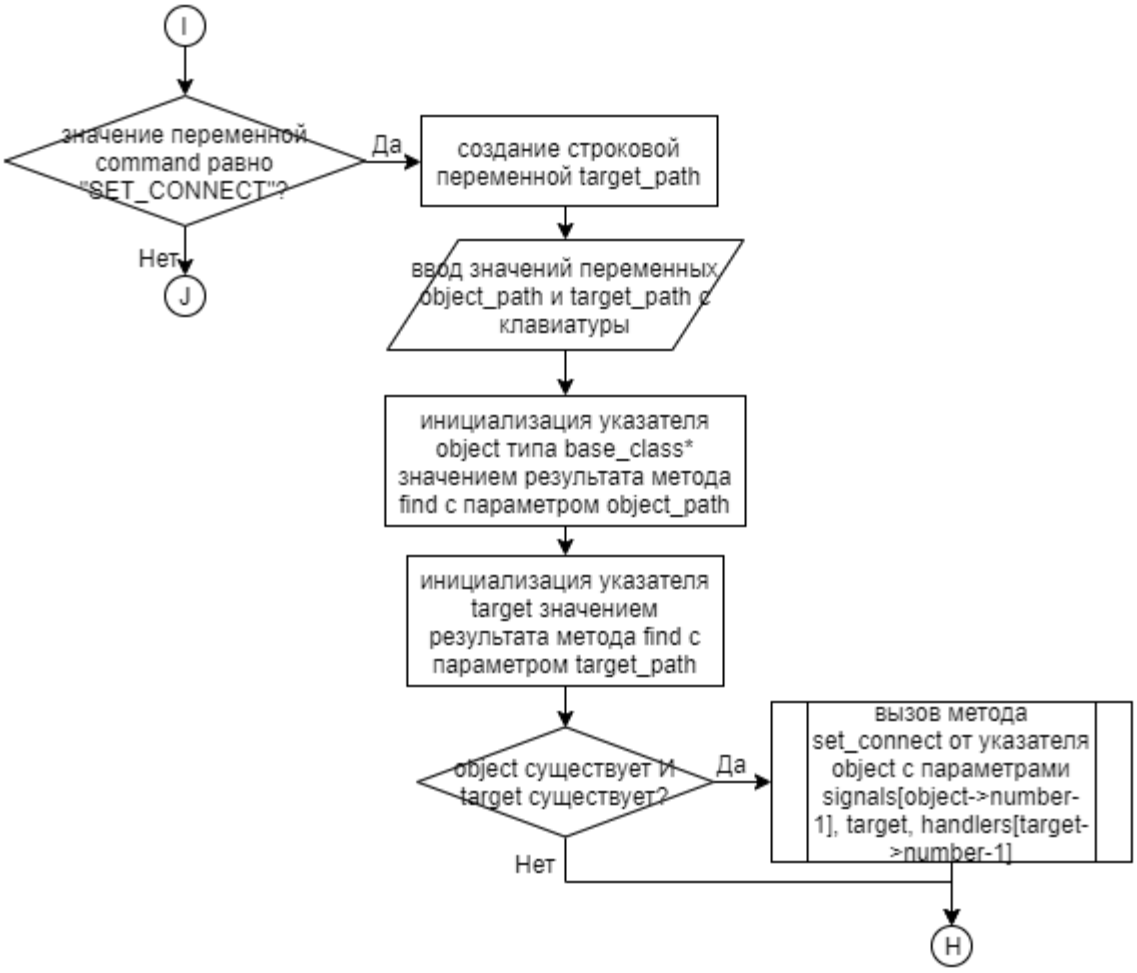

Рисунок 14 – Блок-схема алгоритма

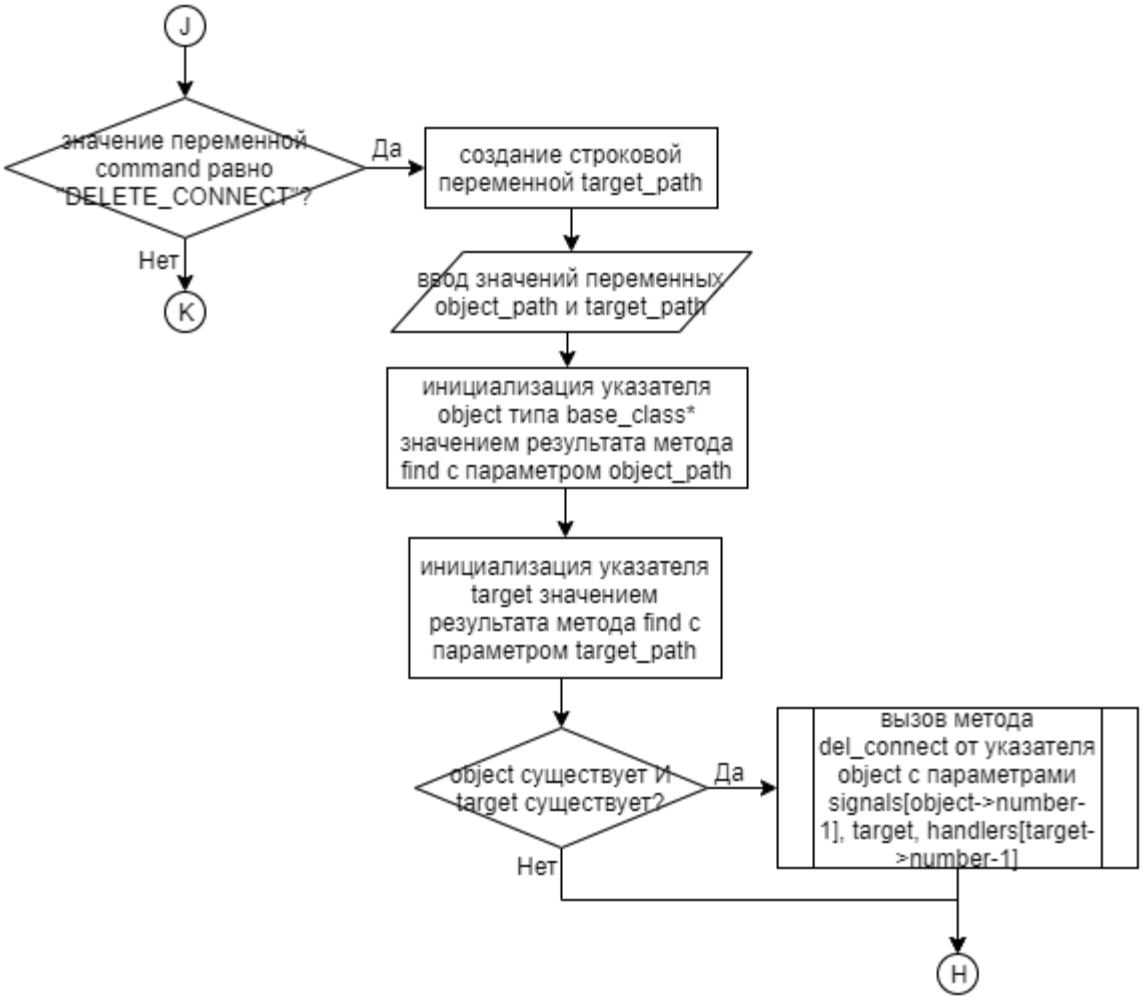

Рисунок 15 – Блок-схема алгоритма

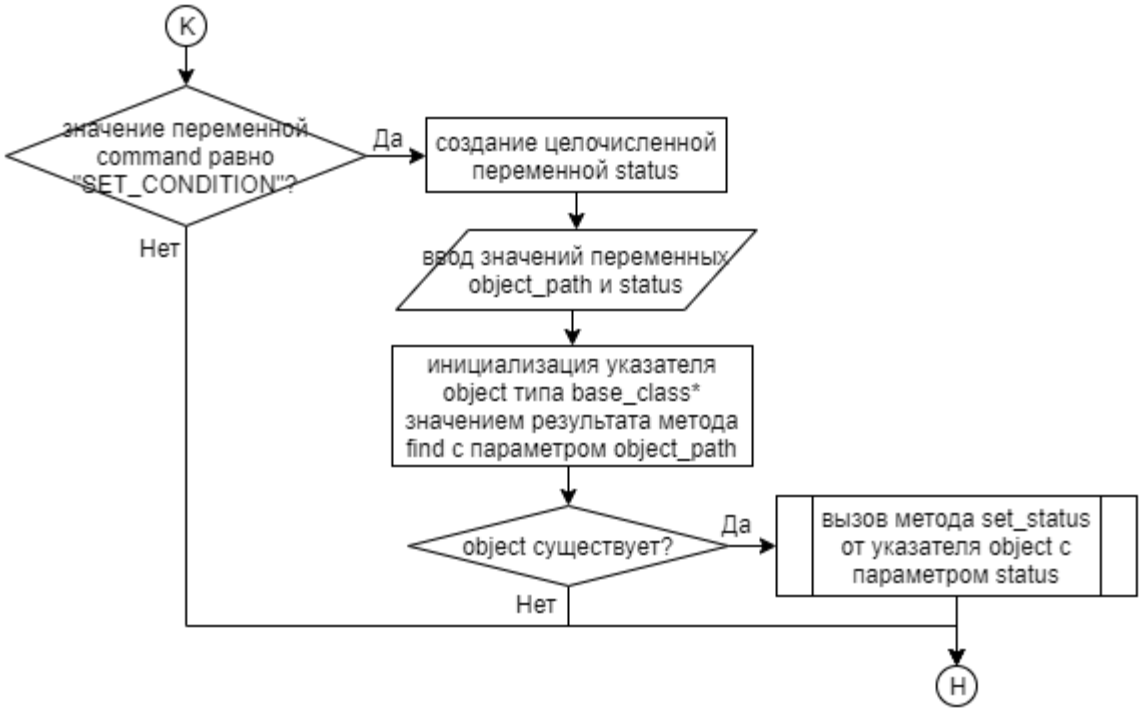

Рисунок 16 – Блок-схема алгоритма

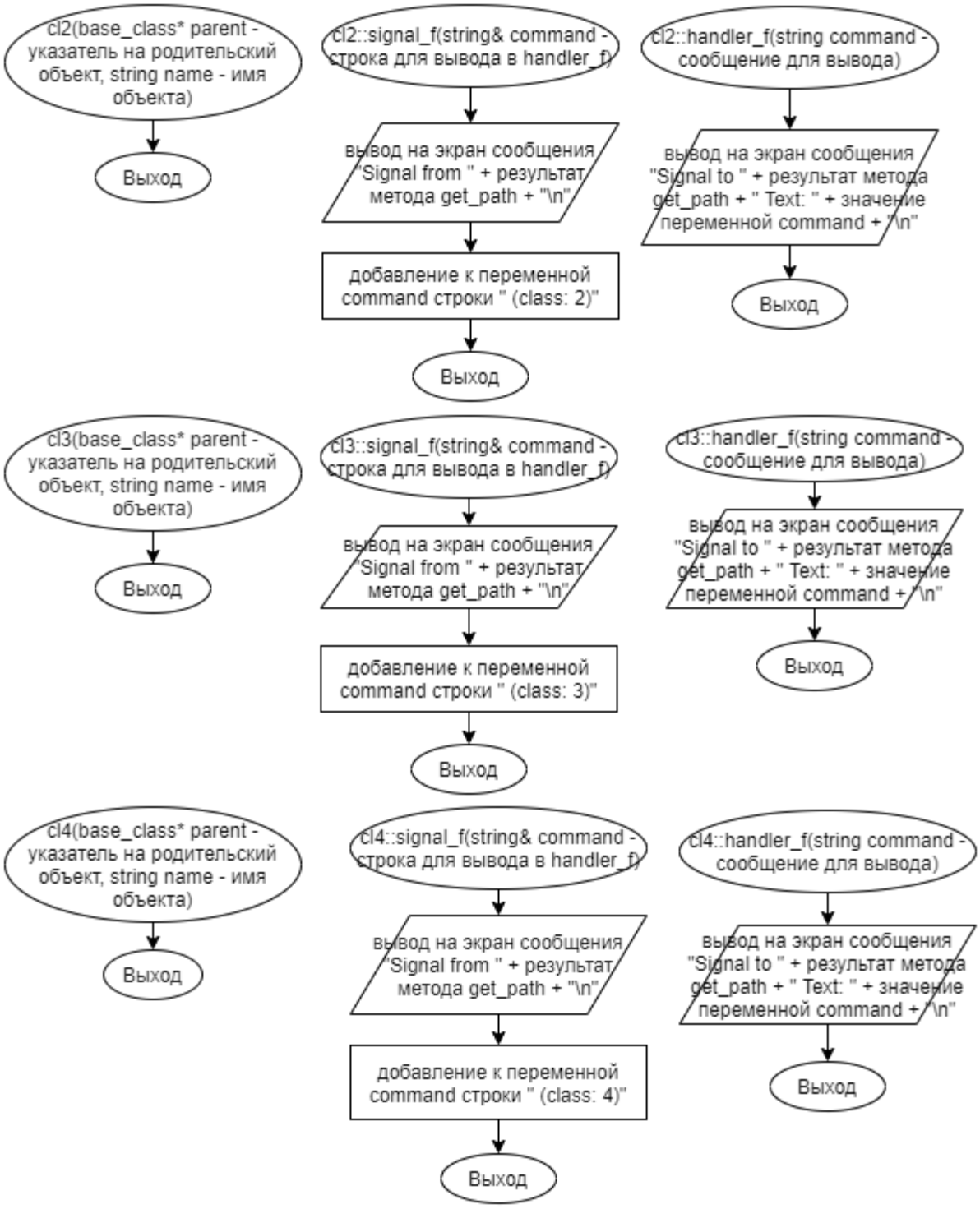

Рисунок 17 – Блок-схема алгоритма

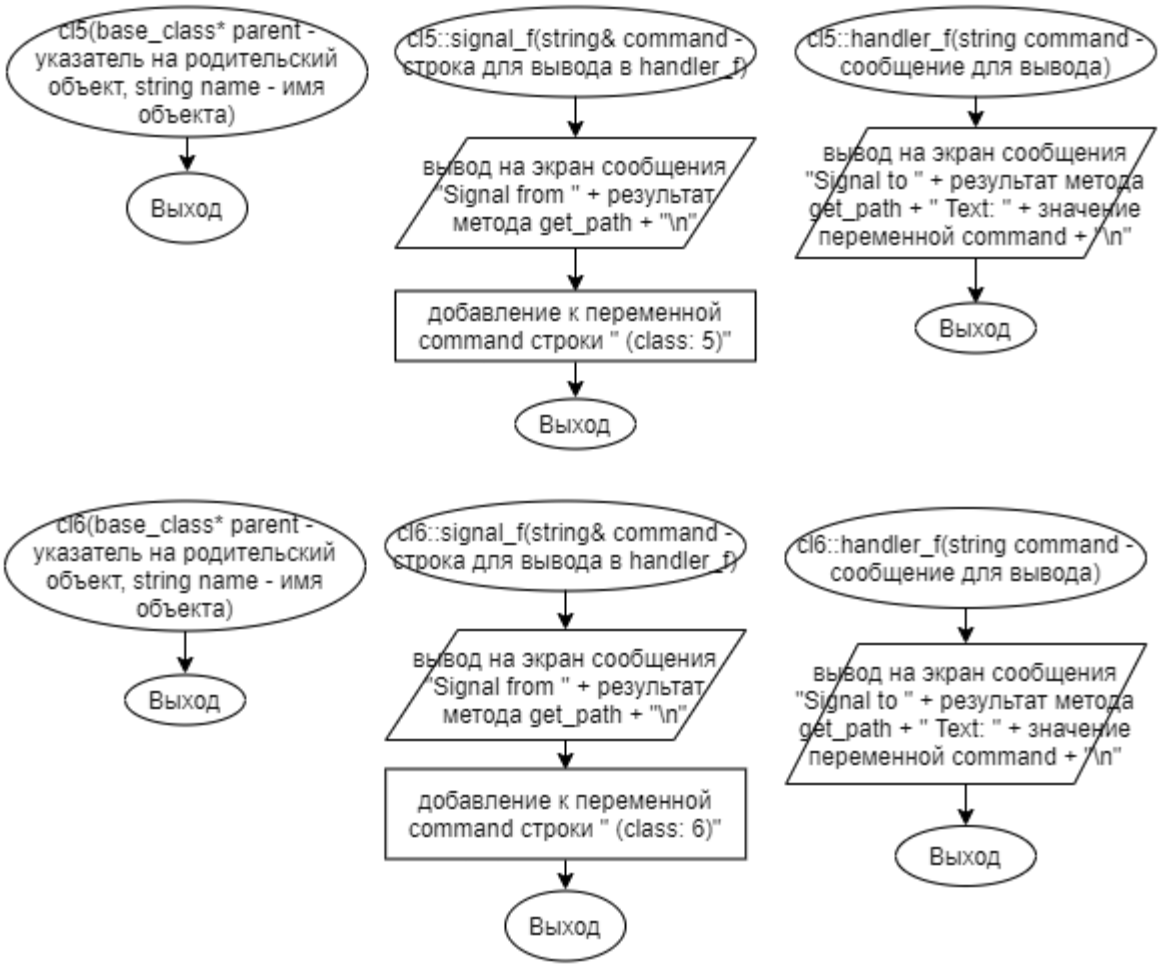

Рисунок 18 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
#include "cl2.h"
#include "cl3.h"
#include "cl4.h"
#include "cl5.h"
#include "cl6.h"
#include <iostream>

application::application(base_class*     parent,     std::string     name):
base_class(parent, name, 1) { }

void application::build_tree() // changed
{
   int number;
   std::string parent_name, child_name, parent_coords;
   std::cin >> parent_name;
   set_name(parent_name);
   base_class* parent = this;
   while (std::cin >> parent_coords && parent_coords != "endtree")
   {
      std::cin >> child_name >> number;
      parent = parent->find(parent_coords);
      if (!parent)
      {
         std::cout << "Object tree\n";
         print_branch();
         std::cout << "The head object " << parent_coords << " is not found\
n";
         exit(1);
      }
      if (parent->get_child(child_name))
         std::cout << parent_coords << "     Dubbing the names of subordinate
objects\n";
      else
      {
         if (number == 2)
            new cl2(parent, child_name);
         else if (number == 3)
            new cl3(parent, child_name);
```

```
            else if (number == 4)
               new cl4(parent, child_name);
            else if (number == 5)
               new cl5(parent, child_name);
            else if (number == 6)
               new cl6(parent, child_name);
         }
      }
      std::string emit_path, object_path;
      TYPE_SIGNAL signals[]
      {
         SIGNAL(application::signal_f),                    SIGNAL(cl2::signal_f),
   SIGNAL(cl3::signal_f),     SIGNAL(cl4::signal_f),     SIGNAL(cl5::signal_f),
   SIGNAL(cl6::signal_f)
      };
      TYPE_HANDLER handlers[]
      {
         HANDLER(application::handler_f),               HANDLER(cl2::handler_f),
   HANDLER(cl3::handler_f),  HANDLER(cl4::handler_f),  HANDLER(cl5::handler_f),
   HANDLER(cl6::handler_f)
      };
      while (std::cin >> emit_path && emit_path != "end_of_connections")
      {
         std::cin >> object_path;
         base_class* emit = find(emit_path);
         base_class* object = find(object_path);
         if (emit && object)
            emit->set_connect(signals[emit->number      -      1],      object,
   handlers[object->number - 1]);
      }
   }
   int application::exec_app() // changed
   {
      std::cout << "Object tree\n";
      print_branch();
      set_status_on_branch(1);
      std::string command;
      TYPE_SIGNAL signals[]
      {
         SIGNAL(application::signal_f),                    SIGNAL(cl2::signal_f),
   SIGNAL(cl3::signal_f),     SIGNAL(cl4::signal_f),     SIGNAL(cl5::signal_f),
   SIGNAL(cl6::signal_f)
      };
      TYPE_HANDLER handlers[]
      {
         HANDLER(application::handler_f),               HANDLER(cl2::handler_f),
   HANDLER(cl3::handler_f),  HANDLER(cl4::handler_f),  HANDLER(cl5::handler_f),
   HANDLER(cl6::handler_f)
      };
      while (std::cin >> command && command != "END")
      {
         std::string object_path;
         if (command == "EMIT")
         {
            std::cin >> object_path;
```

```
            std::string message;
            std::getline(std::cin, message);
            base_class* object = find(object_path);
            if   (object)   object->emit_signal(signals[object->number   -   1],
message);
         }
         else if (command == "SET_CONNECT")
         {
            std::string target_path;
            std::cin >> object_path >> target_path;
            base_class* object = find(object_path);
            base_class* target = find(target_path);
            if (object && target) object->set_connect(signals[object->number -
1], target, handlers[target->number - 1]);
         }
         else if (command == "DELETE_CONNECT")
         {
            std::string target_path;
            std::cin >> object_path >> target_path;
            base_class* object = find(object_path);
            base_class* target = find(target_path);
            if (object && target) object->del_connect(signals[object->number -
1], target, handlers[target->number - 1]);
         }
         else if (command == "SET_CONDITION")
         {
            int status;
            std::cin >> object_path >> status;
            base_class* object = find(object_path);
            if (object) object->set_status(status);
         }
      }
      return 0;
   }

   void application::signal_f(std::string& command) // new
   {
      std::cout << "Signal from " + get_path() + "\n";
      command += " (class: 1)";
   }
   void application::handler_f(std::string command) // new
   { std::cout << "Signal to " + get_path() + " Text: " + command + "\n"; }
```

5.2 Файл application.h

Листинг 2 – application.h

```
   #ifndef __APPLICATION__H
   #define __APPLICATION__H
   #include "base_class.h"
```

```
class application: public base_class
{
public:
   application(base_class* parent, std::string name = "root");
   void build_tree(); // changed
   int exec_app(); // changed
   void signal_f(std::string& command); // new
   void handler_f(std::string command); // new
};

#endif
```

5.3 Файл base_class.cpp

Листинг 3 – base_class.cpp

```
#include "base_class.h"
#include <iostream>
#include <functional>

base_class::base_class(base_class*    parent,    std::string    name,    int
number):number(number) // changed
{
   this->parent = parent;
   this->name = name;
   if (parent) parent->children.push_back(this);
}
base_class::~base_class()
{ for (base_class* child: children) delete child; }

bool base_class::set_name(std::string name)
{
   if (parent && parent->get_child(name))
      return false;
   this->name = name;
   return true;
}
std::string base_class::get_name()
{ return name; }
base_class* base_class::get_parent()
{ return parent; }
base_class* base_class::get_child(std::string name)
{
   for (base_class* child: children)
      if (child->get_name() == name)
         return child;
   return nullptr;
}
```

```
base_class* base_class::search_in_branch(std::string name)
{
   std::function <int(std::string name, base_class* object)> count = [&count]
(std::string name, base_class* object)
   {
      int c = 0;
      if (object->get_name() == name) c++;
      for (int i = 0; i < object->children.size(); i++)
         c += count(name, object->children[i]);
      return c;
   };
   if (count(name, this) != 1) return nullptr;
   if (get_name() == name) return this;
   else
      for (base_class* child: children)
      {
         base_class* object = child->search_in_branch(name);
         if (object) return object;
      }
   return nullptr;
}
base_class* base_class::search_in_tree(std::string name)
{
   base_class* parent = this;
   while (parent->get_parent())
      parent = parent->get_parent();
   return parent->search_in_branch(name);
}

void base_class::print_branch()
{
   base_class* parent = get_parent();
   while (parent)
   {
      std::cout << "    ";
      parent = parent->get_parent();
   }
   std::cout << name << std::endl;
   for (base_class* child: children)
      child->print_branch();
}
void base_class::print_branch_status()
{
   base_class* parent = get_parent();
   while (parent)
   {
      std::cout << "    ";
      parent = parent->get_parent();
   }
   std::cout << name << " is ";
   if (!status)
      std::cout << "not ";
   std::cout << "ready\n";
   for (base_class* child: children)
      child->print_branch_status();
```

```
}

void base_class::set_status(int status) // changed
{
   if (status == 0)
   {
      this->status = 0;
      for (base_class* child: children)
         child->set_status(0);
   }
   else
   {
      if (get_parent() && get_parent()->status == 0)
         this->status = 0;
      else
         this->status = status;
   }
}
void base_class::set_status_on_branch(int status) // new
{
   set_status(status);
   if (status == 0) return;
   for (base_class* child: children)
      child->set_status_on_branch(status);
}

base_class* base_class::find(std::string coords)
{
   if (coords.empty()) return nullptr;
   base_class* parent = this;
   if (coords[0] == '.')
   {
      if (coords == ".")
         return this;
      coords.erase(0, 1);
      return this->search_in_branch(coords);
   }
   else if (coords[0] == '/')
   {
      while (parent->get_parent())
         parent = parent->get_parent();
      if (coords == "/")
         return parent;
      if (coords[1] == '/')
      {
         coords.erase(0, 2);
         return parent->search_in_tree(coords);
      }
      coords.erase(0, 1);
   }
   std::string current = "";
   for (int i = 0; i < coords.length(); i++)
   {
      if (coords[i] == '/')
      {
```

```
            parent = parent->get_child(current);
            if (!parent) return parent;
            current = "";
         }
         else current += coords[i];
      }
      return parent->get_child(current);
   }
   void base_class::delete_child(std::string name)
   {
      base_class* child = get_child(name);
      if (child)
         for (int i = 0; i < children.size(); i++)
            if (children[i] == child)
            {
               children.erase(children.begin() + i);
               delete child;
               break;
            }
   }
   bool base_class::set_parent(base_class* object)
   {
      if  (!object  ||  !this->parent  ||  object->get_child(this->get_name()))
   return false;
      base_class* parent = object;
      while (parent)
      {
         if (parent == this) return false;
         parent = parent->get_parent();
      }
      base_class* current = this->parent;
      for (int i = 0; i < current->children.size(); i++)
         if (current->children[i] == this)
         {
            current->children.erase(current->children.begin() + i);
            break;
         }
      this->parent = object;
      object->children.push_back(this);
      return true;
   }

   base_class::connect::connect(TYPE_SIGNAL    signal,    base_class*    object,
   TYPE_HANDLER handler) // new
   {
      this->signal = signal;
      this->object = object;
      this->handler = handler;
   }
   void   base_class::set_connect(TYPE_SIGNAL   signal,   base_class*   object,
   TYPE_HANDLER handler) // new
   {
      for (int i = 0; i < connects.size(); i++)
         if (connects[i]->signal == signal && connects[i]->object == object &&
   connects[i]->handler == handler)
```

```
            return;
      connects.push_back(new connect(signal, object, handler));
   }
   void   base_class::del_connect(TYPE_SIGNAL   signal,   base_class*   object,
   TYPE_HANDLER handler) // new
   {
      for (int i = 0; i < connects.size(); i++)
         if (connects[i]->signal == signal && connects[i]->object == object &&
   connects[i]->handler == handler)
         {
            connects.erase(connects.begin() + i);
            return;
         }
   }
   void  base_class::emit_signal(TYPE_SIGNAL  signal,  std::string&  command)  //
   new
   {
      if (status == 0) return;
      TYPE_HANDLER handler;
      base_class* object;
      (this->*signal) (command);
      for (int i = 0; i < connects.size(); i++)
         if (connects[i]->signal == signal)
         {
            handler = connects[i]->handler;
            object = connects[i]->object;
            if (object->status != 0)
               (object->*handler) (command);
         }
   }
   std::string base_class::get_path() //new
   {
      base_class* current = get_parent();
      if (!current) return "/";
      std::string path = '/' + get_name();
      while (current->get_parent())
      {
         path = '/' + current->get_name() + path;
         current = current->get_parent();
      }
      return path;
   }
```

5.4 Файл base_class.h

Листинг 4 – base_class.h

```
   #ifndef __BASE_CLASS__H
   #define __BASE_CLASS__H
```

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

class base_class
{
public:
   const int number; // new
   base_class(base_class* parent, std::string name = "default", int number =
0); // changed
   ~base_class();
   bool set_name(std::string name);
   std::string get_name();
   base_class* get_parent();
   base_class* get_child(std::string name);
   base_class* search_in_branch(std::string name);
   base_class* search_in_tree(std::string name);
   void print_branch();
   void print_branch_status();
   void set_status(int status); // changed
   base_class* find(std::string coords);
   void delete_child(std::string name);
   bool set_parent(base_class* object);

   typedef void (base_class::*TYPE_SIGNAL) (std::string&); // new
   typedef void (base_class::*TYPE_HANDLER) (std::string); // new
   void  set_connect(TYPE_SIGNAL  signal,  base_class*  object,  TYPE_HANDLER
handler); // new
   void  del_connect(TYPE_SIGNAL  signal,  base_class*  object,  TYPE_HANDLER
handler); // new
   void emit_signal(TYPE_SIGNAL signal, std::string& command); // new
   std::string get_path(); //new
   void set_status_on_branch(int status); // new
private:
   std::string name;
   base_class* parent;
   std::vector <base_class*> children;
   int status = 0;

   struct connect // new
   {
      TYPE_SIGNAL signal;
      base_class* object;
      TYPE_HANDLER handler;
      connect(TYPE_SIGNAL signal, base_class* object, TYPE_HANDLER handler);
   };
   std::vector <connect*> connects; // new
};
#define SIGNAL(signal_func) (TYPE_SIGNAL) (&signal_func) // new
#define HANDLER(handler_func) (TYPE_HANDLER) (&handler_func) // new

#endif
```

5.5 Файл cl2.cpp

Листинг 5 – cl2.cpp

```
#include "cl2.h"

cl2::cl2(base_class* parent, std::string name): base_class(parent, name, 2)
{ } // changed

void cl2::signal_f(std::string& command) // new
{
   std::cout << "Signal from " + get_path() + "\n";
   command += " (class: 2)";
}
void cl2::handler_f(std::string command) // new
{ std::cout << "Signal to " + get_path() + " Text: " + command + "\n"; }
```

5.6 Файл cl2.h

Листинг 6 – cl2.h

```
#ifndef __CL2__H
#define __CL2__H
#include "base_class.h"

class cl2: public base_class
{
public:
   cl2(base_class* parent, std::string name = "cl2"); // changed
   void signal_f(std::string& command); // new
   void handler_f(std::string command); // new
};

#endif
```

5.7 Файл cl3.cpp

Листинг 7 – cl3.cpp

```
#include "cl3.h"

cl3::cl3(base_class* parent, std::string name): base_class(parent, name, 3)
{ } // changed
```

```
void cl3::signal_f(std::string& command) // new
{
   std::cout << "Signal from " + get_path() + "\n";
   command += " (class: 3)";
}
void cl3::handler_f(std::string command) // new
{ std::cout << "Signal to " + get_path() + " Text: " + command + "\n"; }
```

5.8 Файл cl3.h

Листинг 8 – cl3.h

```
#ifndef __CL3__H
#define __CL3__H
#include "base_class.h"

class cl3: public base_class
{
public:
   cl3(base_class* parent, std::string name = "cl3"); // changed
   void signal_f(std::string& command); // new
   void handler_f(std::string command); // new
};

#endif
```

5.9 Файл cl4.cpp

Листинг 9 – cl4.cpp

```
#include "cl4.h"

cl4::cl4(base_class* parent, std::string name): base_class(parent, name, 4)
{ } // changed

void cl4::signal_f(std::string& command) // new
{
   std::cout << "Signal from " + get_path() + "\n";
   command += " (class: 4)";
}
void cl4::handler_f(std::string command) // new
{ std::cout << "Signal to " + get_path() + " Text: " + command + "\n"; }
```

5.10 Файл cl4.h

Листинг 10 – cl4.h

```
#ifndef __CL4__H
#define __CL4__H
#include "base_class.h"

class cl4: public base_class
{
public:
   cl4(base_class* parent, std::string name = "cl4"); // changed
   void signal_f(std::string& command); // new
   void handler_f(std::string command); // new
};

#endif
```

5.11 Файл cl5.cpp

Листинг 11 – cl5.cpp

```
#include "cl5.h"

cl5::cl5(base_class* parent, std::string name): base_class(parent, name, 5)
{ } // changed

void cl5::signal_f(std::string& command) // new
{
   std::cout << "Signal from " + get_path() + "\n";
   command += " (class: 5)";
}
void cl5::handler_f(std::string command) // new
{ std::cout << "Signal to " + get_path() + " Text: " + command + "\n"; }
```

5.12 Файл cl5.h

Листинг 12 – cl5.h

```
#ifndef __CL5__H
#define __CL5__H
#include "base_class.h"
```

```
class cl5: public base_class
{
public:
   cl5(base_class* parent, std::string name = "cl5"); // changed
   void signal_f(std::string& command); // new
   void handler_f(std::string command); // new
};

#endif
```

5.13 Файл cl6.cpp

Листинг 13 – cl6.cpp

```
#include "cl6.h"

cl6::cl6(base_class* parent, std::string name): base_class(parent, name, 6)
{ } // changed

void cl6::signal_f(std::string& command) // new
{
   std::cout << "Signal from " + get_path() + "\n";
   command += " (class: 6)";
}
void cl6::handler_f(std::string command) // new
{ std::cout << "Signal to " + get_path() + " Text: " + command + "\n"; }
```

5.14 Файл cl6.h

Листинг 14 – cl6.h

```
#ifndef __CL6__H
#define __CL6__H
#include "base_class.h"

class cl6: public base_class
{
public:
   cl6(base_class* parent, std::string name = "cl6"); // changed
   void signal_f(std::string& command); // new
   void handler_f(std::string command); // new
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "application.h"

int main()
{
   application tree(nullptr);
   tree.build_tree();
   return tree.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 31.

Таблица 31 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 EMIT /object_s2/object_s4 Send message 2 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 4 END	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6	Object tree appls_root object_s1 object_s7 object_s2 object_s4	Object tree appls_root object_s1 object_s7 object_s2 object_s4

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2 Send TEXT1 EMIT /object_s2/object_s4 Send TEXT2 SET_CONNECT /object_s1 /object_s2/object_s4 SET_CONDITION / 0 SET_CONDITION / 1 SET_CONDITION /object_s2 1 END	object_s6 object_s13 Signal from /object_s2 Signal to /object_s1/object_s7 Text: Send TEXT1 (class: 2) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send TEXT2 (class: 4) Signal to / Text: Send TEXT2 (class: 4)	object_s6 object_s13 Signal from /object_s2 Signal to /object_s1/object_s7 Text: Send TEXT1 (class: 2) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send TEXT2 (class: 4) Signal to / Text: Send TEXT2 (class: 4)
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	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	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

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
EMIT /object_s2/object_s4 Send message 3 EMIT /object_s2 Send message 4 EMIT /object_s2/object_s4 Send message 5 EMIT /object_s2/object_s4 Send message 3 EMIT /object_s1 Send message 2 END	(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_s2 Signal to /object_s1/object_s7 Text: Send message 4 (class: 2) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 5 (class: 4) Signal to / Text: Send message 5 (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	(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_s2 Signal to /object_s1/object_s7 Text: Send message 4 (class: 2) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 5 (class: 4) Signal to / Text: Send message 5 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 3 (class: 4) Signal to / Text: Send message 3 (class: 4) Signal from /object_s1
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 /	Object 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	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

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
/object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 DELETE_CONNECT /object_s2/object_s4 /object_s2/object_s6 EMIT /object_s2/object_s4 Send message 2 SET_CONNECT /object_s2/object_s4 /object_s2/object_6 SET_CONDITION / 0 EMIT /object_s2/object_s4 Send message 3 SET_CONDITION / 1 SET_CONDITION /object_s2 1 EMIT /object_s2/object_s4 Send message 4 SET_CONDITION /object_s2/object_s4 1 EMIT /object_s2/object_s4 Send message 5 SET_CONDITION /object_s2/object_s6 1 EMIT /object_s2/object_s4 Send message 6 EMIT /object_s1 Send message 7 SET_CONNECT /object_s1 /object_s2/object_s4 EMIT /object_s1 Send message 8 SET_CONDITION /object_s2/object_s4 1 EMIT /object_s1 Send message 9 SET_CONDITION /object_s1 1 EMIT /object_s1 Send	(class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 5 (class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 6 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s4 Text: Send message 10 (class: 3)	(class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 2 (class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 5 (class: 4) Signal from /object_s2/object_s4 Signal to / Text: Send message 6 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s4 Text: Send message 10 (class: 3)

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
message 10 END		

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).