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.....13
3.1 Алгоритм метода find_obj_by_coord класса cl_base.....13
3.2 Алгоритм метода delete_sub_obj класса cl_base.....15
3.3 Алгоритм метода set_head_obj класса cl_base.....16
3.4 Алгоритм метода build_tree_objects класса cl_application.....17
3.5 Алгоритм функции main.....19
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....21
5 КОД ПРОГРАММЫ.....28
5.1 Файл cl_2.cpp.....28
5.2 Файл cl_2.h.....28
5.3 Файл cl_3.cpp.....28
5.4 Файл cl_3.h.....29
5.5 Файл cl_4.cpp.....29
5.6 Файл cl_4.h.....30
5.7 Файл cl_5.cpp.....30
5.8 Файл cl_5.h.....30
5.9 Файл cl_6.cpp.....31
5.10 Файл cl_6.h.....31
5.11 Файл cl_application.cpp.....31
5.12 Файл cl_application.h.....35
5.13 Файл cl_base.cpp.....35
5.14 Файл cl_base.h.....41

5.15 Файл main.cpp.....42
6 ТЕСТИРОВАНИЕ.....43
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....47

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:

- для решения используется то же, что и в КВ2.

Класс cl_base:

- функционал:
    - метод find_obj_by_coord — метод поиска объекта по заданной координате;
    - метод delete_sub_obj — метод удаления подчиненного объекта по имени;
    - метод set_head_obj — метод смены головного элемента.

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 find_obj_by_coord класса cl_base

Функционал: метод поиска объекта по заданной координате.

Параметры: string s_coord - координата искомого объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода find_obj_by_coord класса cl_base

№	Предикат	Действия	№ перехода
1		p_root присваивается указатель на текущий объект Инициализация i_slash_2 = 0 , для хранения индекса 2-ого слэша Строковый тип s_name, для хранения пути объекта	2
2	s_coord равен "/"		3
			4
3	p_root не равен пустому указателю	p_root присваивается значение вызываемого метода get_p_head() от p_root	3
		возвращаем p_root	∅
4	s_coord равен "."	возвращаем текущий объект	∅
			5
5	s_coord[0] равен "/" и s_coord[1] равен "/"	от текущего объекта вызывается метод search_object_from_tree с параметром s_coord с встроенном методом substr(2) (начиная с 3 символа)	∅

№	Предикат	Действия	№ перехода
			6
6	s_coord[0] равен "."	от текущего объекта вызывается метод search_object_from_tree с параметром s_coord с встроенном методом substr(1) (начиная со 2 символа)	∅
			7
7		i_slah_2 присваивается индекс второго слэша, если он не найден он равен -1	8
8	s_coord[0] равен "/"		9
			14
9	головной объект не равен пустому указателю	p_root присваивается значение вызываемого метода get_p_head() от p_root	9
			10
10	i_slah_2 не равен -1	s_name присваивается значение объекта до следующего слэша из переменной s_coord	11
		s_name присваивается значение объекта с 1 символа s_coord	13
11		p_obj присваивается значение вызываемого метода get_subordibane_object с параметром s_name объекта p_root	12
12	p_obj не равен пустому указателю	возвращаем вызываем метод find_obj_by_coord с параметром s_coord (отсекается строка от найденного индекса слэша) объекта p_obj	∅
		возвращаем p_obj	∅
13		возвращаем вызываемый метод get_subordinate_object с парараметром s_name объекта p_root	∅
14	i_slah_2 не равен -1	s_name присваивается значение объекта до следующего слэша из переменной s_coord	15

№	Предикат	Действия	№ перехода
		возвращаем вызываемый метод get_subordinate_object с параметром s_coord текущего объекта	∅
15		p_obj присваивается значение вызываемого метода get_subordibane_object с параметром s_name текущего объекта	16
16	p_obj не равен пустому указателю	возвращаем вызываемый метод find_obj_by_coord с параметром s_coord (строка, после найденного первого слэша) объекта p_obj	∅
		возвращаем p_obj	∅

3.2 Алгоритм метода delete_sub_obj класса cl_base

Функционал: метод удаления подчиненного объекта по имени.

Параметры: string s_name - имя удаляемого объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода delete_sub_obj класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация i = длины subordinates_objects - 1	2
2	i >= 0		3
			∅
3	i-ый элемент subordinates_objects с полученным именем s_object_name равен s_name	удаляем этот i-ый элемент subordinate_objects	∅
			4
4		--i	2

3.3 Алгоритм метода set_head_obj класса cl_base

Функционал: метод смены головного элемента.

Параметры: cl_base* p_new_head - имя нового головного объекта.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода set_head_obj класса cl_base

№	Предикат	Действия	№ перехода
1		создаем дубликат указатель p_temp = p_new_head	2
2	p_new_head равен нулевому указателю	возвращаем false	∅
			3
3	корневой элемент равен нулевому указателю	возвращаем false	∅
			4
4	у p_new_head есть подчиненые с одинаковым именем	возвращаем false	∅
			5
5	p_temp не равен пустому множеству		6
			8
6	p_temp равен текущему объекту	возвращаем false	∅
			7
7		p_temp присвается значение p_head_object p_temp	5
8		у текущего головного объекта удаляеются подчиненые объекты	9
9		присваиваем p_head_object текущего объекта	10

№	Предикат	Действия	№ перехода
		p_new_head	
10		p_new_head добавляется подчиненные объекты текущего объекта	11
11		возвращаем true	∅

3.4 Алгоритм метода build_tree_objects класса cl_application

Функционал: используется для построения дерева объектов и обработки подаваемых команд.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		все тоже самое, что и в кв2	2
2		p_current присваивается указатель на текущий объект объявление строковых переменных s_command, s_coordinate Объявление указателя p_obj	3
3		ввод значения для s_command	4
4	s_command равен "END"		∅
			5
5		ввод значения для s_coirdinate	6
6		p_obj присвается значение метода find_obj_by_coord с параметром s_coordinate объекта p_current	7
7	s_command равен "SET"		8

№	Предикат	Действия	№ перехода
			10
8	p_obj не равен пустому указателю	p_current присваивается значение p_obj	9
		вывод переноса строки << "The object was not found at the specified coordinate: " << s_coordinate	10
9		вывод переноса строки << "Object is set: " << вызов метода get_name объекта p_current	10
10	s_command равен "FIND"		11
			12
11	p_obj не равен пустому указателю	вывод переноса строки << s_coordinate << " Object name: " << вызов метода get_name объекта p_obj	12
		вывод переноса строки << s_coordinate << " Object is not found"	12
12	s_command равен "MOVE"		13
			18
13	метод set_head_obj с параметорм p_obj объекта p_current вернет true	вывод << перенос строки << "New head object: " << вызов метода get_name объекта p_obj	18
			14
14		указатель p_temp (дубликат) присваивается значение p_obj	15
15	p_obj равен пустому указтелю	вывод переноса строки << s_coordinate << " Head object is not found"	16
			16
16	у головного объекта p_obj есть пордчиненные объекты с похожим именем p_obj	вывод переноса строки <<" Dubbing the names of subordinate objects"	17
			17

№	Предикат	Действия	№ перехода
17	вызываемый метод get_p_head текущего объекта равен пустому указателю	вывод переноса строки << s_coordinate << " Redefining the head object failed"	18
			18
18	s_command равен "DELETE"		19
			25
19	p_obj не равен пустому указателю	объявляется строковая переменная s_abs_path, для хранения абсолютного пути объекта объявляется указатель дубликат p_object хранящий значение p_obj	20
			25
20	головной объект p_object не равен пустому указателю	s_abs_path присваивается "/" + get_name() объекта p_object + s_abs_path	21
			22
21		p_object = get_p_head() объекта p_object	20
22		вывод переноса строки << "The object " << s_abs_path << " has been deleted"	23
23		от текущего объекта p_current вызывается метод delete_sub_obj с параметром p_obj->get_name() для удаления подчиненных объектов	24
24		деструктор найденного объекта p_obj	25
25		ввод значения для s_command	∅

3.5 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 5.

Таблица 5 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		выполнение алгоритма данной функции из предыдущей работы КВ2	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-7.

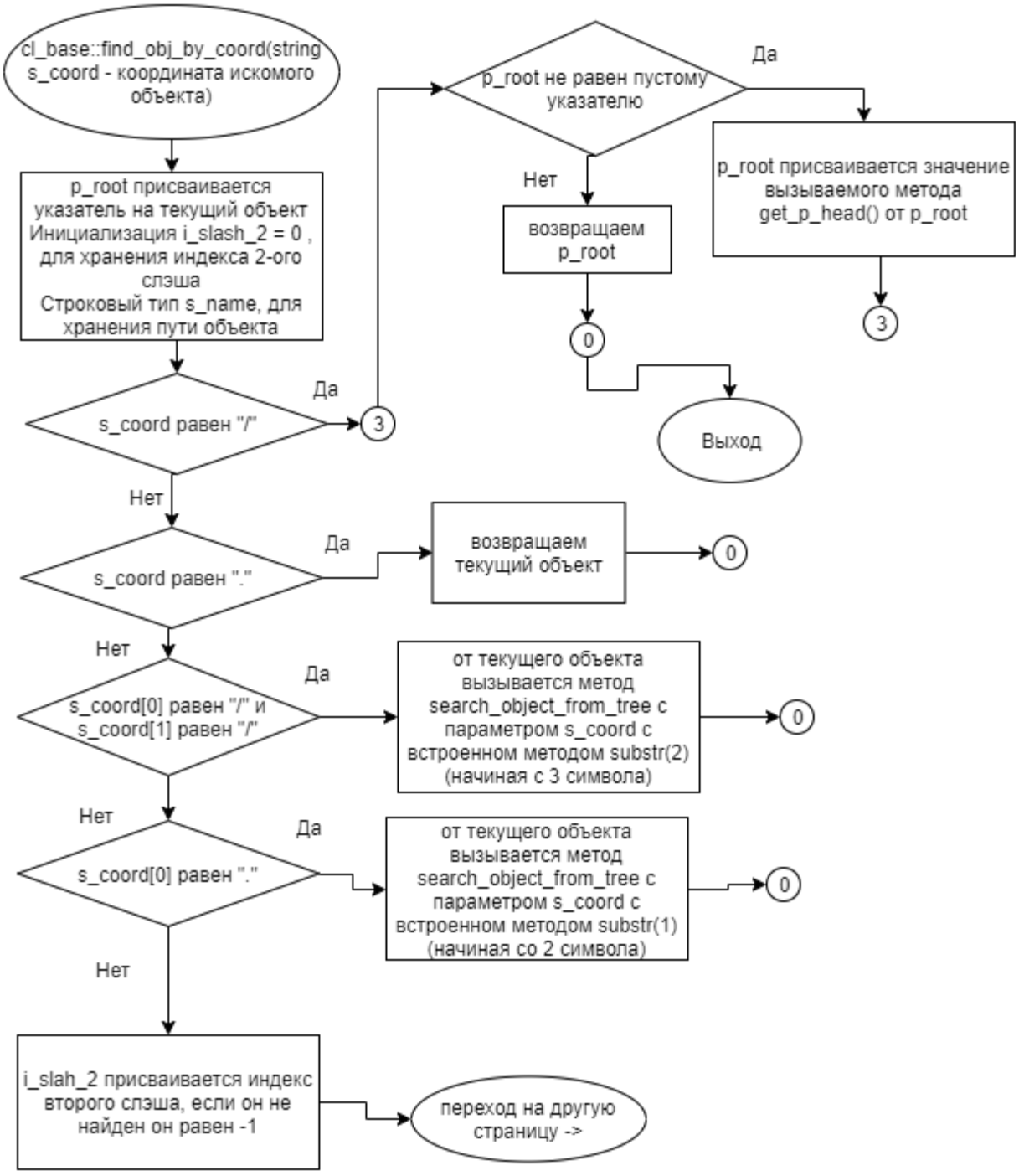

Рисунок 1 – Блок-схема алгоритма

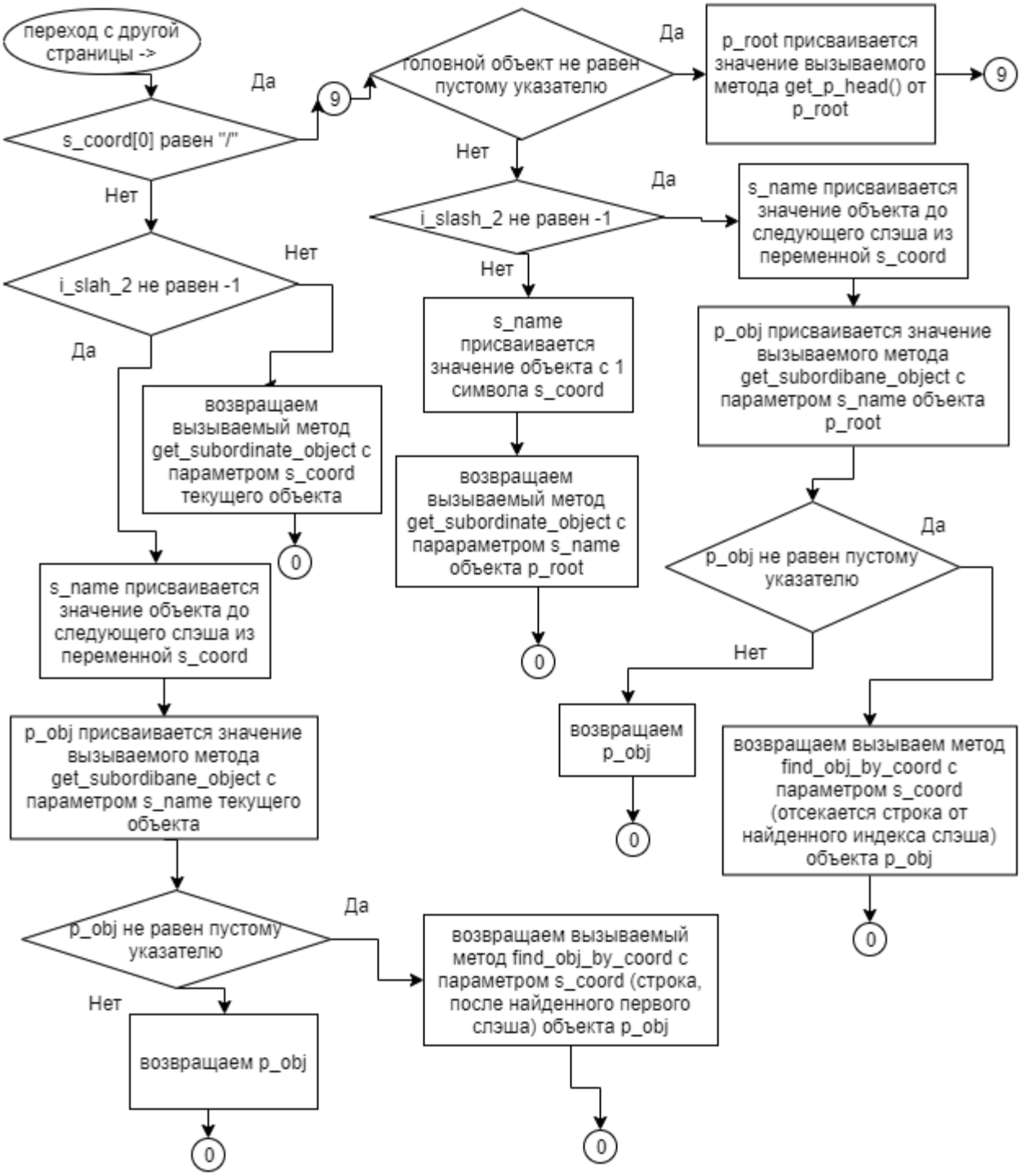

Рисунок 2 – Блок-схема алгоритма

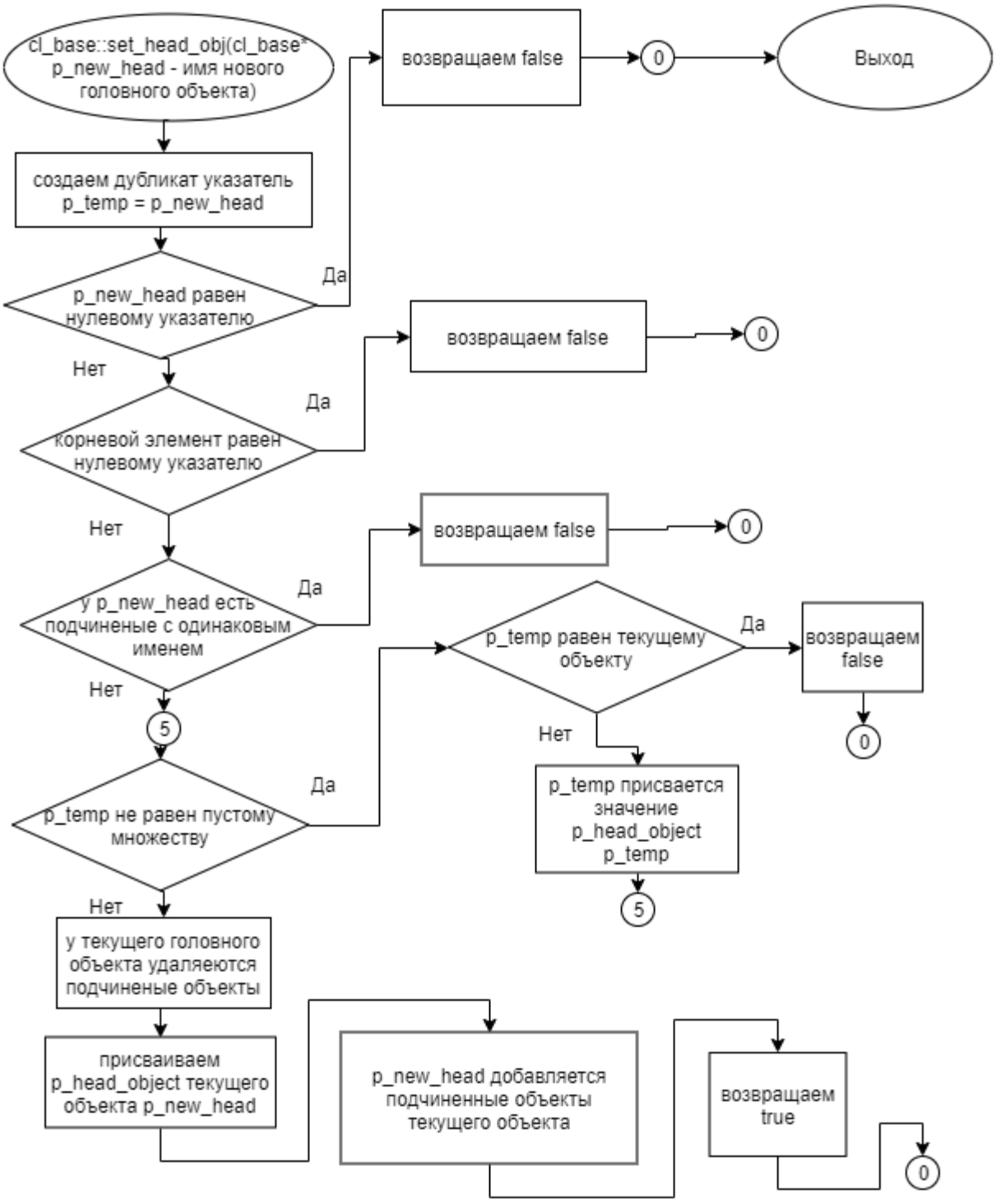

Рисунок 3 – Блок-схема алгоритма

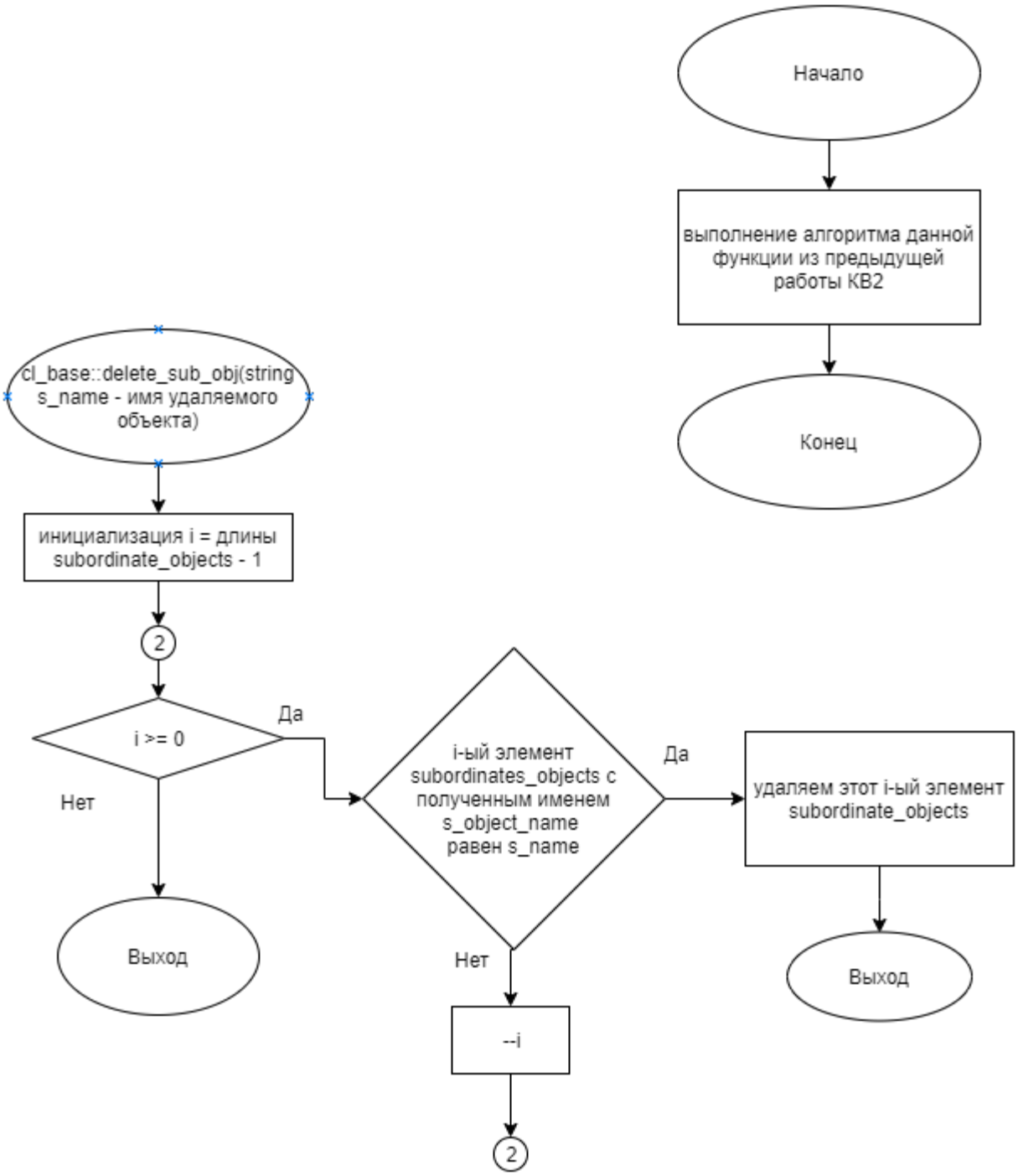

Рисунок 4 – Блок-схема алгоритма

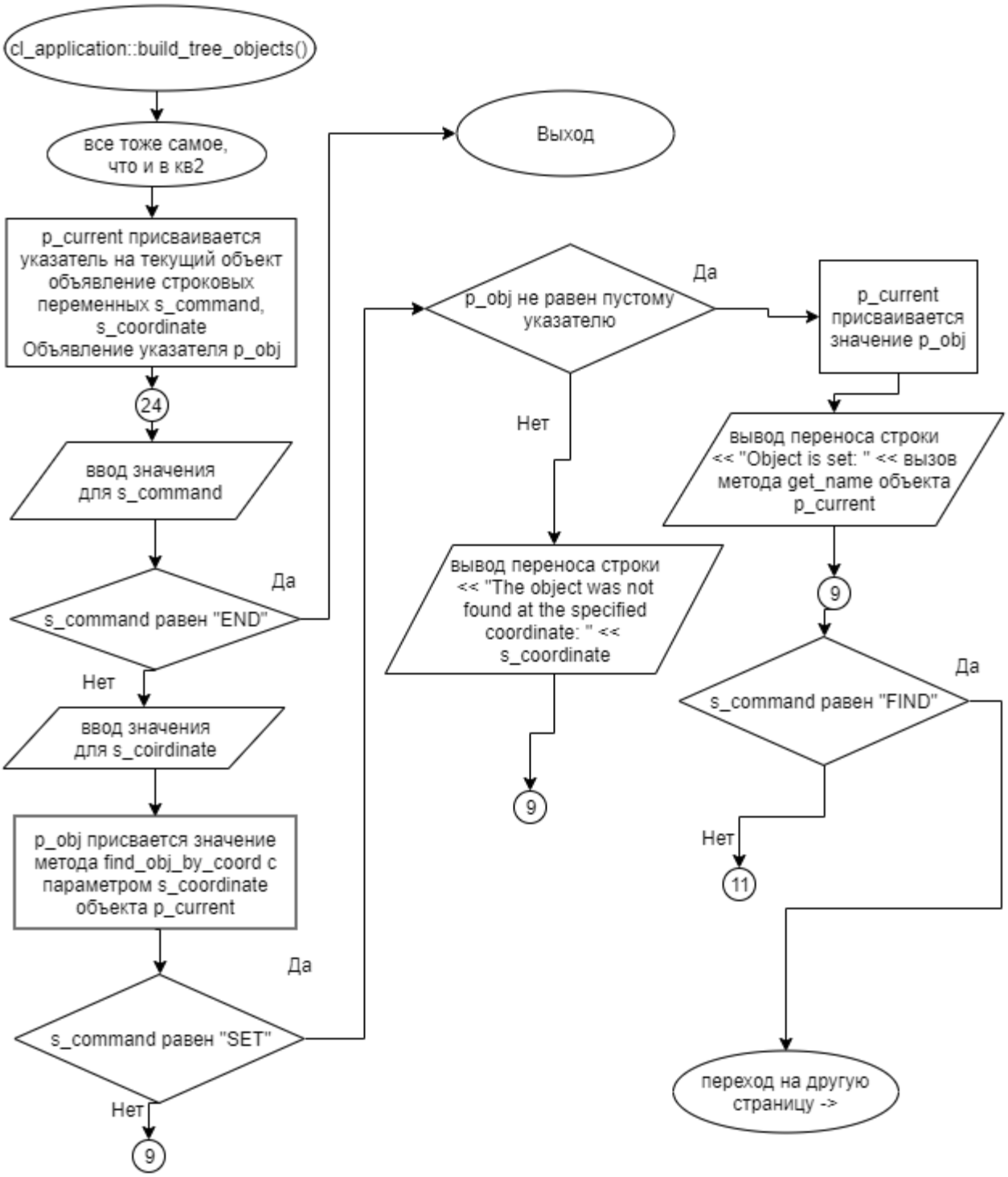

Рисунок 5 – Блок-схема алгоритма

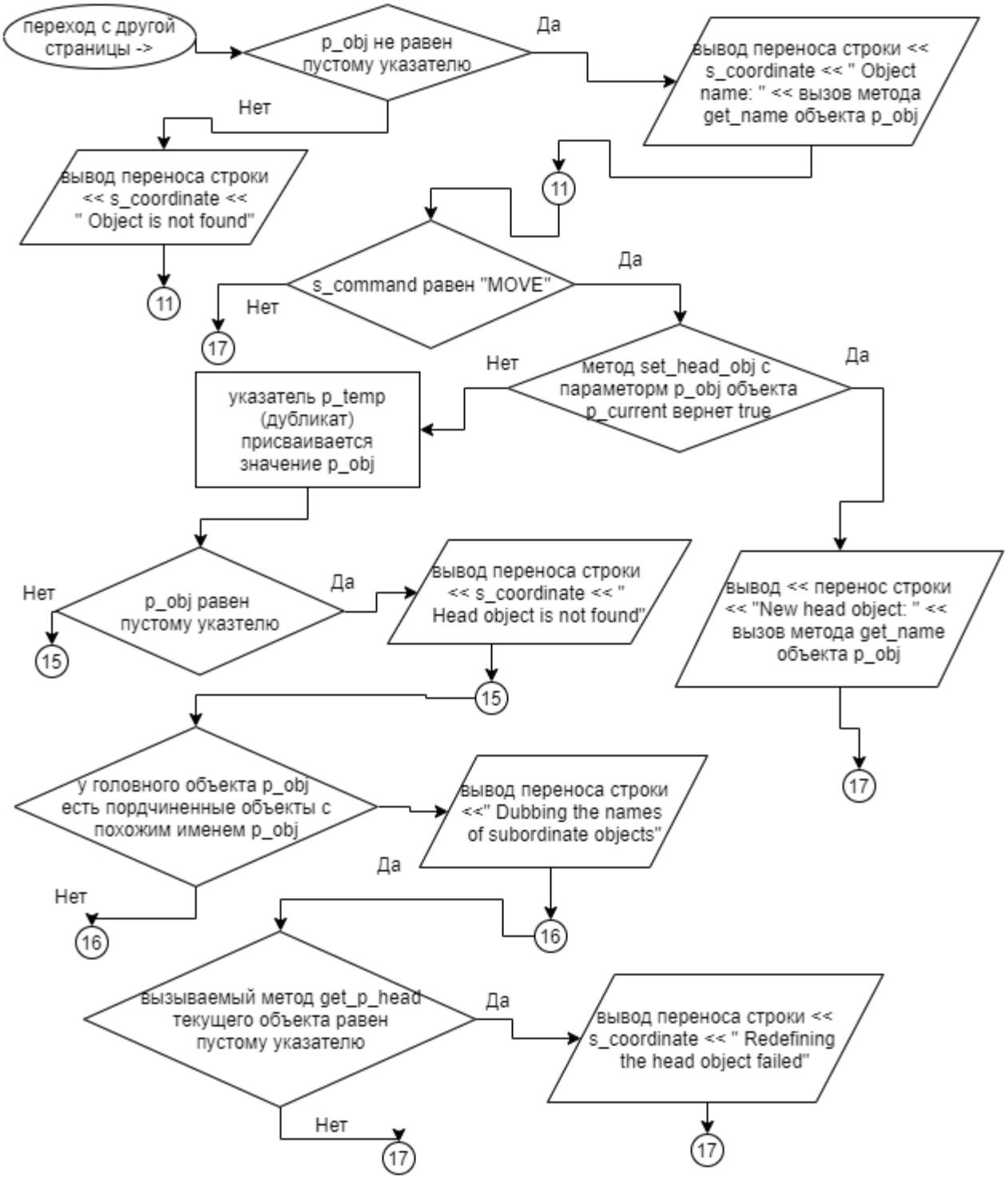

Рисунок 6 – Блок-схема алгоритма

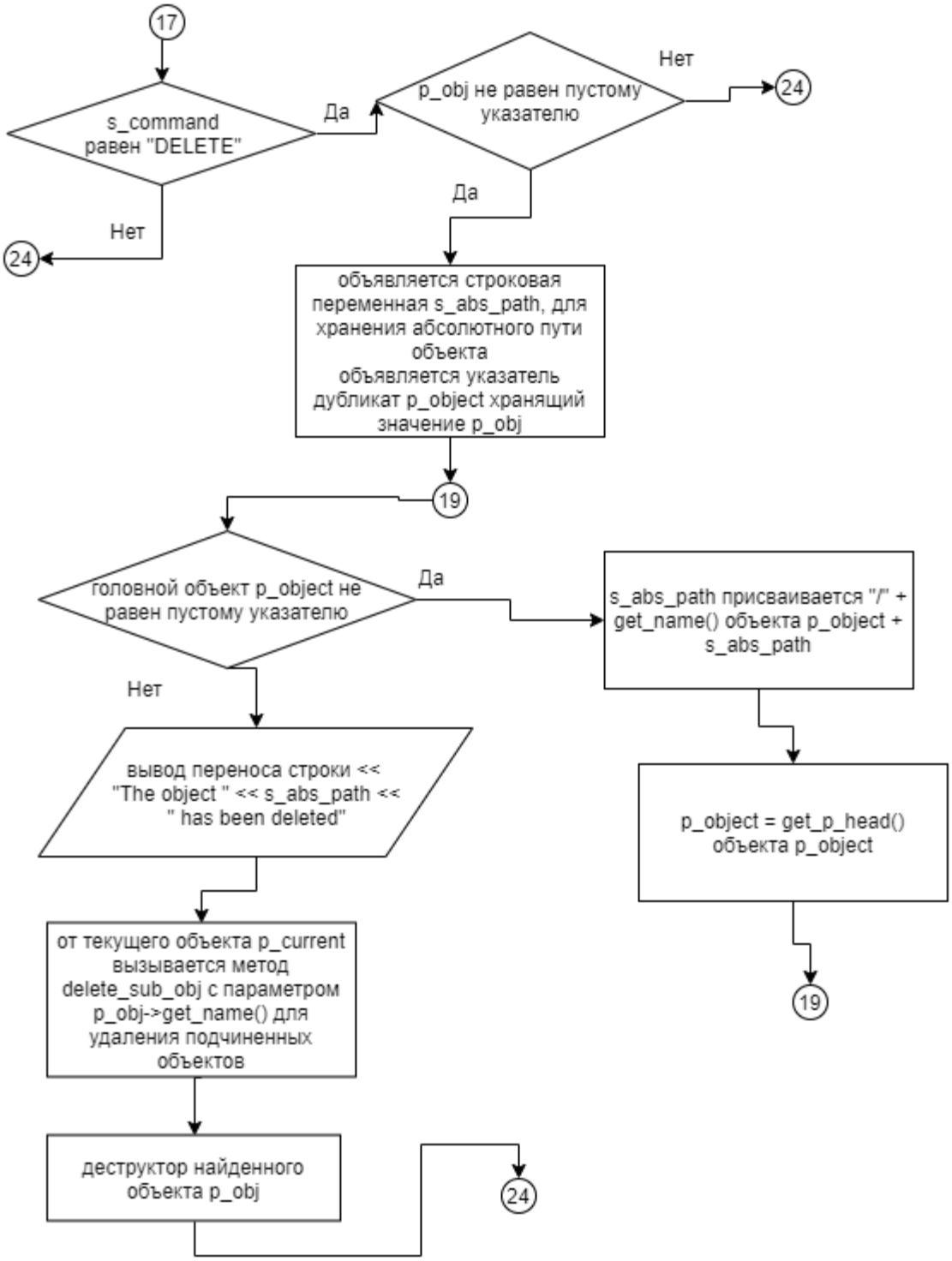

Рисунок 7 – Блок-схема алгоритма

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"

// вызов параметризированного конструктора класса cl_base с параметрами p_head_object и s_object_name
cl_2::cl_2(cl_base*      p_head_object,      string      s_object_name)      : cl_base(p_head_object, s_object_name){}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H

#include "cl_base.h"


class cl_2: public cl_base{
public:
   // конструктор, создающий объект класса cl_2
   cl_2(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
```

```
// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_3::cl_3(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_name){}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H

#include "cl_base.h"


class cl_3: public cl_base{
public:
   // конструктор, создающий объект класса cl_3
   cl_3(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_4::cl_4(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_name){}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H

#include "cl_base.h"


class cl_4: public cl_base{
public:
   // конструктор, создающий объект класса cl_4
   cl_4(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_5::cl_5(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_name){}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H

#include "cl_base.h"


class cl_5: public cl_base{
public:
   // конструктор, создающий объект класса cl_5
   cl_5(cl_base * p_head_object, string s_object_name);
```

```
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_6::cl_6(cl_base*     p_head_object,     string     s_object_name)     :
cl_base(p_head_object, s_object_name){}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H

#include "cl_base.h"


class cl_6: public cl_base{
public:
   // конструктор, создающий объект класса cl_6
   cl_6(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"

// вызов параметризированного конструктора класса cl_base с параметром
```

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


void cl_application::build_tree_objects(){
   /*
   метод построения дерева иерархии объектов
   */
   cout << "Object tree";

   string s_head_name, s_sub_name; // имя головного объекта, подчиненного
объекта
   cl_base* p_head;
   cl_base* p_sub = nullptr;
   int class_number, object_state; // номер класса, номер состояния

   cin >> s_head_name;
   // вызов метода set_name этого объекта с параметром s_head_name
   this -> set_name(s_head_name);
   // присваиваем p_head этот объект
   p_head = this;

   cin >> s_head_name;
   // ввод иерархии объектов
   while(s_head_name != "endtree"){
      cin >> s_sub_name >> class_number;
      //поиск головного объекта
      p_head = find_obj_by_coord(s_head_name); // кв3
      // если головной объект ненулевой
      if (p_head == nullptr){
         print_tree();
         cout << endl << "The head object " << s_head_name << " is not
found";
         exit(1);
      }
      else{
         // далее - существует единсвенный головной объект p_head и
подчиненного с таким именем не существует
         switch(class_number){
            case 2:
               p_sub = new cl_2(p_head, s_sub_name);
               break;
            case 3:
               p_sub = new cl_3(p_head, s_sub_name);
               break;
            case 4:
               p_sub = new cl_4(p_head, s_sub_name);
               break;
            case 5:
               p_sub = new cl_5(p_head, s_sub_name);
               break;
            case 6:
               p_sub = new cl_6(p_head, s_sub_name);
               break;
```

```
            }
         }
         cin >> s_head_name;
      }
      //вызов метода print_tree этого объекта
      this->print_tree();

      string s_command, s_coordinate; // команда, координата
      cl_base* p_current = this; // указатель на текущий объект (головной объект
   класса cl_application)
      cl_base* p_obj;




      cin >> s_command;
      while(s_command != "END"){


         cin >> s_coordinate;
         p_obj = p_current->find_obj_by_coord(s_coordinate); // ищем переданный
   объект по координатам

         if(s_command == "SET"){
            // если объект явл. ненулевым указателем (т.е нашелся)
            if (p_obj != nullptr){
               p_current = p_obj; // текущему объекту присваиваем найденный
   объект
               cout << endl << "Object is set: " << p_current->get_name();
            }
            else{
               cout << endl << "The object was not found at the specified
   coordinate: " << s_coordinate;
            }
         }
         else if(s_command == "FIND"){
            // если объект явл. ненулевым указателем (т.е нашелся)
            if (p_obj != nullptr){
               cout << endl << s_coordinate << "      Object name: " << p_obj-
   >get_name();
            }
            else{
               cout << endl << s_coordinate << "     Object is not found";
            }
         }
         // смена головного объекта
         else if(s_command == "MOVE"){
            // если метод, вызванный через текущий объект с параметром
   найденного объекта вернет true
            if (p_current->set_head_obj(p_obj)){
               cout << endl << "New head object: " << p_obj->get_name();
            }
            else{
               // если головной элемент равен нулевому указателю
```

```
            if (p_obj == nullptr){
               cout << endl << s_coordinate << "       Head object is not
found";
            }
            // если у нового головного объекта есть подчиненные с одним
именем
            //                                    (this->)
            if  (p_obj->  get_subordinate_object(p_current->  get_name())  !=
nullptr){
               cout << endl << s_coordinate << "      Dubbing the names of
subordinate objects";
            }
            //        (this-> замена)
            // если корневой элемент равен нулевому указателю
            else if (this->get_p_head() == nullptr){
               cout << endl << s_coordinate << "       Redefining the head
object failed";
            }
         }
      }
      else if(s_command == "DELETE"){
         // если p_obj не равен пустому указателю, то мы нашли объект,
который нужно удалить => он существует
         if (p_obj != nullptr){
            string s_abs_path = ""; // абсолютный путь объекта
            cl_base* p_object = p_obj;
            // поднимаемся по дереву, пока головной объект не равен пустому
указателю
            while (p_object -> get_p_head() != nullptr){
               s_abs_path  =  "/"  +  p_object->get_name()  +  s_abs_path;  //
составляем путь объекта
               p_object = p_object->get_p_head();
            }
            cout  <<  endl  <<  "The  object  "  <<  s_abs_path  <<  "  has  been
deleted";
            //                         p_obj->get_name()  s_coordinate
            p_current->  delete_sub_obj(p_obj->get_name());  //  от  текущего
объекта удаляется подчиненные объекты p_object->get_name()
            delete p_obj; // деструктор найденного объекта
         }
      }
      cin >> s_command;
   }
}

int cl_application::exec_app(){

   cout << endl << "Current object hierarchy tree";
   this->print_tree();
   return 0;
}
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H

#include "cl_base.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"


class cl_application : public cl_base{
public:
   cl_application(cl_base * p_head_object); // конструктор, создающий объект
класса cl_app..
   int exec_app(); // метод запуска приложения

   void build_tree_objects(); // метод, создающий иерархию объекта
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base * p_head_object, string s_object_name)
: p_head_object(p_head_object) , s_object_name(s_object_name){
   /*
   параметризированный конструктор
   p_head_object - указатель на головной объект
   s_object_name - имя узла дерева
   */

   //полю p_head_object этого объекта присваивается параметр p_head_object
   //полю s_object_name этого объекта присваивается параметр s_object_name

   // если p_head_object ненулевой, то в subordinate_objects добавляется
указатель на этот объект
   if (p_head_object){
      p_head_object -> subordinate_objects.push_back(this);
   }
```

```
}
cl_base::~cl_base(){
   // проходимся по каждому элементу subordinate_objects и удаляем его
   for   (auto p_sub: subordinate_objects){
      delete p_sub;
   }
}

bool cl_base::set_name(string new_name){
   /*
   метод редактирования имени объекта
   new_name - новое имя узла дерева
   */
   // проходимся по каждому элементу вектора указателей subordinate_objects
объекта по указателю p_head_object
   // если он равен new_name, то возвращаем false
   if (p_head_object != nullptr){
      for (auto p_sub: p_head_object -> subordinate_objects){
         if (p_sub-> get_name() == new_name){
            return false;
         }
      }
   }
   // полю s_object_name этого объекта присваивается new_name
   this -> s_object_name = new_name;
   return true;
}

string cl_base::get_name(){
   // возвращаем s_object_name
   return this->s_object_name;
}

cl_base * cl_base::get_p_head(){
   // возвращаем p_head_object
   return this->p_head_object;
}

cl_base * cl_base::get_subordinate_object(string search_name){
   /*
   получение указателя на непосредственно подчиненный объект по имени
   search_name - имя искомого объекта
   */

   // проходимся по элементам subordinate_objects, если он равен search_name
   // возвращаем i-ый subordinate_objects
   for (auto p_sub: subordinate_objects){
      if (p_sub -> get_name() == search_name){
         return p_sub;
      }
   }
   // иначе возвращается нулевой указатель
   return nullptr;
}
```

```
cl_base* cl_base::search_object_from_current(string s_name){
   /*
   метод поиска объекта по имене в поддереве (в ветке) (обход графа в ширину)
   s_name - имя искомого объекта
   */
   cl_base* p_found = nullptr; // указатель на объект, который был найден
   queue<cl_base*> q; // очередь элементов

   q.push(this); // добавляем в очередь текущий элемент

   // пока очередь не пустая
   while(!q.empty()){
      cl_base* p_front = q.front(); // хранится указатель на наш объект

      q.pop(); // удаляем элемент из начала очереди, чтобы пройтись по всем
элементам
      if (p_front -> get_name() == s_name){ // если имя указателя на элемент
в очереди совпадает с имкомым объектом
         if (p_found     ==  nullptr)  //  указатель  не  пустой  и  объект  не
уникальный
            p_found = p_front; // присваиваем указатель на элемент в очереди
         else // нашли дубликат
            return nullptr;
      }
      // добавляем дочерние элементы p_front в очередь
      for (auto p_sub : p_front->subordinate_objects){
         q.push(p_sub);
      }
   }
   return p_found;
}

cl_base* cl_base::search_object_from_tree(string s_name){
   /*
   поиск объекта по имене во всем дереве
   s_name - имя искомого объекта
   */

   cl_base* p_root = this;
   // пока вышестоящий объект в дереве не пустой, поднимаемся по дереву
   while (p_root-> get_p_head() != nullptr){
      p_root = p_root-> get_p_head();
   }

   return p_root -> search_object_from_current(s_name);
}
void cl_base::print_tree(int layer){
   /*
   метод вывода иерархии объектов (дерева/ветки) от текущего объекта
   layer - уровень на дереве иерархии
   */

   // выводим layer-ое кол-во ' ' и имя объекта
```

```
   cout << endl << string(layer, ' ') << this->get_name();
   // проходимся по элемента subordinate_objects и вызываем рекурсию
   for (auto p_sub : subordinate_objects)
      p_sub->print_tree(layer + 4);

}

void cl_base::print_status_tree(int layer){
   /*
   метод вывода дерева/ветки иерархии объектов и их статуса от текущего
объекта
   layer - уровень на дереве иерархии
   */
   cout << endl;
   for(int i =0; i < layer; ++i){
      cout << "    ";
   }
   // проверка на статус текущего объекта
   if (this-> status!=0){
      cout << this -> get_name() << " is ready";
   }
   else{
      cout << this -> get_name() << " is not ready";
   }
   for (int i = 0; i < subordinate_objects.size(); ++i){
      subordinate_objects[i] -> print_status_tree(layer + 1);
   }
}
void cl_base::set_status(int status){
   /*
   метод установки статуса объекта
   status - номер состояния
   */

   // если значение status ненулевое у головного объекта
   if (p_head_object == nullptr || p_head_object -> status != 0){
      this -> status = status;
   }
   if (status == 0){
      this ->status = status;
      for (int i = 0; i < subordinate_objects.size(); i++){
         subordinate_objects[i] -> set_status(status);
      }
   }
}

cl_base* cl_base::find_obj_by_coord(string s_coord){
   /*
   метод поиска объекта по координате
   s_coord - координата искомого объекта
   */

   cl_base* p_root = this; // указатель на текущий объект
   int i_slash_2 = 0; // хранит индекс 2-ого слэша
   string s_name = "";
```

```
   cl_base* p_obj;

   if (s_coord == "/"){
      //поднимаемся по корню дереву
      while(p_root -> get_p_head() != nullptr){ // пока головной объект не
равен пустому указателю
         p_root = p_root -> get_p_head();
      }
      return p_root; // нашли корневой объект
   }
   if (s_coord == "."){
      return this; // вовзращаем текущий объект, т.к. считаем, что метод
вызван от текущего объекта
   }
   if (s_coord[0] == '/' && s_coord[1] == '/'){ // второй символ строки равен
/ (/ '/' )
      return  this->search_object_from_tree(s_coord.substr(2));  //  текущему
объекту вызывваем метод с парам. s_coord (начиная с 3 символа)
   }
   if (s_coord[0] == '.'){
      return this->search_object_from_current(s_coord.substr(1)); // текущему
объекту вызывваем метод с парам. s_coord (начиная со 2 символа)
   }

   i_slash_2 = s_coord.find("/",1);
   if (s_coord[0] == '/'){
      //поднимаемся по корню дереву
      while(p_root -> get_p_head() != nullptr){ // пока головной объект не
равен пустому указателю
         p_root = p_root -> get_p_head();
      }

      if (i_slash_2 != -1){ // нашли индекс слэша
         s_name = s_coord.substr(1, i_slash_2 - 1); // получили имя объекта,
после слэша и до след. слэша
         p_obj  =  p_root  ->  get_subordinate_object(s_name);  //  корневому
объекту вызываем метод поиска подчиненных объектов с парам. s_name
         if (p_obj != nullptr) // если p_obj не равен нулевому указателю
            return p_obj->find_obj_by_coord(s_coord.substr(i_slash_2 + 1));
         else
            return p_obj;
      }
      else{
         s_name = s_coord.substr(1);
         return   p_root->get_subordinate_object(s_name);   //   от   корневого
объекта ищем подчиненные
      }
   } // ob1/ob2/ob3
   else{
      if (i_slash_2 != -1){ // если слэш был найден в s_coord
         s_name = s_coord.substr(0, i_slash_2); // получили имя объекта
         p_obj = this -> get_subordinate_object(s_name); // текущему объекту
вызываем метод с парам. s_name
         if (p_obj != nullptr) // если p_obj не равен нулевому указателю
            return     p_obj->find_obj_by_coord(s_coord.substr(i_slash_2     +
```

```
1)); // рекурсивно вызываем этот метод, передавая строку, после / в s_coord
         else
            return p_obj;
      }
      else{ // второго слэша не нашлось (ob3)
         return this->get_subordinate_object(s_coord); // от текущего объекта
ищем подчиненные
      }
   }

   return nullptr;
}
void cl_base::delete_sub_obj(string s_name){
   /*
   метод удаления подчиненного объекта по имени
   s_name - имя удаляемого объекта
   */

   // проходимся по подчиненным элементам
   for (int i = subordinate_objects.size() - 1; i>=0; --i){
      if (subordinate_objects[i]-> get_name() == s_name){ // если подчиненный
элемент равен имени удаляемого объекта
         //delete subordinate_objects[i];
         subordinate_objects.erase(subordinate_objects.begin() + i);
         break;
      }
   }
}
bool cl_base::set_head_obj(cl_base* p_new_head){
   /*
   метод смены головного элемента
   p_new_head - имя нового головного объекта
   */

   cl_base* p_temp = p_new_head; // указатель

   // если головной элемент равен нулевому указателю
   if (p_new_head == nullptr){
      return false;
   }
   // если корневой элемент равен нулевому указателю
   if (this->get_p_head() == nullptr){
      return false;
   }
   // если у нового головного объекта есть подчиненные с одним именем
   if (p_new_head-> get_subordinate_object(this-> get_name()) != nullptr){
      return false;
   }
   // поднимаеся по дереву от нашего нового головного объекта, пока не найдем
до корня
   while (p_temp != nullptr){
      if (p_temp == this){ // если p_temp равен текущему объекту (является
подчиненным - недопустимо)
         return false;
      }
```

```
            else
               p_temp = p_temp -> get_p_head();
         }
         // головной объект удалили из подчиненных у текущего головного объекта
         this -> get_p_head()->delete_sub_obj(this->get_name());
         // присваиваем имя нового головного объекта

         // новому головному объекту добавляем в подчиненные объекты текущий объект
         p_new_head->subordinate_objects.push_back(this);
         return true;
      }
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
      #ifndef __CL_BASE__H
      #define __CL_BASE__H

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

      using namespace std;

      class cl_base {
      private:
         string s_object_name; // имя объекта
         cl_base * p_head_object; // указатель на родительский объект
         vector <cl_base *> subordinate_objects; // вектор подчиненных объектов
         int status = 0; // статус состояния объекта
      public:
         cl_base(cl_base * p_head_object, string s_object_name = "Base object"); //
      параметризированный конструктор
         string get_name(); // метод получения имени
         cl_base * get_p_head(); // метод получения указателя на родительский
      объект
         bool set_name(string new_name); //метод редактирования имени объекта
         cl_base * get_subordinate_object(string search_name); // получение
      указателя на непосредственно подчиненный объект по имени
         ~cl_base(); // деструктор
         cl_base* search_object_from_current(string); // поиск объекта на ветке
      иерархии от текущего по имени
         cl_base* search_object_from_tree(string); // поиск по дереву (в корне)
      иерархии
         void set_status(int status); // метод установки статуса объекта
         void print_tree(int layer = 0); // метод вывода дерева иерархии
         void print_status_tree(int layer = 0); // метод вывода дерева/ветки
      иерархии объектов и их статуса от текущего объекта
```

```
   cl_base* find_obj_by_coord(string); // метод поиска объекта по заданной
координате
   void delete_sub_obj(string); // метод удаления подчиненного объекта по
имени
   bool set_head_obj(cl_base*); // метод смены головного элемента
};



#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_application.h"

int main()
{
   cl_application ob_cl_application(nullptr); // создание корневого объекта
   ob_cl_application.build_tree_objects();   //   конструирование   системы,
построение дерева иерархии

   return (ob_cl_application.exec_app()); // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 6.

Таблица 6 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4
object_7 3
endtree
FIND
object_2/object_4
SET /object_2
FIND //object_7
FIND
object_4/object_7
FIND .
FIND .object_7
FIND
object_4/object_7
MOVE .object_7
SET
object_4/object_7
MOVE //object_1
MOVE /object_3
END
	Object tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_7
	object_5
	object_3
	object_3
	object_2/object_4
	Object name:
	object_4
	Object is set:
	object_2
	//object_7
	Object is not found
	object_4/object_7
	Object name:
	object_7
	. Object name:
	object_2
	.object_7 Object
	name: object_7
	object_4/object_7
	Object name:
	object_7
	.object_7
	Redefining the head
	object failed
	Object is set:
	object_7
	//object_1
	Dubbing the names of
	subordinate objects
	New head object:
	object_3
	Current object
	hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_5
		Object tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_7
		object_5
		object_3
		object_3
		object_2/object_4
		Object name:
		object_4
		Object is set:
		object_2
		//object_7
		Object is not found
		object_4/object_7
		Object name:
		object_7
		. Object name:
		object_2
		.object_7 Object
		name: object_7
		object_4/object_7
		Object name:
		object_7
		.object_7
		Redefining the head
		object failed
		Object is set:
		object_7
		//object_1
		Dubbing the names of
		subordinate objects
		New head object:
		object_3
		Current object
		hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_5

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_3 object_3 object_7	object_3 object_3 object_7
rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree DELETE /object_2 END	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 The object /object_2 has been deleted Current object hierarchy tree rootela object_1 object_7 object_3	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 The object /object_2 has been deleted Current object hierarchy tree rootela object_1 object_7 object_3
rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 FIND //object_7 SET /object_2 FIND //object_7 Find object_4/Object_7 FIND /./././. FIND ./././. FIND /// FIND .object_7 FIND object_4/object_7 MOVE .object_7 SET object_4/object_7 MOVE //object_1 MOVE /object_3	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_2/object_4 Object name: object_4 //object_7 Object is not found Object is set: object_2 //object_7 Object is not found /./././. Object is not found ././. Object is not found /// Object is not found .object_7 Object name: object_7 object_4/object_7 Object name: 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_7 Object is not found Object is set: object_2 //object_7 Object is not found /./././. Object is not found ././. Object is not found /// Object is not found .object_7 Object name: object_7 object_4/object_7 Object name:

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
SET /object_2 DELETE object_5 DELETE /object_7 END	.object_7 Redefining the head object failed Object is set: object_7 //object_1 Dubbing the names of subordinate objects New head object: object_3 Object is set: object_2 The object /object_2/object_5 has been deleted Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_3 object_3 object_7	object_7 .object_7 Redefining the head object failed Object is set: object_7 //object_1 Dubbing the names of subordinate objects New head object: object_3 Object is set: object_2 The object /object_2/object_5 has been deleted Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_3 object_3 object_7

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
r / a 2 / b 3 /a b 3 /a/b c 4 /a/b/c a 2 /a/b/c/a a 2 /a/b/c/a b 3 endtree SET /a MOVE /b END	Object tree r a b c a a b b Object is set: a New head object: b Current object hierarchy tree r b a b c a a b	Object tree r a b c a a b b Object is set: a New head object: b Current object hierarchy tree r b a b c a a b

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).