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 конструирования и запуска системы;
- Объект стандартного потока ввода с клавиатуры cin;
- Объект стандартного потока вывода на экран cout;
- Условный оператор if..else;
- Оператор цикла for;
- Оператор цикла с предусловием while.

Класс Cl_base:

- свойства/поля:
    - поле Состояние объекта:
        - наименование — object_state;
        - тип — int;
        - модификатор доступа — private;
    - поле Наименование объекта:
        - наименование — s_object_name;
        - тип — string;
        - модификатор доступа — private;
    - поле Указатель на головной объект для текущего объекта:
        - наименование — p_head_object;
        - тип — Cl_base*;
        - модификатор доступа — private;
    - поле Динамический массив указателей на объекты, подчиненные текущему объекту:
        - наименование — subordinate_object;

        - тип — vector<Cl_base*>;
        - модификатор доступа — private;
- функционал:
    - метод change_head_object — метод переопределения головного объекта для текущего в дереве иерархии;
    - метод delete_sub_object_by_name — метод удаления подчиненного объекта по наименованию;
    - метод get_object_by_coordinate — метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты).

Класс Cl_application:

- функционал:
    - метод build_tree_object — метод построения исходного дерева иерархии объектов;
    - метод exec_app — метод запуска приложения.

Класс Cl_child_2:

Собственные свойства и методы отсутствуют.

Класс Cl_child_3:

Собственные свойства и методы отсутствуют.

Класс Cl_child_4:

Собственные свойства и методы отсутствуют.

Класс Cl_child_5:

Собственные свойства и методы отсутствуют.

Класс Cl_child_6:

Собственные свойства и методы отсутствуют.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	Cl_base			Базовый класс	
		Cl_application	public		2
		Cl_child_2	public		3
		Cl_child_3	public		4
		Cl_child_4	public		5
		Cl_child_5	public		6
		Cl_child_6	public		7
2	Cl_application			Класс приложение	
3	Cl_child_2			Производный класс	
4	Cl_child_3			Производный класс	
5	Cl_child_4			Производный класс	
6	Cl_child_5			Производный класс	
7	Cl_child_6			Производный класс	

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_head_object класса Cl_base

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

Параметры: Указатель p_head_object на объект класса Cl_base.

Возвращаемое значение: Логическое true или false.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода change_head_object класса Cl_base

№	Предикат	Действия	№ перехода
1	p_head_object равен nullptr или get_head_object равен nullptr?	Вернуть false	∅
			2
2	Результат метода get_sub_object_by_name объекта по адресу p_head_object с параметром this -> get_object_name() не равен nullptr?	Вернуть false	∅
			3
3	Результат метода get_branch_object_by_name объекта с параметром	Вернуть false	∅

№	Предикат	Действия	№ перехода
	p_head_object -> s_object_name равен p_head_object?		
			4
4		Вызов метода delete_sub_object_by_name объекта по адресу get_head_object с параметром get_object_name	5
5		Свойсву p_head_object присвоить значение параметра p_head_object	6
6		Добавить текущий объект в массив subordinate_object объекта по адресу p_head_object	7
7		Вернуть true	∅

3.2 Алгоритм метода delete_sub_object_by_name класса Cl_base

Функционал: метод удаления подчиненного объекта по наименованию.

Параметры: Строка s_object_name с наименование объекта.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода delete_sub_object_by_name класса Cl_base

№	Предикат	Действия	№ перехода
1	s_object_name пустая?	Выход из метода	∅
			2
2	Перебираем элементы вектора subordinate_objects, subordinate_object - указатель на текущий элемент, k -		3

№	Предикат	Действия	№ перехода
	индекс текущего элемента		
			∅
3	Результат метода get_object_name объекта по адресу subordinate_object равен s_object_name?	Удалить k-ый элемент из вектора subordinate_objects	2
			2

3.3 Алгоритм метода get_object_by_coordinate класса Cl_base

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

Параметры: Строка coordinate с координатой объекта на дереве.

Возвращаемое значение: Указатель на объект класса Cl_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_object_by_coordinate класса Cl_base

№	Предикат	Действия	№ перехода
1	coordinate не пустая?	Объявление указателя base на объект класса Cl_base и инициализация указателем на текущий объект	2
		Вернуть nullptr	∅
2	Первый элемент строки coordinate точка?		3
			4
3	Длина строки coordinate равна 1?	Вернуть указатель на текущий объект	∅

№	Предикат	Действия	№ перехода
		Вернуть результат метода get_branch_object_by_name с параметром подстроки coordinate, начинающейся со 2 элемента	∅
4	Первый элемент строки coordinate слеш?		5
			9
5	Результат метода get_head_object объекта по адресу base не равен nullptr?	base присвоить результат метода get_head_object объекта по адресу base	5
			6
6	Длина строки coordinate равна 1?	Вернуть base	∅
			7
7	Второй элемент строки coordinate равен слеш?	Вернуть результат метода get_branch_object_by_name с параметром подстроки coordinate, начинающейся с 3 элемента	∅
			8
8		coordinate присвоить значение подстроки coordinate, начинающейся со 2 элемента	9
9		Объявление вектора строк names	10
10	В строке coordinate найден слеш?	Добавить в вектор names наименование объекта до слеша	11
		Добавить в вектор names строку coordinate	12
11		coordinate присвоить значение подстроки coordinate после первого слеша	10
12	Перебираем элементы вектора names, name - указатель на текущий элемент	base присвоить резульат метода get_sub_object_by_name объекта по адресу base с параметром name	13

№	Предикат	Действия	№ перехода
			14
13	base равен nullptr?		14
			12
14		Вернуть base	∅

3.4 Алгоритм метода build_tree_object класса Cl_application

Функционал: метод построения исходного дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода build_tree_object класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных head и sub, указателя head_object на объект класса Cl_base, целочисленной переменной iClass	2
2		Ввод значения head с клавиатуры	3
3	head равно "endtree"?		∅
		Ввод значений sub и iClass с клавиатуры	4
4		Присвоение указателю head_object результата метода get_object_by_coordinate с параметром head	5
5	head_object не равен nullptr?		6
			8
6	Результат метода get_sub_object_by_name объекта по адресу head_object с параметром sub не равен nullptr?	Вывод на экран "(head) Dubbing the names of subordinate objects"	2

№	Предикат	Действия	№ перехода
			7
7		Создание объекта класса Cl_child_i с параметрами head_object и sub, где i - номер класса iClass	2
8		Вывод на экран "Object tree"	9
9		Вызов метода show_object_tree	10
10		Вывод на экран "The head object (head) is not found"	11
11		Выход из программы с кодом 1	∅

3.5 Алгоритм метода exec_app класса Cl_application

Функционал: метод запуска приложения.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода exec_app класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных command и args, векторов строк commands и callback, указателя target на текущий объект и указателя from со значением nullptr	2
2		Ввод строки command с клавиатуры	3
3		Добавление command в commands	4
4	command равно "END"?		5
			2
5		Вывод на экран "Object tree"	6
6		Вызов метода show_object_tree	7
7	Перебираем элементы	command присвоить doing	8

№	Предикат	Действия	№ перехода
	вектора commands, doing - указатель на текущий элемент		
			28
8	command равно "END"?	Добавить "Current object hierarchy tree" в callback	28
			9
9		args присвоить подстроку command после пробела	10
10		command присвоить подстроку command до пробела	11
11		from присвоить результат метода get_object_by_coordinate объекта по адресу target с параметром args	12
12	command равно "SET"?		13
			15
13	target не равно nullptr?	from присвоить target	14
		Добавить "The object was not found at the specified coordinate" в callback	7
14		Добавить "Object is set: (from -> get_object_name())" в callback	7
15	command равно "FIND"?		16
			17
16	target не равно nullptr?	Добавить "(args) Object name: (target -> get_object_name())" в callback	7
		Добавить "(args) Object is not found" в callback	7
17	command равно "MOVE"?		18
			22
18	target не равно nullptr?		19
		Добавить "(args) Head object is not found" в callback	7

№	Предикат	Действия	№ перехода
19	Результат метода get_branch_object_by_name объекта по адресу from с параметром target -> s_object_name равен target?	Добавить "(args) Redefinding the head object failed" в callback	7
			20
20	Результат метода get_sub_object_by_name объекта по адресу target с параметром from -> s_object_name не равен nullptr?	Добавить "(args) Dubbing the name of subordinate objects" в callback	7
			21
21	Результат метода change_head_object объекта по адресу from с параметром target не равен nullptr?	Добавить "New head object: (from -> get_head_object() -> get_object_name())" в callback	7
			7
22	command равно "DELETE"?		23
			7
23	target не равно nullptr?	Объявление указателя to и инициализация результатом метода get_head_object объекта по адресу target	24
			7
24	to не равно nullptr?	Объявление строки absolute и инициализация результатом метода get_object_name объекта по адресу target	25
			7
25		Вызов метода delete_sub_object_by_name объекта	26

№	Предикат	Действия	№ перехода
		по адресу to с параметром absolute	
26	Результат метода get_head_object объекта по адресу to не равен nullptr?	Добавление в начало строки absolute результата метода get_object_name объекта по адресу to и слеша	27
		Добавить "The object /(absolute) has been deleted" в callback	7
27		to присвоить результат метода get_head_object объекта по адресу to	26
28	Перебираем элементы вектора callback, back - указатель на текущий элемент	Вывод на экран (back)	28
		Вызов метода show_tree_object	∅

3.6 Алгоритм функции main

Функционал: Конструирование и запуск системы.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм функции представлен в таблице 7.

Таблица 7 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Объявление объекта ob_cl_application класса Cl_application с параметром nullptr	2
2		Вызов метода build_tree_object объекта ob_cl_application	3
3		Возврат значения метода exec_app объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-8.

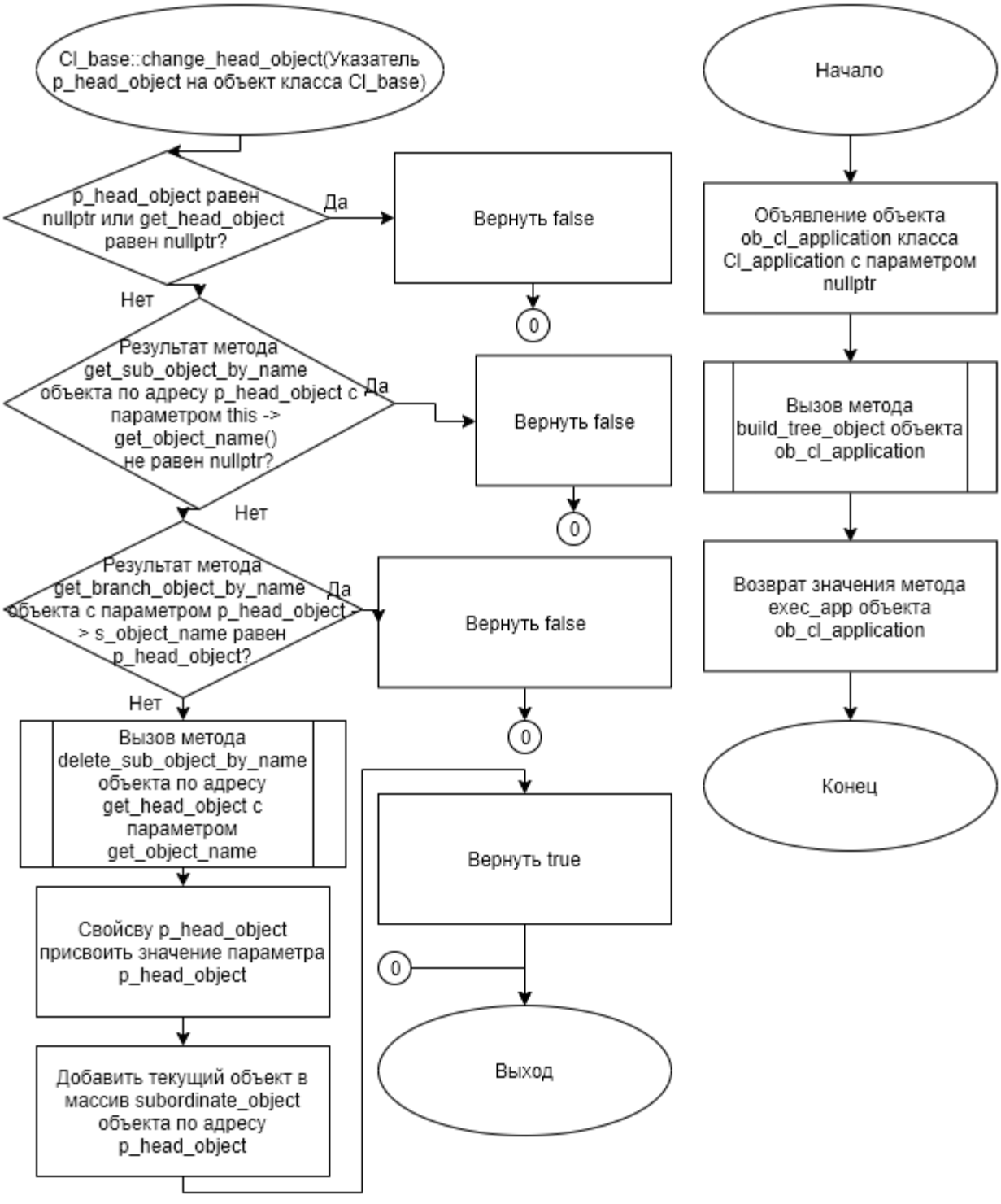

Рисунок 1 – Блок-схема алгоритма

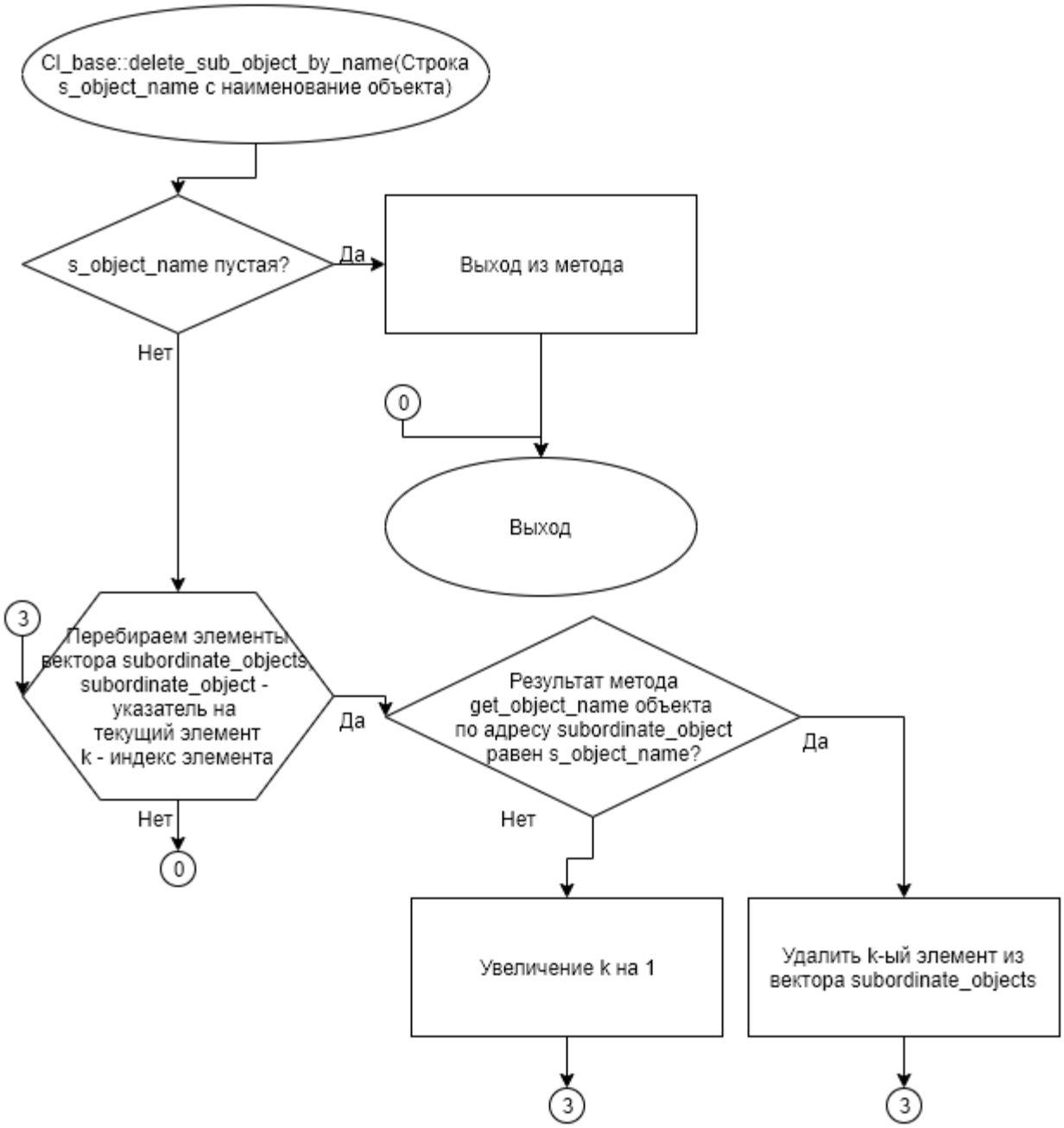

Рисунок 2 – Блок-схема алгоритма

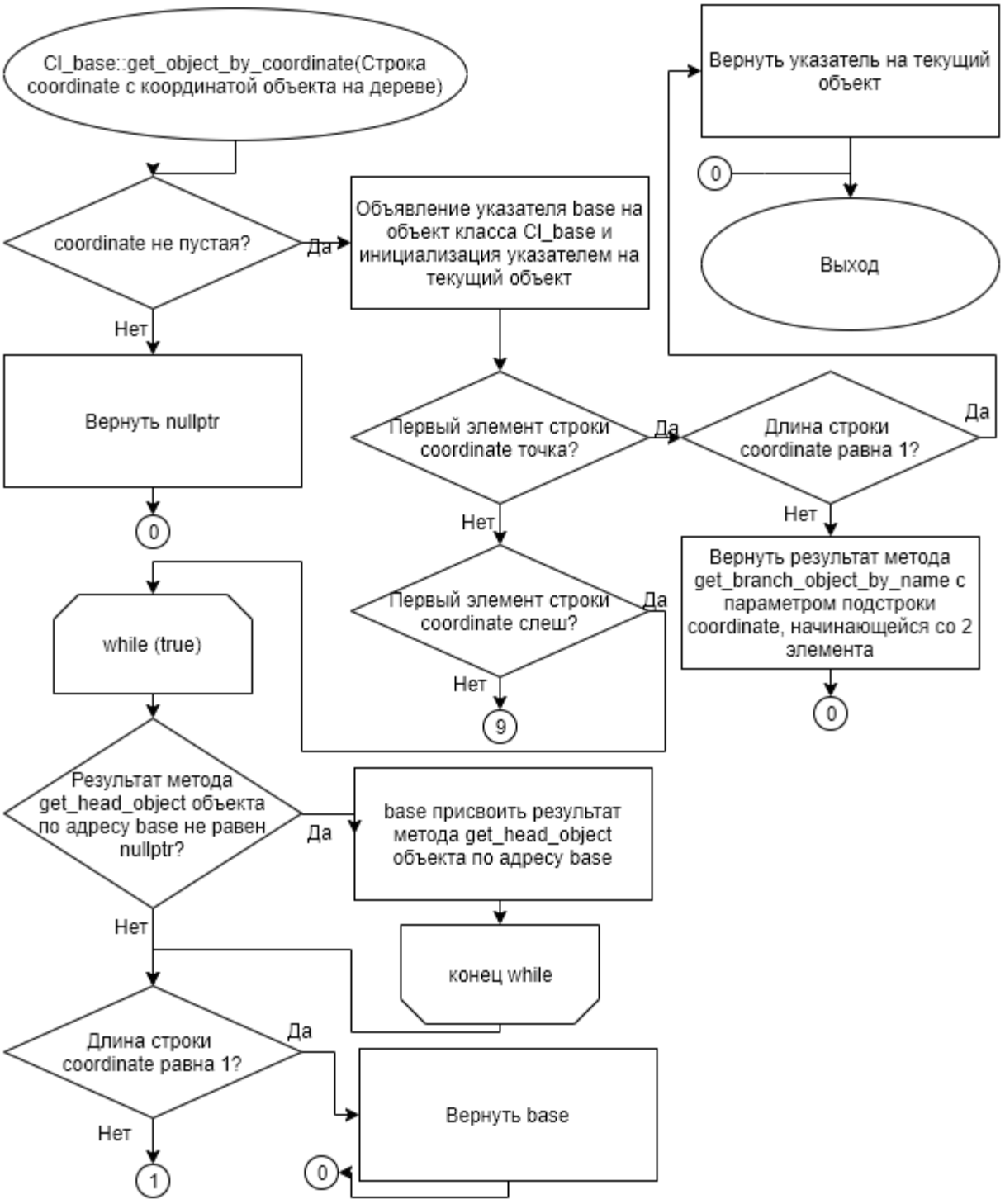

Рисунок 3 – Блок-схема алгоритма

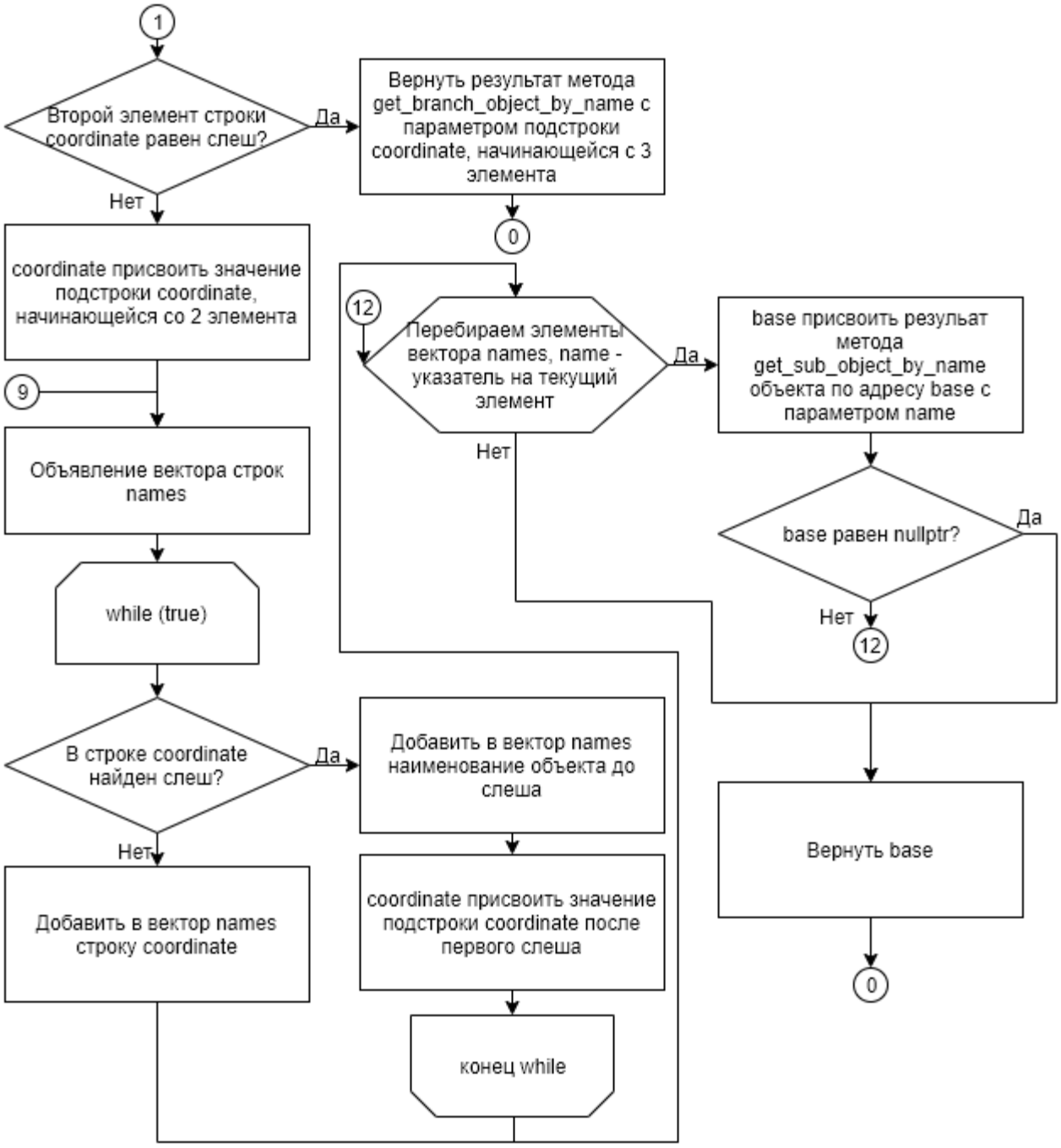

Рисунок 4 – Блок-схема алгоритма

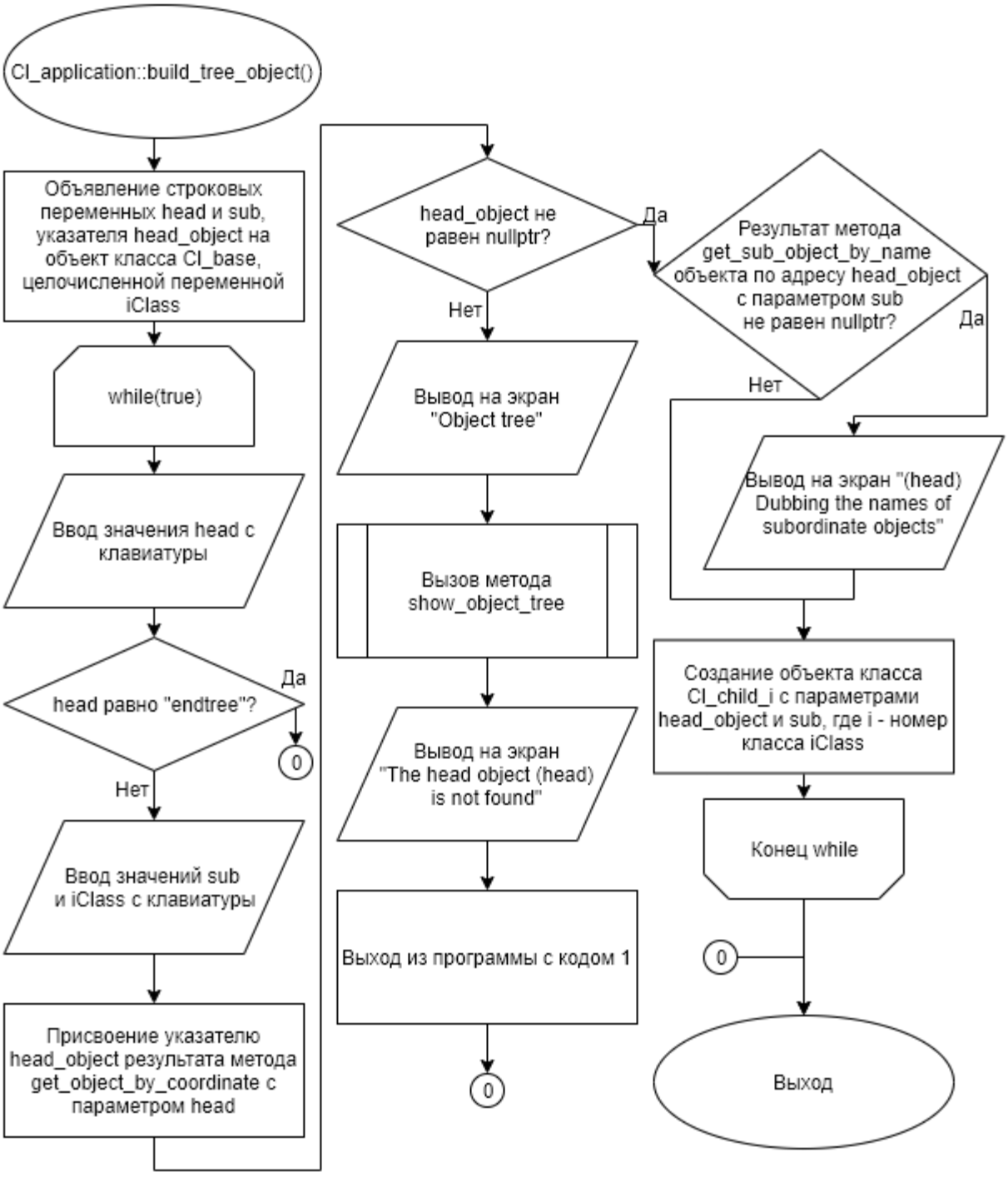

Рисунок 5 – Блок-схема алгоритма

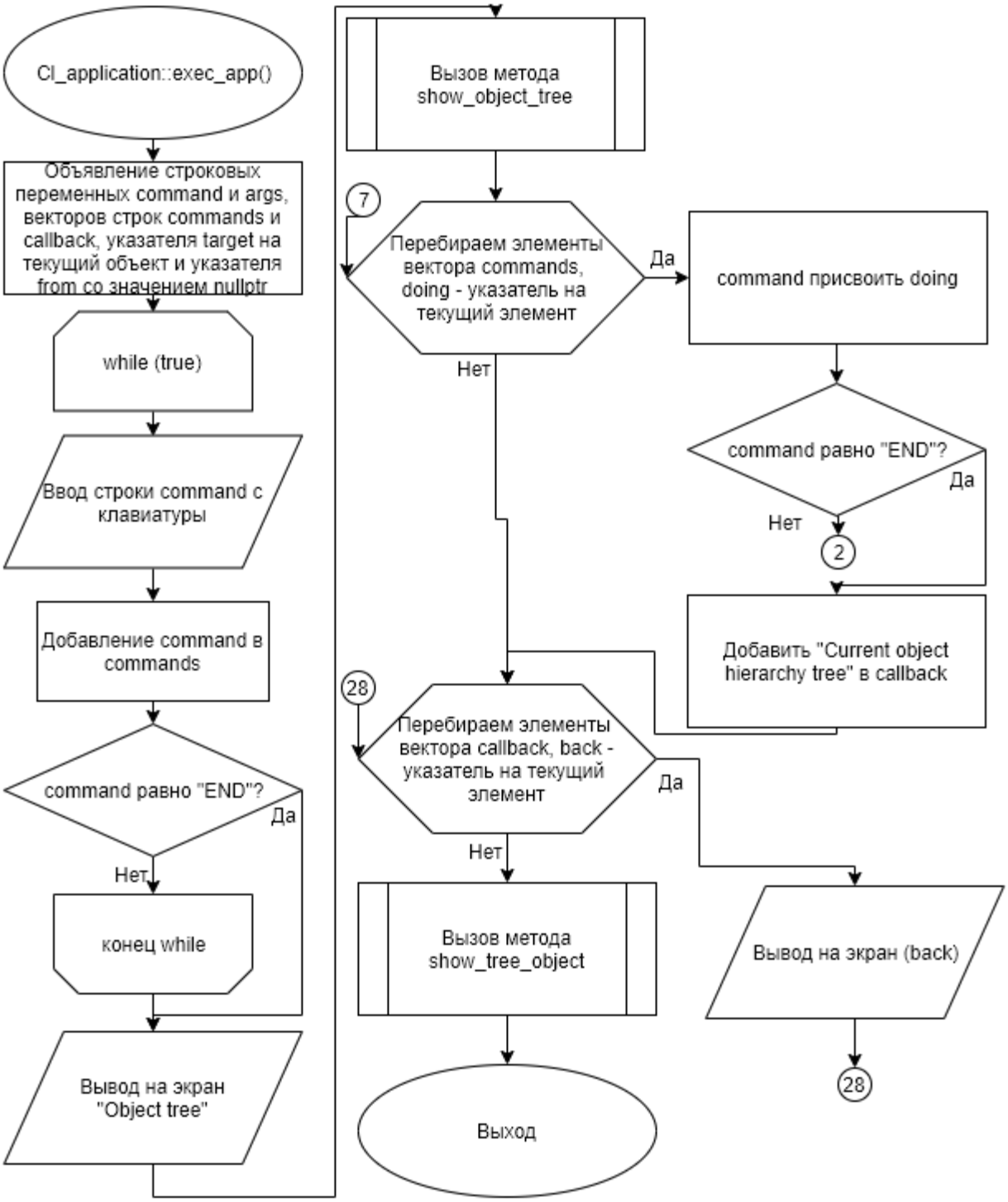

Рисунок 6 – Блок-схема алгоритма

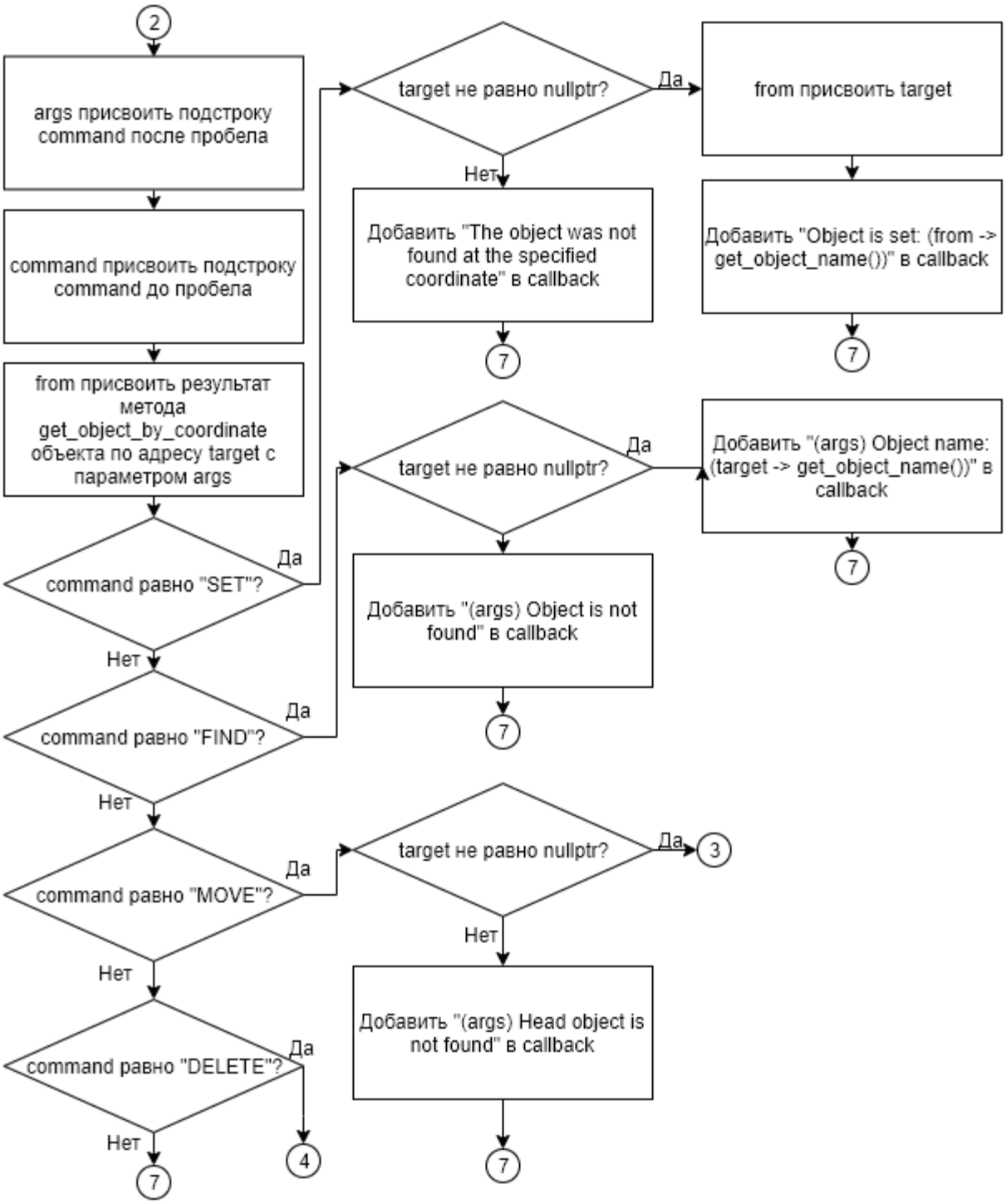

Рисунок 7 – Блок-схема алгоритма

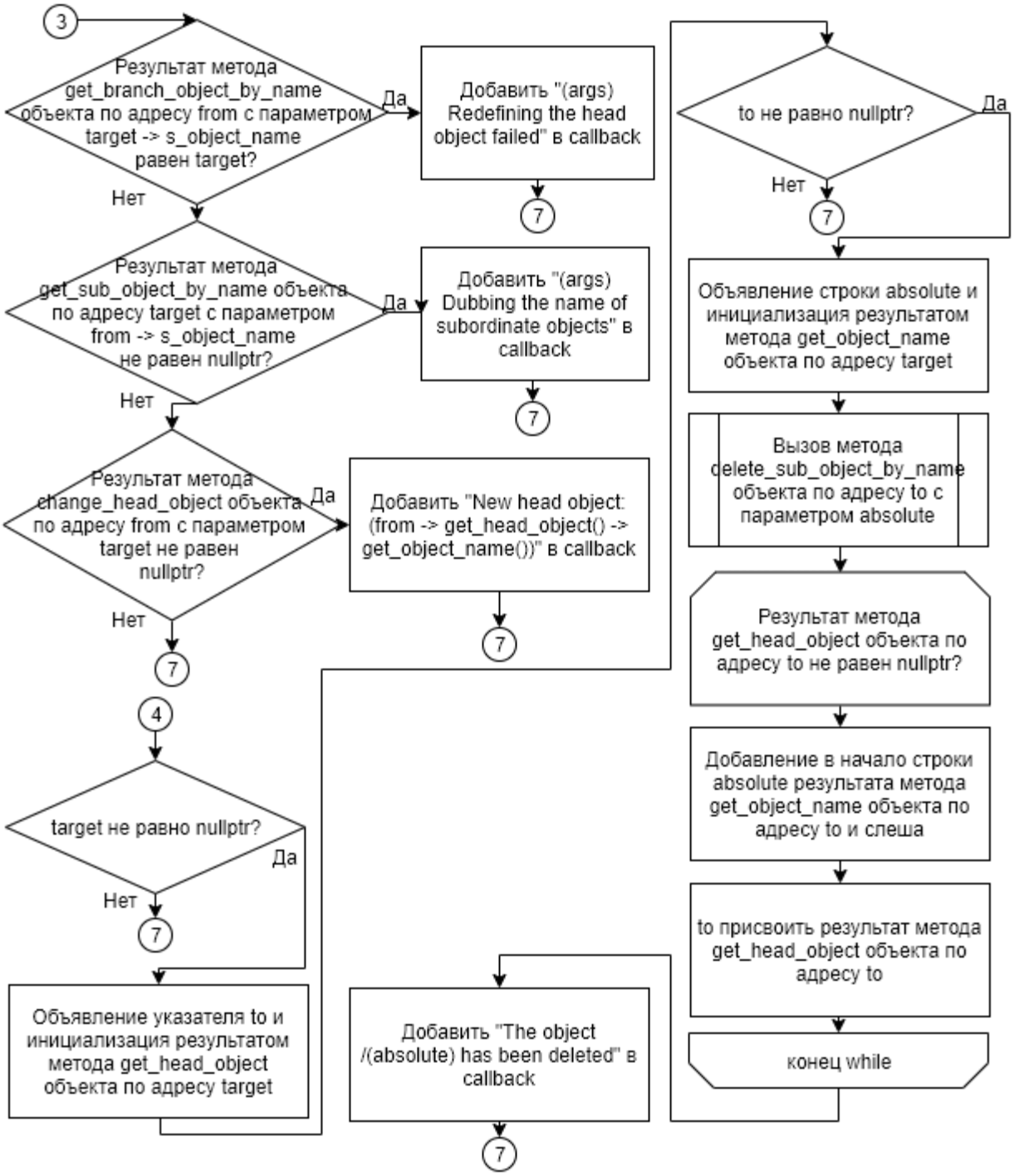

Рисунок 8 – Блок-схема алгоритма

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*         p_head_object,         string
s_name_object):Cl_base(p_head_object, s_name_object)
{
   cin >> s_name_object;//ввод имени объекта
   change_object_name(s_name_object);//изменение имени объекта
}

void Cl_application::build_tree_objects()
{
   string head, sub;//переменные имен объектов
   Cl_base* head_object;//указатель на головной объект
   int iClass;//номера классов
   while (true) {//ввод дерева
      cin >> head;//ввод имени головного объекта
      if (head == "endtree") {//если endtree то выход из цикла
         break;
      }
      cin >> sub >> iClass;//ввод номера класса
      head_object = get_object_by_coordinate(head);//поиск годовного объекта
по координате
      if (head_object) {//Координата не nullptr?
         if(head_object -> get_sub_object_by_name(sub)) {//Есть ли дубль?
            cout << head << "      Dubbing the names of subordinate objects"
<< endl;//сообщение о дубле
            continue;
         }
         switch(iClass)//создание объекта по номеру класса
         {
            case 2:
               new Cl_child_2(head_object, sub);
               break;
            case 3:
               new Cl_child_3(head_object, sub);
               break;
            case 4:
               new Cl_child_4(head_object, sub);
               break;
```

```
            case 5:
               new Cl_child_5(head_object, sub);
               break;
            case 6:
               new Cl_child_6(head_object, sub);
               break;
         }
      }
      else {
         cout << "Object tree" << endl;
         this -> show_object_tree();//Вывод построенного дерева
         cout << "The head object " << head << " is not found";//вывод
сообщения об ошибке
         exit(1);
      }
   }
}

int Cl_application::exec_app()
{
   string command, args;//строки для парсинга команды
   vector<string> commands, callback;//вектора команд и вывода
   Cl_base* from = this,* target = nullptr;//указатели на текущий и целевой
объект
   while (true) {
      getline(cin, command);//чтение команды
      commands.push_back(command);//добавление команды в список
      if (command == "END") {//команда end?
         break;
      }
   }
   cout << "Object tree" << endl;
   show_object_tree();//вывод дерева объектов
   for (string doing : commands) {//для всех элементов вектора
      command = doing;//обновление команды
      if (command == "END") {//команда end?
         callback.push_back("Current   object   hierarchy   tree");//добавить
сообщение в вывод
         break;
      }
      args = command.substr(command.find(' ') + 1);//извлечение координаты
      command = command.substr(0, command.find(' '));//извлечение команды
      target = from -> get_object_by_coordinate(args);//определение целевого
объекта
      if (command == "SET") {//команда set?
         if (target) {//существует?
            from = target;//обновление указателя
            callback.push_back("Object    is    set:    "    +    from    ->
get_object_name());//добавить сообщение в вывод
         }
         else {
            callback.push_back("The object was not found at the specified
coordinate " + args);//добавить сообщение в вывод
         }
      }
```

```
      else if (command == "FIND") {//команда find?
         if (target) {//существует?
            callback.push_back(args + "        Object name: " + target ->
get_object_name());//добавить сообщение в вывод
         }
         else {
            callback.push_back(args + "     Object is not found");//добавить
сообщение в вывод
         }
      }
      else if (command == "MOVE") {//команда move?
         if (target) {//существует?
            if     (from     ->     get_branch_object_by_name(target     ->
get_object_name()) == target) {//находится в ветке?
               callback.push_back(args + "       Redefining the head object
failed");//добавить сообщение в вывод
            }
            else    if    (target    ->    get_sub_object_by_name(from    ->
get_object_name())) {//создается дубль?
               callback.push_back(args + "          Dubbing the names of
subordinate objects");//добавить сообщение в вывод
            }
            else if (from -> change_head_object(target)) {//смена головного
объекта
               callback.push_back("New   head   object:   "   +   from   ->
get_head_object() -> get_object_name());//добавить сообщение в вывод
            }
         }
         else {
            callback.push_back(args   +   "            Head   object   is   not
found");//добавить сообщение в вывод
         }
      }
      else if (command == "DELETE") {//команда delete?
         if (target) {//существует?
            Cl_base* to = target -> get_head_object();//указатель на головной
объект целевого
            if (to) {//существует?
               string absolute = target -> get_object_name();//добавление имя
target в координату
               to -> delete_sub_object_by_name(absolute);//удаление absolute
из списка подчиненных
               while (to -> get_head_object()) {//существует головной объект?
                  absolute   =   to   ->   get_object_name()   +   '/'   +
absolute;//добавление координаты
                  to = to -> get_head_object();//обновение указателя
               }
               callback.push_back("The object /" + absolute + " has been
deleted");//добавить сообщение в вывод
            }
         }
      }
   }

   for (string back : callback) {//для всех элементов вектора
```

```
      cout << back << endl;//вывод элемента
   }
   show_object_tree();//вывод дерева объектов
   return 0;
}
```

5.2 Файл Cl_application.h

Листинг 2 – Cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "Cl_child_2.h"
#include "Cl_child_3.h"
#include "Cl_child_4.h"
#include "Cl_child_5.h"
#include "Cl_child_6.h"

class Cl_application: public Cl_base//наследование класса
{
public:
   Cl_application(Cl_base*          p_head_object,          string          =
"Base_object");//параметризированный конструктор
   void build_tree_objects();//метод построения дерева
   int exec_app();//метод запуска системы
};

#endif
```

5.3 Файл Cl_base.cpp

Листинг 3 – Cl_base.cpp

```
#include "Cl_base.h"

Cl_base::Cl_base(Cl_base* p_head_object, string s_object_name)
{
   this   ->   p_head_object   =   p_head_object;//присвоение   указателя   на
родительский объект
   this -> s_object_name = s_object_name;//присвоение имени объекта
   if ( p_head_object ) {//есть родительский объект?
      p_head_object   ->   subordinate_objects.push_back(this);//добавить   в
производные объекты
   }
}
```

```
bool Cl_base::change_object_name(string s_object_name)
{
   if (s_object_name.empty()) {//пустая строка?
      return false;
   }
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
      if (subordinate_object -> get_object_name() == s_object_name) {//если
имя равно искомому
         return false;
      }
   }
   this -> s_object_name = s_object_name;//сменить имя
   return true;
}

string Cl_base::get_object_name()
{
   return s_object_name;//вернуть имя объекта
}

Cl_base* Cl_base::get_head_object()
{
   return p_head_object;//вернуть указатель на родительский объект
}

void Cl_base::show_object_tree()
{
   Cl_base* head_object = p_head_object;//указатель на головной объект
   while (head_object != nullptr) {//существует головной объект?
      cout << "    ";//отступ
      head_object = head_object -> p_head_object;//обновление головного
объекта
   }
   cout << s_object_name << endl;//вывод имени объекта
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
      subordinate_object -> show_object_tree();//уход в рекурсию
   }
}

Cl_base* Cl_base::get_sub_object_by_name(string s_object_name)
{
   if (!s_object_name.empty()) {//строка не пустая?
      for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
         if (subordinate_object -> get_object_name() == s_object_name) {//имя
равно искомому?
            return subordinate_object;//вернуть указатель на подчиненный
объект
         }
      }
   }
   return nullptr;
```

```
}

Cl_base* Cl_base::get_branch_object_by_name(string s_object_name)
{
  if (this -> s_object_name == s_object_name) {//строка совпадает с именем
объекта?
     return this;//вернуть объект
  }
  for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объеков
     if (subordinate_object -> get_object_name() == s_object_name) {//строка
совпадает с именем объекта?
        return subordinate_object;//вернуть объект
     }
  }
  for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
     if (subordinate_object -> get_branch_object_by_name(s_object_name))
{//есть в ветви такое имя?
        return                    subordinate_object                    ->
get_branch_object_by_name(s_object_name);//вернуть объект, если есть
     }
  }
  return nullptr;
}

Cl_base* Cl_base::get_object_by_name(string s_object_name)
{
  Cl_base* base = this;//указатель на текущий объект
  while (true) {
     if (base -> get_head_object()) {//существует головной объект?
        base = base -> get_head_object();//обновить текущий объект
     }
     else {
        break;
     }
  }
  if (base -> get_branch_object_by_name(s_object_name)) {//есть в дереве
такое имя?
     return  base  ->  get_branch_object_by_name(s_object_name);//вернуть
объект, если есть
  }
  return nullptr;
}

void Cl_base::show_object_tree_full()
{
  Cl_base* head_object = p_head_object;//указатель на головной объект
  while (head_object != nullptr) {//головной объект существует?
     cout << "    ";//отступ
     head_object  =  head_object  ->  p_head_object;//обновление  головного
объекта
  }
  cout << s_object_name;//вывод имени объекта
  if (object_state != 0) {//вывод состояния
```

```
            cout << " is ready" << endl;
        }
        else {
            cout << " is not ready" << endl;
        }
        for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
            subordinate_object -> show_object_tree_full();//уход в рекурсию
        }
    }

    void Cl_base::change_object_state(int object_state)
    {
        if (object_state != 0) {//состояние отлично от 0?
            Cl_base* head_object = p_head_object;//указатель на головной объект
            bool f = true;//объявление флага
            while (head_object != nullptr) {//головной объект существует?
                if (head_object -> object_state == 0) {//состояние головного объекта
0?
                    f = false;
                    break;
                }
                head_object = head_object -> p_head_object;//обновление головного
объекта
            }
            if (f) {
                this -> object_state = object_state;//обновление состояния объекта
            }
        }
        else {
            this -> object_state = 0;//обнуление состояния объекта
            for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
                subordinate_object -> change_object_state(0);//обнуление
            }
        }
    }

    bool Cl_base::change_head_object(Cl_base* p_head_object)
    {
        if (!p_head_object || !get_head_object()) {//Создается новый корневой
объект или переопределяется корневой объект?
            return false;
        }
        if (p_head_object -> get_sub_object_by_name(this -> get_object_name()) !=
nullptr) {//При переопределеии появляется дубль?
            return false;
        }
        if (this -> get_branch_object_by_name(p_head_object -> s_object_name) ==
p_head_object) {//Новый головной объектв ветви текущего объекта?
            return false;
        }
        get_head_object()       ->       delete_sub_object_by_name(this       ->
get_object_name());//Удаление объекта из списка подчиненных
        this -> p_head_object = p_head_object;//Переопределение указателя на
```

```
головной объект
   this -> p_head_object -> subordinate_objects.push_back(this);//Добавление
объекта в список подчиненных
   return true;
}

void Cl_base::delete_sub_object_by_name(string s_object_name)
{
   if (s_object_name.empty()) {//Строка пустая?
      return;
   }
   int k = 0;//переменная счетчик
   for  (Cl_base*  subordinate_object  :  subordinate_objects)  {//для  всех
подчиненных объектов
      if  (subordinate_object  ->  get_object_name()  ==  s_object_name)  {//имя
объекта равняетя искомому?
         subordinate_objects.erase(subordinate_objects.begin()              +
k);//удаление текущего объекта из списка подчиненных
      }
      else {
         k++;//увеличение счетчика
      }
   }
}

Cl_base* Cl_base::get_object_by_coordinate(string coordinate)
{
   if (!coordinate.empty()) {//координата не пустая?
      Cl_base* base = this;//Указатель на текущий объект
      if (coordinate[0] == '.') {//Кооринатa начинается с точки?
         if (coordinate.length() == 1) {//Длина равна 1?
            return this;//Вернуь текущий объект
         }
         return     get_branch_object_by_name(coordinate.substr(1));//Вернуть
объект найденный по имени на ветке
      }
      if (coordinate[0] == '/') {//Координата начинается со слеша?
         while (true) {
            if (base -> get_head_object()) {//существует головной объект?
               base = base -> get_head_object();//обновить текущий объект
            }
            else {
               break;
            }
         }
         if (coordinate.length() == 1) {//Длина равна 1?
            return base;//Вернуть корневой объект
         }
         if (coordinate[1] == '/') {//Второй элемент координаты равен слеш?
            return                              base                          ->
get_sub_object_by_name(coordinate.substr(2));//Вернуть    объект    найденный
среди подчиненных по имени
         }
         coordinate   =   coordinate.substr(1);//Отрезать   первый   элемент
координаты
```

```
        }
        vector<string> names;//массив для записи имен объектов в координате
        while (true) {
           if (coordinate.find('/') != string::npos) {//В координате есть слеш?
              names.push_back(coordinate.substr(0,
coordinate.find('/')));//добавить в массив имя объекта до слеша
              coordinate     =     coordinate.substr(coordinate.find('/')     +
1);//отрезать первое имя и слеш
           }
           else {
              names.push_back(coordinate);//добавить в массив имя объекта
              break;
           }
        }
        for (string name : names) {//для всех элементов массива
           base  =  base  ->  get_sub_object_by_name(name);//обновить  текущий
объект
           if (!base) {//Нет подчиненного с таким именем?
              break;
           }
        }
        return base;//вернуть текущий объект
     }
     return nullptr;
  }
```

5.4 Файл Cl_base.h

Листинг 4 – Cl_base.h

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

  using namespace std;

  class Cl_base//наименование класса
  {
  public:
     Cl_base     (Cl_base*     p_head_object,     string     s_object_name     =
  "Base_object");//параметризированный конструктор
     bool change_object_name(string);//метод изменения имени
     string get_object_name();//метод получения имени
     Cl_base*  get_head_object();//метод  получения  указателя  на  родительский
  объект
     Cl_base*   get_sub_object_by_name(string);//метод   поиска   подчиненного
  объекта по имени

     Cl_base* get_branch_object_by_name(string);//метод поиска объекта на ветке
```

```
по имени
   Cl_base* get_object_by_name(string);//метод поиска объекта по имени
   void show_object_tree();//метод вывода дерева объектов
   void show_object_tree_full();//метод вывода дерева объектов и состояния
   void change_object_state(int);//метод установки состояния

   bool   change_head_object(Cl_base*);//метод   переопределения   головного
объекта
   void   delete_sub_object_by_name(string);//метод   удаления   подчиненного
объекта по наименованию
   Cl_base*  get_object_by_coordinate(string);//метод получения указателя на
объект по его координате

private:
   int object_state;//состояние объекта
   string s_object_name;//имя объекта
   Cl_base* p_head_object;//указатель на родительский объект
   vector <Cl_base*> subordinate_objects;//подчиненные объекты
};



#endif
```

5.5 Файл Cl_child_2.cpp

Листинг 5 – Cl_child_2.cpp

```
#include "Cl_child_2.h"

Cl_child_2::Cl_child_2(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.6 Файл Cl_child_2.h

Листинг 6 – Cl_child_2.h

```
#ifndef __CL_CHILD__H
#define __CL_CHILD__H
#include "Cl_base.h"

class Cl_child_2 : public Cl_base//наследование класса
```

```
{
public:
   Cl_child_2(Cl_base*, string);//параметризированный конструктор
};

#endif
```

5.7 Файл Cl_child_3.cpp

Листинг 7 – Cl_child_3.cpp

```
#include "Cl_child_3.h"

Cl_child_3::Cl_child_3(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.8 Файл Cl_child_3.h

Листинг 8 – Cl_child_3.h

```
#ifndef __CL_CHILD_3__H
#define __CL_CHILD_3__H
#include "Cl_base.h"

class Cl_child_3 : public Cl_base//наследование класса
{
public:
   Cl_child_3(Cl_base*, string);//параметризированный конструктор
};


#endif
```

5.9 Файл Cl_child_4.cpp

Листинг 9 – Cl_child_4.cpp

```
#include "Cl_child_4.h"

Cl_child_4::Cl_child_4(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.10 Файл Cl_child_4.h

Листинг 10 – Cl_child_4.h

```
#ifndef __CL_CHILD_4__H
#define __CL_CHILD_4__H
#include "Cl_base.h"

class Cl_child_4 : public Cl_base//наследование класса
{
public:
   Cl_child_4(Cl_base*, string);//параметризированный конструктор
};


#endif
```

5.11 Файл Cl_child_5.cpp

Листинг 11 – Cl_child_5.cpp

```
#include "Cl_child_5.h"

Cl_child_5::Cl_child_5(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.12 Файл Cl_child_5.h

Листинг 12 – Cl_child_5.h

```
#ifndef __CL_CHILD_5__H
#define __CL_CHILD_5__H
#include "Cl_base.h"

class Cl_child_5 : public Cl_base//наследование класса
{
public:
   Cl_child_5(Cl_base*, string);//параметризированный конструктор
};


#endif
```

5.13 Файл Cl_child_6.cpp

Листинг 13 – Cl_child_6.cpp

```
#include "Cl_child_6.h"

Cl_child_6::Cl_child_6(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.14 Файл Cl_child_6.h

Листинг 14 – Cl_child_6.h

```
#ifndef __CL_CHILD_6__H
#define __CL_CHILD_6__H
#include "Cl_base.h"

class Cl_child_6 : public Cl_base//наследование класса
{
public:
   Cl_child_6(Cl_base*, string);//параметризированный конструктор
};
```

```
#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 объекта приложения
   ob_cl_application.build_tree_objects();//конструирование системы
   return ob_cl_application.exec_app();//запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 8.

Таблица 8 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4
object_7 3
endtree
FIND
object_2/object_4
SET /object_2
FIND //object_7
FIND
object_4/object_7
FIND .
FIND .object_7
FIND
object_4/object_7
MOVE .object_7
SET
object_4/object_7
MOVE //object_1
MOVE /object_3
END
	Object tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_7
	object_5
	object_3
	object_3
	object_2/object_4
	Object name:
	object_4
	Object is set:
	object_2
	//object_7
	Object is not found
	object_4/object_7
	Object name:
	object_7
	. Object name:
	object_2
	.object_7 Object
	name: object_7
	object_4/object_7
	Object name:
	object_7
	.object_7
	Redefining the head
	object failed
	Object is set:
	object_7
	//object_1
	Dubbing the names of
	subordinate objects
	New head object:
	object_3
	Current object
	hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_5
		Object 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_12 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_2 object_4 The head object /object_12 is not found	Object tree rootela object_1 object_2 object_4 The head object /object_12 is not found
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_8 FIND //object_15 FIND object_4/object_7 FIND . FIND .object_7 FIND	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 The object was not found at the specified coordinate /object_8 //object_15 Object is not found object_4/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 The object was not found at the specified coordinate /object_8 //object_15 Object is not found object_4/object_7

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
object_4/object_7 MOVE .object_3 SET object_6/object_7 MOVE //object_1 MOVE /object_3 END	Object is not found . Object name: rootela .object_7 Object name: object_7 object_4/object_7 Object is not found .object_3 Redefining the head object failed The object was not found at the specified coordinate object_6/object_7 //object_1 Redefining the head object failed /object_3 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 is not found . Object name: rootela .object_7 Object name: object_7 object_4/object_7 Object is not found .object_3 Redefining the head object failed The object was not found at the specified coordinate object_6/object_7 //object_1 Redefining the head object failed /object_3 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 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_7/object_4 END	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_7/object_4 Object is not found Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_7	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 object_7/object_4 Object is not found Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_7

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_5 object_3 object_3	object_5 object_3 object_3
root / 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_6 DELETE //object_2 DELETE //object_15 END	Object tree root 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_6 The object /object_2 has been deleted Current object hierarchy tree root object_1 object_7 object_3	Object tree root 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_6 The object /object_2 has been deleted Current object hierarchy tree root 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 SET /object_2 MOVE /object_2 MOVE rootela MOVE //object_5 END	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 Object is set: object_2 /object_2 Redefining the head object failed rootela Head object is not found //object_5 Head object is not found Current object hierarchy tree rootela object_1 object_7 object_2	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 Object is set: object_2 /object_2 Redefining the head object failed rootela Head object is not found //object_5 Head object is not found Current object hierarchy tree rootela object_1 object_7 object_2

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_4 object_7 object_5 object_3 object_3	object_4 object_7 object_5 object_3 object_3
rootela / object_1 3 / object_1 2 /object_2 object_4 3 /object_12 object_5 4 / object_2 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree	/ Dubbing the names of subordinate objects Object tree rootela object_1 The head object /object_2 is not found	/ Dubbing the names of subordinate objects Object tree rootela object_1 The head object /object_2 is not found
rootela / object_7 3 / object_7 4 /object_7 object_7 4 /object_7 object_6 2 endtree SET object_7/object_6 MOVE / END	/ Dubbing the names of subordinate objects Object tree rootela object_7 object_7 object_6 Object is set: object_6 New head object: rootela Current object hierarchy tree rootela object_7 object_7 object_6	/ Dubbing the names of subordinate objects Object tree rootela object_7 object_7 object_6 Object is set: object_6 New head object: rootela Current object hierarchy tree rootela object_7 object_7 object_6
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	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 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:

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

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_3
SET /object_3
MOVE /object_2
SET /
MOVE /object_1
SET //object_10
END
	//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
	New head object: object_3
	Object is set: object_3
	/object_2
	Dubbing the names of subordinate objects
	Object is set: rootela
	/object_1
	Redefining the head object failed
	The object was not found at the specified coordinate
	//object_10
	Current object hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_5
	object_3
	object_3
	object_7
		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
		New head object: object_3
		Object is set: object_3
		/object_2
		Dubbing the names of subordinate objects
		Object is set: rootela
		/object_1
		Redefining the head object failed
		The object was not found at the specified coordinate
		//object_10
		Current object hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_5
		object_3
		object_3
		object_7

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

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

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

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

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