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 дисциплины)

Тема курсовой работы **Разработка программы для проверки позиции объекта в дереве иерархии.**

Студент группы **ИКБО-29-20 Хан Анастасия Александровна** _____
(учебная группа, фамилия, имя отчество, студента) (подпись студента)

Руководитель курсовой работы **Доцент, к.ф-м.н. Путуридзе З.Ш.** _____
(должность, звание, ученая степень) (подпись руководителя

Рецензент (при наличии) _____ _____
(должность, звание, ученая степень) (подпись рецензента)

Работа представлена к защите « ____ » _____ 2021г.

Допущен к защите « ____ » _____ 2021г.

Москва 2021 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой _____
Подпись
Платонова О.В.
ФИО
«25» февраля 2021 г.

ЗАДАНИЕ

на выполнение курсовой работы по дисциплине

«Объектно-ориентированное программирование»

Студент Хан Анастасия Александровна Группа ИКБО-29-20
Тема: Разработка программы для проверки позиции объекта в дереве иерархии.

Исходные данные: Исходная иерархия расположения объектов, координаты искомых объектов.
Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Реализовать алгоритм размещения всех объектов в составе программы на иерархическом дереве объектов.
2. Реализовать алгоритм поиска объекта на дереве иерархии.
3. Блок-схема реализованных алгоритмов.

Срок представления к защите курсовой работы: **до «31» мая 2021г.**

Задание на курсовую работу выдал _____ (Путуридзе З.Ш.)
Подпись руководителя *Ф.И.О. руководителя*

Задание на курсовую работу получил «25» февраля 2021г

_____ (Хан А.А.)
Подпись обучающегося Ф.И.О. исполнителя

Москва 2021 г.

ОТЗЫВ
на курсовую работу
по дисциплине «Объектно-ориентированное программирование»

Студент(ка) Хан Анастасия Александровна Группа ИКБО-29-20

Характеристика курсовой работы

Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме			
2. Соответствие курсовой работы заданию			
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.			
4. Полнота выполнения всех пунктов задания			
5. Логичность и системность содержания курсовой работы			
6. Отсутствие фактических грубых ошибок			

Рекомендуемая оценка: удовлетворительно, хорошо, отлично

Путуридзе З.Ш.

Подпись руководителя *(ФИО руководителя)*

«___»_____2021 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

За годы развития компьютерных наук появилось множество подходов к написанию программного обеспечения. Набор принципов и правил, по которым формируется структура программы называется парадигмой. Парадигма программирования определяет, как компьютер организует вычисления при выполнении программы. Одной из самых популярных парадигм является объектно-ориентированное программирование. Согласно принципам этой парадигмы, программа представляет собой систему объектов различных типов, взаимодействующих между собой. Подобное восприятие упрощает процесс моделирования и реализации компьютерных систем, так как окружающий нас мир состоит из объектов, которые влияют друг на друга.

Парадигма ООП лежит в основе многих мейнстримовых языков. ОО-языки: C++, Python, Java, JavaScript, Swift и др. То есть топовые языки, ряд языков общего назначения, языки для Android-, веб- и IOS-разработки. Популярные современные языки - мультипарадигмальные, по принципам ООП можно писать и на языках с другими парадигмами. Это уже говорит об актуальности,востребованности знания ООП.

Объектно-ориентированное программирование всегда будет актуальным из-за своей выразительности. Средства данной парадигмы позволяют создавать большие и расширяемые решения сложных прикладных задач. ООП позволяет упростить восприятие функциональных взаимоотношений компонентов системы. Актуальность и востребованность ООП обусловлена тем, что позволяет абстрагироваться от деталей реализации и сосредоточиться на построении правильной архитектуры. Изящность ООП заменила собой запутанность процедурной парадигмы.

Если объединить все факторы, можно сказать, что ООП - это некий способ моделирования реального мира. Оно дает контроль над зависимостями в коде.Это способ сделать так, чтобы высокоуровневый код не зависел от низкоуровневой

реализации. Что дает возможность облегчить работу программистам.

ООП актуально сейчас и будет актуально еще очень долгое время. Подтверждением служит то, что одним из требований при приеме на работу программиста, является требование " понимание ООП".

Постановка задачи

Определение указателя на объект по его координате

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

В составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов.

В качестве параметра методу передать путь объекта от корневого. Путьзадать в следующем виде:

/root/ob_1/ob_2/ob_3

Уникальность наименования требуется только относительно множества подчиненных объектов для любого головного объекта.

Если система содержит объекты с уникальными именами, то в методе реализовать определение указателя на объект посредством задания координаты в виде:

//«наименование объекта»

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в контрольной работе № 1.

Единственное различие: в строке ввода первым указать не наименование головного объекта, а путь к головному объекту.

Подразумевается, что к моменту ввода очередной строки соответствующая ветка на дереве иерархии уже построена.

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2, 3, 4, 5, 6.

Пример ввода иерархии дерева объектов.

```
root
/root object_1 3 1
```

/root object_2 2 1

/root/object_2 object_4 3 -1

/root/object_2 object_5 4 1

/root object_3 3 1

/root/object_2 object_3 6 1

/root/object_1 object_7 5 1

/root/object_2/object_4 object_7 3 -1

endtree

Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии.

Структура данных для ввода согласно изложенному в фрагменте методического указания [3] в контрольной работе № 1.

После ввода состава дерева иерархии построчно вводятся координаты искомых объектов.

Ввод завершается при вводе: //

Описание выходных данных

Первая строка:

Object tree

Со второй строки вывести иерархию построенного дерева.

Далее, построчно:

«координата объекта» Object name: «наименование объекта»

Разделитель один пробел.

Если объект не найден, то вывести:

«координата объекта» Object not found

Разделитель один пробел.

Метод решения

Иерархия наследования отображена в таблице 1.

Таблица 1. «Описание иерархии наследования классов»

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	Base			Базовый класс в иерархии классов. Содержит основные поля и методы.	
		Application	public		2
		Derived	public		3
		Derived1	public		4
		Derived2	public		5
		Derived3	public		6
		Derived4	public		7
2	Application			Класс корневого объекта(приложения).	
3	Derived			Классы объектов, подчиненных корневому объекту Application.	
4	Derived1				
5	Derived2				
6	Derived3				
7	Derived4				

Детальное описание свойств и функционала классов:

Класс Base:

- **Свойства/поля:**
    - Поле, отвечающее за наименование объекта
        - Наименование - name;
        - Тип - строковое;
        - Модификатор доступа - private;
    - Поле, отвечающее за указатель на головной объект для текущего объекта
        - Наименование - parent;
        - Тип - указатель на объект класса Base(или его наследников);
        - Модификатор доступа - private;
    - Поле, отвечающее за список указателей на объекты, подчиненных к текущему объекту в дереве иерархии
        - Наименование - children;
        - Тип - контейнер Vector из указателей на объекты класса Base(или его наследников);
        - Модификатор доступа - private;
    - Поле, отвечающее за итератор списка указателей на объекты, подчиненных к текущему объекту
        - Наименование - children_iterator;
        - Тип - итератор контейнера Vector из указателей на объекты класса Base(или его наследников);
        - Модификатор доступа - private;

- Поле, отвечающее за статус объекта
    - Наименование - state;
    - Тип - целочисленное;
    - Модификатор доступа - private;

- **Методы:**
    - Конструктор Base:
        - Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии и наименованием объекта;
    - Метод SetName:
        - Функционал - используется для определения имени объекта;
    - Метод GetName:
        - Функционал - используется для получения имени объекта;
    - Метод SetParent:
        - Функционал - используется для определения головного объекта для текущего объекта;
    - Метод GetParent:
        - Функционал - используется для получения головного объекта текущего объекта;
    - Метод PrintName:
        - Функционал - используется для вывода наименований объектов в дереве иерархии слева-направо и сверху-вниз;
    - Метод find:

        - Функционал - поиск заданного объекта в дереве иерархии;
    - Метод SetState:
        - Функционал - используется для определения статуса текущего объекта;
    - Метод GetState:
        - Функционал - используется для получения статуса текущего объекта;
    - Метод PrintState:
        - Функционал - используется для вывода статуса объектов;
    - Метод FindPath:
        - Функционал - поиск объекта с помощью пути корневого объекта
    - Метод SearchObject:
        - Функционал - поиск объекта по его пути и вывод на экран пути от корневого объекта и имя объекта
    - Метод PrintPath:
        - Функционал - вывод пути до вызывающего объекта

Класс Application:

- **Свойства/поля:**
    - Поле, отвечающее за указатель на головной объект(используется для хранения головного объекта при построении дерева объектов)
        - Наименование - p_parent;
        - Тип - указатель на объект класса Base (или его наследников);
        - Модификатор доступа - private;

- Поле, отвечающее за указатель на подчиненный объект (используется для хранения подчиненного объекта при построении дерева объектов)
    - Наименование - c_children;
    - Тип - указатель на объект класса Base (или его наследников);
    - Модификатор доступа - private;

- **Методы:**
    - Конструктор Application:
        - Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии, наименование объекта и статус объекта;
    - Метод BuildTree:
        - Функционал - используется для построения дерева иерархии;
    - Метод ExecApp:
        - Функционал - используется для запуска основного алгоритма приложения;

Класс Derived:

- Собственные поля отсутствуют
- **Методы:**
    - Конструктор Derived
        - Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии, наименование объекта и статус объекта;

Класс Derived1:

- Собственные поля отсутствуют
- **Методы:**
    - Конструктор Derived1
        - Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии, наименование объекта и статус объекта;

Класс Derived2:

- Собственные поля отсутствуют
- **Методы:**
    - Конструктор Derived2
        - Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии, наименование объекта и статус объекта;

Класс Derived3:

- Собственные поля отсутствуют
- **Методы:**
    - Конструктор Derived3
        - Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии, наименование объекта и статус объекта;

Класс Derived4:

- Состенные поля отсутствуют
- **Методы:**

- Конструктор Derived4
    - Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии, наименование объекта и статус объекта;

Для решения поставленной задачи воспользуемся объектами стандартных потоков ввода и вывода - cin/cout из библиотеки <iostream>, условным оператором if..else, операторами цикла - while/for, объектами класса vector из библиотеки <vector>, объектами класса string из библиотеки <string>.

Описание алгоритма

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

Класс объекта: Base

Модификатор доступа: public

Метод: FindPath

Функционал: Поиск объекта с помощью пути от корневого объекта

Параметры: Строковое path - путь до объекта

Возвращаемое значение: Указатель на искомый объект

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода FindPath класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Создание с инициализацией указателя типа Base parent= нулевому указателю	2	
2	Путь к объекту начинается с двух слэшей	Удаляем два первых символа значения переменной path	3	
			4	
3		Возврат результата вызова метода find с параметром path	∅	
4		Объявление целочисленной переменной с инициализацие й cnt=2	5	

Продолжение таблицы 2.

5		Объявление строковой переменной name	6	
6	В переменной path не осталось символов	Возврат текущего указателя на объект	∅	
		Инициализация переменной name присваиваем ей значение второго символа переменной path	7	
7	Считывается имя текущего объекта и мы не дошли до конца пути	Добавляем в переменную name текущий символ	8	
			9	
8		Инкрементируем значение переменной cnt	7	
9		Стираем из переменной path имя считанного объекта	10	
10	Имя текущего объекта совпадает с искомым	Присваиваем указателю ParentName указатель на текущий объект	11	
			11	
11		Запись в поле children_iterator текущего объекта итератора первого подчиненного объекта	12	
12	Обработаны НЕ все подчиненные объекты текущего объекта		13	
			14	

13	Имя текущего подчиненного объекта совпадает с искомым	Присваиваем указателю ParentName указатель на текущий подчиненный объект	15	
		Инкремент итератора children_iterator	12	
14	Не было найдено подчиненного объекта с искомым именем	Возврат нуля	∅	
			15	
15		Присваиваем указателю ParentName результат вызова метода FindPath с параметром path	16	
16		Возврат значения указателя ParentName	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: SearchObject

Функционал: Поиск объекта по его пути и вывод на экран пути от корневого объекта и имя объекта

Параметры: -

Возвращаемое значение: void

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода SearchObject класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление строковой переменной path	2	
2		Ввод значения переменной path	3	

Продолжение таблицы 3.

3	Введеный путь не "//"	Объявление указателя temp тип Base	4	
			∅	
4		Присваиваем указателю temp результат вызова метода FindPath с параметром path	5	
5	Объект был найден в иерархии	Вывод перехода на новую строку	6	
			8	
6		Вызов метода PrintPath объекта temp	7	
7		Вывод "Object name: "+результат вызова метода GetName объекта temp	9	
8		Вывод перехода на новую строку+значения переменной path+" Object not found"	9	
9		Ввод значения переменной path	3	

Класс объекта: Base

Модификатор доступа: public

Метод: PrintPath

Функционал: Вывод пути до вызывающего объекта

Параметры: -

Возвращаемое значение: void

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода PrintPath класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление указателя тип Base с инициализацией parentPtr= результат вызова метода GetParent текущего объекта	2	
2	У текущего объекта есть родительский объект	Вызов метода PrintPath объекта parentPtr	3	
			3	
3		Вывод "/"+результат вызова метода GetName текущего объекта	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: BuildTree

Функционал: Построение дерева иерархии объектов

Параметры: -

Возвращаемое значение: void

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода BuildTree класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление переменных строкового типа parent_name и child_name	2	
2		Объявление переменных	3	

Продолжение таблицы 5.

		целочисленного типа ClassNum и state		
3		Ввод значения переменной parent_name	4	
4		Вызов метода SetName текущего объекта с параметром parent_name	5	
5		Запись в поле p_parent корневого объекта указателя на текущий(корневой) объект	6	
6		Запись в поле p_app корневого объекта указателя на текущий(корневой) объект	7	
7		Ввод имени нового объекта	8	
8	Имя нового объекта endtree		∅	
			9	
9	Введеное имя родительского объекта не совпадает с прошлым именем родительского объекта	Присваивание указателю p_parent результат вызова метода FindPath с параметром parent_name	10	
			10	
10		Ввод имени текущего подчиненного объекта, помера класса текущего подчиненного объекта и состояние текущего подчиненного объекта	11	

11	Создаваемый объект принадлежит первому классу	Создание объекта класса Derived с помощью конструктора, в качестве параметров - поле p_parent корневого объекта, переменной child_name и переменную state	7	
	Создаваемый объект принадлежит второму классу	Создание объекта класса Derived1 с помощью конструктора, в качестве параметров - поле p_parent корневого объекта, переменной child_name и переменную state	7	
	Создаваемый объект принадлежит третьему классу	Создание объекта класса Derived2 с помощью конструктора, в качестве параметров - поле p_parent корневого объекта, переменной child_name и переменную state	7	
	Создаваемый объект принадлежит четвертому классу	Создание объекта класса Derived3 с помощью конструктора, в качестве параметров - поле p_parent корневого объекта, переменной child_name и переменную state	7	
	Создаваемый объект принадлежит пятому классу	Создание объекта класса Derived4 с помощью конструктора, в качестве параметров - поле p_parent корневого объекта, переменной child_name и переменную state	7	
			7	

Класс объекта: Application

Модификатор доступа: public

Метод: ExecApp

Функционал: Запуск основного алгоритма приложения

Параметры: -

Возвращаемое значение: Целое число - индикатор корректности завершения программы,0

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода ExecApp класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран "Object tree"	2	
2		Вызов метода PrintState() текущего объекта с целочисленным параметром 4	3	
3		Вызов метода SearchObject	4	
4		Возврат нуля	∅	

Функция: main

Функционал: Основной алгоритм программы

Параметры: -

Возвращаемое значение: Целое число - индикатор корректности завершения программы, 0

Алгоритм функции представлен в таблице 7.

Таблица 7. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Создание объекта app класса Application с помощью конструктора, аргументы не передаются	2	
2		Вызов метода BuildTree объекта app.	3	
3		Вызов метода ExecApp объекта app.	4	
4		Возврат результата предыдущего вызова метода ExecApp объекта app.	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: Base

Функционал: Параметризированный конструктор, устанавливает головной объект и наименование для текущего объекта

Параметры: parent - указатель на головной объект класса Base(или его наследников), по умолчанию - нулевой; строка name - наименованиетекущего объекта, по умолчанию - "base", state - целое число, состояние объекта по умолчанию 1

Возвращаемое значение: Ничего не возвращает

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода Base класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода SetParent текущего объекта с параметром	2	

		parent		
2		Вызов метода SetName текущего объекта с параметром name	3	
3		Вызов метода SetName текущего объекта с параметром name	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: SetName

Функционал: Определение имени текущего объекта

Параметры: Строковое name - новое наименование текущего объекта

Возвращаемое значение: void

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода SetName класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Инициализация поля name текущего объекта значением параметра name	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: GetName

Функционал: Возвращает наименование текущего объекта

Параметры: -

Возвращаемое значение: Строка - наименование текущего объекта

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода GetName класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат значения поля name текущего объекта	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: SetParent

Функционал: Определение головного объекта для текущего объекта

Параметры: parent - указатель на головной объект класса Base(или его наследников)

Возвращаемое значение: void

Алгоритм метода представлен в таблице 11.

Таблица 11. Алгоритм метода SetParent класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Запись в поле parent текущего объекта значения параметра parent	2	
2	Указатель на головной объект не нулевой	Вызов метода push_back поля children головного объекта для текущего с параметром - указатель на текущий объект	∅	
			∅	

Класс объекта: Base

Модификатор доступа: public

Метод: GetParent

Функционал: Возвращение указателя на головной объект для текущегообъекта

Параметры: -

Возвращаемое значение: Указатель на головной объект класса Base (или его наследников) для текущего объекта

Алгоритм метода представлен в таблице 12.

Таблица 12. Алгоритм метода GetParent класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат значения поля parent текущего объекта	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

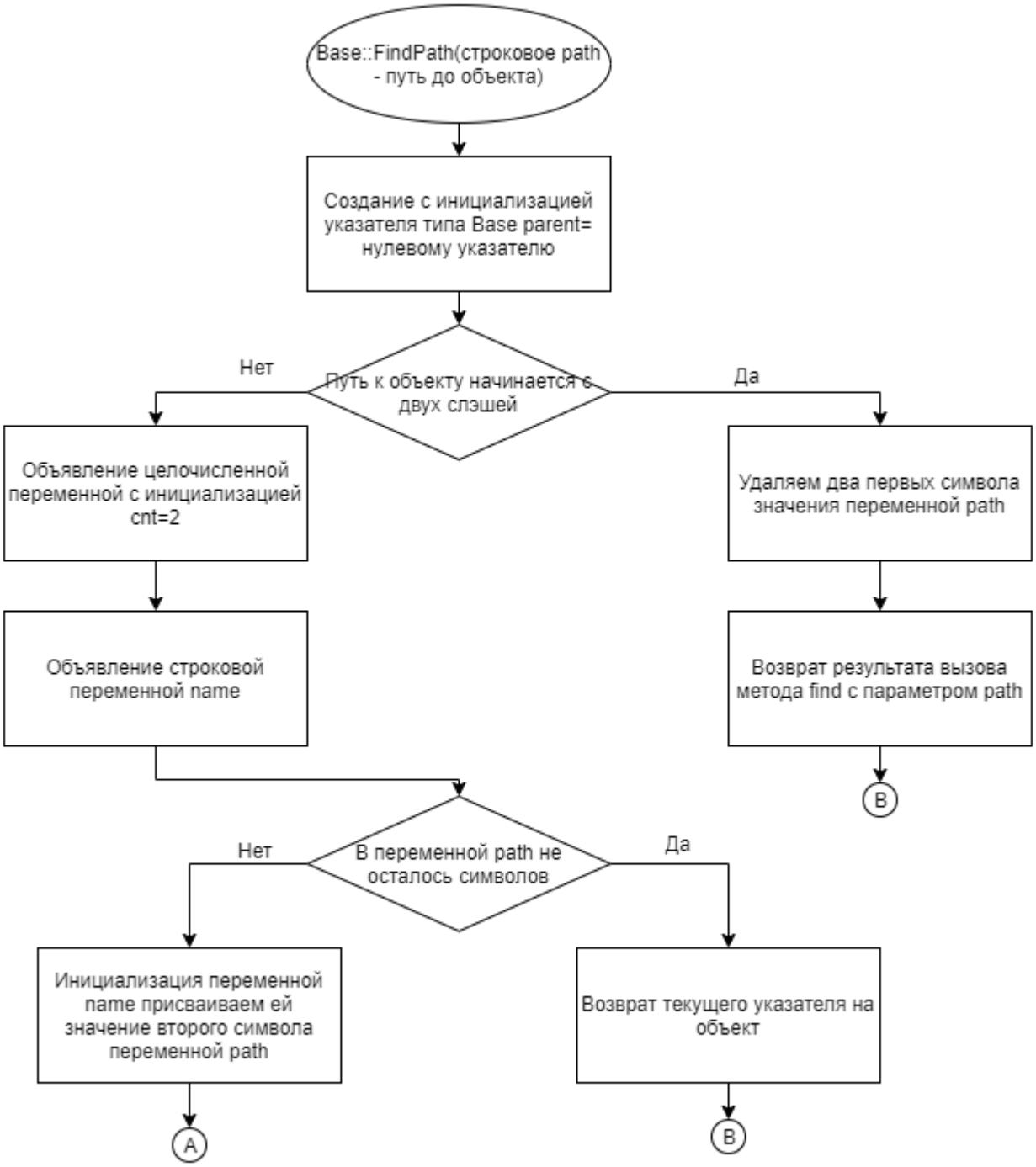

Рис. 1. Блок-схема алгоритма.

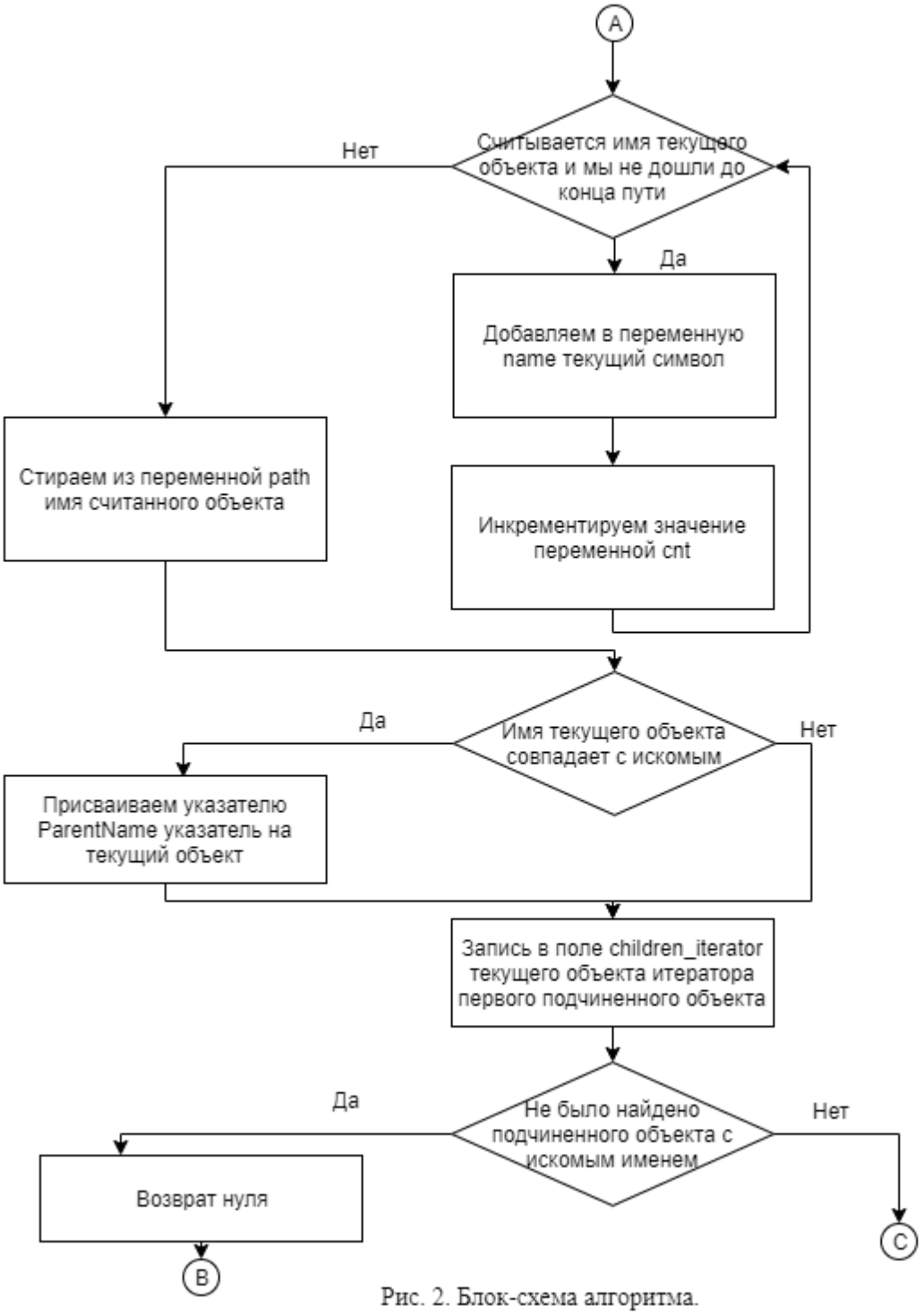

Рис. 2. Блок-схема алгоритма.

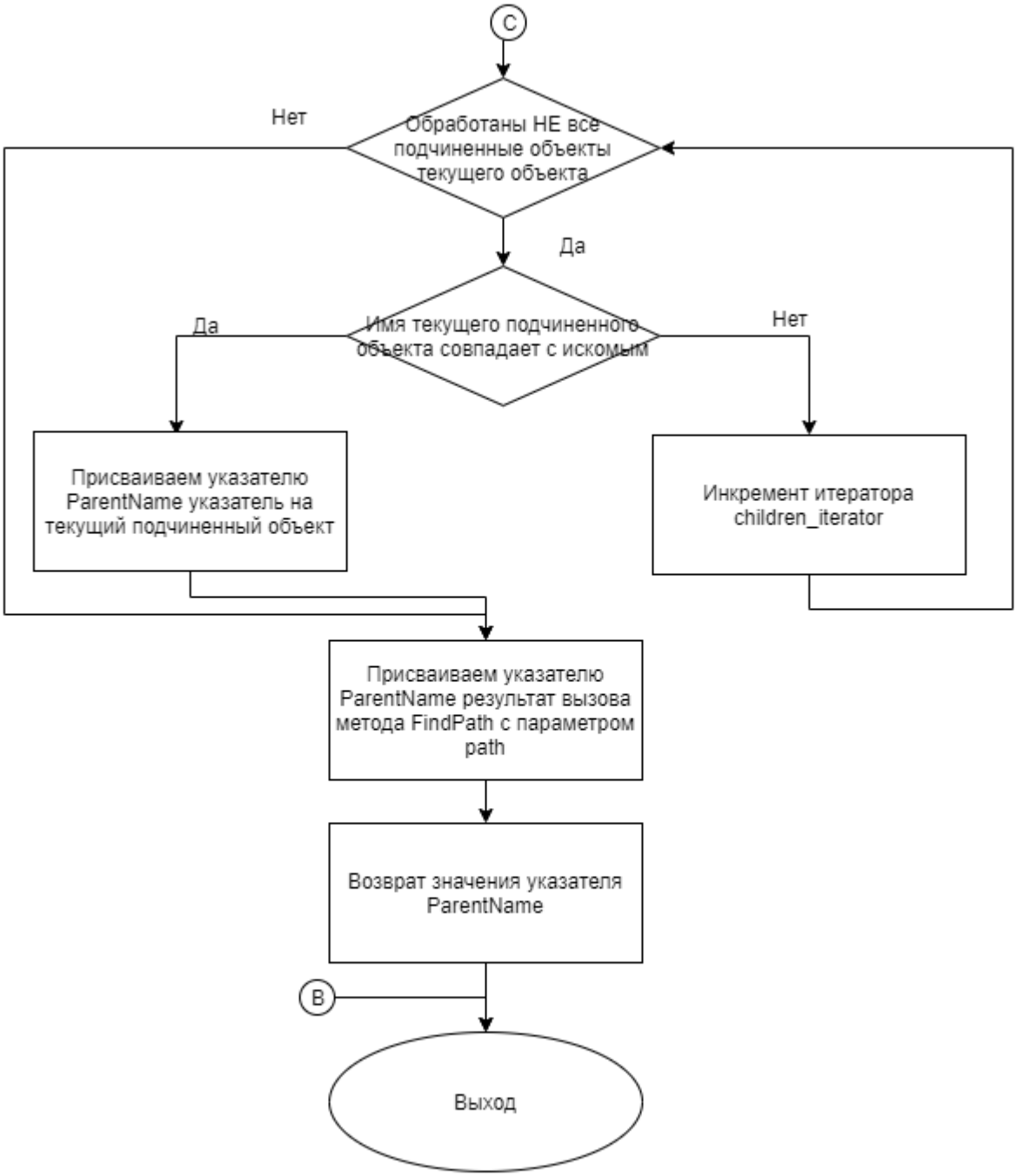

Рис. 3. Блок-схема алгоритма.

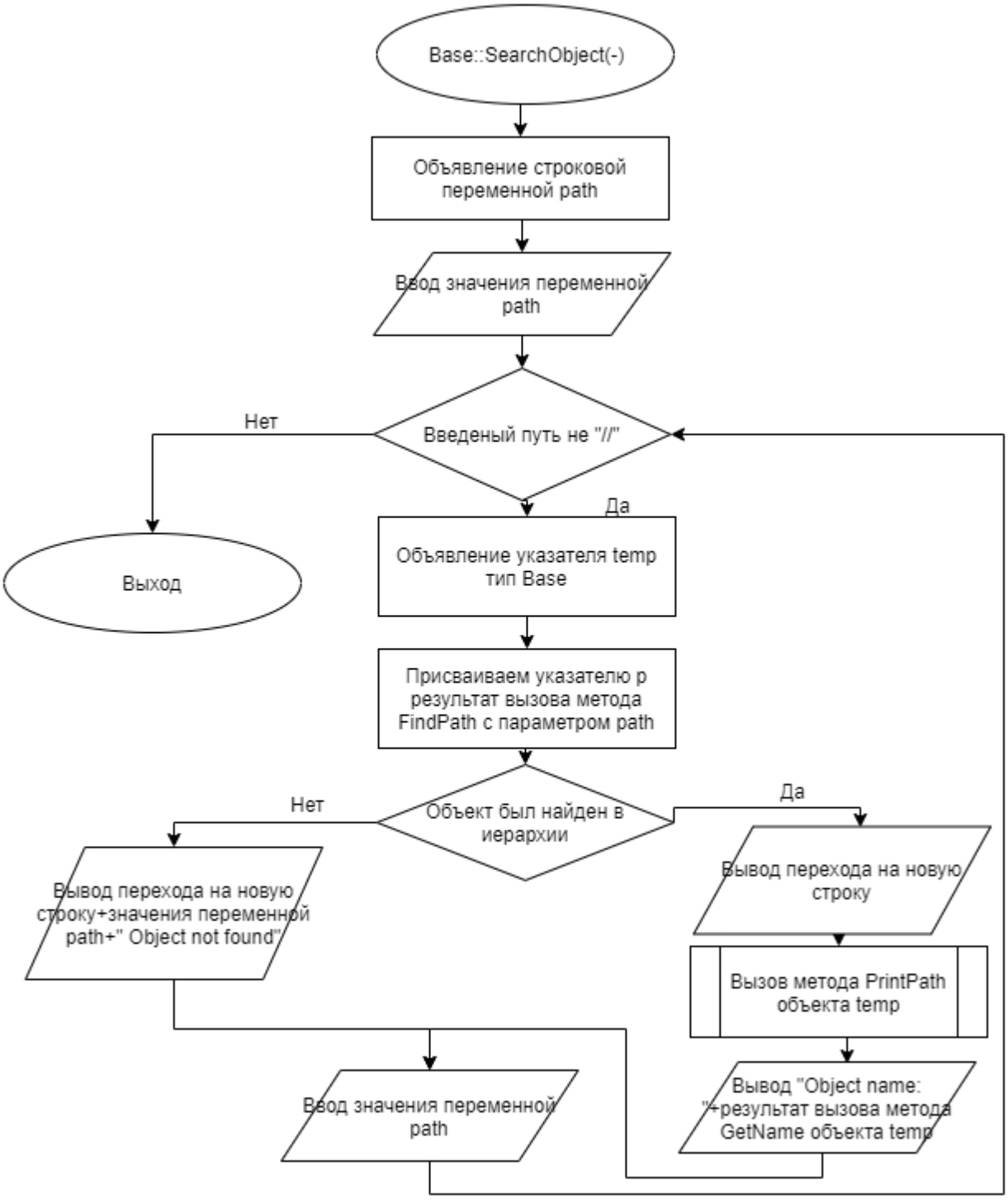

Рис. 4. Блок-схема алгоритма.

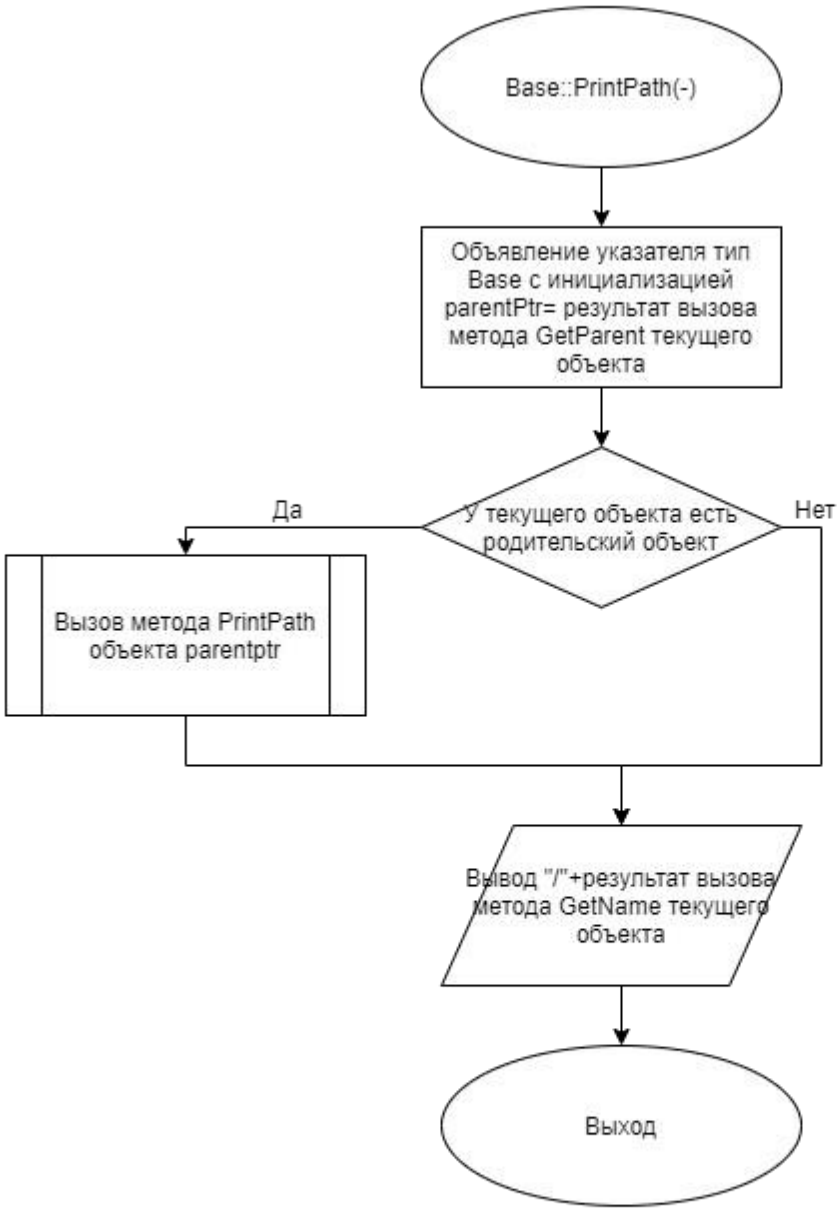

Рис. 5. Блок-схема алгоритма.

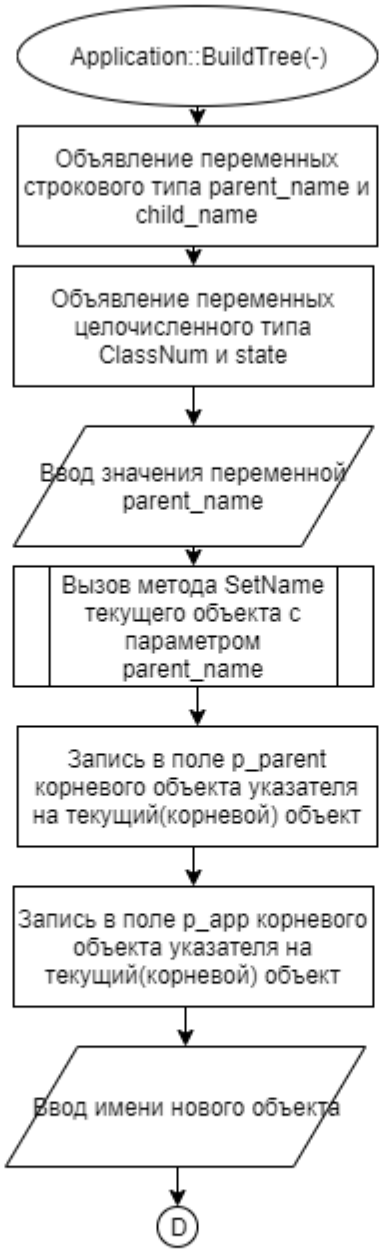

Рис. 6. Блок-схема алгоритма.

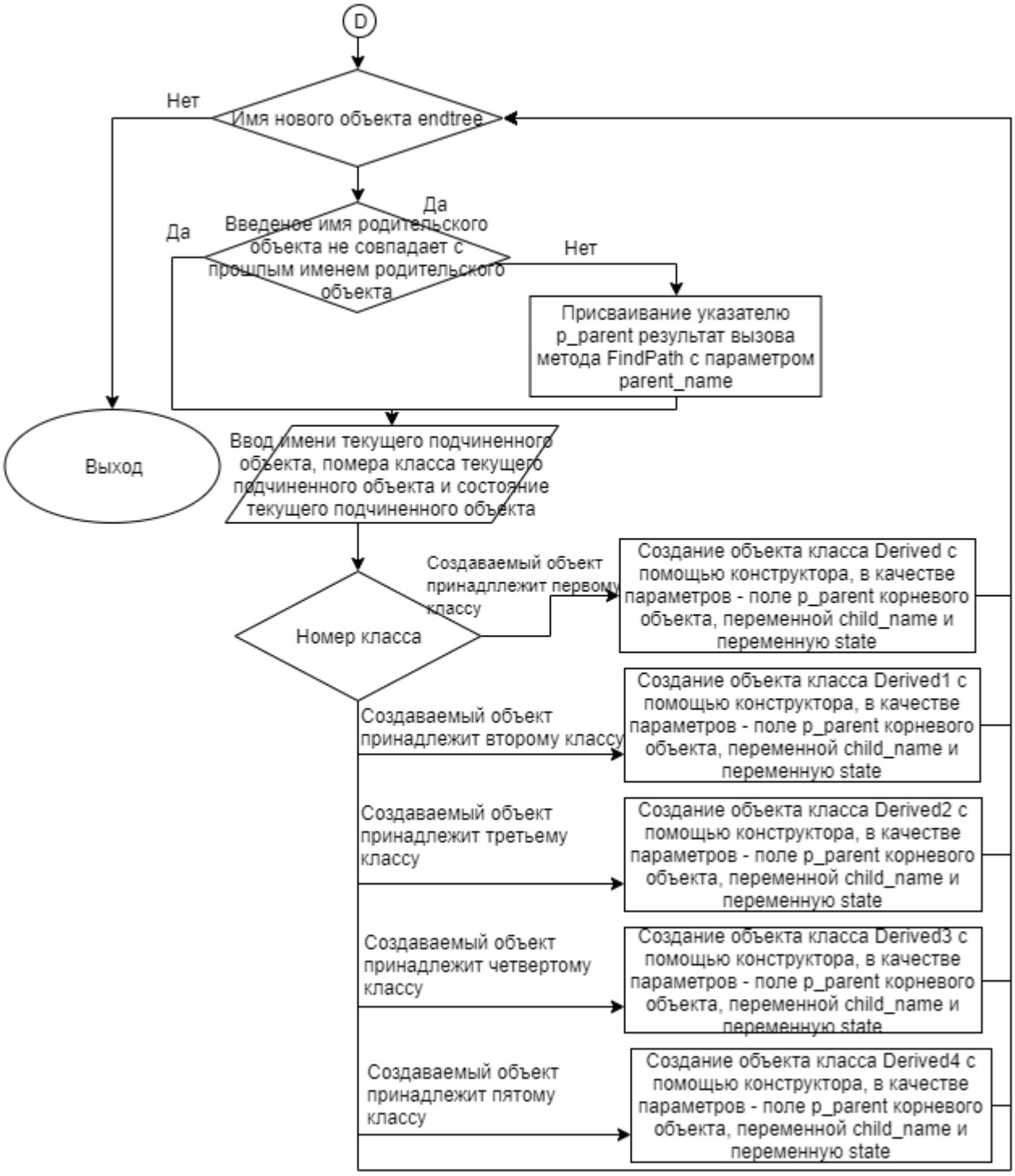

Рис. 7. Блок-схема алгоритма.

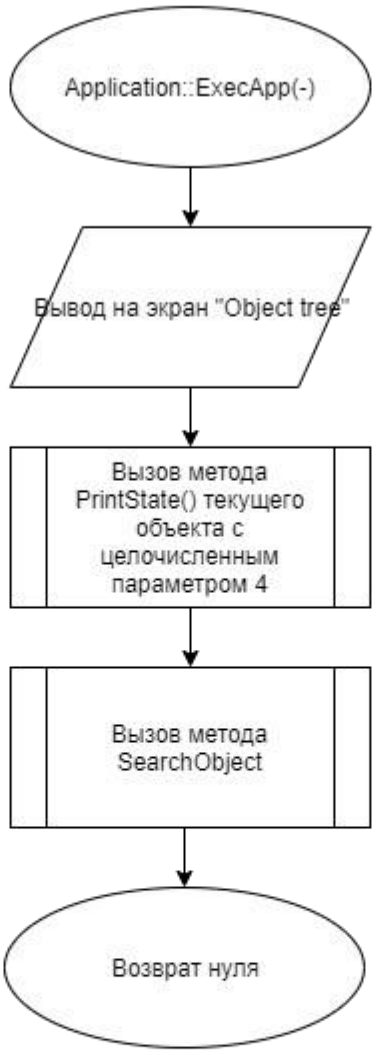

Рис. 8. Блок-схема алгоритма.

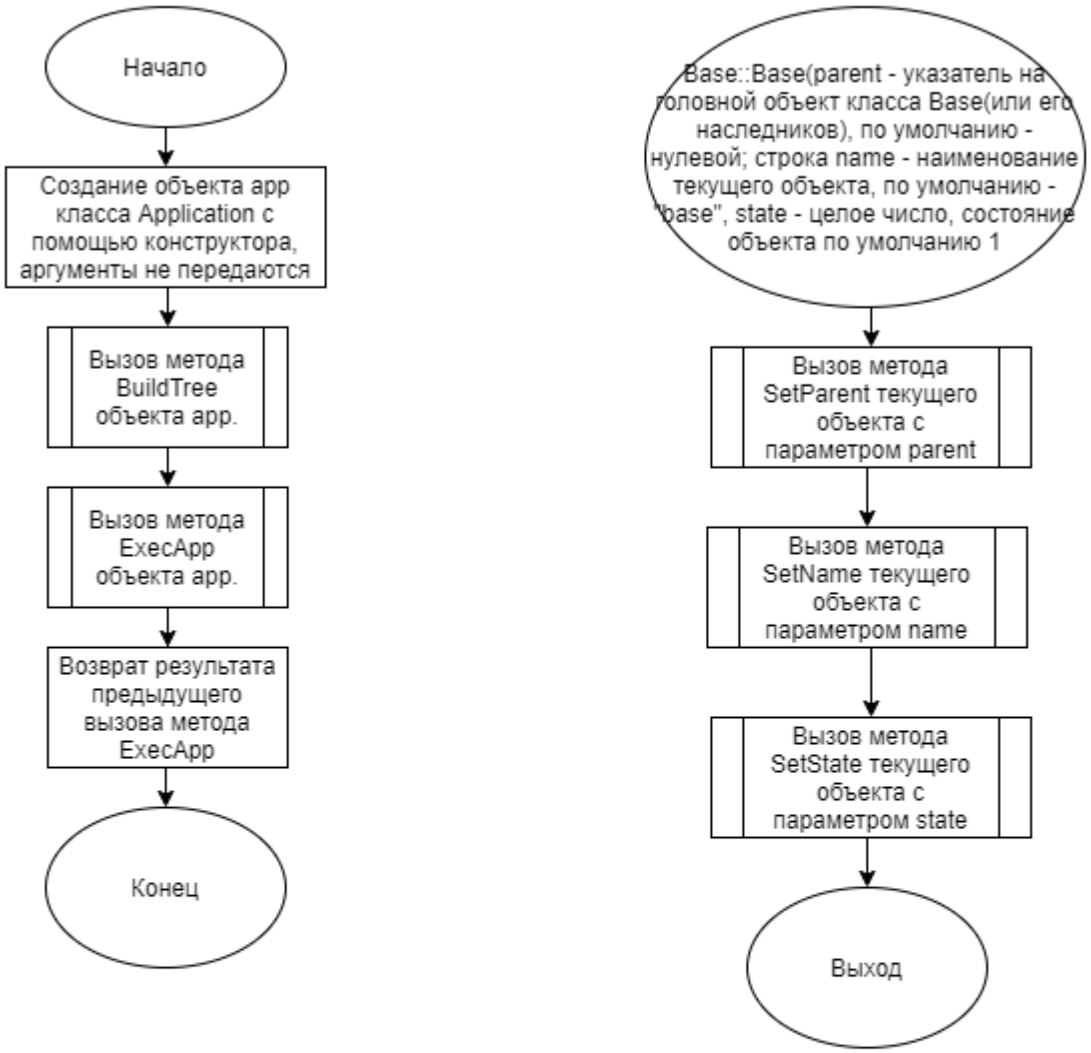

Рис. 9. Блок-схема алгоритма.

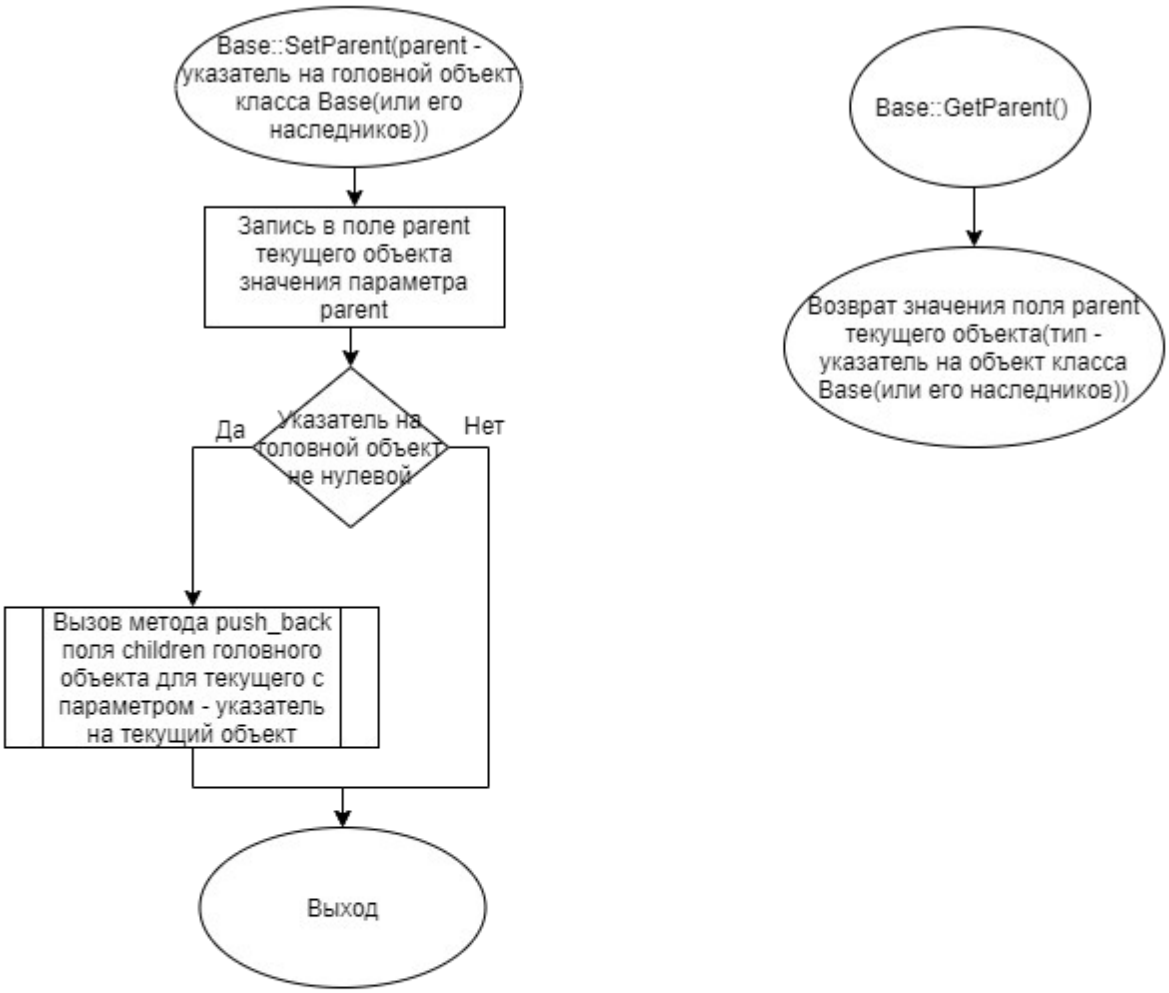

Рис. 10. Блок-схема алгоритма.

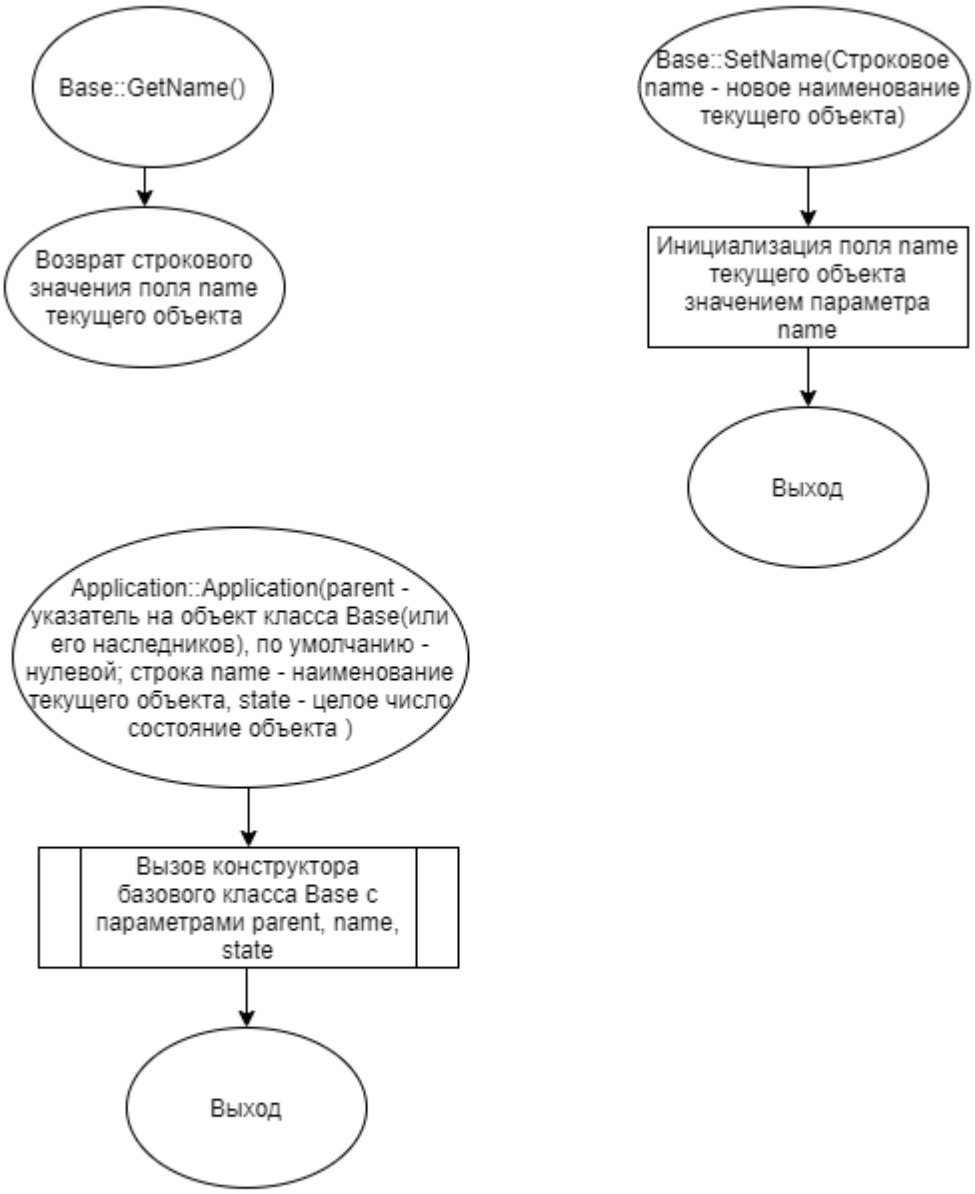

Рис. 11. Блок-схема алгоритма.

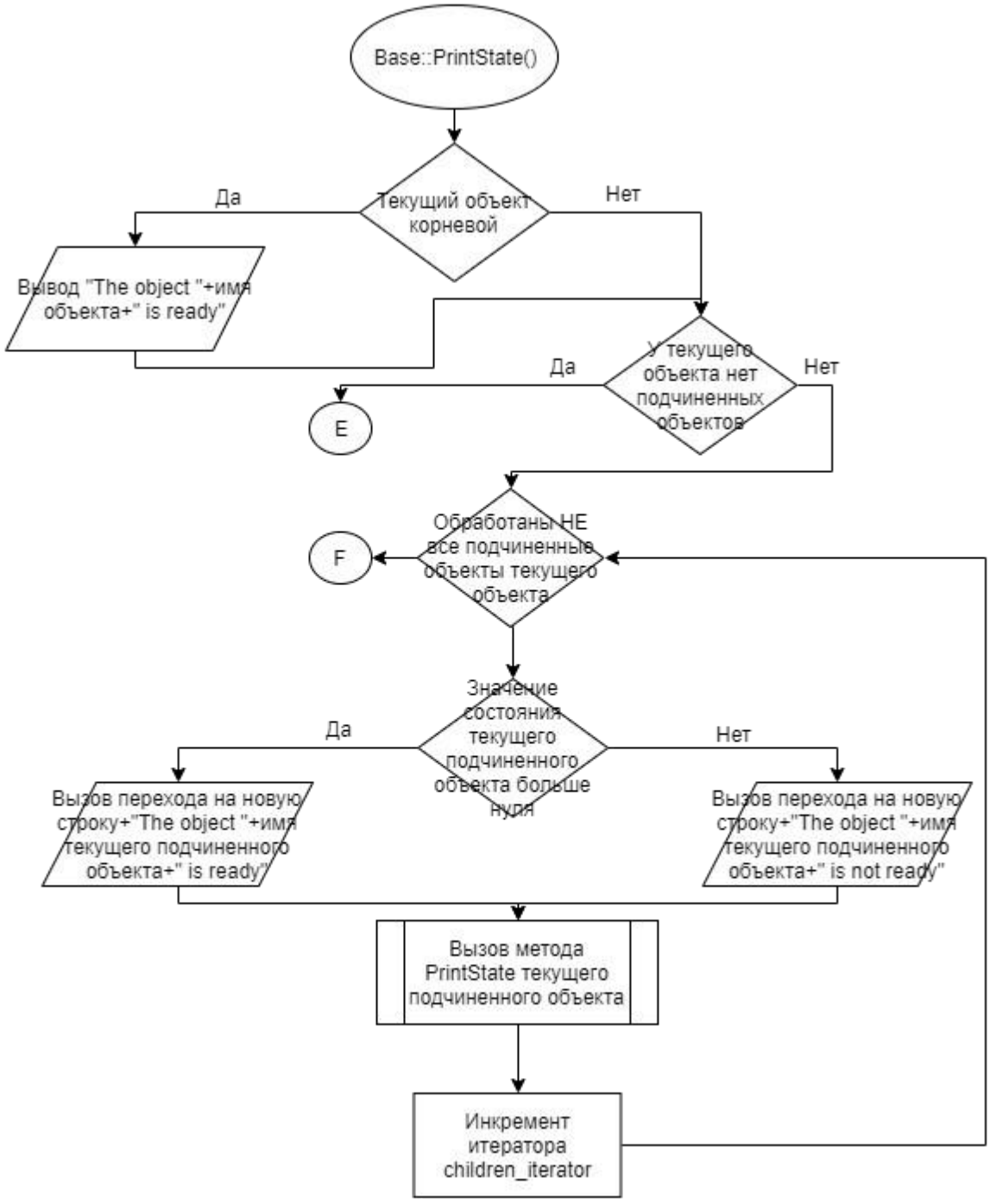

Рис. 12. Блок-схема алгоритма.

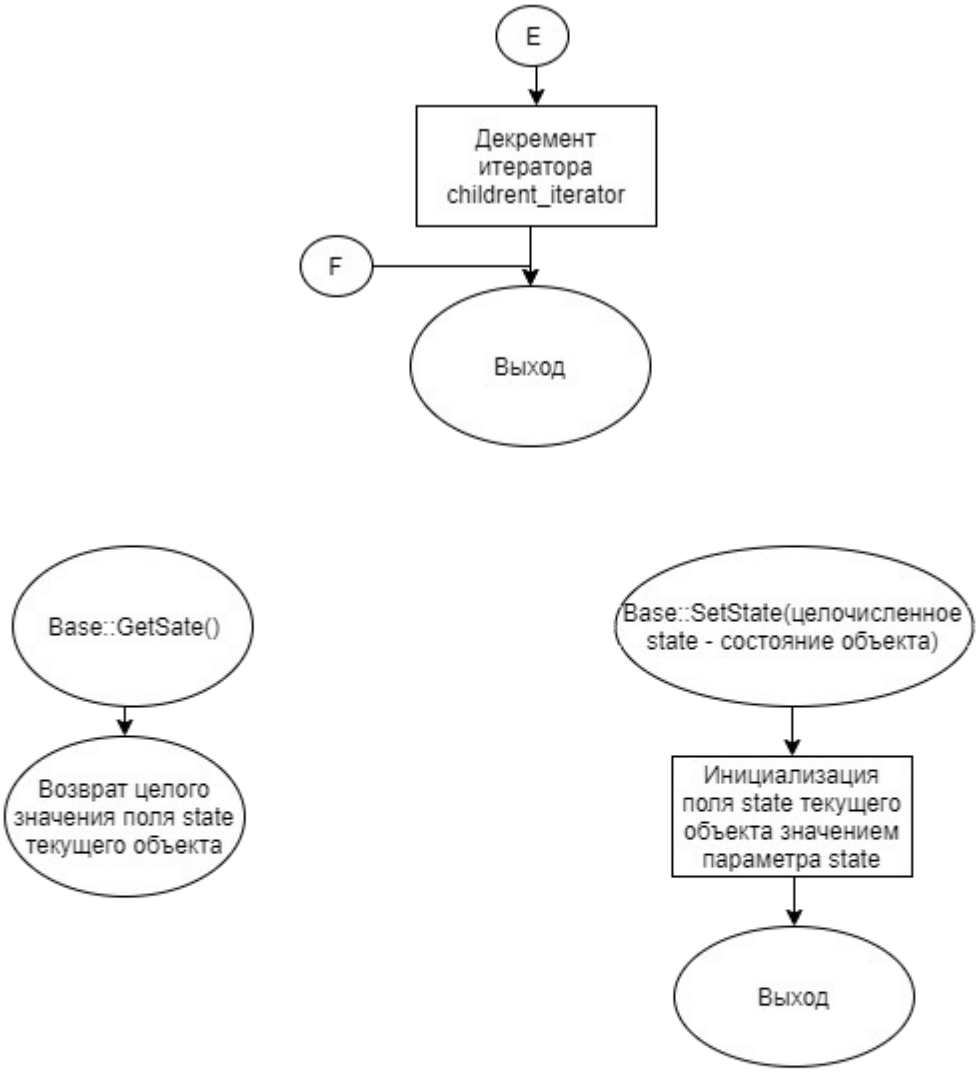

Рис. 13. Блок-схема алгоритма.

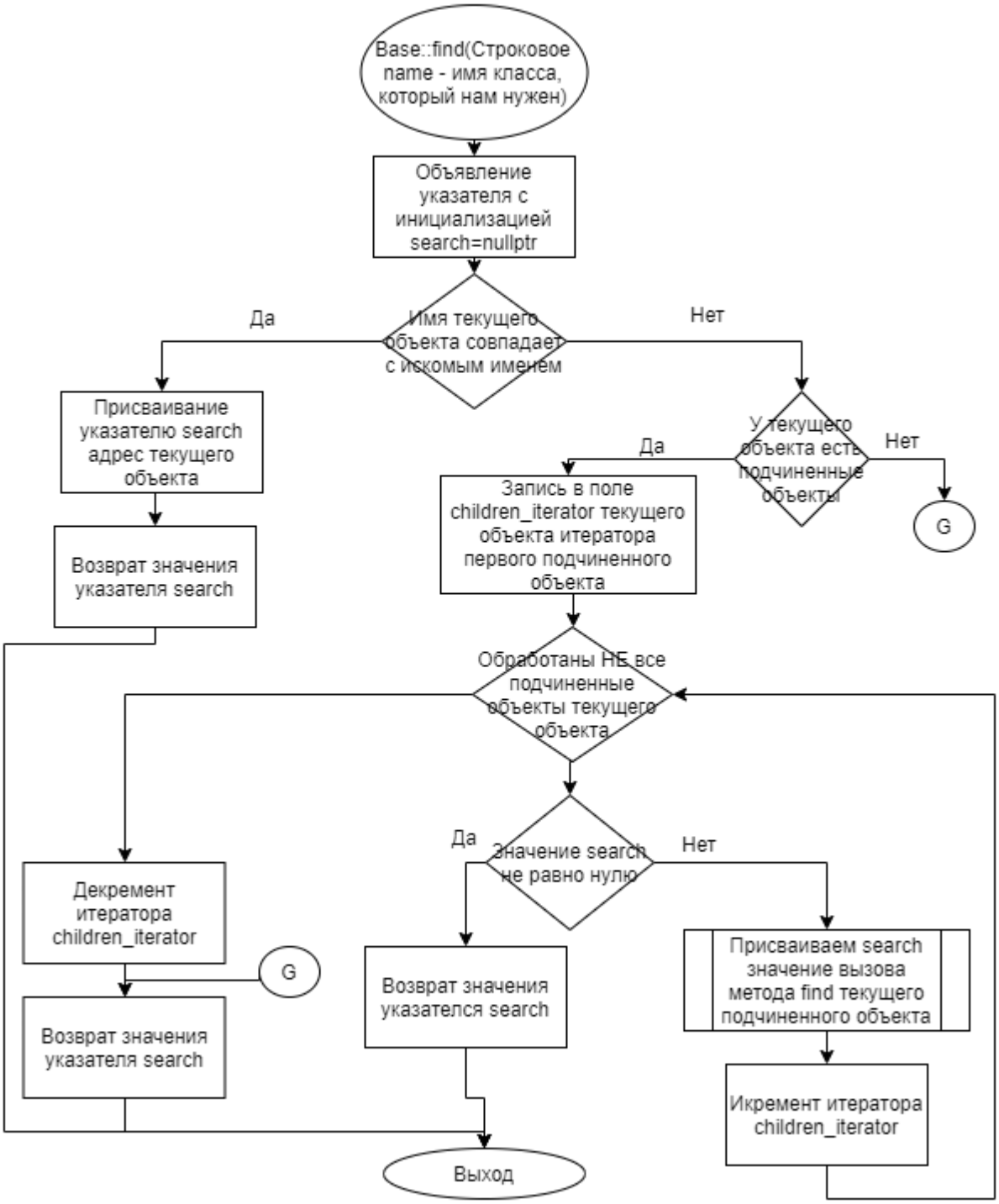

Рис. 14. Блок-схема алгоритма.

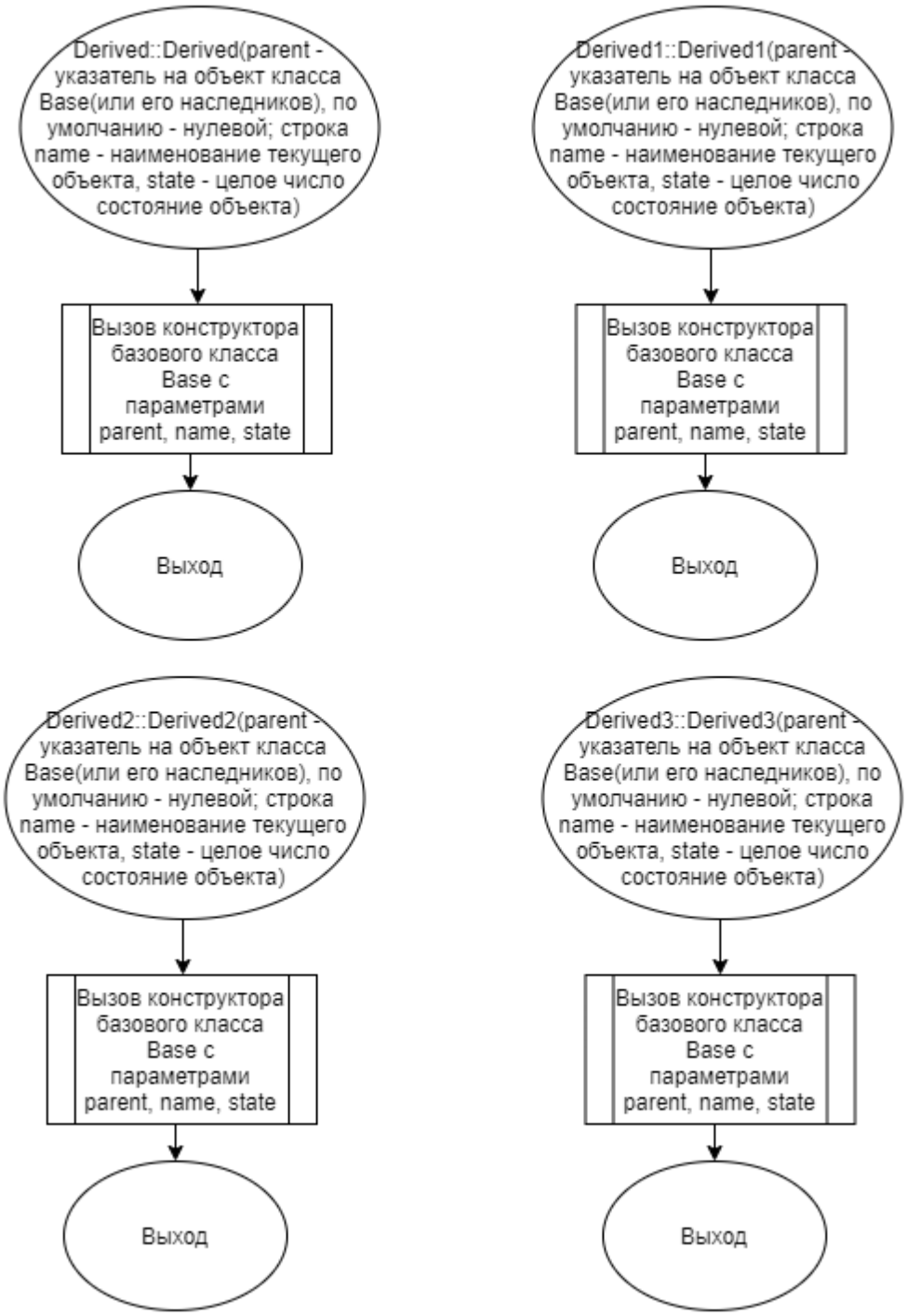

Рис. 15. Блок-схема алгоритма.

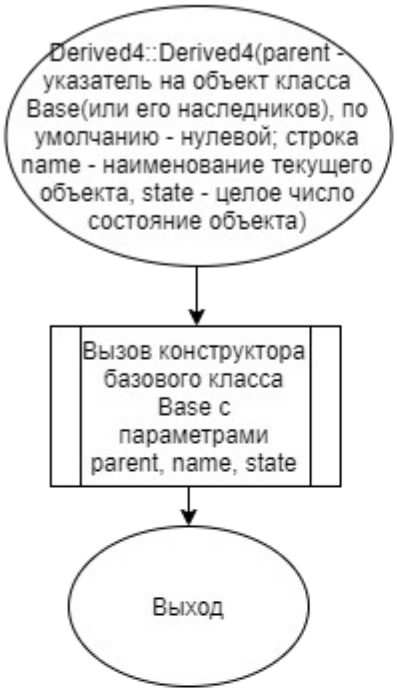

Рис. 16. Блок-схема алгоритма.

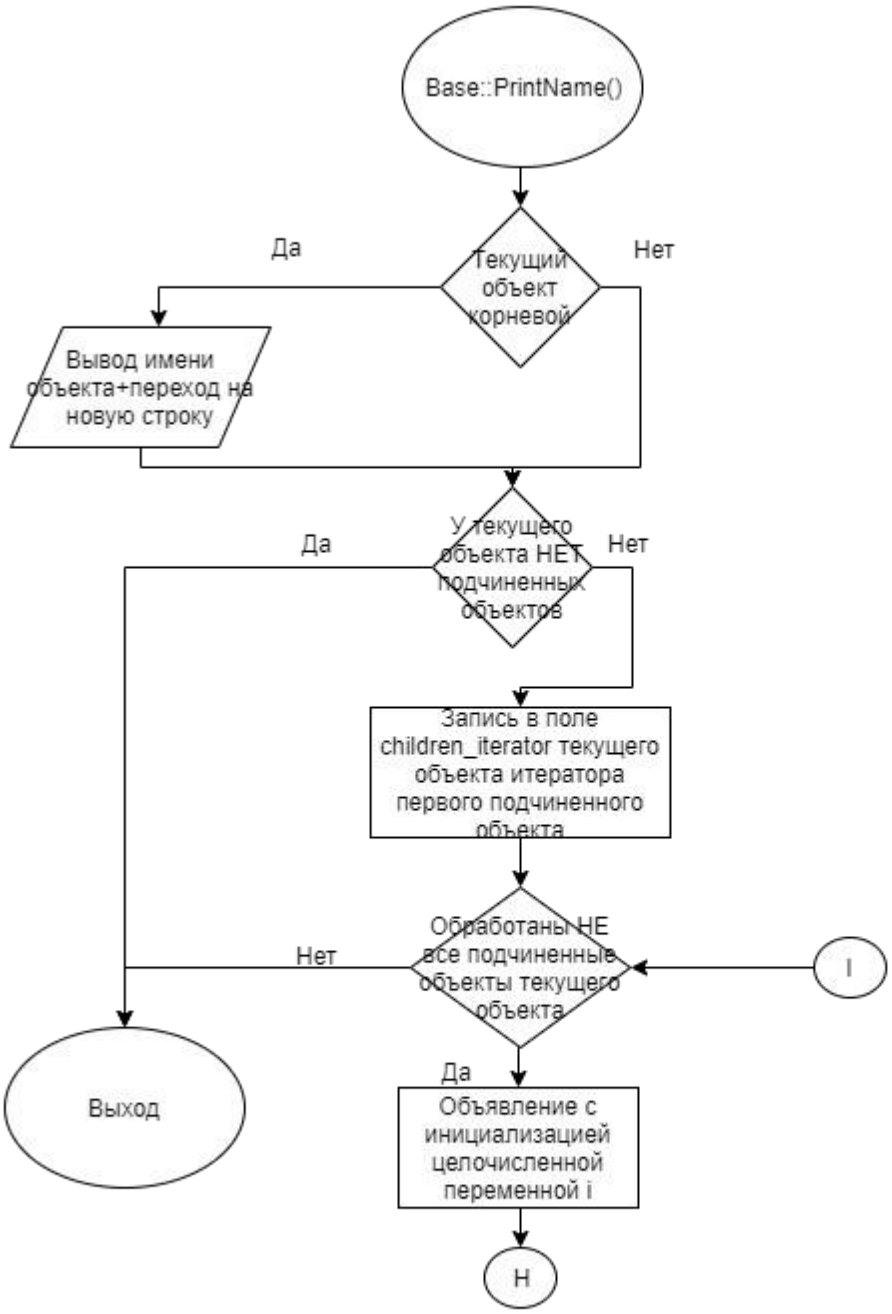

Рис. 17. Блок-схема алгоритма.

Рис. 18. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Base.cpp

```
#include "Base.h"
#include <iostream>

using namespace std;

Base::Base(Base*parent, int state, string name)
{
	SetParent(parent);
	SetName(name);
	SetState(state);
}
void Base::SetName(string name)
{
	this->name=name;
}
void Base::SetState(int state)
{
	this->state=state;
}
string Base::GetName()
{
	return this->name;
}
int Base::GetState()
{
	return this->state;
}
void Base::SetParent(Base* parent)
{
	this->parent=parent;
	if (parent)
	{
		parent->children.push_back(this);
	}
}
Base* Base::GetParent()
{
	return this->parent;
}
void Base::PrintName(int k)
```

```
{
	 if (this->parent==0)
	{
		cout << this->name;
	}
	if  (children.empty()) return;
	children_iterator=children.begin();
	while(children_iterator!=children.end())
	{
		cout << "\n";
		for (int i=0; i<k; i++) cout << " ";
		cout << (*children_iterator)->GetName();
		(*children_iterator)->PrintName(k+4);
		children_iterator++;
	}
}
void Base::PrintState()
{
	if (this->parent==0)
	{
		cout<<"The object "<<this->name<<" is ready";
	}
	if (children.empty()) return;
	children_iterator=children.begin();
	while (children_iterator != children.end())
	{
		if ((*children_iterator)->GetState()>0)
		{
			cout<<"\n"<< "The object "<<(*children_iterator)->GetName()<<" is ready";
			(*children_iterator)->PrintState();
		}
		else
		{
			cout<<"\n"<< "The object "<<(*children_iterator)->GetName()<<" is not
ready";
			(*children_iterator)->PrintState();
		}
		children_iterator++;
	}
}
Base* Base::find(string ParentName)
{
	Base* search=nullptr;
	if (this->name==ParentName)
	{
		search=this;
		return search;
	}
	else
```

```
	{
		if(!children.empty())
		{
			children_iterator=children.begin();
			while (children_iterator!=children.end())
			{
				if (search!=0) return search;
				search=(*children_iterator)->find(ParentName);
				++children_iterator;
			}
			--children_iterator;
		}
		return(search);
	}
}
Base* Base::FindPath(std::string path)
{
	Base* ParentName=nullptr;
	if (path[0]=='/' && path[1]=='/')
	{
		path.erase(0,2);
		return find(path);
	}
	else
	{
		int cnt=2;
		std::string name;
		if (path.size()==0)
			return this;
		name=path[1];
		while (path[cnt]!='/' && cnt<path.size()) {
			name+=path[cnt];
			cnt++;
		}
		path.erase(0,cnt);
		if (this->GetName()==name)
			ParentName=this;
		children_iterator=children.begin();
		while (children_iterator!=children.end()) {
			if ((*children_iterator)->GetName()==name){
				ParentName=*children_iterator;
				break;
			}
			++children_iterator;
		}
		if (ParentName==nullptr)
			return nullptr;
		ParentName=ParentName->FindPath(path);
	}
```

```
	return ParentName;
}
void Base::SearchObject()
{
	std::string path;
	cin>>path;
	while (path!="//")
	{
		Base* temp;
		temp=FindPath(path);
		if (temp!=nullptr)
		{
			cout<<"\n";
			cout<<path<<" Object name: "<<temp->GetName();
		}
		else
			cout<<"\n"<<path<<" Object not found";
		cin>>path;
	}
}
void Base::PrintPath()
{
	Base* parentPtr=this->GetParent();
	if (parentPtr!=0)
	{
		parentPtr->PrintPath();
	}
	cout<<"/"<<this->GetName();
}
```

Файл Base.h

```
#ifndef HEAD_H
#define HEAD_H
#include <string>
#include<iostream>
#include<vector>
using namespace std;
	class Base
	{
	private:
		string name;
		int state;
		Base* parent;
		vector <Base*> children;
		vector <Base*>::iterator children_iterator;
	public:
		Base(Base*, int, string="base");
		void SetName(string);
		string GetName();
		void SetParent(Base*);
		Base* GetParent();
		Base* find(string);
		void SetState(int);
		int GetState();
		void PrintName(int);
		void PrintState();
		Base* FindPath(string);
		void SearchObject();
		void PrintPath();
};
#endif
```

Файл Application.cpp

```
#include "Application.h"
#include "Derived.h"
#include "Derived1.h"
#include "Derived2.h"
#include "Derived3.h"
#include "Derived4.h"
#include <iostream>

using namespace std;

Application::Application (Base*parent, int state) : Base(parent, state){}
Application::Application (Base* parent, int state, string name): Base(parent, state, name){}
```

```
void Application::BuildTree(){
	string parent_name, child_name;
	int ClassNum, state;
	cin>> parent_name;
	SetName(parent_name);
	p_parent=this;
	app_parent=this; //выполняет роль указателя на корневой объект
	do {
		cin>>parent_name;
		if (parent_name=="endtree") return;
		p_parent=FindPath(parent_name);
		cin>> child_name>>ClassNum>>state;
		switch (ClassNum)
		{
			case 2:
				c_child = new Derived (p_parent, state, child_name);
				break;
			case 3:
				c_child = new Derived1 (p_parent, state, child_name);
				break;
			case 4:
				c_child = new Derived2 (p_parent, state, child_name);
				break;
			case 5:
				c_child = new Derived3 (p_parent, state, child_name);
				break;
			case 6:
				c_child = new Derived4 (p_parent, state, child_name);
				break;
		}
	}
	while(true);
}
int Application::ExecApp()
{
	cout<<"Object tree"<< endl;
	PrintName(4);
	SearchObject();
	return 0;
}
```

Файл main.cpp

```
#include "Base.h"
#include "Application.h"
int main()
{
	Application app;
	app.BuildTree();
	return app.ExecApp();
}
```

Файл Application.h

```
#ifndef APP_H
#define APP_H
#include "Base.h"

	class Application: public Base
	{
	Base* p_parent;
	Base* c_child;
	Base* app_parent;
	public:
	Application(Base* parent=0, int state=1);
	Application(Base* parent, int state, string name);
	void BuildTree();
	int ExecApp();
};
#endif
```

Файл Derived1.cpp

```
#include "Derived1.h"
#include <string>

using namespace std;

Derived1::Derived1 (Base* parent, int state, string name): Base(parent, state, name){}
```

Файл Derived1.h

```
#ifndef DERIVED1_H
#define DERIVED1_H
#include "Base.h"
class Derived: public Base
{
public:
```

```
        Derived1(Base*, int, string);
};
#endif
```

Файл Derived2.cpp

```
#include "Derived2.h"
#include <string>

using namespace std;

Derived2::Derived2(Base* parent, int state, string name): Base(parent, state, name){}
```

Файл Derived2.h

```
#ifndef DERIVED2_H
#define DERIVED2_H
#include "Base.h"
class Derived2: public Base
{
public:
        Derived2(Base*, int, string);
};
#endif
```

Файл Derived3.cpp

```
#include "Derived3.h"
#include <string>

using namespace std;

Derived3::Derived3(Base* parent, int state, string name): Base(parent, state, name){}
```

Файл Derived3.h

```
#ifndef Derived3_H
#define Derived3_H
#include "Base.h"
class Derivede3: public Base
{
public:
        Derived3(Base*, int, string);
};
#endif
```

Файл Derived4.cpp

```
#include "Derived4.h"
#include <string>

using namespace std;

Derived4::Derived4(Base* parent, int state, string name): Base(parent, state, name){}
```

Файл Derived4.h

```
#ifndef DERIVED4_H
#define DERIVED4_H
#include "Base.h"
class Derived4: public Base
{
public:
	Derived4(Base*, int, string);
};
#endif
```

Файл Derived.cpp

```
#include "Derived.h"
#include <string>

using namespace std;

Derived::Derived(Base* parent, int state, string name): Base(parent, state, name){}
```

Файл Derived.h

```
#ifndef DERIVED_H
#define DERIVED_H
#include "Base.h"
class Derived: public Base
{
public:
	Derived(Base*, int, string);
};
#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
app endtree //	Object tree app	Object tree app
app /app object1 3 1 /app object2 2 1 /app/object2 object4 3 -1 /app/object2 object5 4 1 /app object3 3 1 /app/object2 object3 6 1 /app/object1 object7 5 1 /app/object2/object4 object7 3 -1 endtree /object1 /object2 /object7 /object12 /app/object2 //	Object tree app object1 object7 object2 object4 object7 object5 object3 object3 /object1 Object name: object1 /object2 Object name: object2 /object7 Object not found /object12 Object not found /app/object2 Object name: object2	Object tree app object1 object7 object2 object4 object7 object5 object3 object3 /object1 Object name: object1 /object2 Object name: object2 /object7 Object not found /object12 Object not found /app/object2 Object name: object2

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы я поняла, что в самом общем смысле объектно-ориентированное программирование - это программирование, основанное на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.ООП как стиль написания программ подразумевает построение структуры, состоящей из взаимодействующих объектов. В рамках ООП нужно научиться мыслить этими объектами. Объекты расположены в иерархии, самостоятельно и как-то взаимодействуют. Программа состоит из модулей - блоков, которые решают задачи. Изменения в этих участках могут не отражаться на других участках.

Благодаря ООП, студенты учатся мыслить гибко, быстро, выстраивая правильную архитектуру. Лично для себя, я выделила следующие обретенные навыки:

- Разрабатывать базовый класс для объектов;
- Определять общий функционал для используемых в рамках приложения объектов;
- Разрабатывать операции добавления, удаления, изменения позиции объекта в рамках иерархического дерева;
- Построение дерева иерархии объектов;
- Освоение алгоритмов обработки структур данных в виде дерева;
- Переключать состояния объектов и определять их готовность к работе;
- Выводить на печать дерево иерархии объектов;
- Искать указатель на объект по координате, подереву иерархии объектовили по имени, при уникальности наименований объектов.

Моя точка зрения: при хорошем уровне объектного мышления и правильном выборе абстракций и аналогий ООП упрощает систему, позволяет писать без

повторов и быстро управлять кодом, добавляя объекты в классы. Многие популярные языки программирования - ОО, объектно - ориентированные,поэтому, понимая принципы ООП, легче изучать языки.

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_laboratorny h_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).