Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ.....6
1.1 Описание входных данных.....8
1.2 Описание выходных данных.....9
2 МЕТОД РЕШЕНИЯ.....12
3 ОПИСАНИЕ АЛГОРИТМОВ.....14
3.1 Алгоритм функции main.....14
3.2 Алгоритм метода set_parent класса cl_base.....14
3.3 Алгоритм метода delete_child класса cl_base.....16
3.4 Алгоритм метода get_by_coordinate класса cl_base.....17
3.5 Алгоритм метода build_tree_objects класса cl_application.....19
3.6 Алгоритм метода exec_app класса cl_application.....22
3.7 Алгоритм деструктора класса cl_base.....25
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....27
5 КОД ПРОГРАММЫ.....45
5.1 Файл cl_2.cpp.....45
5.2 Файл cl_2.h.....45
5.3 Файл cl_3.cpp.....45
5.4 Файл cl_3.h.....46
5.5 Файл cl_4.cpp.....46
5.6 Файл cl_4.h.....46
5.7 Файл cl_5.cpp.....47
5.8 Файл cl_5.h.....47
5.9 Файл cl_6.cpp.....47
5.10 Файл cl_6.h.....47
5.11 Файл cl_application.cpp.....48
5.12 Файл cl_application.h.....51

5.13 Файл cl_base.cpp.....52
5.14 Файл cl_base.h.....57
5.15 Файл main.cpp.....59
6 ТЕСТИРОВАНИЕ.....60
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....62

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 используются:

- объект стандартного потока ввода std::cin (используется для ввода с клавиатуры);
- объект стандартного потока вывода std::cout (используется для вывода на экран);
- оператор цикла со счётчиком for;
- оператор цикла с условием while;
- условный оператор if..else;
- объекты классов cl_application, cl_2, cl_3, cl_4, cl_5, cl_6.

Класс cl_base:

- Свойства (поля): дополнительных свойств добавлено не было;
- Методы:
  - Метод set_parent:
    - Функционал - установка нового головного объекта;
    - Возвращаемое значение - логическое значение;
    - Модификатор доступа - открытый
    - Параметры:
      - new_parent - указатель на новый головной объект;
  - Метод delete_child:
    - Функционал - удаление подчинённого объекта по имени;
    - Возвращаемое значение - координата удалённого объекта (строка);
    - Модификатор доступа - открытый
    - Параметры:
      - child_name - имя объекта, который требуется удалить

(строка);

- Метод get_by_coordinate:
  - Функционал - получение объекта из дерева иерархии объектов по координате;
  - Возвращаемое значение - указатель на объект класса cl_base;
  - Модификатор доступа - открытый
  - Параметры:
    - coordinate - координата искомого объекта (строка).

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: отсутствуют.

Возвращаемое значение: целочисленный код завершения работы программы.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_cl_application класса cl_application с использованием параметризованного конструктора и нулевого указателя (nullptr) в качестве параметра	2
2		Вызов метода build_tree_objects объекта ob_cl_application	3
3		Возвращение значения, возвращённого методом exec_app класса ob_cl_application	∅

3.2 Алгоритм метода set_parent класса cl_base

Функционал: установка нового головного объекта.

Параметры: new_parent - указатель на новый головной объект.

Возвращаемое значение: логическое значение.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_parent класса cl_base

№	Предикат	Действия	№ перехода
1	Значение поля parent равно нулевому указателю или значение указателя new_parent равно нулевому указателю или значение, возвращённое методом get_child объекта new_parent с полем name в качестве аргумента, не равно нулевому указателю	Возврат значения логическая ложь (false)	∅
			2
2		Инициализация указателя на объект класса cl_base temp значением указателя new_parent	3
3	Значение указателя temp не равно нулевому указателю		4
			5
4	Значение указателя temp равно значению указателя this (указателя на текущий объект)	Возврат значения логическая ложь (false)	∅
		Присваивание указателю temp поля parent объекта, на который указывает указатель temp	3
5		Инициализация целочисленной переменной i значением 0	6
6	Значение переменной i меньше значения, возвращённого методом size поля children поля parent		7

№	Предикат	Действия	№ перехода
			9
7	Значение элемента поля children поля parent с индексом i равно значению указателя this (указателя на текущий объект)	Вызов метода erase поля children поля parent с суммой значения переменной i и значения, возвращённого методом begin поля children поля parent, в качестве аргумента	8
			8
8		Увеличене значения переменной i на 1	6
9		Присваивание полю parent указателя new_parent	10
10		Вызов метода push_back поля children объекта, на который указывает new_parent с указателем this в качестве аргумента	11
11		Возврат значения логическая истина (true)	∅

3.3 Алгоритм метода delete_child класса cl_base

Функционал: удаление подчинённого объекта по имени.

Параметры: child_name - имя объекта, который требуется удалить (строка).

Возвращаемое значение: координата удалённого объекта (строка).

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода delete_child класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя на объект cl_base to_delete значением, возвращённым методом get_child со значением child_name в качестве параметра	2
2		Объявление строковой переменной	3

№	Предикат	Действия	№ перехода
		delete_coordinate	
3		Инициализация указателя на объект cl_base temp значением указателя to_delete	4
4	Значение, возвращённое методом get_parent объекта temp не равно нулевому указателю		5
			7
5		Присваивание переменной delete_coordinate суммы строки "/", значения, возвращённого методом get_parent объекта temp и значения переменной delete_coordinate	6
6		Присваивание указателю temp значения, возвращённого методом get_parent объекта temp	4
7		Вызов метода disconnect_child со значением child_name в качестве параметра	8
8		Удаление объекта to_delete с помощью оператора delete	9
9		Возвращение значения delete_coordinate	∅

3.4 Алгоритм метода get_by_coordinate класса cl_base

Функционал: получение объекта из дерева иерархии объектов по координате.

Параметры: coordinate - координата искомого объекта (строка).

Возвращаемое значение: указатель на объект класса cl_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_by_coordinate класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя на объект класса cl_base root значением, возвращённым методом get_root	2
2	Значение параметра coordinate равно "/"	Возврат значения указателя root	∅
			3
3	Значение параметра coordinate равно "."	Возврат значения указателя this	∅
			4
4	Значение, возвращённое методом substr параметра coordinate со значениеями 0 и 2 в качестве параметров, равно "//"		∅
			5
5	Значение элемента параметра coordinate с индексом 0 равно '.'		∅
			6
6	Значение элемента параметра coordinate с индексом 0 равно '/'		∅
			7
7		Объявление строковой переменной object_name	8
8		Инициализация целочисленной переменной i значением 0	9
9	Значение переменной i меньше значения, возвращённого методом		10

№	Предикат	Действия	№ перехода
	length параметра coordinate		
			14
10	Значение элемента параметра coordinate с индексом i не равно '/'	Присваивание переменной object_name значения суммы значения object_name и значения элемента аргумента coordinate с индексом i	13
			11
11	Значение, возвращённое методом get_child со значением переменной object_name в качестве параметра равно нулевому указателю	Возврат нулевого указателя	∅
			12
12		Возврат значения, возвращённого методом get_by_coordinate объекта, возвращённого методом get_child со значением параметра object_name в качестве параметра, вызванного со значениями длины object_name + 1 и длины coordinate - 1 в качестве параметров	∅
13		Увеличение значения переменной i на 1	9
14		Возврат значения, возврашённого методом get_child со значением object_name в качестве параметра	∅

3.5 Алгоритм метода build_tree_objects класса cl_application

Функционал: построение исходного дерева иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной parent_coordinate	2
2		Объявление строковой переменной child_name	3
3		Инициализация целочисленной переменной class_number значением 0	4
4		Инициализация указателя на объект класса cl_base input_parent значением nullptr (нулевой указатель)	5
5		Инициализация указателя на объект класса cl_base input_child значением указателя this (указатель на текущий объект)	6
6		Ввод значения переменной child_name	7
7		Вызов метода set_name со значением переменной child_name в качестве параметра	8
8		Ввод значения переменной parent_coordinate	9
9	Значение переменной parent_coordinate равно "endtree"		∅
			10
10		Ввод значения переменной child_name и значения переменной class_number	11
11		Присваивание указателю input_parent значения, возвращённого методом get_by_coordinate с значением parent_coordinate в качестве параметра	12
12	Значение указателя input_parent не равно нулевому указателю (nullptr)		13

№	Предикат	Действия	№ перехода
			19
13	Значение, возвращённое методом get_child объекта input_parent с значением child_name в качестве аргумента, равно нулевому указателю		14
		Вывод значения переменной parent_coordinate и строки " Dubbing the names of subordinate objects\n"	8
14	Значение переменной class_number равно 2	Присваивание указателю input_child адреса нового объекта класса cl_2, созданного с помощью оператора new с использованием параметризованного конструктора	8
			15
15	Значение переменной class_number равно 3	Присваивание указателю input_child адреса нового объекта класса cl_3, созданного с помощью оператора new с использованием параметризованного конструктора	8
			16
16	Значение переменной class_number равно 4	Присваивание указателю input_child адреса нового объекта класса cl_4, созданного с помощью оператора new с использованием параметризованного конструктора	8
			17
17	Значение переменной class_number равно 5	Присваивание указателю input_child адреса нового объекта класса cl_5, созданного с помощью оператора new с использованием параметризованного конструктора	8

№	Предикат	Действия	№ перехода
			18
18	Значение переменной class_number равно 6	Присваивание указателю input_child адреса нового объекта класса cl_6, созданного с помощью оператора new с использованием параметризованного конструктора	8
			8
19		Вывод строки "Object tree\n"	20
20		Вызов метода print_tree со строкой " " (4 пробела) в качестве аргумента	21
21		Вывод строки "\nThe head object ", значения parent_coordinate и строки " is not found"	∅

3.6 Алгоритм метода exec_app класса cl_application

Функционал: запуск приложения.

Параметры: отсутствуют.

Возвращаемое значение: целочисленный код завершения работы приложения.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод строки "Object tree\n"	2
2		Вызов метода print_tree со строкой " " (4 пробела) в качестве аргумента	3
3		Вывод символа '\n' (переход на новую строку)	4
4		Объявление строковых переменных coordinate и command	5

№	Предикат	Действия	№ перехода
5		Инициализация указателя на объект класса cl_base current_object значением указателя this	6
6		Ввод значения переменной command	7
7	Значение переменной command равно "END"		8
			10
8		Вывод строки "Current object hirarchy tree\n"	9
9		Вызов метода print_tree со строкой " " (4 пробела) в качестве аргумента	25
10		Ввод значения переменной coordinate	11
11		Инициализация указателя на объект cl_base found значением, возвращённым методом get_by_coordinate объекта current_object со значением переменной coordinate в качестве параметра	12
12	Значение переменной command равно "SET"		13
			16
13	Значение указателя found не равно нулевому указателю		14
		Вывод строки "The object was not found at the specified coordinate: ", значения переменной coordinate и символа '\n'	6
14		Присваивание указателю current_object значения указателя found	15
15		Вывод строки "Object is set: ", значения, возвращённого методом get_name объекта current_object и символа '\n'	6

№	Предикат	Действия	№ перехода
16	Значение переменной command равно "FIND"		17
			18
17	Значение указателя found не равно нулевому указателю	Вывод значения переменной coordinate, строки " Object name: ", значения, возвращённого методом get_name объекта found и символа '\n'	6
		Вывод значения переменной coordinate и строки " Object is not found\n"	6
18	Значение переменной command равно "MOVE"		19
			22
19	Значение указателя found равно нулевому указателю	Вывод значения переменной coordinate и строки " Head object is not found\n"	6
			20
20	Значение, возвращённое методом get_child объекта found с значением, возвращённым методом get_name объекта current_object, в качестве аргумента, не равно нулевому указателю	Вывод значения переменной coordinate и строки " Dubbing the names of subordinate objects\n"	6
			21
21	Значение, возвращённое методом set_parent объекта current_objects со значением указателя found в качестве аргумента, равно логической лжи	Вывод значения переменной coordinate и строки " Redefining the head object failed\n"	6

№	Предикат	Действия	№ перехода
		Вывод строки "New head object: ", значения, возвращённого методом get_name объекта found и символа '\n'	6
22	Значение переменной command равно "DELETE"		23
			6
23		Инициализация строковой переменной deleted_coordinate значением, возвращённым методом delete_child объекта current_object со значением coordinate в качестве параметра	24
24		Вывод строки "The object ", значения переменной deleted_coordinate и строки " has been deleted\n"	6
25		Возврат значения 0	∅

3.7 Алгоритм деструктора класса cl_base

Функционал: удаление объекта cl_base.

Параметры: отсутствуют.

Алгоритм деструктора представлен в таблице 7.

Таблица 7 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i значением 0	2
2	Значение переменной i меньше значения, возвращённого методом size поля children	Удаление элемента поля children с индексом i с помощью оператора delete	3

№	Предикат	Действия	№ перехода
			∅
3		Увеличение значения переменной i на 1	2

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

Рисунок 1 – Блок-схема алгоритма

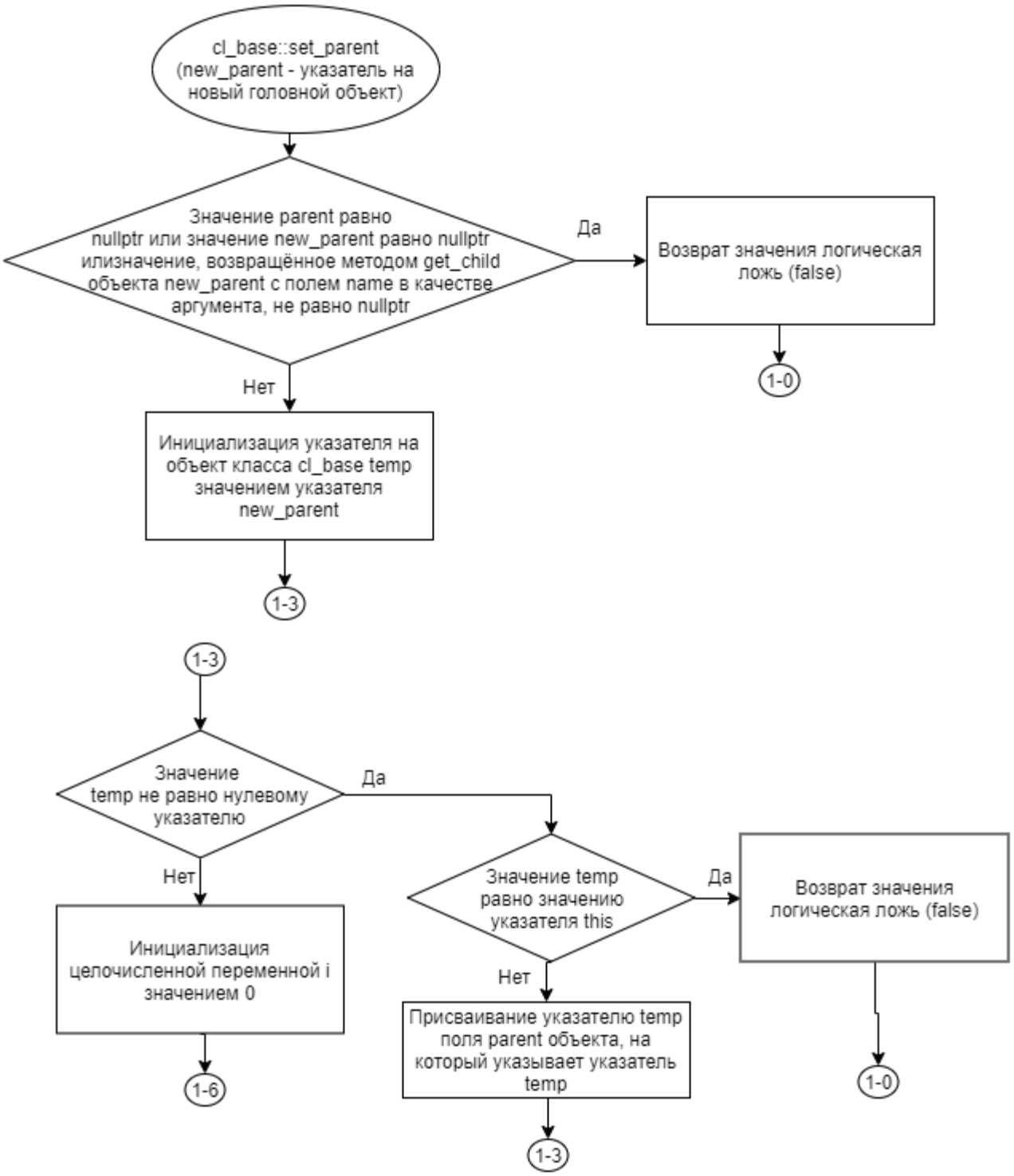

Рисунок 2 – Блок-схема алгоритма

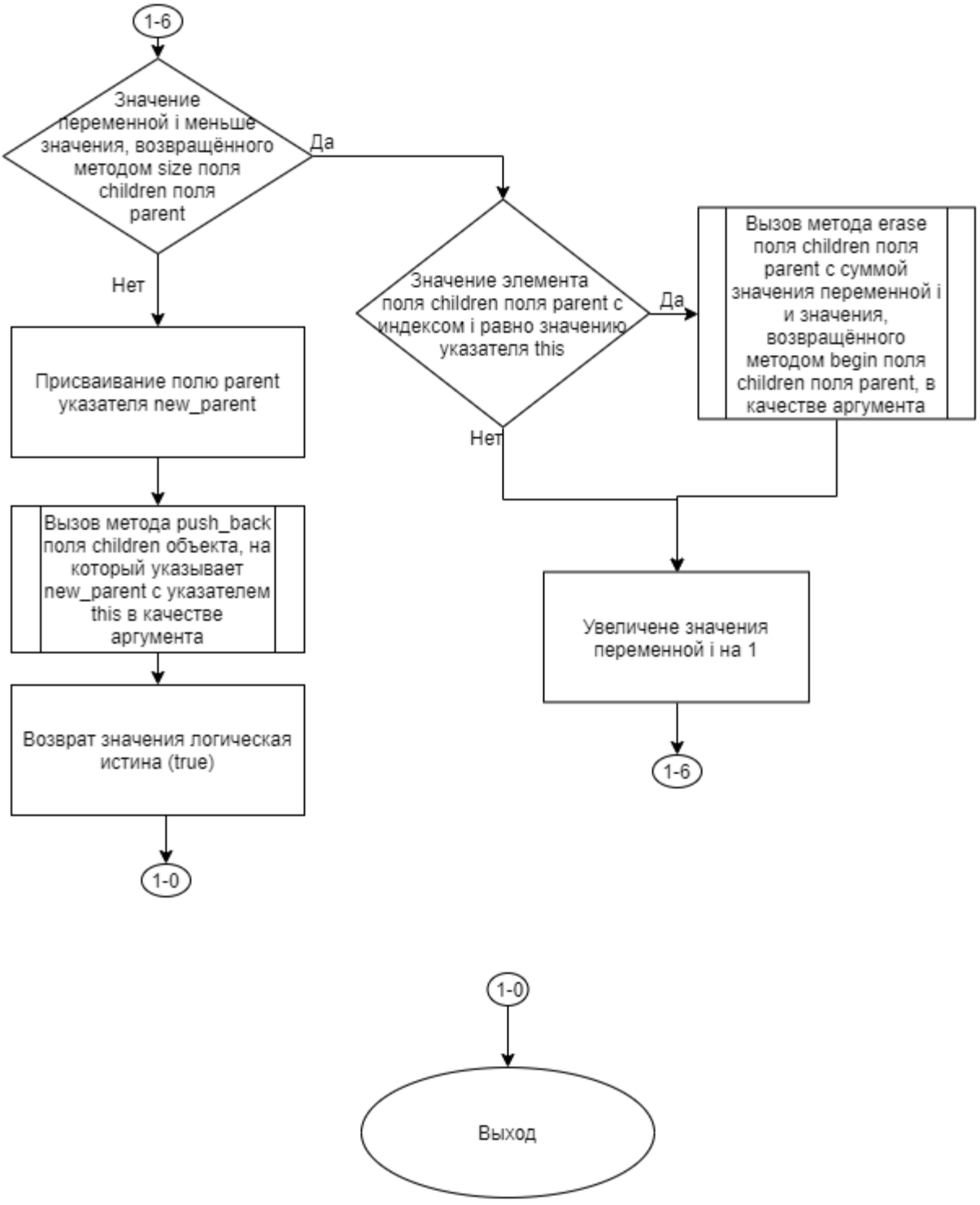

Рисунок 3 – Блок-схема алгоритма

Рисунок 4 – Блок-схема алгоритма

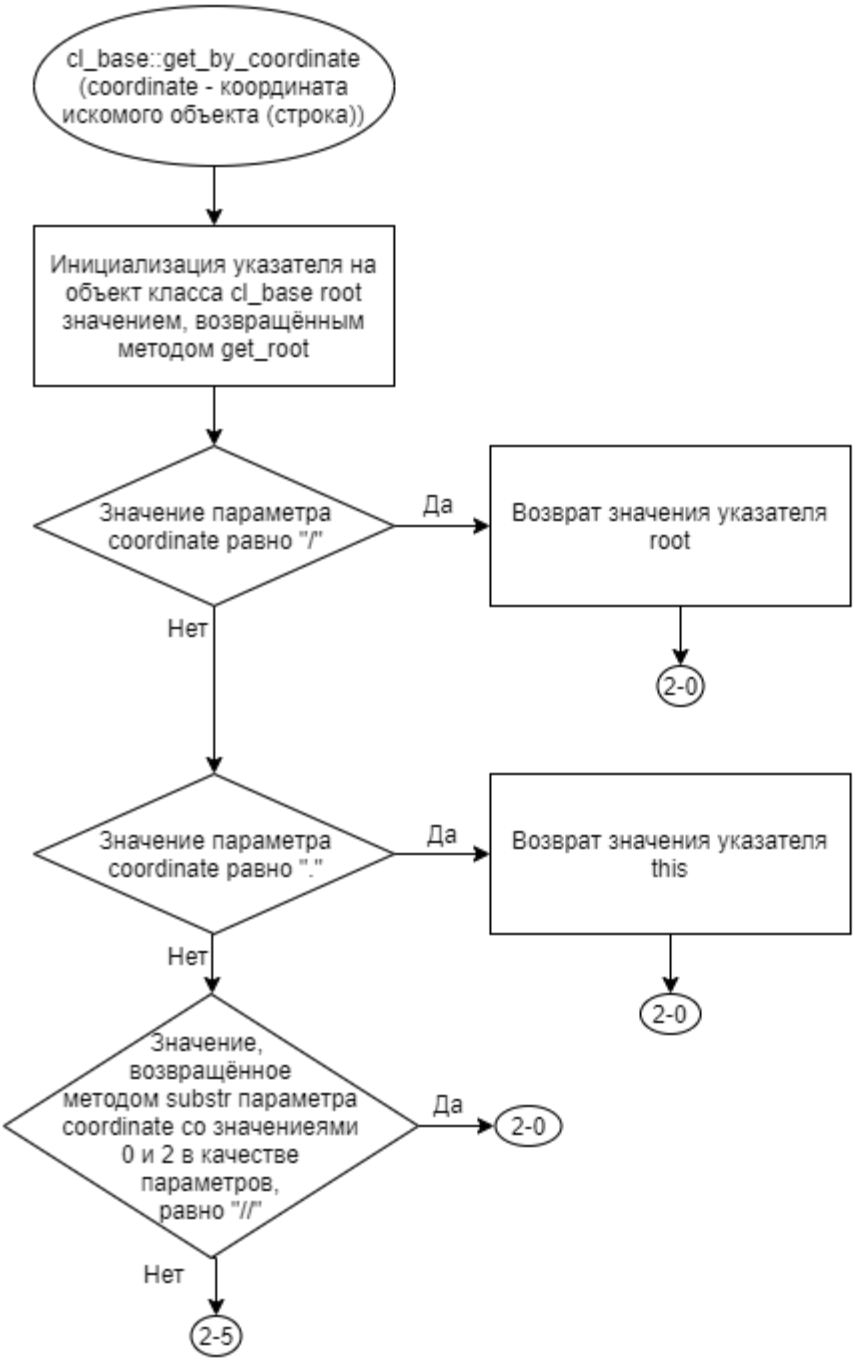

Рисунок 5 – Блок-схема алгоритма

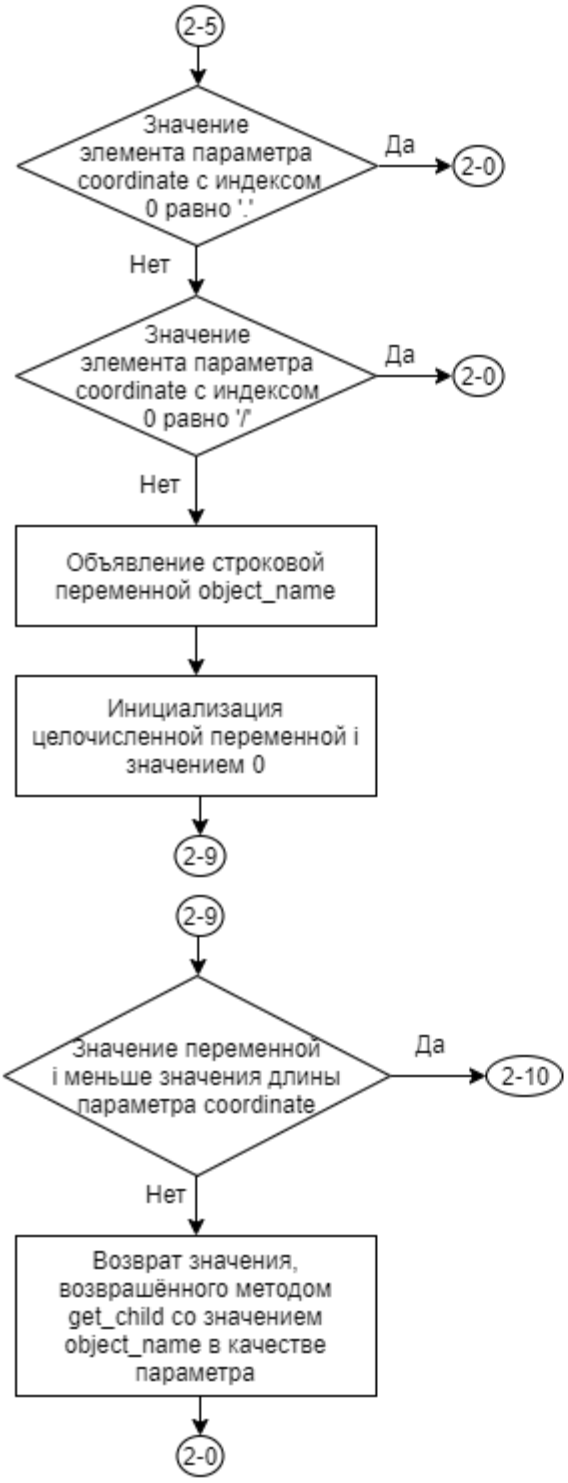

Рисунок 6 – Блок-схема алгоритма

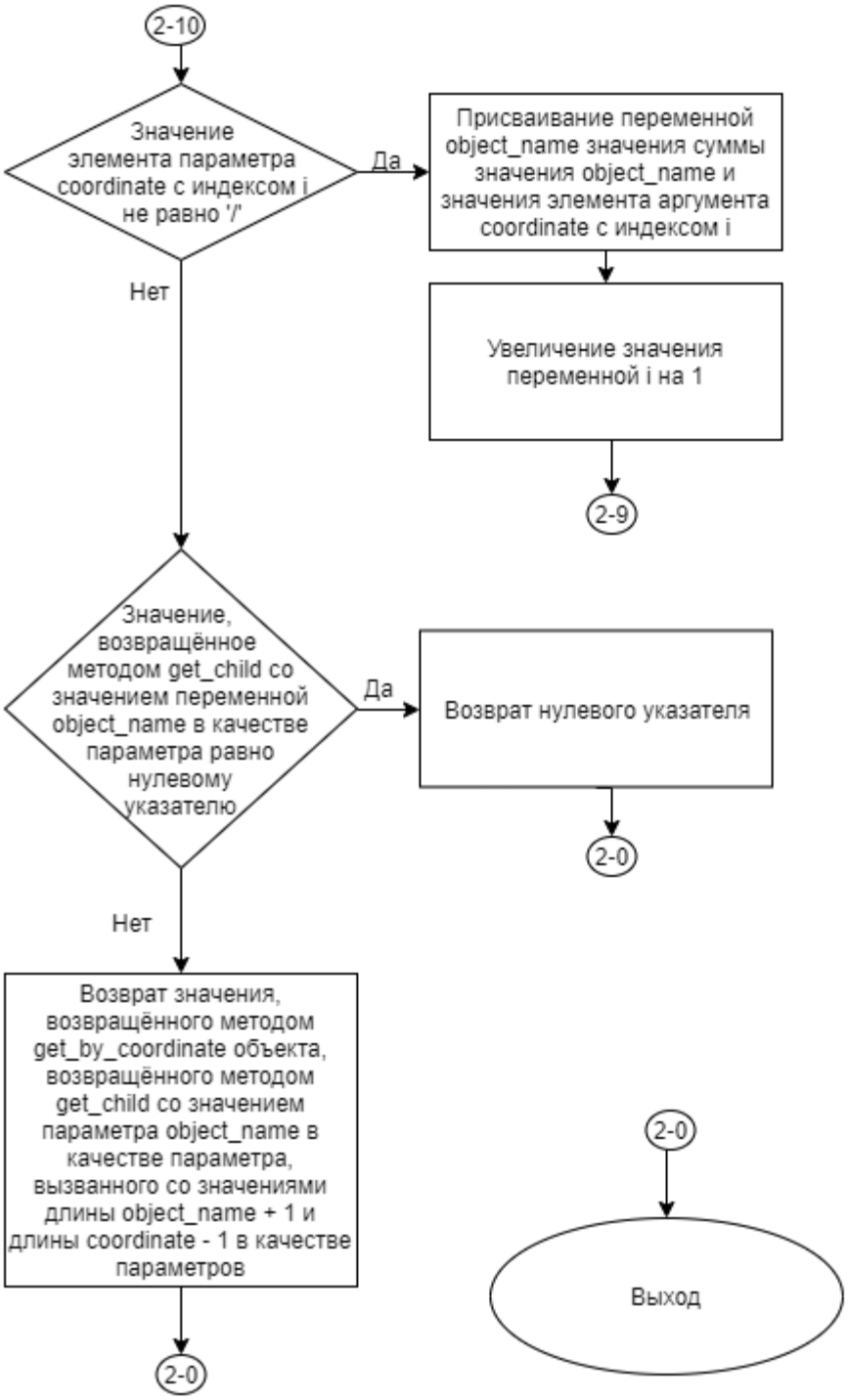

Рисунок 7 – Блок-схема алгоритма

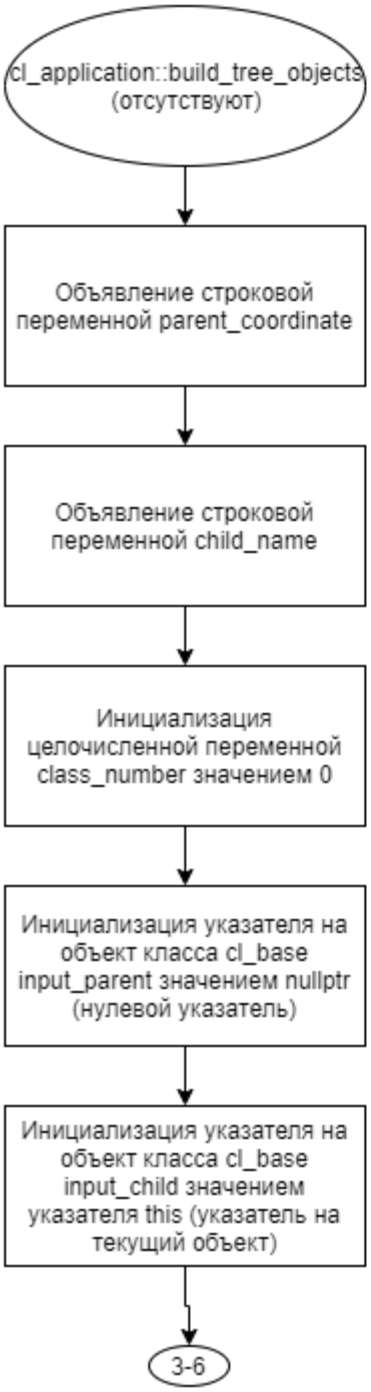

Рисунок 8 – Блок-схема алгоритма

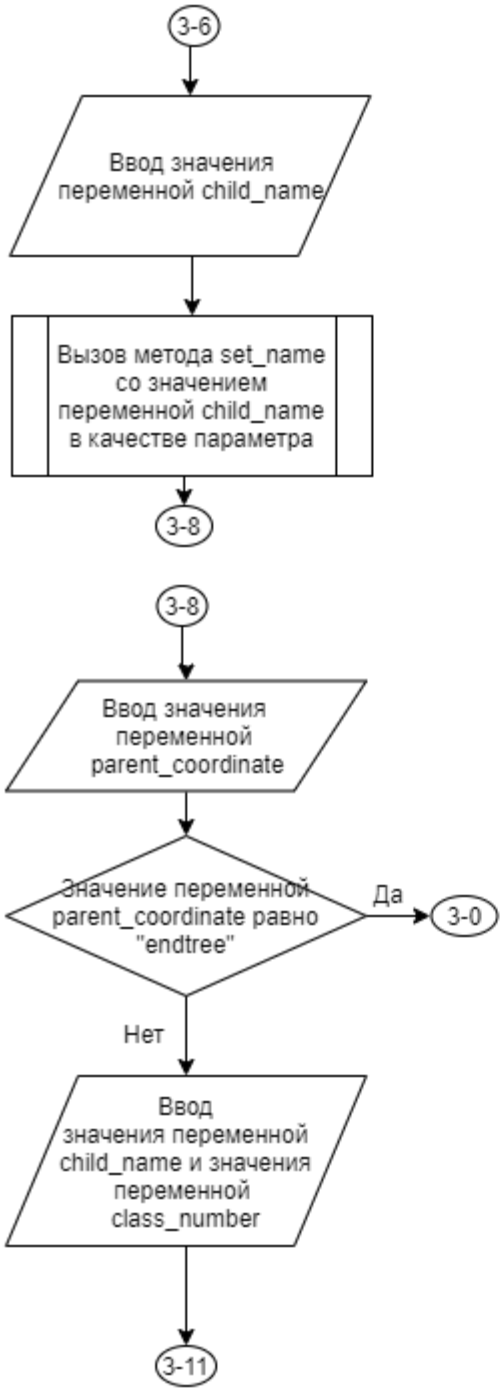

Рисунок 9 – Блок-схема алгоритма

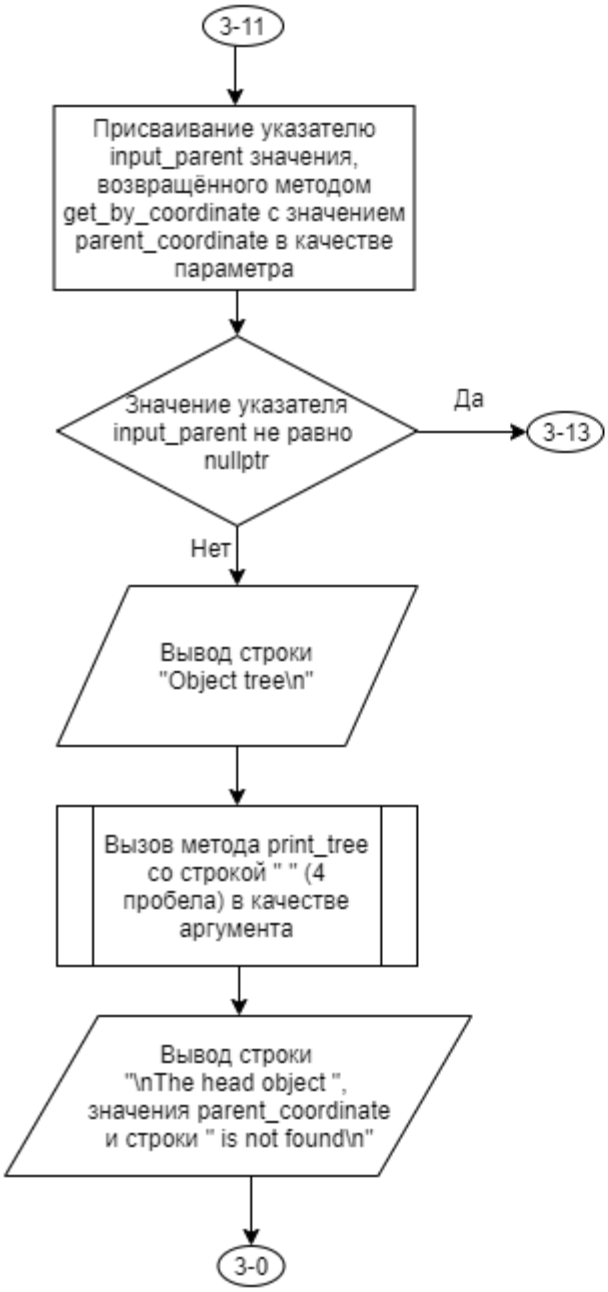

Рисунок 10 – Блок-схема алгоритма

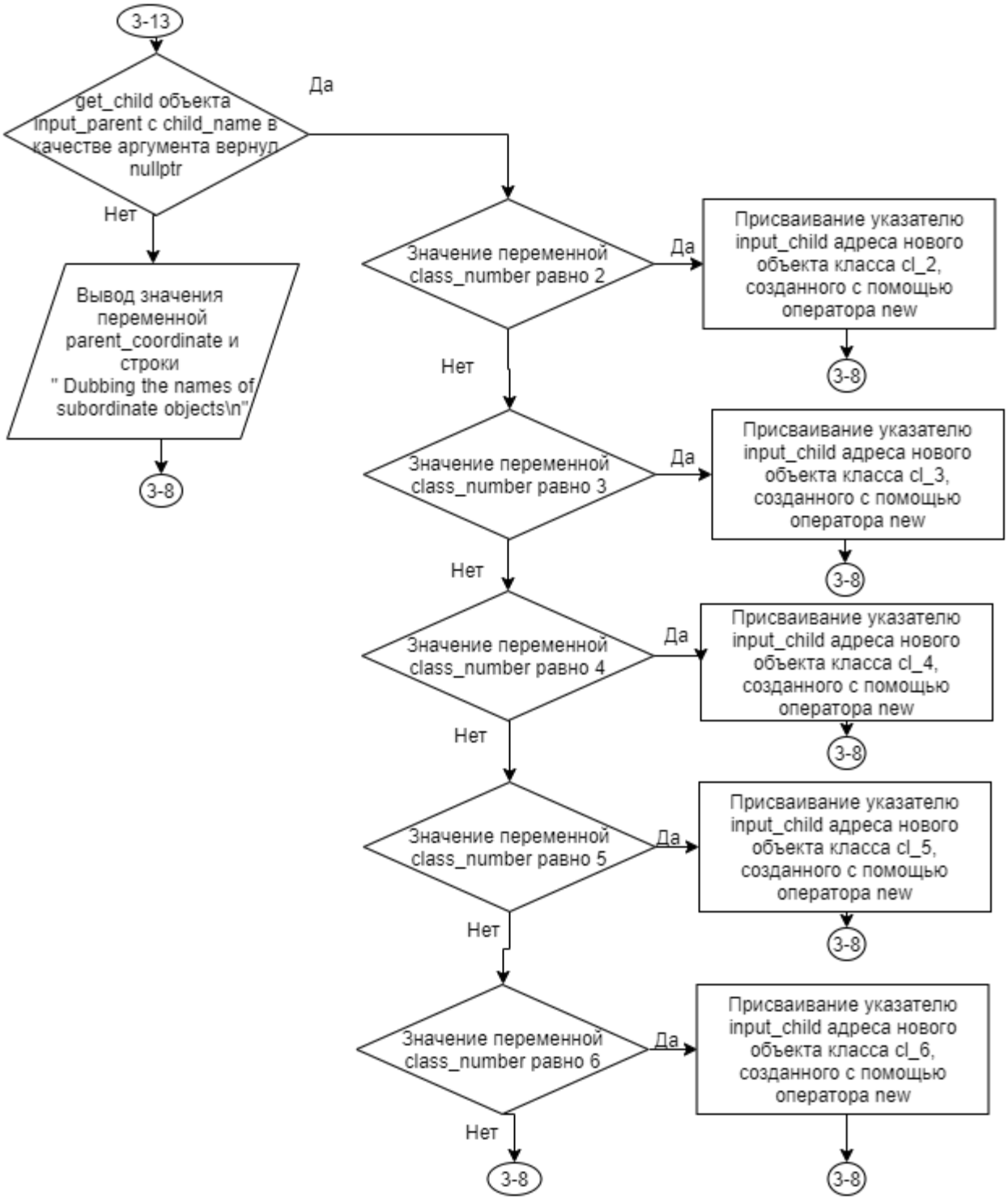

Рисунок 11 – Блок-схема алгоритма

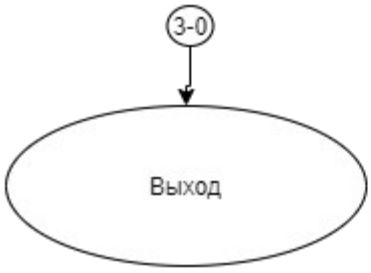

Рисунок 12 – Блок-схема алгоритма

Рисунок 13 – Блок-схема алгоритма

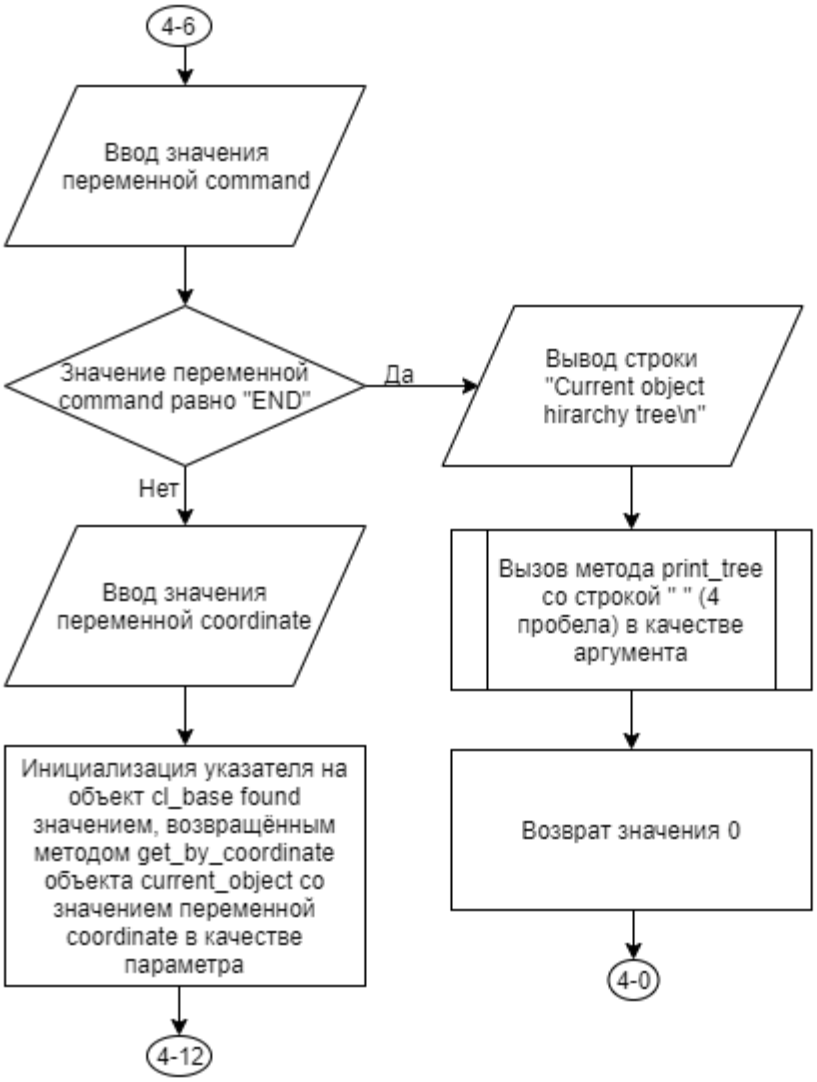

Рисунок 14 – Блок-схема алгоритма

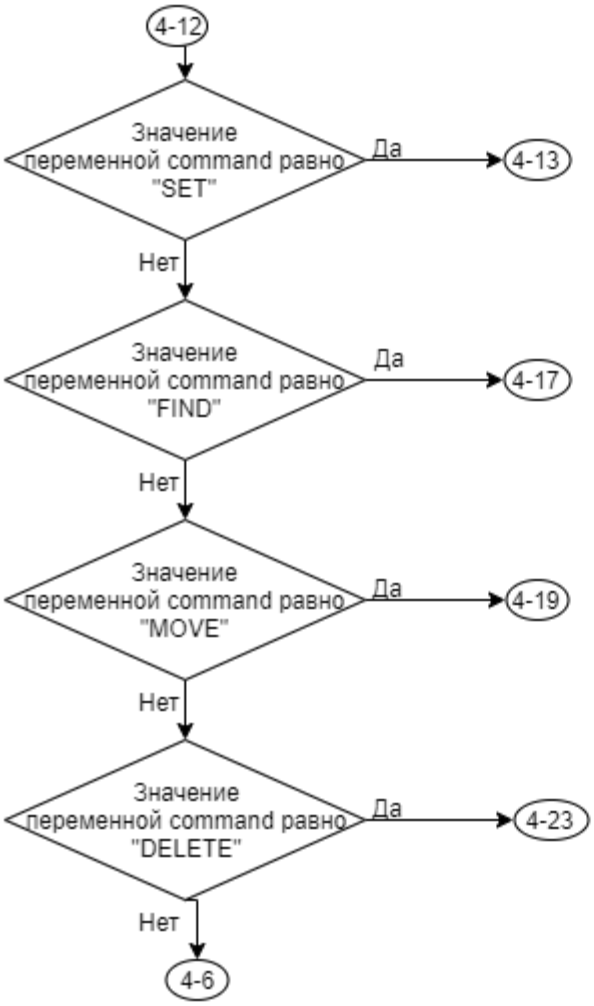

Рисунок 15 – Блок-схема алгоритма

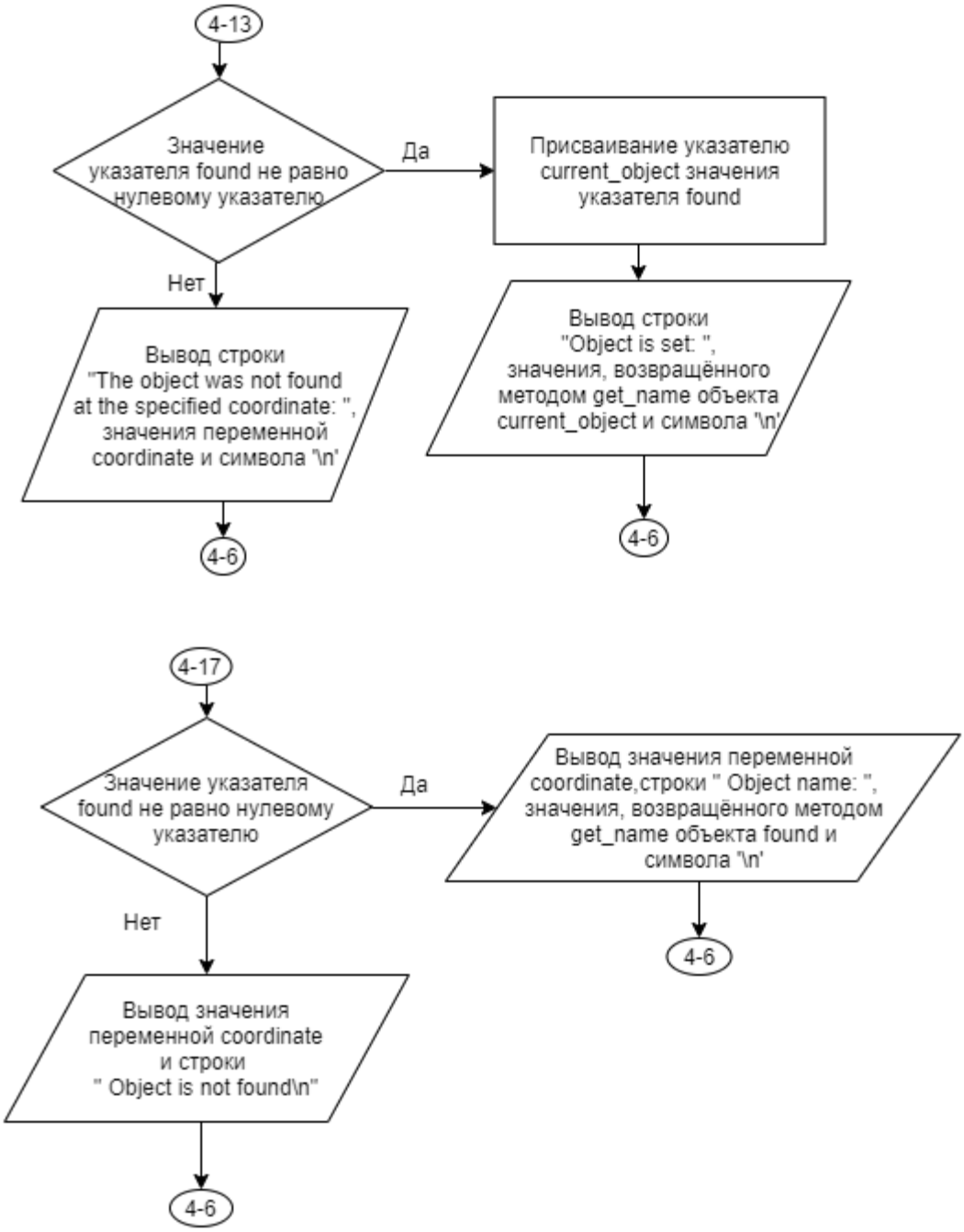

Рисунок 16 – Блок-схема алгоритма

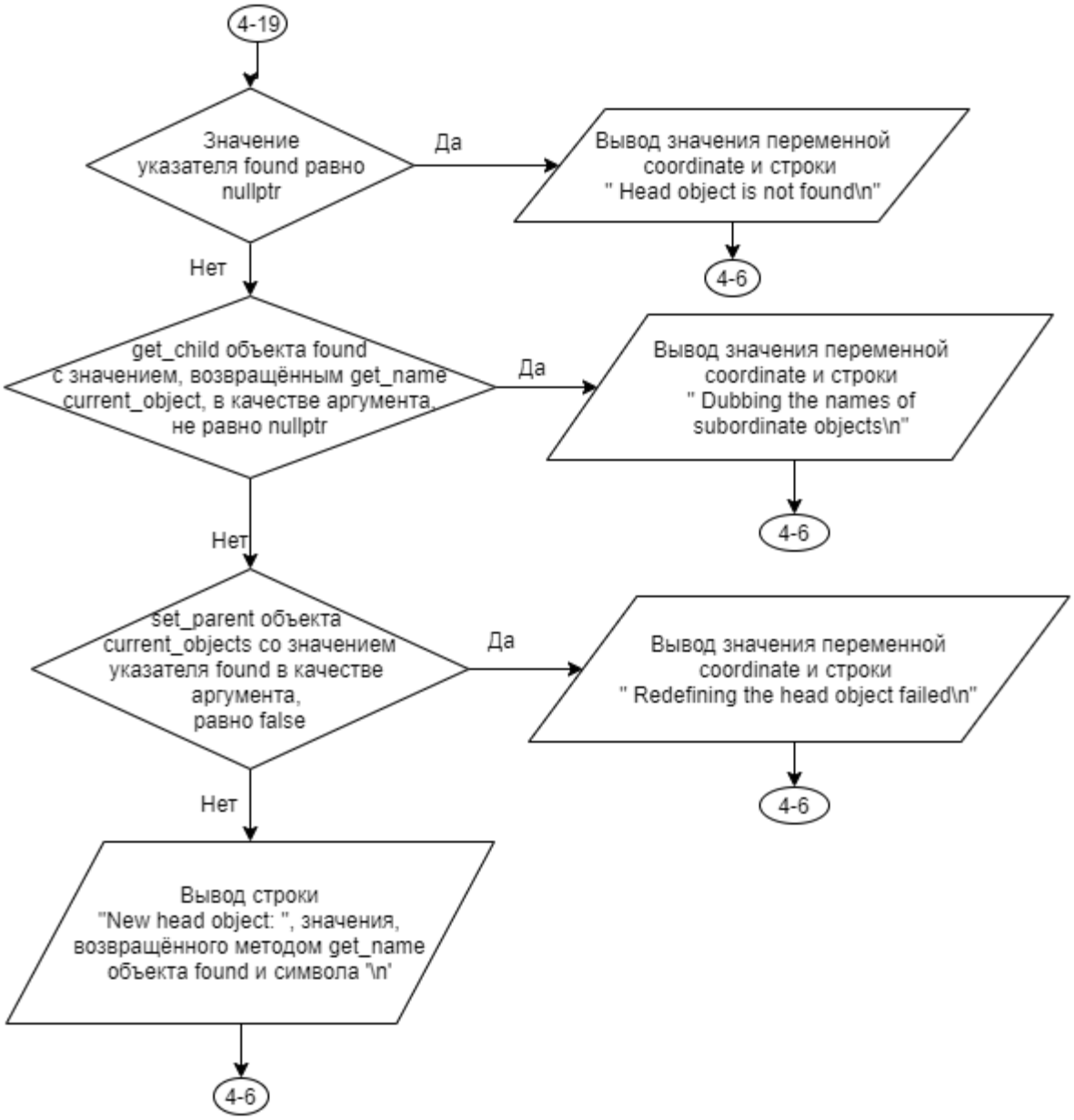

Рисунок 17 – Блок-схема алгоритма

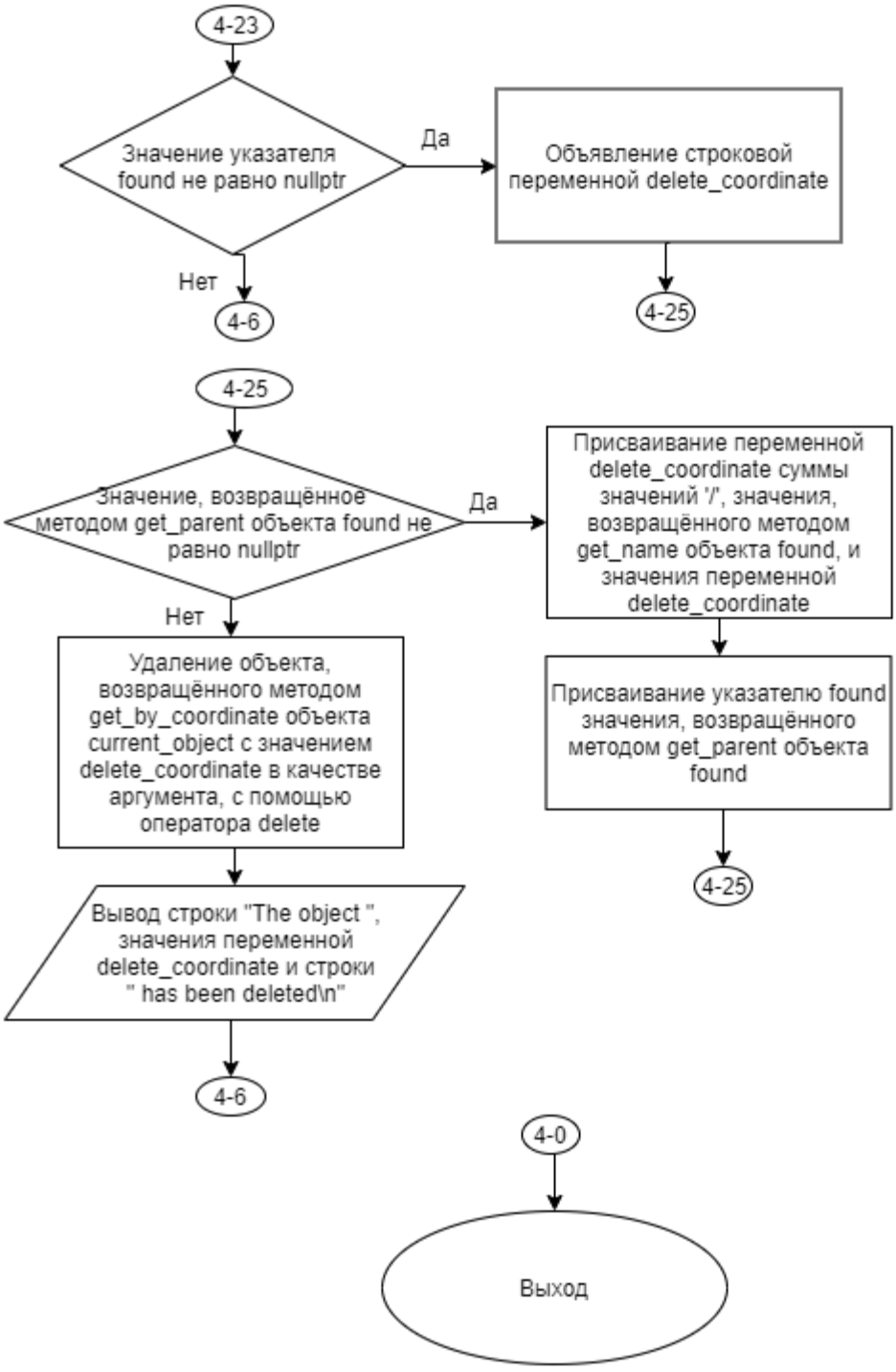

Рисунок 18 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(cl_base* parent, std::string name)
	: cl_base(parent, name) {}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H

#include "cl_base.h"
#include <string>

class cl_2 : public cl_base
{
public:
	cl_2(cl_base* parent, std::string name);
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* parent, std::string name)
	: cl_base(parent, name) {}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H

#include "cl_base.h"
#include <string>

class cl_3 : public cl_base
{
public:
	cl_3(cl_base* parent, std::string name);
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* parent, std::string name)
	: cl_base(parent, name) {}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H

#include "cl_base.h"
#include <string>

class cl_4 : public cl_base
{
public:
	cl_4(cl_base* parent, std::string name);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* parent, std::string name)
      : cl_base(parent, name) {}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H

#include "cl_base.h"
#include <string>

class cl_5 : public cl_base
{
public:
      cl_5(cl_base* parent, std::string name);
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* parent, std::string name)
      : cl_base(parent, name) {}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H

```

```
#include "cl_base.h"
#include <string>

class cl_6 : public cl_base
{
public:
	cl_6(cl_base* parent, std::string name);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"

// Параметризованный конструктор
cl_application::cl_application(cl_application* parent) : cl_base(parent) {}

// Метод построения исходного дерева иерархии объектов
void cl_application::build_tree_objects()
{
	std::string parent_coordinate;
	std::string child_name;
	int class_number = 0;
	cl_base* input_parent = nullptr;
	cl_base* input_child = this;

	std::cin >> child_name;
	set_name(child_name);

	while (true)
	{
		std::cin >> parent_coordinate;
		if (parent_coordinate == "endtree")
		{
			break;
		}

		std::cin >> child_name >> class_number;
		input_parent = get_by_coordinate(parent_coordinate);

		if (input_parent != nullptr)
		{
			if (input_parent->get_child(child_name) == nullptr)
			{
				if (class_number == 2)
				{
					input_child = new cl_2(input_parent, child_name);
				}
				else if (class_number == 3)
				{
```

```
                              input_child = new cl_3(input_parent, child_name);
                        }
                        else if (class_number == 4)
                        {
                              input_child = new cl_4(input_parent, child_name);
                        }
                        else if (class_number == 5)
                        {
                              input_child = new cl_5(input_parent, child_name);
                        }
                        else if (class_number == 6)
                        {
                              input_child = new cl_6(input_parent, child_name);
                        }
                  }
                  else
                  {
                        std::cout << parent_coordinate
                              << "    Dubbing the names of subordinate objects\n";
                  }
            }
            else
            {
                  std::cout << "Object tree\n";
                  print_tree("    ");
                  std::cout << "\nThe head object "
                        << parent_coordinate << " is not found";
                  exit(0);
            }
      }
}

// Метод запуска приложения
int cl_application::exec_app()
{
      std::cout << "Object tree\n";
      print_tree("    ");
      std::cout << '\n';

      std::string coordinate, command;
      cl_base* current_object = this;

      while (true)
      {
            std::cin >> command;
            if (command == "END")
            {
                  std::cout << "Current object hierarchy tree\n";
                  print_tree("    ");
                  break;
            }

            std::cin >> coordinate;
            cl_base* found = current_object->get_by_coordinate(coordinate);

            if (command == "SET")
            {
```

```
                  if (found != nullptr)
                  {
                        current_object = found;
                        std::cout  <<  "Object  is  set:  "  <<  current_object-
>get_name() << '\n';
                  }
                  else
                  {
                        std::cout << "The object was not found at the specified
coordinate: "
                              << coordinate << '\n';
                  }
            }
            else if (command == "FIND")
            {
                  if (found != nullptr)
                  {
                        std::cout << coordinate << "     Object name: " << found-
>get_name() << '\n';
                  }
                  else
                  {
                        std::cout << coordinate << "     Object is not found\n";
                  }
            }
            else if (command == "MOVE")
            {
                  if (found == nullptr)
                  {
                        std::cout << coordinate << "     Head object is not found\
n";
                  }
                  else    if    (found->get_child(current_object->get_name())    !=
nullptr)
                  {
                        std::cout  <<  coordinate  <<  "       Dubbing  the  names  of
subordinate objects\n";
                  }
                  else if (current_object->set_parent(found) == false)
                  {
                        std::cout  <<  coordinate  <<  "        Redefining  the  head
object failed\n";
                  }
                  else
                  {
                        std::cout << "New head object: " << found->get_name() <<
'\n';
                  }
            }
            else if (command == "DELETE")
            {
                  std::string      deleted_coordinate      =      current_object-
>delete_child(coordinate);
                  if (deleted_coordinate != "")
                  {
                        std::cout << "The object " << deleted_coordinate << " has
been deleted\n";
```

```
                  }
            }
      }

      return 0;
}

void cl_application::read_states()
{
      std::string input_name;
      int input_state = 0;

      while (std::cin >> input_name >> input_state)
      {
            cl_base* obj = find_in_tree(input_name);
            if (obj != nullptr)
            {
                  obj->set_state(input_state);
            }
      }
}
```

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"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <iostream>
#include <string>

class cl_application : public cl_base
{
private:
public:

      // Параметризованный конструктор
      cl_application(cl_application* parent);

      // Метод построения исходного дерева иерархии объектов
      void build_tree_objects();

      // Метод запуска приложения
      int exec_app();

      // Метод для считывания имён объектов и их готовностей
```

```
	void read_states();
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"

// Параметризированный конструктор
cl_base::cl_base(cl_base* parent, std::string name)
{
	// Установка родительского объекта
	this->parent = parent;
	// Установка имени объекта
	this->name = name;

	if (parent != nullptr)
	{
		// Добавление объекта к подчинённым
		// объектом родительского объекта
		parent->children.push_back(this);
	}
}

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

// Метод редактирования имени объекта
bool cl_base::set_name(std::string new_name)
{
	if (parent != nullptr)
	{
		for (int i = 0; i < parent->children.size(); i++)
		{
			if (parent->children[i]->name == new_name)
			{
				// Объект с имененм, совпадающим
				// с новым именем найден среди
				// подчинённых объектов родительского
				return false;
			}
		}
	}
```

```
	// Установка нового имени объекта
	this->name = new_name;
	return true;
}

// Метод получения имени объекта
std::string cl_base::get_name()
{
	return name;
}

// Метод получения указателя на головной объект текущего объекта
cl_base* cl_base::get_parent()
{
	return parent;
}

// Метод вывода наименований объектов в дереве иерархии слева
// направо и сверху вниз
void cl_base::print_tree(std::string prefix)
{
	if (parent == nullptr)
	{
		// Вывод имени текущего объекта
		std::cout << name;
	}

	// Вывод имён всех подчинённых объектов
	for (int i = 0; i < children.size(); i++)
	{
		std::cout << '\n' << prefix << children[i]->name;
		children[i]->print_tree(prefix + "    ");
	}
}

// Метод получения указателя на непосредственно подчиненный
// объект по его имени
cl_base* cl_base::get_child(std::string child_name)
{
	for (int i = 0; i < children.size(); i++)
	{
		if (children[i]->name == child_name)
		{
			// Среди подчинённых объектов
			// найден объект с именем,
			// совпадающим с искомым
			return children[i];
		}
	}

	// Объект не найден
	return nullptr;
}

cl_base* cl_base::find_in_tree(std::string wanted_name)
{
	if (name == wanted_name)
```

```
	{
		return this;
	}

	for (int i = 0; i < children.size(); i++)
	{
		cl_base* found = children[i]->find_in_tree(wanted_name);
		if (found != nullptr)
		{
			return found;
		}
	}

	return nullptr;
}

void cl_base::print_tree_with_state(std::string prefix)
{
	if (parent == nullptr)
	{
		std::cout << name << (state == 0 ? " is not ready" : " is ready");
	}

	for (int i = 0; i < children.size(); i++)
	{
		std::cout << "\n" << prefix << children[i]->get_name()
			<< (children[i]->get_state() == 0 ? " is not ready" : " is
ready");
		children[i]->print_tree_with_state(prefix + "    ");
	}
}

int cl_base::get_state()
{
	return state;
}

void cl_base::set_state(int state)
{
	if (parent == nullptr || parent->state != 0)
	{
		this->state = state;
	}
	else
	{
		this->state = 0;
	}

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

```
cl_base* cl_base::get_root()
{
	if (parent == nullptr)
	{
		return this;
	}
	else
	{
		return parent->get_root();
	}
}

cl_base* cl_base::find_in_tree_from_root(std::string wanted_name)
{
	int amount = get_root()->count_in_tree(wanted_name);
	if (amount == 1)
	{
		return get_root()->find_in_tree(wanted_name);
	}

	return nullptr;
}

int cl_base::count_in_tree(std::string wanted_name)
{
	int res = 0;
	if (name == wanted_name)
	{
		res++;
	}

	for (int i = 0; i < children.size(); i++)
	{
		res += children[i]->count_in_tree(wanted_name);
	}

	return res;
}

bool cl_base::set_parent(cl_base* new_parent)
{
	if (parent == nullptr || new_parent == nullptr ||
		new_parent->get_child(name) != nullptr)
	{
		return false;
	}

	cl_base* temp = new_parent;
	while (temp != nullptr)
	{
		if (temp == this)
		{
			return false;
		}
		temp = temp->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;
	new_parent->children.push_back(this);
	return true;
}

std::string cl_base::delete_child(std::string child_name)
{
	cl_base* to_delete = get_child(child_name);
	if (to_delete == nullptr)
	{
		return "";
	}

	std::string delete_coordinate;

	cl_base* temp = to_delete;
	while (temp->get_parent() != nullptr)
	{
		delete_coordinate = '/' + temp->get_name() + delete_coordinate;
		temp = temp->get_parent();
	}

	disconnect_child(child_name);
	delete to_delete;

	return delete_coordinate;
}

cl_base* cl_base::get_by_coordinate(std::string coordinate)
{
	cl_base* root = get_root();

	if (coordinate == "/")
	{
		return root;
	}

	if (coordinate == ".")
	{
		return this;
	}

	if (coordinate.substr(0, 2) == "//")
	{
		return find_in_tree_from_root(coordinate.substr(2, coordinate.length()
- 1));
	}

	if (coordinate[0] == '.')
```

```
	{
		return find_in_tree(coordinate.substr(1, coordinate.length() - 1));
	}

	if (coordinate[0] == '/')
	{
		return 			root->get_by_coordinate(coordinate.substr(1,
coordinate.length() - 1));
	}

	std::string object_name;
	for (int i = 0; i < coordinate.length(); i++)
	{
		if (coordinate[i] != '/')
		{
			object_name += coordinate[i];
		}
		else if (get_child(object_name) == nullptr)
		{
			return nullptr;
		}
		else
		{
			return get_child(object_name)
				->get_by_coordinate(
					coordinate.substr(object_name.length()	+	1,
coordinate.length() - 1));
		}
	}

	return get_child(object_name);
}

void cl_base::disconnect_child(std::string child_name)
{
	for (int i = 0; i < children.size(); i++)
	{
		if (children[i]->name == child_name)
		{
			children.erase(children.begin() + i);
			break;
		}
	}
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H

#include <iostream>
```

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

class cl_base
{
private:

	int state;

	// Наименование объекта
	std::string name;

	// Указатель на головной объект для текущего объекта
	cl_base* parent;

	// Динамический массив указателей на объекты,
	// подчиненные к текущему объекту в дереве иерархии
	std::vector<cl_base*> children;

public:

	// Параметризированный конструктор
	cl_base(cl_base* parent, std::string name="");

	// Деструктор
	~cl_base();

	// Метод редактирования имени объекта
	bool set_name(std::string new_name);

	// Метод получения имени объекта
	std::string get_name();

	// Метод получения указателя на головной объект текущего объекта
	cl_base* get_parent();

	// Метод вывода наименований объектов в дереве иерархии слева
	// направо и сверху вниз
	void print_tree(std::string prefix);

	// Метод получения указателя на непосредственно подчиненный
	// объект по его имени
	cl_base* get_child(std::string child_name);

	// Метод поиска объекта по заданному имени в дереве иерархии объектов
	cl_base* find_in_tree(std::string wanted_name);

	// Метод вывода наименований объектов в дереве иерархии слева
	// направо и сверху вниз вместе с их готовностью
	void print_tree_with_state(std::string prefix);

	// Метод получения состояния объекта
	int get_state();

	// Метод получения состояния объекта
	void set_state(int state);
```

```
	// Метод получения корня дерева
	cl_base* get_root();

	// Метод поиска по всему дереву из произвольной точки
	cl_base* find_in_tree_from_root(std::string wanted_name);

	// Метод подсчёта объектов в дереве с заданным именем
	int count_in_tree(std::string wanted_name);

	// Метод изменения головного объекта
	bool set_parent(cl_base* new_parent);

	// Метод удаления подчинённого объекта
	std::string delete_child(std::string child_name);

	// Метод получения объекта по координате
	cl_base* get_by_coordinate(std::string coordinate);

	void disconnect_child(std::string child_name);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_application.h"

int main()
{
	cl_application ob_cl_application(nullptr);
	ob_cl_application.build_tree_objects();
	return ob_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 8.

Таблица 8 – 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_3
	object_3
	object_7
		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

rootela / object_1 2 / object_2 2 / object_3 3 /object_1 object_7 4	Object tree rootela object_1 object_7 object_2	Object tree rootela object_1 object_7 object_2

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
/object_2 object_4 4 /object_2 object_5 4 /object_2 object_3 4 /object_3 object_7 6 endtree END	object_4 object_5 object_3 object_3 object_7 Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_5 object_3 object_3 object_7	object_4 object_5 object_3 object_3 object_7 Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_5 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).