Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Формирование и работа с иерархией объектов программы-системы.

Создание объектов и построение исходного иерархического дерева объектов.

Система собирается из объектов, принадлежащих определенным классам. В тексте постановки задачи классу соответствует уникальный номер. Относительно номера класса определяются требования (свойства, функциональность).

Первоначальная сборка системы (дерева иерархии объектов, программы) осуществляется исходя из входных данных. Данные вводятся построчно.

Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Корневой объект объявляется в основной программе (main).

Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных программах динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей,

существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны.

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

Расширить функциональность базового класса:

- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr);
- метод вывода дерева иерархии объектов;
- метод вывода дерева иерархии объектов и отметок их готовности;
- метод установки готовности объекта реализовать (доработать) следующим образом.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется.

При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

1. Построить дерево объектов системы (в методе коневого объекта построения исходного дерева объектов).

2. В методе корневого объекта запуска моделируемой системы реализовать:

2.1. Вывод на консоль иерархического дерева объектов в следующем виде:

root

ob_1

ob_2

ob_3

ob_4

ob_5

ob_6

ob_7

где: root - наименование корневого объекта (приложения).

2.2. Переключение готовности объектов согласно входным данным (командам).

2.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

root is ready

ob_1 is ready

ob_2 is ready

ob_3 is ready

ob_4 is not ready

ob_5 is not ready

ob_6 is ready

ob_7 is not ready

1.1 Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии.

Последовательность ввода организовано так, что головной объект для очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка

«Наименование корневого объекта»

Со второй строки

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

. . . . .

endtree

Со следующей строки вводятся команды включения или отключения объектов

«Наименование объекта» «Номер состояния объекта»

. . . . .

Признаком завершения ввода является конец потока входных данных.

Пример ввода

app_root

app_root object_01 3

app_root object_02 2

object_02 object_04 3

object_02 object_05 5

object_01 object_07 2

endtree

app_root 1

object_07 3

object_01 1

object_02 -2

object_04 1

1.2 Описание выходных данных

Вывести иерархию объектов в следующем виде:

Object tree

«Наименование корневого объекта»

«Наименование объекта 1»

«Наименование объекта 2»

«Наименование объекта 3»

. . . . .

The tree of objects and their readiness

«Наименование корневого объекта» «Отметка готовности»

«Наименование объекта 1» «Отметка готовности»

«Наименование объекта 2» «Отметка готовности»

«Наименование объекта 3» «Отметка готовности»

. . . . .

«Отметка готовности» - равно «is ready» или «is not ready»

Отступ каждого уровня иерархии 4 позиции.

Пример вывода

```
Object tree
app_root
    object_01
        object_07
    object_02
        object_04
        object_05
The tree of objects and their readiness
app_root is ready
    object_01 is ready
        object_07 is not ready
    object_02 is ready
        object_04 is ready
        object_05 is not ready
```

2 МЕТОД РЕШЕНИЯ

Для решения поставленной задачи используются:

Объекты стандартных потоков ввода и вывода cin и cout

Объект root класса Application

Объеты классов Two, Three, Four, Five, Six

Класс CBase

Свойства/поля:

- поле, хранящее значения состояния объекта
  - имя - readiness
  - тип целочисленный
  - модификатор доступа - private

Методы

5.0printTree (целочисленная перемения level со значением по умолчанию 1, булевая переменная readiness со занчением по умолчанию false)

1. функционал - вывод дерева объектов в терминал
2. модификатор доступа - public

5.1printTreeWithReadiness()

функционал - вывод дерева объектов и отметки их готовности в терминал

модификатор доступа - public

5.2findByName(переменная строкового типа name, указатель на currentRoot со значением по умолчанию nullptr)

функционал - поиск обьекта по имени в дереве

модификатор доступа - public

5.3setReadiness(целочисленная переменная readiness)

функционал - установка готовности объекта

модификатор доступа - public

5.4getRoot()

функционал - возвращает указатель на корневой объект

модификатор доступа - private

5.5getChild(строка name - имя объекта)

функционал - возвращает указатель на подчиненный объект

модификатор доступа - public

5.6CBase(указатель на base, переменная строкового типа name)

функционал - конструктор

модификатор доступа - public

Класс Application

Методы

bildTree()

функционал - построение полного дерева объектов

модификатор доступа - public

execute()

функционал - вызов необходимых для выполнения задания методов, возвращение кода ошибки

модификатор доступа - public

Класс Two

Методы

конструктор наследованный у класса CBase

Класс Three

Методы

конструктор наследованный у класса CBase

Класс Four

Методы

конструктор наследованный у класса CBase

Класс Five

Методы

конструктор наследованный у класса CBase

Класс Six

Методы

конструктор наследованный у класса CBase

Таблица 1 – Иерархия наследования классов

номер класса	имя класса	классы - наследники	модификатор доступа при наследовании	описание	номер	комментарий
1	CBase	Application	public	Базовый класс	2	
		Two	public		3	
		Three	public		4	
		Four	public		5	
		Five	public		6	
		Six	public		7	
2	Aplication	отсутсвует		класс приложение		
3	Two	отсутсвует		номерной класс		
4	Three	отсутсвует		номерной класс		
5	Four	отсутсвует		номерной класс		

6	Five	отсутсвует		номерной класс		
7	Six	**отсутсвует**		номерной класс		

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода printTree класса CBase

Функционал: Вывод дерева обьектов..

Параметры: int level с значением по умолчанию 1 - количество отступов, bool state с значением по умолчанию false - флаг для определения формата вывода.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода printTree класса CBase

№	Предикат	Действия	№ перехода
1	Текущий объект - корневой	Вывод назван	2
			4
2	параметр readiness имеет знаяение true		3
			4
3	Поле данного объекта readiness равно 0	Вывод в консоль "is ready"	4
		Вывод в консоль "is not ready"	4
4	Есть следующий элемент вектора указателей на подчиненные объекты	Вывод с новой строки отступов, количество которых определяется параметров level и название объекта из текущего элемента вектора	5
			∅
5	Параметр readiness имеет значение true		6

№	Предикат	Действия	№ перехода
			7
6	Поле объекта элемента вектора по указателю readiness не равно 0	Вывод в консоль "is ready"	7
		Вывод в консоль "is not ready"	7
7		Рекурсивный вызов метода для объектов по указателю из текущего элемента вектора с параметрами level + 1 и readiness	5

3.2 Алгоритм метода printTreeWithReadiness класса CBase

Функционал: Вывод дерева объектов и состояния их готовности в консоль.

Параметры: -.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода printTreeWithReadiness класса CBase

№	Предикат	Действия	№ перехода
1		Вызов метода текущего объетка printTree со значениями аргументов 1, true	∅

3.3 Алгоритм метода setReadiness класса CBase

Функционал: Установка готовности объекта.

Параметры: int readiness - значение состояния объекта.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода setReadiness класса CBase

№	Предикат	Действия	№ перехода
1	Параметр readiness равен 0	Полю readiness текущего объекта присваива значение параметра readiness	2
		Инициализация указателя на CBase root с указателем на текущий объект	3
2	Есть следующий элемент вектора указателей на подчиненные объекты	Рекурсивный вызов метода объекта по указателю из текущего элемента вектора с параметором readiness	2
			∅
3	Объект по указателю root не является корневым		4
		Присвоение полю readiness текущего объекта значение параметра readiness	∅
4	Значение поля readiness объекта выше по иерархии равно 0	Полю readiness текущего объекта присваивается значение 0	∅
		Присвоение указателю root указателя на объект выше по иерархии от текущего объекта	3

3.4 Алгоритм конструктора класса CBase

Функционал: Присвоение значений аргументов приватным полям класс, инициализация поля readiness со значением 0.

Параметры: CBase* parent - указатель на родительский объект, std::string name - название объекта.

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм конструктора класса CBase

№	Предикат	Действия	№ перехода
1		Присвоение полю name значение из параметров name	2
2		Присвоение полю parent значение из параметров parent	3
3		Присвоение полю readiness значение 0	4
4	указатель parent не равен nullptr	Добавление указателя на текущий объект в массив указателей подчиненных объектов для родительского объекта	∅
			∅

3.5 Алгоритм метода getRoot класса CBase

Функционал: Возвращает указатель на корневой объект в иерархии.

Параметры: -.

Возвращаемое значение: CBase*.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода getRoot класса CBase

№	Предикат	Действия	№ перехода
1		Инициализация указателя на CBase root указателем на текущий объект	2
2	Текущий объект не является корневым	Присвоение указателю root указателя на объект выше по иерархии от текущего объекта	2
		Возвращение указателя root	∅

3.6 Алгоритм метода buildTree класса Application

Функционал: Построение полного дерева иерархии объетов.

Параметры: -.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода buildTree класса Application

№	Предикат	Действия	№ перехода
1		Объявление переменных rootName и derivedName типа std::string	2
2		Объявление переменной classNum	3
3		Ввод пользователем значения переменной rootName	4
4		Вызов пользователем значения переменной rootName	5
5		Ввод пользователем значения переменной rootName	6
6	Значение переменной rootName равно "endtree"		∅
		Ввод пользователем значения переменных defivedName и classNum	7
7	Значение переменной classNum равно 2	Создание объекта класса Two с параметрами значения вызова функции findByName с параметрами rootName и derivedName	5
	Значение переменной classNum равно 3	Создание объекта класса Three с параметрами значения вызова функции findByName с параметрами rootName и derivedName	5
	Значение переменной classNum равно 4	Создание объекта класса Four с параметрами значения вызова функции findByName с параметрами rootName и derivedName	5
	Значение переменной classNum равно 5	Создание объекта класса Five с параметрами значения вызова функции findByName с параметрами rootName и derivedName	5

№	Предикат	Действия	№ перехода
	Значение переменной classNum равно 6	Создание объекта класса Six с параметрами значения вызова функции findByName с параметрами rootName и derivedName	5
			∅

3.7 Алгоритм метода execute класса Application

Функционал: Вызов необходимых методов работоспособности объекта и возвращение кода ошибки.

Параметры: -.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода execute класса Application

№	Предикат	Действия	№ перехода
1		Вывод в консоль "Object tree" с переходом на следующую строку	2
2		Вызов метода printTree для текущего объекта	3
3	Конец потока ввода еще не достигнут	Объявление переменной name типа std::string	4
		Вывод в консоль с новой строки "The tree of objects and their readiness"	7
4		Объявление переменной readiness типа int	5
5		Ввод пользователем значений переменных name и readiness	6
6		Вызов метода setReadiness с параметром name от текущего объекта	3
7		Вызов метода printTreeWithReadiness от текущего объекта	8

№	Предикат	Действия	№ перехода
8		Возвращение значение 0	∅

3.8 Алгоритм конструктора класса Two

Функционал: Создание объекта.

Параметры: CBase* parent - указатель на родительский объект, std::string name - название объекта.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса Two

№	Предикат	Действия	№ перехода
1		Создание объекта путем наследования конструктора CBase с параметрами CBase* parent и std::string name	∅

3.9 Алгоритм конструктора класса Three

Функционал: Создание объекта.

Параметры: CBase* parent - указатель на родительский объект, std::string name - название объекта.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса Three

№	Предикат	Действия	№ перехода
1		Создание объекта путем наследования конструктора CBase с параметрами CBase* parent и std::string name	∅

3.10 Алгоритм конструктора класса Four

Функционал: Создание объекта.

Параметры: CBase* parent - указатель на родительский объект, std::string name - название объекта.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса Four

№	Предикат	Действия	№ перехода
1		Создание объекта путем наследования конструктора CBase с параметрами CBase* parent и std::string name	∅

3.11 Алгоритм конструктора класса Five

Функционал: Создание объекта.

Параметры: CBase* parent - указатель на родительский объект, std::string name - название объекта.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса Five

№	Предикат	Действия	№ перехода
1		Создание объекта путем наследования конструктора CBase с параметрами CBase* parent и std::string name	∅

3.12 Алгоритм конструктора класса Six

Функционал: Создание объекта.

Параметры: CBase* parent - указатель на родительский объект, std::string name - название объекта.

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса Six

№	Предикат	Действия	№ перехода
1		Создание объекта путем наследования конструктора CBase с параметрами CBase* parent и std::string name	∅

3.13 Алгоритм функции main

Функционал: Точка входа в программу.

Параметры: -.

Возвращаемое значение: код ошибки или 0.

Алгоритм функции представлен в таблице 14.

Таблица 14 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объета root класса Application с параметром nullptr	2
2		Вызов метода buildTree объекта root	3
3		Вызов метода execute объекта root и возвращение значения этого метода	∅

3.14 Алгоритм метода findByName класса CBase

Функционал: Поиск объекта в дереве по имени.

Параметры: std::string name - имя объекта для поиска в дереве .

Возвращаемое значение: CBase* - указатель на найденный объект.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода findByName класса CBase

№	Предикат	Действия	№ перехода
1	имя текущего объекта (значение поля name) =	вернуть указатель на текущий объект	∅

№	Предикат	Действия	№ перехода
	имени из параметров (значению переменной name)		
			2
2		инициализация указателя на CBase root указателем на текущий объкт	3
3	root не равен nullptr - указателю на родителя корневого объекта		4
			5
4	поле name - имя родителя объекта ,на который указывает root = имени из параметров - значению переменной name	вернуть указатель на родителя root	∅
		root = указателю на родительский объект root	3
5		вернуть результат вызова метода getChild(name) для объекта на который указывает root - корня дерева	∅

3.15 Алгоритм метода getChild класса CBase

Функционал: Поиск объекта по имени в списке детей объекта и ниже по иерархии.

Параметры: std::string name - имя объекта для поиска.

Возвращаемое значение: CBase* - найденный объект.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода getChild класса CBase

№	Предикат	Действия	№ перехода
1		инициализация указателя на CBase child = первому элементу вектора children	2
2	есть следующий указатель на объект child в векторе children текущего объекта		3
			4
3	имя объекта child (поле name) = значению имени из параметров (name)	вернуть child - указетль на найденный объект	∅
		child = следующий элемент вектора children	2
4		инициализация указателя на CBase child = первому элементу вектора children	5
5	есть следующий указатель на объект child в векторе children текущего объекта		6
		вернуть nullptr	∅
6	размер вектора children объекта на который указывает child > 0		7
			9
7		инициализация указателя на CBase tempObj значением полученным рекурсивным вызовом функции getChild с параметром name для объекта child - идем в низ по дереву	8
8	tempObj != nullptr - то есть имя нашлось	вернуть tempObj	∅
			9
9		child = следующий элемент вектора children	5

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-17.

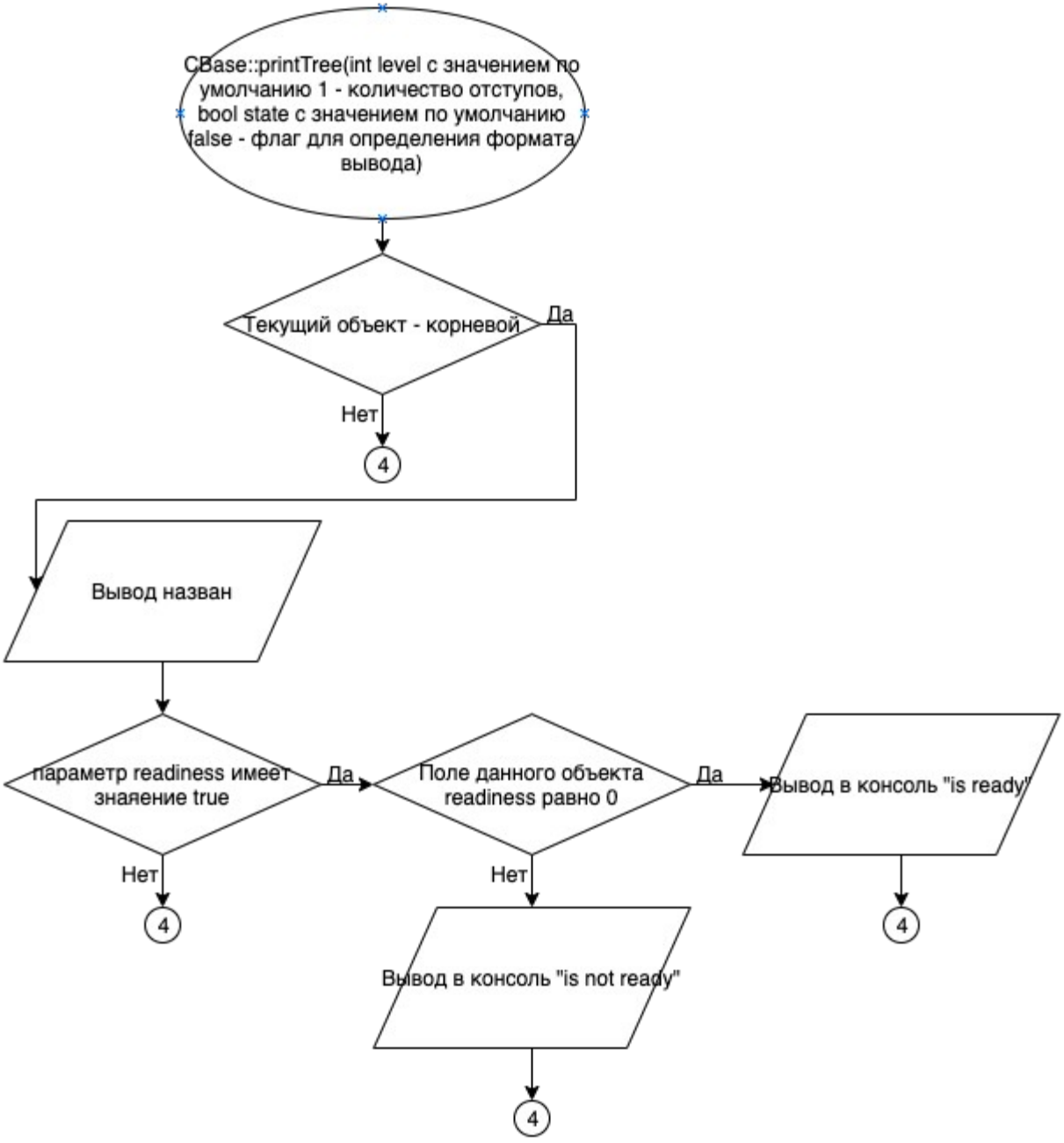

Рисунок 1 – Блок-схема алгоритма

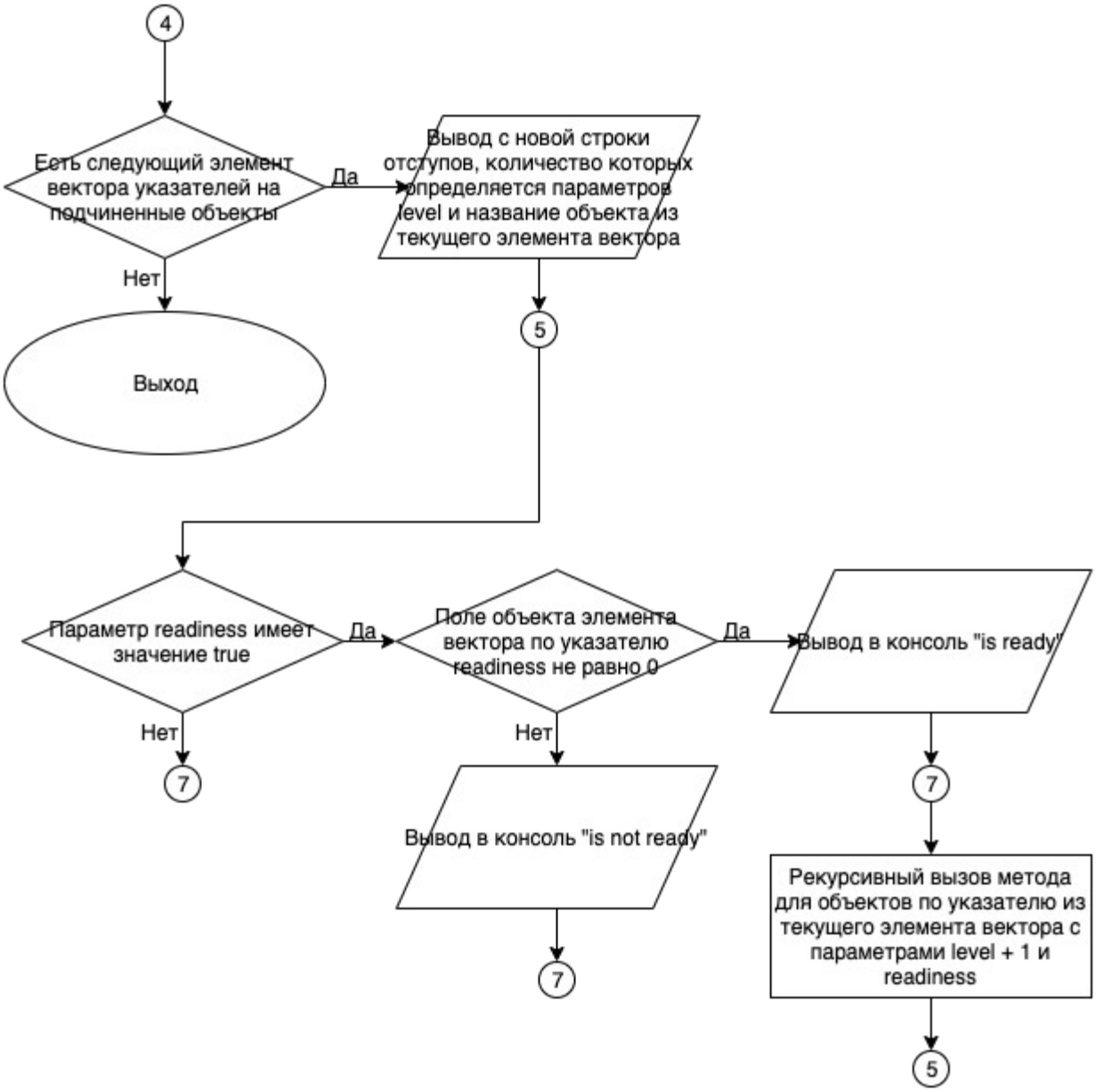

Рисунок 2 – Блок-схема алгоритма

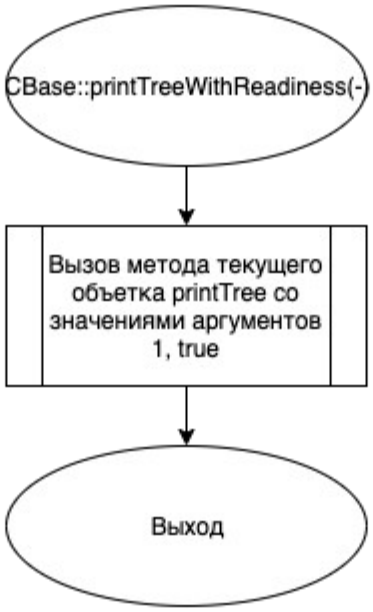

Рисунок 3 – Блок-схема алгоритма

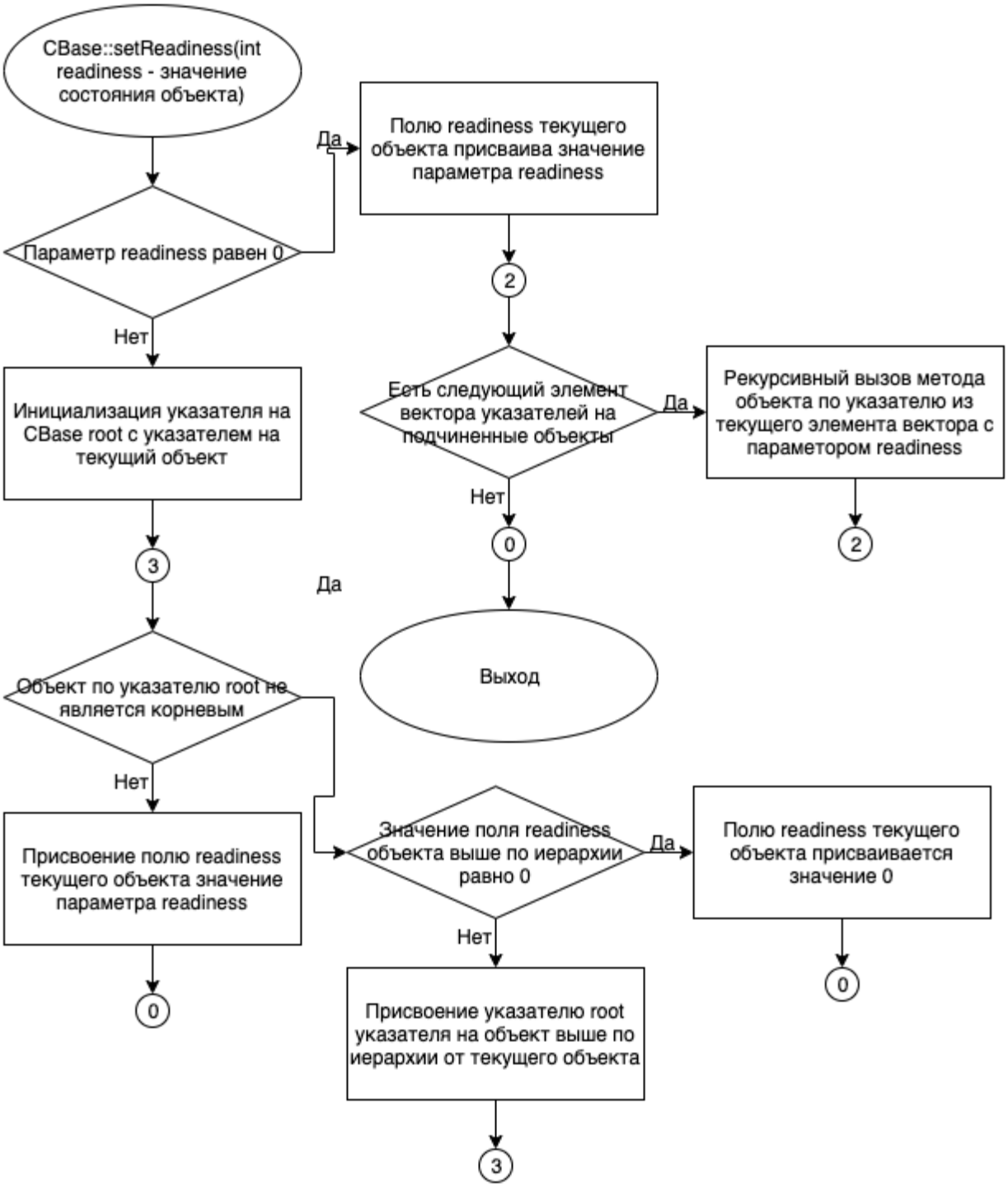

Рисунок 4 – Блок-схема алгоритма

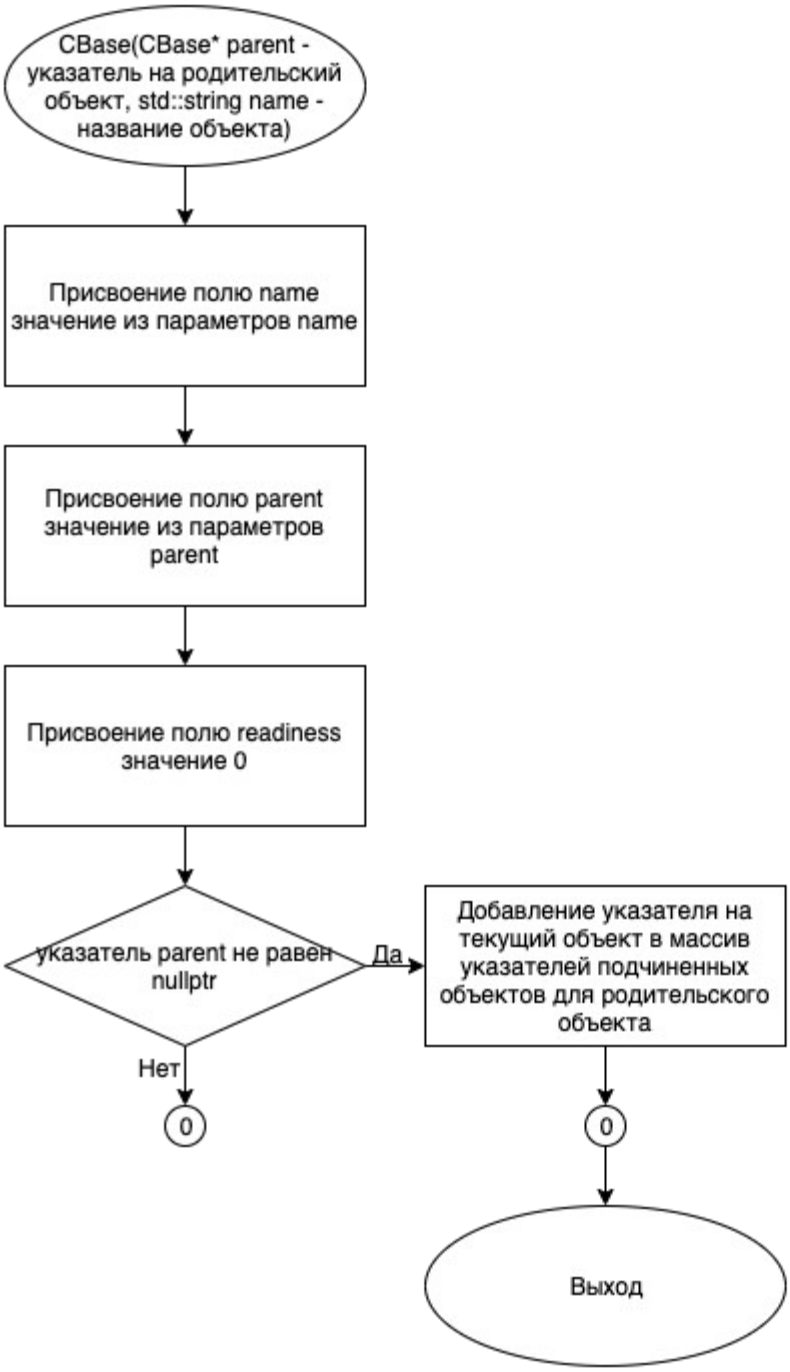

Рисунок 5 – Блок-схема алгоритма

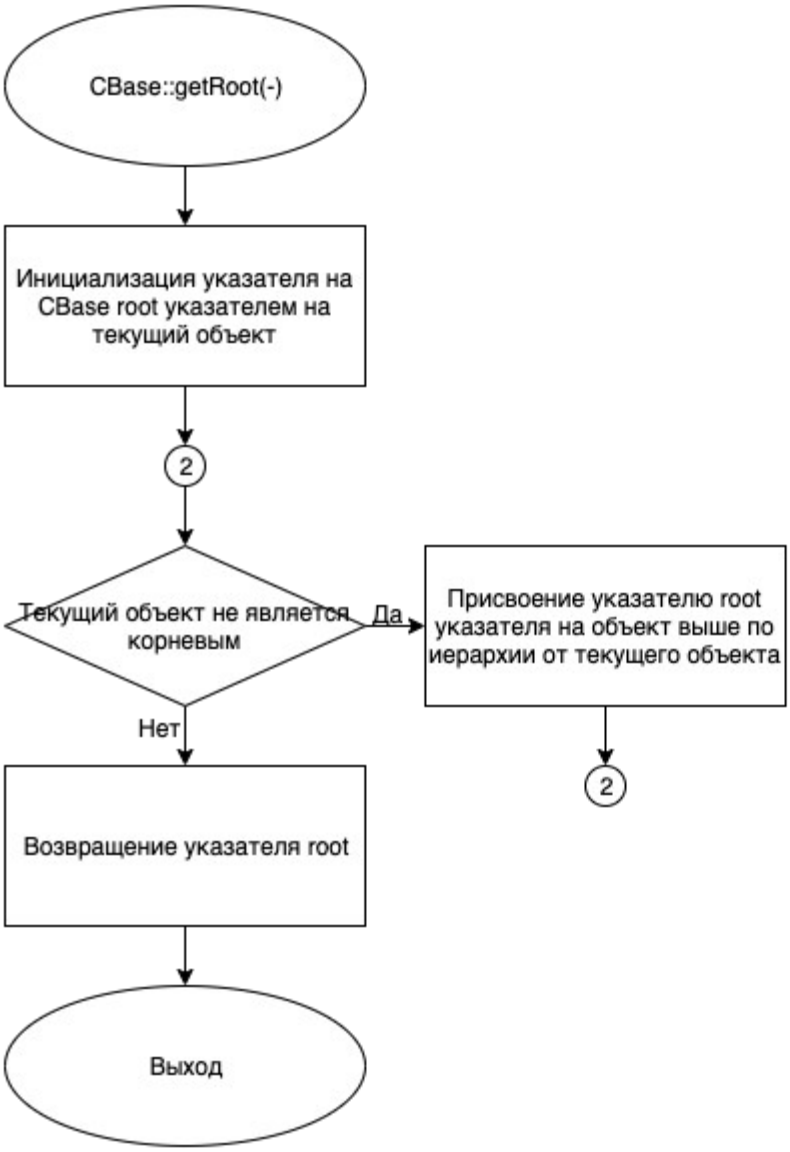

Рисунок 6 – Блок-схема алгоритма

Рисунок 7 – Блок-схема алгоритма

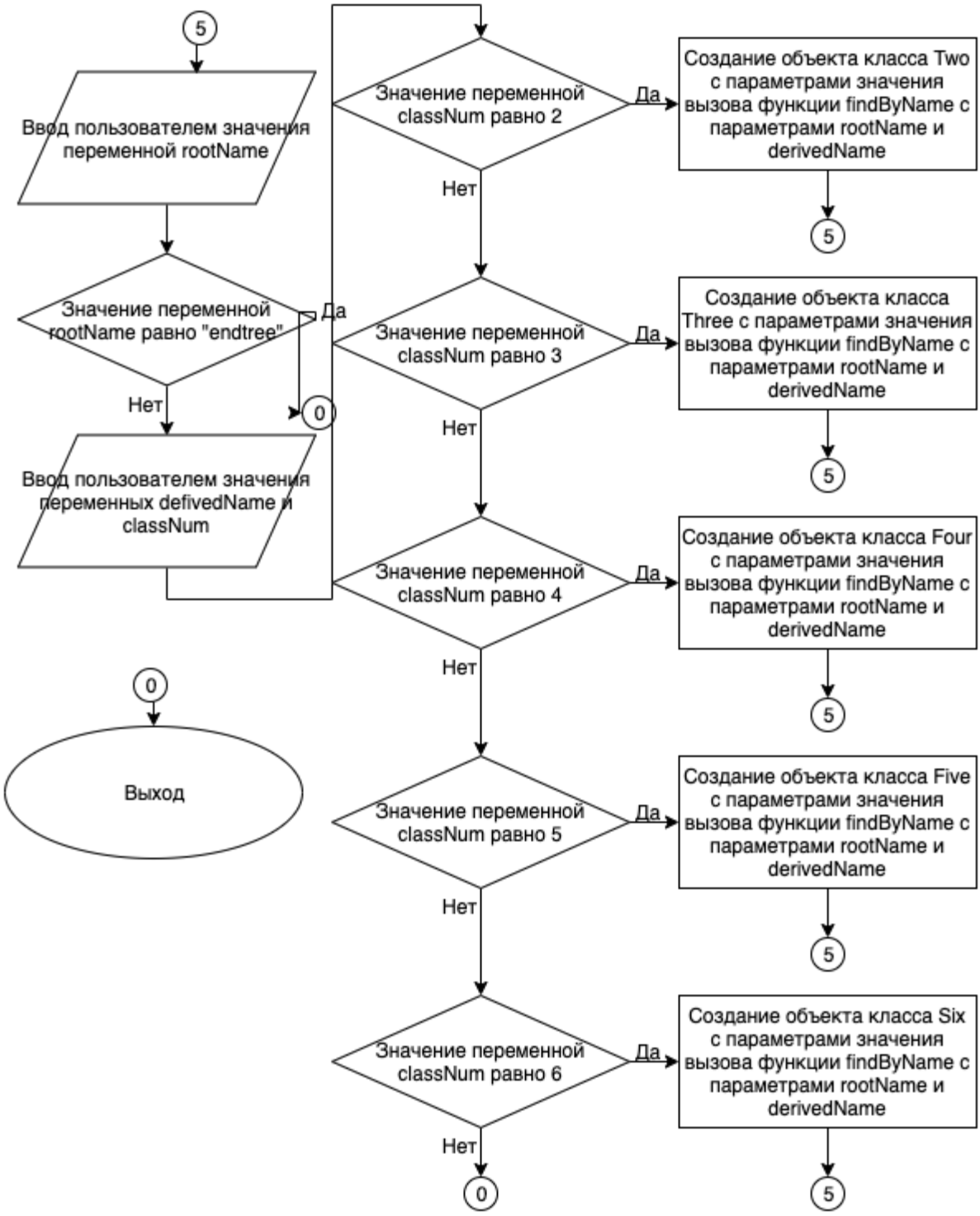

Рисунок 8 – Блок-схема алгоритма

Рисунок 9 – Блок-схема алгоритма

Рисунок 10 – Блок-схема алгоритма

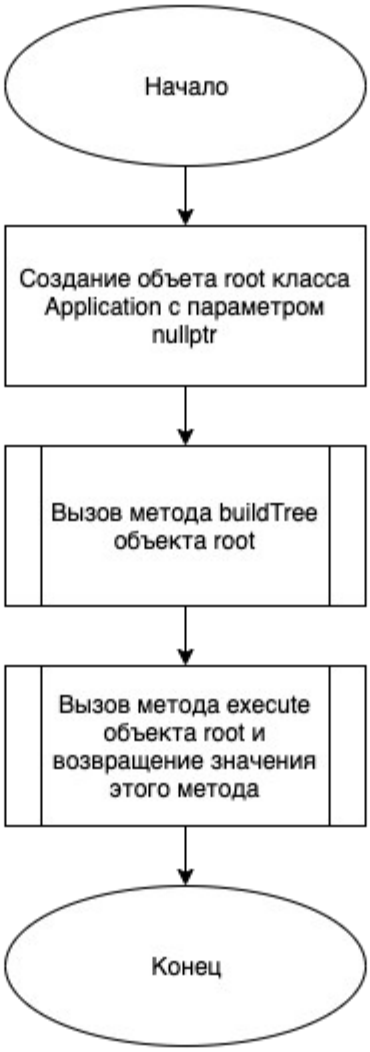

Рисунок 11 – Блок-схема алгоритма

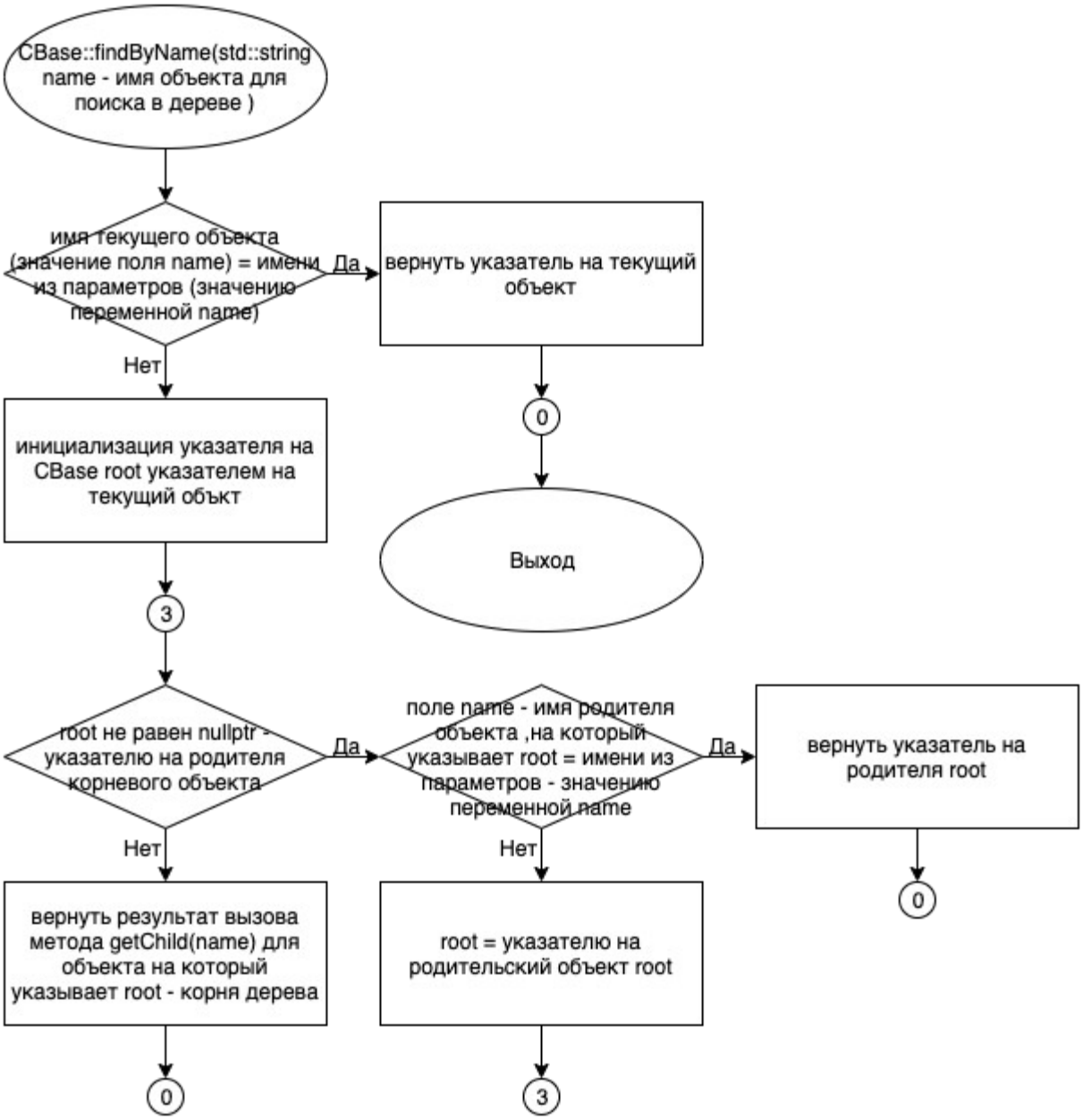

Рисунок 12 – Блок-схема алгоритма

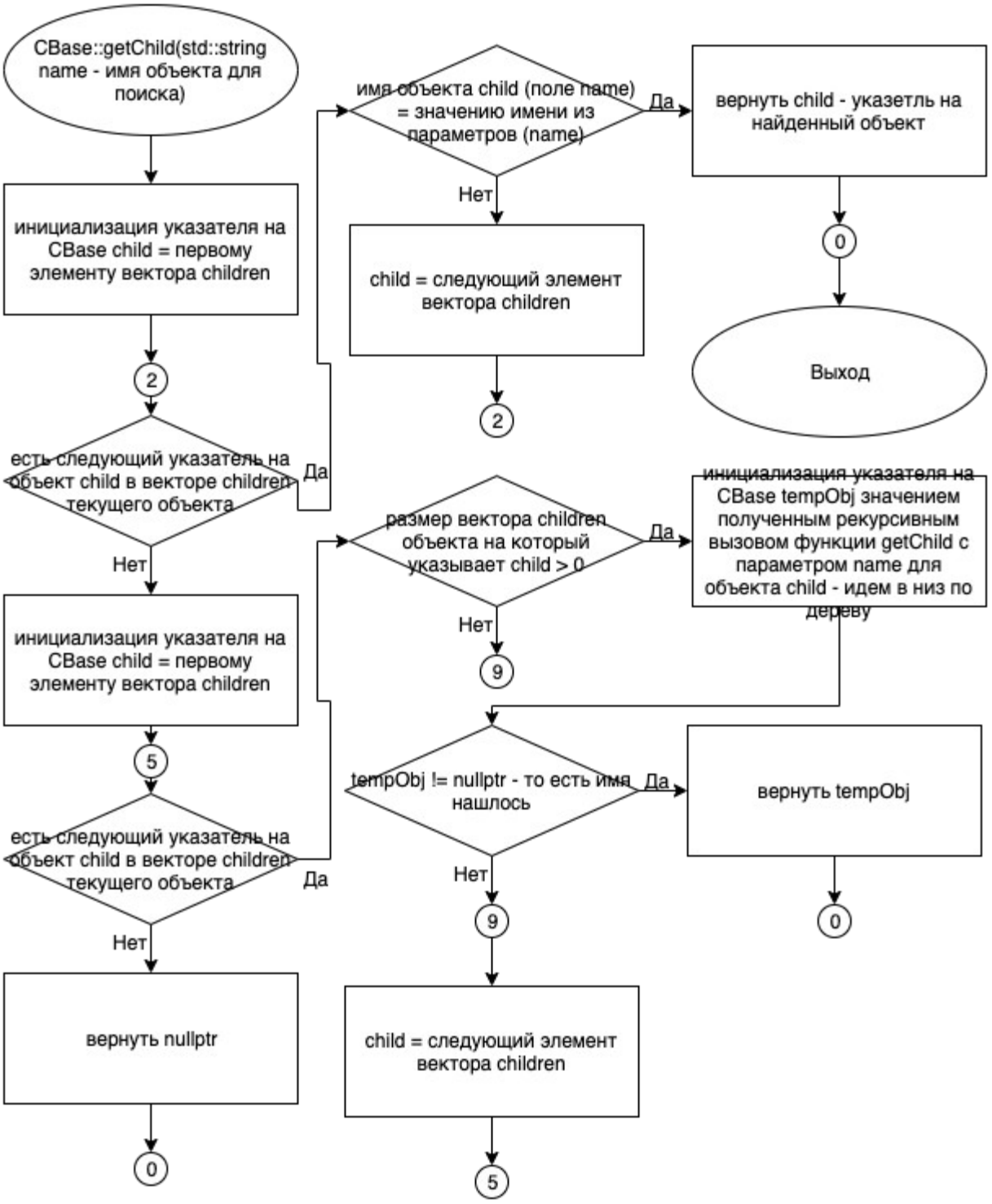

Рисунок 13 – Блок-схема алгоритма

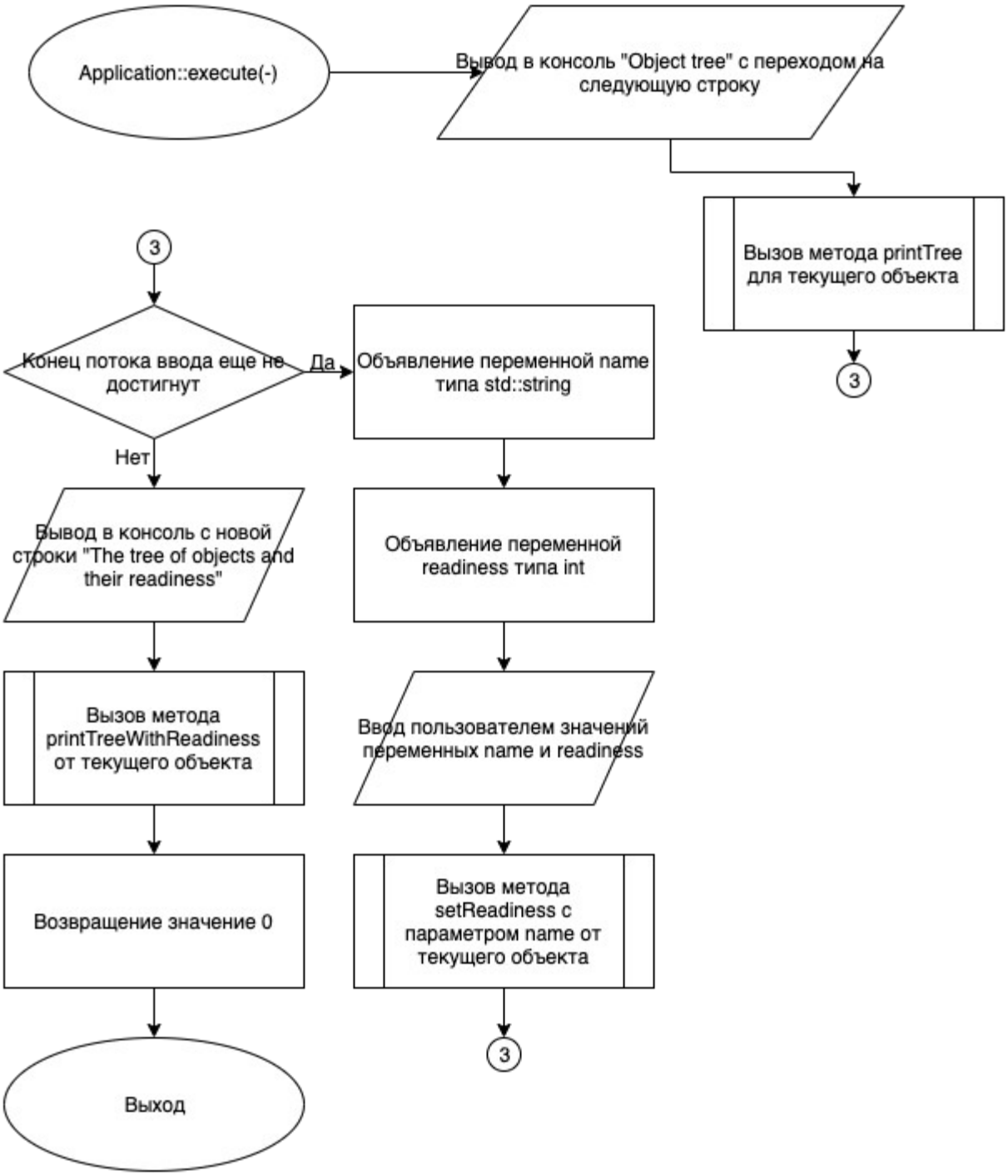

Рисунок 14 – Блок-схема алгоритма

Рисунок 15 – Блок-схема алгоритма

Рисунок 16 – Блок-схема алгоритма

Рисунок 17 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.7 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"

Application::Application(CBase* parent, std::string name): CBase(parent, name){}

void Application::buildTree() {
	std::string parentName, childName;
	int classNum;
	std::cin >> parentName;
	this->setName(parentName);
	while(true) {
		std::cin >> parentName;
		if (parentName == "endtree") return;
		std::cin >> childName >> classNum;
		switch(classNum) {
			case 2:
				new Two(this->findByName(parentName), childName);
				break;
			case 3:
				new Three(this->findByName(parentName), childName);
				break;
			case 4:
				new Four(this->findByName(parentName), childName);
				break;
			case 5:
				new Five(this->findByName(parentName), childName);
				break;
			case 6:
				new Six(this->findByName(parentName), childName);
				break;
			default:
				break;
		}
	}
}

int Application::execute() {
	std::cout << "Object tree" << '\n';
	this->printTree();
	while(!std::cin.eof()) {
		std::string name;
		int readiness;
```

```
            std::cin >> name >> readiness;
            this->findByName(name)->setReadiness(readiness);
      }
      std::cout << '\n' << "The tree of objects and their readiness" << '\n';
      this-> printTreeWithReadiness();
      return 0;
}
```

5.8 Файл application.h

Листинг 2 – application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H

#include <iostream>

#include "cobject.h"
#include "cbase.h"
#include "two.h"
#include "three.h"
#include "four.h"
#include "five.h"
#include "six.h"

class Application: public CBase {
public:
      Application(CBase* parent, std::string name="Application");
      void buildTree();
      int execute();
};

#endif // APPLICATION_H
```

5.9 Файл cbase.cpp

Листинг 3 – cbase.cpp

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

CBase::CBase(CBase*   parent,   std::string   name):   parent(parent),   name(name),
readiness(0) {
      if(parent != nullptr)
            this->parent->children.push_back(this);
}

void CBase::setName(std::string name) {
      this->name = name;
```

```
}

std::string CBase::getName() {
	return this->name;
}

void CBase::setReadiness(int readiness) {
	if(readiness == 0) {
		this->readiness = readiness;
		for(CBase* child : children)
			child->setReadiness(readiness);
	}
	else {
		CBase* root = this;
		while (root->parent != nullptr) {
			if (root->parent->readiness == 0) {
				this->readiness = 0;
				return;
			}
			root = root-> parent;
		}
		this->readiness = readiness;
	}
}

CBase* CBase::findByName(std::string name) {
	if (this->name == name) return this;
	CBase* root = this;
	while(root->parent != nullptr) {
		if(root->parent->name == name) return root->parent;
		root = root->parent;
	}
	return root->getChild(name);
}

CBase* CBase::getChild(std::string name) {
	for(CBase* child : children)
		if(child->name == name) return child;
	for(CBase* child : children) {
		if((child->children).size() > 0) {
			CBase* tempObj = child->getChild(name);
			if (tempObj != nullptr)
				return tempObj;
		}
	}
	return nullptr;
}

CBase* CBase::getRoot() {
	CBase* root = this;
	while (root->parent != nullptr)
		root = root->parent;
	return root;
}


void CBase::setParent(CBase* newParent) {
```

```
	CBase* oldParent = this->parent;
	if (oldParent == nullptr || newParent == nullptr) return;
	std::vector<CBase*> oldPatherChildren = oldParent->children;
	for (size_t i=0; i < oldPatherChildren.size(); ++i)
		if(oldPatherChildren[i] == this) {
			oldPatherChildren.erase(oldPatherChildren.begin()+i);
			break;
		}
	this->parent = newParent;
	parent->children.push_back(this);
}

void CBase::printTree(int level, bool readiness) {
	if (parent == nullptr) {
		std::cout << name;
		std::cout << (readiness ? ((bool)(this->readiness) ? " is ready" : "
is not ready") : "");
	}
	for (CBase* child : children) {
		std::cout << '\n' << std::string(4*level, ' ') << child->name;
		std::cout << (readiness ? ((bool)child->readiness ? " is ready" : " is
not ready") : "");
		child->printTree(level +1, readiness);
	}
}

void CBase::printTreeWithReadiness() {
	this->printTree(1, true);
}

CBase::~CBase() {
	for(size_t i=0; i < children.size(); ++i)
		delete children[i];
}
```

5.10 Файл cbase.h

Листинг 4 – cbase.h

```
#ifndef CBASE_H
#define CBASE_H

#include <string>
#include <vector>

class CBase {
private:
	CBase* parent;
	std::string name;
	int readiness;
	std::vector<CBase*> children;
	CBase* getRoot();
public:
```

```
      CBase(CBase* parent, std::string name="Base");
      void setName(std::string name);
      std::string getName();
      void setReadiness(int readiness);
      void setParent(CBase* parent);
      CBase* getParent();
      CBase* getChild(std::string name);
      CBase* findByName(std::string name);
      void printTree(int level=1, bool readiness=false);
      void printTreeWithReadiness();
      ~CBase();
};

#endif // CBASE_H
```

5.11 Файл cobject.cpp

Листинг 5 – cobject.cpp

```
#include "cobject.h"


Cobject::Cobject(CBase* parent, std::string name): CBase(parent, name){}
```

5.12 Файл cobject.h

Листинг 6 – cobject.h

```
#ifndef COBJECT_H
#define COBJECT_H

#include "cbase.h"

class Cobject: public CBase {
public:
      Cobject(CBase* parent, std::string name);
};

#endif // COBJECT_H
```

5.13 Файл five.cpp

Листинг 7 – five.cpp

```
#include "five.h"
```

```
Five::Five(CBase* parent, std::string name): CBase(parent, name){}
```

5.14 Файл five.h

Листинг 8 – five.h

```
#ifndef FIVE_H
#define FIVE_H

#include "cbase.h"

class Five: public CBase {
public:
      Five(CBase* parent, std::string name);
};

#endif // FIVE_H
```

5.15 Файл four.cpp

Листинг 9 – four.cpp

```
#include "four.h"

Four::Four(CBase* parent, std::string name): CBase(parent, name){}
```

5.16 Файл four.h

Листинг 10 – four.h

```
#ifndef FOUR_H
#define FOUR_H

#include "cbase.h"

class Four: public CBase {
public:
      Four(CBase* parent, std::string name);
};

#endif // FOUR_H
```

5.17 Файл main.cpp

Листинг 11 – main.cpp

```
#include "application.h"


int main() {
      Application root(nullptr);
      root.buildTree();
      return root.execute();
}
```

5.18 Файл six.cpp

Листинг 12 – six.cpp

```
#include "six.h"

Six::Six(CBase* parent, std::string name): CBase(parent, name){}
```

5.19 Файл six.h

Листинг 13 – six.h

```
#ifndef SIX_H
#define SIX_H

#include "cbase.h"

class Six: public CBase {
public:
      Six(CBase* parent, std::string name);
};

#endif // SIX_H
```

5.20 Файл three.cpp

Листинг 14 – three.cpp

```
#include "three.h"
```

```
Three::Three(CBase* parent, std::string name): CBase(parent, name){}
```

5.21 Файл three.h

Листинг 15 – three.h

```
#ifndef THREE_H
#define THREE_H

#include "cbase.h"

class Three: public CBase {
public:
	Three(CBase* parent, std::string name);
};

#endif // THREE_H
```

5.22 Файл two.cpp

Листинг 16 – two.cpp

```
#include "two.h"

Two::Two(CBase* parent, std::string name): CBase(parent, name){};
```

5.23 Файл two.h

Листинг 17 – two.h

```
#ifndef TWO_H
#define TWO_H

#include "cbase.h"

class Two: public CBase {
public:
	Two(CBase* parent, std::string name);
};

#endif // TWO_H
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 17.

Таблица 17 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

R
R Ob_1 3
R Ob_2 2
Ob_2 Ob_4 3
Ob_2 Ob_5 5
Ob_1 Ob_7 2
endtree
R 1
Ob_7 3
Ob_1 1
Ob_2 -2
Ob_4 1
	Object tree
	R
	    Ob_1
	        Ob_7
	    Ob_2
	        Ob_4
	        Ob_5
	The tree of objects and their readiness
	R is ready
	    Ob_1 is ready
	        Ob_7 is not ready
	    Ob_2 is ready
	        Ob_4 is ready
	        Ob_5 is not ready
		Object tree
		R
		    Ob_1
		        Ob_7
		    Ob_2
		        Ob_4
		        Ob_5
		The tree of objects and their readiness
		R is ready
		    Ob_1 is ready
		        Ob_7 is not ready
		    Ob_2 is ready
		        Ob_4 is ready
		        Ob_5 is not ready

R
R A 2
R B 3
R C 4
C D 5
C E 6
B F 6
endtree
R 1
B 4
D 5
F 7
	Object tree
	R
	    A
	    B
	        F
	    C
	        D
	        E
	The tree of objects and their readiness
	R is ready
	    A is not ready
	    B is ready
	        F is ready
	    C is not ready
	        D is not ready
	        E is not ready
		Object tree
		R
		    A
		    B
		        F
		    C
		        D
		        E
		The tree of objects and their readiness
		R is ready
		    A is not ready
		    B is ready
		        F is ready
		    C is not ready
		        D is not ready
		        E is not ready

СПИСОК ИСПОЛЬЗОВАННЫХ 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).