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).

1.1. Переключение готовности объектов согласно входным данным (командам).

1.2. 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 ob_cl_application класса Cl_application предназначен для конструирования и запуска системы;
- Объект стандартного потока ввода с клавиатуры cin;
- Объек стандартного потока вывода на экран cout;
- Условный оператор if..else;
- Оператор цикла for;
- Оператор цикла с предусловием while.

Класс Cl_base:

- свойства/поля:
  - поле Состояние объекта:
    - наименование — object_state;
    - тип — int;
    - модификатор доступа — private;
  - поле Наименование объекта:
    - наименование — s_object_name;
    - тип — string;
    - модификатор доступа — private;
  - поле Указатель на головной объект для текущего объекта:
    - наименование — p_head_object;
    - тип — Cl_base*;
    - модификатор доступа — private;
  - поле Динамический массив указателей на объекты, подчиненные текущему объекту в дереве иерархии:
    - наименование — subordinate_objects;

        - тип — vector<Cl_base*>;
        - модификатор доступа — private;
- функционал:
    - метод get_branch_object_by_name — Метод поиска объекта на ветке дереве иерархии от текущего по имени;
    - метод get_object_by_name — Метод поиска объекта на дереве иерархии по имени;
    - метод show_object_tree — Метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
    - метод show_object_tree_full — Метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;
    - метод change_object_state — Метод установки готовности объекта.

Класс Cl_application:

- функционал:
    - метод build_tree_objects — Метод построения исходного дерева иерархии объектов;
    - метод exec_app — Метод запуска приложения.

Класс Cl_child_2:

- функционал:
    - метод Cl_child_2 — Параметризированный конструктор.

Класс Cl_child_3:

- функционал:
    - метод Cl_child_3 — Параметризированный конструктор.

Класс Cl_child_4:

- функционал:
    - метод Cl_child_4 — Параметризированный конструктор.

Класс Cl_child_5:

- функционал:
    - метод Cl_child_5 — Параметризированный конструктор.

Класс Cl_child_6:

- функционал:
    - метод Cl_child_6 — Параметризированный конструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	Cl_base			Базовый класс	
		Cl_application	public		2
		Cl_child_2	public		3
		Cl_child_3	public		4
		Cl_child_4	public		5
		Cl_child_5	public		6
		Cl_child_6	public		7
2	Cl_application			Класс приложение	
3	Cl_child_2			Производный класс	
4	Cl_child_3			Производный класс	
5	Cl_child_4			Производный класс	
6	Cl_child_5			Производный класс	
7	Cl_child_6			Производный класс	

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм метода get_branch_object_by_name класса Cl_base

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

Параметры: Строка s_object_name с наименованием объекта.

Возвращаемое значение: Указатель на объект класса Cl_base.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода get_branch_object_by_name класса Cl_base

№	Предикат	Действия	№ перехода
1	Значение свойства s_object_name равно s_object_name?	Вернуть текущий объект	∅
			2
2		Объявление указателя subordinate_object на объект класса Cl_base	3
3	Перебираем элементы вектора subordinate_objects, subordinate_object- указатель на текущий элемент		4
			5
4	Результат метода get_object_name объекта по	Вернуть subordinate_object	∅

№	Предикат	Действия	№ перехода
	адресу subordinate_object равен s_object_name?		
			3
5		Объявление указателя subordinate_object на объект класса Cl_base	6
6	Перебираем элементы вектора subordinate_objects, subordinate_object- указатель на текущий элемент		7
			8
7	Результат метода get_branch_object_by_name с параметром s_object_name объекта по адресу subordinate_object не равен nullptr?	Вернуть результат метода get_branch_object_by_name с параметром s_object_name объекта по адресу subordinate_object	∅
			6
8		Вернуть nullptr	∅

3.2 Алгоритм метода get_object_by_name класса Cl_base

Функционал: Метод поиска объекта на дереве иерархии по имени.

Параметры: Строка s_object_name с наименованием объекта.

Возвращаемое значение: Указатель на объект класса Cl_base.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода get_object_by_name класса Cl_base

№	Предикат	Действия	№ перехода
1		Объявлеие и инициализация указателя base на	2

№	Предикат	Действия	№ перехода
		текущий объект	
2	get_head_object объекта base не равен nullptr?	base присвоить p_head_object объекта base	2
			3
3	Результат метода get_branch_object_by_name с параметром s_object_name объекта по адресу base не равен nullptr?	Вернуть результат метода get_branch_object_by_name с параметром s_object_name объекта по адресу base существует	∅
			4
4		Вернуть nullptr	∅

3.3 Алгоритм метода show_object_tree класса Cl_base

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

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода show_object_tree класса Cl_base

№	Предикат	Действия	№ перехода
1		Объявление указателя head_object и инициализация значением p_head_object	2
2	head_object не равен nullptr?	Увеличение отступа	3
			4
3		head_object присвоить p_head_object объекта head_object	2
4		Вывод на экран значения s_object_name	5

№	Предикат	Действия	№ перехода
5		Объявление указателя subordinate_object на объект класса Cl_base	6
6	Перебираем элементы вектора subordinate_objects, subordinate_object- указатель на текущий элемент	Вызов метода show_object_tree объекта по адреу subordinate_object	6
			∅

3.4 Алгоритм метода show_object_tree_full класса Cl_base

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

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода show_object_tree_full класса Cl_base

№	Предикат	Действия	№ перехода
1		Объявление указателя head_object и инициализация значением p_head_object	2
2	head_object не равен nullptr?	Увеличение отступа	3
			4
3		head_object присвоить p_head_object объекта head_object	2
4		Вывод на экран значения s_object_name	5
5	object_state не равен 0?	Вывод на экран " is ready"	6
		Вывод на экран " is not ready"	6
6		Объявление указателя subordinate_object на объект класса Cl_base	7

№	Предикат	Действия	№ перехода
7	Перебираем элементы вектора subordinate_objects, subordinate_object- указатель на текущий элемент	Вызов метода show_object_tree объекта по адреу subordinate_object	7
			∅

3.5 Алгоритм метода change_object_state класса Cl_base

Функционал: Метод установки готовности объекта.

Параметры: Целочисленная переменная object_state с номером состояния.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода change_object_state класса Cl_base

№	Предикат	Действия	№ перехода
1	object_state не равен 0?		4
		Свойству object_state присвоить 0	2
2		Объявление указателя subordinate_object на объект класса Cl_base	3
3	Перебираем элементы вектора subordinate_objects, subordinate_object- указатель на текущий элемент	Вызов метода change_object_state объекта по адресу subordinate_object с параметром 0	3
			∅
4		Объявление указателя head_object и инициализация значением p_head_object	5
5		Объявление флага f и инициализация true	6
6	head_object не равен nullptr?		7
			9

№	Предикат	Действия	№ перехода
7	Результат метода object_state объекта по адресу head_object равен 0?	Присвоение флагу f значения false	9
			8
8		head_object присвоить p_head_object объекта head_object	6
9	Значение f равняется true?	Свойству object_state присвоить значение object_state	∅
			∅

3.6 Алгоритм метода build_tree_objects класса Cl_application

Функционал: Метод построения исходного дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: Ничего.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода build_tree_objects класса Cl_application

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных head и sub, указателя head_object на объект класса Cl_base, целочисленных переменных iClass и iState	2
2		Ввод значения head с клавиатуры	3
3	head равно "endtree"?		7
			4
4		Ввод значений sub и iClass с клавиатуры	5
5		Присвоение указателю head_object результата метода get_object_by_name с параметром head	6
6	head_object существует и на	Создание объекта класа Cl_child_i с параметрами	2

№	Предикат	Действия	№ перехода
	ветви нет объектов с именем sub?	head_object и sub, где i - номер класса iClass	
			2
7	head успешно считано с клавиатуры?	Ввод значения iState с клавиатуры	8
			∅
8		Вызов метода change_object_state с параметром iState для объекта с адресом результата метода get_object_by_name с параметром head	7

3.7 Алгоритм метода exec_app класса Cl_application

Функционал: Метод запуска приложения.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода exec_app класса Cl_application

№	Предикат	Действия	№ перехода
1		Вывод на экран "Object tree" и переноса строки	2
2		Вызов метода show_object_tree	3
3		Вывод на экран "The tree of objects and their readiness" и переноса строки	4
4		Вызов метода show_object_tree_full	5
5		Возврат 0	∅

3.8 Алгоритм конструктора класса Cl_child_2

Функционал: Параметризированный конструктор.

Параметры: нет.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса Cl_child_2

№	Предикат	Действия	№ перехода
1		Передача параметров p_head_object и s_object_name параметризированному конструктору класса Cl_base	∅

3.9 Алгоритм конструктора класса Cl_child_3

Функционал: Параметризированный конструктор.

Параметры: нет.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса Cl_child_3

№	Предикат	Действия	№ перехода
1		Передача параметров p_head_object и s_object_name параметризированному конструктору класса Cl_base	∅

3.10 Алгоритм конструктора класса Cl_child_4

Функционал: Параметризированный конструктор.

Параметры: нет.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса Cl_child_4

№	Предикат	Действия	№ перехода
1		Передача параметров p_head_object и s_object_name параметризированному конструктору класса Cl_base	∅

3.11 Алгоритм конструктора класса Cl_child_5

Функционал: Параметризированный конструктор.

Параметры: нет.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса Cl_child_5

№	Предикат	Действия	№ перехода
1		Передача параметров p_head_object и s_object_name параметризированному конструктору класса Cl_base	∅

3.12 Алгоритм конструктора класса Cl_child_6

Функционал: Параметризированный конструктор.

Параметры: нет.

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса Cl_child_6

№	Предикат	Действия	№ перехода
1		Передача параметров p_head_object и s_object_name параметризированному конструктору класса Cl_base	∅

3.13 Алгоритм функции main

Функционал: Конструирование и запуск системы.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм функции представлен в таблице 14.

Таблица 14 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Объявление объекта ob_cl_application класса Cl_application с параметром nullptr	2
2		Вызов метода build_tree_objects объекта ob_cl_application	3
3		Возврат значения метода exec_app объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-6.

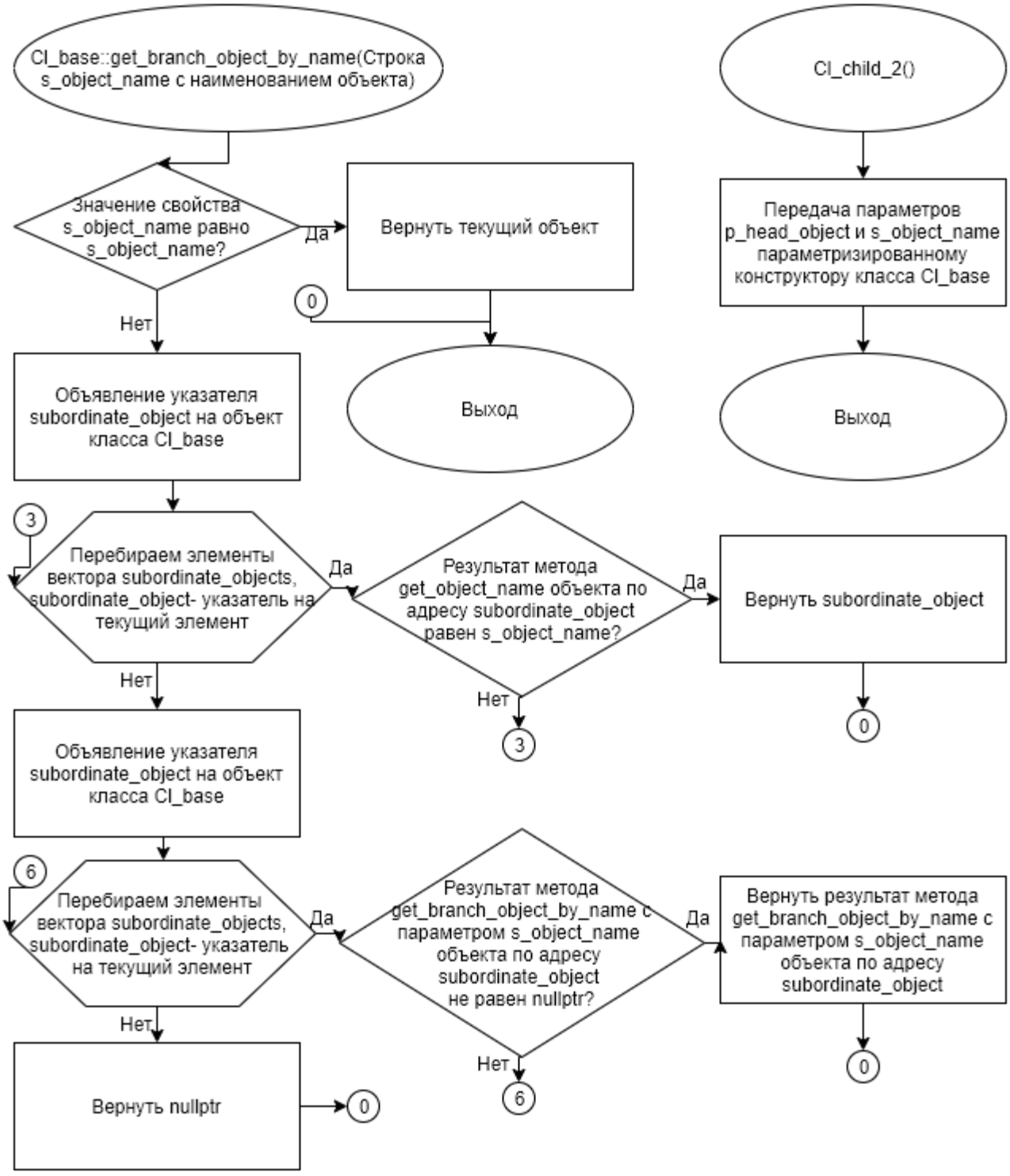

Рисунок 1 – Блок-схема алгоритма

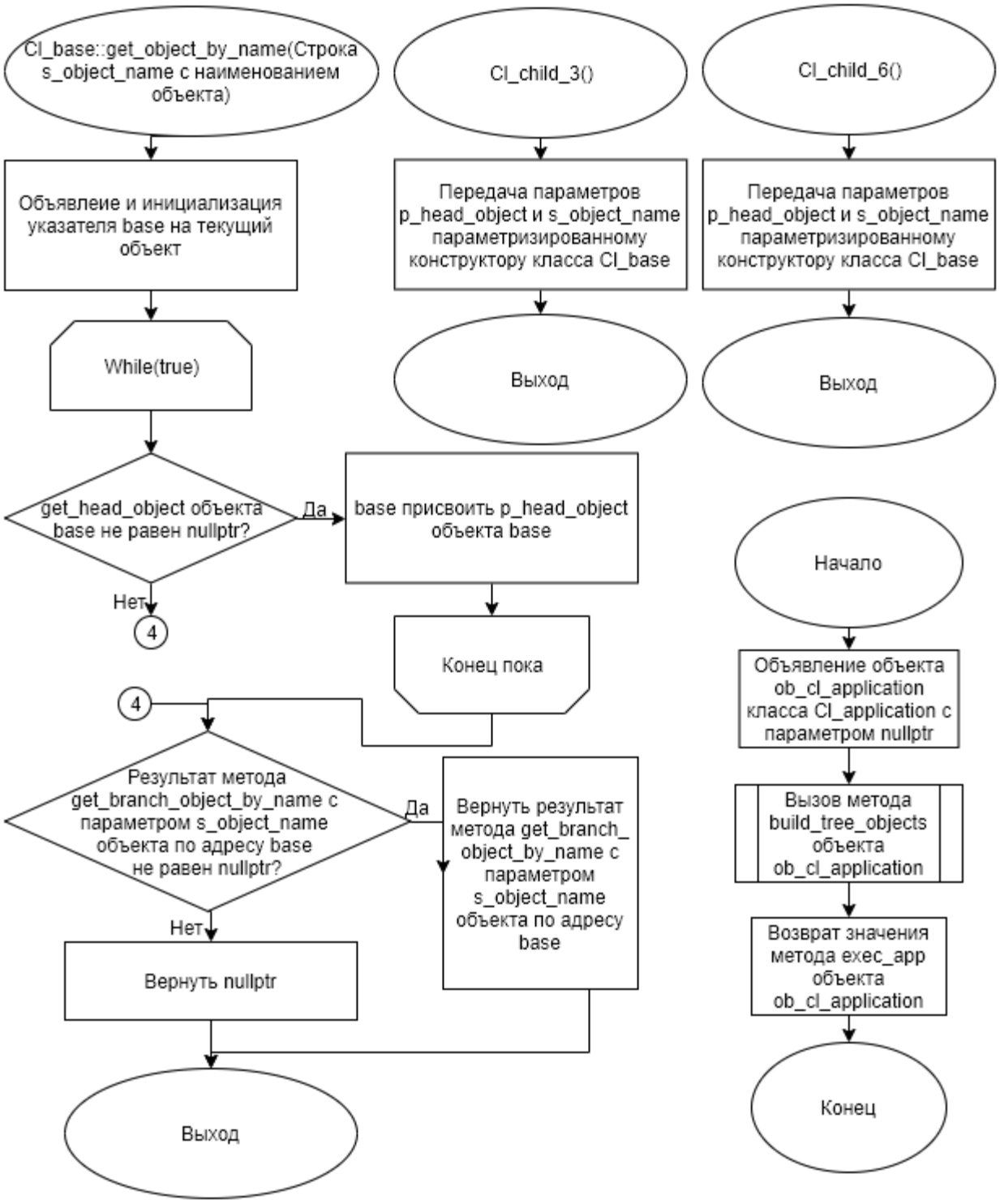

Рисунок 2 – Блок-схема алгоритма

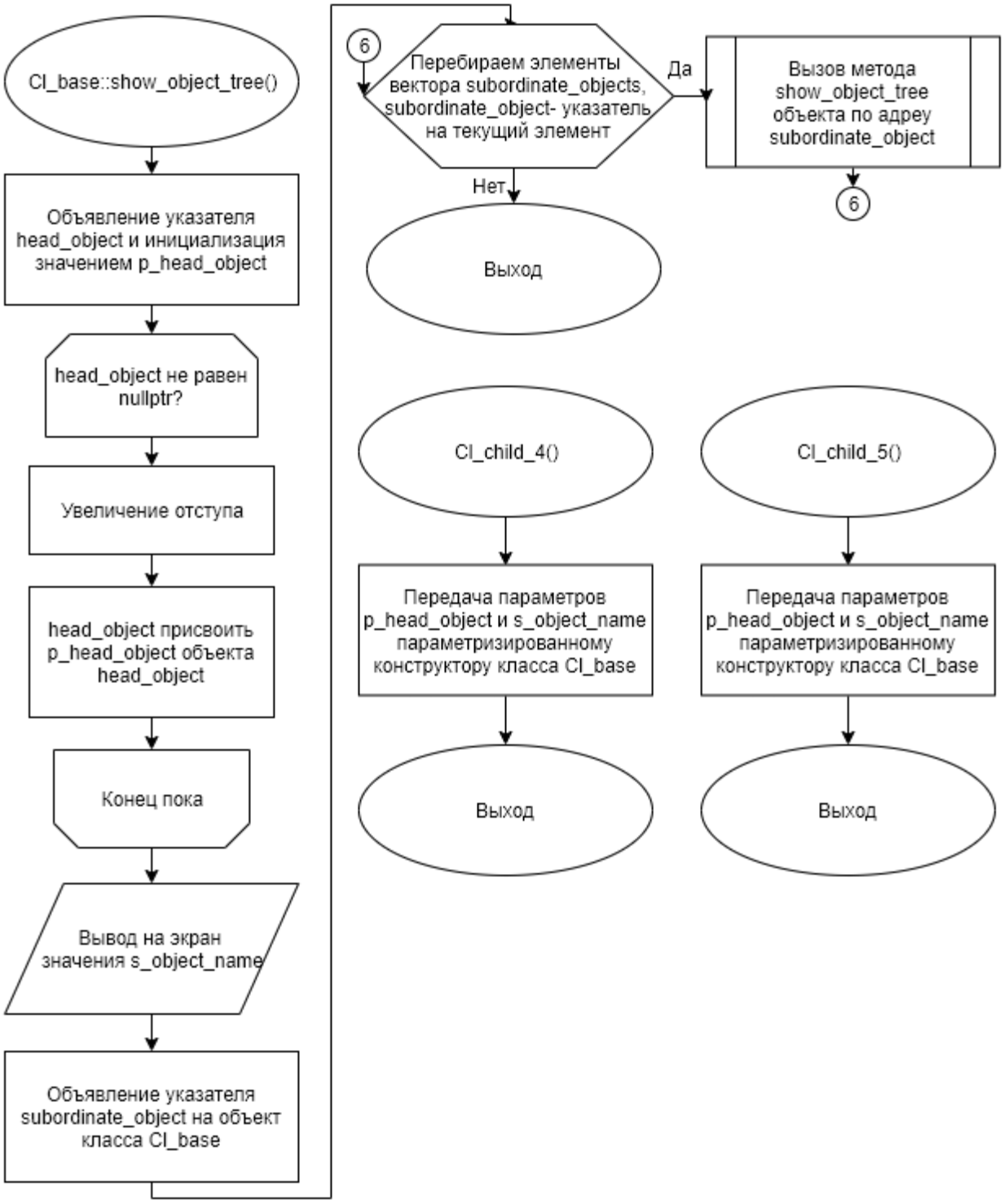

Рисунок 3 – Блок-схема алгоритма

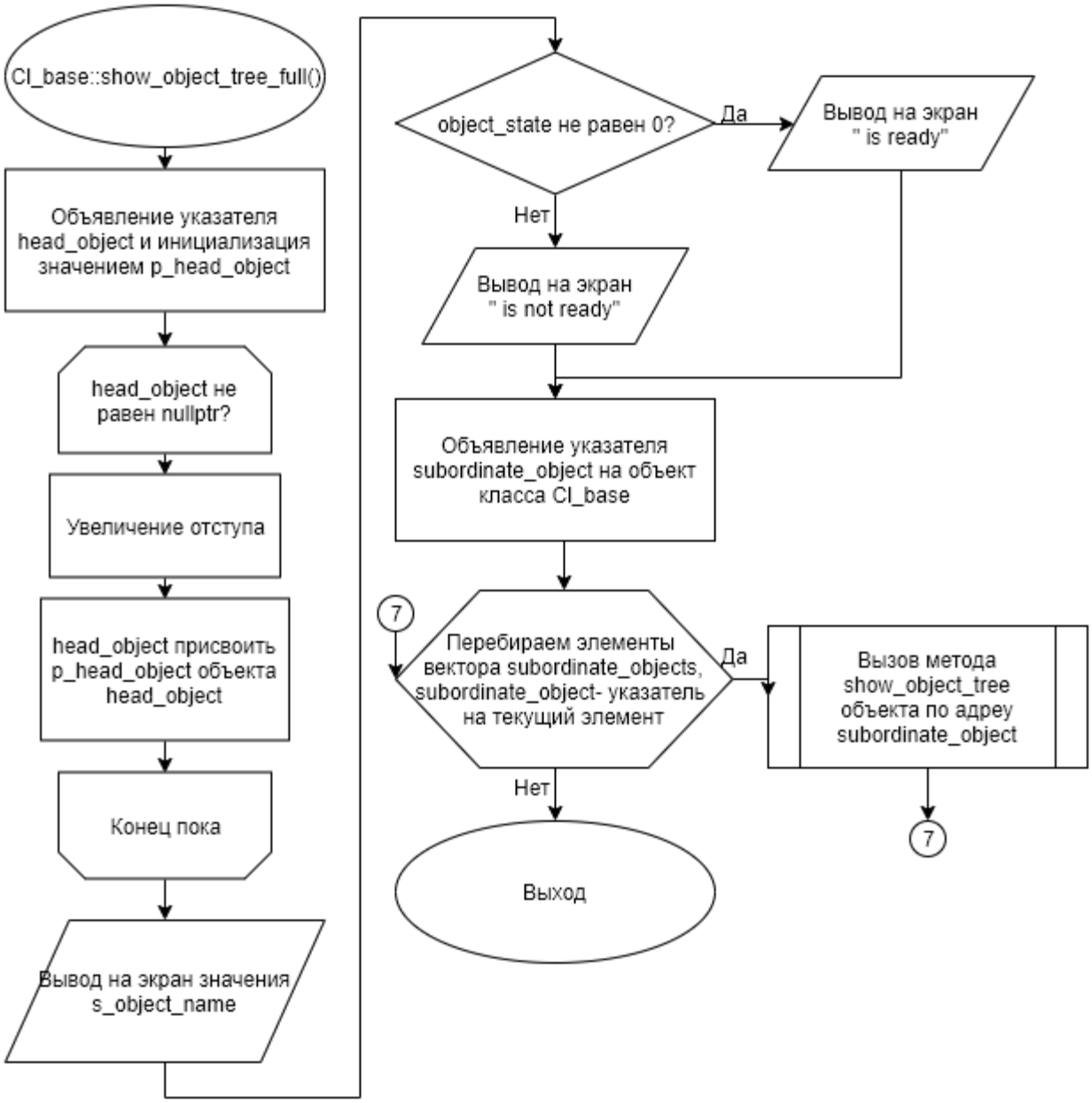

Рисунок 4 – Блок-схема алгоритма

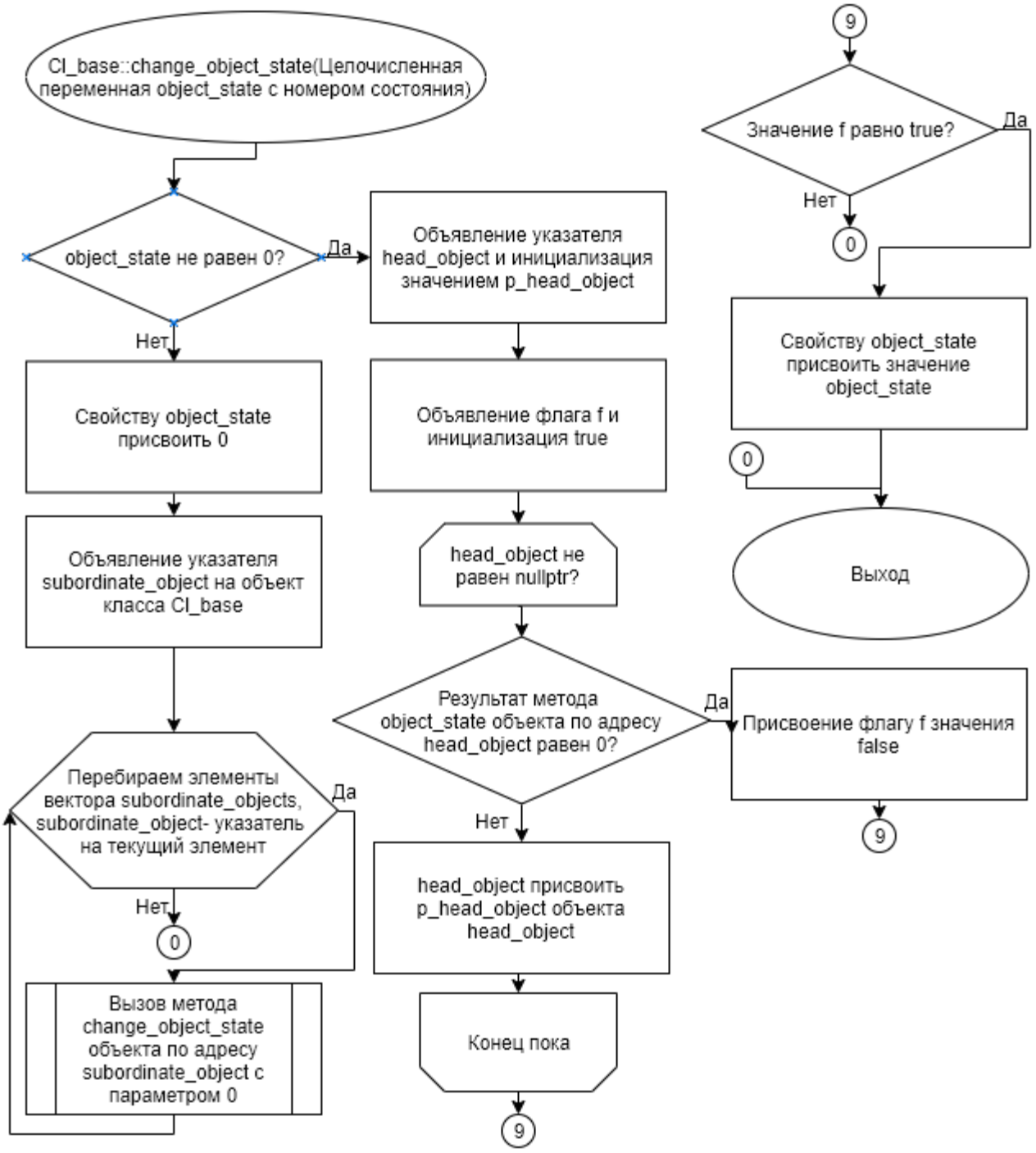

Рисунок 5 – Блок-схема алгоритма

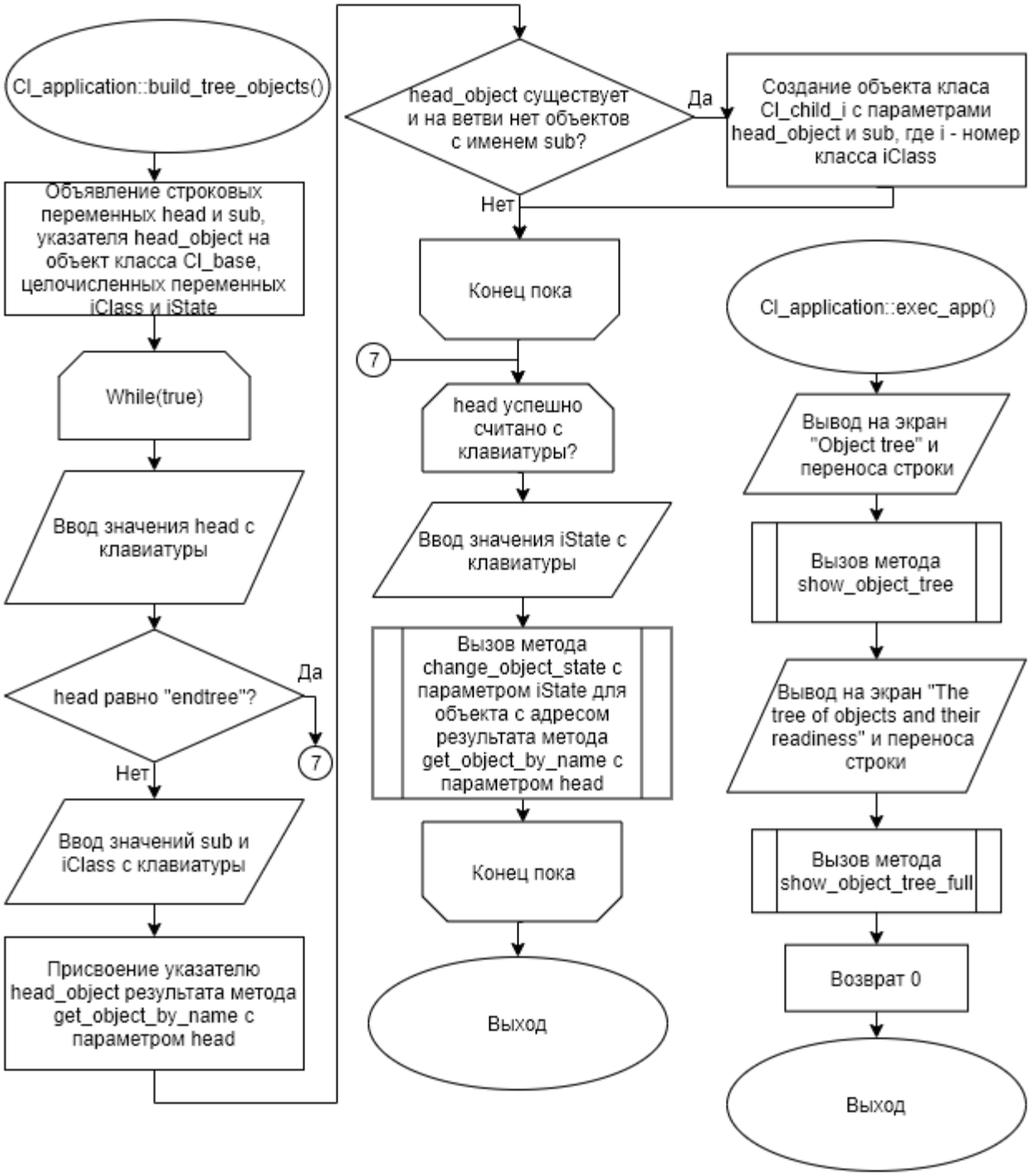

Рисунок 6 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл Cl_application.cpp

Листинг 1 – Cl_application.cpp

```
#include "Cl_application.h"

Cl_application::Cl_application(Cl_base*         p_head_object,         string
s_name_object):Cl_base(p_head_object, s_name_object)
{
   cin >> s_name_object;//ввод имени объекта
   change_object_name(s_name_object);//изменение имени объекта
}

void Cl_application::build_tree_objects()
{
   string head, sub;//переменные имен объектов
   Cl_base* head_object;//указатель на головной объект
   int iClass, iState;//номера классов и состояний
   while (true) {//ввод дерева
      cin >> head;//ввод имени головного объекта
      if (head == "endtree") {//если endtree то выход из цикла
         break;
      }
      cin >> sub >> iClass;//ввод номера класса
      head_object  =  get_object_by_name(head);//поиск  годовного  объекта  по
имени
      if      (head_object      !=      nullptr      &&      head_object      ->
get_branch_object_by_name(sub)  ==  nullptr)  {//если  есть  головной  объект  и
нет повторений имен в подчиенных
         switch(iClass)//создание объекта по номеру класса
         {
            case 2:
               new Cl_child_2(head_object, sub);
               break;
            case 3:
               new Cl_child_3(head_object, sub);
               break;
            case 4:
               new Cl_child_4(head_object, sub);
               break;
            case 5:
               new Cl_child_5(head_object, sub);
               break;
```

```
            case 6:
               new Cl_child_6(head_object, sub);
               break;
         }
      }
   }
   while (cin >> head) {//Цикл ввода состояний объектов
      cin >> iState;//ввод состояния
      get_object_by_name(head)     ->     change_object_state(iState);//Смена
состояния
   }
}

int Cl_application::exec_app()
{
   cout << "Object tree" << endl;
   show_object_tree();//вывод дерева объектов
   cout << "The tree of objects and their readiness" << endl;
   show_object_tree_full();//вывод состояний объектов
   return 0;
}
```

5.2 Файл Cl_application.h

Листинг 2 – Cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "Cl_child_2.h"
#include "Cl_child_3.h"
#include "Cl_child_4.h"
#include "Cl_child_5.h"
#include "Cl_child_6.h"

class Cl_application: public Cl_base//наследование класса
{
public:
   Cl_application(Cl_base*          p_head_object,          string          =
"Base_object");//параметризированный конструктор
   void build_tree_objects();//метод построения дерева
   int exec_app();//метод запуска системы
};

#endif
```

5.3 Файл Cl_base.cpp

Листинг 3 – Cl_base.cpp

```
#include "Cl_base.h"

Cl_base::Cl_base(Cl_base* p_head_object, string s_object_name)
{
   this -> p_head_object = p_head_object;//присвоение указателя на
родительский объект
   this -> s_object_name = s_object_name;//присвоение имени объекта
   if ( p_head_object ) {//есть родительский объект?
      p_head_object -> subordinate_objects.push_back(this);//добавить в
производные объекты
   }
}

bool Cl_base::change_object_name(string s_object_name)
{
   if (s_object_name.empty()) {//пустая строка?
      return false;
   }
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
объектов из списка
      if (subordinate_object -> get_object_name() == s_object_name) {//если
имя равно искомому
         return false;
      }
   }
   this -> s_object_name = s_object_name;//сменить имя
   return true;
}

string Cl_base::get_object_name()
{
   return s_object_name;//вернуть имя объекта
}

Cl_base* Cl_base::get_head_object()
{
   return p_head_object;//вернуть указатель на родительский объект
}

void Cl_base::show_object_tree()
{
   Cl_base* head_object = p_head_object;//указатель на головной объект
   while (head_object != nullptr) {//существует головной объект?
      cout << "    ";//отступ
      head_object = head_object -> p_head_object;//обновление головного
объекта
   }
   cout << s_object_name << endl;//вывод имени объекта
   for (Cl_base* subordinate_object : subordinate_objects) {//для всех
подчиненных объектов
```

```
            subordinate_object -> show_object_tree();//уход в рекурсию
         }
      }

      Cl_base* Cl_base::get_sub_object_by_name(string s_object_name)
      {
         if (!s_object_name.empty()) {//строка не пустая?
            for (Cl_base* subordinate_object : subordinate_objects) {//для всех
      объектов из списка
               if (subordinate_object -> get_object_name() == s_object_name) {//имя
      равно искомому?
                  return subordinate_object;//вернуть указатель на подчиненный
      объект
               }
            }
         }
         return nullptr;
      }

      Cl_base* Cl_base::get_branch_object_by_name(string s_object_name)
      {
         if (this -> s_object_name == s_object_name) {//строка совпадает с именем
      объекта?
            return this;//вернуть объект
         }
         for (Cl_base* subordinate_object : subordinate_objects) {//для всех
      подчиненных объеков
            if (subordinate_object -> get_object_name() == s_object_name) {//строка
      совпадает с именем объекта?
               return subordinate_object;//вернуть объект
            }
         }
         for (Cl_base* subordinate_object : subordinate_objects) {//для всех
      подчиненных объектов
            if (subordinate_object -> get_branch_object_by_name(s_object_name))
      {//есть в ветви такое имя?
               return                    subordinate_object                    ->
      get_branch_object_by_name(s_object_name);//вернуть объект, если есть
            }
         }
         return nullptr;
      }

      Cl_base* Cl_base::get_object_by_name(string s_object_name)
      {
         Cl_base* base = this;//указатель на текущий объект
         while (true) {
            if (base -> get_head_object()) {//существует головной объект?
               base = base -> get_head_object();//обновить текущий объект
            }
            else {
               break;
            }
         }
         if (base -> get_branch_object_by_name(s_object_name)) {//есть в дереве
```

```
такое имя?
      return   base   ->   get_branch_object_by_name(s_object_name);//вернуть
объект, если есть
   }
   return nullptr;
}

void Cl_base::show_object_tree_full()
{
   Cl_base* head_object = p_head_object;//указатель на головной объект
   while (head_object != nullptr) {//головной объект существует?
      cout << "    ";//отступ
      head_object  =  head_object  ->  p_head_object;//обновление  головного
объекта
   }
   cout << s_object_name;//вывод имени объекта
   if (object_state != 0) {//вывод состояния
      cout << " is ready" << endl;
   }
   else {
      cout << " is not ready" << endl;
   }
   for  (Cl_base*  subordinate_object  :  subordinate_objects)  {//для  всех
подчиненных объектов
      subordinate_object -> show_object_tree_full();//уход в рекурсию
   }
}

void Cl_base::change_object_state(int object_state)
{
   if (object_state != 0) {//состояние отлично от 0?
      Cl_base* head_object = p_head_object;//указатель на головной объект
      bool f = true;//объявление флага
      while (head_object != nullptr) {//головной объект существует?
         if (head_object -> object_state == 0) {//состояние головного объекта
0?
            f = false;
            break;
         }
         head_object  =  head_object  ->  p_head_object;//обновление  головного
объекта
      }
      if (f) {
         this -> object_state = object_state;//обновление состояния объекта
      }
   }
   else {
      this -> object_state = 0;//обнуление состояния объекта
      for  (Cl_base*  subordinate_object  :  subordinate_objects)  {//для  всех
подчиненных объектов
         subordinate_object -> change_object_state(0);//обнуление
      }
   }
}
```

5.4 Файл Cl_base.h

Листинг 4 – Cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Cl_base//наименование класса
{
public:
   Cl_base     (Cl_base*     p_head_object,     string     s_object_name     =
"Base_object");//параметризированный конструктор
   bool change_object_name(string);//метод изменения имени
   string get_object_name();//метод получения имени
   Cl_base*  get_head_object();//метод  получения  указателя  на  родительский
объект
   Cl_base*   get_sub_object_by_name(string);//метод   поиска   подчиненного
объекта по имени

   Cl_base* get_branch_object_by_name(string);//метод поиска объекта на ветке
по имени
   Cl_base* get_object_by_name(string);//метод поиска объекта по имени
   void show_object_tree();//метод вывода дерева объектов
   void show_object_tree_full();//метод вывода дерева объектов и состояния
   void change_object_state(int);//метод установки состояния

private:
   int object_state;//состояние объекта
   string s_object_name;//имя объекта
   Cl_base* p_head_object;//указатель на родительский объект
   vector <Cl_base*> subordinate_objects;//подчиненные объекты
};



#endif
```

5.5 Файл Cl_child_2.cpp

Листинг 5 – Cl_child_2.cpp

```
#include "Cl_child_2.h"

Cl_child_2::Cl_child_2(Cl_base*             p_head_object,             string
```

```
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.6 Файл Cl_child_2.h

Листинг 6 – Cl_child_2.h

```
#ifndef __CL_CHILD__H
#define __CL_CHILD__H
#include "Cl_base.h"

class Cl_child_2 : public Cl_base//наследование класса
{
public:
   Cl_child_2(Cl_base*, string);//параметризированный конструктор
};

#endif
```

5.7 Файл Cl_child_3.cpp

Листинг 7 – Cl_child_3.cpp

```
#include "Cl_child_3.h"

Cl_child_3::Cl_child_3(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.8 Файл Cl_child_3.h

Листинг 8 – Cl_child_3.h

```
#ifndef __CL_CHILD_3__H
#define __CL_CHILD_3__H
```

```
#include "Cl_base.h"

class Cl_child_3 : public Cl_base//наследование класса
{
public:
   Cl_child_3(Cl_base*, string);//параметризированный конструктор
};


#endif
```

5.9 Файл Cl_child_4.cpp

Листинг 9 – Cl_child_4.cpp

```
#include "Cl_child_4.h"

Cl_child_4::Cl_child_4(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.10 Файл Cl_child_4.h

Листинг 10 – Cl_child_4.h

```
#ifndef __CL_CHILD_4__H
#define __CL_CHILD_4__H
#include "Cl_base.h"

class Cl_child_4 : public Cl_base//наследование класса
{
public:
   Cl_child_4(Cl_base*, string);//параметризированный конструктор
};


#endif
```

5.11 Файл Cl_child_5.cpp

Листинг 11 – Cl_child_5.cpp

```
#include "Cl_child_5.h"

Cl_child_5::Cl_child_5(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.12 Файл Cl_child_5.h

Листинг 12 – Cl_child_5.h

```
#ifndef __CL_CHILD_5__H
#define __CL_CHILD_5__H
#include "Cl_base.h"

class Cl_child_5 : public Cl_base//наследование класса
{
public:
   Cl_child_5(Cl_base*, string);//параметризированный конструктор
};


#endif
```

5.13 Файл Cl_child_6.cpp

Листинг 13 – Cl_child_6.cpp

```
#include "Cl_child_6.h"

Cl_child_6::Cl_child_6(Cl_base*             p_head_object,             string
s_object_name):Cl_base(p_head_object, s_object_name)
{

}
```

5.14 Файл Cl_child_6.h

Листинг 14 – Cl_child_6.h

```
#ifndef __CL_CHILD_6__H
#define __CL_CHILD_6__H
#include "Cl_base.h"

class Cl_child_6 : public Cl_base//наследование класса
{
public:
   Cl_child_6(Cl_base*, string);//параметризированный конструктор
};


#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "Cl_application.h"

int main()
{
   Cl_application ob_cl_application(nullptr);//создание объекта приложения
   ob_cl_application.build_tree_objects();//конструирование системы
   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
base base abcd_01 3 base abcd_02 2 abcd_02 abcd_04 3 abcd_02 abcd_05 5 abcd_01 abcd_07 2 endtree base 1 abcd_07 3 abcd_01 1 abcd_02 -2 abcd_04 1	Object tree base abcd_01 abcd_07 abcd_02 abcd_04 abcd_05 The tree of objects and their readiness base is ready abcd_01 is ready abcd_07 is not ready abcd_02 is ready abcd_04 is ready abcd_05 is not ready	Object tree base abcd_01 abcd_07 abcd_02 abcd_04 abcd_05 The tree of objects and their readiness base is ready abcd_01 is ready abcd_07 is not ready abcd_02 is ready abcd_04 is ready abcd_05 is not ready
app_root app_root object_01 3 app_root object_02 2 object_02 object_04 3	Object tree app_root object_01 object_07	Object tree app_root object_01 object_07

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
object_02 object_05 5 object_01 object_07 2 endtree app_root 1 object_07 3 object_01 1 object_02 0 object_04 1	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 not ready object_04 is not ready object_05 is not ready	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 not ready object_04 is not ready object_05 is not ready
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 object_07 object_35 6 endtree app_root 1 object_07 3 object_01 1 object_02 -2 object_04 1 object_35 4	Object tree app_root object_01 object_07 object_35 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_35 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_35 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_35 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).