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:

- объект obj_cl_app класса cl_application предназначен для постраения дерева и запуска приложения;
- функция main для работоспособности программы(основной алгоритм программы);
- if - условный оператор;
- for - цикл со счетчиком;
- while - цикл с уловием;
- объекты класса cl_2, cl_3, cl_4, cl_5, cl_6 количество которых определено самим пользователем.

Класс cl_base:

- свойства/поля:
    - поле поле индикатор состояния объекта:
        - наименование — status;
        - тип — int;
        - модификатор доступа — private;
- функционал:
    - метод find_object_from_current — метод поиска объекта на ветке дерева иерархии от текущего по имени;
    - метод find_object_from_root — метод поиска объекта на дереве иерархии по имени;
    - метод print_branch — метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
    - метод print_branch_status — метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;

- метод set_status — метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Класс cl_2:

- функционал:
  - метод cl_2 — параметризированный конструктор.

Класс cl_3:

- функционал:
  - метод cl_3 — параметризированный конструктор.

Класс cl_4:

- функционал:
  - метод cl_4 — параметризированный конструктор.

Класс cl_5:

- функционал:
  - метод cl_5 — параметризированный конструктор.

Класс cl_6:

- функционал:
  - метод cl_6 — параметризированный конструктор.

Класс cl_application:

- функционал:
  - метод build_tree_object — метод построения исходного дереваиерархии объектов;
  - метод exec_app — метод запуска приложения.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			основной класс программы	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
		cl_2	public		2
		cl_3	public		3
		cl_4	public		4
		cl_5	public		5
		cl_6	public		6
2	cl_2			класс объекта дерева	
3	cl_3			класс объекта дерева	
4	cl_4			класс объекта дерева	
5	cl_5			класс объекта дерева	
6	cl_6			класс объекта дерева	
7	cl_application			класс приложения	

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 find_object_from_current класса cl_base

Функционал: метод поиска объекта на ветке дерева иерархии от текущего по имени.

Параметры: string s_name - наименование объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода find_object_from_current класса cl_base

№	Предикат	Действия	№ перехода
1		Объявления очереди q, которая принимает указатели на объекты класса cl_base	2
2		Инициализация пустого указателя found на объекты класса cl_base	3
3		Добавление в конце очереди q указателя на текущий объект	4
4	очередь q содержит элементы	Инициализация указателя e на объект класса cl_base значением первого элемента в очереди q	5
			11
5		удаление первого элемента очереди q	6
6	имя объекта по указателю e равно параметру s_object_name		7
			8

№	Предикат	Действия	№ перехода
7	found не нулевой	возврат нулевого указателя	∅
		присвоение found значение e	8
8		инициализация целочисленной переменной i со значением 0	9
9	i < размер вектора p_sub_objects объекта с указателем e	добавление в конец очереди q элемента p_sub_objects[i] объекта e	10
			4
10		увеличение переменной i на единицу	11
11		возврат found	∅

3.2 Алгоритм метода find_object_from_root класса cl_base

Функционал: метод поиска объекта на дереве иерархии по имени.

Параметры: string s_name - наименование объекта.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода find_object_from_root класса cl_base

№	Предикат	Действия	№ перехода
1	p_head_objects не нулевой	возврат результатавызова метода find_object_from_root с переметром s_object_name по указателю p_head_object	∅
		возврат результатавызова метода find_object_from_current с переметром s_object_name	∅

3.3 Алгоритм метода print_branch класса cl_base

Функционал: метод вывода иерархии объектов (дерева или ветки) от текущего объекта.

Параметры: int layer - номер уровня дерева.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода print_branch класса cl_base

№	Предикат	Действия	№ перехода
1		вывод переноса на новую строку	2
2		инициализация целочисленной переменной i со значением 0	3
3	i < layer	вывод четырех пробелов	4
			5
4		увеличение переменной i на единицу	3
5		вывод имени текущего объекта с помощью get_name()	6
6		инициализация целочисленной переменной i со значением 0	7
7	i < длины вектора p_sub_object	вызов метода print_branch с параметром layer + 1 объекта p_sub_object[i]	8
			∅
8		увеличение переменной i на единицу	7

3.4 Алгоритм метода print_branch_status класса cl_base

Функционал: метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта.

Параметры: int layer - номер уровня дерева.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода print_branch_status класса cl_base

№	Предикат	Действия	№ перехода
1		вывод переноса на новую строку	2
2		инициализация целочисленной переменной i со значением 0	3
3	i < layer	вывод черырех пробелов	4
			5
4		увеличение i на единицу	5
5	status не нулевой	вывои имени текущего объекта и сообщение " is ready"	6
		вывои имени текущего объекта и сообщение " is not ready"	6
6		инициализация целочисленной переменной i со значением 0	7
7	i < длины вектора p_sub_object	вызов метода print_branch_status с параметром layer + 1 объекта p_sub_object[i]	8
			∅
8		увеличение i на единицу	7

3.5 Алгоритм метода set_status класса cl_base

Функционал: метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Параметры: int status - индикатор состояния объекта.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода set_status класса cl_base

№	Предикат	Действия	№ перехода
1	p_head_objects нулевой или поле status головного объекта не равно нулю	присвоение скрытому полю текущего объекта status параметра status	2
			2
2	status нулевой	присвоение скрытому полю текущего объекта status параметра status	3
			∅
3		инициализация целочисленной переменной i со значением 0	4
4	i < длины вектора p_sub_object	вызов метода set_status с параметром status объекта p_sub_object[i]	5
			∅
5		увеличение i на единицу	4

3.6 Алгоритм конструктора класса cl_2

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head - указатель на головной объект, string s_name - наименование объекта.

Алгоритм конструктора представлен в таблице 7.

Таблица 7 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head и s_name	∅

3.7 Алгоритм конструктора класса cl_3

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head - указатель на головной объект, string s_name - наименование объекта.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head и s_name	∅

3.8 Алгоритм конструктора класса cl_4

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head - указатель на головной объект, string s_name - наименование объекта.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head и s_name	∅

3.9 Алгоритм конструктора класса cl_5

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head - указатель на головной объект, string s_name - наименование объекта.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head и s_name	∅

3.10 Алгоритм конструктора класса cl_6

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head - указатель на головной объект, string s_name - наименование объекта.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		вызов параметризированного конструктора класса cl_base с параметрами p_head и s_name	∅

3.11 Алгоритм метода build_tree_object класса cl_application

Функционал: метод построения исходного дереваиерархии объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода build_tree_object класса cl_application

№	Предикат	Действия	№ перехода
1		объявление строковых переменных s_sub_name и s_head_name	2
2		объявление целочисленной переменной class_number	3

№	Предикат	Действия	№ перехода
3		инициализация указателя p_haed_object на объект класса cl_base указателем на текущий объект	4
4		ввод значений переменной s_haed_name	5
5		вызов метода set_name с параметром s_head_name	6
6		ввод значений переменной s_head_name	7
7	s_head_name равно "endtree"		16
			8
8		ввод значений переменной s_sub_name и class_number	9
9		присвоение объекту p_head_object результат работы метода find_object_from_root с параметром s_head_name	10
10	p_head_object не нулевой и результат метода find_object_from_root с параметром s_head_name, вызванного через указатель p_head_object равно нулю		11
			6
11	class_number равен 2	создание объекта класса cl_2 с параметрами p_head_object и s_sub_name с помощью оператора new	6
			12
12	class_number равен 3	создание объекта класса cl_3 с параметрами p_head_object и s_sub_name с помощью оператора new	6
			13
13	class_number равен 4	создание объекта класса cl_4 с параметрами p_head_object и s_sub_name с помощью оператора	6

№	Предикат	Действия	№ перехода
		new	
			14
14	class_number равен 5	создание объекта класса cl_5 с параметрами p_head_object и s_sub_name с помощью оператора new	6
			15
15	class_number равен 6	создание объекта класса cl_6 с параметрами p_head_object и s_sub_name с помощью оператора new	6
			16
16		объявление целочисленной переменной object_status	17
17	введено значение переменной s_head_name	ввод значения переменной object_status	18
			∅
18		вызов метода set_name с параметром object_status по указателю результата вызова метода find_object_from_root с параметром s_head_name	17

3.12 Алгоритм метода exec_app класса cl_application

Функционал: метод запуска приложения.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2

№	Предикат	Действия	№ перехода
2		вызов метода print_branch	3
3		вывод "The tree of objects and their readiness"	4
4		вызов метода print_branch_status	∅

3.13 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности работы программы.

Алгоритм функции представлен в таблице 14.

Таблица 14 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_cl_app класса cl_application в качестве параметра подается нулевой указатель	2
2		вызов метода build_tree_objects объекта ob_cl_app	3
3		вызов метода exec_app объекта ob_cl_app	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-9.

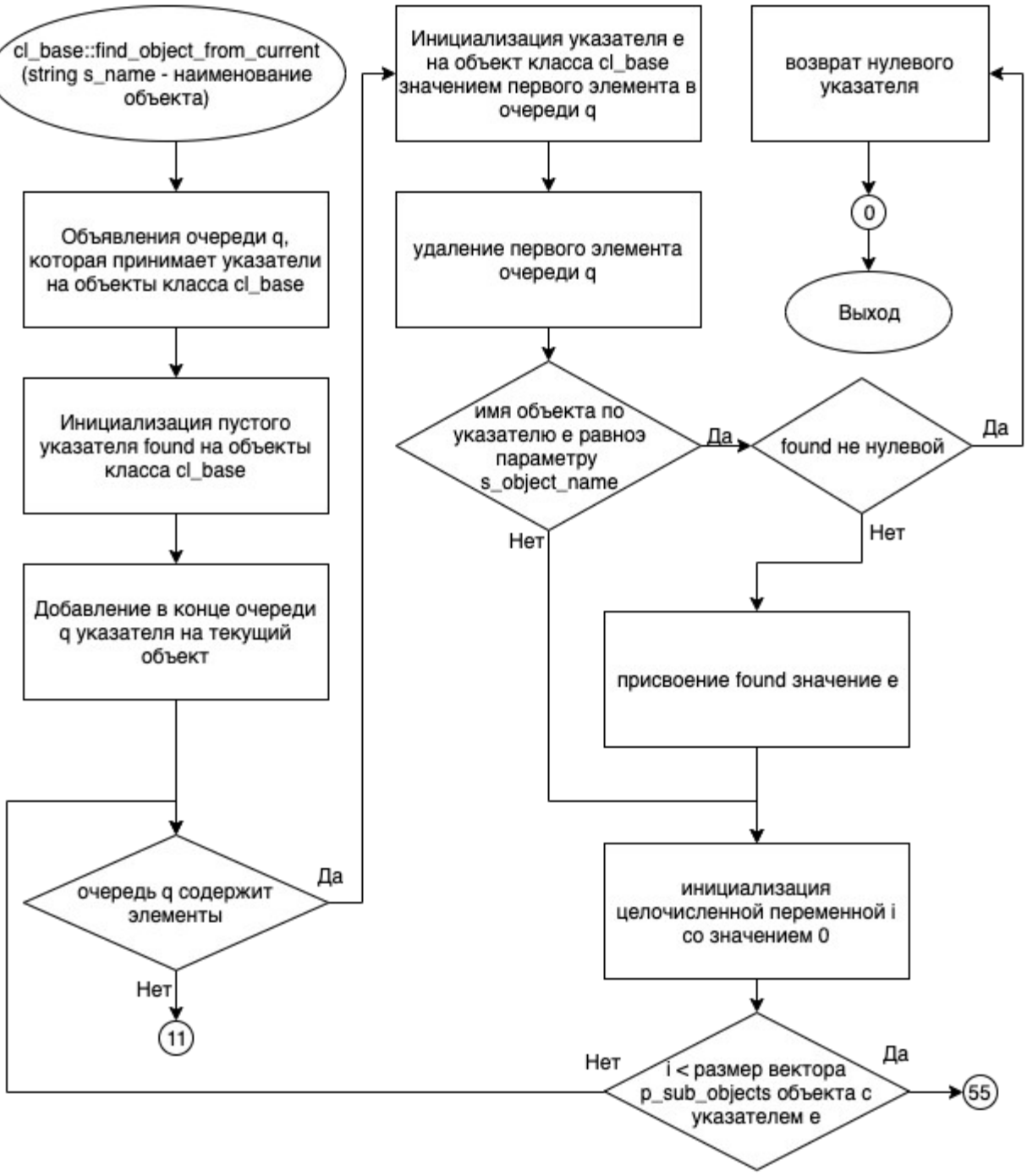

Рисунок 1 – Блок-схема алгоритма

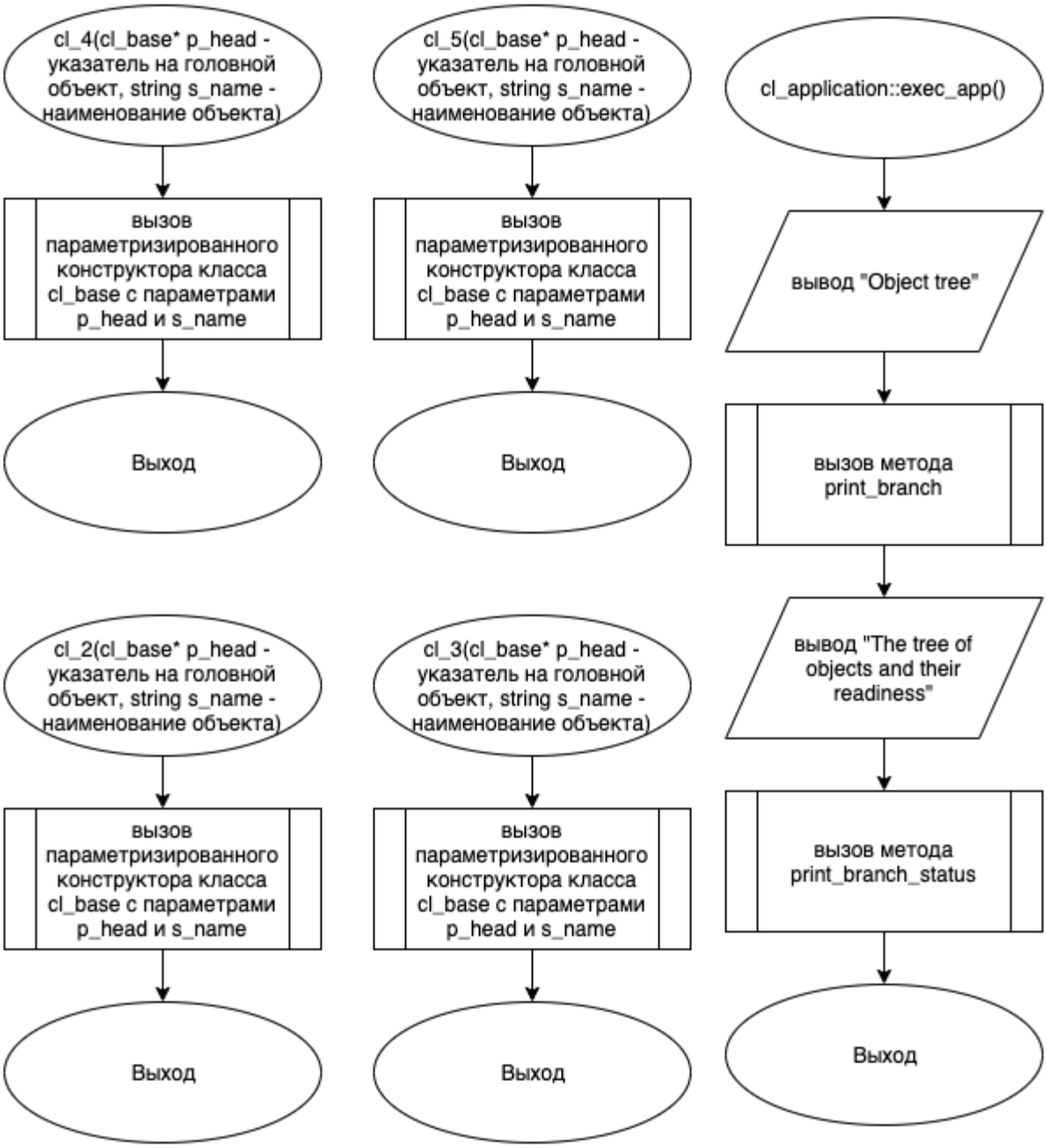

Рисунок 2 – Блок-схема алгоритма

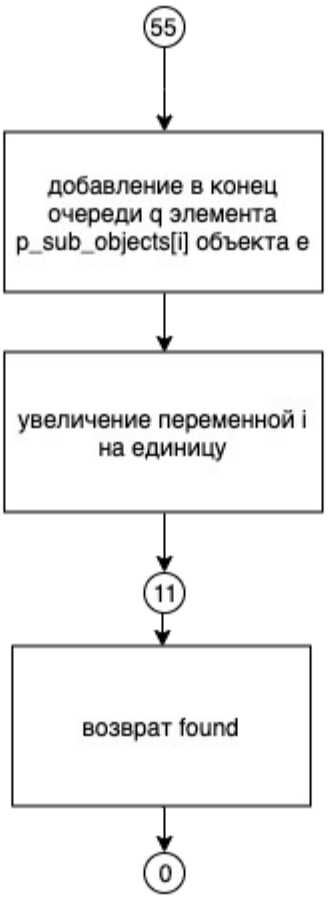

Рисунок 3 – Блок-схема алгоритма

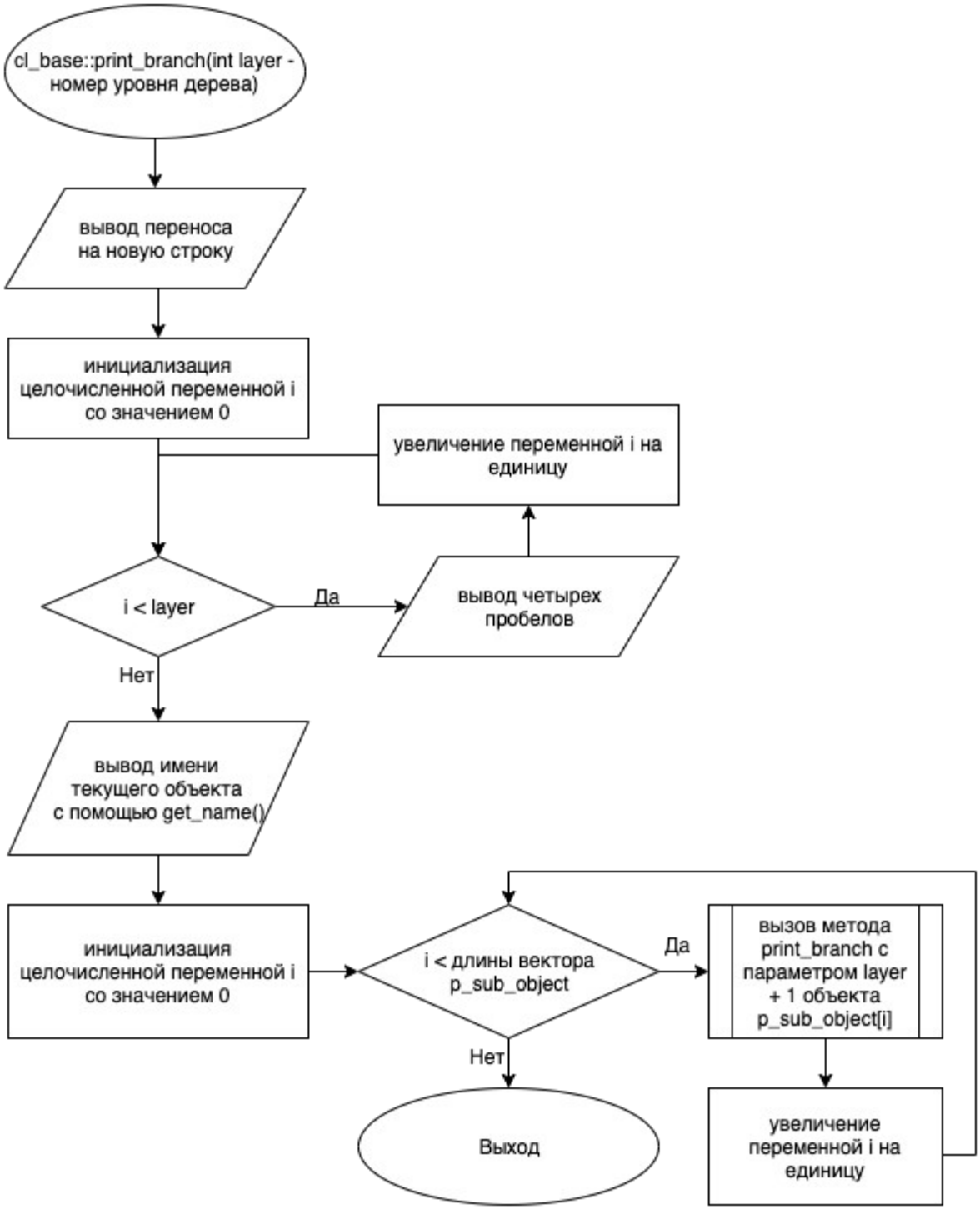

Рисунок 4 – Блок-схема алгоритма

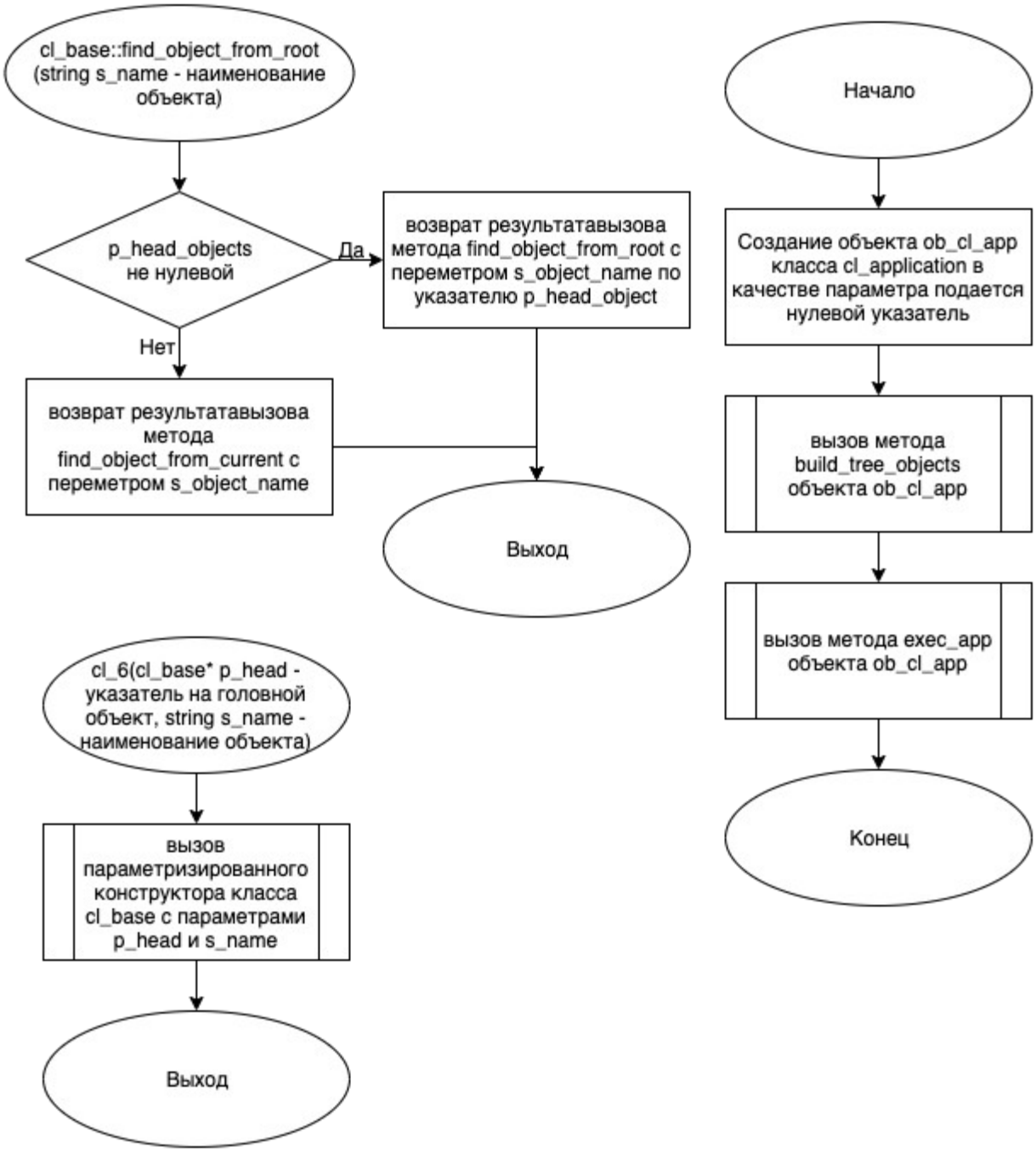

Рисунок 5 – Блок-схема алгоритма

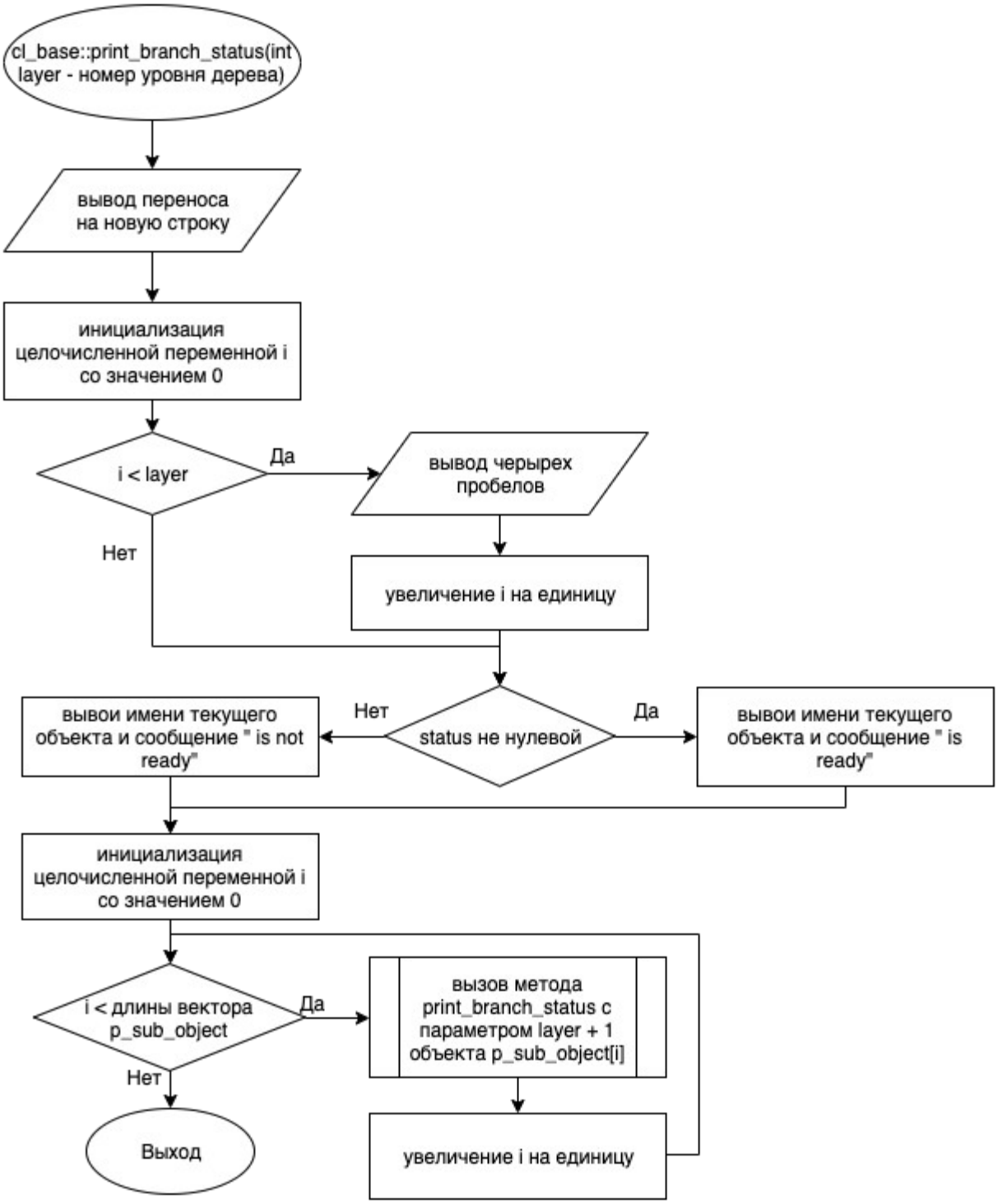

Рисунок 6 – Блок-схема алгоритма

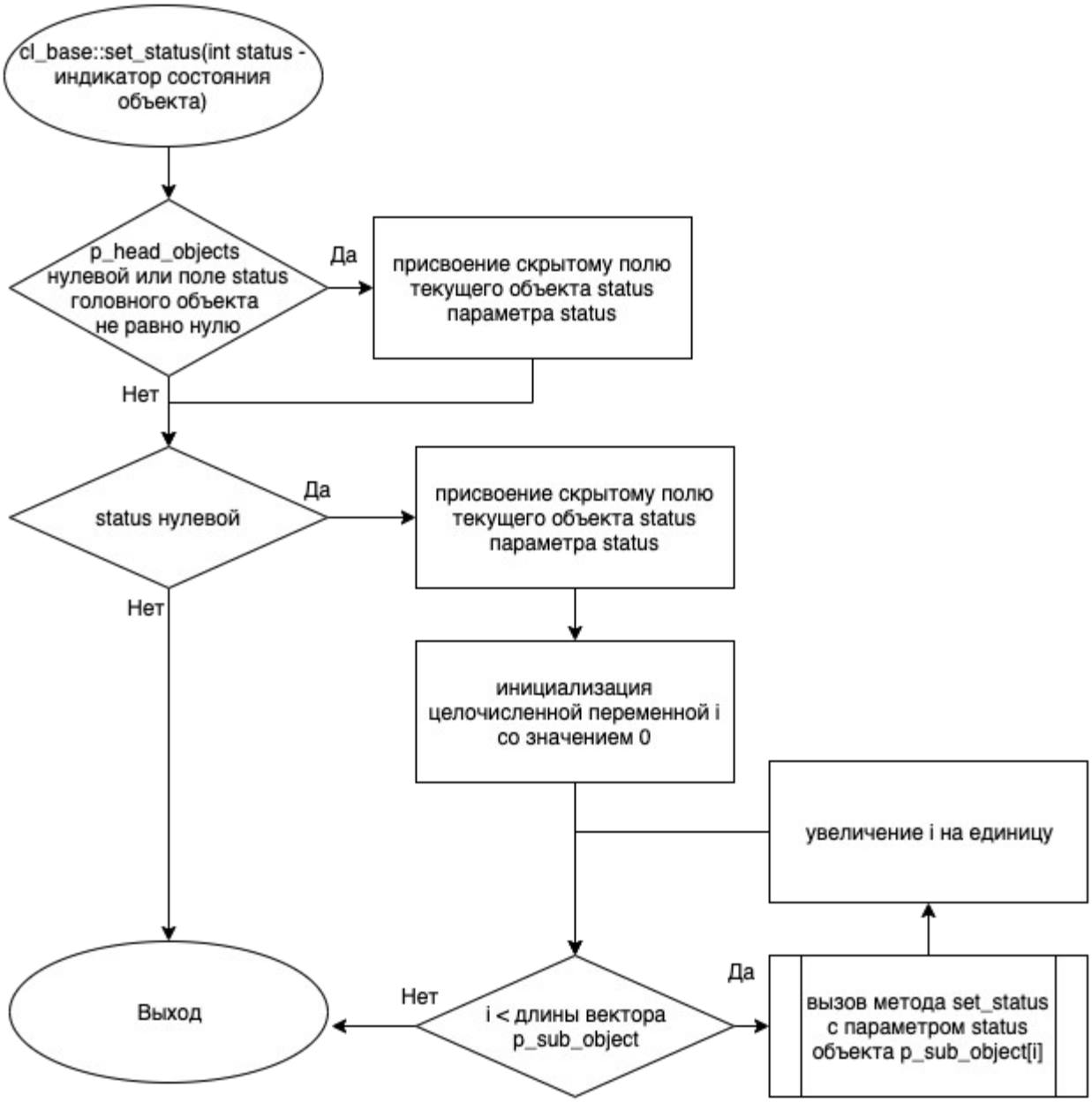

Рисунок 7 – Блок-схема алгоритма

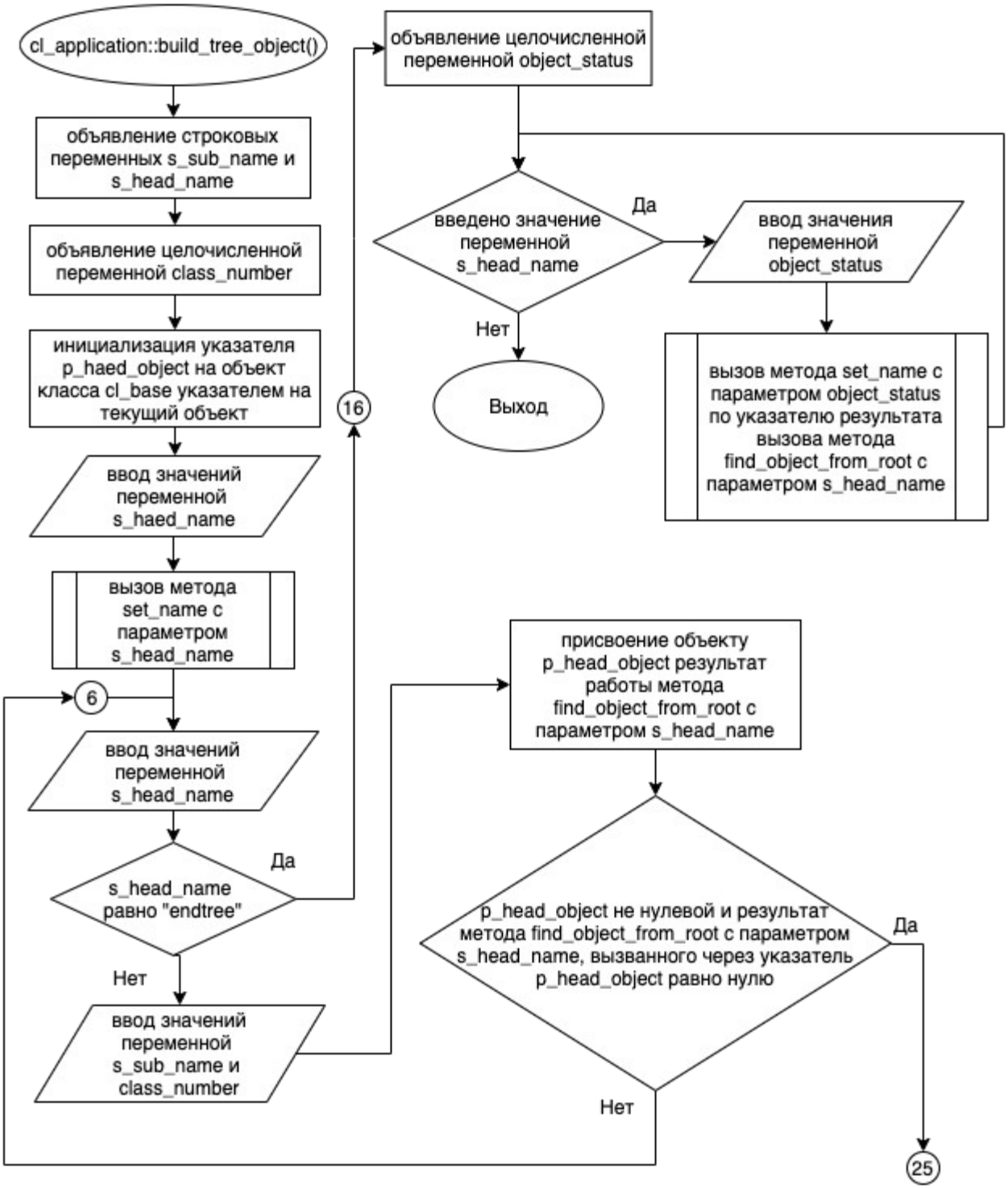

Рисунок 8 – Блок-схема алгоритма

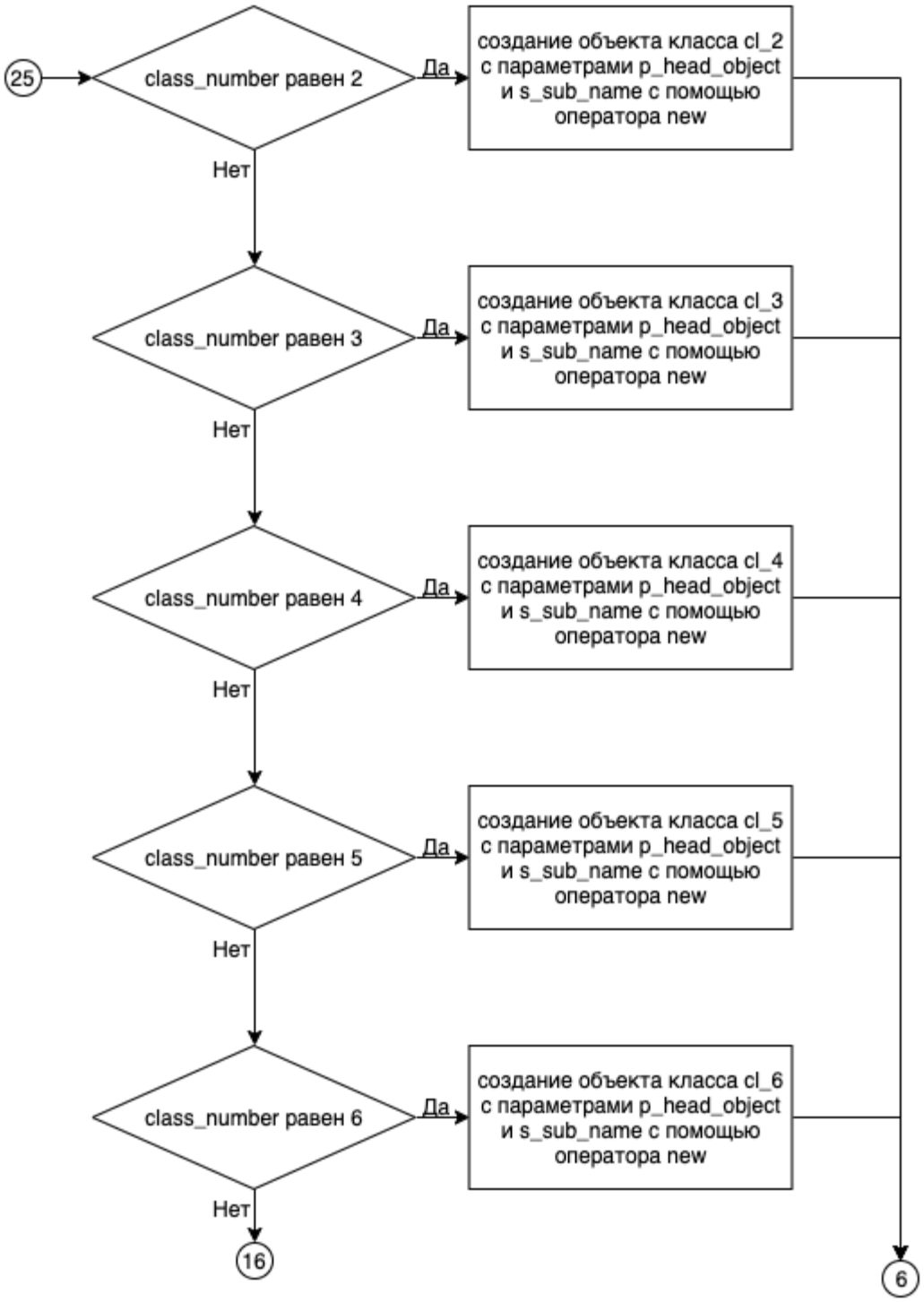

Рисунок 9 – Блок-схема алгоритма

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"

cl_2::cl_2(cl_base* p_head, string s_name) : cl_base(p_head, s_name){}
```

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, string s_name);
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* p_head, string s_name) : cl_base(p_head, s_name){}
```

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, string s_name);
};



#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* p_head, string s_name) : cl_base(p_head, s_name){}
```

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, string s_name);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* p_head, string s_name) : cl_base(p_head, s_name){}
```

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, 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, string s_name) : cl_base(p_head, s_name){}
```

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, string s_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"

cl_application::cl_application(cl_base * p_head) : cl_base(p_head){}

void cl_application::build_tree_objects() {
   string s_head_name, s_sub_name;
   int class_number;
   cl_base* p_head = this;
   cin >> s_head_name;
   set_name(s_head_name);

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

      p_head = find_object_from_root(s_head_name);
      if (p_head != nullptr && p_head -> find_object_from_root(s_sub_name) ==
nullptr)
      {
         switch(class_number)
         {
            case 2:
               new cl_2(p_head, s_sub_name);
               break;
            case 3:
               new cl_3(p_head, s_sub_name);
```

```
                  break;
               case 4:
                  new cl_4(p_head, s_sub_name);
                  break;
               case 5:
                  new cl_5(p_head, s_sub_name);
                  break;
               case 6:
                  new cl_6(p_head, s_sub_name);
                  break;
               default: break;
            }
         }
      }
      int object_status;
      while(cin >> s_head_name)
      {
         cin >> object_status;
         find_object_from_root(s_head_name) -> set_status(object_status);
      }
   }

   int cl_application::exec_app()
   {
      cout << "Object tree";
      print_branch();
      cout << "\nThe tree of objects and their readiness";
      print_branch_status();
      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"

   class cl_application : public cl_base
   {
      public:
         cl_application(cl_base* p_head_object);
         void build_tree_objects();
         int exec_app();
   };
   #endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base * p_head, string s_name)
{
   this -> p_head_object = p_head;
   this -> s_object_name = s_name;
   if (p_head_object != nullptr)
   {
      p_head_object -> p_sub_objects.push_back(this);
   }
}

cl_base::~cl_base(){
   for (int i = 0; i < p_sub_objects.size(); i++){
      delete p_sub_objects[i];
   }
}

bool cl_base::set_name(string s_new_name){
   if (p_head_object != nullptr)
   {
      for (int i = 0; i < p_head_object -> p_sub_objects.size(); i++)
      {
         if (p_head_object -> p_sub_objects[i] -> get_name() == s_new_name)
         {
            return false;
         }
      }
   }
   this -> s_object_name = s_new_name;
   return true;
}

string cl_base::get_name(){
   return this -> s_object_name;
}

cl_base* cl_base::get_head()
{
   return this -> p_head_object;
}

cl_base* cl_base::get_sub_object(string s_name)
{
   for (int i = 0; i < p_sub_objects.size(); i++)
   {
      if (p_sub_objects[i] -> get_name() == s_name)
      {
         return p_sub_objects[i];
      }
```

```
   }
   return nullptr;
}

cl_base* cl_base::find_object_from_current(string s_name)
{
   queue<cl_base*> q;
   cl_base* found = nullptr;
   q.push(this);
   while (!q.empty())
   {
      cl_base* e = q.front();
      if (e -> get_name() == s_name)
      {
         if (found != nullptr)
            return nullptr;
         else
            found = e;
      }
      for (int i = 0; i < e -> p_sub_objects.size(); i++)
      q.push(e -> p_sub_objects[i]);
      q.pop();
   }
   return found;
}

cl_base* cl_base::find_object_from_root(string s_name)
{
   if (p_head_object != nullptr)
   {
      return p_head_object -> find_object_from_root(s_name);
   }
   else
   {
      return find_object_from_current(s_name);
   }
}

void cl_base::print_branch(int layer)
{
   cout << "\n";
   for (int i = 0; i < layer; i++)
   {
      cout << "    ";
   }
   cout << this -> get_name();
   for (int i = 0; i < p_sub_objects.size(); i++)
      p_sub_objects[i] -> print_branch(layer + 1);
}

void cl_base::print_branch_status(int layer)
{
   cout << "\n";
   for (int i = 0; i < layer; i++)
   {
```

```
            cout << "    ";
        }
        if (this -> status != 0)
        {
            cout << this -> get_name() << " is ready";
        }
        else
        {
            cout << this -> get_name() << " is not ready";
        }
        for (int i = 0; i < p_sub_objects.size(); i++)
            p_sub_objects[i] -> print_branch_status(layer + 1);
    }

    void cl_base::set_status(int status)
    {
        if (p_head_object == nullptr || p_head_object -> status != 0)
            this -> status = status;
        if (status == 0)
        {
            this -> status = status;
            for (int i = 0; i < p_sub_objects.size(); i++)
                p_sub_objects[i] -> set_status(status);
        }
    }
```

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>
    #include <queue>

    using namespace std;

    class cl_base {
        public:
            cl_base(cl_base* p_head, string s_name = "Base object");
            ~cl_base();
            bool set_name(string s_new_name);
            string get_name();
            cl_base* get_head();
            cl_base* get_sub_object(string s_name);
            cl_base* find_object_from_current(string s_name);
            cl_base* find_object_from_root(string s_name);
            void print_branch(int layer = 0);
```

```
        void print_branch_status(int layer = 0);
        void set_status(int status);
    private:
        int status = 0;
        string s_object_name;
        cl_base* p_head_object;
        vector <cl_base*> p_sub_objects;
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

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

int main()
{
   cl_application obj_cl_app(nullptr);
   obj_cl_app.build_tree_objects();
   return (obj_cl_app.exec_app());
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 15.

Таблица 15 – 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).