Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. 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).

Поиск головного объекта выполняется от последнего созданного объекта. Первоначально последним созданным объектом считается корневой объект. Если для головного объекта обнаруживается дубляж имени в непосредственно подчиненных объектах, то объект не создается. Если обнаруживается дубляж имени на дереве иерархии объектов, то объект не создается. Если номер класса объекта задан некорректно, то объект не создается.

Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем 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 ветке дереве иерархии от текущего по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на искомой ветке дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
- метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;
- метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

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»

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 задачи необходимы те же объекты, циклы и операторы, которые были необходимы для решения работы КВ1. Также требуются классы cl_2, cl_3, cl_4, cl_5, cl_6.

Класс cl_base базовый класс (с изменениями)

Поля:

скрытые элементы:

bool state;

методы:

открытые:

void show_object_tree(bool show_state=false) - устанавливает состояние объекта, проверяя и корректируя состояния его родителей и детей

void show_object_next(int i_level, bool show_state) - вывод иерархию классов от текущего объекта с или без их состояния

bool get_state() - возвращает состояние объекта

void set_state(bool state) - устанавливает состояние объекта, проверяя и корректируя состояния его родителей и детей

cl_base* find(string& object_name) - возвращает указатель на найденный на ветке объект

cl_base* find_on_whole_tree(string& object_name) - возвращает указатель на найденный на дереве объект

Класс cl_application наследует класс cl_base (с изменениями)

Поля:

методы:

void build_tree_objects() - строит иерархию классов

void exec_app() - вывод иерархию классов с корневого объекта, сначала бех, а потом с состояними

Классы cl_2, cl_3, cl_4, cl_5, cl_6 унаследованы от cl_base

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 build_tree_objects класса cl_application

Функционал: строение иерархии классов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		инициализация строковых переменных root_name, child_name, а также инициализация целочисленных переменных class_num, obj_state	2
2		ввод root_name	3
3		создание динамического массива created, который хранит в себе указатели на объекты класса cl_base, в котором находится указатель на корневой объект	4
4		изменение имени корневого объекта на root_name	5
5		ввод root_name	6
6	root_name равно "endtree"		9
		ввод child_name и class_num	7
7		вызов метода поиска по всему дереву объекта и запись результата в переменную p	8
8	объект root_name не найден на дереве или у объекта		5

№	Предикат	Действия	№ перехода
	root_name есть ребенок с именем child_name		
	class_num == 2	создание нового объекта класса cl_2 с child_name и p	5
	class_num == 3	создание нового объекта класса cl_3 с child_name и p	5
	class_num == 4	создание нового объекта класса cl_4 с child_name и p	5
	class_num == 5	создание нового объекта класса cl_5 с child_name и p	5
	class_num == 6	создание нового объекта класса cl_6 с child_name и p	5
			5
9	root_name и obj_state введены	вызов метода поиска по всему дереву объекта с именем root_name	10
			∅
10	Объект root_name найден	вызов метода установки состояния obj_state у объекта с именем root_name	9
			9

3.2 Алгоритм метода exec_app класса cl_application

Функционал: вывод иерархии классов с корневого объекта.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2
2		вызов метода вывода дерева объектов без состояний	3
3		вывод "Tre tree of objects and their readiness	4
4		вызов метода вывода дерева объектов с состояниями	∅

3.3 Алгоритм метода get_state класса cl_base

Функционал: возвращение состояние объекта.

Параметры: нет.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода get_state класса cl_base

№	Предикат	Действия	№ перехода
1		возврат состояния объекта	∅

3.4 Алгоритм метода set_state класса cl_base

Функционал: устанавливает состояние объекта, проверяя и корректируя состояния его родителей и детей.

Параметры: bool state.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_state класса cl_base

№	Предикат	Действия	№ перехода
1	у объекта есть родитель и состояние родителя равно	установить состояние текущего объекта в false	∅

№	Предикат	Действия	№ перехода
	false		
		установить состояние текущего объекта в значение параметра state	2
2	состояние равно false		3
			∅
3	с принадлежит children	вызов метода изменения состояния в state у с	3
			∅

3.5 Алгоритм метода find класса cl_base

Функционал: возвращает указатель на найденный на ветке объект.

Параметры: string name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода find класса cl_base

№	Предикат	Действия	№ перехода
1	параметр object_name == имени текущего объекта	возврат указателя на текущий объект	∅
			2
2	с принадлежит children	вызов метода поиска объекта по имени object_name у объекта с	3
		возврат nullptr	∅
3	объект object_name найден	возврат указателя на найденный объект	∅
			2

3.6 Алгоритм метода find_on_whole_tree класса cl_base

Функционал: возвращает указатель на найденный на дереве объект.

Параметры: string object_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода find_on_whole_tree класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация переменной p равной this	2
2	p есть родитель	установить p равной ее родителю	2
		вызов метода поиска объекта по имени object_name от объекта p и возврат результата	∅

3.7 Алгоритм метода show_object_tree класса cl_base

Функционал: вывод иерархии классов от текущего объекта с или без их состояния.

Параметры: bool show_state.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода show_object_tree класса cl_base

№	Предикат	Действия	№ перехода
1		вызов метода show_object_next(0, show_state)	∅

3.8 Алгоритм метода show_object_next класса cl_base

Функционал: вывод иерархию классов от текущего объекта с или без их состояния.

Параметры: int i_level, bool show_state.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода show_object_next класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация пустой строковой переменной s_space	2
2	i_level > 0	заполнение переменной s_space пробелами в количестве равном i_level * 4	3
			3
3		вывод s_space и имени текущего объекта	4
4	show_state == true	вывод состояния объекта	5
			5
5	c принадлежит children	вывод символа переноса строки	6
			∅
6		вызов метода show_object_next с параметрами i_level + 1 и show_state	5

3.9 Алгоритм функции main

Функционал: главный метод программы.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 9.

Таблица 9 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта класса	2
2		вызов метода build_tree_objects	3
3		вызов метода exec_app	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-8.

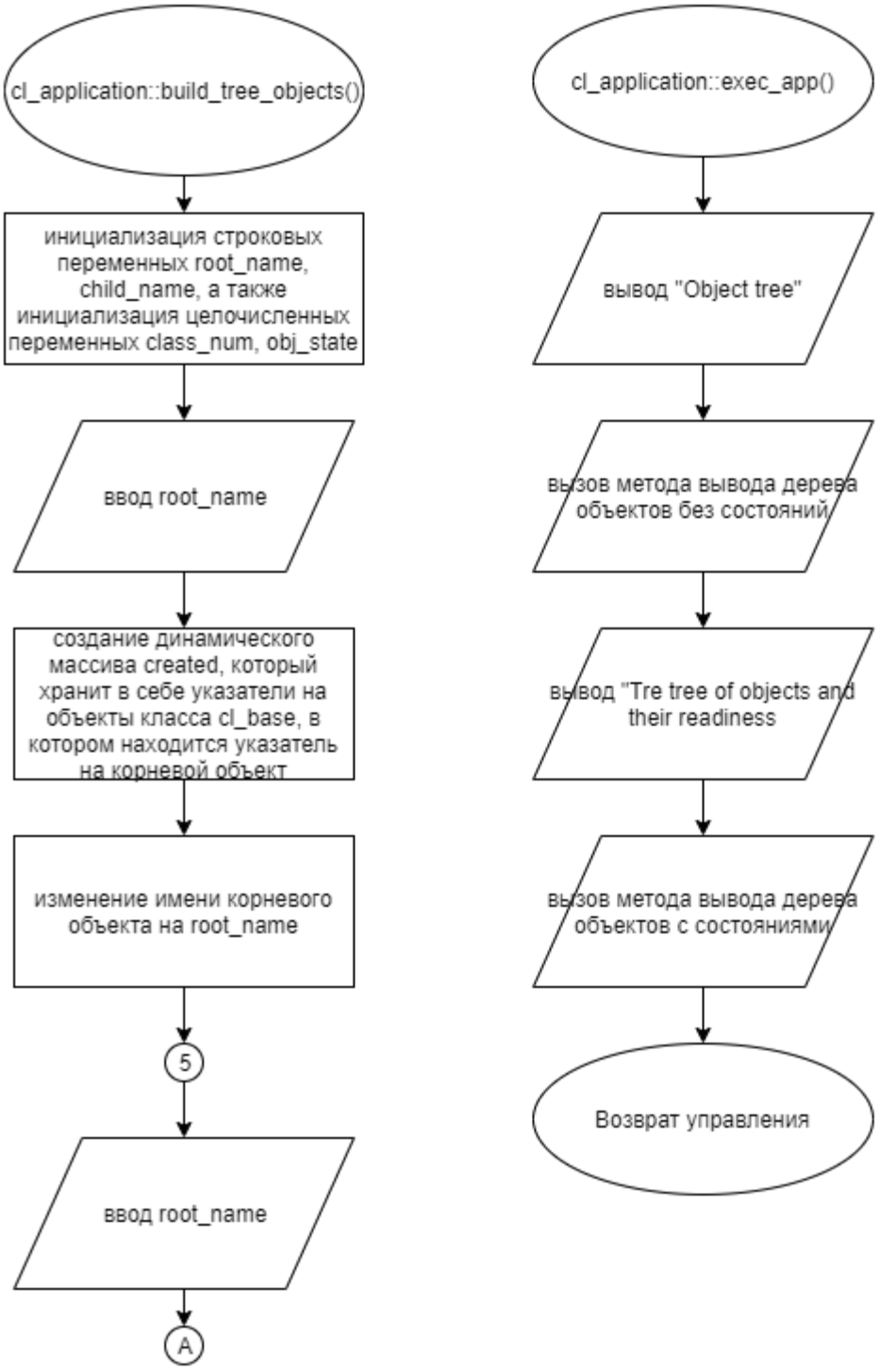

Рисунок 1 – Блок-схема алгоритма

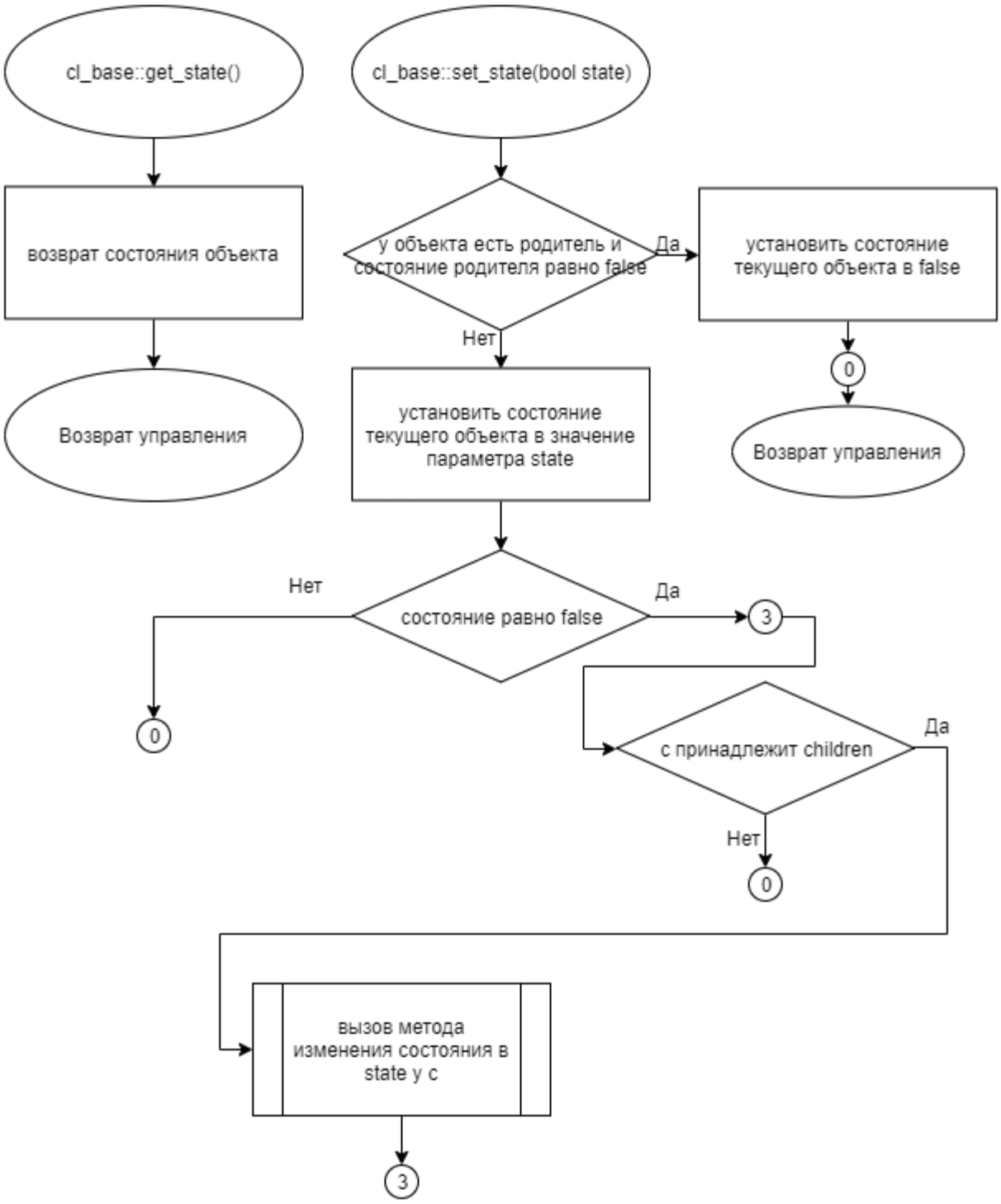

Рисунок 2 – Блок-схема алгоритма

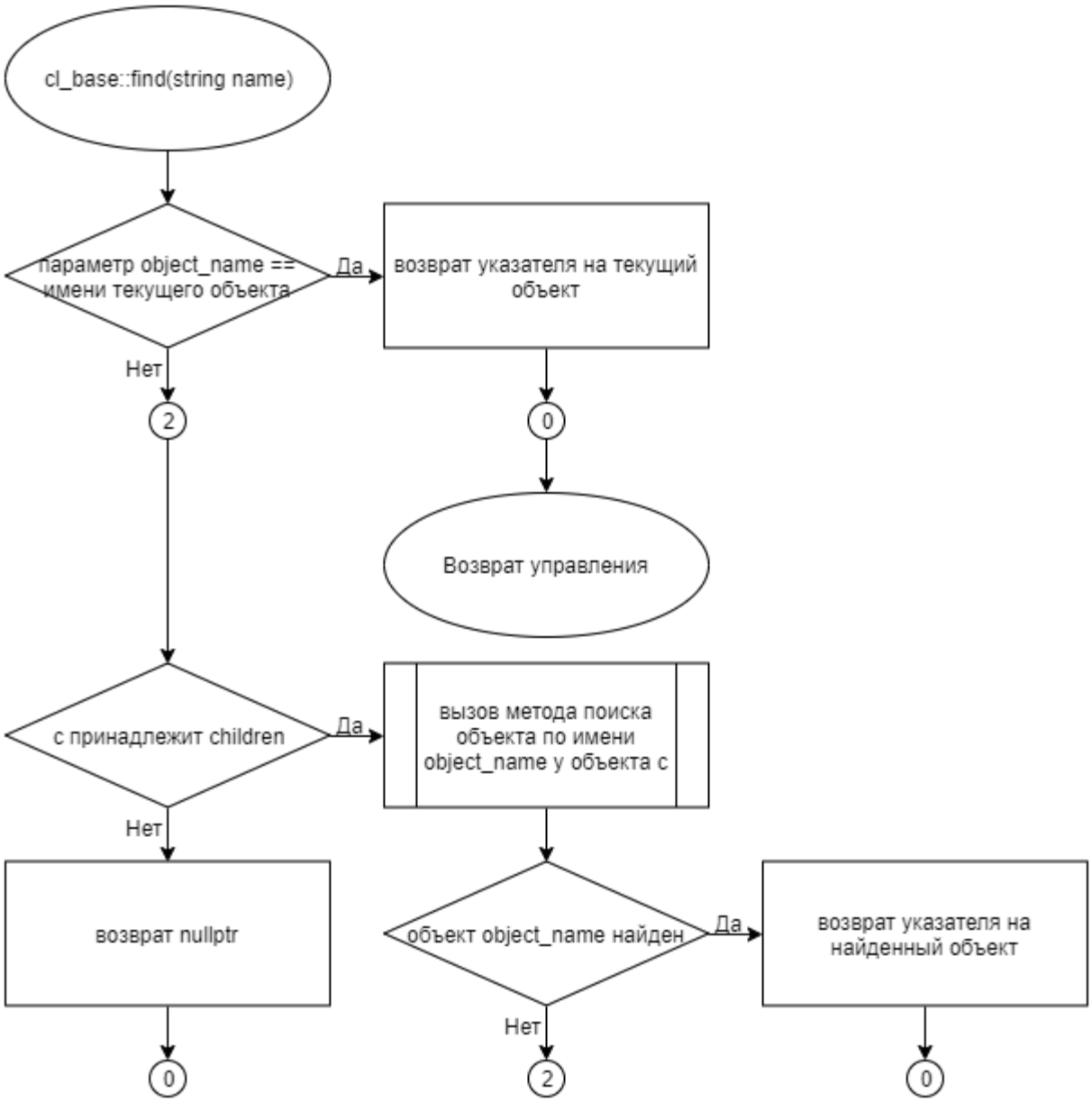

Рисунок 3 – Блок-схема алгоритма

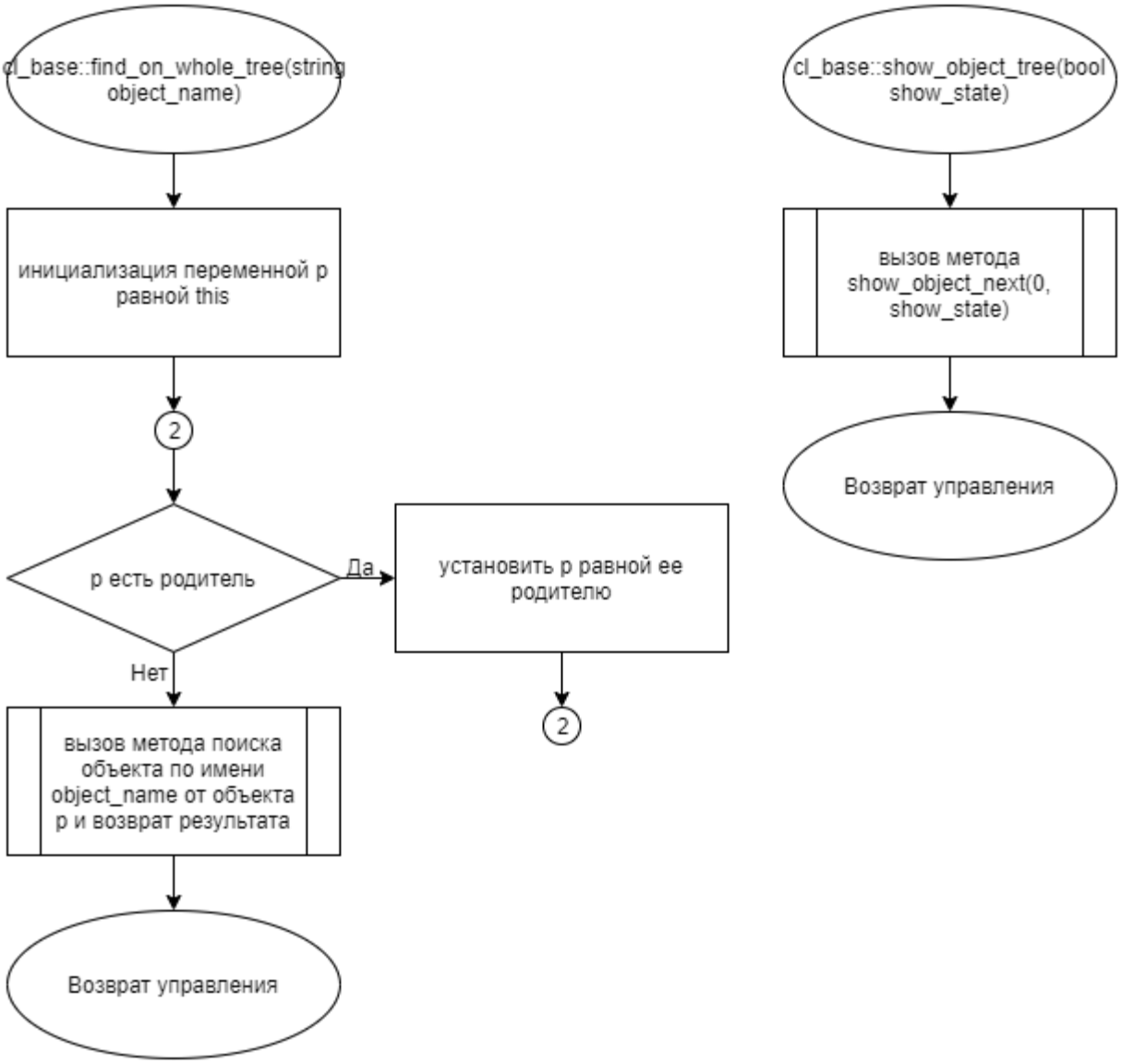

Рисунок 4 – Блок-схема алгоритма

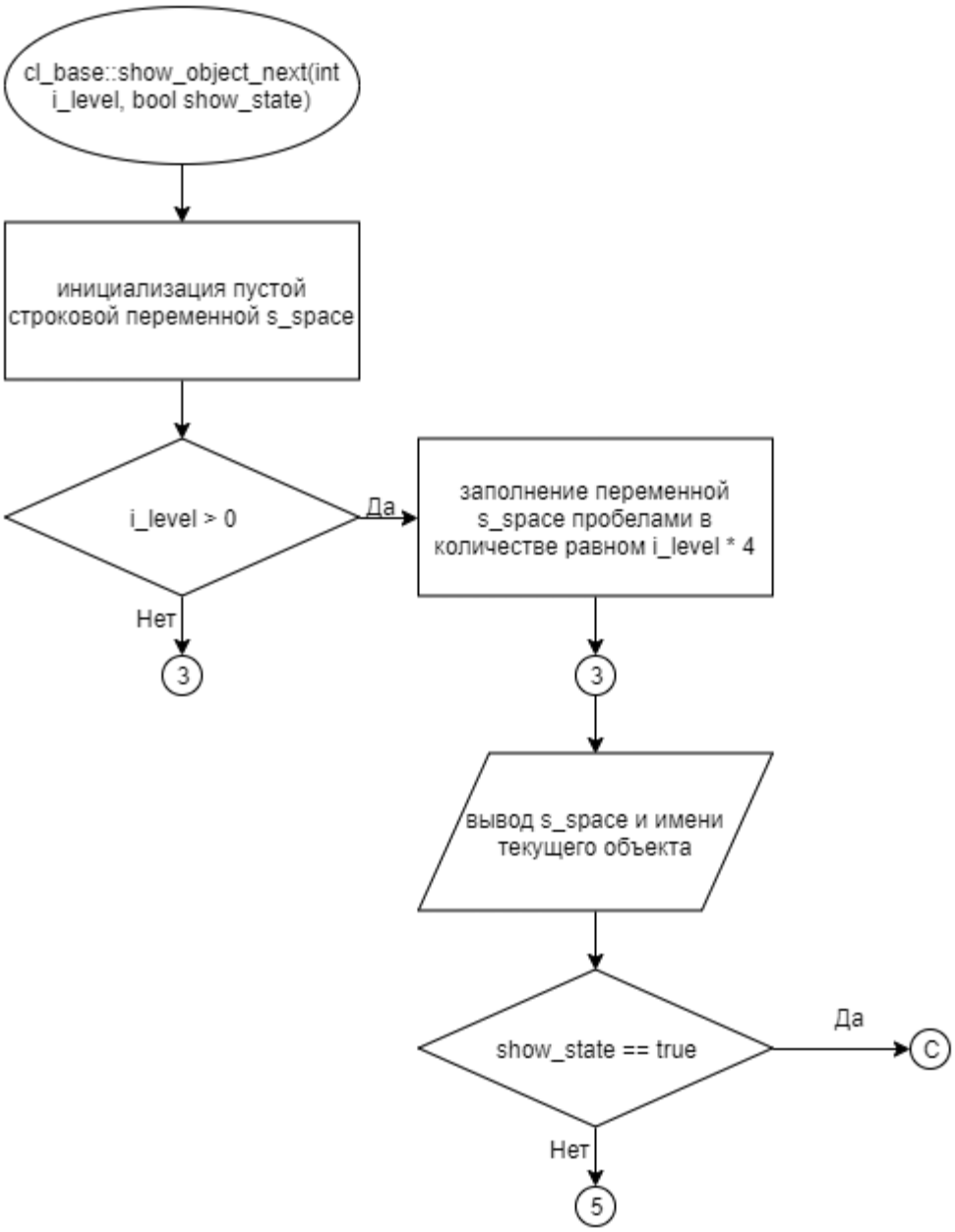

Рисунок 5 – Блок-схема алгоритма

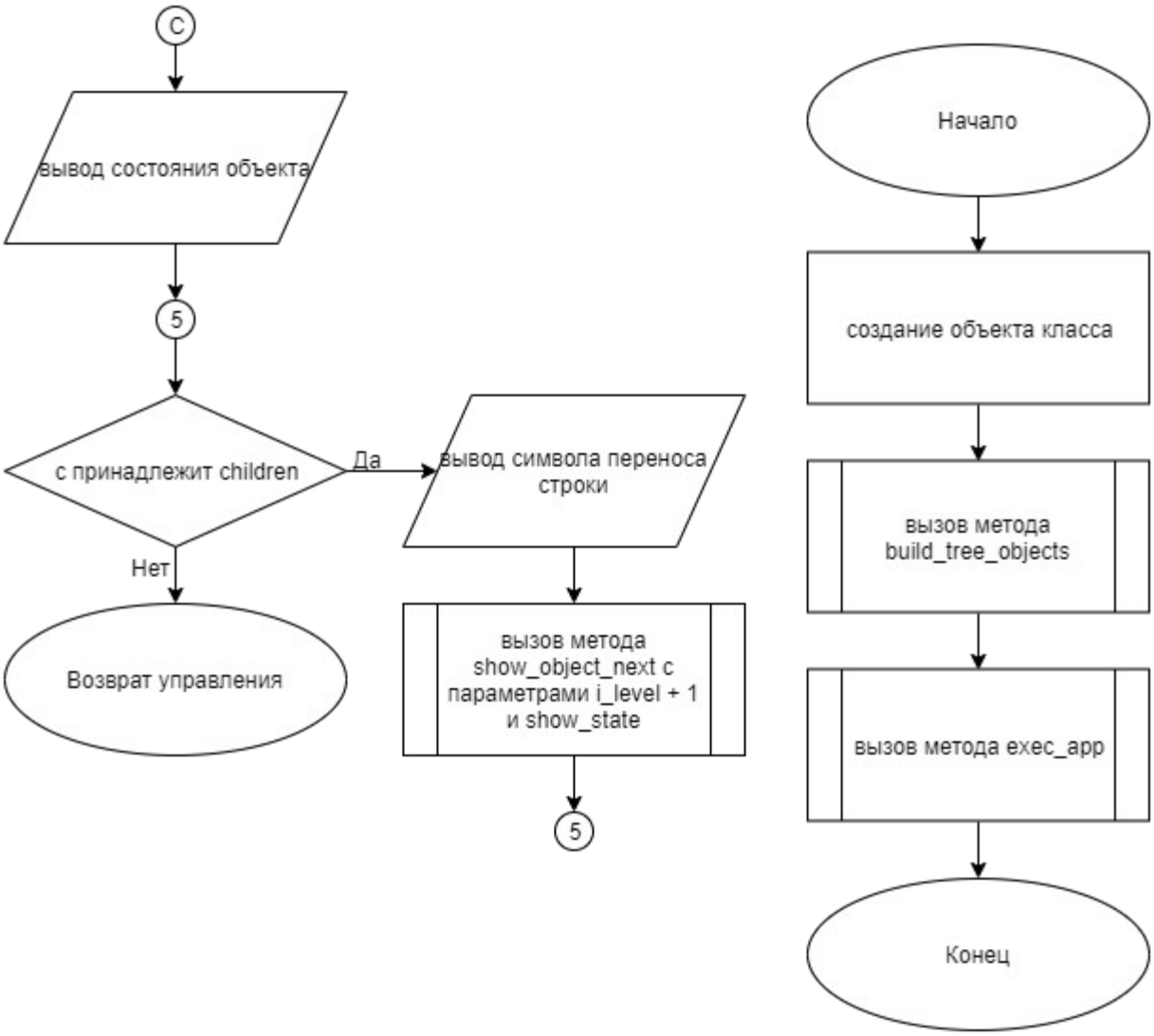

Рисунок 6 – Блок-схема алгоритма

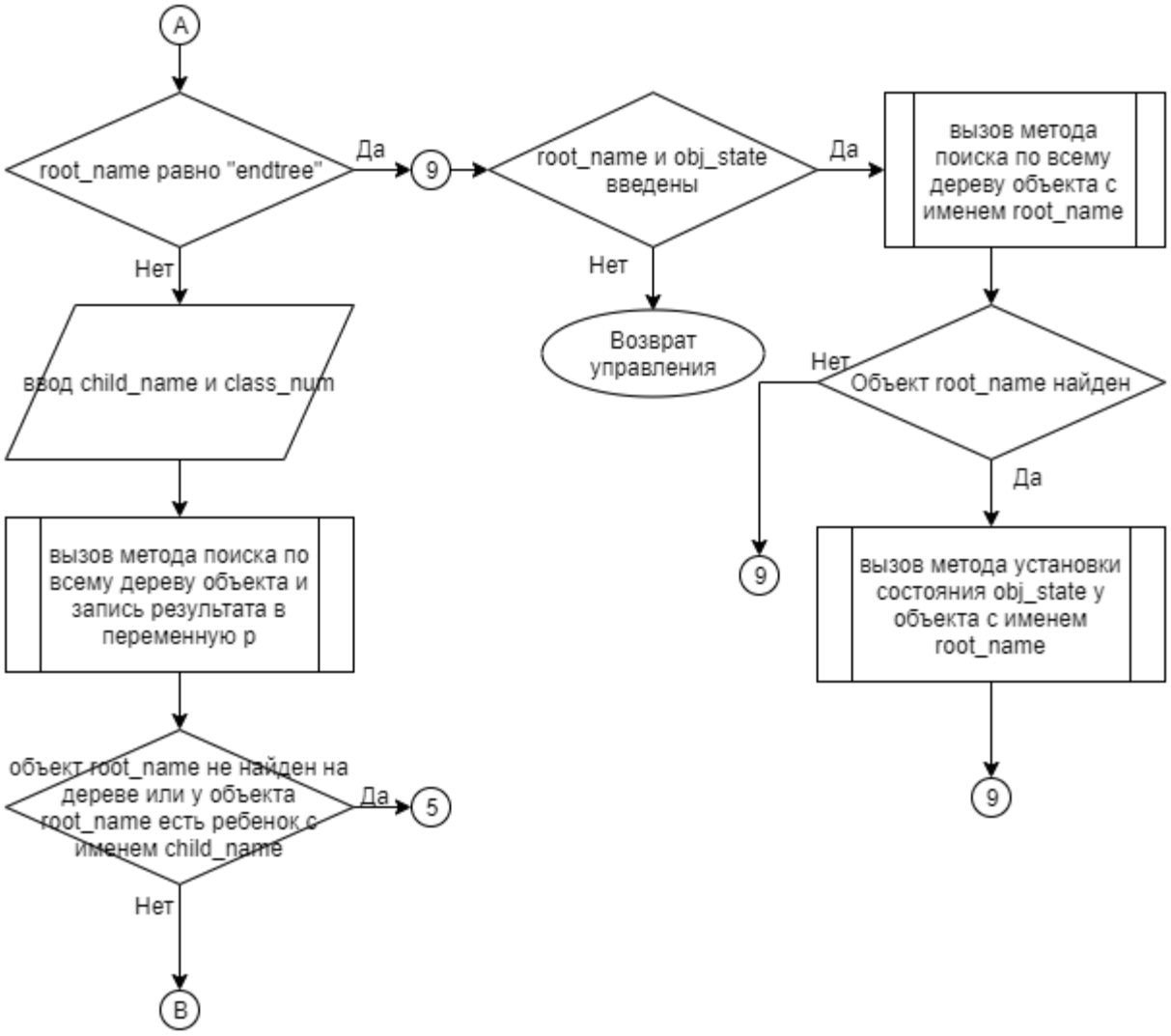

Рисунок 7 – Блок-схема алгоритма

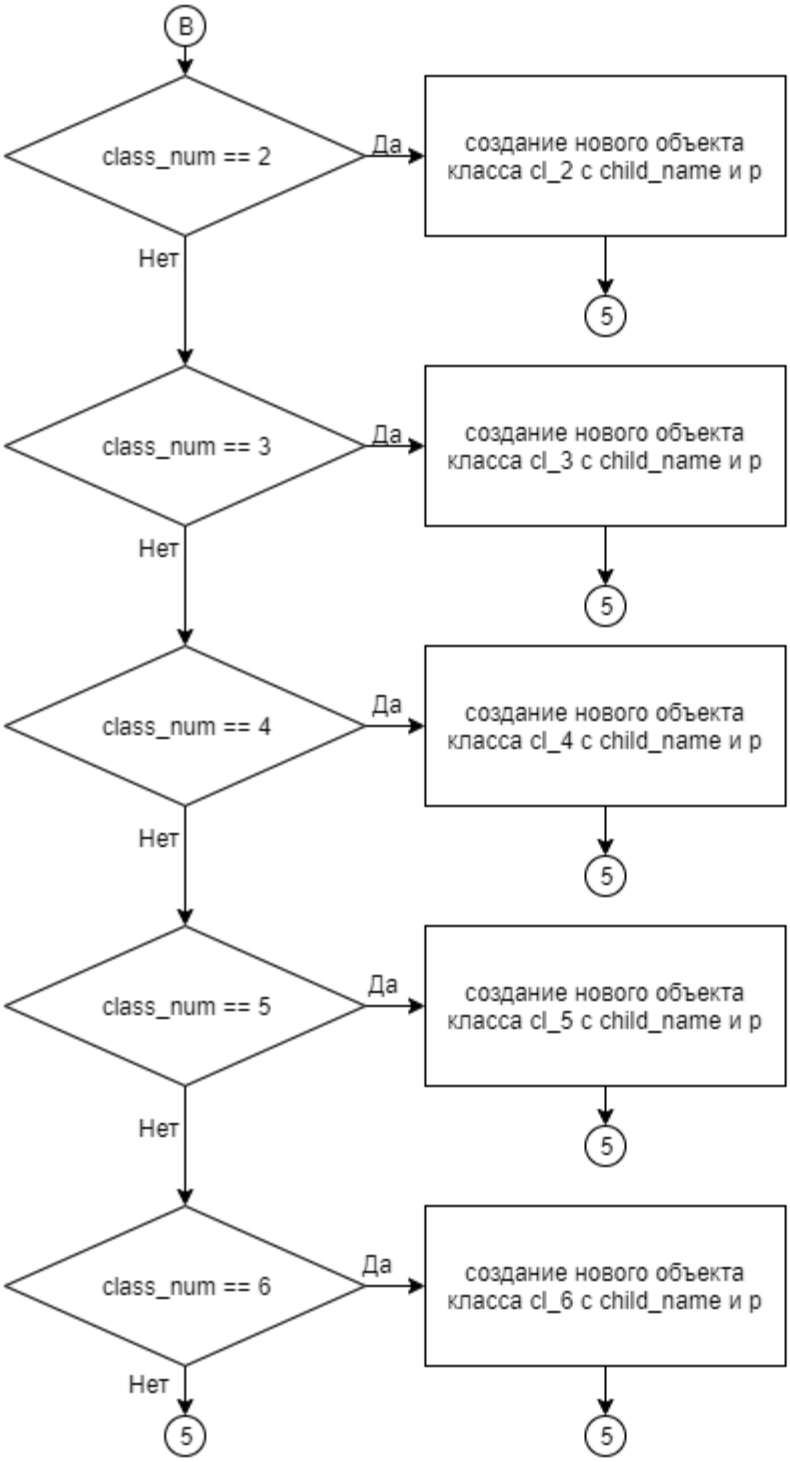

Рисунок 8 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_2::cl_2(cl_base* parent, string name) : cl_base(parent, name) {};
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_2 : public cl_base
{
public:
	cl_2(cl_base* parent=nullptr, string name="");
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_3::cl_3(cl_base* parent, string name) : cl_base(parent, name) {};
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_3 : public cl_base
{
public:
	cl_3(cl_base* parent=nullptr, string name="");
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
cl_4::cl_4(cl_base* parent, string name) : cl_base(parent, name) {};
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_4 : public cl_base
{
public:
	cl_4(cl_base* parent=nullptr, string name="");
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_5::cl_5(cl_base* parent, string name) : cl_base(parent, name) {};
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include "cl_base.h"
#include <iostream>
```

```
#include <string>
#include <vector>

using namespace std;

class cl_5 : public cl_base
{
public:
	cl_5(cl_base* parent=nullptr, string name="");
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_6::cl_6(cl_base* parent, string name) : cl_base(parent, name) {};
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_6 : public cl_base
{
public:
	cl_6(cl_base* parent=nullptr, string name="");
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_application::cl_application() : cl_base(nullptr, "") {};

void cl_application::build_tree_objects()
{
	string root_name, child_name;
	int class_num = 0, obj_state = 0;
	cin >> root_name;

	set_object_name(root_name);

	while (true)
	{
		cin >> root_name;
		if (root_name == "endtree")
		{
			break;
		}

		cin >> child_name >> class_num;

		auto p = find_on_whole_tree(root_name);
		if (!p || p->get_child(child_name))
		{
			continue;
		}

		switch (class_num)
		{
			case 2:
				new cl_2(p, child_name);
				break;
			case 3:
				new cl_3(p, child_name);
				break;
			case 4:
				new cl_4(p, child_name);
				break;
			case 5:
				new cl_5(p, child_name);
```

```
                        break;
                  case 6:
                        new cl_6(p, child_name);
                        break;
            }
      }

            while (cin >> root_name >> obj_state)
            {
                  auto obj = find_on_whole_tree(root_name);
                  if (obj)
                  {
                        obj->set_state(obj_state);
                  }
            }
}

void cl_application::exec_app()
{
      cout << "Object tree" << endl;
      show_object_tree(false);
      cout << endl << "The tree of objects and their readiness" << endl;
      show_object_tree(true);
}
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_application : public cl_base
{
public:
      cl_application();
      void build_tree_objects();
      void exec_app();
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

cl_base::cl_base(cl_base*     parent,     string     name):     p_parent(parent),
object_name(name)
{
      if (parent != nullptr)
      {
            parent->children.push_back(this);
      }
}

bool cl_base::set_object_name(string object_name)
{
      if (!p_parent)
      {
            this->object_name = object_name;
            return true;
      }

      it_child = children.begin();

      while (it_child != children.end())
      {
            if (((*it_child)->get_object_name() == object_name) && ((*it_child) !=
this))
            {
                  return false;
            }
            it_child++;
      }

      this->object_name = object_name;
      return true;
}

string cl_base::get_object_name()
{
      return object_name;
}

cl_base* cl_base::get_parent()
{
      return p_parent;
}

void cl_base::show_object_tree(bool show_state)
```

```
{
	show_object_next(0, show_state);
}

void cl_base::show_object_next(int i_level, bool show_state)
{
	string s_space = "";
	if (i_level > 0)
	{
		s_space.append(4 * i_level, ' ');
	}
	cout << s_space << get_object_name();

	if (show_state)
	{
		if (this->get_state())
		{
			cout << " is ready";
		}
		else
		{
			cout << " is not ready";
		}
	}

	for (auto c : children)
	{
		cout << endl;
		c->show_object_next(i_level + 1, show_state);
	}

}

cl_base* cl_base::get_child(string child_name)
{
	for (auto child : children)
	{
		if (child->get_object_name() == child_name)
		{
			return child;
		}
	}

	return nullptr;
}

bool cl_base::get_state()
{
	return state;
}

void cl_base::set_state(bool state)
{
	if (get_parent() && !get_parent()->get_state())
	{
		this->state = false;
	}
```

```
	else
	{
		this->state = state;
	}
	if (!state)
	{
		for (auto c : children)
		{
			c->set_state(state);
		}
	}
}

cl_base* cl_base::find(string& object_name)
{
	if (object_name == get_object_name())
	{
		return this;
	}

	for (auto c : children)
	{
		auto obj = c->find(object_name);
		if (obj)
		{
			return obj;
		}
	}

	return nullptr;
}

cl_base* cl_base::find_on_whole_tree(string& object_name)
{
	auto p = this;
	while (p->get_parent())
	{
		p = p->get_parent();
	}
	return p->find(object_name);
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
class cl_base
{
private:
	string object_name;
	cl_base* p_parent;
	bool state;
public:
	vector <cl_base*> children;
	vector <cl_base*> :: iterator it_child;
	cl_base(cl_base* parent=nullptr, string name="");
	bool set_object_name(string object_name);
	string get_object_name();
	cl_base* get_parent();
	void show_object_tree(bool show_state=false);
	void show_object_next(int i_level, bool show_state);
	cl_base* get_child(string child_name);
	bool get_state();
	void set_state(bool state);
	cl_base* find(string& object_name);
	cl_base* find_on_whole_tree(string& object_name);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "cl_application.h"

int main()
{
	cl_application ob_cl_application;
	ob_cl_application.build_tree_objects();
	ob_cl_application.exec_app();
	return(0);
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 10.

Таблица 10 – Результат тестирования программы

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

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).