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_application класса cl_application предназначен для построения дерева и запуска приложения;
- cin/cout - операторы ввода/вывода, if - условный оператор, for - цикл со счетчиком, while - цикл с условием.

Класс cl_base:

- свойства/поля:
  - поле индикатор состояния объекта:
    - наименование — status;
    - тип — int;
    - модификатор доступа — private;
- функционал:
  - метод search_object_from_current — метод поиска объекта на ветке дереве иерархии от текущего по имени;
  - метод search_object_from_tree — метод поиск объекта по имене во всем дереве;
  - метод print_tree — метод вывода иерархии объектов (дерева/ветки) от текущего объекта;
  - метод print_status_tree — метод вывода дерева/ветки иерархии объектов и их статуса от текущего объекта;
  - метод set_status — метод установки статуса объекта.

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_objects — метод, создающий иерархию объекта;
    - 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	
		cl_2	public		2
		cl_3	public		3
		cl_4	public		4
		cl_5	public		5
		cl_6	public		6
		cl_application	public		7
2	cl_2			класс объекта древа	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
3	cl_3			класс объекта древа	
4	cl_4			класс объекта древа	
5	cl_5			класс объекта древа	
6	cl_6			класс объекта древа	
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 search_object_from_current класса cl_base

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

Параметры: string s_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода search_object_from_current класса cl_base

№	Предикат	Действия	№ перехода
1		объявление очереди q, которая принимает указатели на объекты класса cl_base	2
2		инициализация пустого указателя p_found на объекты класса cl_base	3
3		добавление в конец очереди q указателя на текущий объект	4
4	очередь q не пустая	инициализация указателя p_front 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 p_front равно параметру		7

№	Предикат	Действия	№ перехода
	s_name		
			8
7	p_found ненулевой	возврат нулевого указателя	∅
		присваивание p_found значение p_front	8
8		инициализация i со значением 0	9
9	i < размер вектора subordinate_objects объекта с указателем p_front	добавление в конец очереди q элемента subordinate_objects[i] объекта p_front	10
			4
10		i++	11
11		возврат p_found	∅

3.2 Алгоритм метода search_object_from_tree класса cl_base

Функционал: метод поиск объекта по имене во всем дереве.

Параметры: string s_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода search_object_from_tree класса cl_base

№	Предикат	Действия	№ перехода
1	вышестоящий объект p_root в дереве не пустой	p_root присваивается имя головного объекта	1
			2
2		возврат результат вызова метода search_object_from_current с параметром s_name	∅

3.3 Алгоритм метода print_tree класса cl_base

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

Параметры: int layer.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1		вывод переноса строки	2
2		инициализация i со значением 0	3
3	i< layer	вывод четырех пробелов	4
			5
4		i++	5
5		вывод имени текущего объекта с помощью get_name()	6
6		инициализация i со значением 0	7
7	i< длины вектора subordinate_objects	вызов метода print_tree с параметром layer+1 объекта subordinate_objects[i]	8
			∅
8		i++	7

3.4 Алгоритм метода print_status_tree класса cl_base

Функционал: метод вывода дерева/ветки иерархии объектов и их статуса от текущего объекта.

Параметры: int layer.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода print_status_tree класса cl_base

№	Предикат	Действия	№ перехода
1		вывод переноса строки	2
2		инициализация i со значением 0	3
3	i < layer	вывод четырех пробелов	4
			5
4		i++	5
5	status ненулевой	вывод имени текущего объекта и " is ready"	6
		вывод имени текущего объекта и " is not ready"	6
6		инициализация i со значением 0	7
7	i < размер вектора subordinate_objects	вызов метода print_status_tree с параметром layer+1 объекта subordinate_objects[i]	8
			∅
8		i++	7

3.5 Алгоритм метода set_status класса cl_base

Функционал: метод установки статуса объекта.

Параметры: int status.

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 родительского объекта != 0	присвоение скрытому полю текущего объекта status параметра status	2
			2
2	status == 0	присвоение скрытому полю текущего объекта status параметра status	3

№	Предикат	Действия	№ перехода
			∅
3		инициализация i = 0	4
4	i < размер вектора subordinate_objects	вызов метода set_status с параметром status объекта subordinate_objects[i]	5
			∅
5		i++	4

3.6 Алгоритм конструктора класса cl_2

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

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_objects и s_object_name	∅

3.7 Алгоритм конструктора класса cl_3

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

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_objects и s_object_name	∅

3.8 Алгоритм конструктора класса cl_4

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

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_objects и s_object_name	∅

3.9 Алгоритм конструктора класса cl_5

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

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_objects и s_object_name	∅

3.10 Алгоритм конструктора класса cl_6

Функционал: параметризированный конструктор.

Параметры: cl_base* p_head_objects, string s_object_name.

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_objects и s_object_name	∅

3.11 Алгоритм метода build_tree_objects класса cl_application

Функционал: метод, создающий иерархию объекта.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		объявление строковых переменных s_head_name и s_sub_name	2
2		объявление int class_number, object_state	3
3		инициализация указателя p_head на объект класса cl_base указателем на текущий объект	4
4		ввод значения s_head_name	5
5		вызов метода set_name с параметром s_head_name текущего объекта	6
6		ввод значений s_head_name	7
7	s_head_name равно "endtree"		16
			8
8		ввод значение s_sub_name и class_number	9
9		присвоение объекту p_head результат работы метода search_object_from_tree с параметром s_head_name	10

№	Предикат	Действия	№ перехода
10	p_head ненулевой и результат метода search_object_from_tree с параметром s_sub_name не равен нулевому указателю		11
			6
11	class_number равен 2	создание объекта класса cl_2 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			12
12	class_number равен 3	создание объекта класса cl_3 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			13
13	class_number равен 4	создание объекта класса cl_4 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			14
14	class_number равен 5	создание объекта класса cl_5 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			15
15	class_number равен 6	создание объекта класса cl_6 с параметрами p_head_objects и s_sub_name с помощью оператора new	6
			16
16	введено значение s_sub_name	ввод значение object_status	17
			∅

№	Предикат	Действия	№ перехода
17		присвоение p_sub результат вызова метода search_object_from_tree с параметром s_sub_name	18
18	p_sub ненулевой	вызов метода set_status с параметром object_state по указателю p_sub	16
			16

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
2		вызов метода print_tree	3
3		вывод "The tree of objects and their readiness"	4
4		вызов метода print_status_tree	∅

3.13 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 14.

Таблица 14 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_cl_applicatiob класса cl_application в качестве параметра подается нулевой указатель	2
2		вызываем метод build_tree_objects объекта ob_cl_application	3
3		вызываем метода exec_app объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-10.

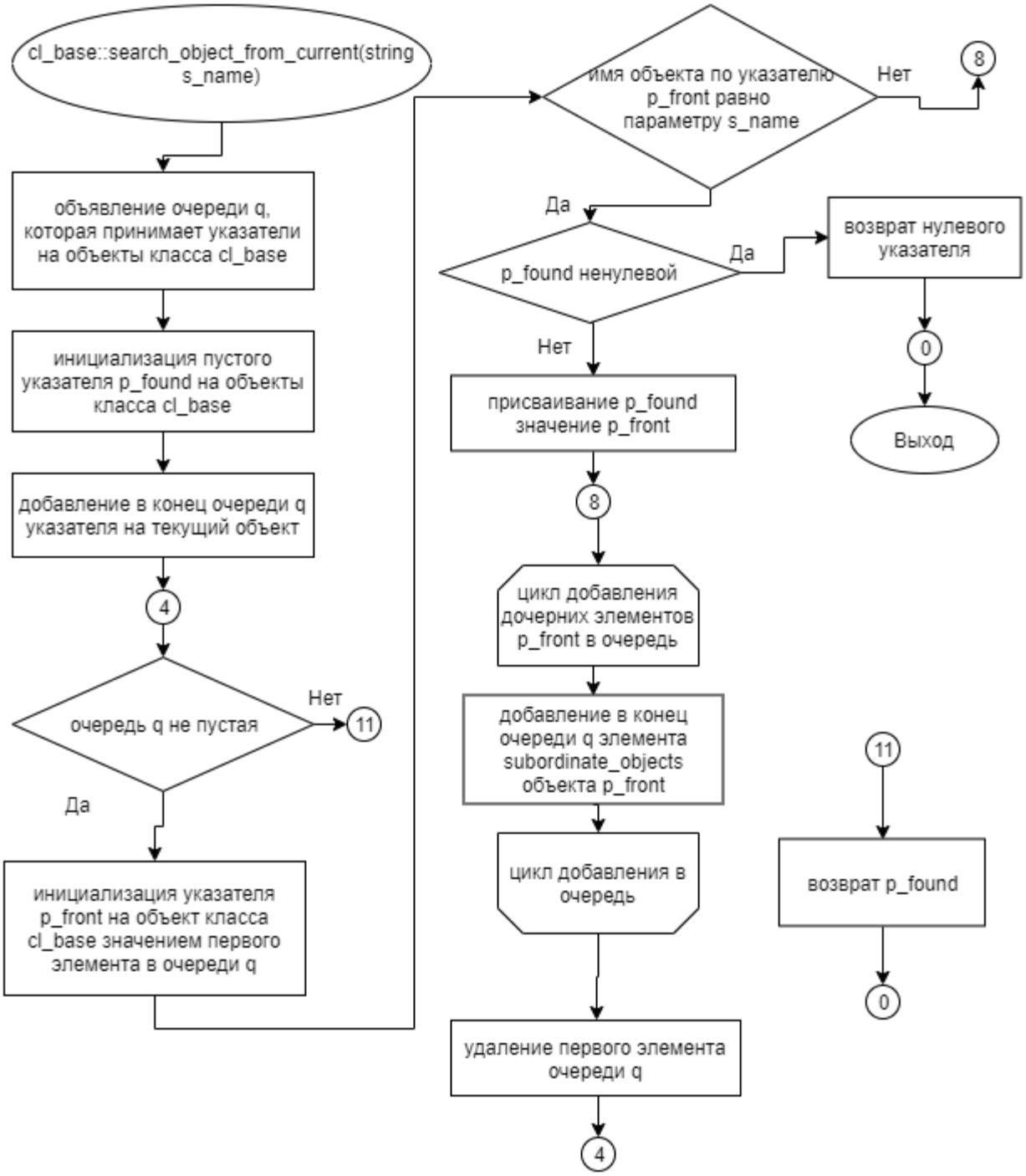

Рисунок 1 – Блок-схема алгоритма

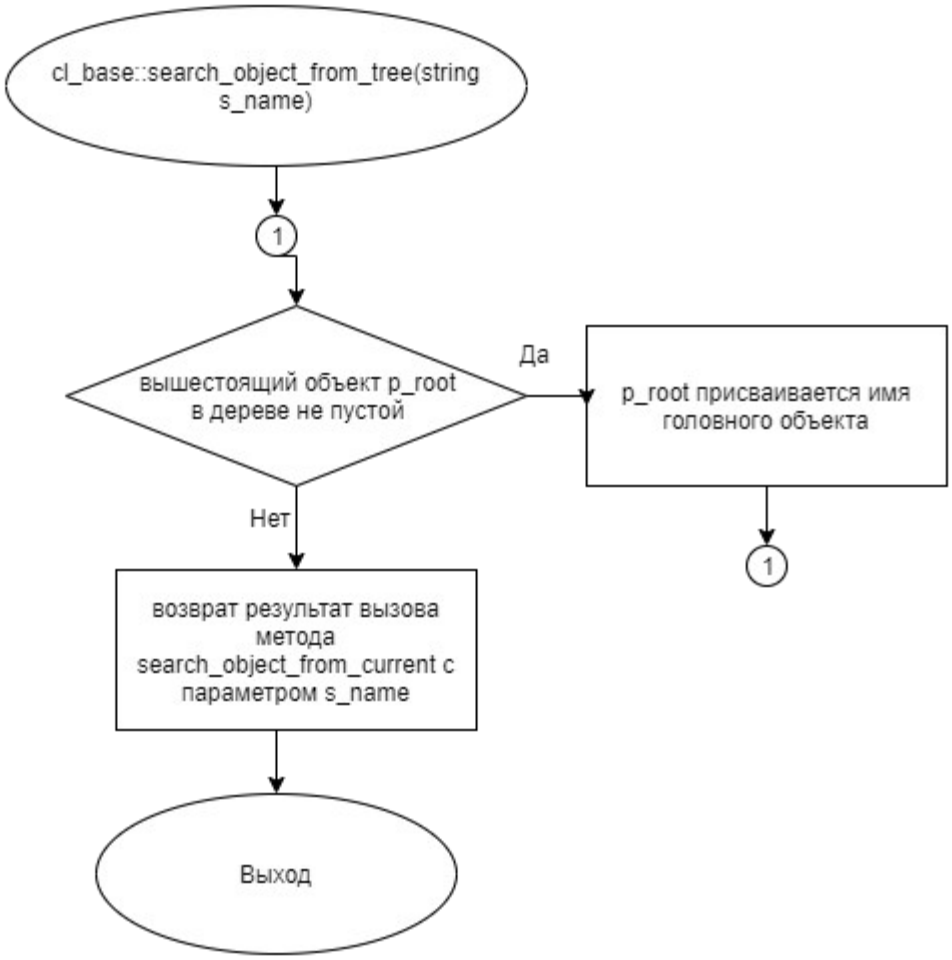

Рисунок 2 – Блок-схема алгоритма

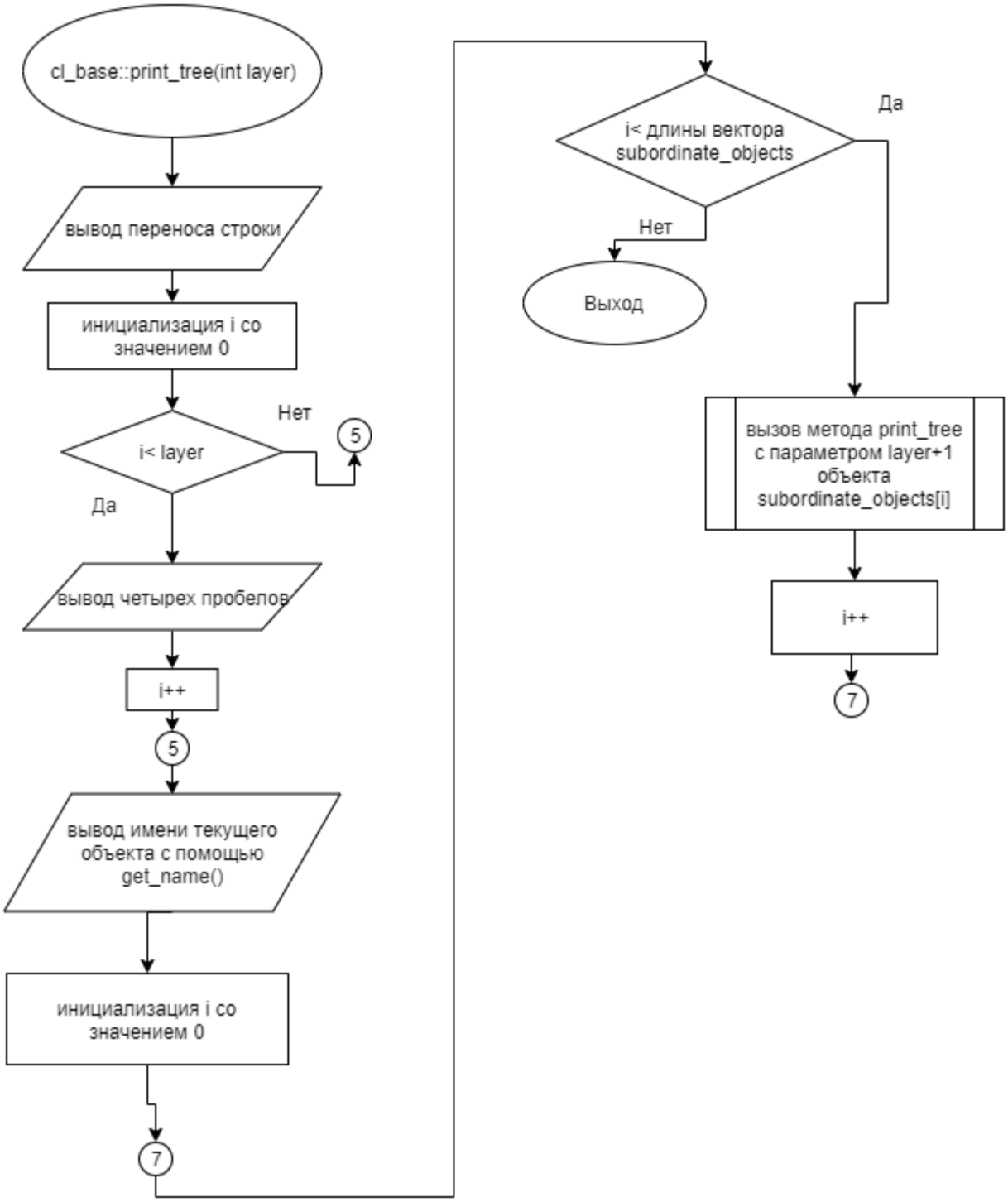

Рисунок 3 – Блок-схема алгоритма

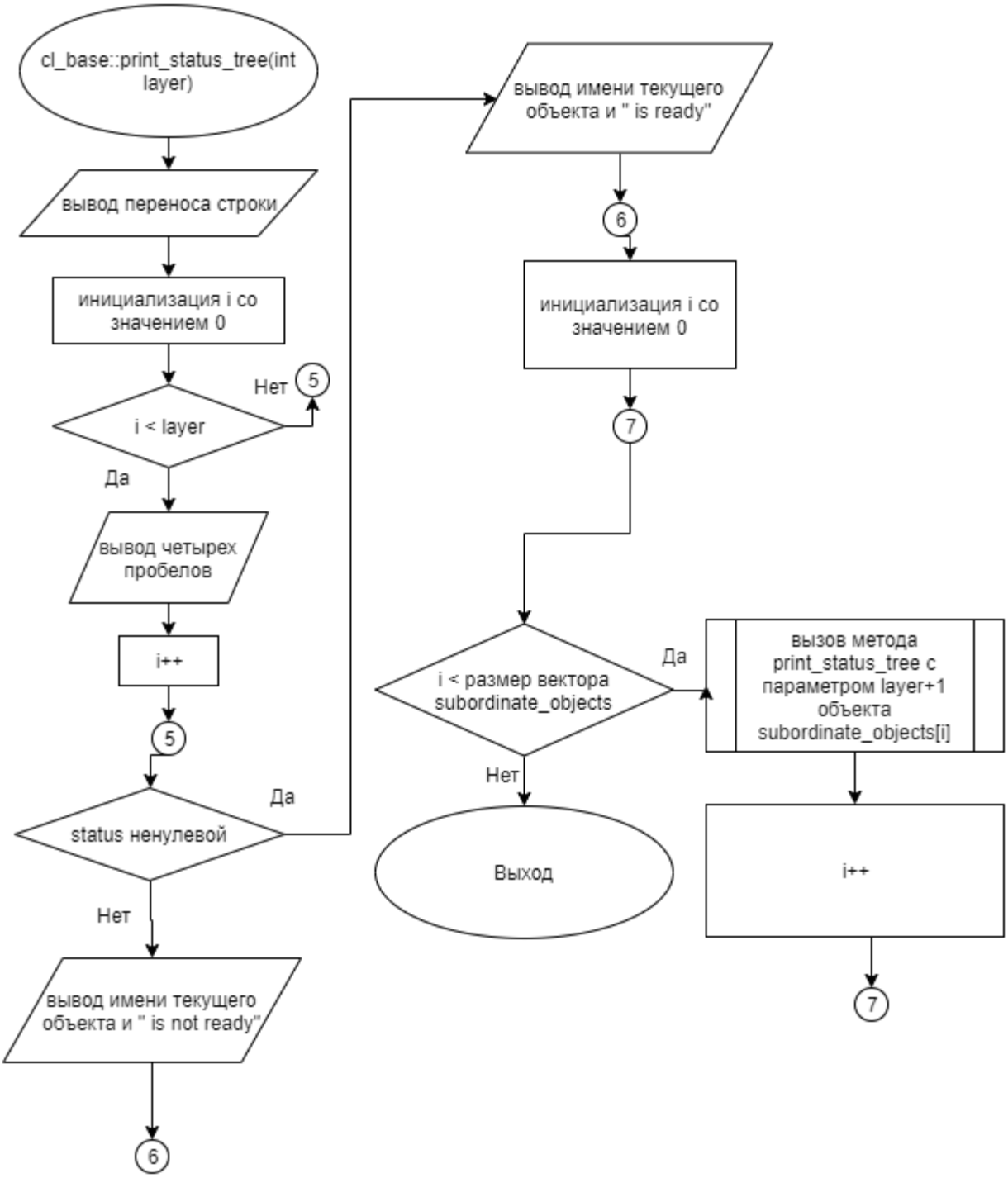

Рисунок 4 – Блок-схема алгоритма

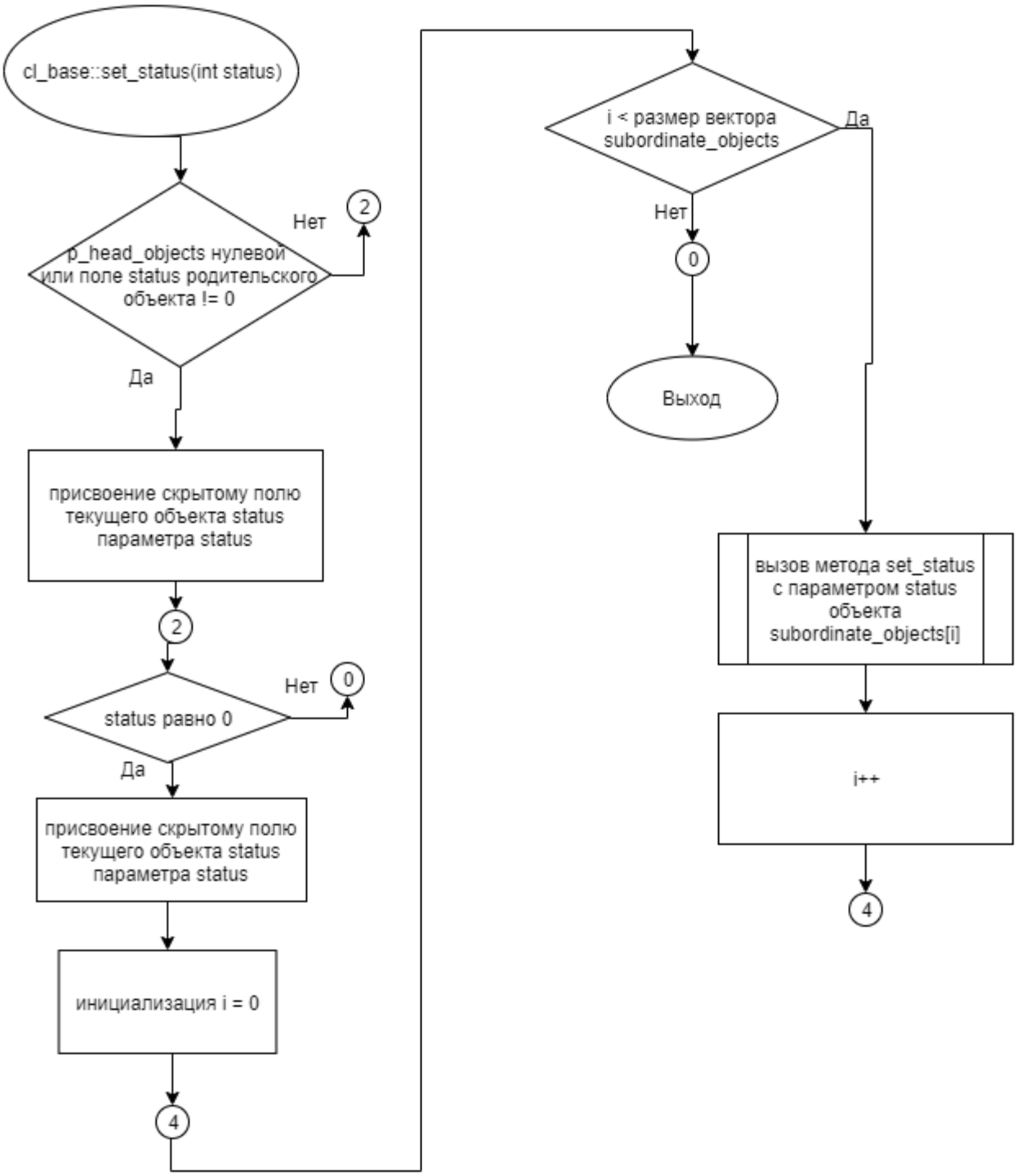

Рисунок 5 – Блок-схема алгоритма

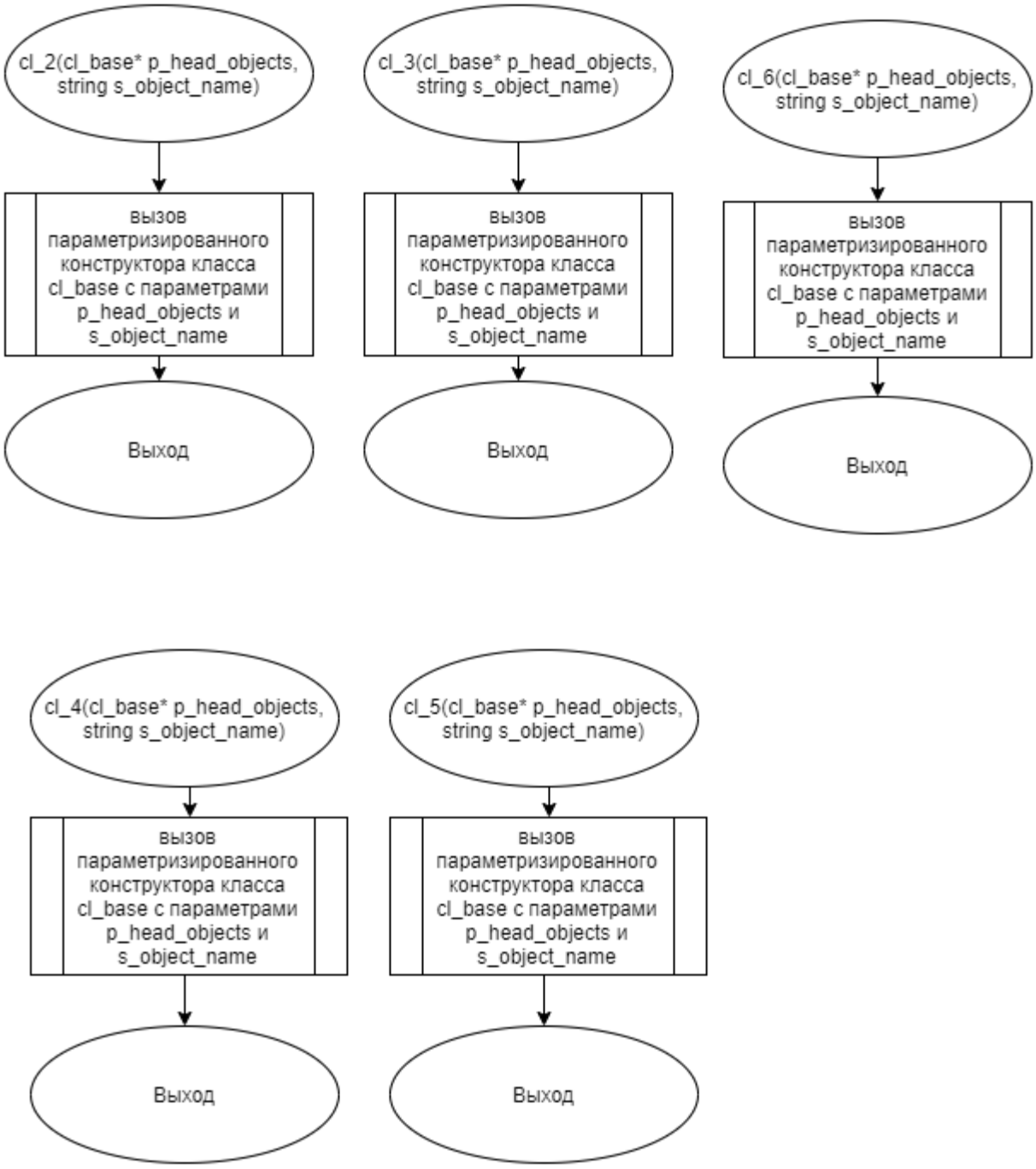

Рисунок 6 – Блок-схема алгоритма

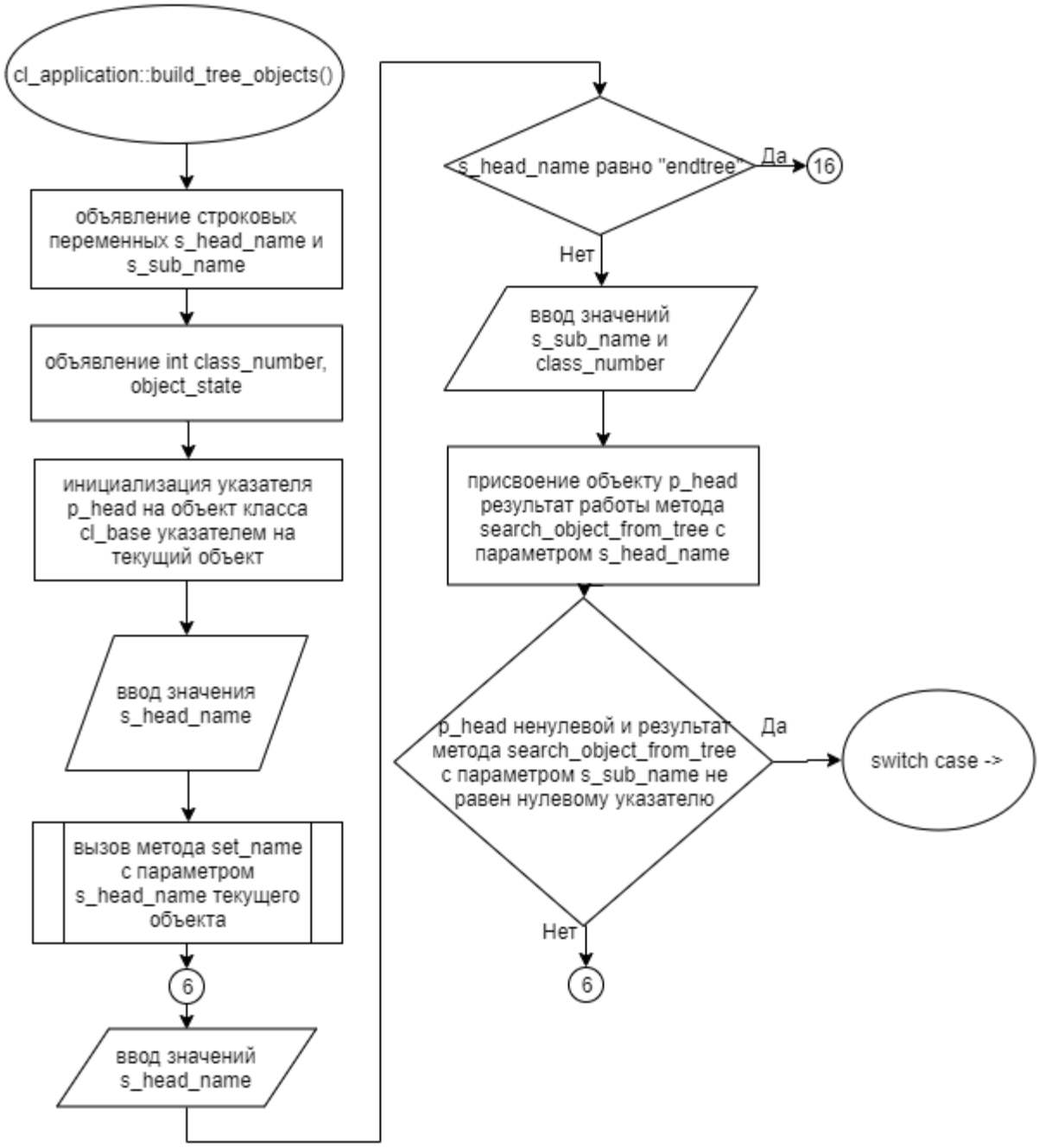

Рисунок 7 – Блок-схема алгоритма

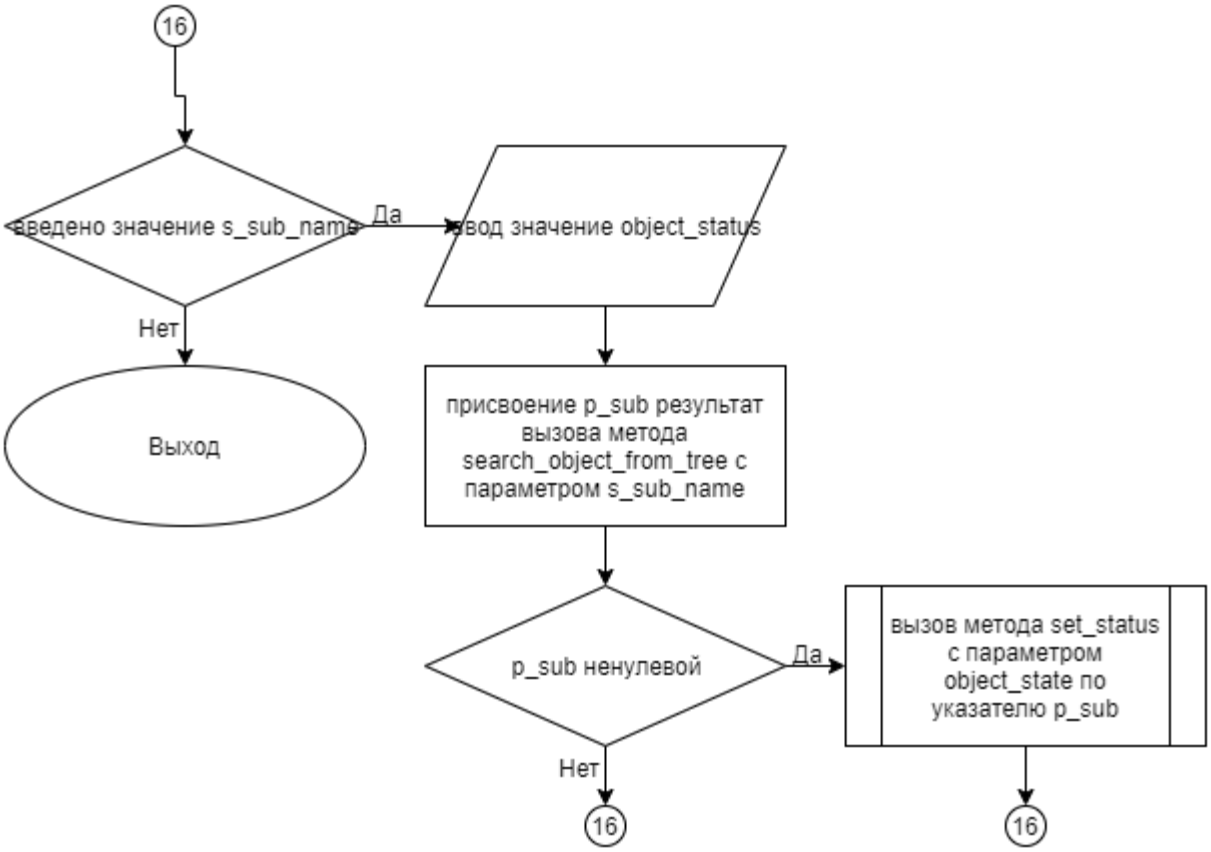

Рисунок 8 – Блок-схема алгоритма

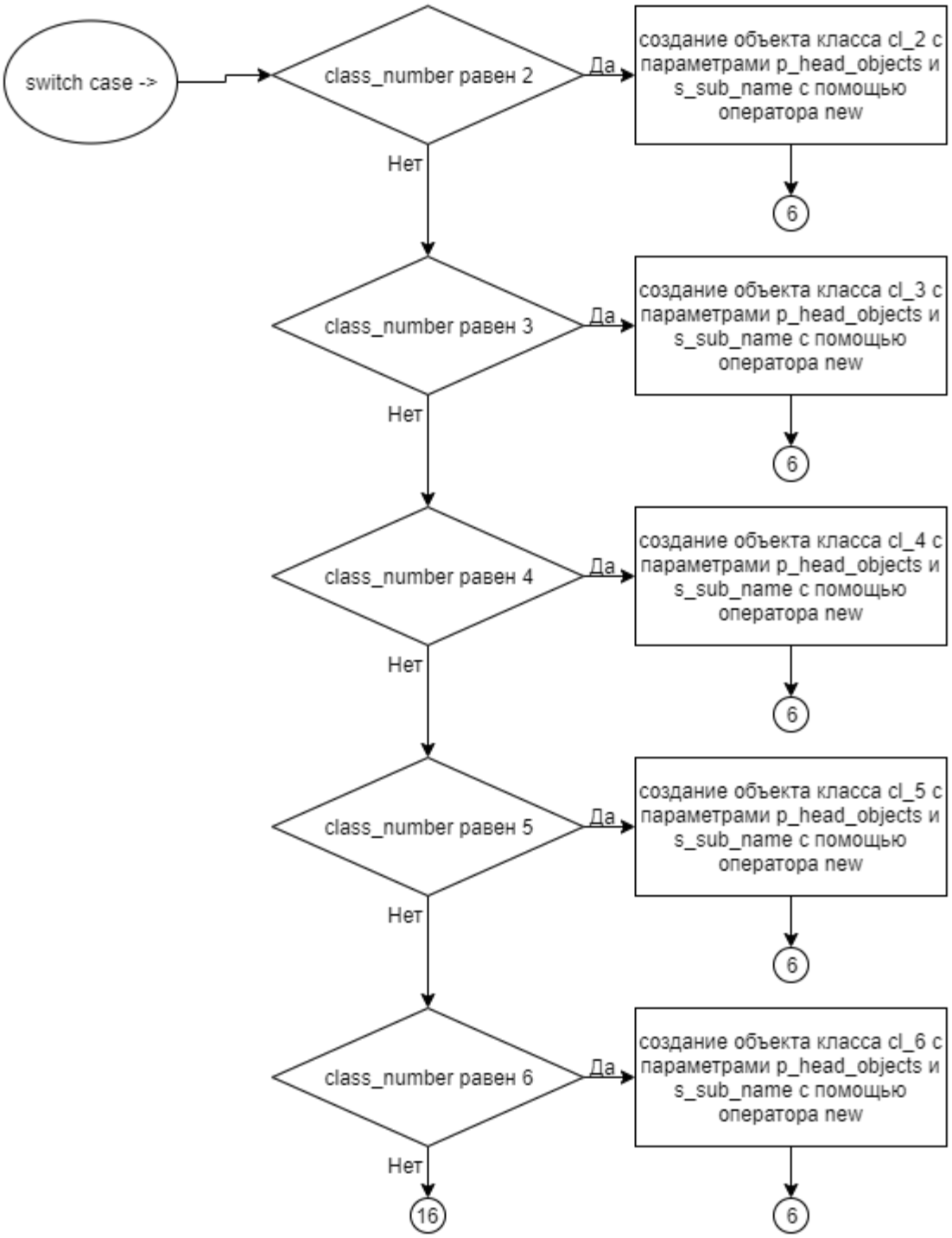

Рисунок 9 – Блок-схема алгоритма

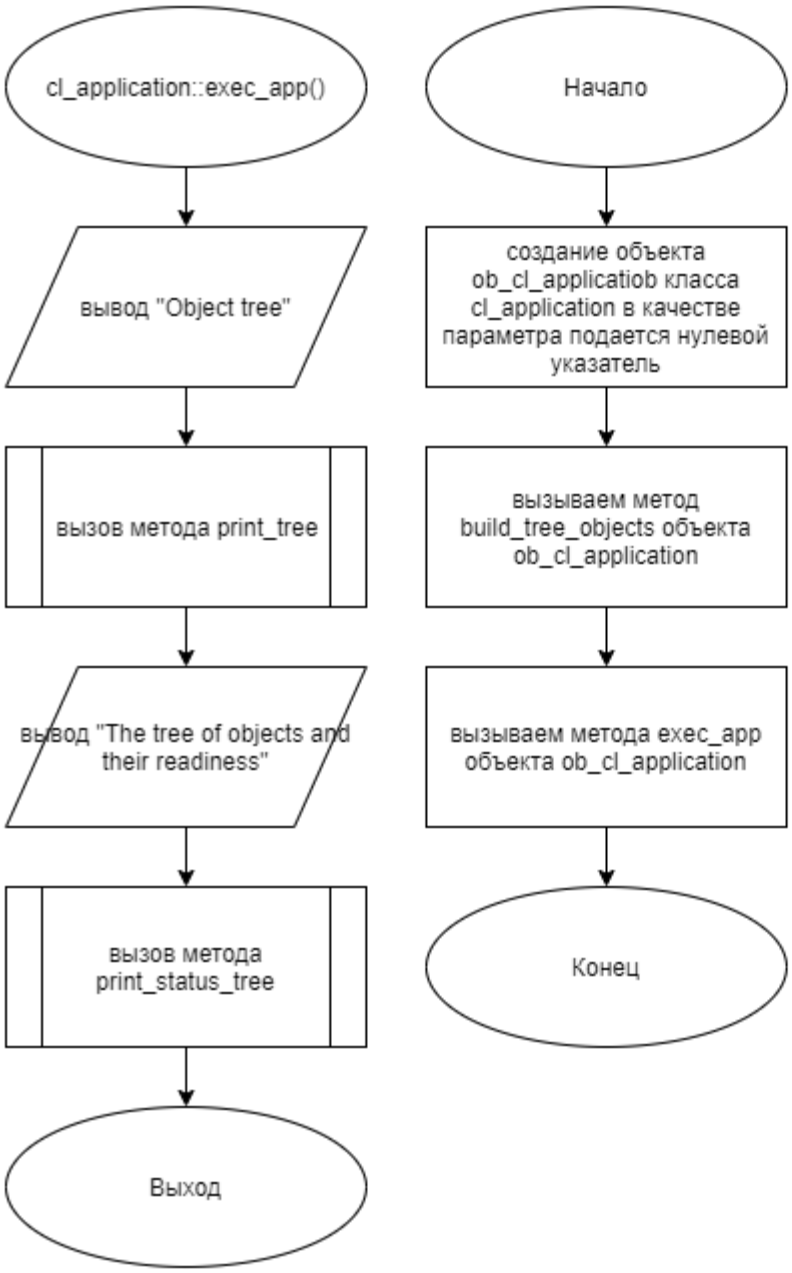

Рисунок 10 – Блок-схема алгоритма

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"

// вызов параметризированного конструктора класса cl_base с параметрами p_head_object и s_object_name
cl_2::cl_2(cl_base*      p_head_object,      string      s_object_name)      : cl_base(p_head_object, s_object_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:
   // конструктор, создающий объект класса cl_2
   cl_2(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
```

```
// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_3::cl_3(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_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:
   // конструктор, создающий объект класса cl_3
   cl_3(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_4::cl_4(cl_base*      p_head_object,      string      s_object_name)      :
cl_base(p_head_object, s_object_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:
   // конструктор, создающий объект класса cl_4
   cl_4(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_5::cl_5(cl_base*     p_head_object,     string     s_object_name)     :
cl_base(p_head_object, s_object_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:
   // конструктор, создающий объект класса cl_5
   cl_5(cl_base * p_head_object, string s_object_name);
```

```
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

// вызов параметризированного конструктора класса cl_base с параметрами
p_head_object и s_object_name
cl_6::cl_6(cl_base*     p_head_object,     string     s_object_name)     :
cl_base(p_head_object, s_object_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:
   // конструктор, создающий объект класса cl_6
   cl_6(cl_base * p_head_object, string s_object_name);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"

// вызов параметризированного конструктора класса cl_base с параметром
```

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


void cl_application::build_tree_objects(){
   /*
   метод построения дерева иерархии объектов
   */

   string s_head_name, s_sub_name; // имя головного объекта, подчиненного
объекта
   cl_base* p_head;
   cl_base* p_sub = nullptr;
   int class_number, object_state; // номер класса, номер состояния

   cin >> s_head_name;
   // вызов метода set_name этого объекта с параметром s_head_name
   this -> set_name(s_head_name);
   // присваиваем p_head этот объект
   p_head = this;

   // ввод иерархии объектов
   while(true){
      cin >> s_head_name;

      if (s_head_name == "endtree")
         break;

      cin >> s_sub_name >> class_number;

      //поиск головного объекта
      p_head = search_object_from_tree(s_head_name);
      // если p_head ненулевой и есть ли s_sub_name в поддереве не равный
nullptr
      if (p_head == nullptr && search_object_from_tree(s_sub_name) !=
nullptr)
         continue; // продолжаем новую итерацию

      // далее - существует единсвенный головной объект p_head и подчиненного
с таким именем не существует
      switch(class_number){
         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;
         }
      }
      // установка состояний готовности для объектов
      while (cin >> s_sub_name){
         cin >> object_state;
         p_sub = search_object_from_tree(s_sub_name);

         if (p_sub != nullptr)
            p_sub->set_status(object_state); // устанавливаем вводимое состояние
p_sub
      }
}

int cl_application::exec_app(){
   cout << "Object tree";
   //вызов метода print_tree этого объекта
   this -> print_tree();
   cout << endl << "The tree of objects and their readiness";
   this -> print_status_tree();
   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"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"


class cl_application : public cl_base{
public:
   cl_application(cl_base * p_head_object); // конструктор, создающий объект
класса cl_app..
   int exec_app(); // метод запуска приложения

   void build_tree_objects(); // метод, создающий иерархию объекта
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base * p_head_object, string s_object_name){
   /*
   параметризированный конструктор
   p_head_object - указатель на головной объект
   s_object_name - имя узла дерева
   */

   //полю p_head_object этого объекта присваивается параметр p_head_object
   //полю s_object_name этого объекта присваивается параметр s_object_name
   this -> p_head_object = p_head_object;
   this -> s_object_name = s_object_name;

   // если p_head_object ненулевой, то в subordinate_objects добавляется
указатель на этот объект
   if (p_head_object){
      p_head_object -> subordinate_objects.push_back(this);
   }
}
cl_base::~cl_base(){
   // проходимся по каждому элементу subordinate_objects и удаляем его
   for  (int i =0; i < subordinate_objects.size(); i++){
      delete subordinate_objects[i];
   }
}

bool cl_base::set_name(string new_name){
   /*
   метод редактирования имени объекта
   new_name - новое имя узла дерева
   */

   // проходимся по i-ому указателю вектора указателей subordinate_objects
объекта по указателю p_head_object
   // если он равен new_name, то возвращаем false
   if (p_head_object != nullptr){
      for (int i = 0; i < p_head_object -> subordinate_objects.size(); i++){
         if (p_head_object -> subordinate_objects[i] -> get_name() ==
new_name){
            return false;
         }
      }
   }
   // полю s_object_name этого объекта присваивается new_name
   this -> s_object_name = new_name;
   return true;
}

string cl_base::get_name(){
```

```
   // возвращаем s_object_name
   return this->s_object_name;
}

cl_base * cl_base::get_p_head(){
   // возвращаем p_head_object
   return this->p_head_object;
}

cl_base * cl_base::get_subordinate_object(string search_name){
   /*
   получение указателя на непосредственно подчиненный объект по имени
   search_name - имя искомого объекта
   */

   // проходимся по i-ому указателю вектора указателей subordinate_objects,
если он равен search_name
   // возвращаем i-ый subordinate_objects
   for (int i = 0; i < subordinate_objects.size(); i++){
      if (subordinate_objects[i] -> get_name() == search_name){
         return subordinate_objects[i];
      }
   }
   // иначе возвращается нулевой указатель
   return nullptr;
}



cl_base* cl_base::search_object_from_current(string s_name){
   /*
   метод поиска объекта по имене в поддереве (в ветке) (обход графа в ширину)
   s_name - имя искомого объекта
   */
   cl_base* p_found = nullptr; // указатель на объект, который был найден
   queue<cl_base*> q; // очередь элементов

   q.push(this); // добавляем в очередь текущий элемент

   // пока очередь не пустая
   while(!q.empty()){
      cl_base* p_front = q.front(); // хранится указатель на наш объект


      if (p_front -> get_name() == s_name){ // если имя указателя на элемент
в очереди совпадает с имкомым объектом
         if (p_found    ==  nullptr)  //  указатель  не  пустой  и  объект  не
уникальный
            p_found = p_front; // присваиваем указатель на элемент в очереди
         else // нашли дубликат
            return nullptr;
      }
      // добавляем дочерние элементы p_front в очередь
      for (auto p_sub : p_front->subordinate_objects){
         q.push(p_sub);
```

```
      }
      q.pop(); // удаляем элемент из начала очереди, чтобы пройтись по всем
элементам
   }
   return p_found;
}

cl_base* cl_base::search_object_from_tree(string s_name){
   /*
   поиск объекта по имене во всем дереве
   s_name - имя искомого объекта
   */

   cl_base* p_root = this;
   // пока вышестоящий объект в дереве не пустой, поднимаемся по дереву
   while (p_root-> get_p_head() != nullptr){
      p_root = p_root-> get_p_head();
   }

   return p_root -> search_object_from_current(s_name);
}
void cl_base::print_tree(int layer){
   /*
   метод вывода иерархии объектов (дерева/ветки) от текущего объекта
   layer - уровень на дереве иерархии
   */
   cout << endl;
   for(int i =0; i < layer;i++){
      cout << "    ";
   }
   // выводим дерево объекта
   cout << this -> get_name();
   if (subordinate_objects.size() != 0){
      // проходимся по элементам subordinate_objects
      for (int i = 0; i < subordinate_objects.size(); i++){
         // создается рекурсия, если у i-ого subordinate_objects есть еще
какие-то привязанные объекты
         subordinate_objects[i] -> print_tree(layer + 1);
      }
   }

}

void cl_base::print_status_tree(int layer){
   /*
   метод вывода дерева/ветки иерархии объектов и их статуса от текущего
объекта
   layer - уровень на дереве иерархии
   */
   cout << endl;
   for(int i =0; i < layer; ++i){
      cout << "    ";
   }
   // проверка на статус текущего объекта
   if (this-> status!=0){
```

```
            cout << this -> get_name() << " is ready";
        }
        else{
            cout << this -> get_name() << " is not ready";
        }
        for (int i = 0; i < subordinate_objects.size(); ++i){
            subordinate_objects[i] -> print_status_tree(layer + 1);
        }
    }
    void cl_base::set_status(int status){
        /*
        метод установки статуса объекта
        status - номер состояния
        */

        // если значение status ненулевое у головного объекта
        if (p_head_object == nullptr || p_head_object -> status != 0){
            this -> status = status;
        }
        if (status == 0){
            this ->status = status;
            for (int i = 0; i < subordinate_objects.size(); i++){
                subordinate_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 <string>
    #include <vector>
    #include <iostream>
    #include <queue>

    using namespace std;

    class cl_base {
    private:
        string s_object_name; // имя объекта
        cl_base * p_head_object; // указатель на родительский объект
        vector <cl_base *> subordinate_objects; // вектор подчиненных объектов
        int status = 0; // статус состояния объекта
    public:
        cl_base(cl_base * p_head_object, string s_object_name = "Base object"); //
```

```
параметризированный конструктор
   string get_name(); // метод получения имени
   cl_base * get_p_head(); // метод получения указателя на родительский
объект
   bool set_name(string new_name); //метод редактирования имени объекта
   cl_base * get_subordinate_object(string search_name); // получение
указателя на непосредственно подчиненный объект по имени
   ~cl_base(); // деструктор
   cl_base* search_object_from_current(string); // поиск объекта на ветке
иерархии от текущего по имени
   cl_base* search_object_from_tree(string); // поиск по дереву (в корне)
иерархии
   void set_status(int status); // метод установки статуса объекта
   void print_tree(int layer = 0); // метод вывода дерева иерархии
   void print_status_tree(int layer = 0); // метод вывода дерева/ветки
иерархии объектов и их статуса от текущего объекта
};



#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#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 иерархии

   return ob_cl_application.exec_app(); // запуск системы
}
```

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
app_root app_root object_01 3 app_root object_02 2 app_root object_03 5 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 object_03 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_03 is not ready	Object tree app_root object_01 object_07 object_02 object_04 object_05 object_03 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_03 is not ready

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
app_root app_root object_01 3 app_root object_02 2 app_root object_03 5 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 app_root 0	Object tree app_root object_01 object_07 object_02 object_04 object_05 object_03 The tree of objects and their readiness app_root is not ready object_01 is not ready object_07 is not ready object_02 is not ready object_04 is not ready object_05 is not ready object_03 is not ready	Object tree app_root object_01 object_07 object_02 object_04 object_05 object_03 The tree of objects and their readiness app_root is not ready object_01 is not ready object_07 is not ready object_02 is not ready object_04 is not ready object_05 is not ready object_03 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).