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:

- find - Функция из стандартного пространста имён поиска значения в векторе.

Класс cl_base:

- функционал:
  - метод change_header — Изменение головного объекта;
  - метод delete_by_name — Удаление подчинённого объекта по имени;
  - метод find_by_coord — Поиск объекта по координате;
  - метод find_on_branch — Поиск объект на ветке (уникальный).

Класс cl_application:

- функционал:
  - метод exec_app — Метод запуска работы системы и обработки вводимиых команд;
  - метод build_tree_objects — Метод построения дерева объектов системы.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс - элемент дерева объектов	
		cl_application	public		2
2	cl_application				

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 change_header класса cl_base

Функционал: Изменение головного объекта.

Параметры: cl_base* new_header - указатель на новывй головной объект.

Возвращаемое значение: bool - Успешность переопределения головного объекта.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода change_header класса cl_base

№	Предикат	Действия	№ перехода
1	Текущий объект равен результату вызова метода get_root или new_header равен нулевому указателю		2
	Результат вызова метода get_child у объекта по указателю new_header с параметром поле name текущего объекта не равен нулевому указателю		2
			3
2		Возврат false	∅
3		Инициализация указателя current типа cl_base* равному new_header	4

№	Предикат	Действия	№ перехода
4	Значение вызова метода get_hedaer у объекта current не равно нулевому указателю		5
			7
5	current = текущему объекту	Возврат false	∅
			6
6		current = Значение вызова метода get_hedaer у объекта current	4
7		Удаляем из вектора подчинённых объектов children_ptr родительского объекта header_ptr текущий объект с помощью метода erase и функции поиска find	8
8		Установка header_ptr равному new_header	9
9		Добавление текущего объекта в вектор подчинённых объектов childre_ptr объекта new_header с помощью метода push_back вектора	10
10		Возврат true	∅

3.2 Алгоритм метода delete_by_name класса cl_base

Функционал: Удаление подчинённого объекта по имени.

Параметры: string name - Наименование удаляемого объекта.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода delete_by_name класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация i = 0 типа int	2

№	Предикат	Действия	№ перехода
2	i < Рамзер вектора children_ptr		3
		Возврат	∅
3	Имя объекта в children_ptr i-й равно параметру name	Удаление элемента из children_ptr по индексу i и выход из функции	∅
			4
4		i++	2

3.3 Алгоритм метода find_by_coord класса cl_base

Функционал: Поиск объекта по координате.

Параметры: string coord - искомая координата.

Возвращаемое значение: cl_base* - указатель на найденный объект.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода find_by_coord класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление переменной s_name типа string	2
2	coord - пустая строка	Возврат нулевого указателя	∅
			3
3		Инициализация указателя root типа cl_base* указателем на текущий объект	5
4	Значение вызова get_header() у root не нулевой указатель	root = Значение вызова get_header() у root	4
			5
5	coord = "/"	Возврат root	∅
			6
6	Первые два символа coord = "//"	s_name = строка coord начиная со второго индекса	7

№	Предикат	Действия	№ перехода
			8
7		Возврат значения вызова метода find_on_tree с параметром s_name	∅
8	coord = "."	Возврат указателя на текущий объект	∅
			9
9	Первый символ coord = "."	s_name = строка coord начиная с первого индекса	10
			11
10		Возврат значения вызова метода find_on_branch с параметром s_name	∅
11		Инициализация указателя current типа cl_base* указателем на текущий объект	12
12	Первый символ coord = "/"	current = root	13
			14
13		coord = строка coord начиная с первого индекса	14
14	"/" есть в строке coord	Инициализация переменной nsl индексом символа "/" найденным в строке coord с помощью функции find	15
			18
15		current = значение вызова метода get_child у объекта current с параметром строки равной срезу строки coord с 0 по nsl индекс	16
16	current нулевой указатель	Возврат нулевого указателя	∅
			17
17		coord = срез строки coord с индекса nsl + 1	14
18		Возврат значения вызова метода get_child у объекта current с параметром coord	∅

3.4 Алгоритм метода exec_app класса cl_application

Функционал: Метод запуска работы системы и обработки вводимых команд.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода print	2
2		Инициализация указателя curent типа cl_base* указателем на текущий объект	3
3		Объявление переменных cmd, coord типа string	4
4		Ввод значения cmd	5
5	cmd = "END"		20
			6
6		Инициализация указателя p_obj типа cl_base* равному значению вызова метода find_by_coord с параметром coord у объекта current	7
7	cmd = "SET"		8
			9
8	p_obj не нулевой указатель	current = p_obj Вывод "Object is set: " Значение вызова метода get_name у объекта current	3
		Вывод "The object was not found at the specified coordinate: " значение coord	3
9	cmd = "FIND"		10
			11
10	p_obj не нулевой указатель	Вывод значение coord " Object name: " Значение	3

№	Предикат	Действия	№ перехода
		вызова метода get_name у объекта p_obj	
		Вывод значение coord " Object is not found"	3
11	cmd = "MOVE"		12
			13
12	p_obj нулевой указатель	Вывод значение coord " Head object is not found"	3
	значение вызова метода get_header у p_obj != значение вызова метода get_header у current && значение вызова метода find_on_branch у current с параметром значения вызова метода get_name у p_obj пустой указатель	Вывод значение coord " Redefining the head object failed"	3
	значение вызова метода find_on_branch у p_obj с параметром значения вызова метода get_name у current пустой указатель	Вывод значение coord " Dubbing the names of subordinate obects"	3
	значение вызова метода change_header у current с параметром p_obj пустой указатель	Вывод значение coord " Redefining the head object failed"	3
		Вывод "New head object: " значение вызова метода get_name у объекта p_obj	3
13	cmd = "DELETE"		14
			3
14	p_obj не нулевой указатель	Инициализация указателя fheader типа cl_base* значение вызова метода get_header у объекта p_obj	15

№	Предикат	Действия	№ перехода
			3
15		Объявление перемнной path типа string	16
16		Вызов метода delete_by_name у объекта current с параметром значения вызова функции get_name у объекта p_obj	17
17	fheader не нулевой указатель		18
			3
18	Значение вызова метода get_header у объекта fheader не нулевой указатель	path += "/" + значение вызова метода get_name у объекта fheader	19
			19
19		Вывод "The object " path + "/" + coord " has been deleted"	3
20		Вывод "Current object hierarchy tree"	21
21		Вызов метода print	22
22		Возврат значения 0	∅

3.5 Алгоритм метода build_tree_objects класса cl_application

Функционал: Метод построения дерева объектов системы.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Object Tree"	2
2		Объявление переменных parentCoord, childName типа string и childClass типа int	3

№	Предикат	Действия	№ перехода
3		Ввод значения parentCoord	4
4		Вызов метода edit_name с передачей параметра parentCoord	5
5		Ввод значения parentCoord	6
6	parentCoord = "endtree"		∅
			7
7		Инициализация указателя parent типа cl_base* значением вызова метода find_by_coord с передачей параметра parentCoord	8
8	parent нулеовй указатель	Вызов метода print	9
			11
9		Вывод "The head object " parentCoord " is not found"	10
10		Завершение программы с кодом 1	∅
11		Ввод значений childName и childClass	12
12	Значение вызова метода get_child у объекта parent с параметром childName - нулевой указатель		13
		Вывод "parentCoord Dubbing the names of subordinate objects"	5
13	childClass = 2	Выделение памяти объекту класса cl_obj2 с передачей парметров конструктору parent и childName	5
	childClass = 3	Выделение памяти объекту класса cl_obj3 с передачей парметров конструктору parent и childName	5
	childClass = 4	Выделение памяти объекту класса cl_obj4 с передачей парметров конструктору parent и childName	5
	childClass = 5	Выделение памяти объекту класса cl_obj5 с	5

№	Предикат	Действия	№ перехода
		передачей парметров конструктору parent и childName	
	childClass = 6	Выделение памяти объекту класса cl_obj6 с передачей парметров конструктору parent и childName	5
			5

3.6 Алгоритм функции main

Функционал: Главная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 7.

Таблица 7 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Алгоритм из предыдущего задания КВ-2	∅

3.7 Алгоритм метода find_on_branch класса cl_base

Функционал: Поиск объект на ветке (уникальный).

Параметры: string name - наименование искомого объекта, int* count - указатель на кол-во найденных объектов (nullptr по умолчанию).

Возвращаемое значение: cl_base* - указатель на найденный объект.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода find_on_branch класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация переменной c = 0 типа int	2
2	count нулевой указатель	count = указатель на c	3
			3
3		Инициализация указателя fchild равному нулевому указателю типа cl_base*	4
4	поле name = параметр name	fchild = указатель на текущий объект	5
			6
5		инкремент объекта по указателю count	6
6		Инициализация итератора child типа cl_base* по вектору children_ptr	7
7	child в children_ptr	Инициализация указателя f равному вызову метода find_on_branch у объекта child с параметрами name, count	8
			9
8	f не нулевой указатель	fchild = f	7
			7
9	объект по указателю count = 1 && fchild не нулевой указатель	Возврат fchild	∅
		Возврат nullptr	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

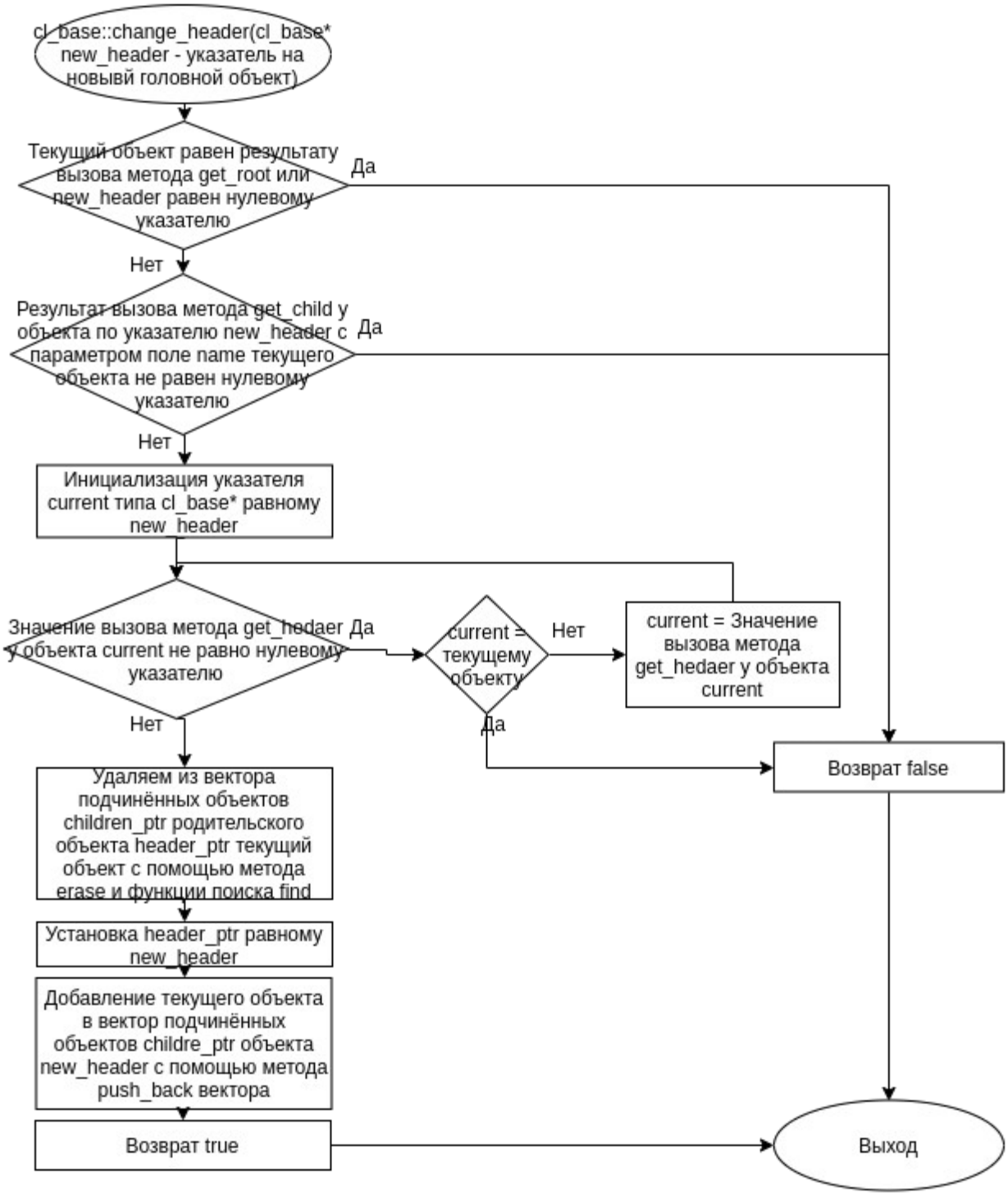

Рисунок 1 – Блок-схема алгоритма

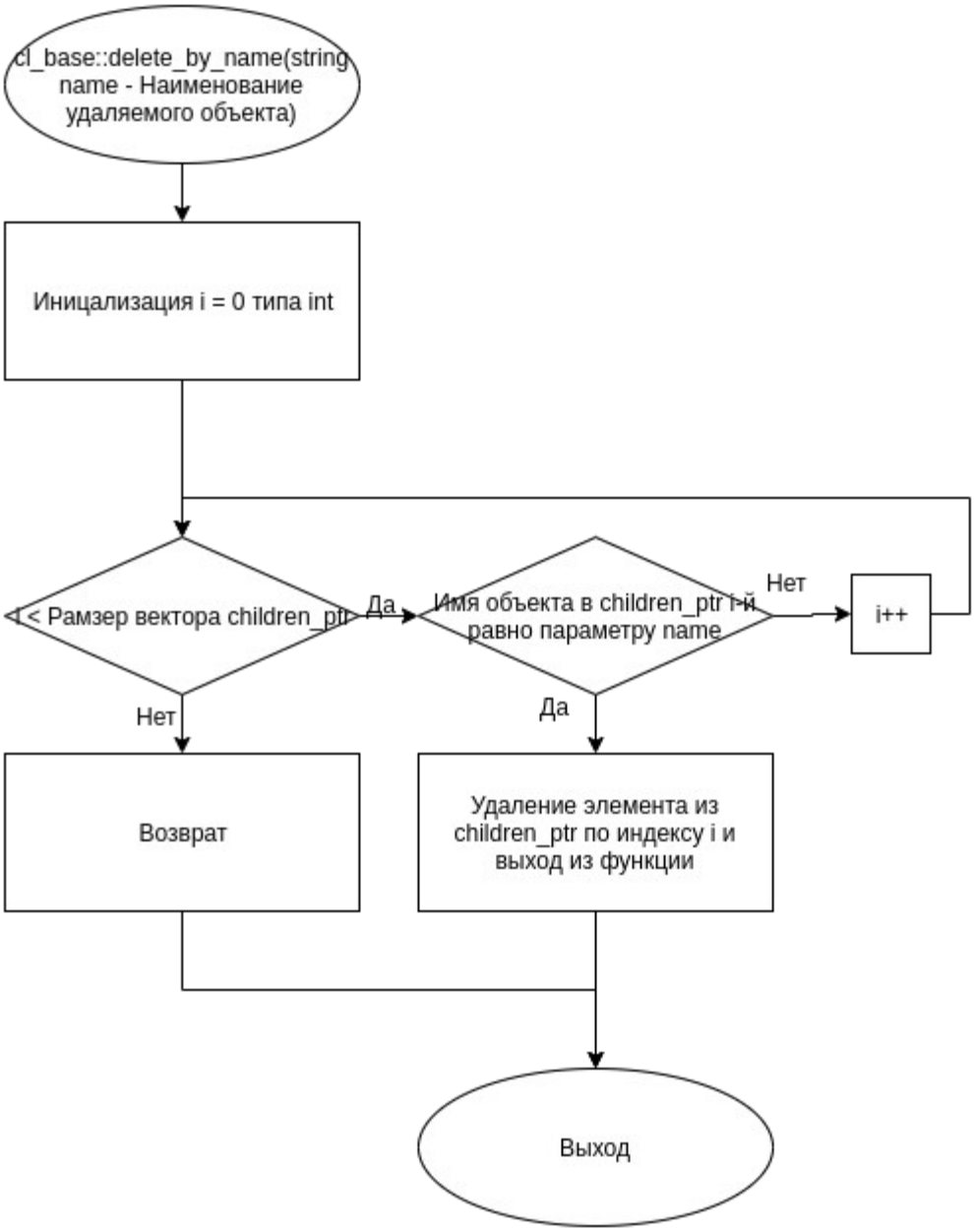

Рисунок 2 – Блок-схема алгоритма

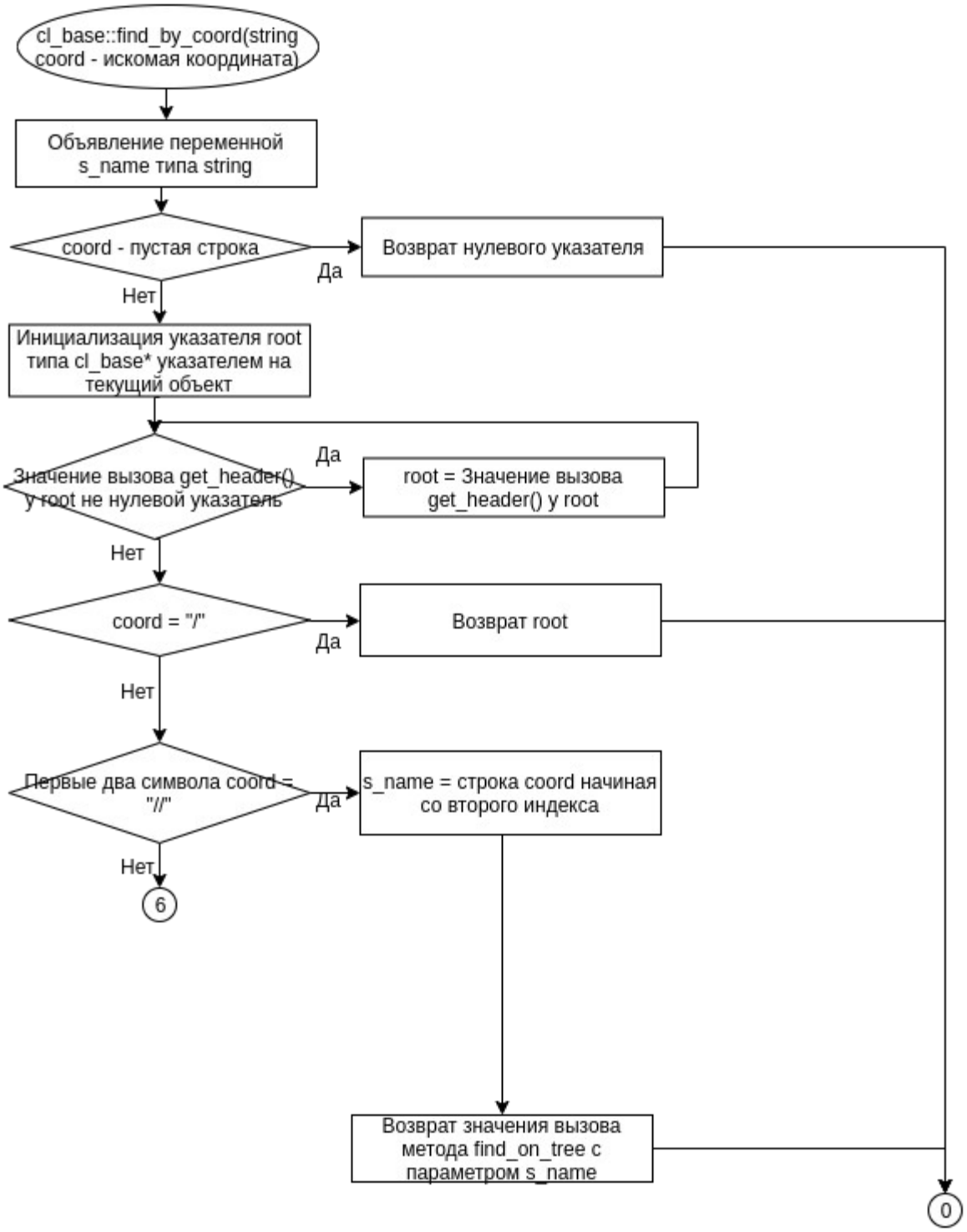

Рисунок 3 – Блок-схема алгоритма

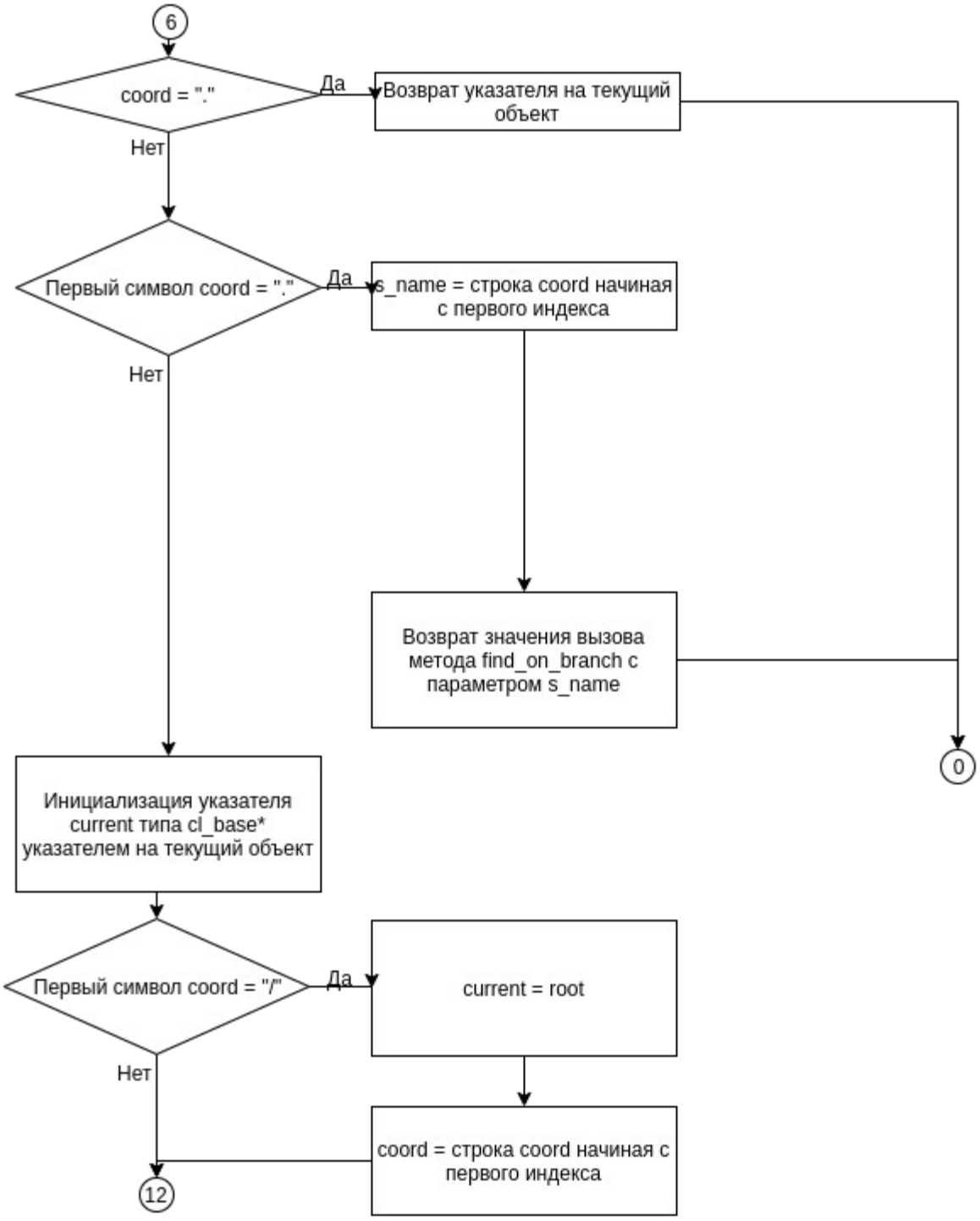

Рисунок 4 – Блок-схема алгоритма

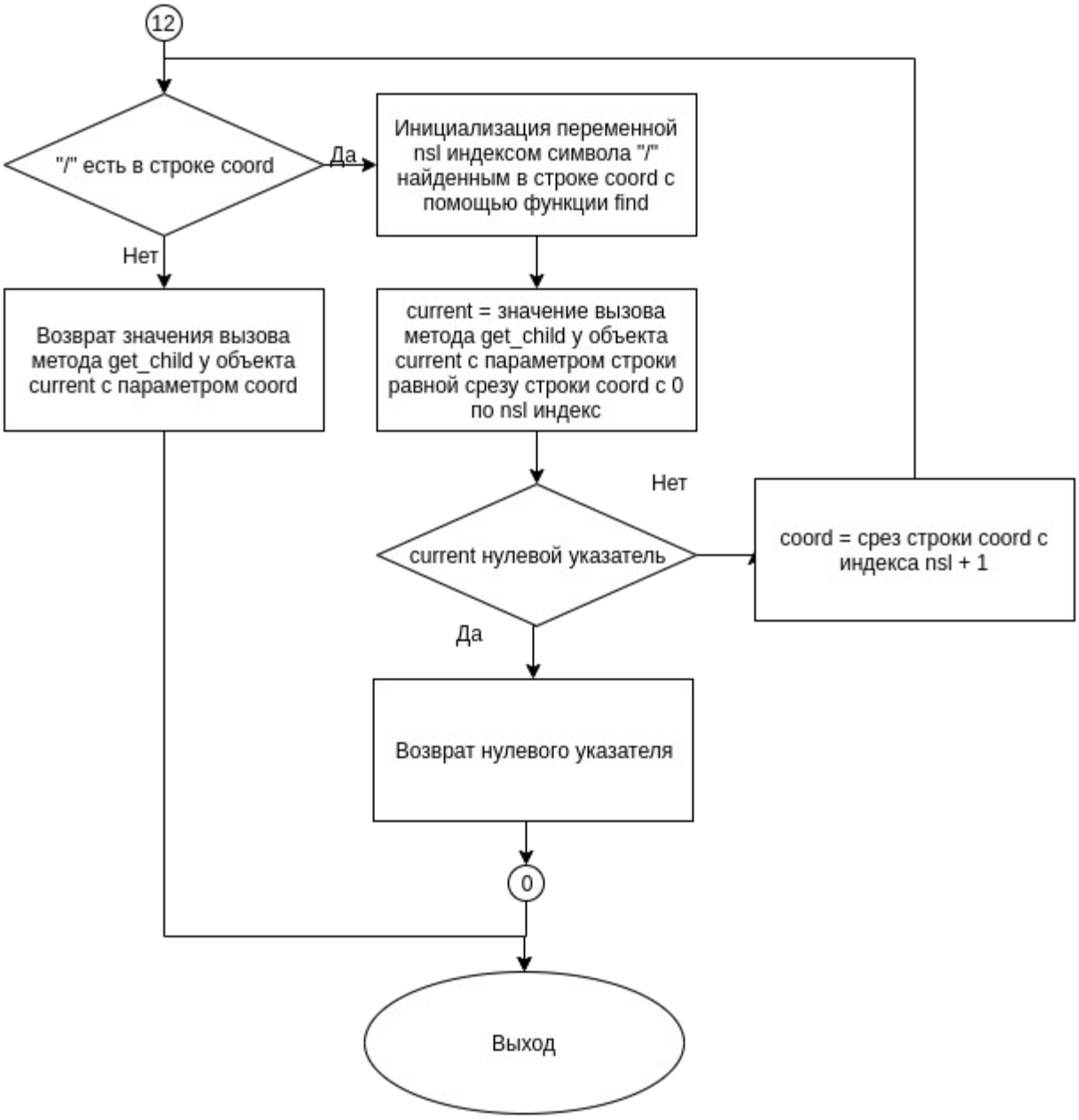

Рисунок 5 – Блок-схема алгоритма

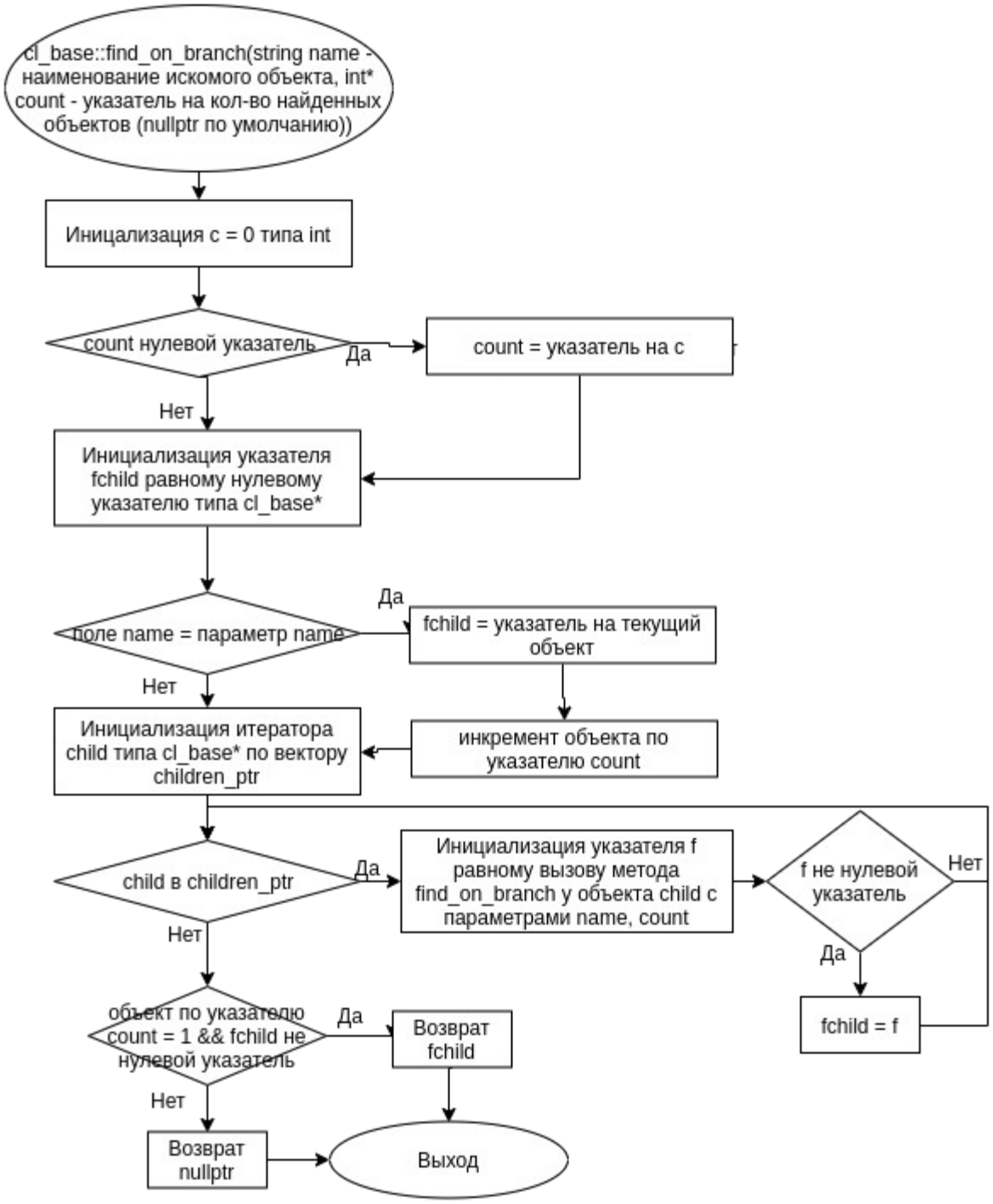

Рисунок 6 – Блок-схема алгоритма

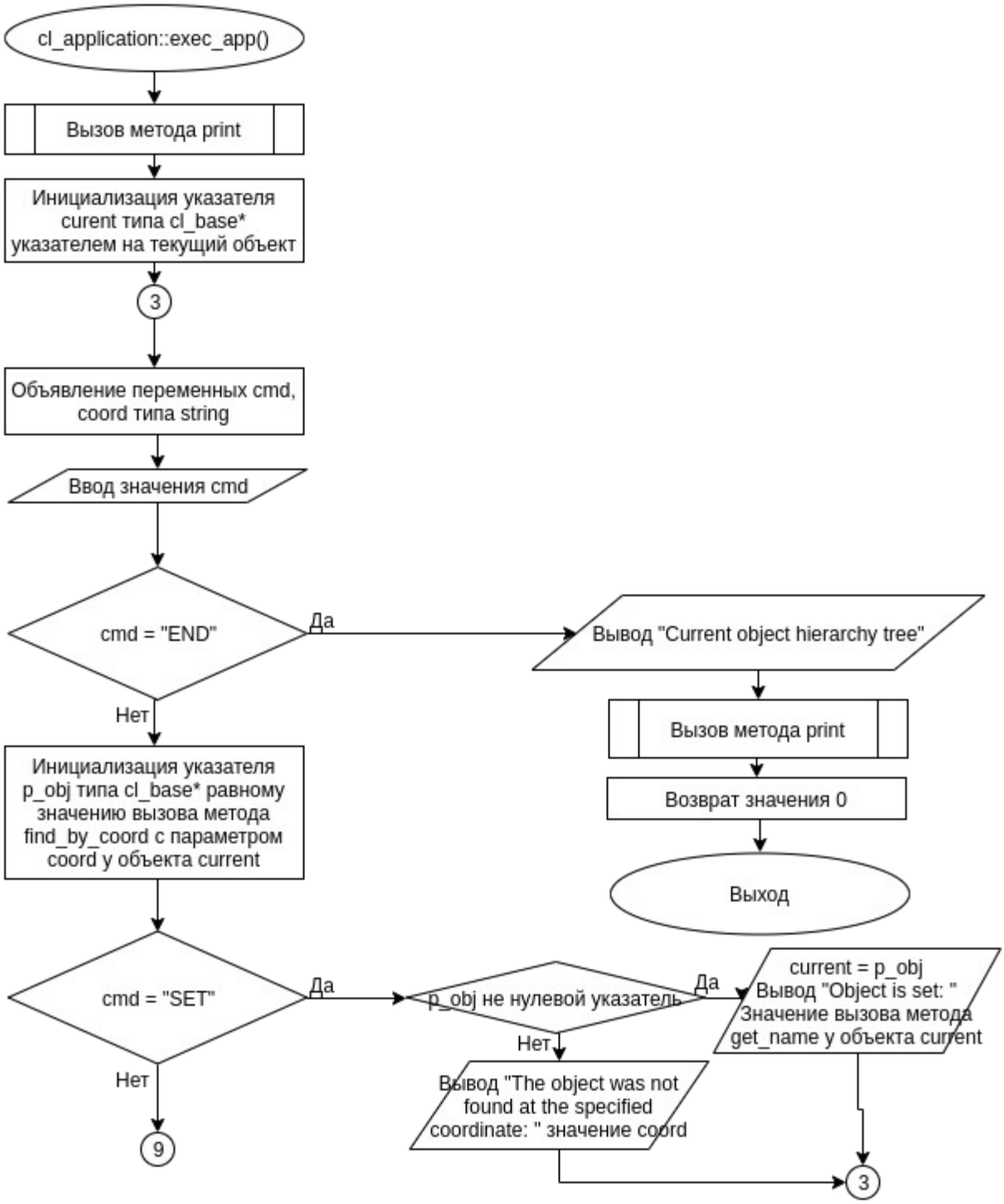

Рисунок 7 – Блок-схема алгоритма

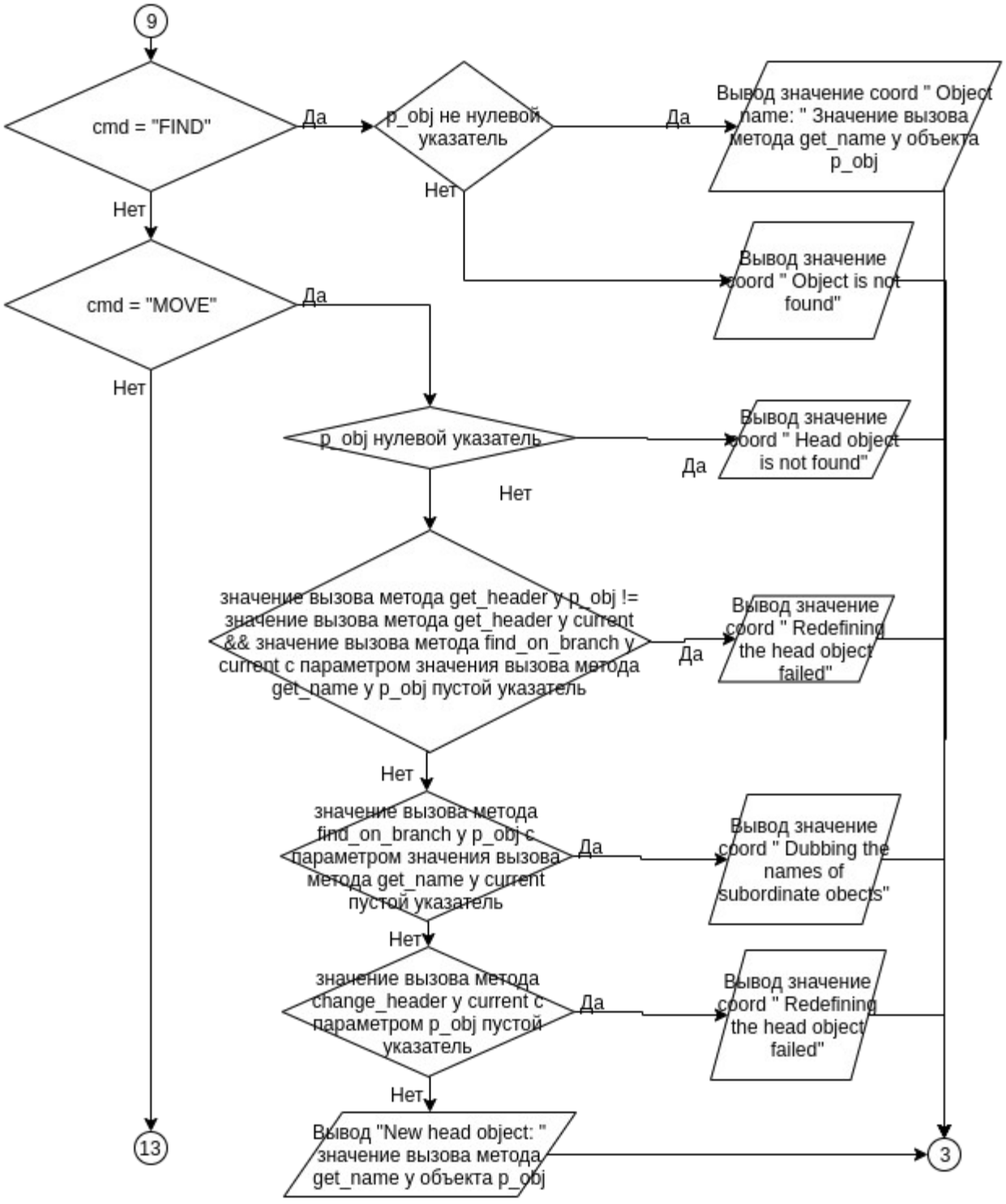

Рисунок 8 – Блок-схема алгоритма

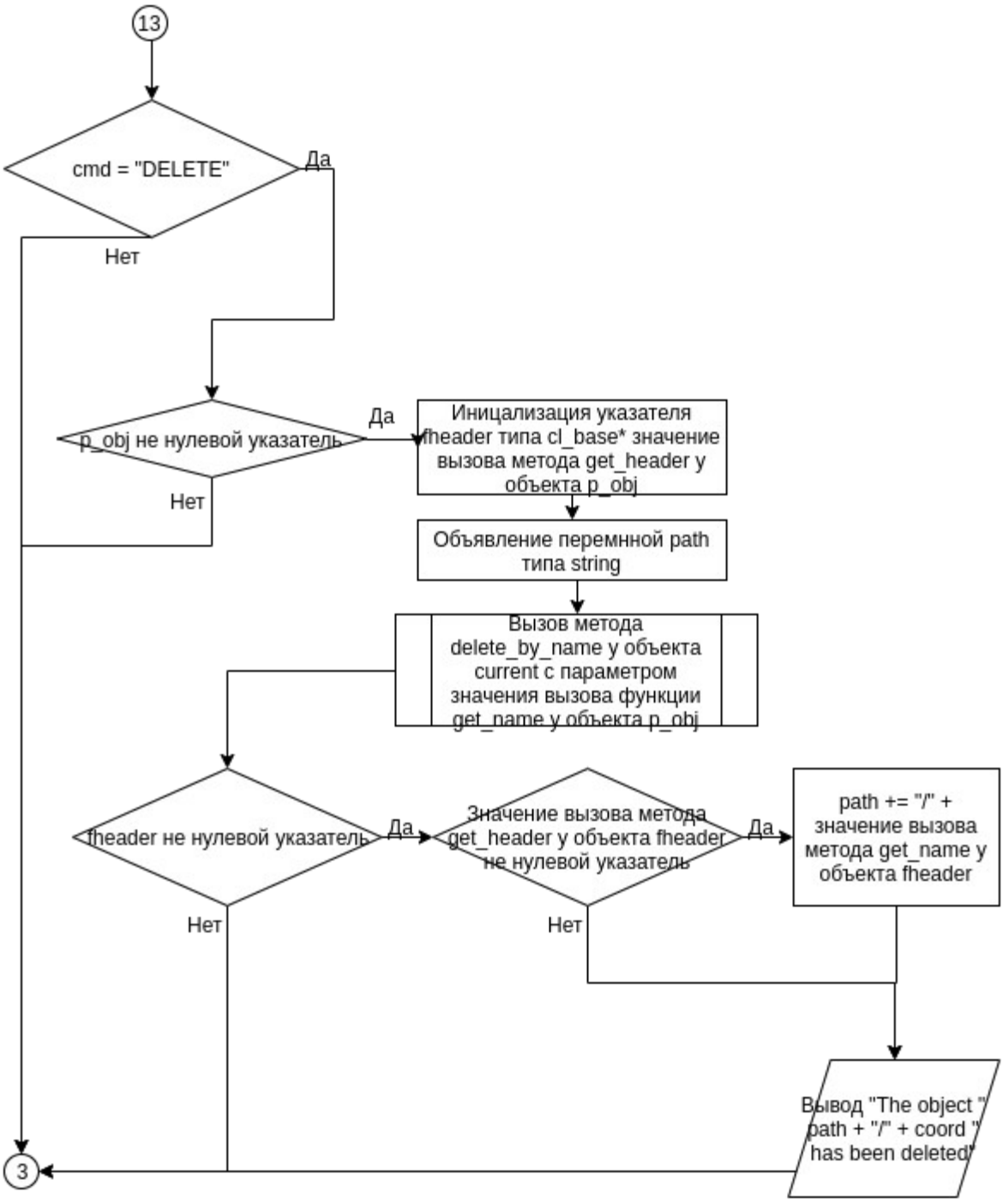

Рисунок 9 – Блок-схема алгоритма

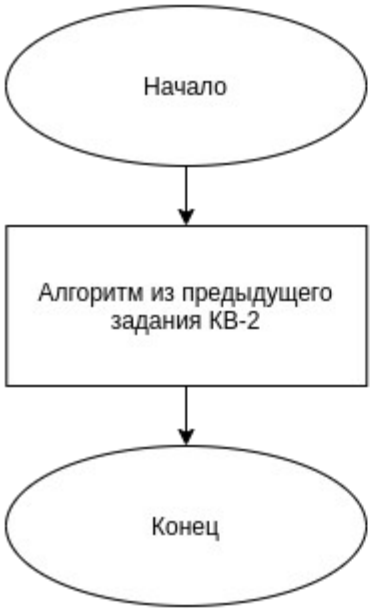

Рисунок 10 – Блок-схема алгоритма

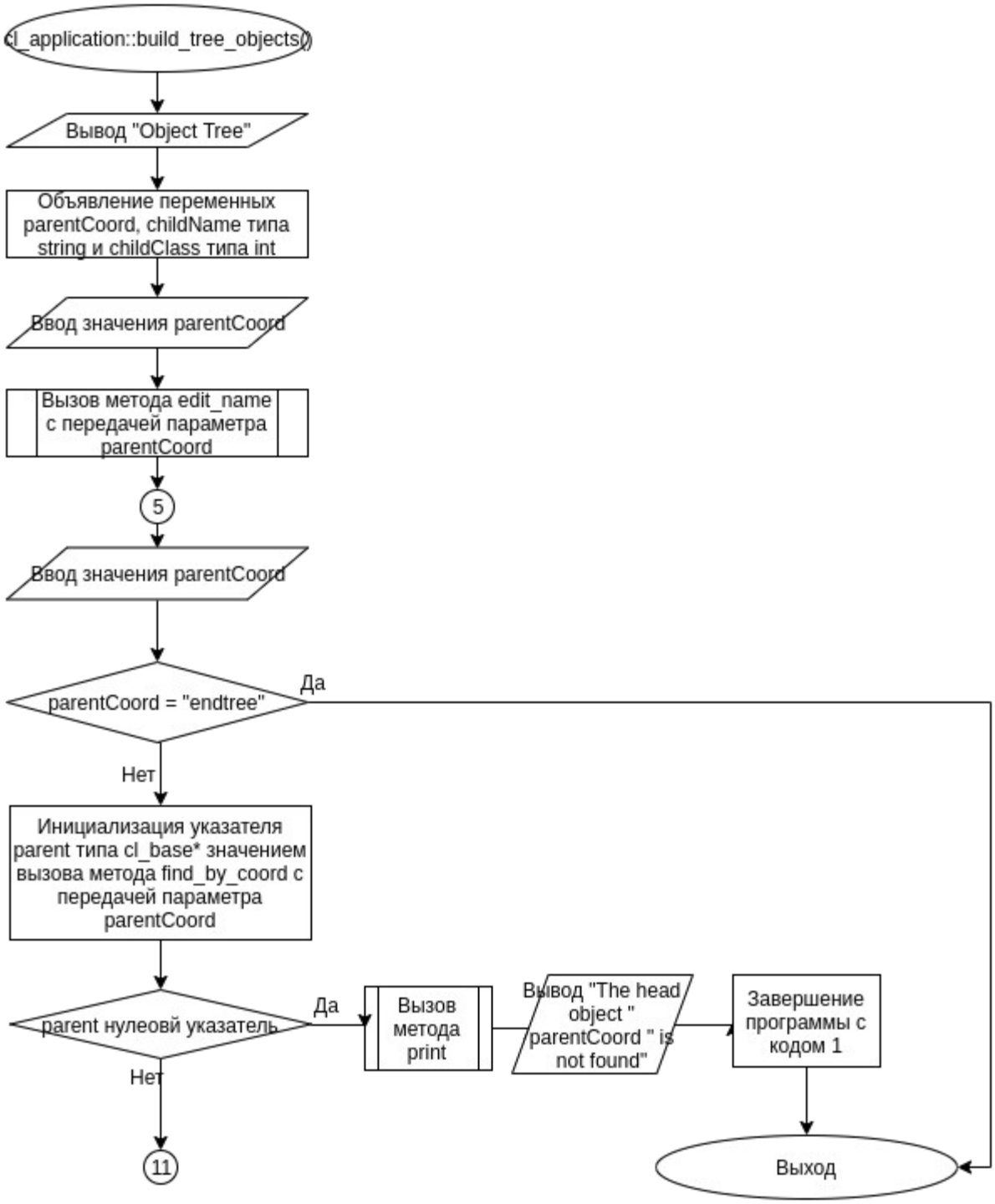

Рисунок 11 – Блок-схема алгоритма

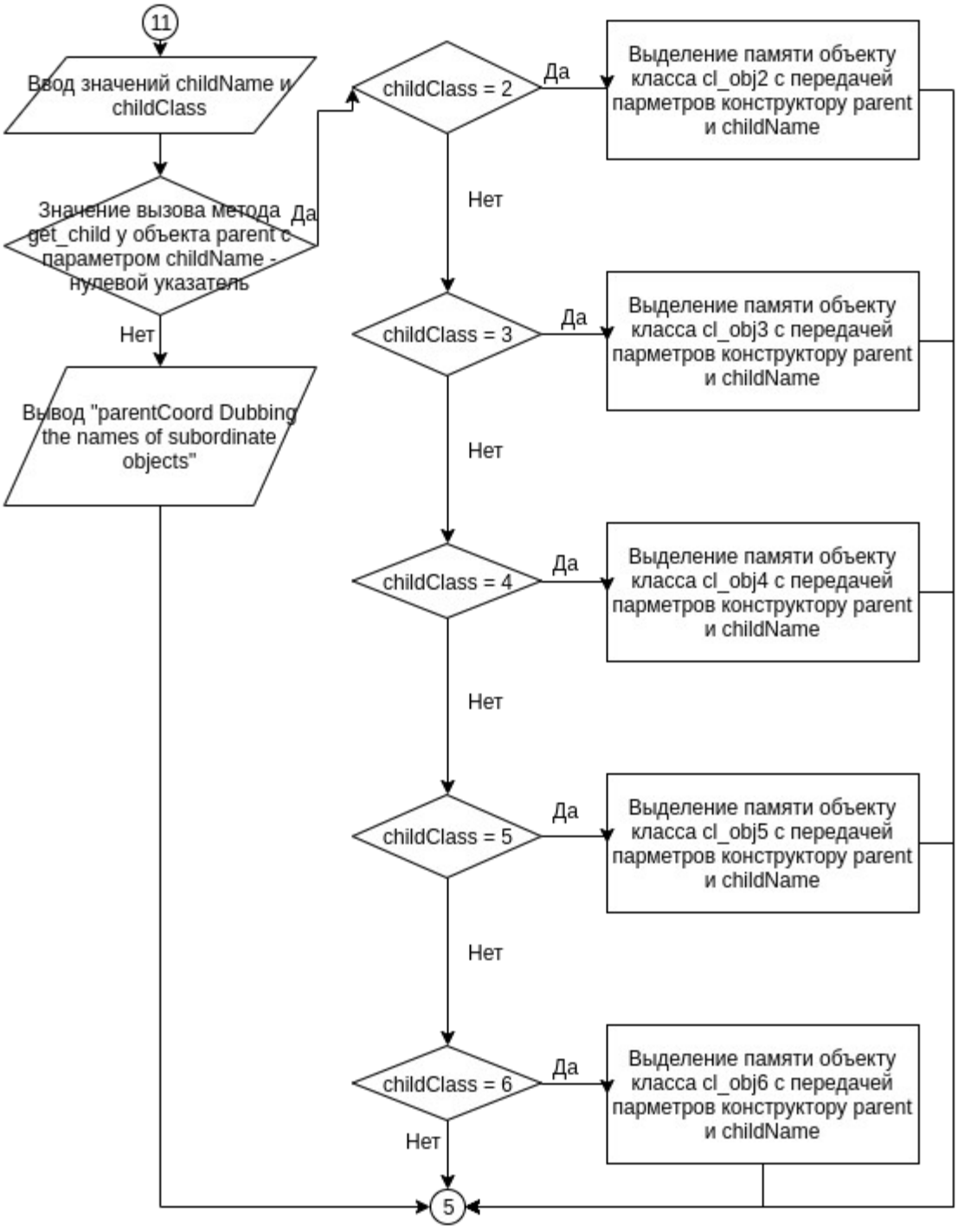

Рисунок 12 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_application.cpp

Листинг 1 – cl_application.cpp

```
#include "cl_application.h"


cl_application::cl_application(cl_base* header) : cl_base(header) {};

void cl_application::build_tree_objects() {
   cout << "Object tree" << endl;

   string parentCoord, childName;
   int childClass;

   cin >> parentCoord;
   this->edit_name(parentCoord);
   while(true)
   {
      cin >> parentCoord;
      if(parentCoord == "endtree")
         break;

      cl_base* parent = find_by_coord(parentCoord);

      if(!parent) {
         print();
         cout << endl << "The head object " << parentCoord << " is not
found";
         exit(1);
      }

      cin >> childName >> childClass;

      if(!parent->get_child(childName))
      {
         switch(childClass)
         {
            case 2:
            {
               new cl_obj2(parent, childName);
               break;
            }
```

```
                case 3:
                {
                   new cl_obj3(parent, childName);
                   break;
                }
                case 4:
                {
                   new cl_obj4(parent, childName);
                   break;
                }
                case 5:
                {
                   new cl_obj5(parent, childName);
                   break;
                }
                case 6:
                {
                   new cl_obj6(parent, childName);
                   break;
                }
             }
          }
          else {
             cout << parentCoord << "       Dubbing the names of subordinate
      objects" << endl;
          }
       }
      };
      int cl_application::exec_app()
      {
         print();
         cout << endl;

         cl_base* current = this;

         // Обработка команд
         while(true){
            string cmd, coord;
            cin >> cmd;
            if(cmd == "END") break;
            cin >> coord;

            cl_base* p_obj = current->find_by_coord(coord);
            if(cmd == "SET") {
               if(p_obj){
                  current = p_obj;
                  cout << "Object is set: " << current->get_name() << endl;
               }
               else {
                  cout << "The object was not found at the specified coordinate: "
      << coord << endl;
               }
          }
          else if(cmd == "FIND") {
             if(p_obj){
                cout << coord << "      Object name: " << p_obj->get_name() <<
```

```
endl;
            }
            else {
               cout << coord << "     Object is not found" << endl;
            }
         }
         else if(cmd == "MOVE"){
            if(!p_obj) {
               cout << coord << " Head object is not found" << endl;
            }
            else if(p_obj->get_header() != current->get_header()
                  && current->find_on_branch(p_obj->get_name()))
            {
               cout << coord << "       Redefining the head object failed" <<
endl;
            }
            else if (p_obj->get_child(current->get_name())) {
               cout << coord << "     Dubbing the names of subordinate objects"
<< endl;
            }
            else if (!current->change_header(p_obj)){
               cout << coord << "       Redefining the head object failed" <<
endl;
            }
            else {
               cout << "New head object: " << p_obj->get_name() << endl;
            }
         }
         else if(cmd == "DELETE") {
            if(p_obj) {
               cl_base* fheader = p_obj->get_header();
               string path;
               current->delete_by_name(p_obj->get_name());
               while(fheader != nullptr) {
                  if(fheader->get_header()) path += "/" + fheader->get_name();
                  fheader = fheader->get_header();
               }
               cout << "The object " << path + "/" + coord << " has been
deleted" << endl;
            }
         }
      }
      // Обработка команд

      cout << "Current object hierarchy tree" << endl;

      print();
      return 0;
};
```

5.2 Файл cl_application.h

Листинг 2 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
#include "cl_obj2.h"
#include "cl_obj3.h"
#include "cl_obj4.h"
#include "cl_obj5.h"
#include "cl_obj6.h"

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

#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"


cl_base::cl_base(cl_base* header, string name){
   this->name = name;
   this->header_ptr = header;
   if(header_ptr){
      header_ptr-> children_ptr.push_back(this);
   }
}

bool cl_base::edit_name(string new_name){
   this->name = new_name;
   return true;
}

string cl_base::get_name(){
   return this->name;
}

cl_base* cl_base::get_header(){
   return this->header_ptr;
```

```
}

cl_base* cl_base::get_child(string name){
   for(cl_base* child : children_ptr) {
      if(child->name == name) return child;
   }
   return nullptr;
}

cl_base* cl_base::find_on_branch(string name, int* count) {
   int c = 0;
   if(count == nullptr){
      count = &c;
   }

   cl_base* fchild = nullptr;
   if(this->name == name){
      fchild = this;
      (*count)++;
   }

   for(cl_base* child : children_ptr) {
      cl_base* f = child->find_on_branch(name, count);
      if(f)
         fchild = f;
   }

   if( (*count)==1 && fchild) {
      return fchild;
   }
   return nullptr;
}

cl_base* cl_base::find_on_tree(string name) {
   cl_base* fheader = this;
   while(fheader->get_header()){
      fheader = fheader->get_header();
   }
   return fheader->find_on_branch(name);
}

// КВ-3
bool cl_base::change_header(cl_base* new_header) {
   // Нельзя переопределять корневой и создавать новый корень
   if(header_ptr == nullptr || new_header == nullptr)
      return false;

   // У нового головного нельзя чтобы появились два подчиненных объекта с
одинаковым наименованием.
   if(new_header->get_child(name))
      return false;


   // Новый объект Не должен принадлежать к ветке текущего
   cl_base* current = new_header;
```

```
   while(current->get_header()){
      if(current == this) return false;
      current = current->get_header();
   }


   // Удаляем у текущего родителя текущий объект из списка подчинённых
   this->header_ptr->children_ptr.erase(find(this->header_ptr-
>children_ptr.begin(), this->header_ptr->children_ptr.end(), this));

   // Переопределяем головной объект
   this->header_ptr = new_header;
   new_header->children_ptr.push_back(this);

   return true;
}

void cl_base::delete_by_name(string name) {
   for(int i = 0; i < children_ptr.size(); i++) {
      if(children_ptr[i]->name == name){
         children_ptr.erase(children_ptr.begin() + i);
         return;
      }
   }
}

cl_base* cl_base::find_by_coord(string coord) {
   string s_name;

   // Пустая координата
   if(coord == "")
      return nullptr;

   cl_base* root = this;
   while(root->get_header()){
      root = root->get_header();
   }

   // Только Корень
   if(coord == "/") {
      return root;
   }

   // Поиск по уникальной имени от корневого
   if(coord.substr(0,2)=="//") {
      s_name = coord.substr(2);
      return this->find_on_tree(s_name);
   }

   // Только текущий объект
   if(coord == ".")
      return this;

   // Поиск по уникальной имени от текущего
   if(coord.substr(0,1)==".") {
```

```
            s_name = coord.substr(1);
            return this->find_on_branch(s_name);
        }

        // Если в начале стоит / преобразуем абсолютную координату
        // в относительную и ищем относительную от корня
        cl_base* current = this;

        // Абсолютная координата от корневого объекта
        if(coord.substr(0,1) == "/")
        {
            current = root;
            coord = coord.substr(1);
        }


        // Относительная координата от текущего объекта
        while(coord.find("/") != string::npos)  {
            int nsl = coord.find("/", 1);
            current = current->get_child(coord.substr(0, nsl));

            if(!current) {
                return nullptr;
            }

            coord = coord.substr(nsl + 1);
        }

        return current->get_child(coord);
    }

    // КВ-3


    void cl_base::print() {
        cout << this->name;

        int tab = 0;
        cl_base* par = this;
        while(par->get_header()){
            par = par->get_header();
            tab += 1;
        }


        if(!children_ptr.empty())
        {
            for(cl_base* child : children_ptr) {
                cout << endl;
                for(int i = 0; i <= tab; i++) cout << "    ";
                child->print();
            }
        }
    }
```

```
void cl_base::print_state() {
   cout << this->name;
   if(this->state == 0) cout << " is not ready";
   else cout << " is ready";

   int tab = 0;
   cl_base* par = this;
   while(par->get_header()){
      par = par->get_header();
      tab += 1;
   }


   if(!children_ptr.empty())
   {
      for(cl_base* child : children_ptr) {
         cout << endl;
         for(int i = 0; i <= tab; i++) cout << "    ";
         child->print_state();
      }
   }
}

void cl_base::set_state(int state) {
   if(header_ptr && header_ptr->state == 0) this->state = 0;
   else this->state = state;

   if(state == 0) {
      for(cl_base* child : children_ptr){
         child->set_state(0);
      }
   }
}
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <string>
#include <vector>
#include <iostream>
#include <algorithm>


using namespace std;

class cl_base {
```

```
protected:
   string name;
   cl_base *header_ptr;
   vector<cl_base*> children_ptr;

   int state;
public:
   cl_base(cl_base* base, string name = "Object");
   bool edit_name(string new_name);
   string get_name();
   cl_base* get_header();
   cl_base* get_child(string name);

   cl_base* find_on_branch(string name, int* count = nullptr); // Поиск на
ветке
   cl_base* find_on_tree(string name); // Поиск на всем дереве

   // КВ-3
   bool change_header(cl_base* new_header); // Переопределение головного
объекта
   void delete_by_name(string name); // Удаление объекта по наименованию
   cl_base* find_by_coord(string coord); // Поиск по координате
   // КВ-3


   void print();
   void print_state();

   void set_state(int state);

};

#endif
```

5.5 Файл cl_obj2.cpp

Листинг 5 – cl_obj2.cpp

```
#include "cl_obj2.h"

cl_obj2::cl_obj2(cl_base* header, string name) : cl_base(header, name) {}
```

5.6 Файл cl_obj2.h

Листинг 6 – cl_obj2.h

```
#ifndef __CL_OBJ2__H
#define __CL_OBJ2__H
#include "cl_base.h"

class cl_obj2 : public cl_base
{
public:
   cl_obj2(cl_base* header, string name);
};

#endif
```

5.7 Файл cl_obj3.cpp

Листинг 7 – cl_obj3.cpp

```
#include "cl_obj3.h"

cl_obj3::cl_obj3(cl_base* header, string name) : cl_base(header, name) {}
```

5.8 Файл cl_obj3.h

Листинг 8 – cl_obj3.h

```
#ifndef __CL_OBJ3__H
#define __CL_OBJ3__H

#include "cl_base.h"

class cl_obj3 : public cl_base
{
public:
   cl_obj3(cl_base* header, string name);
};

#endif
```

5.9 Файл cl_obj4.cpp

Листинг 9 – cl_obj4.cpp

```
#include "cl_obj4.h"

cl_obj4::cl_obj4(cl_base* header, string name) : cl_base(header, name) {}
```

5.10 Файл cl_obj4.h

Листинг 10 – cl_obj4.h

```
#ifndef __CL_OBJ4__H
#define __CL_OBJ4__H
#include "cl_base.h"

class cl_obj4 : public cl_base
{
public:
   cl_obj4(cl_base* header, string name);
};

#endif
```

5.11 Файл cl_obj5.cpp

Листинг 11 – cl_obj5.cpp

```
#include "cl_obj5.h"

cl_obj5::cl_obj5(cl_base* header, string name) : cl_base(header, name) {}
```

5.12 Файл cl_obj5.h

Листинг 12 – cl_obj5.h

```
#ifndef __CL_OBJ5__H
#define __CL_OBJ5__H
```

```
#include "cl_base.h"

class cl_obj5 : public cl_base
{
public:
   cl_obj5(cl_base* header, string name);
};


#endif
```

5.13 Файл cl_obj6.cpp

Листинг 13 – cl_obj6.cpp

```
#include "cl_obj6.h"

cl_obj6::cl_obj6(cl_base* header, string name) : cl_base(header, name) {}
```

5.14 Файл cl_obj6.h

Листинг 14 – cl_obj6.h

```
#ifndef __CL_OBJ6__H
#define __CL_OBJ6__H
#include "cl_base.h"

class cl_obj6 : public cl_base
{
public:
   cl_obj6(cl_base* header, string name);
};



#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

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

#include "cl_application.h"


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

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 9.

Таблица 9 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

rootela
/ object_1 3
/ object_2 2
/ object_2 4
/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
SET /object_3
DELETE object_7
END
	Object tree
	/ Dubbing the names of subordinate objects
	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
	Object is set: object_3
	The object /object_3/object_7 has been deleted
	Current object
		Object tree
		/ Dubbing the names of subordinate objects
		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
		Object is set: object_3
		The object /object_3/object_7 has been deleted
		Current object

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	hierarchy tree rootela object_1 object_7 object_2 object_4 object_5 object_3 object_3	hierarchy tree rootela object_1 object_7 object_2 object_4 object_5 object_3 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
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 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

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_2 object_4 object_5 object_3 object_3 object_7	object_2 object_4 object_5 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 SET /object_12 DELETE object_12 MOVE /object_2/object_4 SET /object_2 MOVE /object_2/object_4 END	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 The object was not found at the specified coordinate: /object_12 /object_2/object_4 Redefining the head object failed Object is set: object_2 /object_2/object_4 Redefining the head object failed Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 The object was not found at the specified coordinate: /object_12 /object_2/object_4 Redefining the head object failed Object is set: object_2 /object_2/object_4 Redefining the head object failed Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3
rootela / object_1 3 /object_2 object_3 6	Object tree rootela object_1 The head object /object_2 is not found	Object tree rootela object_1 The head object /object_2 is not found
r / a 2 /a b 2	Object tree r a	Object tree r a

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
/a/b c 2 /a/b/c a 2 /a/b/c/a a 3 /a/b/c/a b 3 / b 3 endtree SET a MOVE /b END	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	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).