Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....7
1 ПОСТАНОВКА ЗАДАЧИ.....8
1.1 Описание входных данных.....10
1.2 Описание выходных данных.....12
2 МЕТОД РЕШЕНИЯ.....14
3 ОПИСАНИЕ АЛГОРИТМОВ.....20
3.1 Алгоритм функции main.....20
3.2 Алгоритм конструктора класса Base.....20
3.3 Алгоритм метода WayPtr класса Base.....21
3.4 Алгоритм метода Signal класса Base.....22
3.5 Алгоритм метода Handler класса Base.....23
3.6 Алгоритм метода SetConnection класса Base.....24
3.7 Алгоритм метода DeleteConnections класса Base.....24
3.8 Алгоритм метода BuildPath класса Base.....25
3.9 Алгоритм метода SetSignal класса Base.....26
3.10 Алгоритм метода SetStatus класса Base.....26
3.11 Алгоритм метода BuildTree класса App.....27
3.12 Алгоритм метода exe класса App.....29
3.13 Алгоритм конструктора класса Class2.....31
3.14 Алгоритм метода Signal класса Class2.....31
3.15 Алгоритм метода Handler класса Class2.....31
3.16 Алгоритм конструктора класса Class3.....32
3.17 Алгоритм метода Signal класса Class3.....32
3.18 Алгоритм метода Handler класса Class3.....33
3.19 Алгоритм конструктора класса Class4.....34
3.20 Алгоритм метода Signal класса Class4.....34

3.21 Алгоритм метода Handler класса Class4.....34
3.22 Алгоритм конструктора класса Class5.....35
3.23 Алгоритм метода Signal класса Class5.....35
3.24 Алгоритм метода Handler класса Class5.....36
3.25 Алгоритм конструктора класса Class6.....37
3.26 Алгоритм метода Signal класса Class6.....37
3.27 Алгоритм метода Handler класса Class6.....37
3.28 Алгоритм метода PrintTree класса Base.....38
3.29 Алгоритм метода SetName класса Base.....39
3.30 Алгоритм метода GetName класса Base.....39
3.31 Алгоритм конструктора класса App.....40
3.32 Алгоритм метода FindPtr класса Base.....40
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....42
5 КОД ПРОГРАММЫ.....74
5.1 Файл App.cpp.....74
5.2 Файл App.h.....76
5.3 Файл Base.cpp.....77
5.4 Файл Base.h.....80
5.5 Файл Class2.cpp.....81
5.6 Файл Class2.h.....81
5.7 Файл Class3.cpp.....82
5.8 Файл Class3.h.....82
5.9 Файл Class4.cpp.....82
5.10 Файл Class4.h.....83
5.11 Файл Class5.cpp.....83
5.12 Файл Class5.h.....84
5.13 Файл Class6.cpp.....84

5.14 Файл Class6.h.....85
5.15 Файл main.cpp.....85
6 ТЕСТИРОВАНИЕ.....86
ЗАКЛЮЧЕНИЕ.....87
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....88

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответсвии с требованиями ГОСТ Единой системы программной документации (ЕСПД). Все этапы решения задача курсовой работы фиксированы, соответсвуют методике разработки объекто-ориентированных программ [1-2] и требованиям приведенным в методическом пособии для проведения практических заданий контрольных курсовых работ по дисциплине "Объектно-ориентированное программирование" [3-4].

В современном мире крайне актуально написание программного обеспечения с применением принципов ООП.

Целью данной работы является ознакомление с навыками проектирования и реализации программ на языке C++, ознакомление с основными принципами ООП такими, как инкапсуляция, наследование, полиморфизм, работа с алгоритмами сигналов и обработчиков.

Задача: написание программного обеспечения, моделирующей работу сигналов и обработчиков между объектами с применением основных принципов ООП в соответсвтии с поставленными условиями.

1 ПОСТАНОВКА ЗАДАЧИ

Реализация сигналов и обработчиков

Для организации взаимодействия объектов вне схемы взаимосвязи используется механизм сигналов и обработчиков. Вместе с передачей сигнала еще передается определенное множество данных. Механизм сигналов и обработчиков реализует схему взаимодействия объектов один ко многим.

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:

1. Установления связи между сигналом текущего объекта и обработчиком целевого объекта;

2. Удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;

3. 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. Вызов метода сигнала с передачей строковой переменной по ссылке.

2. Цикл по всем связям сигнал-обработчик текущего объекта.

2.1. Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то вызвать метод обработчика очередного целевого объекта и передать в качестве аргумента строковую переменную по значению.

3. Конец цикла.

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

В базовый класс добавить метод определения абсолютного 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.

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»

Реализовать алгоритм работы системы:

1. В методе построения системы:

1.1. Построение дерева иерархии объектов согласно вводу.

1.2. Ввод и построение множества связей сигнал-обработчик для заданных пар объектов.

2. В методе отработки системы:

2.1. Привести все объекты в состоянии готовности.

2.2. Цикл до признака завершения ввода.

2.2.1. Ввод наименования объекта и текста сообщения.

2.2.2. Вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной, содержащей текст сообщения.

2.3. 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

DELETE_CONNECT /object_s2/object_s4 /

EMIT /object_s2/object_s4 Send message 2

SET_CONDITION /object_s2/object_s4 0

EMIT /object_s2/object_s4 Send message 3

SET_CONNECT /object_s1 /object_s2/object_s6

EMIT /object_s1 Send message 4

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 from /object_s1

Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)

2 МЕТОД РЕШЕНИЯ

Для решения поставленной задачи используются:

объекты стандартных потоков ввода и вывода cin и cout соответственно для ввода и вывода на экран;

объект app класса App;

библиотека контейнеров vectror для создания векторов, которые хранят указатели на дочерние объекты и указатели на связанные объекты;

библиотека string для создания переменных строкового типа;

класс Base, являющийся базовым для классов App, Class2,Class3, Class4, Class5, Class6;

классы Class2,Class3, Class4, Class5, Class6, которые являются основными классами подчиненных объектов.

Класс Base:

1. поля:
   - поле имени объекта:
     1. наименование - name;
     2. тип - строка;
     3. модификатор доступа - protected.
   - поле указателя на объект:
     1. наименование - head;
     2. тип - указательно на объект класса Base;
     3. модификатор доступа - protected.
   - поле неправильной коодинаты:

наименование - wrong;

тип - строка;

модификатор доступа - protected.

- поле номера на сигнал текущего объекта:

наименование - class0fsignal;

тип - символьный;

модификатор доступа - protected.

- поле статуса готовности объекта:

наименование - status;

тип - целочисленный;

модификатор доступа - protected.

- поле номера класса:

наименование - number;

тип - символьный;

модификатор доступа - public.

- поле контейнера указателей на установленные связи:

наименование - connects;

тип - контейнер указателей на объекты типа Base.

модификатор доступа - public.

2. Функционал:

метод Base - параметризированный конструктор;

метод SetName - сеттер поля name;

метод GetName - геттер поля name;

метод FindPtr - поиск указателя на объект по имени;

метод WayPtr - поиск указателя на объект по указателю;

метод PrintTree - вывод дерева иерархии;

метод Signal - метода сигнала;

метод Handler - метод обработчика;

метод SetConnections - установка связи;

метод DeleteConnections - удаление связи;

метод BuildPath - построение координаты к объекту;

метод SetSignal - установка сигнала;

метод SetStatus - установка готовности объекта.

Структура o_sh:

поля:

поле указателя на метод сигнала:

наименование - p_signal;

тип - TYPE_SIGNAL.

модификатор доступа - public.

поле указателя на второй объект:

наименование - p_cl_base;

тип - указатель на класс Base.

модификатор доступа - public.

поле указателя на метод обработчика:

наименование - p_handler;

тип - TYPE_SIGNAL.

модификатор доступа - public.

функционал:

отсутсвуют.

Класс App:

поля:

Отсутсвуют.

функционал:

метод App - делегирующий конструктор;

метод BuildTree - метод построения дерева;

метод exe - метод запуска приложения.

Класс Class2:

поля:

отсутсвуют.

функционал:

метод Class2 - делегирующий конструктор;

метод Signal - метода сигнала;

метод Handler - метод обработчика.

Класс Class3:

поля:

отсутсвуют.

функционал:

метод Class3 - делегирующий конструктор;

метод Signal - метода сигнала;

метод Handler - метод обработчика.

Класс Class4:

поля:

отсутсвуют.

функционал:

метод Class4 - делегирующий конструктор;

метод Signal - метода сигнала;

метод Handler - метод обработчика.

Класс Class5:

поля:

отсутсвуют.

функционал:

метод Class5 - делегирующий конструктор;

метод Signal - метода сигнала;

метод Handler - метод обработчика.

Класс Class6:

поля:

отсутсвуют.

функционал:

метод Class6 - делегирующий конструктор;

метод Signal - метода сигнала;

метод Handler - метод обработчика.

Таблица 1 – Иерархия наследования классов

№	Базовый класс	Классы-наследники	Модификатор доступа	Описание	Номер	Комментарий
1	Base	App	public	Базовый класс	2	
		Class2	public		3	
		Class3	public		4	
		Class4	public		5	
		Class5	public		6	
		Class6	public		7	
2	App			Класс подчиненных объектов классу Base		
3	Class2			Класс подчиненных объектов		

				классу Base		
4	Class3			Класс подчиненных объектов классу Base		
5	Class4			Класс подчиненных объектов классу Base		
6	Class5			Класс подчиненных объектов классу Base		
7	Class6			Класс подчиненных объектов классу 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 функции main

Функционал: Основной алгоритм программы.

Параметры: Отсутсвуют.

Возвращаемое значение: Целочисленное значение - код возврата.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Инициализация главного объекта app типа App и передача в его конструктор указатель nullptr	2
2		Вызов у объекта app метод BuildTree	3
3		Возврат результата метода exe у объекта app	∅

3.2 Алгоритм конструктора класса Base

Функционал: Параметризированный конструктор.

Параметры: Указатель на объект типа Base, строка - name.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса Base

№	Предикат	Действия	№ перехода
1		Присвоение полю name значения параметра name	2
2		Присвоение полю head значения параметра head	3

№	Предикат	Действия	№ перехода
3		Обнуление поля wrong	4
4		Присвоение полю status значение 1 (отключение готовности объекта)	5
5	head не нулевой указатель	Добавление объекта в вектор children у объекта по указателю head	∅
			∅

3.3 Алгоритм метода WayPtr класса Base

Функционал: Поиск указателя на объект по координате.

Параметры: строка directory, булева переменная f.

Возвращаемое значение: Указатель на объект типа Base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода WayPtr класса Base

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной text пустой строкой	2
2		Инициализация указателя temp на объект класса Base и присвоения ему нулевого указателя	3
3	Переданная координата имеет в начале строки "//"	Удаление // в начале строки, присвоение указателю temp итога выполнения метода FindPtr с параметрами directory,this,false. Возврат temp	∅
			4
4		Инициализация указателя temp2 на объект класса Base с присвоением ему значения this	5
5	Цикл со счетчиком		6
			∅
6	Символ из строки directory		7

№	Предикат	Действия	№ перехода
	равен '/' или это последний символ в строке		
		Прибавление к переменной text символа directory[i]	12
7	Символ из строки не равен '/'	Прибавление к переменной text символа directory[i]	8
			8
8		Присвоение переменной temp результата выполнения метода FindPtr с параметрами text,temp2,true	9
9	temp2 равен нулевому указателю	Возврат temp2	∅
			10
10	Больше нет символов в строке	Возврат temp2	∅
			11
11		Обнуление строки text	12
12	i меньше размера вектора children	Увеличение i на 1	5
			5

3.4 Алгоритм метода Signal класса Base

Функционал: Метод сигнала.

Параметры: Строка - name.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода Signal класса Base

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением ей результата выполнения метода BuildPath с параметром this	2
2	Переменная temp не равно /	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала от объекта по координате temp	∅

3.5 Алгоритм метода Handler класса Base

Функционал: Обработчик сигнала.

Параметры: Строка - message.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода Handler класса Base

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением итога BuildPath с параметром this	2
2	Переменная temp не равна '/'	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала в объект по координате temp с текстом message от объекта класса ptr->class0fsignal	∅

3.6 Алгоритм метода SetConnection класса Base

Функционал: Метод установки связи.

Параметры: Указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Возвращаемое значение: Отсутсвует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода SetConnection класса Base

№	Предикат	Действия	№ перехода
1	Цикл со счетчиком i		2
			3
2	Переданные параметры равны полям объекта по указателю connects	Выход из функции	4
		Увеличение i на 1	1
3		Создание нового объекта типа o_sh с присвоением в его поля переданных значений	4
4		Запись объекта в вектор connects	∅

3.7 Алгоритм метода DeleteConnections класса Base

Функционал: Удаление связи между объектами.

Параметры: Указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Возвращаемое значение: Отсутсвует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода DeleteConnections класса Base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i=0	2
2	i меньше размера вектора connects		3
			∅
3	Найден объект по индексу i с индентичными полями в векторе connects	Удаление из вектора connects данного указателя	∅
		Увелечение i на 1	2

3.8 Алгоритм метода BuildPath класса Base

Функционал: Метод построения координаты по объекту.

Параметры: Указатель на класс Base, стока path.

Возвращаемое значение: Отсутсвует.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода BuildPath класса Base

№	Предикат	Действия	№ перехода
1	Переданный указатель не равен нулю		2
		Вперед пустой строки	∅
2		Прибавление к параметру path символа '/'	3
3	Указатель не родителя не нулевой указатель		4
			7
4		Инициализация строки переменной temp равной строке path	5
5		Обнуление строки path	6

№	Предикат	Действия	№ перехода
6		Прибавление к строке path итога вызова метода BuildPath от указателя на родителя head + итог выполнения метода GetName() от данного объекта + строка temp	7
7		Возврат строки path	∅

3.9 Алгоритм метода SetSignal класса Base

Функционал: Метод установки сигнала.

Параметры: Указатель на метод сигнала текущего объекта, строка message.

Возвращаемое значение: Отсутсвует.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода SetSignal класса Base

№	Предикат	Действия	№ перехода
1	Объект готов		2
			∅
2		Вызов метода Signal с параметром результата выполнения метода GetName, инициализация целочисленной переменной i	3
3	i меньше размера вектора connects	Присвоение всем полям class0fsignal объектам по указателям connects[i] значения number, вызов у каждого объекта метода Handler, увеличение i на единицу	3
			∅

3.10 Алгоритм метода SetStatus класса Base

Функционал: Метод установки готовности объекта и его дочерние объектам.

Параметры: Целочисленная переменная n.

Возвращаемое значение: Отсутсвует.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода SetStatus класса Base

№	Предикат	Действия	№ перехода
1		Присвоение полю status значения параметра n	2
2	Цикл со счетчиком i	Вызов метода SetStatus по указателю children[i], увеличение i на 1	∅
			∅

3.11 Алгоритм метода BuildTree класса App

Функционал: Построение дерева иерархии.

Параметры: Отсутсвуют.

Возвращаемое значение: Отсутсвует.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода BuildTree класса App

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных a,b	2
2		Объявление целочисленной переменной cl	3
3		Считывание с клавиатуры значения переменной a	4
4		Присваивание полю name головного объекта значения переменной a	5
5	Бесконечный цикл	Считывание с клавиатуры значения переменной a	6
			∅
6	Значение a равняется "endtree"		∅
			7

№	Предикат	Действия	№ перехода
7	Значение a равняется "/"	Присвоение переменной a поля name головного объекта	8
			8
8		Считывание с клавиатуры значения переменных b,cl	9
9	Значение переменной a равняется полю name головного объекта	Создание в зависимости от значения cl объекта класса Class2/3/4/5/6 с параметрами this,b	5
			10
10		Инициализация указателя на объект типа Base с присвоением результата выполнения метода WayPtr с параметрами a, true	11
11	temp не равняется нулевому указателю	Создание в зависимости от значения cl объекта класса Class2/3/4/5/6 с параметрами this,b	5
		Присвоение полю wrong значения переменной a, выход из цикла	12
12	Поле wrong равняется пустой строке	Объявление строковых переменных SignalTo, SignalFrom	13
			∅
13	Бесконечный цикл	Считывание с клавиатуры значения переменной SignalFrom	14
			∅
14	Значение переменной SignalFrom равняется "end_of_connections"	Выход из цикла	∅
		Считывание с клавиатуры значения переменной SignalTo	15
15		Инициализация указателя ptrTo на класс Base результатом выполнения метода WayPtr с	16

№	Предикат	Действия	№ перехода
		параметром SingleFrom	
16	Результат выполнения метода WayPtr с параметром SingleFrom не нулевой указатель и ptrTo не нулевой указатель	Вызов по указателю ptrTo метода SetConnections с параметрами signals[WayPtr(SignalTo)->number-'0'-1],WayPtr(SignalTo),handlers[WayPtr(SignalTo)->number-'0'-1]	13
			∅

3.12 Алгоритм метода exe класса App

Функционал: Метод запуска приложения.

Параметры: Отсутсвуют.

Возвращаемое значение: Целочисленное значение - код возврата.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода exe класса App

№	Предикат	Действия	№ перехода
1		Вывод на экран "Object tree"	2
2		Вывод на экран результата метода GetName текущего объекта	3
3		Вызов метода PrintTree с параметром ""	4
4	Поле wrong не пустая строка	Вывод на экран сообщения об ошиибочной координате	∅
			5
5		Вызов метода SetStatus с параметром 1, объявление строковых переменных function, object, переход на новую строку	6
6	Бесконечный цикл		7
			∅

№	Предикат	Действия	№ перехода
7		Считывание с клавиатуры значения переменной function	8
8	function равна "END"	Выход из цикла	∅
			9
9		Считывание с клавиатуры значения переменной object	10
10	function равна "EMIT"	Объявление строки message, считывание с клавиатуры ее значения	11
			12
11	Не удалось найти объект по координате	Вывод на экран сообщения об ошиибочной координате	6
		Вызов метода SetSignal по указателю ptr	6
12	function равна "DELETE_CONNECT"	Объявление строки object2, считывание с клавиатуры ее значения	13
			14
13	Не удалось найти объект по координате	Вывод на экран сообщения об ошиибочной координате	6
		Вызов метода DeleteConnections	6
14	function равна "SET_CONDITION"	Объявление целочисленной переменной status и считывание ее значения с клавиатуры, вызов метода SetStatus по указателю ptr	6
			15
15	function равна "SET_CONNECT"	Объявление строки object2, считывание с клавиатуры ее значения	16
			6
16	Не удалось найти объект по координате	Вывод на экран сообщения об ошиибочной координате	6
		Вызов метода SetConnections	6

3.13 Алгоритм конструктора класса Class2

Функционал: Делегирующий конструктор.

Параметры: Указатель на объект типа Base, строка - name.

Алгоритм конструктора представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса Class2

№	Предикат	Действия	№ перехода
1		Присвоение полю number символа '2'	∅

3.14 Алгоритм метода Signal класса Class2

Функционал: Метод сигнала.

Параметры: Строка - name.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода Signal класса Class2

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением ей результата выполнения метода BuildPath с параметром this	2
2	Переменная temp не равно /	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала от объекта по координате temp	∅

3.15 Алгоритм метода Handler класса Class2

Функционал: Обработчик сигнала.

Параметры: Строка - message.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода Handler класса Class2

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением итога BuildPath с параметром this	2
2	Переменная temp не равна '/'	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала в объект по координате temp с текстом message от объекта класса ptr->class0fsignal	∅

3.16 Алгоритм конструктора класса Class3

Функционал: Делегирующий конструктор.

Параметры: Указатель на объект типа Base, строка - name.

Алгоритм конструктора представлен в таблице 17.

Таблица 17 – Алгоритм конструктора класса Class3

№	Предикат	Действия	№ перехода
1		Присвоение полю number символа '3'	∅

3.17 Алгоритм метода Signal класса Class3

Функционал: Метод сигнала.

Параметры: Строка - name.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода Signal класса Class3

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением ей результата выполнения метода BuildPath с параметром this	2
2	Переменная temp не равно /	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала от объекта по координате temp	∅

3.18 Алгоритм метода Handler класса Class3

Функционал: Обработчик сигнала.

Параметры: Строка - message.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода Handler класса Class3

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением итога BuildPath с параметром this	2
2	Переменная temp не равна '/'	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала в объект по координате temp с текстом message от объекта класса ptr->class0fsignal	∅

3.19 Алгоритм конструктора класса Class4

Функционал: Делегирующий конструктор.

Параметры: Указатель на объект типа Base, строка - name.

Алгоритм конструктора представлен в таблице 20.

Таблица 20 – Алгоритм конструктора класса Class4

№	Предикат	Действия	№ перехода
1		Присвоение полю number символа '4'	∅

3.20 Алгоритм метода Signal класса Class4

Функционал: Метод сигнала.

Параметры: Строка - name.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода Signal класса Class4

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением ей результата выполнения метода BuildPath с параметром this	2
2	Переменная temp не равно /	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала от объекта по координате temp	∅

3.21 Алгоритм метода Handler класса Class4

Функционал: Обработчик сигнала.

Параметры: Строка - message.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода Handler класса Class4

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением итога BuildPath с параметром this	2
2	Переменная temp не равна '/'	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала в объект по координате temp с текстом message от объекта класса ptr->class0fsignal	∅

3.22 Алгоритм конструктора класса Class5

Функционал: Делегирующий конструктор.

Параметры: Указатель на объект типа Base, строка - name.

Алгоритм конструктора представлен в таблице 23.

Таблица 23 – Алгоритм конструктора класса Class5

№	Предикат	Действия	№ перехода
1		Присвоение полю number символа '5'	∅

3.23 Алгоритм метода Signal класса Class5

Функционал: Метод сигнала.

Параметры: Строка - name.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода Signal класса Class5

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением ей результата выполнения метода BuildPath с параметром this	2
2	Переменная temp не равно /	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала от объекта по координате temp	∅

3.24 Алгоритм метода Handler класса Class5

Функционал: Обработчик сигнала.

Параметры: Строка - message.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 25.

Таблица 25 – Алгоритм метода Handler класса Class5

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением итога BuildPath с параметром this	2
2	Переменная temp не равна '/'	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала в объект по координате temp с текстом message от объекта класса ptr->class0fsignal	∅

3.25 Алгоритм конструктора класса Class6

Функционал: Делегирующий конструктор.

Параметры: Указатель на объект типа Base, строка - name.

Алгоритм конструктора представлен в таблице 26.

Таблица 26 – Алгоритм конструктора класса Class6

№	Предикат	Действия	№ перехода
1		Присвоение полю number символа '6'	∅

3.26 Алгоритм метода Signal класса Class6

Функционал: Метод сигнала.

Параметры: Строка - name.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода Signal класса Class6

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением ей результата выполнения метода BuildPath с параметром this	2
2	Переменная temp не равно /	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала от объекта по координате temp	∅

3.27 Алгоритм метода Handler класса Class6

Функционал: Обработчик сигнала.

Параметры: Строка - message.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода Handler класса Class6

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной temp с присвоением итога BuildPath с параметром this	2
2	Переменная temp не равна '/'	Удаление из переменной temp последнего символа '/'	3
			3
3		Вывод на экран информации о поступлении сигнала в объект по координате temp с текстом message от объекта класса ptr->class0fsignal	∅

3.28 Алгоритм метода PrintTree класса Base

Функционал: Вывод дерева иерархии.

Параметры: Строка space - количество пробелов.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 29.

Таблица 29 – Алгоритм метода PrintTree класса Base

№	Предикат	Действия	№ перехода
1	Цикл со счетчиком i		2
			5
2	Элемент в векторе детей также имеет своих детей		3
		Вывод на экран space+4 пробела, вывод итога выполнения метода GetName у элемента индекса i,	5

№	Предикат	Действия	№ перехода
		переход на следующую строку	
3		Вывод на экран space+4 пробела, вывод итога выполнения метода GetName у элемента индекса i, переход на следующую строку	4
4		Вызов метода PrintTree у элемента индекса i	5
5	i меньше размера вектора	Увеличение i на 1	1
			∅

3.29 Алгоритм метода SetName класса Base

Функционал: Сеттер поля name.

Параметры: Строка - имя объекта.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 30.

Таблица 30 – Алгоритм метода SetName класса Base

№	Предикат	Действия	№ перехода
1		Присвоение полю name значения передаваемого параметра name	∅

3.30 Алгоритм метода GetName класса Base

Функционал: Геттер поля name.

Параметры: Отсутствуют.

Возвращаемое значение: Строка - имя объекта.

Алгоритм метода представлен в таблице 31.

Таблица 31 – Алгоритм метода GetName класса Base

№	Предикат	Действия	№ перехода
1		Возврат поля name	∅

3.31 Алгоритм конструктора класса App

Функционал: Делегирующий конструктор.

Параметры: Указатель на объект типа Base, строка - name.

Алгоритм конструктора представлен в таблице 32.

Таблица 32 – Алгоритм конструктора класса App

№	Предикат	Действия	№ перехода
1		Передача параметров head, name в конструктор Base	∅

3.32 Алгоритм метода FindPtr класса Base

Функционал: поиск объекта по имени.

Параметры: строка - имя, указатель на объект класса Base, булева переменная f.

Возвращаемое значение: указатель на класс Base.

Алгоритм метода представлен в таблице 33.

Таблица 33 – Алгоритм метода FindPtr класса Base

№	Предикат	Действия	№ перехода
1	ptr не нулевой указатель		2
		Возврат нулевого указателя	∅
2	Результат выполнения метода GetName объекта ptr равно name	Возврат указателя ptr	∅
			3
3	Цикл со счетчиком i		4
			∅
4	Результат выполнения метода GetName объекта ptr-	Возврат указателя ptr->children[i]	∅

№	Предикат	Действия	№ перехода
	>children[i] равно name		
	У ptr->children[i] есть дочерние объекты и f равно false	Возврат значения метода FindPtr с параметрами name, ptr->children[i]	∅
			3

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-32.

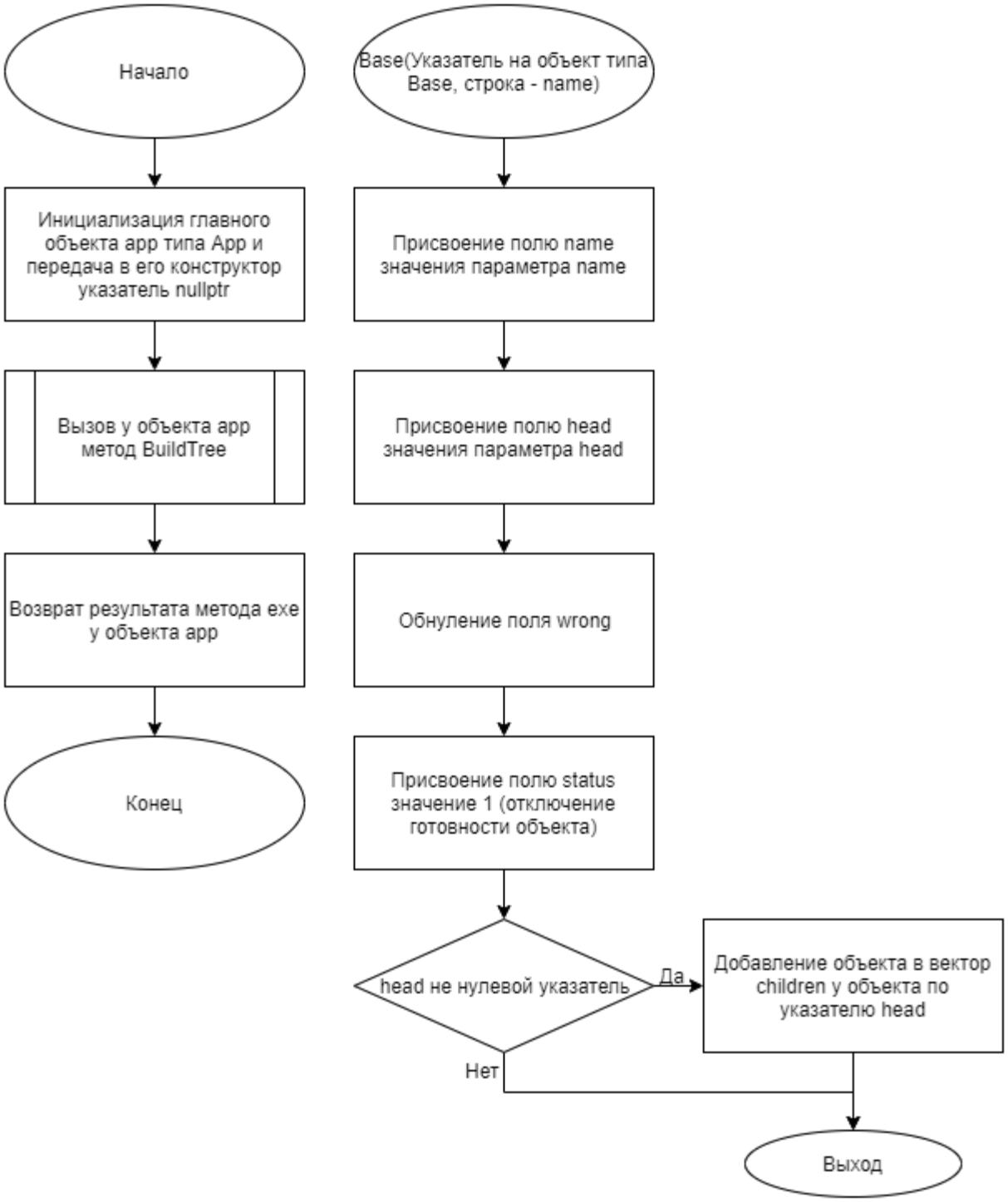

Рисунок 1 – Блок-схема алгоритма

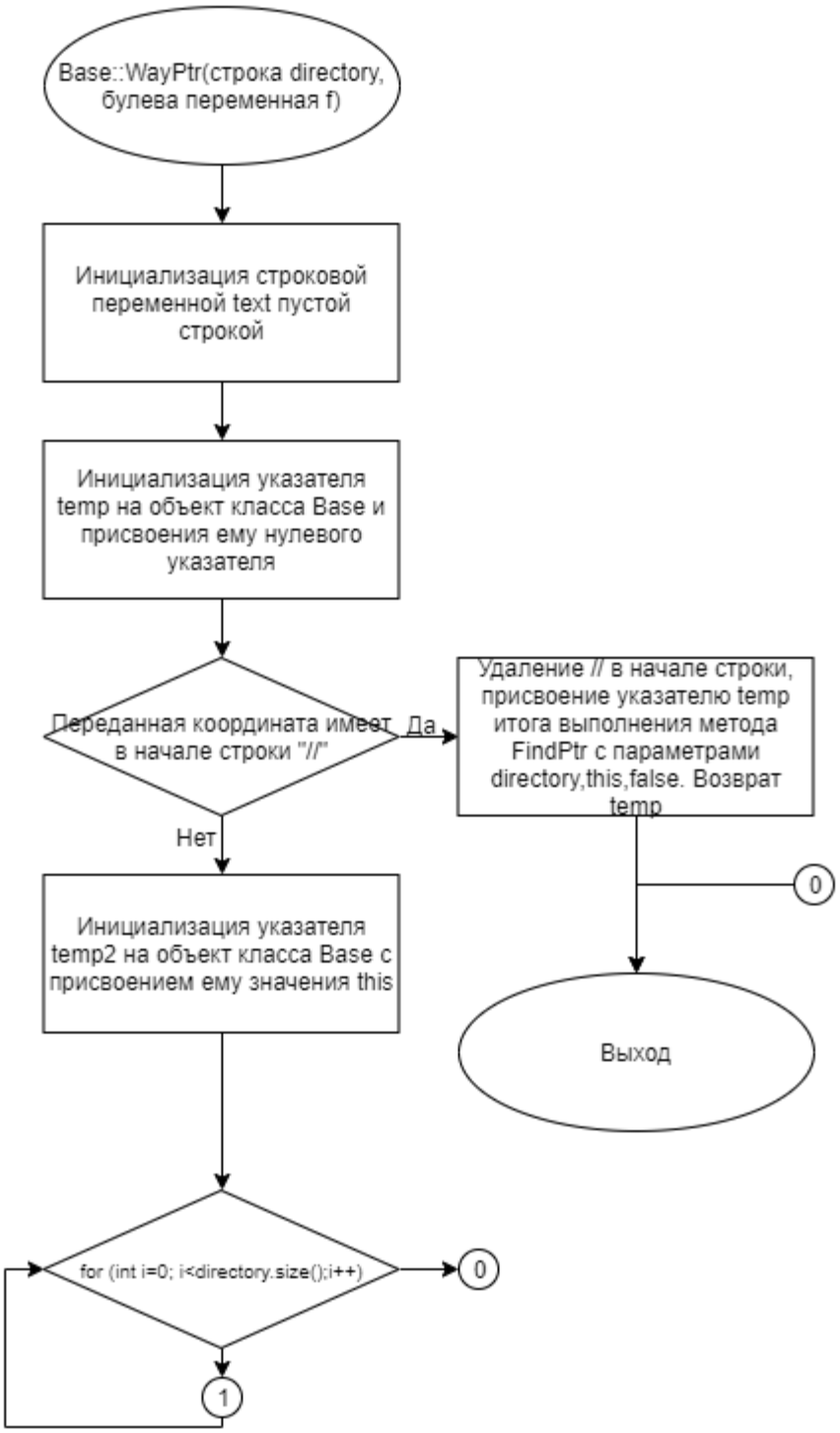

Рисунок 2 – Блок-схема алгоритма

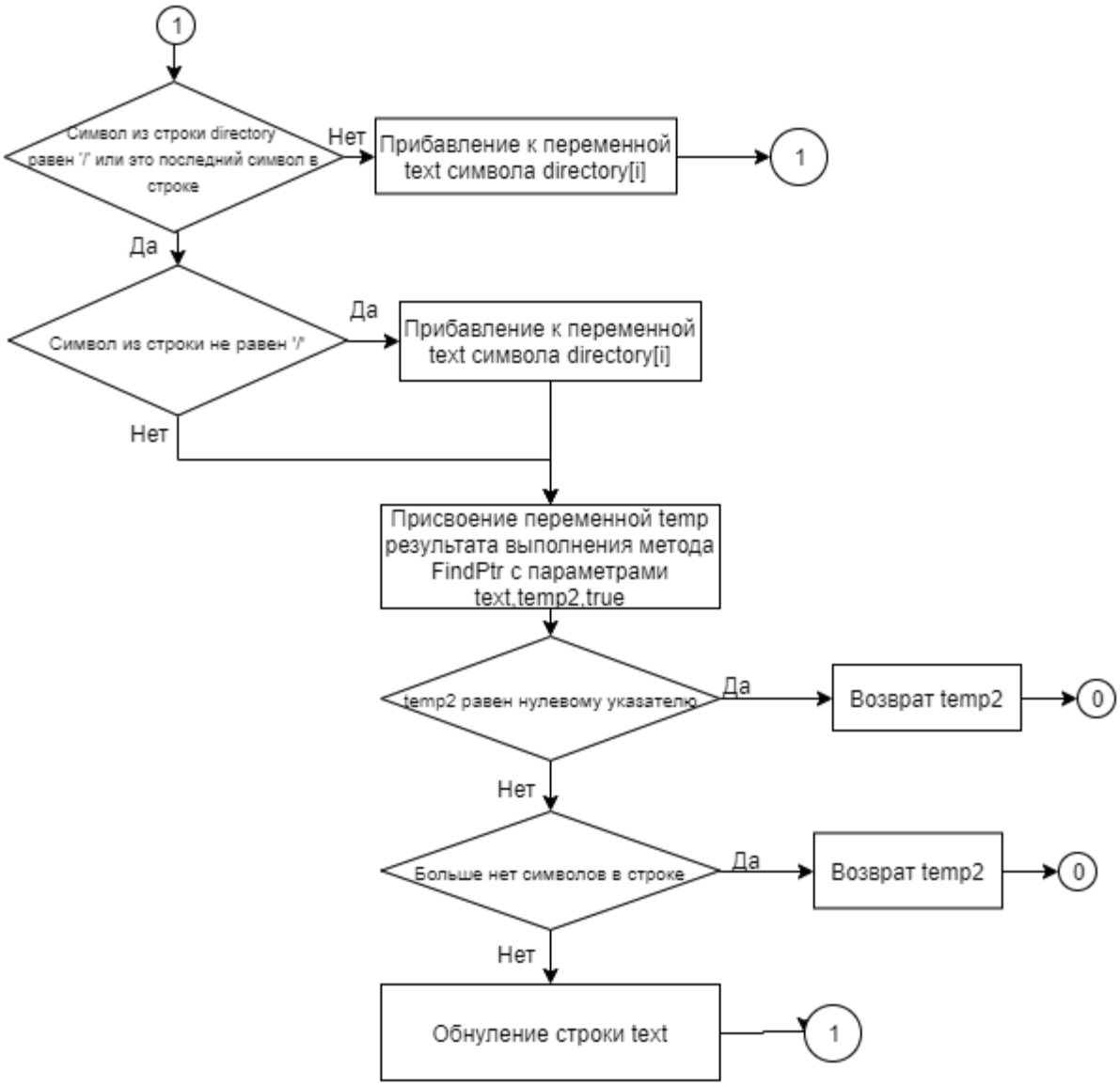

Рисунок 3 – Блок-схема алгоритма

Рисунок 4 – Блок-схема алгоритма

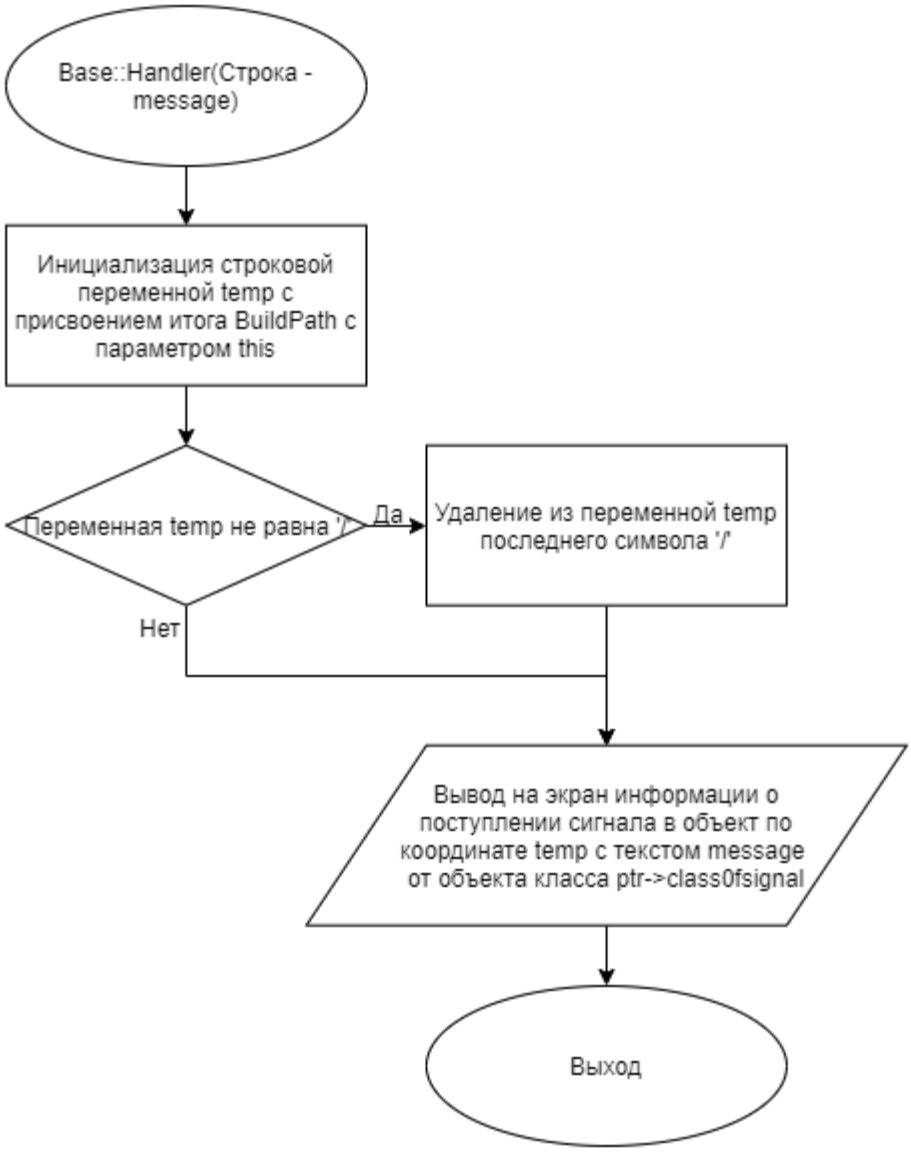

Рисунок 5 – Блок-схема алгоритма

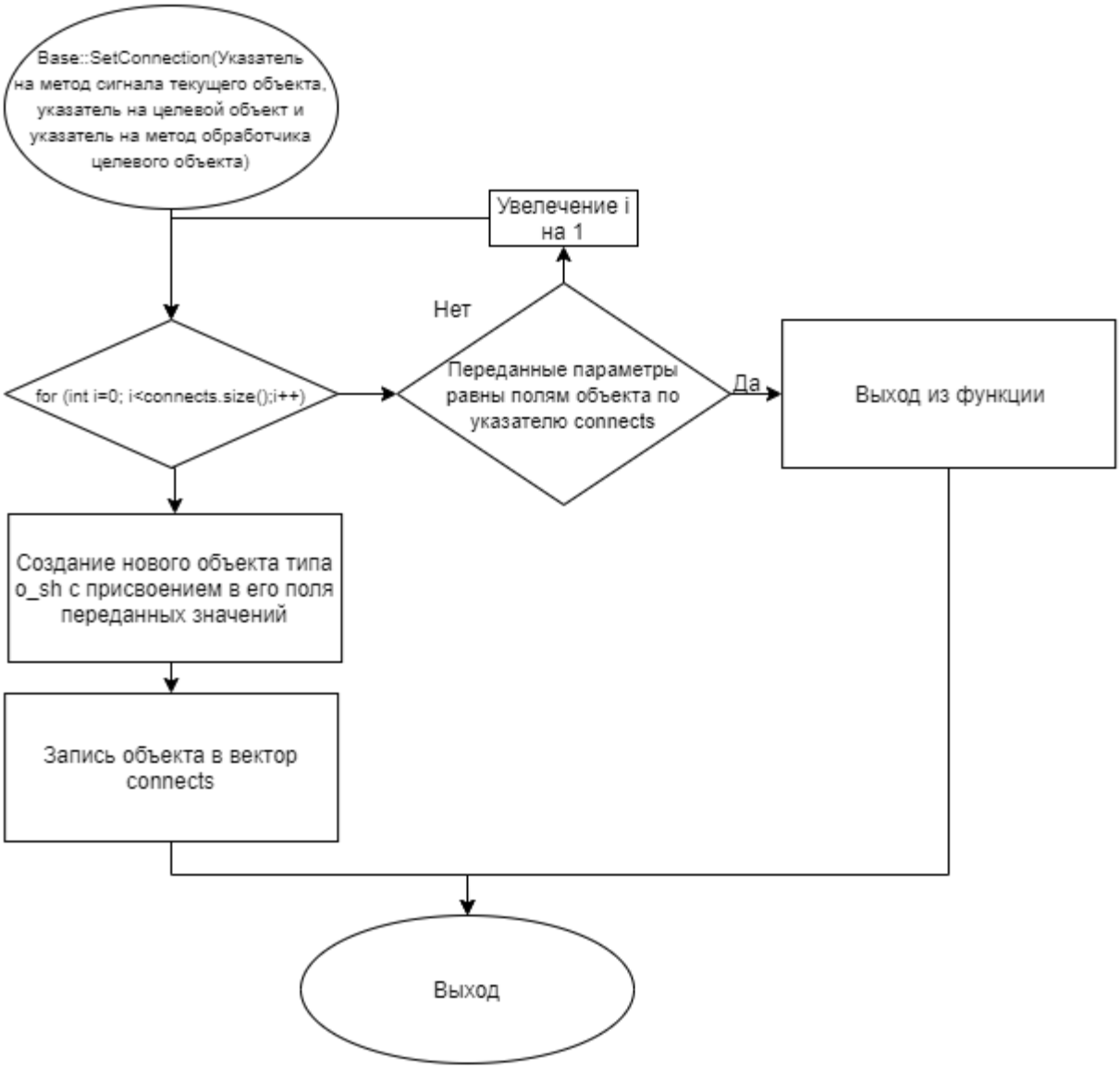

Рисунок 6 – Блок-схема алгоритма

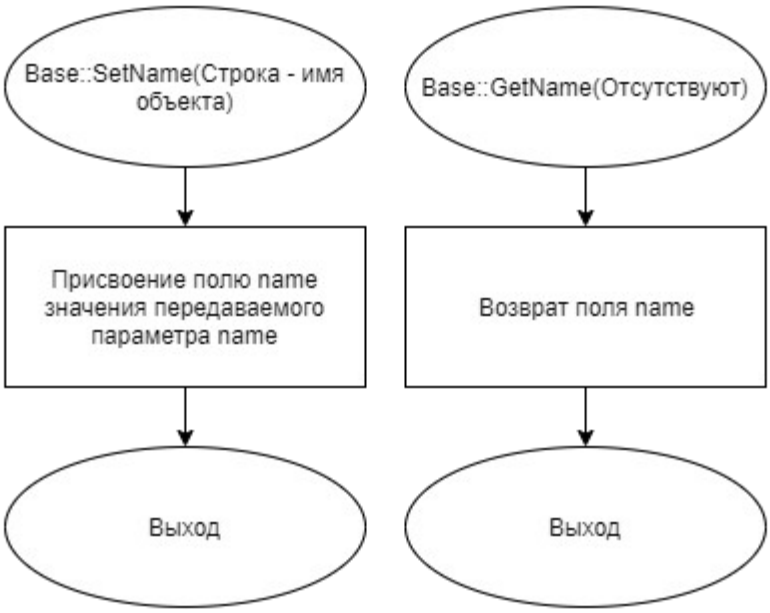

Рисунок 7 – Блок-схема алгоритма

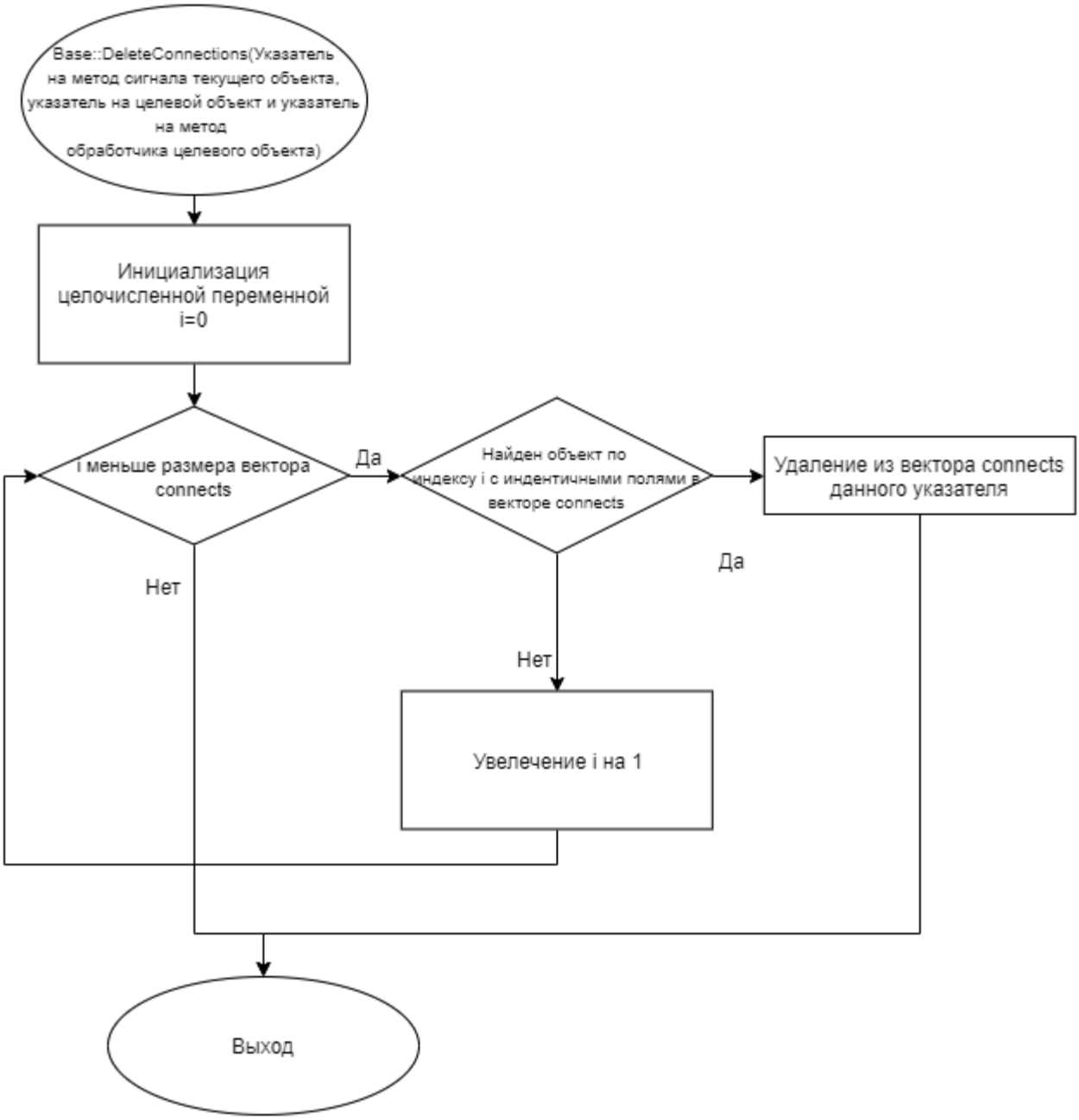

Рисунок 8 – Блок-схема алгоритма

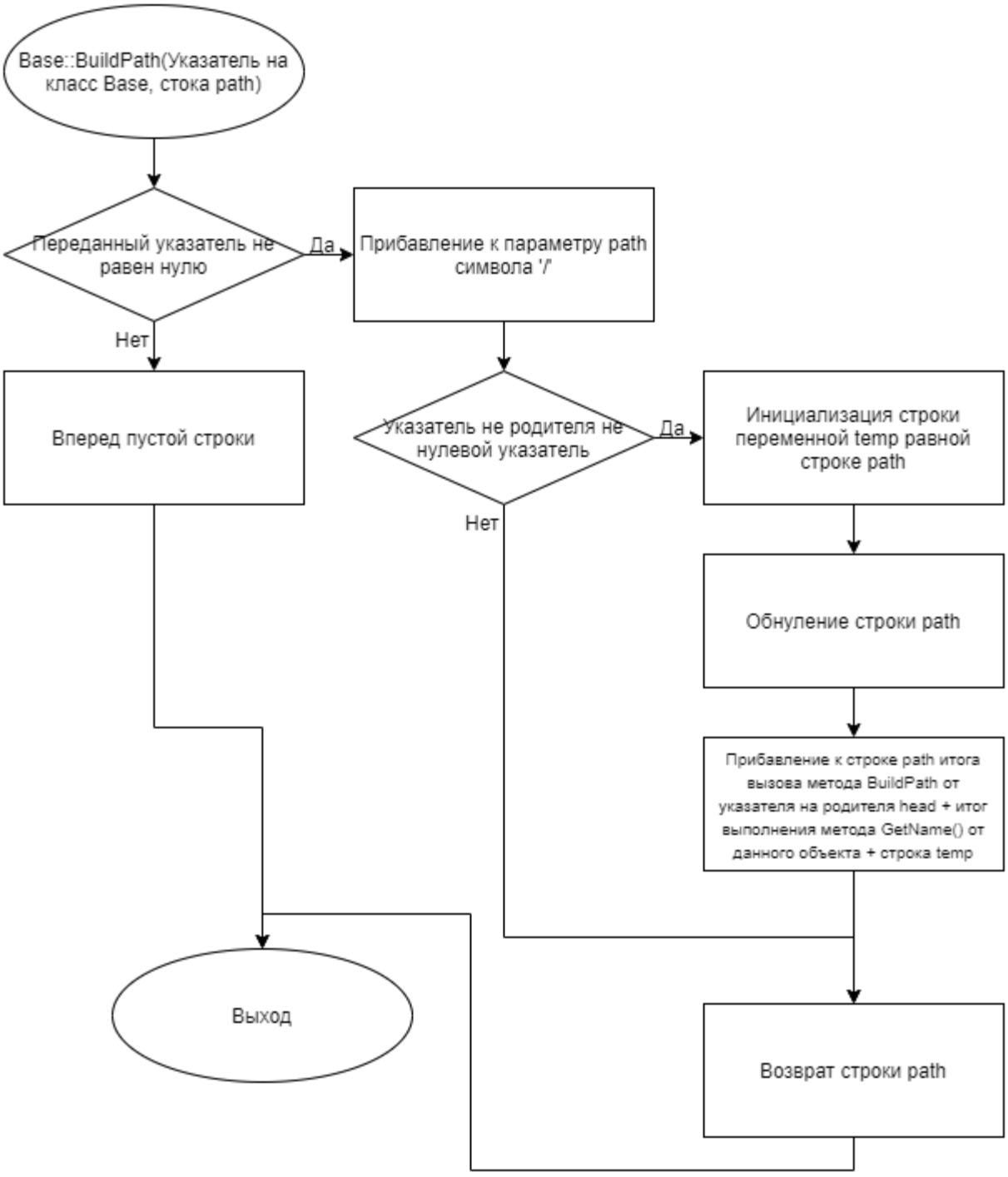

Рисунок 9 – Блок-схема алгоритма

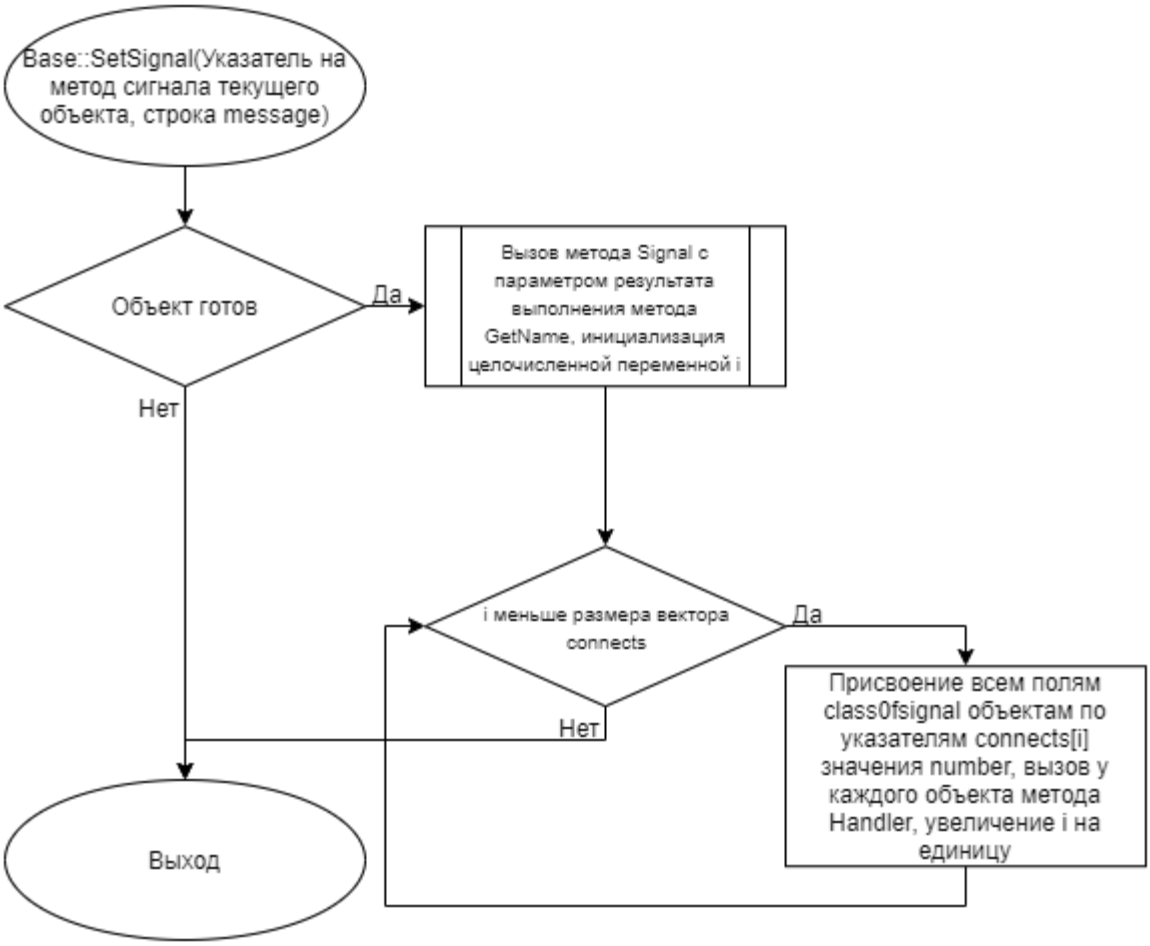

Рисунок 10 – Блок-схема алгоритма

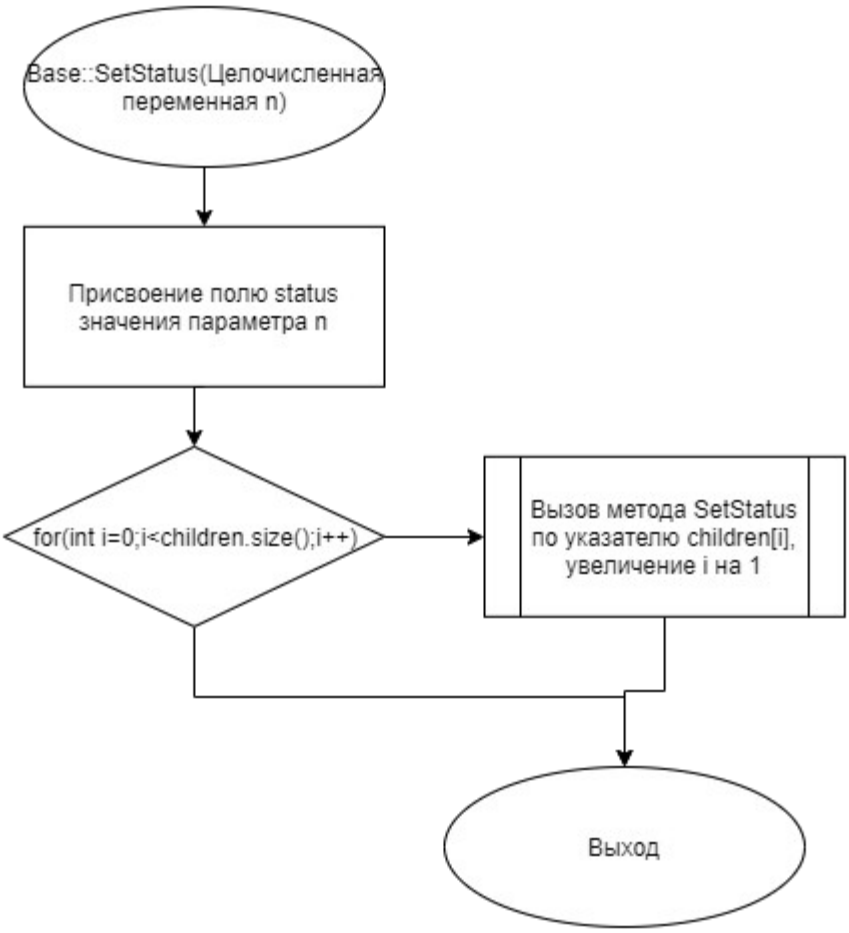

Рисунок 11 – Блок-схема алгоритма

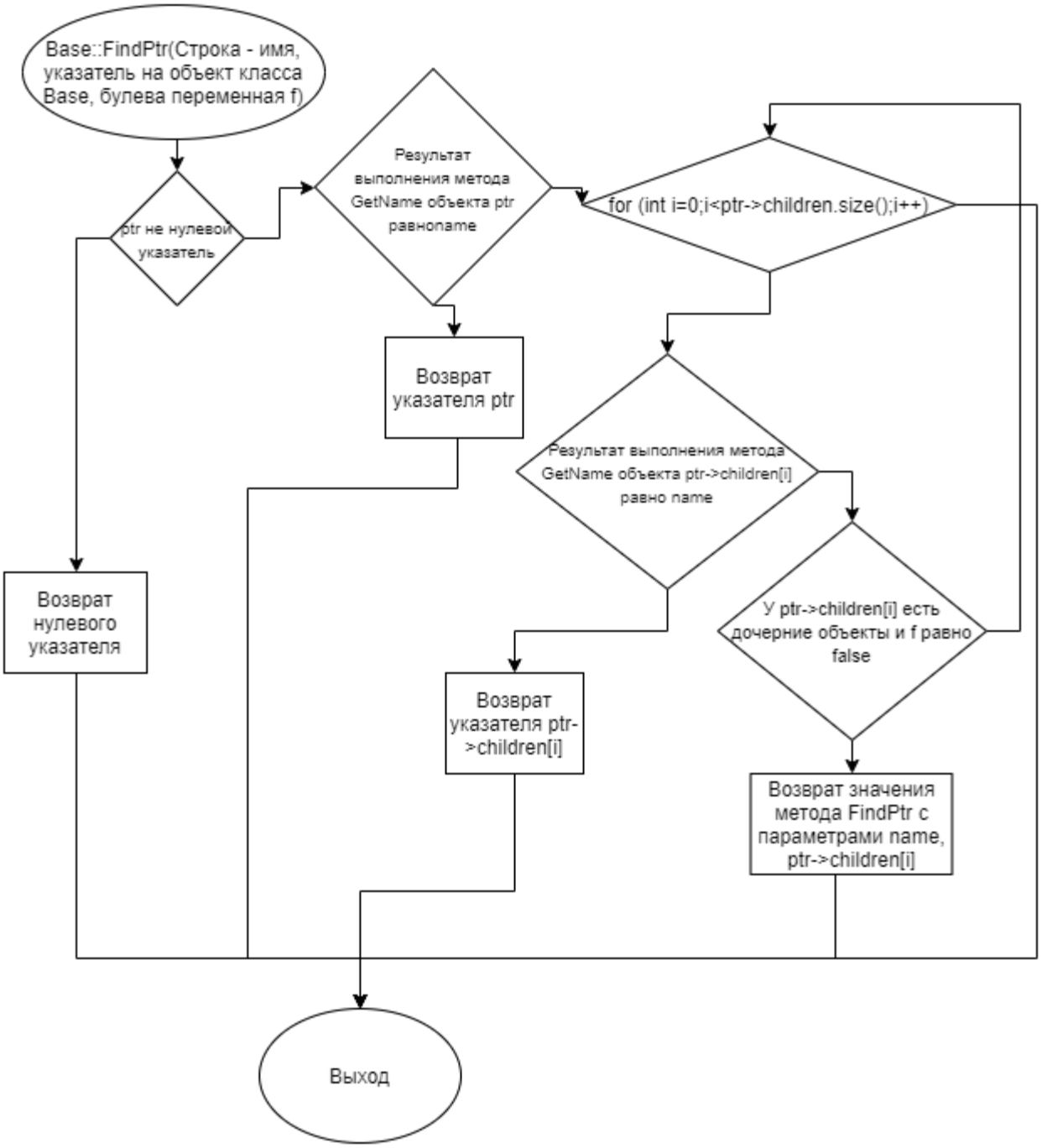

Рисунок 12 – Блок-схема алгоритма

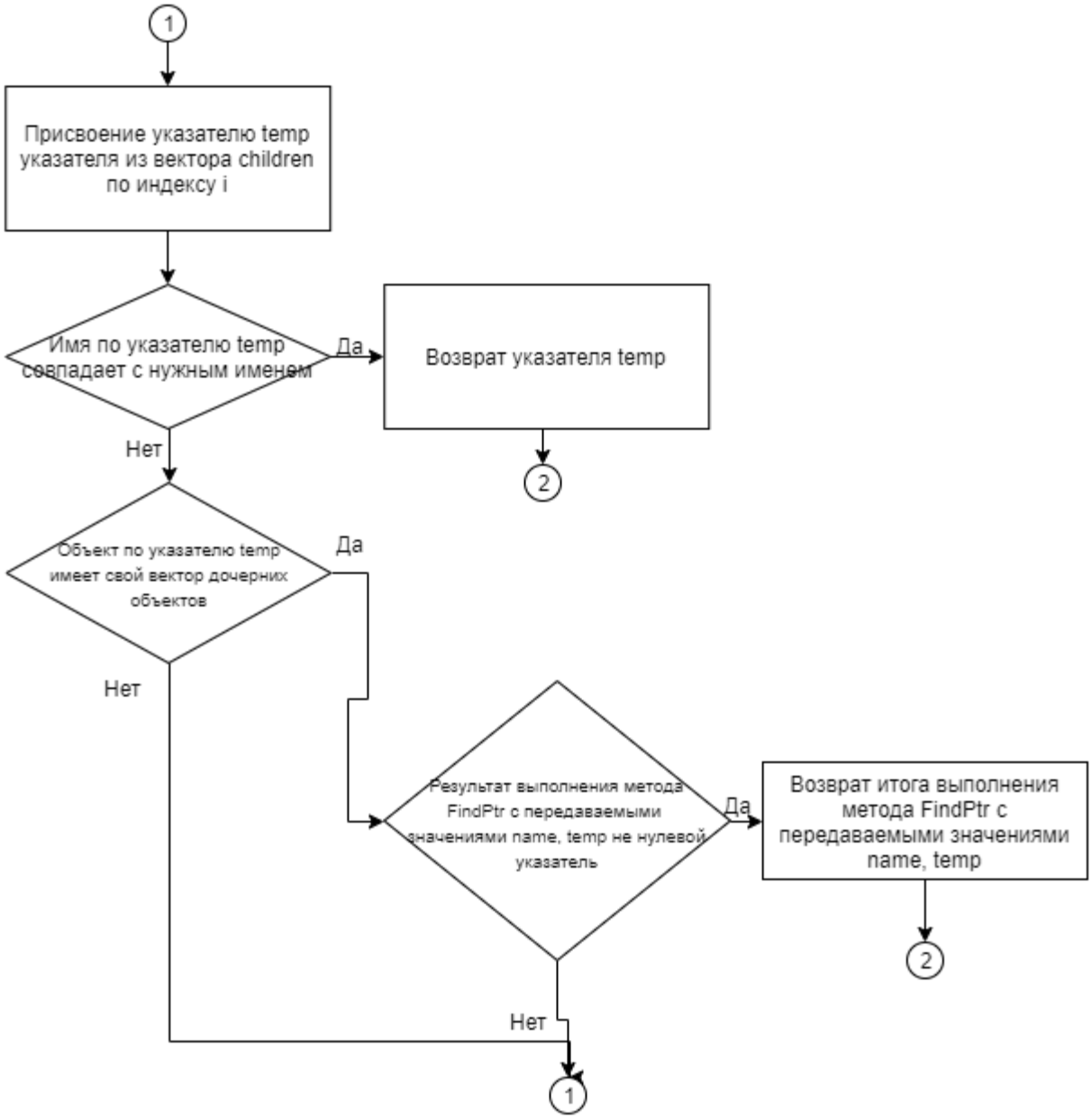

Рисунок 13 – Блок-схема алгоритма

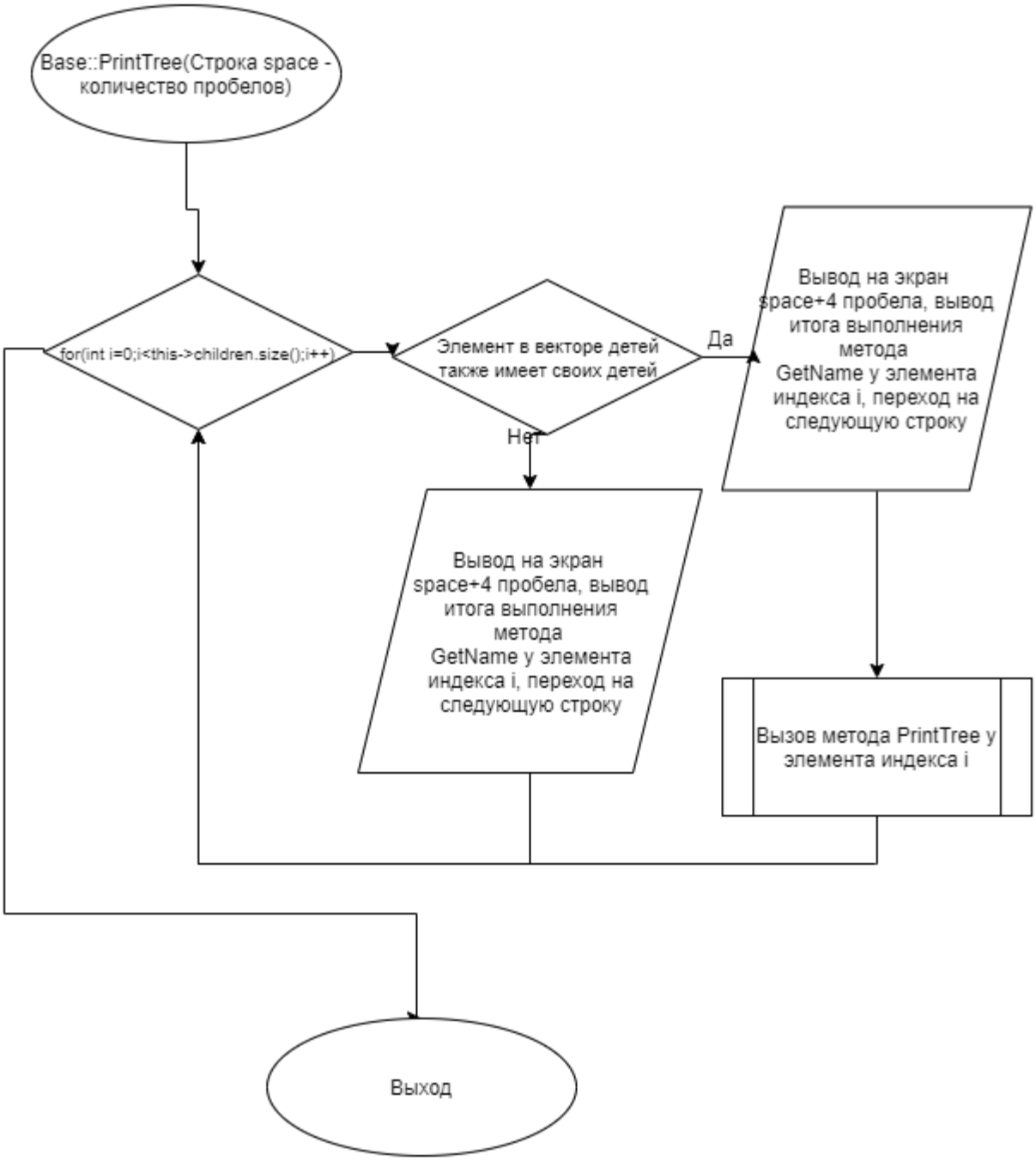

Рисунок 14 – Блок-схема алгоритма

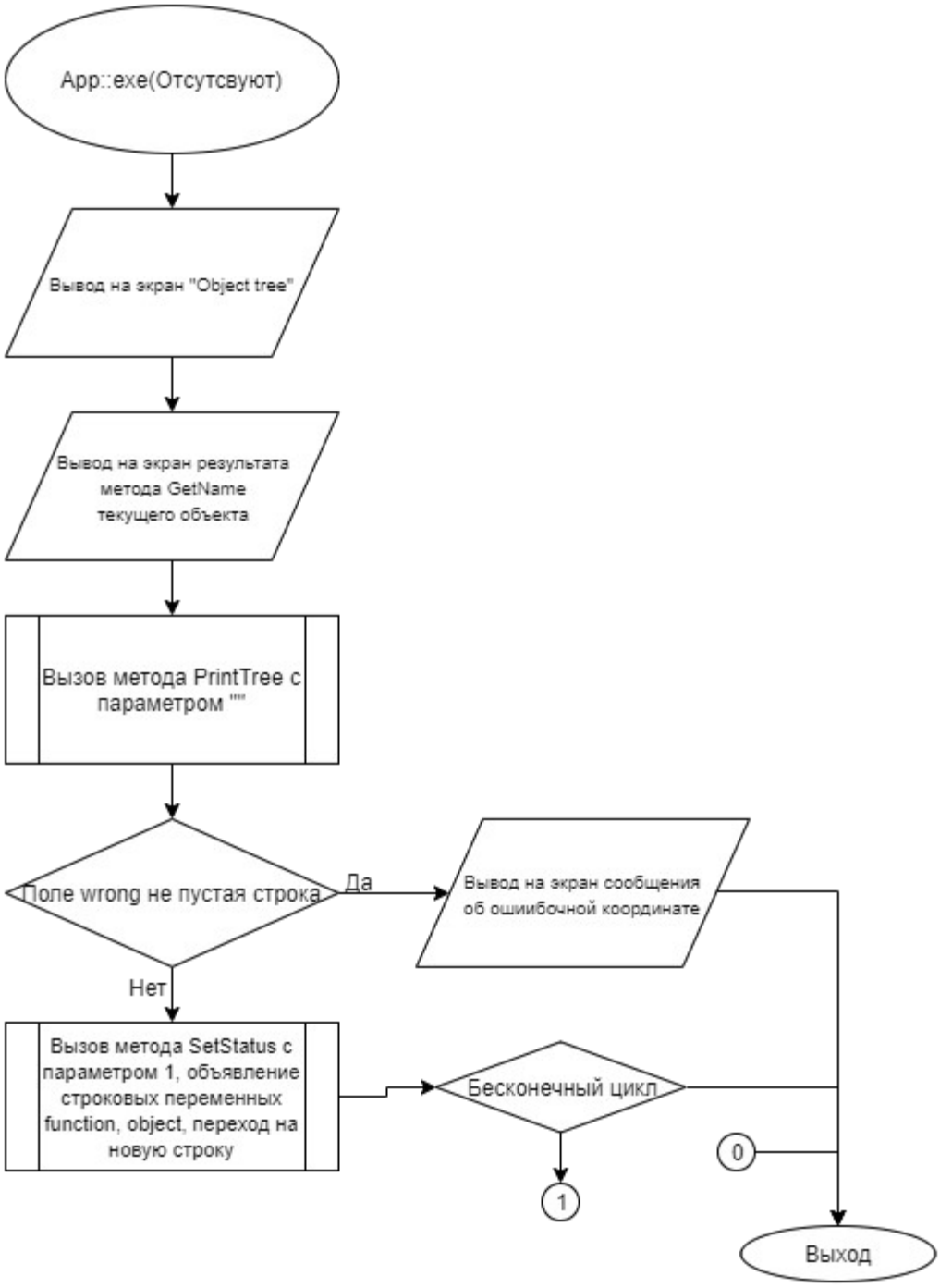

Рисунок 15 – Блок-схема алгоритма

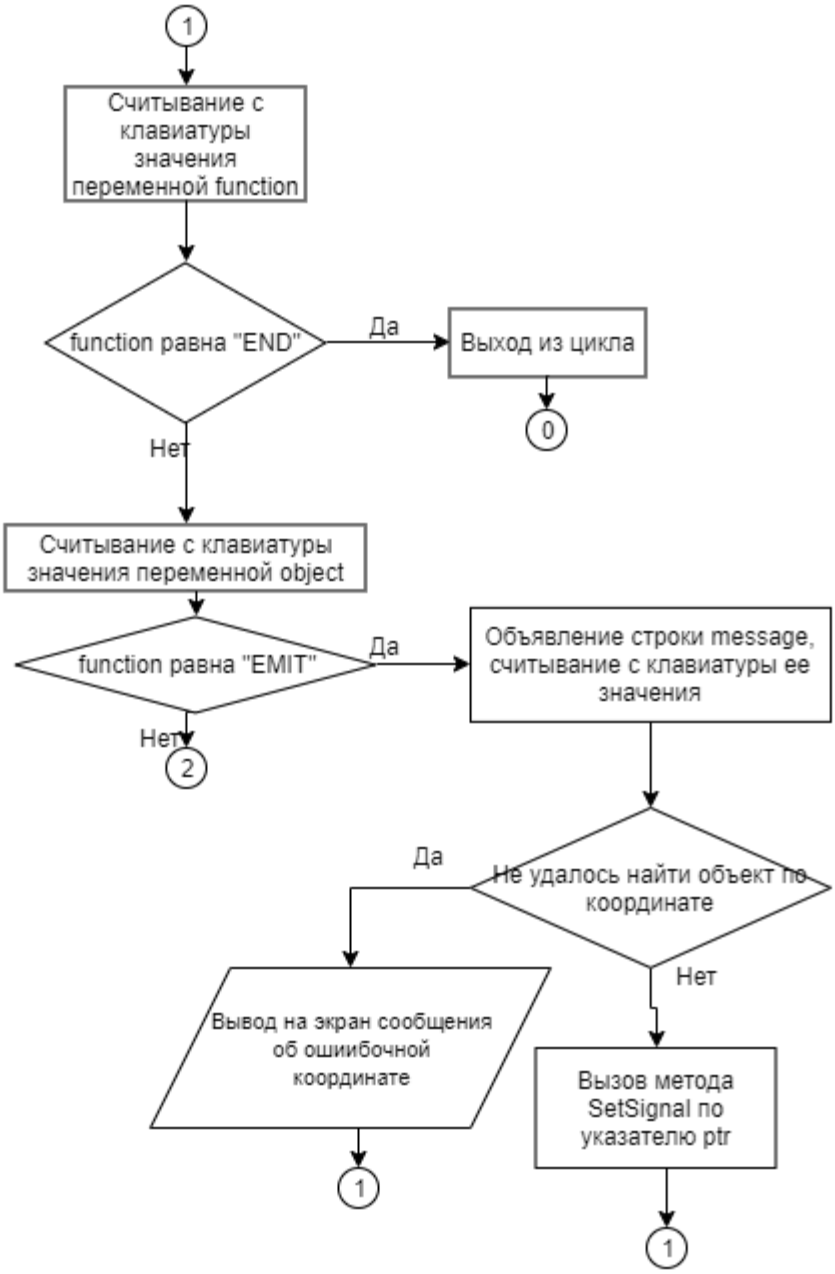

Рисунок 16 – Блок-схема алгоритма

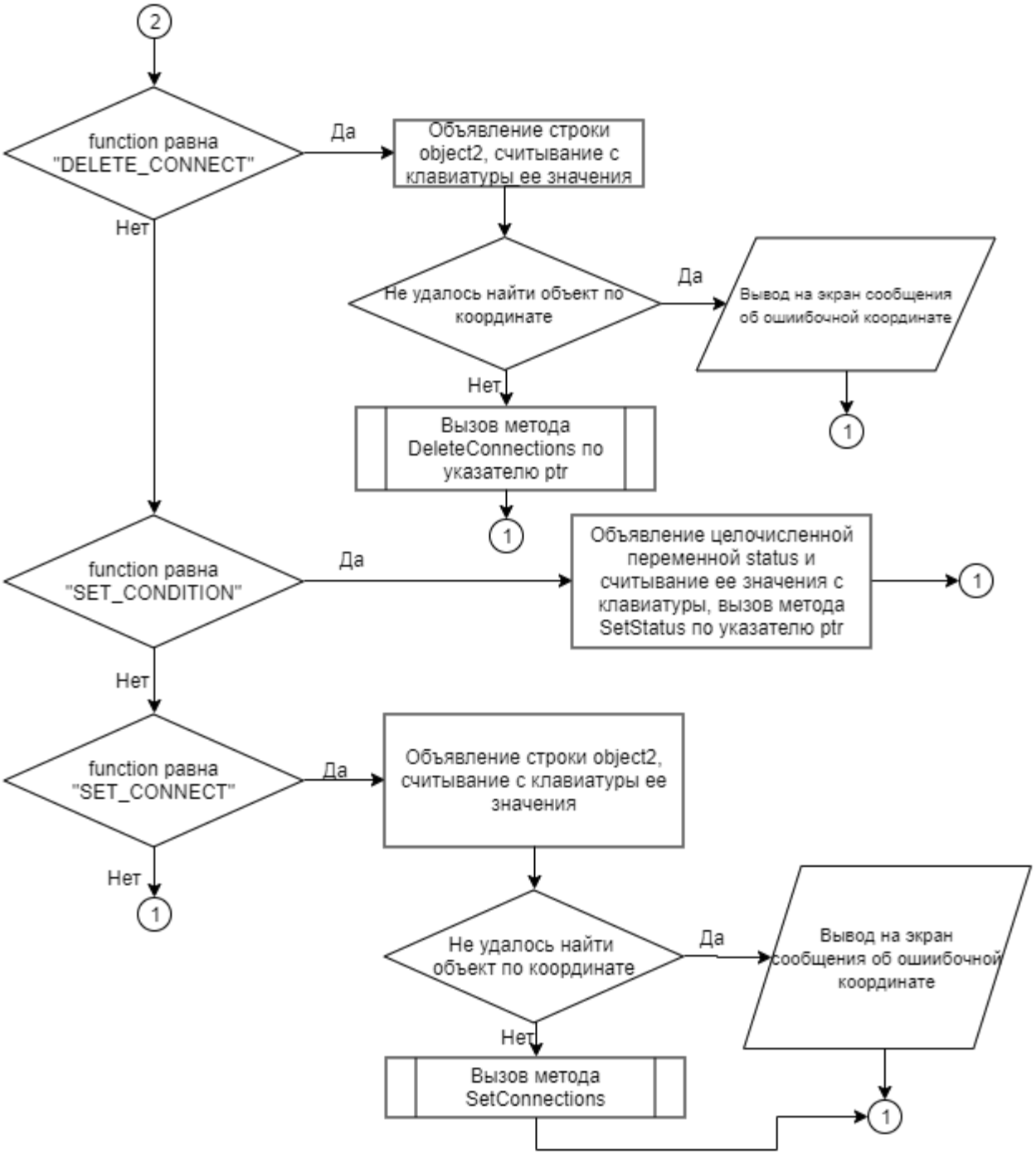

Рисунок 17 – Блок-схема алгоритма

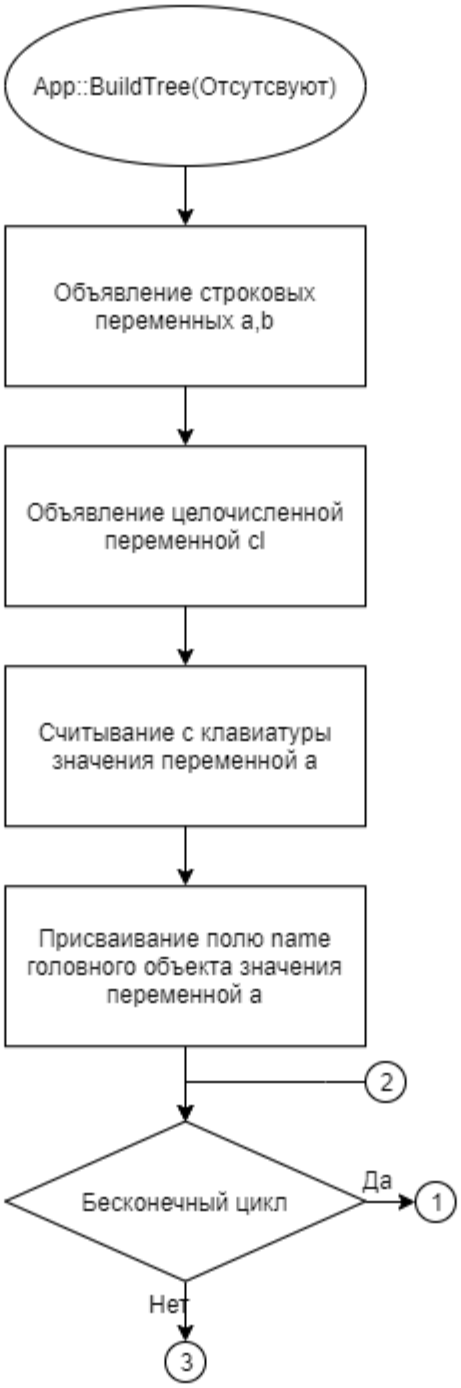

Рисунок 18 – Блок-схема алгоритма

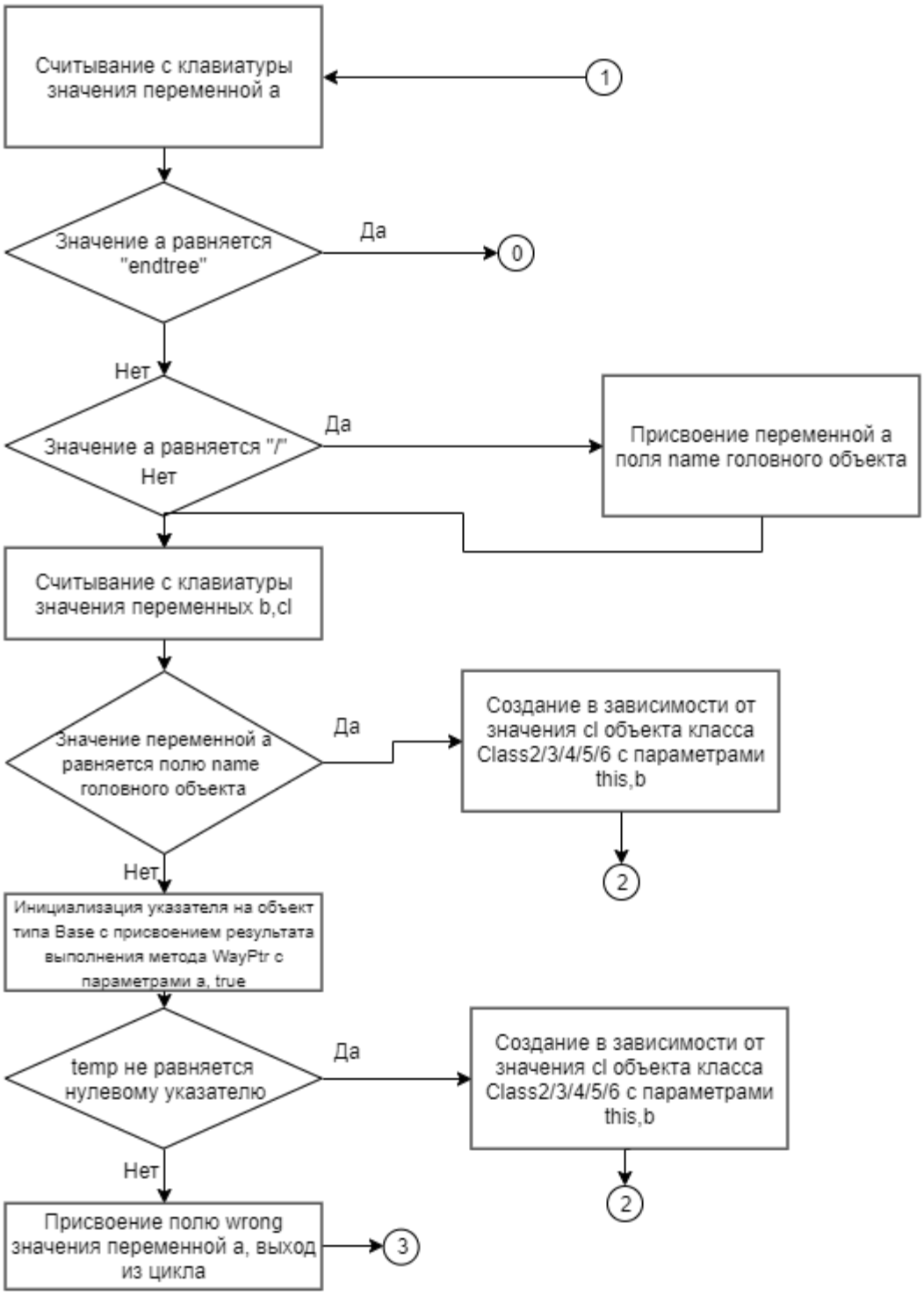

Рисунок 19 – Блок-схема алгоритма

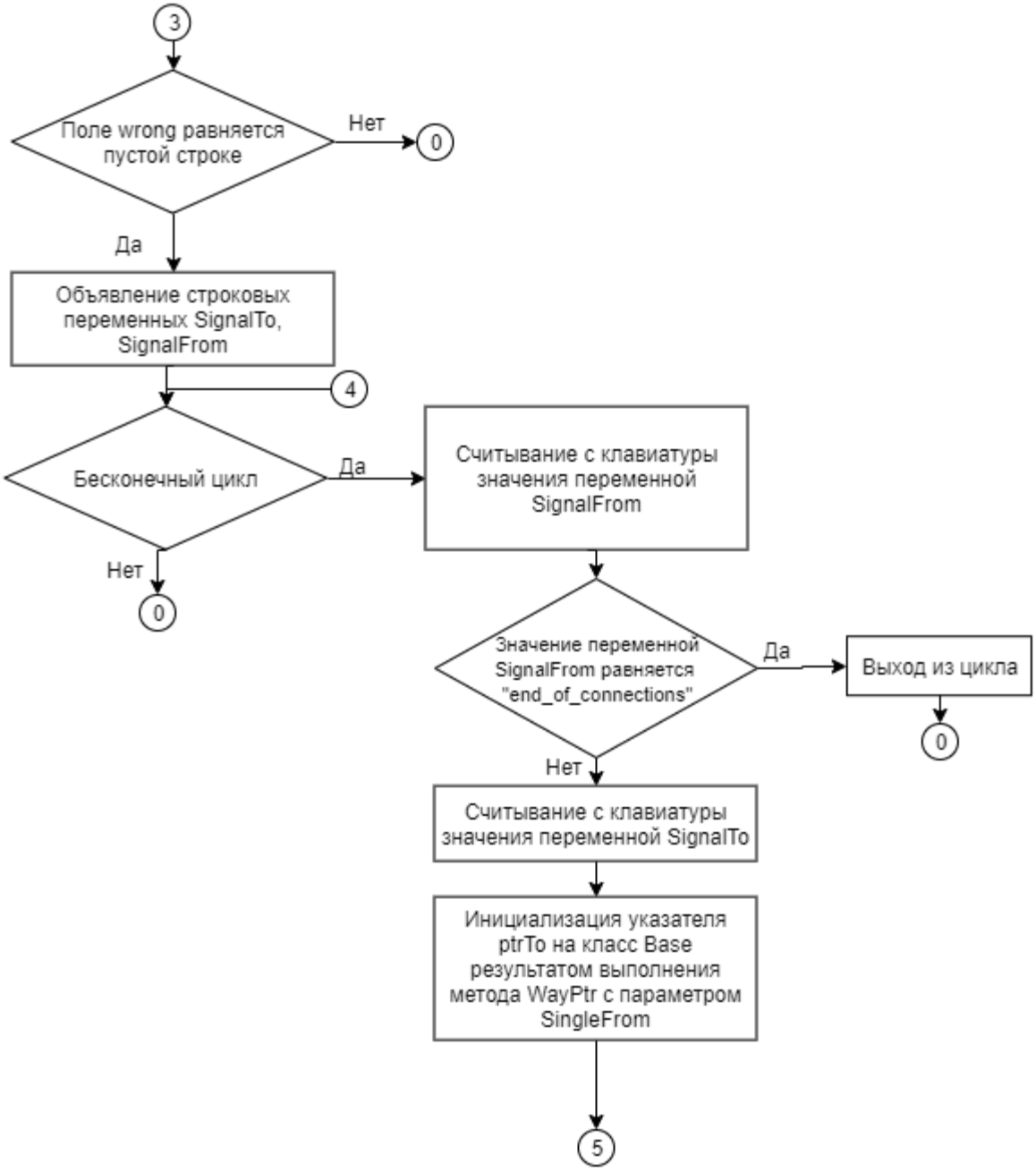

Рисунок 20 – Блок-схема алгоритма

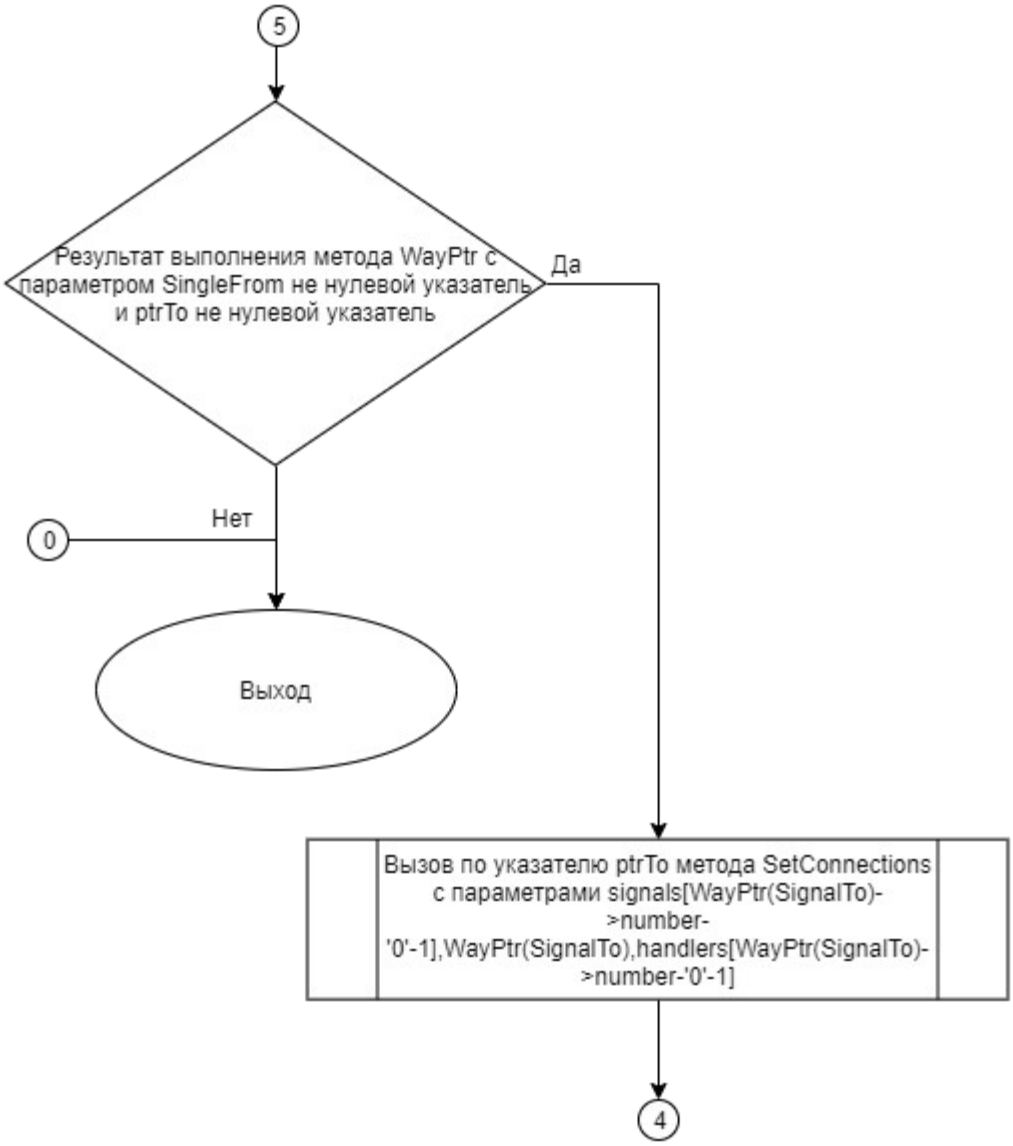

Рисунок 21 – Блок-схема алгоритма

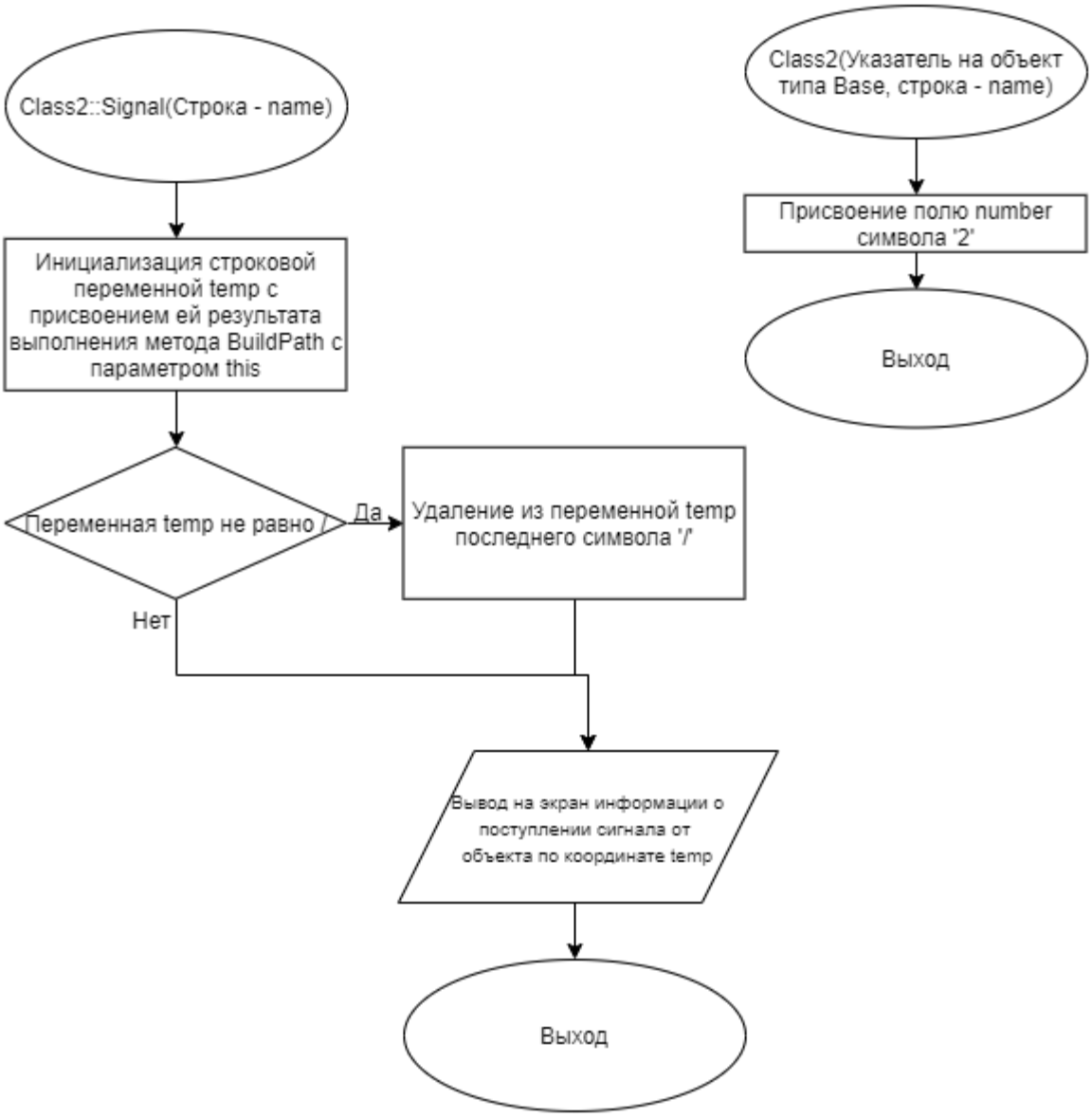

Рисунок 22 – Блок-схема алгоритма

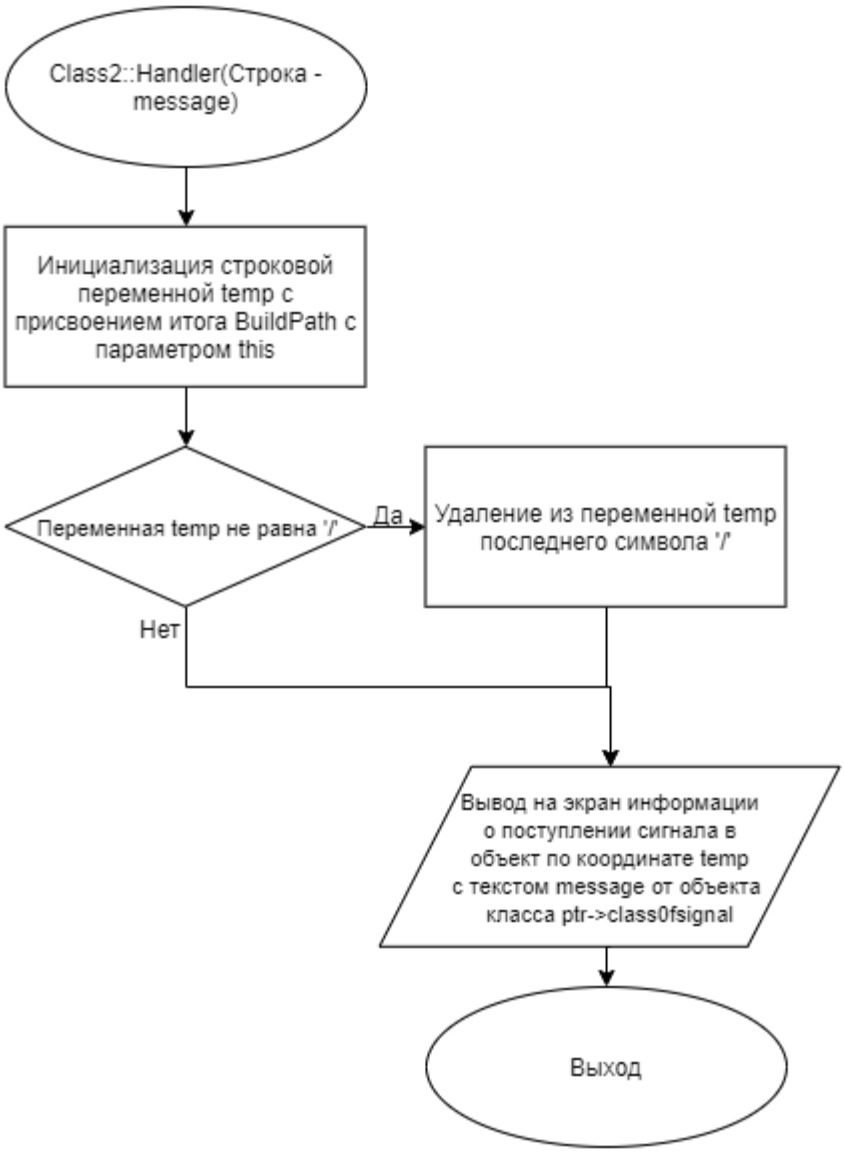

Рисунок 23 – Блок-схема алгоритма

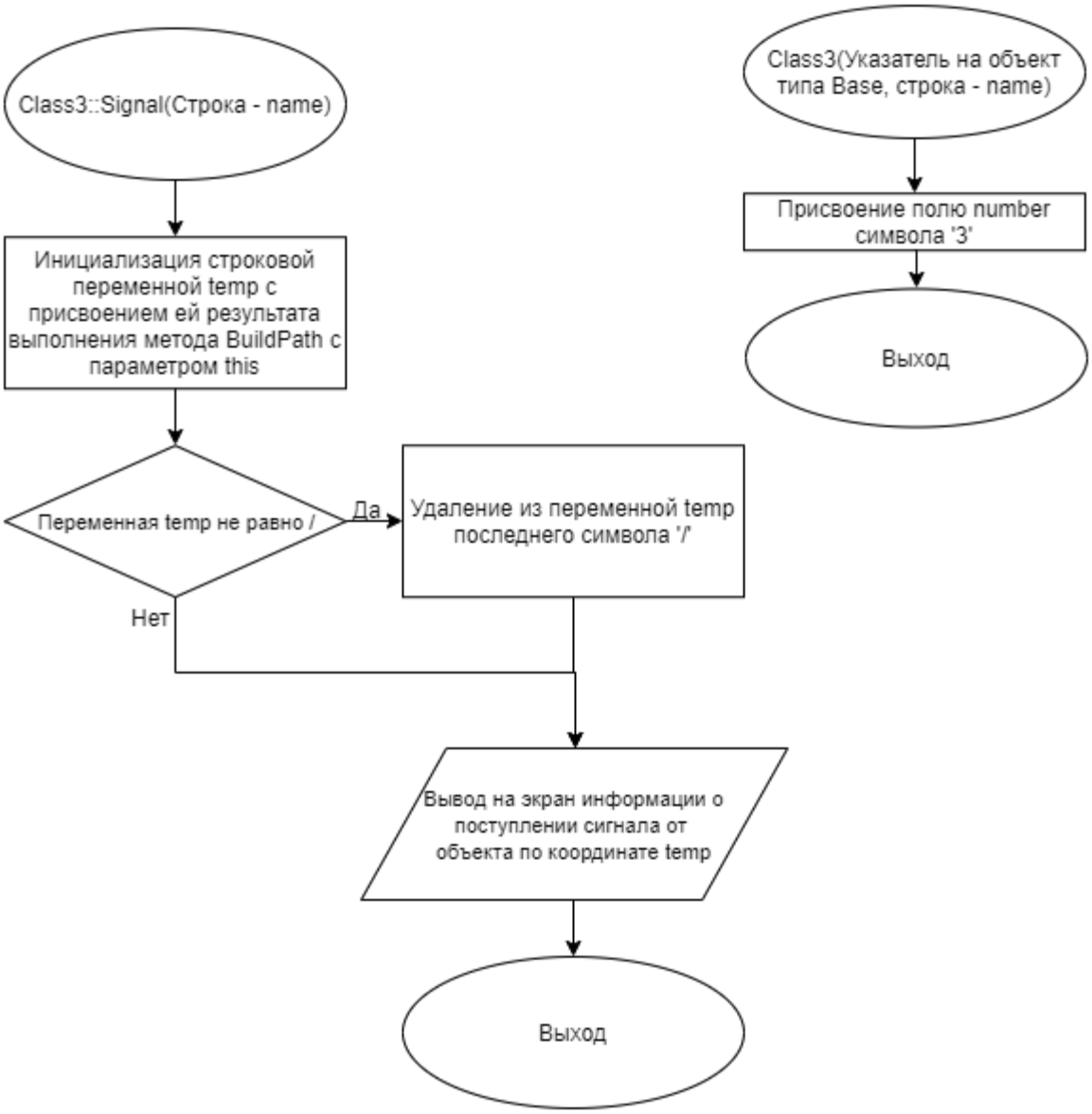

Рисунок 24 – Блок-схема алгоритма

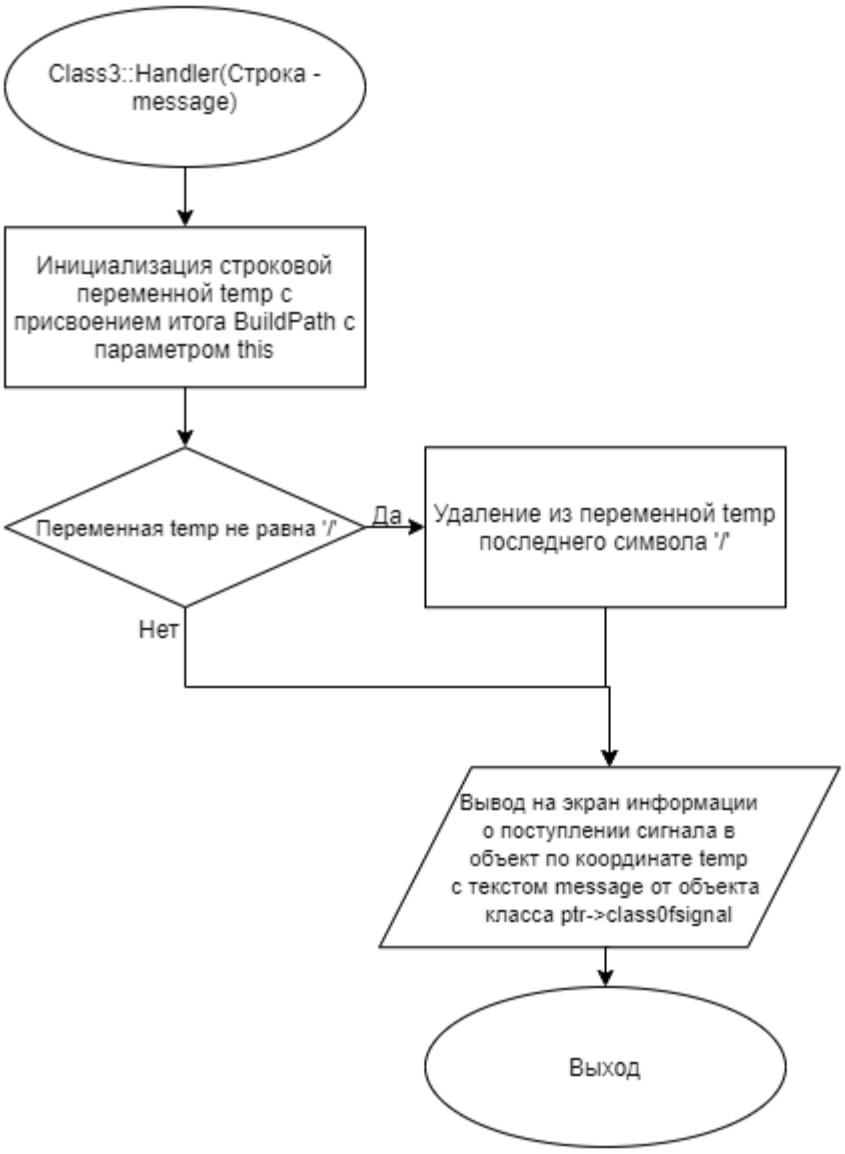

Рисунок 25 – Блок-схема алгоритма

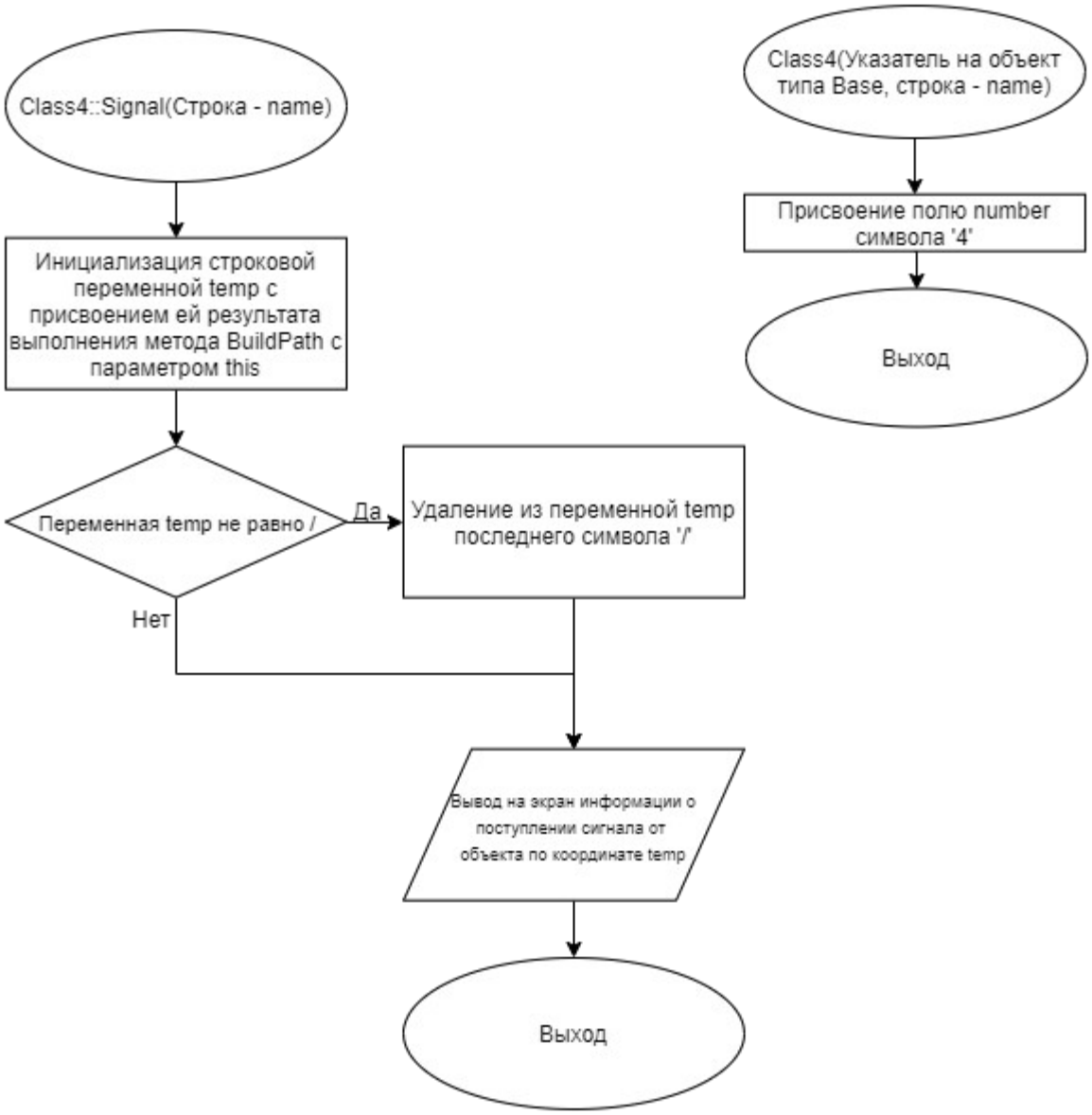

Рисунок 26 – Блок-схема алгоритма

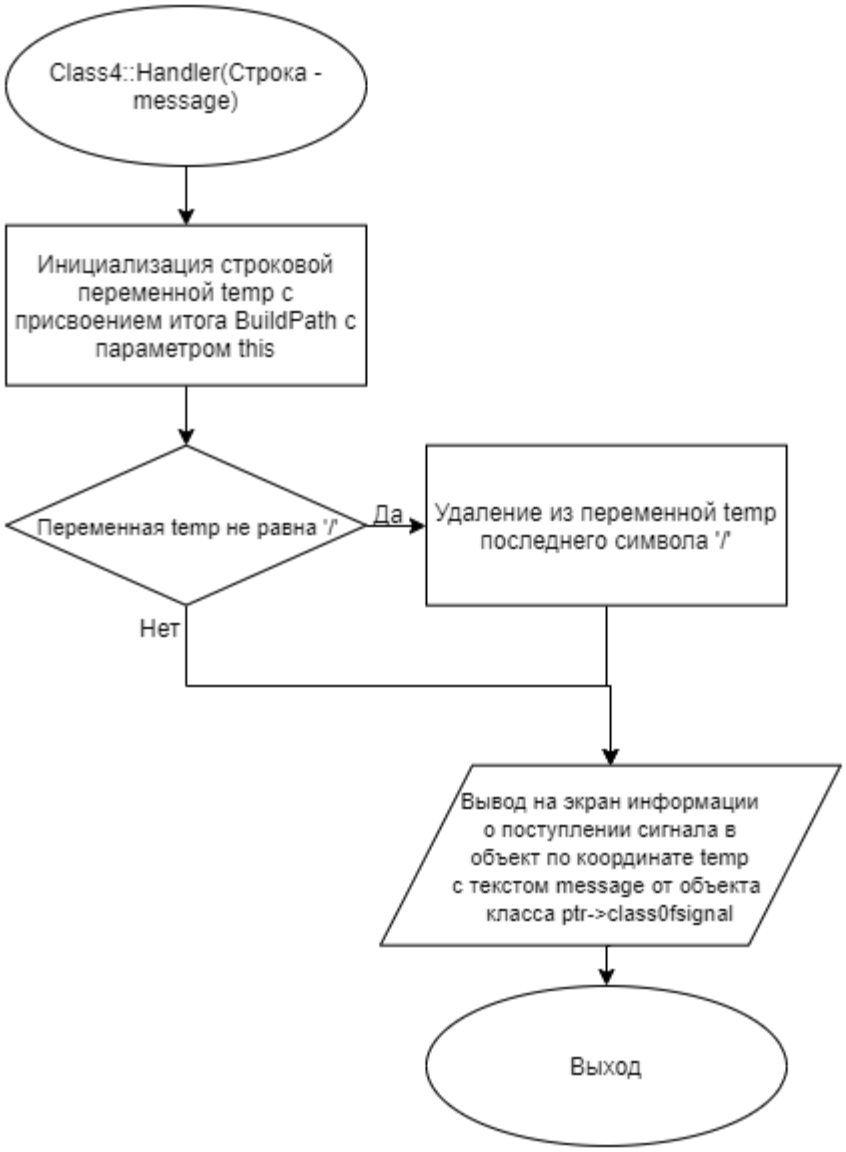

Рисунок 27 – Блок-схема алгоритма

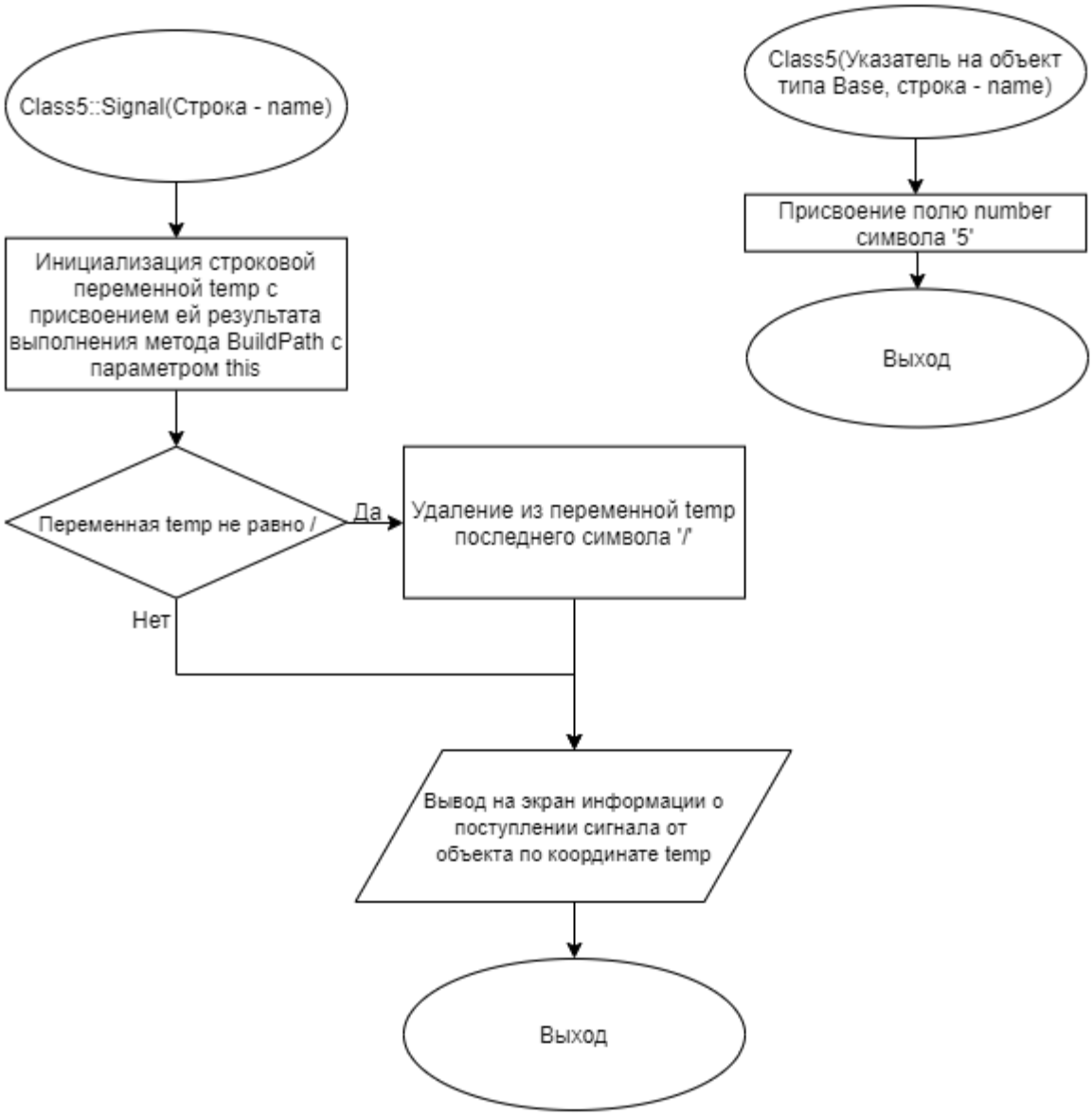

Рисунок 28 – Блок-схема алгоритма

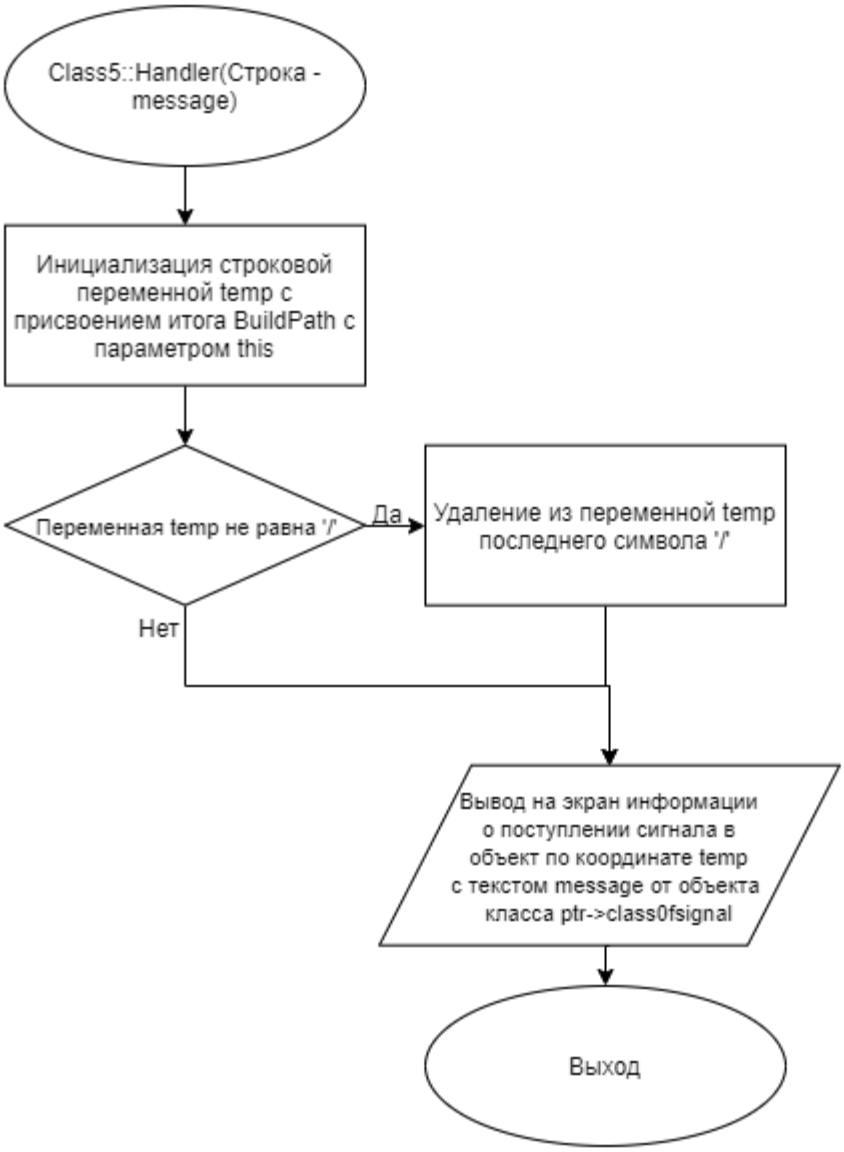

Рисунок 29 – Блок-схема алгоритма

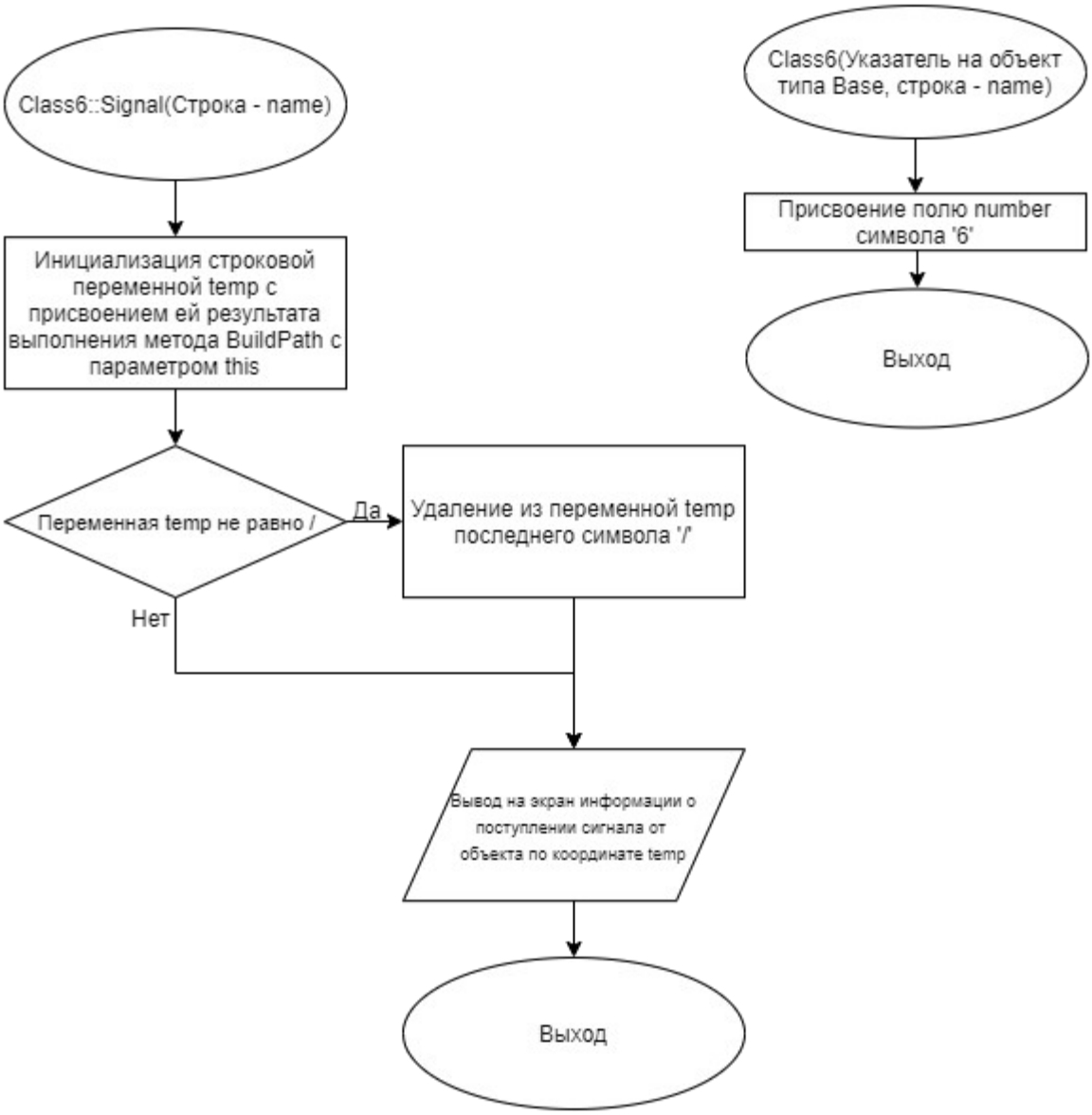

Рисунок 30 – Блок-схема алгоритма

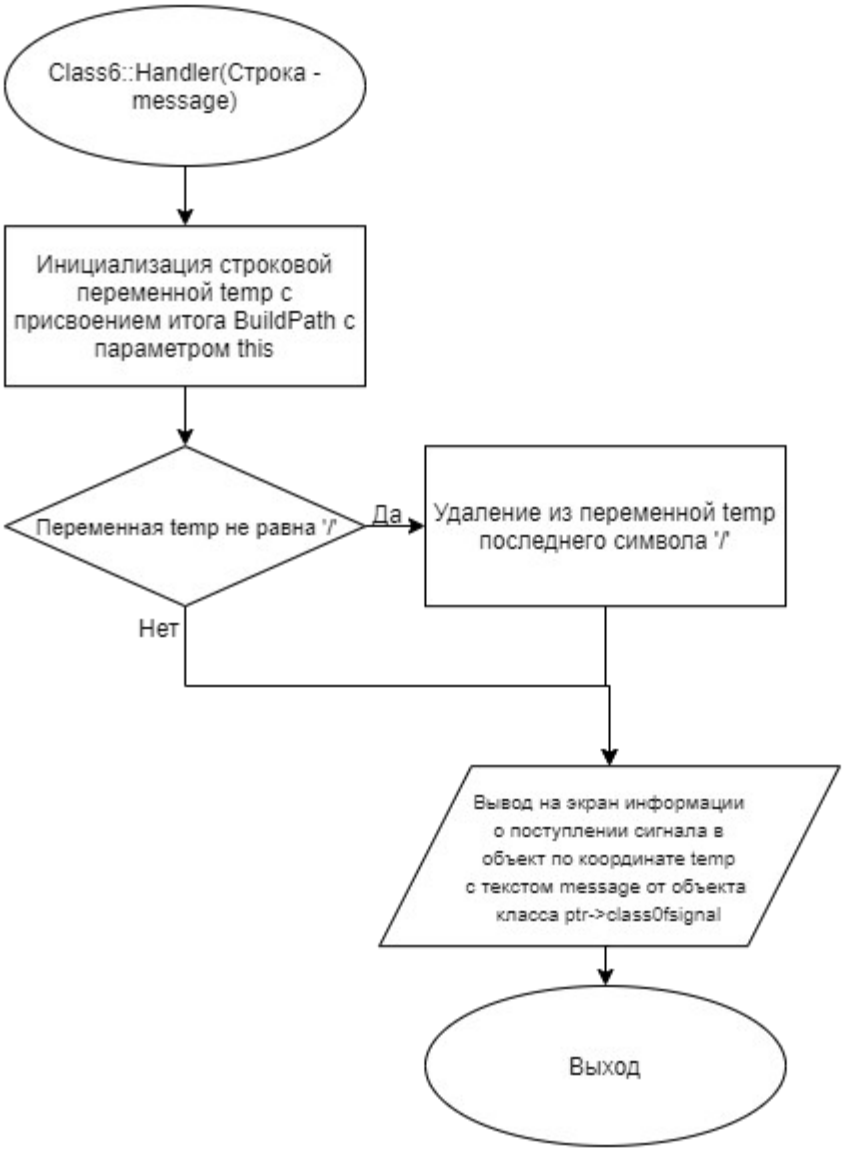

Рисунок 31 – Блок-схема алгоритма

Рисунок 32 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл App.cpp

Листинг 1 – App.cpp

```
#include "App.h"
#include "Class2.h"
#include "Class3.h"
#include "Class4.h"
#include "Class5.h"
#include "Class6.h"
using namespace std;
#define SIGNAL(path)(TYPE_SIGNAL)(&path)
#define HANDLER(message)(TYPE_HANDLER)(&message)
TYPE_SIGNAL
signals[]={SIGNAL(Base::Signal),SIGNAL(Class2::Signal),SIGNAL(Class3::Signal),SIGN
AL(Class4::Signal),SIGNAL(Class5::Signal),SIGNAL(Class6::Signal)};
TYPE_HANDLER
handlers[]={HANDLER(Base::Handler),HANDLER(Class2::Handler),HANDLER(Class3::Handle
r),HANDLER(Class4::Handler),HANDLER(Class5::Handler),HANDLER(Class6::Handler)};
int App::exec(){
      cout<<"Object tree"<<endl;
      cout<<this->GetName();
      PrintTree("");
      if (wrong!=""){
            cout<<endl<<"The head object "<<wrong<<" is not found";
            return 0;
      }
      this->SetStatus(1);
      string function, object;
      while (true){
            cin>>function;
            if(function=="END"){
                  break;
            }
            cin>>object;
            Base* ptr=WayPtr(object);
            if (function=="EMIT"){
                  string message;
                  getline(cin,message);
                  if (ptr==nullptr){
                        cout<<endl<<"Object "<<object<<" not found";
                  }
                  else{
                        ptr->SetSignal(signals[(ptr->number-'0')-1],message);
                  }
```

```
            }
            else if (function=="DELETE_CONNECT"){
                  string object2;
                  cin>>object2;
                  Base* ptr2=WayPtr(object2);
                  if(ptr==nullptr){
                        cout<<endl<<"Object "<<object<<" not found";
                  }
                  else if(ptr2==nullptr){
                        cout<<endl<<"Handler object "<<object2<<" not found";
                  }
                  else {
                        ptr->DeleteConnection(signals[((ptr2->number)-'0')-
1],ptr2, handlers[((ptr2->number)-'0')-1]);
                  }
            }
            else if (function=="SET_CONDITION"){
                  int status;
                  cin>>status;
                  if (ptr==nullptr){
                        cout<<endl<<"Object "<<object<<" not found";
                  }
                  else{
                        ptr->SetStatus(status);
                  }
            }
            else if (function=="SET_CONNECT"){
                  string object2;
                  cin>>object2;
                  Base* ptr2=WayPtr(object2);
                  if(ptr==nullptr){
                        cout<<endl<<"Object "<<object<<" not found";
                  }
                  else if(ptr2==nullptr){
                        cout<<endl<<"Handler object "<<object2<<" not found";
                  }
                  else {
                        ptr->SetConnection(signals[((ptr->number)-'0')-1],ptr2,
handlers[((ptr2->number)-'0')-1]);
                  }
            }
      }
      return 0;
}
void App::BuildTree(){
      string a,b;
      int cl;
      cin>>a;
      this->SetName(a);
      while(true){
            cin>>a;
            if (a=="endtree"){
                  break;
            }
            if (a=="/"){
                  a=this->name;
            }
```

```
            cin>>b>>cl;
            if (this->GetName()==a){
                  if (cl==2) new Class2(this, b);
                  if (cl==3) new Class3(this, b);
                  if (cl==4) new Class4(this, b);
                  if (cl==5) new Class5(this, b);
                  if (cl==6) new Class6(this, b);
            }
            else{
                  Base* temp=WayPtr(a, true);
                  if (temp!=nullptr){
                        if (cl==2) new Class2(temp, b);
                        if (cl==3) new Class3(temp, b);
                        if (cl==4) new Class4(temp, b);
                        if (cl==5) new Class5(temp, b);
                        if (cl==6) new Class6(temp, b);
                  }
                  else{
                        wrong=a;
                        break;
                  }
            }
      }
      if(wrong==""){
            string SignalFrom, SignalTo;
            while(true){
                  cin>>SignalFrom;
                  if(SignalFrom=="end_of_connections"){
                        break;
                  }
                  cin>>SignalTo;
                  Base* ptrTo=WayPtr(SignalFrom);
                  if(WayPtr(SignalTo)!=nullptr && ptrTo!=nullptr){
                        ptrTo->SetConnection(signals[WayPtr(SignalTo)->number-'0'-
1],WayPtr(SignalTo),handlers[WayPtr(SignalTo)->number-'0'-1]);
                  }
            }
      }
}
```

5.2 Файл App.h

Листинг 2 – App.h

```
#ifndef APP_H
#define APP_H
#include "Base.h"
#define SIGNAL(path)(TYPE_SIGNAL)(&path)
#define HANDLER(message)(TYPE_HANDLER)(&message)
typedef void(Base:: *TYPE_SIGNAL)(string&);
typedef void(Base:: *TYPE_HANDLER)(string);
class App : public Base{
public:
```

```
      App(Base* head, string name=""):Base(head, name){};
      void BuildTree();
      int exec();
};
#endif
```

5.3 Файл Base.cpp

Листинг 3 – Base.cpp

```
#include "Base.h"
#define SIGNAL(path)(TYPE_SIGNAL)(&path)
#define HANDLER(message)(TYPE_HANDLER)(&message)
typedef void(Base:: *TYPE_SIGNAL)(string&);
typedef void(Base:: *TYPE_HANDLER)(string);
Base::Base(Base* head, string name){
      this->name=name;
      this->head=head;
      wrong = "";
      status=1;
      if (head!=nullptr){
            head->children.push_back(this);
      }
}
void Base::SetName(string name){
      this->name=name;
}
string Base::GetName(){
      return name;
}
void Base::PrintTree(string space){
      for(int i=0;i<this->children.size();i++){
            cout<<endl;
            if (children[i]->children.size()>0){
                  cout<<space+"    "<<children[i]->GetName();
                  ((Base*)children[i])->PrintTree(space+"    ");
            }
            else{
                  cout<<space+"    "<<children[i]->GetName();
            }
      }
}

Base* Base::FindPtr(string name,Base* ptr, bool f){
      if (ptr!=nullptr){
            if (ptr->GetName()==name){
                  return ptr;
            }
            for (int i=0;i<ptr->children.size();i++){
                  if (ptr->children[i]->GetName()==name){
                        return ptr->children[i];
                  }
                  else if (ptr->children[i]->children.size()>0 && f==false){
```

```
                        return FindPtr(name,ptr->children[i]);
                  }
            }
      }
      return nullptr;
}

Base* Base::WayPtr(string directory, bool f){
      string text="";
      Base* temp=nullptr;
      if(directory=="/"){
            return this;
      }
      if (directory[0]=='/' && directory[1]=='/'){
            directory.erase(directory.begin()+0);
            directory.erase(directory.begin()+0);
            temp=FindPtr(directory,this,false);
            return temp;
      }
      else{
            Base* temp2=this;
            for(int i=0;i<directory.size();i++){
                  if (directory[i]=='/' || i==directory.size()-1){
                        if (text!=""){
                              if (directory[i]!='/'){
                                    text+=directory[i];
                              }
                              temp2=FindPtr(text,temp2,true);
                              if(temp2==nullptr){
                                    return temp2;
                              }
                              if (i==directory.size()-1){
                                    return temp2;
                              }
                              text="";
                        }
                  }
                  else{
                        text+=directory[i];
                  }
            }
            return nullptr;
      }
}
void Base::Signal(string& name){
      if (this->status!=0){
            string temp=BuildPath(this);
            if (temp!="/"){
                  temp.erase(temp.size()-1);
            }
            cout<<endl<<"Signal from "<<temp;
      }
}
void Base::Handler(string message){
      if (this->status!=0){
            string temp=BuildPath(this);
            if(temp!="/"){
```

```
                  temp.erase(temp.size()-1);
            }
            cout<<endl<<"Signal to "<<temp<<" Text: "<<message<<" (class: "<<this-
>class0fsignal<<")";
      }
}
void  Base::SetConnection(TYPE_SIGNAL  p_signal,  Base  *p_cl_base,  TYPE_HANDLER
p_handler){
      for(int i=0;i<connects.size();i++){
            if       (connects[i]->p_signal==p_signal       &&       connects[i]-
>p_cl_base==p_cl_base && connects[i]->p_handler==p_handler){
                  return;
            }
      }
      o_sh* temp = new o_sh();
      temp->p_signal=p_signal;
      temp->p_cl_base=p_cl_base;
      temp->p_handler=p_handler;
      connects.push_back(temp);
}
void  Base::DeleteConnection(TYPE_SIGNAL  p_signal,  Base  *p_cl_base,  TYPE_HANDLER
p_handler){
      int i=0;
      while(i<this->connects.size()){
            if       (connects[i]->p_signal==p_signal       &&       connects[i]-
>p_cl_base==p_cl_base && connects[i]->p_handler==p_handler){
                  connects.erase(connects.begin()+i);
                  break;
            }
            i++;
      }
}
string Base::BuildPath(Base* ptr, string path){
      if (ptr!=nullptr){
            path+='/';
            if (this->head!=nullptr){
                  string temp=path;
                  path="";
                  path+=this->head->BuildPath(this->head,    path)+this->GetName()
+temp;
            }
            return path;
      }
      return "";
}
void Base::SetSignal(TYPE_SIGNAL p_signal, string& message){
      if (this->status==0){
            return;
      }
      (this->*(p_signal))(message);
      for (int i=0;i<connects.size();i++){
            connects[i]->p_cl_base->class0fsignal=this->number;
            (connects[i]->p_cl_base->*(connects[i]->p_handler))(message);
      }
}
void Base::SetStatus(int n){
      this->status=n;
```

```
	for(int i=0;i<children.size();i++){
		children[i]->SetStatus(n);
	}
}
```

5.4 Файл Base.h

Листинг 4 – Base.h

```
#ifndef BASE_H
#define BASE_H
#include <iostream>
#include <string>
#include <vector>
#define SIGNAL(path)(TYPE_SIGNAL)(&path)
#define HANDLER(message)(TYPE_HANDLER)(&message)
using namespace std;
class Base{
protected:
	string name;
	string wrong;
	Base* head;
	char class0fsignal='1';
	short int status;
public:
	char number='1';
	Base(Base* head, string name="");
	void SetName(string name); //метод определения имени объекта
	string GetName(); //метод получения имени объекта
	void PrintTree(string space); //метод вывода наименований объектов в дереве иерархии
	vector<Base*>children;  //массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии
	Base* FindPtr(string name, Base* ptr, bool f=false); // метод поиска объекта на дереве иерархии по имени
	Base* WayPtr(string directory, bool f=false);
	typedef void(Base:: *TYPE_SIGNAL)(string&);
	typedef void(Base:: *TYPE_HANDLER)(string);
	struct o_sh{
		TYPE_SIGNAL p_signal;
		Base *p_cl_base;
		TYPE_HANDLER p_handler;
	};
	virtual void Signal(string& name);
	virtual void Handler(string message);
	void  SetConnection(TYPE_SIGNAL  p_signal,  Base*  p_cl_base,  TYPE_HANDLER p_handler);
	void  DeleteConnection(TYPE_SIGNAL  p_signal,  Base*  p_cl_base,  TYPE_HANDLER p_handler);
	void SetSignal(TYPE_SIGNAL p_signal, string& message);
	void SetStatus(int n);
	string BuildPath(Base* p_cl_base, string path="");
	vector<o_sh*> connects;
```

```
};
#endif
```

5.5 Файл Class2.cpp

Листинг 5 – Class2.cpp

```
#include "Class2.h"
void Class2::Signal(string& name){
	if(this->status!=0){
		string temp=BuildPath(this);
		if(temp!="/"){
			temp.erase(temp.size()-1);
		}
		cout<<endl<<"Signal from "<<temp;
	}
}
void Class2::Handler(string message){
	if (this->status!=0){
		string temp=BuildPath(this);
		if (temp!="/"){
			temp.erase(temp.size()-1);
		}
		cout<<endl<<"Signal to "<<temp<<" Text: "<<message<<" (class: "<<this-
>class0fsignal<<")";
	}
}
```

5.6 Файл Class2.h

Листинг 6 – Class2.h

```
#ifndef _CLASS2_H
#define _CLASS2_H
#include "Base.h"
class Class2:public Base {
public:
	Class2(Base* ptr, string name=""):Base(ptr, name){number='2';};
	void Signal(string& name);
	void Handler(string message);
};
#endif
```

5.7 Файл Class3.cpp

Листинг 7 – Class3.cpp

```
#include "Class3.h"
void Class3::Signal(string& name){
	if(this->status!=0){
		string temp=BuildPath(this);
		if(temp!="/"){
			temp.erase(temp.size()-1);
		}
		cout<<endl<<"Signal from "<<temp;
	}
}
void Class3::Handler(string message){
	if (this->status!=0){
		string temp=BuildPath(this);
		if (temp!="/"){
			temp.erase(temp.size()-1);
		}
		cout<<endl<<"Signal to "<<temp<<" Text: "<<message<<" (class: "<<this-
>class0fsignal<<")";
	}
}
```

5.8 Файл Class3.h

Листинг 8 – Class3.h

```
#ifndef _CLASS3_H
#define _CLASS3_H
#include "Base.h"
class Class3:public Base {
public:
	Class3(Base* ptr, string name=""):Base(ptr, name){number='3';};
	void Signal(string& name);
	void Handler(string message);
};
#endif
```

5.9 Файл Class4.cpp

Листинг 9 – Class4.cpp

```
#include "Class4.h"
void Class4::Signal(string& name){
	if(this->status!=0){
```

```
            string temp=BuildPath(this);
            if(temp!="/"){
                  temp.erase(temp.size()-1);
            }
            cout<<endl<<"Signal from "<<temp;
      }
}
void Class4::Handler(string message){
      if (this->status!=0){
            string temp=BuildPath(this);
            if (temp!="/"){
                  temp.erase(temp.size()-1);
            }
            cout<<endl<<"Signal to "<<temp<<" Text: "<<message<<" (class: "<<this-
>class0fsignal<<")";
      }
}
```

5.10 Файл Class4.h

Листинг 10 – Class4.h

```
#ifndef _CLASS4_H
#define _CLASS4_H
#include "Base.h"
class Class4:public Base {
public:
      Class4(Base* ptr, string name=""):Base(ptr, name){number='4';};
      void Signal(string& name);
      void Handler(string message);
};
#endif
```

5.11 Файл Class5.cpp

Листинг 11 – Class5.cpp

```
#include "Class5.h"
void Class5::Signal(string& name){
      if(this->status!=0){
            string temp=BuildPath(this);
            if(temp!="/"){
                  temp.erase(temp.size()-1);
            }
            cout<<endl<<"Signal from "<<temp;
      }
}
void Class5::Handler(string message){
      if (this->status!=0){
```

```
            string temp=BuildPath(this);
            if (temp!="/"){
                  temp.erase(temp.size()-1);
            }
            cout<<endl<<"Signal to "<<temp<<" Text: "<<message<<" (class: "<<this-
>class0fsignal<<")";
      }
}
```

5.12 Файл Class5.h

Листинг 12 – Class5.h

```
#ifndef _CLASS5_H
#define _CLASS5_H
#include "Base.h"
class Class5:public Base {
public:
      Class5(Base* ptr, string name=""):Base(ptr, name){number='5';};
      void Signal(string& name);
      void Handler(string message);
};
#endif
```

5.13 Файл Class6.cpp

Листинг 13 – Class6.cpp

```
#include "Class6.h"
void Class6::Signal(string& name){
      if(this->status!=0){
            string temp=BuildPath(this);
            if(temp!="/"){
                  temp.erase(temp.size()-1);
            }
            cout<<endl<<"Signal from "<<temp;
      }
}
void Class6::Handler(string message){
      if (this->status!=0){
            string temp=BuildPath(this);
            if (temp!="/"){
                  temp.erase(temp.size()-1);
            }
            cout<<endl<<"Signal to "<<temp<<" Text: "<<message<<" (class: "<<this-
>class0fsignal<<")";
      }
}
```

5.14 Файл Class6.h

Листинг 14 – Class6.h

```
#ifndef _CLASS6_H
#define _CLASS6_H
#include "Base.h"
class Class6:public Base {
public:
	Class6(Base* ptr, string name=""):Base(ptr, name){number='6';};
	void Signal(string& name);
	void Handler(string message);
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "App.h"
int main(){
	App app(nullptr);
	app.BuildTree();
	return app.exec();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 34.

Таблица 34 – 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 DELETE_CONNECT /object_s2/object_s4 / EMIT /object_s2/object_s4 Send message 2 SET_CONDITION /object_s2/object_s4 0 EMIT /object_s2/object_s4 Send message 3 SET_CONNECT /object_s1 /object_s2/object_s6 EMIT /object_s1 Send message 4 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 from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была написана программа, моделирующая работу сигналов и обработчиков сигналов, были получены навыки проектирования и реализации задач в соответсвии с принципами ООП.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. 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).

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).