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 коневого 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

Таблица Иерархии:

Таблица 1 – Иерархия наследования классов

№	Наименование	Классф наследники	Модификатор доступа	Описание	Номер класса наследника	Комментарий
1	cl_base			Базовый класс		
		cl_application	public		2	
		cl_1	public		3	
		cl_2	public		4	
		cl_3	public		5	
		cl_4	public		6	
		cl_5	public		7	
		cl_6	public		8	
2	cl_application			Класс, производный от cl_base		
3	cl_1			Класс, производный от cl_base		
4	cl_2			Класс, производный от		

				cl_base		
5	cl_3			Класс,производный отcl_base		
6	cl_4			Класс,производный отcl_base		
7	cl_5			Класс, производный от cl_base		
8	cl_6			Класс,производный отcl_base		

- Класс cl_base
  - Поля/свойства:
    - Строковое поле
      - Наименование: s_name
      - Тип данных: string
      - Модификатор доступа: private
    - Указатель на объект
      - Наименование: p_head_object
      - Тип данных: указатель
      - Модификатор доступа: private
    - Вектор указателей на объекты
      - Наименование: p_sub_objects
      - Тип данных: vector

        - Модификатор доступа: private
- Методы:
    - Метод count
        - Функционал: Подсчёт совпадания имён
    - Метод search_by_name
        - Функционал: Поиск по имени
    - Метод search_cur
        - Функционал: Поиск по имени
    - Метод search_from_root
        - Функционал: поиск по имени от корня
    - Метод set_ready
        - Функционал: изменение состояния объекта
    - Метод get_ready
        - Функционал: Вывод состояния объекта
    - Метод ready_num
        - Функционал: возвращение готовности объекта

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 count класса cl_base

Функционал: Подсчёт совпадания имён.

Параметры: string name.

Возвращаемое значение: int count.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода count класса cl_base

№	Предикат	Действия	№ перехода
1		Определить функцию count с параметром name	2
2		Присвоить переменной count значение 0.	3
3	проверяет, соответствует ли имя текущего объекта	инкременция счётчика	4
			4
4	перебирает все подчиненные объекты текущего объекта	увеличение значения счётчика посредством рекурсивного вызова метода count с параметром name для вектора значений	5
			5
5		return count	∅

3.2 Алгоритм метода search_by_name класса cl_base

Функционал: Поиск по имени.

Параметры: string name.

Возвращаемое значение: this.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода search_by_name класса cl_base

№	Предикат	Действия	№ перехода
1	проверяет, соответствует ли имя текущего объекта	return this	2
			2
2		инициализация p_result со значением пустого указателя	3
3	перебирает все подчиненные объекты текущего объекта	p_result = рекурсивному вызову метода search_by_name с параметром name для вектора значений	4
			∅
4	проверяет, был ли найден объект с именем	вернуть p_result	∅
		вернуть пустой указатель	∅

3.3 Алгоритм метода search_cur класса cl_base

Функционал: Возвращение имени.

Параметры: string name.

Возвращаемое значение: имя.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода search_cur класса cl_base

№	Предикат	Действия	№ перехода
1	проверяет количество объектов с именем совпадающим	вернуть пустой указатель	2

№	Предикат	Действия	№ перехода
			2
2		вернуть вызов метода search_by_name с параметром name для вектора значений	∅

3.4 Алгоритм метода search_from_root класса cl_base

Функционал: поиск по имени от корня.

Параметры: string name.

Возвращаемое значение: имя.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода search_from_root класса cl_base

№	Предикат	Действия	№ перехода
1	проверяет, существует ли корневой объект в иерархии объектов	вернуть вызов метода search_from_root с параметром name для вектора значений	∅
		вернуть вызов метода search_cur с параметром name	∅

3.5 Алгоритм метода set_ready класса cl_base

Функционал: Меняет состояние объекта.

Параметры: int state.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода set_ready класса cl_base

№	Предикат	Действия	№ перехода
1		p_ready = s_new_ready	∅

3.6 Алгоритм метода get_ready класса cl_base

Функционал: Вывод состояния объекта.

Параметры: string name.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get_ready класса cl_base

№	Предикат	Действия	№ перехода
1	проверяет, соответствует ли имя текущего объекта искомому имени		2
			3
2	проверяет состояние объекта	вывести имя объекта и " is ready"	3
		вывести имя объекта и " is not ready"	3
3	перебирает все подчиненные объекты текущего объекта	вернуть результат рекурсивного вызова метода	∅
			∅

3.7 Алгоритм конструктора класса cl_2

Функционал: Объект класса.

Параметры: cl_base* p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		cl_base(p_head_object, s_name)	∅

3.8 Алгоритм конструктора класса cl_3

Функционал: Объект класса.

Параметры: cl_base* p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		cl_base(p_head_object, s_name)	∅

3.9 Алгоритм конструктора класса cl_4

Функционал: Объект класса.

Параметры: cl_base* p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		cl_base(p_head_object, s_name)	∅

3.10 Алгоритм конструктора класса cl_5

Функционал: Объект класса.

Параметры: cl_base* p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		cl_base(p_head_object, s_name)	∅

3.11 Алгоритм конструктора класса cl_6

Функционал: Объект класса.

Параметры: cl_base* p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		cl_base(p_head_object, s_name)	∅

3.12 Алгоритм метода exec_app класса cl_application

Функционал: Запуск программы.

Параметры: .

Возвращаемое значение: nullptr.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		вывод имени	2
2		вызов метода print_tree()	3
3		вывести в консоль"The tree of objects and their readiness"	4
4		вызов метода print_ready()	5
5		вернуть 0	∅

3.13 Алгоритм метода build_tree_objects класса cl_application

Функционал: Построение дерева.

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		объявление строковых переменных s_head, s_sub	2
2		объявление целочисленных переменных s_num_obj, s_redy_obj	3
3		инициализация указателей cl_base* p_head = this, * p_sub = nullptr	4
4		считывание из консоли поля s_head	5
5		вызов метода set_name с параметром s_head	6
6	бесконечный цикл		7
			7
7		считывание из консоли поля s_head	8
8	s_head == "endtree"	break	9
			9
9		считывание из консоли s_sub и s_num_obj	10
10	существует ли объект с определенным именем в иерархии объектов, и имеет ли этот объект только один экземпляр		11
			12
11		p_head = вызову метода search_from_root с параметром s_head	12
12	свич кейс		13
			14
13	s_num_obj == 1	создание объектов разных классов	14
			14
14	бесконечный цикл ввода	вызов метода set_ready с параметрами s_head, s_redy_obj	∅

№	Предикат	Действия	№ перехода
			∅

3.14 Алгоритм функции main

Функционал: Основная функция.

Параметры: .

Возвращаемое значение: Текст.

Алгоритм функции представлен в таблице 15.

Таблица 15 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Объявление объекта класса cl_application	2
2		вызов метода построения дерева для этого объекта	3
3		запуск программы	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-8.

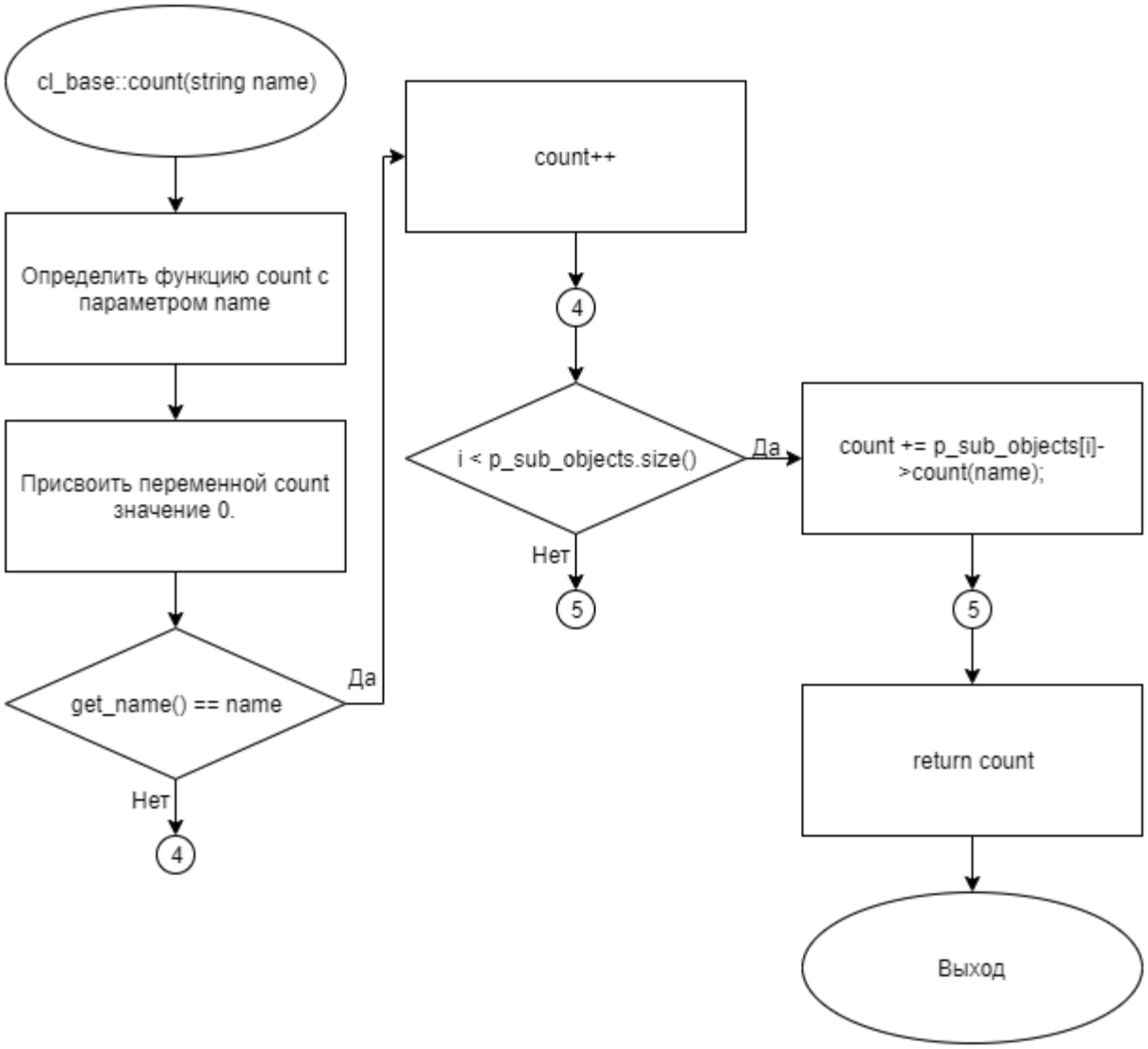

Рисунок 1 – Блок-схема алгоритма

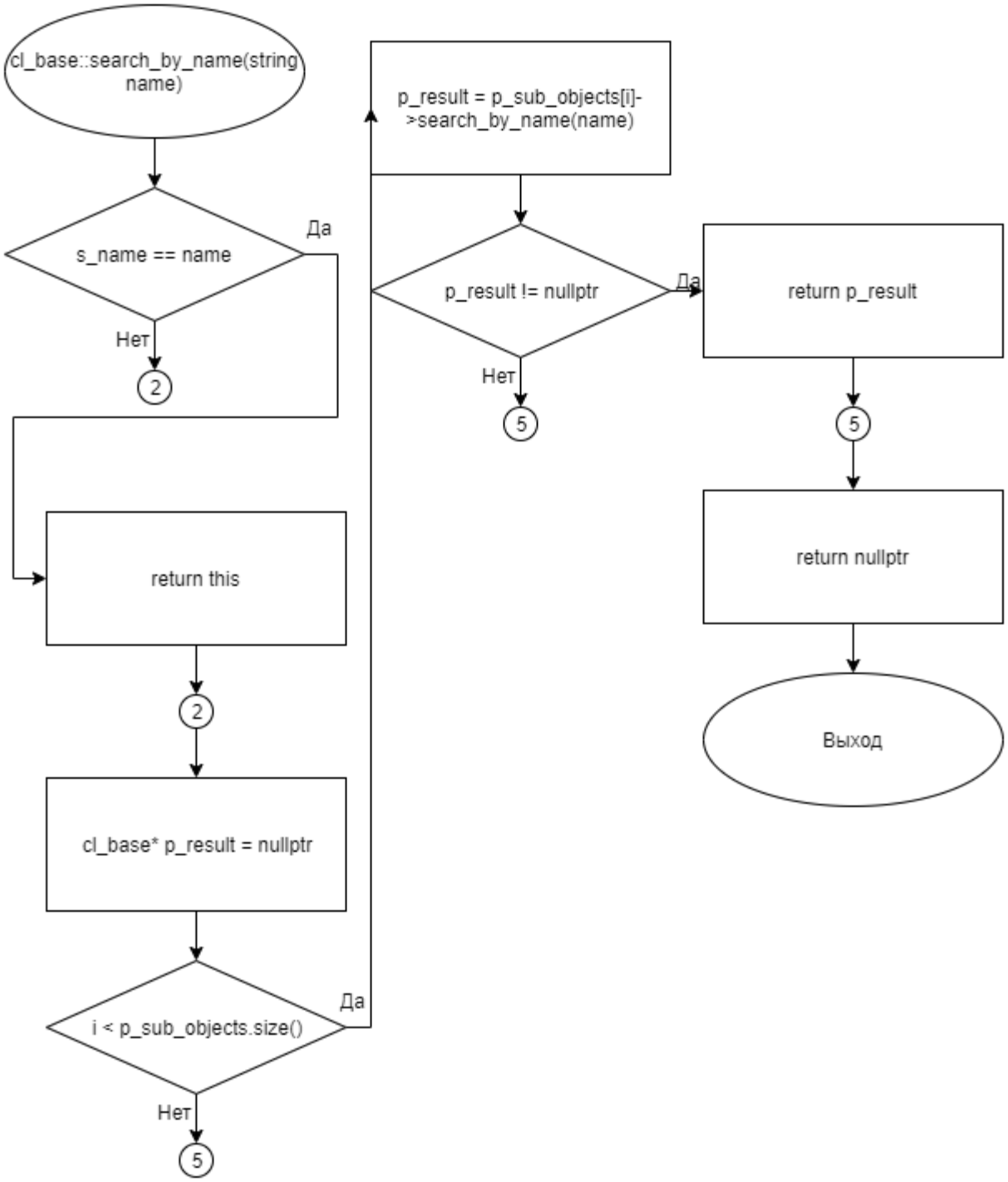

Рисунок 2 – Блок-схема алгоритма

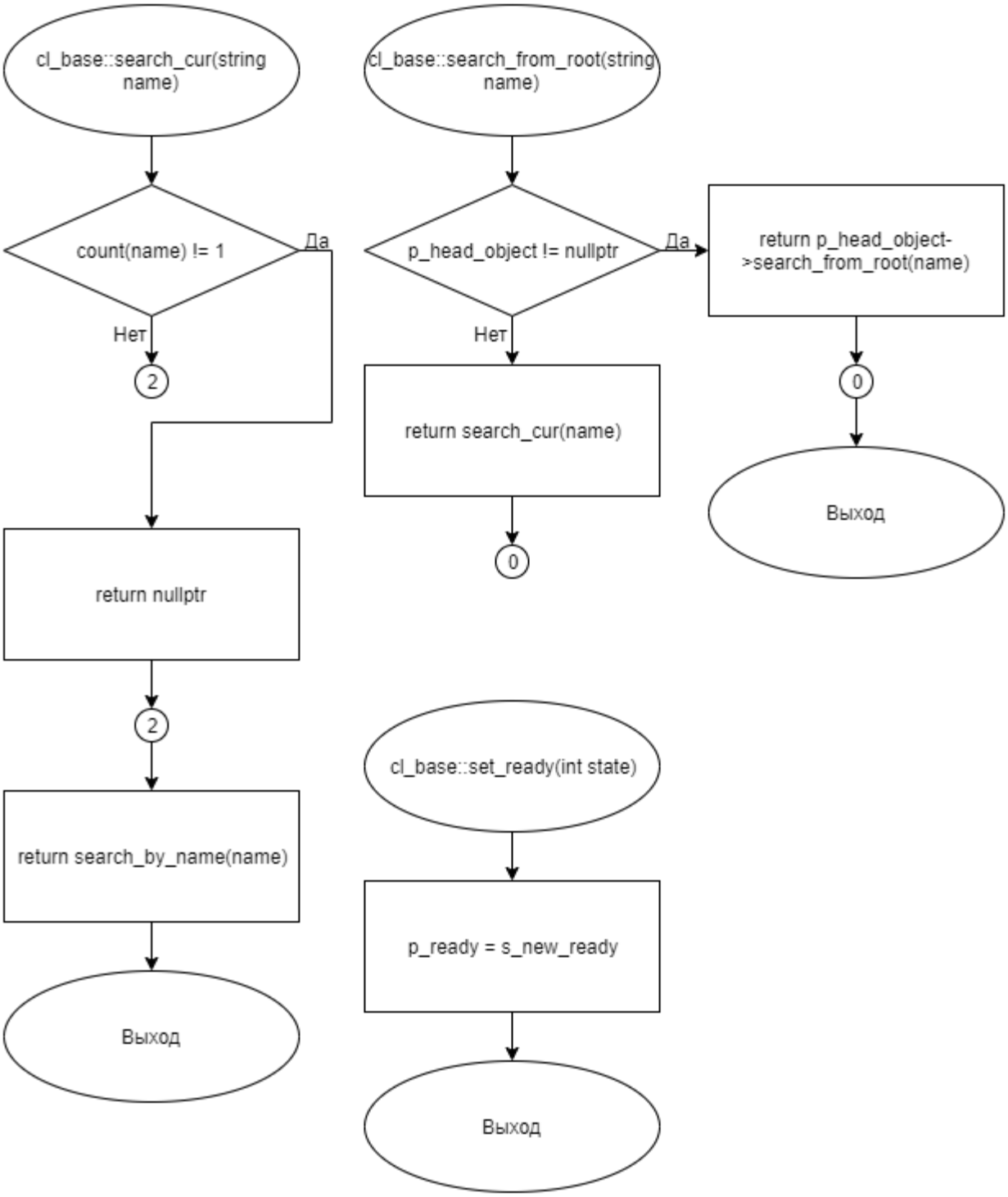

Рисунок 3 – Блок-схема алгоритма

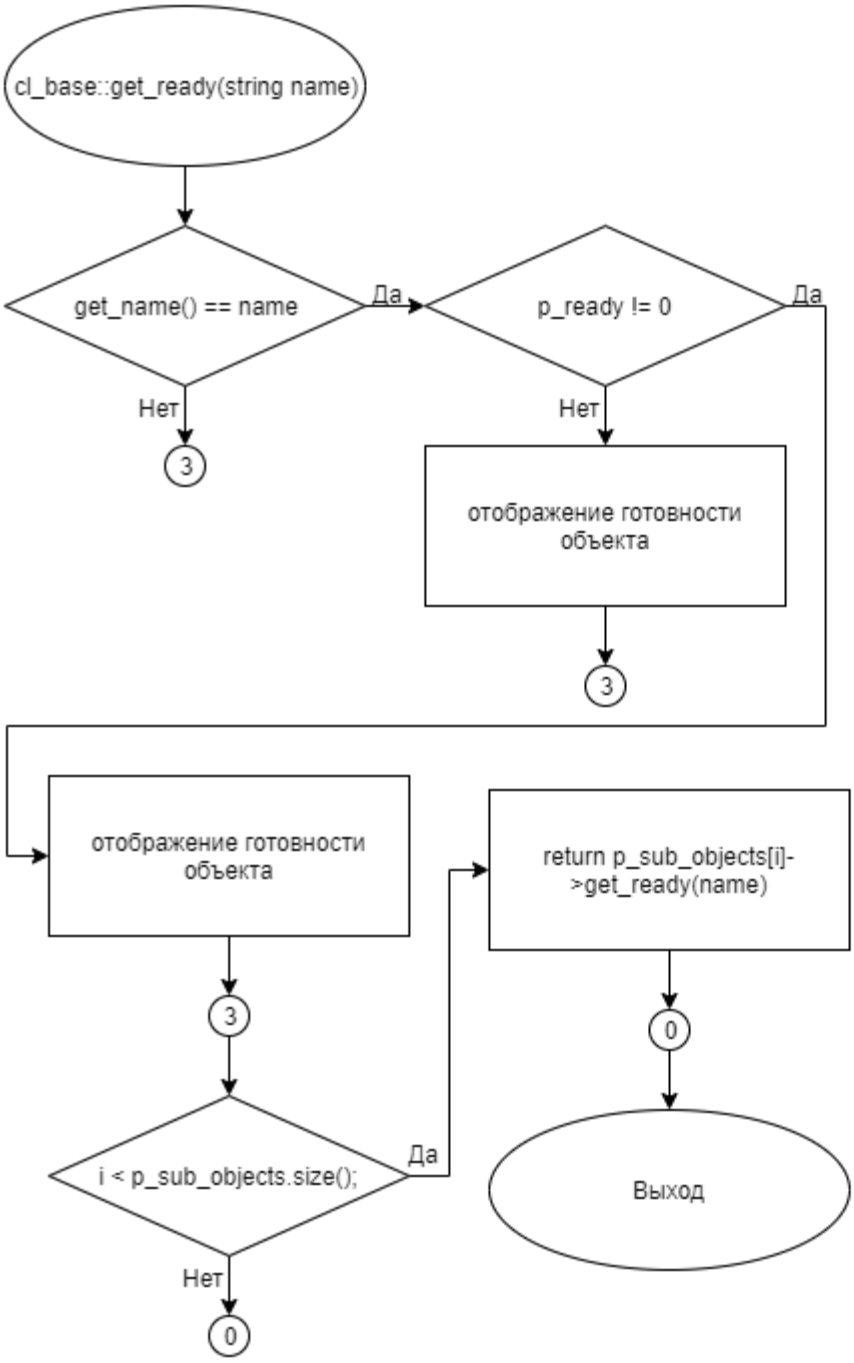

Рисунок 4 – Блок-схема алгоритма

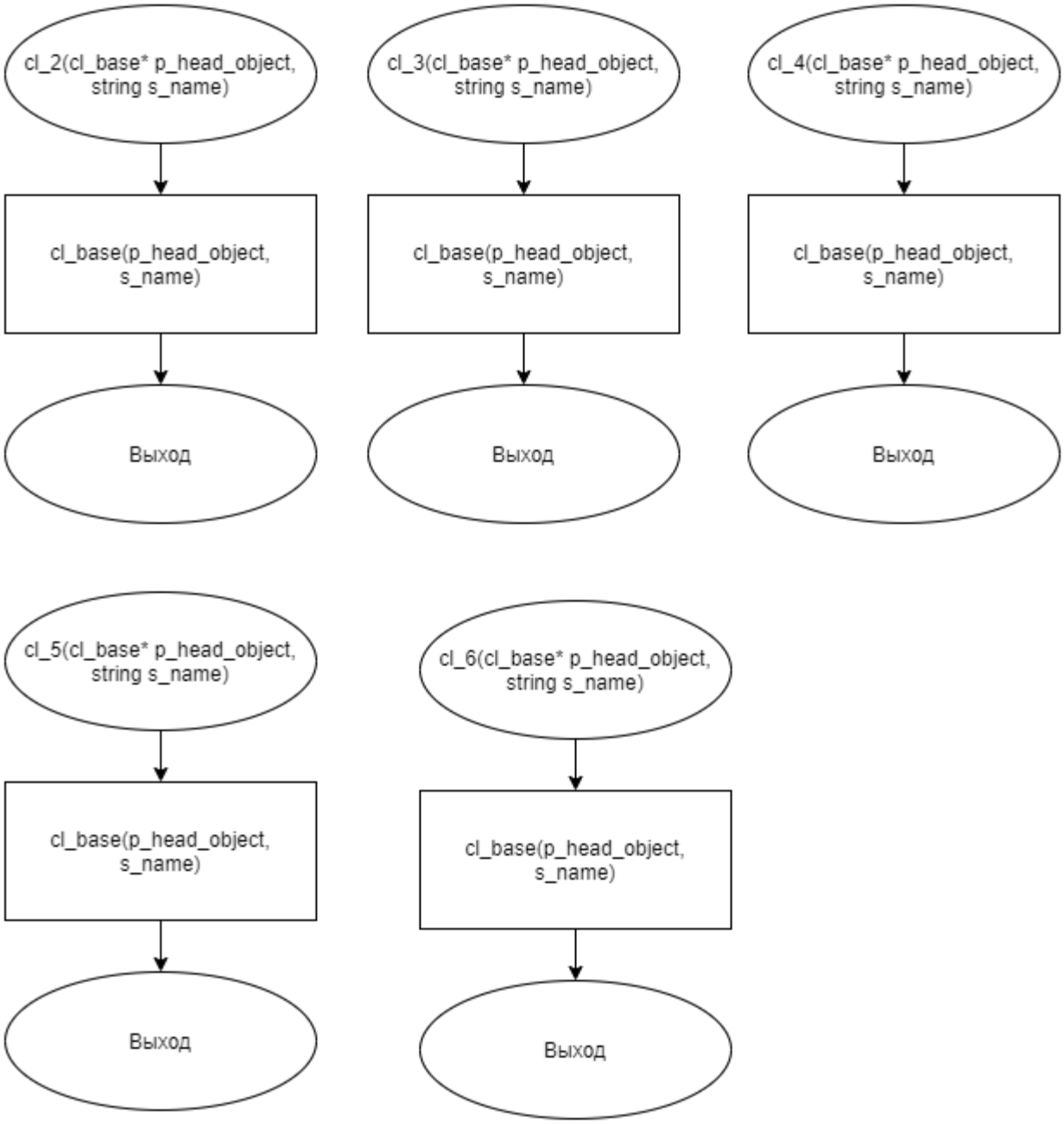

Рисунок 5 – Блок-схема алгоритма

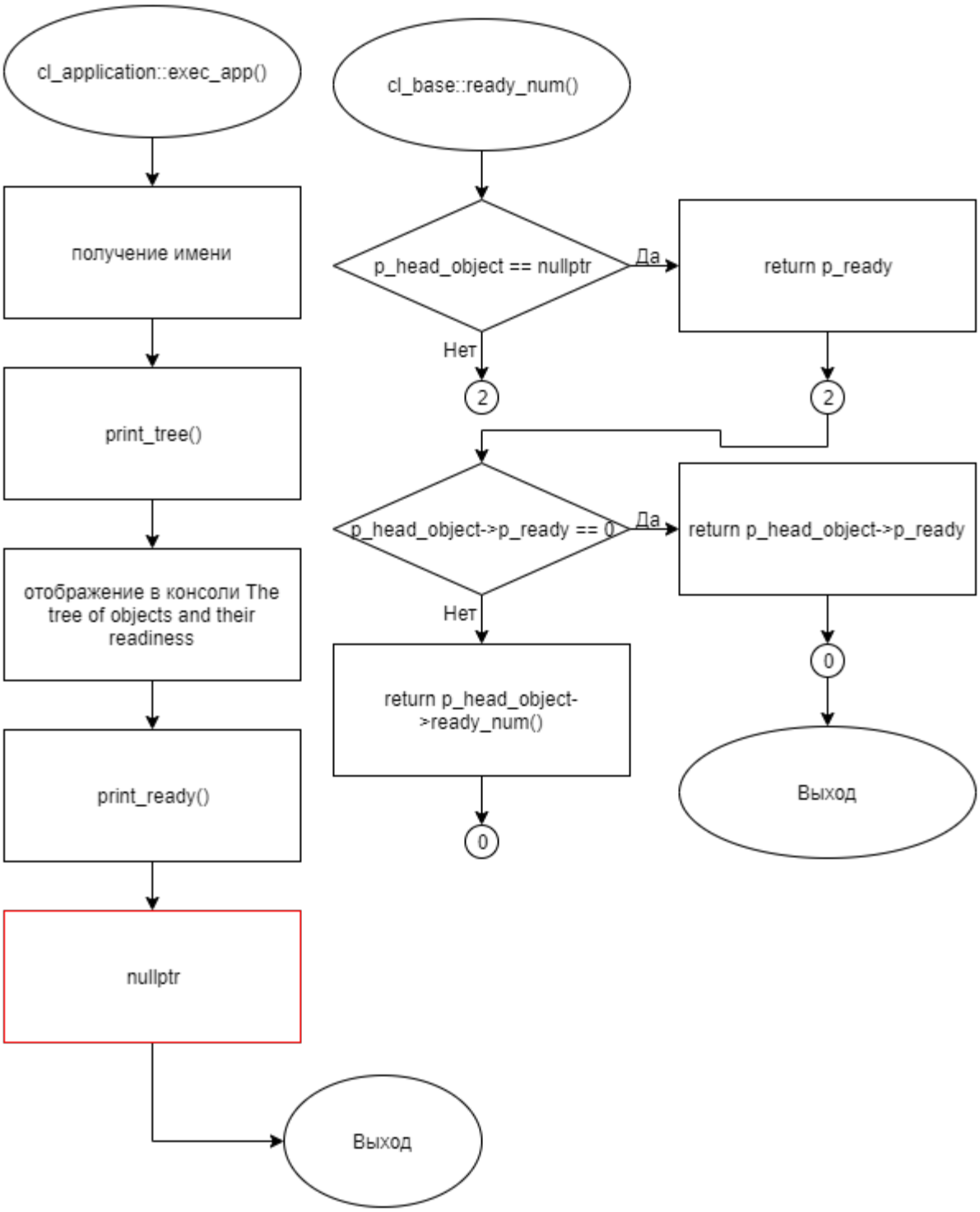

Рисунок 6 – Блок-схема алгоритма

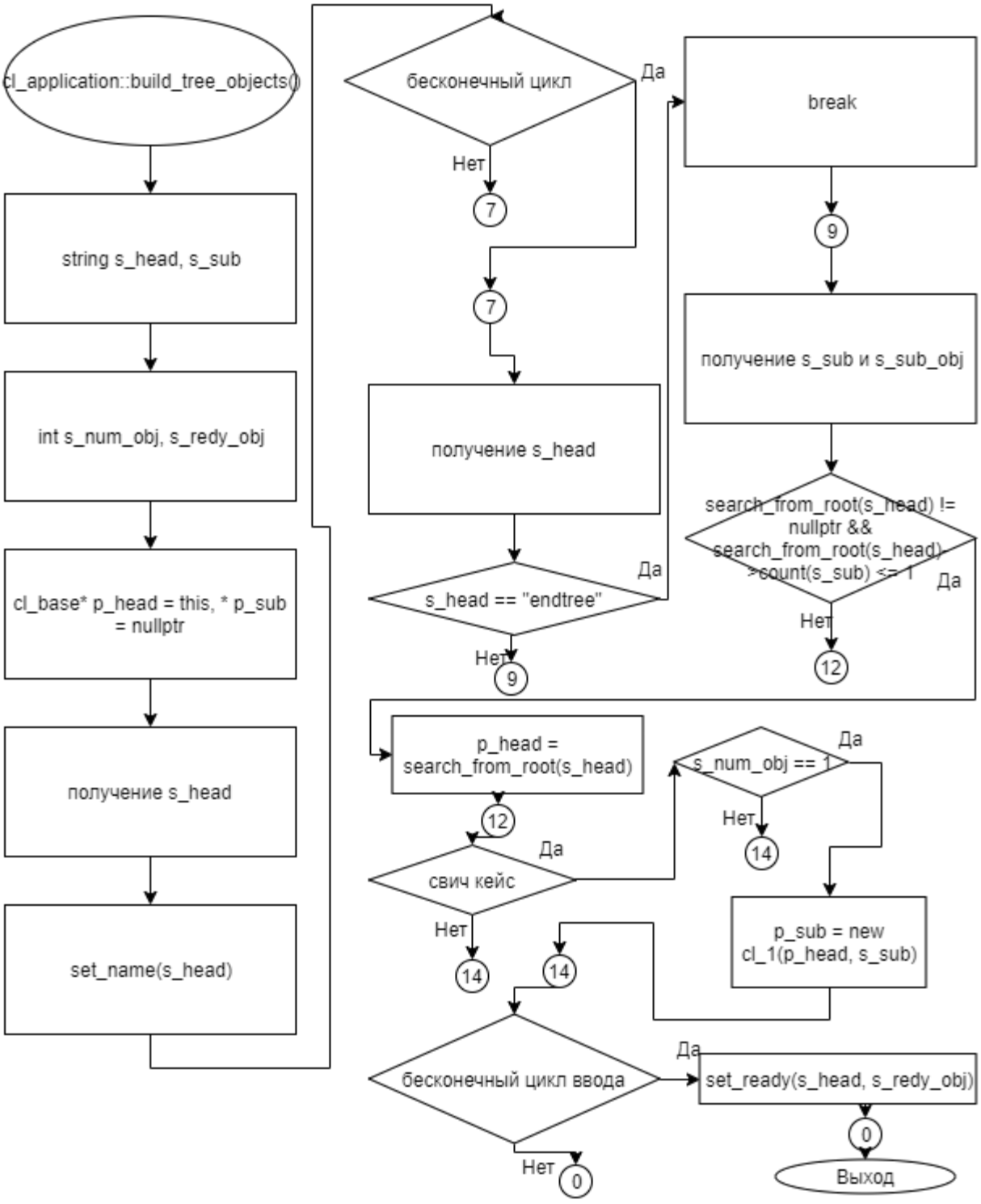

Рисунок 7 – Блок-схема алгоритма

Рисунок 8 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"

cl_1::cl_1(cl_base* p_head_object, string s_name) :cl_base(p_head_object, s_name)
{}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef __CL_1_H__
#define __CL_1_H__
#include "cl_base.h"

class cl_1 : public cl_base
{
public:
      cl_1(cl_base* p_head_object, string s_name);
};
#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_name) :cl_base(p_head_object, s_name)
{}
```

5.4 Файл cl_2.h

Листинг 4 – cl_2.h

```
#ifndef KW_CL_2_H
#define KW_CL_2_H

#include "cl_base.h"
class cl_2:public cl_base{
public:
	cl_2(cl_base* p_head_object, string s_name);
};


#endif //KW_CL_2_H
```

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_name) :cl_base(p_head_object, s_name)
{}
```

5.6 Файл cl_3.h

Листинг 6 – cl_3.h

```
#ifndef KW_CL_3_H
#define KW_CL_3_H
#include "cl_base.h"

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


#endif //KW_CL_3_H
```

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_name) :cl_base(p_head_object, s_name)
{}
```

5.8 Файл cl_4.h

Листинг 8 – cl_4.h

```
#ifndef KW_CL_4_H
#define KW_CL_4_H
#include "cl_base.h"

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


#endif //KW_CL_4_H
```

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_name) :cl_base(p_head_object, s_name)
{}
```

5.10 Файл cl_5.h

Листинг 10 – cl_5.h

```
#ifndef KW_CL_5_H
#define KW_CL_5_H

#include "cl_base.h"
class cl_5:public cl_base{
```

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


#endif //KW_CL_5_H
```

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_name) :cl_base(p_head_object, s_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_name);
};
#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.h"

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

void cl_application::build_tree_objects()
{
	string s_head, s_sub;
	int s_num_obj, s_redy_obj;
	cl_base* p_head = this, * p_sub = nullptr;
	cin >> s_head;
```

```
	set_name(s_head);
	while (true)
	{
		cin >> s_head;
		if (s_head == "endtree")
		{
			break;
		}
		cin >> s_sub >> s_num_obj;
		if (search_from_root(s_head) != nullptr && search_from_root(s_head)-
>count(s_sub) <= 1)
		{
			p_head = search_from_root(s_head);
			switch (s_num_obj)
			{
			case 1:
				p_sub = new cl_1(p_head, s_sub);
				break;
			case 2:
				p_sub = new cl_2(p_head, s_sub);
				break;
			case 3:
				p_sub = new cl_3(p_head, s_sub);
				break;
			case 4:
				p_sub = new cl_4(p_head, s_sub);
				break;
			case 5:
				p_sub = new cl_5(p_head, s_sub);
				break;
			case 6:
				p_sub = new cl_6(p_head, s_sub);
				break;
			}
		}
	}
	while (cin >> s_head >> s_redy_obj)
	{
		search_from_root(s_head)->set_ready(s_redy_obj);
	}
}

int cl_application::exec_app()
{
	cout << "Object tree";
	print_tree();
	cout << endl << "The tree of objects and their readiness";
	print_ready();
	return 0;
}
```

5.14 Файл cl_application.h

Листинг 14 – cl_application.h

```
#ifndef __CL_APPLICATION_H__
#define __CL_APPLICATION_H__
#include "cl_base.h"
#include "cl_1.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);
	void build_tree_objects();
	int exec_app();
};
#endif
```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
#include "cl_base.h"

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

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

bool cl_base::set_name(string s_new_name)
{
	if (get_head() != nullptr)
	{
		for (int i = 0; i < get_head()->p_sub_objects.size(); i++)
```

```
            {
                  if (get_head()->p_sub_objects[i]->get_name() == s_new_name)
                  {
                        return false;
                  }
            }
      }
      s_name = s_new_name;
      return true;
}

void cl_base::print_tree(string delay)
{
      cout << endl << delay << get_name();
      for (auto p_sub : p_sub_objects)
      {
            p_sub->print_tree(delay + "    ");
      }
}

void cl_base::print_ready(string delay)
{
      cout << endl << delay;
      get_ready(get_name());
      for (auto p_sub : p_sub_objects)
      {
            p_sub->print_ready(delay + "    ");
      }
}

string cl_base::get_name()
{
      return s_name;
}

cl_base* cl_base::get_head()
{
      return p_head_object;
}

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

int cl_base::count(string name)
{
      int count = 0;
      if (get_name() == name)
      {
```

```
            count++;
      }
      for (int i = 0; i < p_sub_objects.size(); i++)
      {
            count += p_sub_objects[i]->count(name);
      }
      return count;
}

cl_base* cl_base::search_by_name(string name)
{
      if (s_name == name)
      {
            return this;
      }
      cl_base* p_result = nullptr;
      for (int i = 0; i < p_sub_objects.size(); i++)
      {
            p_result = p_sub_objects[i]->search_by_name(name);
            if (p_result != nullptr)
            {
                  return p_result;
            }
      }
      return nullptr;
}

cl_base* cl_base::search_cur(string name)
{
      if (count(name) != 1)
      {
            return nullptr;
      }
      return search_by_name(name);
}

cl_base* cl_base::search_from_root(string name)
{
      if (p_head_object != nullptr)
      {
            return p_head_object->search_from_root(name);
      }
      else
      {
            return search_cur(name);
      }
}

void cl_base::set_ready(int s_new_ready)
{
      if (s_new_ready != 0)
      {
            if  (p_head_object  ==  nullptr  ||  p_head_object  !=  nullptr  &&
p_head_object->p_ready != 0)
            {
                  p_ready = s_new_ready;
            }
```

```
	}
	else
	{
		p_ready = s_new_ready;
		for (int i = 0; i < p_sub_objects.size(); i++)
		{
			p_sub_objects[i]->set_ready(s_new_ready);
		}
	}
}

void cl_base::get_ready(string name)
{
	if (get_name() == name)
	{
		if (p_ready != 0)
		{
			cout << get_name() << " is ready";
		}
		else
		{
			cout << get_name() << " is not ready";
		}

	}
	else
	{
		for (int i = 0; i < p_sub_objects.size(); i++)
		{
			return p_sub_objects[i]->get_ready(name);
		}
	}
}
```

5.16 Файл cl_base.h

Листинг 16 – cl_base.h

```
#ifndef __CL_BASE_H__
#define __CL_BASE_H__
#include <iostream>
#include <string>
#include <vector>
using namespace std;


class cl_base
{
private:
	string s_name;
	cl_base* p_head_object;
	vector <cl_base*> p_sub_objects;
	int p_ready = 0;
```

```
public:
      cl_base(cl_base* p_head_object, string s_name = "Base Object");
      bool set_name(string s_new_name);
      string get_name();
      cl_base* get_head();
      void print_tree(string delay = "");
      cl_base* get_sub_obj(string s_name);
      ~cl_base();
      int count(string name);
      cl_base* search_by_name(string name);
      cl_base* search_cur(string name);
      cl_base* search_from_root(string name);
      void set_ready(int s_new_ready);
      void get_ready(string name);
      void print_ready(string delay = "");
};
#endif
```

5.17 Файл main.cpp

Листинг 17 – main.cpp

```
#include "cl_application.h"
int main()
{
      cl_application  ob_cl_application ( nullptr );
      ob_cl_application.build_tree_objects ( );
      return ob_cl_application.exec_app ( );
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 16.

Таблица 16 – 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).