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 (допускается использовать один целочисленный параметр со значением по-умолчанию);
- метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта (допускается использовать один целочисленный параметр со значением по-умолчанию);
- метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Устаревший метод вывода из задачи КВ_1 убрать.

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 корневого 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 используется:

- объекты классов cl_2, cl_3, cl_4, cl_5, cl_6 имена и количество задаются пользователем;
- объект класса cl_application;
- условный оператор if...else;
- оператор цикла со счетчиком for;
- оператор множественного выбора switch;
- оператор цикла с предусловием while.

Класс cl_base:

- свойства/поля:
    - поле наименование объекта:
        - наименование — s_name;
        - тип — string;
        - модификатор доступа — private;
    - поле указатель на головной объект данного:
        - наименование — p_head_object;
        - тип — указатель на объект класса cl_base или его наследников;
        - модификатор доступа — private;
    - поле список указателей на подчиненные объекты:
        - наименование — p_sub_objects;
        - тип — контейнер vector указателей на объекты класса cl_base или его наследников;
        - модификатор доступа — private;
    - поле состояние объекта:
        - наименование — state;

        - тип — int;
        - модификатор доступа — private;
- функционал:
    - метод set_state — задание состояние объекта;
    - метод find_obj_current — поиск объекта с заданным именем, поиск ведется от данного объекта;
    - метод find_obj_root — поиск объекта с заданным именем, поиск ведется от корневого объекта;
    - метод print_names_recur — рекурсивный вывод имен объектов дерева иерархии.

Класс cl_application:

- функционал:
    - метод build_tree_objects — построение дерева объектов.

Класс cl_2:

- функционал:
    - метод cl_2 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Класс cl_3:

- функционал:
    - метод cl_3 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Класс cl_4:

- функционал:
    - метод cl_4 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя

создаваемого объекта.

Класс cl_5:

- функционал:
  - метод cl_5 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Класс cl_6:

- функционал:
  - метод cl_6 — параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс	
		cl_application	public		2
		cl_2	public		3
		cl_3	public		4
		cl_4	public		5
		cl_5	public		6
		cl_6	public		7
2	cl_application			Класс-приложение	
3	cl_2			Дочерний класс	
4	cl_3			Дочерний класс	
5	cl_4			Дочерний класс	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
6	cl_5			Дочерний класс	
7	cl_6			Дочерний 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 set_state класса cl_base

Функционал: задание состояние объекта.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_state класса cl_base

№	Предикат	Действия	№ перехода
1	state == 0?	Поле state становится нулем	2
			4
2		Инициализация указателя на объект класса cl_base - p_sub_object первым элементом вектора указателей p_sub_objects	3
3		Для каждого элемента вектора p_sub_objects вызов set_state с gfhfvtnhjv 0	∅
4		Инициализация булевого значения par_ready = true	5
5		Инициализация указателя root_obj на объект класса cl_base указателем на текущий объект	6
6	Есть головный объект для root_obj?	Присвоение root_obj указателя на головной объект текущего	7
			9
7	Поле state головного объекта равен нулю?	par_ready = false	8

№	Предикат	Действия	№ перехода
			6
8		Выход из цикла	9
9	par_ready = true	Присвоение полю state текущего объекта значения параметра state	∅
			∅

3.2 Алгоритм метода find_obj_current класса cl_base

Функционал: поиск объекта с заданным именем, поиск ведется от данного объекта.

Параметры: строка name.

Возвращаемое значение: указатель на объект класса cl_base.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода find_obj_current класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление очереди q, хранящейуказатели на объекты класса cl_base	2
2		Инициализация указателя на объект класса cl_base	3
3		Указатель на текущий объект помещается в конец очереди	4
4	Очередь не пуста?		5
		Возврат p_found	∅
5	Имя первого элемента очереди совпадает с параметром?		6
			7
6	p_found == nullptr?	Присвоение p_found значения первого элемента очереди	7

№	Предикат	Действия	№ перехода
		Возврат nullptr	∅
7		Все дочерние объекты текущего добавляются в конец очереди	8
8		Удаление первого элемента очереди	9
9		Возврат p_found	∅

3.3 Алгоритм метода find_obj_root класса cl_base

Функционал: поиск объекта с заданным именем, поиск ведется от корневого объекта.

Параметры: строка name.

Возвращаемое значение: указатель на объект класса cl_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода find_obj_root класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя p_root_object класса cl_base указателем на текущий объект	2
2	У текущего элемента есть родитель?	Присвоение указателю p_root_object указателя на головной объект текущего	2
			3
3		Возврат результата find_object_curr от p_root_object	∅

3.4 Алгоритм метода print_names_recur класса cl_base

Функционал: рекурсивный вывод имен объектов дерева иерархии.

Параметры: целое число level - показатель количества отступов от края окна.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода print_names_recur класса cl_base

№	Предикат	Действия	№ перехода
1		Перенос вывода на новую строку	2
2		Инициализация счетчика i = 0	3
3	i < 4*level?	Вывод пробела	4
			5
4		i+=1	5
5		Вывод имени текущего объекта	6
6		Вызов метода print_names_recur для всех дочерних элементов текущего, в качестве параметра передается level+1	∅

3.5 Алгоритм метода build_tree_objects класса cl_application

Функционал: построение дерева объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Инициализация указателей p_head b p_sub указателями на текущий объект	2
2		Объявление строк s_sub_name, s_head_name	3
3		Ввод значения s_head_name	4
4		Задание имени головному объекту посредством вызова функции set_name с параметром s_head_name	5

№	Предикат	Действия	№ перехода
5		Инициализация целочисленных переменных i_class, i_state	6
6		Ввод значения s_head_name	7
7	s_head_name != "endtree"?	Ввод значений s_sub_name и i_class	8
		Выход из цикла	∅
8		Присвоение p_head результата работы метода find_obj_root с параметром s_head_name по указателю p_sub	9
9	Результат вызова find_ob_root с параметром s_sub_name != nullptr?	Пропуск действия цикла	7
			10
10	i_class == 2?	Создание объекта класса cl_2, в параметр конструктора передаются p_head - корневой объект и имя объекта s_sub_name	7
			11
11	i_class == 3?	Создание объекта класса cl_3, в параметр конструктора передаются p_head - корневой объект и имя объекта s_sub_name	7
			12
12	i_class == 4?	Создание объекта класса cl_4, в параметр конструктора передаются p_head - корневой объект и имя объекта s_sub_name	7
			13
13	i_class == 5?	Создание объекта класса cl_5, в параметр конструктора передаются p_head - корневой объект и имя объекта s_sub_name	7
			14
14	i_class == 6?	Создание объекта класса cl_6, в параметр конструктора передаются p_head - корневой	7

№	Предикат	Действия	№ перехода
		объект и имя объекта s_sub_name	
			7

3.6 Алгоритм конструктора класса cl_2

Функционал: параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 7.

Таблица 7 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве параметров в него передаются p_head_object и s_name	∅

3.7 Алгоритм конструктора класса cl_3

Функционал: параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве параметров в него передаются p_head_object и s_name	∅

3.8 Алгоритм конструктора класса cl_4

Функционал: параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве параметров в него передаются p_head_object и s_name	∅

3.9 Алгоритм конструктора класса cl_5

Функционал: параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве параметров в него передаются p_head_object и s_name	∅

3.10 Алгоритм метода exec_app класса cl_application

Функционал: запуск алгоритма программы.

Параметры: нет.

Возвращаемое значение: код завершения работы алгоритма.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree"	2
2		Вызов метода print_names_recur()	3
3	Был вывод значения s_head_name?	Ввод значения i_state	4
			6
4		Присвоение p_head результата работы метода find_ob_root с параметром s_head_name по указателю p_sub	5
5		Вызов метода set_state с параметром i_state по указателю p_head	6
6		Вывод с новой строки 'The tree of objects and their readiness"	7
7		Вызов метода print_states_recur()	8
8		Возврат значения	∅

3.11 Алгоритм конструктора класса cl_6

Функционал: параметризированный конструктор. В качестве параметров передается указатель на головной объект и имя создаваемого объекта.

Параметры: p_head_object - указатель на объект класса cl_base, s_name - строка.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		Вызов конструктора объекта класса cl_base, в качестве параметров в него передаются p_head_object и s_name	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-12.

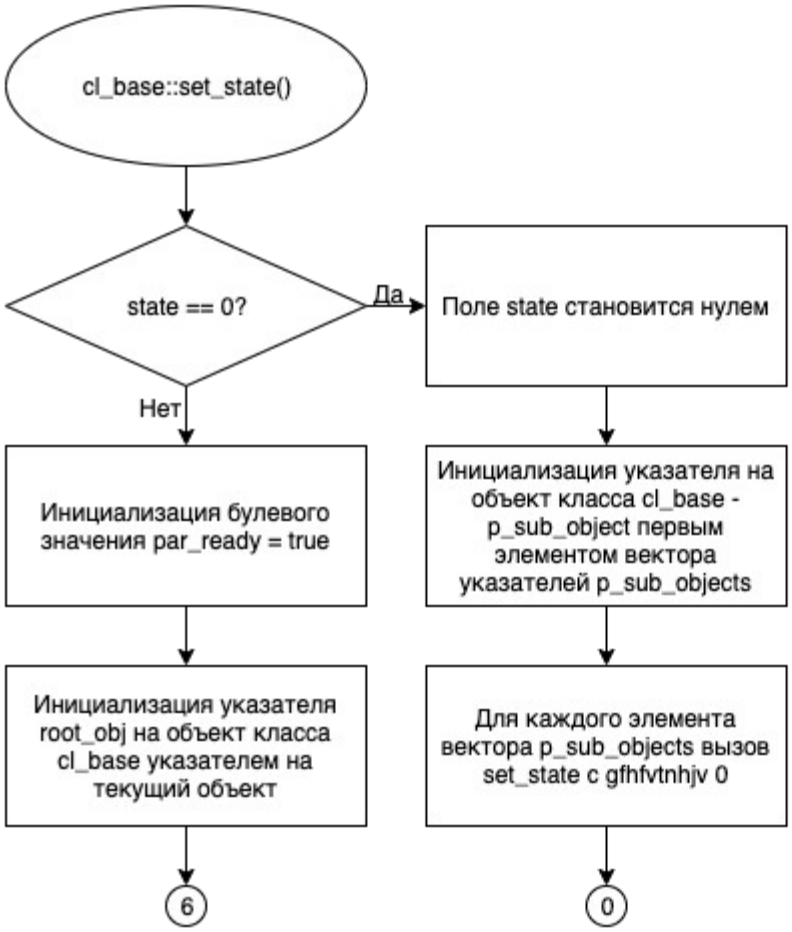

Рисунок 1 – Блок-схема алгоритма

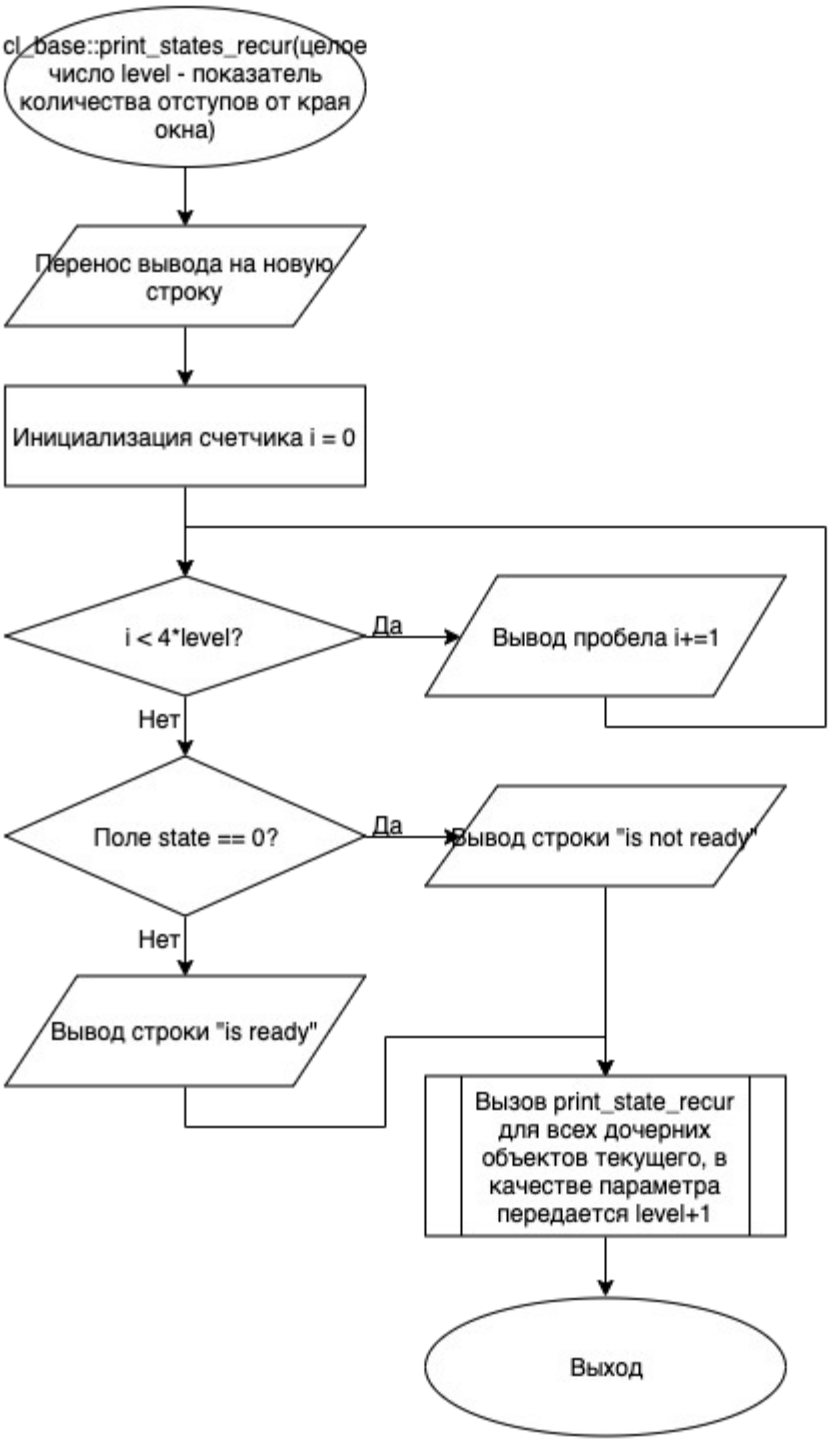

Рисунок 2 – Блок-схема алгоритма

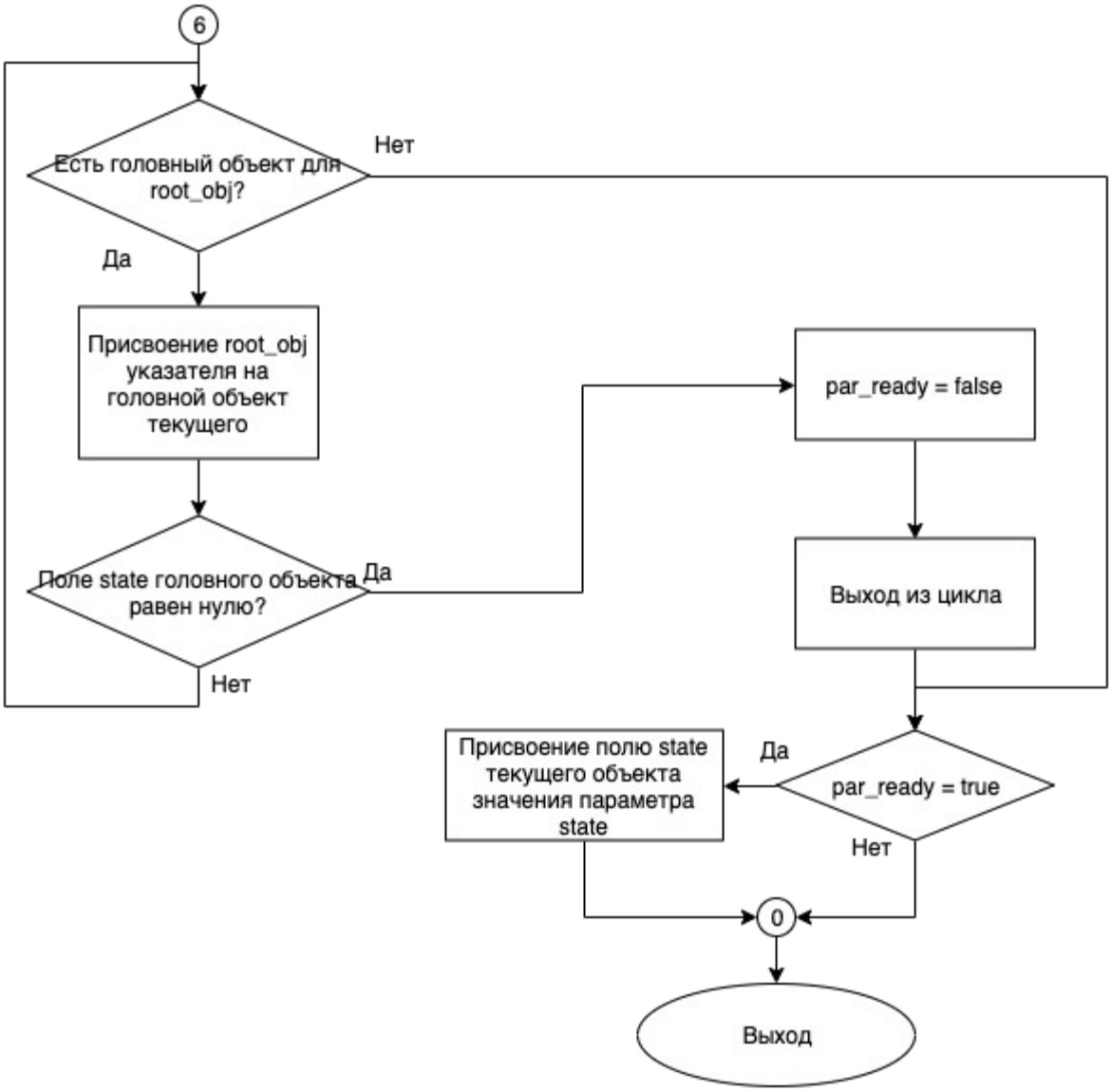

Рисунок 3 – Блок-схема алгоритма

Рисунок 4 – Блок-схема алгоритма

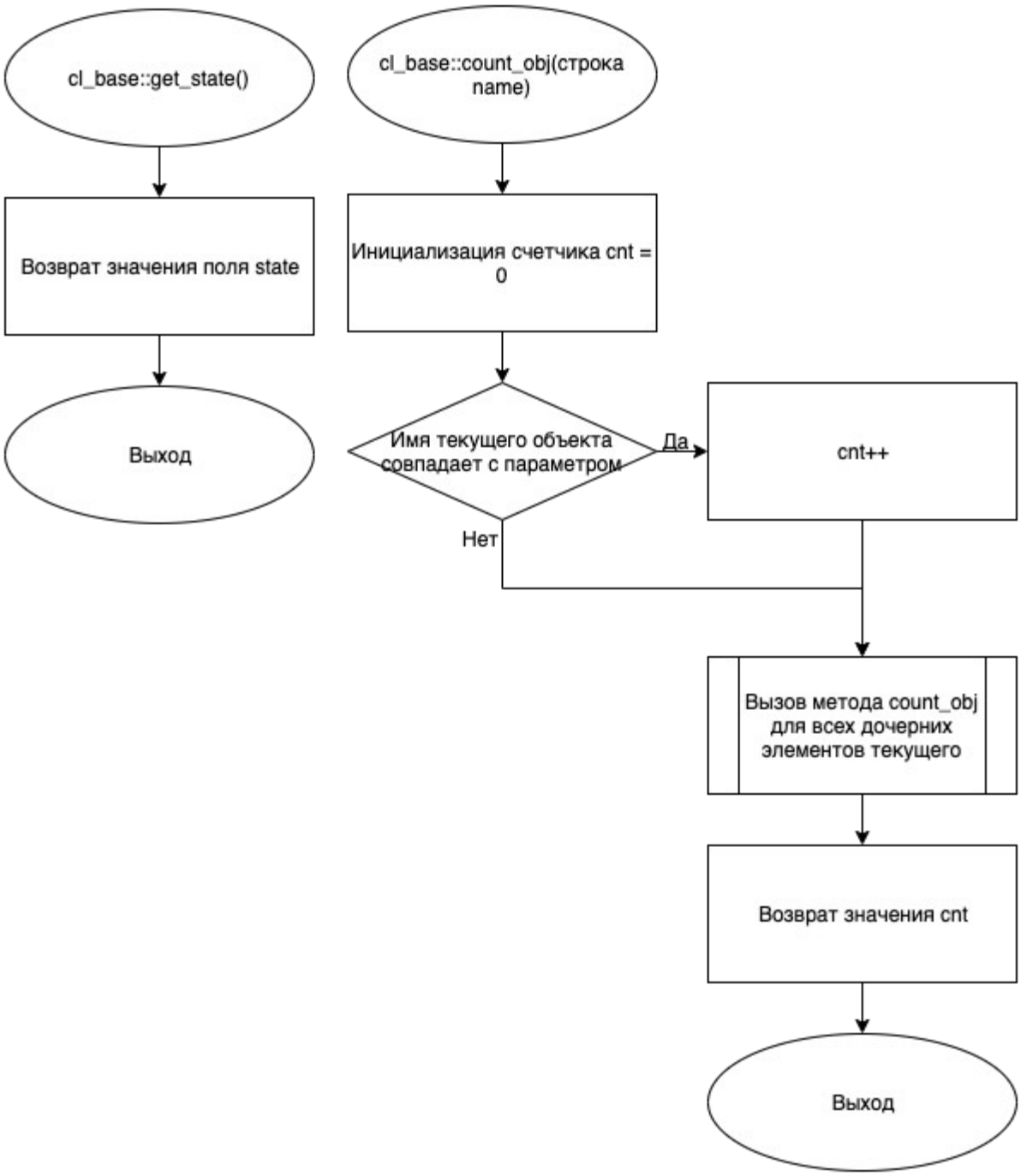

Рисунок 5 – Блок-схема алгоритма

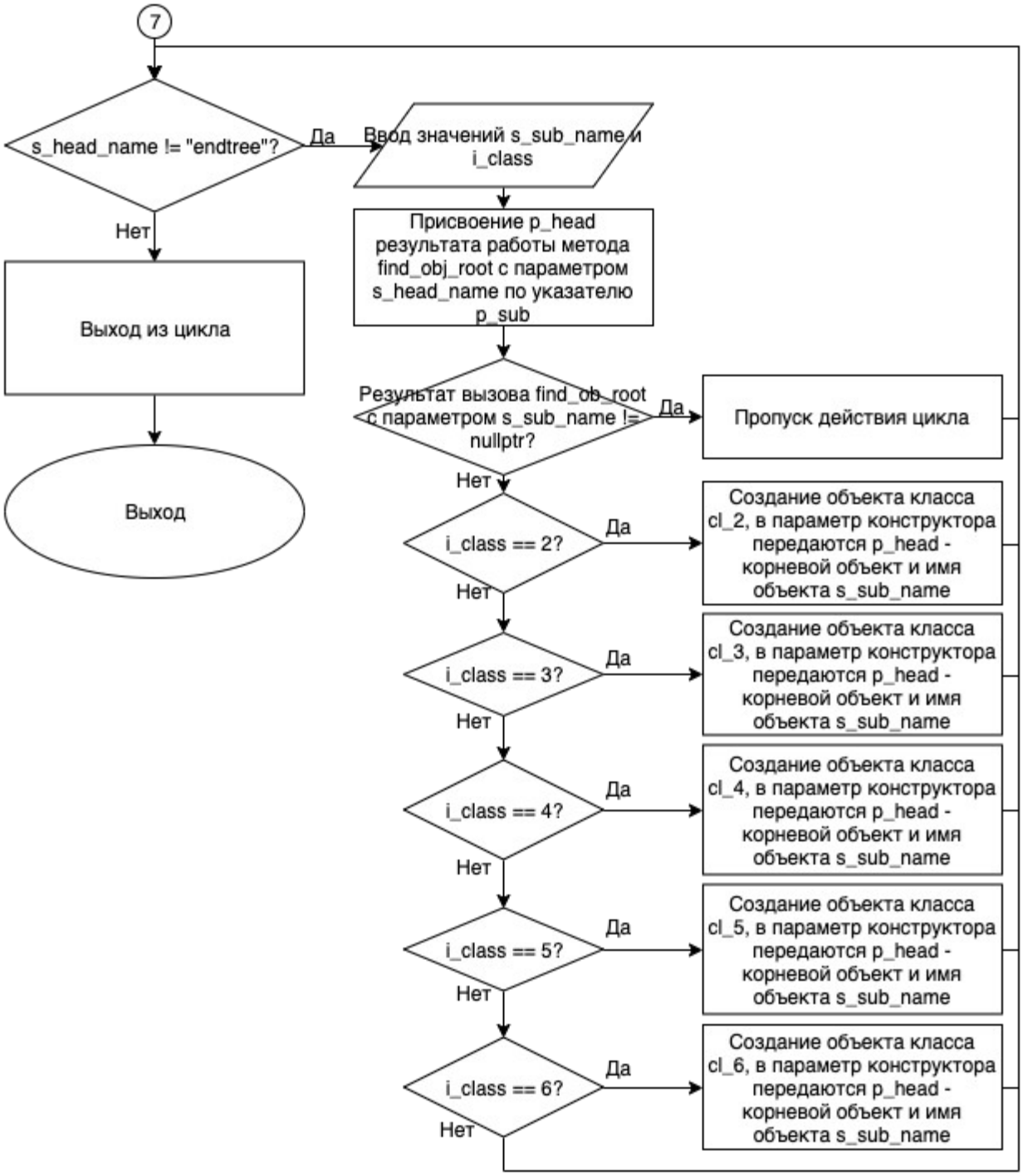

Рисунок 6 – Блок-схема алгоритма

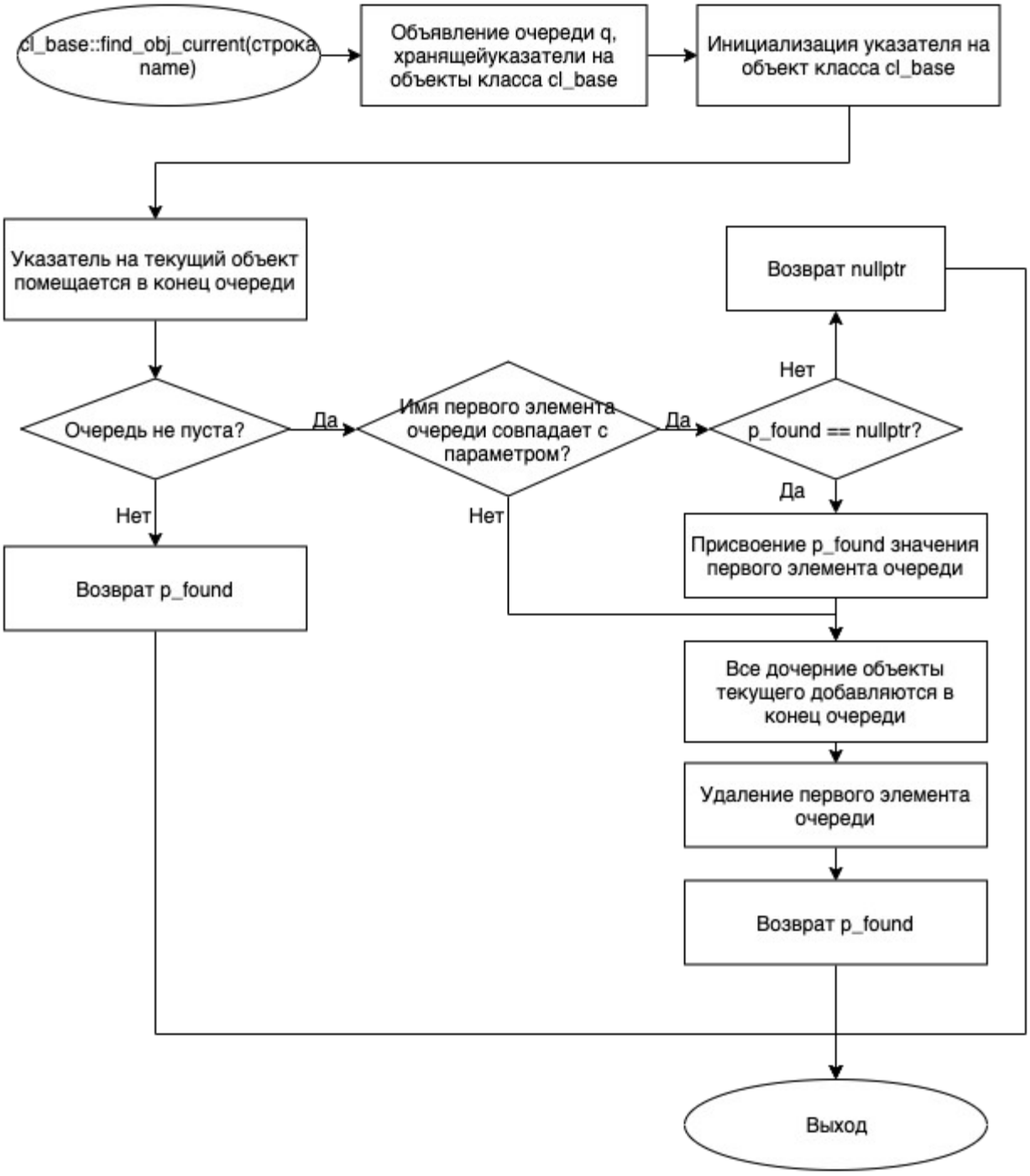

Рисунок 7 – Блок-схема алгоритма

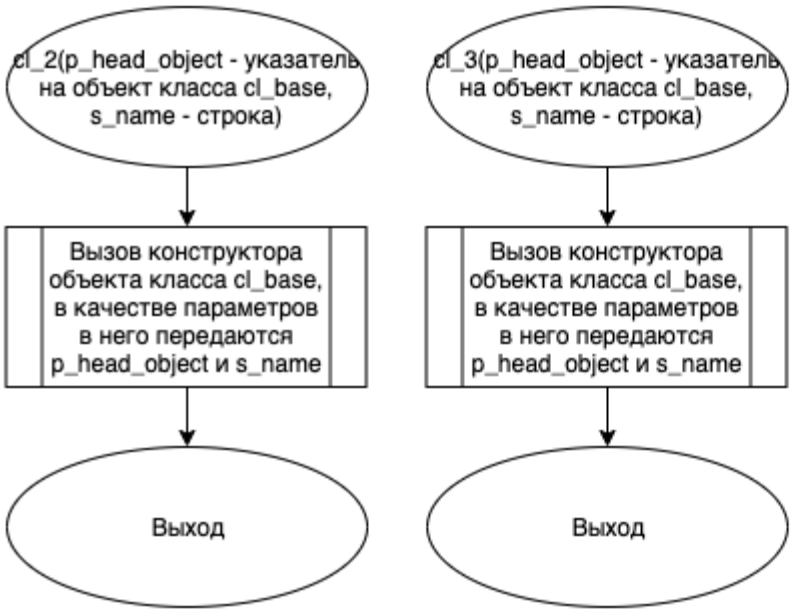

Рисунок 8 – Блок-схема алгоритма

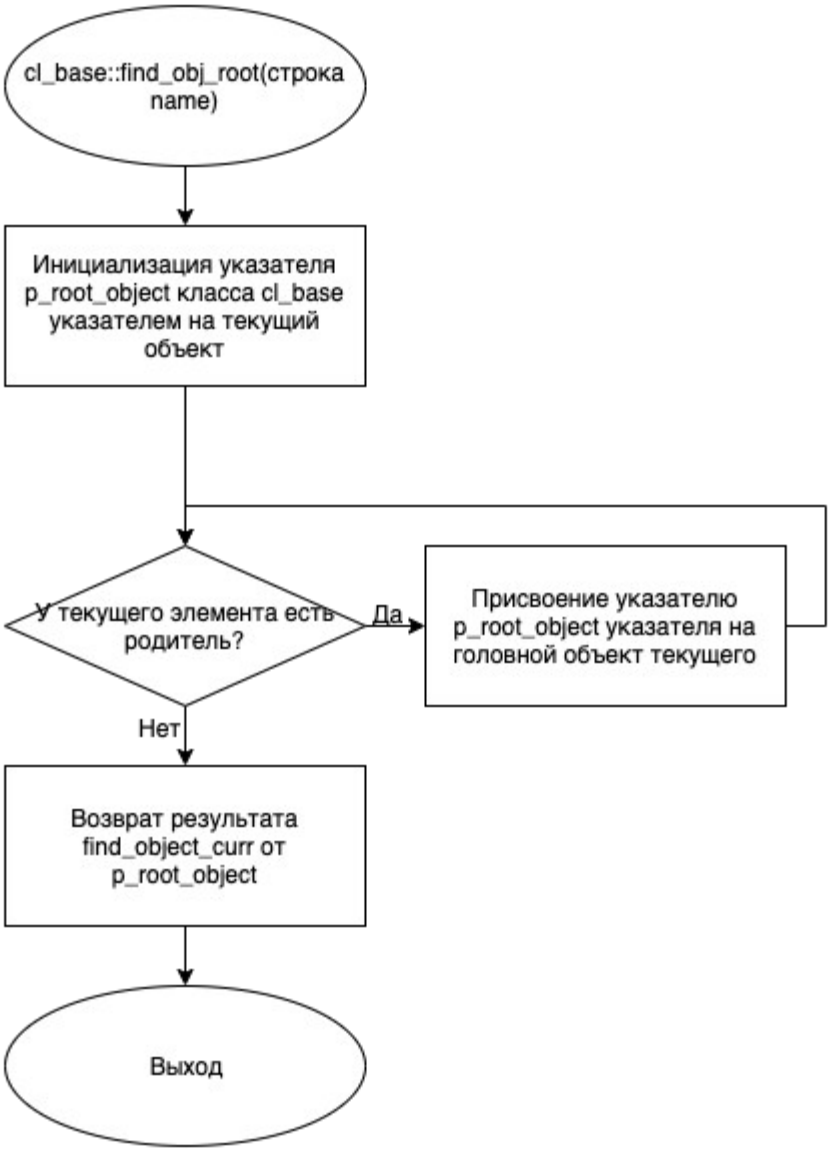

Рисунок 9 – Блок-схема алгоритма

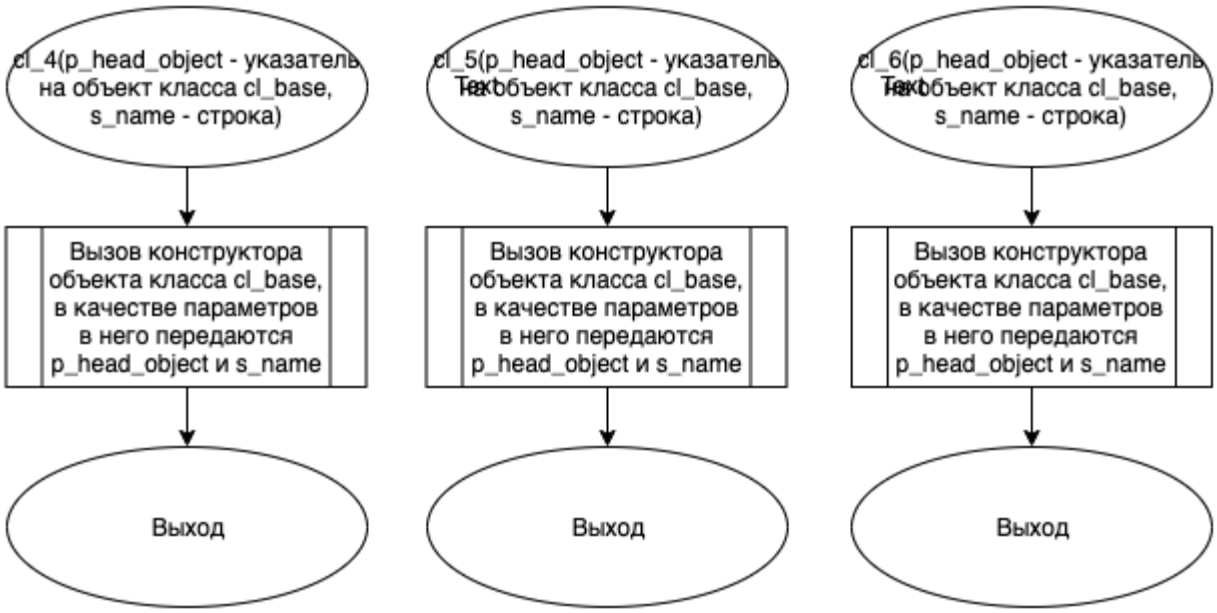

Рисунок 10 – Блок-схема алгоритма

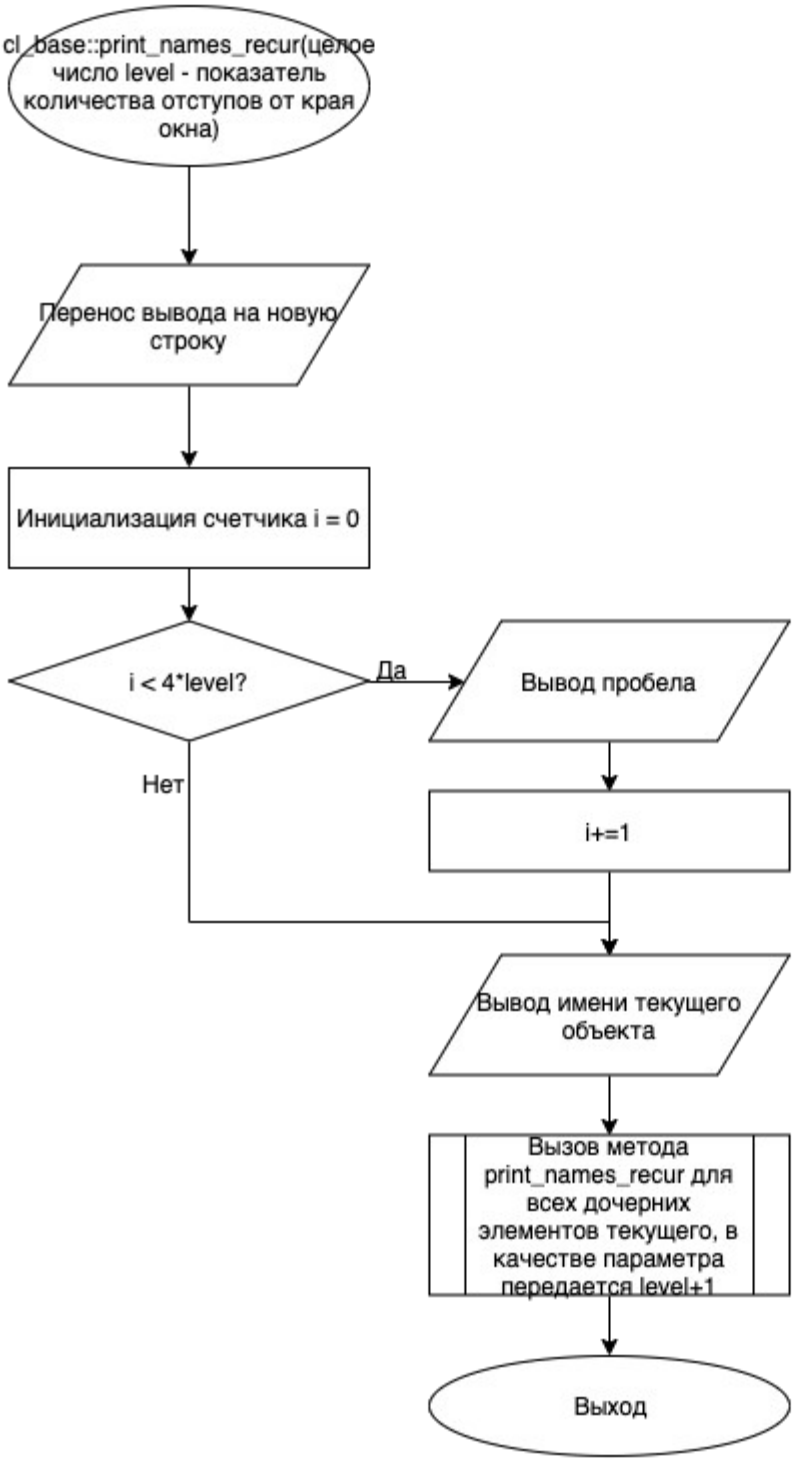

Рисунок 11 – Блок-схема алгоритма

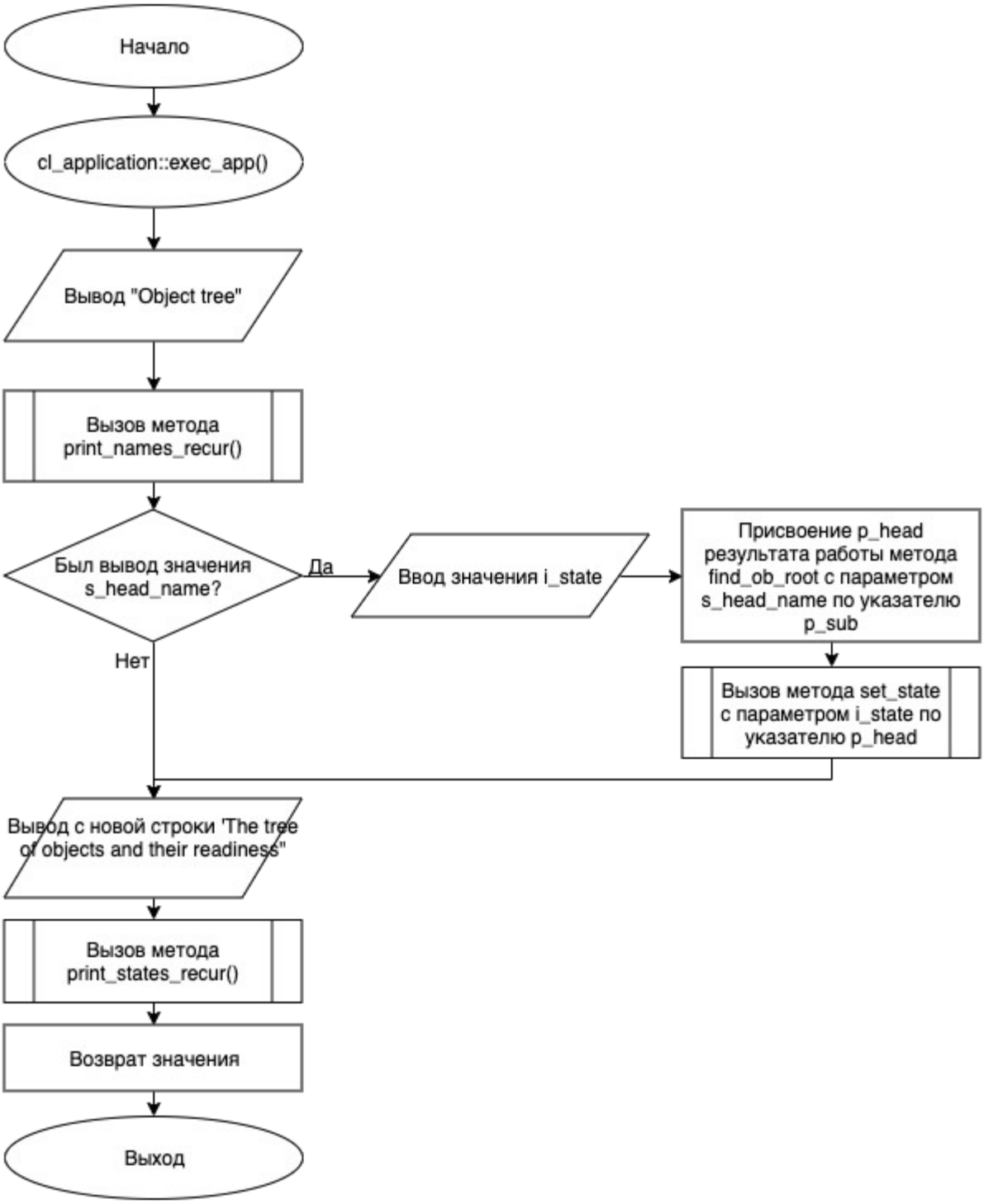

Рисунок 12 – Блок-схема алгоритма

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"
using namespace std;

cl_2::cl_2(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_2
}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"

class cl_2 : public cl_base{
public:
   cl_2(cl_base* p_head_object, string s_name);
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
using namespace std;
```

```
cl_3::cl_3(cl_base*  p_head_object,  string  s_name)  :  cl_base(p_head_object,
s_name){
   // конструктор объекта cl_3
}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"

class cl_3 : public cl_base{
public:
   cl_3(cl_base* p_head_object, string s_name);
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"
using namespace std;

cl_4::cl_4(cl_base*  p_head_object,  string  s_name)  :  cl_base(p_head_object,
s_name){
   // конструктор объекта cl_4
}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
```

```
#include "cl_base.h"

class cl_4 : public cl_base{
public:
   cl_4(cl_base* p_head_object, string s_name);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"
using namespace std;

cl_5::cl_5(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_5
}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include "cl_base.h"

class cl_5 : public cl_base{
public:
   cl_5(cl_base* p_head_object, string s_name);
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base*  p_head_object,  string  s_name)  :  cl_base(p_head_object,
s_name){
   // конструктор объекта cl_6
}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "cl_base.h"

class cl_6 : public cl_base{
public:
   cl_6(cl_base* p_head_object, string s_name);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

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

cl_application::cl_application(cl_base*           p_head_object)           :
cl_base(p_head_object){

};
```

```
void cl_application::build_tree_objects(){
   string s_sub_name, s_head_name;
   cl_base* p_head = this;
   cl_base* p_sub = nullptr;

   cin >> s_head_name;

   set_name(s_head_name);

   int i_class, i_state;

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

      cin >> s_sub_name >> i_class;
      p_head = find_obj_root(s_head_name);

      if (p_head -> find_obj_root(s_sub_name) != nullptr)
         continue;
      switch (i_class){
         case 2:
            p_sub = new cl_2(p_head, s_sub_name);
            break;
         case 3:
            p_sub = new cl_3(p_head, s_sub_name);
            break;
         case 4:
            p_sub = new cl_4(p_head, s_sub_name);
            break;
         case 5:
            p_sub = new cl_5(p_head, s_sub_name);
            break;
         case 6:
            p_sub = new cl_6(p_head, s_sub_name);
            break;
      }
   }
}

int cl_application::exec_app(){
   string s_head_name;
   cl_base* p_head;
   int i_state;
   cout << "Object tree";
   print_names_recur();

   while (cin >> s_head_name){
      cin >> i_state;
      p_head = find_obj_root(s_head_name);
      p_head -> set_state(i_state);
   }
   cout << endl << "The tree of objects and their readiness";
```

```
   print_states_recur();
   return 0;
}
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H

#include "cl_base.h"
#include "cl_2.h"

class cl_application : public cl_base{
public:
   cl_application(cl_base* p_head_object = nullptr);
   void build_tree_objects();
   int exec_app();
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
#include <iostream>
using namespace std;

// 1

cl_base::cl_base(cl_base* p_head_object, string s_name){
   this->p_head_object = p_head_object;
   if (p_head_object){
      p_head_object -> p_sub_objects.push_back(this);
   }
   this -> state = 0;
   set_name(s_name);
}

string cl_base::get_name(){
   return s_name; // возврат имени объекта
}
```

```
cl_base* cl_base::get_head(){
   return p_head_object;
   // возвращение указателя на головной объект
}

cl_base* cl_base::get_sub_object(string s_name){
   for (auto obj : p_sub_objects){
      if (obj->get_name() == s_name) return obj;
   }
   return nullptr;
}

// 2

cl_base* cl_base::find_obj_current(string name){
   queue <cl_base*> q;
   cl_base* p_found = nullptr;
   q.push(this);
   while (!q.empty()){
      if (q.front() -> get_name() == name){
         if (p_found == nullptr)
            p_found = q.front();
         else
            return nullptr;
      }
      for (auto p_sub_object : q.front() -> p_sub_objects)
         q.push(p_sub_object);
      q.pop();
   }
   return p_found;
}

// 3

cl_base* cl_base::find_obj_root (string name){
   cl_base* p_root_object = this;
   while (p_root_object -> get_head() != nullptr)
      p_root_object = p_root_object -> get_head();
   return p_root_object -> find_obj_current(name);
}

bool cl_base::set_name(string s_new_name){
   if (p_head_object)
      for (auto obj : p_head_object -> p_sub_objects)
         if(obj -> get_name() == s_new_name)
            return false;
   this -> s_name = s_new_name;
   // присвоение дочернему объекту имени, если не имеется других дочерних
объектов с аналогичным именем
   return true;
}

// 4
```

```
void cl_base::set_state(int state){
   if (state == 0){
      this -> state = 0;
      for (auto p_sub_objects : p_sub_objects){
         p_sub_objects -> set_state(0);
      }
   }
   else {
      bool par_ready = true;
      cl_base* root_obj = this;

      while (root_obj -> get_head() != nullptr){
         root_obj = root_obj -> get_head();
         if (root_obj -> state == 0){
            par_ready = false;
            break;
         }
      }
      if (par_ready)
         this -> state = state;
   }
}

// 5

void cl_base::print_names_recur(int level){
   cout << endl;
   for (int i = 0; i < 4*level; i++){
      cout << " ";
   }
   cout << get_name();

   for (auto p_sub_object : p_sub_objects){
      p_sub_object -> print_names_recur(level+1);
   }
}


void cl_base::print_states_recur(int level){
   cout << endl;
   for (int i = 0; i < 4*level; i++){
      cout << " ";
   }
   cout << get_name() << (state == 0 ? " is not ready" : " is ready");

   for (auto p_sub_object : p_sub_objects)
      p_sub_object -> print_states_recur(level+1);
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H

#include <iostream>
#include <vector>
#include <string>
#include <queue>

using namespace std;

class cl_base{

private:
   string s_name;
   cl_base* p_head_object;
   vector <cl_base*> p_sub_objects;

   int state;

public:
   cl_base(cl_base* p_head_object, string s_name = "Base object");

   bool set_name(string s_new_name);
   string get_name();
   cl_base* get_head();

   void set_state(int state);

   cl_base* get_sub_object(string s_name);
   cl_base* find_obj_current(string name);
   cl_base* find_obj_root(string name);

   void print_names_recur(int level = 0);
   void print_states_recur(int level = 0);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "cl_base.h"
#include "cl_application.h"
```

```
int main()
{
   cl_application ob_cl_application(nullptr); // создание корневого объекта
   ob_cl_application.build_tree_objects();   //   конструирование   системы,
построение дерева объектов
   return ob_cl_application.exec_app(); // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 13.

Таблица 13 – 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. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения 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).

3. 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).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

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).