Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

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

Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:

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

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется. При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

1. Построить дерево объектов системы (в методе 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

Класс cl_base:

- Свойства/поля:
  - Поле состояния объекта:
    - Наименование - state;
    - Тип - int;
    - Модификатор доступа - private;
- Функционал:
  - Метод findObjOnBranch выполняет поиск объекта на ветке дереве иерархии от текущего по имени;
  - Метод findObjOnTree выполняет поиск объекта на дереве иерархии по имени;
  - Метод printBranch выполняет вывод иерархии объектов (дерева или ветки) от текущего объекта;
  - Метод printBranchWithState выполняет вывод иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;
  - Метод setState устанавливает готовность объекта;

Класс cl_2:

- Функционал:
  - Параметризированный конструктор cl_2, который в качестве параметров принимает указатель на головной объект в дереве иерархии и возможное наименование текущего объекта;

Класс cl_3:

- Функционал:
  - Параметризированный конструктор cl_3, который в качестве параметров принимает указатель на головной объект в дереве

иерархии и возможное наименование текущего объекта;

Класс cl_4:

- Функционал:
  - Параметризированный конструктор cl_4, который в качестве параметров принимает указатель на головной объект в дереве иерархии и возможное наименование текущего объекта;

Класс cl_5:

- Функционал:
  - Параметризированный конструктор cl_5, который в качестве параметров принимает указатель на головной объект в дереве иерархии и возможное наименование текущего объекта;

Класс cl_6:

- Функционал:
  - Параметризированный конструктор cl_6, который в качестве параметров принимает указатель на головной объект в дереве иерархии и возможное наименование текущего объекта;

Используются объекты классов cl_node, cl_2, cl_3, cl_4, cl_5, cl_6. Их количество зависит от ввода пользователя;

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификаторы доступа при наследовании	Описание	Номер	Комментарий
1	cl_base			Базовый класс в иерархии		

				классов. Содержит основные свойства и методы		
		application			2	
		cl_node			3	
		cl_2			4	
		cl_3			5	
		cl_4			6	
		cl_5			7	
		cl_6			8	
2	application			Класс корневого объекта (приложения)		
3	cl_node			Класс объекта дерева		
4	cl_2			Класс объекта дерева		
5	cl_3			Класс объекта дерева		
6	cl_4			Класс объекта дерева		

7	cl_5			Класс объекта дерева		
8	cl_6			Класс объекта дерева		

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 конструктора класса cl_2

Функционал: Создание объекта класса.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base с параметрами p_head_object и s_object_name	∅

3.2 Алгоритм конструктора класса cl_3

Функционал: Создание объекта класса.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base с параметрами p_head_object и s_object_name	∅

3.3 Алгоритм конструктора класса cl_4

Функционал: Создание объекта класса.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 4.

Таблица 4 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base с параметрами p_head_object и s_object_name	∅

3.4 Алгоритм конструктора класса cl_5

Функционал: Создание объекта класса.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base с параметрами p_head_object и s_object_name	∅

3.5 Алгоритм конструктора класса cl_6

Функционал: Создание объекта класса.

Параметры: cl_base* p_head_object, string s_object_name.

Алгоритм конструктора представлен в таблице 6.

Таблица 6 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base с параметрами p_head_object и s_object_name	∅

3.6 Алгоритм метода findObjOnBranch класса cl_base

Функционал: Поиск объекта по имени в ветке дерева иерархии объектов.

Параметры: string s_object_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода findObjOnBranch класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация пустого указателя found на объект класса cl_base	2
2		Объявление очереди elementsQueue, которая принимает указатели на объекты класса cl_base	3
3		Добавление в конец очереди elementsQueue указателя на текущий объект	4
4	Очередь elementsQueue содержит элементы	Инициализация указателя elem на объект класса cl_base значением первого элемента в очереди elementsQueue	5
			11
5		Удаление первого элемента очереди elementsQueue	6
6	Имя объекта по указателю elem равно параметру s_object_name		7
			8
7	found != nullptr	Возврат нулевого указателя	∅

№	Предикат	Действия	№ перехода
		found = elem	8
8		Инициализация целочисленной переменной i значением 0	9
9	i < размер вектора children объекта с указателем elem	Добавление в конец очереди elementsQueue элемента children[i] объекта elem	10
			4
10		i += 1	9
11		Возвратить found	∅

3.7 Алгоритм метода findObjOnTree класса cl_base

Функционал: Поиск объекта по имени во всём дереве иерархии объектов.

Параметры: string s_object_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода findObjOnTree класса cl_base

№	Предикат	Действия	№ перехода
1	parent != nullptr	Возврат результата вызова метода findObjOnTree с параметром s_object_name по указателю p_head_object	∅
		Возврат результата вызова метода findObjOnBranch с параметром s_object_name	∅

3.8 Алгоритм метода printBranch класса cl_base

Функционал: Вывод дерева иерархии объектов от текущего объекта.

Параметры: int level.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода printBranch класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод переноса на новую строку	2
2		Инициализация целочисленной переменной i значением 0	3
3	i < level	Вывод четырёх пробелов	4
			5
4		i += 1	3
5		Вывод имени текущего объекта	6
6		Инициализация целочисленной переменной i значением 0	7
7	i < длины вектора children	Вызов метода printBranch с параметром (level + 1) объекта children[i]	8
			∅
8		i += 1	7

3.9 Алгоритм метода printBranchWithState класса cl_base

Функционал: Вывод дерева иерархии объектов от текущего с параметром готовности.

Параметры: int level.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода printBranchWithState класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод переноса на новую строку	2
2		Инициализация целочисленной переменной i значением 0	3
3	i < level	Вывод четырёх пробелов	4
			5
4		i += 1	3
5	state != 0	Вывод имени текущего объекта и "is ready"	6
		Вывод имени текущего объекта и "is not ready"	6
6		i = 0	7
7	i < длины вектора children	Вызов метода printBranchWithState с параметром (level + 1) объекта children[i]	8
			∅
8		i += 1	7

3.10 Алгоритм метода setState класса cl_base

Функционал: Изменение значения поля state.

Параметры: int state.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода setState класса cl_base

№	Предикат	Действия	№ перехода
1	parent = nullptr или поле state головного объекта != 0	Присвоение скрытому полю текущего объекта state параметра state	2
			2
2	state = 0	Присвоение скрытому полю текущего объекта state параметра state	3

№	Предикат	Действия	№ перехода
			∅
3		Инициализация целочисленной переменной i значением 0	4
4	i < длины вектора children	Вызов метода setState с параметром state объекта children[i]	5
			∅
5		i += 1	4

3.11 Алгоритм метода exec_app класса application

Функционал: Запуск системы.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree"	2
2		Вызов метода printBranch	3
3		Вывод "The tree of objects and their rediness"	4
4		Вызов метода printBranchWithState	∅

3.12 Алгоритм метода build_tree_objects класса application

Функционал: Строит дерево иерархии.

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода build_tree_objects класса application

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных s_sub_name s_head_name	2
2		Объявление целочисленной переменной class_num	3
3		Инициализация указателя p_head_object на объект класса cl_base ссылкой на текущий объект	4
4		Ввод значения переменной s_head_name	5
5		Вызов метода setName с параметром s_head_name	6
6		Ввод значения переменной s_head_name	7
7	s_head_name == "endtree"		16
			8
8		Ввод значений переменных s_sub_name и class_num	9
9		Присвоение объекту p_head_object результат работы метода findObjOnTree с параметром s_head_name	10
10	p_head_object != nullptr и результат метода findObjOnTree с параметром s_sub_name, вызванного через указатель p_head_object = nullptr		11
			6
11	class_num == 2	Создание объекта класса cl_2 с параметрами p_head_object и s_sub_name с помощью оператора функции new	6
			12
12	class_num == 3	Создание объекта класса cl_3 с параметрами p_head_object и s_sub_name с помощью оператора функции new	6

№	Предикат	Действия	№ перехода
			13
13	class_num == 4	Создание объекта класса cl_4 с параметрами p_head_object и s_sub_name с помощью оператора функции new	6
			14
14	class_num == 5	Создание объекта класса cl_5 с параметрами p_head_object и s_sub_name с помощью оператора функции new	6
			15
15	class_num == 6	Создание объекта класса cl_6 с параметрами p_head_object и s_sub_name с помощью оператора функции new	6
			16
16		Объявление целочисленной переменной state_obj	17
17	Было введено значение переменной s_head_name	Ввод значения переменной state_obj	18
			∅
18		Вызов метода setName с параметром state_obj по указателю результата вызова метода findObjOnTree с параметром s_head_name	17

3.13 Алгоритм функции main

Функционал: Основная программа.

Параметры: .

Возвращаемое значение: int - код возврата.

Алгоритм функции представлен в таблице 14.

Таблица 14 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_application класса application с использованием параметрезированного конструктора и передачей в него в качествепараметра пустого указателя	2
2		Вызов метода build_tree_objects объекта ob_application	3
3		Возвращение результата работы метода exec_app() для объекта ob_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-7.

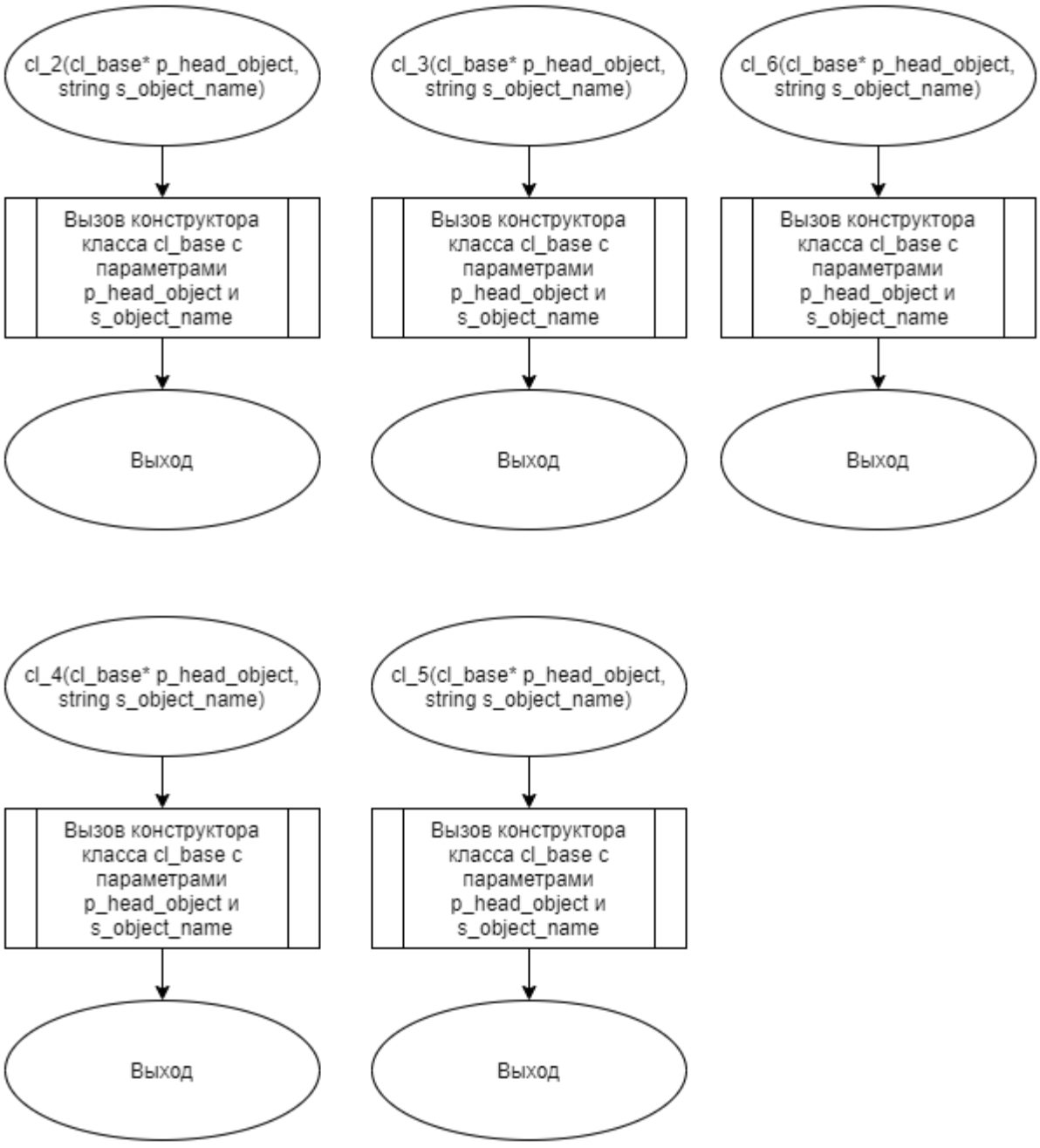

Рисунок 1 – Блок-схема алгоритма

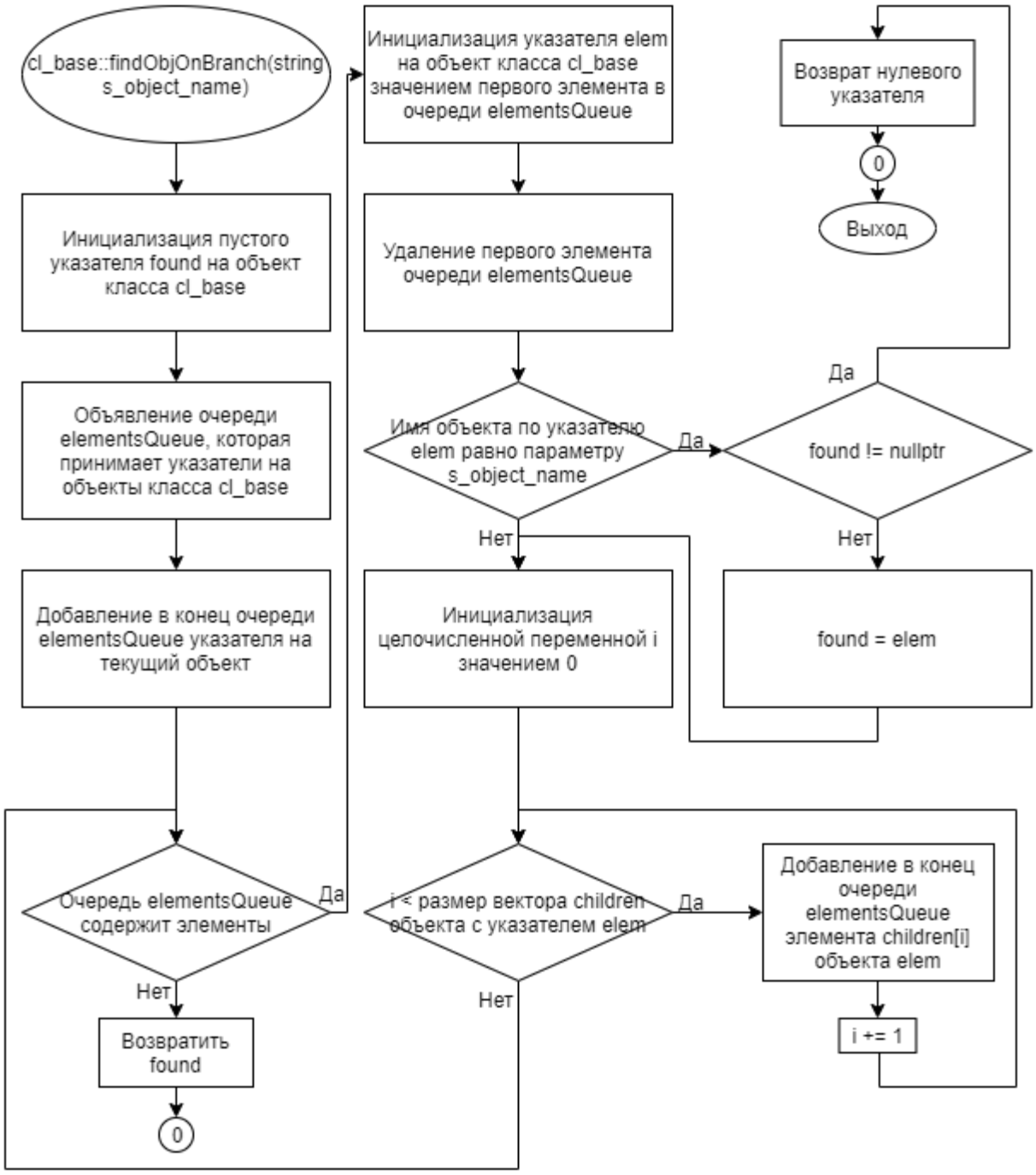

Рисунок 2 – Блок-схема алгоритма

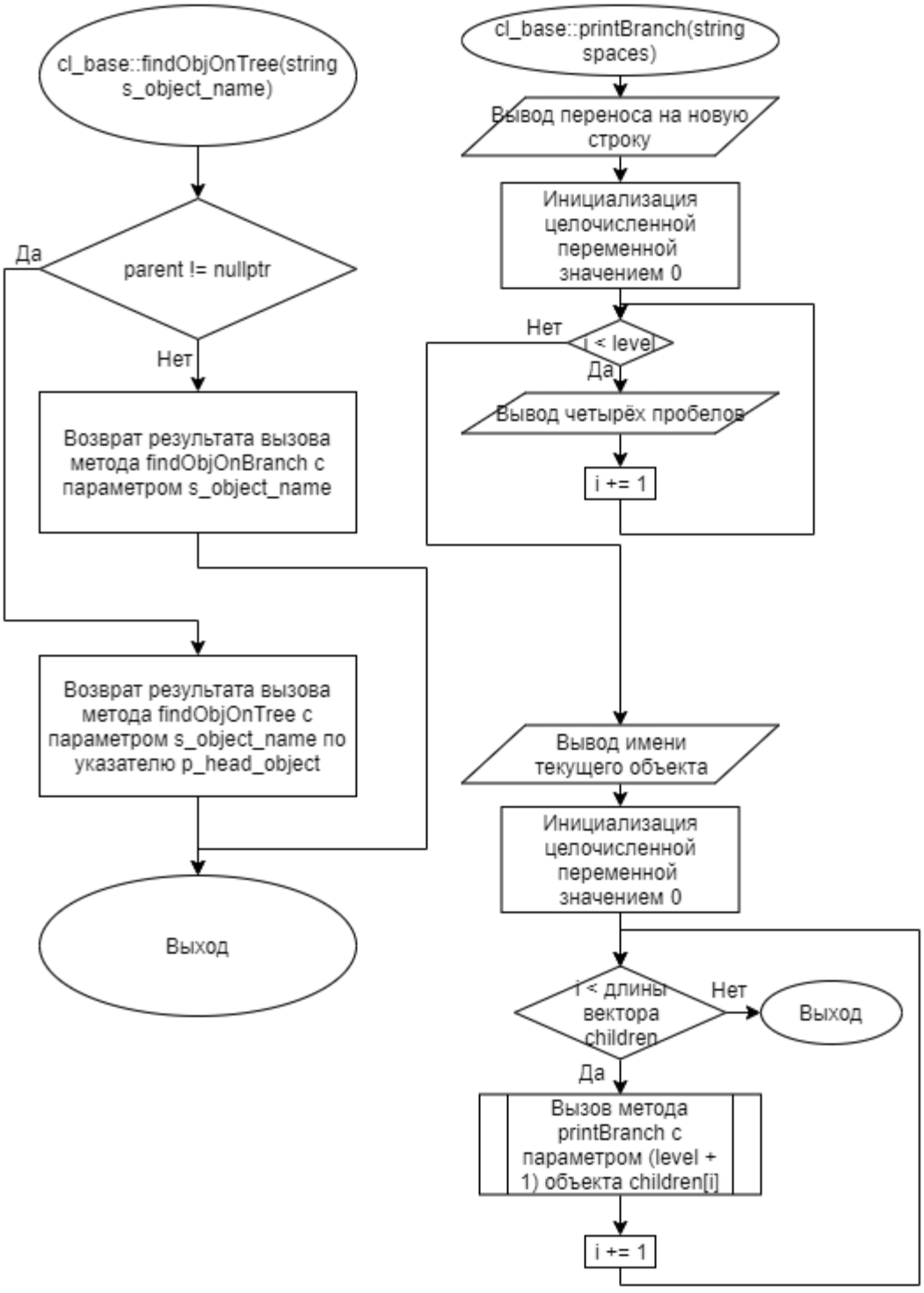

Рисунок 3 – Блок-схема алгоритма

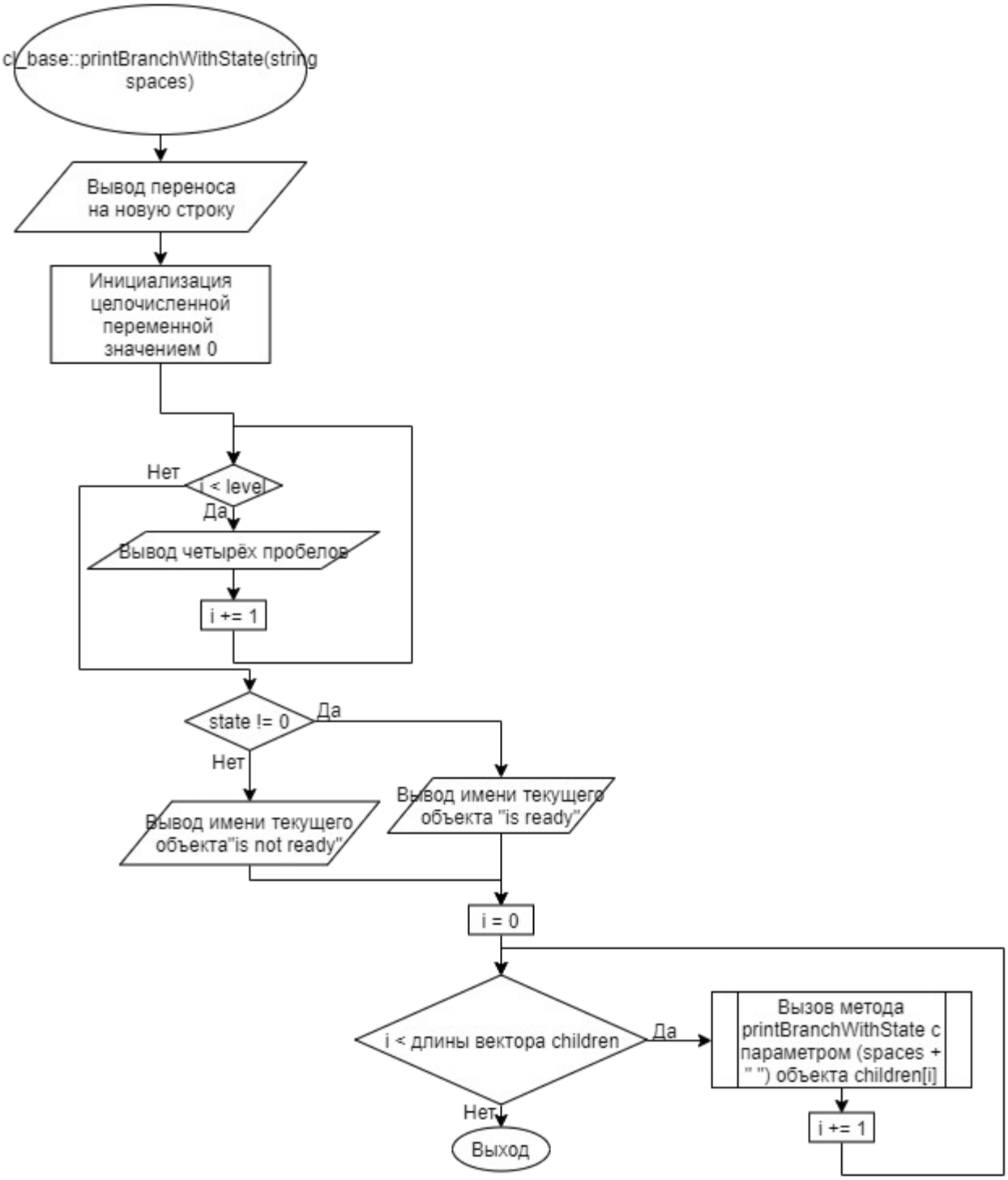

Рисунок 4 – Блок-схема алгоритма

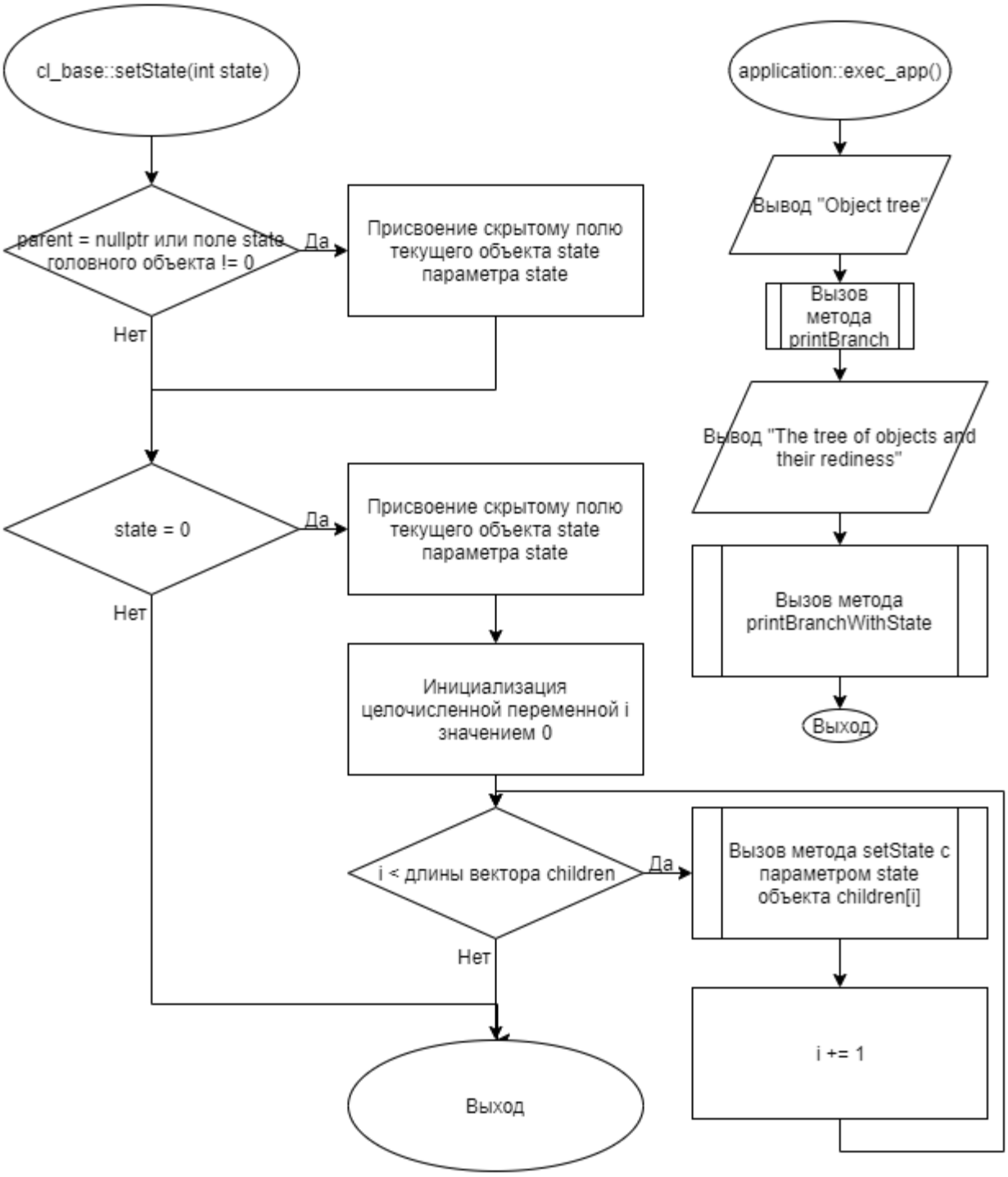

Рисунок 5 – Блок-схема алгоритма

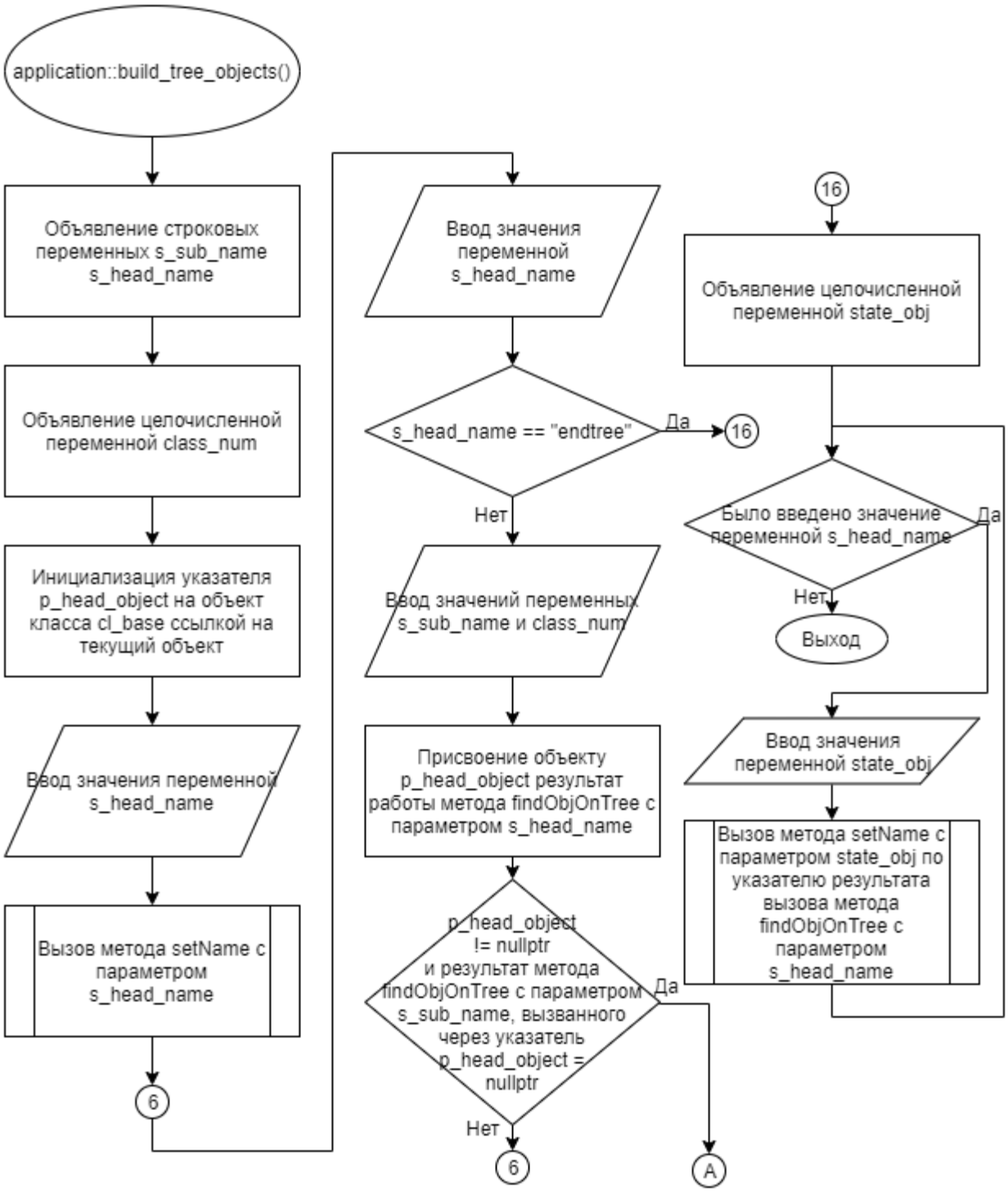

Рисунок 6 – Блок-схема алгоритма

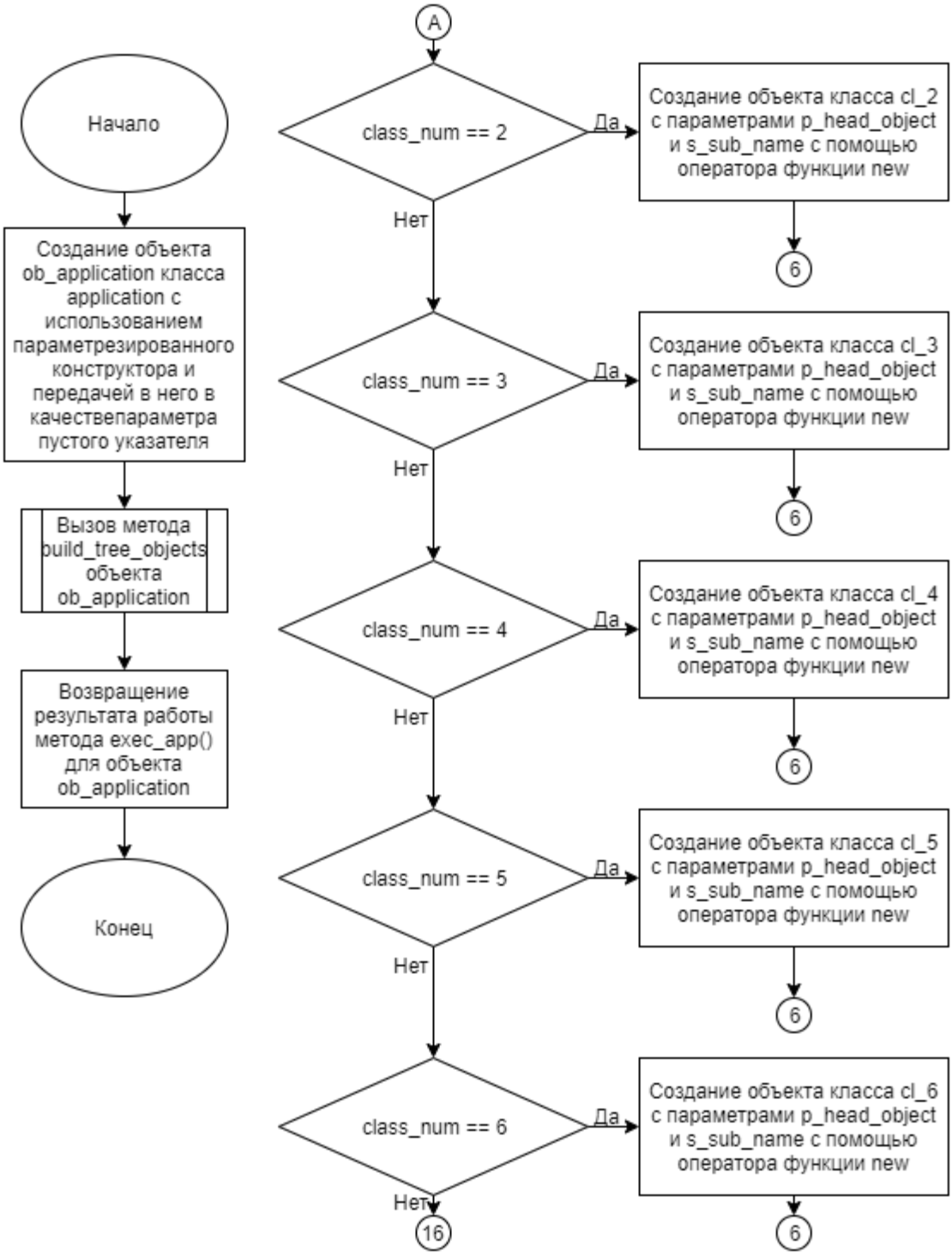

Рисунок 7 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"

application::application(cl_base* parent): cl_base(parent) {}

int application::exec_app() {
      cout << "Object tree";
      printBranch();
      cout << '\n' <<"The tree of objects and their readiness";
      printBranchWithState();
      return 0;
}

void application::build_tree_objects() {
      string s_sub_name, s_head_name;
      int class_num;
      cl_base* p_head_object = this;
      cin >> s_head_name;
      setName(s_head_name);
      while (true) {
            cin >> s_head_name;
            if (s_head_name == "endtree") {
                  break;
            }
            cin >> s_sub_name >> class_num;
            p_head_object = findObjOnTree(s_head_name);
            if      (p_head_object      !=      nullptr      &&      p_head_object-
>findObjOnTree(s_sub_name) == nullptr) {
                  switch(class_num) {
                        case 2:
                              new cl_2(p_head_object, s_sub_name);
                              break;
                        case 3:
                              new cl_3(p_head_object, s_sub_name);
                              break;
                        case 4:
                              new cl_4(p_head_object, s_sub_name);
```

```
                              break;
                        case 5:
                              new cl_5(p_head_object, s_sub_name);
                              break;
                        case 6:
                              new cl_6(p_head_object, s_sub_name);
                              break;
                        default:
                              break;
                  }
            }
      }
      int state_obj;
      while (cin >> s_head_name) {
            cin >> state_obj;
            findObjOnTree(s_head_name)->setState(state_obj);
      }
}
```

5.2 Файл application.h

Листинг 2 – application.h

```
#ifndef __APPLICATION__H
#define __APPLICATION__H

#include "cl_base.h"
class application: public cl_base {
public:
      application(cl_base* parent);
      void build_tree_objects();
      int exec_app();
};


#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
```

5.4 Файл cl_2.h

Листинг 4 – 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_object_name);
};

#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
```

5.6 Файл cl_3.h

Листинг 6 – 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_object_name);
};


#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
```

5.8 Файл cl_4.h

Листинг 8 – 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_object_name);
};

#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
```

5.10 Файл cl_5.h

Листинг 10 – 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_object_name);
};

#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base*  p_head_object,  string  s_object_name):cl_base(p_head_object,
s_object_name){

}
```

5.12 Файл cl_6.h

Листинг 12 – 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_object_name);
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"

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

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

string cl_base::get_name() const {return name;}
cl_base* cl_base::get_parent() const {return parent;}

bool cl_base::setName(string name1) {
	if (get_parent() != nullptr && get_parent() -> get_child_by_name(name1) !=
nullptr) {
		return false;
	}
	name = name1;
	return true;
}

cl_base* cl_base::get_child_by_name(string name) {
	for (auto child: children) {
		if (child->name == name) return child;
	}
	return nullptr;
}

//2 часть

cl_base* cl_base::findObjOnBranch(string s_object_name) {
	cl_base* found = nullptr;
	queue <cl_base*> elementsQueue;
	elementsQueue.push(this);
	while(!elementsQueue.empty()) {
		cl_base* elem = elementsQueue.front();
		elementsQueue.pop();
		if (elem->name == s_object_name) {
			if (found != nullptr) {
				return nullptr;
			}
			else {
				found = elem;
			}
		}
		for (int i = 0; i < elem->children.size();i++) {
			elementsQueue.push(elem->children[i]);
		}
	}
	return found;
}
cl_base* cl_base::findObjOnTree(string s_object_name) {
	if (parent != nullptr) {
		return parent->findObjOnTree(s_object_name);
	}
	else {
		return findObjOnBranch(s_object_name);
	}
}
void cl_base::printBranch(int level) {
	cout << endl;
```

```
	for (int i = 0; i < level; ++i) {
		cout << "    ";
	}
	cout << this->get_name();
	for (int i = 0; i < children.size(); ++i) {
		children[i]->printBranch(level + 1);
	}
}
void cl_base::printBranchWithState(int level) {
	cout << endl;
	for (int i = 0; i < level; ++i) {
		cout << "    ";
	}
	if (this->state != 0) {
		cout << this->get_name() << " is ready";
	}
	else {
		cout << this->get_name() << " is not ready";
	}
	for (int i= 0; i < children.size(); ++i) {
		children[i]->printBranchWithState(level + 1);
	}
}
void cl_base::setState(int state) {
	if (parent == nullptr || parent->state != 0) {
		this->state = state;
	}
	if (state == 0) {
		this->state = state;
		for (int i = 0; i < children.size(); i++) {
			children[i]->setState(state);
		}
	}
}
```

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:
	int state = 0;
	cl_base* parent;
	vector <cl_base*> children;
	string name;
```

```
public:
	cl_base(cl_base* parent, string name = "Object_root");
	~cl_base();
	bool setName(string name);
	string get_name() const;
	cl_base* get_parent() const;
	cl_base* get_child_by_name(string name);
	//2 часть
	cl_base* findObjOnBranch(string name);
	cl_base* findObjOnTree(string name);
	void printBranch(int level = 0);
	void printBranchWithState(int level = 0);
	void setState(int state);
};


#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include "application.h"
int main() {
	application ob_application(nullptr);
	ob_application.build_tree_objects();
	return (ob_application.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. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).