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 необходимы те же объекты, циклы и операторы, которые были использованы для решения работы КВ2.

Класс cl_base базовый класс (с изменениями)

Поля:

открытые методы:

добавлены:

~cl_base() - удаляет подчинные объекты

void remove_child(cl_base* child) - удаляет из списка детей переданный объект

cl_base* find_internal(string object_name, int& count) - поиск объекта и подсчитывает количество найденных в иерархии объектов

bool move(cl_base* new_parent) - изменяет головной объект текущего

void delete_child(string child_name) - удаляет ребёнка текущего объекта по имени

cl_base* get_by_path(string path) - возвращает объект по его координате

string get_abs_path() - возвращает абсолютный путь до объекта

bool is_subordinate(cl_base*) - проверяет является объект подчиненным текущего

методы с изменениями:

cl_base* find(string object_name) - поиск объекта от текущего по имени, если их несколько с таким именем возвращаем nullptr

Класс cl_application наследует класс cl_base (с изменениями)

Поля:

открытые методы с изменениями:

void build_tree_objects() - строит иерархию классов

void exec_app() - вывод иерархию классов с корневого объекта

Классы cl_2, cl_3, cl_4, cl_5, cl_6 унаследованы от cl_base

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм функции main

Функционал: главный метод программы.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта	2
2		вызов метода build_tree_objects	∅

3.2 Алгоритм метода exec_app класса cl_application

Функционал: вывод иерархию классов с корневого объекта.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2
2		вызов метода show_object_tree()	∅

3.3 Алгоритм метода build_tree_objects класса cl_application

Функционал: строение иерархии классов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		ввод имени корневого объекта	2
2		установка имени корневого объекта	3
3		ввод координаты головногообъекта	4
4	root_name == "endtree"		10
		ввод имени подчиненного объекта и номера класса	5
5		вызов метода получения головного объекта по его координате	6
6	головного объекта нет	вызов метода exec_app	7
			9
7		вывод "The head object root_name is not found"	8
8		завершение программы	∅
9	у головного объекта есть ребёнок с именем, равным имени подчиненного объекта	вывод root_name "Dubbing the names of subordinate objects"	3
	class_num == 2	создание нового объекта класса cl_2 с child_name и p	3
	class_num == 3	создание нового объекта класса cl_3 с child_name и p	3
	class_num == 4	создание нового объекта класса cl_4 с child_name и p	3
	class_num == 5	создание нового объекта класса cl_5 с child_name и	3

№	Предикат	Действия	№ перехода
		p	
	class_num == 6	создание нового объекта класса cl_6 с child_name и p	3
			3
10		вызов метода exec_app	11
11		переход на новую строку	12
12		установить текущий объект равным этому объекту	13
13		ввод команды	14
14	cmd == "END"		23
		ввод координаты	15
15	cmd == "DELETE"	поиск объекта среди детей текущего по имени	16
		поиск объекта по координате от текущего	16
16	cmd == "SET"		17
	cmd == "FIND"		19
	cmd == "MOVE"		20
	cmd == "DELETE"		21
			13
17	объект не найден	вывод координата объекта	13
		установка текущего объекта равным найденному объекту	18
18		вывод имени найденного объекта	13
19	объект найден	вывод координата объекта и имя найденного объекта	13
		вывод координата объекта	13

№	Предикат	Действия	№ перехода
20	объект не найден или не удалось изменить головной объект текущего	вывод об ошибке	13
		вывод имя найденного объекта	13
2	объект найден	вывод абсолютного пути к объекту	22
1			13
22		удалить подчиненный текущему объект по имени	13
23		вывод "Current object hierarchy tree"	24
24		вызов метода exec_app	∅

3.4 Алгоритм деструктора класса cl_base

Функционал: удаляет подчиненные объекты.

Параметры: нет.

Алгоритм деструктора представлен в таблице 4.

Таблица 4 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1	с принадлежит children	удаление с	1
			∅

3.5 Алгоритм метода remove_child класса cl_base

Функционал: удаляет из списка детей переданный объект.

Параметры: cl_base* child.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода remove_child класса cl_base

№	Предикат	Действия	№ перехода
1	it не равен конечному элементу children		2
			∅
2	объект *it == child	удалить it из children	∅
			1

3.6 Алгоритм метода find_internal класса cl_base

Функционал: поиск объекта и подсчитывает количество найденных в иерархии объектов.

Параметры: string object_name, int& count.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода find_internal класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация указателя на найденный объект	2
2	имя текущего объекта == object_name	count увечить на 1	3
			4
3		установить найденный объект равным текущему	4
4	с принадлежит children	вызов метода find_internal	5
			6
5	метод find_internal вернул указатель на существующий объект	установить найденный объект равным результату вызова метода find_internal	4

№	Предикат	Действия	№ перехода
			4
6		возврат найденного объекта	∅

3.7 Алгоритм метода move класса cl_base

Функционал: изменяет головной объект текущего.

Параметры: cl_base* new_parent.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода move класса cl_base

№	Предикат	Действия	№ перехода
1	это корневой объект или new_parent == nullptr или у new_parent уже есть ребенок с именем текущего объекта или new_parent является подчиненным текущего объекта	возврат false	∅
		удалить текущий объект из списка детей его старого родителя	2
2		добавить текущий объект в список детей нового родителя	3
3		p_parent = new_parent	4
4		возврат true	∅

3.8 Алгоритм метода delete_child класса cl_base

Функционал: удаляет ребёнка текущего объекта по имени.

Параметры: string child_name.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода delete_child класса cl_base

№	Предикат	Действия	№ перехода
1	it не равен конечному элементу children		2
			∅
2	имя *it == child_name	удалить *it	3
			1
3		удалить it из списка children	∅

3.9 Алгоритм метода get_by_path класса cl_base

Функционал: возвращает объект по его координате.

Параметры: string path.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_by_path класса cl_base

№	Предикат	Действия	№ перехода
1	path начинается с "//"	вызов метода find_on_whole_tree	∅
	path равно "."	возврат этого объекта	∅
	path начинается с "."	вызов метода find	∅
	path начинается с "/"	вызов метода get_by_path	∅
		получить ребёнка текущего объекта по части координаты	2
2	ребёнок не найден или это конец координаты	возврат ребёнка	∅

№	Предикат	Действия	№ перехода
		вызов метода get_by_path	∅

3.10 Алгоритм метода get_abs_path класса cl_base

Функционал: возвращает абсолютный путь до объекта.

Параметры: нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода get_abs_path класса cl_base

№	Предикат	Действия	№ перехода
1	это корневой объект дерева	возврат ""	∅
		возврат метода get_abs_path и имени текущего объекта	∅

3.11 Алгоритм метода is_subordinate класса cl_base

Функционал: проверяет является объект подчиненным текущего.

Параметры: cl_base* obj.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода is_subordinate класса cl_base

№	Предикат	Действия	№ перехода
1	child принадлежит children		2
		возврат false	∅
2	child == obj	возврат true	∅
	obj является подчиненным child	возврат true	∅

№	Предикат	Действия	№ перехода
			1

3.12 Алгоритм метода find класса cl_base

Функционал: поиск объекта от текущего по имени.

Параметры: string object_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода find класса cl_base

№	Предикат	Действия	№ перехода
1		вызов метода find_internal с передачей object_name и ссылки на переменную-счетчик	2
2	count > 1	возврат nullptr	∅
		возврат результата выполнения метода find_internal	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

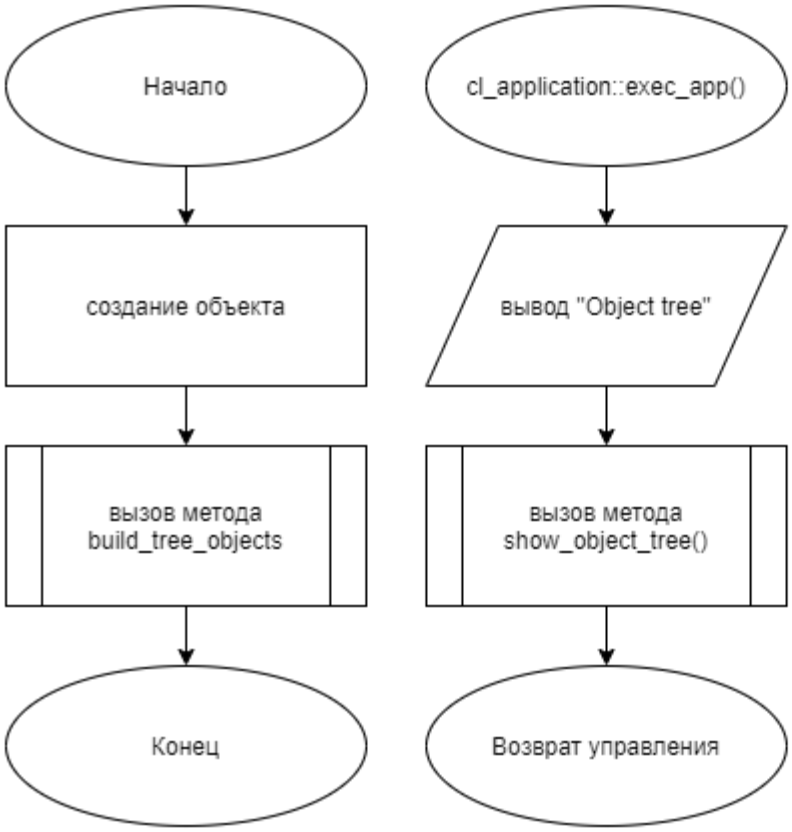

Рисунок 1 – Блок-схема алгоритма

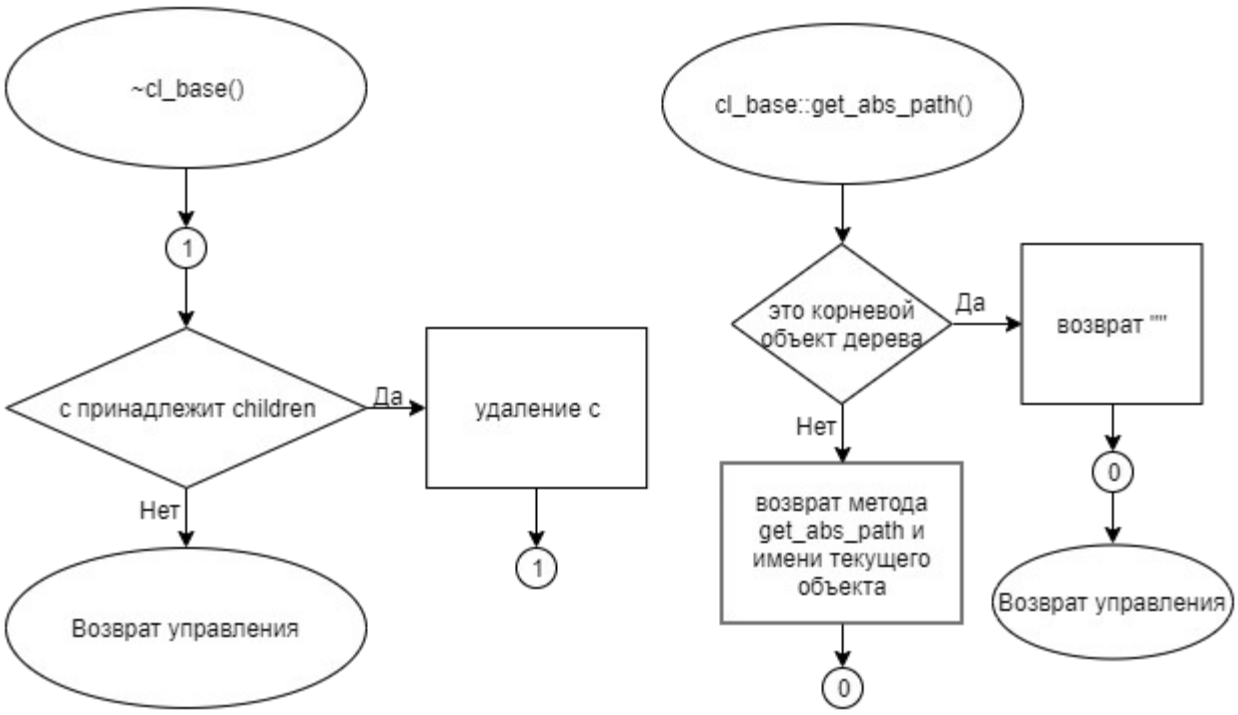

Рисунок 2 – Блок-схема алгоритма

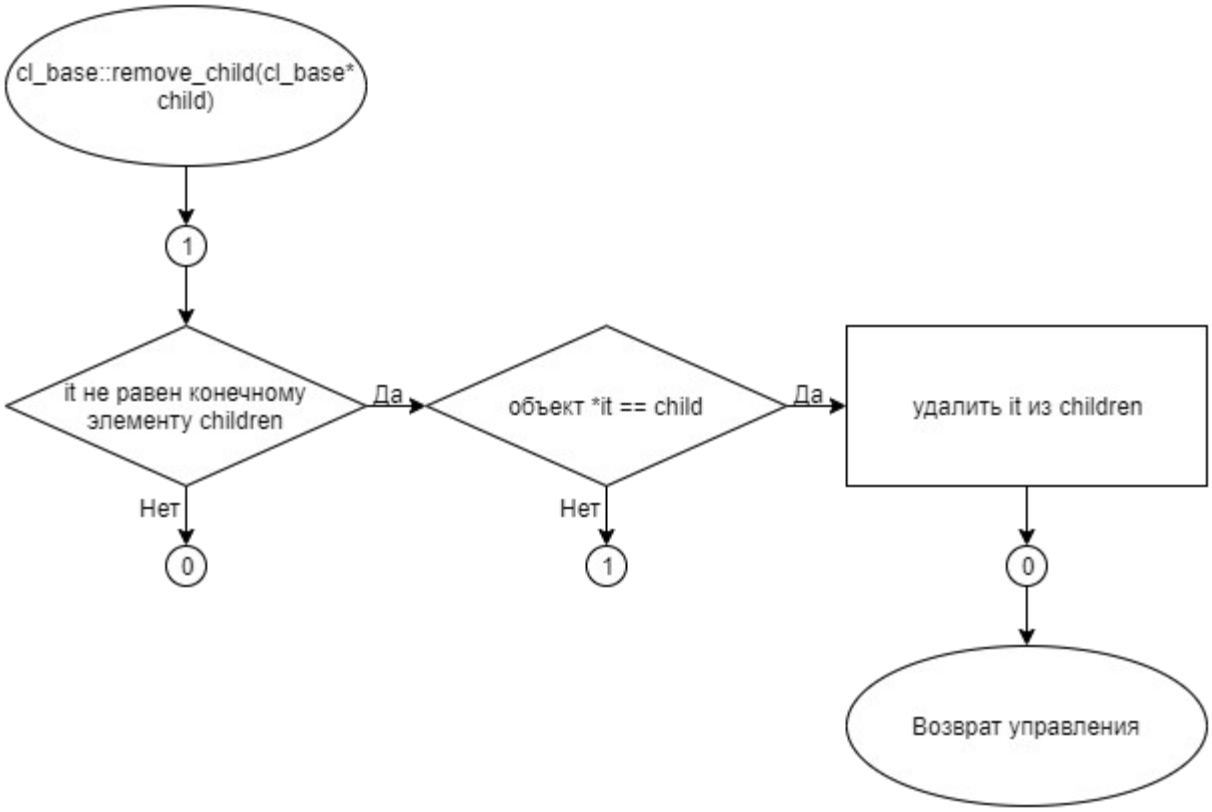

Рисунок 3 – Блок-схема алгоритма

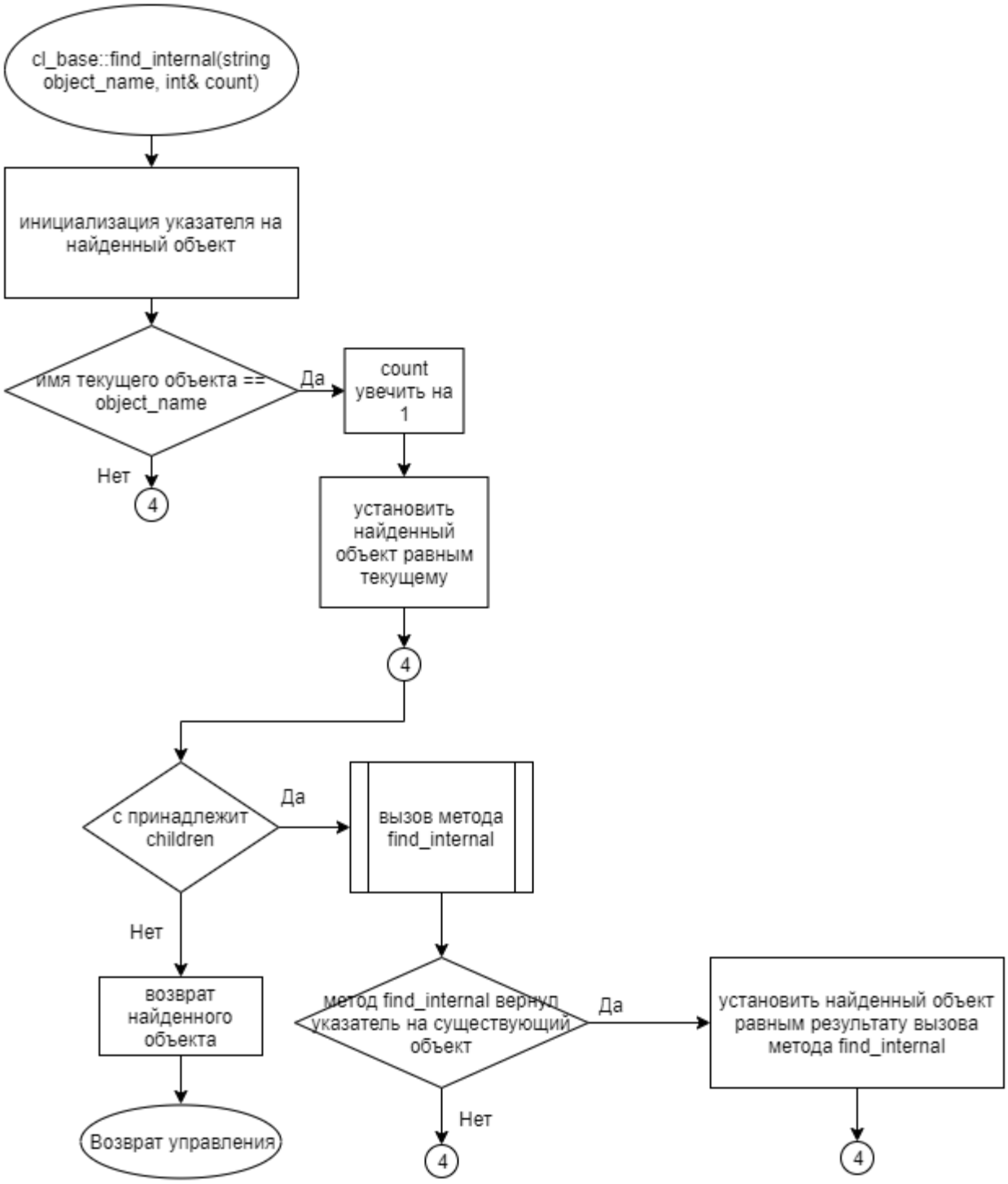

Рисунок 4 – Блок-схема алгоритма

Рисунок 5 – Блок-схема алгоритма

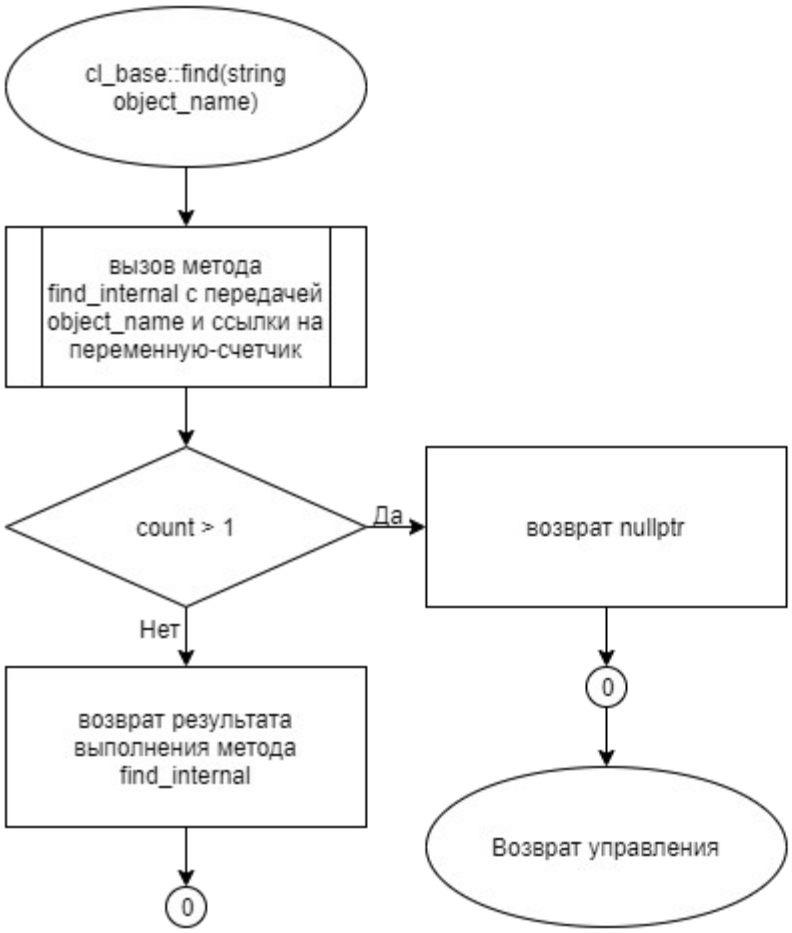

Рисунок 6 – Блок-схема алгоритма

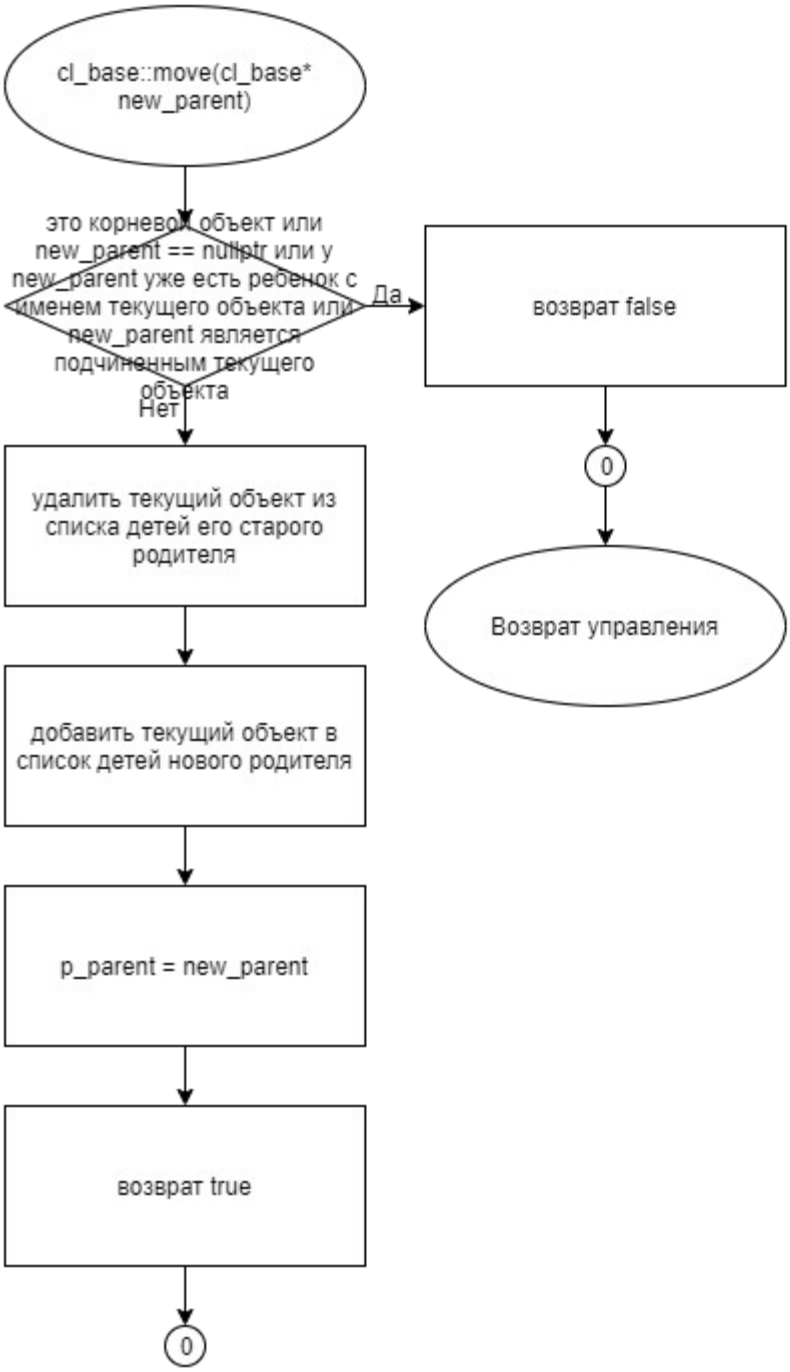

Рисунок 7 – Блок-схема алгоритма

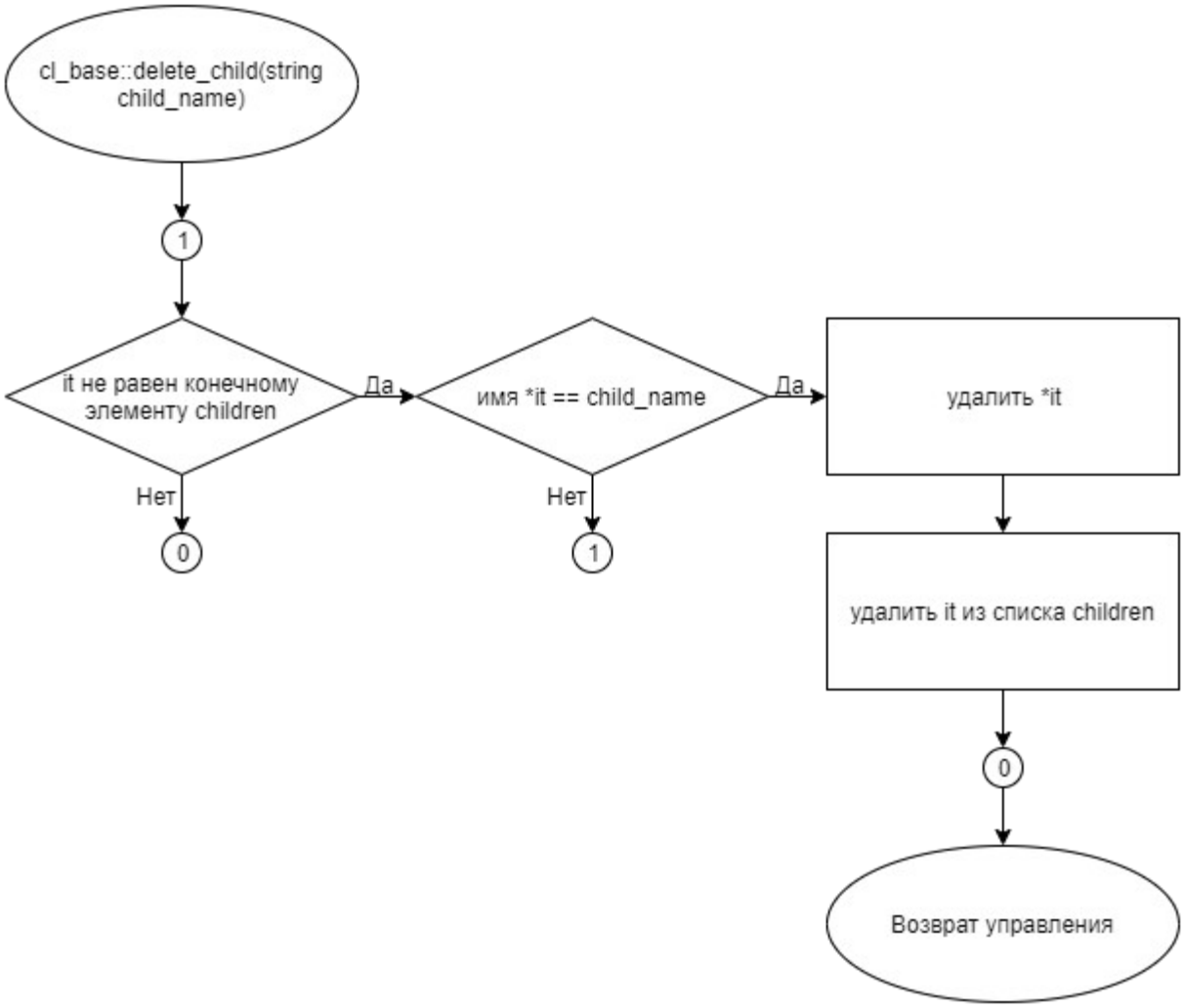

Рисунок 8 – Блок-схема алгоритма

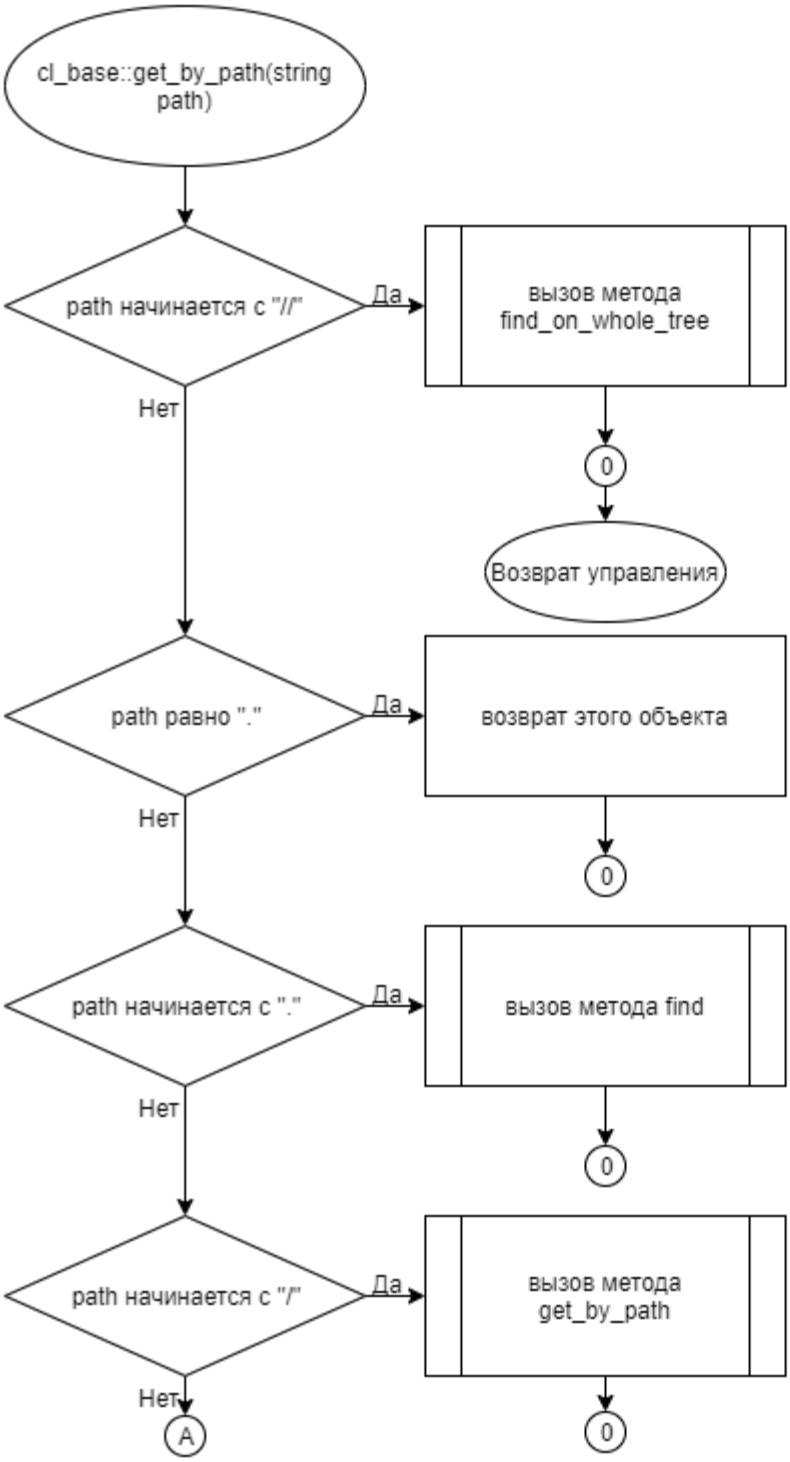

Рисунок 9 – Блок-схема алгоритма

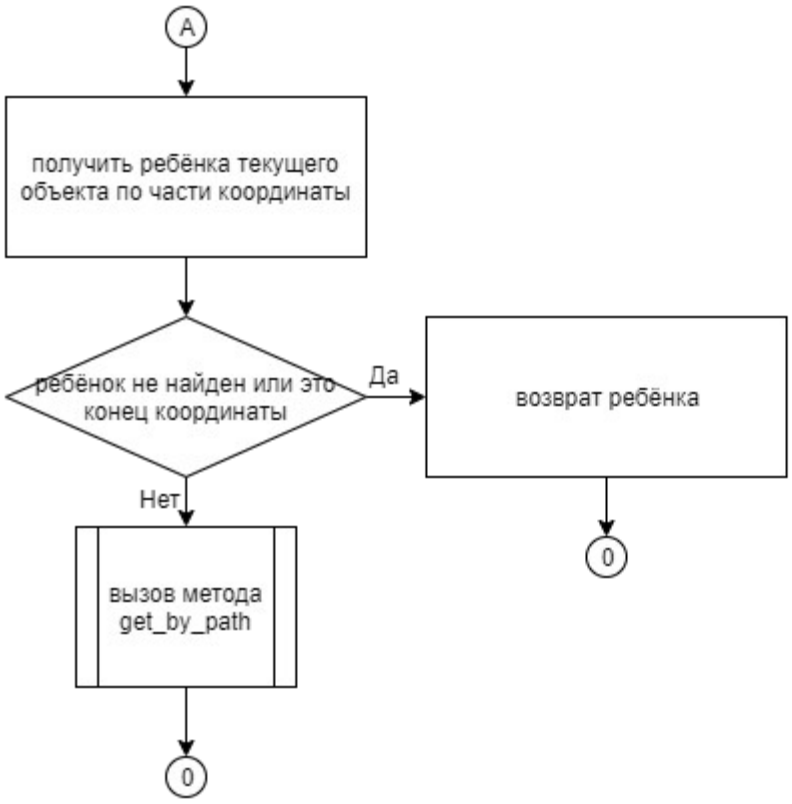

Рисунок 10 – Блок-схема алгоритма

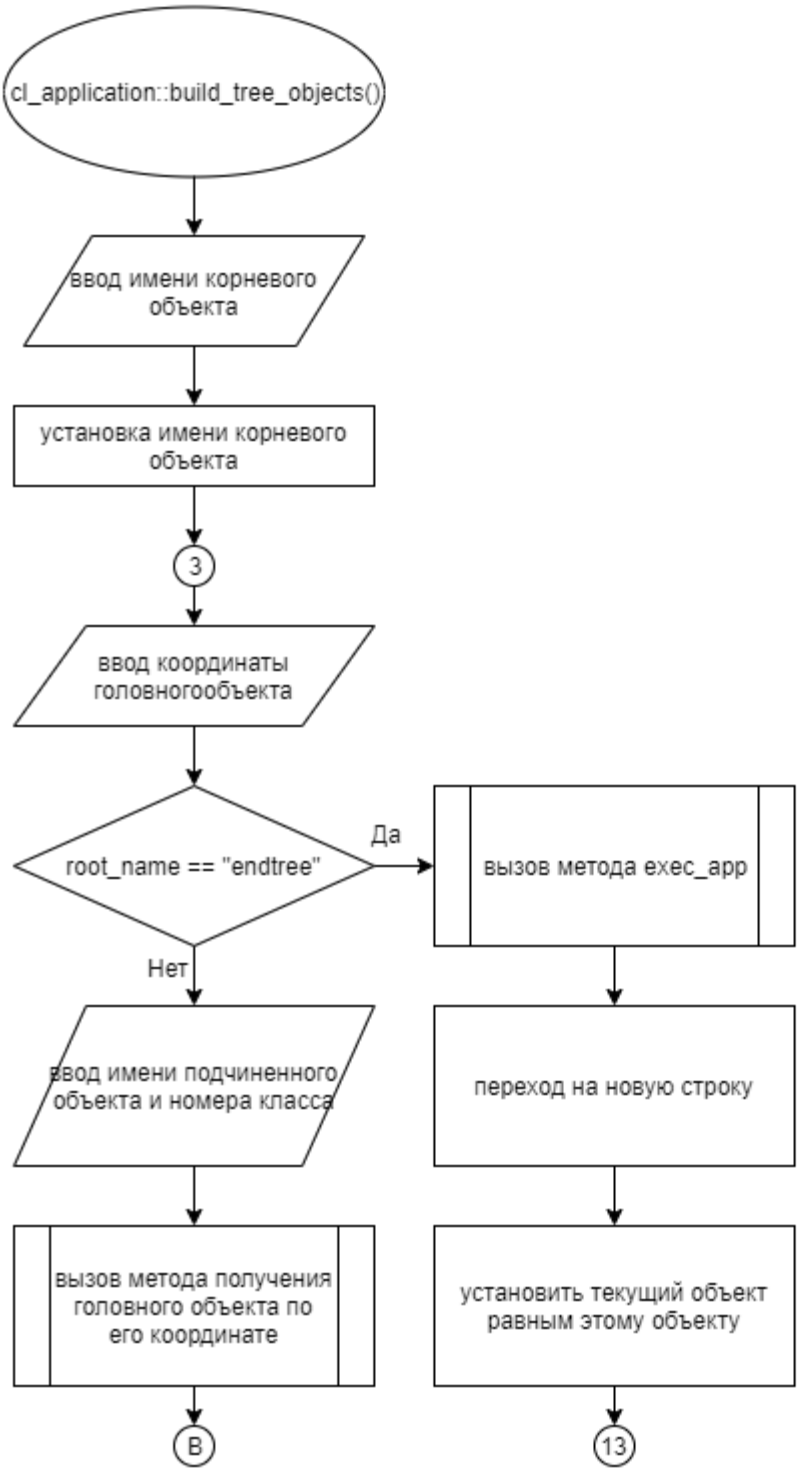

Рисунок 11 – Блок-схема алгоритма

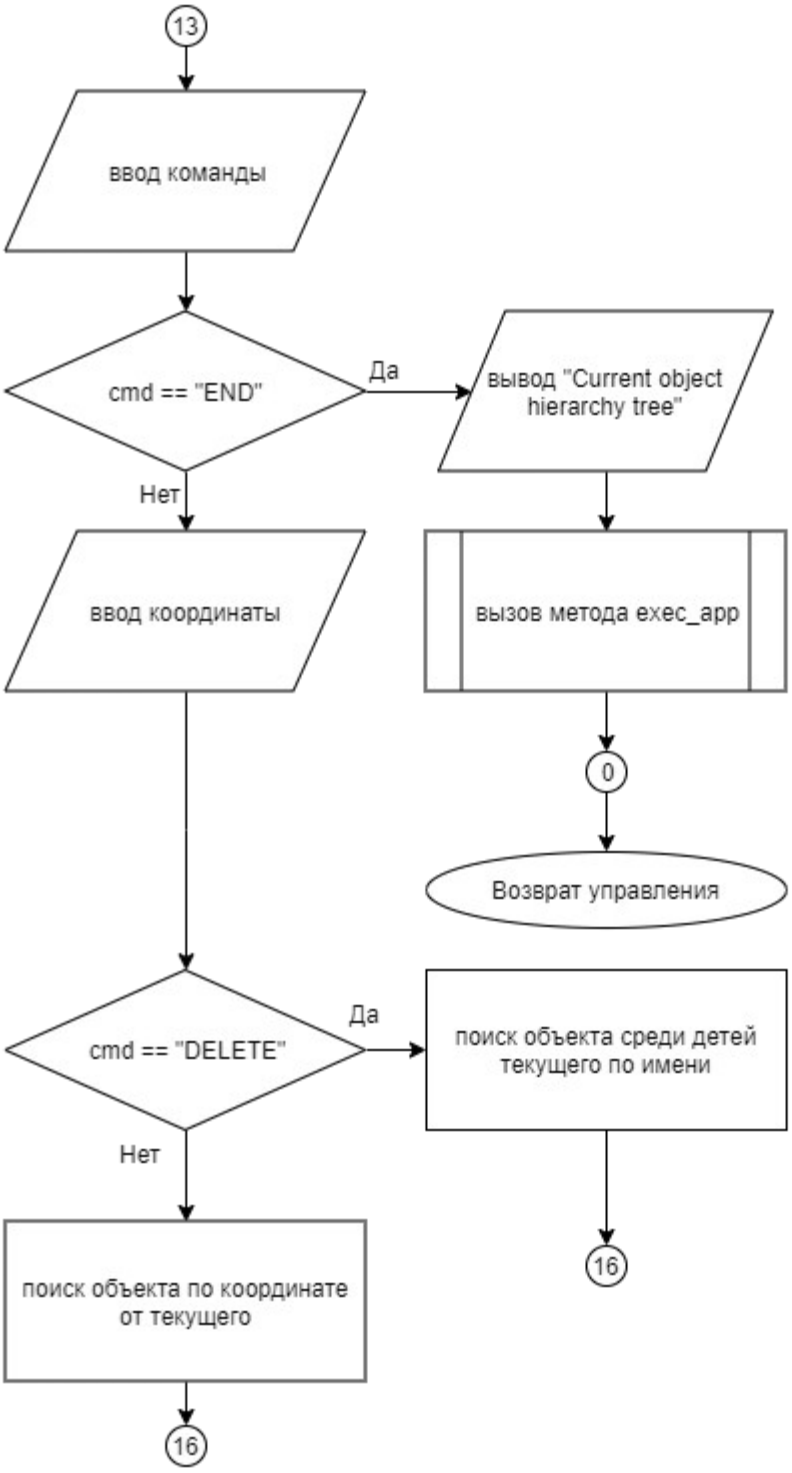

Рисунок 12 – Блок-схема алгоритма

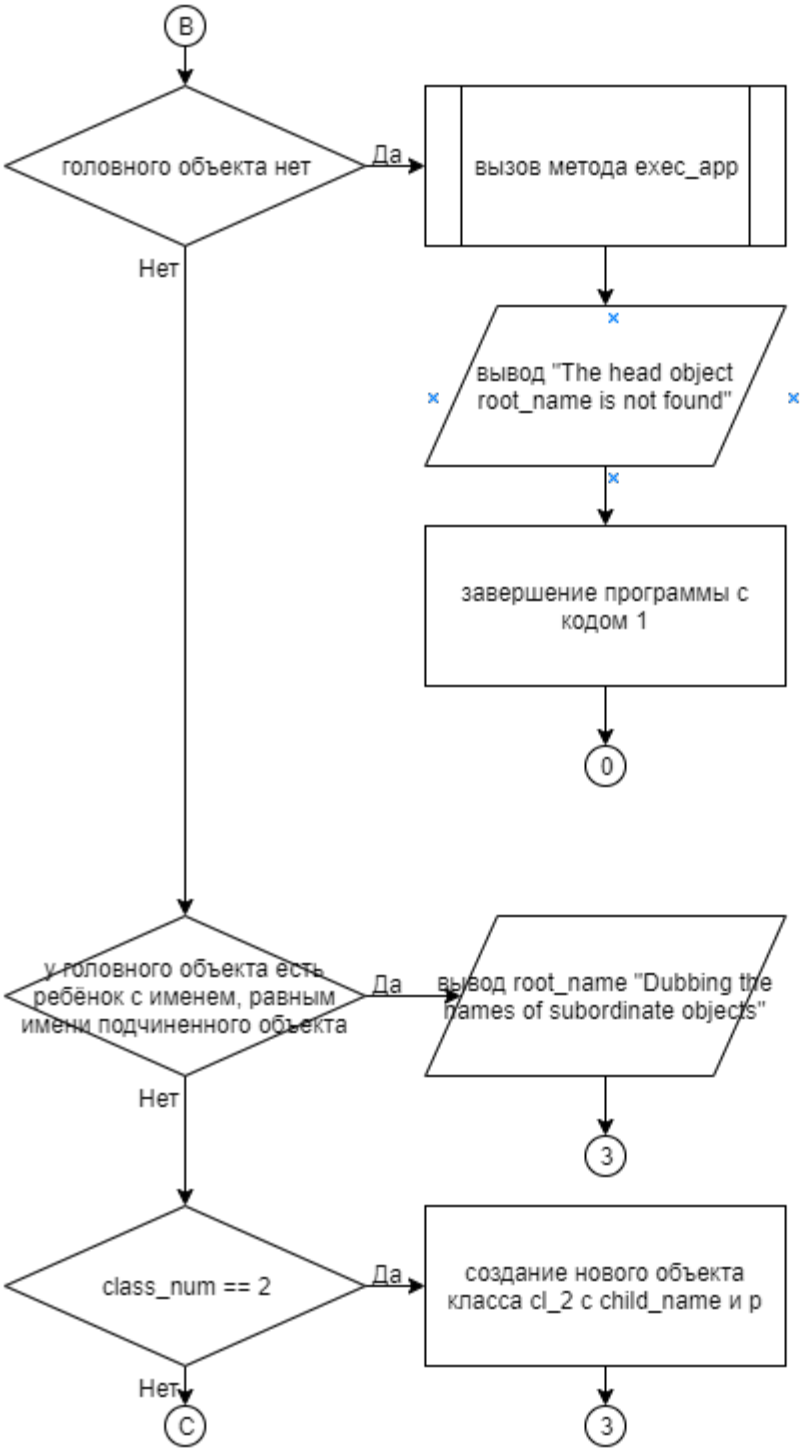

Рисунок 13 – Блок-схема алгоритма

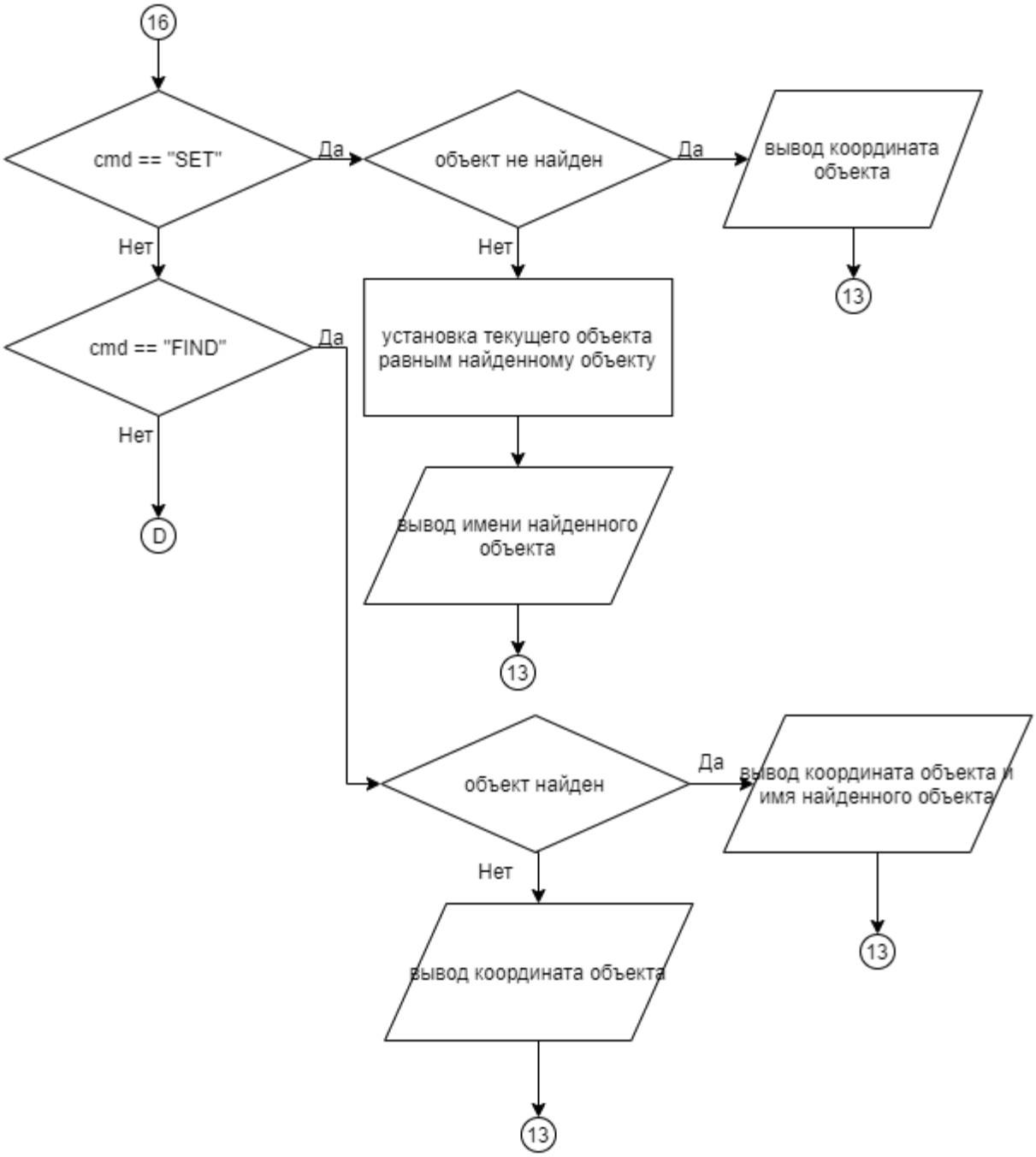

Рисунок 14 – Блок-схема алгоритма

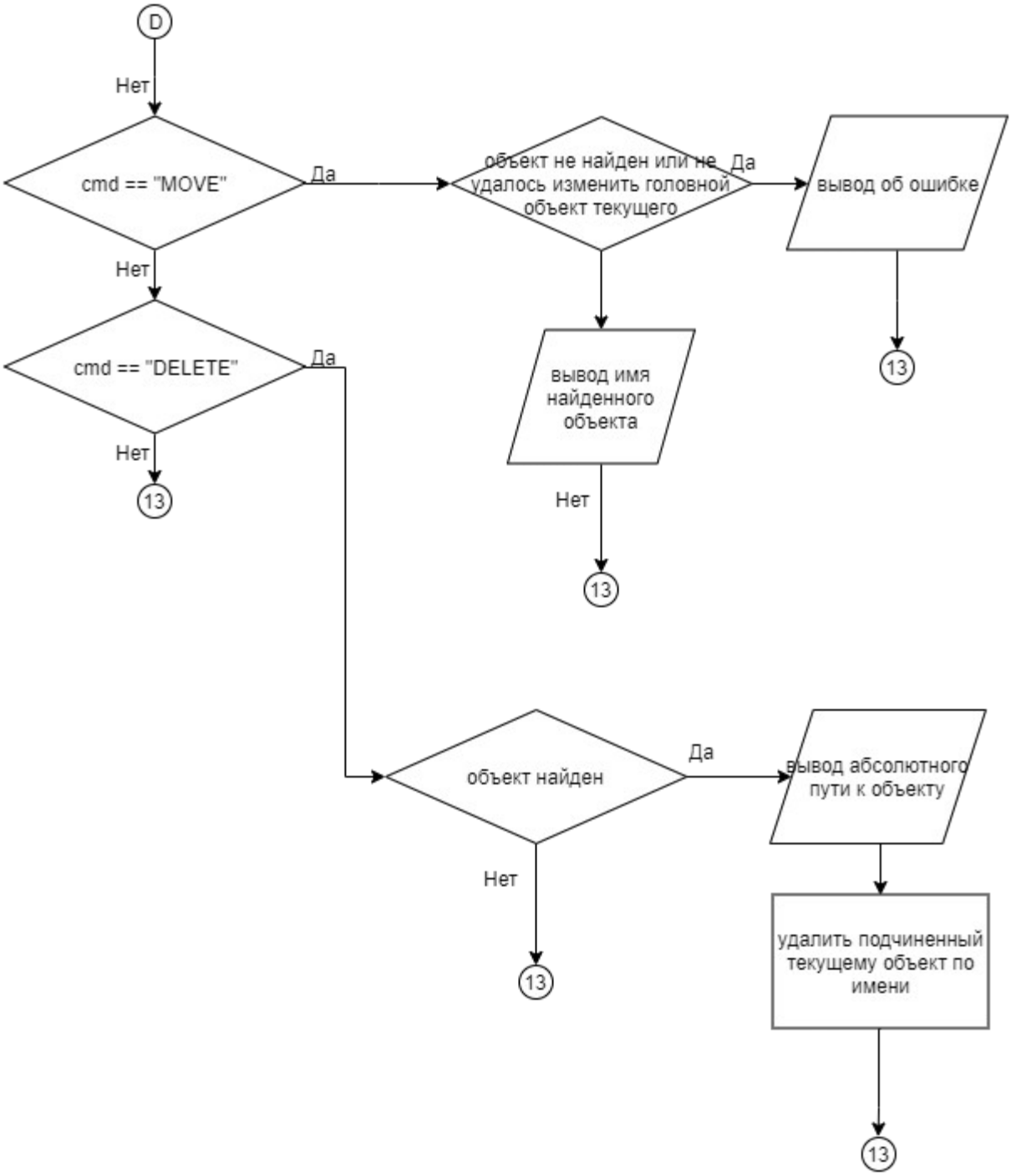

Рисунок 15 – Блок-схема алгоритма

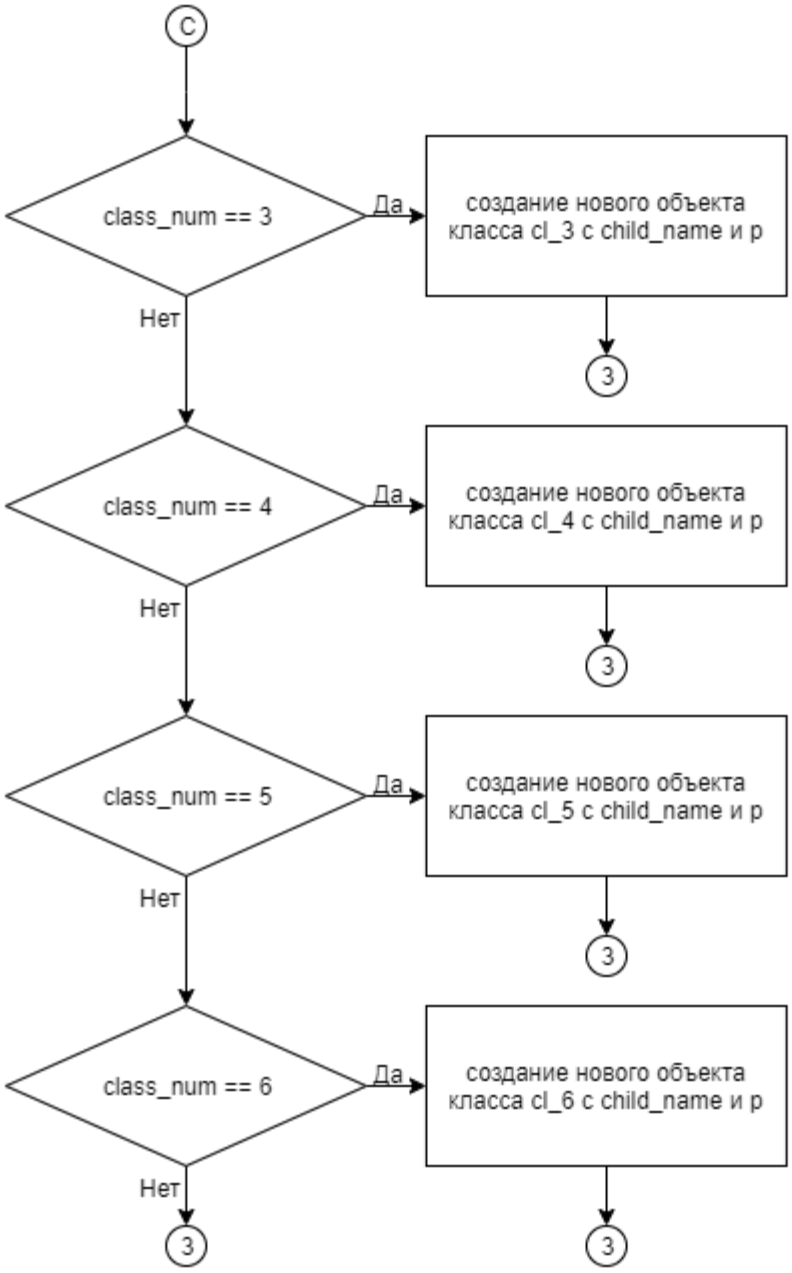

Рисунок 16 – Блок-схема алгоритма

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

using namespace std;

cl_2::cl_2(cl_base* parent, 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 <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_2 : public cl_base
{
public:
	cl_2(cl_base* parent=nullptr, string name="");
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_3::cl_3(cl_base* parent, 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 <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_3 : public cl_base
{
public:
	cl_3(cl_base* parent=nullptr, string name="");
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
cl_4::cl_4(cl_base* parent, 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 <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_4 : public cl_base
{
public:
	cl_4(cl_base* parent=nullptr, string name="");
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_5::cl_5(cl_base* parent, 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 <iostream>
```

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

using namespace std;

class cl_5 : public cl_base
{
public:
	cl_5(cl_base* parent=nullptr, string name="");
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_6::cl_6(cl_base* parent, 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 <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_6 : public cl_base
{
public:
	cl_6(cl_base* parent=nullptr, string name="");
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.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>
#include <vector>

using namespace std;

cl_application::cl_application() : cl_base(nullptr, "") {};

void cl_application::build_tree_objects()
{
	string root_name, child_name, cmd, path;
	int class_num = 0;
	cin >> root_name;

	set_object_name(root_name);

	while (true)
	{
		cin >> root_name;
		if (root_name == "endtree")
		{
			break;
		}

		cin >> child_name >> class_num;

		auto p = get_by_path(root_name);

		if (!p)
		{
			exec_app();
			cout << endl << "The head object " << root_name << " is not
found" << endl;
			exit(1); // exit?
		}

		if (p->get_child(child_name))
		{
			cout << root_name << "        Dubbing the names of subordinate
objects" << endl;
			continue;
		}

		switch (class_num)
		{
```

```
                  case 2:
                        new cl_2(p, child_name);
                        break;
                  case 3:
                        new cl_3(p, child_name);
                        break;
                  case 4:
                        new cl_4(p, child_name);
                        break;
                  case 5:
                        new cl_5(p, child_name);
                        break;
                  case 6:
                        new cl_6(p, child_name);
                        break;
            }
      }

      exec_app();
      cout << endl;

      cl_base* current = this;
      while (true)
      {
            cin >> cmd;
            if (cmd == "END")
            {
                  break;
            }
            cin >> path;
            auto found = cmd == "DELETE" ? current->get_child(path) : current-
>get_by_path(path); // ?
            if (cmd == "SET")
            {
                  if (!found)
                  {
                        cout << "The object was not found at the specified
coordinate: " << path << endl;
                        continue;
                  }
                  current = found;
                  cout << "Object is set: " << found->get_object_name() << endl;
            }
            else if (cmd == "FIND")
            {
                  cout << path << "      " << (found ? "Object name: " + found-
>get_object_name() : "Object is not found") << endl; // ?
            }
            else if (cmd == "MOVE")
            {
                  if (!found || !current->move(found))
                  {
                        cout << path << "      " << (!found ? "Head object is not
found" : found->get_child(current->get_object_name()) ? "Dubbing the names of
subordinate objects" : "Redefining the head object failed") << endl;
                  }
                  else
```

```
                    {
                          cout << "New head object: " << found->get_object_name() <<
endl;
                    }
              }
              else if (cmd == "DELETE")
              {
                    if (!found)
                    {
                          continue;
                    }
                    cout << "The object " << found->get_abs_path() << " has been
deleted" << endl;
                    current->delete_child(path);
              }
      }
      cout << "Current object hierarchy tree" << endl;
      show_object_tree();
}

void cl_application::exec_app()
{
      cout << "Object tree" << endl;
      show_object_tree(false);
}
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_application : public cl_base
{
public:
      cl_application();
      void build_tree_objects();
      void exec_app();
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

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

using namespace std;

cl_base::cl_base(cl_base*     parent,     string     name):     p_parent(parent),
object_name(name)
{
      if (parent != nullptr)
      {
            parent->children.push_back(this);
      }
}

cl_base::~cl_base()
{
      for (auto c: children)
      {
            delete c;
      }
}

bool cl_base::set_object_name(string object_name)
{
      if (!p_parent)
      {
            this->object_name = object_name;
            return true;
      }

      it_child = children.begin();

      while (it_child != children.end())
      {
            if (((*it_child)->get_object_name() == object_name) && ((*it_child) !=
this))
            {
                  return false;
            }
            it_child++;
      }

      this->object_name = object_name;
      return true;
}

string cl_base::get_object_name()
{
      return object_name;
```

```
}

cl_base* cl_base::get_parent()
{
	return p_parent;
}

void cl_base::show_object_tree(bool show_state)
{
	show_object_next(0, show_state);
}

void cl_base::show_object_next(int i_level, bool show_state)
{
	string s_space = "";
	if (i_level > 0)
	{
		s_space.append(4 * i_level, ' ');
	}
	cout << s_space << get_object_name();

	if (show_state)
	{
		if (get_state())
		{
			cout << " is ready";
		}
		else
		{
			cout << " is not ready";
		}
	}

	for (auto c : children)
	{
		cout << endl;
		c->show_object_next(i_level + 1, show_state);
	}

}

cl_base* cl_base::get_child(string child_name)
{
	for (auto child : children)
	{
		if (child->get_object_name() == child_name)
		{
			return child;
		}
	}

	return nullptr;
}

bool cl_base::get_state()
{
	return state;
```

```
}

void cl_base::set_state(bool state)
{
	if (get_parent() && !get_parent()->get_state())
	{
		this->state = false;
	}
	else
	{
		this->state = state;
	}
	if (!state)
	{
		for (auto c : children)
		{
			c->set_state(state);
		}
	}
}

cl_base* cl_base::find(string object_name)
{
	int count = 0;
	auto found = find_internal(object_name, count);
	if (count > 1)
	{
		return nullptr;
	}
	return found;
}

cl_base* cl_base::find_internal(string object_name, int& count)
{
	cl_base* found = nullptr;
	if (object_name == get_object_name())
	{
		count++;
		found = this;
	}

	for (auto c : children)
	{
		auto obj = c->find_internal(object_name, count);
		if (obj)
		{
			found = obj;
		}
	}

	return found;
}

cl_base* cl_base::find_on_whole_tree(string object_name)
{
	auto p = this;
	while (p->get_parent())
```

```
	{
		p = p->get_parent();
	}
	return p->find(object_name);
}

bool cl_base::is_subordinate(cl_base* obj)
{
	for (auto child: children)
	{
		if (child == obj)
		{
			return true;
		}

		if (child->is_subordinate(obj))
		{
			return true;
		}
	}
	return false;
}

bool cl_base::move(cl_base* new_parent)
{
	if (!get_parent() || !new_parent || new_parent->get_child(get_object_name())
|| is_subordinate(new_parent))
	{
		return false;
	}
	get_parent()->remove_child(this);
	new_parent->children.push_back(this);
	p_parent = new_parent;
	return true;
}

void cl_base::delete_child(string child_name)
{
	for (auto it = children.begin(); it != children.end(); it++)
	{
		if ((*it)->get_object_name() == child_name)
		{
			delete *it;
			children.erase(it);
			break;
		}
	}
}

cl_base* cl_base::get_by_path(string path)
{
	if (path.substr(0, 2) == "//")
	{
		return find_on_whole_tree(path.substr(2));
	}
	else if (path == ".")
	{
```

```
            return this;
      }
      else if (path[0] == '.')
      {
            return find(path.substr(1));
      }
      else if (path[0] == '/')
      {
            auto new_path = path.substr(1);
            auto cur = this;
            while (cur->get_parent())
            {
                  cur = cur->get_parent();
            }
            if (path == "/")
            {
                  return cur;
            }
            return cur->get_by_path(new_path);
      }
      auto pos = path.find('/');
      auto child = get_child(path.substr(0, pos));
      if (!child || pos == string::npos) // -1
      {
            return child;
      }
      return child->get_by_path(path.substr(pos + 1));
}

void cl_base::remove_child(cl_base* child)
{
      for (auto it = children.begin(); it != children.end(); it++)
      {
            if ((*it) == child)
            {
                  children.erase(it);
                  break;
            }
      }
}

string cl_base::get_abs_path()
{
      if (!get_parent())
      {
            return "";
      }
      return get_parent()->get_abs_path() + "/" + get_object_name();
}
```

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>

using namespace std;

class cl_base
{
private:
      string object_name;
      cl_base* p_parent;
      bool state;
public:
      vector <cl_base*> children;
      vector <cl_base*> :: iterator it_child;
      cl_base(cl_base* parent=nullptr, string name="");
      ~cl_base();
      bool set_object_name(string object_name);
      string get_object_name();
      cl_base* get_parent();
      void show_object_tree(bool show_state=false);
      void show_object_next(int i_level, bool show_state);
      cl_base* get_child(string child_name);
      bool get_state();
      void set_state(bool state);
      cl_base* find(string object_name);
      cl_base* find_on_whole_tree(string object_name);
      void remove_child(cl_base* child);
      cl_base* find_internal(string object_name, int& count);
      bool move(cl_base* new_parent);
      void delete_child(string child_name);
      cl_base* get_by_path(string path);
      string get_abs_path();
      bool is_subordinate(cl_base* obj);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
```

```
#include "cl_application.h"

int main()
{
      cl_application ob_cl_application;
      ob_cl_application.build_tree_objects();
      return(0);
}
```

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_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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

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

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

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

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