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_1 Вывод иерархического дерева »

С тудент группы	ИКБО-33-21	Евсеев Г.Е.
Руководитель практики	Старший преподаватель	Грач Е.П.
Работа представлена	«___»_____2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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

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
```

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

Метод решения

Для решения задачи используются:

- Объект класса cl_application, cl_base, cl_1, cl_2, cl_3, cl_4, cl_5;
- Класс cl_base:
  - Свойства/поля:
    - Поле, хранящее состояние объекта:
      - Наименование - state;
      - Тип - целого типа;
      - Модификатор доступа - private;
    - Методы:
      - Метод print_new:
        - Функционал - вычисляет длину пути от данной вершины до корня дерева иерархии
      - Метод print_new_state:
        - Функционал - выводит дерево иерархии на экран с отступами, соответствующими расположению объектов в ней, начиная с данной вершины
      - Метод find_object_by_name:
        - Функционал - находи объект по имени среди вершин, низлежащих в дереве иерархии относительно данного объекта или указывает на его отсутствие
      - Метод set_state:
        - Функционал - устанавливает переданное состоянииe объекту
      - Метод get_state:

- Функционал - возвращает состояние текущего объекта

- Класс cl_1, cl_2, cl_3, cl_4, cl_5:
  - Методы:
    - Метод cl_i:
      - Функционал - параметризированный конструктор с параметрами имени объекта и указателя на его родителя;

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	cl_base	cl_application	public	Базовый класс в иерархии классов, содержит основные методы	2	
		cl_1	public		3	
		cl_2	public		4	
		cl_3	public		5	
		cl_4	public		6	
		cl_5	public		7	
2	cl_application			Класс приложение, содержащий метод запуска программы и построения дерева		
3	cl_1			Класс первого объекта		
4	cl_2			Класс второго объекта		
5	cl_3			Класс третьего		

				объекта		
6	cl_4			Класс четвертого объекта		
7	cl_5			Класс пятого объекта		

Описание алгоритма

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

Конструктор класса: cl_base_cl_1_cl_2_cl_3_cl_4_cl_5

Модификатор доступа: public

Функционал: конструктор класса

Параметры: строка name, указатель cl_base* parent

Алгоритм конструктора представлен в таблице 2.

Таблица 2. Алгоритм конструктора класса cl_base_cl_1_cl_2_cl_3_cl_4_cl_5

№	Предикат	Действия	№ перехода	Комментарий
1		присваивание полю name текущего объекта значение name	2	
2		присваивание полю parent текущего объекта значение parent	3	
3	parent != пустой указатель	добавление в объект parent в поле children текущего объекта	∅	
			∅	

Конструктор класса: cl_application

Модификатор доступа: public

Функционал: конструктор класса

Параметры: строка name

Алгоритм конструктора представлен в таблице 3.

Таблица 3. Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		вызов конструктора cl_base(name)	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: print_new

Функционал: вывод дерева объектов

Параметры: строка depth

Возвращаемое значение: пустого типа

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода print_new класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		вывод: переход на новую строку + depth + get_name()	2	
2	перебор детей в child	вызов метода print_new(depth + " ") для объекта child	2	
			∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: print_new_state

Функционал: вывод состояний ветвей дерева объектов

Параметры: строка depth

Возвращаемое значение: пустого типа

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода print_new_state класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		вывод: переход на новую строку + depth + get_name()	2	
2	state == 0	вывод " is not ready"	3	
		вывод " is ready"	∅	
3	перебор детей в child	вызов метода print_new_state(depth + " ") для объекта child	3	
			∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: find_object_by_name

Функционал: ищет объект по заданному имени

Параметры: строка name

Возвращаемое значение: указатель на объект класса cl_base

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода find_object_by_name класса cl_base

№	Предикат	Действия	№ перехода	Комментарий

1	name текущего объекта == name	возвращает указатель на этот объект	∅	
	перебор всех детей (child) из children	инициализация указателя на объект result класса cl_base с помощью метода find_object_by_name(name) от объекта child	2	
			3	
2	result != пустой указатель	возвращает result	1	
			1	
3		возвращает пустой указатель	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_state

Функционал: возвращает значение состояния

Параметры: отсутствуют

Возвращаемое значение: целое

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода get_state класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		возвращает состояние этого объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_state

Функционал: устанавливает состояние

Параметры: целое state

Возвращаемое значение: логическое

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода set_state класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1	state == 0		2	
			6	
2	parent != пустой указатель and parent->get_state() == 0	возвращает true	∅	
			3	
3	перебор детей (child) в children	child->set_state(0)	3	
			4	
4		state текущего объекта = 0	5	
5		возвращает true	∅	
6	parent != пустой указатель and parent->get_state() == 0	возвращает false	∅	
		присваивание state текущего объекта = state	7	
7		возвращает true	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: build_tree_objects

Функционал: строит дерево объектов

Параметры: отсутствуют

Возвращаемое значение: пустого типа

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		объявление строковых parent_name, child_name	2	
2		объявление указателей на cl_base parent_ptr, child_ptr	3	
3		ввод parent name	4	
4		set_name(parent_name)	5	
5		объявление целого class_number	6	
6	истина	ввод parent_name	7	
			∅	
7	parent_name =="endtree"		∅	
			8	
8		ввод child name и class_number	9	
9		присваивание parent_ptr = find_object_by_name(parent_name)	10	
10	class_number == 2	присваивание child_ptr = new cl_1(child_name, parent_ptr)	6	
	class_number == 3	присваивание child_ptr = new cl_2(child_name, parent_ptr)	6	
	class_number	присваивание child_ptr =	6	

	== 4	new cl_3(child_name, parent_ptr)		
	class_number == 5	присваивание child_ptr = new cl_4(child_name, parent_ptr)	6	
	class_number == 6	присваивание child_ptr = new cl_5(child_name, parent_ptr)	6	

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app

Функционал: запуск программы

Параметры: отсутствуют

Возвращаемое значение: целого типа

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		вывод "Object tree"	2	
2		вызов метода print_new("")	3	
3		объявление строки object_name и целого new_state	4	
4	ввод object_name >> new_state	инициализация указателя на объект cl_base cl = find_object_by_name(object_name)	5	
			6	
5		вызов метода cl-	4	

		>set_state(new_state)		
6		вывод: переход на новую строку + "The tree of objects and their readiness"	7	
7		вызов метода print_new_state("")	∅	

Функция: main

Функционал: основная функция

Параметры: отсутствуют

Возвращаемое значение: целого типа

Алгоритм функции представлен в таблице 11.

Таблица 11. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		объявление объекта application класса cl_application	2	
2		вызов метода application.build_tree_objects()	3	
3		возвращение значения метода application.exec_app()	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

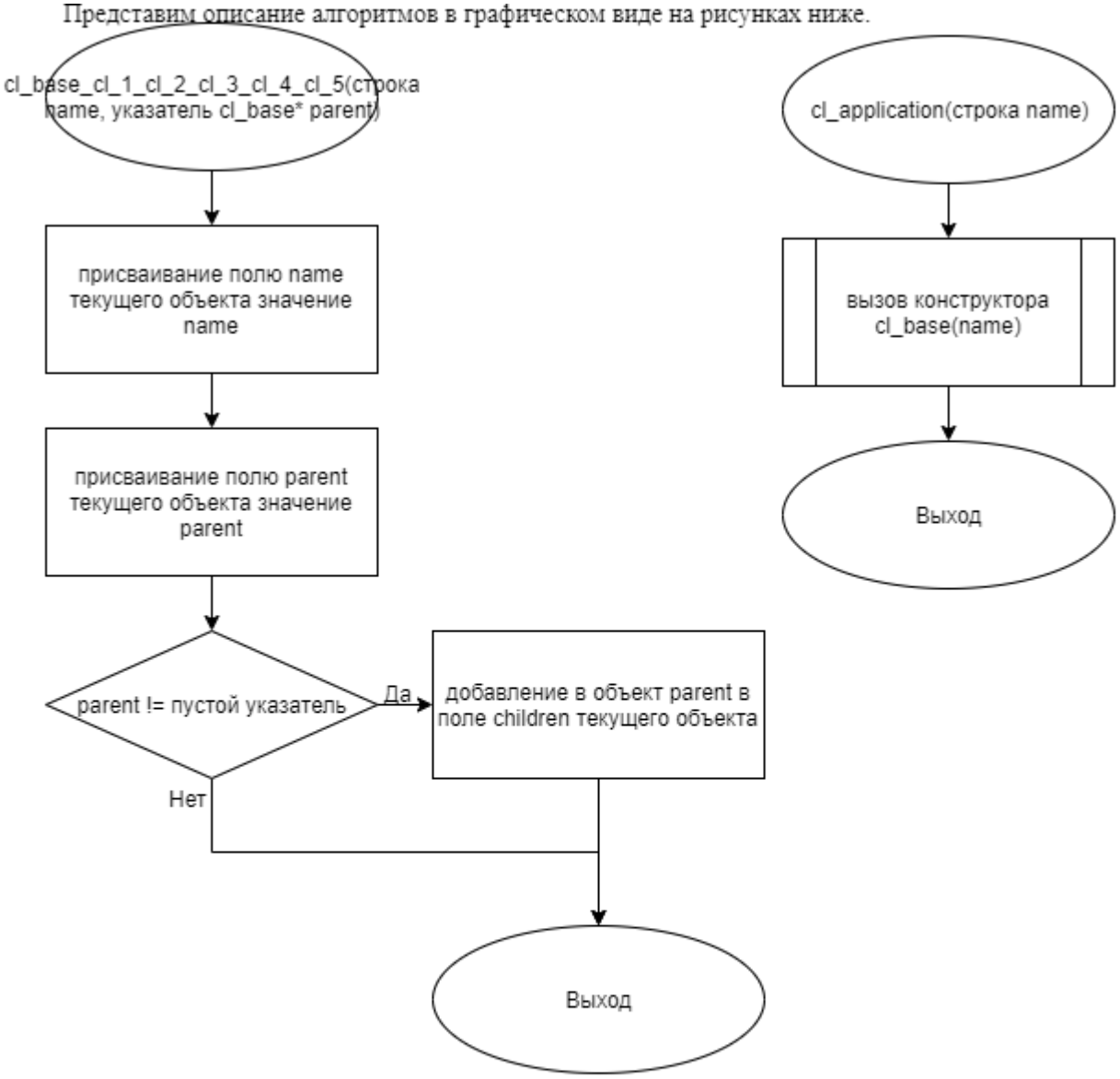

Рис. 1. Блок-схема алгоритма.

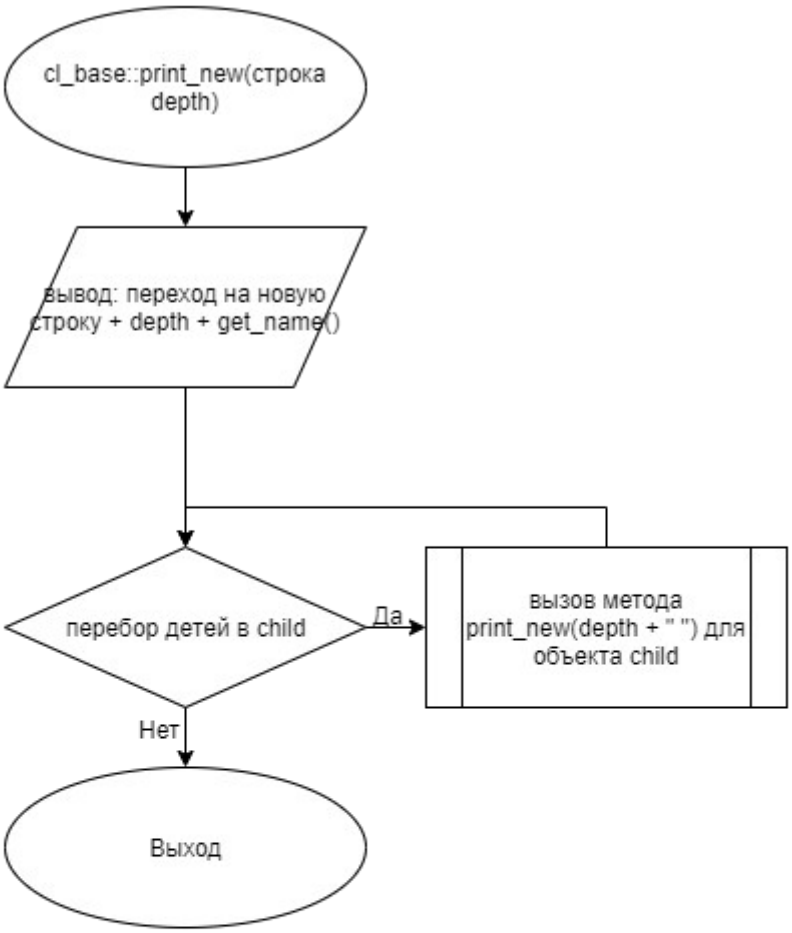

Рис. 2. Блок-схема алгоритма.

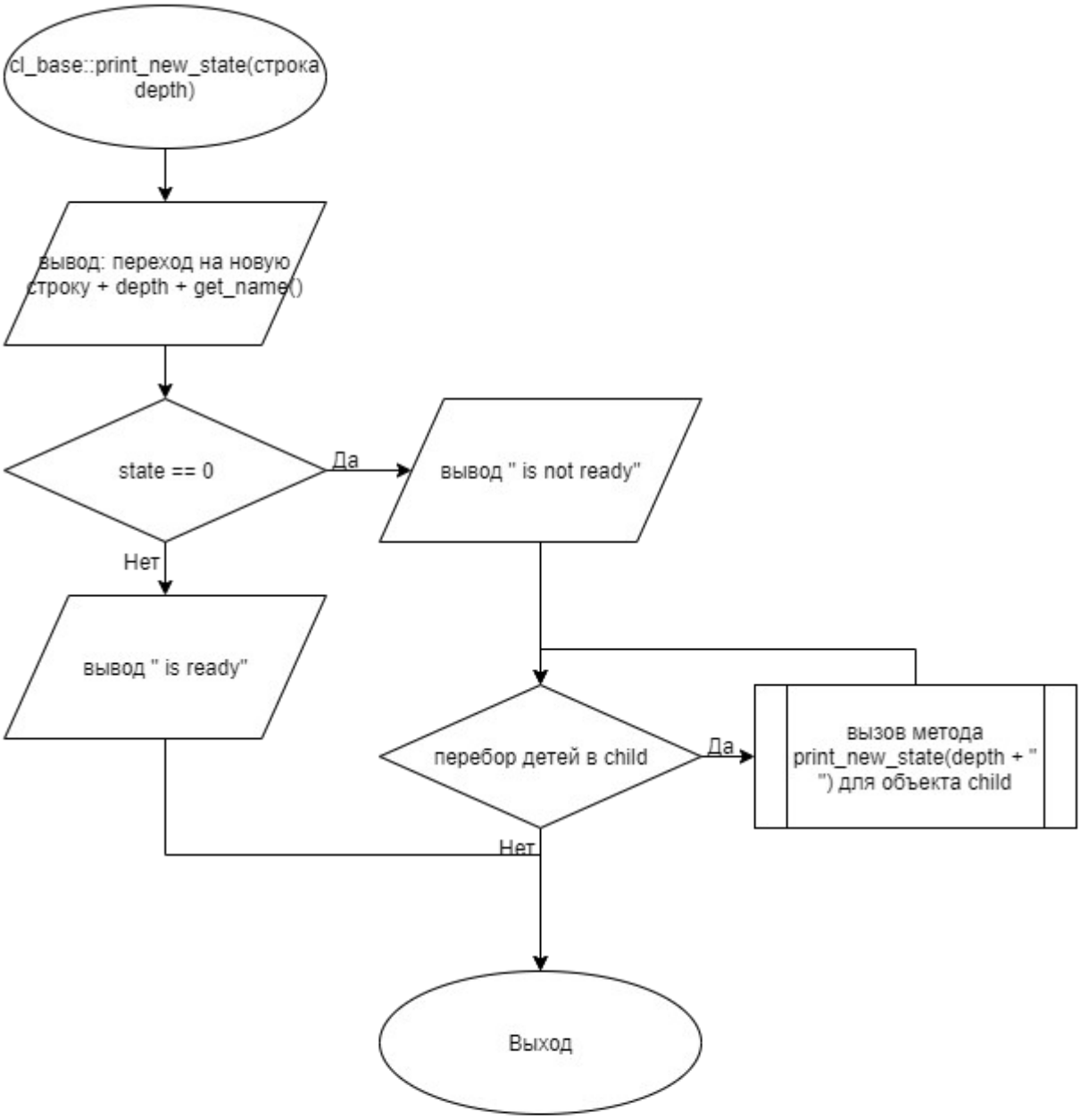

Рис. 3. Блок-схема алгоритма.

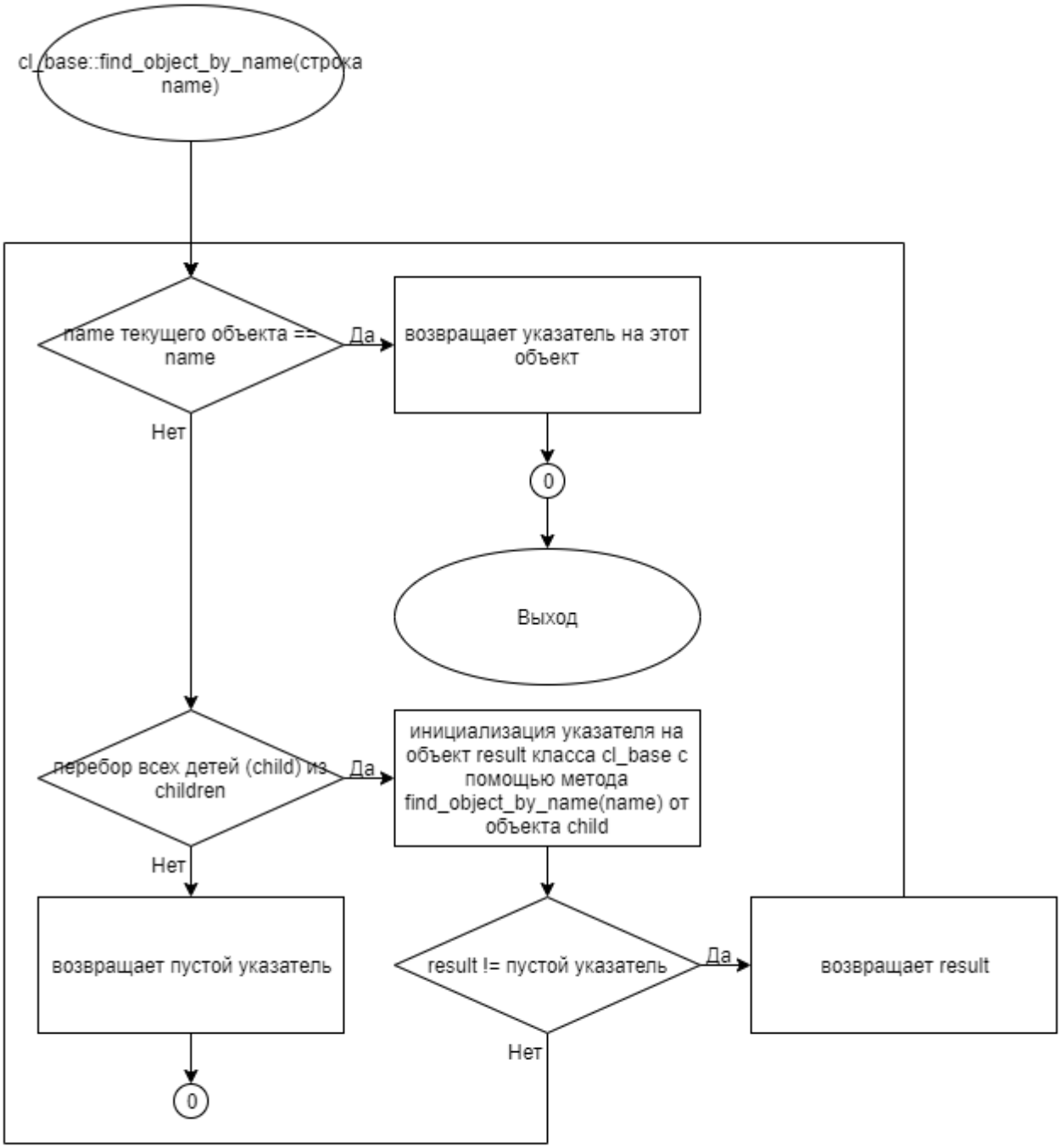

Рис. 4. Блок-схема алгоритма.

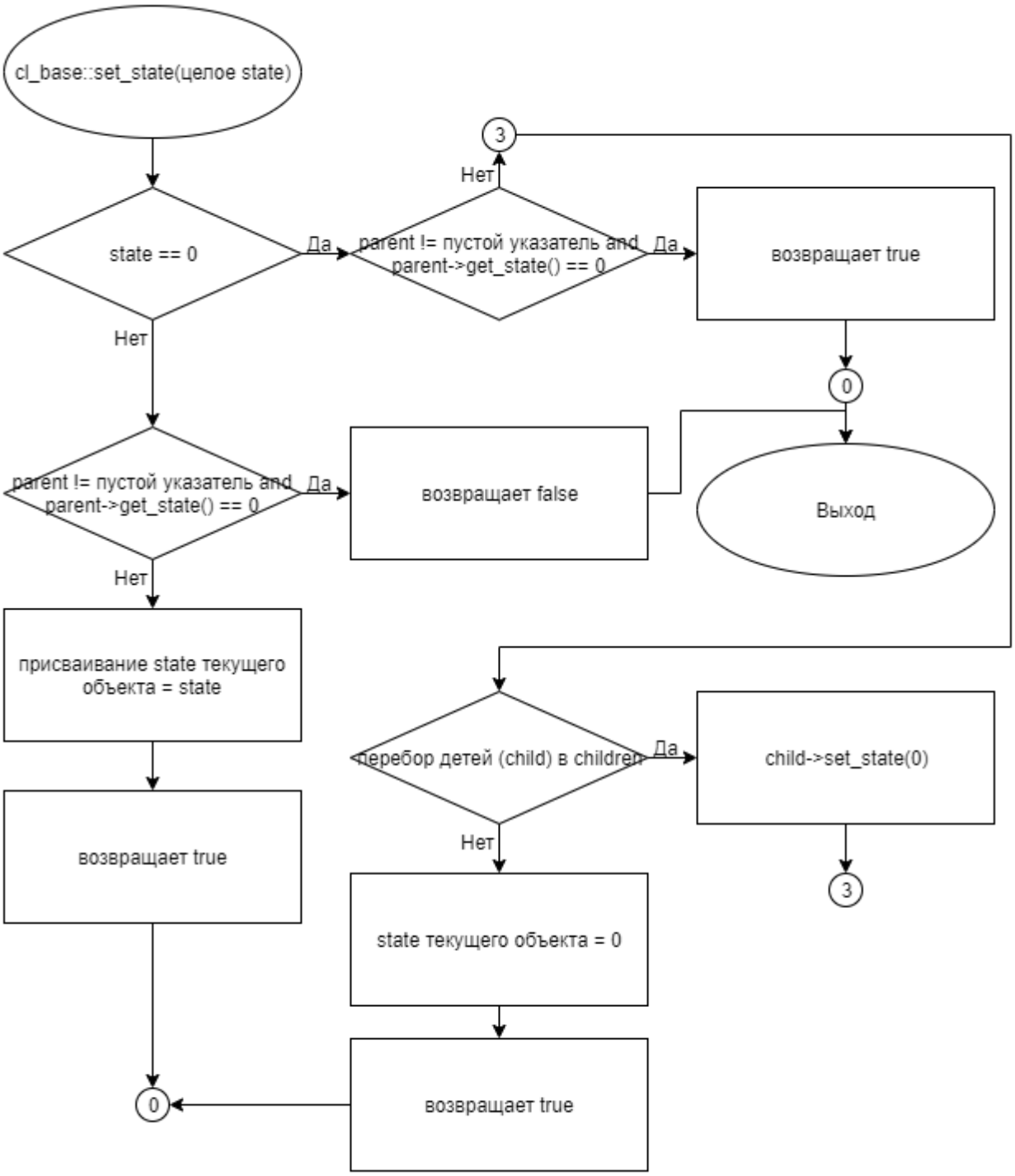

Рис. 5. Блок-схема алгоритма.

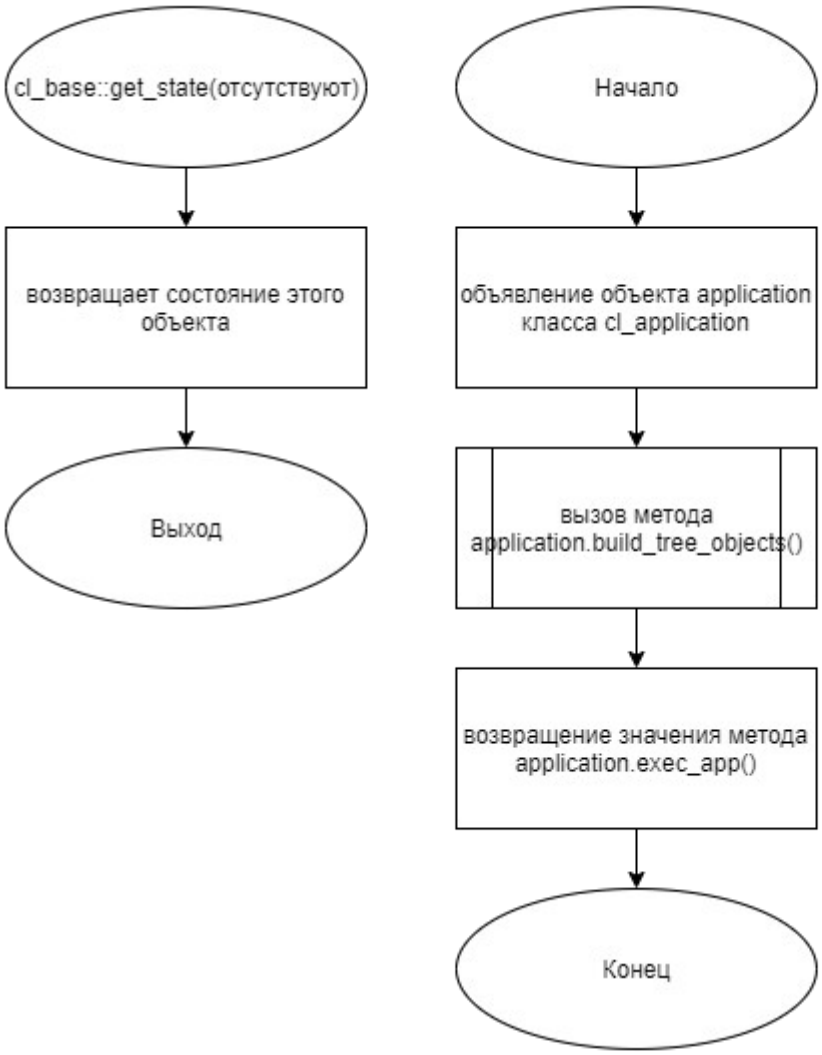

Рис. 6. Блок-схема алгоритма.

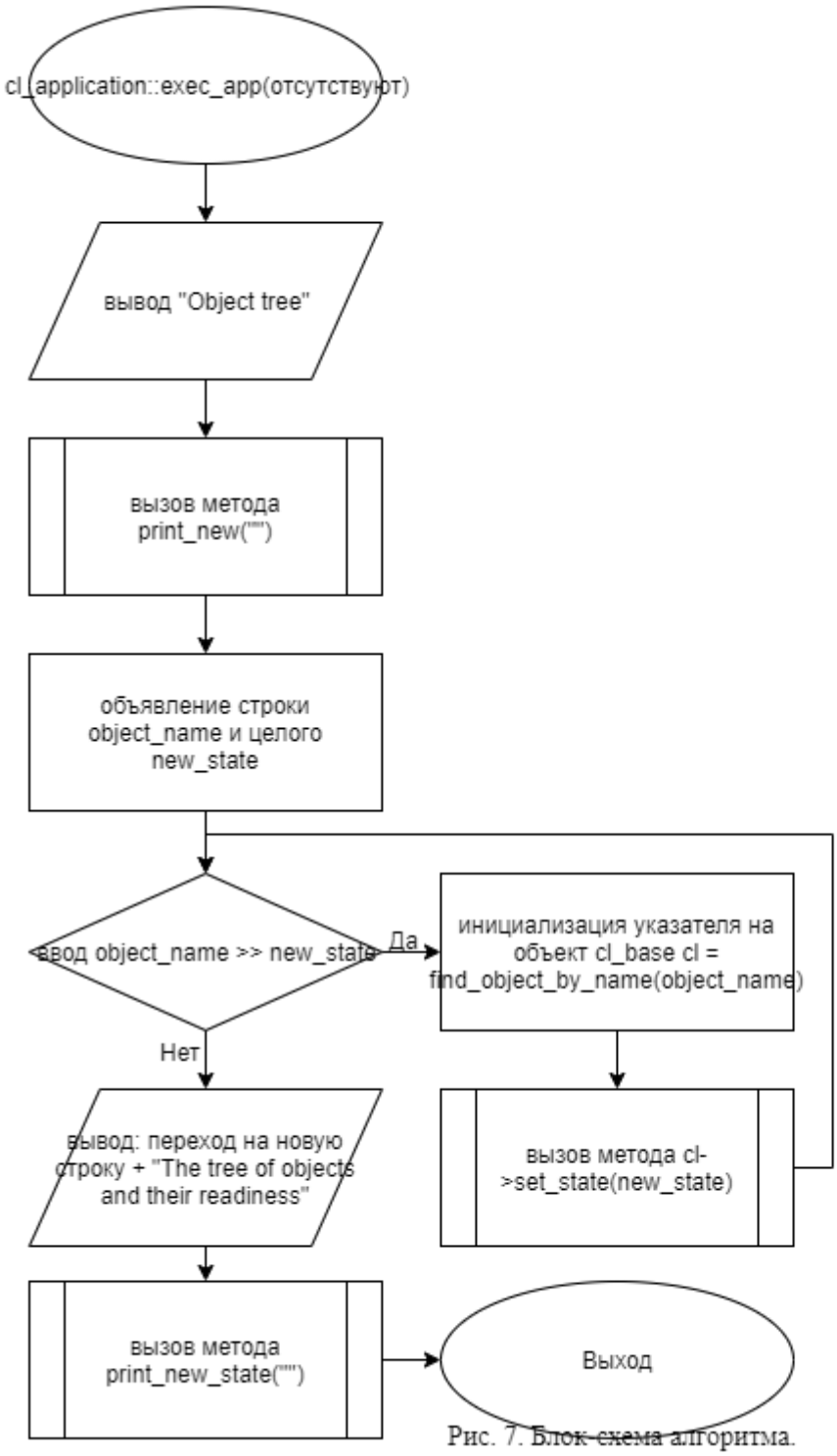

Рис. 7. Блок-схема алгоритма.

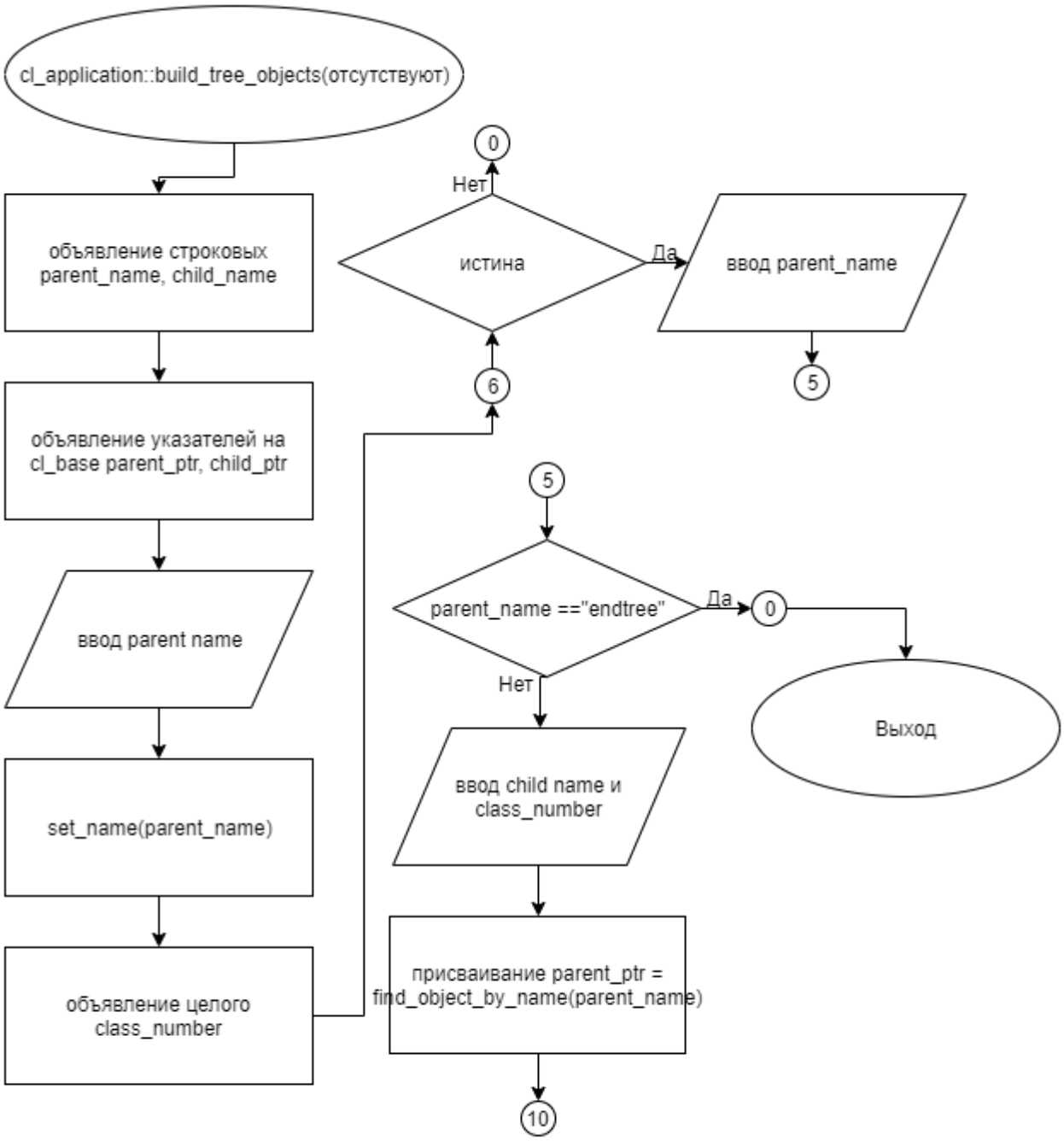

Рис. 8. Блок-схема алгоритма.

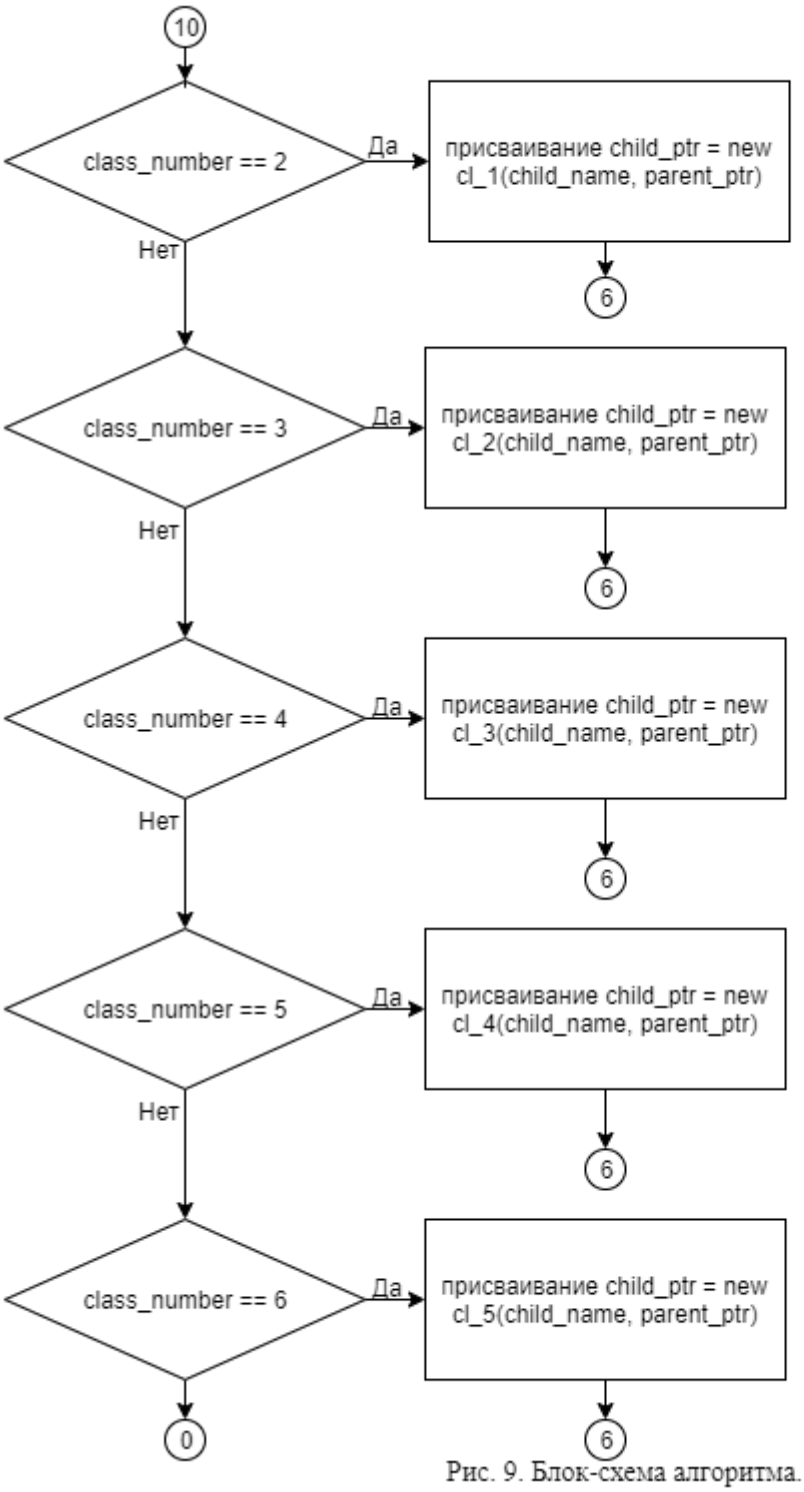

Рис. 9. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл cl_1.h

```
#ifndef __CL_1_H__
#define __CL_1_H__
#include "cl_base.h"
class cl_1 : public cl_base
{
public:
        cl_1(string name, cl_base* parent) : cl_base(name, parent) {};
};
#endif
```

Файл cl_2.h

```
#ifndef __CL_2_H__
#define __CL_2_H__
#include "cl_base.h"
class cl_2: public cl_base
{
public:
        cl_2(string name, cl_base* parent): cl_base(name, parent) {};
};
#endif
```

Файл cl_3.h

```
#ifndef __CL_3_H__
#define __CL_3_H__
#include "cl_base.h"
class cl_3: public cl_base
{
public:
        cl_3(string name, cl_base* parent): cl_base(name, parent) {};
};
#endif
```

Файл cl_4.h

```
#ifndef __CL_4_H__
#define __CL_4_H__
#include "cl_base.h"
class cl_4: public cl_base
{
public:
        cl_4(string name, cl_base* parent): cl_base(name, parent) {};
};
#endif
```

Файл cl_5.h

```
#ifndef __CL_5_H__
#define __CL_5_H__
#include "cl_base.h"
class cl_5: public cl_base
{
public:
        cl_5(string name, cl_base* parent): cl_base(name, parent) {};
};
#endif
```

Файл cl_application.cpp

```
#include "cl_application.h"
void cl_application::build_tree_objects()
{
        string parent_name, child_name;
        cl_base* parent_ptr, * child_ptr;
        cin >> parent_name;
        set_name(parent_name);
        int class_number;
        while (true)
        {
                cin >> parent_name;
                if (parent_name == "endtree")
                        break;
                cin >> child_name >> class_number;
                parent_ptr = find_object_by_name(parent_name);
                switch (class_number)
                {
                case(2):
                        child_ptr = new cl_1(child_name, parent_ptr);
                        break;
                case(3):
                        child_ptr = new cl_2(child_name, parent_ptr);
                        break;
                case(4):
                        child_ptr = new cl_3(child_name, parent_ptr);
```

```
                                break;
                case(5):
                        child_ptr = new cl_4(child_name, parent_ptr);
                        break;
                case(6):
                        child_ptr = new cl_5(child_name, parent_ptr);
                        break;
                }
        }
}
int cl_application::exec_app()
{
        cout << "Object tree";
        print_new("");
        string object_name;
        int new_state;
        while (cin >> object_name >> new_state)
        {
                cl_base* cl = find_object_by_name(object_name);
                cl->set_state(new_state);
        }
        cout << endl << "The tree of objects and their readiness";
        print_new_state("");
        return 0;
}
```

Файл cl_application.h

```
#ifndef __CL_APPLICATION_H__
#define __CL_APPLICATION_H__
#include "cl_base.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
class cl_application : public cl_base
{
public:
        cl_application(string name = "") : cl_base(name) {};
        void build_tree_objects();
        int exec_app();
};
#endif
```

Файл cl_base.cpp

```
#include "cl_base.h"
cl_base::cl_base(string name, cl_base* parent)
{
```

```
        this->name = name;
                this->parent = parent;
        if (parent)
                parent->children.push_back(this);
        return;
}
cl_base::~cl_base()
{
        for (int i = 0; i < children.size(); i++)
                delete children[i];
}
string cl_base::get_name()
{
        return name;
}
void cl_base::set_name(string& name)
{
        this->name = name;
        return;
}
void cl_base::set_parent(cl_base* new_parent)
{
        if (parent)
                for (int i = 0; i < parent->children.size(); i++)
                        if (parent->children[i] == this)
                                parent->children.erase(parent ->
children.begin() + i);
        parent = new_parent;
        if (new_parent)
                new_parent->children.push_back(this);
        return;
}
cl_base* cl_base::get_parent()
{
        return this->parent;
}
void cl_base::print()
{
        if (children.size())
        {
                cout << endl << get_name();
                for (int i = 0; i < children.size(); i++)
                        cout << " " << children[i]->get_name();
                for (int i = 0; i < children.size(); i++)
                        children[i]->print();
        }
        return;
}
void cl_base::print_new(string depth)
{
        cout << endl << depth << get_name();
        for (auto child : children)
                child->print_new(depth + "    ");
}
void cl_base::print_new_state(string depth)
{
        cout << endl << depth << get_name();
```

```
        if (!state)
                cout << " is not ready";
        else
                cout << " is ready";
        for (cl_base* child : children)
                child->print_new_state(depth + "    ");
}
cl_base* cl_base::find_object_by_name(string name)
{
        if (this->name == name)
                return this;
        else
        {
                for (cl_base* child : children)
                {
                        cl_base* result = child->find_object_by_name(name);
                        if (result)
                                return result;
                }
                return nullptr;
        }
}
int cl_base::get_state()
{
        return this->state;
}
bool cl_base::set_state(int state)
{
        if (!state)
                if ((parent && parent->get_state() == 0) || get_state() == 0)
                        return true;
                else
                {
                        for (cl_base* child : children)
                                child->set_state(0);
                        this->state = 0;
                        return true;
                }
        else
                if (parent && parent->get_state() == 0)
                        return false;
                else
                {
                        this->state = state;
                        return true;
                }
}
```

Файл cl_base.h

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

```
class cl_base
{
        int state = 0;
        string name;
        cl_base* parent;
        vector<cl_base*> children;
public:
        cl_base(string name = "", cl_base* parent = nullptr);
        ~cl_base();
        string get_name();
        void set_name(string& name);
        cl_base* get_parent();
        void set_parent(cl_base* new_parent);
        void print();
        void print_new(string);
        void print_new_state(string);
        virtual cl_base* find_object_by_name(string name);
        bool set_state(int state);
        int get_state();
};
#endif
```

Файл main.cpp

```
#include "cl_application.h"
int main()
{
        cl_application application;
        application.build_tree_objects();
        return application.exec_app();
}
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
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	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	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
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	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	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
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 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 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_01 1 object_02 -2 object_04 1	object_07 is not ready object_02 is ready object_04 is ready object_05 is not ready	object_07 is not ready object_02 is ready object_04 is ready object_05 is not ready

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).