



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»
(наименование дисциплины)

Тема курсовой работы Моделирование движения падающих шариков
(наименование темы)

Студент группы ИВБО-08-21 Хворостин Никита Романович
(учебная группа) (Фамилия Имя Отчество) (подпись студента)

Руководитель курсовой работы доцент каф.ВТ Путуридзе З.Ш.
(Должность, звание, ученая степень) (подпись руководителя)

Консультант доцент каф.ВТ Путуридзе З.Ш.
(Должность, звание, ученая степень) (подпись консультанта)

Работа представлена к защите « 16 » мая 2022 г.

Допущен к защите « 16 » мая 2022 г.

Москва 2022 г.

3 (8506)

З.Ш. Путуридзе



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий

Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой

Платонова О.В.

ФИО

« 14 » марта 2022г.

ЗАДАНИЕ

На выполнение курсовой работы

по дисциплине «Объектно-ориентированное программирование»

Студент Хворостин Никита Романович Группа ИББО-08-21

Тема Моделирование движения падающих шариков

Исходные данные:

1. Описания исходной иерархии дерева объектов.
2. Описание схемы взаимодействия объектов.
3. Множество команд для управления функционированием моделируемой системы.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Построение версий программ.
2. Построение и работа с деревом иерархии объектов.
3. Взаимодействия объектов посредством интерфейса сигналов и обработчиков.
4. Блок-схемы алгоритмов.
5. Управление функционированием моделируемой системы

Срок представления к защите курсовой работы: до « 16 » мая 2022 г.

Задание на курсовую работу выдал

Подпись

(Путуридзе З.И.)

ФИО консультанта

« 28 » февраля 2022 г.

Задание на курсовую работу получил

Подпись

(Хворостин Н.Р.)

ФИО исполнителя

« 28 » февраля 2022 г.

Москва 2022г.

ОТЗЫВ

на курсовую работу

по дисциплине «Объектно-ориентированное программирование»

Студент Хворостин Никита Романович группа ИББО-08-21
(ФИО студента) (Группа)

Характеристика курсовой работы

Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме	✓		
2. Соответствие курсовой работы заданию			✓
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.	✓		
4. Полнота выполнения всех пунктов задания		✓	
5. Логичность и системность содержания курсовой работы	✓		
6. Отсутствие фактических грубых ошибок	✓		

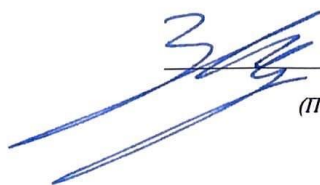
Замечаний:

Рекомендуемая оценка:

Не полностью выполнена

Не все пункты выполнены

3 (удов)



доцент каф. ВТ Путуридзе З.Ш.

(Подпись руководителя)

(ФИО руководителя)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 ПОСТАНОВКА ЗАДАЧИ.....	8
1.1 Описание входных данных	9
1.2 Описание выходных данных	11
2 МЕТОД РЕШЕНИЯ	13
3 ОПИСАНИЕ АЛГОРИТМОВ.....	18
3.0 Алгоритм метода get_object_by_coordinate класса base	18
3.1 Алгоритм метода bild_tree_objects класса application	21
3.2 Алгоритм метода exes_app класса application.....	24
3.3 Алгоритм функции main	26
3.4 Алгоритм метода find_object_by_name класса base.....	27
3.5 Алгоритм метода print_object_tree класса base	29
3.6 Алгоритм метода print_state класса base	30
3.7 Алгоритм метода install_state класса base	32
3.8 Алгоритм метода turn_off_state класса base.....	33
3.9 Алгоритм метода get_object_by_name класса base	33
3.10 Алгоритм метода get_root класса base.....	34
3.11 Алгоритм конструктора класса base.....	35
3.12 Алгоритм метода change_parent класса base	36
3.13 Алгоритм метода get_parent класса base	38
3.14 Алгоритм метода set_state класса base	38
3.15 Алгоритм метода get_state класса base.....	39
3.16 Алгоритм метода get_child_by_name класса base	39
3.17 Алгоритм деструктора класса base	40
4 БЛОК-СХЕМЫ АЛГОРИТМОВ	42
5 КОД ПРОГРАММЫ.....	61

5.0 Файл application.cpp	61
5.1 Файл application.h	62
5.2 Файл base.cpp	63
5.3 Файл base.h.....	66
5.4 Файл cl_1.cpp	67
5.5 Файл cl_1.h.....	67
5.6 Файл cl_2.cpp	67
5.7 Файл cl_2.h.....	67
5.8 Файл cl_3.cpp	68
5.9 Файл cl_3.h.....	68
5.10 Файл cl_4.cpp	68
5.11 Файл cl_4.h.....	69
5.12 Файл cl_5.cpp	69
5.13 Файл cl_5.h.....	69
5.14 Файл cl_6.cpp	69
5.15 Файл cl_6.h.....	70
5.16 Файл main.cpp	70
6 ТЕСТИРОВАНИЕ	71
ЗАКЛЮЧЕНИЕ	72
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	73

ВВЕДЕНИЕ

Объектно-ориентированное программирование (ООП) в наши дни это одна из самых популярных парадигм программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

ООП это максимально простой и понятный для человека способ описать окружающий мир и построить работу программы по подобию реального мира, она позволяет структурировать информацию с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при создании крупных проектов.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

Инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды "что делать", без уточнения, как именно делать;

Наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя все остальное, учтённое в предыдущих шагах;

Полфирмизм для определения точки, в которой утиное управление лучше разделить или наоборот - собрать воедино;

Язык C++ также прекрасно подходит для изучения ООП благодаря тому, что он сочетает в себе высокоуровневые и низкоуровневые средства. Использование таких средств, как указатели и динамическое выделение памяти, позволяет понять (или в дальнейшем способствует пониманию), что такое стек, куча, стек вызовов, раскрутка и т.д. Помимо этого, на практике понимание концепции адрессов и адресной арифметики. На примерах демонстрируется, что память надо выделять и освобождать, что существуют утечки памяти. Чётко разграниченные уровни доступа к членам класса, возможность множественного наследования и динамический полиморфизм дают возможность быстро усвоить основные концепции ООП. Так-же C++ не ограничивает использованием только ООП, позволяет понимать функциональное программирование, что учит сочетать разные средства разработки, ведь ООП не может существовать полностью без процедурного программирования

1 ПОСТАНОВКА ЗАДАЧИ

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

В составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задается в следующем виде:

/ - корневой объект;
//«имя объекта» - поиск объекта по уникальному имени от корневого (для однозначности уникальность требуется в рамках дерева);
. - текущий объект;
«имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;
/«имя объекта 1»[/«имя объекта 2»] . . . - абсолютная координата от корневого объекта.

Примеры координат:

/
//ob_3
.
ob_2/ob_3
ob_2
/ob_1/ob_2/ob_3

Если координата пустая строка или объект не найден, то вернуть нулевой указатель.

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы.

Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

При построении дерева уникальность наименования относительно множества непосредственно подчиненных объектов для любого головного объекта соблюдены.

Добавить проверку допустимости исходной сборки. Собрать дерево невозможно, если по заданной координате головной объект не найден (например, ошибка в наименовании или еще не расположен на дереве объектов).

Система обрабатывает следующие команды:

SET «координата» – устанавливает текущий объект;

FIND «координата» – находит объект относительно текущего;

END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим.

При вводе данных в названии команд ошибок нет. Условия уникальности имен объектов для однозначной обработки соответствующих команд соблюдены.

1.1 Описание входных данных

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы.

Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

После ввода состава дерева иерархии построчно вводятся команды:

SET «координата» - установить текущий объект;

FIND «координата» - найти объект относительно текущего;

END – завершить функционирование системы (выполнение программы).

Команды SET и FIND вводятся произвольное число раз. Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов.

root

/ object_1 3

/ object_2 2

/object_2 object_4 3

/object_2 object_5 4

/ object_3 3

/object_2 object_3 6

/object_1 object_7 5

/object_2/object_4 object_7 3

endtree

FIND object_2/object_4

SET /object_2

FIND //object_5

FIND /object_15

FIND .

FIND object_4/object_7

END

1.2 Описание выходных данных

Первая строка:

Object tree

Со второй строки вывести иерархию построенного дерева как в курсовой работе версия №2.

При ошибке определения головного объекта, прекратить сборку, вывести иерархию уже построенного фрагмента дерева, со следующей строки сообщение: The head object «координата головного объекта» is not found и прекратить работу программы.

Если дерево построено, то далее построчно:

для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

Object is not found: «имя текущего объекта» «искомая координата объекта»

для команд FIND вывести:

«искомая координата объекта» Object name: «наименование объекта»

Если объект не найден, то:

«искомая координата объекта» Object is not found

Пример вывода иерархии дерева объектов.

Object tree

root

 object_1

 object_7

 object_2

 object_4

 object_7

 object_5

 object_3

 object_3

object_2/object_4 Object name: object_4

Object is set: object_2

//object_5 Object name: object_5

/object_15 Object is not found
.
Object name: object_2
object_4/object_7 Object name: object_7

2 МЕТОД РЕШЕНИЯ

Для решения задачи используются:

- Объект стандартного потока вывода на экран cout;
- Объект стандартного потока ввода с клавиатуры cin;
- Стандартная функция substr(pos, count) библиотеки <string> -возвращает подстроку данной строки начиная с символом с индексом pos количеством count или до конца строки.
- Объект ob_application класса application;
- Объекты классов cl_2, cl_3, cl_4, cl_5, cl_6 (имена и количество задаются пользователем);

• Класс base:

Свойства/поля:

- Поле, отвечающее за наименование объекта
- Наименование - object_name;
- Тип - строковый;
- Модификатор доступа - private.
- Поле, отвечающее за указатель на головной объект для текущего объекта .
- Наименование - p_parent;
- Тип - указатель на объект класса base (или его наследников);
- Модификатор доступа - private.
- Поле, отвечающее за состояние объекта
- Наименование - state;
- Тип - целочисленный;
- Модификатор доступа - private.
- Поле, отвечающее за массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии:

- Наименование - children;
- Тип - вектор ;
- Модификатор доступа - priva
- методы:

• Конструктор base:

• Функционал - параметризованный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта (имеет значение по умолчанию).

• set_object_name:

• Функционал - используется для установки имени объекта.

• get_object_name:

• Функционал - используется для получения имени объекта.

• change_parent:

• Функционал - используется для переопределения головного объекта для текущего в дереве иерархии.

• get_parent:

• Функционал - используется для получения указателя на головной объект текущего объекта.

• set_state:

• Функционал - используется для определения состояние объекта.

• get_state:

• Функционал - используется для получения состояние объекта.

• get_child_by_name:

• Функционал - используется для получения указателя объект-потомок по имени объекта

• print_object_tree:

• Функционал - используется для вывода наименований объектов в дереве иерархии слева направо и сверху вниз.

- Деструктор ~base:
 - Функционал - деструктор, удаление объектов иерархического дерева.
-
- get_object_by_coordinate()
 - Функционал - получения указателя на любой объект в составе дерева иерархии объектов согласно пути(координаты).
-
- Класс applicatoin
 - методы:
 - Конструктор application:
 - Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии
 - bild_tree_objects ()
 - Функционал - построение дерева иерархии объектов.
 - exes_app()
 - Функционал - метод запуска приложения (начало функционирования системы, выполнения алгоритм решения задачи).
-
- Класс cl_1 (наследуется от base):
 - Методы
 - Конструктор cl_1
 - Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.
-
- Класс cl_2 (наследуется от base):
 - Методы:
 - Конструктор cl_2()
 - Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.
-
- Класс cl_3 (наследуется от base):
 - Методы:
 - Конструктор cl_3()
 - Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.
-
- Класс cl_4 (наследуется от base):
 - Методы:
 - Конструктор cl_4()
 - Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.
-
- Класс cl_5 (наследуется от base):
 - Методы:
 - Конструктор cl_5()
 - Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.
-
- Класс cl_6(наследуется от base):
 - Методы:
 - Конструктор cl_6()

- Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.0 Алгоритм метода `get_object_by_coordinate` класса `base`

Функционал: получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты).

Параметры: строковый, `path` - путь(координата) объекта.

Возвращаемое значение: указатель на объект класса `base`.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода `get_object_by_coordinate` класса `base`

№	Предикат	Действия	№ перехода
1	Значение параметра <code>path</code> равно "/"	Возврат результата вызова метода <code>get_root()</code>	∅
	Значение параметра <code>path</code> равно "."	Возврат значения указателя на текущий объект	∅
	Два первых символа параметра <code>path</code> равны "/"	Возврат результата вызова метода <code>get_object_by_name()</code> от корневого объекта с параметром значения подстроки <code>path</code> , начиная с третьего символа	∅
	Первый символ параметра <code>path</code> равен "/"	Возврат результата вызова метода <code>get_object_by_coordinate()</code> для корневого объекта с параметром значения подстроки, начиная со второго символа	∅
			2

№	Предикат	Действия	№ перехода
2		Инициализация целочисленной переменной index значением равным размерности строки path	3
3		Инициализация целочисленной переменной счетчика i значением 0	4
4	Значение переменной i меньше размерности строки path		5
			6
5	Значение i - го элемента строки равно '/'	Присвоение переменной index значения переменной i	6
		Увеличение переменной счетчика i на 1	4
6		Инициализация строковой переменной s значением подстроки с первого элемента длиной index	7
7		Инициализация целочисленной переменной счетчика i значением 0	8
8	значение переменной i меньше размера массива children		9
		Возврат нулевого указателя	∅
9	Результат вызова метода get_object_name подчиненного объекта равен значению переменной s		10
			11

№	Предикат	Действия	№ перехода
10	Размер строки path равен значению переменной index	Возврат указателя на подчиненный объект для текущего	∅
		Возврат результата вызова метода get_object_by_name() подчиненного объекта к текущему с параметром значения подстроки path, начиная с символа index+1	∅
11		Увеличение переменной счетчика i на 1	8

3.1 Алгоритм метода `build_tree_objects` класса `application`

Функционал: Построение дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 2.

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

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных string,path	2
2		Объявление целочисленной переменной number	3
3		Ввод с клавиатуры значения переменной name	4
4		Вызов метода set_object_name() текущего объекта с значением параметра name	5
5		Объявление указателя	6

№	Предикат	Действия	№ перехода
		obj на объект класса base	
6	Осуществляется ввод данных		7
			∅
7	Значение переменной path = "endtree"		∅
		Ввод с клавиатуры значения переменных name,number	8
8	Результат вызова метода get_object_by_coordinate с параметром path равен 0	вызов метода print_object_tree с параметрами указателя на текущий объект и 0	9
			11
9		Вывод перехода на новую строку + "The head object + path + " is not found"	10
10		завершение программы	∅
11	Значения переменной number = 2	Создание объекта класса cl_2 посредством параметризованного конструктора	6
			12
12	Значения переменной number = 3	Создание объекта класса cl_3 посредством параметризованного конструктора	6
			13
13	Значения переменной number = 4	Создание объекта класса cl_4 посредством параметризованного конструктора	6
			14
14	Значения переменной number = 5	Создание объекта класса cl_5 посредством параметризованного конструктора	6
			15
15	Значения переменной number = 6	Создание объекта класса cl_6 посредством параметризованного конструктора	6
			6

3.2 Алгоритм метода exes_app класса application

Функционал: метод запуска приложения (начало функционирования системы, выполнения алгоритма решения задачи).

Параметры: нет.

Возвращаемое значение: целочисленное, 0 - при успешном выполнении, в другом случае - код ошибки.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода exes_app класса application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree"	2
2		вызов метода print_object_tree с параметрами указателя на текущий объект и 0	3
3		Объявление строковых переменных command, path	4
4		Инициализация указателя p на класс base значением указателя на текущий объект	5
5	конец входных данных		∅
			6
6	Значение переменной command = "END"		∅
		ввод значения переменной path	7
7	Значение переменной command = "SET"		8
			10
8	Результат вызова метода get_object_by_coordinate() по указателю p не равен 0	Присвоение указателю p результата вызова метода get_object_by_coordinate() по указателю p	9
		Вывод перехода на новую	5

№	Предикат	Действия	№ перехода
		строку +"Object is not found: "+ результат вызова метода get_object_name() по указателю p+ значения переменной path	
9		Вывод перехода на новую строку + "Object is set: " + результат вызова метода get_object_name() по указателю p	5
10	Значение переменной command = "FIND"		11
			5
11	Результат вызова метода get_object_by_coordinate() по указателю p не равен 0	вывод перехода на новую строку+значения переменной path + "Object is found: "+результат вызова метода get_object_name с координатой path	5
			5

3.3 Алгоритм функции main

Функционал: основанная функция.

Параметры: нет.

Возвращаемое значение: целочисленное, результат выполнения программы.

Алгоритм функции представлен в таблице 4.

Таблица 4 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_application класса application посредством параметризованного конструктора с параметром	2

№	Предикат	Действия	№ перехода
		нулевого указателя	
2		вызов метода bild_tree_objects для объекта ob_application	3
3		возврат результата вызова метода exes_app для объект ob_applicaton	∅

3.4 Алгоритм метода find_object_by_name класса base

Функционал: Поиск объекта на дереве иерархии по имени (метод возвращаетуказатель на найденный объект или nullptr).

Параметры: строковый, name - наименование объекта.

Возвращаемое значение: указатель на объект класса base.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода find_object_by_name класса base

№	Предикат	Действия	№ перехода
1		Инициализация указателя p на объект класса base значением нулевого указателя	2
2	Результат вызова метода get_object_name для текущего объекта равен значению параметра name	вовращение указателя на текущий объект	∅
		Указателю p присвоить значение указателя на текущий объект	3
3		Инициализация целочисленной переменной счетчика i значением 0	4
4	Значение переменной i		5

№	Предикат	Действия	№ перехода
	меньше размера массива children		
			7
5	Результат вызова метода get_object_name для объекта подчиненного текущему с именем name равно значению параметра name	возврат результата вызова метода find_object_by_name для объекта подчиненного текущему	∅
			6
6		Увеличение переменной счетчика i на 1	4
7	Указатель на головной объект для текущего равен нулевому и результат вызова метода get_object_name не равен значению параметра name	возврат нулевого указателя	∅
		возврат значение указателя p	∅

3.5 Алгоритм метода print_object_tree класса base

Функционал: вывод дерева иерархии объектов.

Параметры: указатель на класс base, parent - указатель на головной объект; целочисленный, level - уровень вывода объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода print_object_tree класса base

№	Предикат	Действия	№ перехода
1		Объявление строковой	2

№	Предикат	Действия	№ перехода
		переменной s	
2	Значение параметра level больше нуля	Добавление к строке s количество пробелов равное level*4	3
			3
3		вывод переменной s+ результата вызова метода get_object_name+переход на новую строку	4
4	размер массива children равен 0		∅
			5
5		Инициализация целочисленной переменной счетчика i значением 0	6
6	Значение переменной i меньше размера массива children	вызов метода print_object_tree с параметрами значения указателя на подчиненный объект и level+1	7
			∅
7		Увеличение переменной счетчика i на 1	6

3.6 Алгоритм метода print_state класса base

Функционал: вывод дерева иерархии объектов и отметок их готовности.

Параметры: указатель на класс base, parent - указатель на головной объект; целочисленный, level - уровень вывода объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода print_state класса base

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной s	2
2	Значение	Добавление к строке s	3

№	Предикат	Действия	№ перехода
	параметра level больше нуля	количество пробелов равно level*4	
			3
3		вывод перехода на новую строку+переменной s+ результата вызова метода get_object_name+пробел	4
4	Результат вызова метода get_state не равен 0	вывод "is ready"	5
		вывод "is not ready"	5
5	размер массива children равен 0		∅
			6
6		Инициализация целочисленной переменной счетчика i значением 0	7
7	Значение переменной i меньше размера массива children	вызов метода print_state с параметрами значения указателя на подчиненный объект и level+1	8
			∅
8		Увеличение переменной счетчика i на 1	7

3.7 Алгоритм метода install_state класса base

Функционал: установка готовности объекта.

Параметры: целочисленный, _state - состояние готовности объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода install_state класса base

№	Предикат	Действия	№ перехода
1	Значение параметра state не равно 0		2

№	Предикат	Действия	№ перехода
			4
2	Значение указателя на головной объект для текущего равен нулевому	Присвоение полю state текущего объекта значение параметра _state	∅
			3
3	Значение поля state для головного объекта не равно 0	Присвоение полю state текущего объекта значение параметра _state	∅
			∅
4	Значение поля state для текущего объекта не равно 0	вызов метода turn_off_state	∅
			∅

3.8 Алгоритм метода turn_off_state класса base

Функционал: Функционал - отключение готовности для головного объекта и его подчиненных.

Параметры: Нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода turn_off_state класса base

№	Предикат	Действия	№ перехода
1		Присвоение полю state для текущего объекта значение 0	2
2	размер массива children равен 0		∅
			3
3		Инициализация целочисленной переменной счетчика i значением 0	4
4	Значение переменной i меньше размера массива children	вызов метода turn_off_state для объекта подчиненного к текущему	5

№	Предикат	Действия	№ перехода
			∅
5		Увеличение переменной счетчика <i>i</i> на 1	4

3.9 Алгоритм метода `get_object_by_name` класса `base`

Функционал: получение значения указателя на объект по имени.

Параметры: строковый, `name` - наименование объекта.

Возвращаемое значение: указатель на объект класса `base`.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода `get_object_by_name` класса `base`

№	Предикат	Действия	№ перехода
1		возврат результата метода <code>find_object_by_name</code> с параметром <code>name</code> для корневого объекта	∅

3.10 Алгоритм метода `get_root` класса `base`

Функционал: Получение указателя на головной объект.

Параметры: нет.

Возвращаемое значение: указатель на объект класса `base` (указатель на корневой объект).

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода `get_root` класса `base`

№	Предикат	Действия	№ перехода
1		Инициализация указателя <code>p</code> на класс <code>base</code> значением указателя на текущий объект	2
2	Значение	Присвоение указателю <code>p</code>	2

№	Предикат	Действия	№ перехода
	указателя на головной объект не равно 0	значение указателя на головной объект	
			3
3		возврат значения p	∅

3.11 Алгоритм конструктора класса base

Функционал: параметризированный конструктор, устанавливается имяобъекта и головной объект для текущего.

Параметры: p_parent - указатель на объект класса base (указатель на головнойобъект в дереве иерархии); object_name - строковый, наименование объекта.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса base

№	Предикат	Действия	№ перехода
1		Присвоение полю object_name текущего объекта значение параметра object_name	2
2		Присвоение полю p_parent текущего объекта значение параметра p_parent	3
3		Присвоение полю state текущего объекта значения 0	4
4	Указатель на головной объект p_parent для текущего объекта не нулевой	Добавление для головного объекта текущий объект в массив подчиненных объектов	∅
			∅

3.12 Алгоритм метода `change_parent` класса `base`

Функционал: используется для переопределения головного объекта для текущего в дереве иерархии.

Параметры: .

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода `change_parent` класса `base`

№	Предикат	Действия	№ перехода
1	Указатель на головной объект для текущего объекта равен нулевому указателю или значение параметра <code>new_parent</code> равно нулевому указателю		∅
			2
2		Инициализация целочисленной переменной счетчика <code>i</code> значением 0	3
3	Значение переменной счетчика меньше размера массива подчиненных объектов <code>children</code> для текущего головного		4
			5
4	Значение <code>i</code> -ого элемента массива <code>children</code> текущего головного равно указателю на текущий объект	удаление текущего объекта из списка наследников объектародителя	5
		увеличение переменной счетчика <code>i</code> на 1	3
5		Присвоение полю	6

№	Предикат	Действия	№ перехода
		р_parent текущего объекта значения параметра new_parent	
6		добавление текущего объекта в массив наследников нового головного объекта new_parent	∅

3.13 Алгоритм метода get_parent класса base

Функционал: используется для получения указателя на головной объект для текущего объекта.

Параметры: нет.

Возвращаемое значение: указатель на объект класса base (указатель на головной объект для текущего объекта).

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода get_parent класса base

№	Предикат	Действия	№ перехода
1		возврат указателя р_parent на головной объект для текущего объекта	∅

3.14 Алгоритм метода set_state класса base

Функционал: определение состояния объекта.

Параметры: целочисленный, state, состояние объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода *set_state* класса *base*

№	Предикат	Действия	№ перехода
1		Присвоение полю <i>state</i> текущего объекта значение параметра <i>state</i>	∅

3.15 Алгоритм метода *get_state* класса *base*

Функционал: используется для получения состояние объекта.

Параметры: нет.

Возвращаемое значение: целочисленное, состояние объекта.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода *get_state* класса *base*

№	Предикат	Действия	№ перехода
1		возврат поля <i>state</i> текущего объекта	∅

3.16 Алгоритм метода *get_child_by_name* класса *base*

Функционал: Получение указателя на подчиненный объект по имени объекта.

Параметры: строковый, *object_name*, наименование объекта.

Возвращаемое значение: указатель на объект класса *base* (указатель на подчиненный объект).

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода *get_child_by_name* класса *base*

№	Предикат	Действия	№ перехода
1	Размер массива подчиненных объектов <i>children</i> равен 0		∅

№	Предикат	Действия	№ перехода
		Инициализация целочисленной переменной счетчика i значением 0	2
2	Значение переменной счетчика i меньше размера массива children		3
			∅
3	Имя объекта i-ого элемента массива children равен значению параметра object_name	возврат i-ого элемента массива children	∅
		Увеличение переменной счетчика i на 1	2

3.17 Алгоритм деструктора класса base

Функционал: деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 18.

Таблица 18 – Алгоритм деструктора класса base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной счетчика i значением 0	2
2	Значение переменной счетчика i меньше размера массива children	Удаление значения i-ого элемента массива children	3
			∅
3		увеличение переменной счетчика i на 1	2

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-19.

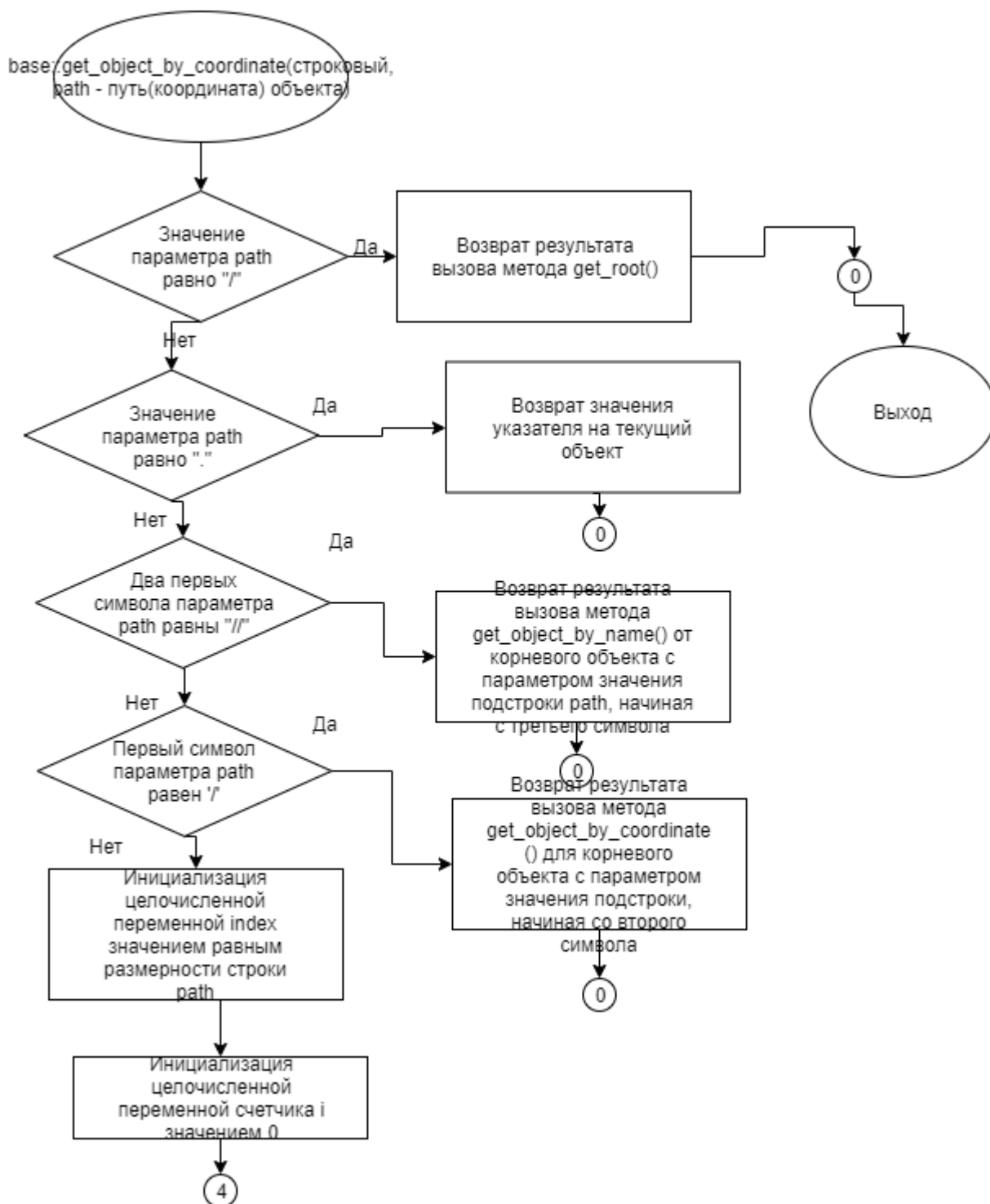


Рисунок 1 – Блок-схема алгоритма

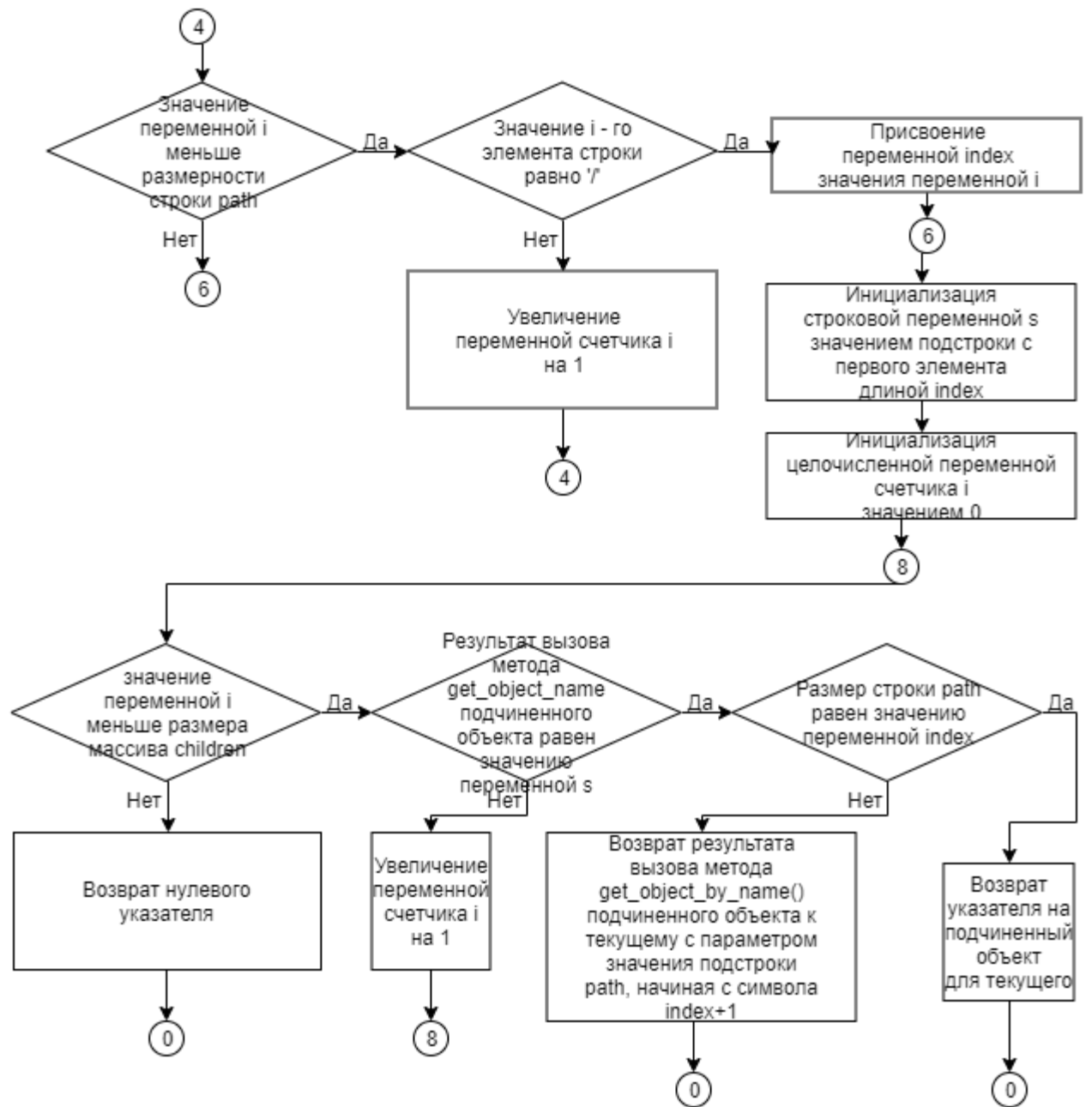


Рисунок 2 – Блок-схема алгоритма

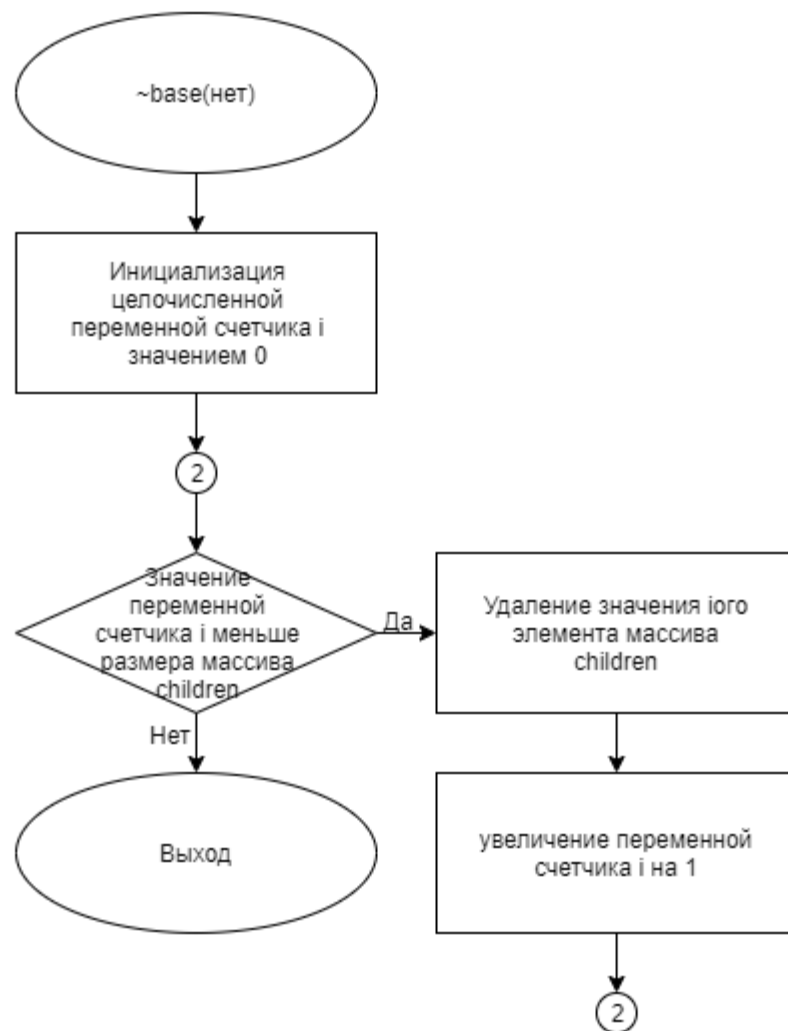


Рисунок 3 – Блок-схема алгоритма

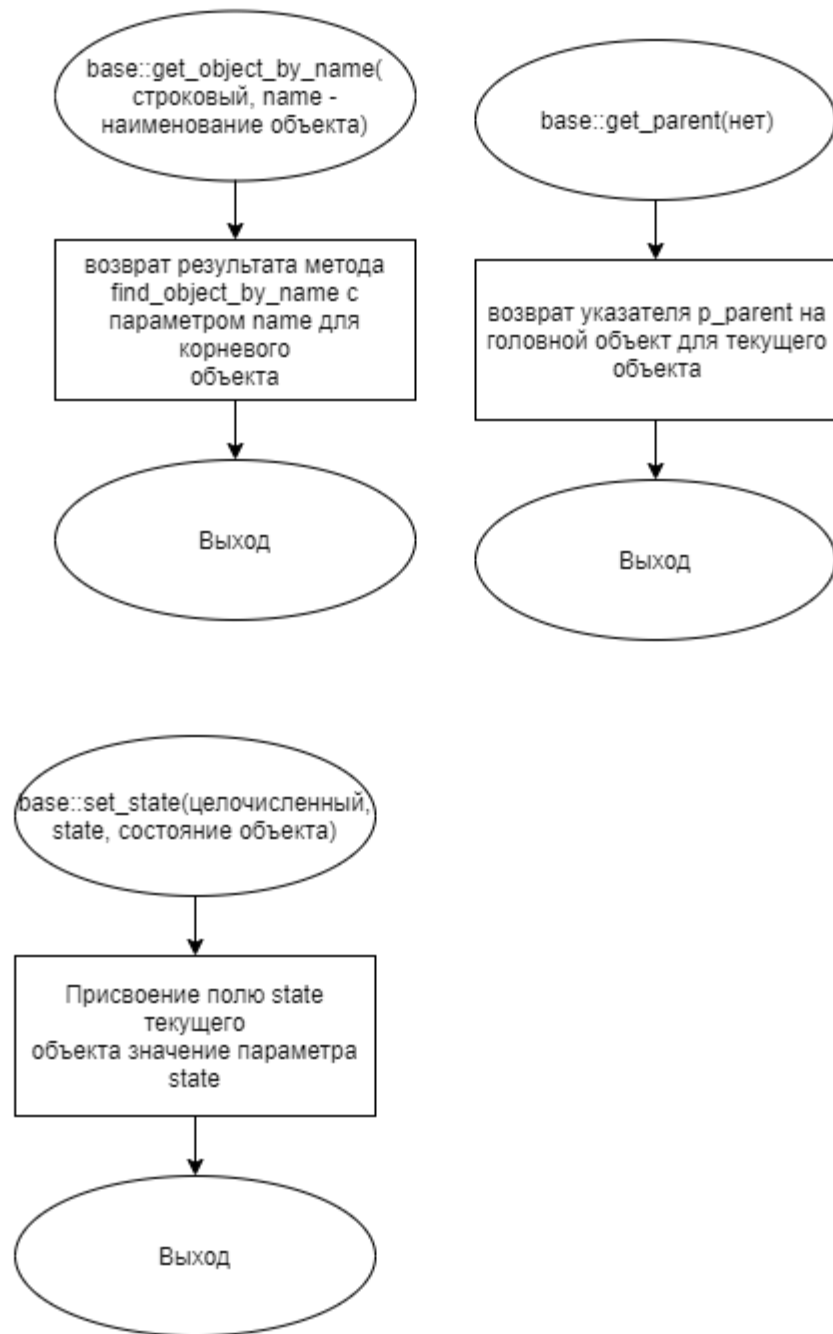


Рисунок 4 – Блок-схема алгоритма

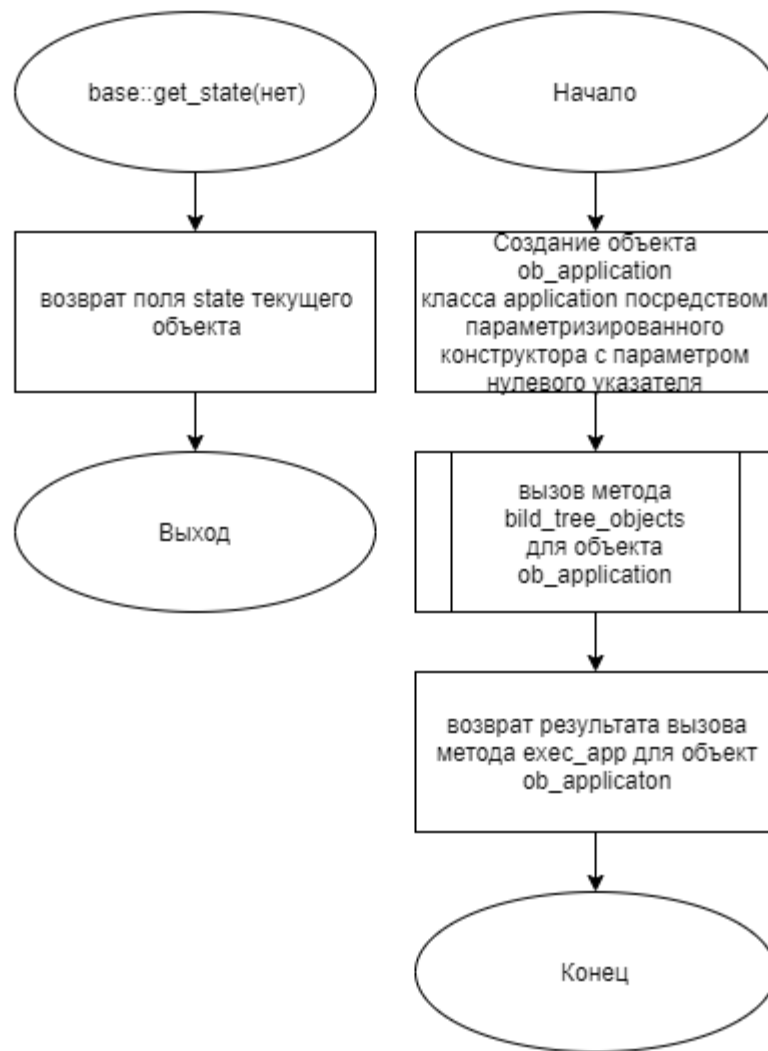


Рисунок 5 – Блок-схема алгоритма

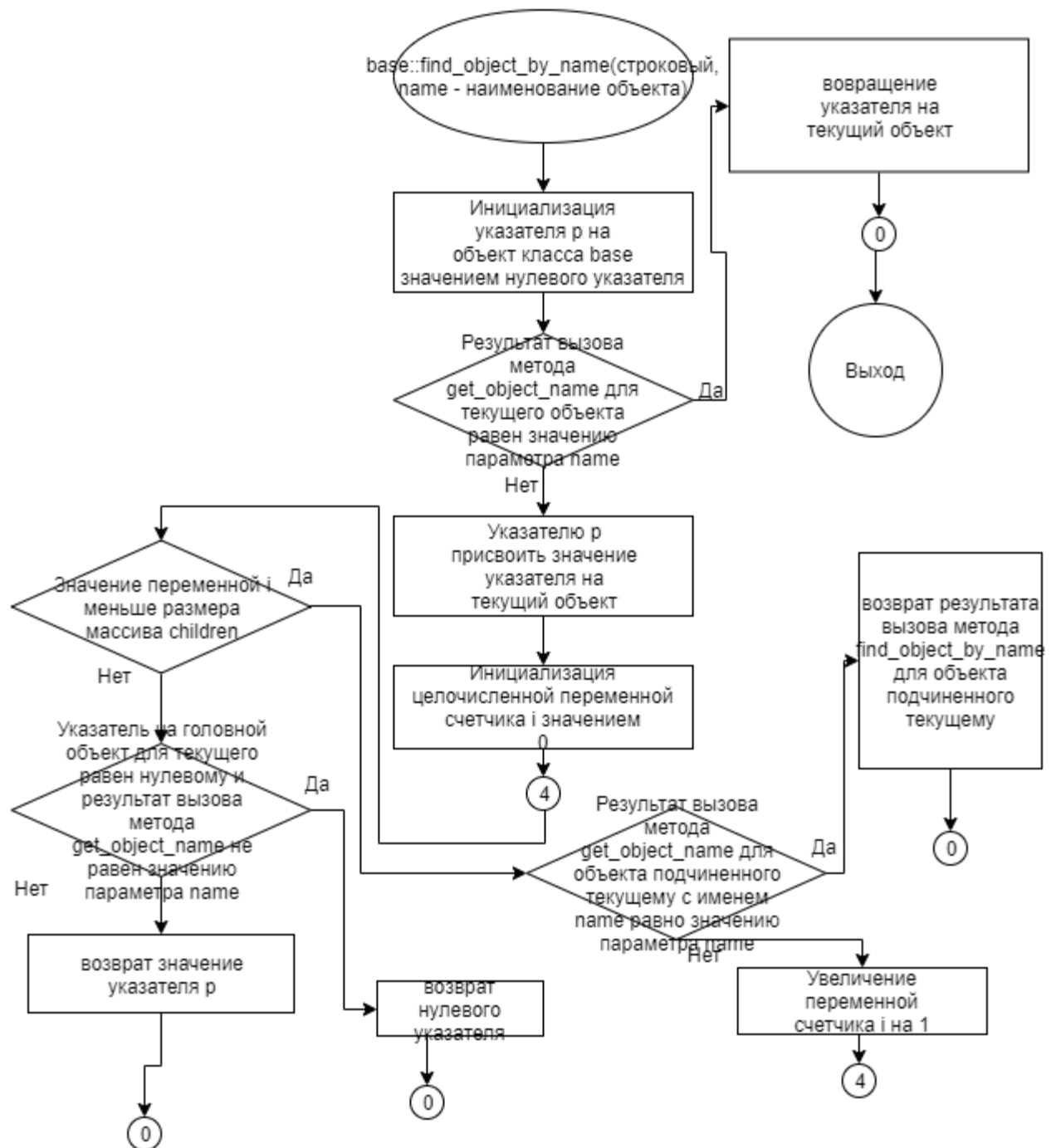


Рисунок 6 – Блок-схема алгоритма

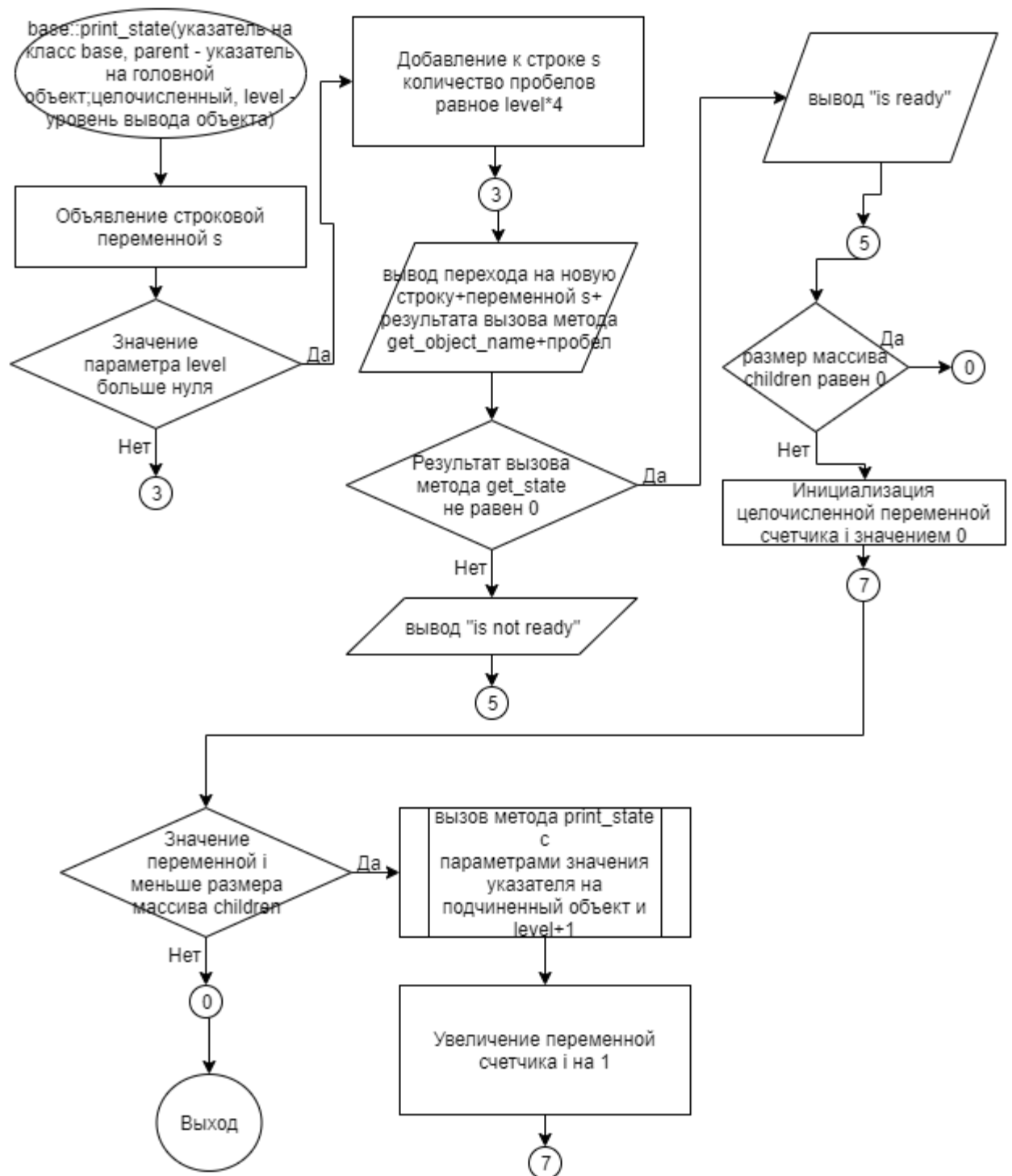


Рисунок 7 – Блок-схема алгоритма

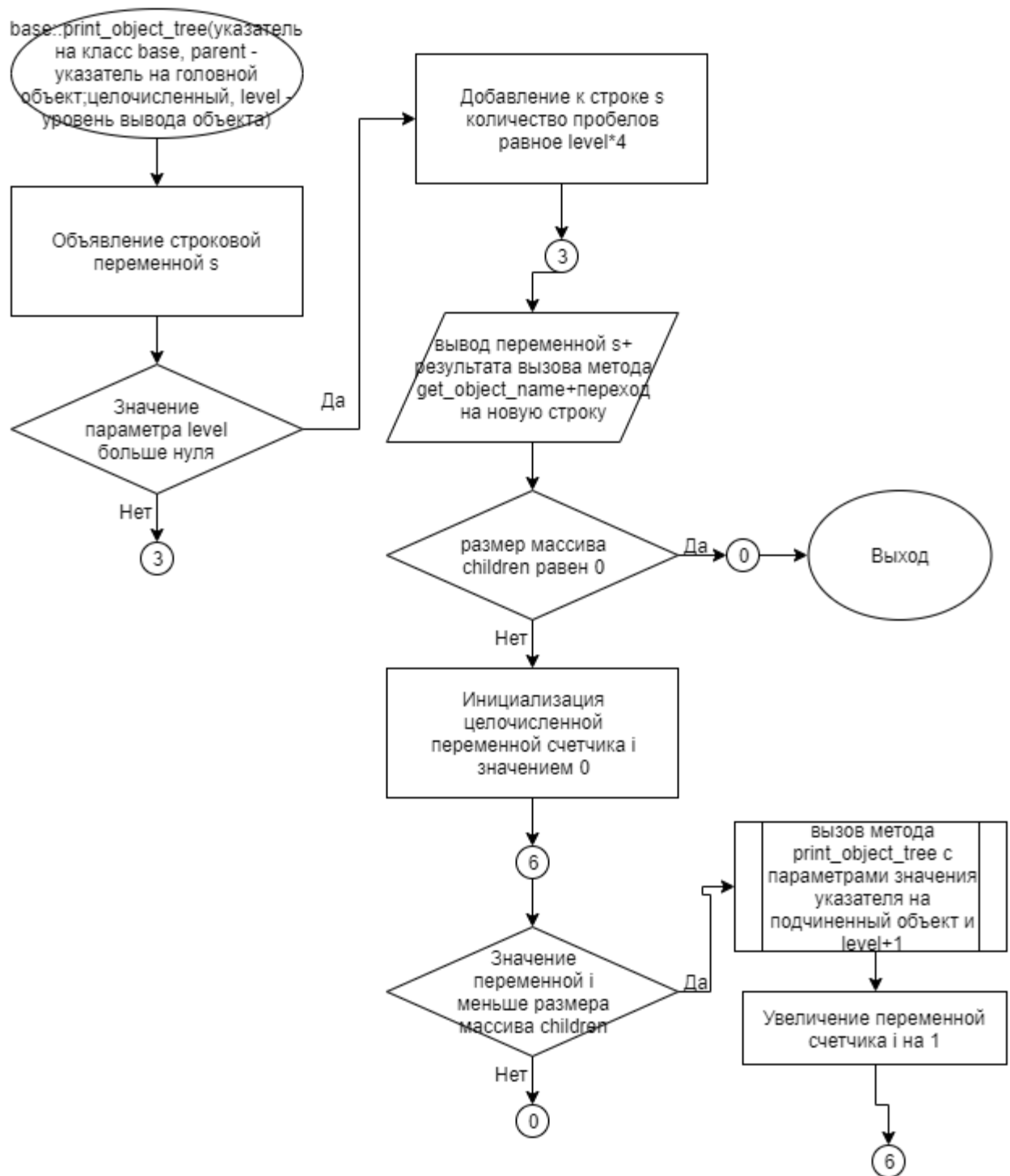


Рисунок 8 – Блок-схема алгоритма

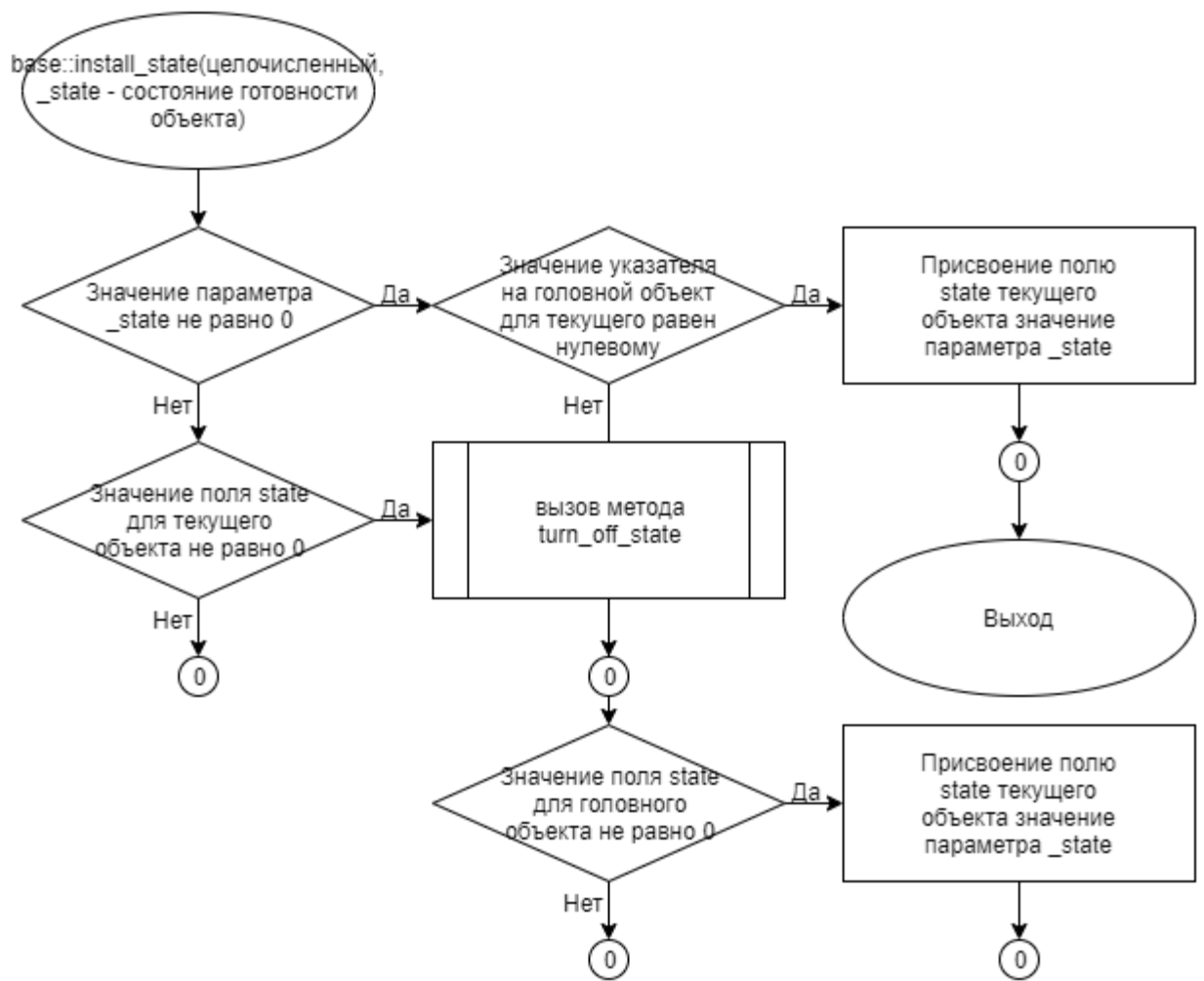


Рисунок 9 – Блок-схема алгоритма

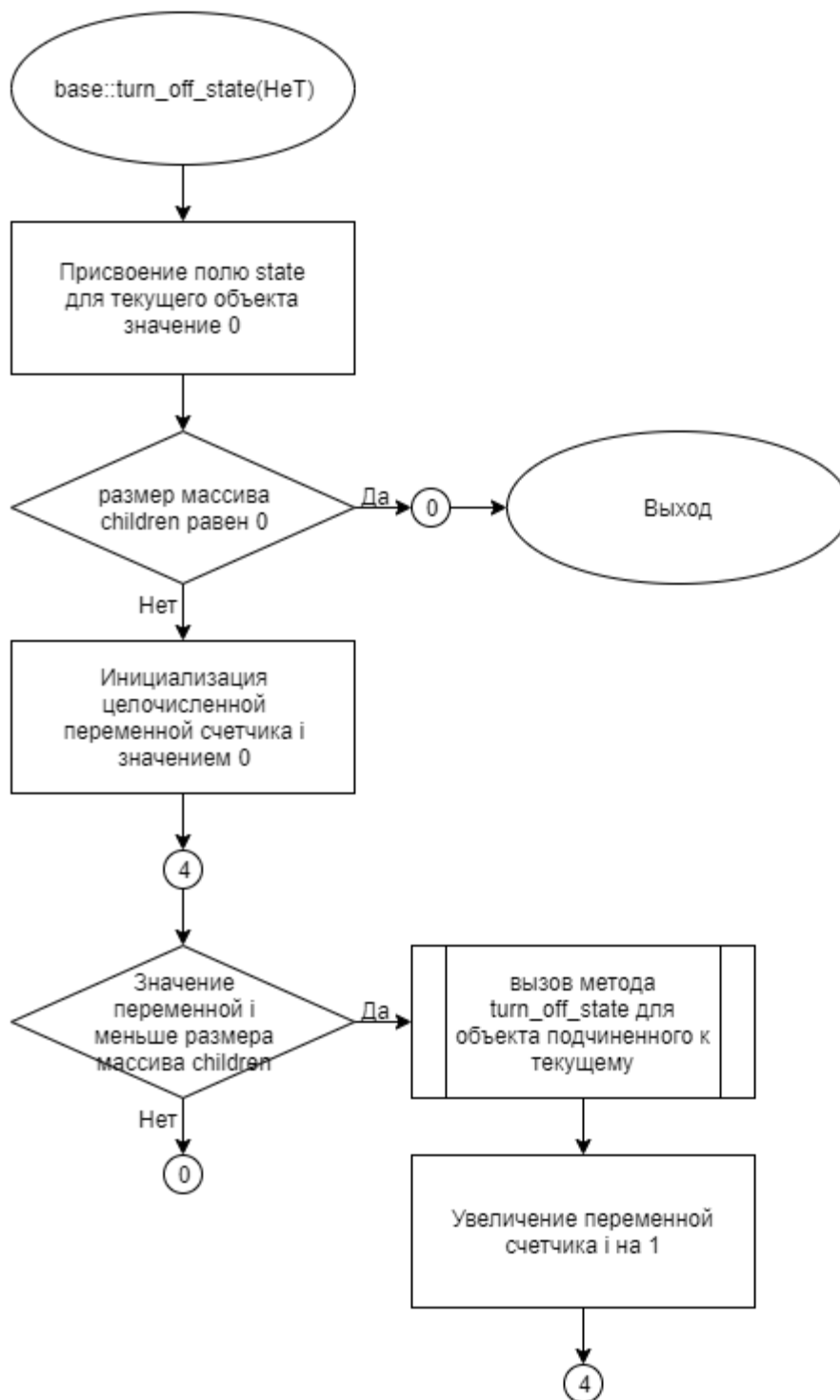


Рисунок 10 – Блок-схема алгоритма

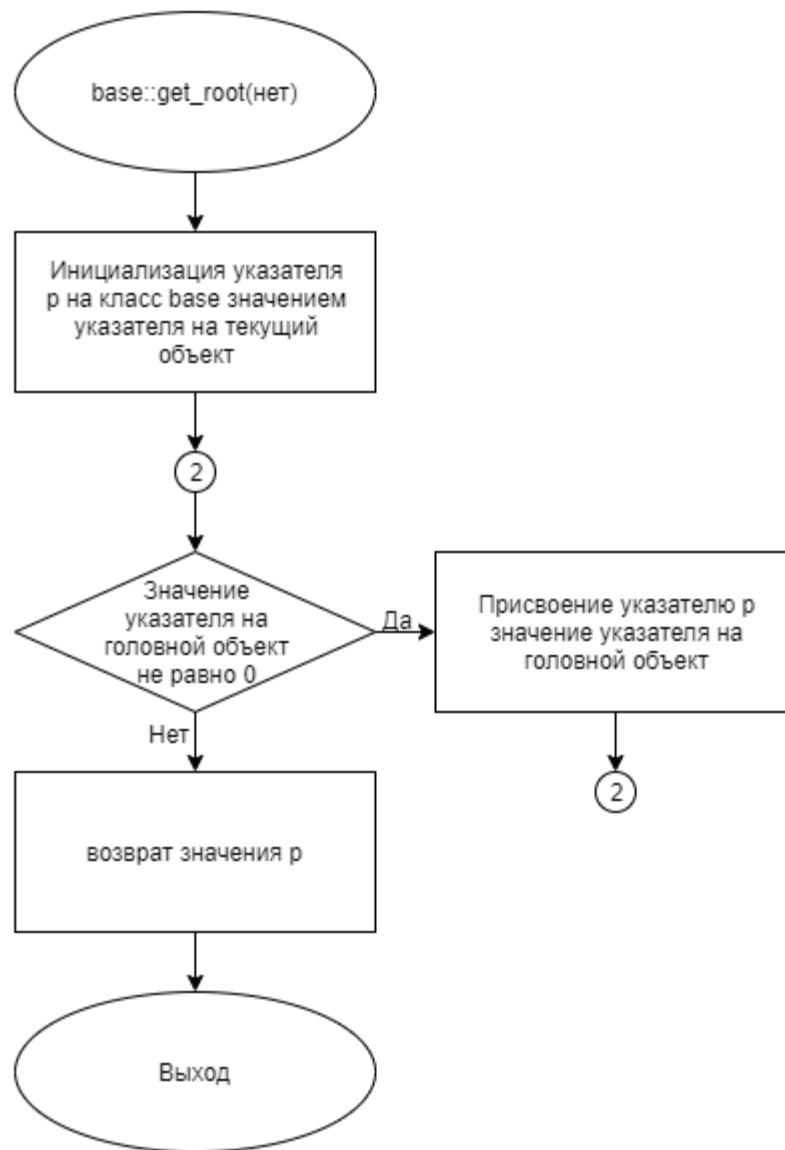


Рисунок 11 – Блок-схема алгоритма

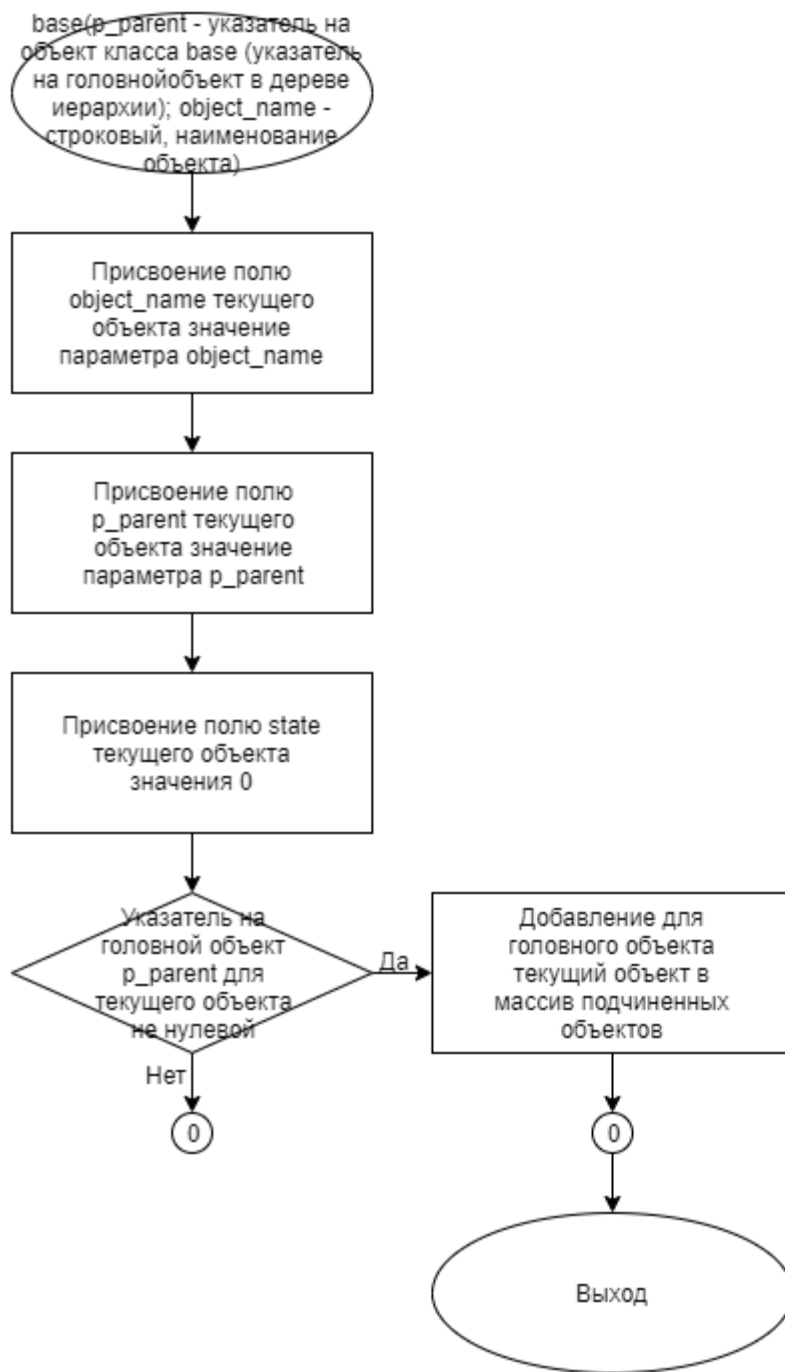


Рисунок 12 – Блок-схема алгоритма

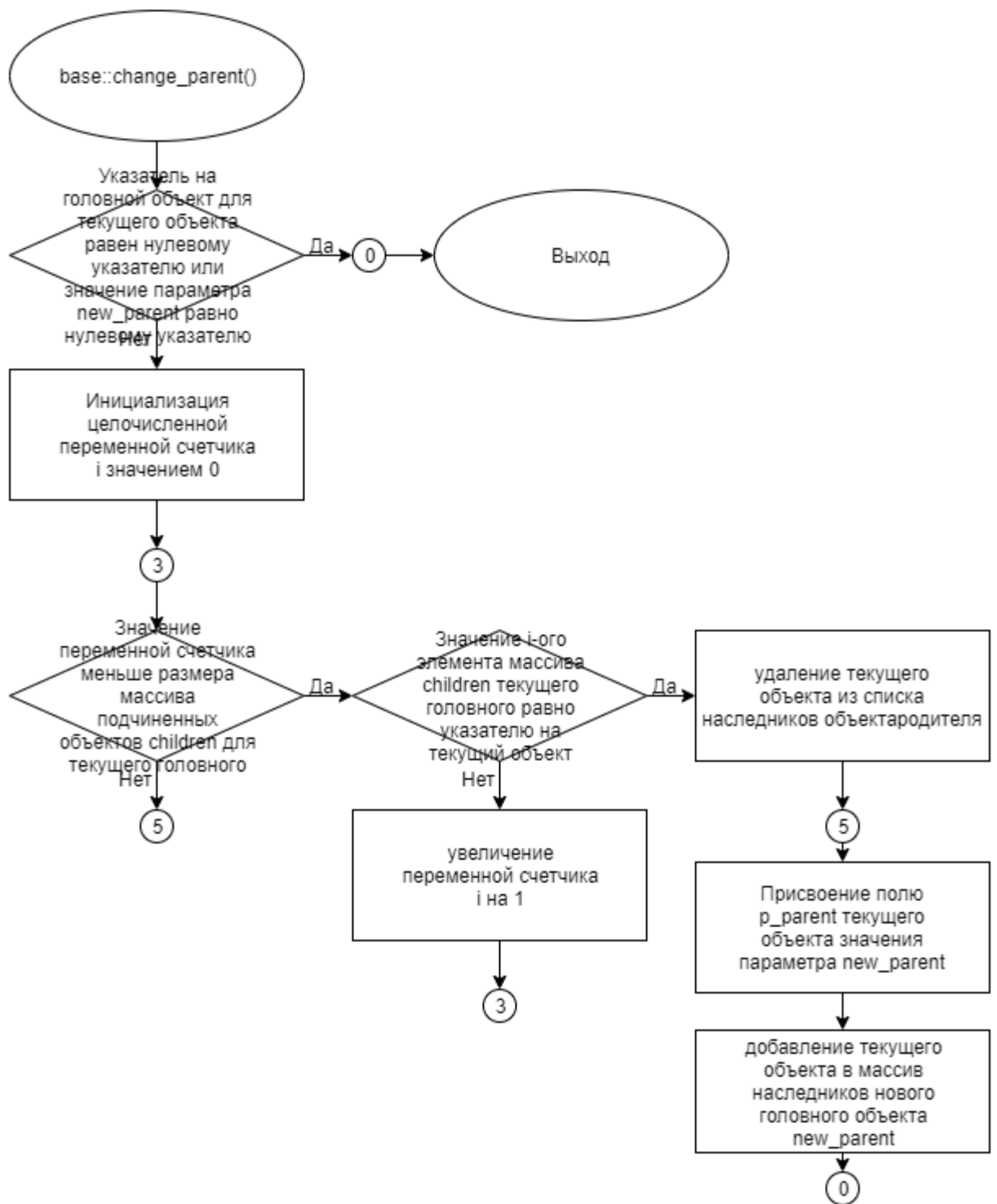


Рисунок 13 – Блок-схема алгоритма

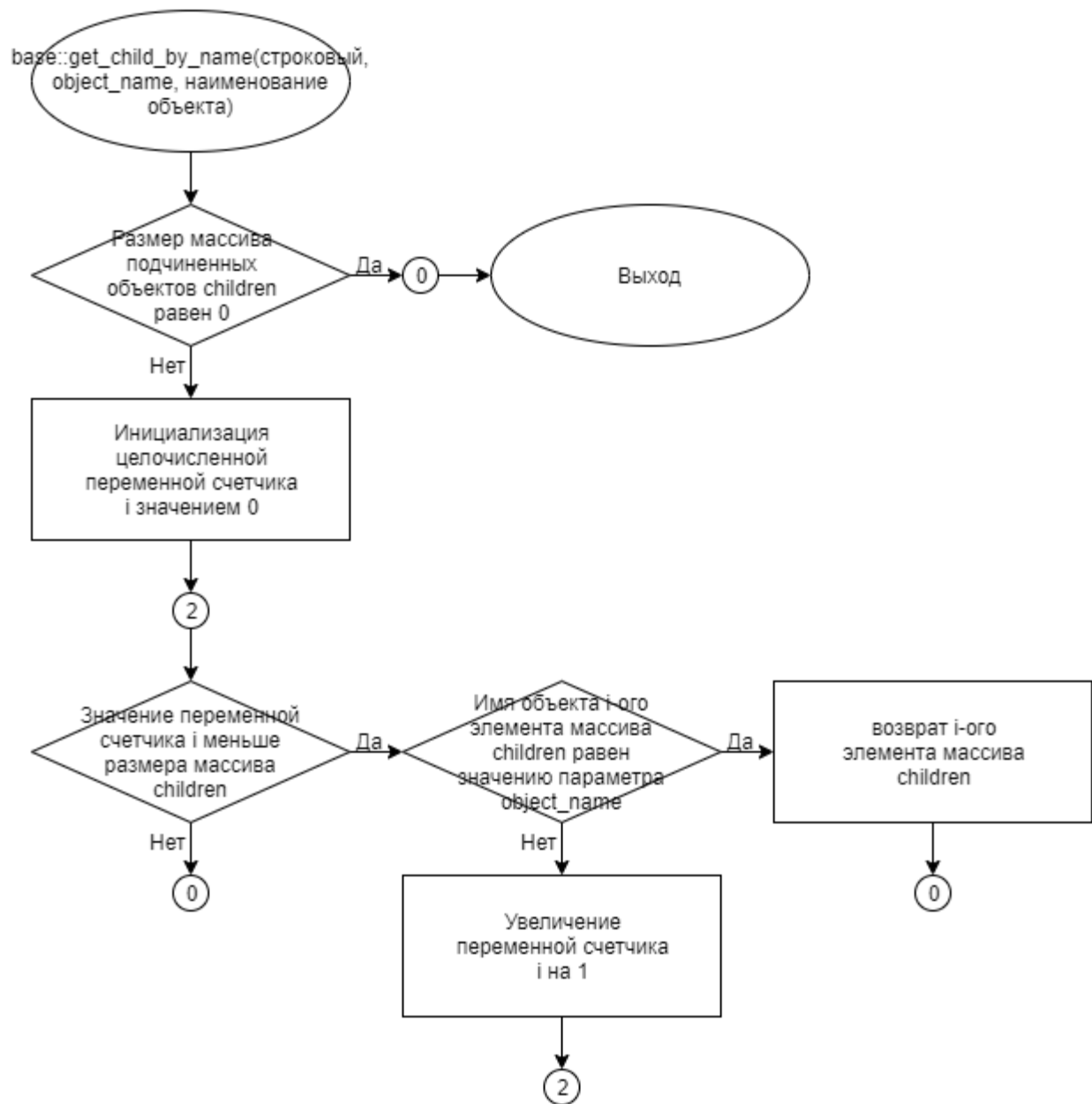


Рисунок 14 – Блок-схема алгоритма



Рисунок 15 – Блок-схема алгоритма

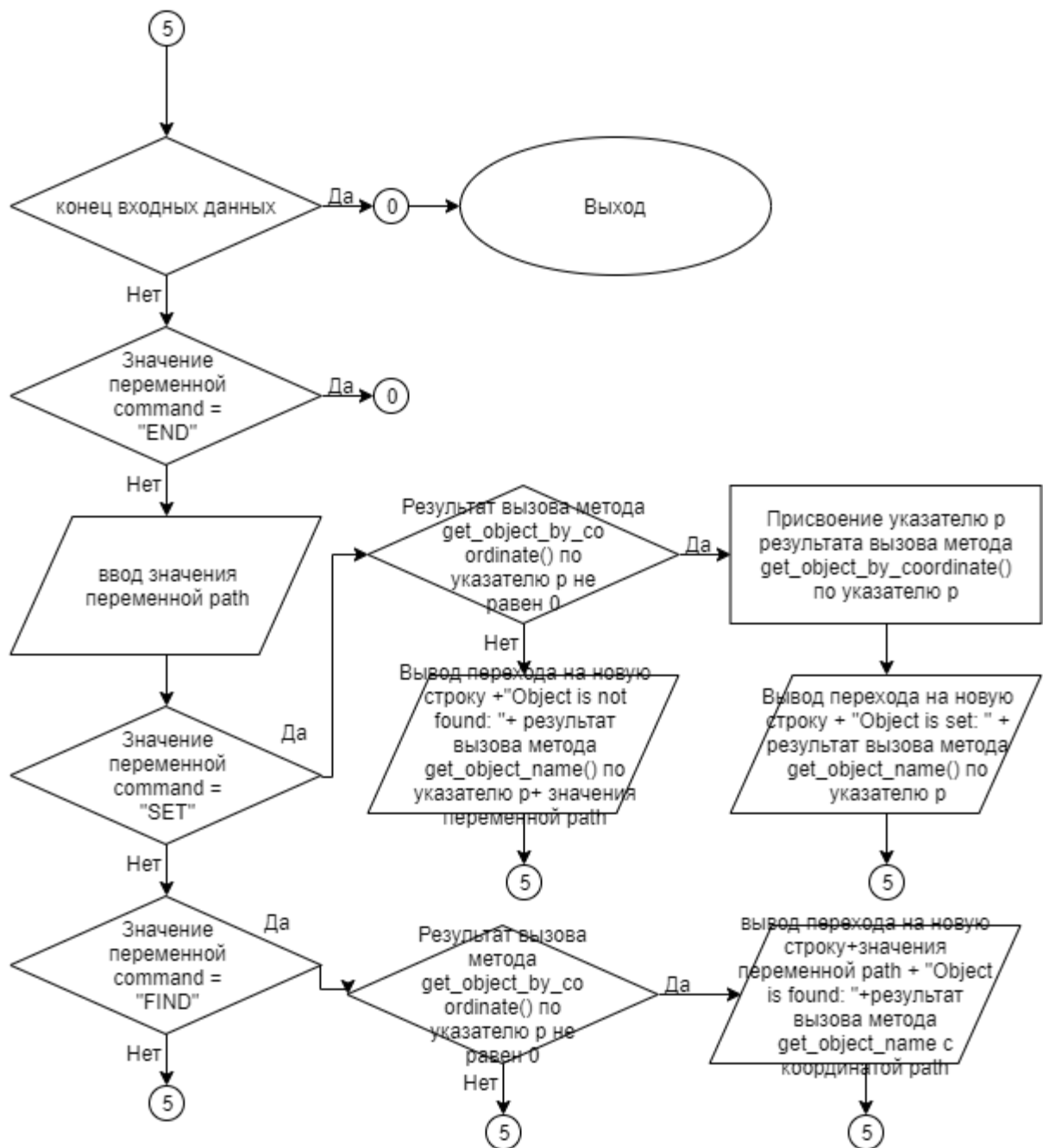


Рисунок 16 – Блок-схема алгоритма



Рисунок 17 – Блок-схема алгоритма

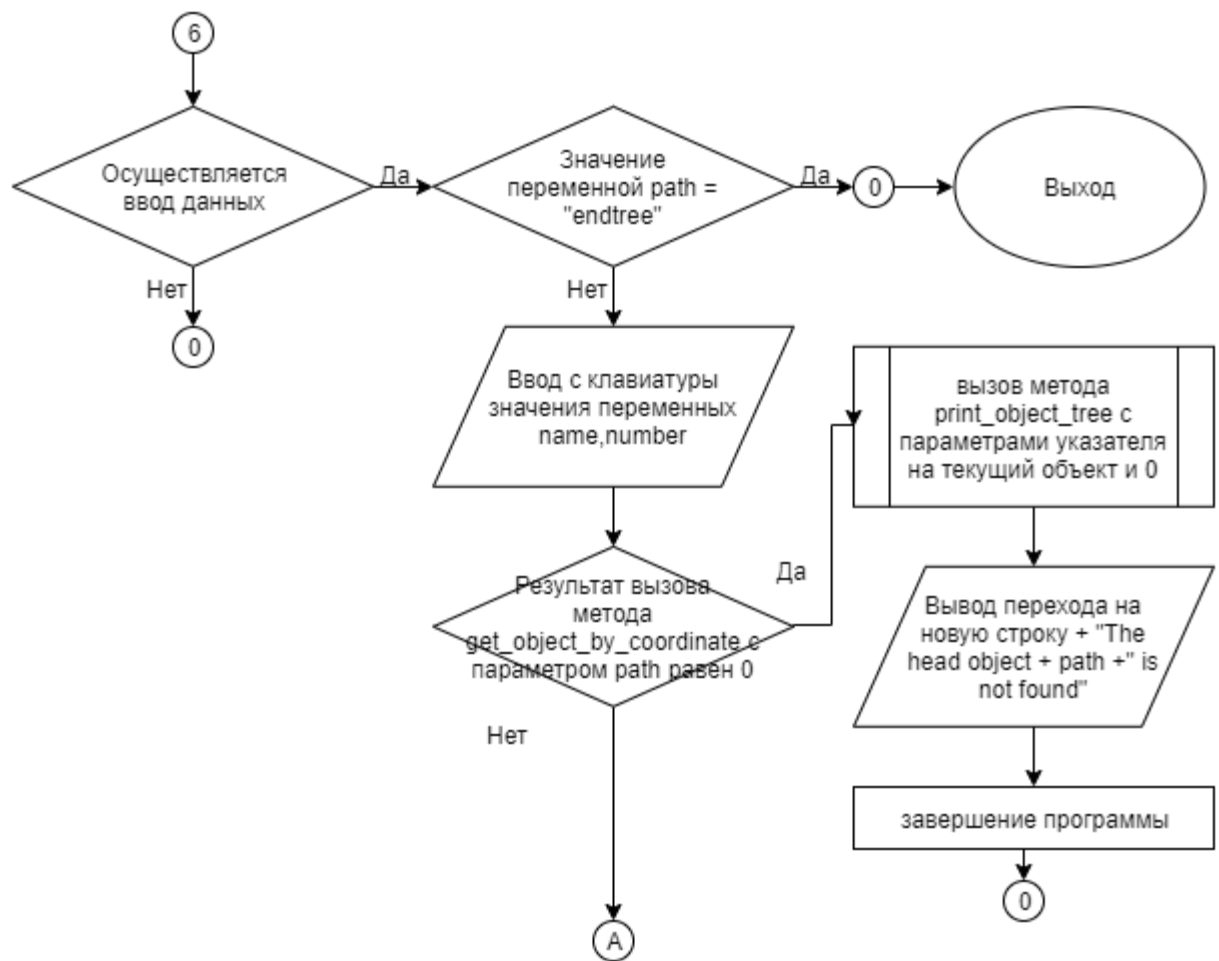


Рисунок 18 – Блок-схема алгоритма

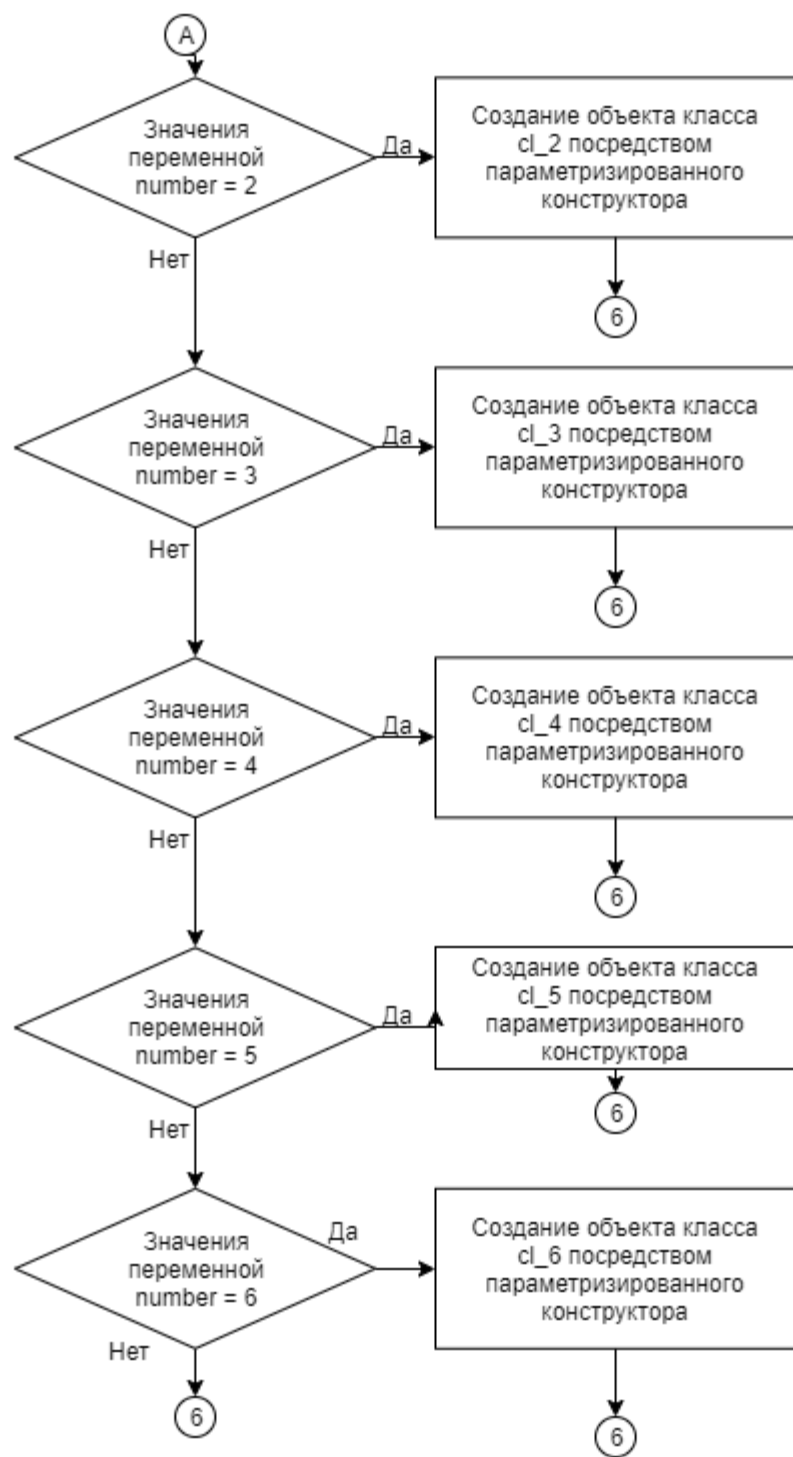


Рисунок 19 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.0 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
application::application(base* p_parent):base(p_parent)
{}

void application:: build_tree_objects()
{
    string path, name;
    int number;
    cin>>name;
    this->set_object_name(name);
    base* obj;
    string s;
    while (cin>>path)
    {
        if (path == "endtree")
            return ;
        if (!get_object_by_coordinate(path))
        {
            cout<<"Object tree"<<endl;
            print_object_tree(this,0);
            cout <<endl<<"The head object "<< path <<" is not found";
            exit(0);
        }
        else
        {
            cin>>name>>number;
            if (number == 2) obj = new cl_2(get_object_by_coordinate(path),
name);
            else if (number == 3) obj = new
cl_3(get_object_by_coordinate(path), name);
            else if (number == 4) obj = new
cl_4(get_object_by_coordinate(path), name);
            else if (number == 5) obj = new
cl_5(get_object_by_coordinate(path), name);
            else if (number == 6) obj = new
cl_6(get_object_by_coordinate(path), name);
        }
    }
}

int application::exec_app()
```

```

{
    cout<<"Object tree"<<endl;
    print_object_tree(this,0);
    string command,path;
    base* p = this;
    while (cin>>command)
    {
        if (command == "END")
            return 0;
        cin>>path;
        if (command == "SET")
        {
            if (p-> get_object_by_coordinate(path))
            {
                p = p->get_object_by_coordinate(path);
                cout<<endl<<"Object is set: "<<p->get_object_name();
            }
            else
                cout<<endl<<"Object is not found: "<<p->get_object_name()<<' ' <<path;
        }
        if (command == "FIND")
        {
            if (p-> get_object_by_coordinate(path))
                cout<<endl<<path<<" Object name: "<<p->get_object_by_coordinate(path)->get_object_name();
            else
                cout<<endl<<path<<" Object is not found";
        }
    }
    return 0;
}

```

5.1 Файл application.h

Листинг 2 – application.h

```

#ifndef APPLICATION_H_
#define APPLICATION_H_
#include <string>
#include "base.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
class application : public base
{
public:
    application(base* p_parent);
    void bild_tree_objects();// метод построения дерева иерархии
    int exes_app();//метод запуска приложения
};

```

```
#endif
```

5.2 Файл base.cpp

Листинг 3 – base.cpp

```
#include "base.h"
base::base(base* p_parent, string object_name)//конструктор базового класса
{
    this->object_name=object_name;
    this->p_parent=p_parent;
    state = 0;//отметка неготовности объекта
    if (p_parent)
        p_parent->children.push_back(this);
}

void base::set_object_name(string object_name)
{
    this->object_name = object_name;
}

string base::get_object_name()
{
    return object_name;
}

void base:: change_parent (base* new_parent)//переопределить головной объект для
текущего
{
    if (this ->p_parent==nullptr || new_parent == nullptr)
        return;
    for (int i = 0; i<p_parent ->children.size(); i++)
    {
        if (p_parent->children[i] == this)
        {
            p_parent->children.erase(p_parent-
>children.begin()+i);//удаление текущего объекта из списка наследников объекта-
родителя
            return ;
        }
    }
    this->p_parent;
    new_parent->children.push_back(this);//добавление текущего объекта в список
наследников нового головного объекта
}

base* base :: get_parent() //получение указателя на головной объект
{return p_parent;}

int base::get_state()//получить состояние объекта
{
    return state;
}
```

```

base* base::get_child_by_name(string object_name)
{
    if (children.size()==0)
        return nullptr;
    for( int i = 0; i<children.size(); i++)
    {
        if (children[i]->get_object_name()==object_name)
            return children[i];
    }
    return nullptr;
}

base* base::get_root()
{
    base* p = this;
    while (p->p_parent)
        p = p->p_parent;
    return p;
}

base* base::get_object_by_name(string name)
{
    return get_root()->find_object_by_name(name);
}

base* base::find_object_by_name(string name )
{
    base* p;
    if (this->object_name==name) return this;
    for (int i = 0; i<children.size();i++)
    {
        p = children[i]->find_object_by_name(name);
        if (p!= nullptr)
            return p;
    }
    return nullptr;
}

void base::install_state(int _state)
{
    if (_state)
    {
        if (p_parent==nullptr)
            state = _state;
        else if (p_parent->state)
            state = _state;
    }
    else if (state)
        turn_off_state();
}

void base::turn_off_state()
{
    this -> state = 0;
    if (children.size()==0)
        return;
    for (int i = 0; i<children.size();i++)
    {

```

```

        children[i]->turn_off_state();
    }
}

void base::print_object_tree(base* parent, int level)
{
    string s;
    if (level>0)
        s.append(4*level, ' ');
    if (parent== this)
        cout<<parent->get_object_name();
    else
        cout<<endl<<s<<parent->get_object_name();
    if (parent->children.size()==0)
        return;
    for (int i = 0; i<parent->children.size();i++)
        print_object_tree(parent->children[i],level+1);
}

void base::print_state(base* parent, int level)
{
    string s;
    if (level>0) s.append(4*level, ' ');
    cout<<endl<<s<<parent->get_object_name()<<' ';
    if (parent->get_state())
        cout<<"is ready";
    else
        cout<<"is not ready";
    if (parent->children.size()==0)
        return;
    for (int i =0; i<parent->children.size();i++)
        print_state(parent->children[i],level+1);
}

base* base::get_object_by_coordinate(string path)
{
    if (path == "/" )
        return get_root();
    if (path == ".")
        return this;
    if (path.find("//") == 0)
        return get_root()->get_object_by_name(path.substr(2,path.size()-2));
    if (path[0] == '/')
        return get_root()->get_object_by_coordinate(path.substr(1,path.size()-
1));
    int index = path.size();
    for (int i = 0; i< path.size(); i++)
    {
        if (path[i]=='/')
        {
            index = i;
            break;
        }
    }
    string s = path.substr(0,index);
    for (int i = 0; i< this->children.size(); i++)
    {

```

```

        if (children[i]->get_object_name() == s)
        {
            if (path.size()==index)
                return children[i];
            else
                return children[i]-
>get_object_by_coordinate(path.substr(index+1,path.size() - index - 1));
        }
    }
    return nullptr;
}

base::~~base()
{
    for(int i = 0; i<children.size();i++)
        delete children[i];
}

```

5.3 Файл base.h

Листинг 4 – base.h

```

#ifndef BASE_H_
#define BASE_H_
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class base
{
private:
    string object_name;//наименование объекта
    base* p_parent;//указатель на головной объект
    int state; //состояние объекта
    vector <base*>children;//массив указателей на объекты, подчиненные к текущему
    объекту в дереве иерархии
    base* get_root();//получение указателя на корневой объект
public:
    base( base* p_parent , string object_name = "base" );
    void set_object_name(string object_name);
    string get_object_name();//получить имя объекта
    void change_parent( base* new_parent);//переопределить головной объект для
    текущего
    base* get_parent();//получить указатель на головной объект для текущего
    int get_state ();//получить состояние объекта
    base* get_child_by_name(string object_name);//получить указатель на объект потомок
    по имени объекта
    base* get_object_by_name(string name);//получить указатель на объект по имени
    base* get_object_by_coordinate(string path);
    base* find_object_by_name(string name);//поиск объекта на дереве иерархии по имени
    void print_object_tree(base* parent, int level);//вывод дерева иерархии
    void install_state(int _state);//установка готовности объекта
    void turn_off_state();//отключение готовности объекта

```

```
void print_state(base* parent, int level); //вывод дерева иерархии объектов и
отметок их готовности
~base(); //деструктор
};
#endif
```

5.4 Файл cl_1.cpp

Листинг 5 – cl_1.cpp

```
#include "cl_1.h"
cl_1::cl_1(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.5 Файл cl_1.h

Листинг 6 – cl_1.h

```
#ifndef CL_1_H
#define CL_1_H
#include "base.h"
class cl_1: public base
{
public:
cl_1(base* p_parent, string object_name);
};
#endif
```

5.6 Файл cl_2.cpp

Листинг 7 – cl_2.cpp

```
#include "cl_2.h"
cl_2::cl_2(base* p_parent, string object_name):base(p_parent, object_name)
{}
```

5.7 Файл cl_2.h

Листинг 8 – cl_2.h

```
#ifndef CL_2_H
#define CL_2_H
```



```
#include "base.h"
class cl_2: public base
{
public:
cl_2(base* p_parent, string object_name);
};
#endif
```

5.8 Файл cl_3.cpp

Листинг 9 – cl_3.cpp

```
#include "cl_3.h"
cl_3::cl_3(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.9 Файл cl_3.h

Листинг 10 – cl_3.h

```
#ifndef CL_3_H
#define CL_3_H
#include "base.h"
class cl_3: public base
{
public:
cl_3(base* p_parent, string object_name);
};
#endif
```

5.10 Файл cl_4.cpp

Листинг 11 – cl_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.11 Файл cl_4.h

Листинг 12 – cl_4.h

```
#ifndef CL_4_H
#define CL_4_H
#include "base.h"
class cl_4: public base
{
public:
    cl_4(base* p_parent, string object_name);
};
#endif
```

5.12 Файл cl_5.cpp

Листинг 13 – cl_5.cpp

```
#include "cl_5.h"
cl_5::cl_5(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.13 Файл cl_5.h

Листинг 14 – cl_5.h

```
#ifndef CL_5_H
#define CL_5_H
#include "base.h"
class cl_5: public base
{
public:
    cl_5(base* p_parent, string object_name);
};
#endif
```

5.14 Файл cl_6.cpp

Листинг 15 – cl_6.cpp

```
#include "cl_6.h"
cl_6::cl_6(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.15 Файл cl_6.h

Листинг 16 – cl_6.h

```
#ifndef CL_6_H
#define CL_6_H
#include "base.h"
class cl_6: public base
{
public:
cl_6(base* p_parent, string object_name);
};
#endif
```

5.16 Файл main.cpp

Листинг 17 – main.cpp

```
#include "application.h"
int main()
{
application ob_application(nullptr); //создание объекта класса application
ob_application.build_tree_objects(); //построение дерева иерархии
return ob_application.exec_app(); //запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 19.

Таблица 19 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
<pre> root / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 SET /object_2 FIND //object_5 FIND /object_15 FIND . FIND object_4/object_7 END </pre>	<pre> Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7 </pre>	<pre> Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7 </pre>

ЗАКЛЮЧЕНИЕ

За курс объектно-ориентированного программирования я научился: разрабатывать базовый класс для объектов, определять общий функционал для используемых в рамках приложения объектов, разрабатывать операции добавления, удаления, изменения позиции объекта в рамках иерархического дерева, освоил: алгоритмы обработки структур данных в виде дерева, построение дерева иерархии объектов, переключение состояния объектов или по имени, организацию связей между сигналами и обработчиками объектов, выдачу сигналов от объекта и отработку обработчиков, реализацию алгоритма решения задачи посредством последовательной отправки сигналов и отработку соответствующих обработчиков, реализацию алгоритма решения задачи посредством последовательной отправки сигналов и отработку соответствующих обработчиков, моделирование движения падающих шариков, описание метода решения программы, написание алгоритма, создание блок-схем. Так-же хочу от всех души поблагодарить своих преподавателей: Путуридзе Зураба Шотовича и Рыжёву Анастасию Андреевну за столь подробное объяснение материала и терпение

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-avvora.ru/student/files/methodicheskoe_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-avvora.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).