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:

- объект ob_cl_application класса cl_application предназначен для запуска программы;
- функция main для основной алгоритм программы;
- cin/cout - объекты ввода/вывода;
- if - условный оператор;
- for - цикл со счетчиком;
- while - цикл с условием.

Класс cl_base:

- функционал:
    - метод set_head_object — метод переопределения головного объекта для текущего в дереве иерархии;
    - метод delete_sub_by_name — метод удаления подчиненного объекта по наименованию;
    - метод get_object_by_path — метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты).

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_head_object класса cl_base

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

Параметры: cl_base* new_p_head_object - новое имя головного объекта.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода set_head_object класса cl_base

№	Предикат	Действия	№ перехода
1	головной объект совпадает с новым головным	возвратить true	∅
			2
2	объект является корнем или new_p_head_object нулевой указатель	возвратить false	∅
			3
3	у нового головного объекта уже есть подчиненный объект с именем текущего объекта	возвратить false	∅
			4
4		объявление стека st указателей на объект класса cl_base	5

№	Предикат	Действия	№ перехода
5		добавление в стек текущего объекта	6
6	стек содежит элементы		7
			13
7		инициализация указателя current на объект класса cl_base значением верхнего элемента стека st	8
8		удаление верхнего эдемента стека st	9
9	current равен new_p_head_object	возвратить ложь	∅
			10
10		инициализация целочисленной переменной i значением 0	11
11	i < размера вектора p_sub_objects объекта current	добавление в стек элемент p_sub_objects[i] объекта current	12
			6
12		увеличение значения i на единицу	13
13		инициализация ссылки на вектор v, содержащего указатели на объекты класса cl_base значением свойства p_sub_objects головного объекта	14
14		инициализация целочисленной переменной i знасением 0	15
15	i < размера вектора v		16
		возвратить false	∅
16	имя i-ого объекта равно имени текущего объкекта	удаление i-ого элемента вектора v	18
			17
17		увеличение значения i на единицу	15
18		добавить текущий объект в вектор подчиненных объектов нового родительского объекта	19
19		возвратить true	∅

3.2 Алгоритм метода delete_sub_by_name класса cl_base

Функционал: метод удаления подчиненного объекта по наименованию.

Параметры: string sub_name - переменная строкого типа, содержит имя подчиненного объекта.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода delete_sub_by_name класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация ссылки на вектор v, содержащий указатели на объекты класса cl_base значением свойста p_sub_objects	2
2		инициализация целочисленной переменной i значением 0	3
3	i < размера вектора v		4
			∅
4	имя i-ого объекта равно имени текущего объекта	вызов деструктора для объекта по i-ому указателю вектора v	6
			5
5		увеличиваем значение i на единицу	6
6		удаление i-ого элемента вектора v	∅

3.3 Алгоритм метода get_object_by_path класса cl_base

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

Параметры: string path - переменная строкого типа, содержит указатель на объект.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода get_object_by_path класса cl_base

№	Предикат	Действия	№ перехода
1	path - пустая строка	возврат нулевого указателя	∅
			2
2	path равен "."	вернуть текущий объект	∅
			3
3	первый символ path равен '.'	вернуть результат вызова метода find_object_from_current с параметром path, начиная со второго символа	∅
			4
4	первые два символа path равны "//"	вернуть результат вызова метода find_object_from_root с параметром path, начиная с третьего символа	∅
			5
5	первый символ path не равен '/'	инициализация знаковой целочисленной переменной slash позицией первого символа '/' в строке path	6
			9
6		инициализация указателя sub_ptr на объект класса cl_base результатом вызова метода get_sub_object с параметром path, до первого символа '/'	7
7	sub_ptr нулевой или в строке path нет символа '/'	возврат sub_ptr	∅
		вернуть результат вызова метода get_sub_by_path объекта sub_ptr с параметром path, начиная аосле символа '/'	8
8		вернуть результат вызова метода get_sub_by_path объекта sub_ptr с параметром path, начиная аосле символа '/'	∅

№	Предикат	Действия	№ перехода
9		инициализация указателя root адресом текущего объекта	10
10	у объекта по указателю root есть родитель	присваивание root значение головного объекта по указателю root	10
			11
11	path равен "/"	возврат root	∅
			12
12		вернуть результат вызова метода get_sub_by_path объекта по указателю root с параметром path начиная со второго символа	∅

3.4 Алгоритм метода build_tree_objects класса cl_application

Функционал: строит дерево иерархии объектов.

Параметры: отсутсвуют.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		объявление строковых переменных path и sub_name	2
2		объявление целочисленной перемнной class_number	3
3		ввод sub_name	4
4		вызов метода set_name с параметром sub_name	5
5		объявление указателя parent	6
6		инициализация указателя last_created указателем текущего объекта	7

№	Предикат	Действия	№ перехода
7		ввод path	8
8	path не равно "endtree"	ввод sub_name и class_number	9
			∅
9		присваивание parent результат вызова метода get_object_by_path с параметром path объекта по указателю last_created	10
10	parent ненулевой указатель и у объекта по этому указателю нет подчиненных объектов с именем sub_name		11
			12
11	class_number равен 1	создание объекта класса cl_application с помощью оператора new и вызова конструктора с параметрами parent и sub_name и присваивание указателю last_created адрес этого объекта	12
			8
12	class_number равен 2	создание объекта класса cl_2 с помощью оператора new и вызова конструктора с параметрами parent и sub_name и присваивание указателю last_created адрес этого объекта	13
			8
13	class_number равен 3	создание объекта класса cl_3 с помощью оператора new и вызова конструктора с параметрами parent и sub_name и присваивание указателю last_created адрес этого объекта	14
			8
14	class_number равен 4	создание объекта класса cl_4 с помощью оператора new и вызова конструктора с параметрами parent и sub_name и присваивание	15

№	Предикат	Действия	№ перехода
		указателю last_created адрес этого объекта	
			8
15	class_number равен 5	создание объекта класса cl_5 с помощью оператора new и вызова конструктора с параметрами parent и sub_name и присваивание указателю last_created адрес этого объекта	16
			8
16	class_number равен 6	создание объекта класса cl_6 с помощью оператора new и вызова конструктора с параметрами parent и sub_name и присваивание указателю last_created адрес этого объекта	17
			8
17		ввод path	8

3.5 Алгоритм метода exec_app класса cl_application

Функционал: запуск системы.

Параметры: отсутсвуют.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		объявление строковых переменных command и input	2
2		инициализация указателя current_obj на объект класса cl_base адресом текущего объекта	3
3		объявление указателя extra_obj на объект класса cl_base	4

№	Предикат	Действия	№ перехода
4		объявление стека строк st	5
5		вывод "Object tree"	6
6		вызов метода print_branch	7
7		ввод значения переменной command	8
8	command не равно "END"	ввод input	9
			24
9	command равно "SET"		10
	command равно "FIND"		13
	command равно "MOVE"		15
	command равно "DELETE"		17
			7
10		присваивание extra_obj результат вызова метода get_object_by_path объекта по указателю current_obj с параметром path	11
11	extra_obj ненулевой указатель	current_obj = extra_obj	12
		вывод "The object was not found at the specifiedcoordinate: {input}" с новой строки	7
12		вывод "Object is set: {имя объекта по указателю current_obj}" с новой строки	7
13		присваивание extra_obj результат вызова метода get_object_by_path объекта по указателю current_obj с параметром path	14
14	extra_obj ненулевой указаткль	вывод "{input} Object name: {имя объекта по указателю extra_obj}" с новой строки	7
		вывод "{input} Objects is not found" с новой строки	7
15		присваивание extra_obj результат вызова метода get_object_by_path объекта по указателю current_obj с параметром path	16

№	Предикат	Действия	№ перехода
16	результат вызова метода set_parent объекта по указателю current_obj с параметром extra_obj - истина	вывод "New head object: {имя объекта по указателю extra_obj}" с новой строки	7
	extra_obj нулевой указатель	вывод "{input} Head object is not found" с новой строки	7
	у объекта по указателю extra_obj есть подчиненный с именем объекта по указателю current_obj	вывод "{input} Dubbing the names of subordinate objects" с новой строки	7
		вывод "{input} Redefining the head object failed" с новой строки	7
17		присваивание extra_obj результат вызова метода get_sub_objects объекта по указателю current_obj с параметром input	18
18	extra_obj ненулевой указатель		19
			7
19	у объекта по указателю extra_obj есть головйной объект	добавление имени объекта по указателю extra_obj на вершину стека st	20
		вызов метода delete_sub_by_name у объекта по указателю current_obj с параметром input	21
20		присваивание extra_obj адрес головного объекта extra_obj	19
21		вывод "The object " с новой строки	22
22	стек st содержит элементы	вывод "/{вершина стека}"	23
		вывод " has been deleted"	7

№	Предикат	Действия	№ перехода
23		извлечение вершины стека	22
24		вывод "Current object hierarchy tree" с новой строки	25
25		вызов метода print_branch	26
26		возвратить 0	∅

3.6 Алгоритм функции main

Функционал: основная программа.

Параметры: отсутсвуют.

Возвращаемое значение: int - код возврата.

Алгоритм функции представлен в таблице 6.

Таблица 6 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_cl_application класса cl_application с использованием параметрического конструктора и передачей в него в качестве параметра нулевого указателя	2
2		вызов метода build_tree_objects объекта ob_cl_application	3
3		возвращение результата работы метода exec_app для объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-12.

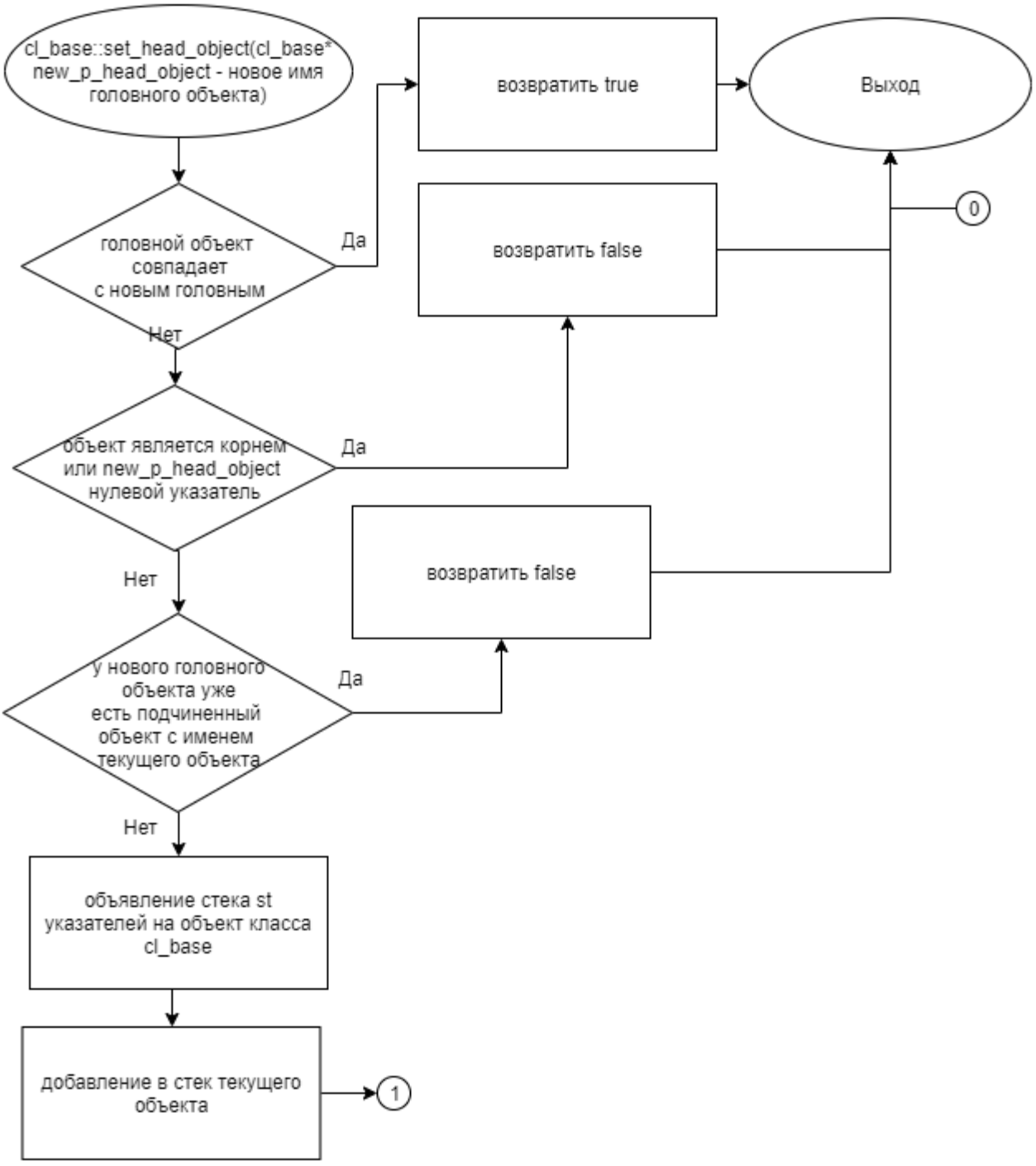

Рисунок 1 – Блок-схема алгоритма

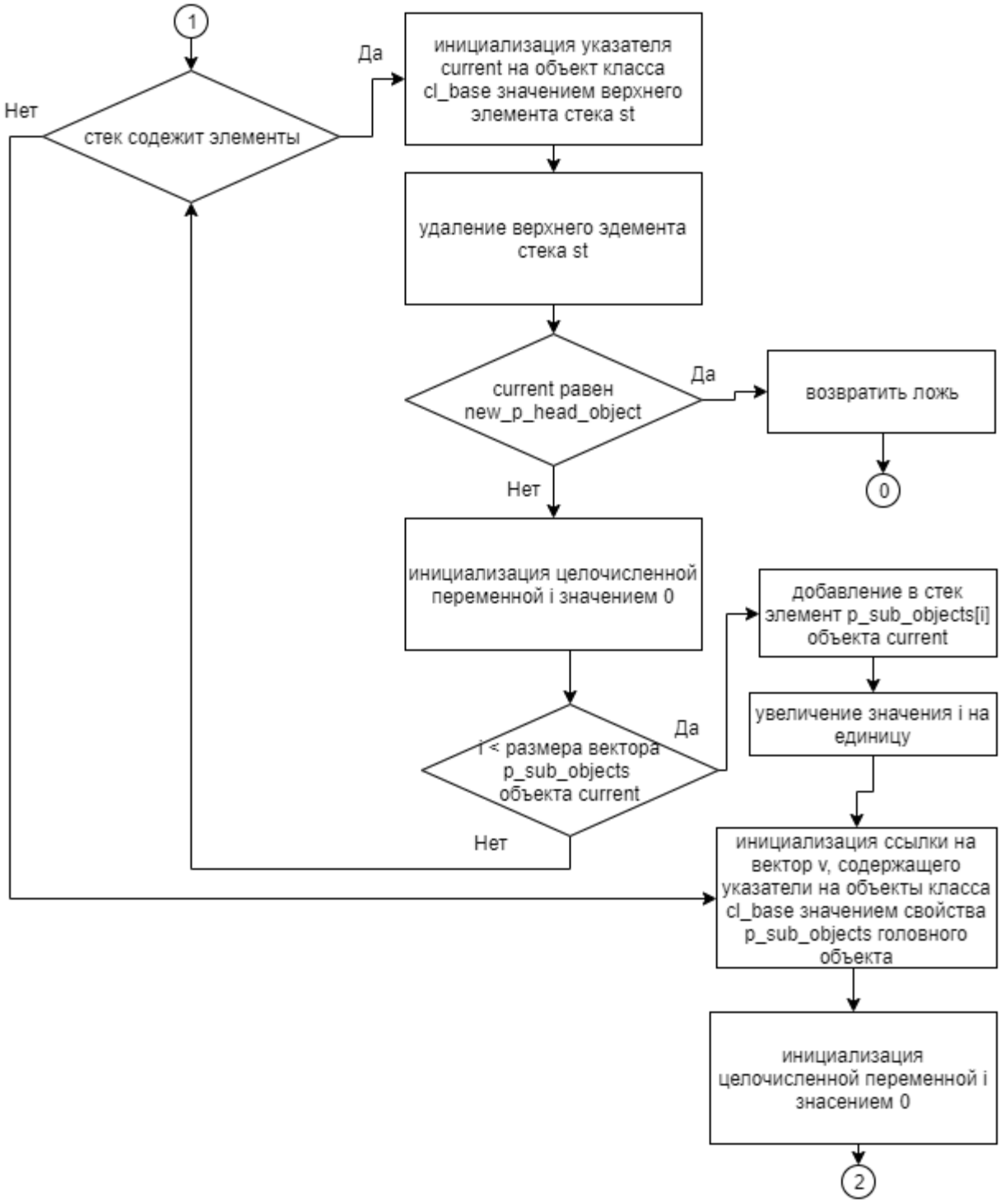

Рисунок 2 – Блок-схема алгоритма

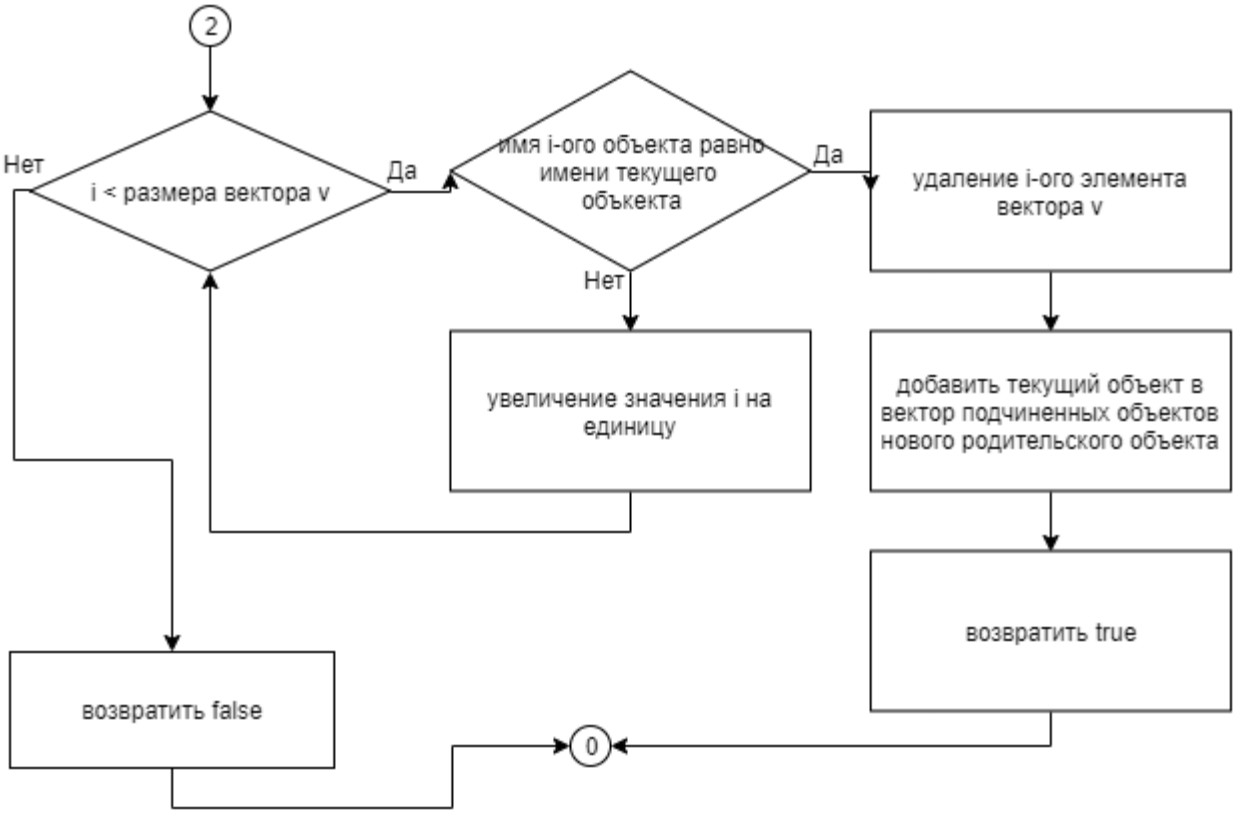

Рисунок 3 – Блок-схема алгоритма

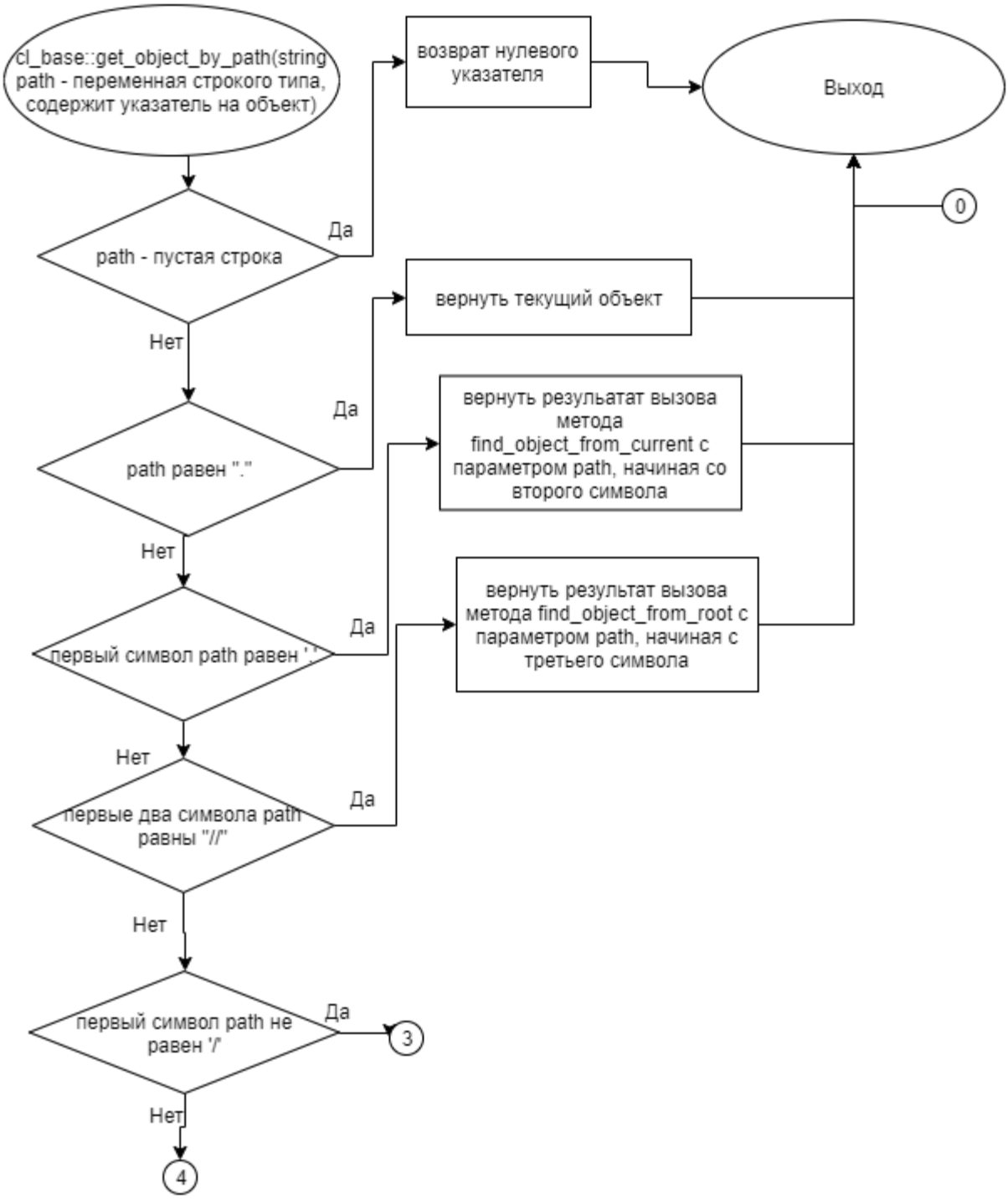

Рисунок 4 – Блок-схема алгоритма

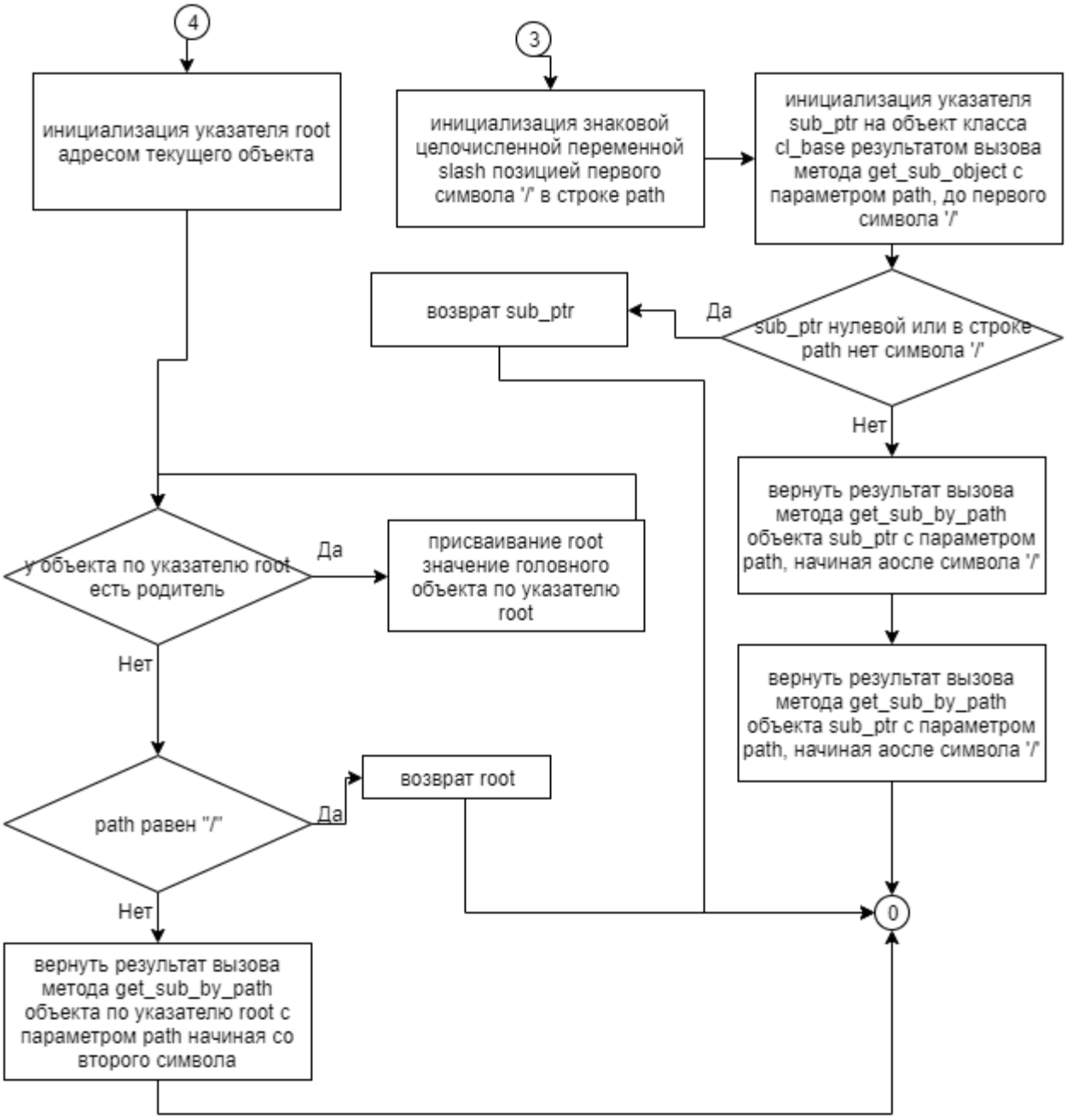

Рисунок 5 – Блок-схема алгоритма

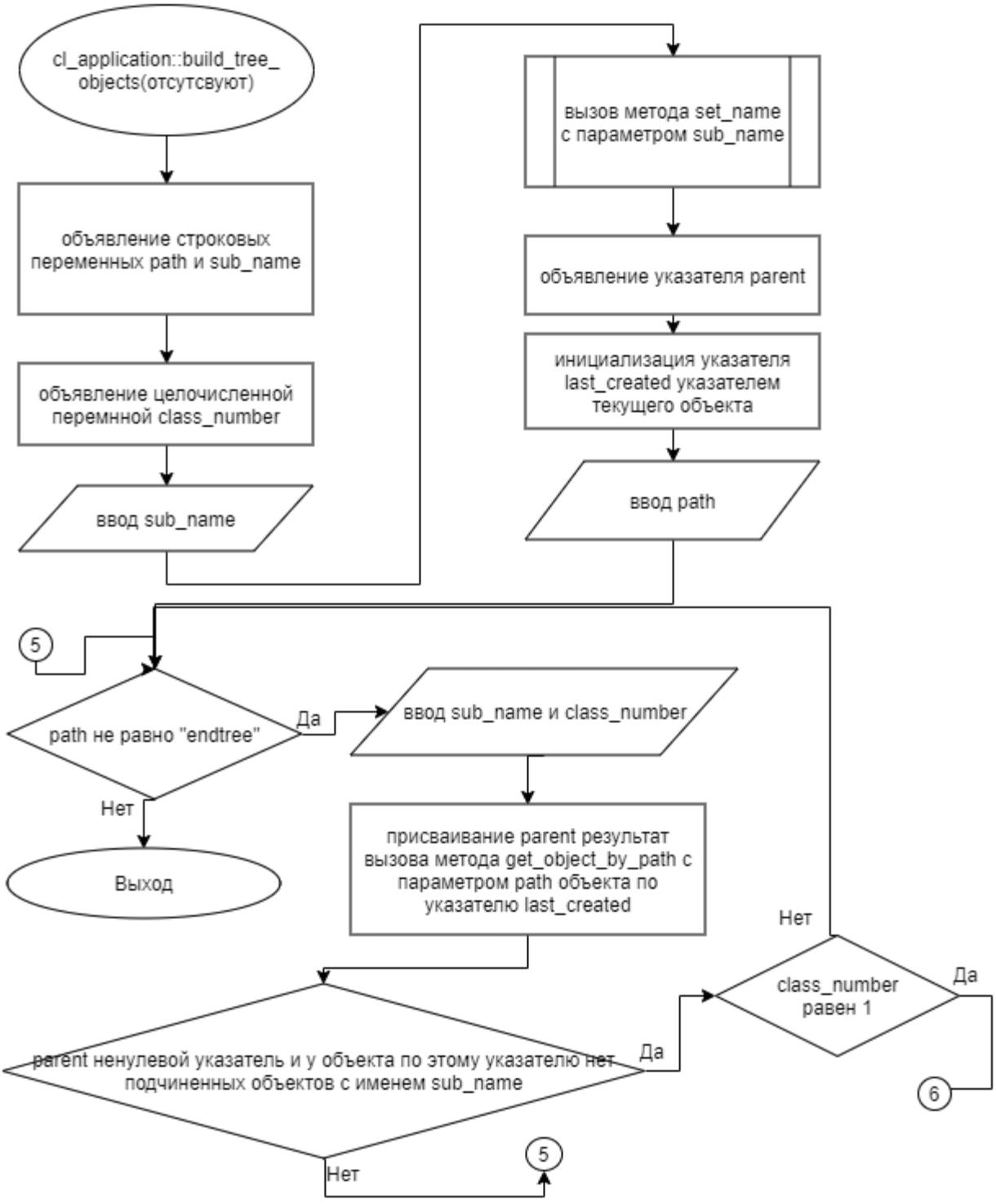

Рисунок 6 – Блок-схема алгоритма

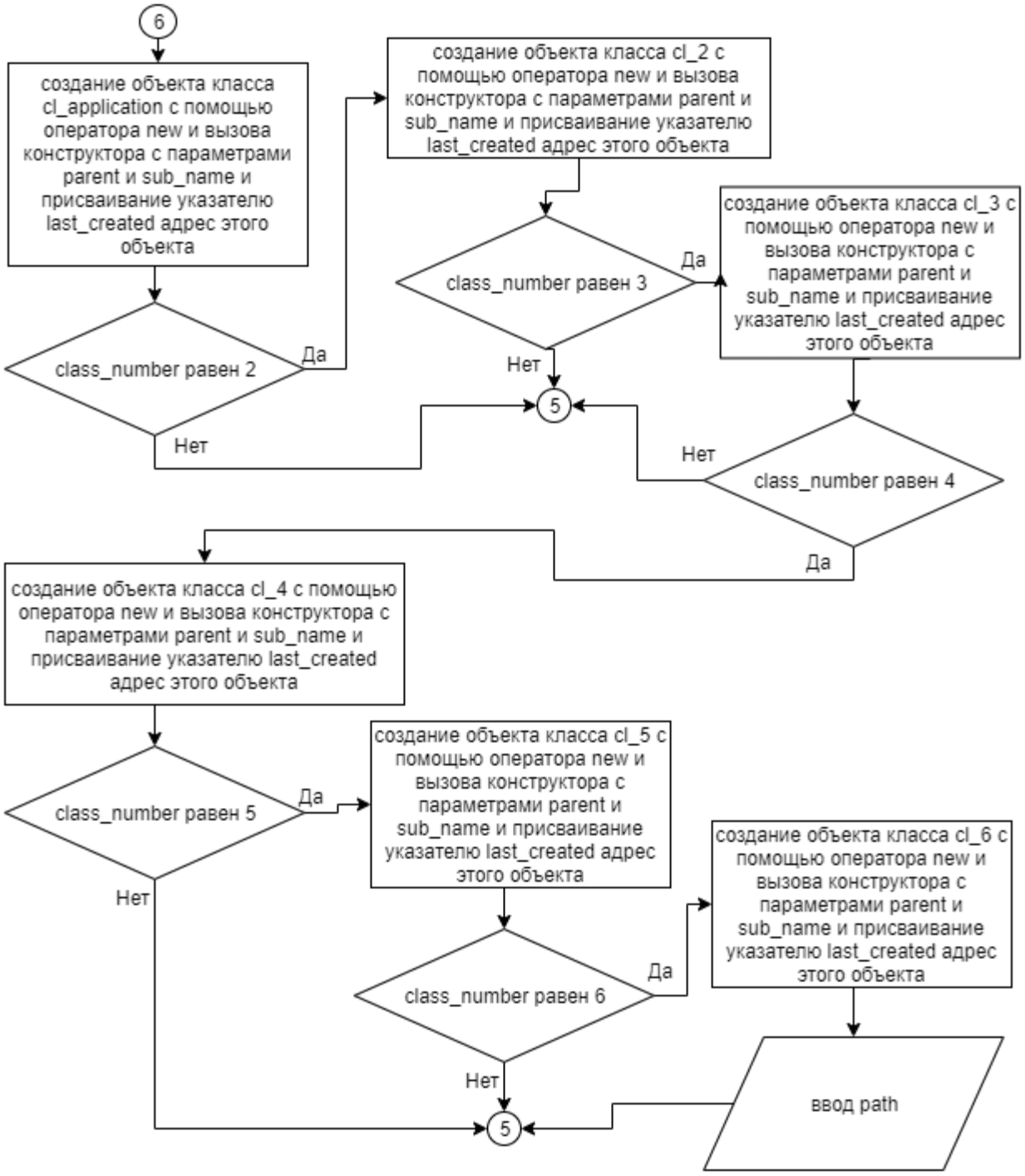

Рисунок 7 – Блок-схема алгоритма

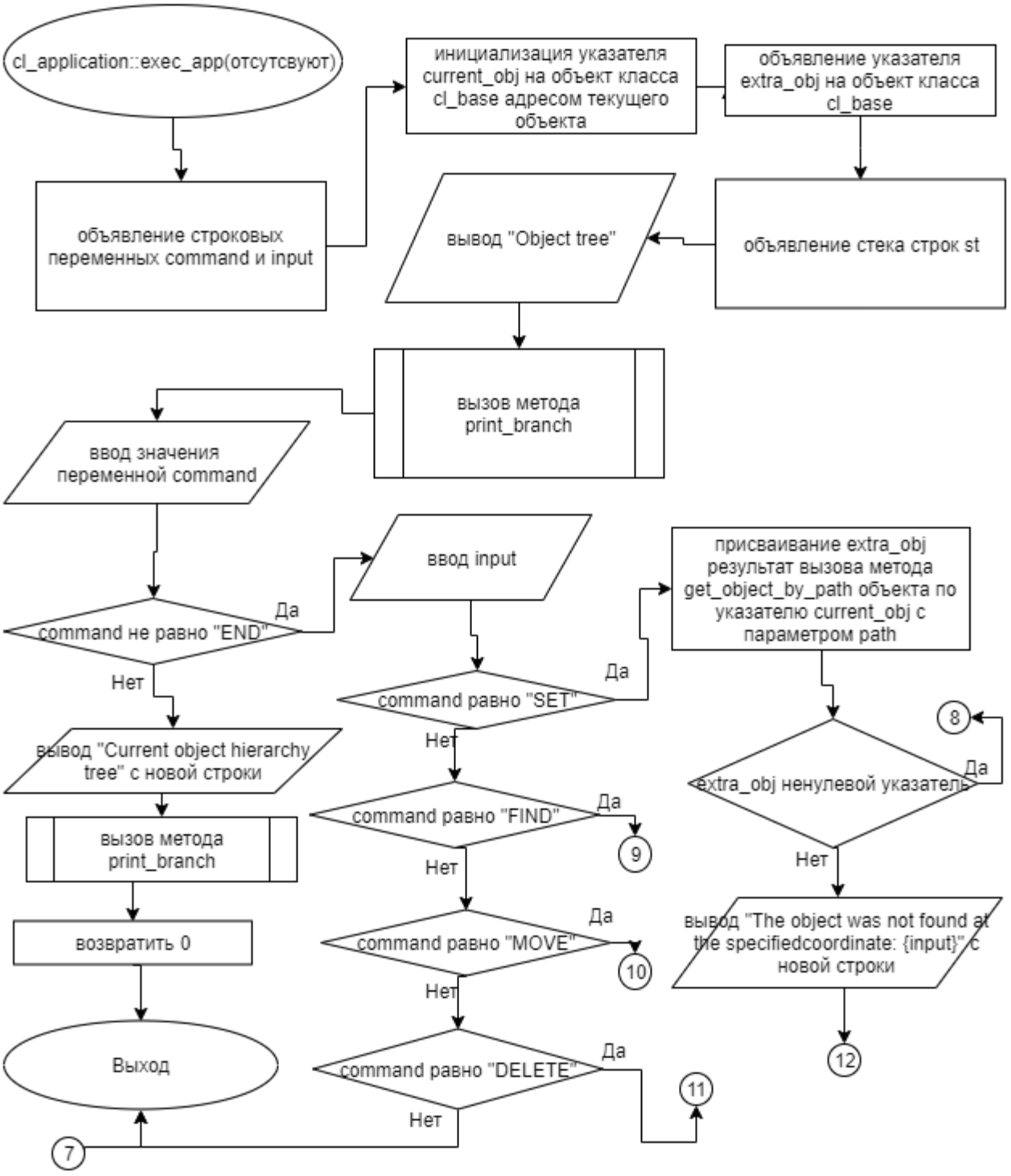

Рисунок 8 – Блок-схема алгоритма

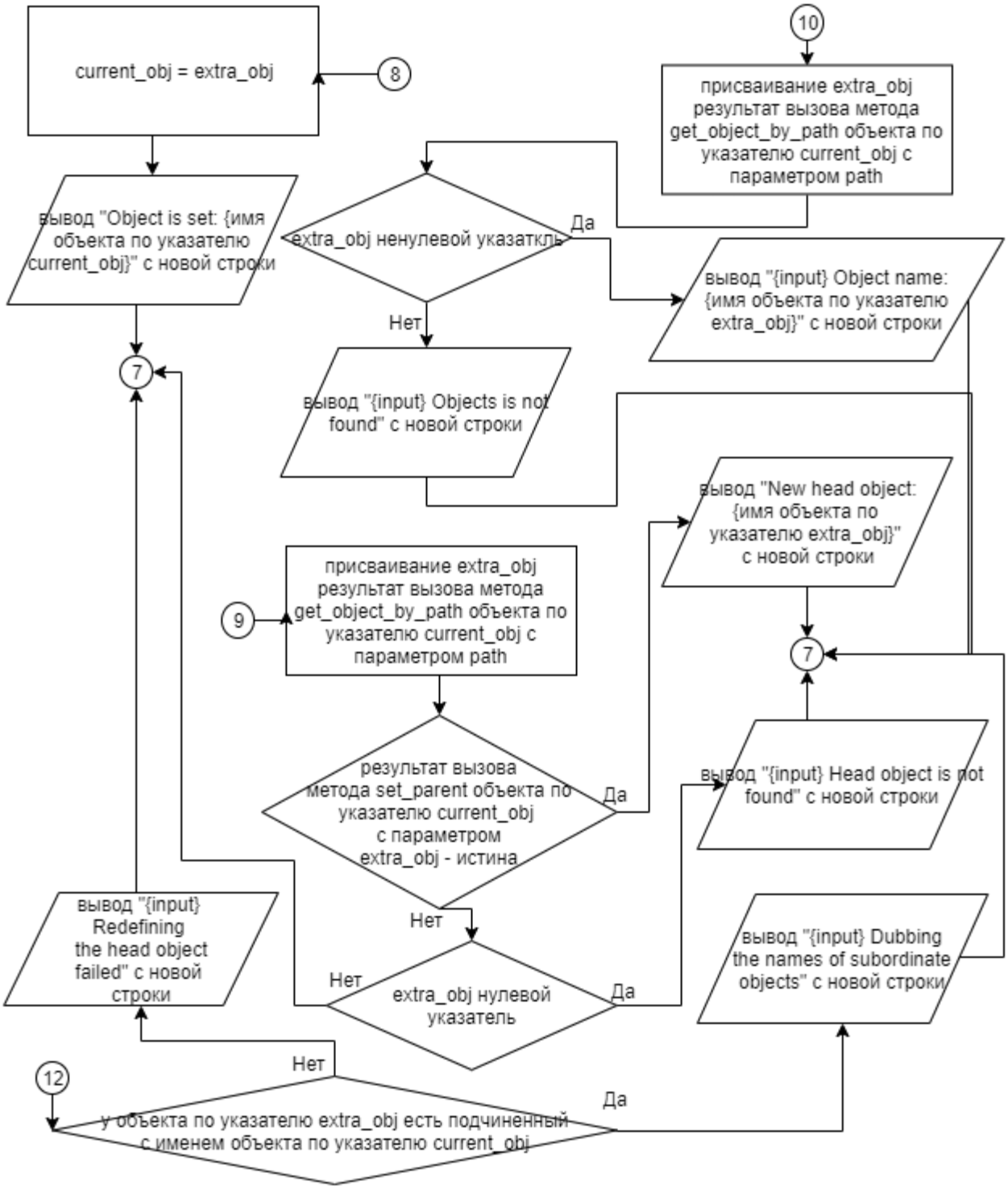

Рисунок 9 – Блок-схема алгоритма

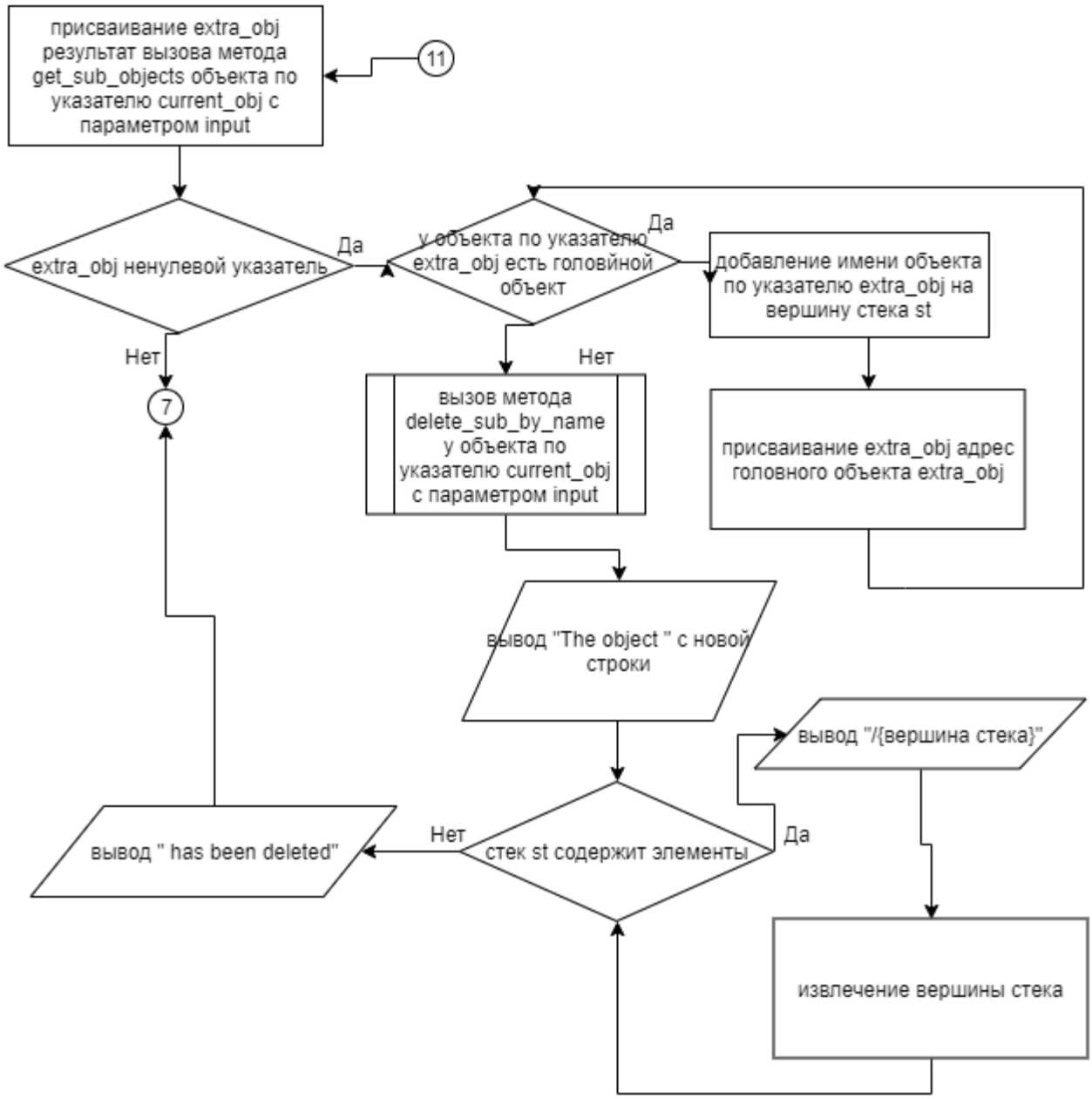

Рисунок 10 – Блок-схема алгоритма

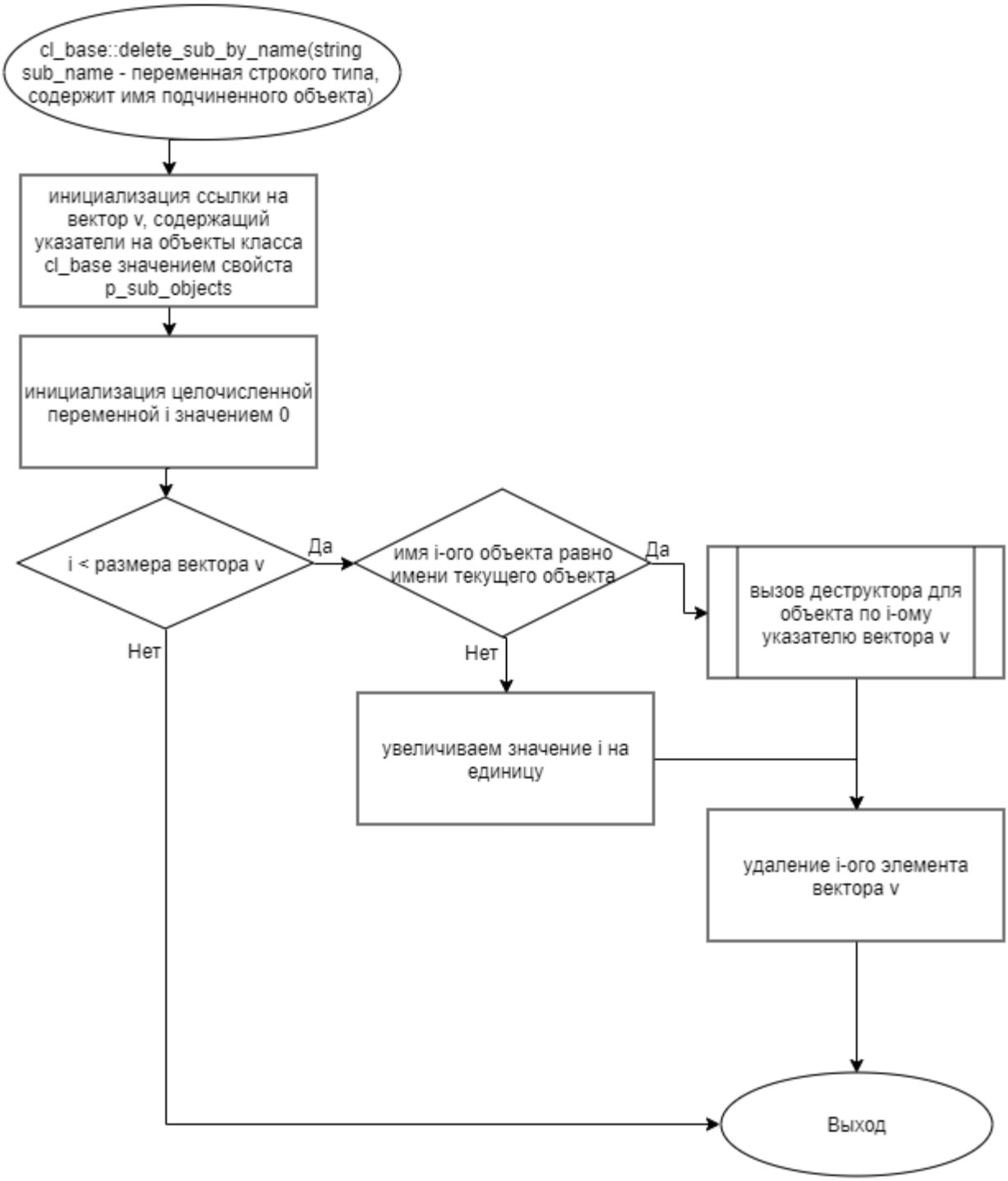

Рисунок 11 – Блок-схема алгоритма

Рисунок 12 – Блок-схема алгоритма

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* p_head, string s_name) : cl_base(p_head, s_name) {} //
параметризированный конструктор, вызывающий конструктор родителя
```

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 // наследование от класса cl_base
{
   public:
      cl_2(cl_base*   p_head,   string   s_name);   //   параметризированный
конструктор
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* p_head, string s_name) : cl_base(p_head, s_name) {} //
```

```
параметризированный конструктор, вызывающий конструктор родителя
```

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 // наследование от класса cl_base
{
   public:
      cl_3(cl_base*   p_head,   string   s_name);   //   параметризированный
конструктор
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* p_head, string s_name) : cl_base(p_head, s_name) {} //
параметризированный конструктор, вызывающий конструктор родителя
```

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 // наследование от класса cl_base
{
```

```
   public:
      cl_4(cl_base*   p_head,   string   s_name);   //   параметризированный
конструктор
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* p_head, string s_name) : cl_base(p_head, s_name) {} //
параметризированный конструктор, вызывающий конструктор родителя
```

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 // наследование от класса cl_base
{
   public:
      cl_5(cl_base*   p_head,   string   s_name);   //   параметризированный
конструктор
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"
```

```
cl_6::cl_6(cl_base* p_head, string s_name) : cl_base(p_head, s_name) {} //
параметризированный конструктор, вызывающий конструктор родителя
```

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 // наследование от класса cl_base
{
   public:
      cl_6(cl_base*   p_head,   string   s_name);   //   параметризированный
конструктор
};

#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 "stack"

cl_application::cl_application(cl_base* p_head) : cl_base(p_head) {}

void cl_application::build_tree_objects()
{
   /*
   Описание: метод построения дерева иерархии объектов
   */
   cout << "Object tree";
   string path, sub_name;
   int class_number;
   cin >> sub_name;
```

```
        this -> set_name(sub_name);
        cl_base* parent;
        cl_base* last_created = this;
        cin >> path;

        //Ввод иерархии объектов
        while(path != "endtree")
        {
           cin >> sub_name >> class_number;

           parent = last_created -> get_object_by_path(path);
           if (parent == nullptr)
           {
              this -> print_branch();
              cout << endl << "The head object " << path << " is not found";
              exit(1);
           }
           if (parent -> get_sub_object(sub_name) != nullptr)
              cout << endl << path << " Dubbing the names of subordinate objects";

           else
           {
              switch(class_number)
              {
                 case 1:
                    last_created = new cl_application(parent);
                    break;
                 case 2:
                    last_created = new cl_2(parent, sub_name);
                    break;
                 case 3:
                    last_created = new cl_3(parent, sub_name);
                    break;
                 case 4:
                    last_created = new cl_4(parent, sub_name);
                    break;
                 case 5:
                    last_created = new cl_5(parent, sub_name);
                    break;
                 case 6:
                    last_created = new cl_6(parent, sub_name);
                    break;
                 default: break;
              }
           }
           cin >> path;
        }
     }

     int cl_application::exec_app()
     {
        /*
        Описание: метод запуска системы
        */
        string command, input;
```

```
   cl_base* current_obj = this;
   cl_base* extra_obj = this;
   stack<string> st;
   this -> print_branch();
   cin >> command;
   while (command != "END")
   {
      cin >> input;

      if (command == "SET")
      {
         extra_obj = current_obj -> get_object_by_path(input);
         if (extra_obj != nullptr)
         {
            current_obj = extra_obj;
            cout << endl << "Object is set: " << current_obj -> get_name();
         }
         else cout << endl << "The object was not found at the
specifiedcoordinate: " << input;
      }

      else if (command == "FIND")
      {
         extra_obj = current_obj -> get_object_by_path(input);
         if (extra_obj != nullptr)
            cout << endl << input << "      Object name: " << extra_obj ->
get_name();
         else cout << endl << input << "     Object is not found";
      }

      else if (command == "MOVE")
      {
         extra_obj = current_obj -> get_object_by_path(input);
         if (current_obj -> set_head_object(extra_obj))
            cout << endl << "New head object: " << extra_obj-> get_name();

         else if (extra_obj == nullptr)
            cout << endl << input << "    Head object is not found";

         else if (extra_obj -> get_sub_object(current_obj -> get_name()) !=
nullptr)
            cout << endl << input << "     Dubbing the names of subordinate
objects";

         else cout << endl << input << "       Redefining the head object
failed";
      }

      else if (command == "DELETE")
      {
         extra_obj = current_obj -> get_sub_object(input);
         if (extra_obj != nullptr)
         {
            while (extra_obj -> get_head() != nullptr)
            {
```

```
                st.push(extra_obj -> get_name());
                extra_obj = extra_obj -> get_head();
             }
             current_obj -> delete_sub_by_name(input);
             cout << endl << "The object ";
             while (!st.empty())
             {
                cout << '/' << st.top();
                st.pop();
             }
             cout << " has been deleted";
          }
       }
       cin >> command;
    }
    cout << endl << "Current object hierarchy tree";
    this -> print_branch();
    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"

class cl_application : public cl_base // наследование от класса cl_base
{
   public:
      cl_application(cl_base*    p_head_object);    //    параметризированный
конструктор
      void build_tree_objects(); // метод построения дерева иерархии объектов
      int exec_app(); // запуск основного алгоритма программы
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
#include "stack"
```

```
cl_base::cl_base(cl_base* p_head, string s_name)
{
   /*
   Описание: параметризированный конструктор
   Параметры:
      p_head_object - указатель на головной объект
      s_name - имя узла дерева
   */

   this -> p_head_object = p_head; // разименование объекта
   this -> s_object_name = s_name; // разименование объекта

   if (p_head_object != nullptr)
      p_head_object -> p_sub_objects.push_back(this);
}

cl_base::~cl_base()
{
   /*
   Описание: деструктор
   */

   for (int i = 0; i < p_sub_objects.size(); i++) // проход по подчиненным
объектам
      delete p_sub_objects[i]; // удаление подчиненного объекта
}

bool cl_base::set_name(string s_new_name)
{
   /*
   Описание: метод редактирования имени объекта
   Параметры:
      s_new_name - новое имя узла дерева
   */

   if (p_head_object != nullptr) // проверяем не нулевой ли указатель на
головной объект
      for (int i = 0; i < p_head_object -> p_sub_objects.size(); i++)
         if (p_head_object -> p_sub_objects[i] -> get_name() == s_new_name)
            return false;

   this -> s_object_name = s_new_name;
   return true;
}

string cl_base::get_name()
{
   /*
   Описание: метод получения имени объекта
   Параметры: нет
   */
   return this -> s_object_name;
}
```

```
cl_base* cl_base::get_head()
{
   /*
   Описание: метод получения указателя на головной объект текущего объекта
   Параметры: нет
   */
   return this -> p_head_object;
}

cl_base* cl_base::get_sub_object(string s_name)
{
   /*
   Описание: получение указателя на непосредственно подчиненный объект по
имени
   Параметры:
      s_name - имя искомого объекта
   */

   for (int i = 0; i < p_sub_objects.size(); i++)
      if (p_sub_objects[i] -> get_name() == s_name)
         return p_sub_objects[i];

   return nullptr;
}

cl_base* cl_base::find_object_from_current(string s_name)
{
   /*
   Описание: метод поиска объекта на ветке по имени(обход графа в ширину)
   Параметры:
      s_name - имя искомого объекта
   */
   queue<cl_base*> q; // очередь элементов
   cl_base* found = nullptr; // объявление нулевого указателя
   q.push(this); // добавдение в очередь всех наших элементов
   while (!q.empty()) // пока очередь не пуста
   {
      cl_base* e = q.front();  //указатель на наш объект
      if (e -> get_name() == s_name) // совпадает ли имя с текущим
      {
         if (found != nullptr) // указатель не пустой и объект не уникальный
            return nullptr;
         else
            found = e; // записываем указатель на этот объект
      }



      for (int i = 0; i < e -> p_sub_objects.size(); i++) // проходим по
подчиненным элементам
      q.push(e -> p_sub_objects[i]); // и каждый элемент добавляем в очередь

      q.pop(); // удаление элемента из начала очереди чтобы пройти по всем
элементам
   }
```

```
   return found;
}

cl_base* cl_base::find_object_from_root(string s_name)
{
   /*
   Описание: метод поиска объекта на дереве иерархии по имени
   Параметры:
      s_name - имя искомого объекта
   */
   if (p_head_object != nullptr)
      return p_head_object -> find_object_from_root(s_name);

   else return find_object_from_current(s_name);
}

void cl_base::print_branch(int layer)
{
   /*
   Описание: метод вывода иерархии объектов (дерева или ветки) от текущего
объекта
   Параметры:
      layer - уровень на дереве иерархии
   */
   cout << endl;

   for (int i = 0; i < layer; i++)
      cout << "    ";

   cout << this -> get_name();
   for (int i = 0; i < p_sub_objects.size(); i++)
      p_sub_objects[i] -> print_branch(layer + 1);
}

void cl_base::print_branch_status(int layer)
{
   /*
   Описание: метод вывода иерархии объектов (дерева или ветки) и отметок их
готовности от текущего объекта
   Параметры:
      layer - уровень на дереве иерархии
   */
   cout << endl;

   for (int i = 0; i < layer; ++i)
      cout << "    ";

   if (this -> status != 0)
      cout << this -> get_name() << " is ready";

   else cout << this -> get_name() << " is not ready";

   for (int i = 0; i < p_sub_objects.size(); ++i)
      p_sub_objects[i] -> print_branch_status(layer + 1);
```

```
}

void cl_base::set_status(int status)
{
   /*
   Описание: метод установки готовности объекта
   Параметры:
      status - переменная целого типа, содержит номер состояния
   */
   if (p_head_object == nullptr || p_head_object -> status != 0)
      this -> status = status;

   if (status == 0)
   {
      this -> status = status;
      for (int i = 0; i < p_sub_objects.size(); i++)
         p_sub_objects[i] -> set_status(status);
   }
}

bool cl_base::set_head_object(cl_base* new_p_head_object)
{
   /*
   Описание: метод переопределения головного объекта для текущего в дереве
иерархии
   Параметры:
      new_p_head_object - переменная содержит новое имя головного объекта
   */
   if (this -> get_head() == new_p_head_object)
      return true;

   if (this -> get_head() == nullptr || new_p_head_object == nullptr)
      return false;

   if (new_p_head_object -> get_sub_object(this -> get_name()) != nullptr)
      return false;

   stack<cl_base*> st;
   st.push(this);
   while (!st.empty())
   {
      cl_base* current = st.top();
      st.pop();
      if (current == new_p_head_object)
         return false;

      for (int i = 0; i < current -> p_sub_objects.size(); i++)
         st.push(current -> p_sub_objects[i]);
   }
   vector<cl_base*> & v = this -> get_head() -> p_sub_objects;
   for (int i = 0; i < v.size(); i++)
      if (v[i] -> get_name() == this -> get_name())
      {
         v.erase(v.begin() + i);
         new_p_head_object -> p_sub_objects.push_back(this);
```

```
            return true;
         }

      return false;
   }

   void cl_base::delete_sub_by_name(string sub_name)
   {
      /*
      Описание: метод удаления подчиненного объекта по наименованию
      Параметры:
         sub_name - переменная строкого типа, содержит имя подчиненного объекта
      */
      vector<cl_base*> & v = this -> p_sub_objects;
      for (int i = 0; i < v.size(); i++)
         if (v[i] -> get_name() == sub_name)
         {
            delete v[i];
            v.erase(v.begin() + i);
            return;
         }
   }

   cl_base* cl_base::get_object_by_path(string path)
   {
      /*
      Описание: метод получения указателя на любой объект в составе дерева
   иерархии объектов согласно пути
      Параметры:
         path - переменная строкого типа, содержит указатель на объект
      */
      if (path.empty())
         return nullptr;

      if (path == ".")
         return this;

      if (path[0] == '.')
         return find_object_from_current(path.substr(1));

      if (path.substr(0, 2) == "//")
         return this -> find_object_from_root(path.substr(2));

      if (path[0] != '/')
      {
         size_t slash = path.find('/');
         cl_base* sub_ptr = this -> get_sub_object(path.substr(0, slash));
         if (sub_ptr == nullptr || slash == string::npos)
            return sub_ptr;

         return sub_ptr -> get_object_by_path(path.substr(slash + 1));
      }

      cl_base* root = this;
      while (root -> get_head() != nullptr)
```

```
            root = root -> get_head();

         if (path == "/")
            return root;

         return root -> get_object_by_path(path.substr(1));
      }
```

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>
      #include <queue>

      using namespace std;

      class cl_base
      {
         private:
            int status = 0; // индикатор состояния объекта
            string s_object_name; // наименование объекта строкового типа
            cl_base* p_head_object; //указатель на головной объект
            vector <cl_base*> p_sub_objects; //вектор подчиненных объектов
         public:
            cl_base(cl_base*   p_head,   string   s_name   =   "Base   object");   //
      параметризированный конструктор
            ~cl_base(); // деструктор
            bool set_name(string s_new_name); // метод установки имени
            string get_name(); // метод получения имени
            cl_base*  get_head();  //  метод  получения  указателя  на  родительский
      объект
            cl_base* get_sub_object(string s_name); // метод получения указателя на
      дочерний объект

            cl_base*  find_object_from_current(string  s_name);  //  метод  поиска
      объекта на ветке дереве иерархии от текущего по имени
            cl_base* find_object_from_root(string s_name); // метод поиска объекта
      на дереве иерархии по имени
            void print_branch(int layer = 0); // метод вывода иерархии объектов
      (дерева или ветки) от текущего объекта
            void  print_branch_status(int  layer  =  0);  //  метод  вывода  иерархии
      объектов (дерева или ветки) и отметок их готовности от текущего объекта
            void set_status(int status); //метод установки готовности объекта, в
      качестве  параметра  передается  переменная  целого  типа,  содержит  номер
      состояния
```

```
     bool set_head_object(cl_base* p_head_object); // метод переопределения
головного объекта для текущего в дереве иерархии
     void   delete_sub_by_name(string   sub_name);   //   метод   удаления
подчиненного объекта по наименованию
     cl_base* get_object_by_path(string path); // метод получения указателя
на любой объект в составе дерева иерархии объектов согласно пути
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_application.h"

int main()
{
   cl_application  ob_cl_application(nullptr);  // создание объекта приложени
   ob_cl_application.build_tree_objects();      // конструирование системы,
   return ob_cl_application.exec_app();         // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 7.

Таблица 7 – 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

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).