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 задачи используются:

• Объект стандартного стандартного потока ввода с клавиатуры cout;

• Объект стандартного ввода с клавиатуры cin;

• Объект ob_application класса application;

• Объекты классов cl_2, cl_3, cl_4, cl_5, cl_6 (имена и количество задаются пользователем).

• Класс base (базовый класс):

• Добавленные/измененые методы:

• find_object_by_name()

• Функционал - поиск объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr).

• print_object_tree()

• Функционал - вывод дерева иерархии объектов.

• print_state()

• Функционал - вывод дерева иерархии объектов и отметок их готовности.

• install_state()

• Функционал - установка готовности объекта.

• turn_off_state()

• Функцинал - отключение готовности для головного объекта и его подчиненных.

• get_root()

• Функционал - получение указателя на корневой

объект.

• get_object_by_name()

• Функционал - получение указателя на объект по имени.

• Класс application (класс приложения):

• Добавленные/измененеые методы:

• bild_tree_object()

• Функционал - построение дерева иерархии объектов.

• exec_app()

• Функционал - метод запуска приложения (начало функционирования системы, выполнения алгоритма решения задачи).

• Класс cl_2 (наследуется от base):

• Методы:

• Конструктор cl_2()

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

• Класс cl_3 (наследуется от base):

• Методы:

• Конструктор cl_3()

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

• Класс cl_4 (наследуется от base):

• Методы:

• Конструктор cl_4()

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.
• Класс cl_5 (наследуется от base):
• Методы:
• Конструктор cl_5()
• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.
• Класс cl_6(наследуется от base):
• Методы:
• Конструктор cl_6()
• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.0 Алгоритм метода find_object_by_name класса base

Функционал: Поиск объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr).

Параметры: строковый, name - наименование объекта.

Возвращаемое значение: указатель на объект класса base.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода find_object_by_name класса base

№	Предикат	Действия	№ перехода
1		Инициализация указателя p на объект класса base значением нулевого указателя	2
2	Результат вызова метода get_object_name для текущего объекта равен значению параметра name	вовращение указателя на текущий объект	∅
		Указателю p присвоить значение	3

№	Предикат	Действия	№ перехода
		указателя на текущий объект	
3		Инициализация целочисленной переменной счетчика i значением 0	4
4	Значение переменной i меньше размера массива children		5
			7
5	Результат вызова метода get_object_name для объекта подчиненного текущему с именем name равно значению параметра name	возврат результата вызова метода find_object_by_name для объекта подчиненного текущему	∅
			6
6		Увеличение переменной счетчика i на 1	4
7	Указатель на головной объект для текущего равен нулевому и результат вызова метода get_object_name не	возврат нулевого указателя	∅

№	Предикат	Действия	№ перехода
	равен значению параметра name		
		возврат значение указателя p	∅

3.1 Алгоритм метода print_object_tree класса base

Функционал: вывод дерева иерархии объектов.

Параметры: указатель на класс base, parent - указатель на головной объект; целочисленный, level - уровень вывода объекта.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода print_object_tree класса base

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной s	2
2	Значение параметра level больше нуля	Добавление к строке s количество пробелов равное level*4	3
			3
3		вывод переменной s+ результата вызова метода get_object_name+переход на новую строку	4
4	размер массива children равен 0		∅
			5

№	Предикат	Действия	№ перехода
5		Инициализация целочисленной переменной счетчика i значением 0	6
6	Значение переменной i меньше размера массива children	вызов метода print_object_tree с параметрами значения указателя на подчиненный объект и level+1	7
			∅
7		Увеличение переменной счетчика i на 1	6

3.2 Алгоритм метода print_state класса base

Функционал: вывод дерева иерархии объектов и отметок их готовности.

Параметры: указатель на класс base, parent - указатель на головной объект; целочисленный, level - уровень вывода объекта.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода print_state класса base

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной s	2
2	Значение параметра level больше нуля	Добавление к строке s количество пробелов равное level*4	3

№	Предикат	Действия	№ перехода
			3
3		вывод перехода на новую строку+переменной s+ результата вызова метода get_object_name+пробел	4
4	Результат вызова метода get_state не равен 0	вывод "is ready"	5
		вывод "is not ready"	5
5	размер массива children равен 0		∅
			6
6		Инициализация целочисленной переменной счетчика i значением 0	7
7	Значение переменной i меньше размера массива children	вызов метода print_state с параметрами значения указателя на подчиненный объект и level+1	8
			∅
8		Увеличение переменной счетчика i на 1	7

3.3 Алгоритм метода install_state класса base

Функционал: установка готовности объекта.

Параметры: целочисленный, _state - состояние готовности объекта.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода install_state класса base

№	Предикат	Действия	№ перехода
1	Значение параметра _state не равно 0		2
			4
2	Значение указателя на головной объект для текущего равен нулевому	Присвоение полю state текущего объекта значение параметра _state	∅
			3
3	Значение поля state для головного объекта не равно 0	Присвоение полю state текущего объекта значение параметра _state	∅
			∅
4	Значение поля state для текущего объекта не равно 0	вызов метода turn_off_state	∅
			∅

3.4 Алгоритм метода turn_off_state класса base

Функционал: Функцинал - отключение готовности для головного объекта и его подчиненных.

Параметры: нет.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода turn_off_state класса base

№	Предикат	Действия	№ перехода
1		Присвоение полю state для текущего объекта значение 0	2
2	размер массива children равен 0		∅
			3
3		Инициализация целочисленной переменной счетчика i значением 0	4
4	Значение переменной i меньше размера массива children	вызов метода turn_off_state для объекта подчиненного к текущему	5
			∅
5		Увеличение переменной счетчика i на 1	4

3.5 Алгоритм метода bild_tree_objects класса application

Функционал: построение дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода bild_tree_objects класса application

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных name_one, name_two	2
2		Объявление целочисленной переменной number	3
3		ввод с клавиатуры значения переменной name_one	4
4		вызов метода set_object_name для текущего объекта с параметром name_one	5
5		Объявление указателя obj на класс base	6
6	Осуществляется ввод переменной name_one		7
			∅
7	значение переменной name_one равно "endtree"		∅
		ввод с клавиатуры значения переменных name_one, number	8
8	Значение переменной number = 2	создание объекта класса cl_2 посредством параметризированного	6

№	Предикат	Действия	№ перехода
		конструктора	
			9
9	Значение переменной number = 3	создание объекта класса cl_3 посредством параметризированного конструктора	6
			10
10	Значение переменной number = 4	создание объекта класса cl_4 посредством параметризированного конструктора	6
			11
11	Значение переменной number = 5	создание объекта класса cl_5 посредством параметризированного конструктора	6
			12
12	Значение переменной number = 6	создание объекта класса cl_6 посредством параметризированного конструктора	6
			6

3.6 Алгоритм метода exec_app класса application

Функционал: метод запуска приложения.

Параметры: нет.

Возвращаемое значение: целочисленное, 0 - при успешном выполнении, в другом случае - код ошибки.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		вывод "Object tree"+переход на новую строку	2
2		вызов метода print_object_tree с параметрами указателя на текущий объект и 0	3
3		объявление строковой переменной name	4
4		объявление целочисленной переменной _state	5
5	Конец входных данных		6
		вызов метода install_state для объекта с именем name с параметром значения переменной _state	5
6		вывод "The tree of objects and their readiness"	7
7		вызов метода print_state с параметрами указателя на текущий объект и 0	∅

3.7 Алгоритм метода get_object_by_name класса base

Функционал: получение значения указателя на объект по имени.

Параметры: строковый, name - наименование объекта.

Возвращаемое значение: указатель на объект класса base.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода get_object_by_name класса base

№	Предикат	Действия	№ перехода
1		возврат результата метода find_object_by_name с параметром name для корневого объекта	∅

3.8 Алгоритм метода get_root класса base

Функционал: Получение указателя на корневой объект.

Параметры: нет.

Возвращаемое значение: указатель на объект класса base(указатель на корневой объект).

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_root класса base

№	Предикат	Действия	№ перехода
1		Инициализация указателя p на класс base значением указателя на текущий объект	2
2	Значение указателя на	Присвоение указателю p значение указателя на	2

№	Предикат	Действия	№ перехода
	головной объект не равно 0	головной объект	
			3
3		возврат значение указателя p	∅

3.9 Алгоритм функции main

Функционал: основная программа.

Параметры: нет.

Возвращаемое значение: целочисленное, результат выполнения, 0 - при успешном выполнении, в другом случае - код ошибки.

Алгоритм функции представлен в таблице 10.

Таблица 10 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_application класса application посредством параметризированного конструктора с параметром нулевого указателя	2
2		вызов метода bild_tree_objects для объекта ob_application	3
3		возврат результата метода exec_app для объекта ob_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

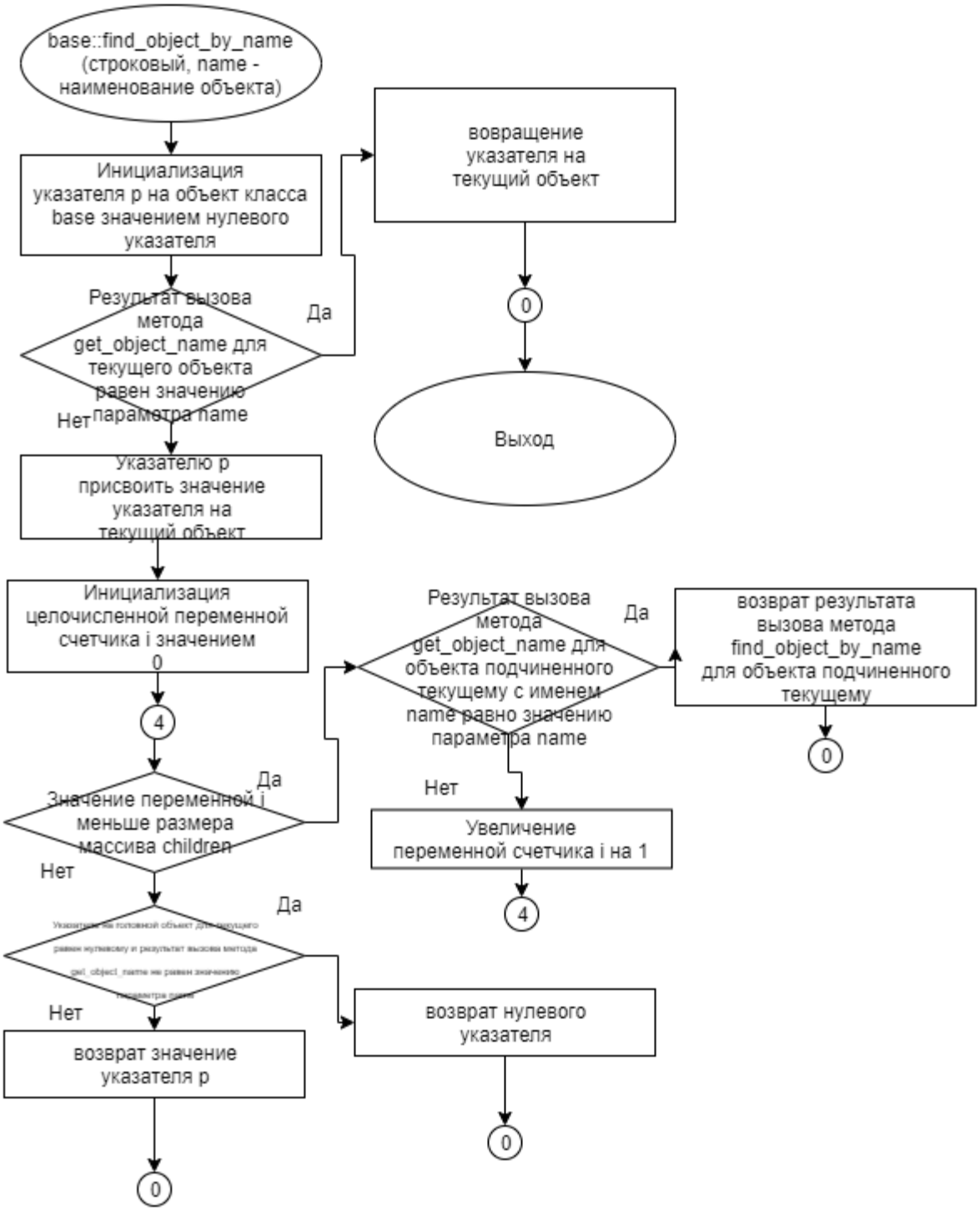

Рисунок 1 – Блок-схема алгоритма

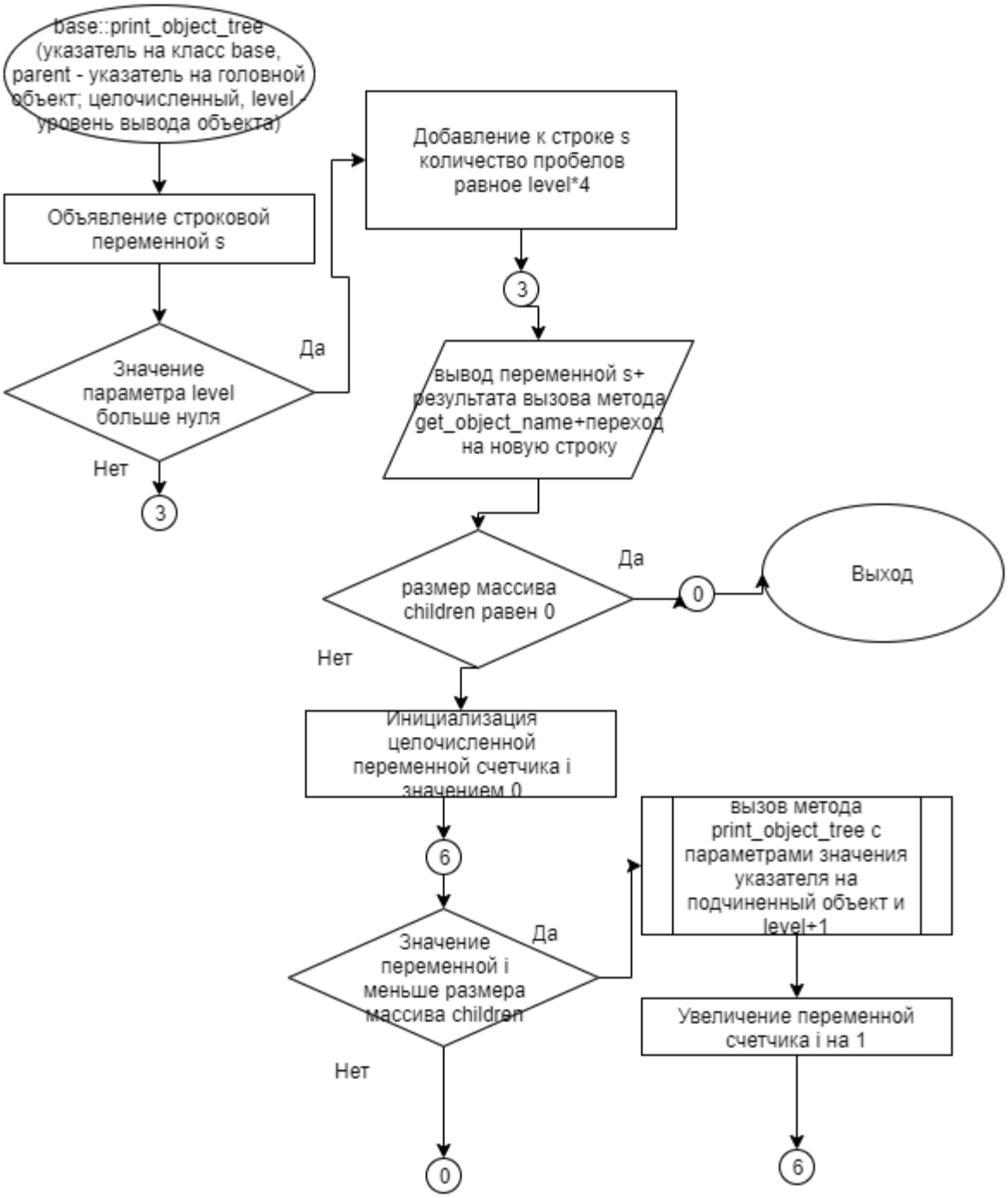

Рисунок 2 – Блок-схема алгоритма

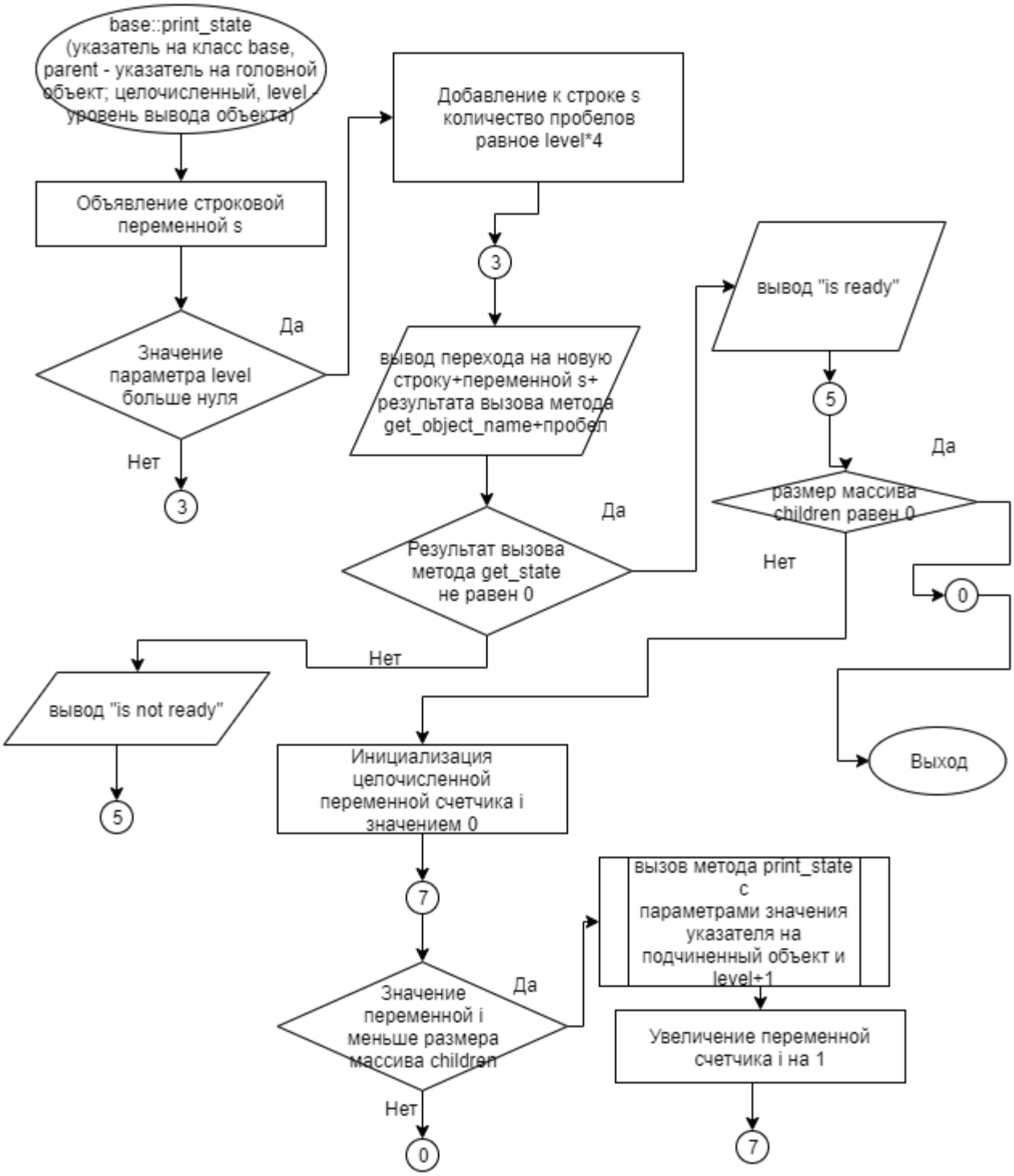

Рисунок 3 – Блок-схема алгоритма

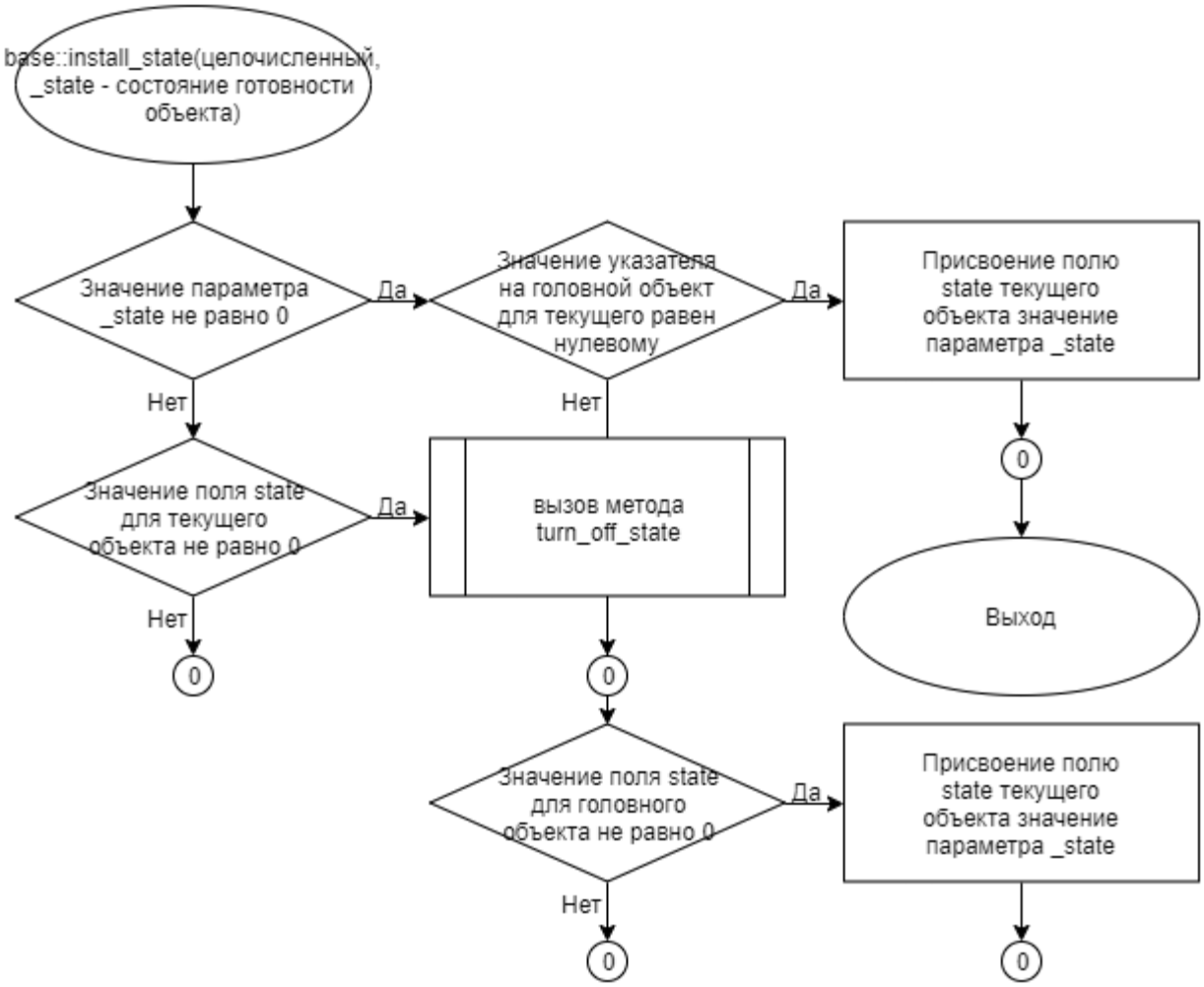

Рисунок 4 – Блок-схема алгоритма

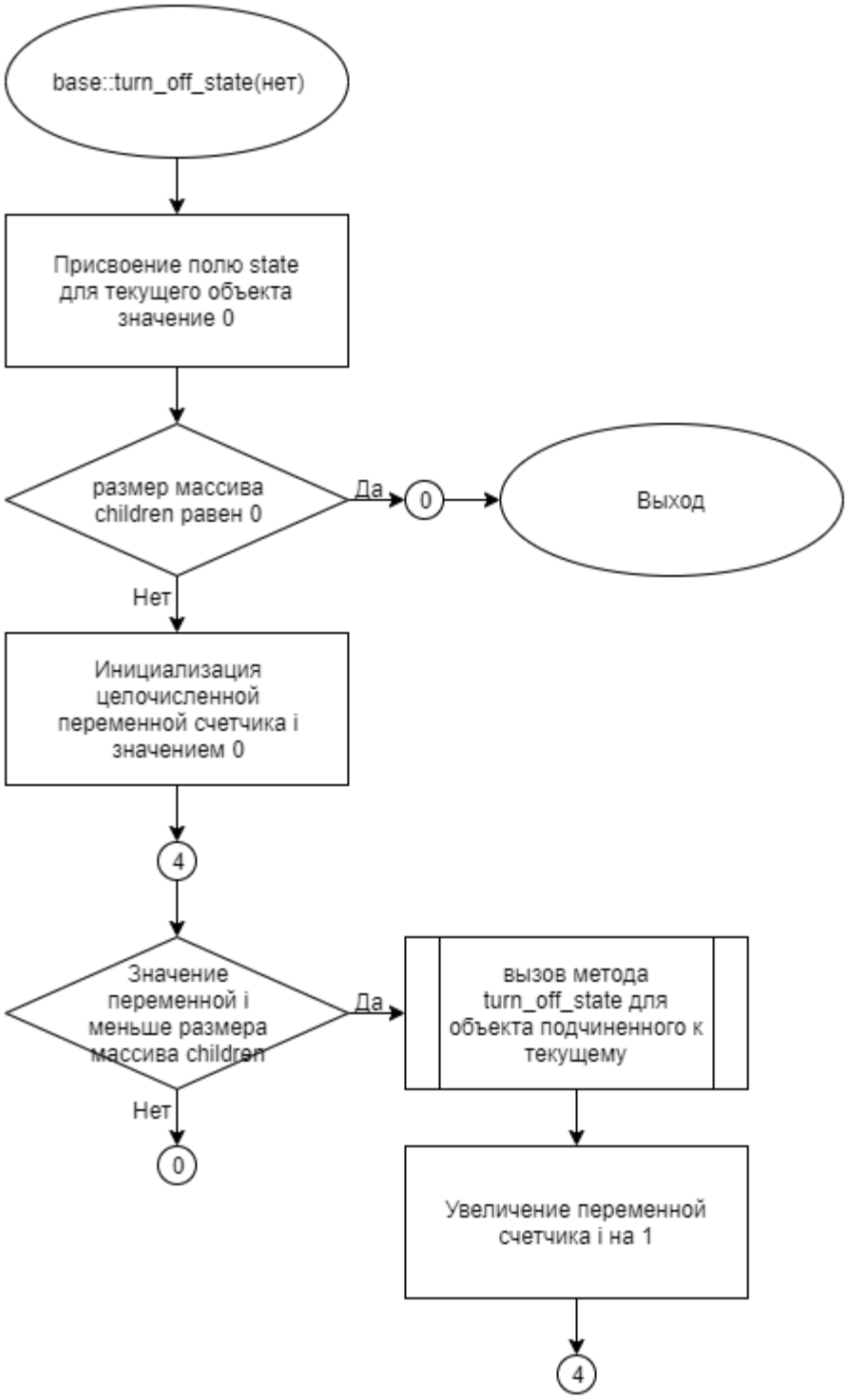

Рисунок 5 – Блок-схема алгоритма

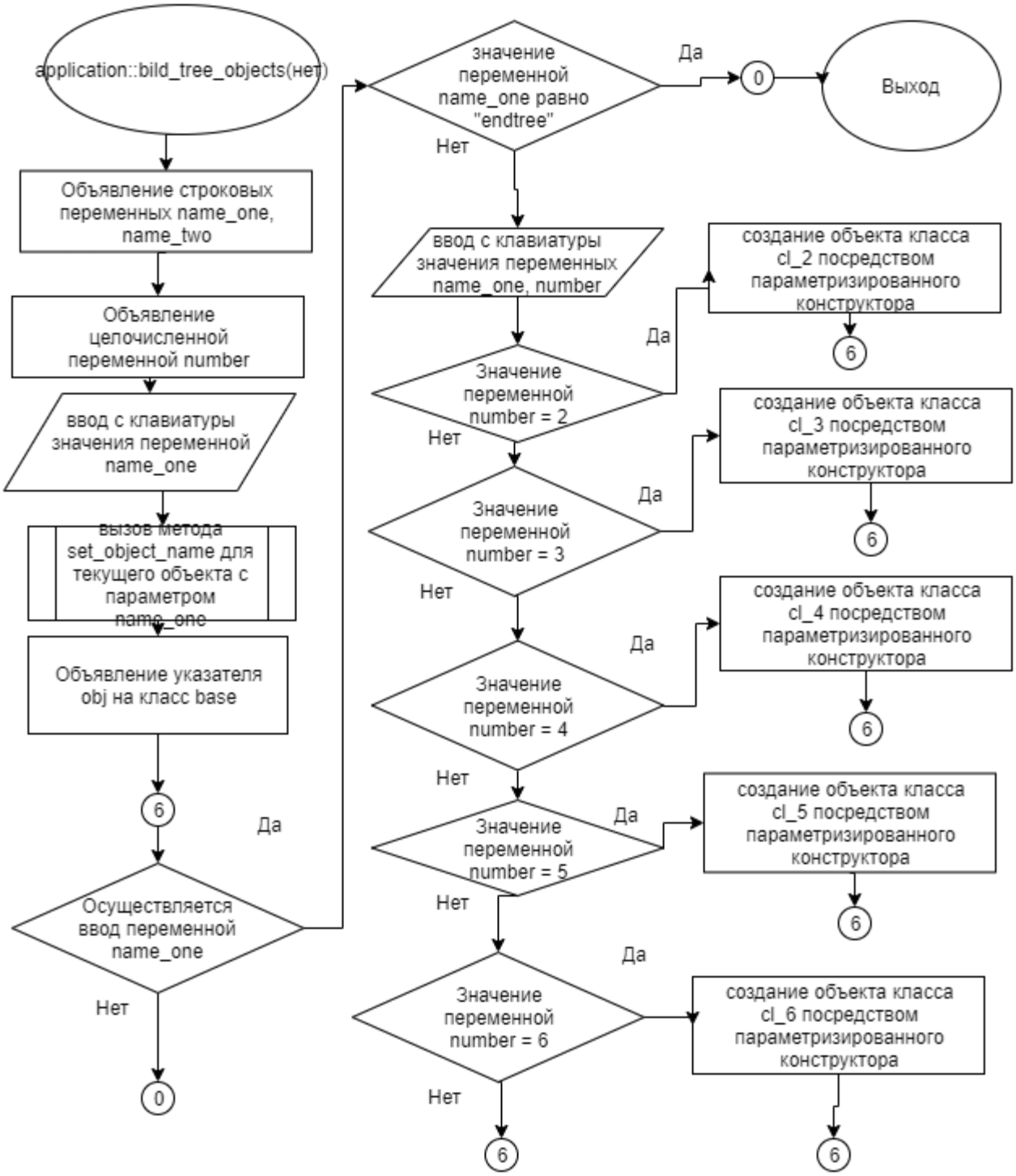

Рисунок 6 – Блок-схема алгоритма

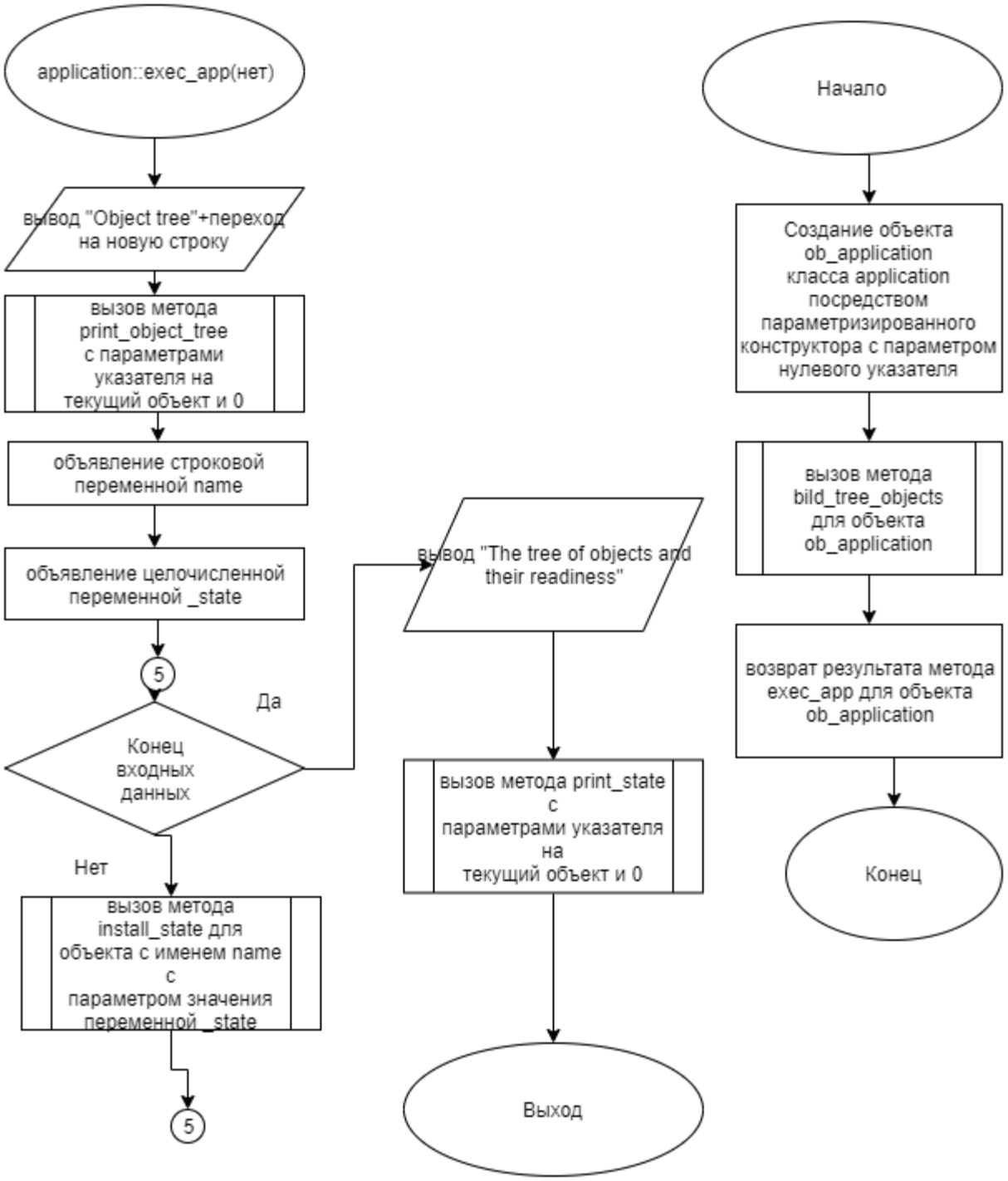

Рисунок 7 – Блок-схема алгоритма

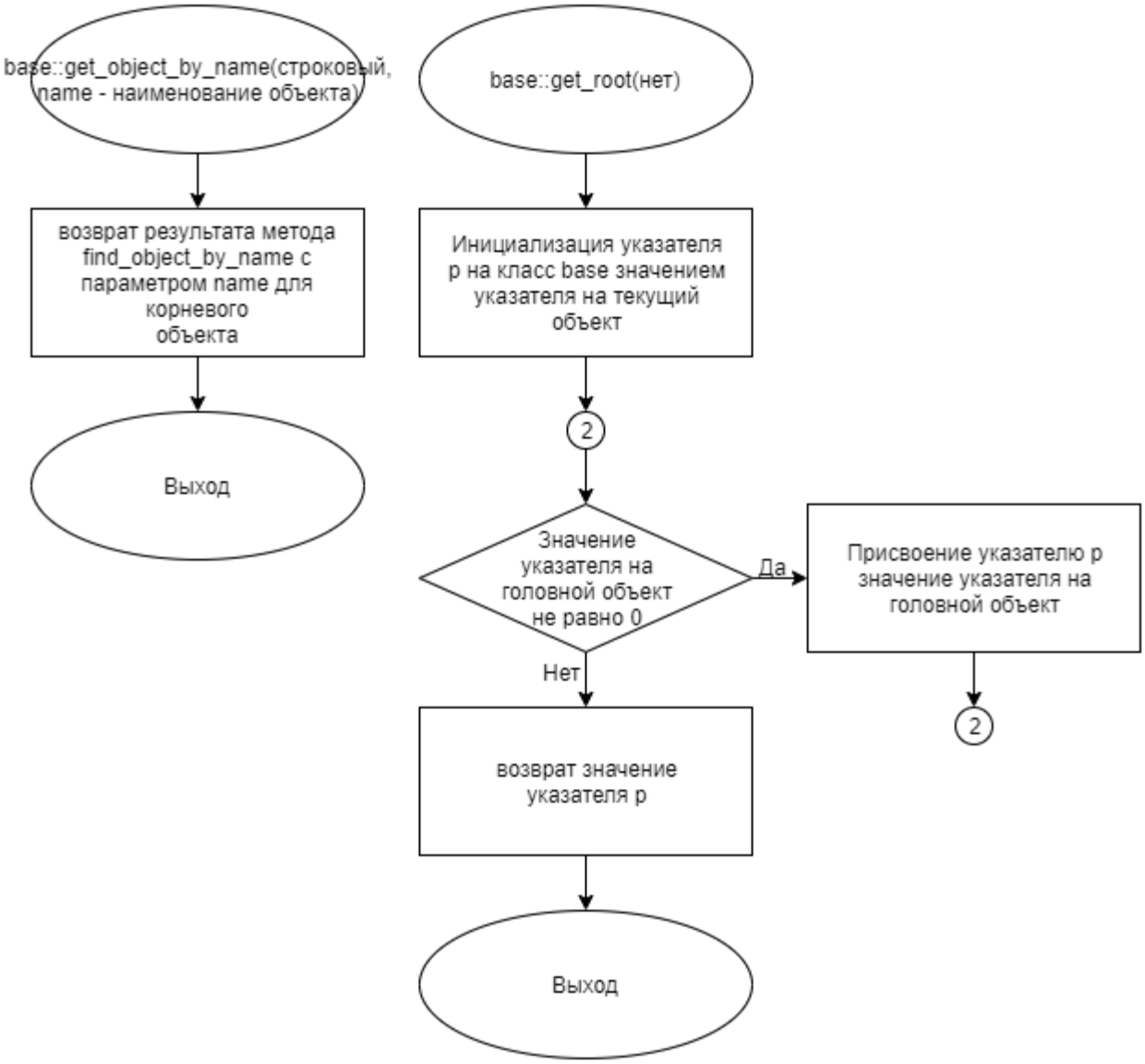

Рисунок 8 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.0 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
application::application(base*   p_parent)   :base(p_parent)//конструктор   класса
приложение
{}

void application::bild_tree_objects()
{
	string name_one, name_two;
	int number;
	cin >> name_one;
	this->set_object_name(name_one);
	base* obj;

	while (cin >> name_one)
	{
		if (name_one == "endtree") break;
		cin >> name_two >> number;
		if (number == 2) obj = new
			cl_2(get_object_by_name(name_one), name_two);
		else if (number == 3) obj = new
			cl_3(get_object_by_name(name_one), name_two);
		else if (number == 4) obj = new
			cl_4(get_object_by_name(name_one), name_two);
		else if (number == 5) obj = new
			cl_5(get_object_by_name(name_one), name_two);
		else if (number == 6) obj = new
			cl_6(get_object_by_name(name_one), name_two);
	}
}

int application::exec_app()
{
	cout << "Object tree" << endl;
	print_object_tree(this, 0);
	string name;
	int _state;
	while (cin >> name >> _state)
	{
		get_object_by_name(name)->install_state(_state);
	}
	cout << "The tree of objects and their readiness";
```

```
	print_state(this, 0);
	return 0;
}
```

5.1 Файл application.h

Листинг 2 – application.h

```
#ifndef APPLICATION_H_
#define APPLICATION_H_
#include <string>
#include "base.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
class application : public base
{
public:
application(base* p_parent);
void bild_tree_objects();// метод построения дерева иерархии
int exec_app();//метод запуска приложения
};
#endif
```

5.2 Файл base.cpp

Листинг 3 – base.cpp

```
#include "base.h"
base::base(base* p_parent, string object_name)//конструктор базового класса
{
	this->object_name = object_name;
	this->p_parent = p_parent;
	state = 0;//отметка неготовности объекта
	if (p_parent)
	{
		p_parent->children.push_back(this);
	}
}
void base::set_object_name(string object_name)
{
	this->object_name = object_name;
}
string base::get_object_name()
{
	return object_name;
}
```

```
void base::change_parent(base * new_parent)//переопределить головной объект для текущего
{
	if (this->p_parent == nullptr || new_parent == nullptr) return;
	for (int i = 0; i < p_parent->children.size(); i++)
	{
		if (p_parent->children[i] == this)
		{
			p_parent->children.erase(p_parent->children.begin()
				+ i);//удаление текущего объекта из списка наследников объекта-родителя
			return;
		}
	}
	this->p_parent = new_parent;//изменение у текущего объекта указатель на головной объект
		new_parent->children.push_back(this);//добавление текущего объекта в список наследников нового головного объекта
}
base* base::get_parent() //получение указателя на головной объект
{
	return p_parent;
}

int base::get_state()//получить состояние объекта
{
	return state;

}
base* base::get_child_by_name(string object_name)
{
	if (children.size() == 0) return nullptr;
	for (int i = 0; i < children.size(); i++)
	{
		if (children[i]->get_object_name() == object_name)
		{
			return children[i];
		}
	}

	return nullptr;
}
base* base::get_root()
{
	base* p = this;
	while (p->p_parent)
	{
		p = p->p_parent;
	}

	return p;
}
base* base::get_object_by_name(string name)
{

	return get_root()->find_object_by_name(name);
```

```
}
base* base::find_object_by_name(string name)
{
	base* p = nullptr;
	if (this->get_object_name() == name)
	{
		return this;
	}
	else p = this;

	for (int i = 0; i < children.size(); i++)
	{
		if (children[i]->find_object_by_name(name)->get_object_name() == name)
		{
			return children[i]->find_object_by_name(name);
		}
	}

	if (p_parent == nullptr && this->get_object_name() != name)
	{
		return nullptr;
	}
	else return p;
}

void base::install_state(int _state)
{

	if (_state)
	{
		if (p_parent == nullptr) state = _state;
		else if (p_parent->state) state = _state;
	}
	else if (state)
	{
		turn_off_state();
	}


}
void base::turn_off_state()
{
	this->state = 0;
	if (children.size() == 0) return;

	for (int i = 0; i < children.size(); i++)
	{
		children[i]->turn_off_state();
	}
}
void base::print_object_tree(base* parent, int level)
{
	string s;
	if (level > 0) s.append(4 * level, ' ');
	cout << s << parent->get_object_name() << endl;
	if (parent->children.size() == 0) return;
```

```
	for (int i = 0; i < parent->children.size(); i++)
	{
		print_object_tree(parent->children[i], level + 1);
	}
}
void base::print_state(base* parent, int level)
{
	string s;
	if (level > 0) s.append(4 * level, ' ');
	cout << endl << s << parent->get_object_name() << ' ';
	if (parent->get_state()) cout << "is ready";
	else cout << "is not ready";
	if (parent->children.size() == 0) return;

	for (int i = 0; i < parent->children.size(); i++)
	{
		print_state(parent->children[i], level + 1);
	}
}
base::~base()
{
	for (int i = 0; i < children.size(); i++)
	{
		delete children[i];
	}

}
```

5.3 Файл base.h

Листинг 4 – base.h

```
#ifndef BASE_H_
#define BASE_H_
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class base
{
private:
	string object_name;//наименование объекта
	base* p_parent;//указатель на головной объект
	int state; //состояние объекта
	vector <base*>children;//массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии
		base* get_root();//получение указателя на корневой объект
public:
	base(base* p_parent, string object_name = "base");
	void set_object_name(string object_name);
	string get_object_name();//получить имя объекта
	void change_parent(base* new_parent);//переопределить головной объект для текущего
```

```
            base*  get_parent();//получить  указатель  на  головной  объект  для
текущего
            int get_state();//получить состояние объекта
      base* get_child_by_name(string object_name);//получить указатель на объект
потомок по имени объекта
            base* get_object_by_name(string name);//получить указатель на объект
по имени
            base*  find_object_by_name(string  name);//поиск  объекта  на  дереве
иерархии по имени
            void print_object_tree(base* parent, int level);//вывод дерев иерархии
            void install_state(int _state);//установка готовности объекта
      void turn_off_state();//отключение готовности объекта
      void print_state(base* parent, int level);//вывод дерева иерархии объектов и
отметок их готовности
            ~base();//деструктор
};
#endif
```

5.4 Файл cl_1.cpp

Листинг 5 – cl_1.cpp

```
#include "cl_1.h"
cl_1::cl_1(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.5 Файл cl_1.h

Листинг 6 – cl_1.h

```
#ifndef CL_1_H
#define CL_1_H
#include "base.h"
class cl_1: public base
{
public:
cl_1(base* p_parent, string object_name);
};
#endif
```

5.6 Файл cl_2.cpp

Листинг 7 – cl_2.cpp

```
#include "cl_2.h"
```

```
cl_2::cl_2(base* p_parent, string object_name):base(p_parent, object_name)
{}
```

5.7 Файл cl_2.h

Листинг 8 – cl_2.h

```
#ifndef CL_2_H
#define CL_2_H
#include "base.h"
class cl_2: public base
{
public:
cl_2(base* p_parent, string object_name);
};
#endif
```

5.8 Файл cl_3.cpp

Листинг 9 – cl_3.cpp

```
#include "cl_3.h"
cl_3::cl_3(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.9 Файл cl_3.h

Листинг 10 – cl_3.h

```
#ifndef CL_3_H
#define CL_3_H
#include "base.h"
class cl_3: public base
{
public:
cl_3(base* p_parent, string object_name);
};
#endif
```

5.10 Файл cl_4.cpp

Листинг 11 – cl_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.11 Файл cl_4.h

Листинг 12 – cl_4.h

```
#ifndef CL_4_H
#define CL_4_H
#include "base.h"
class cl_4: public base
{
public:
cl_4(base* p_parent, string object_name);
};
#endif
```

5.12 Файл cl_5.cpp

Листинг 13 – cl_5.cpp

```
#include "cl_5.h"
cl_5::cl_5(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.13 Файл cl_5.h

Листинг 14 – cl_5.h

```
#ifndef CL_5_H
#define CL_5_H
#include "base.h"
class cl_5: public base
{
public:
cl_5(base* p_parent, string object_name);
};
#endif
```

5.14 Файл cl_6.cpp

Листинг 15 – cl_6.cpp

```
#include "cl_6.h"
cl_6::cl_6(base* p_parent, string object_name):base(p_parent,object_name)
{}
```

5.15 Файл cl_6.h

Листинг 16 – cl_6.h

```
#ifndef CL_6_H
#define CL_6_H
#include "base.h"
class cl_6: public base
{
public:
cl_6(base* p_parent, string object_name);
};
#endif
```

5.16 Файл main.cpp

Листинг 17 – main.cpp

```
#include "application.h"
int main()
{
application ob_application(nullptr);//создание объекта класса application
ob_application.bild_tree_objects();//построение дерева иерархии
return ob_application.exec_app();//запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 11.

Таблица 11 – 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).