МИНОБРНАУКИ РОССИИ

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

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой _____
Подпись

Платонова О.В.
ФИО

« 14 » марта 2022г.

ЗАДАНИЕ
На выполнение курсовой работы
по дисциплине **«Объектно-ориентированное программирование»**

Студент Дмитриев Павел Вячеславович Группа ИКБО-33-21

Тема Моделирование работы инженерного арифметического калькулятора

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 _____ (Грач Е.П.)
Подпись *ФИО консультанта*
« 28 » февраля 2022 г.

Задание на курсовую работу получил _____ (Дмитриев П. В.)
Подпись *ФИО исполнителя*
« 28 » февраля 2022 г.

Москва 2022г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД). Все этапы решения задач курсовой работы фиксированы, соответствуют методике разработки объектно-ориентированных программ [1-2] и требованиям, приведенным в методическом пособии для проведения практических заданий контрольных и курсовых работ по дисциплине "Объектно-ориентированное-программирование" [3-5].

Цель работы: повышение практических навыков в области проектирование и реализации задач на основных принципах объектно-ориентированного программирования.

Задача: разработать инженерный арифметический калькулятор. Реализовать алгоритм выполнения операций над целыми числами.

Инженерный арифметический калькулятор упрощает процесс выполнения операций над числами. Также калькулятор позволяет представить число в различных системах счисления.

Инженерный арифметический калькулятор используется студентами в решении более сложных задач, где калькулятор выполняет второстепенную роль вычисления чисел. Кроме того, он исключает вероятность вычислительной ошибки.

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 отрабатывает 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 отработки 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 поставленной задачи требуется использовать:

1. Объекты стандартных потоков ввода и вывода. Используются для ввода с клавиатуры и вывода на экран
2. Объект ob_cl_application класса cl_application
3. Условный оператор if
4. Операторы циклов for, while
5. Объекты классов Base, cl_application, class1, class2, class3, class4, class5, class6
6. Вектор пар для хранения данных

Класс Base:

- свойства / поля:
    - поле – хранение названия текущего объекта:
        - наименование – name;
        - тип - string;
        - модификатор доступа – private.
    - поле – хранение указателя на родителя для текущего объекта:
        - наименование – *parent;
        - тип – Base;
        - модификатор доступа – private.
    - поле – массив указателей на дочерние объекты текущего объекта:
        - наименование – children;
        - тип - vector;
        - модификатор доступа – private.
    - поле - хранение состояния объекта:

        - наименование – x;
        - тип – int;
        - модификатор доступа – private.
- функционал:
    - конструктор Base - параметризированный конструктор;
    - метод set_name - определение имени объекта;
    - метод get_name - получение имени объекта;
    - метод set_root – определение родителя;
    - метод get_root – получение указателя на родителя;
    - метод print_tree - вывод на экран веток дерева иерархии;
    - метод find_object - поиск объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr);
    - метод print_ready_tree - вывод дерева иерархии объектов и отметок их готовности;
    - метод set_ready - установка готовности объекта;
    - метод get_ready - возврат значения готовности объекта;
    - метод get_children – возврат вектора указателей на детей текущего объекта;
    - метод find_coordinate - возврат указателя на объект согласно пути, заданному пользователем;

Класс cl_application:

- функционал:
    - конструктор cl_application - параметризированный конструктор;
    - метод build_tree_objects - построение дерева иерархии объектов;
    - метод exec_app - запуск приложения.

Класс class1 - наследуется публично от класса Base:

- функционал:
    - конструктор class1 – параметризированный конструктор.

Класс class2 - наследуется публично от класса Base:

- функционал:
    - конструктор class2 – параметризированный конструктор.

Класс class3 - наследуется публично от класса Base:

- функционал:
    - конструктор **class3** – параметризированный конструктор.

Класс class4 - наследуется публично от класса Base:

- функционал:
    - конструктор **class4** – параметризированный конструктор.

Класс class5 - наследуется публично от класса Base:

- функционал:
    - конструктор **class5** – параметризированный конструктор.

Класс class6 - наследуется публично от класса Base:

- функционал:
    - конструктор class6 – параметризированный конструктор.

Таблица 1 – Иерархия наследования объектов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарии
1	Base	cl_application	public		2	
		class1	public		3	
		class2	public		4	
		class3	public		5	
		class4	public		6	
		class5	public		7	

Продолжение Таблицы 1

1		class6	public		8	
2	cl_application			Содержит метод построения дерева иерархии и метод запуска приложения		
3	class1			Класс с параметризированным конструктором первого объекта		
4	class2			Класс с параметризированным конструктором первого объекта		
5	class3			Класс с параметризированным конструктором первого объекта		
6	class4			Класс с параметризированным конструктором первого объекта		
7	class5			Класс с параметризированным конструктором первого объекта		
8	class6			Класс с параметризированным конструктором первого объекта		

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 функционал программы.

Параметры: отсутствуют.

Возвращаемое значение: результат алгоритма.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_cl_application класса cl_application	2
2		Вызов метода build_tree_objects класса cl_application	3
3		Возврат вызова метода exec_app класса cl_application	∅

3.2 Алгоритм конструктора класса Base

Функционал: параметризированный конструктор.

Параметры: имя текущего объекта и указатель на родителя.

Модификатор доступа: public

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса Base

№	Предикат	Действия	№ перехода
1		Присвоение name к именитекущего объекта	2

Продолжение Таблицы 3

2		Присвоение parent кродителю текущегообъекта	3
3	Указатель на родителя существует	Добавление текущегообъекта в массив объектов данного родителя	∅
			∅

3.3 Алгоритм метода set_name класса Base

Функционал: определение имени объекта.

Параметры: string name – имя объекта.

Возвращаемое значение: отсутствует.

Модификатор доступа: public

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_name класса Base

№	Предикат	Действия	№ перехода
1		Присвоение переменной name имени текущего объекта	∅

3.4 Алгоритм метода set_root класса Base

Функционал: определение родителя.

Параметры: new_parent - указатель на родителя.

Возвращаемое значение: отсутствует.

Модификатор доступа: public

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм метода set_root класса Base

№	Предикат	Действия	№ перехода
1	Указатель на родителя		2
			5
2	i<размер children		3
			5
3	Значение в массиве==значение текущего объекта	Удаление текущего значения из children	4
			2
4		i+1	2
5		Присвоение переменной parent значения переменной new_parent	6
6	Указатель на нового родителя существует	Добавление текущего объекта в массив объектов нового родителя	∅
			∅

3.5 Алгоритм метода get_name класса Base

Функционал: получение имени объекта.

Параметры: отсутствуют.

Возвращаемое значение: строка – имя объекта

Алгоритм конструктора представлен в таблице 6.

Таблица 6 – Алгоритм метода get_name класса Base

№	Предикат	Действия	№ перехода
1		Возврат значения переменной name	∅

3.6 Алгоритм метода get_root класса Base

Функционал: получение указателя на родителя.

Параметры: отсутствуют.

Модификатор доступа: public

Возвращаемое значение: указатель на родителя текущего объекта.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_root класса Base

№	Предикат	Действия	№ перехода
1		Возврат значения переменной parent	∅

3.7 Алгоритм метода print_tree класса Base

Функционал: вывод на экран веток дерева иерархии.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода print_tree класса Base

№	Предикат	Действия	№ перехода
1		Переход на новую строку,вывод значения параметра space и вызов метода get_name()	2
2	Перебор элементов в массиве children	Вызов метода print_tree(space + " ") для child	2
			∅

3.8 Алгоритм конструктора класса cl_application

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		Наследование от конструктора класса Base	∅

3.9 Алгоритм метода build_tree_objects класса cl_application

Функционал: построение дерева иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных parent_name и name	2
2		Объявление указателей parent и child класса Base	3
3		Считывание с клавиатуры значения переменной parent_name	4
4		Вызов метода set_name с параметром parent_name	5
5		Объявление целочисленной переменной num	6
6	true	Считывание с клавиатуры значения переменной parent_name	7
			∅
7	parent_name==" endtree"		∅
			8
8		Считывание с клавиатуры значений переменных name и num	9

Продолжение Таблицы 11

9		Присвоение переменной parent метода find_object(parent_name)	10
10	num == 2	child = new class1(name,parent)	6
			11
11	num == 3	child = new class2(name,parent)	6
			12
12	num == 4	child = new class3(name,parent)	6
			13
13	num == 5	child = new class4(name,parent)	6
			14
14	num == 6	child = new class5(name,parent)	6
			15

3.10 Алгоритм метода exec_app класса cl_application

Функционал: запуск приложения.

Параметры: отсутствуют.

Возвращаемое значение: Целочисленное значение - код возврата.

Алгоритм метода представлен в таблице 11.

Таблица 12 – Алгоритм метода exec_app класса cl_application

№	**Предикат**	**Действия**	**№ перехода**
1		Вывод сообщения "Object tree" на экран	2
2		Вызов метода print_tree("")	3
3			4
4			5

Продолжение таблицы 11

5	Считывание с клавиатуры object_name и status	Инициализация указателя find = find_object(object_name)	6
			7
6		Вызов метода set_ready(status) для find	5
7		Переход на новую строку и вывод сообщения "The tree of objects and their readiness" на экран	8
8		Вызов метода print_ready_tree("")	9
9		Возврат 0	∅

3.11 Алгоритм конструктора класса class1

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base* parent - указатель на родителя.

Алгоритм метода представлен в таблице 12.

Таблица 13 – Алгоритм конструктора класса class1

№	Предикат	Действия	№ перехода
1		Присвоение name к имени текущего объекта	2
2		Присвоение parent к родителю текущегообъекта	3
3	Указатель на родителя существует	Добавление текщего объекта в массив объектов данного родителя	∅
			∅

3.12 Алгоритм конструктора класса class2

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base* parent - указатель на родителя.

Алгоритм метода представлен в таблице 13.

Таблица 3 – Алгоритм конструктора класса class2

№	Предикат	Действия	№ перехода
1		Присвоение name к имени текущего объекта	2
2		Присвоение parent к родителю текущегообъекта	3
3	Указатель на родителя существует	Добавление текщего объекта в массив объектов данного родителя	∅
			∅

3.13 Алгоритм конструктора класса class3

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base* parent - указатель на родителя.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса class3

№	Предикат	Действия	№ перехода
1		Присвоение name к имени текущего объекта	2
2		Присвоение parent к родителю текущегообъекта	3
3	Указатель на родителя существует	Добавление текщего объекта в массив объектов данного родителя	∅
			∅

3.14 Алгоритм конструктора класса class4

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base* parent - указатель на родителя.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм конструктора класса class4

№	Предикат	Действия	№ перехода
1		Присвоение name к имени текущего объекта	2
2		Присвоение parent к родителю текущегообъекта	3
3	Указатель на родителя существует	Добавление текщего объекта в массив объектов данного родителя	∅
			∅

3.15 Алгоритм конструктора класса class5

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base* parent - указатель на родителя.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм конструктора класса class5

№	Предикат	Действия	№ перехода
1		Присвоение name к имени текущего объекта	2
2		Присвоение parent к родителю текущегообъекта	3
3	Указатель на родителя существует	Добавление текщего объекта в массив объектов данного родителя	∅
			∅

3.16 Алгоритм конструктора класса class6

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base* parent - указатель на родителя.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм конструктора класса class6

№	Предикат	Действия	№ перехода
1		Присвоение name к имени текущего объекта	2
2		Присвоение parent к родителю текущегообъекта	3
3	Указатель на родителя существует	Добавление текщего объекта в массив объектов данного родителя	∅
			∅

3.17 Алгоритм метода find_object класса Base

Функционал: поиск объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr).

Параметры: string name - имя объекта.

Возвращаемое значение: указатель на объект.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода find_object класса Base

№	Предикат	Действия	№ перехода
1	Имя текущего объекта == name	Возврат указателя на объект	∅
			2
2	перебор элементов массива children	Инициализация указателя res с вызовом метода find_object для элементов массива children	3
			4
3	res существует	Возврат res	2
			2
4		Возврат нулевого указателя	∅

3.18 Алгоритм метода print_ready_tree класса Base

Функционал: вывод дерева иерархии объектов и отметок их готовности.

Параметры: string space.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода print_ready_tree класса Base

№	Предикат	Действия	№ перехода
1		Переход на новую строку, вывод значения параметра space и вызов метода get_name()	2
2	x == false	Вывод сообщения " is not ready" на экран	4
			3
3		Вывод сообщения " is ready" на экран	4
4	перебор элементов в массиве children	Вызов метода print_ready_tree(space + " ") для child	4
			∅

3.19 Алгоритм метода set_ready класса Base

Функционал: установка готовности объекта.

Параметры: отсутствуют.

Возвращаемое значение: int x - готовность текущего объекта.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода set_ready класса Base

№	Предикат	Действия	№ перехода
1	!x		2
			6

Продолжение Таблицы 20

2	parent существует и готовность parent== 0	Возврат true	∅
			3
3	перебор элементов в массиве children	Для всех элементов в массиве children готовность == 0	3
			4
4		x == 0 для текущего объекта	5
5		Возврат true	∅
6	parent существует и готовность parent == 0	Возврат false	∅
			7
7		Присвоение x текущего объекта к x	8
8		Возврат true	∅

3.20 Алгоритм метода get_ready класса Base

Функционал: возврат значения готовности объекта.

Параметры: отсутствуют.

Возвращаемое значение: int x - готовность текущего объекта.

Алгоритм функции представлен в таблице 21.

Таблица 21 – Алгоритм метода get_ready класса Base

№	Предикат	Действия	№ перехода
1		Возврат значения готовности текущего объекта	∅

3.21 Алгоритм get_children класса Base

Функционал: возврат вектора указателей на детей текущего объекта.

Параметры: отсутствуют.

Возвращаемое значение: вектор указателей.

Алгоритм конструктора представлен в таблице 22.

Таблица 22 – Алгоритм конструктора класса Child2

№	Предикат	Действия	№ перехода
1		Возврат вектора children для текущего объекта	∅

3.22 Алгоритм метода find_coordinate класса Base

Функционал: возврат указателя на объект согласно пути, заданному пользователем.

Параметры: Base* this_obj, Base* root_obj, string coordinate.

Возвращаемое значение: указатель на Base.

Алгоритм конструктора представлен в таблице 23.

Таблица 23 – Алгоритм метода find_coordinate класса Base

№	Предикат	Действия	№ перехода
1	coordinate = ""	Возврат нулевого указателя	∅
			2
2	coordinate = "."	Возврат текущего объекта	∅
			3
3	coordinate = "/"	Возврат корневого объекта	∅
			4
4	coordinate[0], coordinate[1]='/'	Объявление строковой переменной name	5
			7
5	i<длина coordinate	В переменную name добавляется строка coordinate	5
			6
6		Возврат вызова метода find_object(name) для root_obj	∅

Продолжение Таблицы 23

7		Объявление указателя curr_obj	8
8		Объявление строковой переменной name	9
9		Объявление целочисленной переменной i	10
10	coordinate[0] == '/'	Присвоение переменной curr_obj значения переменной root_obj	11
			12
11		i = 1	12
12		Присвоение переменной curr_obj значения переменной this_obj	13
13		i = 0	14
14	i<длина coordinate		15
			25
15	coordinate[i]=='/' или i==длина coordinate-1		16
			24
16	i==длина coordinate-1	В переменную name добавляется строка coordinate[i]	17
			17
17	name == "."	Переход к следующей итерации цикла1	14
			18
18		Объявление переменной found типа bool и присвоение ей false	19
19	перебор детей curr_obj		20
			23
20	результат метода get_name() для pointer == name	curr_obj = pointer	21
			19
21		name = ""	22
22		found = true	19

Проодолжение Таблицы 23

23	found не существует	Возврат нулевого указателя	∅
			14
24		В переменную name добавляется строка coordinate[i]	14
25		Возврат curr_obj	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках ниже.

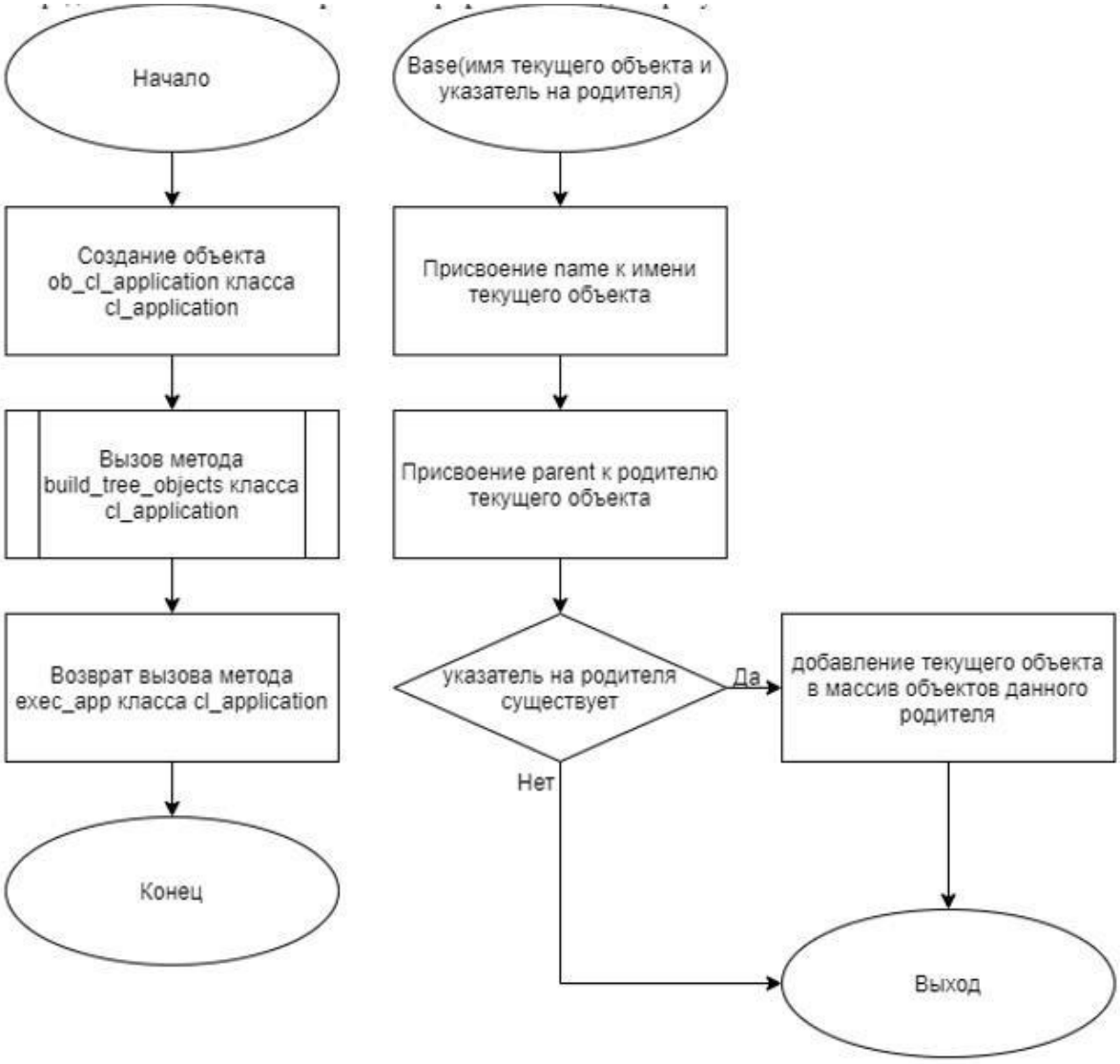

Рисунок 1 – Блок-схема алгоритма

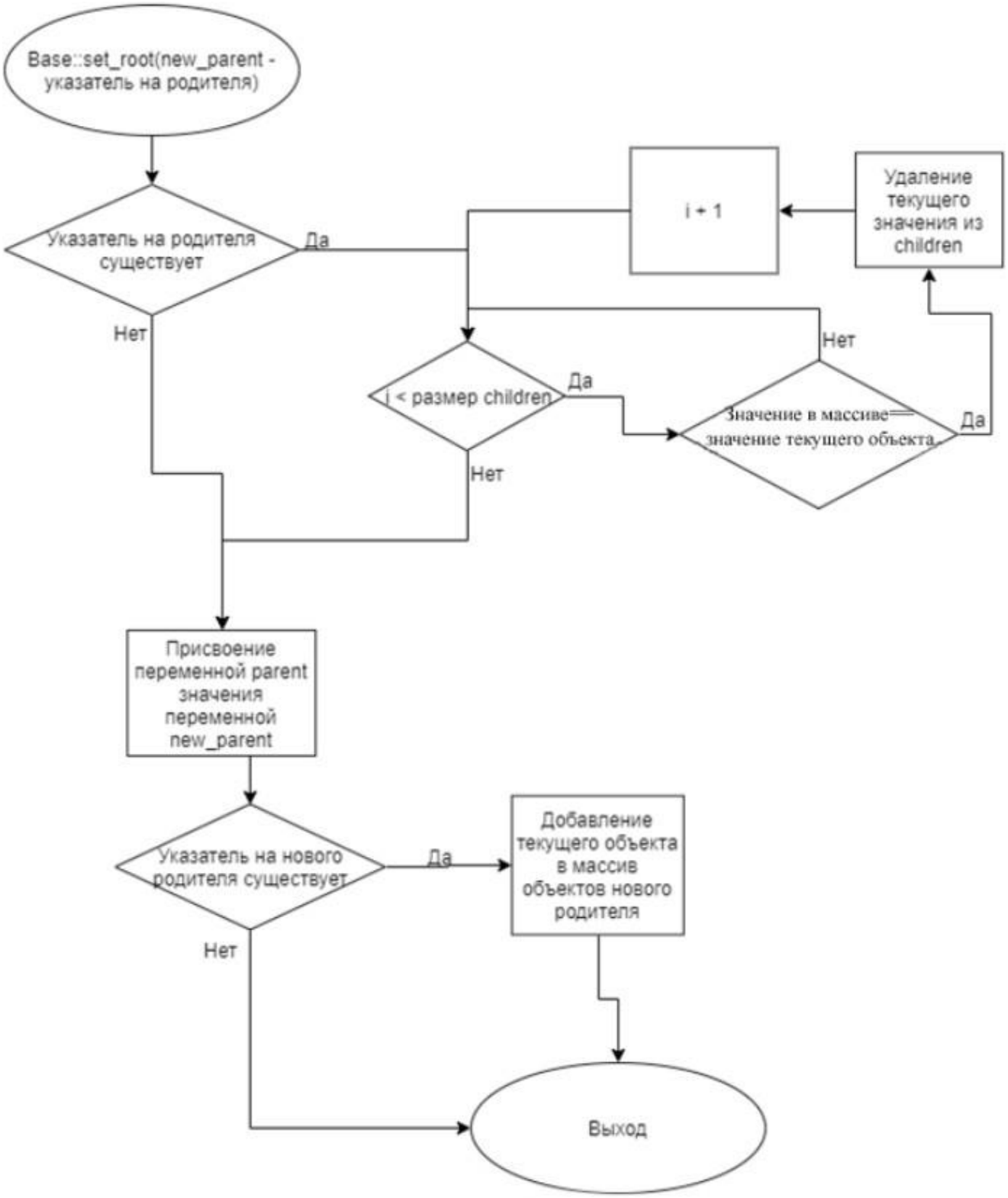

Рисунок 2 – Блок-схема алгоритма

Рисунок 3 – Блок-схема алгоритма

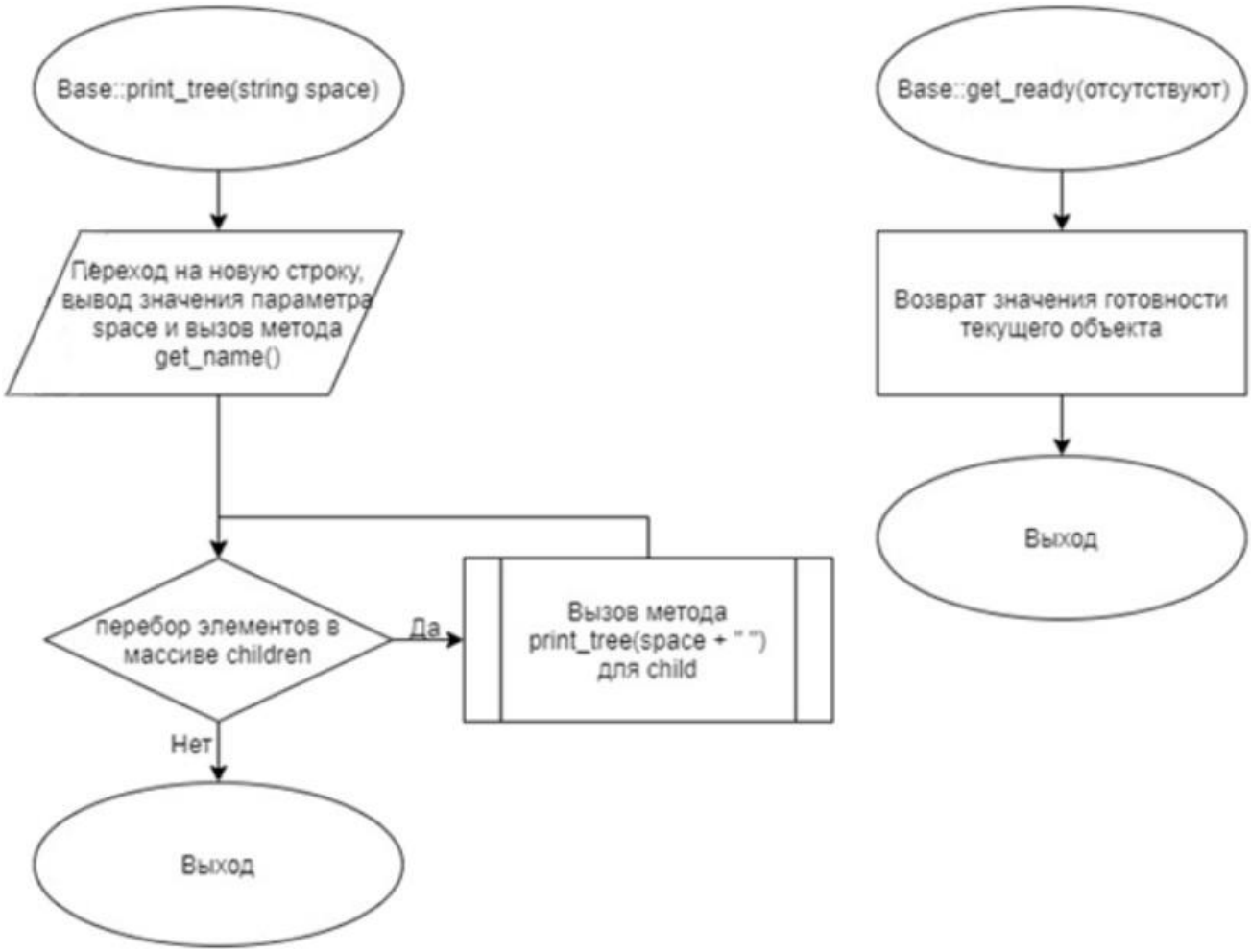

Рисунок 4 – Блок-схема алгоритма

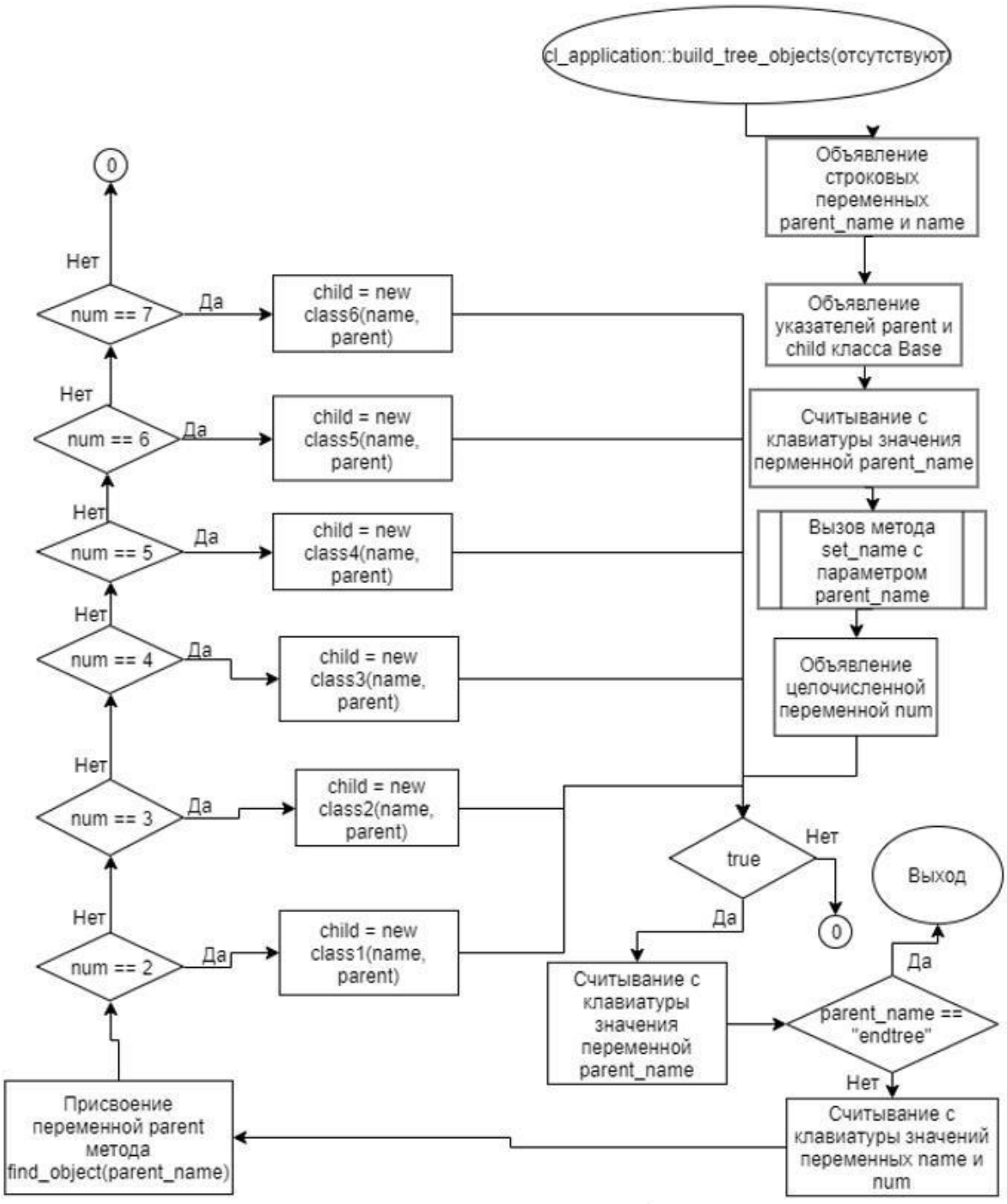

Рисунок 5 – Блок-схема алгоритма

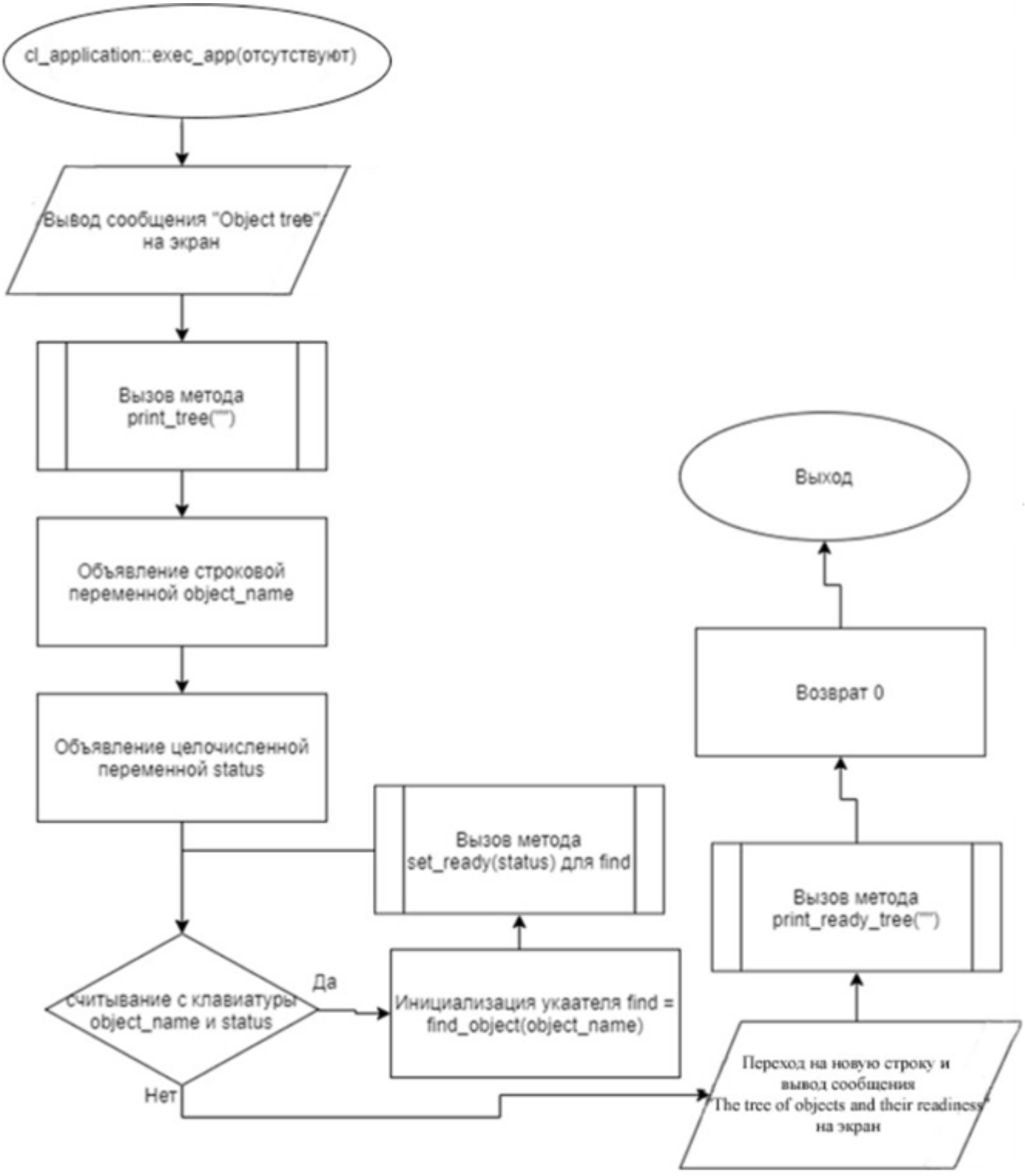

Рисунок 6 – Блок-схема алгоритма

Рисунок 7 – Блок-схема алгоритма

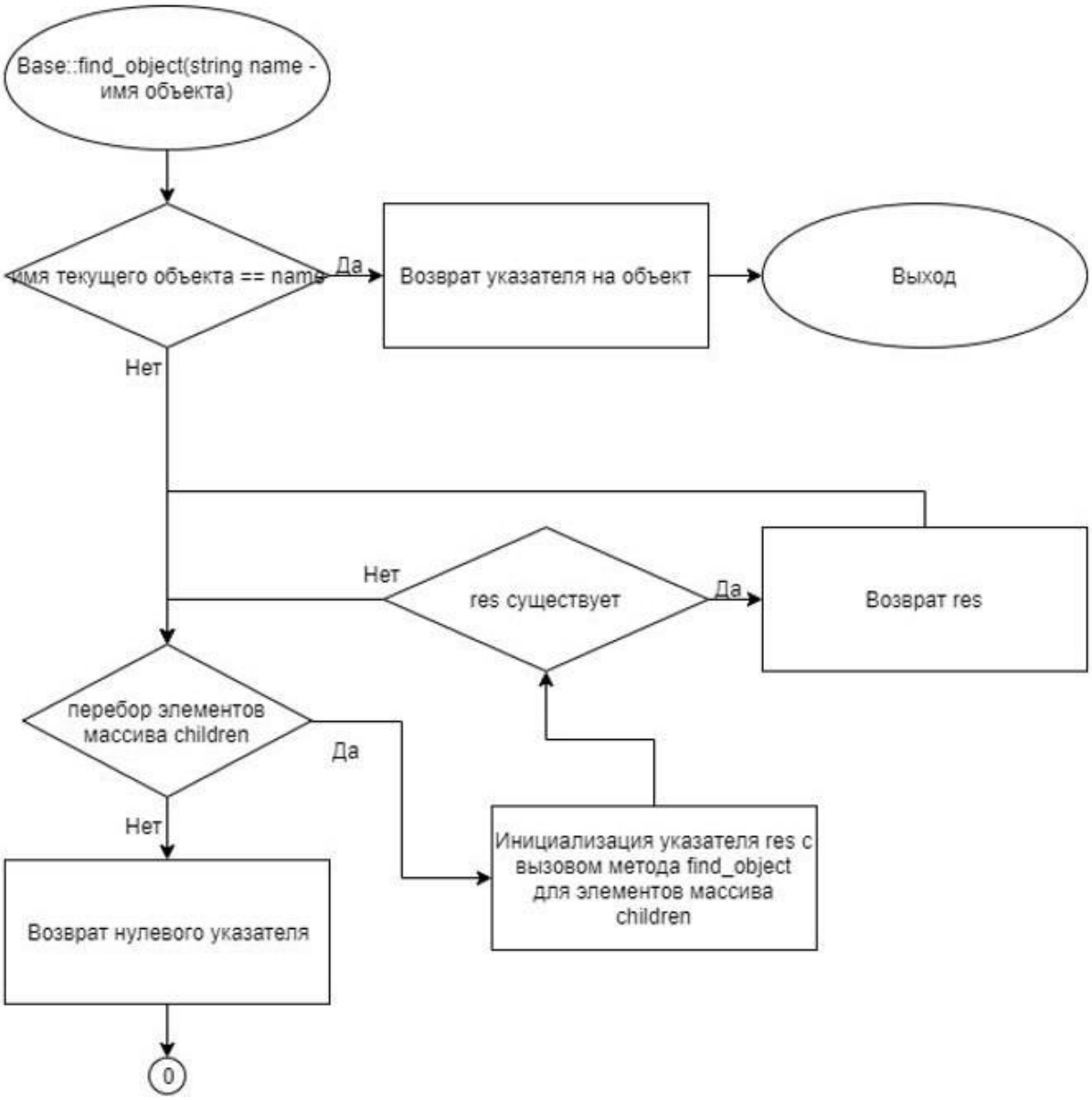

Рисунок 8 – Блок-схема алгоритма

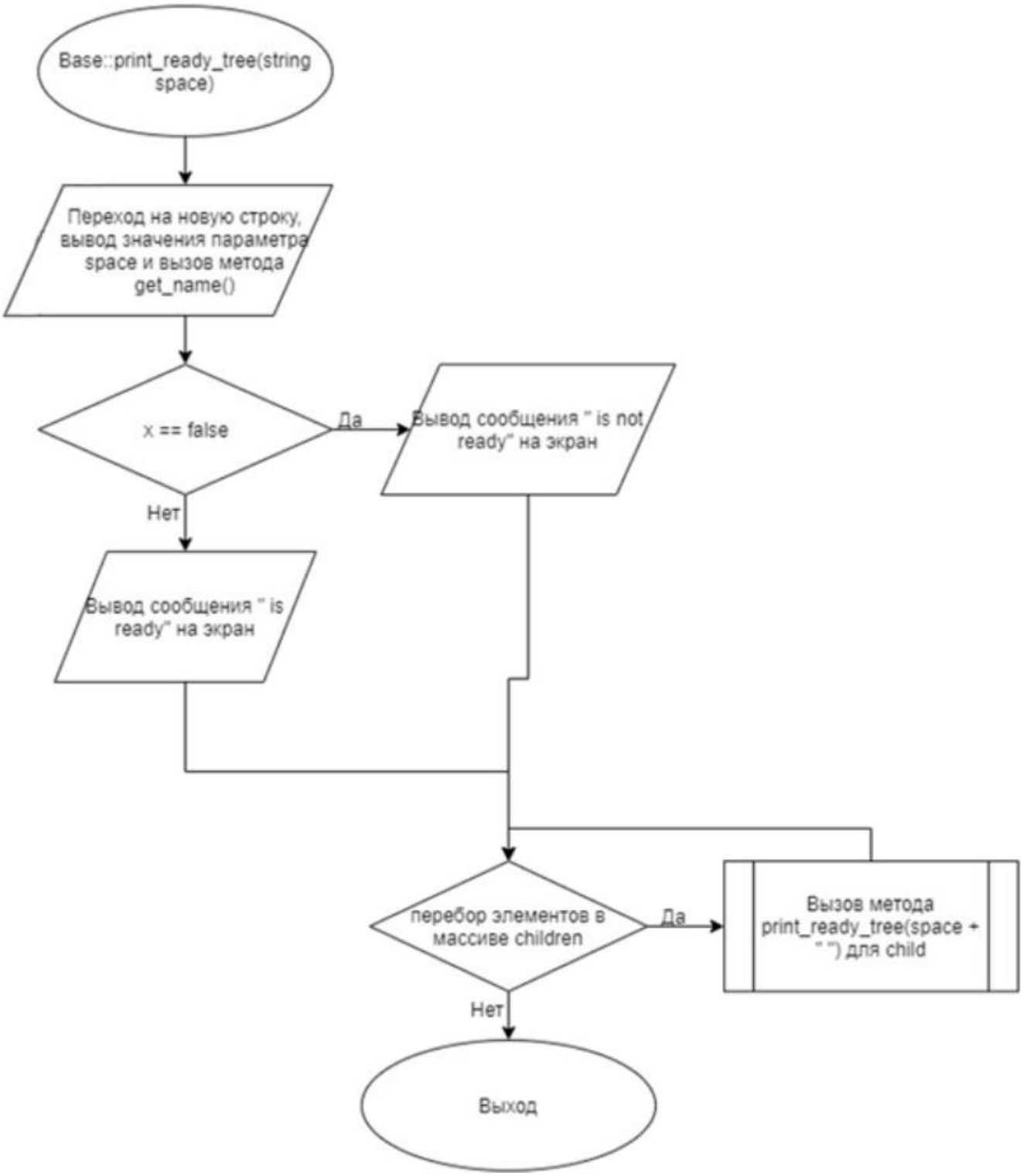

Рисунок 9 – Блок-схема алгоритма

Рисунок 10 – Блок-схема алгоритма

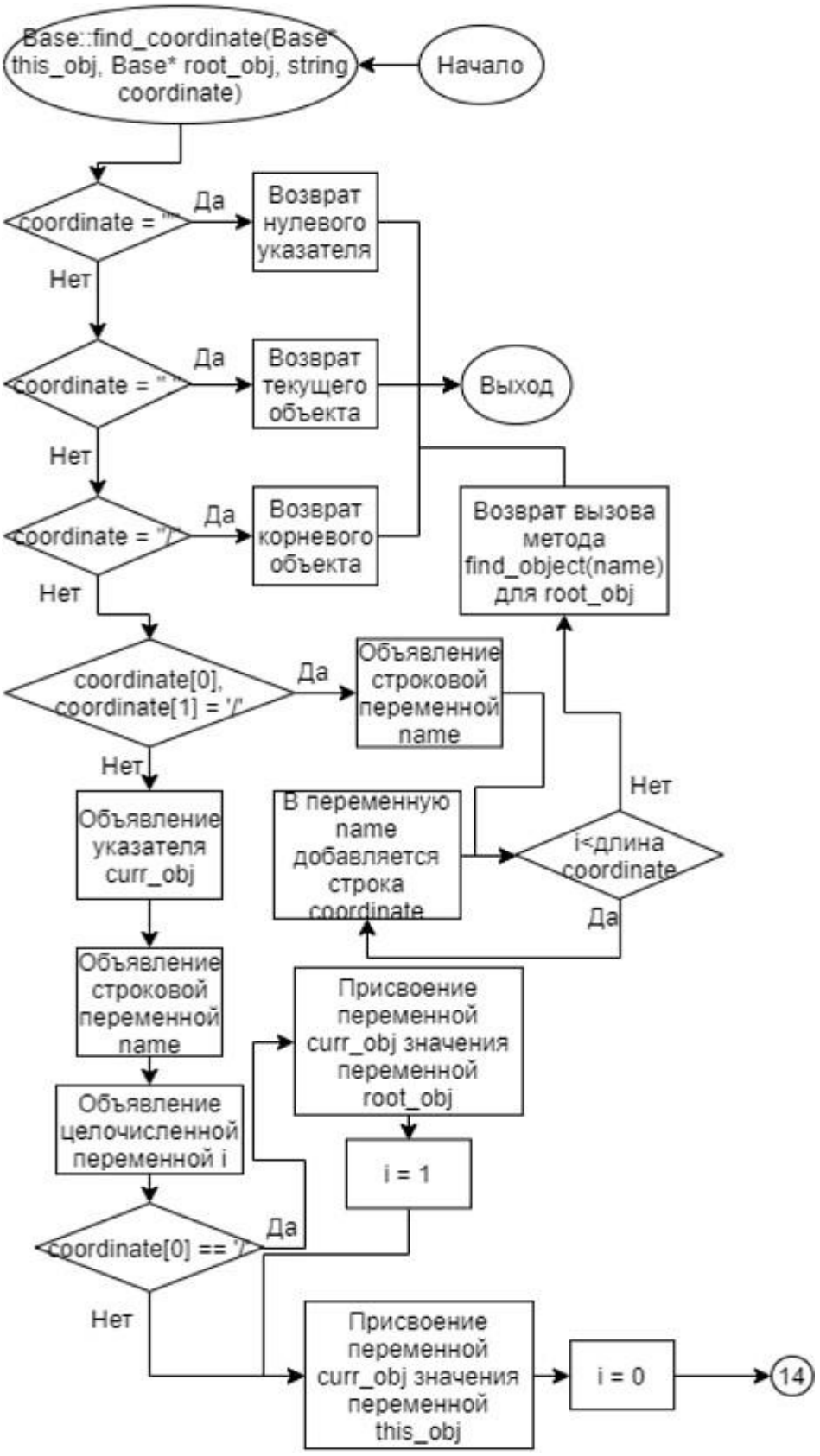

Рисунок 11 – Блок-схема алгоритма

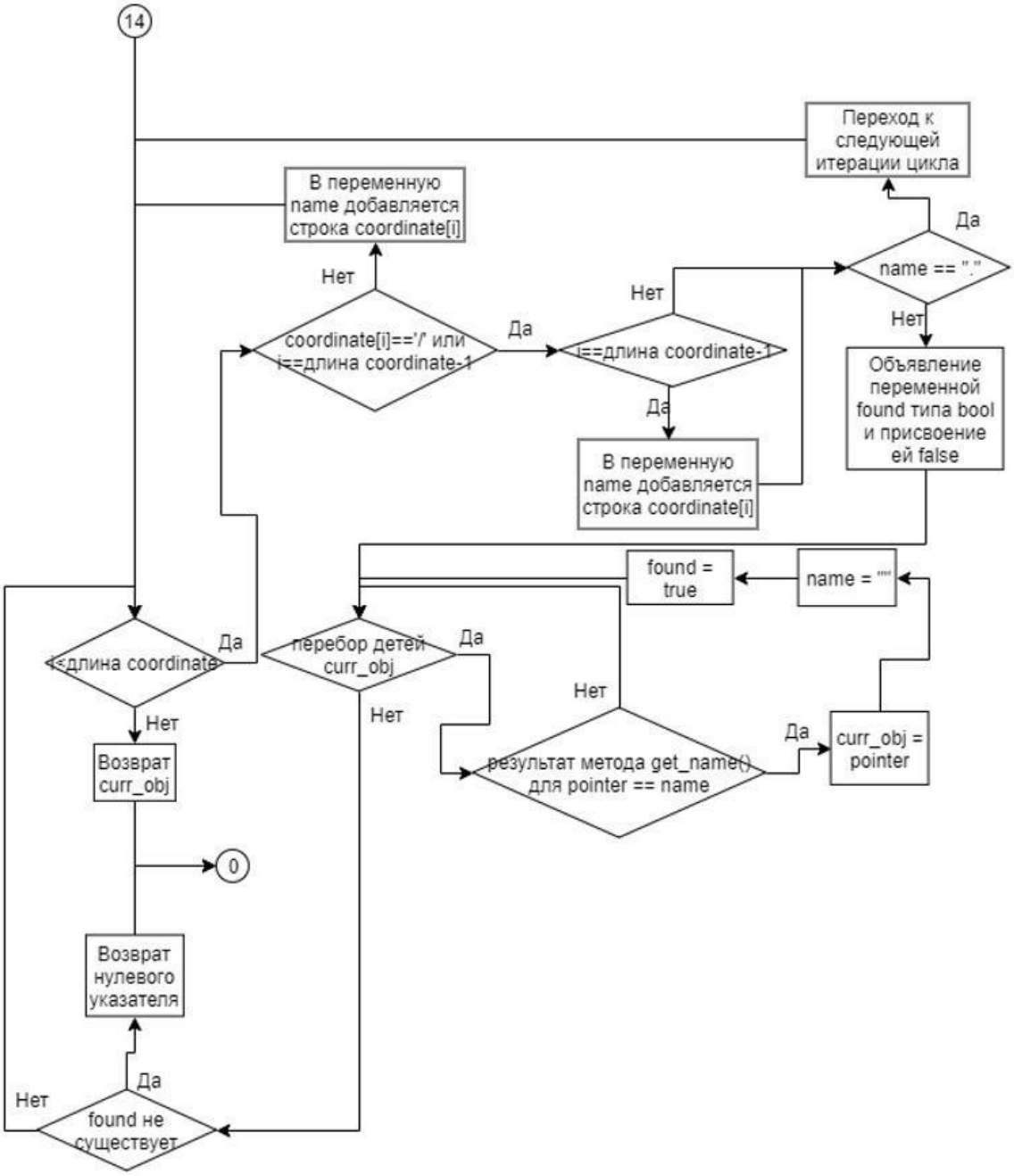

Рисунок 12 – Блок-схема алгоритма

Рисунок 13 – Блок-схема алгоритма

Исходный код программы представлен в приложении А, результаты тестирования представлены в приложении Б.

ЗАКЛЮЧЕНИЕ

В процессе выполнения работы было разработано и реализовано решение поставленной задачи. Описана четкая методология решения, составлена алгоритмическая и графическая составляющая задачи, сформирован исходный код задачи с логическим разделением на подзадачи и распределением этих подзадач по соответствующим файлам. Были проведены отладка и автоматизированное тестирование кода.

В результате выполнения курсовой работы были повышены практические навыки в области объектно-ориентированного программирования, работы с объектами классов, классами и их методами, и полями, логическими выражениями и операторами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» в системе «Аврора» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/js/pdf.js/web/viewer.html?file=/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 15.05.2022).

2. Грач Е. П. Видео-лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://online-edu.mirea.ru/course/view.php?id=3526 (дата обращения: 24.05.2022).

3. Лафоре Р. Объектно-ориентированное программирование в C++. Издательство: Питер СПб, 2018г.

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 24.05.2022).

5. Ингтем Ж. Г. Лекционные материалы по курсу «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://online-edu.mirea.ru/course/view.php?id=927 (дата обращения: 20.05.2022).

Приложение А

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Base.cpp

Листинг 1 – Base.cpp

```
#include "Base.h"
Base::Base(string name, Base* parent) {
	this->name = name;
	this->parent = parent;
	if (parent)
		parent->children.push_back(this);
	return;
}
void Base::set_name(string& name) {
	this->name = name;
	return;
}
string Base::get_name() {
	return name;
}
void Base::set_root(Base* new_parent) {
	if (parent)
		for (int i = 0; i < parent->children.size(); i++)
			if (parent->children[i] == this)
				parent->children.erase(parent -> children.begin() + i);
	parent = new_parent;
	if (new_parent)
		new_parent->children.push_back(this);
	return;
}
Base* Base::get_root() {
	return this->parent;
}
bool Base::set_ready(int x) {
	if (!x) {
		if ((parent and parent->get_ready() == 0) or get_ready() == 0)
			return true;
		else {
			for (Base* child : children)
				child->set_ready(0);
			this->x = 0;
			return true;
		}
	}
	else {
		if (parent and parent->get_ready() == 0)
			return false;
		else {
```

```
                  this->x = x;
                  return true;
            }
      }
}
int Base::get_ready() {
      return this->x;
}
vector<Base*> Base::get_children() {
      return this->children;
}
void Base::print_tree(string space) {
      cout << endl << space << get_name();
      for (auto child : children)
            child->print_tree(space + " ");
}
void Base::print_ready_tree(string space) {
      cout << endl << space << get_name();
      if (!x)
            cout << " is not ready";
      else
            cout << " is ready";
      for (Base* child : children)
            child->print_ready_tree(space + "    ");
}
Base* Base::find_object(string name) {
      if (this->name == name)
            return this;
      else{
            for (Base* child : children){
                  Base* res = child->find_object(name);
                  if(res)
                        return res;
            }
            return nullptr;
      }
}
Base* Base::find_coordinate(Base* this_obj, Base* root_obj, string coordinate) {
      if (coordinate == "")
            return nullptr;
      if (coordinate == ".")
            return this_obj;
      if (coordinate == "/")
            return root_obj;
      if (coordinate[0] == '/' and coordinate[1] == '/') {
            string name = "";
            for (int i = 2; i < coordinate.length(); ++i)
                  name += coordinate[i];
            return root_obj->find_object(name);
      }
      Base* curr_obj;
      string name;
      int i = 0;
      int i = 0;
      if (coordinate[0] == '/') {
            curr_obj = root_obj;
            i = 1;
```

```
	}
	else {
		curr_obj = this_obj;
		i = 0;
	}
	for (; i < coordinate.length(); ++i) {
		if (coordinate[i] == '/' or i == coordinate.length() - 1) {
			if (i == coordinate.length() - 1)
				name += coordinate[i];
			if (name == ".")
				continue;
			bool found = false;
			for (Base* pointer : curr_obj->get_children())
				if (pointer->get_name() == name) {
					curr_obj = pointer; name = "";
					found = true;
				}
			if (!found)
				return nullptr;
		}
		else
			name += coordinate[i];
	}
	return curr_obj;
}
```

Файл Base.h

Листинг 2 – Base.h

```
#ifndef _BASE_H
#define _BASE_H
#include <iostream>
#include <vector>
#include <string>
using namespace std;
class Base {
private:
	string name;
	Base* parent;
	vector<Base*> children;
	int x = 0;
public:
	Base(string name = "",
	Base* parent = nullptr);
	string get_name();
	void set_name(string& name);
	Base* get_root();
	void set_root(Base* new_parent);
	void print_tree(string);
	Base* find_object(string name);
	void print_ready_tree(string);
```

```
	bool set_ready(int x);
	int get_ready();
	vector<Base*> get_children();
	Base* find_coordinate(Base*, Base*, string);
};
#endif
```

Файл cl_application.cpp

Листинг 3 – cl_ application.cpp

```
#include "cl_application.h"
pair<bool, string> cl_application::build_tree_objects() {
	string parent_name, name;
	Base* parent, * child;
	cin >> parent_name;
	set_name(parent_name);
	int num;
	while (true) {
		cin >> parent_name;
		if (parent_name == "endtree")
			return make_pair(true, "");
		cin >> name >> num;
		parent = find_coordinate(this, this, parent_name);
		if (parent == nullptr)
			return make_pair(false, parent_name);
		switch (num) {
		case(2):
			child = new class1(name, parent);
			break;
		case(3):
			child = new class2(name, parent);
			break;
		case(4):
			child = new class3(name, parent);
			break;
		case(5):
			child = new class4(name, parent);
			break;
		case(6):
			child = new class5(name, parent);
			break;
		case(7):
			child = new class6(name, parent);
			break;
		}
	}
}
int cl_application::exec_app(bool err_head, string head_name) {
	if (err_head) {
		cout << "Object tree";
		print_tree("");
		string cmd, coordinate;
```

```
            Base* this_obj = this;
            while (true) {
                  cin >> cmd;
                  if (cmd == "END")
                        break;
                  cin >> coordinate;
                  if (cmd == "SET") {
                        Base* res = find_coordinate(this_obj, this, coordinate);
                        if (res == nullptr)
                              cout << "\n" << "Object is not found: " << this_obj-
>get_name() << " " << coordinate;
                        else {
                              cout << "\n" << "Object is set: " << res->get_name();
                              this_obj = res;
                        }
                  }
                  else if (cmd == "FIND") {
                        Base* res = find_coordinate(this_obj, this, coordinate);
                        cout << "\n" << coordinate << "     ";
                        if (res == nullptr)
                              cout << "Object is not found";
                        else
                              cout << "Object name: " << res->get_name();
                  }
            }
      }
      else {
            cout << "Object tree";
            print_tree("");
            cout << endl << "The head object " << head_name << " is not found";
      }
      return 0;
}
```

Файл cl_application.h

Листинг 4 – cl_application.h

```
#ifndef _CL_APPLICATION_H
#define _CL_APPLICATION_H
#include "Base.h"
#include "class1.h"
#include "class2.h"
#include "class3.h"
#include "class4.h"
#include "class5.h"
#include "class6.h"
class cl_application : public Base {
public:
      cl_application(string name = "") : Base(name) {};
      pair<bool, string> build_tree_objects();
      int exec_app(bool, string);
};
#endif
```

Файл class1.h

Листинг 5 – class1.h

```
#ifndef _CLASS1_H
#define _CLASS1_H
#include "Base.h"
class class1 : public Base {
public:
	class1(string name, Base* parent) : Base(name, parent) {};
};
#endif
```

Файл class2.h

Листинг 6 – class2.h

```
#ifndef _CLASS2_H
#define _CLASS2_H
#include "Base.h"
class class2 : public Base {
public:
	class2(string name, Base* parent) : Base(name, parent) {};
};
#endif
```

Файл class3.h

Листинг 7 – class3.h

```
#ifndef _CLASS3_H
#define _CLASS3_H
#include "Base.h"
class class3 : public Base {
public:
	class3(string name, Base* parent) : Base(name, parent) {};
};
#endif
```

Файл class4.h

Листинг 8 – class4.h

```
#ifndef _CLASS4_H
#define _CLASS4_H
#include "Base.h"
class class4 : public Base {
public:
	class4(string name, Base* parent) : Base(name, parent) {};
};
#endif
```

Файл class5.h

Листинг 9 – class5.h

```
#ifndef _CLASS5_H
#define _CLASS5_H
#include "Base.h"
class class5 : public Base {
public:
	class5(string name, Base* parent) : Base(name, parent) {};
};
#endif
```

Файл class6.h

Листинг 10 – class6.h

```
#ifndef _CLASS6_H
#define _CLASS6_H
#include "Base.h"
class class6 : public Base {
public:
	class6(string name, Base* parent) : Base(name, parent) {};
};
#endif
```

Файл main.cpp

Листинг 11 – main.cpp

```
#include "cl_application.h"
int main() {
	cl_application ob_cl_application;
	pair<bool, string> application = ob_cl_application.build_tree_objects();
	return ob_cl_application.exec_app(application.first, application.second);
}
```

Приложение Б

Результат тестирования программы представлен в таблице Б.

Таблица Б – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
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	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	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