МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« КЛ_3_2 Определение указателя на объект по его координате »

С тудент группы	ИКБО-13-21	Черномуров С.А.
Руководитель практики	Ассистент	Асадова Ю.С.
Работа представлена	«___»_____2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Постановка задачи

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

В составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задается в следующем виде:

/ - корневой объект;
//«имя объекта» - поиск объекта по уникальному имени от корневого (для однозначности уникальность требуется в рамках дерева);
. - текущий объект;
«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_3
ob_2
/ob_1/ob_2/ob_3
```

Если координата пустая строка или объект не найден, то вернуть нулевой указатель.

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 соблюдены.
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).

Система отрабатывает следующие команды:
SET «координата» – устанавливает текущий объект;
FIND «координата»– находит объект относительно текущего;
END – завершает функционирование системы (выполнение программы).
Изначально, корневой объект для системы является текущим.
При вводе данных в названии команд ошибок нет. Условия уникальности имен объектов для однозначной отработки соответствующих команд соблюдены.

Описание входных данных

Состав и иерархия объектов строится 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;

END – завершить функционирование системы (выполнение программы).

Команды SET и FIND вводятся произвольное число раз. Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов.

```
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
FIND                 object_2/object_4
SET                     /object_2
```

```
FIND                    //object_5
FIND                    /object_15
FIND                    .
FIND                    object_4/object_7
END
```

Описание выходных данных

Первая строка:

`Object tree`

Со второй строки вывести иерархию построенного дерева как в курсовой 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.

Если дерево построено, то далее построчно:

для команд SET если объект найден, то вывести:

`Object is set: «имя объекта»`

в противном случае:

`Object is not found: «имя текущего объекта»«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

Пример вывода иерархии дерева объектов.

Object tree

root

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_5 Object name: object_5 /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7

Метод решения

Для решения задачи используются:

- Всё, что использовалось в предыдущей версии приложения КЛ_3_1.
- Методы find и rfind класса string. Используются для разделения координаты объекта.
- Метод erase класса string. Используется для форматирования координаты объекта.
- **Класс Base:**
    - Свойства/поля:
        - Свойства name, parent, children, status взяты из предыдущей версии приложения
    - Методы:
        - Методы Base, SetName, SetParent, GetHead, GetName, PrintTree, PrintTreeAndStatus, SetStatus взяты из предыдущей версии приложения КЛ_3_1.
        - Метод ObjectByName доработан на основе предыдущей версии:
            - Функционал - параметризированный метод, возвращающий указатель на объект по его координате.
- **Класс Application:**
    - Свойства/поля:
        - Свойство (выделено из метода BuildTree класса Application для использования вне метода):
            - Наименование - ukaz;
            - Тип - указатель на объект класса Base;

        - Модификатор доступа - закрытый.
    - Методы:
        - Метод BuildTree взят из предыдущей версии приложения КЛ_3_1 (метод ObjectByName класса Base доработан таким образом, что он сам обрабатывает координату объекта. Таким образом, метод BuildTree изменять не пришлось):
            - Функционал - метод постройки дерева иерархии объектов.
        - Метод StartApp доработан на основе предыдущей версии:
            - Функционал - метод запуска приложения и ввода команд поиска и установки объектов.
- **Классы Child2, Child3, Child4, Child5, Child6:**
    - Свойства/поля:
        - Унаследованы из класса Base.
    - Методы:
        - Унаследованы из класса Base.

Иерархия наследования классов:

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	Base			Базовый класс в иерархии классов. Содержит основные поля и методы		

		Application	public		2	
		Child2	public		3	
		Child3	public		4	
		Child4	public		5	
		Child5	public		6	
		Child6	public		7	
2	Application			Класс корневого объекта (приложения)		
3	Child2			Класс объектов, подчиненных корневому объекту класса Application		
4	Child3			Класс объектов, подчиненных корневому объекту класса Application		
5	Child4			Класс объектов, подчиненных корневому объекту класса Application		
6	Child5			Класс объектов, подчиненных корневому объекту класса		

				Application		
7	Child6			Класс объектов, подчиненных корневому объекту класса Application		

Описание алгоритма

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

Класс объекта: Application

Модификатор доступа: public

Метод: StartApp

Функционал: Метод запуска приложения и ввода команд поиска и установки объектов

Параметры: Отсутствуют

Возвращаемое значение: Целочисленный тип данных - код возврата

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода StartApp класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода PrintTree текущего объекта	2	
2		Объявление с инициализацией указателя на объект класса Base current_obj=текущий объект	3	
3		Объявление строковых переменных command, coordinate	4	
4	Значения command,	Вывод на экран переноса на новую строку	5	

	coordinate считаны с клавиатуры			
			∅	Выход из цикла
5	Значение command равно "END"		∅	
			6	
6	Значение command равно "SET"	Объявление с инициализацией указателя на объект класса Base ob=значение, возвращенное методом ObjectByName с параметрами coordinate, ukaz, current_obj	7	
			9	
7	Значение ob не равно nullptr	Вывод на экран "Object is set: ", значение, возвращенное методом GetName объекта ob	8	
		Вывод на экран "Object is not found: ", значение, возвращенное методом GetName объекта current_obj, " ", значение coordinate	4	
8		Присвоение current_obj значения ob	4	
9	Значение command равно "FIND"	Объявление с инициализацией указателя на объект класса Base ob=значение, возвращенное методом ObjectByName с параметрами coordinate, ukaz, current_obj	10	
			4	
10	Значение ob не равно nullptr	Вывод на экран значения coordinate, " ", "Object name: ", значение, возвращенное методом	4	

		GetName объекта ob		
		Вывод на экран значения coordinate, " Object is not found"	4	

Класс объекта: Base

Модификатор доступа: public

Метод: ObjectByName

Функционал: Параметризированный метод, возвращающий указатель на объект по его координате

Параметры: Строковый параметр name, контейнер класса vector с значениями указателей на объекты класса Base - vec, Указатель на объект класса Base - current

Возвращаемое значение: Указатель на объект класса Base

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода ObjectByName класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	Значение name равно "/"	Объявление указателя на объект класса Base с инициализацией ob=текущий объект	2	
			3	
2	Значение, возвращенное методом GetHead объекта ob не равно nullptr	Присвоение ob значения, возвращенного методом GetHead объекта ob	2	

		Возврат методом значения ob	∅	
3	Значение name равно "."	Возврат методом значения current	∅	
			4	
4	Значение возвращенное методом size объекта name больше 1 и значение name[0] равно '/' и значение name[1] не равно '/'		5	
			33	Выход из условия
5		Объявление целочисленной переменной с инициализацией i=0	6	Использование i в качестве счетчика
6	Значение i меньше значения, возвращенного методом size объекта vec	Объявление строковой переменной с инициализацией copy=name	7	
		Возврат методом nullptr	∅	Выход из цикла
7		Объявление указателя на объект класса Base - object	8	
8		Объявление целочисленной переменной с инициализацией last_slash_pos=значение, возвращенное методом rfind с параметром "/" объекта copy	9	
9		Объявление строковой переменной с	10	

		инициализацией ob=""		
10		Объявление целочисленной переменной с инициализацией k=last_slash_pos+1	11	Использование k в качестве счетчика
11	Значение k меньше значения, возвращенного методом size объекта copy	Присвоение значению ob значения ob+copy[k]	12	
			13	Выход из цикла
12		Инкрементирование k	11	
13	Значение, возвращенное методом GetName объекта vec[i] равно ob	Присвоение object значения vec[i]	14	
			32	
14	Значение, возвращенное методом GetHead объекта object не равно nullptr	Присвоение last_slash_pos значения, возвращенного методом rfind с параметром "/" объекта copy	15	
			31	Выход из цикла с предусловием
15		Объявление строковых переменных с инициализацией ob="", ob2=""	16	
16	Значение, возвращенное методом rfind объекта copy с параметром "/" равно -	Возврат методом nullptr	∅	

	1			
			17	
17		Объявление целочисленной переменной с инициализацией f=last_slash_pos+1	18	Использование f в качестве счетчика
18	Значение f меньше значения, возвращенного методом size объекта copy	Присвоение ob значения ob+copy[f]	19	
			20	Выход из цикла
19		Инкрементирование f	18	
20		Вызов метода erase объекта copy с параметрами last_slash_pos, инкрементированное значение, возвращенное методом size объекта ob	21	
21		Присвоение last_slash_pos значения, возвращенного методом rfind с параметром "/" объекта copy	22	
22	Значение last_slash_pos не равно -1	Объявление целочисленной переменной с инициализацией p=last_slash_pos+1	23	Использование p в качестве счетчика
			25	
23	Значение p меньше значения, возвращенного	Присвоение ob2 значения ob2+copy[p]	24	

	методом size объекта copy			
			25	Выход из цикла
24		Инкрементирование p	23	
25	Значение ob2 не равно ""	Объявление целочисленной переменной с инициализацией j=0	26	Использование j в качестве счетчика
			30	
26	Значение j меньше значения, возвращенного методом size объекта vec		27	
			30	Выход из цикла
27	Значение, возвращенное методом GetName объекта vec[j] равно ob2		30	Досрочное завершение цикла
			28	
28	Значение j равно декрементированному значению, возвращенному методом size объекта vec и значение, возвращенное методом GetName объекта vec[j] не равно ob2		∅	
			29	
29		Инкрементирование j	26	
30	Значение, возвращенное методом GetName объекта object равно ob	Присвоение object значения, возвращенного методом GetHead объекта object	14	

			31	Досрочный выход из цикла с предусловием
31	Значение, возвращенное методом GetHead объекта object равно nullptr	Возврат методом значения vec[i]	∅	
			32	
32		Инкрементирование i	6	
33	Значение name[0] равно '/' и значение name[1] равно '/'	Вызов метода erase объекта name с параметрами 0, 2	34	
			37	Выход из условия
34		Объявление целочисленной переменной с инициализацией i=0	35	Использование i в качестве счетчика
35	Значение i меньше значения, возвращенного методом size объекта vec		36	
		Возврат методом nullptr	∅	Выход из цикла
36	Значение, возвращенное методом GetName объекта vec[i] равно name	Возврат методом значения vec[i]	∅	
		Инкрементирование i	35	
37	Значение, возвращенное методом size объекта name равно нулю или (значение name[0] не равно '/' и значение, возвращенное методом find объекта name с		38	

	параметром "/" больше нуля)			
		Возврат методом nullptr	∅	Выход из условия
38	Строка name начинается не с символа '/'	Присвоение name значения "/"+name	39	
			39	
39		Объявление целочисленной переменной с инициализацией i=0	40	Использование i в качестве счетчика
40	Значение i меньше значения, возвращенного методом size объекта vec		41	
		Возврат методом nullptr	∅	Выход из цикла
41		Объявление строковой переменной с инициализацией ob=""	42	
42		Объявление строковой переменной с инициадизацией copy=name	43	
43		Объявление указателя на объект класса Base - object	44	
44		Объявление целочисленной переменной с инициализацией last_slash_pos=значение, возвращенное	45	

		методом rfind объекта copy с параметром "/"		
45		Присвоение ob последней координаты из пути до объекта	46	
46	Значение, возвращенное методом GetName объекта vec[i] равно ob	Присвоение object значения vec[i]	47	
			61	Выход из условия
47	Значение, возвращенное методом GetHead объекта object не равно current	Присвоение last_slash_pos значения, возвращенного методом rfind объекта copy с параметром "/"	48	
			60	Выход из цикла с предусловием
48		Объявление строковых переменных с инициализацией ob="", ob2=""	49	
49	Значение, возвращенное методом rfind объекта copy с параметром "/" равно -1	Возврат методом nullptr	∅	
			50	
50		Присвоение ob последней координаты из пути до объекта	51	
51		Вызов метода erase объекта copy с параметрами	52	

		last_slash_pos, инкрементированное значение, возвращенное методом size объекта ob		
52		Присвоение last_slash_pos значения, возвращенного методом rfind объекта copy с параметром "/"	53	
53		Присвоение ob2 последней координаты из пути до объекта	54	
54	Значение ob2 не равно ""	Объявление целочисленной переменной с инициализацией j=0	55	
			59	
55	Значение j меньше значения, возвращенного методом size объекта vec		56	
			59	Выход из цикла
56	Значение, возвращенное методом GetName объекта vec[j] равно ob2		59	Досрочное завершение цикла
			57	
57	Значение j равно декрементированному значению, возвращенному методом size объекта	Возврат методом mullptr	∅	

	vec и значение, возвращенное методом GetName объекта vec[j] не равно ob2			
			58	
58		Инкрементирование j	55	
59	Значение, возвращенное методом GetName объекта object равно ob	Присвоение object значения, возвращенного методом GetHead объекта object	60	
			60	
60	Значение, возвращенное методом GetHead объекта object равно current	Возврат методом значения vec[i]	∅	
			61	
61		Инкрементирование i	40	

Функция: main

Функционал: Основной алгоритм программы

Параметры: Отсутствуют

Возвращаемое значение: Целочисленный тип данных - код возврата

Алгоритм функции представлен в таблице 4.

Таблица 4. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Создание объекта app класса Application путем вызова параметризированного конструктора с параметрами nullptr, ""	2	
2		Вызов метода BuildTree объекта	3	

		app		
3		Возврат функцией значения, возвращенного методом StartApp объекта app	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

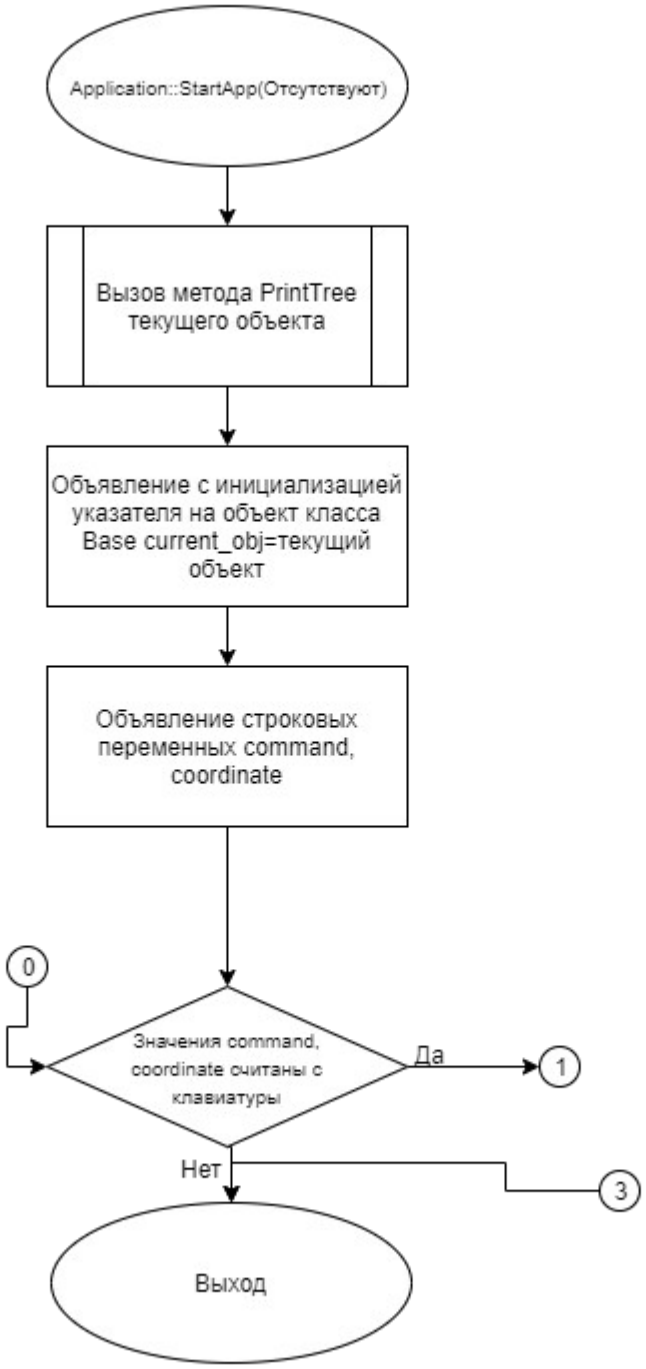

Рис. 1. Блок-схема алгоритма.

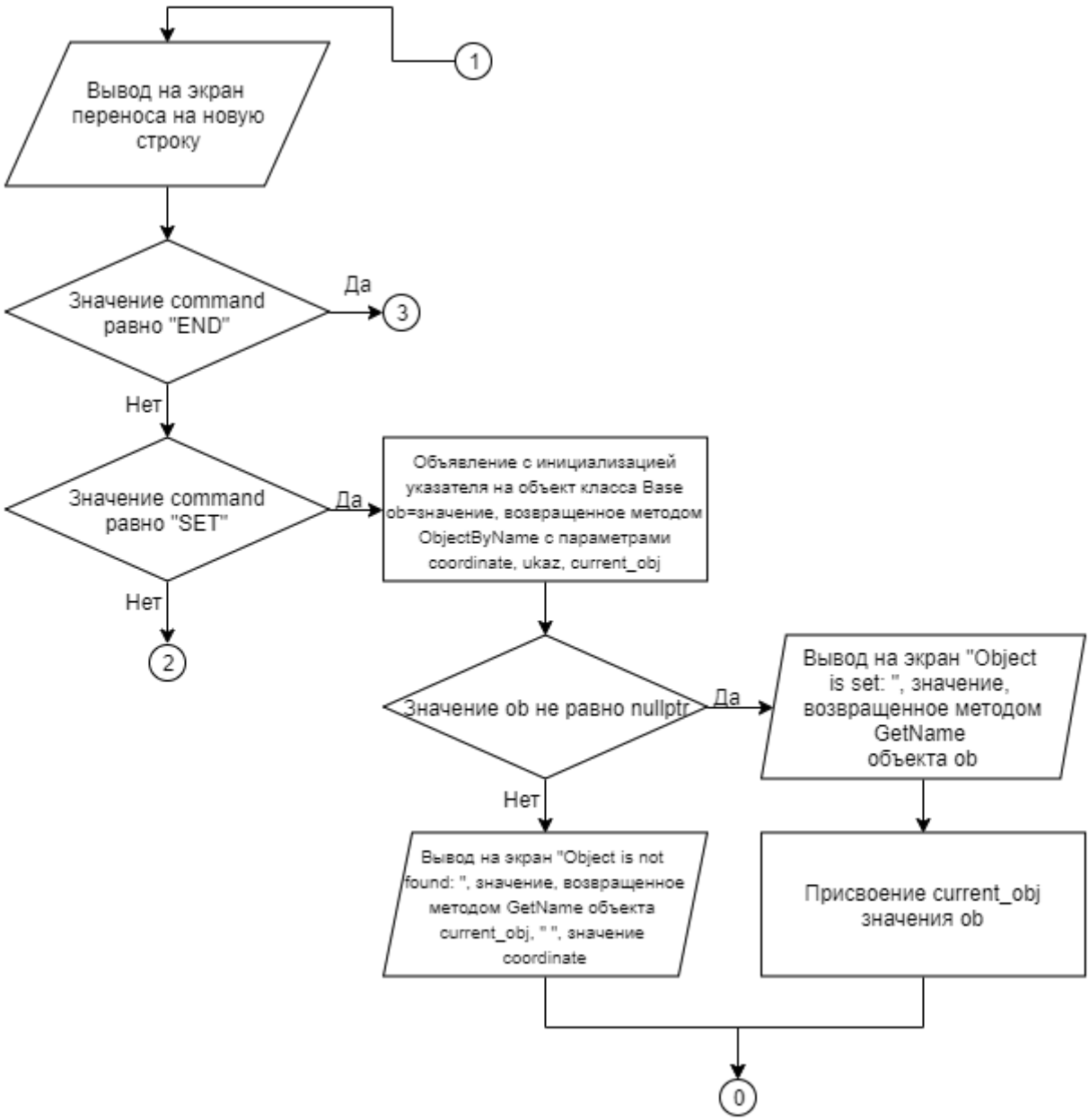

Рис. 2. Блок-схема алгоритма.

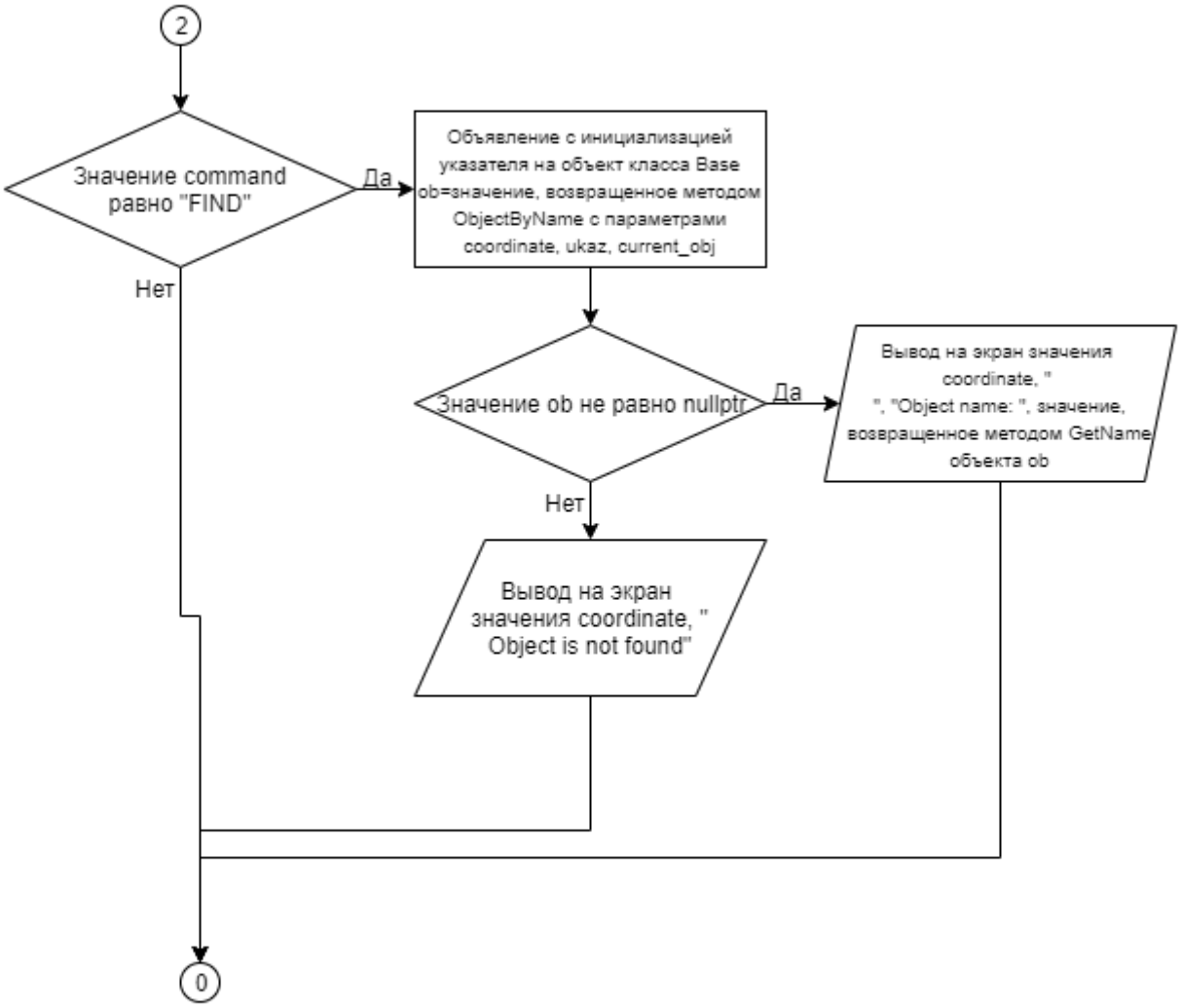

Рис. 3. Блок-схема алгоритма.

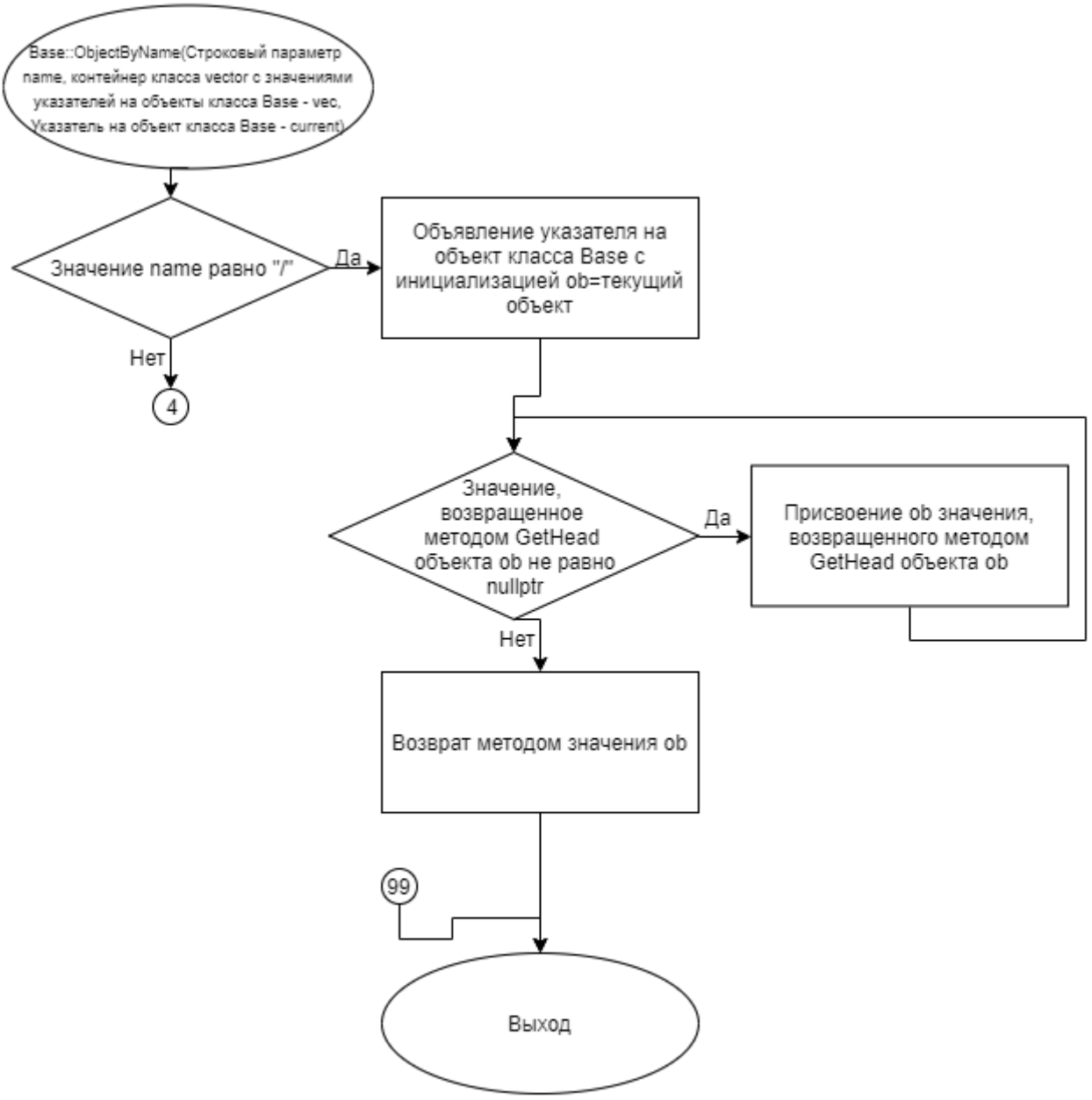

Рис. 4. Блок-схема алгоритма.

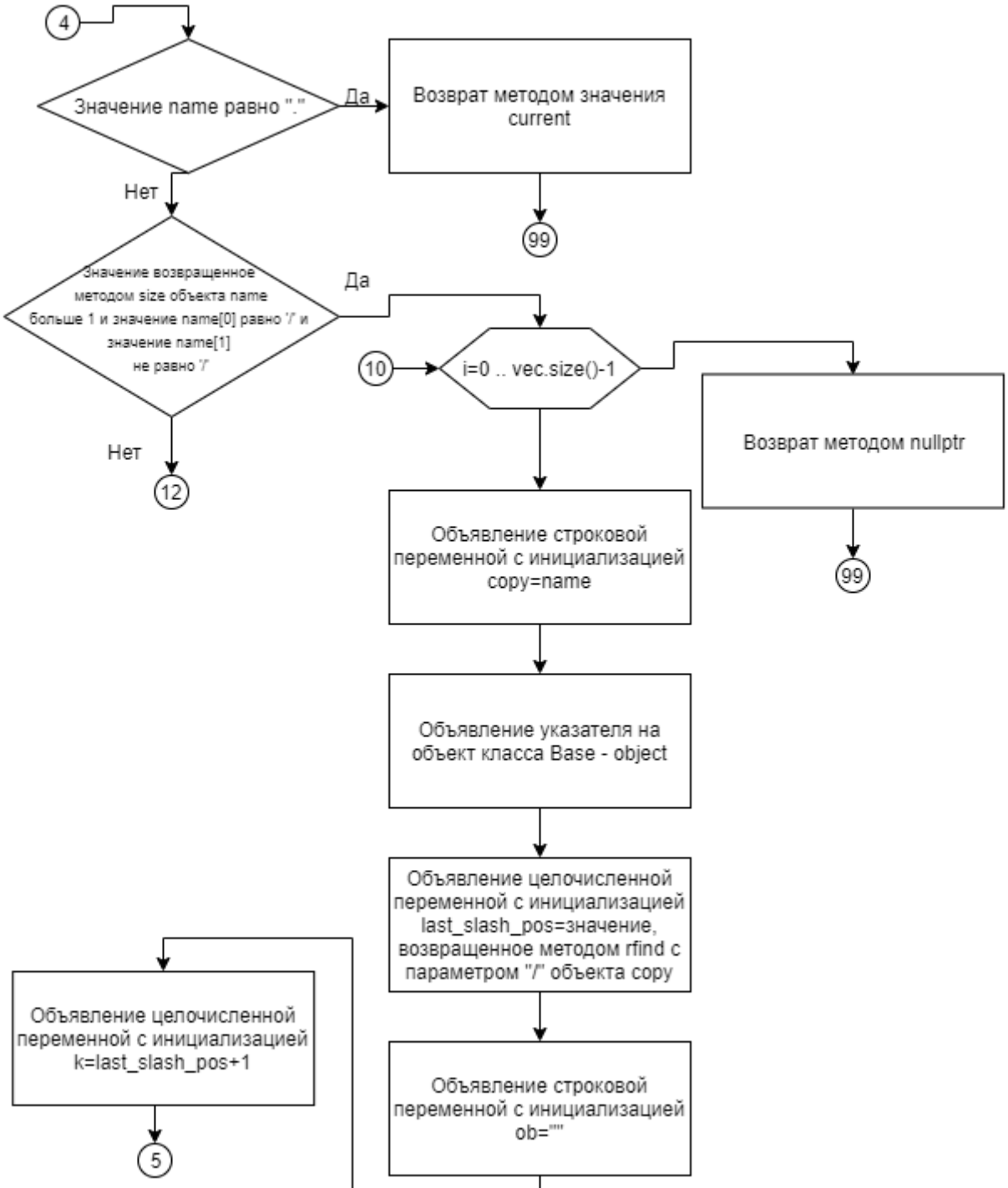

Рис. 5. Блок-схема алгоритма.

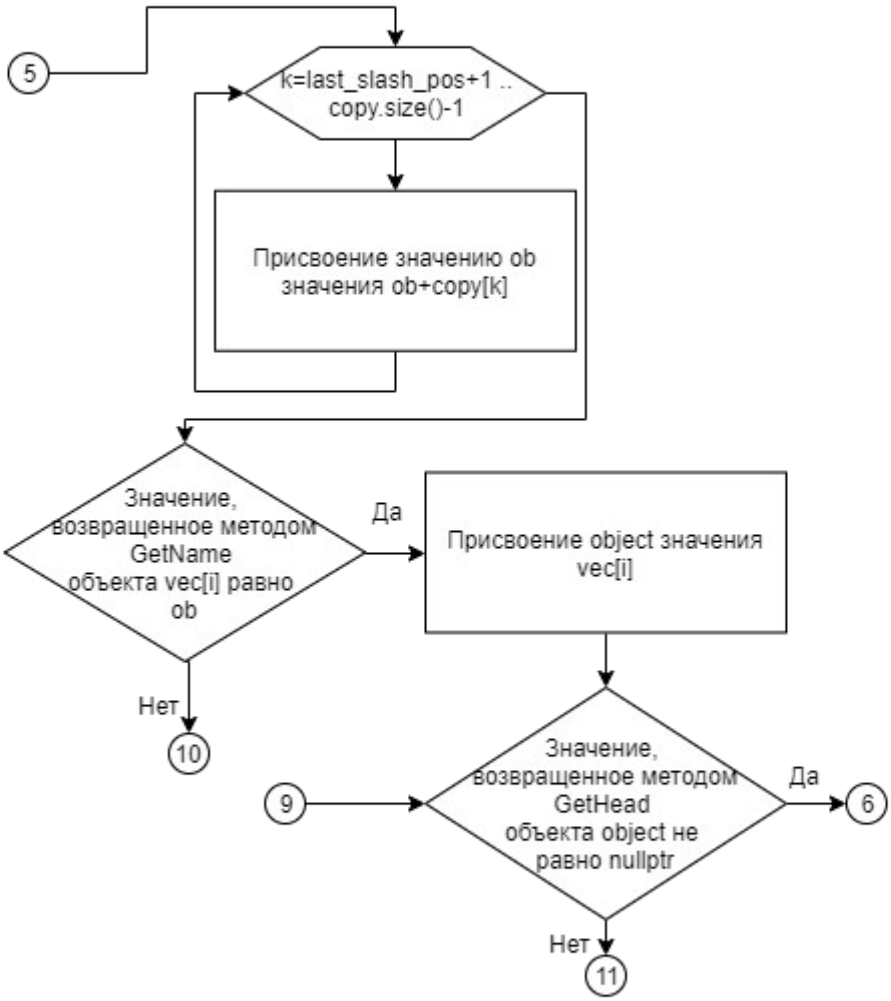

Рис. 6. Блок-схема алгоритма.

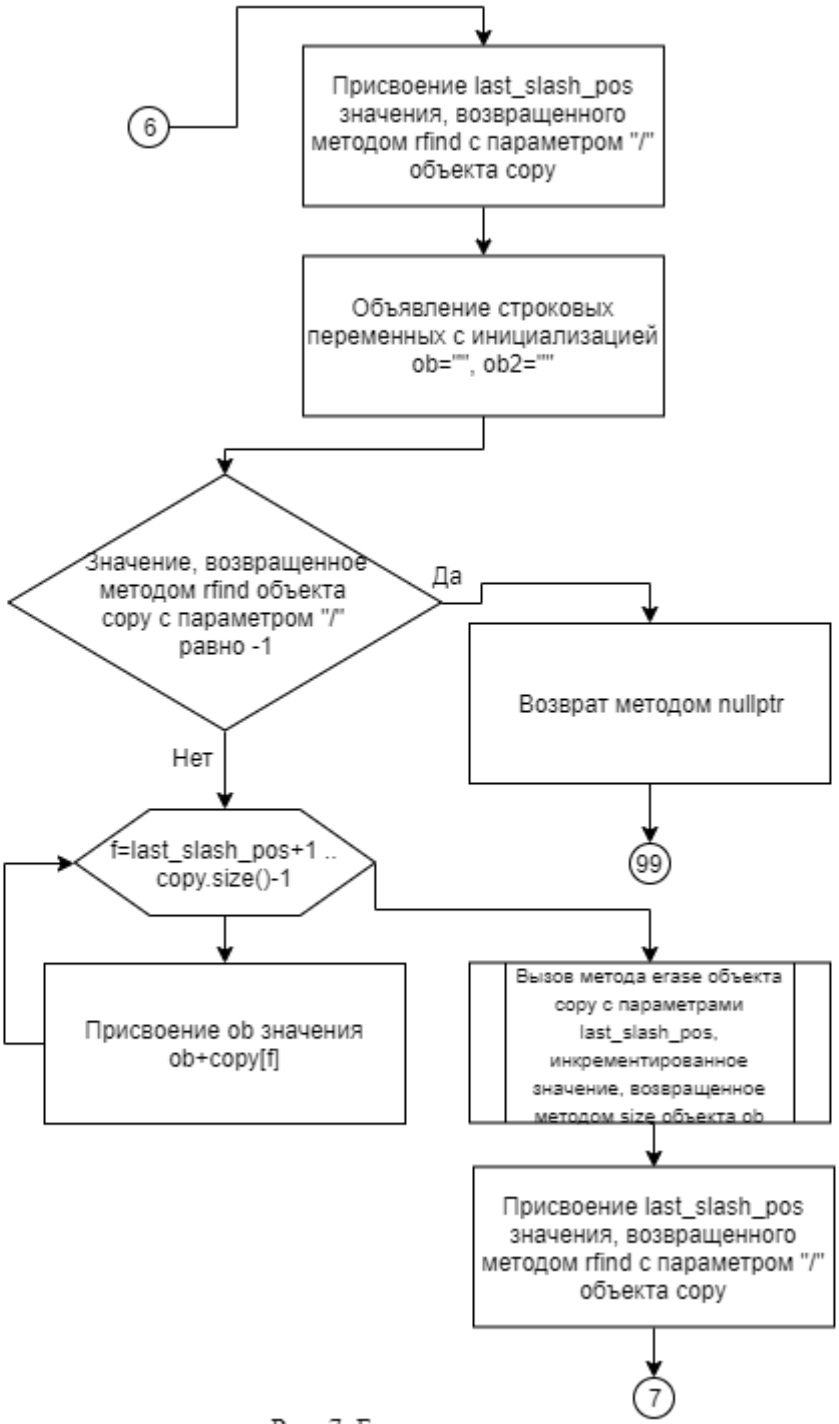

Рис. 7. Блок-схема алгоритма.

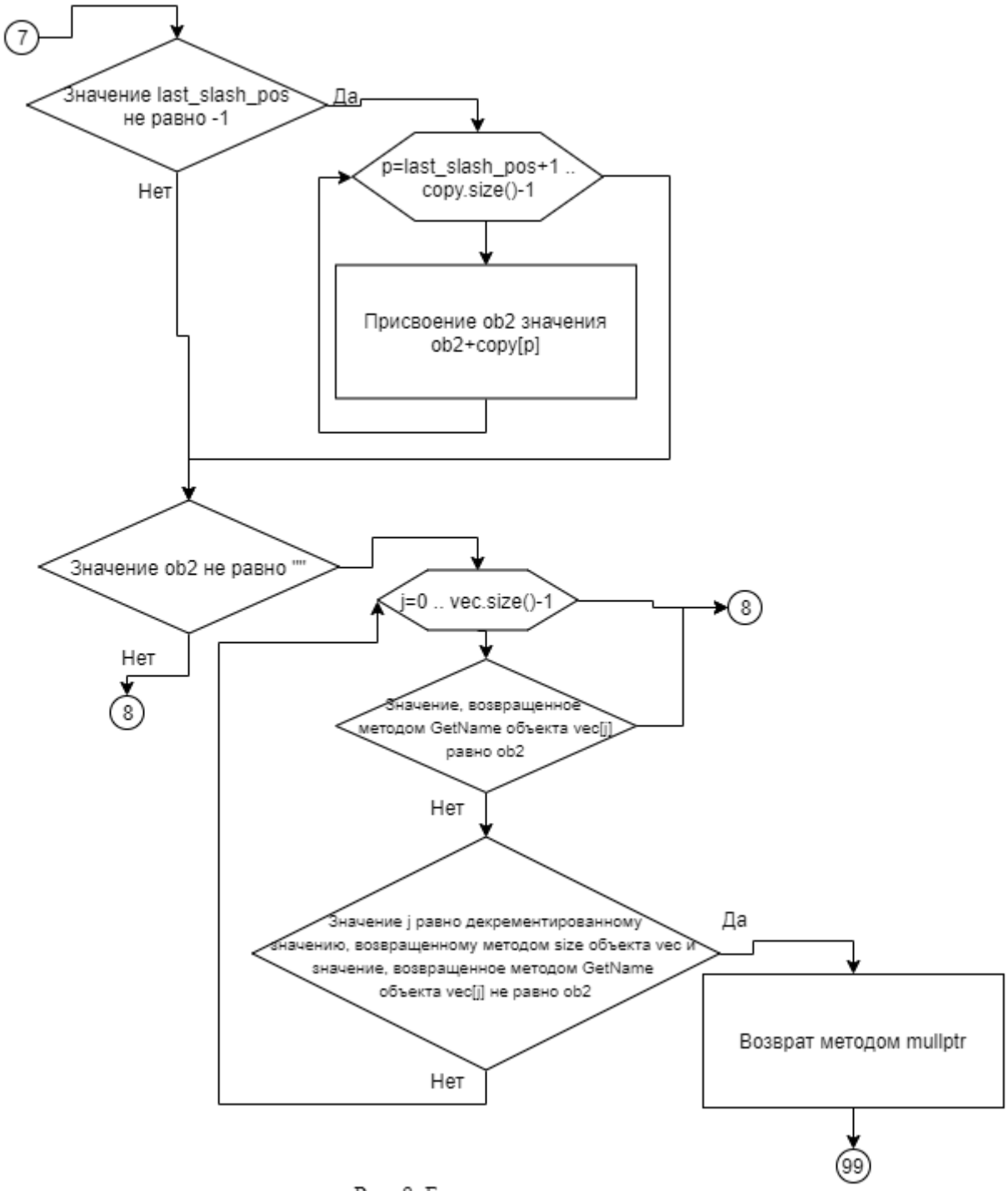

Рис. 8. Блок-схема алгоритма.

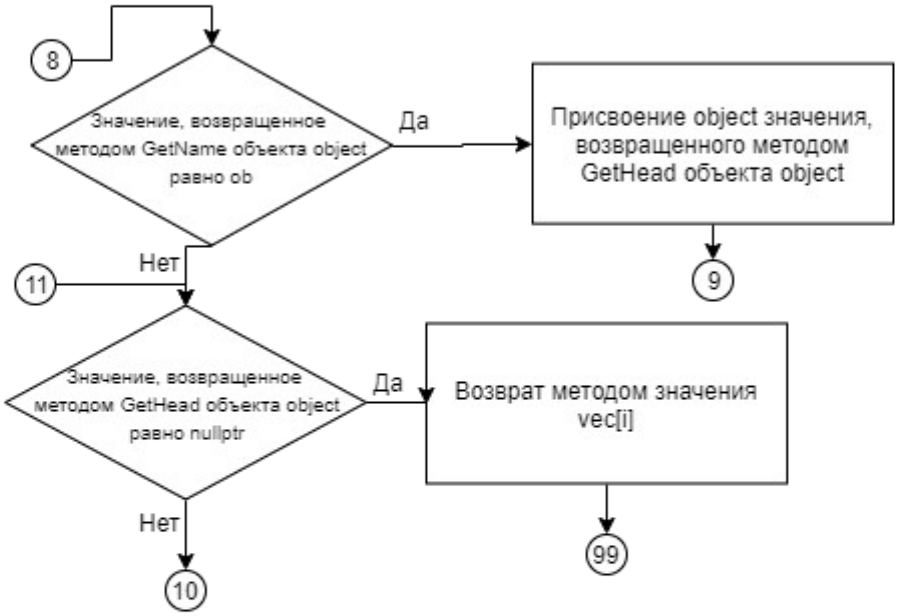

Рис. 9. Блок-схема алгоритма.

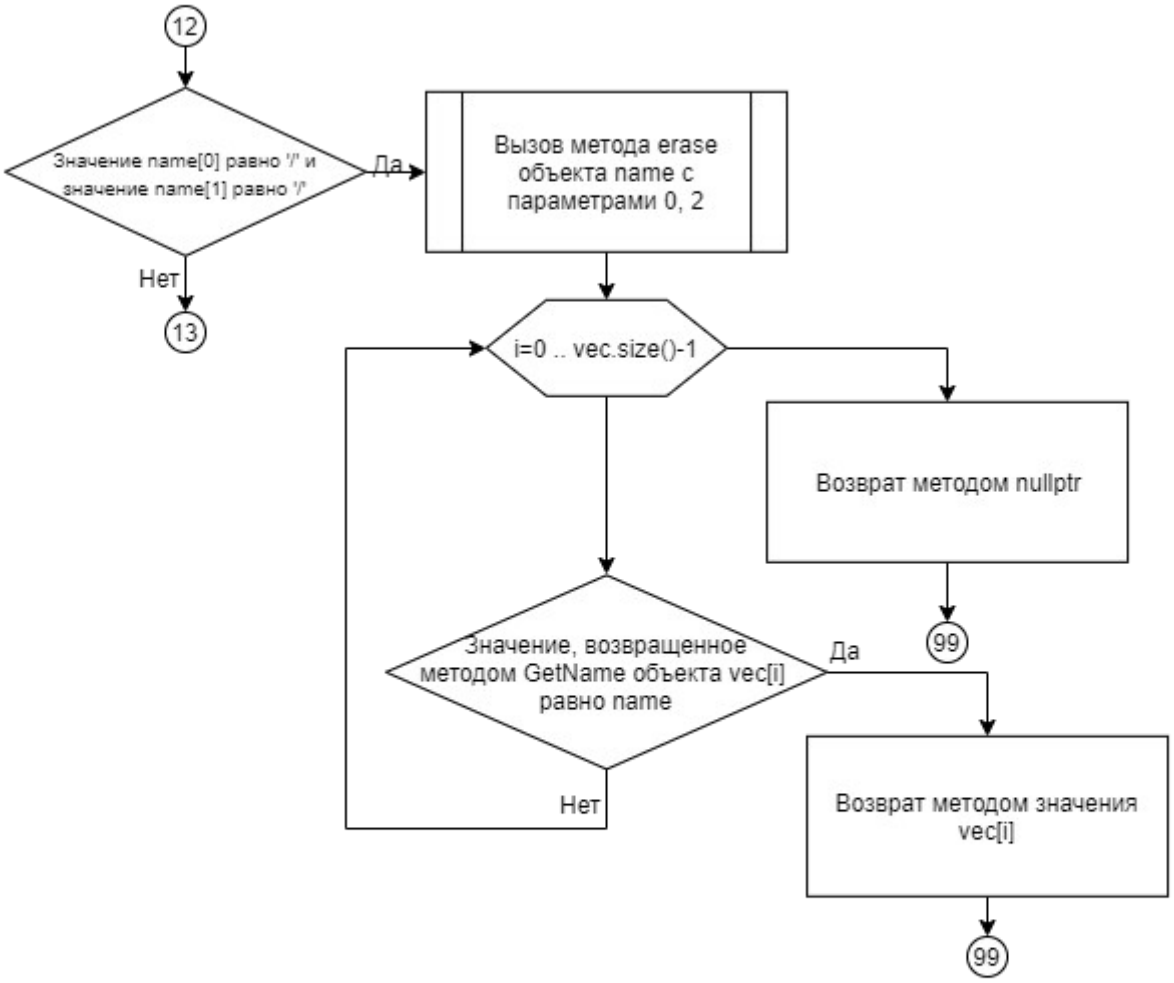

Рис. 10. Блок-схема алгоритма.

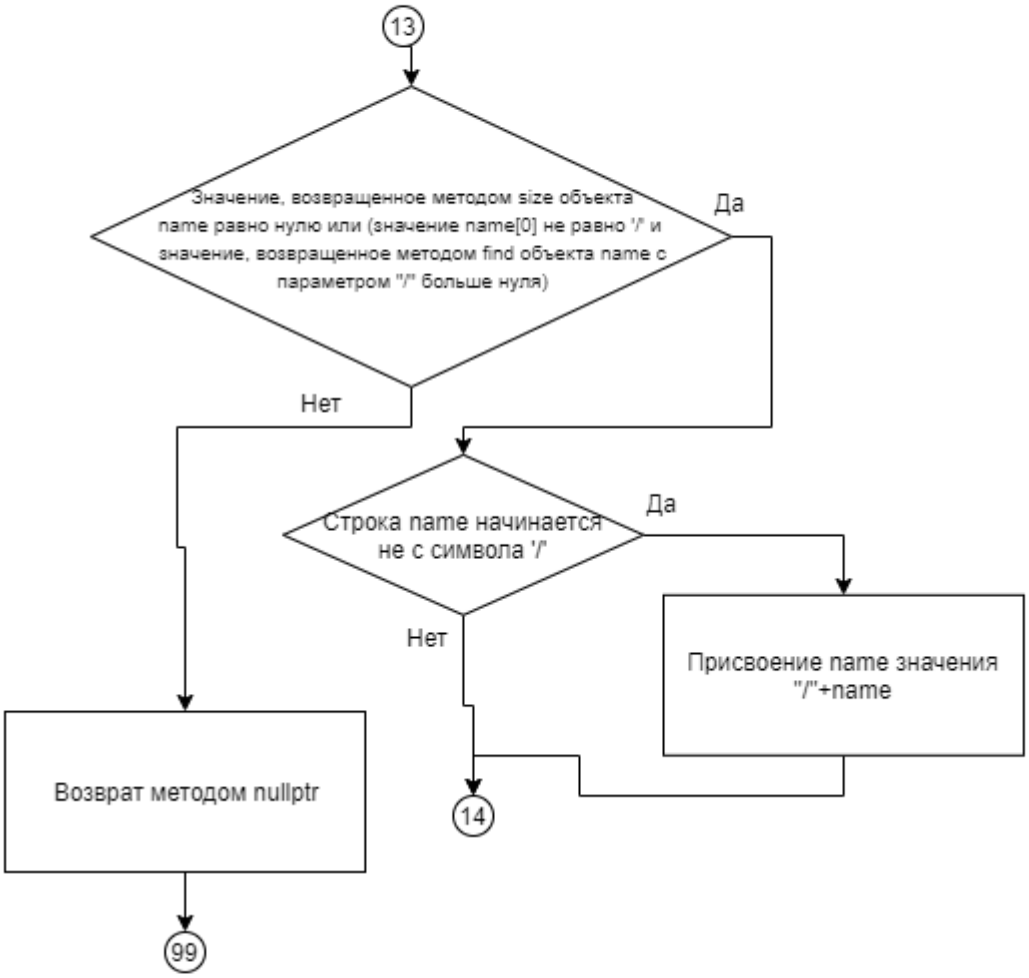

Рис. 11. Блок-схема алгоритма.

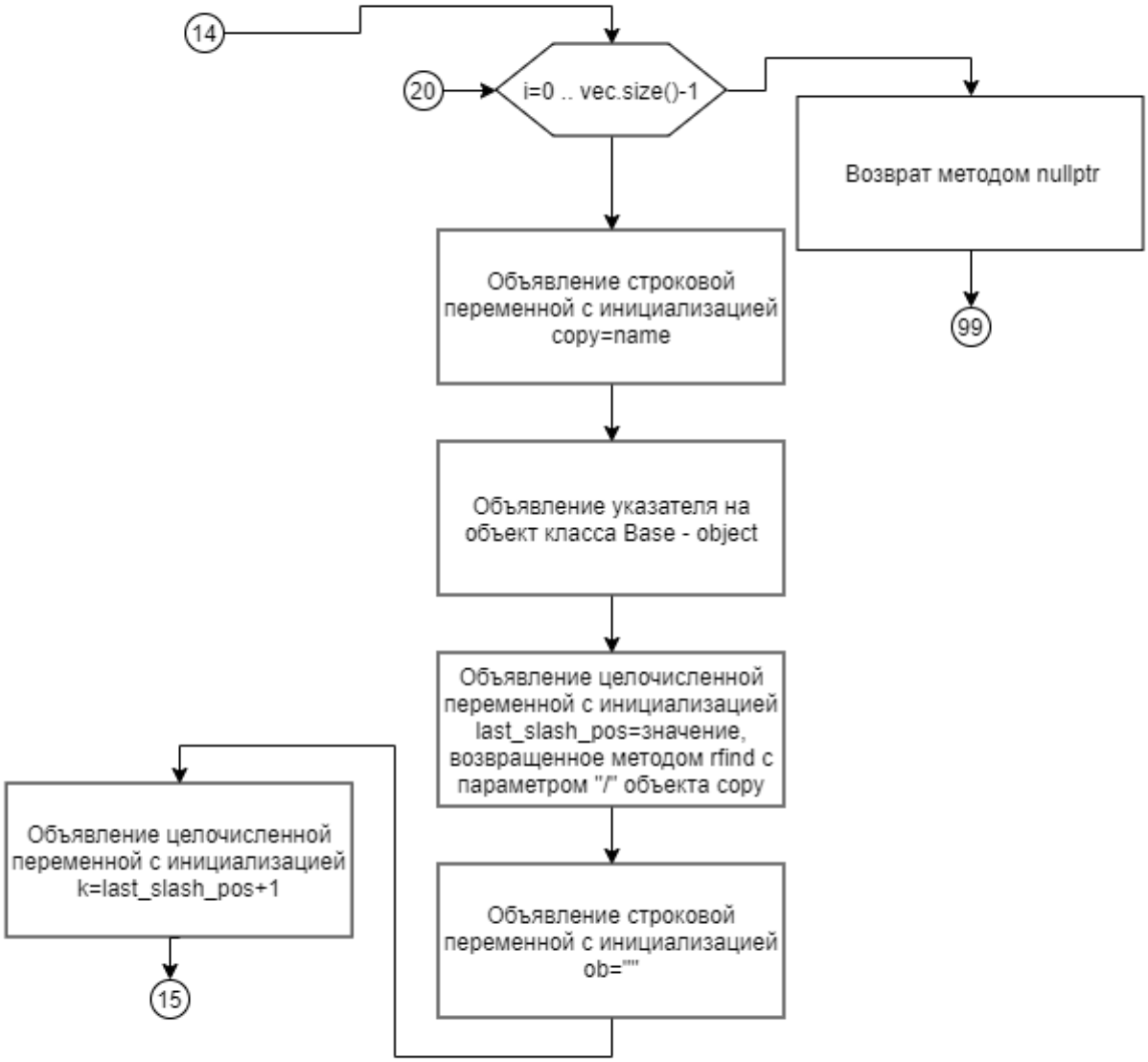

Рис. 12. Блок-схема алгоритма.

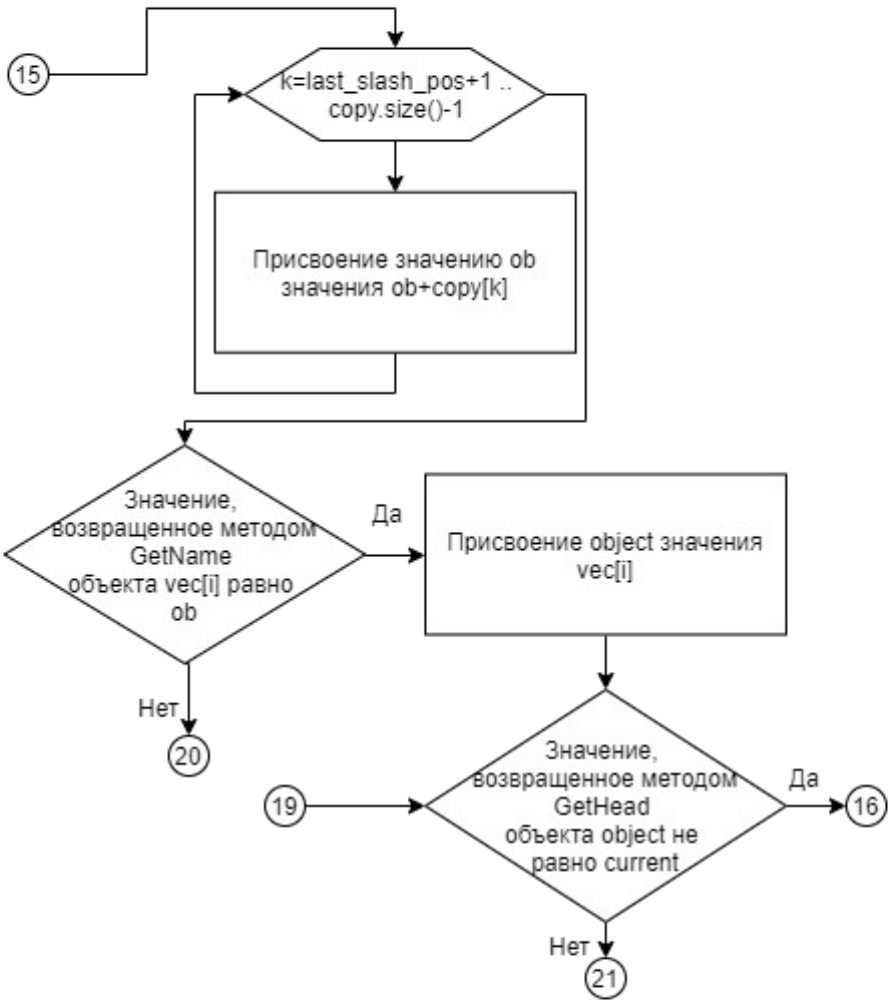

Рис. 13. Блок-схема алгоритма.

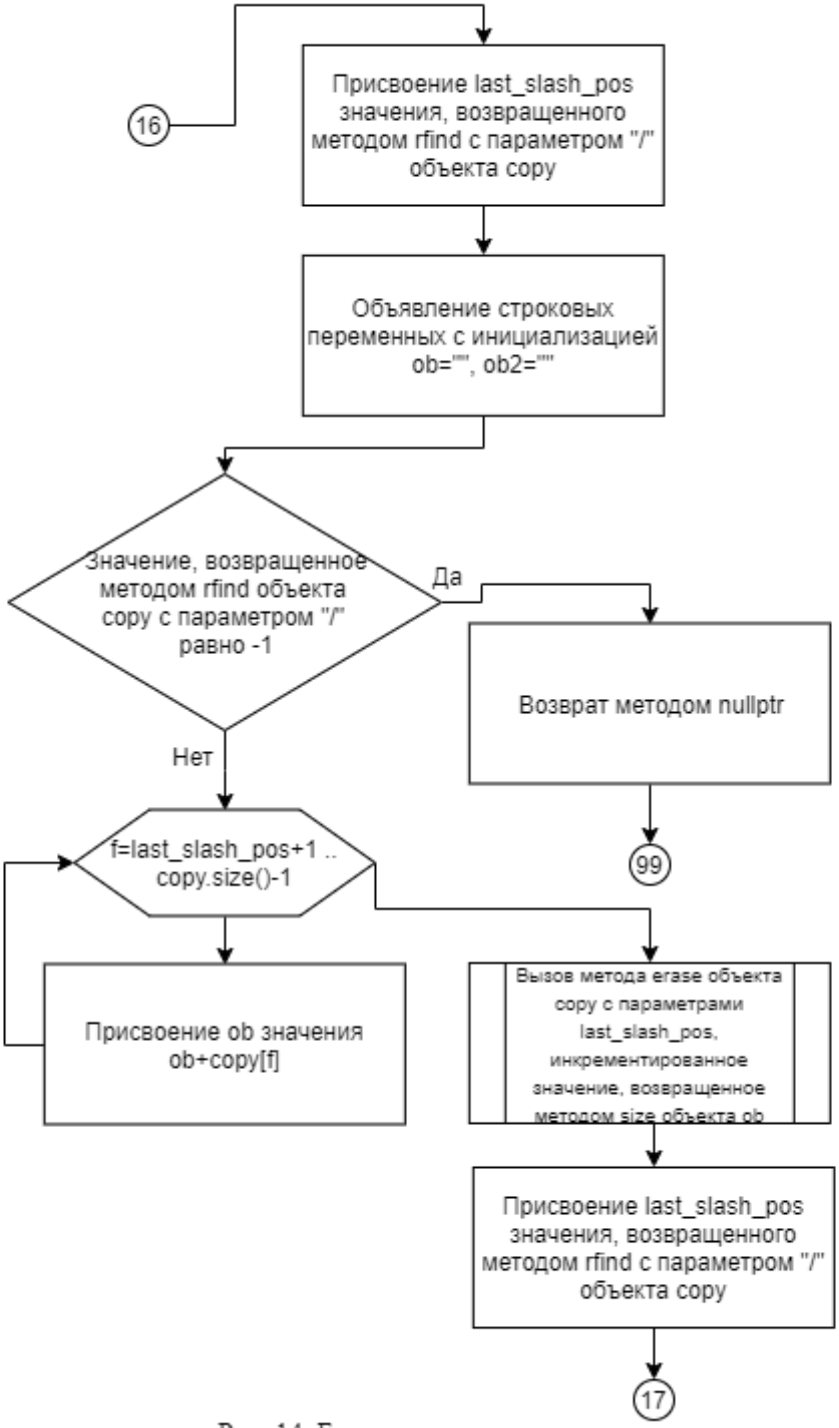

Рис. 14. Блок-схема алгоритма.

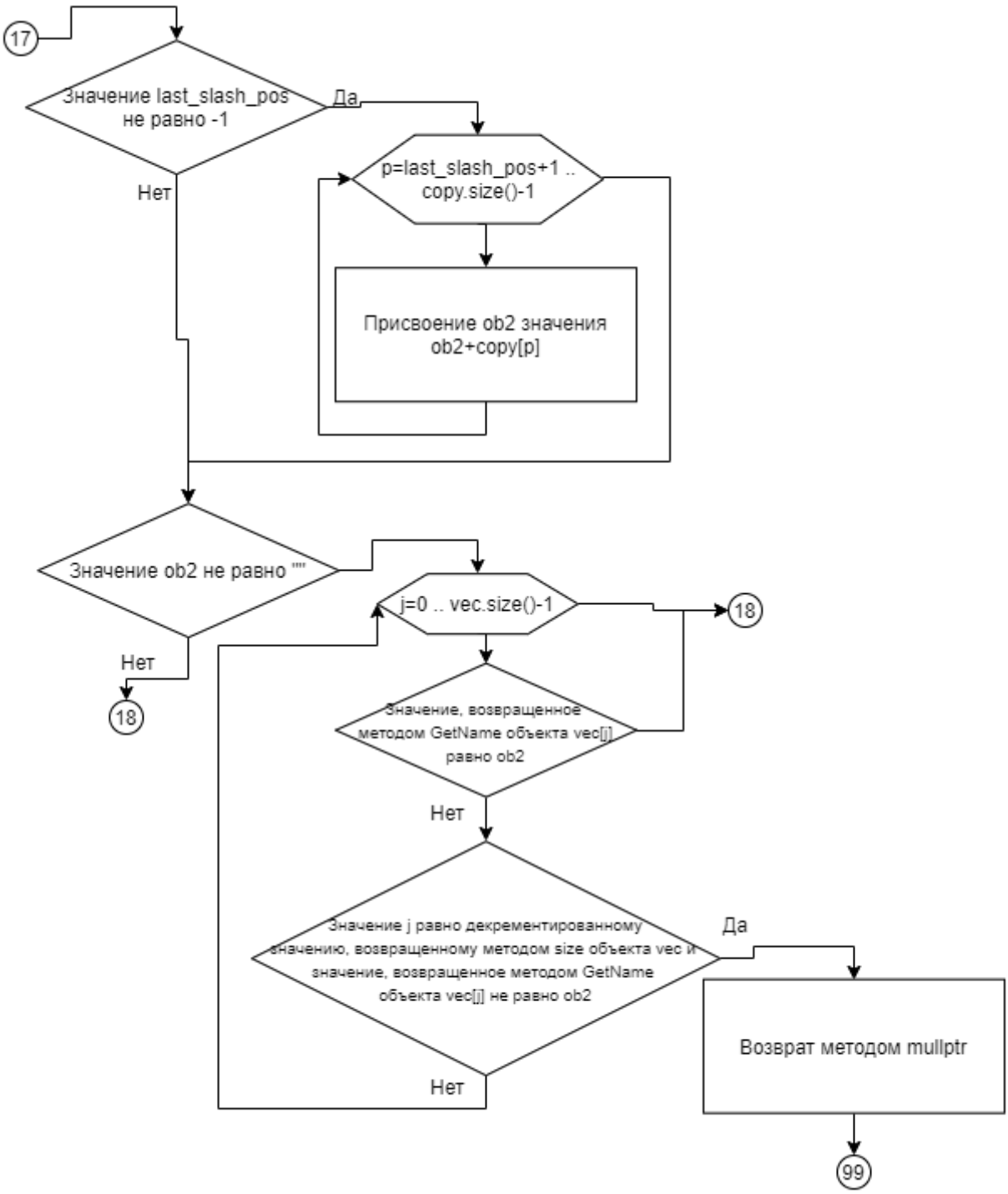

Рис. 15. Блок-схема алгоритма.

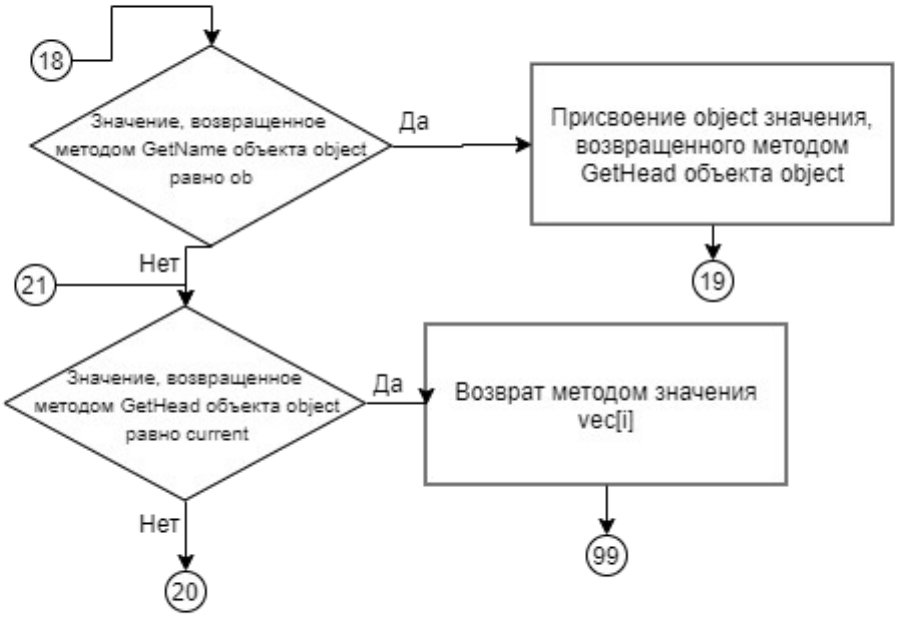

Рис. 16. Блок-схема алгоритма.

Рис. 17. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Application.cpp

```
#include "Application.h"
#include "Child2.h"
#include "Child3.h"
#include "Child4.h"
#include "Child5.h"
#include "Child6.h"
#include <iostream>
using namespace std;

void Application::BuildTree() {
        string name;
        cin >> name;

        this->SetName(name);

        Base* ob_base=this;
        Child2* child2=nullptr;
        Child3* child3=nullptr;
        Child4* child4=nullptr;
        Child5* child5=nullptr;
        Child6* child6=nullptr;



        ukaz.push_back(ob_base);

        string parentname;
        string childname;
        int child_number;

        cout<<"Object tree\n";
        cout << this->GetName();

        cin>>parentname;
                if (parentname!="endtree")
                        cin>>childname>>child_number;

        while (parentname != "endtree") {

                ob_base = ObjectByName(parentname, ukaz, this);

                if (ob_base==nullptr) {PrintTree(); cout<<"\nThe head object
"<< parentname << " is not found"; exit(0);}

                if (child_number == 2) {child2=new Child2(ob_base,childname);
```

```
ukaz.push_back(child2);}
                else
                if (child_number == 3) {child3=new Child3(ob_base,childname);
ukaz.push_back(child3);}
                else
                if (child_number == 4) {child4=new Child4(ob_base,childname);
ukaz.push_back(child4);}
                else
                if (child_number == 5) {child5=new Child5(ob_base,childname);
ukaz.push_back(child5);}
                else
                if (child_number == 6) {child6=new Child6(ob_base,childname);
ukaz.push_back(child6);}

                cin>>parentname;
                if (parentname!="endtree")
                        cin>>childname>>child_number;

        }

// << this->GetName() << "  ";
}

int Application::StartApp() {
        this->PrintTree();

        Base* current_obj=this;
        string command;
        string coordinate;
        while (cin >> command >> coordinate){
                cout<<"\n";
                if (command=="END") return 0;
                else
                if (command=="SET"){
                        Base* ob = ObjectByName(coordinate,ukaz, current_obj);
                        if (ob!=nullptr) { cout<<"Object is set: "<<ob-
>GetName(); current_obj=ob; }
                        else cout<<"Object is not found: "<<current_obj-
>GetName()<<" "<<coordinate;

                }
                else
                if (command=="FIND"){
                        Base* ob = ObjectByName(coordinate,ukaz, current_obj);
                        if (ob!=nullptr) cout<<coordinate<<"     "<<"Object
name: "<<ob->GetName();
                        else cout<<coordinate<<"     Object is not found";
                }
        }
        //this->PrintTreeAndStatus();

        return 0;
}
```

Файл Application.h

```
#ifndef APP_H
#define APP_H

#include "Base.h"

class Application : public Base {
        using Base::Base;
private:
        vector <Base*> ukaz;
public:
        void BuildTree();
        int StartApp();
};

#endif
```

Файл Base.cpp

```
#include "Base.h"
#include <iostream>
using namespace std;


Base::Base(Base* parent, string name) {
        SetName(name);
        SetParent(parent);
        if (parent) { // !=nullptr
                parent->children.push_back(this);
        }
}

void Base::SetName(string name) {
        this->name = name; // наименование объекта
}

string Base::GetName() {
        return name; // возврат имени объекта
}

void Base::PrintTree() {


        for (int i = 0; i < children.size(); i++) {
                int flag=0;
                Base* ob=this;
                string s="    ";

                while (ob->GetHead()!=nullptr){
                        flag++;
                        ob=ob->GetHead();
                }

                for (int j=0;j<flag;j++)
```

```
                        s+="    ";

                cout<<"\n"<<s;

                cout<<children[i]->GetName();

                children[i]->PrintTree();
        }
        }


Base* Base::GetHead() {
        return  parent; // возврат указателя на головной объект
}

void Base::SetParent(Base* parent) {
        this->parent = parent;
}

Base* Base::ObjectByName(string name, vector <Base*> &vec, Base* current){
//получение объекта по имении в дереве иерархи
        if (name=="/"){
                Base* ob=this;
                while (ob->GetHead()!=nullptr){
                        ob=ob->GetHead();
                }
                return ob;
        }
        else
        if (name=="."){
                return current;
        }
        else
        if (name.size()>1 && name[0]=='/' && name[1]!='/'){
                for (int i=0;i<vec.size();i++){
                        string copy=name;
                        Base* object;
                        int last_slash_pos=copy.rfind("/");

                        string ob="";
                        for (int k=last_slash_pos+1;k<copy.size();k++)
                                ob+=copy[k];

                        if (vec[i]->GetName()==ob){
                                object=vec[i];

                                while (object->GetHead() != nullptr){
                                        last_slash_pos=copy.rfind("/");
                                        string ob="", ob2="";
                                        if (copy.rfind("/")==-1) return
nullptr;
                                        for (int
f=last_slash_pos+1;f<copy.size();f++)
                                                ob+=copy[f];

                                        copy.erase(last_slash_pos, ob.size()
+1);


                                        last_slash_pos=copy.rfind("/");
                                        if (last_slash_pos!=-1)
```

```
                                        for (int
p=last_slash_pos+1;p<copy.size();p++)
                                                ob2+=copy[p];

                                        if (ob2!=""){
                                                for (int j=0;j<vec.size();j++)
{
                                                        if (vec[j]-
>GetName()==ob2) break;
                                                        if (j==vec.size()-1 &&
vec[j]->GetName()!=ob2) return nullptr;
                                                }
                                        }
                                        if (object->GetName()==ob)
object=object->GetHead();
                                        else break;
                                }
                                if (object->GetHead()==nullptr) return vec[i];
                        }
                }
                return nullptr;
        }
        if (name[0]=='/' && name[1]=='/'){
                name.erase(0,2);
                for (int i=0;i<vec.size();i++)
                        if (vec[i]->GetName()==name) return vec[i];
                return nullptr;
        }
        else
        if (name.size()==0 || (name[0]!='/' && name.find("/")>0)){
                if (name.find("/")!=0) name="/"+name;

                for (int i=0;i<vec.size();i++){
                        string copy=name;
                        Base* object;
                        int last_slash_pos=copy.rfind("/");

                        string ob="";
                        for (int k=last_slash_pos+1;k<copy.size();k++)
                                ob+=copy[k];

                        if (vec[i]->GetName()==ob){
                                object=vec[i];

                                while (object->GetHead() != current){
                                        last_slash_pos=copy.rfind("/");
                                        string ob="", ob2="";
                                        if (copy.rfind("/")==-1) return
nullptr;
                                        for (int
f=last_slash_pos+1;f<copy.size();f++)
                                                ob+=copy[f];

                                        copy.erase(last_slash_pos, ob.size()
+1);

                                        last_slash_pos=copy.rfind("/");
```

```
                                        if (last_slash_pos!=-1)
                                        for (int
p=last_slash_pos+1;p<copy.size();p++)
                                                ob2+=copy[p];

                                        if (ob2!=""){
                                                for (int j=0;j<vec.size();j++)
{
                                                        if (vec[j]-
>GetName()==ob2) break;
                                                        if (j==vec.size()-1 &&
vec[j]->GetName()!=ob2) return nullptr;
                                                }
                                        }
                                        if (object->GetName()==ob)
object=object->GetHead();
                                        else break;
                                }
                                if (object->GetHead()==current) return vec[i];
                        }
                }
                return nullptr;
        }

        else return nullptr;
}
void Base::SetStatus(int status){
        if (status==0){
                for (int i = 0; i < children.size(); i++)
                        children[i]->SetStatus(0);
        }
        else{
                Base* ob=this;
                while (ob->GetHead()!=nullptr){
                        ob=ob->GetHead();
                        if (ob->status==0) return;
                }
                this->status=status;
        }
}

void Base::PrintTreeAndStatus(){
        for (int i = 0; i < children.size(); i++) {
                int flag=0;
                Base* ob=this;
                string s="    ";

                while (ob->GetHead()!=nullptr){
                        flag++;
                        ob=ob->GetHead();
                }

                for (int j=0;j<flag;j++)
                        s+="    ";

                cout<<"\n"<<s;

                cout<<children[i]->GetName()<<" is ";
                if (children[i]->status!=0) cout<<"ready";
                else cout<<"not ready";
```

```
                children[i]->PrintTreeAndStatus();
        }
}
/*Base* Base::ObjectByName(string name, vector <Base*> &vec, Base* current,
Base* root, int depth, Base* search_ob){ //получение объекта по имении в
дереве иерархи
        if (name=="/"){
                Base* ob=this;
                while (ob->GetHead()!=nullptr){
                        ob=ob->GetHead();
                }
                return ob;
        }
        else
        if (name=="."){
                return current;
        }
        else{
                int obj_pos;
                obj_pos=name.rfind("/");

                string ob_child="";
                for (int i=obj_pos+1;i<name.size();i++)
                        ob_child+=name[i];

                name.erase(obj_pos);

                obj_pos=name.rfind("/");
                string ob_parent="";
                for (int i=obj_pos+1;i<name.size();i++)
                        ob_parent+=name[i];

                if (obj_pos!=0)
                for (int i=0;i<vec.size();i++){
                if (vec[i]->GetName()==ob_child && vec[i]->GetHead()-
>GetName()==ob_parent)
                        if (depth==0) ObjectByName(name,vec,current,root,
depth+1, vec[i]);
                        else ObjectByName(name,vec,current,root, depth+1,
search_ob);
                else return nullptr;
        }
                return search_ob;

}
}
*/

/*      if (name.size()>1 && name.find("/")==0){
                if (depth==0){
                        int find_pos=name.rfind("/");
                        for (int i=find_pos+1;i<name.size();i++)
                                search_obj+=name[i];
                //      cout<<search_obj<<"\n";

                }
```

```
                name.erase(0,1);
                int obj_pos=name.find("/");
                string parent_obj="";
                for (int i=0;i<obj_pos;i++)
                        parent_obj+=name[i];
                //cout<<parent_obj<<"\n";
                name.erase(0,obj_pos);
                //cout<<name<<"\n";
                for (int i=0;i<vec.size();i++){
                //      cout<<vec[i]->GetName()<<" ";
                        if (vec[i]->GetName()==parent_obj)
{ObjectByName(name,vec[i]->children,current, depth+1, search_obj); }
                        //if (vec[i]->GetName()!=parent_obj && i==vec.size()-
1) return nullptr;
                        for (int j=0;j<vec[i]->children.size();j++)
                                if (vec[i]->children[j]-
>GetName()==search_obj) return vec[i]->children[j];
                }
                return nullptr;
}
        else
        {
                if (depth==0){
                        int find_pos=name.rfind("/");
                        for (int i=find_pos+1;i<name.size();i++)
                                search_obj+=name[i];
                name="/"+name;

                }
                name.erase(0,1);
                int obj_pos=name.find("/");
                string parent_obj="";
                for (int i=0;i<obj_pos;i++)
                        parent_obj+=name[i];

                name.erase(0,obj_pos);
                for (int i=0;i<current->children.size();i++){
                        if (current->children[i]->GetName()==parent_obj)
ObjectByName(name,current->children[i]->children,current, depth+1,
search_obj);
                        if (current->children[i]->GetName()==search_obj)
return current->children[i];
                }
                return nullptr;
        }*/

/*int obj_pos;
                obj_pos=name.rfind("/");
                name.erase(0,obj_pos+1);
                for (int i=0;i<vec.size();i++)
                        if (vec[i]->GetName()==name) return vec[i];
                return nullptr;*/

/*//if (name=="") return current;
                if (name.find("/")==0) name.erase(0,1);
                if (name.rfind("/")!=name.size()-1) name+="/";
                int obj_pos=name.find("/");

                string ob="";
                for (int i=0;i<obj_pos;i++)
```

```
                        ob+=name[i];
                //cout<<name<<"    "<<ob<<"\n";
                cout<<ob<<"\n";
                if (current->GetName()==ob) return current;
                name.erase(0,obj_pos);
                cout<<current->GetName()<<"\n\n";
                for (int i=0;i<vec.size();i++){
                        cout<<vec[i]->GetName()<<" ";
                        if (vec[i]->GetName()==ob) ObjectByName(name, vec[i]-
>children, vec[i]);

                }
                cout<<"\n";
                return nullptr;*/

/*while (object->GetHead()!=nullptr){
                                int obj_pos=copy.rfind("/");
                                string ob="";
                                for (int i=obj_pos+1;i<copy.size();i++)
                                        ob+=name[i];
                                //cout<<ob<<"  ";
                                copy.erase(obj_pos,ob.size()+1);
                                if (object->GetName()!=ob) break;
                                else{
                                        object=object->GetHead();
                                }

                        }
                        if (object->GetHead()==nullptr) return vec[i];*/

/*for (int i=0;i<vec.size();i++){
                        string copy=name;
                        Base* object;
                        int obj_pos=copy.rfind("/");
                        string ob="";
                        for (int i=obj_pos+1;i<copy.size();i++)
                                ob+=copy[i];
                        //copy.erase(obj_pos,ob.size()+1);

                        if (vec[i]->GetName()==ob){
                                //cout<<vec[i]->GetName();
                                object=vec[i];

                                while (object->GetHead()!=nullptr){
                                        copy.erase(obj_pos,ob.size()+1);
                                        obj_pos=copy.rfind("/");
                                        string ob="";
                                        for (int i=obj_pos+1;i<copy.size();i+
+)
                                                ob+=copy[i];
                                        if (object->GetName()==ob)
object=object->GetHead();
                                        else break;
                                }
                                if (object->GetHead()==nullptr) return vec[i];
                        }
```

```
                }
                return nullptr;
                }*/

/*for (int i=0;i<vec.size();i++){
                        string copy=name;
                        Base* object;
                        int obj_pos=copy.rfind("/");
                        string ob="";
                        for (int i=obj_pos+1;i<copy.size();i++)
                                ob+=copy[i];
                        //copy.erase(obj_pos,ob.size()+1);

                        if (vec[i]->GetName()==ob){
                                //cout<<vec[i]->GetName();
                                object=vec[i];

                                while (object->GetHead()!=nullptr){

                                        int obj_pos=copy.rfind("/");
                                        string ob="";
                                        for (int i=obj_pos+1;i<copy.size();i+
+)
                                                ob+=copy[i];
                                        copy.erase(obj_pos,ob.size()+1);
                                        if (object->GetName()==ob)
object=object->GetHead();
                                        else break;

                                }
                                if (object->GetHead()==nullptr) return vec[i];
                        }

                }
                return nullptr;
                }*/
```

Файл Base.h

```
#ifndef BASE_H
#define BASE_H

#include <string>
#include <vector>
using namespace std;

class Base {
        protected:
                string name;
                int status=0;
                vector <Base*> children;
        private:
                Base* parent;

        public:
                Base(Base* parent, string name);
```

```
                void SetName(string name);
                void PrintTree();
                void SetParent(Base* parent);

                Base* GetHead();
                Base* ObjectByName(string name, vector <Base*> &vec, Base*
current);

                string GetName();

                void PrintTreeAndStatus();
                void SetStatus(int status);

};

#endif
```

Файл Child2.h

```
#ifndef CHILD2_H
#define CHILD2_H

#include "Application.h"

class Child2 : public Base {
        using Base::Base;
};

#endif
```

Файл Child3.h

```
#ifndef CHILD3_H
#define CHILD3_H

#include "Application.h"

class Child3 : public Base {
        using Base::Base;
};

#endif
```

Файл Child4.h

```
#ifndef CHILD4_H
#define CHILD4_H

#include "Application.h"

class Child4 : public Base {
        using Base::Base;
};

#endif
```

Файл Child5.h

```
#ifndef CHILD5_H
#define CHILD5_H

#include "Application.h"

class Child5 : public Base {
        using Base::Base;
};

#endif
```

Файл Child6.h

```
#ifndef CHILD6_H
#define CHILD6_H

#include "Application.h"

class Child6 : public Base {
        using Base::Base;
};

#endif
```

Файл main.cpp

```
#include "Base.h"
#include "Application.h"
#include "Child2.h"
#include <iostream>
#include <string>
using namespace std;

int main()
```

```
{
        Application app(nullptr, "");
        app.BuildTree();
        return app.StartApp();
}
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root / ob1 2 / ob2 3 /ob1/ob2/ob4 ob3 4 endtree FIND . END	Object tree root ob1 ob2 The head object /ob1/ob2/ob4 is not found	Object tree root ob1 ob2 The head object /ob1/ob2/ob4 is not found
root / object_1 3 / object_2 2 /object_2 object_3 3 /object_2/object_3 obj2 2 / object_3 3 /object_2/object_3 obj 2 /object_2/object_3/obj search 5 endtree SET /object_2/object_3/obj FIND //search END	Object tree root object_1 object_2 object_3 obj2 obj search object_3 Object is set: obj //search Object name: search	Object tree root object_1 object_2 object_3 obj2 obj search object_3 Object is set: obj //search Object name: search
root / object_1 3 / object_2 2 /object_2 object_3 3 / object_3 3 /object_2/object_3 obj 2 /object_2/object_3/obj search 5 endtree FIND . SET /object_4/object_3 FIND object_1 END	Object tree root object_1 object_2 object_3 obj search object_3 . Object name: root Object is not found: root /object_4/object_3 object_1 Object name: object_1	Object tree root object_1 object_2 object_3 obj search object_3 . Object name: root Object is not found: root /object_4/object_3 object_1 Object name: object_1
root / object_1 3 / object_2 2 /object_2 object_3 3 /object_2/object_3 obj2 2 / object_3 3 /object_2/object_3 obj 2 /object_2/object_3/obj search 5 endtree SET	Object tree root object_1 object_2 object_3 obj2 obj search object_3 Object is set: obj search Object name: search	Object tree root object_1 object_2 object_3 obj2 obj search object_3 Object is set: obj search Object name: search

/object_2/object_3/obj FIND search END		

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (ИСТОЧНИКОВ)

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

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

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

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

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

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