Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

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

Расширить функциональность базового класса:

- метод переопределения головного объекта для текущего в дереве иерархии. Метод должен иметь один параметр, указатель на объект базового класса, содержащий указатель на новый головной объект. Переопределение головного объект для корневого объекта недопустимо. Недопустимо создать второй корневой объект. Недопустимо при переопределении, чтобы у нового головного появились два подчиненных объекта с одинаковым наименованием. Новый головной объект не должен принадлежать к объектам из ветки текущего. Если переопределение выполнено, метод возвращает значение «истина», иначе «ложь»;
- метод удаления подчиненного объекта по наименованию. Если объект не найден, то метод завершает работу. Один параметр строкового типа, содержит наименование удаляемого подчиненного объекта;
- метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задаться в следующем виде:
    - / - корневой объект;
    - //«имя объекта» - поиск объекта по уникальной имени от корневого (для однозначности уникальность требуется в рамках дерева);
    - . - текущий объект;
    - .«имя объекта» - поиск объекта по уникальной имени от текущего (для однозначности уникальность требуется в рамках ветви дерева от

текущего объекта);

- «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_2/ob_3
/ob_1/ob_2/ob_3
```

Если координата - пустая строка или объект не найден или определяется неоднозначно (дубляж имен на ветке, на дереве), тогда вернуть нулевой указатель.

Наименование объекта не содержит символы «.» и «/».

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 необходимо соблюдать. Если это требование исходя из входных данных нарушается, то соответствующий подчиненный объект не создается.

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). Если номер класса объекта задан некорректно, то объект не создается.

Собранная система отрабатывает следующие команды:

- SET «координата» – устанавливает текущий объект;
- FIND «координата» – находит объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» задает новый головной объект;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим. При вводе данных в названии команд ошибок нет. Если при переопределении головного объекта нарушается уникальность наименований подчиненных объектов для нового головного, переопределение не производится.

1.1 Описание входных данных

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.

После ввода состава дерева иерархии построчно вводятся команды:

- SET «координата» – установить текущий объект;
- FIND «координата» – найти объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» соответствует новому головному объекту;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершить функционирование системы (выполнение программы).

Команды SET, FIND, MOVE и DELETE вводятся произвольное число раз.

Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов:

```
rootela
/ 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_7
FIND object_4/object_7
FIND .
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева как в 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 с кодом возврата 1.

Если при построении при попытке создания объекта обнаружен дубляж, то вывести:

«координата головного объекта» Dubbing the names of subordinate objects

Если дерево построено, то далее построчно вводятся команды.

Для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

The object was not found at the specified coordinate: «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

Для команд MOVE вывести:

New head object: «наименование нового головного объекта»

Если головной объект не найден, то:

«искомая координата объекта» Head object is not found

Если переопределить головной объект не удалось, то:

«искомая координата объекта» Redefining the head object failed

Если у нового головного объекта уже есть подчиненный с таким же именем, то вывести:

«искомая координата объекта» Dubbing the names of subordinate objects

При попытке переподчинения головного объекта к объекту на ветке, вывести:

«координата нового головного объекта» Redefining the head object failed

Для команды DELETE:

Если подчиненный объект удален, то вывести:

The object «абсолютный путь удаленного объекта» has been deleted

Если объект не найден, то ничего не выводить.

После команды END с новой строки вывести:

```
Current object hierarchy tree
```

Со следующей строки вывести текущую иерархию дерева.

Пример вывода иерархии дерева объектов:

```
Object tree
rootela
    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_7     Object is not found
object_4/object_7     Object name: object_7
.     Object name: object_2
.object_7     Object name: object_7
object_4/object_7     Object name: object_7
.object_7     Redefining the head object failed
Object is set: object_7
//object_1     Dubbing the names of subordinate objects
New head object: object_3
Current object hierarchy tree
rootela
    object_1
        object_7
    object_2
        object_4
        object_5
        object_3
    object_3
        object_7
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- оператор освобождения динамической памяти delete.

Класс cl_base:

- функционал:
    - метод set_new_parent — метод смены родителя для текущего члена иерархии объектов;
    - метод remove_child — метод удаления дочернего объекта по имени;
    - метод get_obj_by_way — метод получения указателя на объект дерева иерархии по пути или координате;
    - метод ~cl_base — деструктор объекта класса, очищает динамическую память.

Класс cl_application:

- функционал:
    - метод build_tree_objects — метод построения дерева иерархии;
    - метод exec_app — метод, запускающий приложение.

Класс cl_2:

- функционал:
    - метод cl_2 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Класс cl_3:

- функционал:
    - метод cl_3 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Класс cl_4:

- функционал:
    - метод cl_4 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Класс cl_5:

- функционал:
    - метод cl_5 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Класс cl_6:

- функционал:
    - метод cl_6 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс	
		cl_application	public		2
		cl_2	public		3
		cl_3	public		4
		cl_4	public		5
		cl_5	public		6
		cl_6	public		7
2	cl_application			Класс-приложение	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
3	cl_2			Дочерний класс	
4	cl_3			Дочерний класс	
5	cl_4			Дочерний класс	
6	cl_5			Дочерний класс	
7	cl_6			Дочерний класс	

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода set_new_parent класса cl_base

Функционал: метод смены родителя для текущего члена иерархии объектов.

Параметры: объект класса cl_base - new_parent.

Возвращаемое значение: булевое значение - если переопределение выполнено, метод возвращает значение «истина», иначе «ложь».

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_new_parent класса cl_base

№	Предикат	Действия	№ перехода
1	Указатель на текущий объект совпадает с указателем на корневой	return false	∅
			2
2	В качестве параметра передан нулевой указатель	return false	∅
			3
3	У нового головного объекта есть подчиненный с именем текущего	return false	∅
			4
4		Инициализация указателя на объект p_obj указателем на объект new_parent класса cl_base	5
5	p_obj - не нулевой указатель		6

№	Предикат	Действия	№ перехода
			8
6	Указатель на p_obj совпадает с указателеми на текущий	return false	∅
			7
7		p_obj становится указателем на свой головной объект	5
8		Удаление вектора подчиненных объектов для текущего	9
9		p_head_object = new_parent	10
10		Добавление текущего объекта в вектор подчиненных новому головному	11
11		return true	∅

3.2 Алгоритм метода remove_child класса cl_base

Функционал: метод удаления дочернего объекта по имени.

Параметры: string sub_name.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода remove_child класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация счетчика i = 0	2
2	i < размер вектора подчиненных объектов текущего		3
			∅
3	Имя i-того объекта совпадает с sub_name	Вызов деструктора	5

№	Предикат	Действия	№ перехода
			4
4		i += 1	2
5		Удаление i-того элемента вектора	∅

3.3 Алгоритм метода get_obj_by_way класса cl_base

Функционал: метод получения указателя на объект дерева иерархии по пути или координате.

Параметры: string way.

Возвращаемое значение: указатель на объект класса cl_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_obj_by_way класса cl_base

№	Предикат	Действия	№ перехода
1	Строка way пуста	return nullptr	∅
			2
2	way == "."	Возврат указателя на текущий объект	∅
			3
3	way[0] == "."	Возвращение результата работы find_obj_current, вызванного от текущего объекта с параметром way со 2 элемента	∅
			4
4	Первые 2 символа way == "//"	Возвращение результата работы find_obj_root, вызванного от текущего объекта с параметром way с 3 элемента	∅
			5
5	Первый символ != "/"	Инициализация беззнаковой переменной ind позицией первого вхождения символа /	6
			8

№	Предикат	Действия	№ перехода
6		инициализация указателя p_sub объекта класса cl_base результатом работы метода get_sub_object с параметром way до первого символа /	7
7	p_sub != nullptr или в строке нет сивола /	возврат p_sub	∅
		возврат результата работы get_obj_by_way от p_sub с параметром way от первого символа /	8
8		инициализация root - указателя на объект класса cl_base указателем на текущий объект	9
9	указатель не равен указателю на корневой объект	root становится указателем на его головной объект	9
			10
10	path == "/"	return root	∅
		возврат результата работы метода get_obj_by_way, вызванного у объекта, на который указывает root, с параметром way от 2 элемента	∅

3.4 Алгоритм деструктора класса cl_base

Функционал: деструктор объекта класса, очищает динамическую память.

Параметры: нет.

Алгоритм деструктора представлен в таблице 5.

Таблица 5 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация счетчика i = 0	2
2	i < размер вектор	очистка памяти по указателю p_sub_objects[i]	3

№	Предикат	Действия	№ перехода
	p_sub_objects		
			∅
3		i += 1	2

3.5 Алгоритм метода build_tree_objects класса cl_application

Функционал: метод построения дерева иерархии.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		вывод "object tree"	2
2		объявление строк way, curname	3
3		объявление int i_class	4
4		ввод curname	5
5		вызов set_name с параметром curname от текущего объекта	6
6		объявление указателя на объект класса cl_base - p_head	7
7		ввод значения переменной way	8
8	way != "endtree"	ввод значения переменных curname, i_class	9
			∅
9		присваивание p_head результата работы get_obj_by_way с параметром way	10
10	p_head - не нулевой указатель	вызов print_names_recur	11
			13

№	Предикат	Действия	№ перехода
11		вывод "The head object ", строки way и " is not found"	12
12		завершение работы программы	∅
13	результат работы get_sub_object от p_head не нулевой указатель	вывод "Dupping the names of subordinate objects" с новой строки	19
			14
14	i_class == 2?	Создание объекта класса cl_2 с помощью конструктора с параметрами p_head и curname	19
			15
15	i_class == 3?	Создание объекта класса cl_3 с помощью конструктора с параметрами p_head и curname	19
			16
16	i_class == 4?	Создание объекта класса cl_4 с помощью конструктора с параметрами p_head и curname	19
			17
17	i_class == 5?	Создание объекта класса cl_5 с помощью конструктора с параметрами p_head и curname	19
			18
18	i_class == 6?	Создание объекта класса cl_6 с помощью конструктора с параметрами p_head и curname	19
			19
19		ввод значения переменной way	8

3.6 Алгоритм метода exec_app класса cl_application

Функционал: метод, запускающий приложение.

Параметры: нет.

Возвращаемое значение: код завершения работы программы.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		объявление строк com и input	2
2		инициализация указателя cur на объект класса cl_base адресом текущего объекта	3
3		объявление указателя help_obj на объект класса cl_base	4
4		объявление стека типа string	5
5		вызов у текущего объекта print_names_recur	6
6		ввод значения переменной com	7
7	com != "END"?	ввод значения переменной input	8
			27
8	com == "SET"?	присваивание help_obj результата работы метода get_obj_by_way, вызванного у объекта, на который указывает cur, с параметром input	9
			11
9	help_obj != nullptr?	cur = help_obj	10
		перенос вывода на следующую строку, вывод "The object was not found at specified coordinate:" и строки input	11
10		перенос вывода на следующую строку, вывод "Object is set:" и имени объекта по указателю help_obj	11
11	com == "FIND" ?	присваивание help_obj результата работы метода get_obj_by_way, вызванного у объекта, на который указывает cur, с параметром input	12
			13
12	help_obj != nullptr?	перенос вывода на следующую строку, вывод "Object name:" и имени объекта по указателю	13

№	Предикат	Действия	№ перехода
		help_obj	
		перенос вывода на следующую строку, вывод input и строки "Object is not found"	13
13	com == "MOVE"?	присваивание help_obj результата работы метода get_obj_by_way, вызванного у объекта, на который указывает cur, с параметром input	14
			17
14	работа метода set_new_parent от объекта по указателю cur завершилась успехом	перенос вывода на следующую строку, вывод "New head object:" и имени объекта по указателю help_obj	17
			15
15	help_obj != nullptr?	перенос вывода на следующую строку, вывод input и строки "Head object is not found"	17
			16
16	Уже есть дочерний элемент с именем объекта по указателю cur	перенос вывода на следующую строку, вывод input и строки "Dubbing the names of subordinate objects"	17
		перенос вывода на следующую строку, вывод input и строки "Dubbing the names of subordinate objects"	17
17	com == "DELETE"?	присваивание help_obj результата работы метода get_obj_by_way, вызванного у объекта, на который указывает cur, с параметром input	18
			26
18	help_obj != nullptr?		19
			26
19	объект по указателю help_obj	добавление в стек s имени объекта по указателю	20

№	Предикат	Действия	№ перехода
	не нулевой	help_obj	
			21
20		присвоение help_obj результата работы get_head, вызванного по указателю help_obj	19
21		вызов remove_child по указателю cur с параметром input	22
22		перенос вывода на новую строку и вывод "The object "	23
23	стек не пустой	вывод на экран "/" и верхнего элемента стека	24
			25
24		удаление верхнего элемента стека	25
25		вывод " has been deleted"	26
26		ввод значения переменной com	7
27		перенос на новую строку и вывод "Current object hierarchy tree"	28
28		вызов print_names_recur у текущего объекта	29
29		return 0	∅

3.7 Алгоритм конструктора класса cl_2

Функционал: Параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве аргументов в него передаются p_head_object и s_name	∅

3.8 Алгоритм конструктора класса cl_3

Функционал: Параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве аргументов в него передаются p_head_object и s_name	∅

3.9 Алгоритм конструктора класса cl_4

Функционал: Параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве аргументов в	∅

№	Предикат	Действия	№ перехода
		него передаются p_head_object и s_name	

3.10 Алгоритм конструктора класса cl_5

Функционал: Параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве аргументов в него передаются p_head_object и s_name	∅

3.11 Алгоритм конструктора класса cl_6

Функционал: Параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве аргументов в него передаются p_head_object и s_name	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-16.

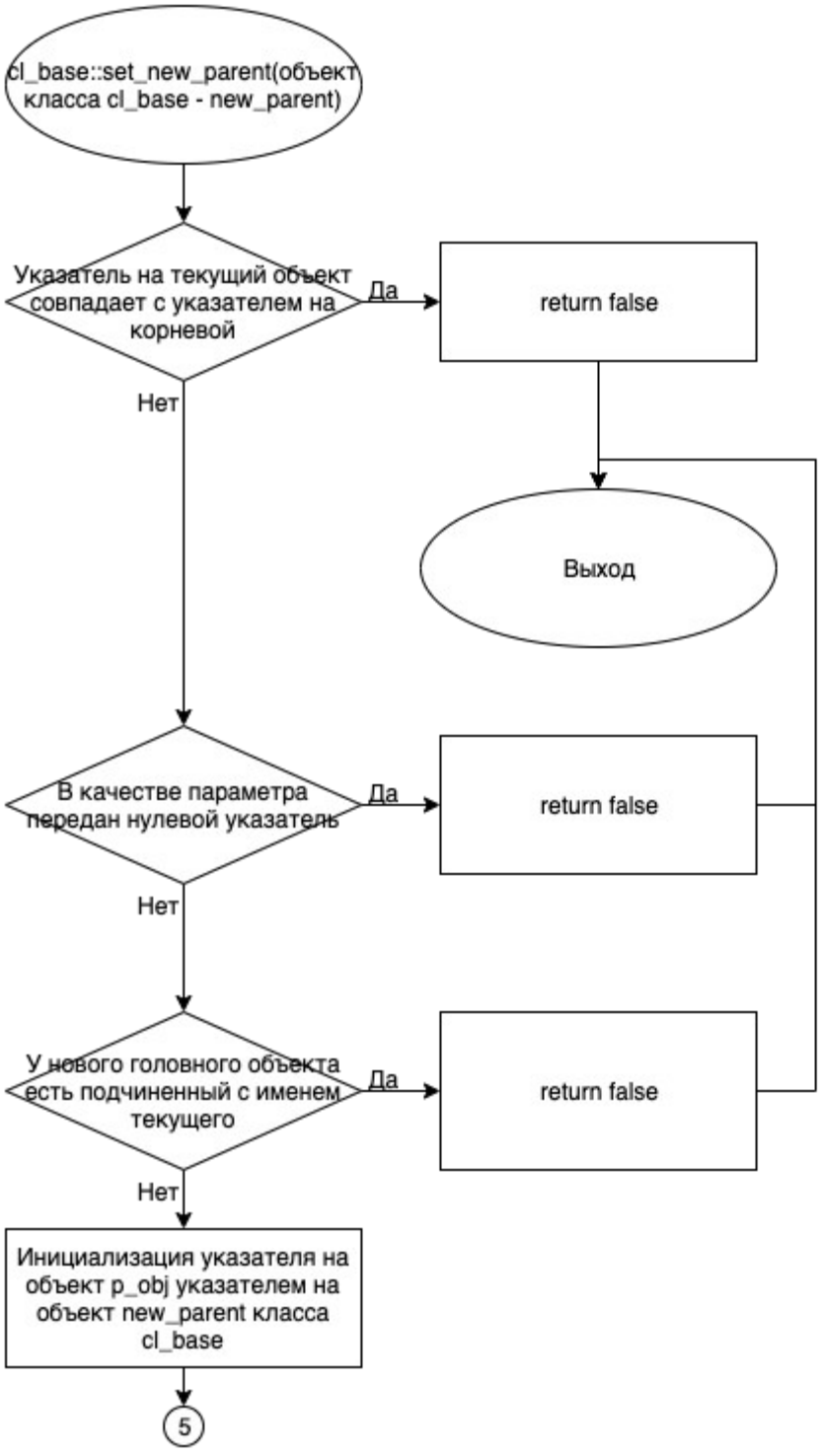

Рисунок 1 – Блок-схема алгоритма

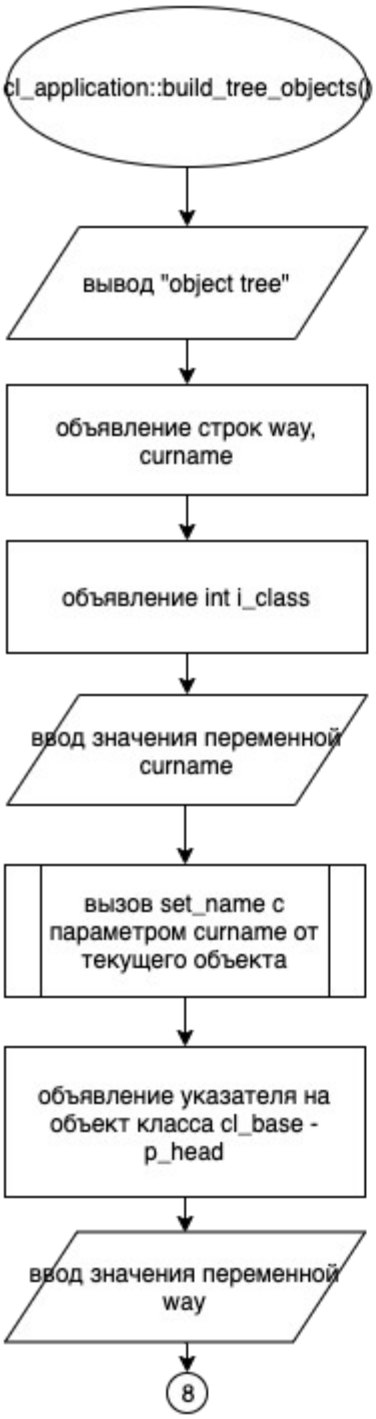

Рисунок 2 – Блок-схема алгоритма

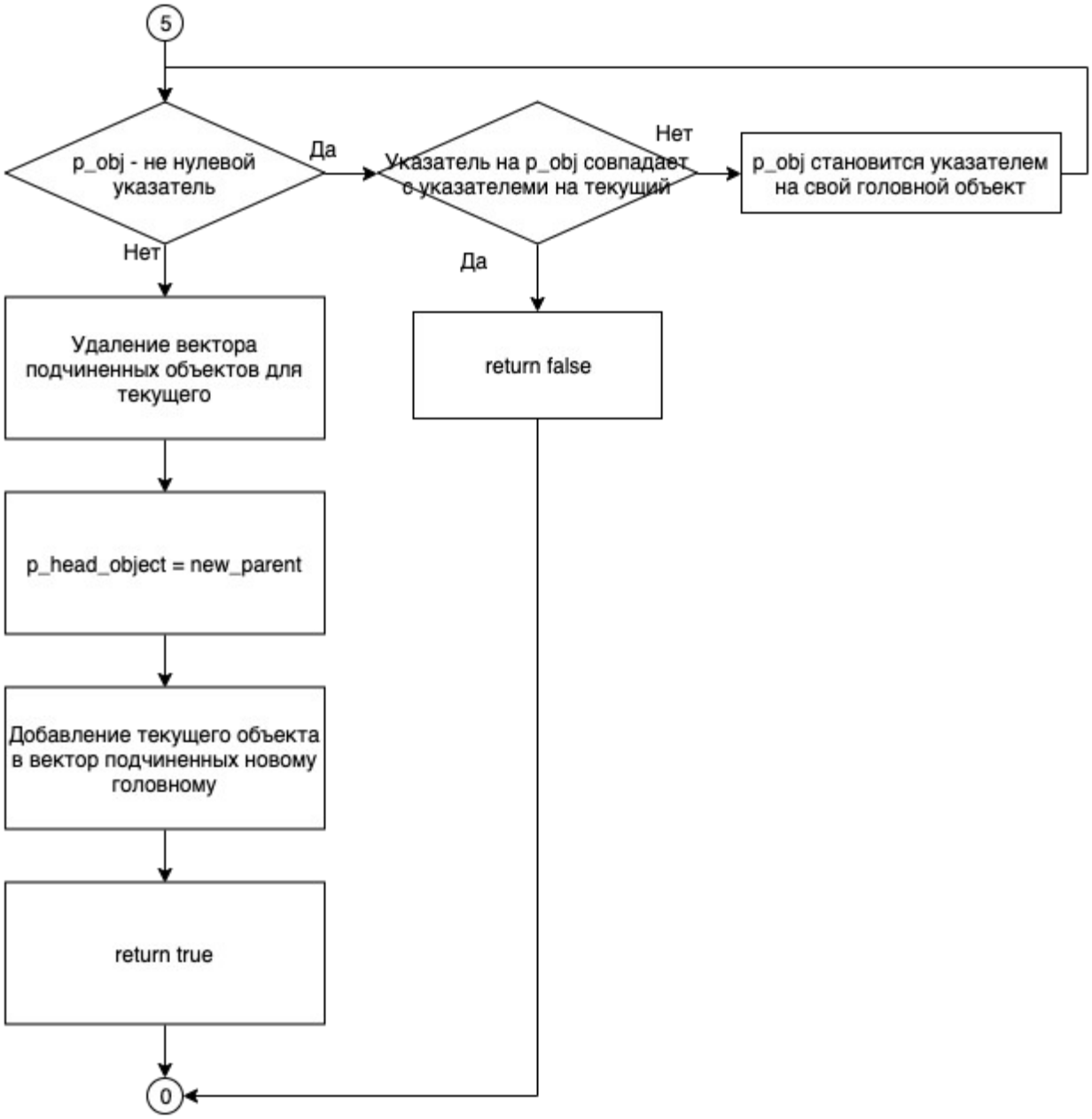

Рисунок 3 – Блок-схема алгоритма

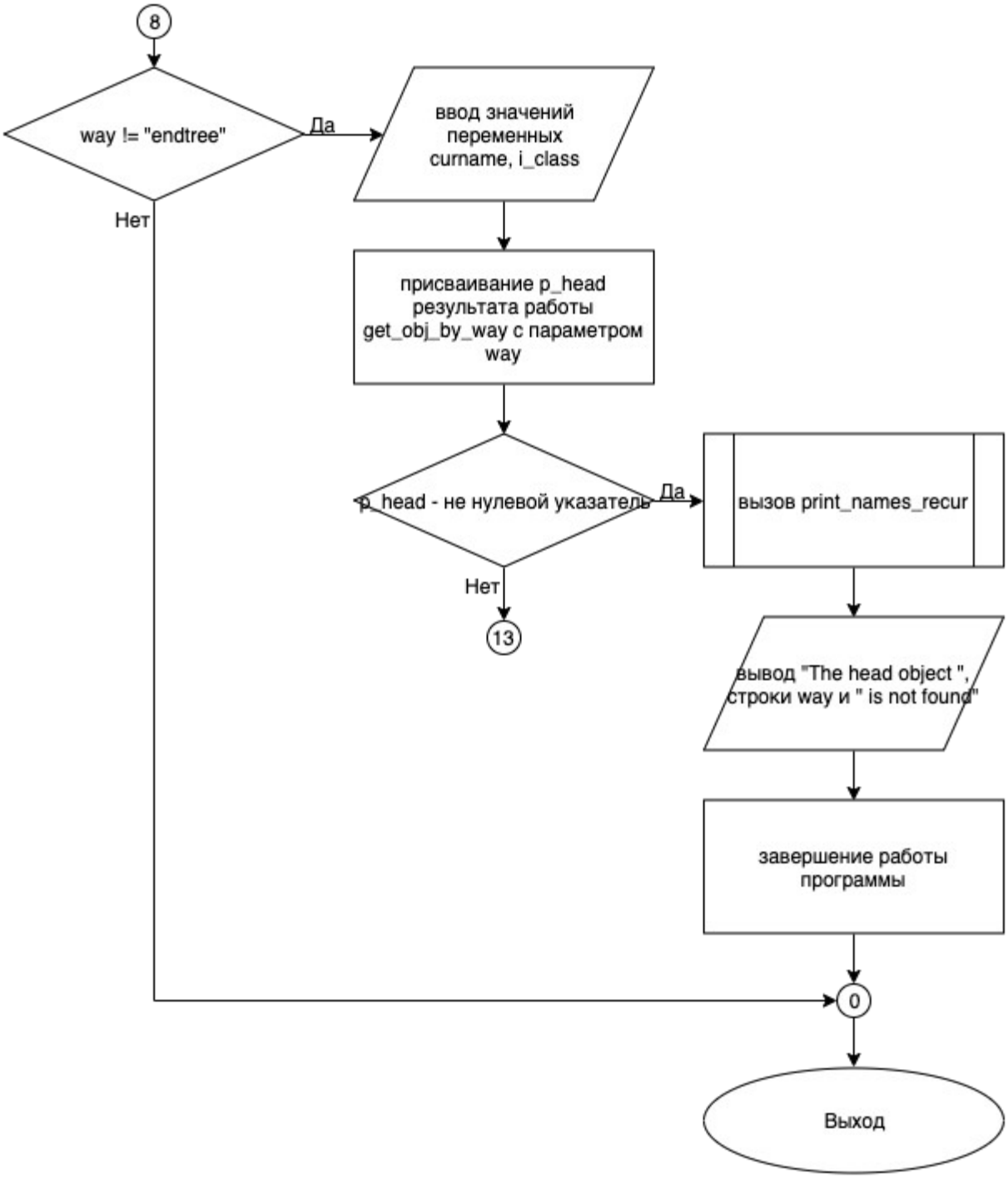

Рисунок 4 – Блок-схема алгоритма

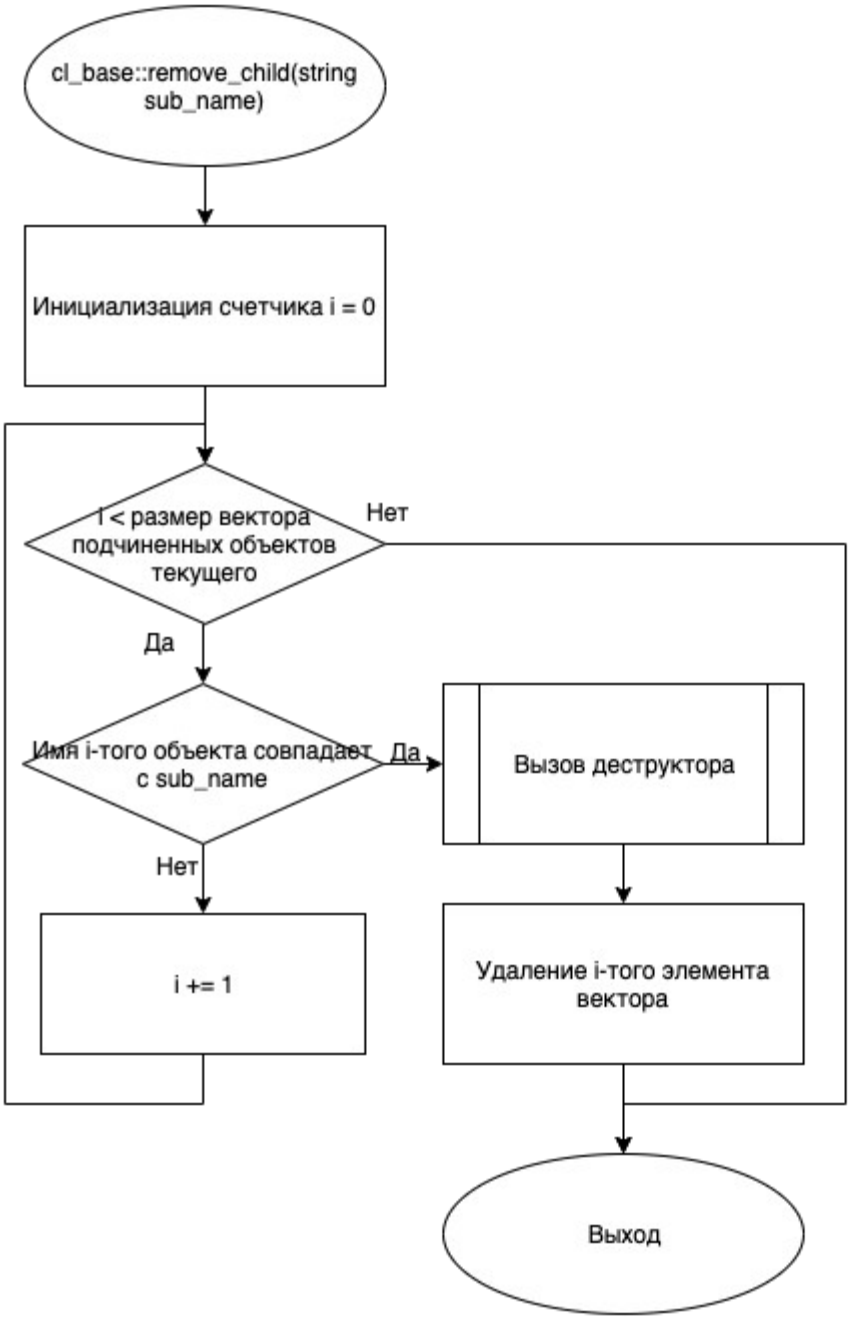

Рисунок 5 – Блок-схема алгоритма

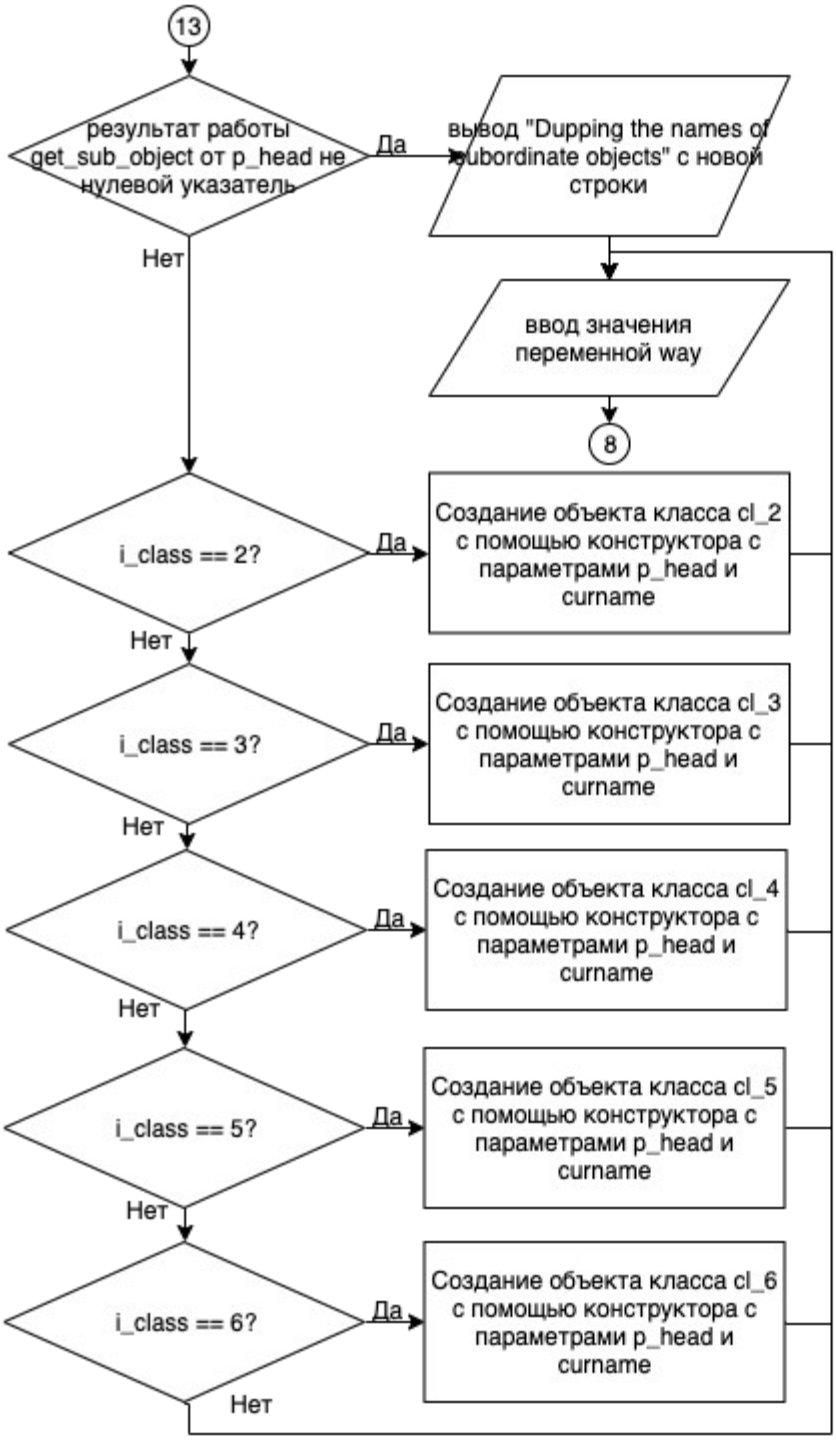

Рисунок 6 – Блок-схема алгоритма

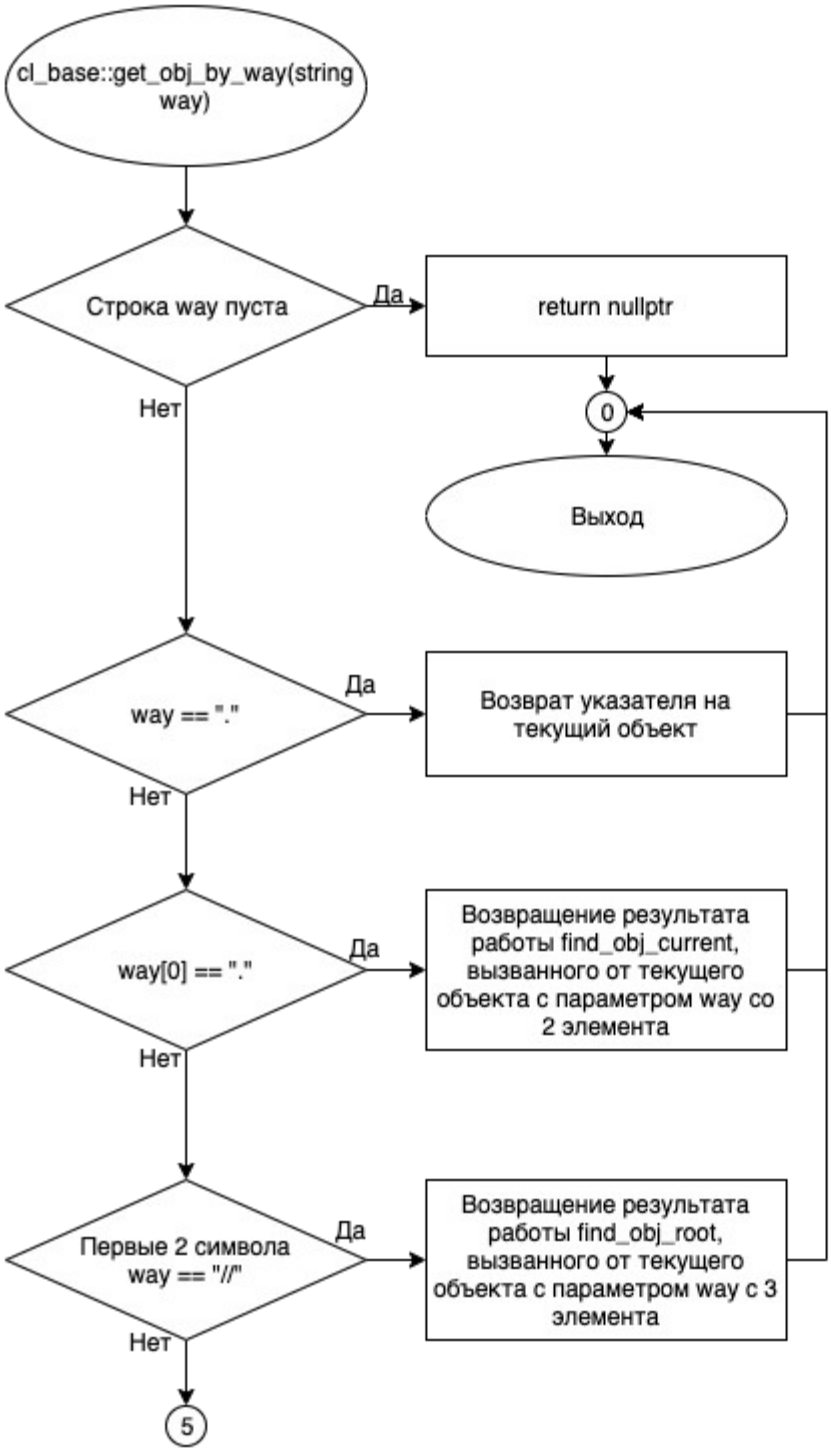

Рисунок 7 – Блок-схема алгоритма

Рисунок 8 – Блок-схема алгоритма

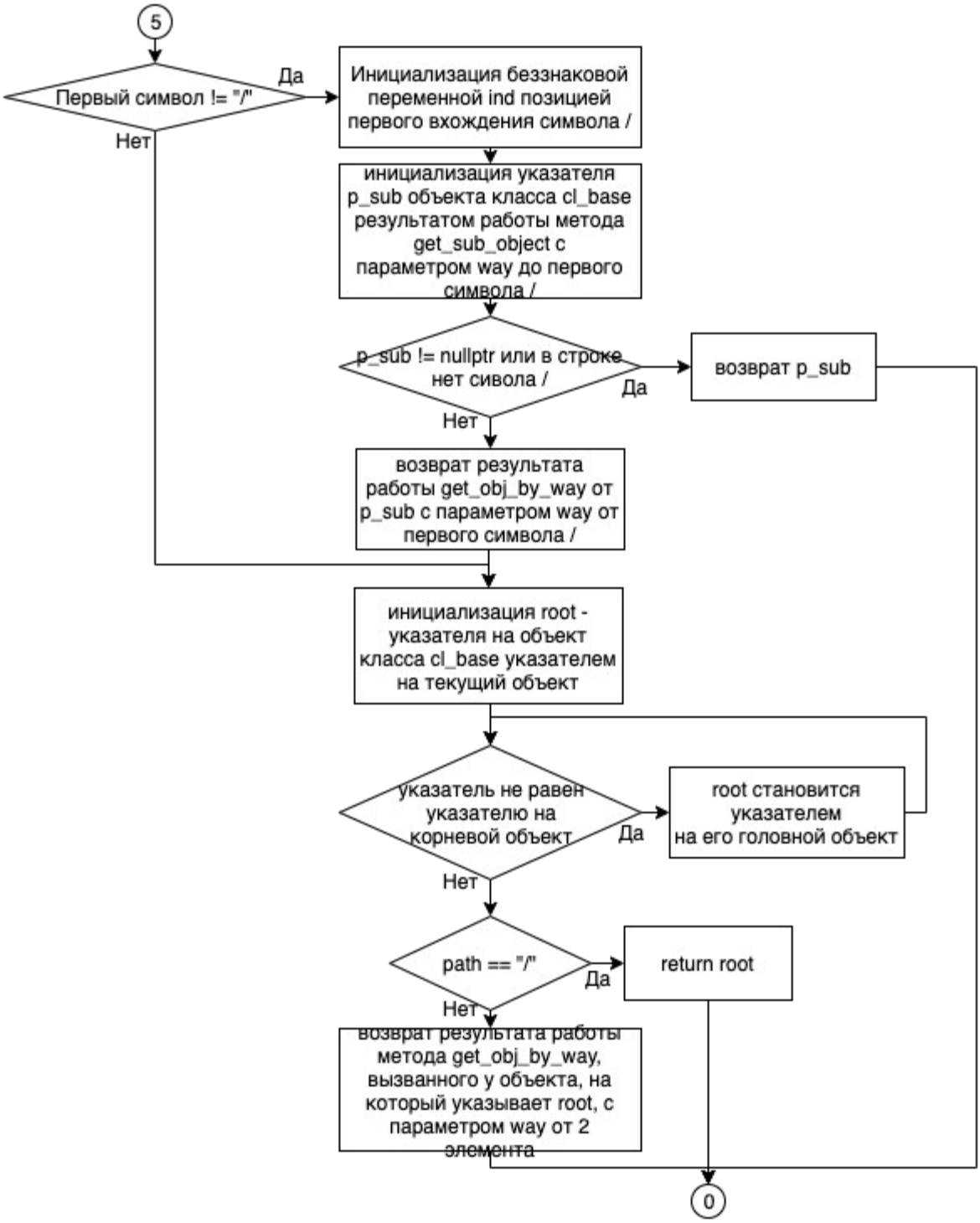

Рисунок 9 – Блок-схема алгоритма

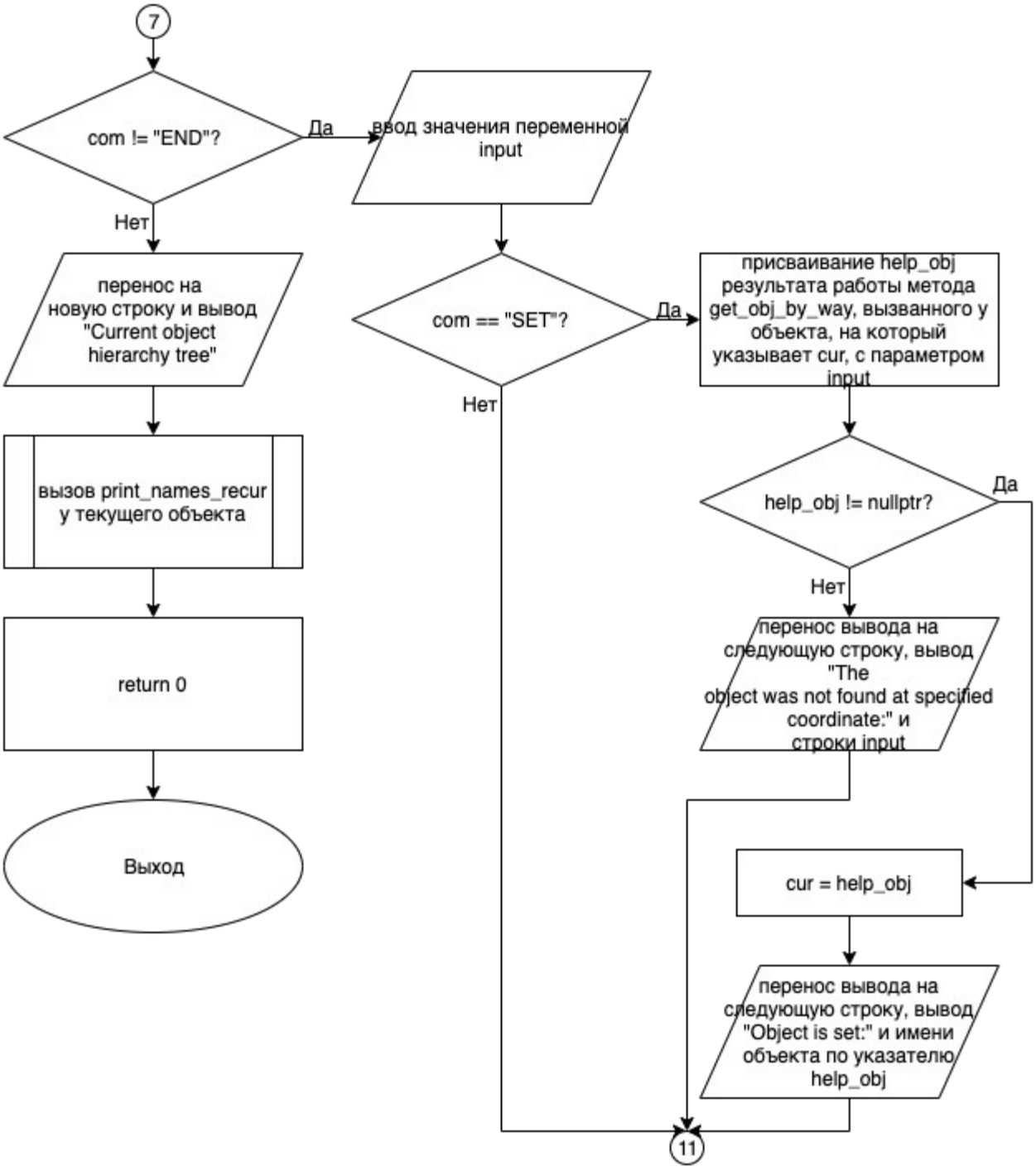

Рисунок 10 – Блок-схема алгоритма

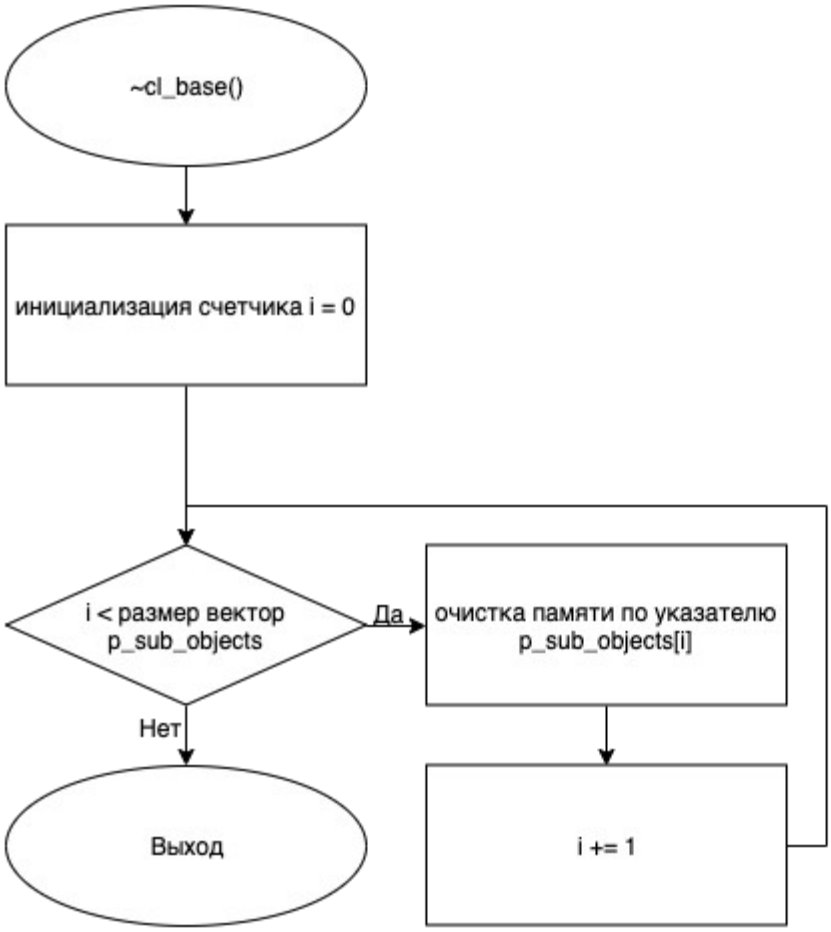

Рисунок 11 – Блок-схема алгоритма

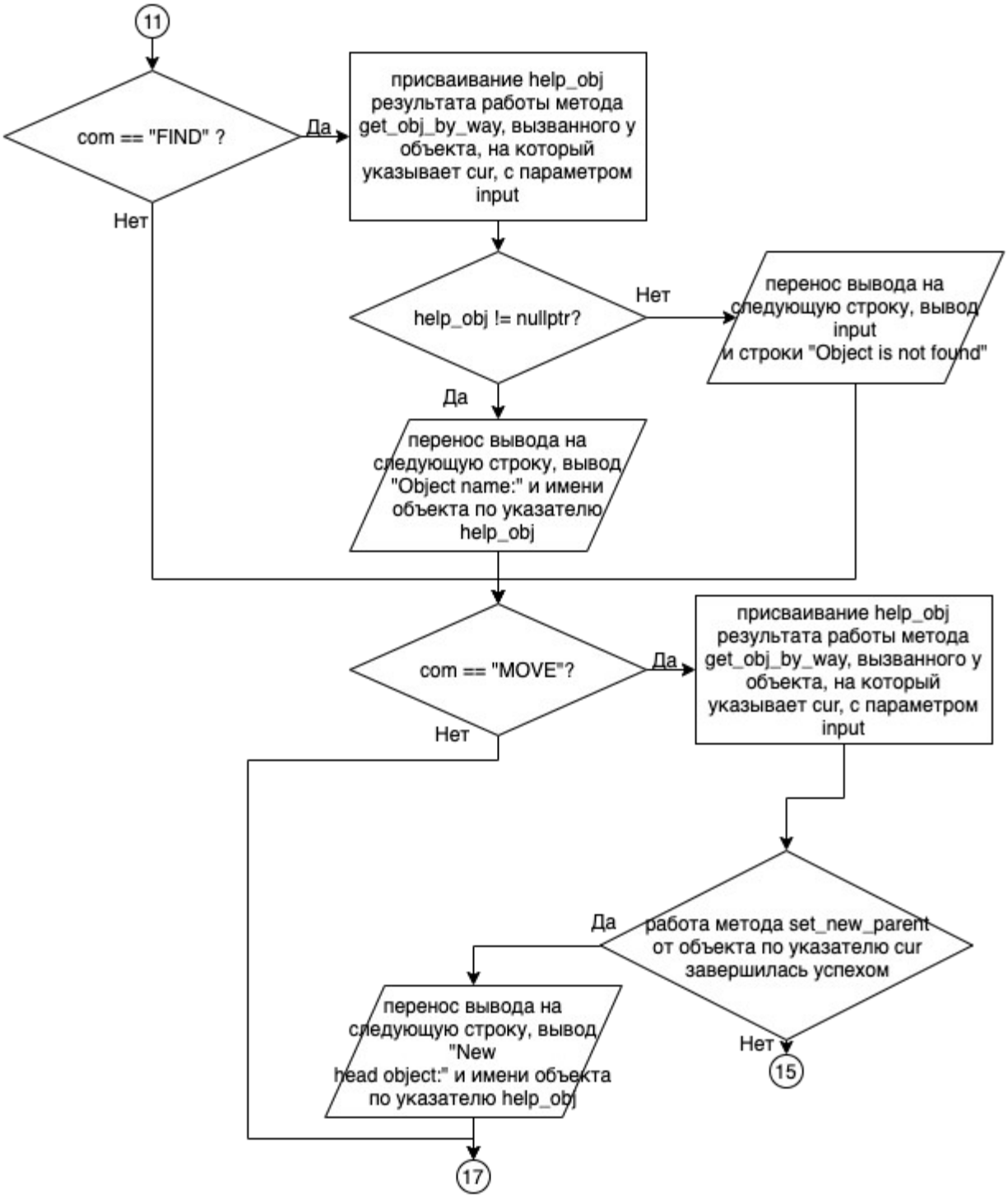

Рисунок 12 – Блок-схема алгоритма

Рисунок 13 – Блок-схема алгоритма

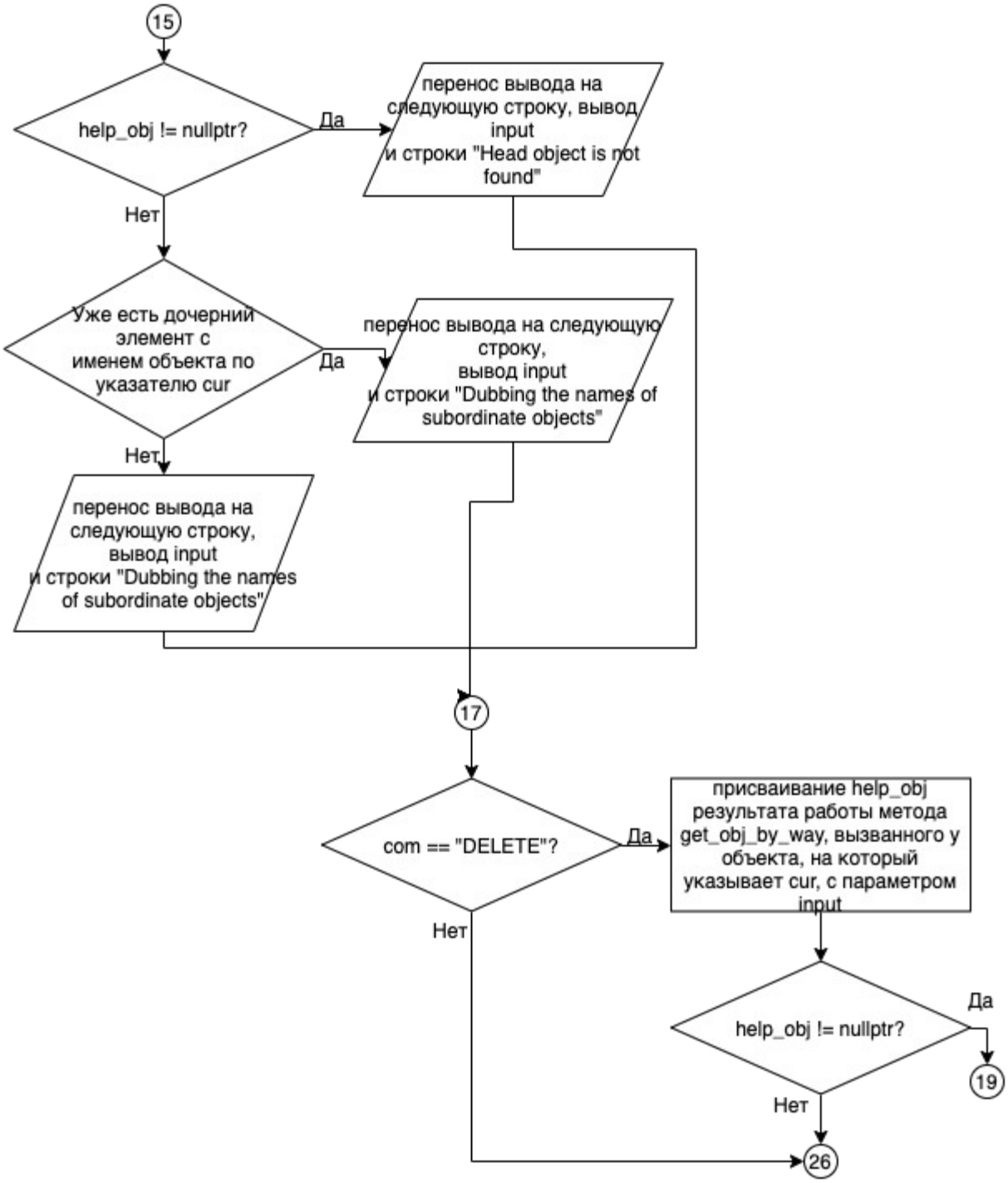

Рисунок 14 – Блок-схема алгоритма

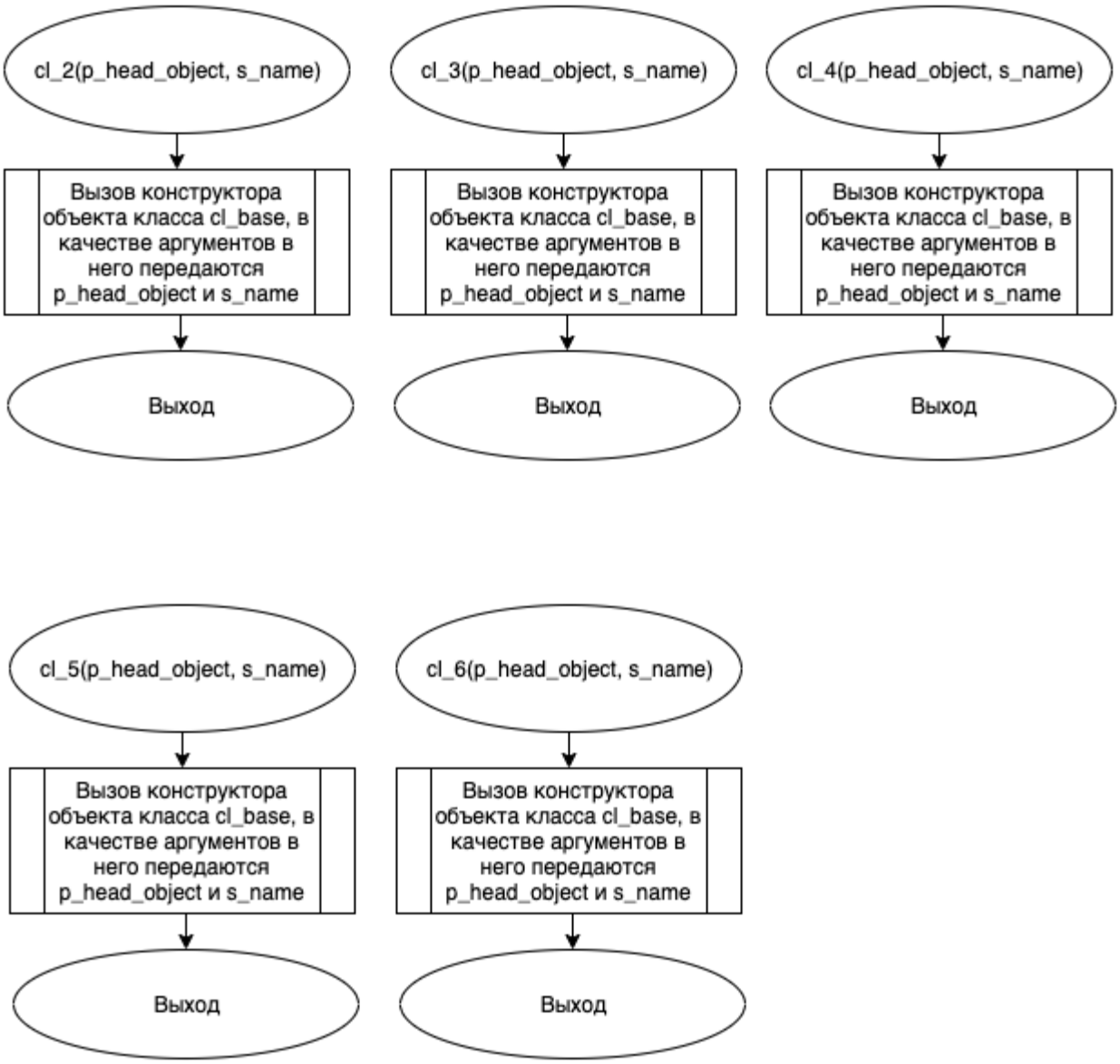

Рисунок 15 – Блок-схема алгоритма

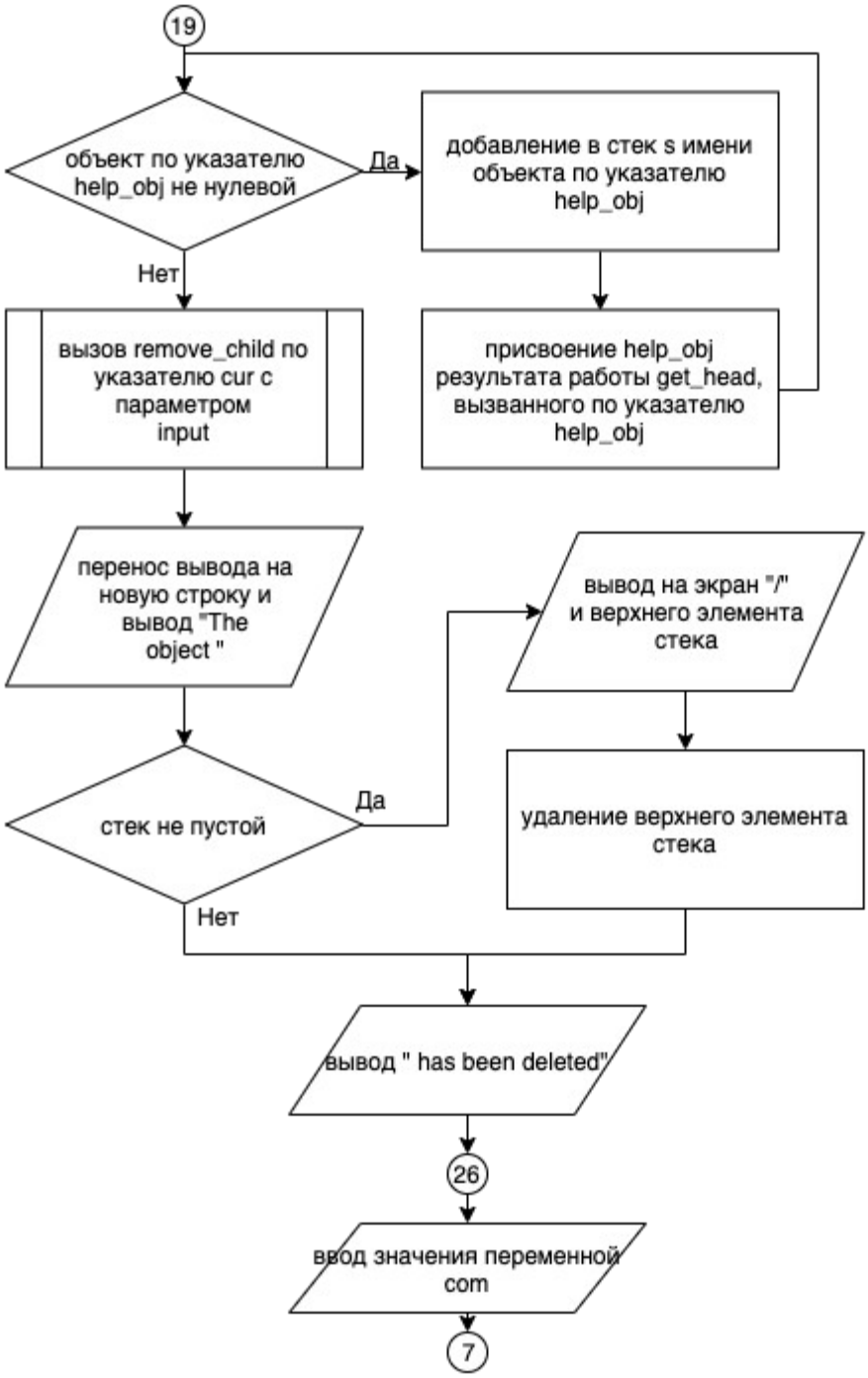

Рисунок 16 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"
using namespace std;

cl_2::cl_2(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_2
}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"

class cl_2 : public cl_base{
public:
   cl_2(cl_base* p_head_object, string s_name);
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
using namespace std;
```

```
cl_3::cl_3(cl_base*  p_head_object,  string  s_name)  :  cl_base(p_head_object,
s_name){
   // конструктор объекта cl_3
}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"

class cl_3 : public cl_base{
public:
   cl_3(cl_base* p_head_object, string s_name);
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"
using namespace std;

cl_4::cl_4(cl_base*  p_head_object,  string  s_name)  :  cl_base(p_head_object,
s_name){
   // конструктор объекта cl_4
}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
```

```
#include "cl_base.h"

class cl_4 : public cl_base{
public:
   cl_4(cl_base* p_head_object, string s_name);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"
using namespace std;

cl_5::cl_5(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_5
}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include "cl_base.h"

class cl_5 : public cl_base{
public:
   cl_5(cl_base* p_head_object, string s_name);
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base*  p_head_object,  string  s_name)  :  cl_base(p_head_object,
s_name){
   // конструктор объекта cl_6
}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "cl_base.h"

class cl_6 : public cl_base{
public:
   cl_6(cl_base* p_head_object, string s_name);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
#include "cl_base.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <iostream>
using namespace std;

cl_application::cl_application(cl_base*           p_head_object)           :
cl_base(p_head_object){
```

```
};

void cl_application::build_tree_objects()
{
   string way, curname;
   int i_class;
   cout << "Object tree";
   cin>>curname;
   set_name(curname);
   cl_base* p_head;
   cl_base* p_sub;
   cin >> way;
   while (way != "endtree")
   {
      cin >> curname >> i_class;
      p_head = get_obj_by_way(way);
      if (!p_head)
      {
         print_names_recur();
         cout << "\n" << "The head object " << way <<" is not found";
         exit(1);
      }
      if (p_head -> get_sub_object(curname))
      {
         cout << "\n" << way << " Dubbing the names of subordinate objects";
      }
      else
      {
         switch (i_class)
         {
            case 2:
               new cl_2(p_head, curname);
               break;
            case 3:
               new cl_3(p_head, curname);
               break;
            case 4:
               new cl_4(p_head, curname);
               break;
            case 5:
               new cl_5(p_head, curname);
               break;
            case 6:
               new cl_6(p_head, curname);
               break;
         }
      }
   cin>>way;
   }
}

// ----------
int cl_application::exec_app() {
   string com, input;
   cl_base* cur = this;
```

```
        cl_base* help_obj;
        stack<string> s;
        print_names_recur();
        cin >> com;

        while (com != "END"){
           cin >> input;
           if (com == "SET")
           {
              help_obj = cur->get_obj_by_way(input);
              if (help_obj)
              {
                 cur = help_obj;
                 cout << endl << "Object is set: " << cur -> get_name();
              }
              else
              {
                 cout  <<  endl  <<  "The  object  was  not  found  as  specified
     coordinate: " << input;
              }
           }
           else if (com == "FIND")
           {
              help_obj = cur->get_obj_by_way(input);
              if (help_obj)
              {
                 cout<<endl<<input<<"          Object  name:  "  <<  help_obj  ->
     get_name();
              }
              else cout << endl << input << "     Object is not found";
           }
           else if (com == "MOVE")
           {
              help_obj = cur -> get_obj_by_way(input);
              if (cur -> set_new_parent(help_obj))
              {
                 cout << endl << "New head object: " << help_obj -> get_name();
              }
              else if (!help_obj)
              {
                 cout << endl << input << " Head object is not found";
              }
              else if (help_obj->get_sub_object(cur->get_name()))
              {
                 cout << endl << input << "      Dubbing the names of subordinate
     objects";
              }
              else
              {
                 cout  <<  endl  <<  input  <<  "        Redefining  the  head  object
     failed";
              }
           }

           else if (com == "DELETE")
```

```
            {
               help_obj = cur -> get_sub_object(input);
               if (help_obj)
               {
                  while (help_obj -> get_head())
                  {
                     s.push(help_obj -> get_name());
                     help_obj = help_obj->get_head();
                  }
                  cur -> remove_child(input);
                  cout << endl << "The object ";
                  while (!s.empty()){
                     cout << '/' << s.top();
                     s.pop();
                  }
                  cout<<" has been deleted";
               }
            }
            cin >> com;
         }
         cout << endl << "Current object hierarchy tree";
         print_names_recur();
         return 0;
      }
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
      #ifndef __CL_APPLICATION__H
      #define __CL_APPLICATION__H

      #include "cl_base.h"
      #include "cl_2.h"

      class cl_application : public cl_base{
      public:
         cl_application(cl_base* p_head_object = nullptr);
         void build_tree_objects();
         int exec_app();
      };

      #endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
#include <iostream>
using namespace std;

// 1

cl_base::cl_base(cl_base* p_head_object, string s_name){
   this->p_head_object = p_head_object;
   if (p_head_object){
      p_head_object -> p_sub_objects.push_back(this);
   }
   this -> state = 0;
   set_name(s_name);
}

string cl_base::get_name(){
   return s_name; // возврат имени объекта
}

cl_base* cl_base::get_head(){
   return p_head_object;
   // возвращение указателя на головной объект
}

cl_base* cl_base::get_sub_object(string s_name){
   for (auto obj : p_sub_objects){
      if (obj->get_name() == s_name) return obj;
   }
   return nullptr;
}

// 2

cl_base* cl_base::find_obj_current(string name){
   queue <cl_base*> q;
   cl_base* p_found = nullptr;
   q.push(this);
   while (!q.empty()){
      if (q.front() -> get_name() == name){
         if (p_found == nullptr)
            p_found = q.front();
         else
            return nullptr;
      }
      for (auto p_sub_object : q.front() -> p_sub_objects)
         q.push(p_sub_object);
      q.pop();
   }
   return p_found;
}
```

```
// 3

cl_base* cl_base::find_obj_root (string name){
   cl_base* p_root_object = this;
   while (p_root_object -> get_head() != nullptr)
      p_root_object = p_root_object -> get_head();
   return p_root_object -> find_obj_current(name);
}

bool cl_base::set_name(string s_new_name){
   if (p_head_object)
      for (auto obj : p_head_object -> p_sub_objects)
         if(obj -> get_name() == s_new_name)
            return false;
   this -> s_name = s_new_name;
   // присвоение дочернему объекту имени, если не имеется других дочерних
объектов с аналогичным именем
   return true;
}

// 4

void cl_base::set_state(int state){
   if (state == 0){
      this -> state = 0;
      for (auto p_sub_objects : p_sub_objects){
         p_sub_objects -> set_state(0);
      }
   }
   else {
      bool par_ready = true;
      cl_base* root_obj = this;

      while (root_obj -> get_head() != nullptr){
         root_obj = root_obj -> get_head();
         if (root_obj -> state == 0){
            par_ready = false;
            break;
         }
      }
      if (par_ready)
         this -> state = state;
   }
}

// 5

void cl_base::print_names_recur(int level){
   cout << endl;
   for (int i = 0; i < 4*level; i++){
      cout << " ";
   }
   cout << get_name();
```

```
   for (auto p_sub_object : p_sub_objects){
      p_sub_object -> print_names_recur(level+1);
   }
}


void cl_base::print_states_recur(int level){
   cout << endl;
   for (int i = 0; i < 4*level; i++){
      cout << " ";
   }
   cout << get_name() << (state == 0 ? " is not ready" : " is ready");

   for (auto p_sub_object : p_sub_objects)
      p_sub_object -> print_states_recur(level+1);
}
// ____________________
cl_base :: ~cl_base()
{
   for (int i = 0; i < p_sub_objects.size(); i++)
   {
      delete p_sub_objects[i];
   }
}

bool cl_base::set_new_parent(cl_base* new_parent)
{
   cl_base* root = this;
   while (root -> p_head_object)
   {
      root = root -> p_head_object;
   }
   if (this == root || new_parent == nullptr)
   {
      return false;
   }
   if (new_parent -> get_sub_object(this -> s_name) != nullptr)
   {
      return false;
   }
   cl_base* p_obj = new_parent;
   while (p_obj != nullptr)
   {
      if (p_obj == this) return false;
      p_obj=p_obj -> get_head();
   }
   for (int i = 0; i < p_head_object -> p_sub_objects.size(); i++)
   {
      if (p_head_object -> p_sub_objects[i] == this)
      {
         p_head_object      ->      p_sub_objects.erase(p_head_object      ->
p_sub_objects.begin() + i);
         break;
      }
   }
```

```
   this -> p_head_object = new_parent;
   new_parent -> p_sub_objects.push_back(this);
   return true;
}

void cl_base::remove_child (string sub_name)
{
   for (int i = 0; i < p_sub_objects.size(); i++)
   {
      if (p_sub_objects[i] -> s_name == sub_name)
      {
         delete p_sub_objects[i];
         p_sub_objects.erase(p_sub_objects.begin() + i);
         return;
      }
   }
}

cl_base* cl_base::get_obj_by_way(string way)
{
   if (way.empty()) return nullptr;
   if (way == ".") return this;
   if (way[0] == '.') return find_obj_current(way.substr(1));
   if (way.substr(0,2) == "//") return find_obj_root(way.substr(2));
   if (way[0] != '/')
   {
      size_t ind = way.find('/');
      cl_base* p_sub = get_sub_object(way.substr(0, ind));
      if (!p_sub || ind == string::npos) return p_sub;
      return p_sub -> get_obj_by_way(way.substr(ind + 1));
   }
   cl_base* root = this;
   while (root->p_head_object)
   {
      root = root -> p_head_object;
   }
   if (way == "/")
   {
      return root;
   }
   return root -> get_obj_by_way(way.substr(1));
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
```

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

using namespace std;

class cl_base
{
private:
   string s_name;
   cl_base* p_head_object;
   vector <cl_base*> p_sub_objects;
   int state;

public:
   cl_base(cl_base* p_head_object, string s_name = "Base object");

   bool set_name(string s_new_name);
   string get_name();
   cl_base* get_head();

   void set_state(int state);

   cl_base* get_sub_object(string s_name);
   cl_base* find_obj_current(string name);
   cl_base* find_obj_root(string name);

   void print_names_recur(int level = 0);
   void print_states_recur(int level = 0);

   ~cl_base();

   bool set_new_parent(cl_base* new_parent);
   void remove_child(string sub_name);
   cl_base* get_obj_by_way(string way);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_base.h"
#include "cl_application.h"

int main()
{
```

```
    cl_application ob_cl_application(nullptr); // создание корневого объекта
    ob_cl_application.build_tree_objects();    //    конструирование    системы,
построение дерева объектов
    return ob_cl_application.exec_app(); // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 13.

Таблица 13 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

rootela
/ 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_7
FIND
object_4/object_7
FIND .
FIND .object_7
FIND
object_4/object_7
MOVE .object_7
SET
object_4/object_7
MOVE //object_1
MOVE /object_3
END
	Object tree
	rootela
	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_7
	Object is not found
	object_4/object_7
	Object name:
	object_7
	. Object name:
	object_2
	.object_7 Object
	name: object_7
	object_4/object_7
	Object name:
	object_7
	.object_7
	Redefining the head
	object failed
	Object is set:
	object_7
	//object_1
	Dubbing the names of
	subordinate objects
	New head object:
	object_3
	Current object
	hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_5
		Object tree
		rootela
		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_7
		Object is not found
		object_4/object_7
		Object name:
		object_7
		. Object name:
		object_2
		.object_7 Object
		name: object_7
		object_4/object_7
		Object name:
		object_7
		.object_7
		Redefining the head
		object failed
		Object is set:
		object_7
		//object_1
		Dubbing the names of
		subordinate objects
		New head object:
		object_3
		Current object
		hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_5

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_3 object_3 object_7	object_3 object_3 object_7
rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_8 object_3 6	Object tree rootela object_1 object_2 object_4 object_5 object_3 The head object /object_8 is not found	Object tree rootela object_1 object_2 object_4 object_5 object_3 The head object /object_8 is not found
rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 endtree SET /object_2 DELETE object_4 DELETE object_5 END	Object tree rootela object_1 object_2 object_4 object_5 object_3 Object is set: object_2 The object /object_2/object_4 has been deleted The object /object_2/object_5 has been deleted Current object hierarchy tree rootela object_1 object_2 object_3	Object tree rootela object_1 object_2 object_4 object_5 object_3 Object is set: object_2 The object /object_2/object_4 has been deleted The object /object_2/object_5 has been deleted Current object hierarchy tree rootela object_1 object_2 object_3
rootela / 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 FIND //object_7 SET /object_2	Object tree rootela 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_7 Object is not found Object is set:	Object tree rootela 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_7 Object is not found Object is set:

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

FIND //object_7
FIND object_4/object_7
FIND /./../.
FIND ./../.
FIND ///
FIND /object_2/./object_4
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
SET /object_2
DELETE object_7
DELETE object_5
END
	object_2
	//object_7
	Object is not found
	object_4/object_7
	Object name: object_7
	/./../. Object is not found
	./../. Object is not found
	/// Object is not found
	/object_2/./object_4
	Object is not found
	.object_7 Object name: object_7
	object_4/object_7
	Object name: object_7
	.object_7
	Redefining the head object failed
	Object is set: object_7
	//object_1
	Dubbing the names of subordinate objects
	New head object: object_3
	Object is set: object_2
	The object /object_2/object_5 has been deleted
	Current object hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_3
	object_3
	object_7
		object_2
		//object_7
		Object is not found
		object_4/object_7
		Object name: object_7
		/./../. Object is not found
		./../. Object is not found
		/// Object is not found
		/object_2/./object_4
		Object is not found
		.object_7 Object name: object_7
		object_4/object_7
		Object name: object_7
		.object_7
		Redefining the head object failed
		Object is set: object_7
		//object_1
		Dubbing the names of subordinate objects
		New head object: object_3
		Object is set: object_2
		The object /object_2/object_5 has been deleted
		Current object hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_3
		object_3
		object_7

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

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

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

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

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

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