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-27-21	Шевелёв И.А.
Руководитель практики	Старший преподаватель	Казанцева Л.В.
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

Объекты ввода/вывода библиотеки <iostream>

Класс string библиотеки <string>

Класс Tree:

- Модификатор доступа private

Указатель на объект current - хранит текущий объект в дереве

- Модификатор доступа public

Метод get_current - получает текущий объект в дереве

Метод set_current - задаёт текущий объект в дереве

Метод findPath - ищет символы "/" "." "//" и возращает указатель на объект

Класс Application (наследует Tree):

- Модификатор доступа public

Метод typeCommand - ввод команд и вывод результата

Описание алгоритма

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

Класс объекта: Tree

Модификатор доступа: public

Метод: findPath

Функционал: Находит путь

Параметры: string path, путь объекта

Возвращаемое значение: Tree*, указатель на объект

Алгоритм метода представлен в таблице 1.

Таблица 1. Алгоритм метода findPath класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1	Введённый символ равен "//"	Вернуть объект по имени	∅	
	Введённый символ равен "."	Вернуть текущий объект в дереве	∅	
	Введённый символ равен "/"	Вернуть текущий объект	∅	
		Найдя символ "/" записываем его позицию в строке в отдельную переменую	2	
2	Первый символ	Берём имя из строки и	∅	

	в строке равен "/"	возращаем объект по имени		
		Вызываем метод поиска имени по пути(findPath) у текущего объекта в дереве	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: get_current

Функционал: Возращает текущий объект в дереве

Параметры: нет

Возвращаемое значение: Tree*, указатель на объект

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода get_current класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Вернуть текущий объект в дереве	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: set_current

Функционал: Задаёт текущий объект в дереве

Параметры: Tree* current, указатель на текущий объект в дереве

Возвращаемое значение: void, возрат без значения

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода set_current класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Присвоить current значение адреса текущего объекта в дереве	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: build

Функционал: Создаёт дерево объектов

Параметры: нет

Возвращаемое значение: bool, возращает успех или неуспех создание дерева

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода build класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление переменных строкового типа данных parentName, childName	2	
2		Объявление целочисленной переменной object_number	3	
3		Ввод имени для корневого объекта	4	
4		Ввод имени объекта, которому будем создавать потомков	5	Бесконечный цикл

5	Ввод имени объекта равен строке "endtree"	Вывести "Object tree", вывести иерархию построенного дерева, вернуть функцию с значение true	∅	
		Ввод имени потомка и принадлежность класса	6	
6		Запоминаем объект в указатель prt с помощью метода findPath	7	
7	Объект не найден	Вывести "Object tree"	8	
			10	
8		Вывести иерархию построенного дерева	9	
9		Вернуть ложь	∅	
10	Введённое значение принадлежности класса равна 2	Создаёем объект класса class2 передавая в параметры введённое имя потомка и найденый объект его родителя	4	
	Введённое значение принадлежности класса равна 3	Создаёем объект класса class3 передавая в параметры введённое имя потомка и найденый объект его родителя	4	
	Введённое значение принадлежности класса равна 4	Создаёем объект класса class4 передавая в параметры введённое имя потомка и найденый объект его родителя	4	
	Введённое значение принадлежности класса равна 5	Создаёем объект класса class5 передавая в параметры введённое имя потомка и	4	

		найденый объект его родителя		
	Введённое значение принадлежности класса равна 6	Создаёем объект класса class6 передавая в параметры введённое имя потомка и найденый объект его родителя	4	

Класс объекта: Application

Модификатор доступа: public

Метод: typeCommand

Функционал: Ввод команд и вывод их результата

Параметры: нет

Возвращаемое значение: void, возрат без значения

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода typeCommand класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление переменной строкового типа данных command	2	
2		Ввод значение в command одной из команд (SET, FIND, END)	3	Бесконечный цикл
3	Значение command равняется "END"	Возрат без значения	∅	
		Ввод пути ('//', '/', '.')	4	
4	Значение command		5	

	равняется "SET"			
			6	
5	Находит объект по переданому пути	Переопределяем текущий объект в дереве, выводим "Object is set:" и имя объекта	6	
		Вывод "Object is not found:" и имени	6	
6	Значение command равняется "FIND"		7	
			2	
7	Объект по пути найден	Вывод "Object name:" и имени найденого объекта	2	
		Вывод " Object is not found" и имя	2	

Функция: main

Функционал: Главная функция программы

Параметры: нет

Возвращаемое значение: int, код возрата (0)

Алгоритм функции представлен в таблице 6.

Таблица 6. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Создание объекта obj класса Application с параметром nullptr	2	
2	Дерево не построилось		∅	

		Вызов метода launch объекта obj	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

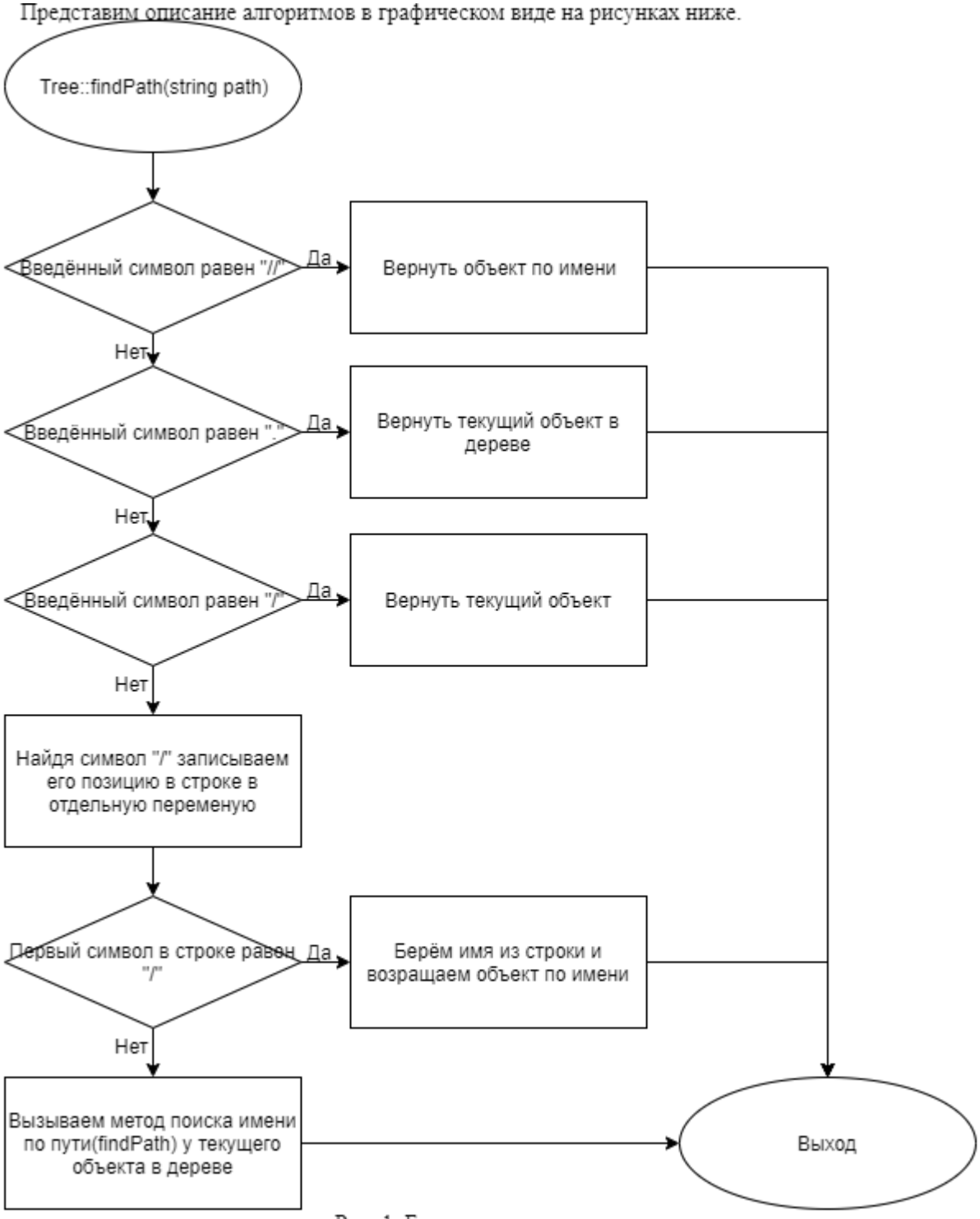

Рис. 1. Блок-схема алгоритма.

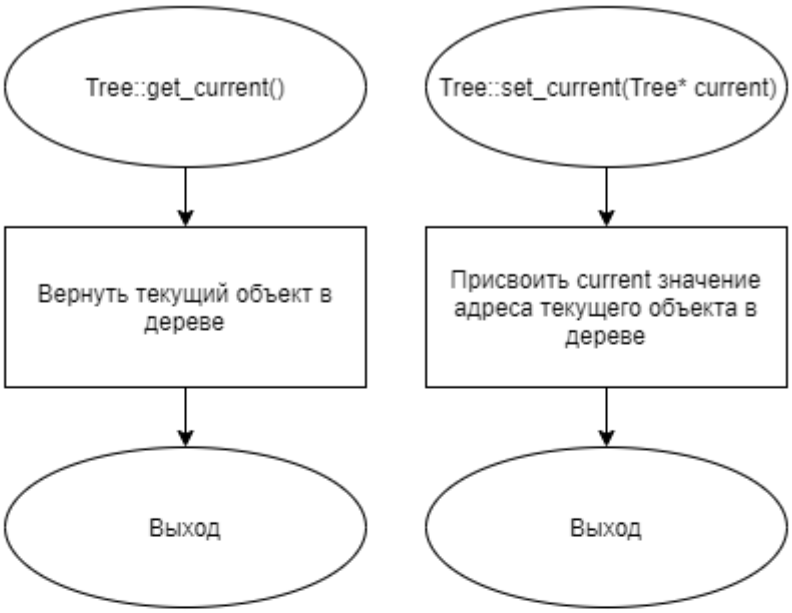

Рис. 2. Блок-схема алгоритма.

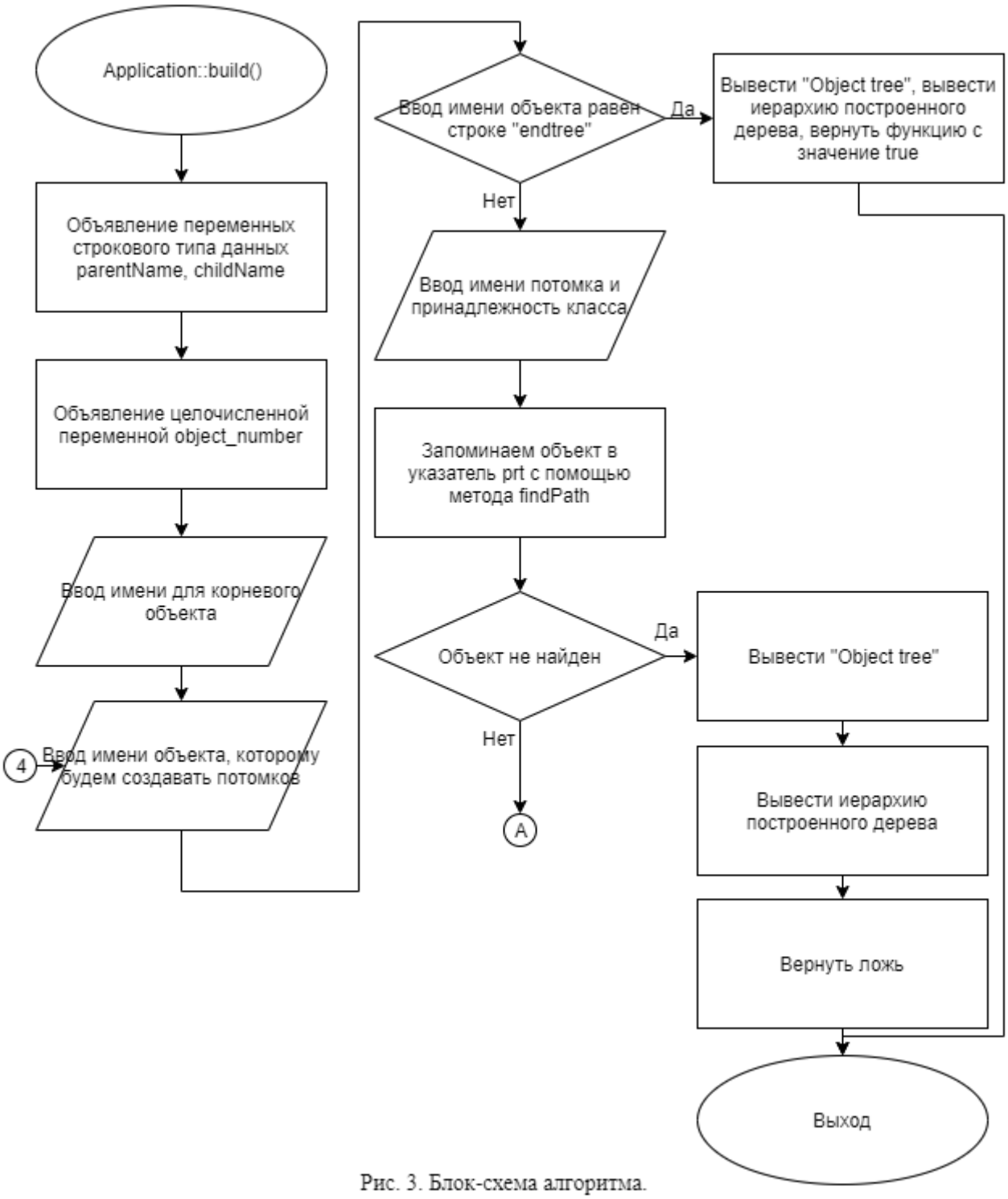

Рис. 3. Блок-схема алгоритма.

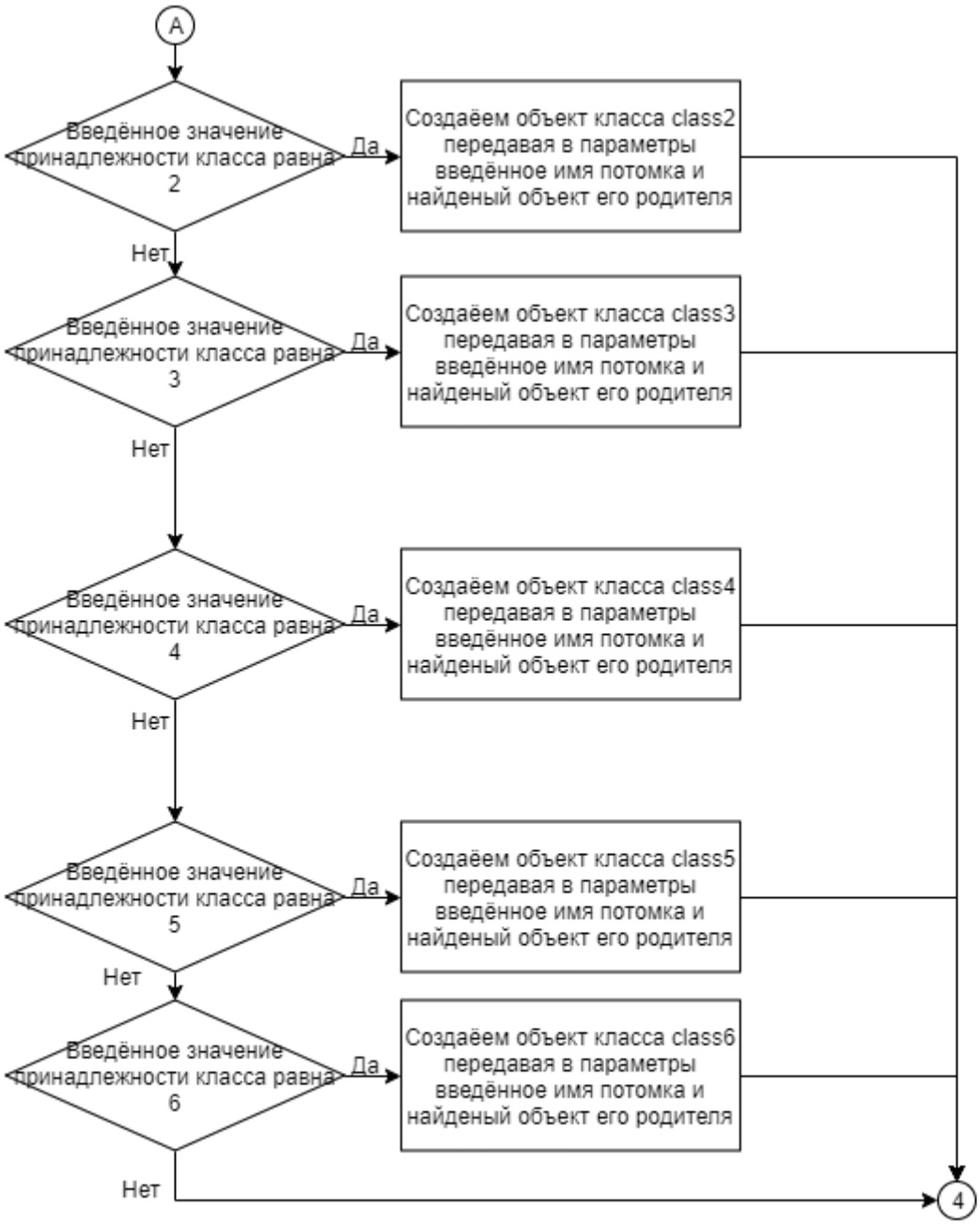

Рис. 4. Блок-схема алгоритма.

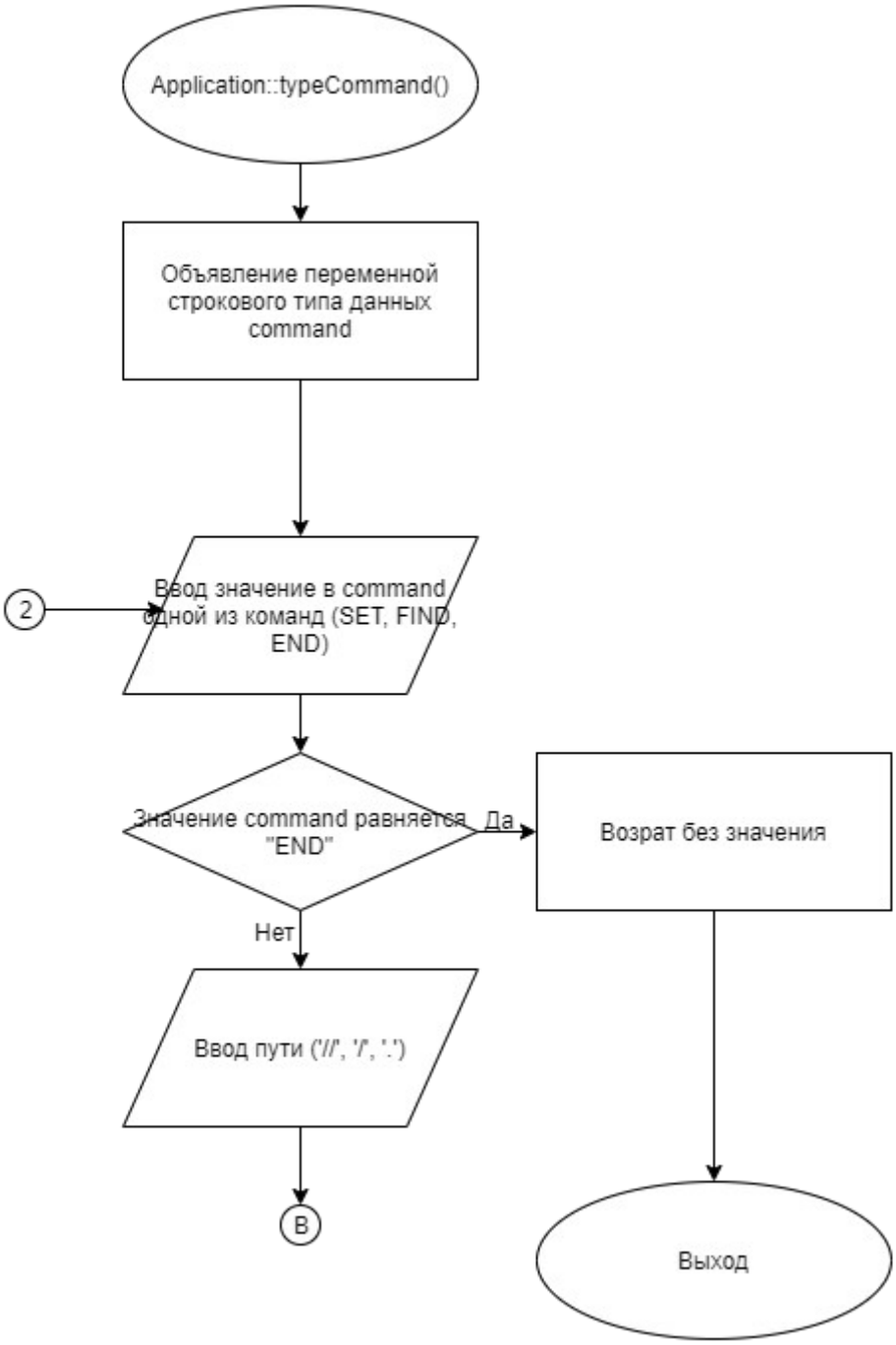

Рис. 5. Блок-схема алгоритма.

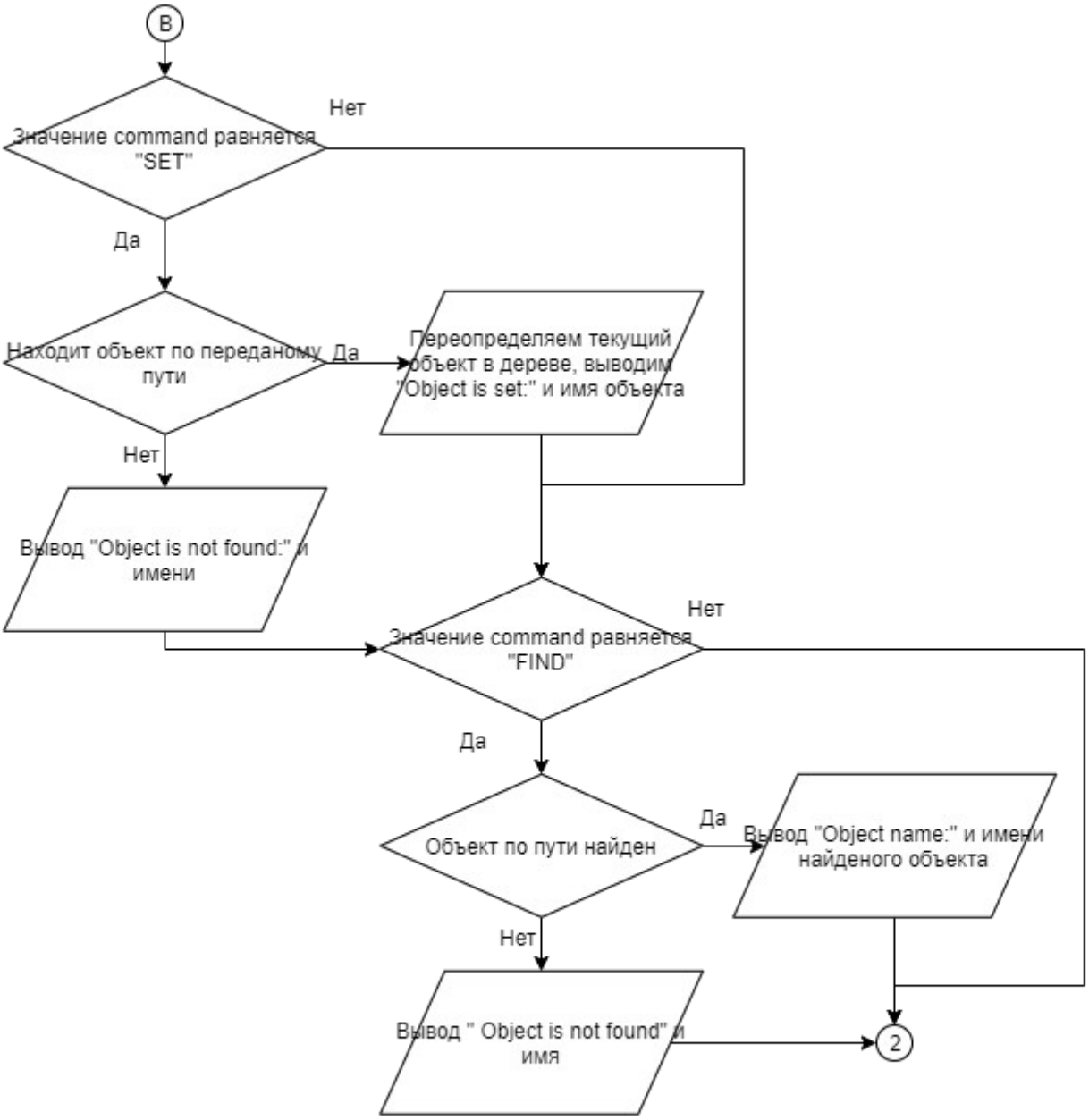

Рис. 6. Блок-схема алгоритма.

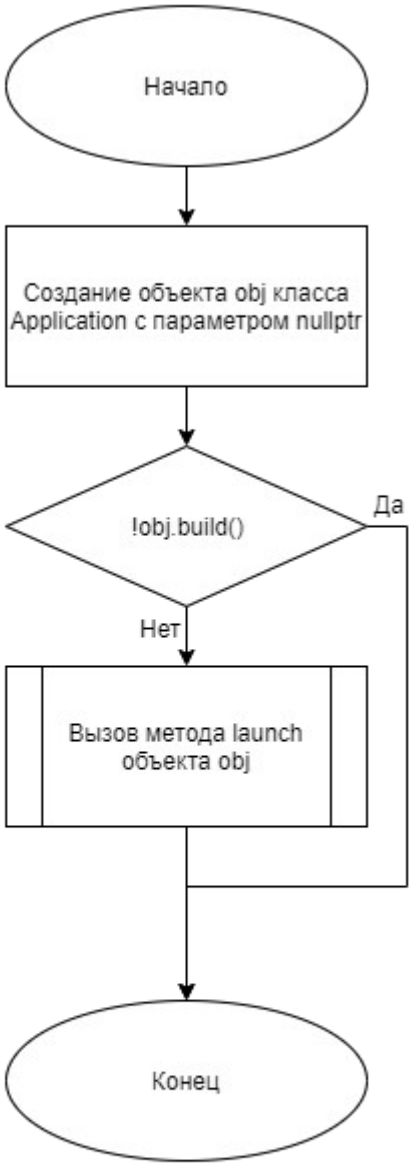

Рис. 7. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Application.cpp

```
#include "Application.h"

Application::Application(Tree* head): Tree(head){}

bool Application::build(){ // Изменено
	string parentName, childName;
	int object_number;
	cin >> name;
	while(true){
		cin >> parentName;
		if(parentName == "endtree"){
			//build_ready();
			cout << "Object tree" << endl;
			print();
			return true;
		}
		cin >> childName >> object_number;

		Tree* ptr = findPath(parentName);
		if(!ptr){ // Если не найдёт текущий объект в дереве
			cout << "Object tree" << endl;
			print();
			cout << "\nThe head object " << parentName << " is not
found";
			return false;
		}

		if(object_number == 2)
			Class2* child = new Class2(ptr, childName);
		else if(object_number == 3)
			Class3* child = new Class3(ptr, childName);
		else if(object_number == 4)
			Class4* child = new Class4(ptr, childName);
		else if(object_number == 5)
			Class5* child = new Class5(ptr, childName);
		else if(object_number == 6)
			Class6* child = new Class6(ptr, childName);
	}
}
void Application::typeCommand(){ // Ввод команд и вывод результата
	string command, path;
	while(true){
		cin >> command;
		if(command == "END"){
			return;
		}
		cin >> path;
		if(command == "SET"){
```

```
                        if(findPath(path)){
                                set_current(findPath(path));
                                cout << "\nObject is set: " << get_current()-
>get_name();
                        } else {
                                cout << "Object is not found: " <<
get_current()->get_name() << ' ' << path;
                        }
                }
                if(command == "FIND"){
                        cout << endl << path;
                        if(findPath(path)){
                                cout << "     Object name: " <<
findPath(path)->get_name();
                        } else {
                                cout << "     Object is not found";
                        }
                }

        }
}
int Application::launch(){
        //cout << "Object tree\n";
        //print();
        typeCommand();
        //cout << "\nThe tree of objects and their readiness\n";
        //print_ready("");
        return 0;
}
```

Файл Application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H
#include "Tree.h"
#include "Class2.h"
#include "Class3.h"
#include "Class4.h"
#include "Class5.h"
#include "Class6.h"

class Application : public Tree{
public:
        Application(Tree* head);
        bool build();
        int launch();
        void typeCommand(); // Ввод команд и вывод результата
};
#endif
```

Файл Class2.h

```
#ifndef CLASS2_H
#define CLASS2_H
#include "Tree.h"

class Class2 : public Tree{
public:
        Class2(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл Class3.h

```
#ifndef CLASS3_H
#define CLASS3_H
#include "Tree.h"

class Class3 : public Tree{
public:
        Class3(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл Class4.h

```
#ifndef CLASS4_H
#define CLASS4_H
#include "Tree.h"

class Class4 : public Tree{
public:
        Class4(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл Class5.h

```
#ifndef CLASS5_H
#define CLASS5_H
#include "Tree.h"

class Class5 : public Tree{
public:
```

```
        Class5(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл Class6.h

```
#ifndef CLASS6_H
#define CLASS6_H
#include "Tree.h"

class Class6 : public Tree{
public:
        Class6(Tree* head, string name) : Tree(head, name){}
};
#endif
```

Файл main.cpp

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

int main()
{
                // program here
                Application obj(nullptr);
                if(!obj.build()){
                        return 0;
                }
                return obj.launch();

}
```

Файл Tree.cpp

```
#include "Tree.h"

Tree::Tree(Tree* head, string name){
        current = this;
        this->name = name;
        this->head = head;
        if(head != nullptr){
                head->children.push_back(this);
        }
        ready = 0;
```

```
}

void Tree::set_name(string name){
        this->name = name;
}
string Tree::get_name(){
        return name;
}
void Tree::set_head(Tree* head){ // Изменено
        if(head != nullptr){
                for(int i = 0; i < head->children.size(); i++){
                        if(head == this){
                                head->children.erase(head->children.begin() +
i);
                        }
                }
        }
        this->head = head;
        if(head != nullptr){
                head->children.push_back(this);
        }
}
Tree* Tree::get_head(){
        return head;
}
void Tree::print(int count){ // Изменено
        if(head == nullptr)
                cout << this->name;
        if(children.empty())
                return;
        for(Tree* element : children){
                cout << endl;
                for(int i = 0; i < count; i++){
                        cout << "    ";
                }
                cout << element->get_name();
                element->print(++count);
                --count;
        }
}
Tree* Tree::getByName(string name){
        if(this->name == name){
                return this;
        }
        Tree* result = nullptr;
        for(int i = 0; i < children.size(); i++){
                result = children[i]->getByName(name);
                if(result != nullptr){
                        return result;
                }
        }
        while(result != nullptr){
                result = head;
                getByName(name);
        }
        return nullptr;
}

// КЛ_3_1
void Tree::set_ready(int input_ready){
```

```
        if(input_ready == 0){
                ready = 0;
                for(int i = 0; i < children.size(); i++){
                        children[i]->set_ready(0);
                }
        }
        else{
                Tree* temp = this->head;
                while(temp != nullptr){
                        if(temp->ready == 0){
                                ready = 0;
                                return;
                        }
                        temp = temp->head;
                }
                this->ready = input_ready;
        }
}
void Tree::build_ready(){
        string name;
        int input_ready;
        while(cin >> name >> input_ready){
                if(getByName(name) != nullptr){
                        (getByName(name))->set_ready(input_ready);
                }
        }
}
void Tree::print_ready(string tab) {
        if(ready > 0 || ready < 0)
                cout << get_name() << " is ready";
        else
                cout << get_name() << " is not ready";
        if(children.size() != 0){
                tab += "    ";
                for(Tree* element : children){
                        cout << endl << tab;
                        element->print_ready(tab);
                }
        }
}

// КЛ_3_2
void Tree::set_current(Tree* current){ // Выбрать(назначает текущим) объект в
дереве
        this->current = current;
}
Tree* Tree::get_current(){ // Получить указатель на выбранный(текущий) в
дереве объект
        return current;
}
Tree* Tree::findPath(string path){ // Ищет символы '/' '.' '//' и возращает
указатель на объект
        if(path.substr(0, 2) == "//") {
                return getByName(path.substr(2));
        }
        else if(path == "."){
                return current;
```

```
        }
        else if(path == "/"){
                return this;
        }

        int posSlash = path.find('/', 1);
        if(path[0] == '/'){
                string nextName;
                if(posSlash == -1){
                        nextName = path.substr(1);
                        for(int i = 0; i < children.size(); i++){
                                if(nextName == children[i]->get_name()){
                                        return children[i];
                                }
                        }
                        return nullptr;
                }
                nextName = path.substr(1, posSlash - 1);
                for(int i = 0; i < children.size(); i++){
                        if(nextName == children[i]->get_name()){
                                return children[i]-
>findPath(path.substr(posSlash));
                        }
                }
                return nullptr;
        }
        return current->findPath("/" + path);
}
```

Файл Tree.h

```
#ifndef TREE_H
#define TREE_H
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Tree{
protected:
        string name;
        Tree* head;
        vector<Tree*> children;
        int ready; // Готовность объекта
        Tree* current; // Указатель на выбранный(текущий) в дереве объект
public:
        Tree(Tree* head, string name = "root");
        void set_name(string name);
        string get_name();
        void set_head(Tree* head);
        Tree* get_head();
        void print(int count = 1);

        // КЛ_3_1
        Tree* getByName(string name);
        void set_ready(int input_ready); // Задаёт готовность объекту
```

```
        void build_ready(); // Установка готовности объектам
        void print_ready(string tab); // Вывод готовности объектов

        // КЛ_3_2
        Tree* get_current(); // Получить указатель на выбранный(текущий) в
дереве объект
        void set_current(Tree* current); // Выбрать(назначает текущим) объект
в дереве
        Tree* findPath(string name); // Ищет символы '/' '.' '//' и возращает
указатель на объект
};
#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
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_1 FIND //object_15 FIND . FIND object_4/object_7 SET object_4/object_7 FIND / SET //object_4 FIND object_3 END	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_1 Object name: object_1 //object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7 Object is set: object_7 / Object name: root Object is set: object_4 object_3 Object is not found	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_1 Object name: object_1 //object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7 Object is set: object_7 / Object name: root Object is set: object_4 object_3 Object is not found
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	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	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

END		object_4/object_7 Object name: object_7
root / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_8 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	Object tree root object_1 object_2 object_4 object_5 object_3 The head object /object_8 is not found	Object tree root object_1 object_2 object_4 object_5 object_3 The head object /object_8 is not found

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).